# Graph Visualization with Good Disparity*

Danyun Wang and Sheung-Hung Poon

University of Nottingham Ningbo China, Ningbo, China
zy22187@nottingham.edu.cn
Sheung-Hung.Poon@nottingham.edu.cn

**Abstract.** We present an algorithm for visualising undirected graphs based on the force model. Our heuristic strives for keeping nodes apart from other nodes, keeping nodes apart from edges and maximize minimum edge crossings angles. The basic idea of our method is as follows. Each vertex can be regarded as particles which will interact with each other. The moving directions of particles will be affected by the attractive forces and the repulsive forces from other particles and edges's fields. Edge fields is a modification repelling the near vertices in a specific range. Then, we regard the optimal layout of undirected graphs as the state in which the virtual dynamic systems have become stable.

**Keywords:** Graph drawing · Network visualization · Force-directed.

## 1 Introduction

Graph is a kind of abstract data structures, which is capable of representing the relation between some pairs of the objects in computers [**?**]. Graphs are increasingly applied in our daily life including circuit schematics, a social network on the social network analysis software, flow charts, data flow diagrams, computer network diagrams, and so on [**?**]. Therefore, the visualisation of graphs has become a significant problem.

We mainly focus on the aesthetically appealing representations of graphs to display underlying structure and meaning with readability and understanding.[**?**]. There are many different aesthetic standards for obtaining good-looking drawings of graphs. In 1995, three common aesthetics have been introduced by Purchase et al.[**?**]: maximized symmetry, minimized edge crossings, and minimized bends. With the development of research on aesthetics, the number of common aesthetics increases [**?**]. The optimality of one aspect often counteracts the optimality of others, so the graph visualization related with these aspects is NP-hard combinatorial optimization problem [**?**].

Among the most flexible approaches for calculating drawing of graphs, force-directed algorithms are the one popular category of them which tends to produce aesthetically pleasing result. We will introduces a effective algorithm for visualising undirected two-dimensional graphs based on the physical force model with

* Supported by organization x.

specific aesthetic criteria. We focus more on the crowded parts of the layout and want to avoid those situations happen. In other words, the minimal values in the graph like angles or the Euclidean distances should be maximized as much as possible. A virtual dynamic system is created in which every two vertices exert attractive and repulsive forces on one another, and every edge forms a special field to exert forces on vertices. As a result, it provides pleasing layouts with a low level of gathering and overlapping.

## 2 Related work

*Force Directed Model* Force-directed model, also known as spring embedders, is one of the most powerful methods for calculating layouts of undirected graphs. It is a heuristic algorithm based on a physical model. The visualization process is to mimic a mechanical system in which vertices are substituted by rings and springs are embedded between pairs of vertices. If the spring rings are too far apart, they will attract the rings, and if they are too close, they will repel the rings [**?**]. Algorithms with force-directed model are capable of producing an aesthetically pleasing visualizations of graphs, which tend to exhibit symmetries and crossing-free for planar graphs [**?**].

### 2.1 Eades Algorithm

In 1984, Eades proposed a method of embedding spring for drawing graph [**?**]. The vertices are replaced with steel rings and each edge with a spring to form a mechanical system. The vertices are placed in the plane to form an initial state. Then, the spring force on the ring can move the system to a state of minimum energy. Two practical adjustments were made to this idea: First, he uses logarithmic strength springs rather than apply Hooke's law. The force applied by a spring is: $c1 * \log(d/c2)$, where d is the length of the spring and c1 and c2 are constants.
Vertices are also mutually exclusive. The force bewteen vertices is $c3/\sqrt{d}$ [**?**] (c3 is a constant). The spring algorithm of Eades utilizes the nature of the spring directly, which is in accordance with conceptual intuitiveness and is simple to understand.

### 2.2 Kamada and Kawai Algorithm

In 1989, Kamada and kawai [**?**] proposed a different algorithm for drawing undirected graphs based on the "spring" ideas from Eades, and they introduced a graph theoretic approach: the desirable geometric distance between two vertices in the layout relates to the graph theoretic distance between them.
Instead of reducing the crossing number, Kamada and Kawai claimed that it was not a good standard, because the drawing with less edge crossing does not definitely look better by comparing some examples. They thought it is even more

important to keep the total balance rather than to reduce the number of edge crossings. The approach to reach a balance is started by building up a dynamic system in a way that each pair of vertices are mutually connected by springs, and the goal is to minimize the energy of the system, which corresponds to minimizing the difference between graph theoretic distances and the geometric distances.

Let $n$ denotes the number of vertices, $p1, p2, p3....pn$ be the particles (vertices), $l_{ij}$ be the original length of the corresponding spring, and $k_{ij}$ be the strength of the spring.

$$E = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{1}{2} k_{i,j}(|p_i - p_j| - l_{i,j})^2$$

By decreasing $E$, the layout can become better. The layout with minimum E is the best layout. In order to minimize $E$, Kamada and Kawai computed local minimum because of the extreme difficulty to get the global one. They used Newton-Raphson method and adopted a way of computing for only one particle $pm$ at a time so $E$ can be viewed as a function of only $x_m$ and $y_m$. For each iteration, the particle with the largest value of $\triangle_m$ will be chosen.

$$\triangle_m = \sqrt{\left(\frac{\partial E}{\partial x_m}\right)^2 + \left(\frac{\partial E}{\partial y_m}\right)^2}$$

The algorithm of Kamada and Kawai shows many good properties, like the small number of edge crossing, the symmetric drawings, and evenly edge lengths. Compared with the Eades algorithm, the drawings it produces are more expanded in some incident edges angles and occupy a larger surface intuitively. However, since it should calculate the graph theoretic distances for all pairs of vertices, $O(n^3)$ time will be cost. Each time of iteration, only one vertex will move and then all the position information should be updated again. The huge amount of computations makes it very slow to get results with the increasing number of vertices. It make a compromise for the better layouts.

### 2.3 Fruchterman and Reingold Algorithm

The Fruchterman and Reingold [**?**] algorithm is based on the Eades' spring-embedder model, which uses a mechanism of electrically charged vertices associated by some springs. It not only reflects symmetry, distributes nodes uniformly, but also makes edge lengths uniform. The model updates vertices based on two forces (attraction and repulsion). Like of Eades, the Fruchterman Reingold algorithm is executed iteratively, and after calculating the force for each iteration, all nodes will move simultaneously. When the total energy of the system is reduced to a minimum, movement will stop and get the best layout. The attractiveness only exists between linking vertices. The attractive force $fa$ repulsive force $fr$ are defined as follows, where $d$ is the distance between two vertices and $k$ is optimal length or natural spring length:

$$f_a(d) = d^2/k \qquad f_r(d) = -k^2/d$$

$$k = C\sqrt{\frac{area}{numberofvertices}}$$

Different from that of Eades, the Fruchterman and Reingold algorithm also introduces the notion of "temperature" from a noted technique called simulated annealing: "the temperature could start at an initial value (say one tenth the width of the frame) and decay to 0 in an inverse linear fashion." The displacement of the vertices is controlled by the temperature. In brief, the displacement will be smaller when the layout is more approaching to the minimum energy of the system. Compared with Eades, the Fruchterman and Reingold algorithm is more powerful.

## 3 The graph drawing algorithm

From the algorithms in section 2, we have already reviewed several kinds of layout methods with spring model and force model. Due to its powerful competence of generating pleasing drawings of graphs, our algorithm is also based on the force-directed model with implementing more aesthetic criteria.

### 3.1 Aesthetic Criteria

According to Kamada and Kawai[**?**], avoiding crossing number are not as effective as we thought. Sometimes drawings with more crossings seem better than those with fewer crossings. The more influential factor will be the crossing angle. Therefore, we adopt the criteria of angles instead of considering the crossing number. In total, We have mainly focus on three criteria, which are: keeping nodes apart from other nodes, keeping nodes apart from edges, and maximize edge crossings angles to make the graph spread out as much as possible.

### 3.2 New Dynamic System

According to Fruchterman and Reingold [**?**], ideals from springs and macro–cosmic gravity [**?**] are used in graph drawing.

> The vertices behave as subatomic particles or celestial bodies, exerting attractive and repulsive forces on one another; the forces induce movement.

Following Fruchterman and Reingold, our algorithm will also resemble this kind of molecular simulations. However, in order to approach the attractive layout of the aesthetic Criteria, we add a new interaction force: the force from the edge fields.
Inspired by electrical field effects [**?**], we suppose every charged particle should be impacted by electrical field that it is in, and the same charges should repel each other. We assume edges are full of same charges repelling the near but non-adjacent vertices, thus the vertices will not be very close to edges. Under the

influence of forces, the vertices will act and move interactively, exerting attractive and repulsive forces on others.
One thing needs to be aware is, the 'force' we mentioned is not the true force in nature which can be defined by acceleration and mass. Instead, it should be regarded as a kind of instantaneous velocity. Since the time interval is the same, forces should be the displacements for vertices for each iteration. Finally, our system leads to a static equilibrium and gives a nicer layout.
In each iteration, there are four steps: Calculate the effect of attractive forces and the effect of repulsive forces on each vertex; check and calculate the effect of forces from the edge fields; limit the displacement on each vertex; and move all the vertex at the same time. The psuedo-code for the algorithm is given in the Algorithm 1.

### 3.3 Modeling the Forces

Based on Fruchterman and Reingold [**?**], we define the desirable distance between vertices in the display plane as

$$k = \sqrt{\frac{area}{numberofvertices}}$$

And the attractive force $fa$ and repulsive force $fr$ are defined as follows, where $d$ is the distance between two vertices and $k$ is optimal length or natural spring length:

$$f_a(d) = d^2/k \qquad f_r(d) = -k^2/d$$

Intuitively, if the distance between two vertices is too small or too large, a violent intensity of correction will be exerted on the system.

### 3.4 Repulsive Direction from Edges

The forces applied by edges should make the nodes not very close to its adjacent edges, so the direction of the repulsive force from one edge should push the nodes away. Intuitively, the direction should be orthogonal to the edge, so we draw perpendicular line from the vertex to this edge. The point of intersection of is called the foot of a perpendicular. We consider the two situations when the foot is outside the edge and the foot is on the edge, which are showed below. Note that v1, v2, and v3 are vertices and there is an edge between v2 and v3.

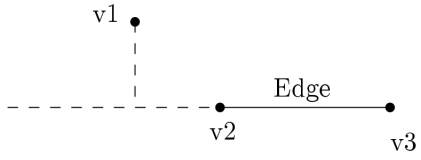



**Fig. 1.** Perpendicular line

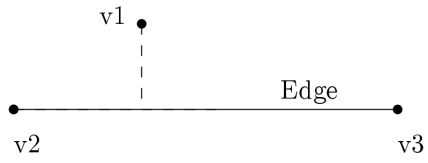



**Fig. 2.** Perpendicular line

For the first situation, the edge will not apply a force to the vertex. It is because the vertex is virtually impacted and repelled by other two vertices at the edge endpoints. In this situation, a greater impact of the spring's forces from the nearest endpoint will push the vertex move to a better position. So, there is no need for an additional force of the perpendicular direction. For the second situation, a repulsive force from this edge will exert on the vertex because the spring's forces are not enough to make the vertex apart from the edge and prevent the case when edges are close to vertices. The force direction is orthogonal to the edge.

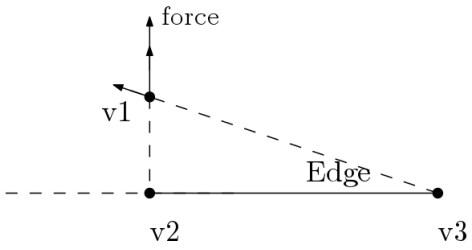



**Fig. 3.** Perpendicular line

The above picture shows a special case. When the foot of a perpendicular for v1 is on the v2 exactly, the direction of edge field on v1 is upward. However, it can be seen that two upward forces will exert on v1. Two forces has a part of overlapping and we do not need this large displacement combined by them. Thus, we also define a special range for edge fields to prevent this kind of redundant displacements.

### 3.5 Repulsive Range from Edges

In a drawing of graph, some edges are far from the vertex and some are close. If the remote edges also exert force to the vertex, the vertex will be more like a random move and the advantages of springs are greatly reduced. In order to prevent those bad movements, we limit the area of repulsing and defined a specific range centered with the edge to push vertices.

When the vertex is in the edges' force field, it will be push by the edges. One analogy is the particle in the electric field with same charge. To define the range, we can easily think of building a square which center is the midpoint of the edge. However, when the foot is near one of endpoints, the force applied by edge is still so strong. To weaken the force, we use circle range, which center is on the midpoint of the edge. In addition to the circle, the ellipse-like (not real ellipses) range can also be adapted by adding a new parameter to determine the $h$, the height of edge repulsive area. To build the range, I have to start splitting in the following situation and illustrate our way for checking whether the vertex is in the repulsive range of a specific edge.
The first situation is in figure 5. The figure 5 shows a special situation when $h = 1/2d = R$ (d is the distance between v1 and v2 on the edge or say edge length). Then the range will be a complete circle. Vertices v3, v4 will be affected and move far from the edge while vertices v5, v6 will not be affected by the edge. Since $h$ is a parameter needs predefined in algorithm, we know the radius of this circle. By comparing the distances from vertices to the midpoint in the edge, it is easy to know whether the vertices will be affected by the edge.
The distances between every pair are different. When the edge length is not equal to $2 * h$, the range is more like the range in figure 4.

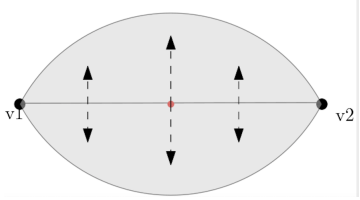



**Fig. 4.** Range of repulsing

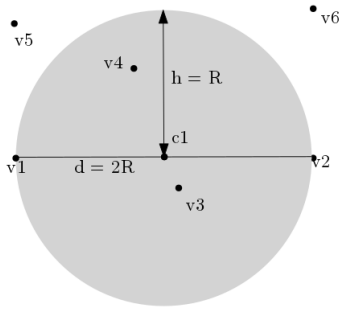



**Fig. 5.** Range of repulsing

To build the edge field, I create two virtual circle intersecting on the edge, and defined the overlapping range of two circles as the edge field range. The parameter $h$ is predefined, so we need to calculate radiuses R for two circles (radiuses are equal) and find the centers of circles. We consider two different situations seen as following (a) and (b). :
a) $h < d/2$
According to the triangle's formulae of sides in figure 3.6, we have the equation: $R^2 = (R-h)^2 + (d/2)^2$ for case 1, when the centers of circles outside the edge repulsion range. Then we get one value for radius called $R1$ by solving the equation.

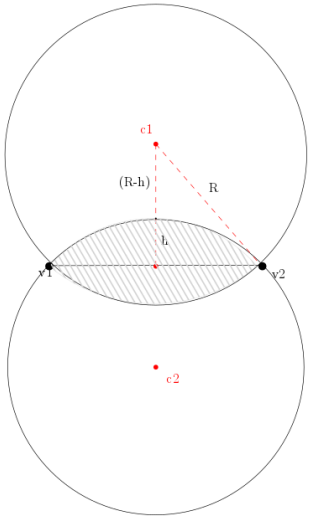



**Fig. 6.** The minimum crossing angle

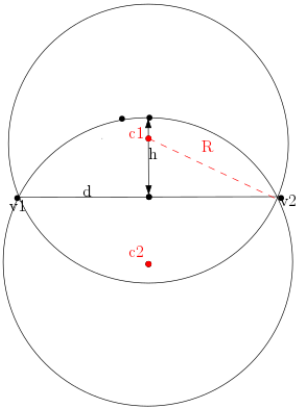



**Fig. 7.** The average of minimum angles for each vertex

As for case 2 that the centers of circles inside the edge repulsion range, we have the equation: $R^2 = (h)^2 + (d/2)^2$. Then we get another value for radius called $R2$ ($R2 > 0$). Only one radius among the R1 and R2 is the right radius. If $R1 > 2h$, then $R = R1$, else $R = R2$. b) $h > d/2$ ($d$ is the edge length)
In this situation, we also define the edge field as the overlapping region with two circles, but the direction has changed. Similar to the previous one, we can easily get the geometric relations from figure 3.8: (1) $R^2 = (R - d/2)^2 + (h)^2$ and (2) $R^2 = (d/2)^2 + (h)^2$. Then we get $R1$ and $R2$ by solving the equation. If $R1 > d$, then $R = R1$, else $R = R2$.
Since we have obtained the radius and we have known the positions of two vertices and $h$, the centers of two circles can be calculated by vectors or linear equations. The way to check is calculating the distances from the vertex to this

two centers. If both distances less than the radius, it is in the range; else not in the range. In this way, we can checking whether a vertex is in edge field.

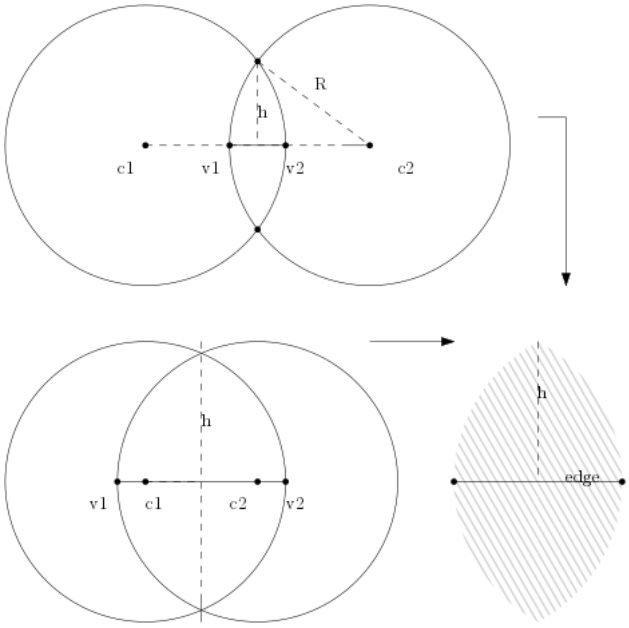



**Fig. 8.** Situation 2: $h > d/2$

### 3.6 Repulsive Force from Edges

Visually, when a vertex is nearer to an edge, we can see that a greater extent of visual confusion appears. In an extreme case that the vertex is on an edge, people cannot identify the previous edge. Instead, they may see some nonexistent edges by mistake. Therefore, the intensity in repulsing ranges should be unevenly distributed. The closer the vertex getting to the edge, the larger force the edges exerting to the vertex. The vertex will be affected by all of its adjacent edges (in those edges field). We define the force from edge field as follows:

$$Fe_{v,e} = c\frac{1}{d_{v,e}^2}$$

where v is the vertex and e is the edge; $d_{v,e}$ is the vertical distance between a vertex v and an edge e; c is a constant. To prevent the case that the repulsive force is infinite when d equals to zero, we should add a very small number to d or check the boundary.

### 3.7 Algorithm

**Algorithm 1** Visualisation with good disparity

**Input:** Graph $G = (V, E)$
**Output:** Straight-line drawing of $G$

```
initialize v_1, v_2, v_3....v_n on a circle;
k ← sqrt(W * L/|V|);
t ← sqrt(W/c0); {W and L are the width and length of the frame}
function fa(v, u) = attractive force(vector) from vertices u to v;
function fr(v, u) = repulsive force(vector) from vertices u to v;
function fe(v, e) = repulsive force(vector) from edge e to v;

for i = 1 to iterations do
  for v in V do
    v.disp = 0;
    for u in V do
      if u ≠ v then
        v.disp = v.disp + fr(v, u)
      end if
    end for
  end for

  for e in E do
    {each edge is an ordered pair of vertices .v and .u}
    v.disp = v.disp − fa(v, u)
    u.disp = u.disp + fa(v, u)

    if i > iterations/c1 then
      {Add edge field force}
      for v in V do
        check if v is in e's edge field range;
        h = min(W/c2, (l + k)/c3); {l is the edge length}
        v.disp = v.disp + fe(v, e);
      end for
    end if
  end for

  for v in V do
    v.pos = v, pos + (v.disp/|v.disp|) * min(v.disp, t);
  end for
  { reduce the temperature as the layout approaches a better configuration }
  t = cool(t);
end for
```

The purpose of algorithm is to minimize the summation of forces on vertices in the virtual dynamic system. On each iteration, all the vertices will move to

new positions simultaneously. The stopping condition is reaching the limited iteration count. When the vertices become moveless (positions change little), we regard the system as stable. The simple framework of the algorithm can be seen Algorithm 3. For the initialization, two ways can be adopted. The first way is to place vertices randomly on the canvas, then draw the edges. The second way is to put vertices on a circle. Compared with the random positions, initial positions on a circle seems better for dispersing.

As for the movement, the relation between vertices' force and edge's force will greatly affect the performance. If edge's force is too strong, it will dominate vertices' force and the movement mainly towards the direction of escaping the current adjacent edges. Then a problem comes up. When the new positions is far from the previous position, the vertex may go to a position close to other edges because those new adjacent edges will not act on this movements before. A lot of meaningless movement will occur under the problem, which greatly reduce the performance of the algorithm.

We adopt two strategies to eliminate this problem. First, we use the idea of 'temperature' from a noted technique simulated annealing, which has been mentioned before. By using 'temperature', we can limit the displacement of the vertices in a certain maximum value to restrict the movement. Second, we do not add the influence of edge fields at the beginning. Using parameter $c1$ to decide when we should activate the edge fields will let vertices to get a better state at the preliminary stage. The effects from edge fields will be combined at the later iterations. Back to 'temperature', this maximum displacement value will decrease with time. The decrease function is cooling function written as *cool(t)*. Therefore, the adjustments will become smaller as the layout improves. In our algorithm, $cool(t) = 0.95t$ when $t > W/c$ and $cool(t) = W/c$ otherwise. We set the initial $t$ equals to 0.15 (the canvas size is 1.0 with equal height and width) and $c = 1000$ in our algorithm. See algorithm in *Algorithm 3* (next page). $c0, c1, c2, c3$, and $iterations$ are constants. We set $c0 = 10, c1 = 2, c3 = 4, iterations = 1000$.

## 4 Experimental Results

### 4.1 Evaluate function

Evaluate function is useful to compare and analyze the results of graph layouts. When graphs become complicated, it is hard for people to identify and judge quickly and directly whether the output is good. Designing evaluate function helps us estimate the goodness with respect to the our aesthetic criteria from a quantitative perspective.

The evaluate function is useful especially for those combinatorial optimization problem, referring to fitness function in genetic algorithm. Vrajitoru has added the *total error*, *Surface*, *Angle*, and *Overlap* constraints in the fitness function, aiming to maximize the surface and volume occupied by the graph, and minimize the absolute difference between the angles and the overlap of the graph facets [**?**,**?**]. Hence, we are also faithful to our aesthetic standards strictly when

we design the evaluation function.

**Algorithm 2** Evaluate function

**Input:** Graph and vertices positions $(x_1, y_1)...(x_n, y_n)$
**Output:** Evaluation value F

{Measurement of minimum}
$minAngle \leftarrow$ the minimum crossing angle in the drawings
$F_1 \leftarrow min(minAngle/60, 1.0)$;

$k \leftarrow 1/\sqrt{1/n}$ {The ideal length}
$d2 \leftarrow$ the minimum distance between nodes
$F_2 \leftarrow$ min(d2/k, 1.0)

$d3 \leftarrow$ the minimum distance from nodes to edges (perpendicular)
$F_3 \leftarrow min(d3/k, 1.0)$

{Measurement of holistic distribution}
$threshold \leftarrow k/2$
$n1 \leftarrow$ the number of pairs of points when the distance between them is short (less than threshold)
$F_4 \leftarrow min((1 - n1/gsize), 1.0)$

$n2 \leftarrow$ the number of point in the range of edge repulsion
$F_5 \leftarrow min((1 - n2/gsize), 1.0)$

$avgdis \leftarrow$ the average distance of edge repulsion
$F_6 \leftarrow (avgdis/threshold)^2$

$avgAngle \leftarrow$ the average angle for all the vertices in the drawings
$F_7 \leftarrow min(avgAngle/60, 1.0)$;

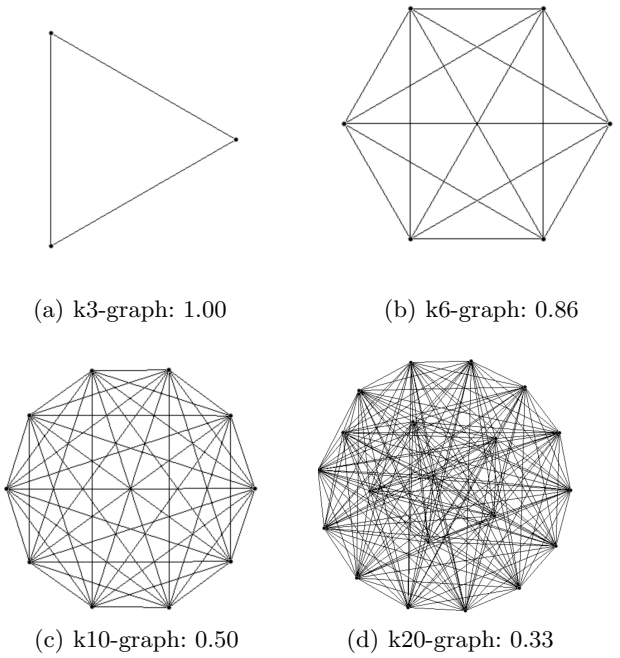
$F = \sum w_i F_i$ {where i=1,2,3...7. $w_i$ is the coefficient in the range (0,1) and $\sum w_i = 1$.}

We consider the measurements for minimum values and the holistic distribution. For the minimum values, the minimum crossing angle, the minimum distance between vertices and the minimum distance from vertices to edges are calculated. For the holistic distribution, the average angle for all the vertices, the average distance of edge repulsion, the number of pairs of points when the distance between them is short and the number of point in the range of edge repulsion will be calculated.
As for angles, We regard the angles greater than sixty degrees as good enough. I calculate the percentage of angles on 60 sixty degrees to determine the extent. As for distance, I use the idea of the desirable distance between vertices which is $k$ from our layout algorithm to calculate the the good percentage of distance.

In other words, the distance larger than $k$ is seen as good. Threshold is also for the distance measurements, which equals to half $k$. We regards the distances less than threshold as bad. So in calculation of $F_4 and F_5$, we subtract the bad percentage to get the good percentage.
All the $F_1, F_2 ... and F_7$ are percentages in range (0,1). Since $F_1, F_2 ... and F_7$ have already keep a balance by operations, we set $w1 = w2 = w3 = w4 = w5 = w7 = 1/7$. Finally, we get the value $F$ which is in range $(0, 1)$. The larger $F$ is, the better the drawing will be. We regards the layouts with $F > 0.5$ are good enough, while $F < 0.3$ is not very good.
This evaluate function evaluates the goodness of layouts, rather than the graphs itself. When the graphs become more complicated, it does not mean the evaluate function result will definitely decrease. Although decreasing is common with increasing graph size, it is due to the difficulty to generate a good layout for large graphs. In our evaluating processes, we has considered the complexity of graphs by introducing the desirable distance between vertices $k$. Therefore, it is properly for evaluating different sizes of graphs.

### 4.2 Drawings with Evaluation Results

The following shows some output drawings and its respective evaluation value.



(a) k3-graph: 1.00 (b) k6-graph: 0.86

(c) k10-graph: 0.50 (d) k20-graph: 0.33

**Fig. 9.** Evaluation Function Result

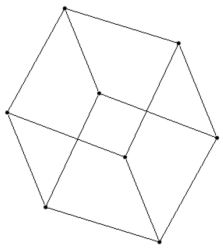


**Fig. 10.** cube: 0.95

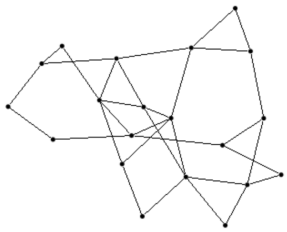


**Fig. 11.** 20 vertices 32 edges: 0.58

### 4.3 Evaluation

For testing the actual performance of our algorithm, we have collected many different kinds of undirected graph to build our data set. We have seen the drawings and their properties of minimum angle, minimum distance from vertices to edges and their evaluating values for many graphs. Most drawings are observed satisfying. In order to prove that our new algorithm can indeed meet our aesthetic standards and conduct a persuasive test, we will make a comparison with other visualisation tool force-directed algorithms to evaluate the performance of our algorithm.

Graphviz is a open source software for doing graph visualization [**?**]. This tool can read descriptions of graphs in a text language, then visualise the graph into diagrams or output the diagrams' description in its useful formats, such as images for web page [**?**]. Apart from its various features for concrete diagrams, such as options for colors and styles, it has a powerful internal layout algorithm which is also based on force model. Therefore, we use the output from graphviz as a comparison to make our evaluation more persuasive.

From the data sets, I randomly chose several undirected graphs of multiple edges graphs with different sizes as the samples, from 20 vertices to 200 vertices. Six properties are taken into account:

- The minimum angle (among edge crossing angles and vertices incident angles),
- The average of minimum angles for each vertex (vertices with no an angle should be ignored),
- The number of vertices which are affected by edge field,
- The minimum angle distance between vertices to edges
- The average of distance between vertices to edges (only for those vertices in edge field)
- The value of evaluation function

The data for these six properties has been calculated and collected, which is showed in following figures. In figures, the blue line is our algorithm output while

the orange line is the graphviz interal algorithm output. Obviously, our algorithm performs much better in the first feature of the minimum angle, especially when the graph size is less than 100. As for the second feature of the average angle for each vertex, there is not significant difference between our algorithm and graphviz algorithm. Therefore, from the pespective on angle measurements, our algorithm seems better in preventing the smallest angle.

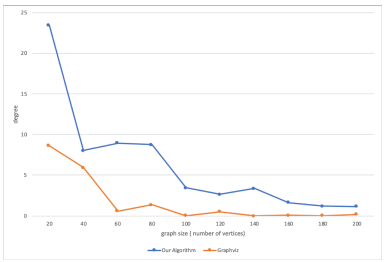



**Fig. 12.** The minimum crossing angle

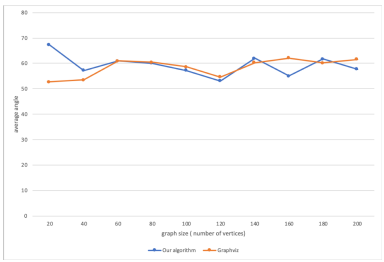



**Fig. 13.** The average of minimum angles for each vertex

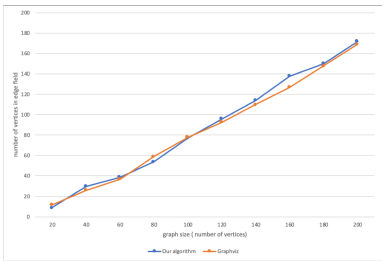



**Fig. 14.** The number of vertices which are affected by edge field

For the measurements of relations between edges and vertices, two algorithms shows a similar result on the number of vertices affected by edge field (in the edge repulsive range), but display significant improvements on other two features, especially for the minimum distance between edges and vertices. Therefore, from the perspective on the relations of edges and vertices, our algorithm also seems better in avoiding vertices and edges stay too close.

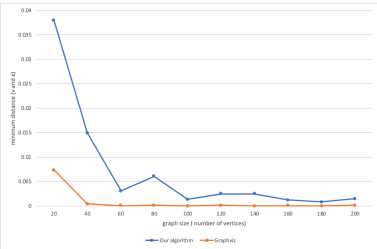



**Fig. 15.** The minimum angle distance between vertices to edges

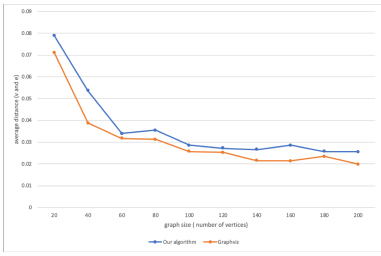



**Fig. 16.** The average of distance between vertices to edges

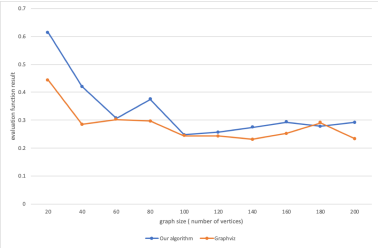



**Fig. 17.** The value of evaluation function

This figure shows the results of our evaluation function. By comparison, our algorithms perform well in most cases. So, our algorithm is indeed effective on our aesthetic criteria.

### Running Time

Assume the graph $G = (V, E)$, we can calculate the time complexity of our algorithm according to the pseudocode in chapter 3.4. Since the iterations count is a constant, the time of the algorithm is $O(|V| * (|V| + |E|))$. The following diagram shows the time in seconds for a set of graphs whose $|V|$ and $|E|$ has linear relationship.

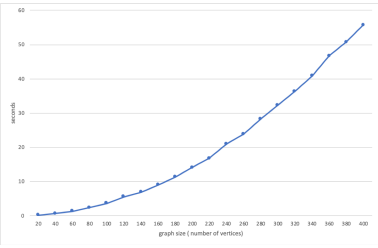
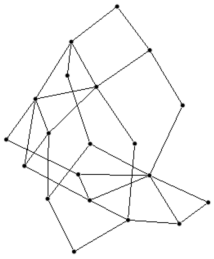
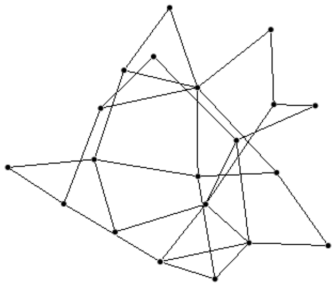



**Fig. 18.** The Time

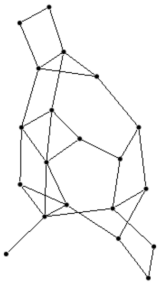
### 4.4 Drawings



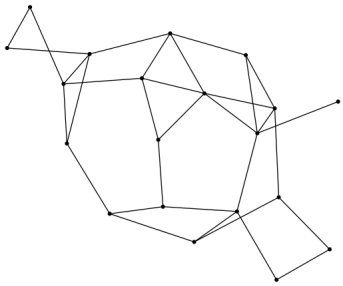
(a) Our algorithm of graph 1



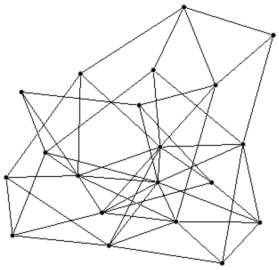
(b) Graphviz algorithm of graph 1



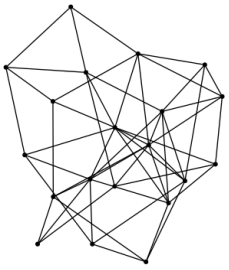
(c) Our algorithm of graph 2



(d) Graphviz algorithm of graph 2



(e) Our algorithm of graph 3



(f) Graphviz algorithm of graph 3

**Fig. 19.** Compare by using same graph

Here we display some diagrams. Each two pictures in a line represents the same graph instance. The left ones are the output of our algorithm, while the right ones are the output of the force-directed algorithm in Graphviz.
We can see some difference between two graph diagrams. For example, figure (d) has a vertex very close to its adjacent edge so the angle is small, but figure (c) eliminates this situation. However, for some graphs, it is not very simple to compare which one is better with the naked eye. Therefore, those similar and more complex graph's output needs to be analyzed from the quantitative perspective that we mentioned previously.
In this project, the simple but effective algorithm for visualising undirected graphs based on the force model has been introduced. The special idea of our method is the introduction of new effects from edge fields, which refers to edge repulsion. This innovative idea is easy to understand but can lead a great contribution towards our goals and aesthetics criteria. We research on related achievements in spring models and gravitational models, which inspired us to build a new virtual force model.
The algorithm is easy to understand and consistent with human perception. The result is of high quality and on accord with our aesthetic criteria.