Research Plan Force-directed Graph Drawing

Michiel van Heusden, 4173309
Maurits van der Veen, 4167287
Kevin Oosterlaak, 4012372 Jesse Ceelen, 4061837

January 27, 2016

Contents

1 intro

Data requires often visualization before people understand what it means. This visualization is done with graphs and charts. This research focusses on the drawing methods of graphs. These graphs consist of objects and the relations between eachother. In graph theory the objects are called vertices and their relations edges. [1] Generating readable graphs becomes difficult according to the amount of vertices and edges. To generate these correctly many algorithms were developed. One family of functions to do this is called *Force Directed Graph Drawing.* The idea behind force directed graph drawing is to use physics based algorithms to calculate the positions of each correctly. In this research a few algorithms wil be examined and how correctly they can generate a graph. Not only will we compare different algorithms, we will also inspect how some algorithm-specific constants influence the results.

1.1 The Algorithms

The method used for each of the tests is the same, except for the way the forces are calculated. Said method is for each vertex to iterate over the other vertices and calculate the repulsive force between them. Then the same is done for the attractive force, but instead of iterating over vertices it iterates over all vertices connected to the current vertex by an edge. All of these forces are added together and applied to the vertex.

The above is applied to every vertex, and after a sufficiently large amount of iterations the graph should have converged to a stable configuration.

1.1.1 Hooke-Coulomb

The Hooke-Coulomb algorithm is not named after its inventors, but after the laws it follows. The attractive forces are calculated using Hooke's Spring Law. The repulsive ones are based on particle physics. To be more precise, they are calculated using Coulomb's law of charged particles.

Hooke's Law can be used to calculate the distance a spring extends when a certain force is applied to it. However, this same law can be used to calculate the required force for a spring to be extended a certain distance. The law itself is rather simple:

$$F = c\delta X \tag{1}$$

Here c is a constant that defines the stiffness of the spring, it is also known as te *spring constant.* The change in length is δX, whilst F is the required force.

Coulomb's law was first published by the French physicist Charles Augustin de Coulomb in 1785. [2] Though it was first proposed for charged metal balls, it holds for charged particles. In his book Coulomb states for two equally electrified metal balls "...the repulsive force that the two balls ...exert on each other, follows the inverse proportion of the square of the distance". This means that the repulsive force grows quadratic the closer the balls are. Coulomb's law can be written down as an equation as well:

$$F = k_e \frac{q_1 q_2}{r^2} \tag{2}$$

In this equation q_1 and q_2 are the charges of the particles, r is the distance between them and k_e is Coulomb's Constant.[1]

Changing Hooke's Law to make it useful for the calculation of forces is rather straightforeward: Each edge corresponds to a spring, where the vertices are objects connected by springs. Thus we can state a rest length l_0 for each edge, for easy calculations we have decided on $l_0 = 1$. Using the ideal length – which can be seen as the starting length of the spring – it is possible to calculate δX:

$$\delta X = l_d - l_0 \tag{3}$$

In this formula l_d is the distance between the two connected vertices. The resulting formula for the attractive forces between vertices i and j will thus be:

$$F_a(i,j) = w_a(l_d - l_0)\hat{r}_{i,j} \tag{4}$$

In this formula the constant w_a is used instead of c for future ease of distinguishing between the constants used in various formulas. Not only could w_a be seen as the stiffness of the spring, it could also be seen as the weight of the attractive force, hence the new name. Thus it can be used to control how much influence the spring has on the vertex. $\hat{r}_{i,j}$ is the normalized vector from vector i to vector j, calculated as $\hat{r}_{i,j} = \vec{r_{i,j}}/|\vec{r_{i,j}}|$ with $\vec{r_{i,j}}$ the vector from vertex i to vertex j and $|\vec{r_{i,j}}|$ the length of said vector.

Rewriting Coulomb's law to be useful in our research requires a bit more effort. However, it results in the very simplified equation below.

This equation is the result of some assumptions. First off, the charge of both the particles is assumed to be 1. Furthermore, since more control over the strength of this force is desired Coulomb's constant can be merged with a weight constant w_r

$$F_r(i,j) = w_r \frac{\hat{r}_{j,i}}{|\vec{r_{j,i}}|^2} \tag{5}$$

In this equation $\hat{r}_{j,i}$ is the normalized vector from vertex j to vertex i. This order is chosen like this to ensure that the force applied to vertex i is calculated instead of the force on vertex j.

[1] $k_e = 8.987551787x10^9 Nm^2C^{-2}$

1.1.2 Fruchterman Reingold

The Fruchterman Reingold algorithm on the otherhand is named after its inventors. [3] For the method Fruchterman and Reingold proposed there are only two principles for graph drawing. Namely that connected verticed should be drawn near eachother and vertices should not be drawn *too* close to eachother. This algorithm can be seen as based off of the idea of ideally distributed vertices. It does not focus so much on each individual node as well as how far each node should be apart from eachother. Like the Hooke Coulomb algorithm it is based off of particle physics: Particles that are close to eachother are repulsive, whilst at a distance they are attracted to eachother. This algorithm uses a few input variables of which three will be changed:

- The allowed surface area W
- The optimal distance between vertices k
- Attractive weight α
- Repulsive weight r

During our testing all but the allowed surface area will be changed. Changing the surface area will only change the scale of the vertices. The influence the other values have to how the graph will look becomes clear from their usage. Both α and r are weights to the forces. Thus directly changing how much influence the attractive and repulsive forces have on the vertices. k on the other hand will not be directly changed, but by changing constant c in its formula.

$$k = c\sqrt{\frac{area}{number of vertices}} \tag{6}$$

The constant k plays an important part in calculation of both the attractive and repulsive forces. These can be calculated using formulas 7 and 8 respectively. In both of these equations d is the distance between the two vertices on which the forces apply.

$$f_a(d) = \alpha\frac{d^2}{k} \tag{7}$$

$$f_r(d) = -r\frac{k^2}{d} \tag{8}$$

However when blindly applying these forces a good looking graph will never converge. Therefore a cooling function is required. This cooling function indicates how much the maximum change is per vertex. As the name suggests, this function returns a lower value every iteration. Thanks to this function eventually the change in the graph per iteration will diminish.

1.1.3 Eades

One person who has made many contributions to the research of force directed graph drawing is Peter Eades. This algorithm is very similar to the Hooke-Coulomb one. It uses springs and Coulomb's law to calculate the new locations of the forces. The difference between them is the type of springs used. Those used in the Hooke-Coulomb algorithm were Hookian – or linear – springs. The force they exert grows linearly, while the ones used in Eades' algorithm where *logartimic* springs. Whenever Hookian springs get extended by a certain amount, no matter their length, results in the same amount of force. Logaritmic springs on the otherhand require more change the longer they are for the same force. The equation for the attractive force between vertices i and j in Eades' algorithm is:

$$f_a(i,j) = c_1 log(\frac{d}{c_2}) \tag{9}$$

Ofcourse there is only an attraction when the vertices are connected. This formula requires two constant parameters c_1 and c_2, these can be used to adjust the output values. The variable d is the length of the spring. [4]

2 Problem Discription

The aforementioned algorithms are meant to make graphs more readable, giving them a clear organized structure. Since readability is a rather vague and abstract measurement criteria in itself, there have to be concrete measurable properties that represent readability. The following properties can be used to express readability: [5]

- Uniformity of vertex distribution
- Uniformity of edge lengths
- Amount of edge crossings

The uniformity of vertex distribution can also be seen as the density of the vertices. Every vertex has an area with a fixed radius where the density can be calculated by counting the amount of vertices in that area. For a readable graph it is important that the densities of all the vertices are similar. This characteristic of the uniformity of vertex distribution is measured with the standard deviation of the densities. The smaller the deviation, the more readable the graph is.

With the uniformity of edge lengths, it is important that the lengths of all the edges are similar. A graph with a high variety of edge lengths is generally harder to read than a graph with edges of equal lengths. This characteristic can be measured with the standard deviation of the edge lengths. If there are more edges with different lengths, the standard deviation will be bigger, thus making the graph hard to read. Therefore, a high quality graph that is easy to read has a small deviation of edge lengths.

For a good graph drawing, it is desirable to have as little amount of edge crossings as possible or none at all even. Having a lot of edges that cross each other makes the graph look more chaotic and harder to read. With that in mind, it obviously is preferred to not have them at all as it makes the graph more readable.

Getting the right properties for these characteristics to make the graph readable is what all the aforementioned algorithms strive for. These algorithms calculate the attractive and repulsive forces differently and make use of them with different equations. This results in getting different graphs with each algorithm with different properties for the three characteristics. Finding out which algorithm comes closest to the desired readablity properties is what this research is meant to focus on. When testing these three algorithms, the uniformity of both the vertex distribution and the edge lengths and the amount of edge crossings will be measured and compared. By doing these tests multiple times, we can conclude which algorithm gives the highest quality graph from the results of these tests. Which brings us to our research question: Which force directed graph drawing algorithm yields the highest quality graphs?

3 The Experiment

4 Result Data

4.1 Tabellen

4.2 Grafieken

4.3 Toelichting

4.4 Statistische Hypothese

4.5 De uitwerking hiervan

5 Conclusion & Discussion

6 Reflection

References

[1] John Adrian Bondy and Uppaluri Siva Ramachandra Murty. *Graph theory with applications*, volume 290. Macmillan London, 1976.

[2] Charles Augustin Coulomb. *Premier-[troisième] mémoire sur l'electricité et le magnétisme.* Académie Royale des sciences, 1785.

[3] Thomas MJ Fruchterman and Edward M Reingold. Graph drawing by force-directed placement. *Softw., Pract. Exper.*, 21(11):1129–1164, 1991.

[4] P Eades. A heuristic for graph drawing. congressus numerantium, 42, 149–160. *Elsevier, BV (2005). Scopus. Retrieved August*, 17:2005, 1984.

[5] Stephen G. Kobourov. Spring embedders and force directed graph drawing algorithms. *CoRR*, abs/1201.3011, 2012.