Applying Layout Algorithms to Hand-drawn Graphs

Peter Reid[1], Fred Hallett-Hook[1], Beryl Plimmer[1], Helen Purchase[2]
Auckland University[1]
Auckland, New Zealand
prei033|fhal010|bplimmer@ec.auckland.ac.nz
Glasgow University[2]
Glasgow, Scotland
hcp@dcs.gla.ac.uk

ABSTRACT

Hand-drawing a node-and-edge graph is a simple visual problem solving technique; however as the graph is built it can easily get untidy and confusing. It is more difficult to understand and interpret a confusing graph. By applying edge morphing techniques and a force-directed algorithm the hand-drawn graph can retain its informal appearance while its layout is improved. Graphs will be more readily understood, making the problem solving process easier.

General Terms
Design, Human Factors

Author Keywords
Layout algorithms, sketch tools, Tablet PC

ACM Classification Keywords
H5.2 [Information interfaces and presentation]: User Interfaces, Graphical user interfaces.

1. INTRODUCTION

Imagine you want to solve a problem using a node-edge graph. Traditional choices such as whiteboard or paper are intuitive to use, but lack the advantages of editing, archiving and searching of a digital copy. A general diagram tool has these advantages, and allows layout changes to be applied, but current widget-based tools, we hypothesise, interfere with the problem solving process. A specific tool for graph layout for use with the tablet PC offers the flexibility of a pen interface as well as the advantages of having a digital copy, including the ability to apply layout algorithms.

Node-and-edge graphs are a useful problem solving tool. Classic computer science problems such as shortest path finding algorithms can be solved by drawing and interpreting a node-edge graph. It has been shown that graphs are easier to understand if certain aesthetics are optimised. The most critical aesthetic elements are crossing edges, bends in edges and the symmetry of a graph [5-7]. Graphs hand-drawn on a tablet PC can be altered to optimize these aesthetics.

The tablet PC offers an intuitive interaction space for users. Using the familiar pen and paper metaphor, the user can sketch on a tablet screen with a stylus. Strokes drawn with the stylus are stored as a series of coordinates, at a high resolution using hi-metric units.

To explore this idea we have worked with undirected graphs that are composed of nodes and edges. Each node may contain some text. Each edge can associate one node with another [2]. The next section discusses the related work and presents software requirements. We then discuss the implementation of the software and the issues involved.

2. RELATED WORK

Prior work on the recognition of hand drawn diagrams and the recording of their underlying semantics exists: Freeman and Plimmer [2] present an implementation of graph recognition using InkKit and describe how several types of graph can be recognised. These include undirected and hierarchical graphs and allow for the exporting of the structure to other forms. InkKit, however, does not include any graph layout algorithms to automatically place the nodes in optimal positions. There are many such algorithms; for this work we have selected a simple force-directed algorithm [3]. It is based on heuristics which calculate the amount of attraction and repulsion force between nodes, while limiting the force to keep the graph from expanding too far.

Purchase et. al. [6] analyses various force based algorithms and how well they meet various conditions that improve the readability of graphs. Tests were done on the differences in performance with different graphs and different layout algorithms with conclusions about the performance of each. Work with sketch tools has shown that maintaining an informal appearance has been proven to be beneficial in other domains [8]. We hypothesize that it will also be beneficial in the domain of abstract problem solving using node-edge graphs. However before this hypothesis can be tested we must replicate the functionality of current graph drawing tools in a sketch tool. It is the creation of this sketch tool for graphs that is the focus of this paper.

3. REQUIREMENTS

Basic digital ink input, editing and persistence must be supported. This is simple and requires no special functionality over that which is provided in the Microsoft Ink SDK. However for the software to be able to layout the graph there are several other requirements.

First, it is necessary to recognize the elements of the graph. Strokes drawn by the user should be recognized as nodes, edges, or text within nodes. Second, the layout algorithm must be applied to the graph to find new

OzCHI 2007 Proceedings, ISBN 978-1-59593-872-5

positions for the nodes. Finally, elegant reflow of edges must be performed as nodes are moved in order to retain the original hand-drawn appearance.

3.1. Recognition

When the graph is drawn the strokes must be classified as text, node or edge. This is done by first running the strokes though a divider which divides text from shapes. The Microsoft text recognizer includes a divider, however it is strongly biased to text, we have used [4] a divider developed specifically for diagrams.

Strokes identified as a shape are passed to an algorithm which then classifies them as node or edge. Our heuristic used the idea that a node (circle) has start and end points relatively close together while edge end points are far apart. Therefore if the distance from the first to the last point of the stroke is less than one third of the length of the stroke then the stroke is categorized as a node. Otherwise the stroke is classified as a edge. As a final step, any strokes drawn inside a node are classified as text regardless of previous classification. The recognised strokes are colour coded for human recognition with a blackboard metaphor. The background colour is black, nodes are drawn as white, text as green and edges as yellow.

Once the nodes and edges are identified logical connections between nodes are established by finding the node that encloses or is within a small distance from the endpoint of a edge. Thus edges are associated with nodes and the structure of the entire graph is represented within the software. These nodes and edges are stored in a graph class which contains a list of its nodes and edges as well as identifiers for the actual strokes in the visual graph.

3.2. Layout Algorithm

There are many different types of algorithm used for optimally rearranging the nodes in a graph such as force-directed algorithms and simulated annealing. The goal of each is to find final positions of the nodes which allow the clearest view or optimization of some feature of the graph (e.g. the minimisation of the number of edge bends or edge crossings). The final solution is normally one of many different solutions which could have been found and some may be better than others. Layout algorithms are highly scalable and can be used for graphs with thousands of nodes or graphs with very few nodes [6].

The type of algorithm which was implemented was a spring force-directed algorithm [3]. The algorithm maximises symmetry, lowers the number of edge crossings, minimises edge bends, thus leading to a more readable graph. Spring force-directed algorithms work by modelling each edge of the graph as a spring or elastic band pulling the nodes at each end of the edge together. Simultaneously each node can be imagined as having a negative charge so each node repels non-connected nodes away from it.

Fruchterman and Reingold's algorithm [3] uses the two heuristics shown in figure 1 for the attraction and repulsion forces based on a factor calculated by the area and number of nodes. The graph G with nodes V and edges E must be rearranged in the available area where x is the distance between two nodes. For a full explanation of this algorithm see [3].

$$G = (V, E)$$
$$k = \sqrt{area / |V|}$$
$$attraction = x^2 / k$$
$$repulsion = k^2 / x$$

Figure 1: Heuristics for attraction and repulsion, reproduced from [3]

These interactions, when repeated, cause the graph to oscillate around an optimum position. This can be likened to the way a pendulum without friction would swing around the rest position. But with friction, the same pendulum would eventually stop in its rest position. Because of this, as the algorithm proceeds the forces must be limited or reduced. The reduction in force is analogous to temperature cooling. This will reduce the amount of movement so that the nodes move into positions where the forces are balanced.

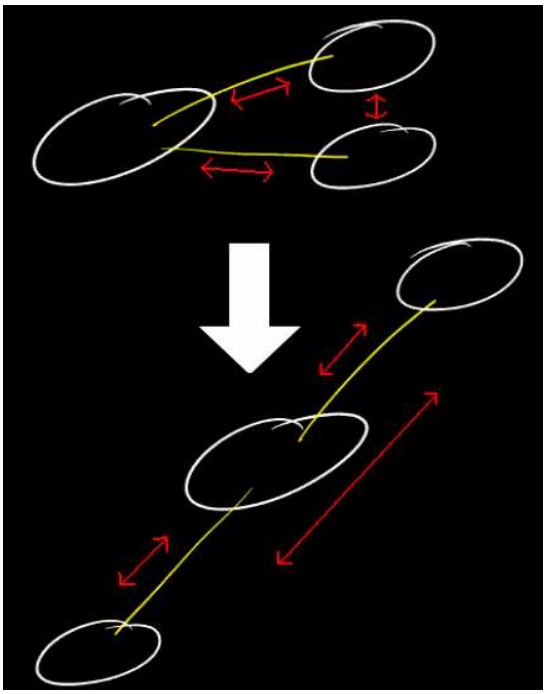

Figure 2: Nodes pull and push themselves into position.

In our implementation this temperature cooling was achieved by reducing the node movement by a set percentage each iteration. This slowly reduces the distance travelled and allows the nodes to settle in positions of balanced force. The node positions are constrained so that nodes do not move outside of the visible area.

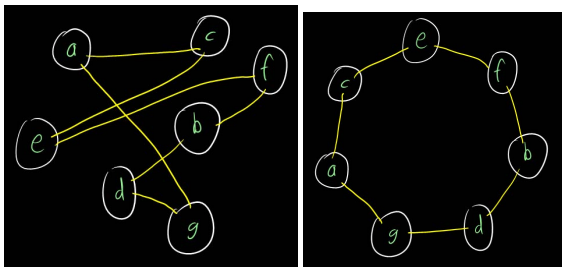

Figure 3: A graph before and after the layout algorithm has been applied.

3.3. Edge Morphing

When a node is moved, every edge attached to the node must be reflowed to reflect the new position of the node. With a formal graph this is simply a matter of repositioning the end points of the lines. However hand-drawn lines need to be morphed to preserve the informal appearance. Our implementation performs these morphs by morphing each edge attached to a repositioned node with respect to the fixed node at the opposite end of each edge. In this way, each of the edges attached to the mobile node is morphed to reflect its new position.

When morphing a line using a string metaphor the line can undergo three types of transformation [1]. These are compressing, stretching and overstretching. A line is represented internally as a series of x, y points. In each case the internal representation of the edge is extended by adding an x, y point to each end of the line to extend it to the centre of the appropriate node before morphing (the added sectors are not rendered on the surface). The centre of a node is approximated using the intersection of the diagonals of the bounding box of the node.

Stretching is achieved using linear interpolation. The new position of each point on the edge stroke is interpolated between its original position and its position on a straight line of the same length as the edge. The line is then translated and rotated into place, so that the extended ends of the line join the centres of the nodes. The visual result is that the line has unfolded. The attachment point and offset data is then used to remove the extensions.

Compression could also be achieved using linear interpolation, between points on the original line, and on a curved elliptical version of the line. We decided against this, in favour of a solution which did not cause the line to bunch up when compressed. The approach we used was to scale the line to compress it in the direction of the line which joins the centres of the two nodes. To achieve the scaling in the appropriate direction, the edge is rotated to horizontal, scaled in the horizontal direction using the Ink resize method, and then rotated back to the original angle. The angle of rotation that occurs as a result of the node movement is then applied. This approach breaks the string metaphor, but better retains the original appearance of the edge.

Once a line unfolds until the point where it is straight, it is in the overstretched state[1]. A working line is constructed between the centres of the two relevant nodes. The attachment point of the edge to each node can then be calculated by finding the intersections of strokes. The offset of each end of the edge from the attachment point can then be applied, as in the stretching of lines.

It is a non-trivial matter to maintain a hand-drawn appearance when morphing a edge. A number of features of the line need to be considered and preserved. It is ideal to maintain an approximately normal angle of incidence of a edge with a node [1]. If the line is extended to the centres of the nodes, this allows the point at which the edge intersects with each node to rotate about the centre of the node. This also maintains an approximately normal angle of incidence. Without such a mechanism in place, the edge runs the risk of being bent so that it enters the node at an awkward angle.

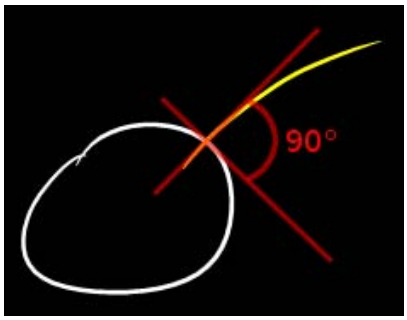

Figure 4: The angle of incidence of the edge is kept approximately normal.

If a edge falls just short of a line, or if it enters the node, the visual appearance of the intersection should be preserved in order to preserve the user's style. This can be achieved by storing the length of the arc from the attachment point to the end of the edge [1]. This length also needs to be recorded as positive or negative. This denotes whether the end of the edge falls within the node, or falls short of it, and is calculated using the centre of the node. The distance from the centre of the node to the end of the edge will be greater than the distance from the centre of the node to the attachment point, if the edge falls short of the node.

4. INTERACTION

To create a graph the user draws directly on the Tablet PC. Ink strokes are immediately recognized and colour coded as a node, edge or text. Eraser, select and move operations are available, and the user can clear the entire graph space. Save and load facilities are provided. The layout algorithm is applied when the user clicks either of the 'optimise layout' or '>' buttons. The first immediately re-renders the graph in a fully optimized form; the second animates the repositioning of each node in the order that the nodes were added to the graph. Additions and alterations can be made to the rearranged graph and the layout algorithm reapplied.

In preparation for evaluations comparing the affect of the visual fidelity of a graph we also render the graph as a formal representation. The user can switch between views by clicking the tabs at the top left of the drawing space. In this prototype the formal view is not editable.

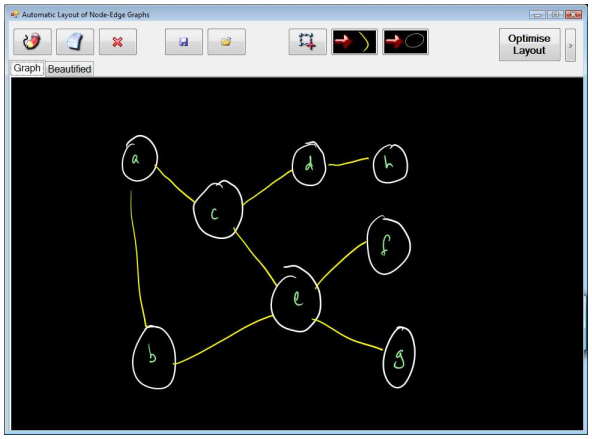

Figure 5: The user interface.

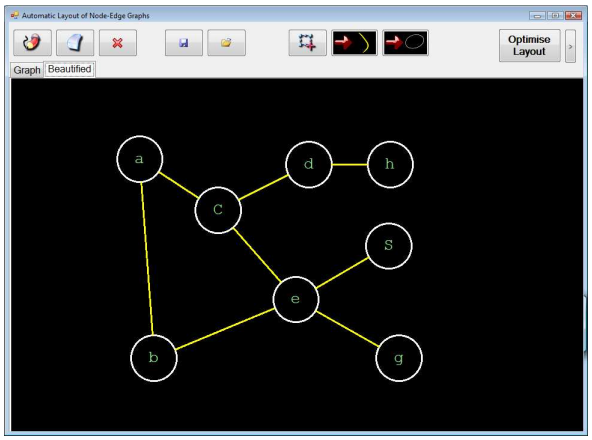

Figure 6: The formalized representation of the graph in Figure 5

5. DISCUSSSION

The recognition algorithm, although simple, provides satisfactory results. The layout algorithm works as expected, however it does not attempt to preserve any aspects of the users initial layout. What effect this reorganisation of the users work has on their spatial memory and understanding of the graph is unclear and a subject for further studies. A key reservation we have about the effectiveness of the edge morphing approach we have implemented is in the likely proliferation of overstretched lines. When applying layout algorithms to graphs, it is likely that many of the edges will at some point move into the overstretched state. The overstretched state, which causes the edge to become a straight line, imposes a formal appearance on the edge. An alternative approach could be used to replace the stretching and overstretching states of the line. A line could be scaled by a ratio in the direction of the baseline. A ratio above one would cause the line to stretch, and a ratio between zero and one would cause the line to compress. This would have the advantage of maintaining the hand-drawn appearance of the edges, but would lose the benefits of the extended line approach currently used in the stretching.

More sophisticated division for nodes and edges could be implemented to support directed graphs and more complex diagrams. The current heuristic does, however, successfully classify strokes in most reasonable small drawings.

The layout algorithm, while it works, is not completely optimized for graphs where nodes and edges are hand drawn because the nodes vary in size. This means that large nodes may overlap with other nodes. In addition highly curved edges may overlap with other graph elements when the new layout is applied.

6. Conclusions

The software we have developed successfully meets the goal of applying a layout algorithm to hand-drawn graphs. This can increase the understandability of the graphs drawn by the user by improving the layout of the nodes and edges. Although further work could yield improvement, the edge morphing techniques applied successfully retain much of the user's drawing style by retaining and recreating features of edge lines.

The next steps in this project are to evaluate the effect on the user of applying layout algorithms to their newly created hand-drawn graph and to compare human comprehension of optimised and non-optimised hand-drawn and formally rendered graphs.

REFERENCES

[1] Arvo, J., Novins, K., Appearance-preserving manipulation of hand-drawn graphs, in proc Graphite, ACM, 61-68 (2006)

[2] Freeman, I., Plimmer, B., Edge Semantics for Sketched Diagram Recognition, in proc AUIC, ACM, 71-78 (2007)

[3] Fruchterman, M. J. T., Reingold, M. E., Graph drawing by force-directed placement, Softw. Pract. Exper., 21, 11, 1129-1164, (1991)

[4] Patel, R., Plimmer, B., Grundy, J., Ihaka, R., Exploring Better Techniques for Diagram Recognition, NZ Computer Science Student Conference, (2007),

[5] Purchase, H. C., Which aesthetic has the greatest effect on human understanding?, in proc Graph Drawing Symposium, (1997)

[6] Purchase, H. C., Carrington, D.A. and Allder J-A., Empirical evaluation of aesthetics-based graph layout, Empirical Software Engineering, 7, 3, 233-255 (2002)

[7] Ware, C., Purchase, H.C., Colpoys, L. and McGill, M., Cognitive Measurements of Graph Aesthetics, Information Visualization, 1, 2, 103-110 (2002)

[8] Yeung, L. W. S., Exploring beautification and the effects of designs' level of formality on the design performance during the early stages of the design process Department of Psychology, MSc, University of Auckland, (2007),