Ross Shannon · Aaron J. Quigley

Considerations in Dynamic Graph Drawing: A Survey

20th June 2007

Abstract In this survey paper we discuss the considerations involved in modeling, visualising and exploring *dynamic* graphs—those graphs which are continually changing in structure to reflect the evolution of the system they are representing. The design of algorithms to analyse and generate visualisations of dynamic graphs pose both a technical and perceptual problem. A range of additional issues beyond those encountered for static graphs are uncovered, from the need for efficient dynamic data structures and algorithms to the difficulties of keeping a visualisation understandable for the user while it is constantly changing its form.

1 Introduction

Graphs are one of the fundamental data structures in computer science, and are used to model structured collections of objects and the relationships between them. Each object in a graph structure is a *node*, or vertex, and the relationship between two or more nodes is known as an *edge* or link. The number of edges that a node has incident to it is called the node's *degree*. Edges may be directed, which means that each edge has a source node and a target node that it connects to. The graph as a whole may be directed, or mixed if it contains both directed and undirected edges. The nodes in a graph may be labeled and hold arbitrary information, and the edges may be weighted with some numerical value, so that different edges incur different "costs" to traverse.

Common operations to perform on graphs include traversing the entire structure, finding shortest paths between a set of nodes (Floyd, 1962), or finding graph-theoretic measures such as bridges, hubs, and authorities (West, 2001), average path lengths, or a variety of more complex roles (Canright & Engø-Monsen, 2004).

Systems Research Group,
School of Computer Science and Informatics
UCD Dublin, Ireland
E-mail: ross.shannon@ucd.ie

Other algorithms involve clustering related nodes, or removing unimportant nodes to speed up subsequent computations. As graphs have been created with many thousands of nodes, the efficiency of these algorithms is increasingly important.

The majority of research in this area has gone into efficient algorithms for working with what we will term *static* graphs. These are graphs which are complete at the time the algorithm is run, whether that algorithm is to draw the graph on a screen or simply to traverse its nodes and edges, applying some computation at each step. In this survey, we tackle *dynamic* graphs—graphs which are continually changed in structure throughout the execution time of an algorithm. These discrete changes may take the form of nodes and edges being added or deleted, the weighting of an edge being changed, or a number of other transformations which reflect changes inherent in the underlying data that is being modeled.

Each of these changes may have a significant impact on the current algorithm, as it may cause it to take a different path through the structure, or in the case of a visualisation algorithm, to modify its generated layout considerably to represent the new topology of the structure. The sensitivity of an algorithm to change is an important consideration, as even small topological modifications have been shown to result in significant changes in behaviour and layout.

Additionally, *interactive* tools which allow a user to manipulate a visualisation of a graph will necessarily employ an algorithm capable of adapting to dynamic changes. The tool may allow the user to explicitly add or remove nodes as part of a simulation, cluster and minimise related nodes or simply modify the positions of individual nodes as they wish. Each of these interactions will require a change in some layout properties and a redrawing of the graph.

There is significant crossover between the problems faced by researchers in modeling, traversing and visualising more traditional static graph structures (Díaz et al., 2002), and performing the same actions on dynamic graphs. These issues in relation to purely static
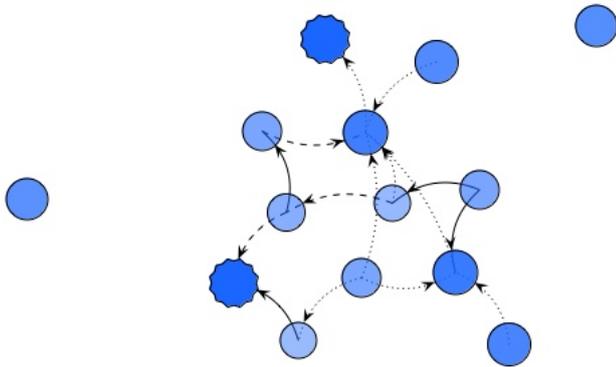
Fig. 1 A typical graph structure, showing a graph with a number of directed edges and two disconnected nodes.

graphs are ably discussed in other excellent surveys, such as Battista et al. (1994); Herman et al. (2000); or books (see Di Battista et al., 1998; Junger & Mutzel, 2003). Though these earlier surveys report on algorithms developed for static graphs, it is often found that these routines can be adapted to work on dynamic graphs. As we shall see, many of the issues researchers have run into in traditional graph drawing, such as requiring efficient algorithms for traversing graphs, and "niceness" concerns when visualising graphs (such as drawing symmetric structures, and limiting edge crossings) are also faced when visualising dynamic graphs, though dynamic graph drawing tasks have an additional set of criteria to balance alongside these concerns.

Before we continue we wish to draw a distinction between dynamic graph drawing and *non-deterministic* graph drawing, two separate problems which may be mistaken as being related. Whereas many graph drawing algorithms will take a graph as input and generate a single visualisation as its output based on a set of rules, non-deterministic algorithms will iteratively perturb the layout of a graph until it reaches some heuristically-defined quiescent state. Non-deterministic layouts like the spring layout (Eades, 1984), are generally used to find optimal layouts for a static data set. The important characteristic of dynamic graph layouts on the other hand is that the underlying data of the graph is changing, which means the algorithm must be able to adapt to these changes, but it is not a requirement that the layout be done iteratively. This is not to say that the two cannot go together; as we shall see, non-deterministic layouts do prove very useful in working with dynamic graph structures due to ease with which they are visualised.

In the general case, when visualised, the nodes and edges of a graph are intuitively drawn as points or circles linked together by straight or curved lines for edges (see Figure 1 for an example). It will be seen that the range

of graph representations are as broad as the applications they serve.

1.1 The Use of Dynamic Graphs

Dynamic graphs are required in scenarios where the underlying data set is constantly in flux, with nodes and edges being added and removed and the weights on nodes and edges changing over time. This makes them particularly appropriate for modeling communication networks in real-time, but any network that changes over time can be visualised after the fact, as long as snapshots of its state at various intervals are available.

A good example of a dynamic network structure is one's own social network. The friends and acquaintances that one builds up throughout their life form an interconnected mesh, since many of the people that you know will also know each other. Social networks exhibit all of the phenomenon we associate with graphs, such as clustering, the existence of "bridge nodes" without which the graph would become disconnected, and a wide range of node degrees. And most importantly, the graph is constantly evolving as new people are encountered or old ties are severed. If data is available of the state of a social network at a number of different time steps, these snapshots can be linked together so that the evolution of the network can be seen.

Another promising application area is the emerging field of pervasive and autonomic communication networks (Dobson et al., 2006). Pervasive systems, such as those in "smart rooms" which are aware of individuals and devices that enter and leave them, are typically concerned with *ad-hoc* topologies and transient communication channels. As devices enter the transmission range of the communication equipment in the system, they begin to transmit relevant contextual data, and in turn receive data from the system. These nodes are unstable and will no longer be part of the network when the user moves out of range. A visualisation of these interactions reveals an evolving network topology (see Shannon et al., 2007), which is useful for application designers to validate that contextual data is being disseminated correctly throughout their system.

Larger scale peer-to-peer networks (Oram, 2001), running on file-sharing networks like Gnutella or FastTrack, are founded on a fluctuating population of host computers. Snapshots of the network topology are maintained and passed around by special nodes, called "supernodes", and these views of the wider network need to be consolidated with each individual node's own view of their neighbourhood. Search and routing algorithms have to be designed to adapt to transient hosts, and the wide range of fluctuating link qualities between them. Client programs for the BitTorrent protocol will often include a visualisation of a file's "swarm", the set of nodes involved in a file's distribution. This again allows a user to get a feeling for the general health of the network.
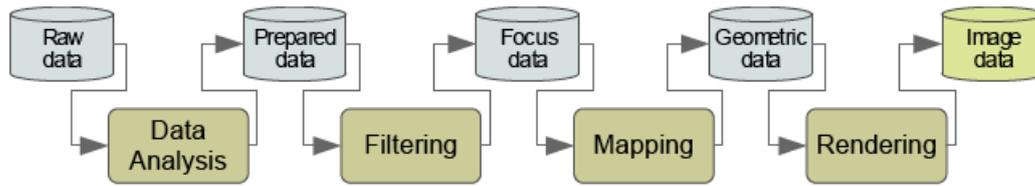
Fig. 2 The visualisation pipeline, as described in Card et al. (1999).

Similarly, the addition of pages to a website and the interconnectivity between them can be modelled as a dynamic graph problem. Watching a time series of visualisations of the graph at different points depicts an evolving network, with patterns of insertions and edge creation.

Many feature-rich visualisation frameworks have been adapted to be capable of dealing with dynamic graphs. The JUNG (Java Universal Network/Graph) framework (O'Madadhain et al., 2005) has multiple facilities for modeling and visualising dynamic graphs, making it easier for researchers to examine the issues involved.

The rest of this survey is laid out as follows: Section 2 surveys key issues in graph drawing, with specific emphasis on how they relate to dynamic graphs. Section 3 discusses a problem unique to dynamic graphs, retaining a user's understanding of a graph over subsequent versions, with a discussion of various techniques used to mitigate the effects of visible changes to a graph drawing. Finally, we conclude with a summary and suggestions for future work.

2 Generating Dynamic Graph Layouts

Figure 2 depicts the well-known Card *et al.* visualisation pipeline[1]. This pipeline (a directed graph, as it happens) shows the various stages a visualisation will go through as it is being generated, from starting as unprocessed data through to being rendered on screen. Visualisations of static graphs will only go through each step here once. On the contrary, visualisations of dynamic graphs revisit steps in the process many times, analysing and introducing new data, recomputing the layout of the nodes and re-rendering some or all of the resultant picture data.

In the following section we will briefly discuss a number of issues involved in the design of graph drawing algorithms for dynamic graphs, with reference to their position in this process.

2.1 Data Modeling

Static graphs have the advantage of being able to use fixed-length data structures such as arrays to model the

adjacency relations between nodes, which confers speed increases over the dynamic data structures required by dynamic graphs.

Data structures that have been specifically designed to model sparse graphs (that is, graphs with many nodes and few edges between them) are often significantly more computationally efficient than more generic data structures (O'Madadhain et al., 2005). This poses a problem for dynamic graphs, as there is often no way of knowing if a graph is going to be sparse or dense, and even then, this property could change over time, making the choice of data structure difficult.

Dynamic graph problems can be classified according to the leniency of updates that is permitted. A problem is *fully dynamic* if it allows unrestricted insertions and deletions, and *partially dynamic* if only either insertions or deletions are allowed. If only insertions are allowed, the problem is *incremental*; if only deletions are allowed it is *decremental*. These formalisms will often hint at the correct data structure to use in a given case.

In either case, limiting the scope of a change in the graph structure can lead to significant time savings, particularly in very large graphs. One approach to keeping changes localised in the data structure is to use *clustering* functions, such as those proposed in Fredrickson (1985), which iteratively partition the graph into a set of connected subgraphs. Each discrete update to the graph will only involve a small number of such clusters. This optimisation technique can be further enhanced in certain drawing conditions if only the nodes from the affected clusters need to be re-drawn.

For a thorough discussion of suitable data structures and traversal algorithms with detailed time complexity measures, we direct the reader to Eppstein et al. (1996).

2.2 Sensitivity to Change

Graph traversal algorithms have been shown to have high sensitivity to change, such as Kleinberg's HITS algorithm for exploring hyperlinked document networks (Kleinberg, 1999; Ng et al., 2001). The sensitivity of many algorithms is proportional to the connectedness of the graph.

Beyond the algorithm's sensitivity to change, we must also consider a viewer's sensitivity to changes in the resulting visualisation. Test subjects, regardless of their familiarity with graph layouts, have been shown to be

---

[1] Reproduced from the Information Visualization Wiki, http://www.infovis-wiki.net/

highly responsive to various encodings of node position (Dengler & Cowan, 1998). Without any explicit labeling of node role or relationship, users correlate visual patterns as logical groupings, infer relationships such as authority and subordination being represented by relative height in a drawing, or believe that a linear sequence of nodes confers some meaning to the group, as in Figure 3.

In the case where the layout of nodes carries some significance to the understanding of the structure, a layout algorithm should try to maintain the relative positions of nodes over successive iterations so that the semantic content of the graph is preserved. Conversely, if no special relationship exists between the nodes, the layout algorithm should not lead the user to make spurious assertions from persistent but incidental features.
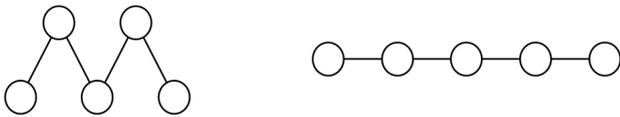

Fig. 3 Though these two graphs drawings represent the same underlying structure, users read different meanings for each. Reproduced from (Dengler & Cowan, 1998).

2.3 Aesthetic Concerns in Graph Drawing

Before they are visualised, most graphs will not have any geometry associated with them, and it is the process of generating a layout for the graph that assigns positions to each of the nodes so that they can be drawn. There exist a set of criteria which can be used to measure how well a particular algorithm performs in this task, which are chosen because of their purported aesthetic contributions to the final drawing.

The set of important aesthetic criteria differ between researchers, but an incomplete list of properties a layout algorithm should have include that it: distributes nodes evenly; tries to make edge lengths uniform; minimises or completely avoids edge crossings; keeps nodes away from the edges; maintains a balanced aspect ratio; minimises edge bends; achieves a reasonable average edge length and maximum edge length and is compact.

Some formal studies have been done which attempt to rank these "niceness" qualities by order of impact, such as those by Purchase (1998), though many come down to personal preference. Each of these criteria can be thought of as an optimisation problem, and as such, the design of a graph drawing algorithm is a balancing act between each of these considerations.

3 Preservation of a User's Mental Map

Early attempts at visualising dynamic graphs would simply treat the new drawing as a separate task and redraw the graph according to the original algorithm, which would often lead to a graph which looked completely different to its previous incarnation. Though straightforward to implement, this was found to frustrate users, as they had to abandon their earlier understanding of the graph's structure and begin again in learning how to interpret this new graph.

This issue has become one of the primary avenues of research for dynamic graphs. On top of the perceptual problems, it is simply inefficient to discard a previously computed visualisation, since it is unlikely that the geometry of the entire graph has changed, and so much of the previous work could be re-used.

When the underlying graph structure changes and a new node is added or one is removed such that the visualisation needs to be changed, it is crucial that this change doesn't interrupt or disorient the user. The understanding of the graph's structure that they had gleaned from observing the graph previously should be retained. Eades *et al.* called this "preserving the mental map" of a drawing (Eades et al., 1991).

The mental map facilitates navigation in the graph. Changes to this map should be minimal and only incurred when required. Misue et al. (1995) discuss three priorities for the mental map:

- Orthogonality: The horizontal and vertical order of nodes should stay the same after a change of node positions.
- Proximity: The distance to all nodes or to the set of neighboring nodes should be close to the same.
- Topology: The dual graph should be maintained between iterations.

3.1 Dealing With Dynamism

As previously stated, there are a range of different changes that can occur to a graph, and each has a different impact on the user's understanding of the structure. A more complete list of possible changes a graph could undergo include: node insertion and deletion, edge insertion and deletion, changes to the weighting of nodes or edges, node elision, cluster formation or dissolution, subgraph composing or expansion.

Of these, the deletion of an edge causes the least problems, since it can just be removed without affecting the rest of the layout. A less naïve approach may see the opportunity to improve the niceness of the layout, since without this edge in the way, a more balanced arrangement of nodes might be possible.

Larger changes in layout brought about by the addition of new nodes can be introduced in a more pleasing way by taking two key views of the layout, before and

after the change, and linking them together with a series of "in-between" frames of animation. This can result in smooth animations and achieves good results if a small number of nodes are moving concurrently, or if an entire cluster is moving without upsetting its internal structure (Friedrich & Houle, 2001).

Force-directed non-deterministic algorithms are particularly adept at dealing with dynamism, as their iterative approach to finding new equilibriums allows a sort of self-correcting behaviour, which lends itself well to animation. Though some changes (such as a large change in an edge weight) will cause a lot of disruption and damage a mental model, approaches like simulated annealing maintain niceness factors and can easily be made to offer animation support for maintaining accurate mental models (Lee et al., 2006).

One approach to avoiding large visible changes to a force-directed layout is to stiffen the springs over time, so that old sections of the layout tend to stay the same. This means that once a user has invested time in understanding a particular area of the graph structure, they are unlikely to need to re-learn characteristics of that part as it will remain largely static unless there is a lot of perturbation to the underlying data in that area. Conversely, new introductions to the graph move about more freely until they find a suitable resting position and then become part of the more permanent structure.

3.2 Minimising Visible Changes

Figure 4 (reproduced from Lee et al. (2006)) shows two different ways a change can be dealt with.
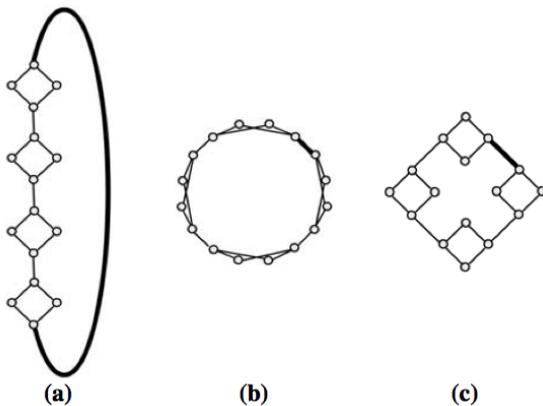

Fig. 4 An example of two different possible adaptations due to a change in the graph's structure.

Figure 4(a) depicts a graph with four connected regular clusters of nodes arranged in a diamon pattern. The introduction of an edge connecting the two outermost edges creates a problem for the layout (in this case, the

"Circular with Radial" layout algorithm from Kaufmann & Wiese (2002)). Given this new graph, it outputs a significantly poorer representation of the structure (Figure 4(b)), with low angular resolution and much of the previous structure obscured. This new layout would necessitate almost a complete replacement of a user's mental map of this graph, and significant additional effort. An alternative layout, shown in Figure 4(c), rearranges the positions of the nodes but maintains many of the patterns and relative positions existing in the original.

This preservation of structure from the previous graph layout is found to be far superior in helping a user to re-orient themselves to a new view. A range of intuitive factors affect the stability of the mental map, which were surveyed extensively by Bridgeman & Tamassia in 2002.

Recent computational advances have made the use of animation to show smooth changes between one drawing and another much more widespread. This is a significant advantage in allowing one's knowledge of a previous state of a graph to transfer to the new version, as the user can track the incremental movement of nodes over time. Additional cues can be added to support the user, such as new nodes being highlighted in a different colour for a few moments, or nodes which are to be removed from the network to slowly fade out of view.

In spite of these advances, the introduction of structural changes to the graph creates difficulty for the user, as they will always have to resolve the differences between the new view and their own internalisation of this graph. As the frequency of changes increases, this problem becomes more serious. As such, on top of the aesthetic concerns already discussed that a graph drawing algorithm should try to balance, a dynamic graph drawing algorithm should also find a good compromise between these and preserving the mental map (such as that described in Diehl & Görg (2002)).

Various techniques have been proposed to minimise the visible changes necessary and so mitigate the confusion that may result (Branke, 2001). This is also known as "maintaining dynamic stability".

For example, one rather simplistic approach is to keep the elements which existed in the previous version of the graph static, and designing an algorithm to add new nodes and edges to the blank areas of the drawing according to some metric, which is generally tied directly to minimisation of common aesthetic criteria (Papakostas et al., 1998). This works well for a small number of additions, but over time the changes build up, and with no change to the original layout of nodes and edges, we end up with a graph with lots of unavoidable edge crossings. It's clear that looser restrictions are needed to deal with these problems effectively.

A modification to this technique is to allow some local change on insertion, but nothing beyond a certain "distance" from the change. This distance is notional, but can be formulated as the set of nodes directly affected by a change together with vertices within a cer-

tain Euclidean distance from the insertion (Böhringer & Paulisch, 1990). This more relaxed set of conditions results in two things: it keeps the majority of the graph unchanged while allowing a localised area to be rearranged to accommodate a change; and it makes re-computing the layout of the graph faster, as much of the previous work in giving coordinates to nodes can be re-used.

3.3 Measuring Iterative Change

Another approach to balancing the preservation of the mental map with the need for the drawing to preserve its aesthetic qualities is to assign some score to a layout, so that two drawings of a graph before and after a change is made can be compared empirically. Effectively this involves computing a measure of similarity between two layouts and assuming that this corresponds reasonably well to the cognitive effort a user will have to expend to resolve the differences between them. Collectively these measures are known as "distance metrics".

There are a range of methods for formulating cost functions for vertex movements. A simple method to start with is to measure the Euclidean distance between a node's original position and its next position in the redrawn graph. The sum of these distances over all of the nodes is the overall cost of this change for the graph.

Relying on absolute point coordinates in this manner is not a very robust solution, as common transformations like rotation, translation and scaling will result in large dissimilarity values while the user will not have a lot of difficulty in resolving the differences. This method is also limited in that it collapses each node's representation down to a single point, thus ignoring any distinctive size or shape that the node might have had. This has the result of throwing away useful orientation information that the user will most likely find helpful to pick out "landmarks" amidst the graph.

It is easy to argue that the relative positions of a set of nodes in relation to each other is more important than any absolute position. This stems from the human visual system's adeptness at pattern matching. One approach to preserving these relative positions is what Eades et al. (1991) called maintaining the *orthogonal ordering*. Later, in Bridgeman et al. (1998) a method was suggested to compare the angles subtended between two nodes in the old and new layout. This has the nice property of allowing larger relative movements for nodes which were already a large distance apart.

The proximity of pairs of nodes in the old and new drawings is another metric we can draw on. Pairwise proximity measures also offer an advantage over absolute positions because they allow whole clusters of nodes to move together without inflating the distance metric, as long as there aren't many movements within the cluster.

Constraining the movement of nodes is a common approach to maintaining the mental map, but comes at a cost of less flexibility and therefore less adherence to the aesthetic metrics. As previously mentioned, one can begin to fix nodes to a certain position over time if they aren't moving very often. Thus, nodes which have recently been moved are more likely to move again if a reflow is necessary. This works very well as the areas a user is most familiar with do not unexpectedly change at any time. A slightly looser approach is to use the graph's dual Voronoi diagram to limit the movement ranges of nodes (Misue et al., 1995). This approach emphasises the topology of the graph, counting the lower-level pixel positions of nodes as less important.

Occasionally an animation of a dynamic graph is built "off-line", meaning that the evolution of the network is already known and the task is to stitch together snapshots of the network previously taken at various intervals. In this case rather than take each new layout change in isolation, we can take a higher level approach and aim to minimise disruption to the mental map throughout the entire animation.

4 Conclusions and future work

Though graph drawing has a long research history, the special cases of dynamic and interactive graph visualisations have until recently received relatively little attention. As more applications that call for continually updating models and visualisations are encountered, the issues unique to dynamic graphs are beginning to be uncovered in more depth.

The main concern in working with dynamic graphs is to maintain a stable view of the layout, despite the changes occurring throughout its structure. This maintenance of the user's internalisation of the graph's structure, which we call the *mental map* of the graph, allows the user to follow changes to the graph without becoming disrupted or disorientated. Techniques which facilitate this dynamic stability include localising changes to small areas of the graph and computing empirical costs for the amount of change a modification is going to cause so that it can be minimised.

These additional priorities need to be applied along with the aesthetic concerns common to all graph drawings, and finding a balance between these two sets of not always complementary forces is an ongoing challenge, particularly since the criteria which have the greatest influence over the mental map are still not fully understood. This suggests that further studies are required to accurately formalise the inputs to a user's mental map, which will allow more dependable distance metrics to be computed. Another avenue for future work is the ongoing transformation of the wide range of layout algorithms designed for static graphs into versions suitable for dynamic graphs.

References

Battista, G. D., Eades, P., Tamassia, R., & Tollis, I. G. (1994). Algorithms for drawing graphs: an annotated bibliography. *Comput. Geom. Theory Appl.*, *4*(5), 235–282.

Böhringer, K., & Paulisch, F. (1990). Using constraints to achieve stability in automatic graph layout algorithms. *Proceedings of the SIGCHI conference on Human factors in computing systems: Empowering people*, (pp. 43–51).

Branke, J. (2001). Dynamic graph drawing. *Springer Lecture Notes In Computer Science*, (pp. 228–246).

Bridgeman, S., Battista, G. D., Didimo, W., Liotta, G., Tamassia, R., & Vismara, L. (1998). Optimal compaction of orthogonal representations.

Bridgeman, S., & Tamassia, R. (2002). A User Study in Similarity Measures for Graph Drawing. *Journal of Graph Algorithms and Applications*, *6*(3), 225–254.

Canright, G., & Engø-Monsen, K. (2004). Roles in networks. *Science of Computer Programming*, *53*(2), 195–214.

Card, S., Mackinlay, J., & Schneiderman, B. (1999). *Readings in Information Visualization: Using Vision to Think*. Morgan Kaufmann.

Dengler, E., & Cowan, W. (1998). Human perception of laid-out graphs. *Graph Drawing (Proc. GD'98)*, *1547*, 441–443.

Di Battista, G., Eades, P., Tamassia, R., & Tollis, I. (1998). *Graph Drawing: Algorithms for the Visualization of Graphs*. Prentice Hall PTR Upper Saddle River, NJ, USA.

Díaz, J., Petit, J., & Serna, M. (2002). A survey of graph layout problems. *ACM Comput. Surv.*, *34*(3), 313–356.

Diehl, S., & Görg, C. (2002). Graphs, they are changing. In *GD '02: Revised Papers from the 10th International Symposium on Graph Drawing*, (pp. 23–30). London, UK: Springer-Verlag.

Dobson, S., Denazis, S., Fernandez, A., Gaïti, D., Gelenbe, E., Massacci, F., Nixon, P., Saffre, F., Schmidt, N., & Zambonelli, F. (2006). A survey of autonomic communications. *ACM Transactions on Autonomous and Autonomic Systems*.

Eades, P. (1984). A heuristic for graph drawing. *Congressus Numerantium*, *42*(149160), 194–202.

Eades, P., Lai, W., Misue, K., & Sugiyama, K. (1991). Preserving the mental map of a diagram. *Proceedings of Compugraphics*, *91*(9), 24–33.

Eppstein, D., Galil, Z., & Italiano, G. F. (1996). Dynamic graph algorithms. Tech. Rep. CS96-11, Univ. Ca' Foscari di Venezia, Dip. di Mathematica Applicata ed Informatica, via Torino 155, 30173 Venezia Mestre, ITALY.

Floyd, R. W. (1962). Algorithm 97: Shortest path. *Commun. ACM*, *5*(6), 345.

Fredrickson, G. (1985). Data Structures for On-line Updating of Minimum Spanning Trees. *SIAM J. Comput*, *14*, 781–798.

Friedrich, C., & Houle, M. (2001). Graph drawing in motion II. *Revised Papers from the 9th International Symposium on Graph Drawing*, (pp. 220–231).

Herman, I., Melançon, G., & Marshall, M. S. (2000). Graph visualization and navigation in information visualization: A survey. *IEEE Transactions on Visualization and Computer Graphics*, *6*(1), 24–43.

Junger, M., & Mutzel, P. (2003). *Graph Drawing Software*. Secaucus, NJ, USA: Springer-Verlag New York, Inc. Translator-M. Junger.

Kaufmann, M., & Wiese, R. (2002). Maintaining the Mental Map for Circular Drawings. *Proc. Graph Drawing, GD*, *2528*, 12–22.

Kleinberg, J. M. (1999). Authoritative sources in a hyperlinked environment. *J. ACM*, *46*(5), 604–632.

Lee, Y.-Y., Lin, C.-C., & Yen, H.-C. (2006). Mental map preserving graph drawing using simulated annealing. In *APVis '06: Proceedings of the 2006 Asia-Pacific Symposium on Information Visualisation*, (pp. 179–188). Darlinghurst, Australia, Australia: Australian Computer Society, Inc.

Misue, K., Eades, P., Lai, W., & Sugiyama, K. (1995). Layout Adjustment and the Mental Map. *Journal of Visual Languages and Computing*, *6*(2), 183–210.

Ng, A. Y., Zheng, A. X., & Jordan, M. I. (2001). Stable algorithms for link analysis. In *SIGIR '01: Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval*, (pp. 258–266). New York, NY, USA: ACM Press.

O'Madadhain, J., Fisher, D., Smyth, P., White, S., & Boey, Y. (2005). Analysis and visualization of network data using JUNG. *Journal of Statistical Software. Accessed on 7th October*.

Oram, A. (2001). *Peer-to-Peer: Harnessing the Power of Disruptive Technologies*. Sebastopol, CA, USA: O'Reilly & Associates, Inc.

Papakostas, A., Tollis, I., Div, N., America, N., & Irving, T. (1998). Interactive orthogonal graph drawing. *Computers, IEEE Transactions on*, *47*(11), 1297–1309.

Purchase, H. C. (1998). Which aesthetic has the greatest effect on human understanding. In G. Di Battista (Ed.) *Graph Drawing, Rome, Italy, September 18-20, 1994*, (pp. pp. 248–261). Springer.

Shannon, R., Williamson, G., Quigley, A., & Nixon, P. (2007). Visualising network communications to evaluate a data dissemination method for ubiquitous systems. In *Proceedings of the first workshop on Ubiquitous Systems Evaluation 2007*.

West, D. (2001). *Introduction to graph theory*. Prentice Hall Upper Saddle River, NJ.