Volume xx (200y), Number z, pp. 1–12

Addressing zero angular resolution in force-directed graph drawing using repulsive forces between nodes and edges

D. Pantůček[†1,2]

[1] Prague College, Czech Republic
[2]School of Computing, Teesside University, Middlesbrough TS1 3BA

Abstract

Visualizing graph structures has its applications in various fields of research. Good readability of a graph is essential for conveying the desired information. In the images depicting graphs, features such as small angles between edges or node images overlapping edges reduce the readability.
This article presents a solution reducing these unwanted features by extending the standard force-directed "spring" model with a repulsive force between edges and nodes not connected to these edges. Apart from the new force, the extended sprint model behaves as the regular spring model.
An evaluation criteria based upon statistical analysis of edges between drawn nodes is also presented, and it can be used for evaluating any graph visualizations where similar problematic features might emerge.
Based upon the evaluation criteria the presented solution improves the situation for more than 99% of test cases using a peer-reviewed testing data set. The improvement is small but consistent over the whole testing data set.

CCS Concepts
•***Human-centered computing*** → *Graph drawings; Visualization design and evaluation methods;*

1. Introduction

Graph theory is the study of mathematical structures consisting of nodes (vertices) and edges (connections). It finds applications in a wide area of research across many fields including biology, social studies and information systems research. When modeling systems with graphs, it is often useful to visualize these systems using two-dimensional embedding where nodes are represented as dots (or other shapes such as images of the objects they represent) and edges are represented as lines [BGL*97], [ST04], [GFV13].

There are several approaches to graph drawing. A recent survey [WT17] discusses methods ranging from processes that derive the positions of graph nodes from symmetrical structures like orthogonal grids or co-centric circles to general purpose algorithms [BBDW17] such as the spring model which uses a force-directed approach in which numerical simulations of physics-like forces are used to compute the positions of the nodes based typically on repulsive forces between unconnected nodes, attractive forces along the edges and possibly other – for example magnetic field-like – forces. A common example of a force-directed graph drawing algorithm is the "spring model" [Tut63], which models edges as stretchable springs (hence the name) and nodes as objects with the same electric potential. For sparse graphs, this algorithm yields visually appealing results – as confirmed by [VDBG*00] evaluation – and is very simple to implement. It is often used for visualizing informational networks or dependency graphs in software engineering.

General-purpose layout algorithms [BBDW17] deal with positioning nodes with respect to each other and do not take into account relative position of nodes to edges or relative positions between edges close to each other. This aspect also means that the angles between edges drawn from one node can be arbitrarily small. This phenomenon is called zero angular resolution of studied algorithms – because the lower angle limit is zero.

As the spring model is an instance of a force-directed graph drawing algorithms class, which fall into the general-purpose layout algorithms, it suffers from the same problem. [LY12] propose solution based on a different approach to force-directed algorithms. It uses only repulsive forces between the edges modeled as conductive wires with electric potential. Other forces between the nodes are not considered.

Instead of trying to overcome the zero angular resolution of traditional simple force-directed graph drawing algorithms, the present research extends the simple spring model with a newly defined force that aims to counter the angles between edges that are

† R. Honzik

too small. There is no definition of what angle sizes are perceived as "small" as this may be very subjective, therefore an exact and measurable criteria is developed based on [VDBG*00] and [LY12] with respect to how close to the ideal angle distribution the sizes get. With the new force definition and the evaluation criteria specified, this work proves that it is possible to address the problems arising from zero angular resolution of the simple spring model by adding a repulsive force between nodes and edges unconnected to such nodes.

This article proposes an extension to the simple spring model algorithm by adding a new force to the simulation and evaluates its improvement to the resulting images with respect to angular resolution. In the first section the simple spring model is described, the second section lays out the evaluation criteria for measuring the angular resolution artifacts, in the third section an extension to the simple spring model is presented. In the fourth section, the testing data set based on [VDBG*00] is presented and the testing process is summarized. And in the last section the results are discussed. Lastly, conclusions and possible research suggestions are presented.

2. The simple spring model

For the purpose of this project, the simple spring model can be described in terms of repulsive forces between all nodes and attractive forces between nodes connected by an edge.

2.1. Forces magnitude

The repulsive forces are modeled similarly to the electrostatic forces and for the purpose of the simple spring model they shrink with distance. For any given node A and B in an arbitrary graph, the size of repulsive force $\vec{\mathbf{F}}_{B-A}$ between nodes A and B is calculated as:

$$|\vec{\mathbf{F}}_{A-B}| = \frac{c_r}{|AB|} \tag{1}$$

The constant c_r gives the force size at unit distance and $|AB|$ is the distance between nodes A and B.

Sometimes, forces more resembling real electrostatic force are used. In such cases the force magnitude shrinks with the square of the distance:

$$|\vec{\mathbf{F}}_{A-B}| = \frac{c_r}{|AB|^2} \tag{2}$$

For the purpose of this project, the force magnitude specified in equation 1 is used. This study focuses on angles between edges and not their lengths. There were two reasons for using this simplification. Using the chosen approach simplifies calculations of the repulsive forces between nodes and lowers its computational cost. In addition to this, it similarly simplifies the newly defined repulsive forces in section 4 The extended spring model.

The attractive forces are modeled as an ideal spring. For a given connected pair of nodes A and B, the size of the attractive force between the two nodes is calculated as:

$$|\vec{\mathbf{F}}_{A+B}| = c_a|AB| \tag{3}$$

The constant c_a gives the attractive force size at unit distance.

The constants c_r and c_a are chosen arbitrarily to affect the lengths of edges in the resulting image. Given just two nodes, connected by an edge, their distance, after the simulation converges and the attractive and repulsive forces come to equilibrium, becomes:

$$|AB| = \sqrt{\frac{c_r}{c_a}} \tag{4}$$

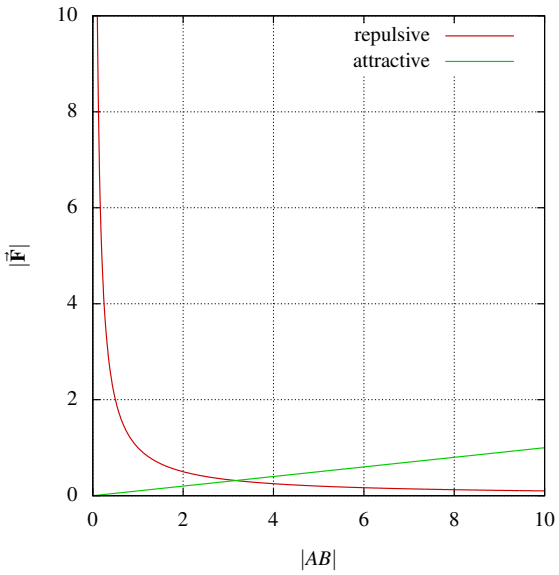

Figure 1: *Equilibrium between repulsive and attractive forces size between two nodes connected by an edge.*

In this project, values of $c_r = 1$ and $c_a = \frac{1}{10}$ are used, therefore the distance between two nodes connected by an edge converges to $|AB| = \sqrt{10}$. The equilibrium between attractive and repulsive forces size can be seen in figure 1. For studying angles between edges, the actual distances are not relevant and these values are chosen arbitrarily. Changing them should not affect the evaluation results, only the image size of graph drawings generated.

2.2. Forces direction

The repulsive forces between two nodes, as shown in figure 2, are always in the opposite direction of each other. Extending the equa-

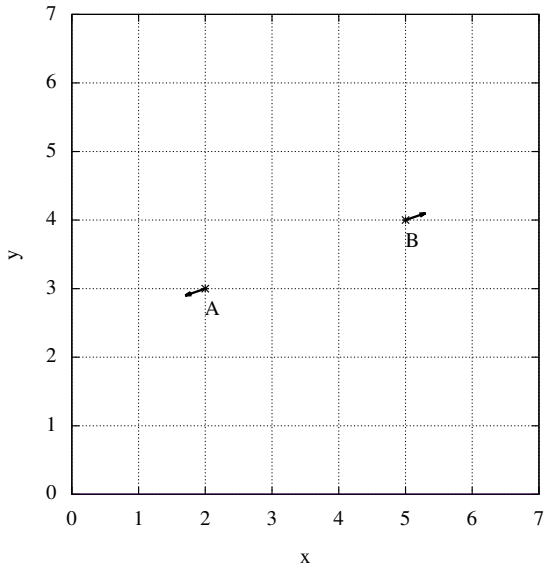

Figure 2: *Repulsive forces between nodes.*

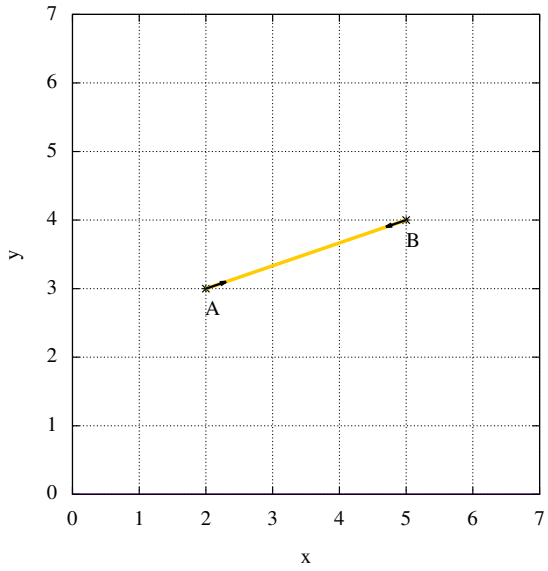

Figure 3: *Attractive forces between connected nodes.*

tion 1, the formulas for repulsive forces between nodes A and B affecting these nodes become:

$$\begin{aligned}\vec{\mathbf{F}}_{A-B} &= \frac{c_r}{|AB|}\frac{\overrightarrow{BA}}{|AB|}\\ \vec{\mathbf{F}}_{B-A} &= \frac{c_r}{|AB|}\frac{\overrightarrow{AB}}{|AB|}\end{aligned} \tag{5}$$

$\vec{\mathbf{F}}_{A-B}$ represents the repulsive force between nodes A and B affecting node A, $\vec{\mathbf{F}}_{B-A}$ is analogous.

As discussed in subsection 2.1 Forces magnitude, the force magnitude shrinks with distance. This has an additional benefit when calculating the vector of repulsive force between nodes. The force $\vec{\mathbf{F}}_{A-B}$ can be expressed as:

$$\vec{\mathbf{F}}_{A-B} = \frac{c_r\overrightarrow{BA}}{|AB|^2} \tag{6}$$

Calculating the square of vector length has smaller computational cost, as there is no need to calculate the square root, only sum of squares is required to calculate in the denominator:

$$\begin{aligned}\overrightarrow{AB} &= \left(\overrightarrow{AB}_x, \overrightarrow{AB}_y\right)\\ |AB| &= \sqrt{\overrightarrow{AB}_x^2 + \overrightarrow{AB}_y^2}\\ |AB|^2 &= \overrightarrow{AB}_x^2 + \overrightarrow{AB}_y^2\end{aligned} \tag{7}$$

Similarly, the attractive forces along the modeled spring, as seen in figure 3, can be calculated – including the direction – by extending equation 3:

$$\begin{aligned}\vec{\mathbf{F}}_{A+B} &= c_a\overrightarrow{AB}\\ \vec{\mathbf{F}}_{B+A} &= c_a\overrightarrow{BA}\end{aligned} \tag{8}$$

Where $\vec{\mathbf{F}}_{A+B}$ represents the attractive force between the connected nodes A and B affecting the node A and the similarly defined $\vec{\mathbf{F}}_{B+A}$ is inverse to it.

2.3. Forces application

For each frame of the simulation, the forces affecting all nodes are calculated first and for each node in the graph a total force is calculated as a sum of all forces affecting it:

$$\vec{\mathbf{F}}_A = \sum_{i=1}^{N}\left(\vec{\mathbf{F}}_{A+i} + \vec{\mathbf{F}}_{A-i}\right) \tag{9}$$

Where $\vec{\mathbf{F}}_{A+i}$ is the attractive force between node A and i-th node in the graph and $\vec{\mathbf{F}}_{A-i}$ is the repulsive force between node A and i-th node in the graph. Attractive forces between unconnected nodes are zero in this equation as well as any force the node exerts upon itself.

Then the total forces are applied to all nodes in the graph by adding the total force vector to given node. For each node A a new position A' is calculated:

$$A' = A + \vec{\mathbf{F}}_A \tag{10}$$

Should the force be applied exactly this way, the immediate displacement may put the node in a position where the total force affecting it may be bigger than it is in the previous state. To counter this problem, an arbitrary scale coefficient $c_f < 1$ is chosen to make the simulation run smoother and closer from discrete to continuous physics-like movement:

$$\begin{aligned} A' &= A + c_f \vec{\mathbf{F}}_A \\ c_f &= \frac{1}{2} \end{aligned} \tag{11}$$

The force scale coefficient for the simple spring model does not need to be too small as the simulation always converges even for relatively large values. Therefore the value $c_f = \frac{1}{2}$ is chosen for all simulations using the simple spring model. Lower values show no improvement in the results with respect to chosen evaluation criteria.

2.4. Simulation convergence

The convergence of the simulation is determined by calculating the total displacement between two successive frames and comparing it to a predefined threshold value. In order to do so, the sum of all total force sizes affecting all nodes is calculated:

$$\Delta = \sum_{i=1}^{N} |\vec{\mathbf{F}}_i| \tag{12}$$

If the value $\Delta < \Delta_{threshold}$ for some small $\Delta_{threshold}$, the simulation has converged and is stopped. In this case, the number of frames to converge is known as well as the total time to run the simulation. It is also possible to evaluate the resulting drawing according to the evaluation criteria.

For this project, value $\Delta_{threshold} = 0.01$ is chosen for simple spring model. This value is chosen empirically based on the test results and no significant improvements have been observed for smaller values of $\Delta_{threshold}$.

2.5. Complexity and cost

For an arbitrary graph with N number of nodes there exists a repulsive force between each pair of the N nodes, for a total of $N(N-1)$ forces. This means the asymptotic time complexity of calculating the forces is $O(n^2)$.

Similarly, if E is the number of edges in a graph, there are E steps required to calculate the attractive forces in the graph. The number of edges for a connected graph is in the range:

$$E \in \left\langle N-1, \frac{N(N-1)}{2} \right\rangle \tag{13}$$

Therefore the total number of steps required to calculate attractive and repulsive forces between nodes is $N(N-1)+E$ which gives maximum total number of steps for a fully connected graph:

$$\begin{aligned} S_{max} &= N(N-1) + E_{max} \\ S_{max} &= N(N-1) + \frac{N(N-1)}{2} \\ S_{max} &= \frac{3}{2} N(N-1) \end{aligned} \tag{14}$$

It is possible to calculate the repulsive forces between nodes in $\frac{N(N-1)}{2}$ steps by calculating the force magnitude and direction between each pair of nodes only for one node in each pair and then negating the resulting vector for for the other node in the pair. This is an optimization that is omitted in this calculation as it makes strictly functional implementation (that is without any programming side effects) impossible.

The asymptotic time complexity of calculating both attractive and repulsive forces in the simple spring model is therefore also $O(n^2)$.

In the case the graph is not connected – that means it contains subgraphs comprised of subsets of nodes where there is no edge connecting a node from one set to another – some repulsive forces are never canceled out by attractive forces along the edges and therefore such simulation using the calculated forces never reaches equilibrium.

As there is some arbitrarily small $\Delta_{threshold}$ specified, the simulation would eventually reach the point where it could be labeled as converged even for a graphs that contain disconnected sub-graphs. In such case the resulting graph drawing would get arbitrarily large and there are currently no known practical applications for this approach.

Computational cost of calculating both the repulsive and attractive forces between nodes can be seen in table 1. Minimum and maximum number of steps required to calculate all the forces for graph with given number of nodes are also given.

	N-N	**N+N**
Divisions	1	1
Multiplications	4	4
Additions	1	1
Subtractions	2	2
Minimum steps	$\frac{N(N-1)}{2}$	$N-1$
Maximum steps	$\frac{N(N-1)}{2}$	$\frac{N(N-1)}{2}$

Table 1: *The computational cost of calculating all repulsive and attractive forces between all nodes of the graph.* ***N-N*** *represents the repulsive force between nodes and* ***N+N*** *is the attractive force between nodes connected by an edge.*

It has been empirically shown that the number of steps required for the simulation to converge grows linearly with the number of nodes [VDBG*00].

The total asymptotic complexity of running the algorithm until convergence is therefore $O(n^3)$.

3. Evaluation criteria

The zero angular resolution of a simple spring model may cause the algorithm to produce edges holding very close angles as studied by [LY12]. Even in the experimental study [VDBG*00] there is no rigorous criteria defined for measuring this phenomenon.

The evaluation criteria analyzes a graph in three steps.

Firstly, for all N nodes in the graph, an ideal angle between edges connected to each node is calculated. As the whole circle around the node i is represented by an angle 2π, dividing this angle evenly by the number of edges E_i connected to this node yields the ideal angle between such edges:

$$\begin{aligned} \alpha_i &= \frac{2\pi}{E_i} \\ i &\in \langle 1;N \rangle \end{aligned} \tag{15}$$

Secondly, for all nodes, actual angles between edges connected to each node are measured, each measured angle is labeled $\beta_{i,j}$ where $j \in \langle 1;E_i \rangle$. The ratio between measured and ideal angle is then:

$$r_{i,j} = \frac{\beta_{i,j}}{\alpha_i} \tag{16}$$

Using this ratio in place of real angles allows for analyzing these ratios over the whole graph regardless of the number of edges around each node. The ratio can be seen as "normalized angle".

Thirdly, performing a statistical analysis of the ratios over the whole graph describes how close to an ideal angle are all angles in the graph. The statistical property used for this is statistical variance.

Lemma: The average value $\bar{r}$ of the ratios between measured and ideal angles over the whole graph is always one.

$$\bar{r} = 1 \tag{17}$$

This lemma allows calculating directly the variance without the need for calculating average value. Proof of this lemma is as follows:

$$\bar{r} = \frac{1}{N} \sum_{i=1}^{N} \frac{1}{E_i} \sum_{j=1}^{E_i} \frac{\beta_{i,j}}{\alpha_i} \tag{18}$$

Substitute for the ideal α_i:

$$\bar{r} = \frac{1}{N} \sum_{i=1}^{N} \frac{1}{E_i} \sum_{j=1}^{E_i} \frac{\beta_{i,j} E_i}{2\pi} \tag{19}$$

Factor out $\frac{E_i}{2\pi}$ from the inner sum:

$$\begin{aligned} \bar{r} &= \frac{1}{N} \sum_{i=1}^{N} \frac{E_i}{E_i 2\pi} \sum_{j=1}^{E_i} \beta_{i,j} \\ \bar{r} &= \frac{1}{N} \sum_{i=1}^{N} \frac{1}{2\pi} \sum_{j=1}^{E_i} \beta_{i,j} \end{aligned} \tag{20}$$

The sum of all angles around each node is always 2π:

$$\begin{aligned} \sum_{j=1}^{E_i} \beta_{i,j} &= 2\pi \\ \bar{r} &= \frac{1}{N} \sum_{i=1}^{N} \frac{1}{2\pi} 2\pi \\ \bar{r} &= \frac{1}{N} \sum_{i=1}^{N} 1 \end{aligned} \tag{21}$$

Which proves the lemma:

$$\begin{aligned} \bar{r} &= \frac{N}{N} \\ \bar{r} &= 1 \end{aligned} \tag{22}$$

Therefore, calculating the variance of measured to ideal angles ratios is done as follows:

$$\sigma^2(r) = \frac{1}{N} \sum_{i=1}^{N} \frac{1}{E_i} \sum_{j=1}^{E_i} (r_{i,j} - 1)^2 \tag{23}$$

However, a definition better suited for implementation can be derived. First, the ratio is expressed in terms of ideal and measured angles:

$$\sigma^2(r) = \frac{1}{N} \sum_{i=1}^{N} \frac{1}{E_i} \sum_{j=1}^{E_i} \left(\frac{\beta_{i,j}}{\alpha_i} - 1 \right)^2 \tag{24}$$

Then the ideal angle is expressed in terms of number of edges connected to each node and measured angles:

$$\sigma^2(r) = \frac{1}{N} \sum_{i=1}^{N} \frac{1}{E_i} \sum_{j=1}^{E_i} \left(\frac{\beta_{i,j} E_i}{2\pi} - 1 \right)^2 \tag{25}$$

This definition can be directly implemented in the software and calculates the variance of ratios of measured angles to ideal angles over the whole graph directly from measured angles and graph definition.

The lower the $\sigma^2(r)$ value, the closer the measured angles in the whole graph are to ideal angles. Any improvement to the simple spring model with respect to countering problems caused by zero angular resolution lowers the value of $\sigma^2(r)$.

4. The extended spring model

To counter the problems associated with zero angular resolution which include angles between edges getting arbitrarily small or nodes crossing edges, described in [LY12], a repulsive force between nodes and edges unconnected to them is proposed.

Introducing a repulsive force between a node and an edge unconnected to it requires expanding the simulated force field to have its source not only at distinct nodes but also on the line representing an edge. An analogy would be to describe the edge as a conductive spring with electric potential the same way a node can be imagined as a point with electric potential. A spring from conductive material should serve as a physical analogy of such edge.

Such force pushes the edges connected to the same node further away from each other and it also repels other nearby nodes away from the edge itself. This is the behavior that counters both problems studied in this project: arbitrarily small angles between edges and node images overlapping with edges.

4.1. Force field

A force field modeled around each edge ensures that any node affected by this field is repelled away from the edge.

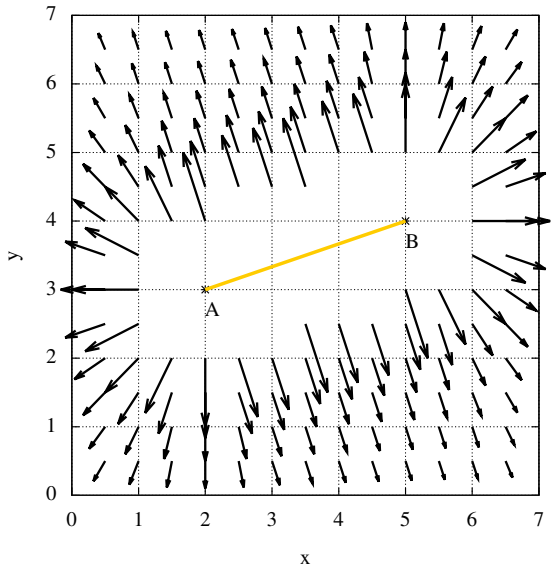

Figure 4: *Visualization of repulsive force field affecting nearby nodes around an edge.*

As seen in figure 4, the repulsive force's direction depends on relative position of an interacting node to given edge. Its scale depends on the distance of such node from this edge. Again, the magnitude of the force is modeled the same way as the magnitude of repulsive forces between nodes in subsection 2.1 Forces magnitude, equation 1, but the distance calculation is more complex in this case.

4.2. Coordinates on an arbitrary axis

In order to calculate the repulsive force between an edge AB and a node P, a relative position of the node to the edge must be calculated. The infinite straight line on which the edge lies is treated as an arbitrary axis l with one node of the edge having coordinate $l = 0$ and the other node being at $l = 1$. A position of node P on this axis is calculated first. It can be calculated as a distance between a line defined by one of the points with normal vector being the vector between the two nodes.

Calculating the distance of a point from a given straight line is simple. The straight line is defined by the following equation:

$$ax + by + c = 0 \tag{26}$$

And the distance of any point in the two-dimensional plane from this line can be computed using:

$$d = \frac{|ax + by + c|}{|\vec{\mathbf{n}}|} \tag{27}$$

Where $\vec{\mathbf{n}}$ is the normal vector of such line. A normal vector can be constructed from the coefficients of the line equation – or the coefficients can be taken from known normal vector as follows:

$$\vec{\mathbf{n}} = (a, b) \tag{28}$$

Therefore the distance can be easily calculated using the following formula:

$$d = \frac{|ax + by + c|}{\sqrt{a^2 + b^2}} \tag{29}$$

By omitting the absolute value from this equation, a distance value with a sign can be calculated. The sign of the distance value represents the semi-plane to which the point belongs with respect to the line:

$$d^* = \frac{ax + by + c}{\sqrt{a^2 + b^2}} \tag{30}$$

A positive sign means the point $[x, y]$ belongs to the same semi-plane as the normal vector is facing towards. A negative sign means it is in the opposite semi-plane. As with regular distance between a point and a line, zero value means the point lies directly on the line.

Calculating the coefficients of the given line equation is performed using known normal vector and a point on given line. Given the following:

$$\begin{aligned} \vec{\mathbf{n}} &= (\vec{\mathbf{n}}_x, \vec{\mathbf{n}}_y) \\ A &= [A_x, A_y] \end{aligned} \tag{31}$$

The coefficients for the line equation then are calculated as:

$$\begin{aligned} a &= \vec{\mathbf{n}}_x \\ b &= \vec{\mathbf{n}}_y \\ c &= -aA_x - bA_y \end{aligned} \tag{32}$$

To calculate the l-axis coordinate for the edge AB, the distance value with sign between a line defined by the point A and normal vector $\overrightarrow{AB}$ and the point P is calculated and divided by the size of vector $\overrightarrow{AB}$ to scale the resulting value to be $l = 1$ at point B. The normal vector used is:

$$\vec{\mathbf{n}} = \overrightarrow{AB} \tag{33}$$

Coefficients a, b and c are calculated as given in equation 32 and the appropriate l-axis coordinate is then calculated as:

$$\begin{aligned} l_p &= \frac{d^*}{|\vec{\mathbf{n}}|} \\ l_P &= \frac{aP_x + bP_y + c}{\sqrt{a^2+b^2}} \cdot \frac{1}{\sqrt{a^2+b^2}} \end{aligned} \tag{34}$$

This can also be simplified to remove the necessity of calculating the square root:

$$l_P = \frac{aP_x + bP_y + c}{a^2+b^2} \tag{35}$$

According to the l_P coordinate, method for calculating the force is chosen.

4.3. Possible force calculation methods

There are three distinct cases that must be addressed. The following choices for the repulsive force between node P and edge AB exist:

$l_P \leq 0$ – it is calculated as a repulsive force between the nodes A and P (point P on figure 5),
$l_P \geq 1$ – it is calculated as a repulsive force between the nodes B and P (point R on figure 5),
$l_P \in (0,1)$ – it is calculated as a repulsive force between the line on which edge AB lies and the node P (point Q on figure 5).

The three distinct cases can be seen in figure 5 superimposed on the three regions where given force calculation must be applied.

From the programming perspective, these are two general cases. First instance is where the force is calculated using the distance between line and point. Second instance is where the force is calculated using the distance from the nearer of the two points defining given edge.

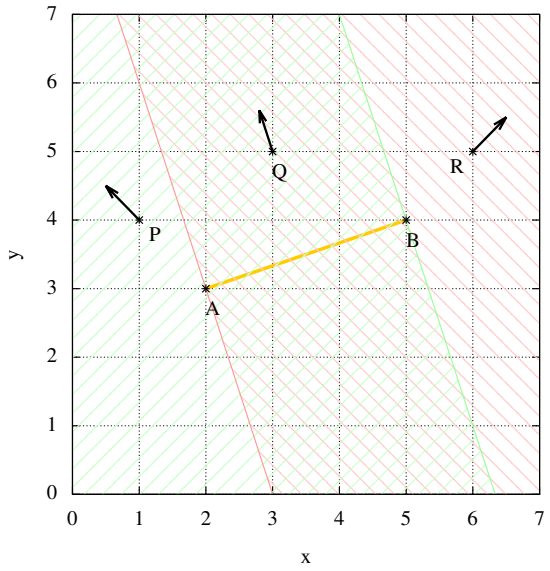

Figure 5: *Visualization of repulsive force affecting three nodes close to an edge in three possible regions with respect to the nodes defining the edge.*

4.4. Repulsive force between a line and a point

In the case the node P has $l_p \in (0,1)$, the distance between the infinite straight line on which the edge AB lies and the point representing node P is used to determine the force magnitude and its direction. The line is specified as in the equation 26 with coefficients calculated similarly to the equation 28.

The normal vector $\vec{\mathbf{n}}$ is obtained by rotating the vector defining the edge by $\frac{\pi}{2}$ counter-clockwise:

$$\begin{aligned} \vec{\mathbf{u}} &= \overrightarrow{AB} \\ \vec{\mathbf{n}} &= (-\vec{\mathbf{u}}_y, \vec{\mathbf{u}}_x) \end{aligned} \tag{36}$$

The distance between the line l and the point P is calculated as:

$$d = \frac{|aP_x + bP_y + c|}{|\vec{\mathbf{n}}|} \tag{37}$$

Then the size of the repulsive force is calculated similarly to calculating the repulsive force between nodes (see equation 1) as:

$$|\vec{\mathbf{F}}_{AB-P}| = \frac{c_r}{d} \tag{38}$$

Where $|\vec{\mathbf{F}}_{AB-P}|$ represents the magnitude of the repulsive force between the edge AB and node P affecting the node P.

The direction of the force is always perpendicular to the edge. Therefore the direction of the force is the same or inverse as of the

calculated normal vector. To calculate the direction, the unit normal vector is calculated first:

$$\vec{\mathbf{n}}_1 = \frac{\vec{\mathbf{n}}}{|\vec{\mathbf{n}}|} \tag{39}$$

A distance value with sign is calculated by removing the absolute value. It is the same approach as used in the equation 35:

$$d^* = \frac{aP_x + bP_y + c}{|\vec{\mathbf{n}}|} \tag{40}$$

If a distance value with a sign is used, the final force vector can be calculated directly from unit normal vector. This is because positive sign means the force vector has the same direction as normal vector that was used and negative sign means it has the opposite direction:

$$\vec{\mathbf{F}}_{AB-P} = \frac{c_r}{d^*} \cdot \vec{\mathbf{n}}_1 \tag{41}$$

This is the vector of repulsive force between edge AB and node P affecting the node P. Calculating the reciprocal force is done similarly as:

$$\vec{\mathbf{F}}_{P-AB} = -\frac{c_r}{d^*} \cdot \vec{\mathbf{n}}_1 \tag{42}$$

Where $\vec{\mathbf{F}}_{P-AB}$ represents the repulsive force between the edge AB and node P affecting the edge AB.

4.5. Force distribution along the edge

The force affecting the edge should exhibit itself exactly on the nearest point on the line representing the edge. This can be greatly simplified by splitting the force at this point to two forces affecting both ends of a given edge by shrinking its magnitude based on the distance from the other end.

If the nearest point on the straight line defined by the two points forming the edge does not lie between them, the distance – and direction – is determined simply by calculating the force against the nearer of the two edge ends. In this case the repulsive force affects effectively only the closer node of given edge and it can be calculated just as the aforementioned repulsive force between two nodes in the equation 5.

For the case where $l_P \in (0,1)$, the force is divided between the two nodes defining the edge by the inverse ratio of the distance from l_P to nodes A and B respectively.

$$\begin{aligned} \vec{\mathbf{F}}_{P-AB(A)} &= (1-l_P) \cdot \vec{\mathbf{F}}_{P-AB} \\ \vec{\mathbf{F}}_{P-AB(B)} &= l_P \cdot \vec{\mathbf{F}}_{P-AB} \end{aligned} \tag{43}$$

The force $\vec{\mathbf{F}}_{P-AB(A)}$ affecting the node A and the force $\vec{\mathbf{F}}_{P-AB(B)}$ affecting the node B are shown in figure 6.

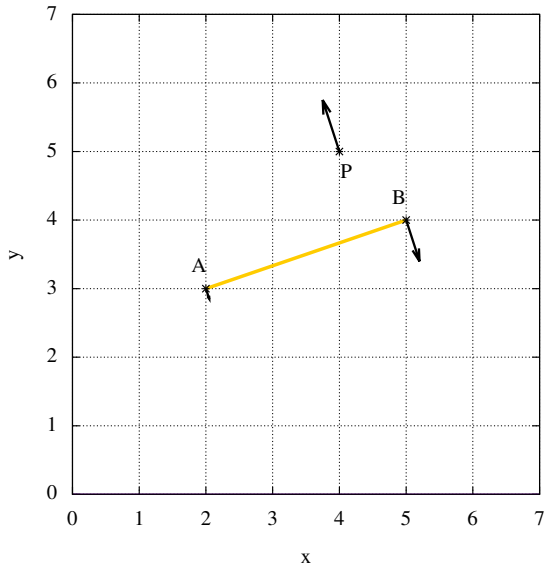

Figure 6: *Visualization of the repulsive force affecting the node close to an edge and respective forces affecting the nodes comprising such edge.*

The sum of magnitudes of the new forces is equal to the magnitude of the original force:

$$|\vec{\mathbf{F}}_{P-AB(A)}| + |\vec{\mathbf{F}}_{P-AB(B)}| = |\vec{\mathbf{F}}_{P-AB}| \tag{44}$$

For $l_P \in (0,1)$ the proof is as follows:

$$\begin{gathered} (1-l_P) \cdot \vec{\mathbf{F}}_{P-AB} + l_P \cdot \vec{\mathbf{F}}_{P-AB} = ((1-l_P)+l_P) \cdot \vec{\mathbf{F}}_{P-AB} \\ ((1-l_P)+l_P) = 1 \end{gathered} \tag{45}$$

For practical implementation, the forces into which the original force gets split are calculated directly using:

$$\begin{aligned} \vec{\mathbf{F}}_{P-AB(A)} &= -\frac{c_r(1-l_P)}{d^*} \cdot \vec{\mathbf{n}}_1 \\ \vec{\mathbf{F}}_{P-AB(B)} &= -\frac{c_r l_P}{d^*} \cdot \vec{\mathbf{n}}_1 \end{aligned} \tag{46}$$

This division of the force translates the edge away from the interacting node P and if $l_P \neq \frac{1}{2}$, rotates the edge around its midpoint as well.

4.6. Application

Newly calculated forces are added to the forces affecting each node. To incorporate this change the equation 9 is extended by adding newly calculated forces to the sum:

$$\vec{\mathbf{F}}_A = \sum_{i=1}^{N} \left(\vec{\mathbf{F}}_{A+i} + \vec{\mathbf{F}}_{A-i} \right) + \sum_{i=1}^{E} \left(\vec{\mathbf{F}}_{i-A} + \sum_{j=1}^{N} \vec{\mathbf{F}}_{j-i(A)} \right) \tag{47}$$

Where $\vec{\mathbf{F}}_A$ is the total force affecting node A, $\vec{\mathbf{F}}_{A+i}$ is the attractive force between node A and i-th node, $\vec{\mathbf{F}}_{A-i}$ is the repulsive force between node A and i-th node, $\vec{\mathbf{F}}_{i-A}$ is the repulsive between the node A and i-th edge affecting node A and $\vec{\mathbf{F}}_{j-i(A)}$ is the repulsive force between i-th edge and j-th node affecting node A.

As the simulation is more complex than in the simple spring model, it takes significantly longer time to fully converge. For all practical purposes, choosing $\Delta_{threshold} = 0.1$ does not have any negative impact on the results for stopping the simulation early.

However, with more force-like interactions in the simulation, higher precision is required to avoid problems of low resolution numerical simulations. With $c_f = 0.1$, all simulations ran smoothly and converged within expected time.

4.7. Complexity

Adding a new force to the spring model requires calculating this force for all nodes for each frame and adding it to the total force affecting nodes.

The number of steps required to calculate the repulsive forces between edges and nodes unconnected to them depends on the number of edges in the graph $E \in \left\langle N-1, \frac{N(N-1)}{2} \right\rangle$. Therefore the number of steps required to calculate the newly added forces is:

$$S \in \left\langle (N-1)\cdot(N-2), \frac{N(N-1)}{2}\cdot(N-2) \right\rangle \tag{48}$$

This has important consequences, as:

$$\begin{aligned} S_{min} &= N^2 - 3N + 3 \\ S_{max} &= \frac{N^3 - 3N^2 + 2N}{2} \end{aligned} \tag{49}$$

The number of steps for the algorithm to converge grows linearly with the number of nodes in the graph – which is confirmed by the experimental results. Adding newly defined force to the model increases the algorithmic complexity between $O(n^3)$ and $O(n^4)$ depending on the number of edges. When working with sparse graphs, for all practical purposes, $O(n^3)$ is an estimation that also fits the experimental results.

Computational cost is also increased compared to the simple spring model. The most expensive function is square root which needs to be calculated when computing the force interaction between line and node. Actual number of distinct operations can be seen in table 2.

	N-E/l	**N-E/d**
Square roots	1	0
Divisions	1	1
Multiplications	6	4
Additions	4	2
Subtractions	2	2
Negations	1	2
Minimum steps	$(n-1)\cdot(n-2)$	
Maximum steps	$\frac{n^2}{2}\cdot(n-2)$	

Table 2: *The computational cost of calculating repulsive forces between nodes and edges.* ***N-E/l*** *represents the case where the interaction is based upon line-point distance and* ***N-E/d*** *shows the cost of calculation based on direct distance between the nearer of the two nodes forming an edge and the affected third node.*

The extended spring model is significantly more computationally expensive than the simple model. This matches the experimental results.

5. The evaluation of the extended spring model

To test the extended spring model, a proof-of-concept implementation was created.

The testing data was obtained from [VDBG*00] which used 11582 graphs gathered from science and engineering applications and additional graphs constructed randomly based on the properties of the collected graphs.

5.1. Implementation

The implementation was created using Racket cross-platform, scheme-based programming environment [Rac17].

Support utilities for running the tests in parallel, converting the original testing data set and analyzing results were implemented as well.

5.2. Testing data set

The testing data set was in a simple text format consisting of two parts listing first the nodes and then listing the edges represented as node pairs.

Set or subset	**Graph count**
Original testing data set	11582
Disconnected graphs	2
Too many nodes (>100)	3
Empty graphs	49
Final testing data set	11528

Table 3: *Erroneous graphs found in the original testing data set and resulting data set graph count.*

In addition to nodes and edges listing, the original format allowed for storing per-node and per-edge weights. This feature was not used in the original testing data set as all such values were zero.

The testing data set was converted to S-expressions based format used by the software developed for this project. The conversion revealed a number of graphs with various kinds of errors in the original data set which are presented in table 3. Erroneous graphs were discarded from the testing data set.

Conversion of the original testing data set was done on an *Intel(R) Core(TM) i7-5600U CPU 2.60GHz*. The data was converted in 68m13s. There were no other errors encountered.

5.3. Evaluation methodology

To assess the improvement of the extended spring model over the simple spring model, the testing data set is processed by both and the results are evaluated with respect to chosen criteria given in section 3 Evaluation criteria.

All graphs in the testing set were processed by the simple spring model first and then the resulting node positions were used as input for the extended spring model. For both parts, the resulting angle ratio variance was calculated after the simulation converged and the time to finish and number of simulation frames were stored as well.

After collecting results from all graphs in the testing data set, a statistical analysis is performed on the complete results. Obtained statistical data is plotted as histogram with respect to number of nodes in graph.

6. Results

Both the simple spring model and the extended version converged for all graphs in the testing set. Running the extended algorithm after the simple one yielded better results for over 99% of these graphs. Only for 108 graphs out of 11518 the results after running the extended algorithm were worse with respect to chosen evaluation criteria. These 108 graphs were not studied further in this project.

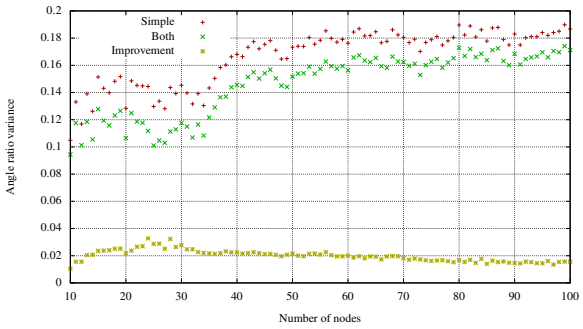

Figure 7: *Averages of angle to ideal angle ratios variance over the whole graph for all graphs in the testing set, grouped by the number of nodes in the graph. The results of running only the simple algorithm and running both algorithms are shown with the improvement added for clarity.*

The averages of the resulting angle ratio variances grouped by the number of nodes in the graph can be seen in figure 7 shown as histogram. The improvement of running both algorithms against running only the simple one is also depicted and although the improvement is less than 10% of the original value, it is consistent over the whole testing set.

The improvement depicted in figure 7 shows slightly decreasing tendency with the increasing number of nodes in testing graphs. As the decrease is really small if approximated with linear function using the least squares fitting method $\left(\text{linear dependence of } \approx (-11.6128 \pm 1.195) \cdot 10^{-05}\right)$, studying this phenomenon would require testing data set of graphs with many more nodes.

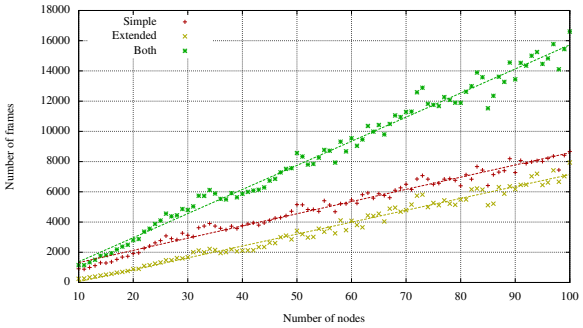

Figure 8: *Averages of the number of frames for both algorithms to converge with respect to the number of nodes in the graph. Both the simple and extended parts of the simulation are shown, as well as the total number of frames for the simulation to finish. Linear approximations are displayed to show that the experimental results match the estimations.*

Running the experiments shows that the average number of frames to converge for both the simple and extended algorithm grow linearly with the number of nodes in a graph. The average number of frames to finish the simulation can be seen in figure 8 and it is shown with respect to the number of nodes in the graph. Also linear approximations using the the least squares fitting method of the Gnuplot software suite [gnu17] are shown.

These results match the expectations from subsection 2.5 Complexity and cost. All simulations converged in less than 20 000 frames. The limit of 100 000 frames set to catch possible errors was never reached.

Unlike the linear growth observed in the number of simulation frames to converge, the computational complexity and also the actual computing cost grow faster with the number of nodes. As given in subsection 2.5 Complexity and cost and subsection 4.7 Complexity, the asymptotic complexity of the simple spring model, based upon required number of steps in the equation 14, is $O(n^3)$ and the extended spring model's asymptotic complexity, based upon the minimum and maximum number of steps given in equation 49, falls between $O(n^3)$ and $O(n^4)$. As the graphs in the testing set are sparse, the $O(n^3)$ approximation of running times fits the results sufficiently.

The average times to converge, with respect to the number of nodes in the graph, are shown in figure 9. The increased algorithmic

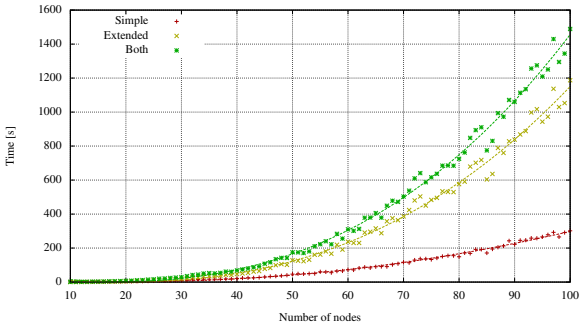

Figure 9: *Average times for the simulations to converge for both variants of the algorithm with respect to the number of nodes in a graph. Cubic estimations are added as dashed curves to show how the experimental results match the predictions.*

complexity and added computing cost is reflected in much longer times needed to run the simulation using the extended algorithm.

Simulations were run on an *Intel(R) Xeon(R) CPU E5-2620 v4 @ 3.00GHz*. There were 8 parallel tasks running using simple partitioning scheme where the testing data set was divided into 8 roughly equally sized bins based on the number of graphs and nodes in those graphs.

Total running CPU time used was 1140h38m35s and all the simulations were finished in less than a week time.

7. Conclusions

An extension to the simple spring model graph drawing algorithm, that should counter the problem of having zero angular resolution, was designed, implemented and rigorously evaluated.

7.1. Test results

It was confirmed that on a publicly available, peer-reviewed, testing data set used for practical evaluation of graph drawing algorithms, the extended spring model yields better results than the simple spring model alone. For less than 1% of the graphs in the testing set, the algorithm yielded worse results.

The extended spring model algorithm cannot be used on its own, initial node positions need to be determined by some other algorithm. Using the simple spring model as the algorithm of choice for finding the initial positions is sufficient.

A downside of the extended spring model algorithm is its asymptotic time complexity and higher computational cost of distinct steps. The simple spring model can run in $O(n^3)$ time, whereas the extended algorithm requires between $O(n^3)$ and $O(n^4)$ – based on the number of edges in given graph.

The evaluation metric developed for this project can be used in the future to measure the performance of other graph drawing algorithms with respect to handling artifacts arising from zero or low angular resolution by measuring how close to the ideal angles distribution around nodes the results get.

7.2. Future improvements

As the extended spring model algorithm was shown to be improving the size of angles between edges towards their ideal size only after running a simple force-directed graph drawing algorithm, a future research should focus on analyzing whether the extended algorithm can also be used after other graph drawing algorithms as a final improvement before actually displaying the result.

To address the algorithmic complexity, study should be conducted to assess possible parallel implementation of certain parts of the algorithm. The forces calculation stage could be – in theory – run in as many threads as there are forces to calculate. Although such thread configuration would pose a different challenge for task scheduling on a massively parallel hardware architecture.

Such hardware architecture, affordable today by virtually anyone, is represented by GPU cards. These can nowadays process many hundreds or thousands tasks in parallel which means that for graphs in the testing set used a separate thread for calculating the forces could be spawned for each node – maybe for each pair of nodes. The most optimistic prediction would be that it should be possible to trade the time complexity for the number of computing cores. The best case scenario would then be to have the algorithm run in linear time with respect to the number of input nodes and edges.

An experimental study with larger graphs is needed to fully understand the nature of the slow decrease in improvement of the variance of measured to ideal angle over the whole graph, discussed in section 6 Results, as the number of nodes in the graph grows.

References

[BBDW17] BECK F., BURCH M., DIEHL S., WEISKOPF D.: A taxonomy and survey of dynamic graph visualization. *Computer Graphics Forum 36*, 1 (2017), 133–159. URL: http://dx.doi.org/10.1111/cgf.12791, doi:10.1111/cgf.12791. 1

[BGL*97] BATTISTA G. D., GARG A., LIOTTA G., TAMASSIA R., TASSINARI E., VARGIU F.: An experimental comparison of four graph drawing algorithms. *Computational Geometry 7*, 5 (1997), 303 – 325. URL: http://www.sciencedirect.com/science/article/pii/S0925772196000053, doi:http://dx.doi.org/10.1016/S0925-7721(96)00005-3. 1

[GFV13] GIBSON H., FAITH J., VICKERS P.: A survey of two-dimensional graph layout techniques for information visualisation. *Information Visualization 12*, 3-4 (2013), 324–357. URL: http://dx.doi.org/10.1177/1473871612455749, arXiv:http://dx.doi.org/10.1177/1473871612455749, doi:10.1177/1473871612455749. 1

[gnu17] GNUPLOT CONTRIBUTORS: gnuplot, 2017. Available online at http://gnuplot.info/. 10

[LY12] LIN C.-C., YEN H.-C.: A new force-directed graph drawing method based on edgeâĂŞedge repulsion. *Journal of Visual Languages & Computing 23*, 1 (2012), 29 – 42. URL: http://www.sciencedirect.com/science/article/pii/S1045926X11000802, doi:https://doi.org/10.1016/j.jvlc.2011.12.001. 1, 2, 5, 6

[Rac17] RACKET ORGANIZATION: Racket (formerly Scheme-PLT), 2017. Available online at http://racket-lang.org/. 9

[ST04] SHAVITT Y., TANKEL T.: Big-bang simulation for embedding network distances in euclidean space. *IEEE/ACM Transactions on Networking 12*, 6 (Dec 2004), 993–1006. doi:10.1109/TNET.2004.838597. 1

[Tut63] TUTTE W. T.: How to draw a graph. *Proceedings of the London Mathematical Society 13*, 52 (1963), 743–768. doi:10.1112/plms/s3-13.1.743. 1

[VDBG*00] VISMARA L., DI BATTISTA G., GARG A., LIOTTA G., TAMASSIA R., VARGIU F.: Experimental studies on graph drawing algorithms. *Software: Practice and Experience 30*, 11 (2000), 1235–1284. URL: http://dx.doi.org/10.1002/1097-024X(200009)30:11<1235::AID-SPE339>3.0.CO;2-B, doi:10.1002/1097-024X(200009)30:11<1235::AID-SPE339>3.0.CO;2-B. 1, 2, 4, 5, 9

[WT17] WANG C., TAO J.: Graphs in scientific visualization: A survey. *Computer Graphics Forum 36*, 1 (2017), 263–287. URL: http://dx.doi.org/10.1111/cgf.12800, doi:10.1111/cgf.12800. 1