Towards a Vertex and Edge Label Aware Force Directed Layout Algorithm

Roman Klapaukh[1] **David J Pearce**[1] **Stuart Marshall**[1]

[1] School of Engineering and Computer Science
Victoria University of Wellington,
New Zealand
Email: {roma,djp,stuart}@ecs.vuw.ac.nz

Abstract

Many automatic graph layout algorithms can cause shaped vertices and edge labels (which have a size when drawn on the screen) to overlap in the resulting visualisation. Overlaps can hide information that users expect to see in cases where the graph is small. We perform two experiments on a large real-world set of small (10-110 vertex) graphs to compare how different combinations of forces in Eades' force directed layout algorithm affect the final graph layout. We identify an optimal combination of forces from those we tested. In particular, we found that adding charged walls, variable node charge and edge label charges, minimises overlaps. We also found that using Hooke's Law over Eades' logarithmic attractive force tends to reduce edge crossings.

1 Introduction

Many different kinds of data can be represented visually by graphs. Automatic graph layout allows for the visualisation of systems ranging from social networks to mind maps and flow charts. In these cases and many other real world applications the vertices and edge labels of the graph will be *shaped*; they will have some shape and size when drawn rather than being a single point. Shapedness comes from the context of the graph. In cases where the user wishes to render large numbers of vertices on the screen, all details of individual vertices and edges is lost due to the lack of screen resolution. In those cases it is the overall pattern and structure of the graph that is interesting. In contrast, when only a small number of vertices are rendered, each can take up a reasonable amount of space on the screen. Each vertex can contain information which a user may wish to see. Overlaps, where vertices or edge labels occlude each other, limit the amount of information that can be conveyed to the user. While a lot of research focuses on the visualisation of large graphs, we believe that there are still problems in the visualisation of small labelled graphs as shown by the existence of small graph benchmarks in venues such as Graph Drawing [12], and the use of small graph layout algorithms as part of large graph layout [24, 1].

Automatic graph layout is meant to position all the objects creating a "good" visualisation. What it means to be "good" is discussed in Section 2, but in this work we are looking particularly at preventing overlaps between any combination of shaped vertices and edge labels. We consider both the count of how many overlaps there are and the proportion of pixels that are hidden. As a sanity check, we also record other properties of the layout and use the widely known and used heuristic of number of edge crossings [26].

Consider the example small shaped graph laid out in two different ways shown in Figure 1. This shows an extract from a social network where vertices contain the name of the person they represent, and edges have the type of relationship between people. In the top layout two of the vertices are overlapping, while in the bottom they are all drawn separately. In the top layout, it is hard to determine the names of the two overlapping vertices, and to determine which edge is connected to which. In the bottom layout there is no such confusion.

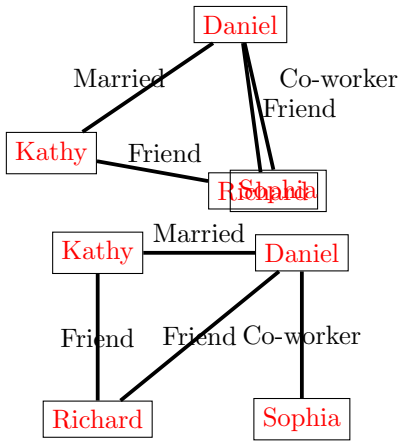

Figure 1: A good and a bad layout of a social network. In the above layout two of the vertices are overlapping and it is hard to read what they say or what edges connect to them.

While the situation of overlapping vertices can be avoided, most of the core layout algorithms assume that vertices and edge labels are not shaped – an assumption which is not upheld in many practical applications with small graphs. Furthermore, previous suggestions about how to resolve this issue have not been evaluated on a large scale (e.g. [14, 5, 4, 17, 9, 18]).

This paper contributes an evaluation of a range of variants to the standard force directed layout algorithm based on Eades' Spring Embedder [8] to prevent overlaps from occurring given shaped vertices and shaped edge labels. We experimentally evaluate a set of variants across a large real-world data set

using a suite of metrics.

We carried out a large scale experiment across 13,720 realistic graphs to evaluate how the numbers of overlaps and edge crossings changes as you use different combinations of forces. We followed this experiment with a second smaller experiment validating the results by ensuring that every combination was covered. The graphs ranged in size from 10 to 110 vertices and come from a graph drawing benchmark containing real world data from a major US corporation[12].

We found that:

1. Using charged walls, degree-based charge and charged edge labels reduce overlaps by an order of magnitude over the base algorithm.
2. Using Hooke's Law instead of the standard logarithmic attractive force tends to give layouts with fewer edge crossings.

Overall we found that the force directed layout algorithm without modifications performs significantly worse than the best modified version with respect to both overlaps and edge crossings. The ideas presented here can be applied to other graph layout algorithms, can be used in the layout of large graphs when using multi-level layout algorithms [1], and can be used for techniques such as multi-dimensional scaling [24].

2 Background

The task of graph layout is to assign a position to each vertex in a graph. This set of positions is called a layout, and should be *good*. While an exact definition of *good* is largely subjective, metrics exist for approximating how good a graph layout is [25, 26]. These metrics quantify different properties of the graph, such as number of edge crossings, angle of separation between edges connected to the same vertex, or enforcing certain rules such as keeping edges as straight lines, or promoting symmetry. Some metrics, such as maximising the similarity of edge lengths, involve solving NP-Complete problems [8, 19]. As there are many metrics, we choose two on which to focus. The first is the number of overlaps, as that is the primary purpose of this paper. The second is the number of edge crossings. This metric is widely known, is used by people performing manual graph layout [26], and serves as a sanity check on how our changes affect other properties of the graph.

2.1 Force Directed Layout

Our work looks at extending the force directed layout algorithm, based on Eades' Spring Embedder [8], with the edges of the screen (or layout area) as an immovable barrier as per Fruchterman and Reingold [10]. There are other graph layout algorithms, such as: the Kamada and Kawai method which also deals with spring systems but solves them using Newton-Raphson on derivatives [20]; likewise spectral layout which works by finding eigenvectors [13]; finally even a method where users manually layout subgraphs [30]. More algorithms can be found in [6]. We chose to use force directed layout for this work because there are many real world graph layout systems that implement it (e.g. [16, 15, 2]); it is iterative, allowing users to interact with it while it is running; it can be used with large graphs [24, 1, 29]; and because it is claimed to promote symmetry where possible [8]. However, the ideas from this work could be extended to other layout algorithms.

The force directed layout algorithm simulates the graph as a physical system (Figure 2). The basic code can be found in Algorithm 1. Edges, like springs, pull vertices together. Vertices, like charged particles, repel each other. The system uses friction to prevent dynamic equilibrium so it tends towards a fixed state. The result of the computation is then used as a visualisation of the graph. The user can interact with the graph while the algorithm runs.

The algorithm terminates either when a certain number of iterations have finished or when the kinetic energy is low enough. The kinetic energy is a measure of the activity of the system. Given the mass (m) and the velocity (v) you have that kineticEnergy = $\frac{1}{2}mv^2$. Low values for the kinetic energy suggest that very little movement is happening so the algorithm can now be stopped.

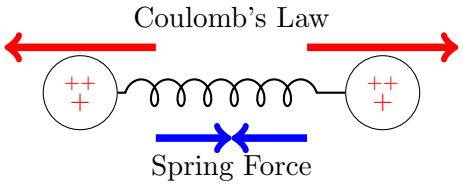

Figure 2: Basic Forces

Various forces can be used to model attraction and repulsion. Almost all the papers we surveyed used electrostatic repulsion (Coulomb's Law) as the repulsive force between vertices [8, 10, 6, 21]. In contrast, two different forces are used for the attractive force. Some papers, including Eades', use a logarithmic attractive force [8, 21, 23]. However, Eades originally describes edges as springs, so Hooke's Law (the physical law for ideal springs) is a natural alternative. It is not clear how they compare, though Battista et al. claim that from their experiences the logarithmic variant does not provide sufficiently better results given the extra computation [6]. We experimentally compare both attractive forces to see if there is a difference.

Algorithm 1 General force directed layout algorithm

```
1:  generateInitialLayout()
2:  for 0 .. maxIterations do
3:    totalEnergy = 0
4:    for all Vertex v do
5:      tempForce = (0,0)
6:      for all Edge {v,w} do
7:        tempForce += springForce(v,w)
8:      end for
9:      for all Vertex w do
10:       if v ≠ w then
11:         tempForce += coulombsLaw(v,w)
12:       end if
13:     end for
14:     v.move(tempForce)
15:     totalEnergy += v.kineticEnergy()
16:   end for
17:   if totalEnergy ≤ energyCutOff then
18:     break
19:   end if
20: end for
```

2.2 Existing Modifications

The original definitions used by Eades do not consider vertices as having any size, or edges as being

labelled [8]. In most graph applications, vertices have a dimension to consider, as they are represented on the screen, and edges may be labelled. For the sake of simplicity, we will refer to everything that is drawn except edges as images (e.g., text, raster images, etc). Having images leads to what we call *occluded* pixels, and *overlaps*. An occluded pixel is a pixel of an image which is covered by a pixel from a different image. An overlap occurs when two images are drawn so that one occludes some pixels of the other. This results in parts of the graph not being visible, reducing clarity and information retrieval from the visualisation, and so should be minimised. Our work focuses on minimising occlusions and overlaps.

Some previous works on reducing overlaps are modifications to the layout algorithm. Wang and Miyamoto implemented modifications that cancel out attraction of occluded vertices, vary constants to account for vertex size, and integrate a constraint solver [28]. They did not do an experimental analysis, except to time the layout of a single graph, and to generate six figures for the paper. Harel and Koren claim that naïve extensions to layout algorithms to deal with shaped vertices often have negative repercussions. They proposed changes to the Kamada and Kawai method and modifications to the spring method [14] that they claim do not have these limitations, but tested their ideas on only 7 graphs. Kumar et al. reduce clutter by giving certain vertices a stronger repulsive force in directed acyclic graphs [21], but test their algorithm on just two graphs. We extend this approach to create our degree-based charge force (discussed in Section 3.1). Lin et al. add torque to allow vertices to pack better [23], but this allows vertex images to end up at arbitrary angles, potentially decreasing readability. However, their analysis only contained 6 graphs.

Other works apply a post-processing step to 'fix' a layout. Force Transfer [18] and Force Scan [22] are two common algorithms to iteratively move vertices apart until they no longer overlap. Each of their experimental evaluations involved looking at only seven graphs. Frishman and Tal propose an algorithm to unclutter an existing layout [9], by mapping from the existing layout to one with a better information density. This algorithm is designed for huge graphs, where details on individual vertices are not visible, but was tested on just 5 graphs. Gansner and North use Voronoi diagrams to move vertex centres away from other each other [11], and notably introduce curved edges. Their experiment consisted of only nine graphs. Dwyer et al. use constraint solving to spread the vertices [7]. They tested their performance on some randomly generated graphs, but only tested the layout quality on a single graph.

None of the works above performed any large scale testing of their algorithms, with the largest test set containing only 12 graphs. This makes it hard to generalise their results. For this reason, we perform a large scale evaluation on over 13,720 realistic graphs, many of which are anonymised graphs from AT&T. Additionally, of all the algorithms we surveyed, only the ePRISM [17] algorithm explicitly considers edge labels, despite their common use in practice.

3 Algorithm Design and Variants

We use the general force directed algorithm as described in Section 2 as the basic algorithm. We describe some of the implementation details below. The system was implemented using Java 1.6.

Initial placement places each vertex at random. The natural length for the attraction force is set as the minimum distance such that the two connected vertices do not overlap [22, 1].

Vertex - vertex repulsion is done with Coulomb's Law and has the form $\mathbf{F} = -k_e \frac{q_1 q_2}{\|\mathbf{r_{21}}\|^2} \mathbf{\hat{r}_{21}}$, where k_e is a constant, q_i is the charge on the given vertex, r_{21} is the distance between the two vertices, and $\mathbf{\hat{r}_{21}}$ is the unit vector between the two vertices. All vertices have the default charge of q_{vertex}.

To move a vertex we convert the total force on it to acceleration using Newton's Law $\left(\mathbf{a} = \frac{\mathbf{F}}{m}\right)$. All vertices have the same mass (m). Each time a vertex is moved dampening reduces the velocity by a fixed proportion, to ensure that the system eventually settles to a static layout. The plane boundaries in our system are an impenetrable barrier representing the edges of the screen. Any vertex that hits a wall has its component of velocity in the direction of the collision reversed.

3.1 Variable Forces

We will now describe the different forces we experimentally investigated. Recall the basic configuration of forces from Section 2.

Hooke's Law (H) Hooke's Law is the physical law for ideal springs, and so is a logical candidate for the spring force to model the edges. Its advantage is that it is less computationally expensive than the logarithmic spring force [6]. It has the form $\mathbf{F} = -k_h (\mathbf{x} - \mathbf{N})$, where $\mathbf{x}$ is the vector between the two vertices, $\mathbf{N}$ is the natural length, and k_h is a constant describing how stiff the spring is.

Logarithmic Spring Force (L) This is the spring force Eades' original paper used, and is also used in other work [8, 6, 22]. It has the form $\mathbf{F} = k_l \log\left(\frac{x}{N}\right)$ where k_l is a constant, x is the distance between the two vertices, and N is the natural length. This gives it a behaviour that is similar in shape to Coulomb's Law, making it a logical, if more computationally expensive, alternative to Hooke's Law.

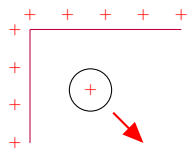

Figure 3: Charged Walls (W)

Charged Walls (W) Davidson and Harel proposed (but did not implement) this as a mechanism for keeping vertices inside fixed boundaries [5]. Wall charge is just Coulomb's Law applied to a line the length of the boundary as in Figure 3. This serves several purposes: it prevents unconnected components from moving infinitely far from each other; it prevents layouts settling close to the boundary where their ability to move is limited, impairing their ability to move into a minimal energy configuration; and it centers the resulting image. Wall charge in Table 2 shows the charge of each wall as used in our experiments.

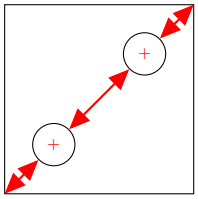

Figure 4: Wrap-Around Forces (A)

Wrap-Around Forces (A) Wrap-around forces cause the repulsive force to act as if the layout is on a torus as seen in Figure 4. This does not change how walls affect the vertices, but Coulomb's Law calculations are performed in both directions. Given any two vertices, this force pushes them to be equal distances from each other both ways around the plane, providing a similar effect to charged walls, where the vertices take the place of the charge on the wall. We hypothesize that combining wrap-around forces with charged walls will center all the graphs neatly. This is a novel force which we are testing.

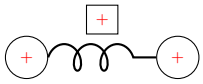

Figure 5: Charged Edge Labels (E)

Charged Edge Labels (E) Edge labels can be occluded by other images. To prevent that, charged edge labels makes labels charged in the same way that vertices already are as in Figure 5. This force is based on the obvious extension to force directed layout where edge labels are treated as special nodes. Each label has charge q_{label}, and so repels vertices to which it is not connected. Labels are unable to move independently so forces applied to a label instead affect the two vertices which it is connected to. This avoids moving the label independently, while allowing labels to affect the layout.

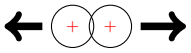

Figure 6: Collisions (C)

Collisions (C) Collisions between vertices is a trivial extension to force directed layout as it is a based on physical simulation. The collisions are ideal billiard ball collisions using the coefficient of restitution. Collisions are only implemented for vertices.

Our implementation computes the velocity changes due to the collision and sets the new velocities appropriately as in Figure 6. While it is possible to move the vertices apart until they are no longer overlapping, we do not do this because there may not always be a sensible layout possible where there are no overlaps. Some graphs may have very dense regions, or just so many vertices that allowing some amount of overlaps is actually beneficial.

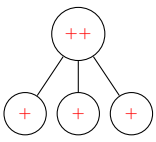

Figure 7: Degree-Based Charge (D)

Degree-Based Charge (D) Kumar et al. proposed to increase the charge on certain vertices to give them more space [21]. Their technique relies on the graph being a directed acyclic graph. We generalise their technique to general graphs, giving high degree vertices higher charge. The larger the degree of a vertex the more space it needs, therefore the more it should repel other images as in Figure 7. Thus we multiply the standard Coulomb repulsion between the vertices by the max of 1 and $\frac{\text{degree}(v_1)*\text{degree}(v_2)}{4}$. We use the product of degrees to increase the strength of the repulsion quickly, as the repulsive force falls off quickly with respect to distance. The original formula cannot be used directly as it requires directed acyclic graphs. The constant 4 is chosen for the denominator so that low degree vertices are not affected (supposing an average low degree vertex has degree 2).

4 Experiment

We ran two experiments comparing different combinations of forces to see how they affect the number overlaps, and edge crossings. To measure overlaps, we recorded both the number of overlapping vertices and the number of occluded pixels. We also recorded the time taken to lay out each graph, to see how the modifications affect performance. We use Coulomb's Law as the repulsive force in all the experiments as it is used in Eades' original work [8], and makes intuitive sense as the relevant physical law. In the first experiment we ran each of the fourteen combinations shown in Table 1 once on each of the 13,720 graphs in the dataset. In the second we ran all combinations of forces on a random sample of 100 graphs, to show that we did not miss any potentially good force combinations and to get a more complete picture of how forces affect computational performance.

Each layout was run until either the kinetic energy (a measure of the activity of the system) dropped below a given force cut off, or the maximum number of iterations was reached. We use short names comprised of a letter for each active force (excluding Coulomb's Law) in this paper for compactness.

Hypothesis We set out to investigate the following hypotheses.

$\mathcal{H}_1$ Changing between H and L has no effect on layout.

$\mathcal{H}_2$ Adding in all the additional forces except for C (i.e. WEDA) results in the lowest number of overlaps.

In order to test the various algorithms we used three test data sets from GraphDrawing.org [12]. The sets are called *Rome*, *North*, and *random-dag*. We use these graphs as they mostly contain graphs from real world applications and are small - containing 10 to 110 vertices, and 9 to 241 edges. There are a total of 13,720 graphs, of which all are used.

Choice of Constants We selected the image size based on a personal social network context such that they are small, but still easily recognisable. The width and height of the plane for embedding was limited to what most modern screens support. The constants were all set by trial and error, such that they looked reasonable on several data sets. Choice of constants is a difficult exercise, as small changes can have unforeseen effects. Many papers have simply ignored the issue entirely e.g. [6, 8, 22, 3, 11, 22, 21, 4]. A standard practice is to use a heuristic method similar to ours, such as optimising for a simple case e.g. [10, 28, 20]. While such methods are not ideal, addressing the core issue of how to choose optimal constant values is beyond the scope of this paper.

In the second experiment, the width and height of the screen were fixed to 1920 x 1080 to mimic a normal screen, and the image size was changed to be square rather than rectangular like a screen. We made these changes to see if small changes to the setup would affect the relative performance of the best combination of modifications. The full set of constants for both experiments is shown in Table 2.

5 Results

We used R version 2.11.1 (2010-05-31) [27] to analyse the results of both experiments. We recorded the number of pixels drawn, the total number of pixels, the number of overlapping images, and the number of edge crossings. The number of pixels drawn compared to the total tells us how many pixels were occluded and therefore how much is hidden. The number of overlaps gives an indication of how crowded the layout is. While a layout can have no overlaps and still be crowded, we only look at crowding that results in occluded pixels. Edge crossings were recorded to see if other features of the layout were affected by our changes, and as minimising them is considered to increase goodness [25, 26].

As the resulting data does not seem to follow a normal distribution, we used the Wilcoxon rank-sum test for significance testing as opposed to the t-test. We hold that the difference is significant at 95% significance ($p < 0.05$).

5.1 Experiment One

The list of medians for each force in the first experiment is shown in Table 3. The entries in bold blue are minima. All reported values have been rounded to 4 significant figures, with trailing 0s omitted.

The medians for the proportion of possible overlaps in the graph (of any size) and the raw count of overlaps can be found in Table 3. In both cases H and L ($p = 0.1176$, 0.3186 resp.) are not significantly different and the best performer is LWED. In terms of area lost due to occlusion the best performer is LWED. With respect to the proportion of edge crossings compared to an estimated upper bound (calculation from [25]), and also as a raw count HWED is the best performer.

5.2 Experiment Two

The list of medians for each force in the second experiment is shown in Table 3. The entries in bold blue are minima. All reported values have been rounded to two decimal places.

LD was the best performing combinations with respect to edge crossings. LWED was best in percentage of overlaps and occluded pixels, and best equal (with HWED) in count of overlaps. In 35 (72%) of cases the same combinations of forces, with H rather than L had less edge crossings.

6 Discussion

The LWED algorithm was the most effective at reducing overlaps in the graph, as well as having the lowest number of hidden pixels. While both results tables shows that in terms of raw counts HWED and LWED are the same, this is only because we are showing medians in the table. If we considered arithmetic means then LWEDs would be lower. This would make it the best algorithm for our original purpose. It also fared well with respect to edge crossings in the first experiment, coming second equal. HWED had the least edge crossings and came second with regards to minimising occluded pixels. An example graph generated using HWED can be see in Figure 9.

This disproves our second hypothesis that HWEDA or LWEDA would be the best force, but does show that adding in extra forces improves performance with respect to the metrics used.

All tests run with wrap-around forces (labelled A) fared poorly. This was a surprising result, as with two vertices this causes vertices to spread out. Upon running some additional simulations with larger graphs, we found that while it did push all the vertices together, it did so too much leading to poor performance. It seemed to perform particularly poorly on sparse non-planar graphs.

The results from the second experiment confirm that the LWED and HWED still perform the best. The changes to the parameters only affected the ordering of force combinations that did not perform well.

They also show that keeping everything else constant, changing from using Hooke's Law (H) to the logarithmic spring force (L) generally increases the number of edge crossings. While we are not sure what causes this, we think it has to do with how the different spring forces change over a small distance while a vertex is trying to move over an edge to make a crossing.

We were interested in how different forces affected performance. We recorded the runtime of each layout in the second experiment and used that to find an average run time for a single iteration of the algorithm for each.

There are three forces which visibly affect program runtime - E, A and C. Figure 8 shows the average time taken for a single iteration of the loop coloured by which combination of these forces is active. The figure shows that the charged edge labels (E) force incurs a clearly noticeable time increase. This is due to this modification requiring an extra inner loop running over all the edges. The presence of both C and A increases the run time more than having both independently would suggest. We believe this happens because the wrap-around forces (A) force promotes collisions, creating more work for the collisions (C) force.

Validity The most notable omission of the experiments is that we do not explore how different combinations of constants affect the output of the program. This is something that has not been explored in other papers. It also has a very large search space, and would take infeasibly long to run.

While we used a large test set of graphs, they do not encompass all possible graphs. They span a variety of graph densities, from 0.8608% to 85.45%, though the majority are less than 8.602%, but all except 3 are connected. Nevertheless, we believe that the set of graphs tested here is representative of many small real life graphs as it is sampled from a real data set. We hypothesis that changes and variances in image size can be accounted for by linked changes to physical constants for each vertex. This is supported by the second experiment where images were smaller, but the best performing forces were the same.

Future Work One difficulty in evaluating algorithms is the selection of constants, and what effect this has on the resulting layout. Further work evaluating the effects of constants would be valuable for

Table 1: Sets of forces we tested in experiment one

Experiment	1	2	3	4	5	6	7	8	9	10	11	12	13	14
H (Hooke's Law)	✓	✓	✓	✓	✓	✓	✓							
L (Logarithmic Attraction)								✓	✓	✓	✓	✓	✓	✓
W (Charged Walls)		✓	✓	✓	✓	✓			✓	✓	✓	✓	✓	
E (Charged Edge Labels)		✓	✓	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓
C (Collisions)			✓							✓				
D (Degree Based Charge)				✓		✓	✓				✓		✓	✓
A (Wrap-Around Forces)					✓	✓	✓					✓	✓	✓

Table 2: Constants used in both experiments

Constant	Value	Constant	Value
All Experiments			
k_e	50,000	Coefficient of Restitution	0.9
k_l	-60	Kinetic Energy Cut Off	3
k_h	0.2	Edge Label Length	2-4 characters
q_{label}	1	Natural Spring Length	Min dist to not overlap
q_{vertex}	3	Dampening	0.9
Wall Charge	1000	Max Iterations	10,000
Vertex Mass	2		
Experiment 1			
w, h	400px $\leq$ #Vertices $\times$ 100 $\leq$ 8000px		
Vertex Image Size	107x87 pixels		
Experiment 2			
$w \times h$	1920 x 1080	Vertex Image Size	80x80 pixels

Table 3: Medians by force from experiment one

Median	# Crossings	Prop. Crossings	# Overlaps	% Overlaps	% Occluded Pixels
H	40	0.02256	13	0.2043	1.213
HWE	38	0.02265	3	0.0407	0.1072
HWEC	39	0.0226	3	0.04024	0.104
HWED	**34**	**0.01984**	**1**	0.01463	0.02526
HWEA	80	0.05337	85	1.948	16.46
HWEDA	73	0.04748	53	1.276	12.21
HEDA	71	0.04523	49	1.144	11.34
L	41	0.02369	13	0.2016	1.195
LWE	42	0.02445	2	0.03487	0.08999
LWEC	41	0.02459	2	0.03518	0.09045
LWED	38	0.02231	**1**	**0.01058**	**0.0159**
LWEA	92	0.0615	103	2.266	18.4
LWEDA	91	0.06051	69	1.744	15.34
LEDA	89	0.05751	63	1.583	14.3

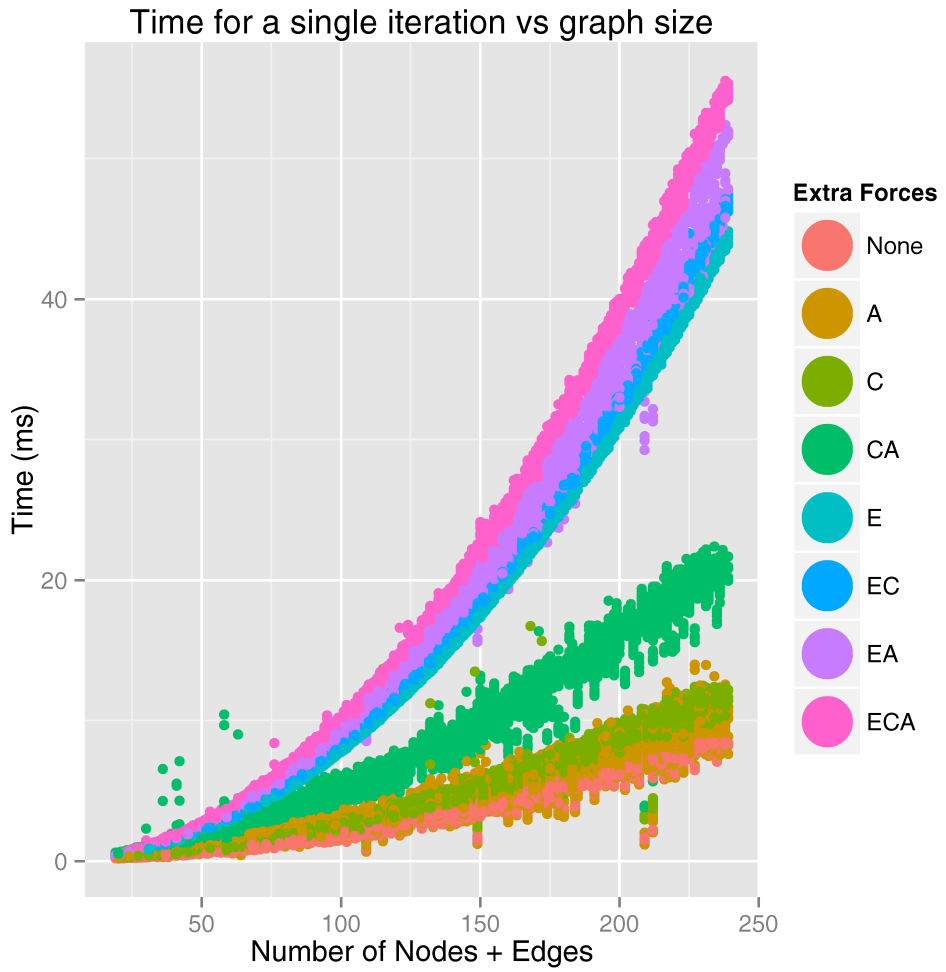

Figure 8: Time for a iteration vs number of vertices and edges in the graph coloured by forces which reduce performance. Times are an average over the whole runtime of the program with all combinations of forces. There are three well defined strata. The top one consists of combinations which contain E. Within this there are further divisions were C or A are also present. The second strata consists solely of combinations containing CA in the force. The final strata shows that A and C slow down the algorithm, and having none of E,C or A is the fastest.

everyone using any graph layout algorithm. Moreover, exploratory studies suggest that size of the layout plane affects the resulting layout significantly and we hope to explore this in future.

7 Conclusion

We looked at modifications to the force directed algorithm to avoid overlaps in small graph layout. We described a range of different forces which can be used with layout algorithms. We then performed two experiments to find which set of forces would produce a layout that was spread out applied to the force directed layout algorithm. We found adding in edge label charges, charged boundaries, and increasing vertex charge proportionally to its degree results in layouts that minimise the number of overlaps and occluded pixels; that using Hooke's law reduces the number of edge crossing; and that while there is a time cost for using charged edge labels, it is not so high as to make the modification too expensive. As such, we conclude that adding in additional forces is a viable way of preventing occluded pixels for graphs with large vertices and edge labels.

References

[1] Archambault, D., Munzner, T. and Auber, D. [2007], 'TopoLayout: Multilevel graph layout by topological features', *IEEE TVCG* **13**(2), 305–317.
URL: *http://dx.doi.org/10.1109/TVCG.2007.46*

[2] AT&T [n.d.], 'Graphviz'. `www.graphviz.org/` accessed: 20 Nov 2012.

[3] Brandes, U., Kääb, V., Löh, A., Wagner, D. and Willhalm, T. [2000], 'Dynamic WWW structures in 3D', *JGAA* **4**(3), 183–191.

[4] Chuang, J.-H., Lin, C.-C. and Yen, H.-C. [2004], Drawing graphs with nonuniform nodes using potential fields, *in* 'Proc. GD, LNCS', pp. 460–465.

[5] Davidson, R. and Harel, D. [1996], 'Drawing graphs nicely using simulated annealing', *ACM TOG* **15**(4), 301–331.

[6] Di Battista, G., Eades, P., Tamassia, R. and Tollis, I. G. [1999], *Graph Drawing: Algorithms for the Visualization of Graphs*, Prentice Hall.

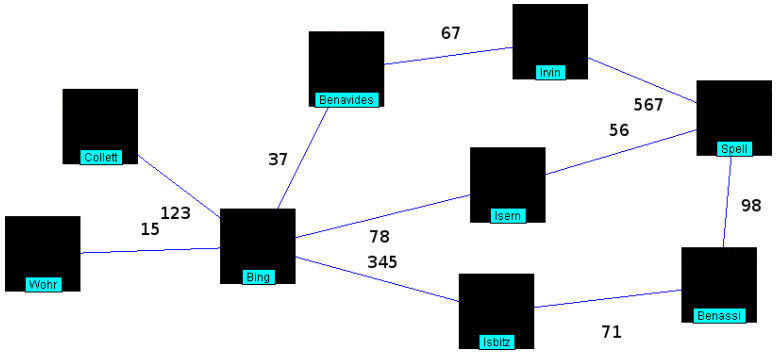

Figure 9: A graph produced by HWED. There are no overlaps or edge crossings allowing a human to see details on the vertices as well as the graph structure

[7] Dwyer, T., Marriott, K. and Stuckey, P. J. [2005], Fast node overlap removal, *in* P. Healy and N. S. Nikolov, eds, 'GD', Vol. 3843 of *LNCS*, pp. 153–164.

[8] Eades, P. [1984], 'A heuristic for graph drawing', *Congressus Numeratum* **42**, 149–160.

[9] Frishman, Y. and Tal, A. [2009], 'Uncluttering graph layouts using anisotropic diffusion and mass transport', *IEEE TVCG* **15**(5), 777–788.

[10] Fruchterman, T. M. J. and Reingold, E. M. [1991], 'Graph drawing by force-directed placement', *Software - Practice and Experience* **21**(11), 1129–1164.

[11] Gansner, E. R. and North, S. C. [1998], Improved force-directed layouts, *in* S. Whitesides, ed., 'GD', Vol. 1547 of *LNCS*, Springer, pp. 364–373.

[12] *graphdrawing.org* [n.d.]. `http://graphdrawing.org/data.html` accessed: 18 May 2011.

[13] Hall, K. M. [1970], 'An r-dimensional quadratic placement algorithm', *Management Science* **17**(3), pp. 219–229.
URL: *http://www.jstor.org/stable/2629091*

[14] Harel, D. and Koren, Y. [2002], Drawing graphs with non-uniform vertices, *in* 'Proc. of the Working Conf. on AVI', AVI 2002, ACM, New York, pp. 157–166.
URL: *http://doi.acm.org/10.1145/1556262.1556288*

[15] Heer, J., Card, S. K. and Landay, J. A. [2005], Prefuse: a toolkit for interactive information visualization, *in* 'Proceedings of the SIGCHI Conference on Human Factors in Computing Systems', CHI '05, ACM, New York, NY, USA, pp. 421–430.
URL: *http://doi.acm.org/10.1145/1054972.1055031*

[16] Hotson, D. [n.d.], 'Springy.js'. `http://getspringy.com/` accessed: 20 Nov 2012.

[17] Hu, Y. [2009], 'Visualizing graphs with node and edge labels', *CoRR* **abs/0911.0626**.

[18] Huang, X., Sajeev, A. S. M. and Lai, W. [2006], A scalable algorithm for adjusting node-node overlaps, *in* 'CGIV', IEEE Computer Society, pp. 43–48.

[19] Johnson, D. S. [1982], 'The NP-completeness column: An ongoing guide', *J. of Alg.* **3**(2), 182–195.
URL: *http://www.sciencedirect.com/science/article/pii/0196677482900189*

[20] Kamada, T. and Kawai, S. [1989], 'An algorithm for drawing general undirected graphs', *Information Processing Letters* **31**(1), 7–15.

[21] Kumar, P., Zhang, K. and Wang, Y. [2008], Visualization of clustered directed acyclic graphs without node overlapping, *in* 'IV', IEEE Comp. Soc., pp. 38–43.

[22] Li, W., Eades, P. and Nikolov, N. S. [2005], Using spring algorithms to remove node overlapping, *in* 'APVIS', Vol. 45 of *CRPIT*, pp. 131–140.

[23] Lin, C.-C., Yen, H.-C. and Chuang, J.-H. [2009], 'Drawing graphs with nonuniform nodes using potential fields', *JVLC* **20**(6), 385–402.

[24] Morrison, A., Ross, G. and Chalmers, M. [2003], 'Fast multidimensional scaling through sampling, springs and interpolation', *Information Visualization* **2**(1), 68–77.

[25] Purchase, H. C. [2002], 'Metrics for graph drawing aesthetics', *JVLC* **13**(5), 501–516.

[26] Purchase, H. C., Pilcher, C. and Plimmer, B. [2012], 'Graph drawing aesthetics - created by users, not algorithms', *IEEE TVCG* **18**(1), 81–92.

[27] R Development Core Team [2010], *R: A Language and Environment for Statistical Computing*, R Foundation for Statistical Computing, Vienna, Austria.
URL: *http://www.R-project.org*

[28] Wang, X. and Miyamoto, I. [1995], Generating customized layouts, *in* F.-J. Brandenburg, ed., 'GD', Vol. 1027 of *LNCS*, Springer, pp. 504–515.

[29] Wong, P. C., Foote, H., Mackey, P., Jr., G. C., Sofia, H. J. and Thomas, J. [2008], 'A dynamic multiscale magnifying tool for exploring large sparse graphs', *Information Visualization* **7**(2), 105–117.

[30] Yuan, X., Che, L., Hu, Y. and Zhang, X. [2012], 'Intelligent graph layout using many users' input', *IEEE TVCG* **18**(12), 2699–2708.

Table 4: Medians by force from experiment two. Short names are used in the column names for compactness. C is Crossings, O is Overlaps and P is Pixels Occluded. The minimum value for each metric is highlighted in blue.

Force	# C	% C	# O	% O	% P	Force	# C	% C	# O	% O	% P
H	34.50	2.28	13.00	0.24	0.91	HA	276.00	18.20	375.50	7.61	80.38
HC	48.00	3.39	19.00	0.34	1.31	HCA	298.00	19.66	172.00	3.12	29.67
HD	31.00	2.09	10.00	0.22	0.82	HDA	294.50	20.57	411.50	8.61	83.47
HDC	32.00	2.25	10.00	0.22	0.83	HDCA	312.00	20.46	172.50	3.10	29.34
HE	38.00	2.72	6.00	0.13	1.16	HEA	95.00	6.94	16.00	0.32	5.72
HEC	41.00	2.87	8.00	0.18	2.02	HECA	55.00	4.18	23.00	0.46	5.76
HED	40.00	2.85	6.00	0.12	1.22	HEDA	176.50	12.02	128.00	2.55	45.02
HEDC	41.50	2.89	6.00	0.14	1.57	HEDCA	93.00	6.51	41.00	0.72	8.96
HG	39.00	2.74	8.00	0.19	1.37	HGA	98.00	7.09	21.00	0.41	6.15
HGC	42.00	3.09	11.00	0.23	2.04	HGCA	60.00	4.57	28.00	0.54	6.15
HGD	40.00	2.88	9.00	0.18	1.38	HGDA	171.00	11.94	100.00	2.00	38.88
HGDC	43.00	3.14	10.00	0.21	1.68	HGDCA	99.00	7.11	44.50	0.77	9.34
HW	55.00	3.36	35.00	0.63	2.42	HWA	284.00	19.04	388.50	7.99	82.60
HWC	123.00	8.57	65.50	1.07	8.99	HWCA	306.50	20.66	177.00	3.34	30.74
HWD	43.00	2.62	22.00	0.40	1.54	HWDA	312.00	21.02	434.50	8.83	84.34
HWDC	48.00	3.46	24.50	0.51	1.74	HWDCA	323.00	21.09	176.00	3.33	30.69
HWE	34.00	2.17	4.00	0.08	0.33	HWEA	146.00	9.80	34.00	0.59	10.54
HWEC	39.00	2.74	9.00	0.18	1.48	HWECA	83.50	6.08	44.00	0.81	10.31
HWED	34.00	2.22	**2.00**	0.05	0.13	HWEDA	192.00	13.20	165.00	3.37	52.25
HWEDC	38.00	2.59	5.00	0.11	0.69	HWEDCA	125.50	8.52	61.00	1.09	13.19
HWG	35.00	2.19	7.00	0.15	0.49	HWGA	143.00	9.94	40.00	0.70	10.62
HWGC	44.00	3.03	13.00	0.27	1.75	HWGCA	98.00	6.98	51.00	0.89	10.75
HWGD	35.00	2.25	6.00	0.13	0.39	HWGDA	197.00	12.89	137.00	2.71	46.65
HWGDC	40.00	2.87	9.00	0.20	0.95	HWGDCA	133.50	9.65	64.00	1.15	12.89
L	33.00	2.09	13.00	0.24	0.91	LA	266.50	18.08	369.00	7.42	79.89
LC	42.00	2.81	17.00	0.30	1.16	LCA	285.50	19.60	172.00	3.03	29.73
LD	**27.00**	**1.87**	9.00	0.20	0.78	LDA	292.50	19.94	418.50	8.40	82.26
LDC	29.00	1.98	9.00	0.20	0.75	LDCA	315.50	20.60	176.00	3.20	29.54
LE	37.00	2.65	4.00	0.10	0.86	LEA	89.00	6.60	12.00	0.26	4.89
LEC	40.00	2.80	6.00	0.12	1.20	LECA	52.00	3.91	20.00	0.41	5.21
LED	39.00	2.77	4.00	0.08	0.73	LEDA	178.00	11.47	106.00	2.06	40.06
LEDC	40.00	2.87	5.00	0.10	1.18	LEDCA	95.00	6.36	40.00	0.71	8.80
LG	36.00	2.72	7.00	0.15	0.99	LGA	95.00	6.74	17.00	0.36	5.89
LGC	42.00	3.03	9.00	0.18	1.54	LGCA	60.00	4.45	24.00	0.50	5.34
LGD	39.00	2.85	7.00	0.16	1.01	LGDA	170.50	11.31	81.00	1.64	34.29
LGDC	40.50	3.02	7.00	0.17	1.22	LGDCA	100.50	7.27	43.00	0.76	9.11
LW	50.00	3.07	36.00	0.63	2.44	LWA	288.00	19.23	401.50	8.04	82.48
LWC	110.00	7.68	65.00	1.04	8.33	LWCA	311.50	20.93	179.00	3.32	30.71
LWD	35.00	2.28	19.50	0.38	1.41	LWDA	315.50	20.97	442.50	8.73	84.21
LWDC	40.00	2.85	22.00	0.45	1.57	LWDCA	322.50	21.50	179.00	3.37	30.78
LWE	32.00	2.10	3.00	0.06	0.25	LWEA	141.50	9.74	32.00	0.53	9.92
LWEC	36.00	2.53	6.00	0.13	0.90	LWECA	84.00	6.03	43.00	0.78	10.14
LWED	34.00	2.16	**2.00**	**0.04**	**0.09**	LWEDA	185.00	12.72	145.00	2.96	49.16
LWEDC	36.00	2.54	3.00	0.07	0.36	LWEDCA	118.00	8.56	61.00	1.08	12.97
LWG	33.00	2.11	6.00	0.14	0.41	LWGA	145.50	9.92	38.00	0.64	10.65
LWGC	41.00	2.95	10.00	0.22	1.15	LWGCA	96.00	6.72	49.00	0.85	10.63
LWGD	35.00	2.23	5.00	0.12	0.37	LWGDA	195.00	12.47	112.00	2.35	43.16
LWGDC	37.00	2.73	7.00	0.16	0.58	LWGDCA	133.00	9.51	66.00	1.14	13.11