An Investigation into Visual Graph Comparison

Alan Gordon Melville

SUBMITTED IN PARTIAL FULFILMENT OF THE REQUIREMENTS OF EDINBURGH NAPIER UNIVERSITY FOR THE DEGREE OF MASTER OF PHILOSOPHY IN COMPUTING

MAY 2012

Abstract

Information Visualisation is extensively used in single graph analysis. However, relatively little work has been done in the field of graph comparison. This work examines and compares the use of two standard graph representations in this area, the Node-Link representation and one based on the graph adjacency matrix. It considers which representation method is superior. In addition it explores whether it is best, for comparison purposes, to combine multiple graphs into single views or to juxtapose single graph representations.

To run this comparison a simple tool was developed and task-based analysis done using that tool to compare multiple versions of a small, locally dense, directed multigraph based on sports data. We are able to demonstrate that it is better to combine views into a single diagram, and that even for small graphs, an analyst is not disadvantaged by the abstract nature of the matrix compared to the intuitive Node-Link diagram.

Contents

Table of figures

List of Publications Derived from this Research

Melville, A.G.; Graham, M.; Kennedy, J.B., “Combined vs. Separate Views in Matrix-based Graph Analysis and Comparison”, *15th International Conference on Information Visualisation (IV), London, 2011,* Pages: 53 - 58

Acknowledgements

There are many people without whom this would not have been possible.

The test groups; many thanks.

The staff at Edinburgh Napier University, from the library assistants and the technicians to the many lecturers who were able to spare me some time to answer what must have sometimes appeared to be quite strange questions.

The research group; the other postgraduate students, especially Jamie and Kevin, now both PhDs, for the much-appreciated (and needed) whisky.

Professor Ben Paechter, my panel chair, for asking the awkward questions.

Most of all though, Jessie and Martin – Professor Jessie Kennedy and Dr Martin Graham – my supervisors who somehow helped me to finish this. Without you two the last four and a bit years of my life would have been wasted; there aren't the words, honestly.

Dedication

To my parents without whose relationship I wouldn’t be here at all.

Chapter 1 Visualisation of Multiple Graphs

1.1 Introduction

The field of information visualisation (InfoVis) is extensive, and deals with how best to display information in such a way as to allow a user to easily perceive important facts, thereby freeing them to utilise their cognitive abilities to understand the reasons behind, or to draw conclusions from, those facts. As such, it predates computers by several centuries. In the modern world the study of InfoVis has defaulted to a large degree to the field of how best to use computer-generated displays to assist in understanding. This is perfectly sensible; computers both generate enormous volumes of data, and are capable of utilising such volumes far more efficiently than any human could hope to by manual methods.

Prior to the use of computers, certain principles were established in the field, and these continue to hold good today. These principles include the elimination of unnecessary/extraneous data, and the need for understandable abstraction of visual representation. The limits of human cognition remain imperfectly understood, but it is a truism that too much information makes for poor cognition. We work best when we can reduce a problem to its vital constituents, and deal with each of these in a clear and effective manner. It is from this understanding of our natures that the most effective visual representations are produced.

One of the most common areas where Information Visualisation is applied is in the realm of graph analysis.

Almost any type of data can be represented as a graph, where aspects of the data are linked to each other in network form. However, mathematical graph theory is complex, and not always directly useful to the analyst. It is here where a visual representation of the data is helpful. Rather than getting bogged down in complex reams of text, with all the cross-checking and referencing those may require, a properly designed visual representation can allow the analyst to focus on what is important, be it structural commonality, relations between data points or the quickest route between points.

Much research has been done on how best to utilise InfoVis for the analysis of single graphs, and the field is well known with well-established concepts. However, data is not static, and old data may often be compared to newer data in order to find any changes that may have occurred, and any areas which are not subject to change. This has practical application in many fields; even relatively simple data structures are rarely set in stone, nor is there often a single definitive method of ordering and displaying a given data set. Thus when attempting to relate changes in data, or simply different aspects of the same data, to each other, we can find ourselves comparing one set of data to another. Even when we are analysing a single data set, it is often helpful to compare it to another data set whose properties are known and already understood. For these and similar reasons, we can clearly identify that there is an ever-growing need to compare graphs to each other.

It is therefore the aim of this research to apply InfoVis principles and techniques to the area of multiple graph comparison. In particular, it examines the major methods of displaying graphs, and assesses them in terms of how best to find commonalities and differences between groups of related graphs. This research utilises standard techniques of Information Visualisation, combined in a simple set of tools. In turn these tools are applied to the accepted main options in graph visualisation and a task-based assessment of effectiveness carried out.

Algorithms have been developed by many researchers to explore aspects of comparison for specific types of graph, or even specific data sets. Such algorithms mathematically and computationally apply graph theory to identify common subgraphs, check isomorphisms (q.v.), or find data trends. However due to the NP-completeness issue of general case graph comparison each algorithm is bespoke. The use of InfoVis techniques as an alternative has not been well-explored. For this reason we start by going back to basics - what is a graph, how can we display a graph, what limitations are known about displaying graphs, and so on – in order to ensure we do not overly complicate our ideas. We consider the different types of graph and how they can each be displayed. We identify the dominant techniques and explore their use for the comparison of general graph, aware that the methods used in the general graph comparison are applicable to the comparison of any type of graph.

The main contribution of this thesis is thus the comparison of display methods for multiple graphs. We explore the display of multiple graphs in matrix form, and compare the performance of these matrices against the much more common node-link rendering. This experiment involves the use of a single data set in each of these two methods of display and also the use of two different means of rendering multiple graphs on the screen at the same time. We are able to demonstrate that the greater abstraction of a matrix is at least as useful for comparing graphs as a node-link based display, and that it is advantageous to combine all the graph renderings into one single display rather than keep them as linked but separate displays. Finally, due to the nature of the tool we designed and built, we are also able to draw some tentative conclusions about the use of filters in graph comparison.

The thesis is organised into several chapters, listed below.

Chapter 2 provides an overview of Graph Theory, and an explanation of the mathematical issues involved in graph comparison. This chapter also explains why algorithmic methods of graph comparison are problematic and how Information Visualisation has been applied to graphs in the past.

Following on from this, **Chapter 3** gives an overview of Information Visualisation as applied to graphs, including the noting of seminal and state-of-the-art IV applications. It finally explains the options considered for visual graph comparison, what has been done in that field, and how and why the analysed comparison techniques were selected.

Chapter 4 describes the decisions behind the development of our software tool, and the reasoning behind the InfoVis techniques adopted.

Chapter 5 outlines the choice of test data set, and the reasoning behind it. It also describes the data set's origins and gives visual illustration of the four different graphs which were used.

Chapter 6 describes the experiments, their set-ups, and prototype test results. It explains changes to the software in the light of early experimental results, and the changes made prior to final tests with four related graphs.

Chapter 7 describes and analyses the results of the main tests and draws conclusions.

Chapter 8 relates the conclusions in chapter 7 to the general body of InfoVis research, and also proposes how these could be applied to future research in the field.

Chapter 2 Information Visualisation for Graphs

Information Visualisation has been used extensively in the analysis of graphs. Before we can examine InfoVis applications in graph analysis, however, it is necessary first to understand what a graph is. The use of the term ‘graph’ to refer to data constructions such as bar, line, and pie charts makes this disambiguation necessary. Formal (mathematical or verbal) definitions of graphs clarify the use of terms over the course of this document and enable the reader to more easily.

Graphs, ***G***, are representations of data which take the form of a set of vertices, or nodes, ***V(G)*** connected by a set of edges ***E(G)***. The edges represent some sort of relation, for example mathematical or semantic, between the different data points which may themselves be of any type. Formal mathematical definitions are given below with comments, but it is important to note that semantic data sets are difficult to define in terms of dimensional space, thus R_n (n-dimensional co-ordinate space) may have no direct semantic analogy. As such they are usually depicted in Real Co-ordinate Space, R_2 or R_3. This is not to imply that they must be two- or three-dimensional, merely that their dimensionality is not easily defined and that our mental model deals more easily with the limit of three dimensions which we are used to in the real world.

Edges can be written (x,y) or simply *xy*, and it is common practice to omit the (*G*) when defining a graph, thus we have *G* = (*V*, *E*). For the edge *xy*, *x* is known as the source and *y* the target. This is the case even when the edge is undirected (q.v.).

If connected by an edge, two vertices are considered **adjacent**, and the vertices are said to be **incident to** the edge. Adjacent edges are those with one vertex in common. The number of edges incident to a given vertex is referred to as the **degree** (or valency) of that vertex. The degree of a graph is defined as the maximum degree of any of its vertices. While a graph of low degree can have a high number of edges evenly displaced across the graph, most graphs tend to have areas where edges are concentrated and areas where edges are fewer in number. These areas of high and low **density** can be considered as subgraphs of various degree. Areas of high density can cause problems for display and analysis of graphs. In particular it is very easy for an analyst to make mistakes on the in- and out-degree of vertices amidst such areas.

A **walk** in a graph is a finite sequence of edges connecting an initial vertex v_0 and a final vertex v_m. A walk in which no edge is repeated is known as a **trail**; a trail where all the intervening vertices between v_0 and v_m are distinct is known as a **path**. A walk which returns to its starting point (i.e. $v_0 = v_m$) is called a **cycle**.

The **adjacency matrix** of a graph is a $|V|$ by $|V|$ matrix of 1s and 0s where a 1 represents the existence of an edge between two vertices and a 0 represents no edge. In undirected graphs (q.v.), adjacency matrices are symmetrical on the diagonal. For directed graphs this need not be the case, as the existence of an edge from x to y does not imply the existence of an edge from y to x.

Complete graphs are graphs where all possible edges exist, i.e. all vertices are adjacent. Complete graphs are often shown as K-numbers, thus K_5 indicates the complete graph with five vertices. K is a reference to the Polish mathematician Kazimierz Kuratowski (1896-1980) who was a pioneer in the field of graph theory. Such graphs have adjacency matrices of the form

$$\begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix}.$$

A **subgraph** of a graph *G* is formally defined as a graph *g* where $V(g) \subseteq V(G)$ and $E(g) \subseteq E(G)$ such that if $x, y \in V(g)$ and $(x, y) \in E(G)$, then $(x, y) \in E(g)$.

2.1 Types of graph

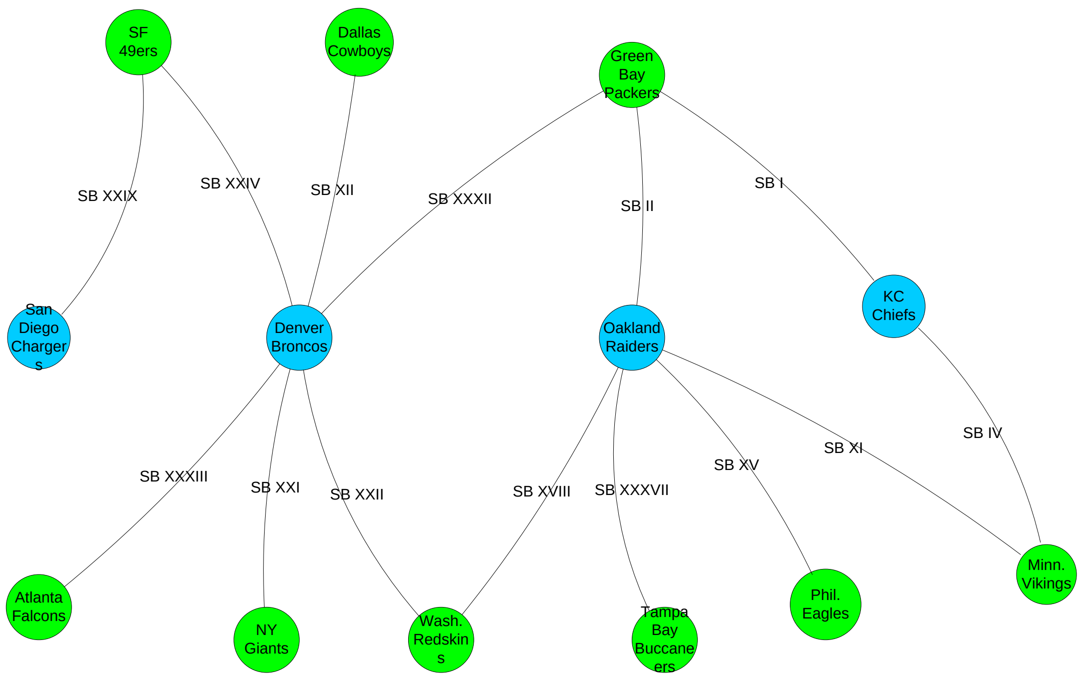
Simple graph: a pair $(V(G), E(G))$ where

$V(G) = \{(x_0, x_1, ..., x_m) \in R_m : V(G) \neq \varnothing\}$

and $E(G) = \{(x, y) \in V(G), x \neq y, (x, y) \equiv (y, x)\}$

Verbally a simple graph is a set of vertices connected by edges where no edge can loop back to its origin, i.e. a vertex does not connect *directly* to itself (loops via other nodes are allowed). The edges are undirected. Simple graphs are easily produced from human activity. Geography in particular tends to be well represented via simple graphs, such as the famous London Underground map.

The following diagram (Figure 1) is a simple graph showing the American football teams from the NFL's AFC West division (in blue) and their SuperBowl opponents. This graph is a subgraph of one used in our testing. Most of the illustrative graphs in this section will use part or all of this graph. Note that this graph only shows who played who; in order to demonstrate the results, we would need to add **labels** or **direction** (q.v.).



Labelled Simple Planar graph showing
The AFC West Division (pale blue) and
Superbowl opponents (green)

Figure 1: A simple graph with labels

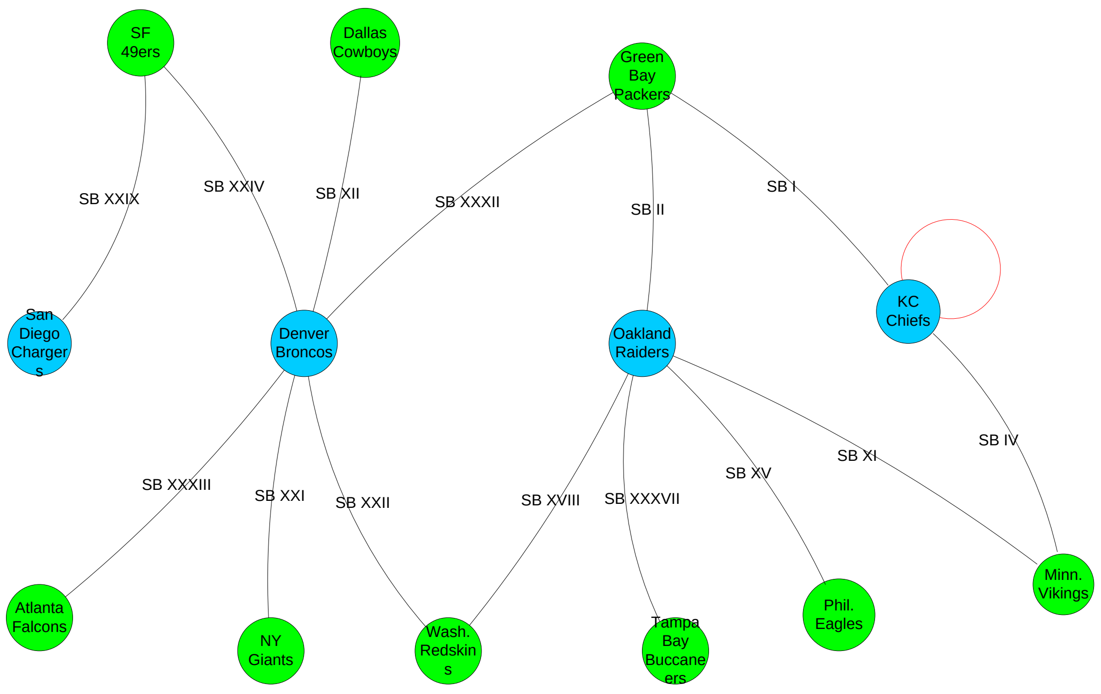
General graph, or graph: a pair $(V(G), E(G))$ where

$$V(G) = \{(x_0, x_1, \ldots, x_m) \in R_m : V(G) \neq \varnothing\}$$

and $E(G) = \{xy : x, y \in V(G), xy \equiv yx\}$

A general graph is like a simple graph, but allowing the edge (*x, x*).

Using the previous figure as an example, it is obvious that a team cannot play itself, however, for illustrative purposes, the edge in red connecting the Kansas City Chiefs to themselves in Figure 2 indicates the difference between a simple and general graph.



Labelled Simple Planar graph showing
The AFC West Division (pale blue) and
Superbowl opponents (green)

Figure 2: The graph from figure 1 turned into a general graph by the addition of one edge (in red)

Directed graph, or digraph: a pair $(V(G), E(G))$ where

$V(G) = \{(x_0, x_1, ..., x_m) \in R_m : V(G) \neq \varnothing\}$

and $E(G) = \{xy : x, y \in V(G), xy \neq yx\}$

This is rather like a general graph, in that loops are allowed, but the edges, or relationships, are one way. The relation (*x*,*y*) does not imply that there is a corresponding relation (*y*,*x*) nor vice versa. Thus the existence of the edge *xy* means that *x* is adjacent to *y*, but *y* is not adjacent to *x* unless there is also a directed edge from *y* to *x*. Edges of digraphs are sometimes referred to as arcs, although this text will not do so.

Digraphs are extremely common because they are good at showing interaction. Thus social networks, for example, will often be shown as digraphs. As previously mentioned, direction can also show a specific type of interaction, defined by the data set. Figure 3 below uses direction to indicate winners (from) and losers (to).

It is important to note that there is a subcategory of digraph where multiple relations in the same direction can exist between the same two vertices. This is particularly common when mapping semantics or designs of real-world entities such as computer programs where vertices may represent complex structures. Such digraphs are known as **multigraphs** (as in Figure 3 shown below).

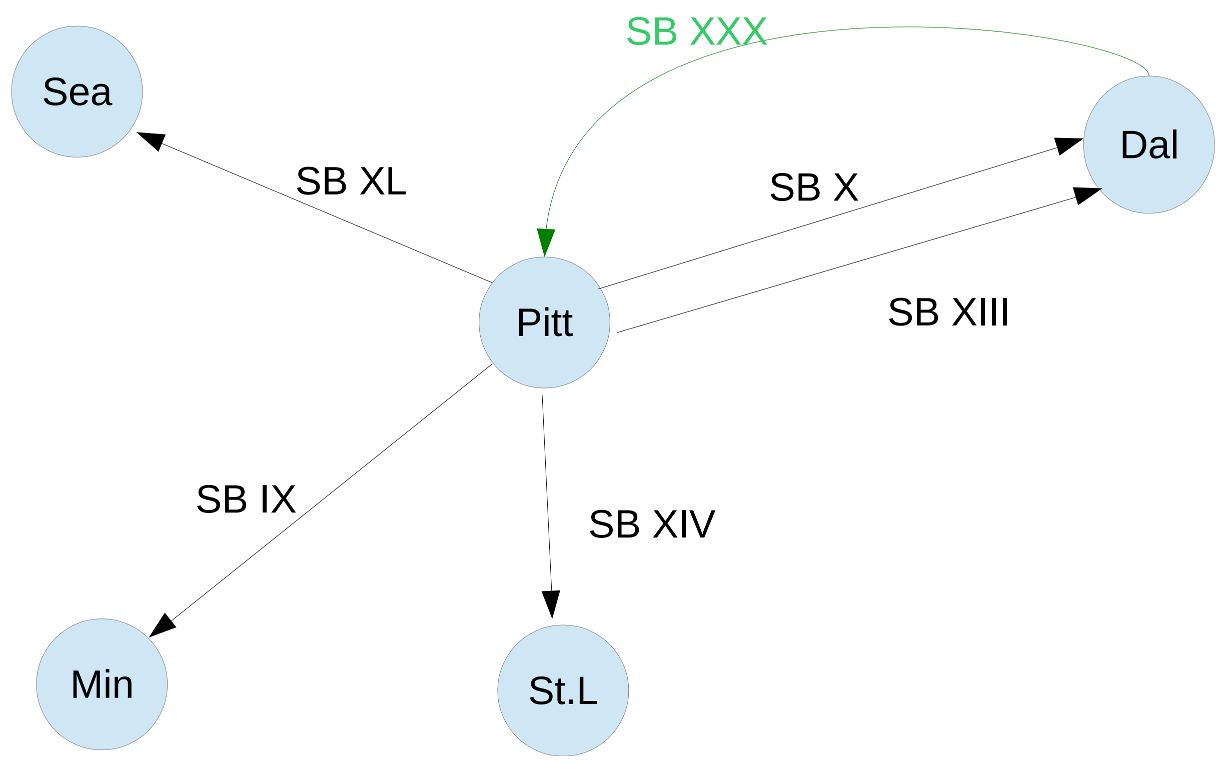


Figure 3: Labelled multigraph showing the Superbowl games of the Pittsburgh Steelers. Arrows point to losing team. Note the arrow in green representing SB XXX where Pittsburgh lost. This produces a *cycle* between Pittsburgh and Dallas.

Directed acyclic graphs, or DAG: a pair $(V(G), E(G))$ where

$V(G) = \{(x_0, x_1, \ldots, x_m) \in R_m : V(G) \neq \emptyset\}$

and

$$E(G)=\{xy: x,y\in V(G), xy\neq yx, \{x_0x_1, x_1x_2,\ldots, x_{i-1}x_i\}\in E(G)\Rightarrow x_ix_0\notin E(G)\}$$

This is a digraph where there are no paths that lead from any given vertex back to itself, that is, there are no cycles. DAGs are quite common graphs. They are frequently generated via classifications and ontologies and so feature quite widely in practical graph analysis. Figure 4 shows a DAG derived by removing the green arrow in Figure 3.

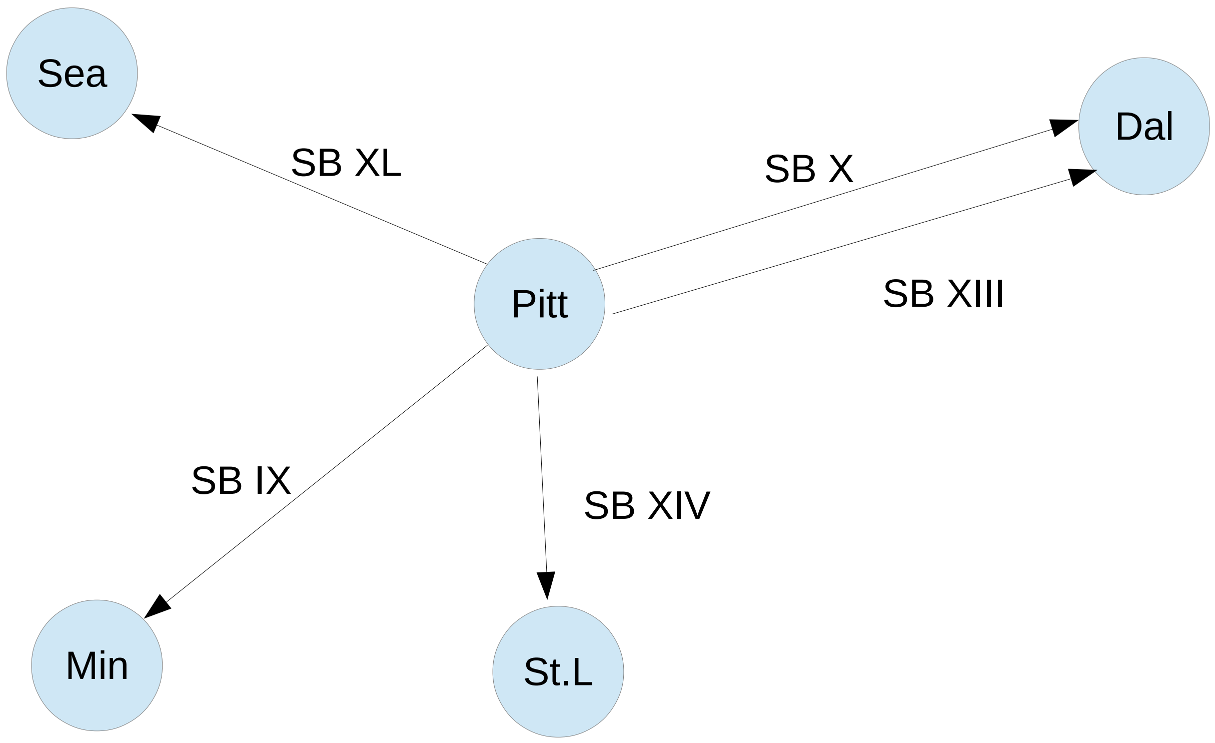


Figure 4: The same figure as figure 3 with the SB XXX removed. This is now a directed acyclic multigraph which shows which teams Pittsburgh has beaten to win its various NFL Championships.

Planar graphs are a special subcategory of graph, and are graphs that can be embedded in the plane (*R_2*) without any edge crossing. These are the subject of Brass et al's 2007 paper [1] which looks at the simultaneous embedding of multiple planar graphs. There has been a great deal of research into planar graphs in the field of mathematics, and the following conditions have been proven which planar graphs must satisfy:

- if the number of vertices $n \geq 3$, the number of edges must be less than $3n$-6;
- if the number of vertices $n > 3$, and there are no cycles of length 3, the graph may not have more than $2n$-4 edges;
- it may not contain either of the so-called Kuratowski graphs, the K_5 complete graph and the bipartite complete $K_{3,3}$ graph of six vertices, nor any subdivision (q.v.) of these. This is known as Kuratowski's Theorem. These two graphs are shown below.

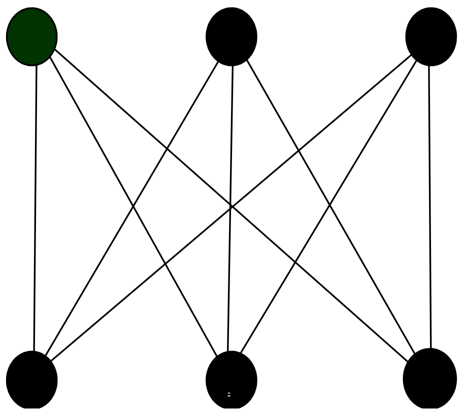


Figure 5: Kuratowski $K_{3,3}$ bi-partite complete graph

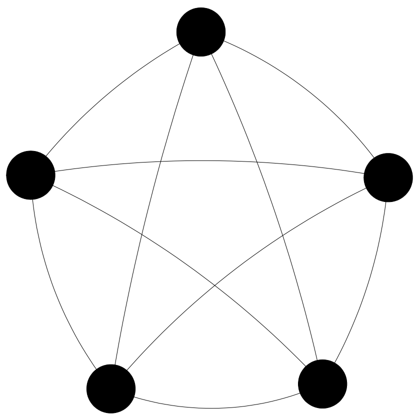


Figure 6: Kuratowski K_5 complete graph

Any graph which cannot be shown as a planar graph is known as **non-planar**.

A **labelled graph** is any graph in which the *edges* are labelled. The *label* can be defined as '*a mapping from a given alphabet to each node and each edge'* [2]. Such labels may be complex semantic descriptions of relations, or merely helpful text to differentiate between relations of the same type. It is important to note that labels need not be unique, and, especially when they represent some sort of relationship type, may be found in several places within the same graph.

All types of graph can be labelled. In practice very few graphs are entirely unlabelled, since the graph represents some real-world data, but relatively few have labels shown throughout. Figures 1 to 4 inclusive (above) are all labelled graphs. The Kuratowski graphs in figures 5 and 6 are unlabelled.

If *G* is a labelled, directed graph it is allowable that there exist parallel edges, *e*, *e'* between vertex *x* and vertex *y*. In this case *G* is called a **labelled multigraph** or more usually, though technically incorrect, just a **multigraph**. It is quite possible that *e* and *e'* have the same label in this case [2].

The following diagram, (Figure 7) representing the first forty-two SuperBowl games, is an example of a labelled non-planar multigraph. The parallel edges are shown in red. It also contains some (somewhat artificial) cycles, highlighted in green. Note that the cycle between Pittsburgh Steelers (Pit) and Dallas Cowboys (Dal) includes either or both of the parallel edges SB X and SB XIII; SB XXX is not considered parallel to these as it is directed differently, the Cowboys having won that particular game, but the edges SBX and SBXIII alone do not constitute a cycle as they are both directed the same way.

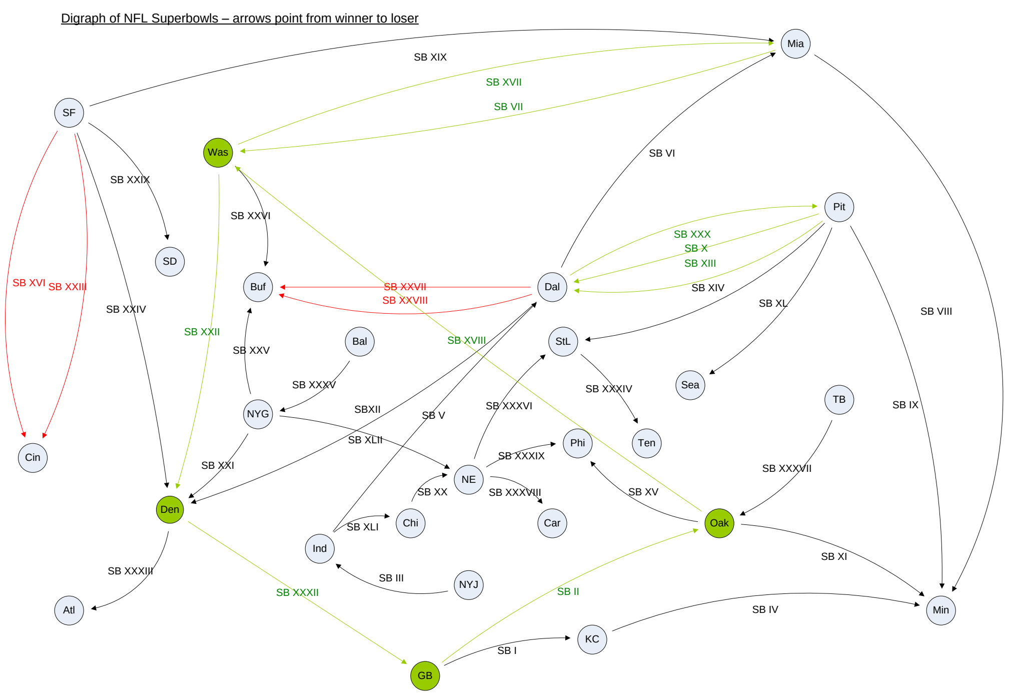


Figure 7: Example of labelled non-planar multigraph

A **weighted graph is a** labelled graph where the labels apply some quantitative value to their respective edges. The values applied to the edges are known as weights. Weighted graphs are very common in areas such as network flow analysis and a specialised variation exists known as a **Sankey diagram** (q.v.) to provide visual assistance to the analyst. In a Sankey diagram the edges increase in width to indicate greater flow.

It is of course possible for one type of graph to also be of another type, for example, a labelled simple graph, or a weighted directed graph. Likewise a subgraph of a given graph might be a specific type in and of itself, as for example in a non-planar graph which includes the complete K_5 subgraph. It is therefore possible to examine multiple graph types at the same time, if the data necessary is available, and an analyst needs to be aware of all properties of the graph being analysed if a good job is to be done.

Our research examines the wider case of graph comparison and to that end uses directed acyclic non-planar graphs with high local degree as being amongst the most complex graphs to visualise and represent clearly on a computer screen.

2.2 Graph Decomposition

Graph decomposition is the process of dividing graphs into subgraphs g_i such that the union of all disjoint subgraphs of *G* is the original graph. Two graphs are considered **disjoint** when they have no vertex in common (although there may exist an edge in the original graph joining two such subgraphs).

Graph decomposition is used extensively in the visualisation of graphs, since it is often much easier to deal with subgraphs both algorithmically and from the user's perspective. Note that is always possible to decompose a graph into a tree and a set of related edges, and research has been done on visualising this type of decomposition as well[[3].

Modular decomposition occurs when a tree is overlaid on *G* such that the root node of the tree contains *G*, all intermediate nodes of the tree contain disjoint subgraphs of *G*, and the leaves of the tree each contain a single vertex of *G*. An overview of this type of decomposition, that is, where the tree is represented by nested rectangles each containing a subgraph, is defined as a **clustered graph** [4]. Algorithms using this sort of nesting process are sometimes used to place such subgraphs into a container in order to provide an analyst with an overview of the graph structure. One example of this is the Voronoi container shown section 3. Another is the hybrid node-link/matrix display developed by Henry and Fekete which uses matrices as the containers. Clustered graphs are thus a useful visualisation tool to reduce occlusion.

A **subdivision** of a graph is obtained by inserting a vertex between two other adjacent vertices. Thus the underlying (original) subgraph remains unchanged.

The reverse of this process, removing vertices and contracting edges to obtain a subgraph, is known as **minoring**. Minoring is a particularly useful technique as it enables an analyst to quickly reduce a graph to a simpler form. The graph in figure 7 can be demonstrated via minoring to reduce to the K_5 Kuratowski graph, and thus prove it is non-planar, where it cannot be instantly shown to be by simply counting edges and vertices as previously explained.

2.3 Graph Symmetry

The subject of graph symmetry has great importance in the visualisation of multiple graphs, since we most typically want to compare one graph to another. This is particularly important when dealing with multiple related graphs such as representations of molecular structures, or when searching for changes in a single graph over a period of time. The ability to separate out symmetrical subgraphs should in theory enable the recognition and tracking of differences elsewhere to be much simplified, especially where the data set is very large. A brief overview of this subject is therefore in order and given below.

There are three types of graph symmetry:

Isomorphism

Automorphism

Homomorphism

All of these bear considerable resemblance to each other, and all are computationally complex.

General graph symmetry, or the so-called ‘graph isomorphism problem’ in mathematical graph theory is that of demonstrating whether one graph can be deformed (by some function) into another. In other words, it is a test of equivalence; two graphs, *G* and *H* are considered **isomorphic** if there is some function mapping all vertices of *G* to all the vertices of *H*, whilst at the same time retaining the same adjacencies. That is, if two vertices are adjacent in *G* they will likewise be adjacent in *H*, and if they are not adjacent in *G* they will not be adjacent in *H*.

Formally this is written:

given two graphs G=(V(G),E(G)) and H=(V(H),E(H)) there exists a one to one mapping ***f*** *such that:*

$$f : V(G) \mapsto V(H), (u,v) \in E(G) \Leftrightarrow (f(u), f(v)) \in E(H) \forall u, v \in V(G)$$

If this is the case, *f* is called an **isomorphism** of *G*, and *G* is said to be isomorphic to *H*. This can also be written as $G \cong H$. Note that isomorphism is a two-way relationship: $G \to H \Rightarrow H \to G$.

Thus in Figure 8 the two graphs, *{u,v,w,x,y,z}* and *{l,m,n,p,q,r}*, are isomorphic to each other

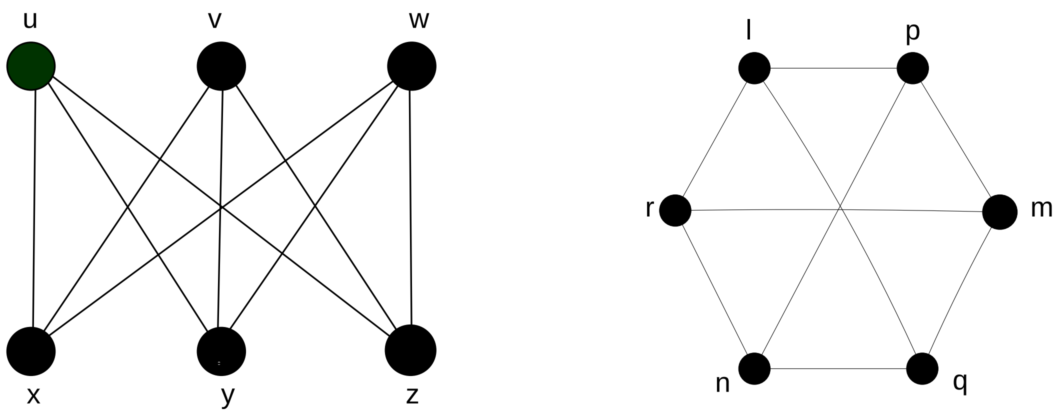


Figure 8: Two layouts of the $K_{3,3}$ graph

In the above diagram, both layouts of a bipartite complete $K_{3,3}$ graph, the vertices map as follows: $u \mapsto l, v \mapsto n, w \mapsto m, x \mapsto r, y \mapsto p, z \mapsto q$ and all connecting edges have direct equivalents. As noted earlier, for a graph to be planar it may not contain an isomorphism or a subdivision of this graph. In information visualisation, direct application of graph isomorphism is not common as most InfoVis applications to graphs deal with single graphs and the layouts thereof.

The graph isomorphism problem is known to be extremely computationally complex; as an example, the obvious brute force algorithm comparing adjacency matrices requires |G!| comparisons of |G|x|G| matrices. Clearly this is impractical for any but the simplest graphs. As of 2010, while it is quick and easy to test any given solution, this problem remains amongst the unsolved NP-complete problems.

Automorphisms are very like isomorphisms except that *f* maps the graph *G* to itself. It could be considered that this is almost a given, since in some respects if two graphs are isomorphic they are the same anyway, but automorphisms have the valuable property of demonstrating symmetry within a graph, by showing how many ways the graph can be presented. The automorphic group of a graph *G*, written $\Gamma(G)$, is defined as the set of all automorphisms of *G.* Thus any different layout of a graph *G* is an automorphism of *G*. This could reasonably have application in layout algorithms, for example in an attempt to eliminate crossings. However, the computational requirements are potentially huge and the area does not appear to have been explored in any practical application as of the date of this thesis.

Formally, an automorphism of a graph *G* = (*V*,*E*) is a permutation σ of the vertex set *V*, such that the pair of vertices (*u*,*v*) form an edge if and only if the pair (σ(*u*),σ(*v*)) also form an edge. That is, it is a graph isomorphism from *G* to itself.

As with the graph isomorphism problem automorphsim belongs to the NP class of problems. Eugene M Luks [5] has developed an algorithmic solution where the degree of the graph is bounded, but a general case algorithmic solution is not yet available.

Southwell [6] demonstrates that

i. if two subgraphs *X,Y* of a graph *G* are automorphically equivalent then both the following are true: $X \cong Y$ **and** $G - X \cong G - Y$;

ii. the reverse also holds i.e. if $X \cong Y$ **and** $G - X \cong G - Y$ then *X,Y* are automorphically equivalent;

iii. automorphically equivalent vertices must have automorphically equivalent neighbours.

This set of results has profound implications for finding symmetry in graphs and subgraphs, and may also have effect upon layout of graphs, since as mentioned above algorithmically finding a non-trivial automorphism remains unsolved. The inclusion of automorphisms as easily identifiable symmetrical subgraphs could be helpful, to say the least, in laying out complex graphs. A good example of how this might be applied can be seen in Figure 14, where the Topolayout program [7] clearly shows such areas of symmetry, although the program did not as far as we are aware use any form of automorphism algorithm.

Homomorphisms are identical to isomorphisms except that the relationship is one way, i.e. $G \to H$ does *not* imply $H \to G$. Homomorphisms are widely utilised in graph colouring where the idea is to colour each node in such a way that no edge is connected at each end to the same colour.

All of the above theory demonstrates how complex the field of graph analysis is; use of algorithms alone is rarely practical and in some cases effectively impossible. It therefore behoves the would-be analyst to find some other method of analysis.

Chapter 3 Graph Visualisation

Having briefly discussed graph theory and few of the issues and limitations arising from it, we can see why visual analysis of graphs can be useful. Humans are well-adapted by evolution to recognise patterns and are visually adept at spotting inconsistencies, following pictorial links and drawing conclusions from images where direct examinations of the underlying information are much more problematic. Our brains are better adapted to taking in data in a pictorial rather than textual form, and we process images much faster than text.

We therefore need to determine the best representations that take advantage of these cognitive abilities when analysing graphs, both when dealing with individual graphs and when examining multiple graphs. Such research pays considerable attention to ease of understanding and the reduction of cognitive load, as well as efficient use of computing resources.

There are several methods of showing a graph on a computer screen:

a) textually, which is highly precise, but visually inconvenient, and, for large graphs, generally unhelpful. A simple example of textual representation might be of the form *V={a,b,c,d,e}, E = {(a,b), (a,e), (b,d), (c,d),(d,e)}*

   Clearly, while describing both *V* and *E* as mathematical sets is easily understood for smallish graphs, it is impractical for graphs with hundreds or thousands of vertices.

b) via an **adjacency matrix** (q.v.) which is an elegant means of input to computers, and lends itself easily to mathematical manipulation. An adjacency matrix can also be put into tabular form to aid understanding; the axes of such a table are made up of the vertices in *V*.

c) display onscreen as vertices and edges; the vertices being some form of dot, blob, or polygon, the edges as arcs or lines between pairs of vertices. There are many different algorithms which have been applied to this type of visualisation. These are commonly known as node-edge (or **Node-Link**) displays. Layouts of Node-Link displays are more extensively explored in section 2 of this chapter.

d) display of vertices as shapes (frequently rectangles or circles) with edges being shown by
   a. touch, known as a **contact diagram** [8]
   b. overlap, known as an **intersection diagram**, or if using circles, a **coin diagram**

e) Display of vertices as horizontal lines and edges as vertical lines. This has been applied to planar graphs and some trees and is known as a **visibility representation**

f) Display of the graph as a **tessellation representation**. This is very similar to a contact graph with the following specific differences: the shapes are always rectangles; tile boundaries intersect only if the corresponding vertices/nodes are incident; the union of all tiles is a rectangle. Tessellation representations have long been applied to planar graphs

g) As a **Sankey diagram** which is designed to demonstrate network flows. The edges vary in thickness according to the amount of traffic between the vertices, which may be sources or sinks (of traffic) or indeed both. Interactive diagrams of this type have been shown effective in analysis of city energy usage [9]. However, they are suitable only for weighted graphs, and were designed for a specific subset of that limited group. Thus they are not widely used outside their designed field.

In the majority of the above, only specific types of graph are usable. Tesselation diagrams are suitable only for planar graphs; visibility representations [10] for planar graphs and some trees; contact and intersection diagrams for very small, undirected graphs.

Occasionally a designer may adapt one of these limited diagrams successfully. For example, Graham et al [11] [12] utilised a similar technique to contact diagrams when drawing large hierarchies(trees), although in these cases it was derived from the requirement to save space onscreen whilst retaining focus and context. In this example nodes were represented as rectangles and contact between nodes on different layers implied an edge while contact between nodes on the same layer (level of the tree) did not. In some respects this is also very like a tessellation representation (q.v.). Figure 9 is taken from Graham and Kennedy's 2008 [13] paper showing how this adaptation was applied.

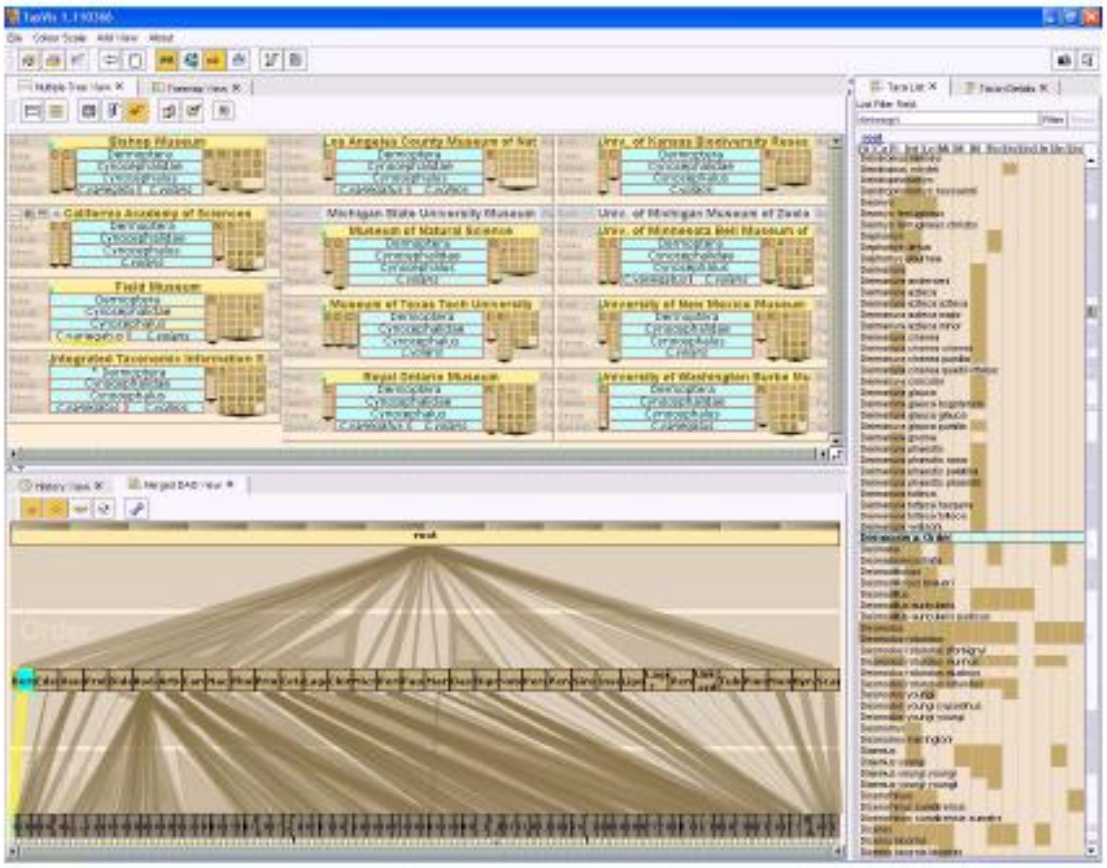

Bishop Museum
Los Angeles County Museum of Nat
Univ. of Kansas Biodiversity Resea
California Academy of Sciences
Michigan State University Museum
Univ. of Michigan Museum of Zoolo
Field Museum
Museum of Natural Science
Univ. of Minnesota Bell Museum of
Museum of Texas Tech University
University of New Mexico Museum
Integrated Taxonomic Information S
Royal Ontario Museum
University of Washington Burke Mu
Dermoptera
Cynocephalidae
Cynocephalus
root
Order

Figure 9: Multiple tree representation from Graham and Kennedy 2008 [13]

Since our purpose is to look at general graphs, including directed non-planar graphs, we are primarily interested in matrices and Node-link diagrams.

3.1 Laying Out Node-link Diagrams

Let us consider the famous London Underground map as it is usually drawn; a collection of circles (stations) and lines (tracks), with colour differentiating between the various tube lines (subgraphs).

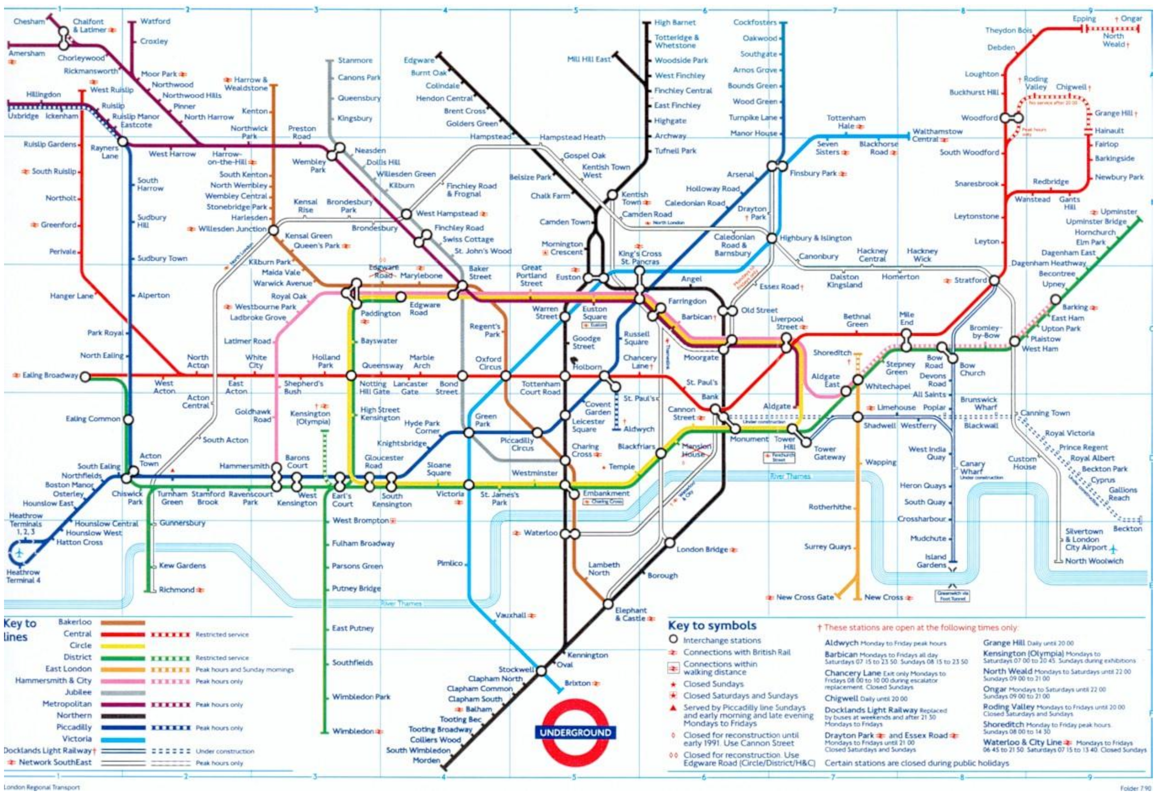


Figure 10: The London Underground as of 1990

This is a very typical display of its type; when dealing with graph data, a Node-Link format is the most common visualisation method. This is because it is intuitively fairly obvious to the user, and because it is also fairly easy to manipulate algorithmically. However, getting a usable Node-Link layout is not always a simple matter. The above diagram has been developed and improved over many iterations for some seventy years, since it was first published. The difficulty is even greater where we have highly complex graphs, and many different methods have been researched.

When displaying a graph onscreen, an important consideration is how best to place the vertices and edges in such a way as to make the graph as clearly understandable as possible. Because Node-Link displays of graphs have difficulty dealing with dense clustering of vertices, or generally dense graphs [14], there have been many attempts to layout the graphs in such a way as to take full advantage of the space available. Collectively these are known as **layout algorithms**.

There are many techniques, which are discussed briefly here. It is important to note that all layout algorithms attempt to make a graph more aesthetically pleasing (and thus understandable) but that different aesthetic constraints (for example minimising the number of crossings, or limits on the size of the graph, or a specific need to cluster subsets of related vertices) may conflict with each other, or be difficult to deal with simultaneously within the algorithm, so trade-offs will be unavoidable.

3.1.1 Force-directed layouts

The first and most common technique is **spring-embedding** or the **force-directed layout** and there are literally dozens of variations on this theme.

Originally described by Eades in 1984 [15] the idea behind spring embedding (or ‘spring-mass embedding’) is that nodes are considered as masses or steel rings and edges as springs connecting the rings. The vertices are placed in some initial layout and the system iterated to simulate its ‘natural’ movement into a minimal energy state, which is according to normal thermodynamic laws considered stable. In the algorithm vertices are given a small mutual repulsion factor which prevents them moving too close to each other so that the graph does not ‘bunch up’ into a degenerative and uninformative nucleus. Variations of the original algorithm have been used extensively in graph visualisation, and it is now the most common technique in layout algorithms [4] [3].

In a spring embedding, those vertices with strong relation to each other are pulled closer together than those which are unrelated or less closely related, providing a structured view of the relationships between different vertices. The idea is to provide a reasonably understandable view of the graph on the computer screen.

The process is fairly simple:

```
Choose an initial layout;
Iterate as follows:
    Calculate the effect of attractive forces on each vertex;
    Calculate the effect of repulsive forces on each vertex;
    Displace the vertices according to the forces acting upon
    them.
```

In Eades' original algorithm each vertex was moved before the forces on the next vertex were calculated. Fruchterman and Reingold [16] calculated all the forces before moving the nodes, and also introduced the idea of an 'optimal' distance between nodes in order that any display remains easily understandable. Various modifications have been made over the years by different practitioners, such as edge repulsion to help prevent bunching [17], or varying masses (weights) to take account of perceived nodal importance [18], but the basic ideas behind the algorithm remain the same.

A simple example of a force-directed layout is shown below, drawn by the *neato* algorithm [19]. For graphs of this small size, this type of algorithm works well.

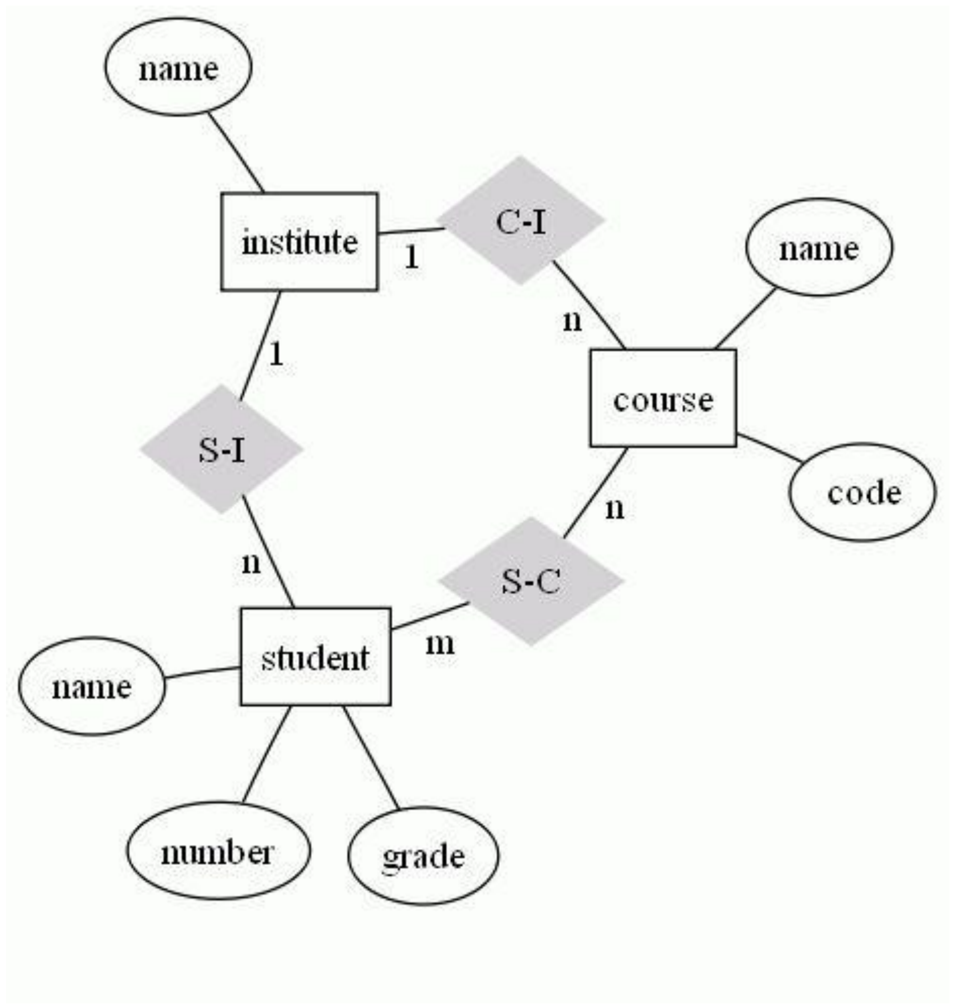


Figure 11: Simple example of force-directed layout, drawn by the *neato* algorithm, and taken from the user manual for that algorithm, 1992, page 12

The major problem of force-directed layouts is that they become inefficient and have difficulty in displaying a helpful image when the graphs have 100+ nodes. This can be alleviated to a degree by amalgamating the embedding with other methods or algorithms, such as Voronoi diagrams [20] or clustering [21] and they remain an important and easily implemented means of utilising the available screen space to the best advantage.

3.1.2 Simulated Annealing

In simulated annealing an attempt is made to copy the process of liquid crystallisation. It is not exclusively associated with graph theory; the technique has been applied extensively to general optimisation problems. While effective, simulated annealing has the drawback of being relatively slow, and highly resource-intensive in computer terms. Thus early uses of it in graph drawing were rapidly overtaken by force-directed layouts and variations thereof. However, more recent work [22] has applied it to the problem of 'beautification', i.e. of improving a layout to make it more aesthetically pleasing and thus (hopefully) clearer. The basic principle of the algorithm remains the same, but the application is designed towards more evenly distributing vertices and reducing the number of crossings, so that there is less clutter and the graph becomes easier to understand. It is not widely found in use for large graphs however due to the slowing of response times it can incur.

3.1.3 Shaped Layouts

Shaped layouts are those based on a specific geometry. Typically they will be one of two types; **orthogonal layouts** and **radial layouts**. Both are used in an attempt to simplify the understanding of the graph displayed by reducing visual clutter and limiting the graph to an overall familiar shape; either rectangular (cuboid) or circular (ovoid).

Orthogonal layouts draw edges as straight lines along the x-, y- or z- axes or some combination of these with 90˚ *bends*. In some fields, such as electrical circuit design, such layouts have long been standard practice; the more general case is covered here.

If all edges involve only the x- and y- axes such layouts are known as *planar orthogonal drawings*. Only graphs of degree four or less can be represented by a planar orthogonal drawing [23]

One of the most important ideas with orthogonal layouts is how to minimise both the number of bends in the display and the number of bends in each edge. Such minimisation makes the layout simpler to understand. This is often combined with attempts to minimise the number of crossings, and different methods may emphasise one minimisation at the expense of the other. For example, the following diagram shows two orthogonal layouts of a graph, one of which minimises the number of bends, while the other minimises the number of edge crossings.

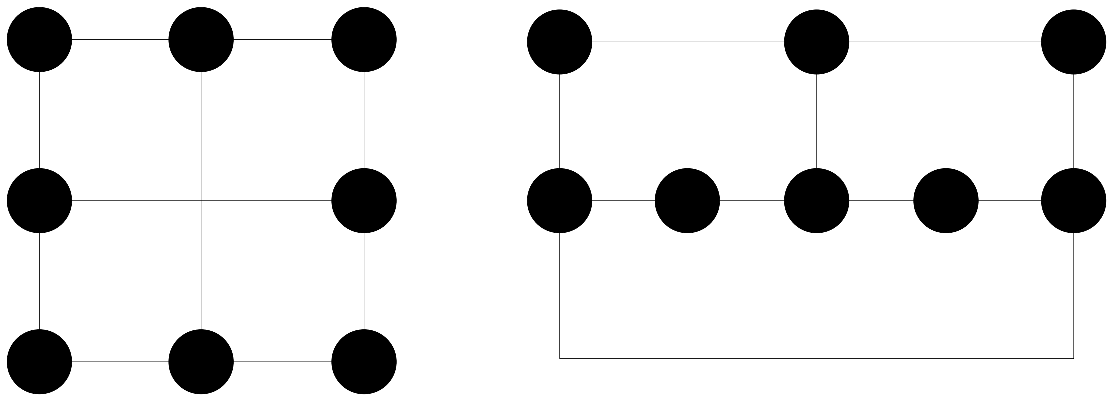


Figure 12 Copied from Di Battista et al, fig 2.4

Radial layouts are those which attempt to place the vertices in a circle or arc, or a series thereof. The idea is to make use of the screen space available effectively whilst providing a recognisable structure for a user. One such readily available algorithm is *circo*, part of the *GraphViz* toolkit. A simple example of a graph produced with this algorithm is shown below.

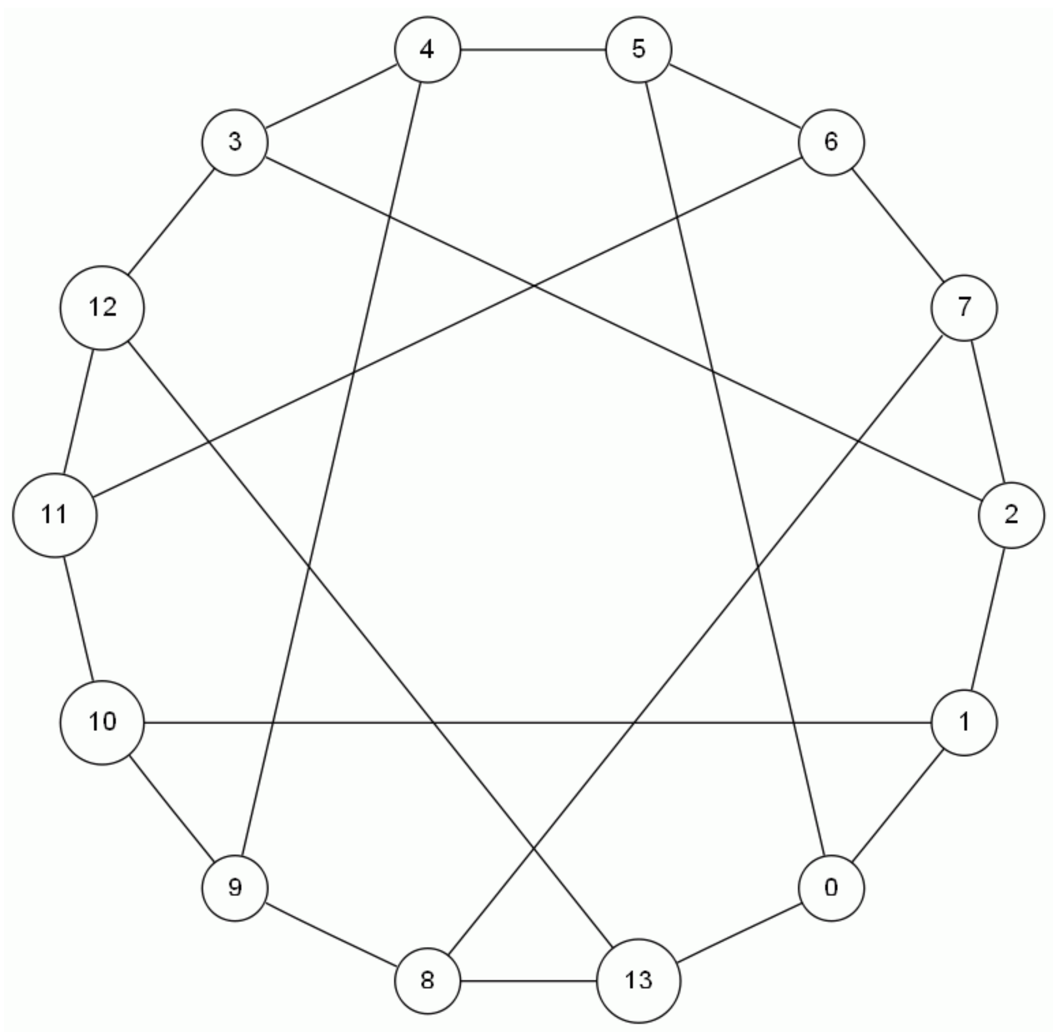


Figure 13: a radial layout of a small graph using the *circo* algorithm

A recent and interesting development of this type of layout involves the problem of graphs with two different types of (related) nodes. The RadialLayout [24], designed for visualisation of protein and protein domain interactions, sets one type of vertex (the domains) as a circle, with the other vertex type (individual proteins) as an outer ring. This minimises crossings and is quite easily understood. It is also quite efficient in terms of utilisation of the available screen space, as is common with circular-type layouts.

Other radial-type layouts involve the utilisation of alternate geometries and projection to a circle or sphere such as the Hyperbolic Browser [25]. These are technically a type of Focus+Context analysis technique, but they do include the idea of projecting the data on a circular rather than rectangular plane in order to maximise the available data, so are mentioned here.

3.1.4 Multi-level layouts

Multi-level layouts is a term used to describe layout algorithms that combine multiple layout algorithms. Due to the size and complexity of modern data sets, and the fact that single algorithms tend to have difficulty in scaling up to match, researchers started to investigate building a graph in a recursive manner. This increases the efficiency of rendering. In multi-level layouts it is typical to recursively apply a coarsening operator thus dividing the input graph into a hierarchy of coarser graphs, so that coarse high-level graphs have vertices which represent subgraphs of lower level graphs. These have been found to both improve running time and layout quality, and there are several examples. Coarsening is very closely related to the ideas behind clustered graphs.

Archambault et al [7] took this approach one step further, and rather than simply coarsen the input graph attempted to break it down into salient features. In this idea, rather than use a single algorithm the graph uses several different algorithms, each tuned for the feature in question. These features are collectively considered to be the graph's **topology**, and the principle of their 'TopoLayout' tool involves breaking down the graph into topological subgraphs, rendered in sequence. Thus the input graph is searched for connected components (to allow independent layout of subgraphs), trees, biconnected components, complete subgraphs, clusters, and so on, and these are rendered using specialised algorithms. This produces rather effective layouts quickly, and remains close to the state of the art in current graph layout algorithms. An example of such a graph is shown below.

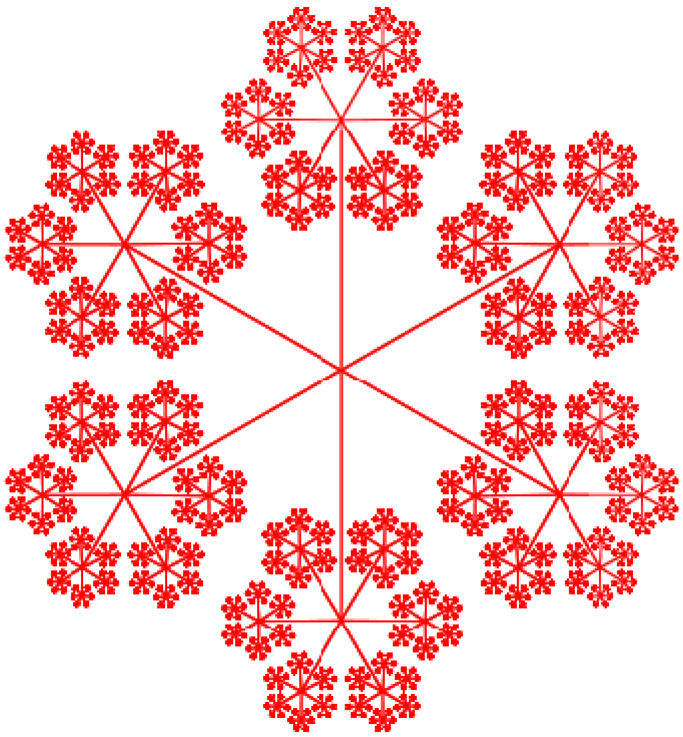


Figure 14: figure 6, page 10 of [7]

3.1.5 Aesthetic considerations in layout algorithms

Measuring the effectiveness of different visualisations is not a simple matter, given that it is both highly contextual and user-dependent. However, several researchers have made efforts in the area [26, 27, 28, 29, 30]. Most of this research involves the measurement of tasks involving the actual use of different visualisation tools. Measurement, in this case, means for the most part how quickly and how correctly the task in question was accomplished, and has normally been averaged across small groups of users without experience of the tool(s) in question, but usually with some experience either of the problem domain or of general interactive computer use.

The criteria against which different layouts were compared in existing literature [31] [26] [27] [30] [28] [29] include the following:

For effectiveness of node-link displays (layout algorithms)

Length of edges;

Uniformity of edge length;

Bends in edges;

Angle of bends;

Orthogonal vs. non-orthogonal layouts;

Number of edge crossings;

Flow direction;

Symmetry of layout;

One of the most interesting results of this area of research was the finding that most of the selected criteria have little measurable effect in general, but that their effects were often circumstantially significant, with the primary circumstance being the size of the graph. Larger graphs clearly showed that artificial forcings, such as orthogonality and circular layouts, made even moderately complex graphs much more difficult to understand. This finding was backed up by more recent research by Dwyer et al [14] who also compared computer-generated layouts to ones produced with human input.

This led to the concept of a 'critical mass' above which the various metrics may become significant. This is most profound in the effect of a large number of edge crossings, which would tend to imply that the effects of large-scale occlusion (on understanding) would be found to be significant. In this context, the finding by Ghoniem et al. [32], that graphs of more than 100 vertices are difficult for a user due to the effects of both edge and vertex occlusion becomes unsurprising.

More recently, Holten and van Wijk [33] demonstrated that the use of an isosceles triangle to indicate direction was much more effective than traditional lines/arcs with arrowheads.

3.1.6 Occlusion

As stated above, one of the major difficulties with Node-Link displays is that as they get larger, and/or move into three dimensions, nodes and/or edges can start to overlap, occluding each other. The matter of when, and when not, to allow occlusion therefore becomes a significant issue. Riehmann et al [34] found that relative edge importance, which tends to be concomitant with thickness in Sankey diagrams, meant that more important edges tend to occlude others in these diagrams but that this was not overly significant in their particular context. In other contexts occlusion may obviously become a greater problem.

Various means of limiting this problem have been looked at in the ongoing development of graph visualisation tools. Occlusion can often be alleviated by e.g. rotating, colouring, or animating the display in order to simplify more detailed selection [35] [36] or by altering the shape of the edges themselves [3] [25] but it remains a difficult issue to resolve. The difficulty of redrawing such displays to provide a better result whilst not adversely impacting retention of the user's mental map adds further complication, not least that of computer response time.

In particular local occlusion, where a graph has areas of closely clustered vertices and edges that are difficult to interpret, has been found to give users difficulty, and the use of containers and clustering as a solution has been driven by the need to alleviate this problem. Some of these solutions have been quite elegant and successful, as Figure 15 (from [20]) indicates, but occlusion remains an issue with Node-Link diagrams and one that is unlikely to find a permanent solution.

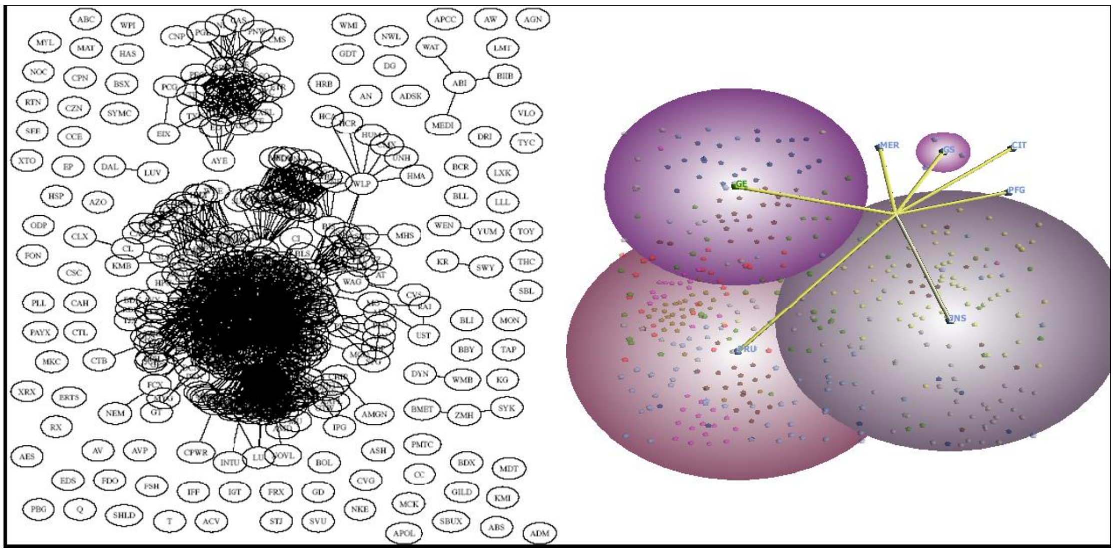


Figure 15: Voronoi containers (right) as used by Kumar and Garland and compared with the equivalent force-directed graph (left) produced by the GraphViz tool. Taken from Kumar and Garland 2006 fig 2

The reason that occlusion will tend to remain a problem in graph visualisation is down to the ongoing issue of data set size increase. A number of very large data sets are today being analysed which are the accumulation of large-scale observations previously difficult to deal with: microarrays, for example can have quarter of a million data points; financial or census information is often the aggregation of hundreds of thousands or even millions of individual data items. Even using containers and broad overview approaches which are designed specifically for context retention and general overview, it can be difficult to draw accurate and reliable conclusions, and still more to gain meaningful detail insight from dense and cluttered Node-Link displays.

One possibility is that a combination of matrix and Node-Link representations could be the best way forward. Such a combinatory tool was developed by Henry and Fekete [37], and demonstrated that the addition of linked Node-Link displays to matrices (in this case for showing social networks) was highly effective in path-related investigation, reducing the difficulties of matrix-only representation in this area. A major advantage of this type of hybrid diagram is that it avoids the computational requirements of constructing complex mathematical structures such as the previously shown Voronoi containers.

This brings us to the second major method of displaying graphs, the adjacency matrix.

3.2 Matrix-based Visualisation

All graphs have an adjacency matrix. In the case of an undirected graph, the adjacency matrix is symmetric on the diagonal; for directed graphs this is not necessarily the case as the edge *xy* does not imply a corresponding edge *yx*.

It is only relatively recently that matrix-based visualisations have become a popular topic, and the main reason for this is quite simple. As an *nxn* matrix, the adjacency matrix is planar, easily scalable to screen size, shows all edges and all vertices, and is not subject to that great bugbear of large data sets, occlusion. It is this last that has popularised matrix-based concepts. Node-Link displays are by and large more intuitively comprehensible, but as the number of nodes grows, they become much more complex and difficult to follow, as described in the previous section. Hence a few researchers in the InfoVis field have started to explore ways in which the strengths of matrix-based methods can be taken advantage of.

Matrix-based displays do however have their own issues for the user.

In the first case, a matrix is not intuitively obvious to a non-expert. While there are fields in which matrix-based representation is the norm, outwith those humans do not instinctively relate a (somewhat) abstract matrix to a network of adjoining points. In particular, finding a path between two connected but well-separated vertices can be extremely difficult without some form of path-finding algorithm to help.

Secondly, a large matrix can make it more difficult for a user to focus on that part of the display they are actually interested in; the sheer volume of blocks can be somewhat overwhelming.

Set against those drawbacks the at first glance overwhelming advantage of occlusion-free oversight is not so clear. This has led to comparisons of the usefulness of matrices with that of Node-Link displays in single graph analysis.

3.3 Comparisons between matrices and Node-Link displays

Different criteria have been used for comparing Node-Link displays and matrices than were used in the research of metrics previously discussed. The criteria used for comparing Node-Link displays against matrix-based displays so far have been:

Size of graph (nodes);

Density of graph (number of edges per node);

The reason for the difference in criteria is clear; a matrix-based display always lays out in the same fashion regardless of which Node-Link layout is selected. Thus the characteristics of the graph itself become more important than the specific algorithm used for the Node-Link display. That said, the research in comparison of matrices vs. Node-Link displays is still in its early stages and since no definitive ‘best’ Node-Link layouts can be said to exist, it would be worthwhile exploring specific Node-Link displays in comparison with their matrix equivalent. So far little appears to have been done on the ordering of rows/columns within the matrix.

It is notable that there has been relatively little research on comparison between domain-specific visualisations, and tasks appertaining thereto. Ghoniem et al. [32], although utilising primarily social networks, have considered generic tasks only when comparing matrices against Node-Link displays, and utilised only one layout algorithm, albeit a very common one.

Likewise, the earlier work of Purchase [26, 27, 28, 29, 30] in analysing graph aesthetic effects focussed more on general tasks than on tasks specific to a given data set/domain. This is a useful development in the direction of graph visualisation research, as it offers the possibility of techniques with a wide scope of application, however it does run the risk of failing to discover new data-specific techniques which are subsequently found adaptable to other data types.

Keller et al [38], in comparing engineering DSMs (a form of adjacency matrix) and Node-Link displays (using the *neato* layout algorithm), did look at real-world semantic data, and found that experience/knowledge of the data set had measurable and statistically significant effect on the efficacy of use; using German road graphs, they found that their German-born subjects, familiar as they were with German geography, were more accurate and faster at performing the tasks set, and that this effect seemed to hold regardless of whether matrix or Node-Link layouts were used. This implies (or more accurately reinforces) that the nature of the user is highly important when designing a visualisation tool. It also indicates that when designing a generic tool it is wise to utilise a data set that does not introduce (or at least minimises) accidental bias. The ideal data set should be easily understandable, but at the same time not widely familiar.

The need for ease of understanding often tends to make researchers prefer real-world to abstract data sets, as the real-world data is more easily understood even to lay testers. The variety of real-world data also offers an enormous range of possible graphs. It is often much easier to simply adapt a real-world data set to fit a research question than it is to produce an abstract data set applicable from first principles.

The tasks used the Keller and Ghoniem experiments [38] [32] to compare Node-Link displays with matrices for single graph analysis were:

Finding specific node(s) given a label or other criterion;

Finding the in- and/or out-degree of a node;

Finding paths between nodes;

Finding the shortest path between nodes;

Estimation of the number of nodes;

Estimation of the number of edges;

Finding a link between two specific nodes;

Finding a common neighbour to two (or more) given nodes;

At first glance, it would be expected that matrices would perform well at finding the degree of vertices and the links between individual vertices, and be relatively unaffected by the size or density of the graph. Occlusion likewise should not be an issue with a matrix. By contrast, Node-Link diagrams would be expected to be strong in the areas of finding neighbours and paths, since they provide a more intuitive visual object, but are likely to be vulnerable to high local density, occlusion, or very large graphs.

The results of the current research bear out these expectations. Both methods were found to have the expected stronger and weaker areas, although both are comprehensible in all the tasks tested. Thus a reasonable conclusion is that a combination of matrix and Node-Link representations could be the best way forward. Such a combinatory tool as already mentioned was developed by Henry and Fekete [37], and demonstrated that the addition of linked Node-Link displays to matrices (in this case for showing social networks) was highly effective in path-related investigation, reducing the difficulties of matrix-only representation in this area.

One area which has not been addressed regarding matrices is how to deal with multigraphs. Clearly a solution will have to be found to the problem of having more than one edge between vertices before matrix-based visualisation can be considered useful for all graphs. However, matrix-based methods do seem to offer at least a partial solution to the ‘large graph’ problem which Node-Link displays cannot at present do effectively.

3.4 Multiple graphs

There are several real-world circumstances in which a user may need to look at multiple graphs:

Examination of changes in a data set over time, e.g. relations between molecules during a chemical reaction

Extrapolating behaviour of an unknown data set from comparison to a known similar data set, e.g. prediction of the effects of a process on a new species via examination of its known effects on a similar species

Examination of an unknown data set via comparison with multiple known data sets, e.g. identification of an unknown skeleton by comparison to existing species' skeletons

Direct comparison to identify similarities/differences in structures of e.g. comparing management overheads and effect in two or more business organisations

Ultimately, whatever the primary reason for the examination, the user needs to identify similarities and differences between the graphs viewed, hence the problem becomes one of **comparison**.

There are thus many potential uses for a tool that enables direct comparison of graphs. For example, one 'generic' task that might benefit from comparison of data sets, indeed requires comparison of data sets, is that of defining (and thus eliminating) noise from data. Given a domain expert user, it may be possible to use comparison of matrices as a speedy means of identifying anomalies between data sets and it is hoped to explore this area during the project by injecting an anomaly into the test data set and see whether it is noted.

There has long been a tradition in the scientific community of graph comparison as a means of analysing multiple data sets [39] [40] [41]. Much of this has involved the use of algorithms based on mathematics and the theory of graphs. In the field of graph theory itself much of this has revolved around graph matching and the isomorphism issue. Graph matching falls into two categories: ‘exact’ and ‘inexact’.

3.4.1 Exact graph matching

Exact graph matching means matching graphs/subgraphs with other graphs in such a way that it is possible to perform a one-to-one mapping between vertices. There are three types of exact graph matching (monomorphism, graph-graph isomorphism and graph-subgraph isomorphism). Of these, subgraph-graph isomorphism is known to be NP-complete, and that may also be the case with graph-graph isomorphism. Monomorphism, being a special case of injective homomorphism, i.e. being a one way rather than a reversible function, does not apply to the area of visual graph comparison at this stage and will be ignored for the rest of this section.

Despite the difficulty of the isomorphism problems there has been much research [42] on the use of algorithms which reduce the computational complexity via some restriction (such as only working on planar graphs [43]) and for most circumstances there is one available which will work adequately for a given set of graphs. However, being algorithmic and not visual, a great deal of contextual information can be lost to a user when utilising these methods exclusively.

3.4.2 Inexact graph matching

Inexact graph matching involves the mapping of a graph or subgraph to another where vertices do not map precisely (i.e. they may map inexactly). This involves mapping either single vertices from one graph to vertex clusters in another, or clusters of vertices in one graph to clusters in another. There has been considerable research in this area, mostly involving pattern/feature matching and shape analysis algorithms and measures of similarity [44]. The primary issue with inexact matching algorithmically is how best to measure similarity of the compared graphs; various measures of similarity exist, but ultimately most have similar basis, viz. they compare the 'average' distances apart of the vertices of each graph. For example, Caelli and Kosinov [44] used a variant of a well-known mathematical cluster validity index taken from the field of data mining. Similar clustering techniques have been utilised in the integration of databases in an attempt to map the data entities [45].

3.4.3 Current examples of visualisation for graph comparison

As we move into the field not only of graph display and analysis, but of graph comparison and matching it is instructive to look at existing areas of Information Visualisation applications. Remarkably these are relatively thin on the ground; most graph matching is algorithmic rather than visual.

One recent ongoing example however is the EMAP (Edinburgh Mouse Atlas Project) project developed at Edinburgh's Medical Research Council's Human Genetic Unit. A visualisation tool for this project was developed at Heriot-Watt University, Edinburgh [46] in which anatomy ontologies are represented as DAGs for comparison purposes. The EMAP project is particularly interesting since it involves comparing the anatomy ontologies of the mouse foetus through the entire development (of the foetus). Thus terms move around within the ontologies, the hierarchies gradually change, new terms appear (as new organs develop) and existing terms disappear (as the development they refer to changes). This means that there are cases of mappings between ontologies which are not one-to-one, and adds a considerable degree of complexity to the problem.

Various techniques were used in the development of the tool, including 2d and 3d graph displays; it is notable that one of the issues users had difficulty with was the occlusion of vertices. In essence though, the use of visual techniques was found to assist users, reducing the cognitive load considerably. An example of the tool's 3d browser is shown in Figure 16 below.

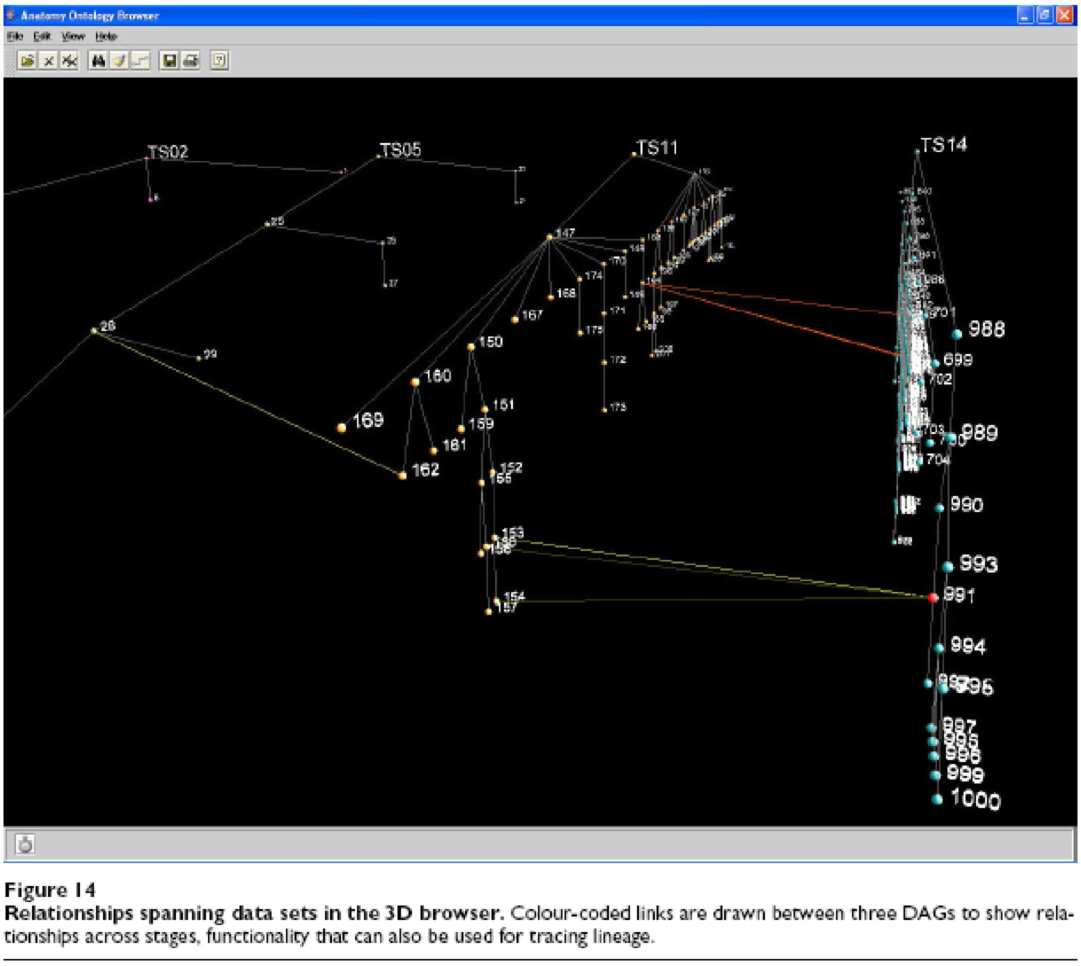


Figure 14
Relationships spanning data sets in the 3D browser. Colour-coded links are drawn between three DAGs to show relationships across stages, functionality that can also be used for tracing lineage.

Figure 16: Taken from Dadzie and Burger 2005 page 8 [46]. The occlusion problem is clearly shown, as are the many-to-one relationships between the different ontologies

Li, Eades & Hong [47] have stated that node occlusion is the single greatest problem in graph visualisation. The various studies by Purchase et al. [26] [27] [30] [28] and the research by Ghoniem, Fekete, Henry and their various partners [3] [32] [37] confirm that it is indeed a serious problem with Node-Link displays. Clearly when combining multiple Node-Link displays the problem will likely increase as common edges start to occlude each other.

Given the size of, for example, typical ontologies (several hundred vertices), and indeed of most modern data sets, it is apparent that some means must be found to solve this problem when comparing graphs too. One possible solution is the use of matrices.

As previously discussed, adjacency matrices are planar and do not suffer from the problems of node occlusion. Moreover they have a very useful property when the matrices of two graphs are compared. That is, they can easily demonstrate isomorphism.

Two graphs, *G1* and *G2*, which are isomorphic have adjacency matrices *M1* and *M2* such that there exists a one-to-one permutation function which will transform *M1* into *M2* [48]. What this means is that, if two vertex sets are mapped to each other, and the matrices are then ordered identically, then any isomorphic subgraphs will appear as identical patterns. But we must be able to ensure that the mapping is correct. Thus one possible solution to the occlusion problem above would appear to be the use of matrices.

An example of this is shown below, using the same two layouts of the $K_{3,3}$ graph from Figure 8.

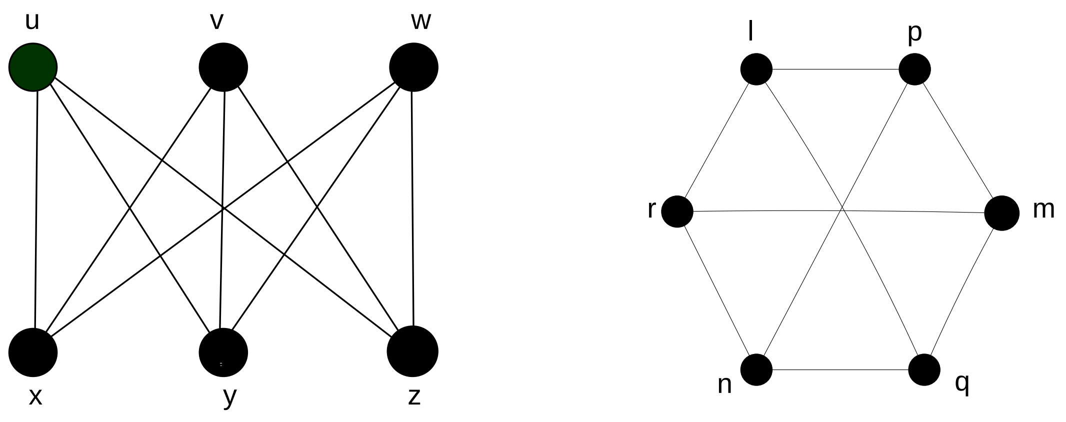


Figure 17: Two layouts of the $K_{3,3}$ graph

The adjacency matrices for the above graphs, with the correct mapping of vertices, show the commonalities clearly.

	u	v	w	x	y	z
u				🟩	🟩	🟩
v				🟩	🟩	🟩
w				🟩	🟩	🟩
x	🟩	🟩	🟩			
y	🟩	🟩	🟩			
z	🟩	🟩	🟩			

	l	m	n	p	q	r
l				🟩	🟩	🟩
m				🟩	🟩	🟩
n				🟩	🟩	🟩
p	🟩	🟩	🟩			
q	🟩	🟩	🟩			
r	🟩	🟩	🟩			

Figure 18: Adjacency matrices for the graphs in Figure 17

It is clear from the identical matrices that the two graphs are isomorphic. This is a highly useful property because it enables a designer to take advantage of the human propensity for pattern recognition. Since the general case algorithmic solution to finding isomorphism is NP-complete as previously explained, this may eventually allow for a non-algorithmic solution, bypassing the entire issue.

Of course in the above case we knew precisely how to order both sets of axes, as we already knew which vertex mapped to which. The question arises then, without such knowledge, does it remain easy to see such commonalities?

Moreover, is it possible to apply InfoVis principles and techniques to the area of multiple graph comparison, and if so, how best do we accomplish this?

This provides our primary research problem: **can we compare the use of matrices and Node-Link diagrams in graph comparison, and in doing so, can we demonstrate which is superior?**

To answer this question we must examine the major methods, node-links and matrices, of displaying graphs, and assess them in terms of how they can be utilised to find commonalities and differences between groups of related graphs. We will be obliged to develop a simple set of tools, and apply these tools towards the typical main options in graph visualisation.

This research therefore must utilise standard techniques of Information Visualisation, and carry out a task-based assessment of the comparative effectiveness of each of our two visualisation types.

Chapter 4 Data Factors and Selection of a Test Data Set

Before any beginning could be made on answering the research question it was first necessary to design and build a tool that would allow comparison of multiple graphs against each other both in node-link and matrix representation. Before that could happen however it was necessary to either find a test data set to use, or at the very least to consider what type of data might best be used. Accordingly we proceeded to examine the various data factors that might affect the research, and to examine potential data sets for use with any proposed tool.

There were several factors that we needed to take into account in the design of this proposed tool. Some of these evolved from the data sets available, and others from the nature of graphs and graph comparison.

4.1 Basic considerations

Our primary desire was to test comparison of general graphs, therefore our data sets used in this experiment had to be small enough to be easily manipulated, as we would need multiple versions, and understandable to non-expert users, as we did not have a readily available pool of workers in any given field to test our hypothesis with. They also needed to be in the public domain to allow other researchers to check if they wished the validity of our experiments.

We decided to look for a data set which would offer itself to comparison with similar (or related) data, but nonetheless provide examples of the primary problems faced in single graph analysis. The reason for this desire was that we would be testing different techniques of visualisation against each other. Therefore we wished a data set that would demonstrate the typical problems faced by visual graph displays, viz., node/edge occlusions, local areas of density and sparseness, non-planarity and asymmetry. We also wished, if possible that our data set would be a multigraph, since if we could demonstrate a superior visualisation for a multigraph, we would still have demonstrated superiority for simpler graphs with one edge between any given pair of vertices. In addition, we were interested as to whether it would be possible to easily display multi-edges in matrix format as that had not been widely examined before. In effect we wanted as complex a graph as possible while still remaining within known limits for single graph analysis.

Our readily available data sets were of one main type; directed graphs with comparable but different static vertex sets. While it remained true that our primary desire was to test general graphs, directed graphs are very common and any conclusion regarding comparison of directed graphs can equally well be applied to undirected graphs, especially in the discussion on matrix-based displays as undirected graphs have much simpler adjacency matrices, being diagonally symmetric, than directed graphs of similar degree based on the same vertex set.

4.2 Data set options

There were several data sets that we might potentially have access to, and we examined each for suitability.

.

4.2.1 Option 1: class relations

Working as we did in a department of computing, we considered using a representation of data structures within versions of an OO computer program. These are however not readily understandable to anyone outwith the computing field. Moreover the nature of objects is such that calls tend to go both up and down in effect producing large areas where the graph will appear symmetric. To confirm this belief we examined the call relationships for various versions of a large program with many hundreds of classes. Each version produced matrices that were symmetric about the diagonal (see Figure 19), which was not what we wanted to consider for a test case as this would mean we could not apply our results to the (asymmetric) directed graphs

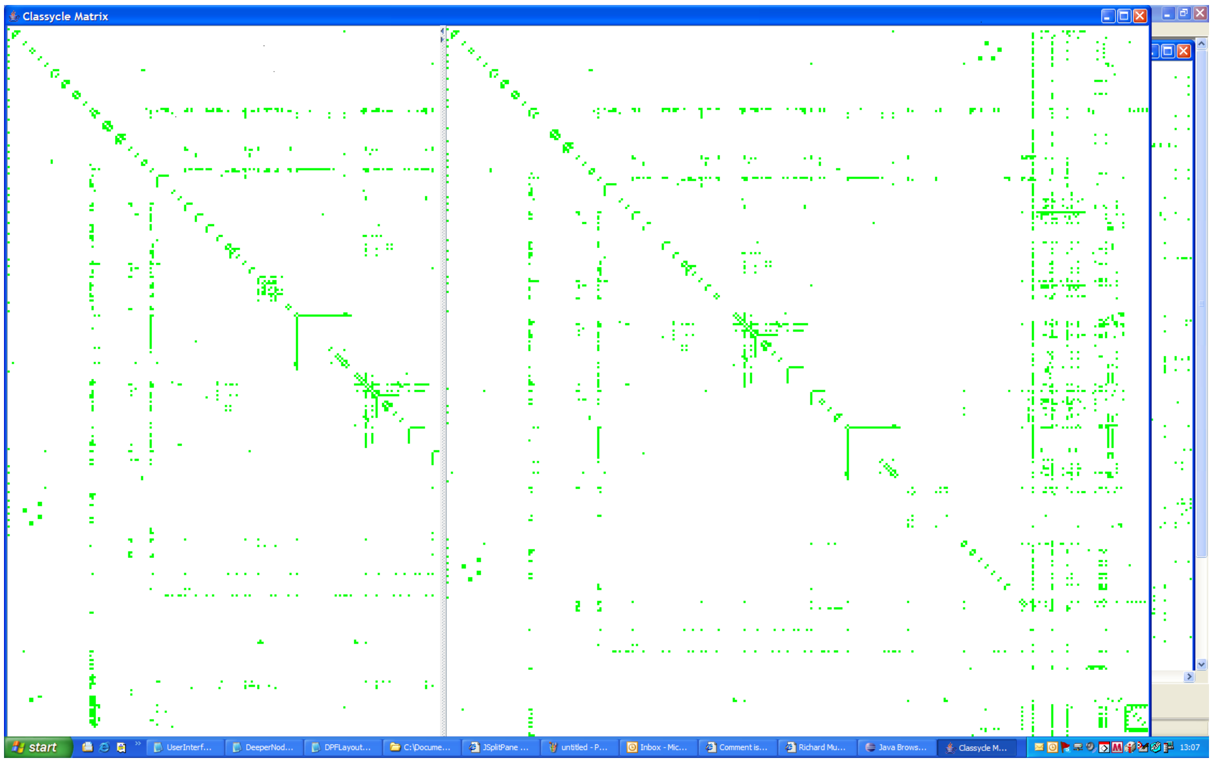


Figure 19: The above is taken from an early matrix prototype. Note that both matrices appear in large part to be symmetric about the diagonal.

The other problem with these graphs was their sheer size (around 8-1200 nodes and 7-10000 edges). This precluded any node-link visualisation for reasons already explored in the literature while at the same time making them more or less incomprehensible to any but an expert designer. We therefore decided to look elsewhere.

4.2.2 Option2: Ontologies

We also considered the use of ontology-based data sets, but concluded that

a) differences between similar ontologies are very much more understandable to expert users than to our expected lay testers,
b) ontologies tend to be very tree-like in structure and thus short of the desired local density and edge crossings
c) ontologies are difficult to provide vertex mapping for due to the frequent one-to-many mappings. This requires an expert user to decide between the vertex mappings.

Again we chose to reject this data set.

4.2.3 Option 3: Sports data

The final option we had available was to utilise some sort of sports data. This would be readily available as most such data is in the public domain, and a set could we hoped be easily found which would meet our needs.

After some exploration of its properties therefore, we decided upon a small graph of circa 40-50 vertices based upon sports results. Such a graph is easily adaptable to multiple variations, and typically includes several vertices of high degree (representing successful athletes/teams) and so providing a suitably complex layout problem. It is also directed (winner to loser), likely non-planar, and easily understood by lay testers.

We chose a data set which consisted of a graph of sports results, specifically the results of the annual SuperBowl game that decides which is the best team in America’s National Football League. The graph showed teams as vertices and the games as directed edges from the winner to the loser. This graph is shown in Figure 20 below.

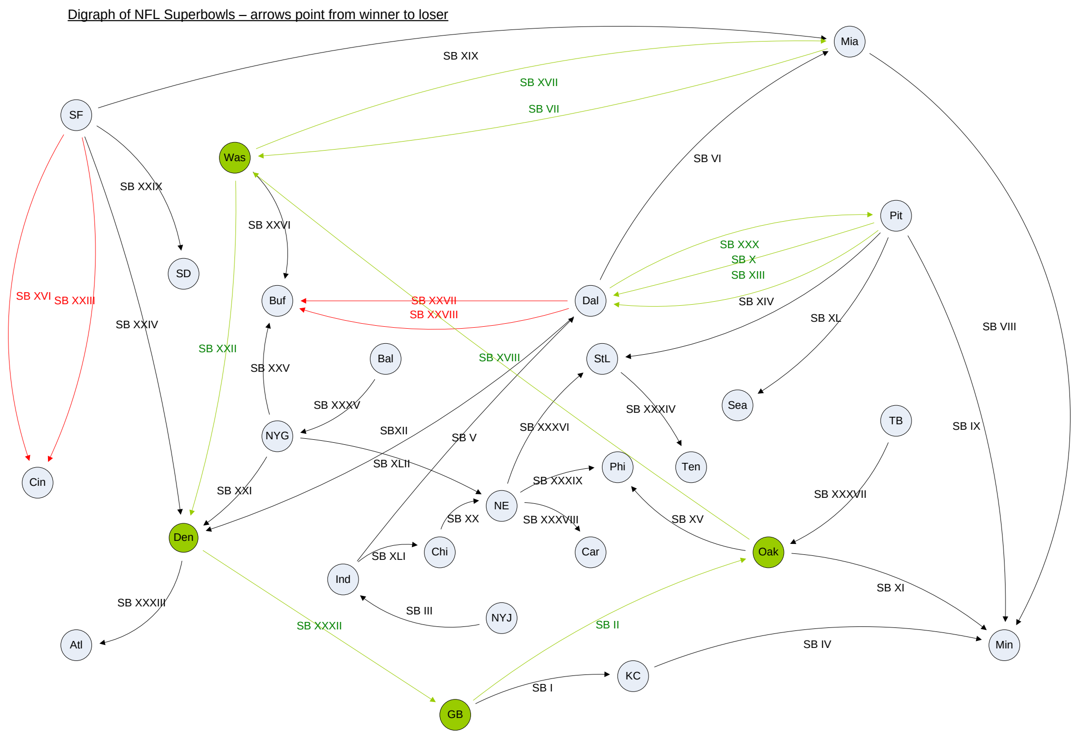


Figure 20: SuperBowl results, to SuperBowl 42. Arrows point from winner to loser

This graph was first of all provably non-planar, as planar graph comparison had already been explored. Although simply calculating the ratio of edges to vertices as mentioned in Section 2.1 did not work, the use of minoring (q.v.) demonstrated that the graph can be decomposed to produce a **K_5** Kuratowski graph and is therefore non-planar. Moreover, although sporting competitions are an easy subject to understand, the graph covers a sport which is not well-known in the UK, and therefore is much less likely to bias test results with prior knowledge. It contains (somewhat artificial) cycles – shown in green above – and also has multiple edges between some vertices (shown in red).

We therefore selected this data set for several reasons:

- It was easily comprehensible by non-expert users;

- It was small (28 vertices and 43 edges) enough to manage the number of differences to test our questions with, but complex enough to require testers take care when finding and interpreting those differences;
- The graphs produced from this data set are directed non-planar multigraphs with sufficient edge crossings and occlusion to offer in microcosm the typical difficulties involved in graph visualisation;
- It can be represented in different ways, allowing for direct comparison between versions;
- It has the rare property of uniquely identifiable edges – each game has a specific number which could be added as a label. This enabled users to more easily identify each specific difference and us to evaluate user accuracy.

We chose to use two graphs based on historic data which would enable information about the teams to be directly obtained from the comparison process. We could then make (different) changes to the historical data to each one of these graphs to produce four closely related graphs with specific differences our testers could be asked to find.

The important point of these differences is that they are not obvious; it requires a user to look for common connections in order to find them. The task is made easier by the useful property of unique edge labelling which allows users (and us as researchers) a ready check on the accuracy of their answers but is nonetheless non-trivial as it requires a logical extrapolation of what the differences might mean.

The only issue which was a cause of worry was the relatively sparse matrix form of each individual graph. However, the most likely occasion of an individual matrix being shown was when it was side by side with other such matrices, so this was considered of minor importance. Given the small size of side by side matrices the clarity of each matrix so laid out would be vital, so any sparseness of the matrix would likely be advantageous as less prone to confuse our testers.

We therefore took full advantage of the possibility to display the same data in multiple ways. The vertex set of our graphs represented teams, the edges the results of each game.

One graph would represent the teams by their current location (or in the case of the New York teams NYG or NYJ respectively). Presenting the data set in this way, the Colts, for example, are shown as IND based on their current home city of Indianapolis, the Rams as STL (St Louis), the Raiders as OAK (Oakland) and so on.

The second variation showed the teams by their home location when the game in question was played. Several NFL teams have moved from city to city over the years, and would be shown different locations for different games. Thus the Colts were shown as BAL (Baltimore) for SuperBowls III and V, but IND for SuperBowl XLI; the Rams under LA for SuperBowl XIV, and STL for SuperBowls XXXIV and XXXVI; the Raiders as OAK for every game they played except XVIII when they are shown under LA. In all there are four games shown differently between the historically accurate versions of these graphs.

We then made small adjustments to the historical results and produced from this adjusted data set two more graphs with locational differences in a similar manner.

Having selected our data set, we proceeded to develop a suitable tool to support its use in answering our research questions. This development was in part carried out while deciding on the data sets to be used and is described in the next chapters.

Chapter 5 Designing the tool

Typically, graph comparison is seen in terms of the matching of patterns within graphs [49]. However there are several other aspects to graph comparison. Gallacher [50] gives a good summary of the major issues involved in pattern matching and of algorithmic means of meeting those challenges. These include inexact matches and, more interesting from our point of view, semantic matches [51] where vertices and/or edges are matched according to data type/content as well as structural issues.

In the field of visualisation, while there has been work done on comparison of specific graph types, such as planar graphs [1] [18] or trees [12] [13] [52], the general graph case has not been as widely examined.

Sairaya et al. [53] examined the issue of graphs associated with time series data and how best to indicate changes to the graph data at points in the time line; Telea et al [54] looked at combining the graphs of RDF schemas with instances of the schemas, but in both cases the visualisations considered means of making alterations to node representations in order to show similarities or differences. Whilst these were effective, such alterations are perforce data dependant rather than independent of both data type and graph type. Accordingly, we decided not to include such a data-bespoke functionality in our tool.

Other research in this area has also tended to look at some means of merging graphs for comparative purposes, and considered how best to compare Node-Link displays [55] [56], but has neither looked at matrices or compared the combined Node-Link against non-combined.

Having decided on our proposed data set we knew we would be working with directed graphs. We therefore needed to examine comparison issues for directed graphs. These would have to be addressed in our proposed software tool.

5.1 Using directed graphs

The questions arising from directed graphs as a data set involve the search for commonalities and differences. These can be broken down into three main areas as follows:

5.1.1 Vertex mapping

This is the mapping of the vertices to a common framework.

If we look at our potential test data sets, for example, with software comparison, vertex mapping can be between classes or between routine and subroutines. In our available sets, the software comparison would, initially at least, have been at the class level rather than the function level. Even so, this provided an enormous data set with many hundreds of vertices and several thousand edges, which is why we chose not to use it.

For, an ontology, vertex mapping involves some means of semantic comparison of ontological terms. While it is not impossible that one-to-many mappings may arise between software versions, it is much more likely that this will occur between ontologies. This last, as previously mentioned, was our primary reason for rejecting the use of an ontology as a data set. It would be much easier to map vertices when we had no one-to-many mappings to deal with. Vertex mapping is clearly easier with a graph based on sports results as variants of the graph will still map relatively simply. For example, if our graph had been based on Spanish football, the team of Real

Madrid in graph *a*, will map to Real Madrid in graph *b,* and to the city of Madrid in graph *c.*

The question was, did we want user input to be required in order to confirm which vertex maps to which? We considered that the project was about testing the best method of comparison, so any vertex mapping should be done by the tool itself prior to the visual representation.

Once vertex sets are mapped to each other, any visualisation tool must ensure that mapped vertices are placed as closely as possible in the same location for each data subset being compared. To do otherwise would produce an unnecessary cognitive load and make the visualisation difficult to use effectively.

5.1.2 Edge mapping

Again we can take our potential data sets to demonstrate the issues involved.

In comparing software, edge mapping is relatively simple; there are few different types of relation (edge) to deal with, and each data set has the same types of relation. This should allow automatic comparisons without user input being needed.

In an ontology there may be several semantic relations, and not all relations may map directly to a relation in another ontology. Thus we have a much more complex problem which will require user input.

While vertices must map to the same locations as described above, the edge sets are expected to vary; there is little likelihood of entirely identical edges appearing between data subsets. That said, direction of edges is an issue for node-link renderings, as an analyst must be able to determine without error whether an edge runs *xy* or *yx*. It is important that the edges of any given type are shown in the same manner, however. As with the placement of vertices, this reduces cognitive load.

In the event, our sports-based data set did not have this issue; each edge represents a unique event, and can therefore be uniquely identified and thus clearly mapped to its equivalent in each variant graph.

5.1.3 Subgraph mapping (isomorphism)

This is the most complex of the three areas. This is due to the fact that it is necessary to take into account possible vertex insertion and minoring. In theory, if the vertices are mapped and the edges are the same for any given subgraphs, the subgraphs are isomorphic; however user input remains necessary to confirm whether this is in fact the case and that the mapping is in fact correct.

There are likely to be very similar data structures within any given data subset, simply because they are structurally efficient for that data type.

Thus, for example, in our potential data set based on software dependencies, we might find a design pattern arising in two variants of the graph, such as one of the standard OO patterns, but they may have been used for different purposes. It is important that the patterns be recognisable, but it must also be possible to confirm a negative correlation between the two subgraphs as well as a positive one. Finding the same subgraph structure is not alone enough to declare sub-graphs are actual isomorphisms or homomorphisms of each other.

Nevertheless one of the fundamental attributes of the human visual system is its ability to see patterns. Any tool must take advantage of that capability. As stated earlier, and shown in Figure 18, adjacency matrices whose mapped vertices are in identical positions show isomorphism as identical patterns.

5.2 Display options

The above suggests that a matrix-based visualisation might be superior in comparing subgraphs as well. However, before that can be tested, there is a complicating factor; we are not limited to displaying our graphs side by side. As discussed by Gleicher et al in 2011 [57] we can instead choose to combine them, or to use a visual tool that explicitly shows relations between the graphs, known as **explicit encoding**. However it is a highly complex matter to explicitly encode relations between many graphs.

An example of explicit encoding can be seen below, taken from Collins and Carpendale's 2007 work [58] where they show relations directly between three graphs laid out as successive pages. As can be seen quite clearly this 'book-type' layout only allows comparison between a 'middle' graph and two others. We intended to look at more than three graphs, and so did not consider this suitable. In addition we considered that the extra information shown would be large for four or more graphs, possibly tending towards the size of the graphs themselves, and so did not utilise explicit encoding. An important facet of InfoVis is to keep representations as free of extraneous information as possible to minimise cognitive load. Given our likely non-expert users, we wished to keep the visualisation as simple as possible.

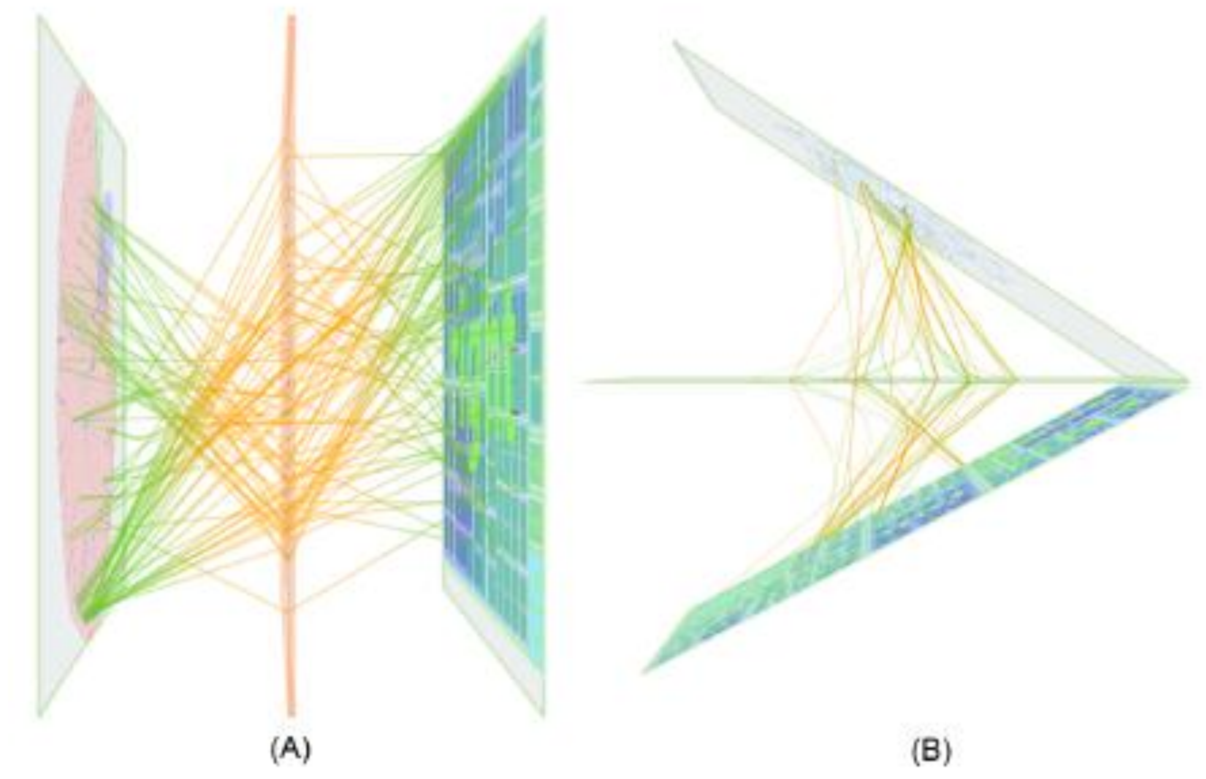


Figure 21: Collins and Carpendale 2007 figure 8

Having decided against explicitly showing relations, we were left with the option of either combining our displays into one, or juxtaposing them side by side, possibly with some sort of linked functionality. To consider our research question properly answered, we therefore needed to determine if a combined display would be better than multiple side by side displays, as well as, would a combined matrix display prove superior to a combined Node-link display or vice versa, and likewise would juxtaposed displays of matrices prove better than of node-links. Our comparison tool thus had to consider four possible display options: a combined Node-Link display; multiple Node-Link displays; a combined Matrix; multiple matrices.

Further examination of Gleicher's work provided us with the tentative hypothesis that **a combined (superposed) view would be superior to juxtaposed views of the individual graphs.**

5.3 Design of the software data structure

In order to project an image of a graph onscreen, it is first necessary to store the data for manipulation.

Goodrich and Tamassia [59] give three common methods of storing graph data in a computer. Although their text is designed for Java programmers, similar storage methods apply across most programming languages. Each of these techniques has advantages and disadvantages in terms of the efficiency of various common manipulation and search functions and of required memory use.

i. The *edge list*, in which edges are stored in a container (list), vertices in a second container, and only the edges have reference to their vertices. Thus it is easy to implement edge-based algorithms, but rather less easy for any function trying to deal with nodes, since a full search of the list of edges is required even to determine which nodes are adjacent to any given node.

ii. The *adjacency list,* which increases the functionality of the edge list by adding a reference from the vertices to their edges. This is done by means of vertices referencing a separate data structure called an *incidence container* which contains references to the edges incident on each vertex. In a digraph there may be separate incidence containers for edges depending upon whether they are into a vertex, out from a vertex or undirected. This is an efficient compromise between storage space and speed of access to the data.

iii. A stored representation using 2-d arrays of the graph’s adjacency matrix. This technique is very fast for finding information about the graph; its major difficulty lies in the adding or removing vertices, which requires the transfer of existing data plus changes into a new array, and thus is highly inefficient for graphs where vertices are frequently altered.

Other authors have utilised different means of storing graph data. For example, Marshall et al [60] offer an unusual idea for large graphs. In order to easily allow manipulation of the graph and its subgraphs (dealt with as node clusters), they consider that it may be necessary to store multiple levels of complex information about said subgraphs, and thus that a given node may belong to multiple subgraphs. In effect they treat the graph object as a meta-node, and class Graph becomes a subclass of class Node. The Node object stores the structural information related to it, and can be queried about its edges in the context of a particular graph. They suggest that this, along with storage of traversals (walks) as collections, makes manipulation by a user much faster.

After some consideration about speed and efficiency of data access, it was decided to utilise a structure based on the adjacency list mentioned above. This allowed for a quick and efficient comparison filter (see section 2 below) to be built which took advantage of the ease of access between different aspects of each graph. The tool was built in Java to take advantage of the many extant display classes which it was hoped could be relatively easily adapted without the necessity for a new rendering class from to be designed from scratch..

5.4 Designing the visualisations

For the purposes of this research, there were several questions that needed to be answered in order to select appropriate visualisations. Some of these questions have been addressed previously, but to recap, these were:

how best to show a node-link display

how best to show a matrix display

how best to show multiples of these

As previously mentioned, Dwyer et al [14] compared user-generated and automatic graph layouts (for node-link displays). In general, they found that user-generated layouts were superior, but importantly their comparison backed up the previous work of Purchase, Huang and their respective varied collaborators in regard to computer-generated layouts, in that artificial forcings, such as orthogonality and circular layouts, made even moderately complex graphs much more difficult to understand and hence that force-directed layouts were the most effective means of using computers to generate node-link displays. In addition, Holten and van Wijk [33] made an important finding in regard to directed graphs, in regard to the most understandable shape of the edge representations on-screen. They found that users found it much easier to follow directed graphs when the edge was represented by an isosceles triangle that by either a straight or curved line with an arrow-head.

Therefore the most effective means of displaying directed graphs has been demonstrated to be a force-directed layout with the edges shown as long triangles or wedges rather than the traditional arrow. We therefore required to select a force-directed algorithm, and use it to produce a triangle-based visualisation.

For reasons of its simplicity and speed, the Barnes-Hut [61] layout algorithm was chosen. This well-known algorithm produces good layouts and responds quickly to manual alterations, so making interactivity easier.

An example of this type of node-link display for a single graph is given in Figure 22 below:

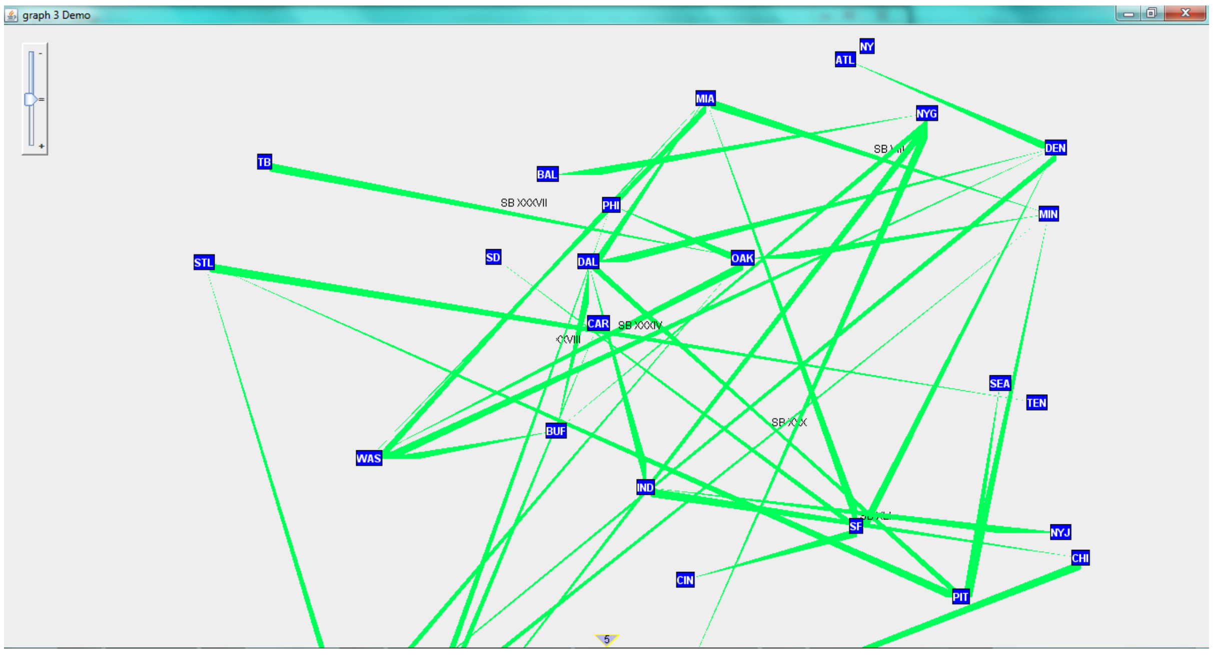


Figure 22: A Node-Link layout of a directed graph. This is generated from the actual test data set used.

However, the visualisation had other issues: we intended to combine multiple graphs into a single image. How best could we display all the edges of all the graphs?

After some consideration, we decided that, since we intended to use isosceles triangles to represent our edges, the best option would to offset those edges so that the edges from different graphs did not entirely overlay each other. This obviously required a clear colour differentiation between graphs, and since we had the simplicity of a numerical value easily assigned to each of the four graphs to be loaded, we decided to experiment with basing a graph's colour on its loading number.

5.5 Displaying matrices

For the matrix-based visualisations, the most difficult issue involved how best to show multiple edges. Because of space constraints there is always some difficulty in arranging the edges in such a way as to clearly show both multiple edges in a single graph, and common edges in multiple graphs.

We considered, and rejected, the concept of dividing the matrix cell between the number of graphs, in a similar manner to the screen-efficient displays pioneered by Keim [62] [63]. The reason behind this rejection was that it required a user to recognise which, if any, graph might be missing from a cell with multiple edges, and therefore to recognise in which part of a cell that graph's edges might be found. It therefore added to the cognitive load of the user, which, in overviewing a number of graphs, might be difficult, especially as the pattern of division would be different depending upon the number of graphs being compared. The additional complication of how best to divide a cell in the first place made this option even less palatable.

Thus we chose to simply offset each edge along the diagonal of the cell, shrinking their representations by whatever number of edges was in the cell.

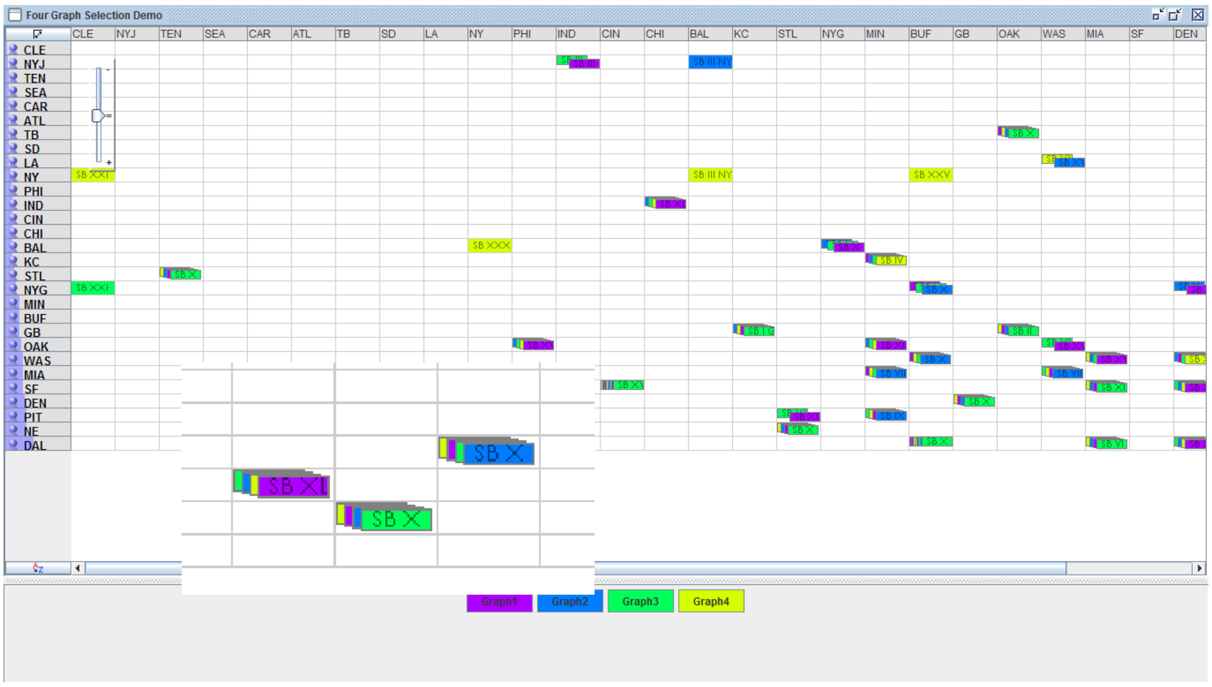


Figure 23: offset edges in a small graph matrix. Note the enlarged area which clearly shows that the edges vary in the order they are placed within each cell. This is a result of allowing Java to place the edges as a group rather than placing all the edges of one graph before moving to the next graph.

As can be seen from Figure 23 this was quite effective and gave clear indication of how many edges were in each cell and to which graphs each edge belonged. By adding a mouseover we would hopefully be able to show additional pertinent information about all the edges in any given cell. The offset would have the additional useful property of being quite similar to the offset used in the node-link displays, so the tool's use remained consistent between types of display.

This type of display also has the advantage of being highly scalable, so it can be used for both small and large graphs. An example of this display for a large graph is given in Figure 24 below.

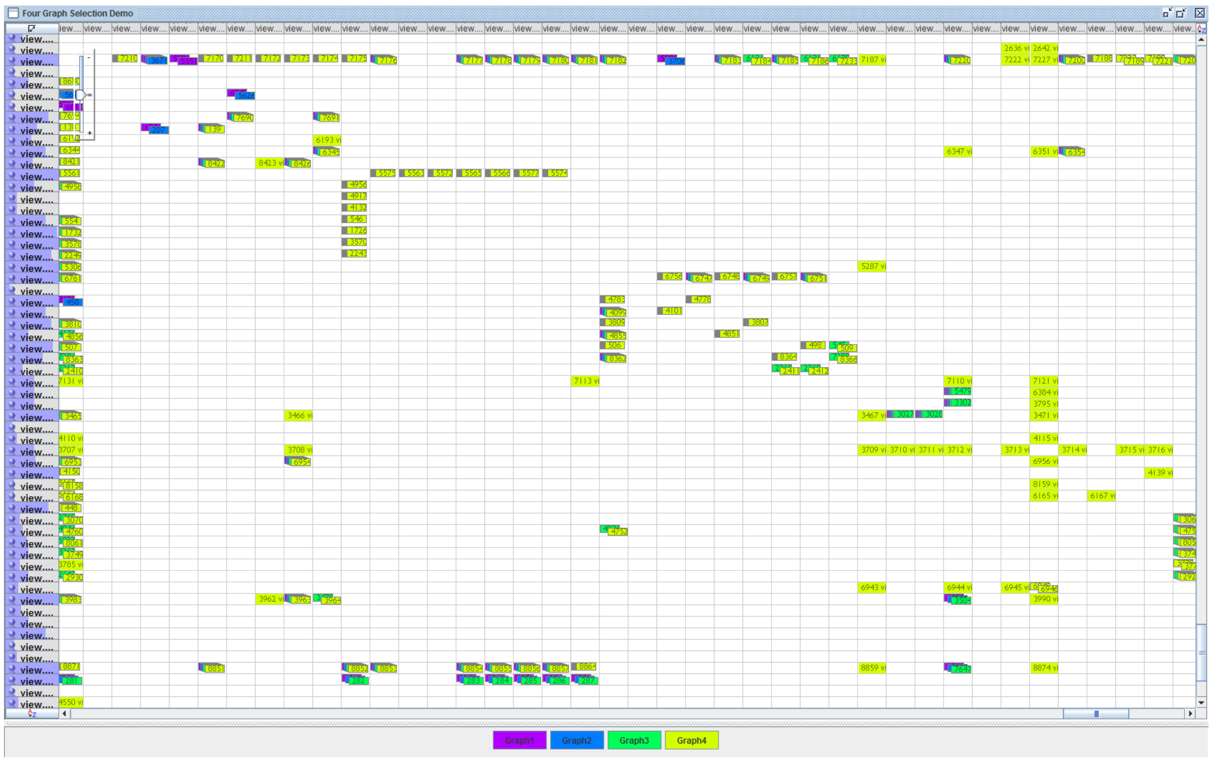


Figure 24: Combined matrix display of four large graphs

5.6 Effects of a common vertex set

The result of having a combined vertex set is that, in the matrix-based displays, the number and ordering of the vertices is consistent. All the individual matrices would show the entire combined vertex set, even where the graph being shown did not contain them. This was intended to make analysis considerably easier for a user; if they alphabetised each axis, they could be sure that alphabetisation of each different matrix would still produce the same ordering of the vertices, and thus that common edges would appear in precisely the same location within each display. This would not be the case if vertices were ordered by, for example, degree, nor would it necessarily be true if each matrix only used the vertices present in the graph it displayed.

A related but different phenomenon occurs with the force-directed node-link displays. For this type of display, where only one graph is rendered but the entire combined vertex set is included, the non-present vertices in the combined set are pushed to the outer edges; there is no attractive force (edge) to keep them close to the other vertices. It is thus possible to see at a glance which vertices are not connected anywhere in the graph. This brought up an important point re the separate node-link displays, viz., is it better to allow each graph to display according to the algorithm without reference to the other graphs, or should the entire vertex set be placed in the same positions on the displays?

Taking the position that users will want to have control over where they put each vertex, and moreover will likely want to treat each graph separately, at least in part, we chose not to enforce the latter. Although the initial layout would be similar, positional changes made by a user in one graph would not be reflected in the linked graphs; this would also improve computer response times. The select/highlight facility, and the option to visually filter out common edges was deemed sufficient, while the advantage of easily seeing non-present vertices in each graph was otherwise not available, indeed given the size of graphs we intended to use in our final set of experiments, it would be almost impossible due to expected occlusion.

5.7 Filtering Unwanted Information

The combination of multiple graphs into a single display also required that we add a **filter**. Filtering is the important technique of suppressing (uninteresting) data in order to concentrate on other areas. This simplifies the display, reducing cognitive load on the user, and enables easier understanding of the differences between the subset of data. In our tool, visualising multiple graphs, a filter might suppress the display of all but one graph, or perhaps show only a common subgraph. Given the possibly large amount of additional data on the normal display, this would enable the user to concentrate only on the data subset they actually required without any distraction.

Thus we wanted to produce a series of simple visualisations:

- a number of separate but linked node-link displays, one for each graph
- a single combined node-link display showing all the graphs at once
- a number of matrix-based displays, one for each graph
- a single combined matrix display showing all the graphs at once

These basic displays would each then be given additional functionality to assist users, as follows:

- for the separate displays, a linked selection and filter; highlighting or selecting in one display highlighted the same edges/nodes in the others.
- For the combined displays, likewise provided with a filter, allowing common edges to be toggled on or off, and also with the option to toggle the visibility of each individual graph's edges independently.

That it was only edges which could be toggled on and off is important. The issue of vertex mapping mentioned previously remained a thorny one; accordingly we decided, because we would be using graphs of known relation to each other, to use a simple label-match on the vertex sets to produce a combined vertex set of which the vertex sets of each graph were subsets. This was possible because we were aware that there was considerable overlap between the sets in the data we were using. In general practice this is not entirely unreasonable; most graphs likely to be compared to each other will have some sort of common basis (such as chemical structures) around which a combined vertex set can be built.

A final important point in the design of the filter involved the use of computing resources; there is no point in a filter that is badly affected by the size of the graph it is filtering..

Technically, the filter searches for common edges between vertices. This is therefore an algorithmic search for common sub-graphs. Such searches can prove very time-consuming and resource intensive, as previously mentioned. However we intended to use a specialised search (examining one edge at a time) and thus avoid this issue (and of course the NP-complete problem), as follows:

The filter would take one graph, and go through it edge by edge to find, based upon the origin and target of that edge, which if any selected additional graph also contains an edge with said origin and target. As such, it is a comparison where we find only subgraphs of degree one. Because it is based around only one primary graph, it only runs a single series of searches and thus runs in O(*kn*) time, where *k* is the number of graphs being searched and *n* the number of edges in the primary graph.

5.8 Building the node-link displays

In actually building our node-link displays, we first had to consider the effect of colour. Would any graph be dominant simply due to its colour, and would there be issues in colours for comparison purposes, for example, green-blue confusion?

As previously described, the obvious way of producing graphs of different colours was to find some sort of multiplier to the graph's load number and use that on the Java colour spectrum. This would make it easy to use exactly the same set of colours every time no matter how many graphs were loaded. Thus the first graph loaded would always be colour *a*, the second colour *b* and so on.

After experimenting with different numbers we settled on a multiplier of fifty which we felt gave us sufficient colour differentials to be clear. This produced four graphs of easily differentiated colours as shown below.

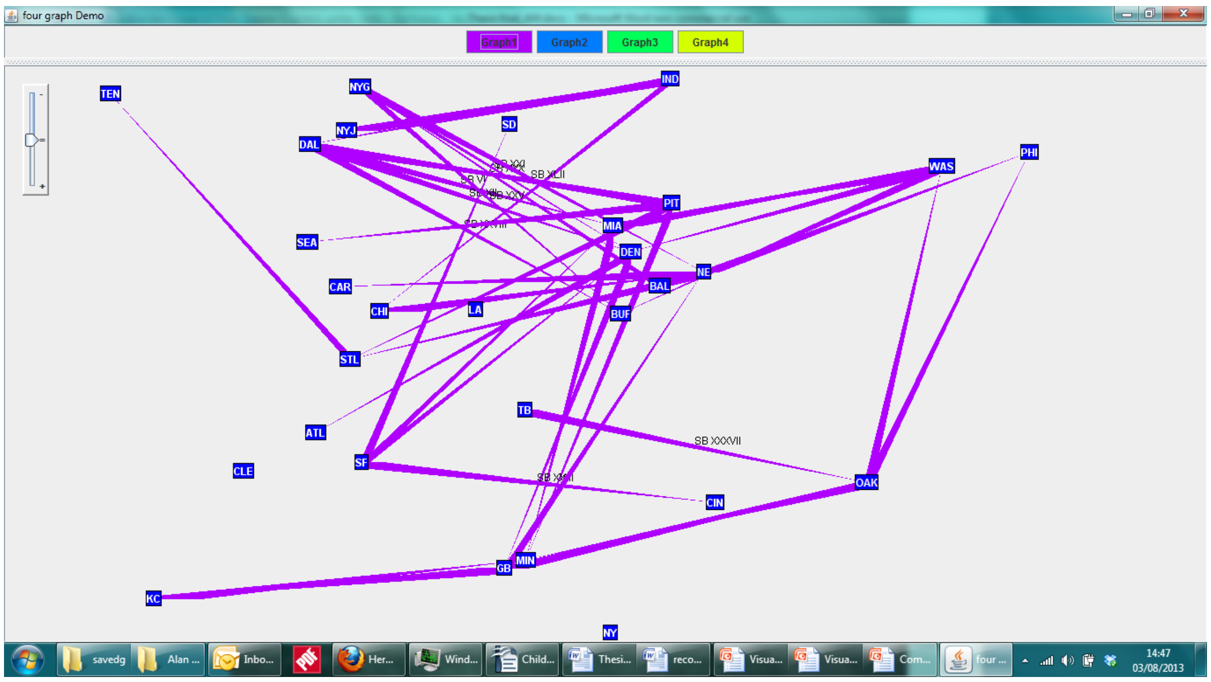


Figure 25: teams by current location

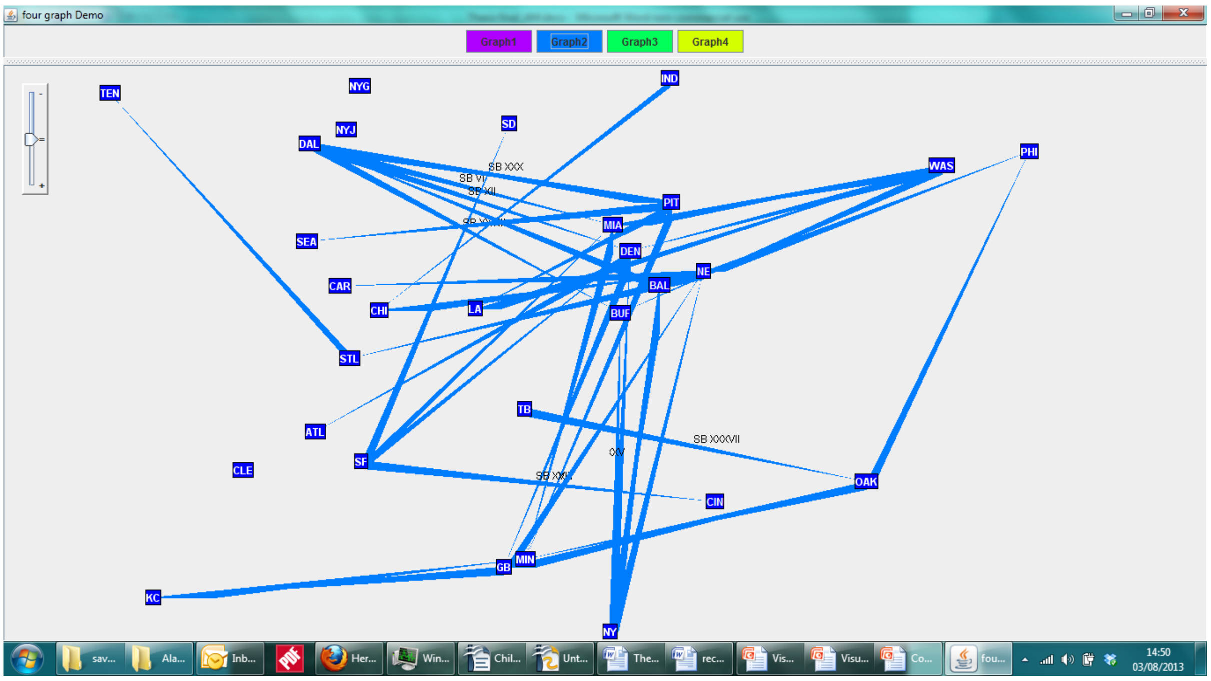


Figure 26: teams by location when game was played

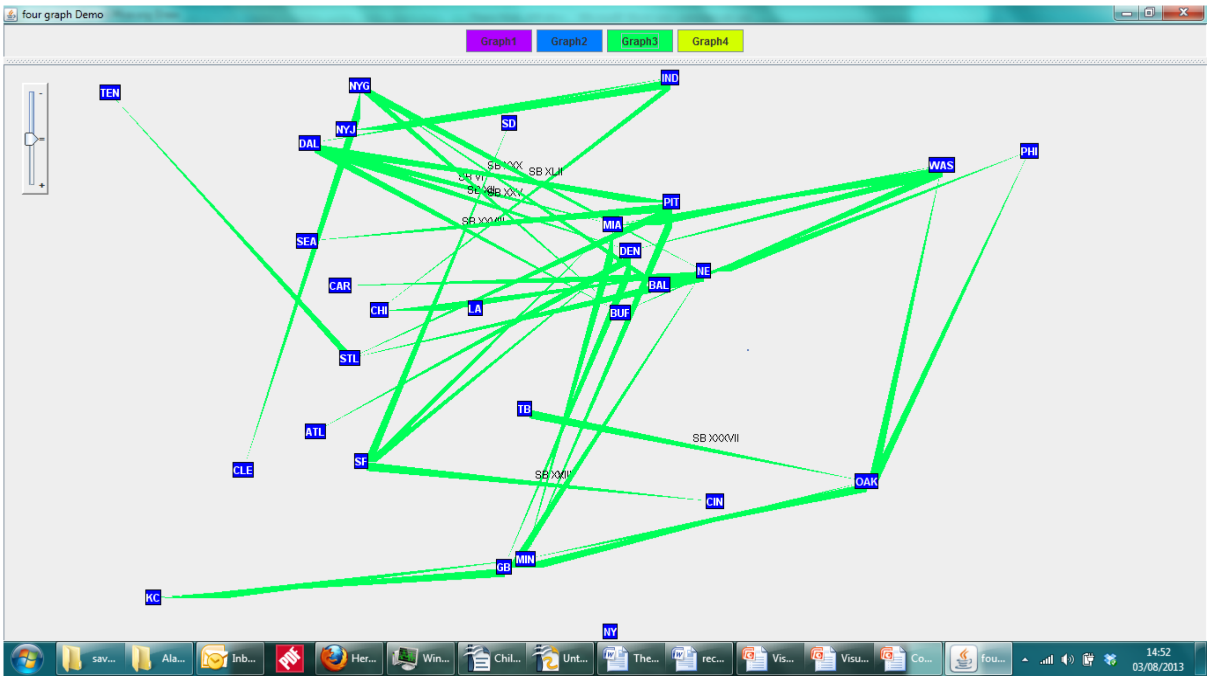


Figure 27: Altered date by current location

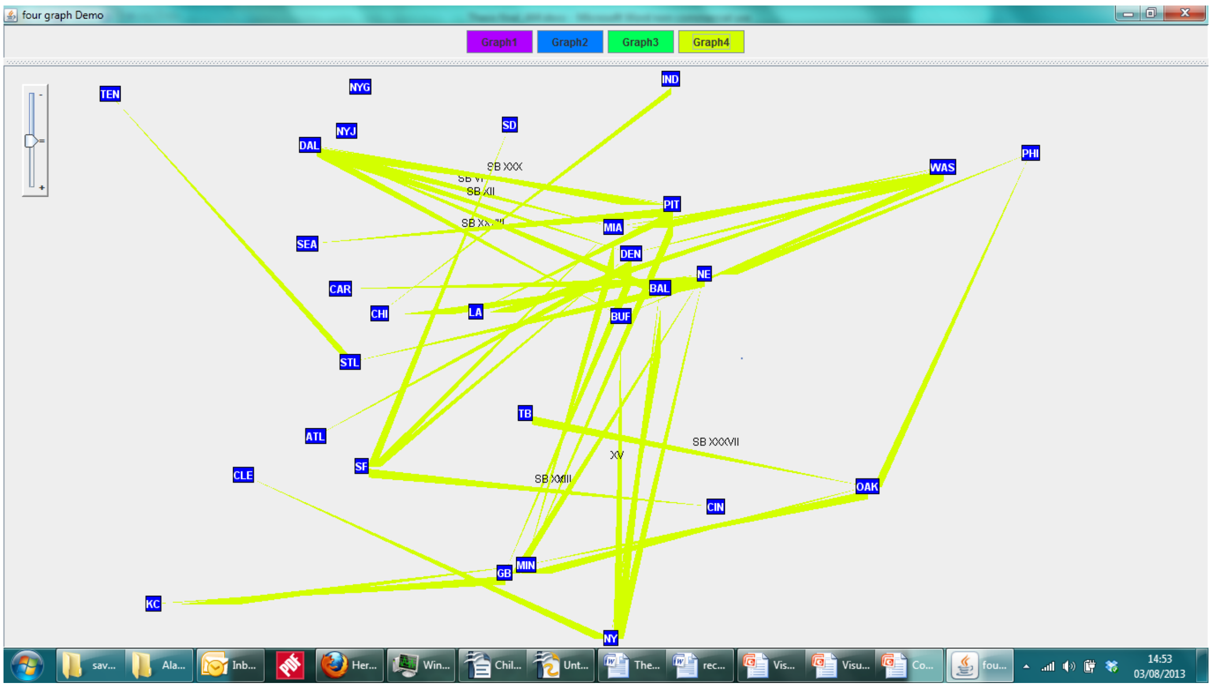


Figure 28: Altered data using team location at time of play

While node-link layouts were fairly easy to consider for each individual graph, the combined image was a potential problem. We had decided as described earlier that we should use the most effective form of node-link display, i.e. we would use straight wedges rather than lines and arrow-heads. These are shown in Figure 25, Figure 26, Figure 27 and Figure 28 above. For multiple graphs, it was obvious that we would have to offset our wedges by a small but noticeable degree and after a few tries this was found to be a simple matter. The real problem was less obvious. We found that the last graph drawn tended to 'drown out' the others as its edges overlaid everything else. This was a tricky issue; we did not want any one graph to dominate the combined display. However, we left the display unchanged for the moment to see whether it would prove a problem for our prototype testers.

Unfortunately this helpful solution gave us another problem; would there not come a point where a user would want to examine each graph individually? As with our matrices, the obvious answer was a filter of some sort. We needed a filter that would work well for both Node-link and matrix based displays, while remaining simple to use and efficient in its use of resources.

Would our proposed filter be useful for both types of display?

After several attempts (see Prototype testing in Chapter 6) we found that combining the data sets and allowing Java to draw the edges on its own was actually the best solution. Although the combined node-link display was fairly complex (see **Error! Reference source not found.** below) no single graph dominated the others.

5.9 The filter

Having decided that a filter would be necessary, the question became, how best to design a filter and what precisely should it do? In the event we produced two different filters.

The first filter was an option to make individual graphs invisible. Each graph had its own control, a simple toggle button. This allowed a user to eliminate any single graph from the display entirely in order to concentrate on the remaining graphs.

Our second filter was slightly more complex; it allowed a user to select one graph as a base, then choose which of the other graphs they wished to find common edges with. Any or all of the remaining graphs could be selected. They could then toggle the visibility of the remaining edges on or off.

By combining the use of both toggles, a user could therefore examine the differences between any or all given graphs loaded.

The filter option was made available to both combined and individual displays, for matrix and node-link, to avoid biasing the results.

The figures below demonstrate use of our filter in side by side (**juxtaposed**) matrices

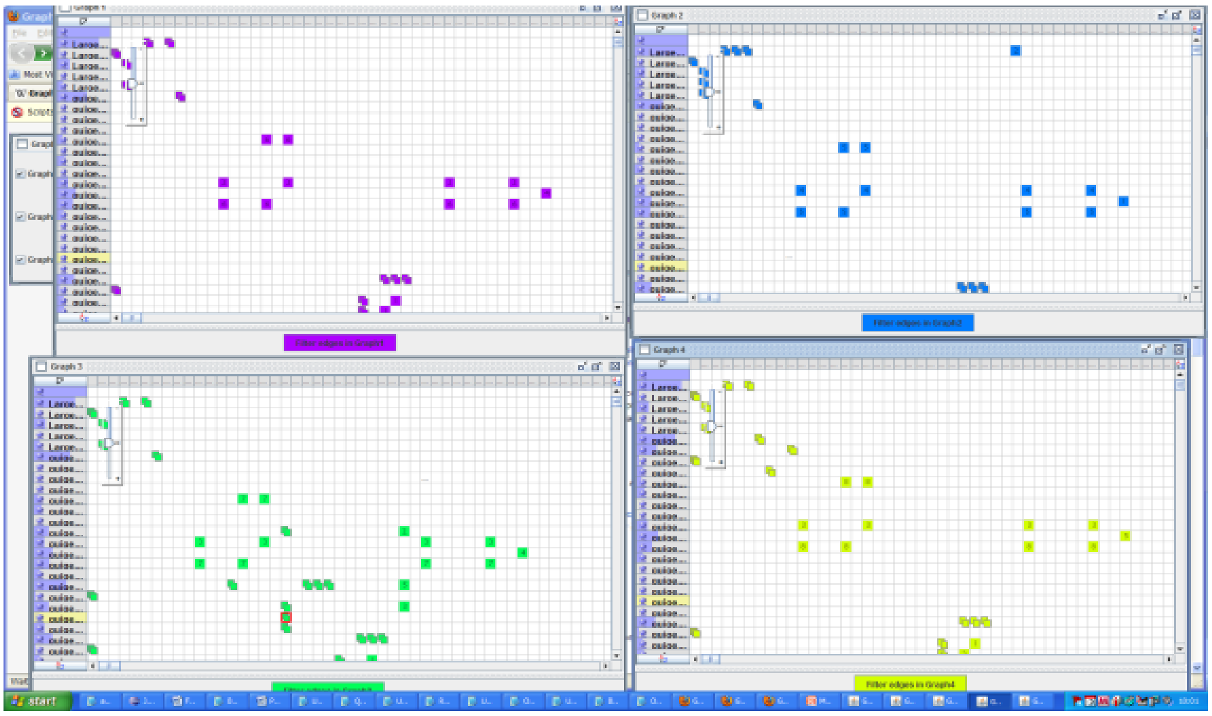


Figure 29: Four graph matrices juxtaposed. The graphs have a common vertex set

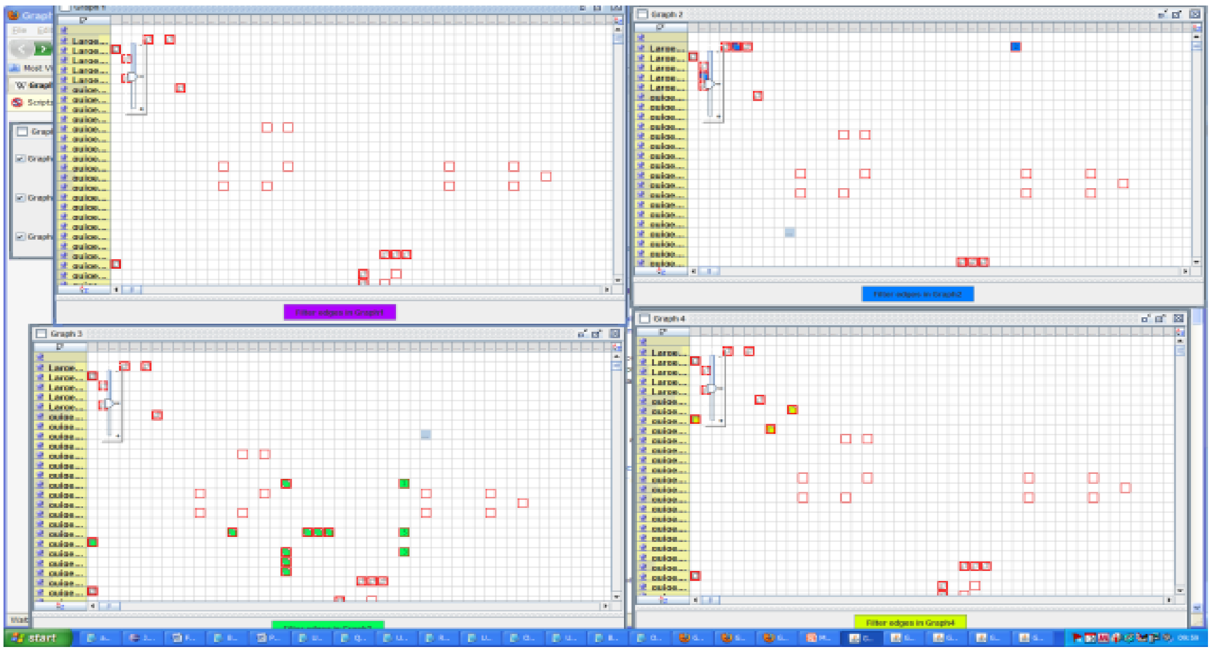


Figure 30: The same four graph matrices with filter applied. Note the red outlines to indicate where edges common to all four graphs have been whited out.

5.10 Building the matrix display

The matrix display was at first sight very simple. Each edge was a square or rectangle, and was placed in a *n* x *n* matrix. All that was required to do was render it on the screen. During development of the matrix display, we had not yet decided on our final data set, and so used the large graphs built from class relations that we had available from the various versions of a software tool built to examine trees. The first attempt at a side-by-side matrix is shown in Figure 31.

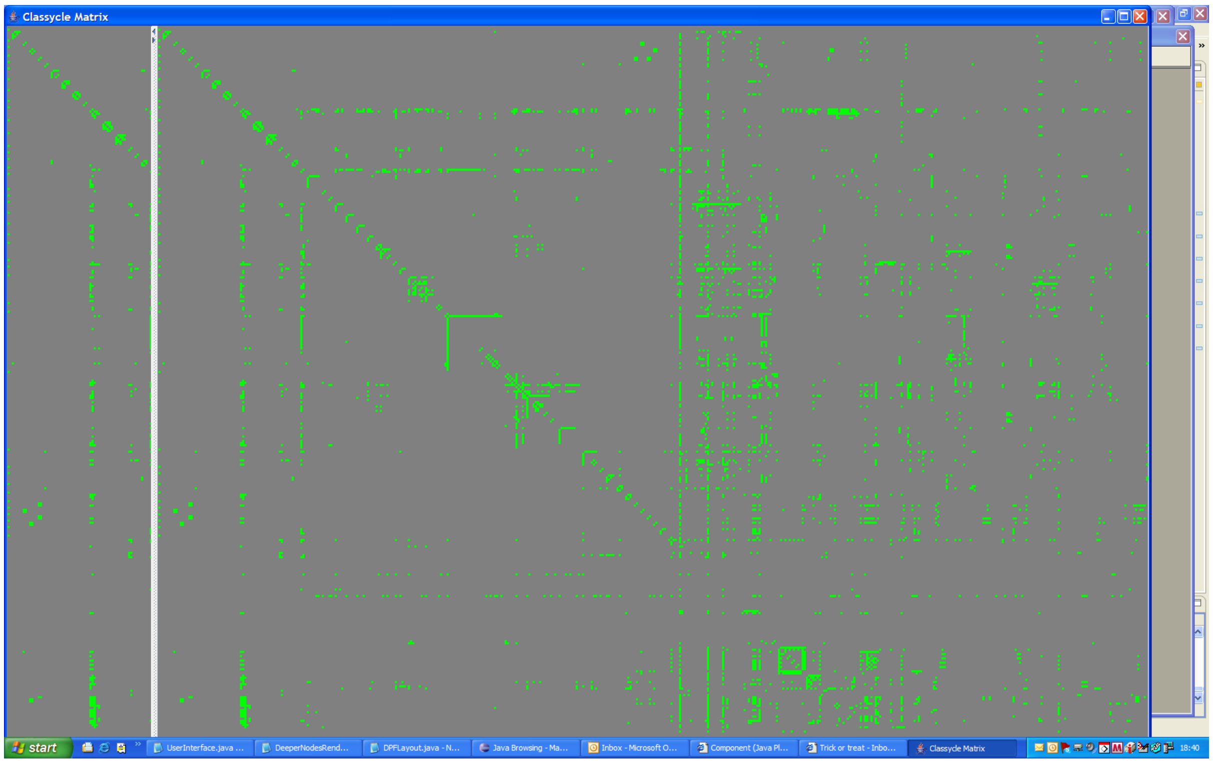


Figure 31: Two graph matrices (in green). The white line on the display is a mouse pick-up point allowing the user to slide the overlaid matrix across

This version was found to have several problems. First, ‘empty’ edges were rendered as grey squares. The program was therefore rendering, individually, the better part of a million squares. This took a couple of minutes to load even on the fastest machines available. A second version was much faster as it left the empty edges blank, only rendering actual edges, as shown in Figure 32.

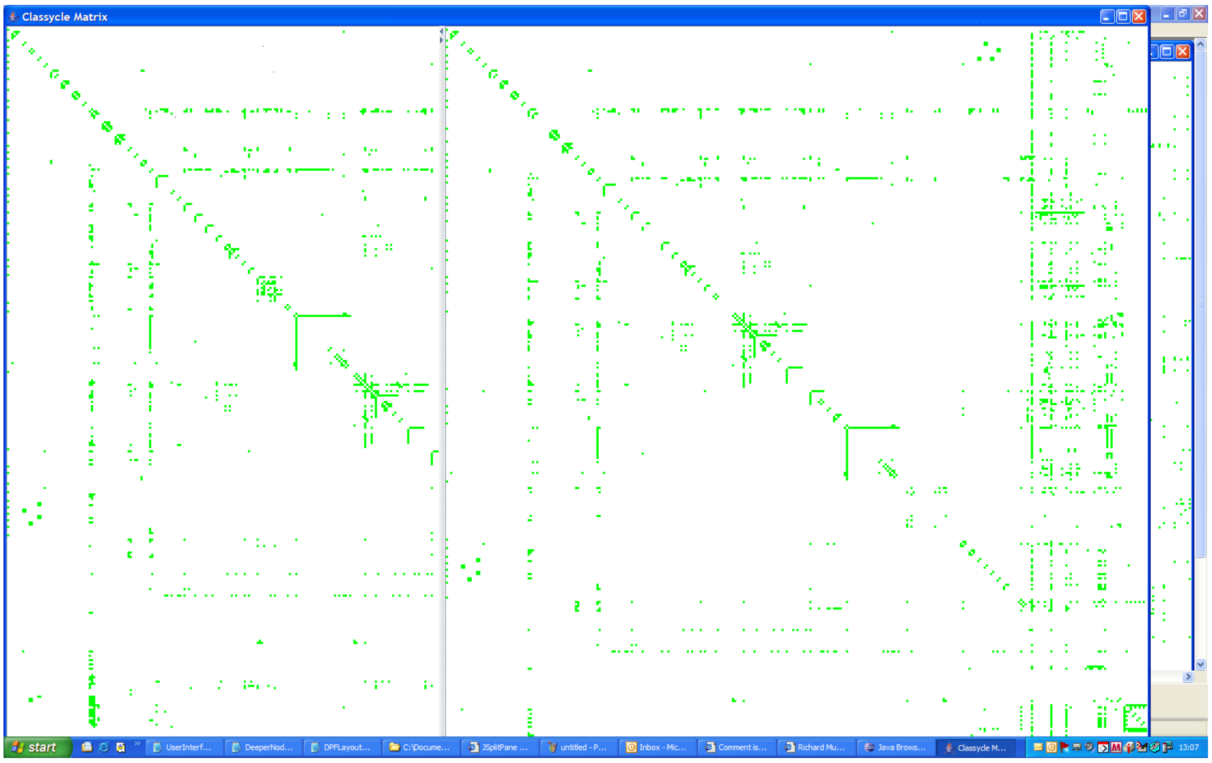


Figure 32: Updated version of previous figure. Only actual edges are rendered

This was considerably faster, but still individually rendered many thousands of objects. Moreover, at this point there was no display for the vertices. We had started to look into the issues of re-ordering, and considered that no user would want to wait for several seconds while the images were redrawn. A different approach was needed, especially since this prototype only offered two graphs and still took around ten seconds to load. While not quite so time-consuming for small graphs, producing even a moderately sized graph of 100 vertices in such fashion would still require the potentials rendering of ten thousand individual images, and re-rendering those images with each change to the display. This was clearly not scalable to any practical degree.

A solution was found in the Java standard classes: the Table. This was readily adaptable to our needs, and did not require multiple individual renderings as the data display was a single entity.

Having decided to use the SuperBowl data set, we were able to produce an easily understandable combined display as shown in Figure 33

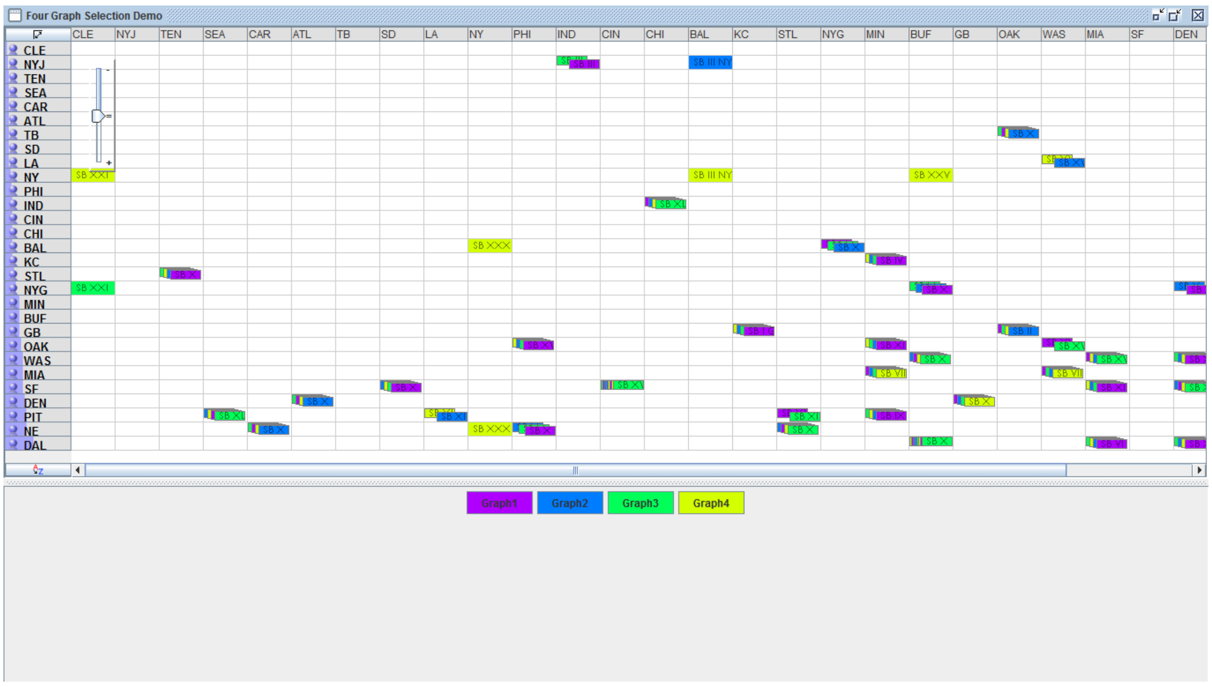


Figure 33: Four matrices on a single table. The buttons toggle individual graph matrices visible/invisible.

A great advantage of the Table class was that it can zoom in and out without difficulty, and that the vertices were simple to include as lists on the side and the top of the image. We were also able to add a mouseover display which showed precisely which edges were contained within any given cell of the Table.

The addition of our filters was a trivial matter; in fact the only difference between the node-link amd matrix programs was the nature of the display, which is as it should be for a test of this nature. The final combined matrix display is shown in Figure 34.

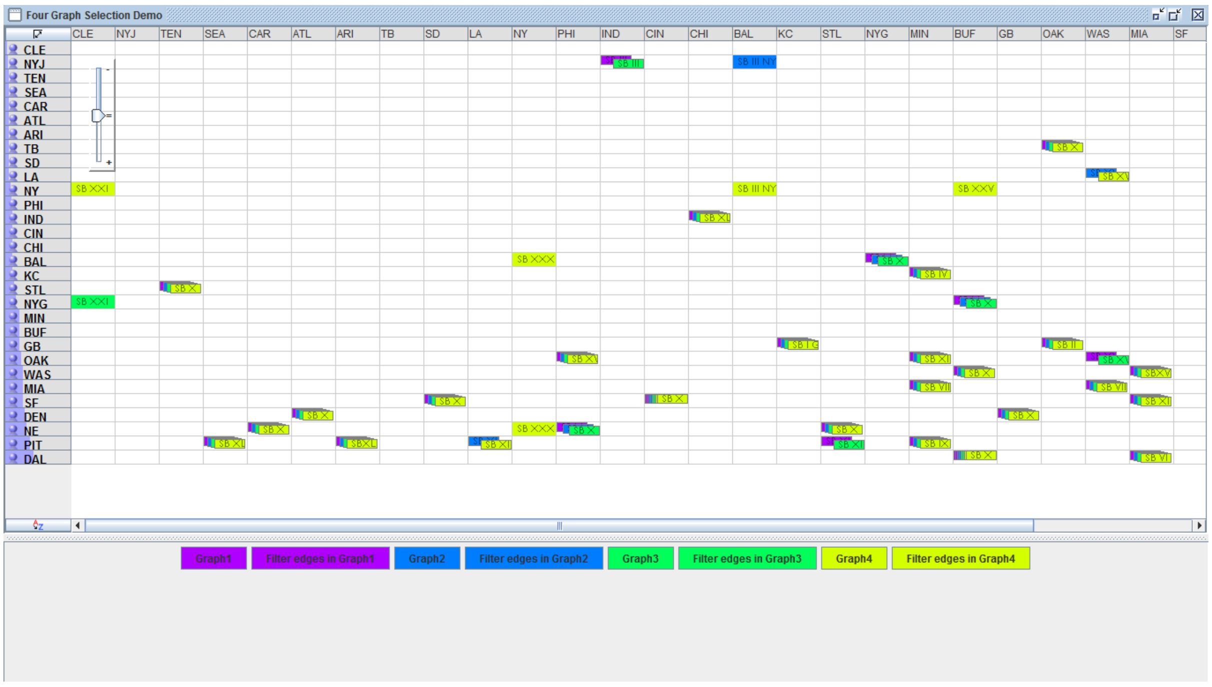


Figure 34: The four different matrices for the SuperBowl data set combined into one matrix. Note how the edges for each graph are now in the same position in each cell with the yellow graph on top, the green under it, then the blue, and the purple on the bottom.

This display was equally useful for the large data sets we had so much trouble with earlier. Using the Table class they rendered almost instantly, even when combining four graphs into a single display (see Figure 35 below).

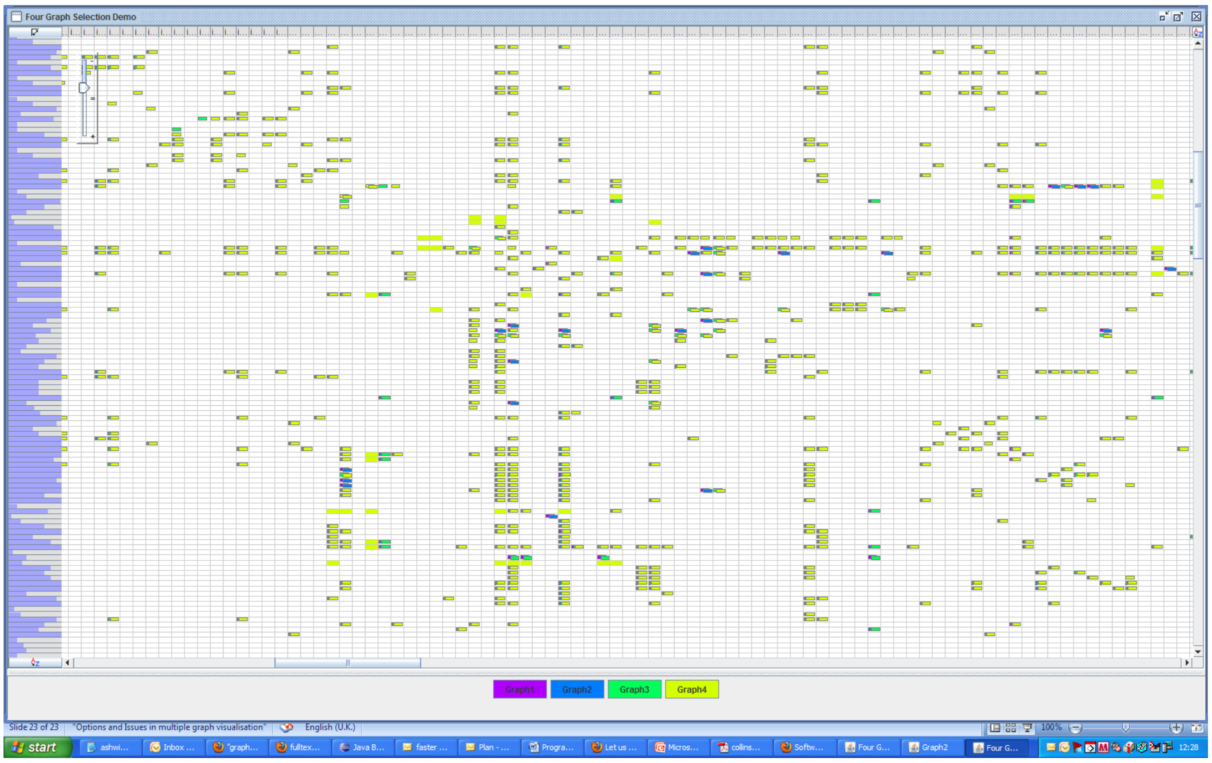

Four Graph Selection Demo
Graph1
Graph2
Graph3
Graph4
Slide 23 of 23
"Options and Issues in multiple graph visualisation"
English (U.K.)
start
12:28

Figure 35: Four large graphs in a single matrix display. There are over 12000 edges displayed across the whole of this matrix.

5.11 Summary

We had now produced a working prototype tool. We could view both juxtaposed and superposed graphs in matrix and node-link form, and had the facility to filter all displays as required to eliminate (temporarily) unrequired data from the visualisation.

The next step was to test it.

Chapter 6 Testing the prototype

Having designed and built a tool, the tests were now ready to proceed. First however we decided to run a simple prototype test using just two graphs in order to ensure there were no obvious problems we might have overlooked. This initial test is described below. Before that however an explanation of the questions asked and reasoning behind them is in order.

As we were not simply considering graph comparison in terms of pattern matching by algorithmic means, we identified a series of 'standard' graph comparison questions. These could then be couched in terms of any experimental data set in order to produce a series of testable comparison tasks.

These tasks were based upon general observation of the literature and related work in the fields of both graph comparison and individual graph analysis. An attempt was made to keep the tasks as direct and simple as possible as it is not immediately obvious how graph features such as trees or cycles could be easily compared. In particular, the works of Ghoniem et al [32] and Keller et al [38] previously mentioned were examined in detail in order to produce an appropriate task list.

The general case graph comparison questions we considered were:

- Given a specific vertex in one graph, find it in a second graph;
- Given a specific vertex in one graph, find its equivalent in a second graph and
    - compare its edges sets in and out
    - compare those vertices to which it is connected as an origin
    - compare those vertices to which it is connected as a target

- Given an identifiable edge in one graph, find it in a second graph;
- Given an identifiable edge in one graph find its equivalent in a second graph
    - Compare its origin
    - Compare its target
- Given two graphs, find the similarities between them in terms of
    - Common vertices
    - Common edges
    - Common sub-graphs
- Given a given sub-graph in one graph, find its equivalent(s) in a second graph

Obviously, in a combined view each vertex only appears once, so the first listed task is only appropriate where each graph is shown in a separate window. Since this task is by its nature included in the finding of common vertices and edges, it was decided not to test for it as an individual task.

Note that finding the same vertex and the equivalent vertex are not necessarily the same task. An equivalent vertex to vertex *a* could be one that has the same in/out edges rather than one that has the same label or identifying characteristic(s) as *a*. This is clearly dependent upon data and context to a large degree.

6.1 Preliminary prototype questions

The two-graph prototype was tested primarily for ease of use. This meant that we were more interested in the ability of the users to answer the questions than in the actual results and accordingly a small group of testers would suffice as little analysis would be done on the results. An additional set of questions was included involving simple tasks of single graph analysis. This was both to familiarise the users with the tool and to see whether the addition of more than one graph would have any detrimental effect.

We subsequently decided to include a similar set of questions in the main testing. Again, this was in order to make it easy for users to familiarise themselves with the tool, especially in switching between graphs.

These single-graph questions were data-specific versions of the following generic questions.

- Locate a specific vertex
- Locate a vertex with given degree
- Locate the vertex with the largest degree (in-, out-, total)
- Given a specific vertex locate those vertices to which it is connected as an origin and/or as a target

To make these specific to our test data set, we would translate these as, for example,

- In which games did Cincinnati Bengals (CIN) play?
- Which team has played in three SuperBowls?
- Which team has played in the most SuperBowls?
- Which teams did Minnesota Vikings lose to in their four SuperBowl games?

The precise wordings of all the test questions are given in Appendix I together with summaries of the results.

6.2 Prototype test set-up

We tested each different type of visualisation (Node-Link and matrix) separately. In each case, testers were asked to use both a combined view and linked separate views to answer a question set, and to help eliminate bias the order in which those views were tested was randomly varied between the testers. Thus each tester received two sets of questions and used one view to answer a given set. Testers used either two matrix-based views, or two node-link views. No tester used both node-link and matrix-based views. This was primarily a result of the same data sets and same questions being asked of each type of view; the possibility of familiarity with the data set and/or questions might have added a bias to the results. Testers were randomly assigned matrix or node-link views. Each test was run individually, under observation. Testers were not limited in time as we were primarily interested in accuracy of results to ensure that our prototype could be used successfully.

After completing the first set of questions each tester attempted to answer the second set of questions using the view which they had not yet tried. Given a familiarity with the data set after answering the first question set, we expected more correct answers from the second question set regardless of visualisation used and this was borne out in practice. After completing the tests, the testers were asked for comments and to express a preference for the type of view.

6.2.1 Prototype test results - matrix

The prototype was tested with a group of 14 students ranging in age from 19 to 22. Ten were male and four female. All had some experience in computer use, primarily office and internet software, but none were actually studying computing. These students were recruited on a first-come basis from volunteers amongst the student body at Edinburgh Napier University. The tests were not limited in time as we primarily wished to ascertain whether our prototype was usable, but the testers were asked for both verbal and written feedback on their experience.

This preliminary test confirmed that testers verbally preferred a combined display to linked side-by-side displays by a margin of two to one.

Testers found the mouseover window very helpful in explaining multi-edges, but otherwise did not describe it as necessary for task completion.

Whilst all but two of our testers used the facility to re-order axes on the matrix views, none suggested that the reordering be linked in the side-by-side views, and when queried on this seemed surprised by the suggestion. This was unexpected, but further questioning indicated that testers preferred the facility to make changes to one view at a time so that they could more clearly follow the results of the changes.

Although we were not limiting time, each question was timed by us, as was total time taken. Only one tester found the display confusing; he stated that it was difficult to determine winners and losers, and gave up after only partially completing the first set of questions. All other testers managed to complete both sets of questions in around twenty minutes.

6.2.2 Prototype test results – Node-Link

This test used a different group of students, likewise 14 strong, with 6 female. The age range was from 19 to 24,

All testers completed both sets of questions within twenty-five minutes, the shortest time being just fourteen.

As with the matrix-based visualisation, the majority of testers, eight to three with three abstentions, stated a preference for the combined display.

Testers reacted favourably to the triangular edges, but almost unanimously stated that having the graph in purple overlay the blue made it quite difficult to see whether there was or was not a blue edge under the purple one. This had been considered as a potential problem earlier, but it was nonetheless welcome to have it confirmed as such by our testers. We therefore decided not to place one graph on top at all times, but to allow random edge overlay. This had a pleasing aesthetic effect with no one colour being dominant, as shown in Figure 36 for four graphs in node link format.

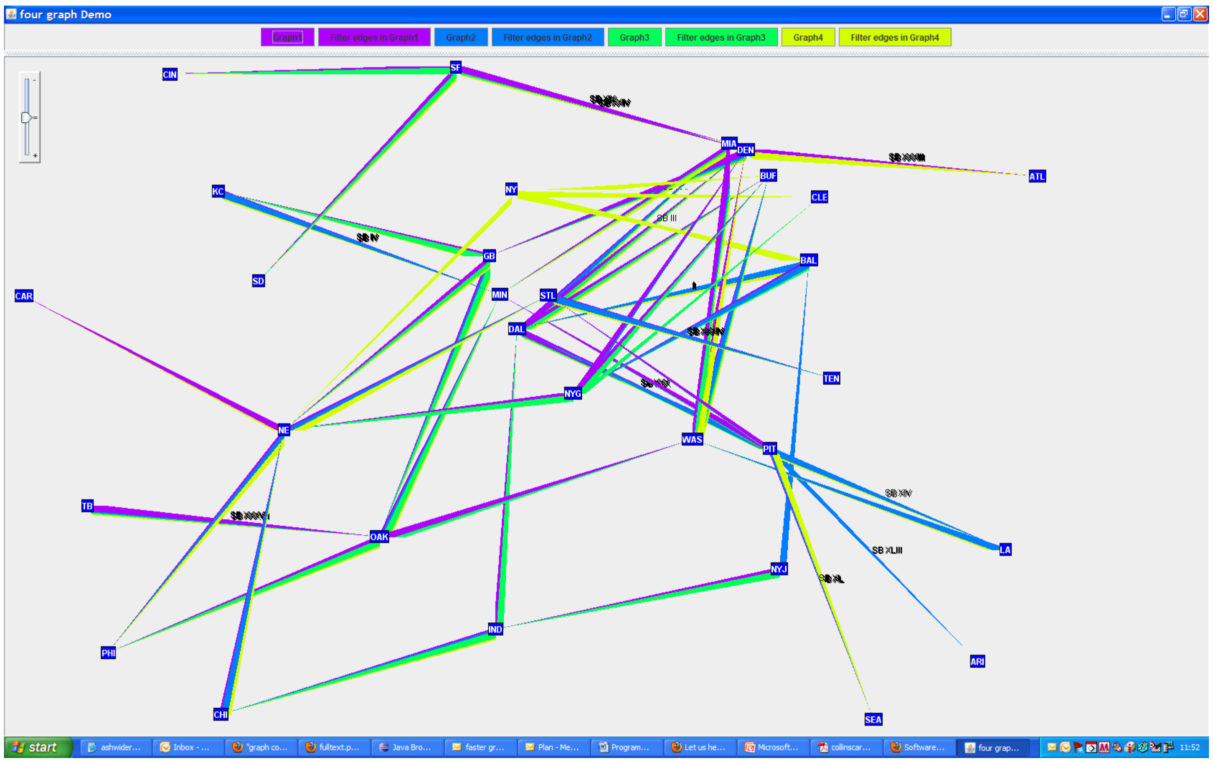


Figure 36: A combined node-link display of four graphs with random edge overlay

6.2.3 Prototype test results – combined vs separate views

A brief analysis of the results was completed which indicated that combined views performed much better in most areas than linked separate views,. We compared the results as percentages of correct answers for each type of question, so we had 13 people (one group of 7 and one of 6) answering each type of question for matrix-based views and 14 (two groups of 7) for node-link views. A full listing of the results from prototype testing is in Appendix A.

These results went fairly much as expected given our hypothesis (see Chapter 5) that combined views would be better than separate ones. However the number of testers was short of the number needed to demonstrate statistical significance in most cases.

Only with the matrix-based tests did we find any results of statistical significance. The statistically significant results were as follows:

We found that when using a combined matrix to identify vertices with a given degree, a task that required the use of only one matrix, the performance was significantly worse whether that degree was out-or in-degree. Despite having the facility to view only one matrix at a time, testers scored only 46% overall with the combined view, whilst the same group of testers scored 78% correct when using single matrices in separate windows. Using a paired two-tailed t-test, these produced a statistically significant P value of 0.0110. This result is shown in Figure 37 below.

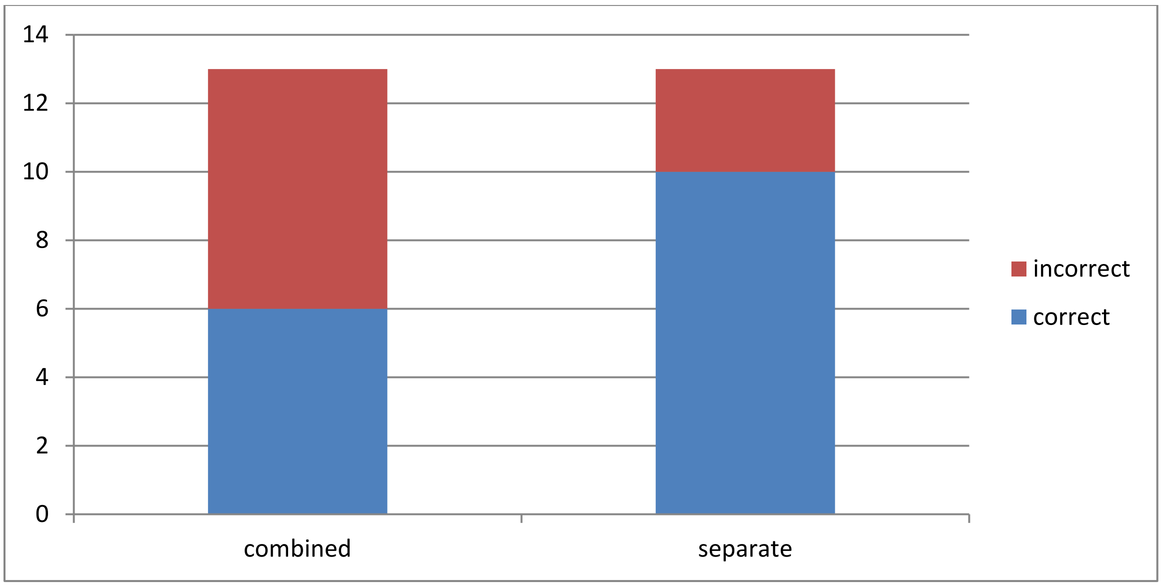


Figure 37: results for finding a vertex of given degree

We also found that when attempting to compare the degrees of named vertices between graphs, that the combined view gave a correct answer 78% of the time compared to separate windows giving the correct answer only 46% of the time. Again we used a two-tailed test. The P value for these results was also 0.0110, which again indicates that these results are unlikely to be random.

Neither of these was unexpected, so we looked at both results in an attempt to determine the underlying causes.

During this further analysis, it was found that the majority of mistakes in the first case were caused by testers who failed to differentiate between graphs in the combined view, and thus miscounted the degree of the vertex in question.

The second result indicated that when asked specifically to take account of the different graphs, testers found no difficulty in the combined view, but did have difficulty relating the separate views to each other. This is somewhat contradictory, but an explanation may lie in testers simply ‘counting blocks’ without regard to colour when asked to find a given degree unless specifically told to take account of colour differences.

Our final difference involved identifying the difference between graphs for edges connecting a given vertex. This was very clearly advantageous to the combined matrix view as shown in Figure 38 below.

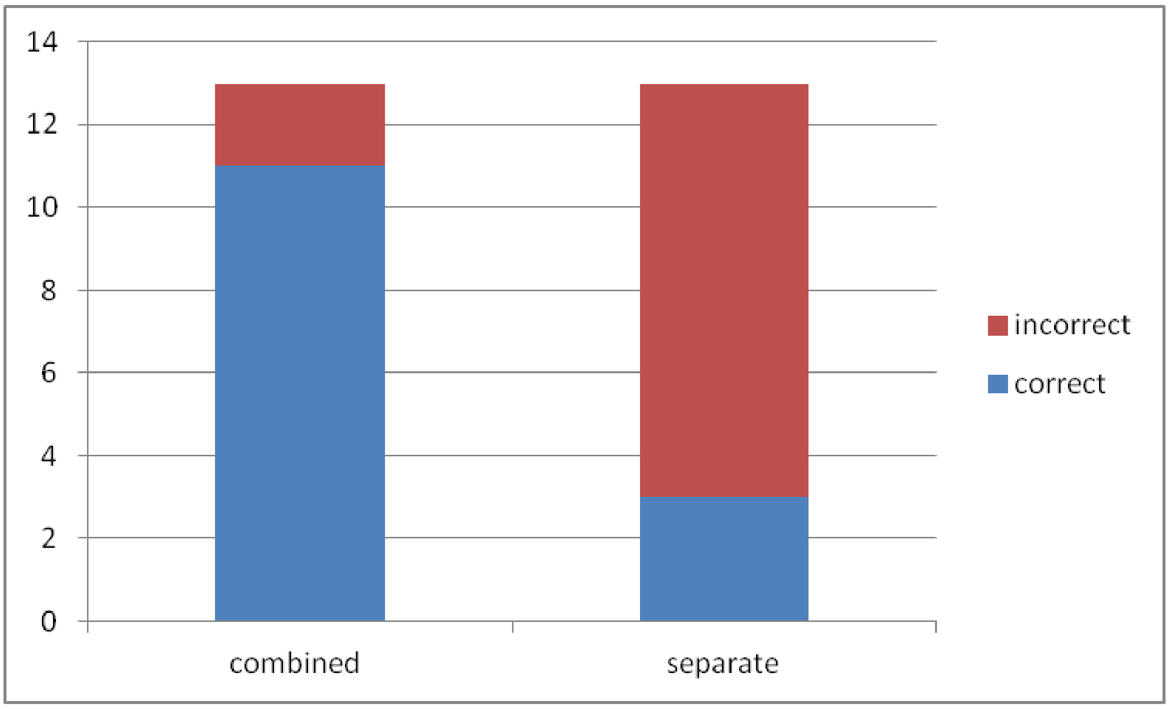


Figure 38: finding edge differences between graphs

The reasons for the above large difference are fairly clear. In a combined view it is obvious which edge has changed target (or origin) compared to its equivalents in other graphs which will appear in the same column or the same row of the matrix. This is especially true when a filter is used and potentially confusing information is stripped away. Finding that similarity when looking between juxtaposed views if not nearly so obvious. What is very interesting about this result however is that was not repeated for node-link diagrams to anything like the same extent. Our testers clearly found identification of equivalent edges easier in the node-link, especially once the filter had been applied as it pointed to or from one of the same vertices. This appears to be an example of the superiority of intuitive node-links over abstract matrices.

After this initial test we considered that the tool worked adequately well, but corrections needed to be made. We removed the forced layering of one graph over another in the Node-link combined view and asked the same group of testers to work again with the result. While familiarity with the data set and questions made statistical analysis of these test unviable due to possible bias, this was not the primary purpose of these tests. What we desired was to discover whether the new combined view was preferable.

All but two confirmed they preferred the new combined view and we therefore decided to retain this for the full test with four graphs.

Therefore, in regard to colour and confusion amongst the users, we found in preliminary testing that re-ordering the edges in the matrix-based display so that each graph was consistently in the same position relative to its peers was very useful (see Figure 34). In regard to the Node-Link display this ordering did not have the same difference; in fact the majority of users preferred the Node-Link display to have a more mixed appearance as they complained that otherwise the preponderance of one graph overlaying the offset edges of the others made it more difficult for them to actually consider those others effectively.

Chapter 7 Results and Analysis

After testing our prototype we made some small changes. We allowed the Java machine to lay edges randomly for node-link diagrams as described in the previous chapter, and we did the opposite for the matrix-based views. The following illustrations (Figure 39 and Figure 40) show what our testers saw onscreen in the combined views. The first two are the matrix-based displays. These demonstrate how the filter operated. Note the empty squares outlined in grey which show where common edges occurred:

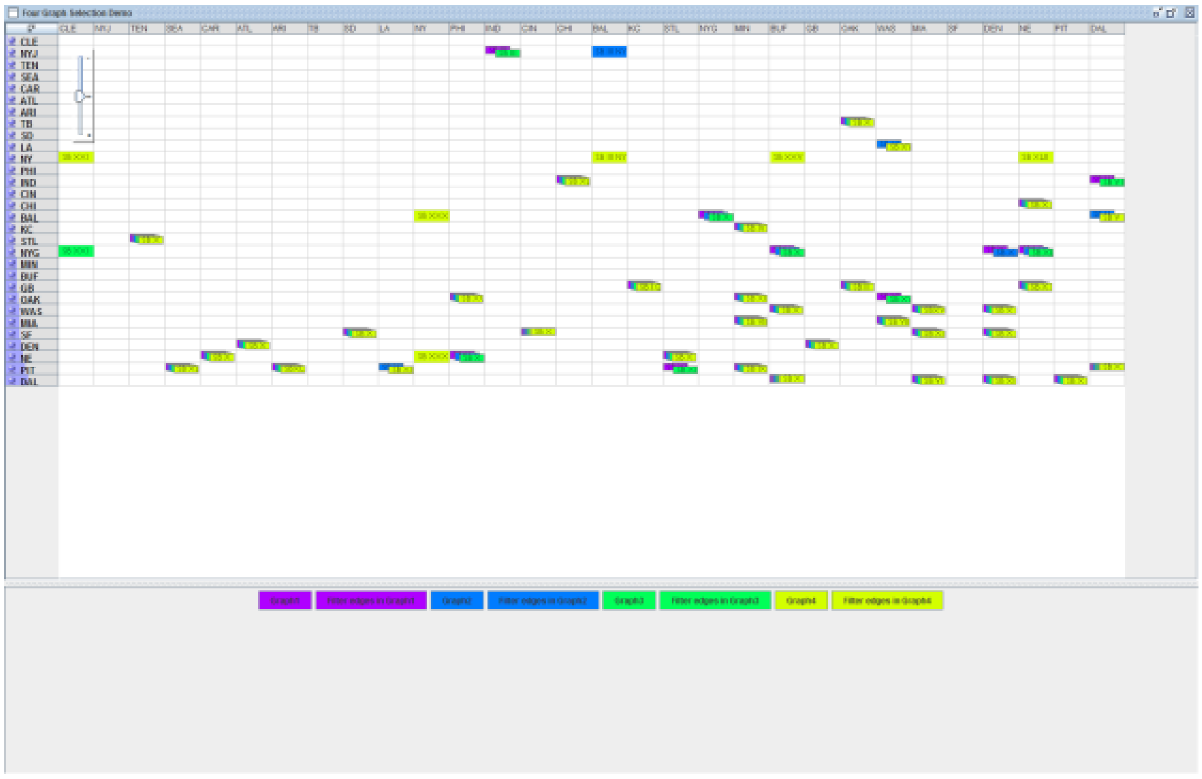


Figure 39: The four graphs in a combined matrix display with common vertex set

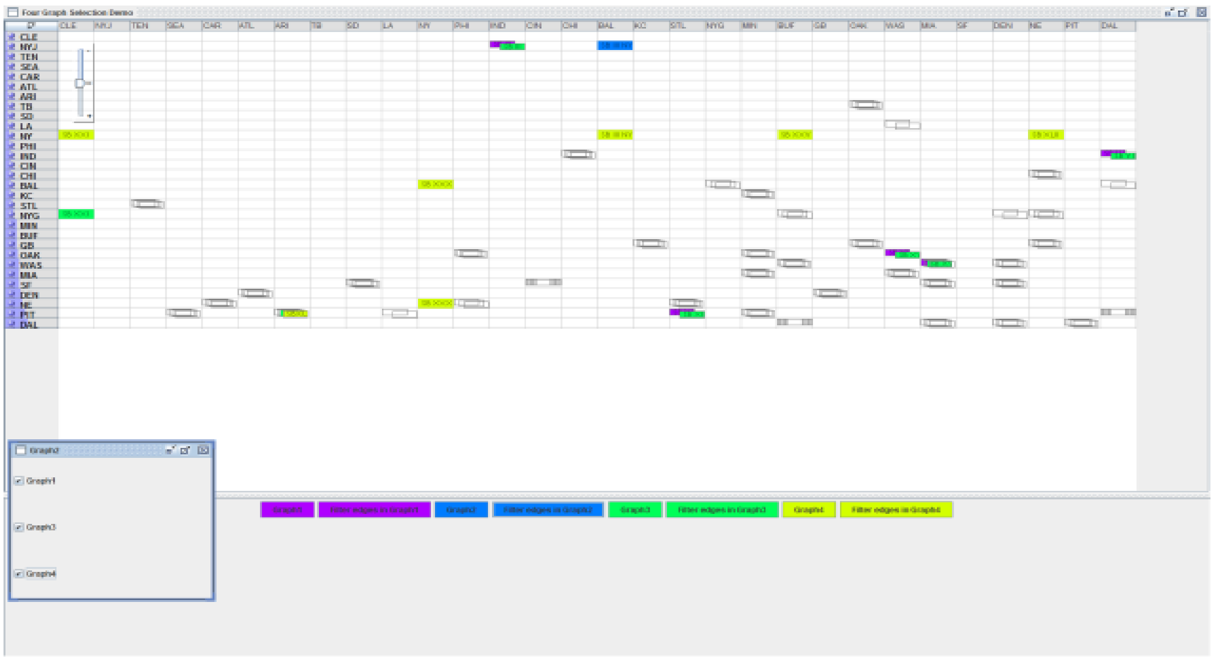


Figure 40: 4 graph combined matrix with filter active to white-out all edges common to all 4 graphs

For the node-link views the filter operated slightly differently, as shown below:

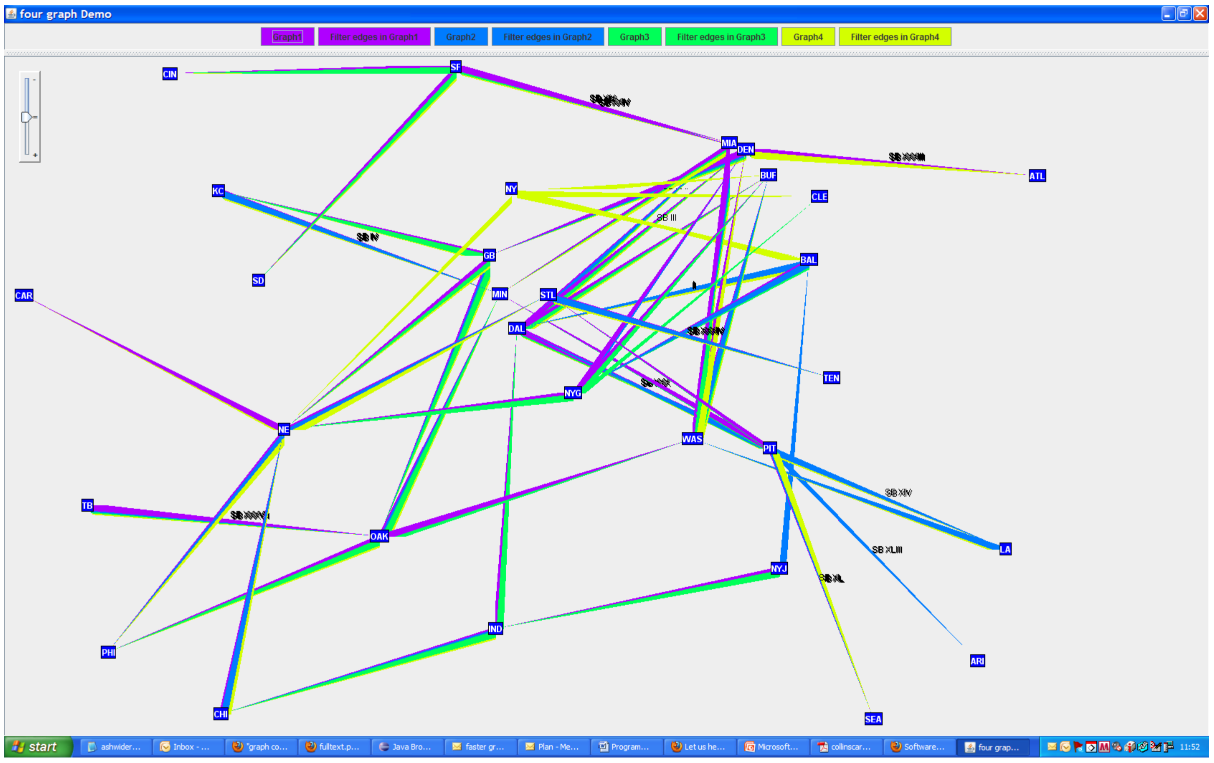


Figure 41: The 4 graphs in node-link combined view with filter dormant

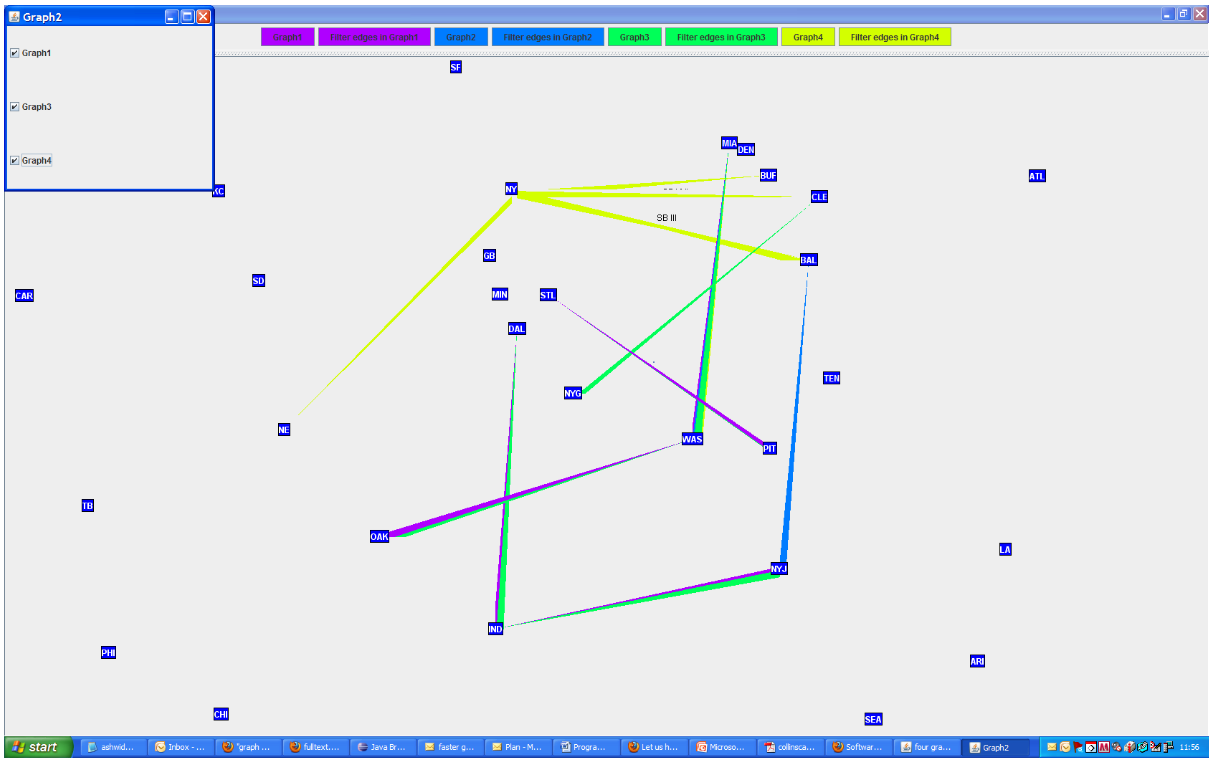


Figure 42: node-link combined view of four graphs with edges common to all four blanked out via filter

As can be seen, the filtered node-link view in Figure 42 does not retain after-images of filtered out edges when compared to its unfiltered equivalent in Figure 41. This is because any such edges continue to occlude the remaining unfiltered edges even when greyed out and therefore continue to add to cognitive load. The whole purpose of a filter is to cut that load and reduce clutter.

Having confirmed the tool worked in the manner desired, we proceeded to the task-based test.

7.1 Testing with four graphs – set up

Considering that there were a fairly limited number of questions available with the small data set, it was decided to test the matrices and Node-Link views separately, using the same format as the prototype tests. Thus each group of testers would use both combined and linked separate views to answer the same sets of questions. If the results were inconclusive, a final test could be run comparing the best Node-link view against the best matrix-based view. However with a sufficient number of testers, the same data set and the same set of questions, this would hopefully not be necessary.

We increased the number of testers to forty, divided into two groups of twenty each. Regrettably some volunteers had to pull out of their scheduled tests, and we found ourselves with just eighteen in each group.

As we had done with the prototype tests, we asked our testers for feedback as to their preferred view.

All tests were run on the same laptop computer, a Dell Inspiron 6000 model with RAM upgraded to 4 GB from its original 2. The tool was a self-contained .jar file which was opened by the researcher and a brief written explanation of it given to each tester individually. A verbal explanation was also given.

The test was task-based requiring each tester to reply to two sets of questions, one set using a combined view, and the other using linked separate views juxtaposed on the screen. As with the prototype testing, to eliminate bias the order in which the two view types were used varied. Nine testers used the combined view first; nine used the separate views first.

As with the prototype testing, each tester worked separately, under observation. Again, the testers were untimed. In hindsight this was probably an error, as the time taken to do a task correctly is an important attribute of how helpful a visualisation tool is to the analyst. However, this was not realised until halfway through the tests, and having failed to time the first series by our testers we considered there to be little point in doing so for the second series as there would not be sufficient data to provide a good basis for analysis.

The tests were split into two sessions, roughly a month apart, for each tester. This allowed us to run the same questions twice with each tester. To eliminate the effect of tester' memory, they were informed that the data had changed. Since they only had access to one view at a time (that is those who had first used the combined view would not be able to see a combined view on the second test) there was no way for them to check this statement. We also added additional questions between the ones we originally tested to further the disguise.

The order of the real test questions was always the same; thus we had nine answer papers from the combined view and nine with the juxtaposed views for each of the two question sets, from each session, a total of thirty-six paired tests in all.

A breakdown of the questions, additional inserted (dummy) questions, and results can be found in Appendix B, however the important test questions were as follows:

- Find a different origin on a specified edge; the difference might appear in more than one graph
- Find an edge unique to one of the four graphs
- Find a vertex that connected on only one graph of the four and its equivalent in the other graphs.
- List all differences between a given graph and all the others.

The above were rewritten to be data specific, and placed in various orders for each tester to help eliminate any bias.

7.2 Method of analysis

With a single group of testers which we had more or less randomly split into two (which group they were assigned to was based on their own availability rather than any deliberate policy of ours) we would no longer be comparing results within each group as individuals, but between the groups as two collective sets of results. That said, it would still be possible to pair every result for a combined view with a result by the same tester for a juxtaposed view of the same data. It would therefore also be possible to run analyses within each group. This would allow us to observe whether an individual’s performance was improved by a given visualisation type (juxtaposed vs superposed).

Moreover, with results of juxtaposed against superposed views available for both node-link and matrix views, observation could be made whether juxtaposition made similar differences to superposition regardless of type. Examination of statistical method indicated that a t-test would be the standard method of dealing with this sort of data; Given that our null hypothesis would be that there was no difference between using juxtaposed and superposed views, we chose to utilise a two-tailed test as it was possible that either one might prove superior.

Likewise, a two-tailed test would be the preferred analysis method for comparison of matrices against node-links as, again, we could consider a null hypothesis of ‘no difference’ and might have results indicative of superior performance by either method.

7.3 Test Results – matrix-based views

The test group consisted of volunteers, all students from the Faculty of Engineering, Computing and the Creative Industries at Edinburgh Napier University. They ranged in age from 18 to 24 and thirteen were male. Three were studying computing or a related subject, but all used computers in their chosen course, with eight engineering students admitting to using specific CAD packages.

In terms of feedback preferences, thirteen out of the eighteen testers preferred the combined view; only four preferred the separate windows of the juxtaposed view and one had no preference.

When asked specifically to compare vertex degrees between different graphs, testers performed well using the combined view, but did have difficulty relating the juxtaposed views to each other. This echoed the results from the prototype test.

These results are shown diagrammatically in Figure 43 and Figure 44 below

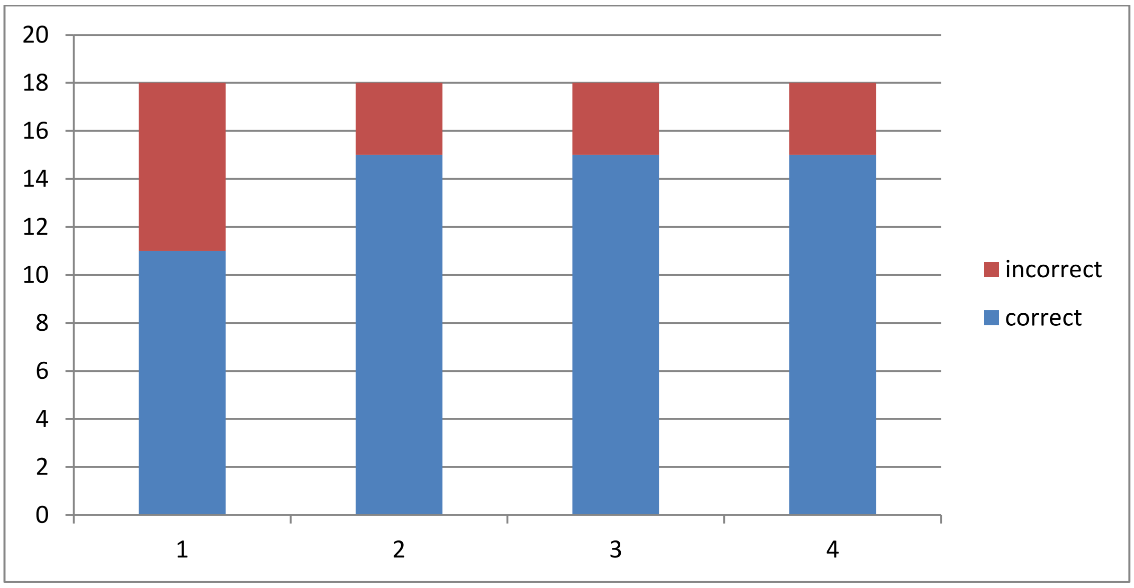


Figure 43: Results for superposed matrix view

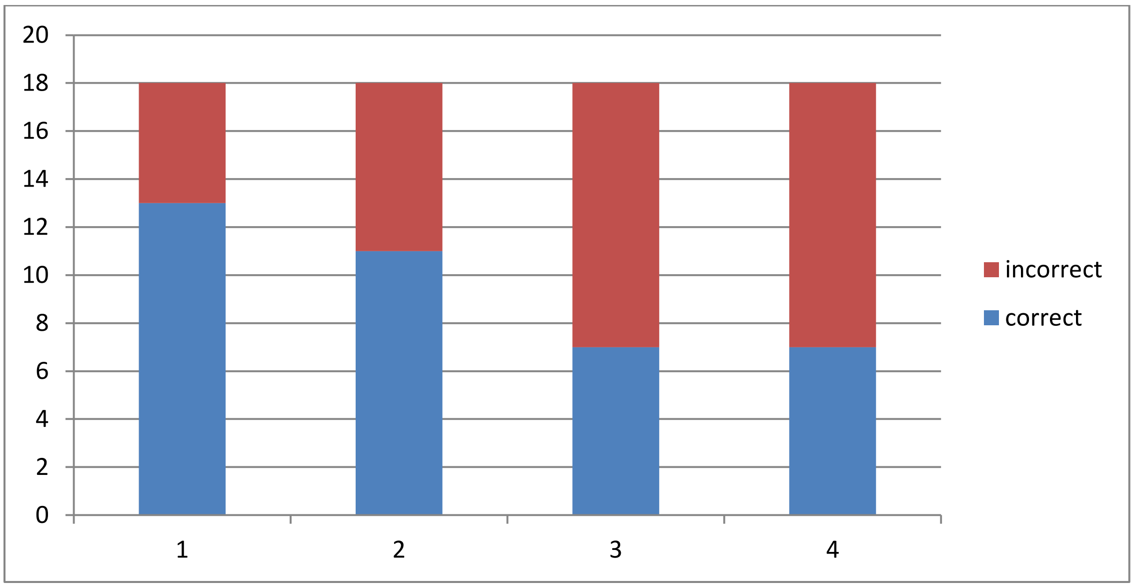


Figure 44: Results of juxtaposed matrix views

To test for whether these results were down to random chance (or not) we first put them into an unpaired two-tailed t-test.

1. When tasked to find differences on a specific edge (a labelled edge with a different origin) the results favoured the juxtaposed view by thirteen to eleven, but this would not be considered statistically significant by normal criteria.
2. When asked to find a specific edge found in only one graph of the four the results favoured the combined view by fifteen to eleven but given the number of testers was likewise not statistically significant.
3. When tasked to find a vertex that connected on only one of the four graphs, the results once again favoured the combined view in this case by fifteen to seven. Once again however the difference would not be considered statistically significant by normal criteria.
4. When tasked to simply list all the differences that could be found between one of the given graphs and each of the others, the results again favoured the combined view by fifteen to seven. Once more however the difference would not be considered statistically significant under the normal criteria for a two-tailed test.

This set of results was interesting; despite the combined view performing much better (around fifty percent more correct answers) the unpaired two-tailed test did not provide sufficient confidence interval for statistical significance.

After some consideration we realised that these tests were also suitable for analysis as a paired t-test, since each combined result had an equivalent (by the same tester) result using separate views. Thus each combined view result paired directly with a separate view result. This means that it is less likely that recurring differences on a given test question are down to random differences in tester performance.

Re-running the t-tests as paired rather than unpaired demonstrated sufficient statistical significance for us to state with a 95% confidence level that

When using matrix-based visual representation of graphs, the combined view is better for finding differences between graphs than side-by-side linked separate views.

These results also support work by Beck and Diehl in analysing software dependency using matrix-based visualisations [64].

7.4 Test results – Node-Link views

The same questions posed for the matrix-based views were asked with the Node-link views. As we had done with the matrix-based tool, we also asked for feedback in order to find out if there were any preferences in regard to combined vs. separate views.

As before we had a group of eighteen testers, in this case eleven were male. The testers ranged in age from 19 to 23 and all were students in the Faculty of Engineering, Computing and the Creative Industries at Edinburgh Napier University. Two of the group were studying computing or a directly related subject; all the others were familiar with computer use and used software packages in the course of their studies.

The feedback given was very positive. Users were unanimous in approving the randomisation of overlays; comments were of the form 'It's nice that no network covers up all the others.' The newly-included filter was also received positively. Several testers stated that they found it helpful, and seven stated, to our surprise, that it was more useful for the juxtaposed views than the combined as it allowed them to keep better track of the relations between the graphs.

The results for this set of tests are shown in Figure 45 and Figure 46 below.

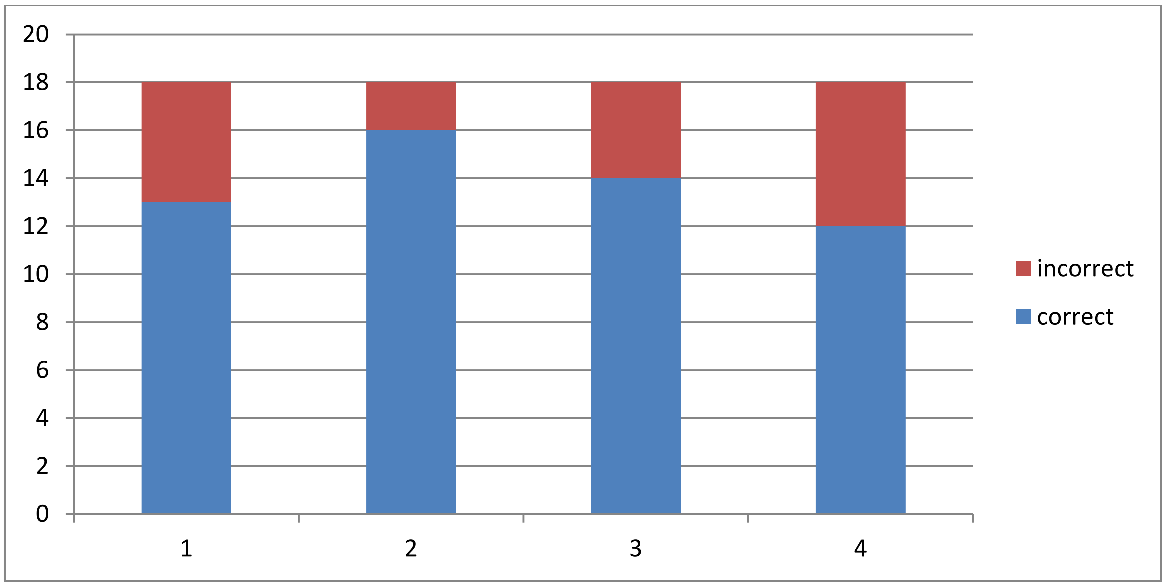


Figure 45: results of superposed node-link view

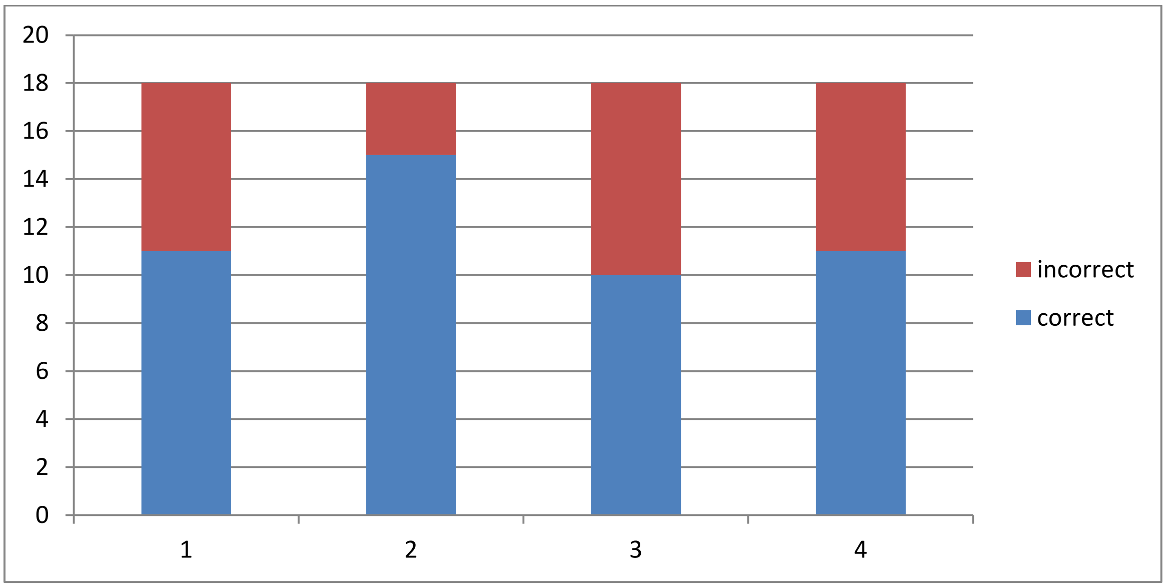


Figure 46: results of juxtaposed node-link views

The results were as follows:

1. When tasked to find differences on a specific labelled edge (the difference was a different origin) the results also favoured the combined view, with thirteen correct to eleven.

2. When asked to simply list all the differences that could be found between one of the given graphs and each of the others, the results again favoured the combined view, by a margin of just one correct answer (sixteen to fifteen).
3. When asked to find a specific edge found in only one graph of the four the results favoured the combined view (fourteen correct to ten);
4. When tasked to find a vertex that connected in only one of the four graphs, the results again favoured the combined view, but only by one correct answer (twelve correct to eleven);

As with the matrix-based views these results were not statistically significant for an unpaired t-test.

For a paired t-test the results were likewise not quite statistically significant with the exception of the first, which had a P-value of 0.0416. Were these results repeated with twice the number of testers we would be able to state without qualification (95% confidence interval) that the combined view is best. As it is however all that can be stated is that the combined view appears to be perform better and is definitely superior when locating edge differences.

Widening the confidence interval to 90% does not provide statistical significance on an unpaired two-tailed t-test, but does do so for a paired test. Given that the paired test compares the results of one tester using one view to that same tester using a different view, we can make the following statement:

We can state with 90% certainty that the combined Node-Link visualisation gives superior results to the equivalent juxtaposed linked views.

7.5 Test results analysis – Node-link vs Matrix

When comparing the results from the matrix-based visualisation to those from the Node-Link, we found that the matrix-based views scored marginally higher; an average improvement in mean score of 0.0555. This could not be considered to be statistically significant however, given the number of testers. To demonstrate categorically that this sort of difference is not down to any factor other than the views themselves would require testing with many more subjects and this was impractical.

However it is safe to confirm that the combined matrix view is superior to juxtaposed matrix or Node-Link views.

It is also true that the combined Node-Link view appeared to give superior results to the juxtaposed matrix views. As before however we are unable to confirm that the difference is statistically significant.

Given that the reason for allowing a paired t-test in the previous sections was because each result had a matched result completed by the same individual – thus eliminating the possibility that differences in results were down to differences between the testers – it is not possible to use a paired t-test for comparing these results between Node-Link and Matrix-based visualisations. This is because variations in performance between individual testers as a reason for difference of results cannot be eliminated by this method.

The full test thus will need to be repeated using the combined views from the Node-Link and matrix-based tools to confirm which performed best at some future date. Our results are not sufficiently conclusive to state which is better or worse.

7.6 Test Results – Conclusion

Unfortunately the test group was insufficiently large to demonstrate a significant difference in performance between matrix-based and Node-Link visualisations.

In all categories, the matrix-based visualisations performed better than their Node-Link equivalent, with an average of 0.056 improvement in mean score. This however could have been down to random variation in the performance of our testers; 0.056 is one correct answer in eighteen.

Accordingly **it is not possible for us to definitively state that it is better to use a matrix-based visualisation than a Node-Link one for comparing graphs.**

However, it is entirely acceptable to state that, based on our test results, **matrix-based displays did not perform worse than node-link displays.**

Chapter 8 Conclusions and Future Work

8.1 Summary

We set out to determine whether it was more effective to use combined rather than juxtaposed views for comparing more than three graphs, and whether those views would give better results if based on the adjacency matrix rather than a Node-Link diagram.

We designed and built a tool that allowed us to compare the effectiveness of combined and juxtaposed views in graph comparison. We utilised in this tool a small enough data set that while still having areas of local density and therefore occlusion problems, would nevertheless allow effective use of both node-link and matrix-based displays.

In our tool we successfully used the technique of offsetting to allow the examination of multiple edges between two vertices. We used this technique in both node-link and matrix based visualisations.

We also successfully added a fast filter to our tool which allowed for elimination of display clutter.

We used task-based testing to examine the performance of combined views and juxtaposed views in both matrix and node-link forms, with filtering, and compared the results to each other.

After looking at our results we are able to draw several conclusions.

8.2 Conclusions

The limitations of this research have been demonstrated via the difficulty in definitively stating which technique (matrix or node-link) is better used in the field of graph comparison. Whilst the methodology was basically sound, the issue will always arise in any task-based assessment of individual performance and how best to eliminate it as a factor. We considered using a greater number of testers, but this is not always practical, nor, in the case of close results like ours, likely to provide much clarity. The most effective method of testing in this manner requires the facility to run a paired statistical test where every result on one technique has a result using the other technique by the same individual. The lack of such facility prevented us from drawing definite conclusion regarding matrices vs node-links.

Those limitations aside however, we can say the following:

First, we can say that when comparing several graphs, it is more helpful to the comparison to have all of them shown on a single combined view rather than on multiple juxtaposed views.

Second, we can say that, for graphs sufficiently small that Node-Link diagrams can be used, the differences in performance between Node-Link views and matrix-based views are not significant for superposed visualisations. Further, although our tests did not find it statistically significant, matrix-based views seemed to perform marginally better than the equivalent Node-link views, and we can therefore state, based on this anecdotal evidence, that a matrix-based visualisation, despite its more abstract nature, does not disadvantage a user when used to compare graph features.

This second point is of interest as it may apply in other fields. If abstraction is not a hindrance to graph comparison, then non-intuitive representations are likely to be helpful in comparing graphs and similar data structures, and therefore designers should not limit themselves to known single-structure analysis tools when exploring this area.

One interesting additional result of our testing is that offsetting edges on the diagonal in matrix-based displays seems to work well, especially when combined with the use of a mouseover cell display. Although we were not actually testing this for a possible visualisation technique it would nonetheless seem to be a useful result for the design of matrix-based visualisations of multiple graphs, or for multigraphs.

In relation to our original research questions, therefore, we can say the following:

We have confirmed that combined views are more effective than juxtaposed views, regardless of type. A combined matrix is better than juxtaposed node-links and also better than juxtaposed matrices, and a combined node-link is likewise better than both juxtaposed node-links and juxtaposed matrices.

We have further confirmed that matrix-based views are not disadvantaged by their greater level of abstraction when compared against Node-Link views of the same graphs. We cannot definitely say that matrix-based representations are better, but we can say that they are, at least, no worse.

8.3 Future Application

There are two interesting areas for future application of this work.

First, there is the issue of large(r) graphs. A large data set based on the data dependencies of the TaxVis software [13] was utilised during tool development. This data set has several hundred vertices in each version with up to 12000 edges, and was felt inappropriate for undertaking comparison of node-link versus matrix visualisations due to the "hairball" effect in such a large graph. It was clear that visual comparison of node-link graphs of such size would be impossible without specially designed functionality to support this. Several screenshots of the matrix view of this data set were shown in the chapter on tool development. However, it would be worthwhile investigating mechanisms to support the comparison of such large graphs, be these represented by node-link or matrix displays.

Second, the use of filters in comparison shows interesting potential. Node-link diagrams, as already stated, are generally considered to have usable limits of circa 150 vertices and 300 edges. Our filter however allows a user to look only at the subgraphs not held in common. Depending on the level of similarity, this could reduce the amount of data onscreen by a considerable amount. An exploration of the limits of this would be worthwhile, as it could allow the use of Node-Link displays in comparison of much larger graphs than would normally be the case for single graph analysis.

While we have already stated that matrix-based displays do not disadvantage analysts, the Node-Link diagram is easily the most common way to show graphs, and the use of a filter similar to ours to reduce the display size would allow users to retain the intuitive mental map that a Node-Link offers. This might ease the finding of common walks, for example.

8.4 Summary

This thesis has examined comparison of four graphs. It set out to answer whether a matrix-based display would be more effective than a node-link display for comparing graphs, and whether it was better to combine or juxtapose the graphs being compared.

It has demonstrated that the use of adjacency matrices in comparing multiple graphs is an effective technique. It has not shown whether matrices are superior or node-link displays or vice versa, but has demonstrated that matrices are at least no worse than the more common node-links.

It has shown that combining the graphs needing comparison into a single visualisation is more effective than juxtaposing them. It has also shown that a combined matrix is better than juxtaposed node-links and that a combined node-link view is better than juxtaposed matrices.

It has demonstrated the use of a simple filter to eliminate clutter, and that this filter was effective. It proposes that such a filter might be useful in comparison of large graphs.

It has likewise demonstrated the practical use of edge offsetting when combined with colour for both matrix-based and node-link graph visualisation and proposes that such might have useful application in the visualisations of both multiple graphs and of multigraphs.

Appendix A – Prototype Test Questions and Results

Question set 1

Question	Combined view **Matrix; node-link**	Separate views **Matrix; node-link**
1a Who won SB XXXII? (*DEN*)	7/7; 7/7	6/6; 6/7
1b Who did they beat? (*GB*)	6/7; 7/7	4/6; 6/7
2a Which team has never lost in 5 SB games? (*SF*)	3/7; 4/7	5/6; 4/7
2b Which team did they beat twice? (*CIN*)	3/7; 3/7	3/6; 3/7
3a The Bills (BUF) lost four consecutive games. Which games? (*SBs XXV, XXVI, XXVII, XXVIII*)	7/7; 7/7	6/6; 7/7
3b Who were their opponents in those games? (*NYG, WAS, DAL, DAL*)	7/7; 6/7	6/6; 6/7
4a The Colts have played in 3 games, but only one since they moved to their current location in Indianapolis (IND). Where were they based for their first two games? (*BAL*)	6/7; 6/7	0/6' 4/7
4b Which was the first game they played? (*SB III*)	6/7; 5/7	3/6; 5/7
4c Which was the second game? (*SB V*)	6/7; 6/7	2/6; 4/7
5 Washington Redskins (WAS) have lost twice. Which games and to whom did they lose? (*SB XVII to OAK/LA; SB VII to MIA*)	7/7; 7/7	6/6; 7/7
6a How many games have Oakland Raiders (OAK) win according to graph 1? (*3*)	6/7; 7/7	3/6; 6/7
6b How many according to graph 2? (*2*)	6/7; 6/7	3/6; 4/7
6c Which game is shown differently and what is the difference? (*SB XVIII*)	7/7; 5/7	3/6; 4/7
6d What is that difference? (*shows as OAK vs WAS in graph, LA vs WAS in graph2*)	4/7; 5/7	1/6; 3/7
6d What does the difference tell you about the Raiders? (*They were based in LA at one time*)	7/7; 4/7	1/6; 3/7

Question set 2

Question	Combined View **Matrix; node-link**	Separate View **Matrix; node-link**

1 Three teams have lost four SuperBowls. Which of these has also won the SuperBowl? (*DEN*)	6/6; 7/7	4/7; 3/7
2a Which team has played in the most SuperBowls? (*DAL*)	3/6; 5/7	6/7; 2/7
2b How many games did they win? (*5*)	3/6; 4/7	5/7; 2/7
2c How many games did they lose? (*3*)	2/6; 3/7	5/7; 1/7
3 Which teams have a 100% winning record in the Superbowl? A) (*SF*)	6/6; 7/7	4/7; 3/7
3… B) (*NYJ*)	5/6; 6/7	5/7; 4/7
3… C) (*TB*)	5/6; 5/7	4/7; 3/7
4 The Los Angeles Rams (LA) lost SuperBowl XIV. They have since moved. To where? (*STL*)	5/6; 5/7	3/7; 3/7
5 Two teams have been based in LA but are no longer. The Rams (above) are one. Which is the other? (*OAK*)	5/6; 7/7	4/7; 3/7
6 How many teams have been based in Baltimore(BAL)? (*2*)	5/6; 4/7	3/7; 4/7

Appendix B Main Test Questions and Results

Session One

Question 1: There are several differences between graph 2 and one or other graph. What difference is there between graph 2 and **all** of the other three graphs?

(Answer: There is an edge between BAL and NYJ labelled SBIII)

Question 2: When a franchise moves location its vertex representation will change in graph 1 and graph 3 (which show the team's current location), but not in graph 2 or graph 4 (which show the location at the time the game was played). From this information, what was the original location of the team currently located in Indianapolis (IND)?

(Answer: Baltimore – BAL)

Question 3: Three teams have lost four games. In one of the graphs this is not correctly shown, and one of the three shows only three losses.

a) Which graph shows the incorrect losing team, and

b) what is that team's location shown as?

(Answer: a) graph 3 shows b) CLE instead of DEN)

Session Two

The following questions were added at different points in the task list for the second session in order to help mask the fact that the questions from session 1 were repeated.

Question 1: An edge represents a game between two teams. In one of the four graphs, there are several edges unique to that graph. Which edge is that?

(Answer: NY)

Question 2: The New York Giants and New York Jets are both shown in graph 4 simply by their location, NY. However, one of their games is shown incorrectly in graph 4. What location is shown instead of NY?

(Answer: PHI)

Question 3: According to graph 4 Cleveland Browns (CLE) lost to a team from New York (NY) in SuperBowl XXI. This is incorrect.

a) Which teams played in that game, and

b) who won?

(Answer: NYG beat DEN)

Results Breakdown - Matrix

Blue – Combined view; Red – juxtaposed views

Session One

Question	1	2	3a	3b
Tester 1	1	1	0	0
Tester 2	1	1	1	1
Tester 3	0	0	0	0
Tester 4	0	0	1	1
Tester 5	1	0	0	0
Tester 6	1	1	1	1
Tester 7	1	1	1	1
Tester 8	1	1	1	1
Tester 9	0	0	1	1
Tester 10	0	0	0	0
Tester 11	1	1	0	0
Tester 12	1	1	1	1
Tester 13	1	1	1	0
Tester 14	0	1	1	1
Tester 15	1	1	1	1
Tester 16	1	1	1	0
Tester 17	1	0	1	1
Tester 18	0	1	1	1

Session Two

Question	1	2	3a	3b
Tester 1	1	1	1	1
Tester 2	1	1	0	0
Tester 3	0	0	0	1
Tester 4	0	0	0	0
Tester 5	1	1	0	0
Tester 6	1	1	0	0
Tester 7	1	1	1	1
Tester 8	1	1	1	1
Tester 9	0	1	1	1
Tester 10	0	0	0	0
Tester 11	1	1	1	1
Tester 12	1	1	1	1
Tester 13	1	1	1	1
Tester 14	1	1	0	0
Tester 15	0	1	1	1
Tester 16	1	1	0	0
Tester 17	1	1	1	1
Tester 18	0	0	0	1

Results Breakdown – Node-Link

Blue – Combined view; Red – juxtaposed views

Session One

Question	1	2	3a	3b
Tester 1	1	1	0	0
Tester 2	0	1	1	1
Tester 3	0	1	0	0
Tester 4	1	1	1	1
Tester 5	0	0	1	1
Tester 6	1	1	1	1
Tester 7	1	1	0	0
Tester 8	1	1	0	0
Tester 9	1	1	1	1
Tester 10	0	0	1	0
Tester 11	1	1	1	1
Tester 12	1	1	1	1
Tester 13	0	1	0	0
Tester 14	1	1	1	1
Tester 15	0	0	0	0
Tester 16	1	1	0	0
Tester 17	0	1	1	1
Tester 18	0	1	0	0

Session Two

Question	1	2	3a	3b
Tester 1	0	1	1	1
Tester 2	1	1	1	1
Tester 3	0	1	1	1
Tester 4	1	1	1	1
Tester 5	1	0	1	0
Tester 6	1	1	1	1
Tester 7	1	1	1	1
Tester 8	0	0	1	1
Tester 9	1	1	1	1
Tester 10	1	1	0	0
Tester 11	1	1	1	1
Tester 12	1	1	1	1
Tester 13	1	1	1	1
Tester 14	1	1	1	1
Tester 15	1	1	0	0
Tester 16	0	1	0	0
Tester 17	1	1	1	1
Tester 18	1	1	0	1

Bibliography

[1] P. Brass, E. Cenel, C. A. Duncan, A. Efrat and E. Cesim, “On simultaneous planar graph embeddings,” *Computational Geometry: Theory and Applications,* vol. Vol. 36, pp. 117-130, 2007.

[2] M. Andries, G. Engels, A. Habel, B. Hoffman, H.-J. Kreowski, S. Kuske, D. Plum, A. Schürr and G. Taentzer, “Graph transformation for specification and programming,” *Science of Computer Programming,* vol. Vol. 34, pp. pp 1-54, 1999.

[3] J.-D. Fekete, D. Wang, N. Dang and C. Plaisant, “Overlaying Graph Links on Treemaps,” in *presented at IEEE InfoVis Poster Compendium*, Seattle, Washington, USA, 2003.

[4] P. Eades and M. L. Huang, “Navigating Clustered Graphs using Force-Directed Methods,” *Journal of Graph Algorithms and Applications,* vol. Vol.4, pp. 157-181, 2000.

[5] E. M. Luks, “Isomorphism of graphs of bounded valence can be tested in polynomial time,” *Journal of Computer and System Sciences ,* vol. 25 (1), pp. 42-65, 1982.

[6] R. Southwell, “Finding Symmetries in Graphs,” in *Dept of Mathematics, vol. MSc Mathematics with modern applications:*, University of York, 2005.

[7] D. Archambault, T. Munzner and D. Auber, “TopoLayout: Multi-Level Graph Layout by Topological Features,” *IEEE Transactions on Visualization and Computer Graphics,* vol. vol. 13, pp. 305-317, 2006.

[8] A. L. Buchsbaum, C. M. Procopiuc, S. Venkatasubr and E. R. Gansner, “Rectangular Layouts and Contact Graphs,” AT&T Technical Report TD-59JQX5, 2005.

[9] P. Riehmann, M. Hanfler and B. Froelich, “Interactive Sankey

Diagrams,” in *InfoVIs'05*, 2005.

[10] G. D. Battista, P. Eades, R. Tamassia and I. G. Tollis, “Algorithms for drawing graphs: an annotated bibliography,” *Computational Geometry Theory and Applications,* vol. Vol 4, pp. 235-282, 1994.

[11] M. Graham and J. Kennedy, “Combining linking & focusing techniques for a multiple hierarchy visualisation,” in *InfoVis 2001*, London, 2001.

[12] M. Graham, J. Kennedy and L. Downey, “Visual Comparison and Exploration of Natural History Collections,” in *Advanced Visual Interfaces (AVI) 2006*, Venice, Italy, 2006.

[13] M. Graham and J. Kennedy, “Multiform Views of Multiple Trees,” in *12th International Conference on Information Visualisation (IV'08)*, London, 2008.

[14] T. Dwyer, B. Lee, D. Fisher, K. I. Quinn, P. Isenberg, G. Robertson and C. North, “A Comparison of User-Generated and Automatic Graph Layouts,” *IEEE Transactions on Visualization and Computer Graphics,* vol. 15, no. 6, pp. 961 - 968, 2009.

[15] P. Eades, “A heuristic for graph drawing,” *Congressus numerantium,* vol. vol. 42, pp. 149-160, 1984.

[16] T. M. J. Fruchterman and E. M. Reingold, “Graph drawing by force-directed placement,” *Software - Practice and Experience,* vol. Vol. 21, pp. 1129-1164, 1991.

[17] T. M. D. Ebbels, B. F. Buxton and D. T. Jones, “SpringScape: Visualisation of microarray and contextual bioinformatic data using spring embedding and an 'information landscape',” *Bioinformatics,* vol. Vol.22, pp. e99-e107, 2006.

[18] C. Erten, S. G. Kobourov, V. Le and A. Navabi, “Simultaneous Graph Drawing: Layout Algorithms and Visualization Schemes,” in *11th*

Symposium on Graph Drawing, 2005.

[19] S. C. North, "Neato User's Guide," 1992. [Online]. Available: ftp://ftp.hgsc.bcm.tmc.edu/pub/data/dicty/docs/neatoguide.pdf. [Accessed 1st June 2012].

[20] G. Kumar and M. Garland, "Visual exploration of time-varying graphs," *IEEE Transactions on Visualization and Computer Graphics,* vol. Vol. 12, 2006.

[21] H. Omote and K. Sugiyama, "Force-directed drawing method for intersecting clustered graphs," in *IEEE Asia-pacific Symposium on Visualisation 2007*, 2007.

[22] R. Davidson and D. Harel, "Drawing graphs nicely using simulated annealing," *ACM Transactions on Graphics,* vol. Vol. 15, pp. 301-331, 1996.

[23] G. Di Battista, P. Eades, R. Tamassia and I. G. Tollis, Graph Drawing: Algorithms for the Visualization of Graphs, 1st ed, Prentice Hall, 1999.

[24] D. Emig, K. Klein, A. Kunert, P. Mutzel and M. Albrecht, "Visualizing domain interaction netwroks and the impact of alternative splicing events," in *IV'08/MediVis'08*, London, 2008.

[25] J. Lamping and R. Rao, "The Hyperbolic Browser: A Focus+Context technique for visualizing large hierarchies," *Journal of Visual Languages and Computing,* vol. Vol.7, pp. 33-55, 1996.

[26] H. C. Purchase, "Effective information visualisation: a study of graph-drawing aesthetics and algorithms," *Interacting with Computers,* vol. vol. 13, pp. 147-162, 2000.

[27] H. C. Purchase, "Metrics for Graph Drawing Aesthetics," *Journal of Visual Languages and Computing,* vol. vol. 13, pp. 501-516, 2002.

[28] H. C. Purchase, M. McGill, L. Colpoys and D. Carrington, “Graph drawing aesthetics and the comprehension of UML class diagrams: an empirical study,” in *2001 Asia-Pacific symposium on Information visualisation*, Sydney, Australia, 2001.

[29] C. Ware, H. C. Purchase, L. Colpoys and M. McGill, “Cognitive measurements of graph aesthetics,” *Information Visualization,* vol. vol.1, pp. 103-110, 2002.

[30] H. C. Purchase, D. Carrington and J.-A. Allder, “Empirical evaluation of aesthetics-based graph layout,” *Empirical Software Engineering,* vol. vol.7, pp. 233-255, 2002.

[31] H.-J. Schulz and H. Schumann, “Visualizing Graphs - a generalised view,” in *Tenth International Conference on Information Visualization*, 05-07 July 2006.

[32] M. Ghoniem, J.-D. Fekete and P. Castagliola, “A Comparison of the Readability of Graphs Using Node-Link and Matrix-Based Representations,” in *IEEE Symposium on Information Visualization*, Austin, Tex., 2004.

[33] D. Holten and J. J. van Wijk, “A User Study on Visualizing Directed Edges in Graphs,” in *CHI '09 Proceedings of the 27th international conference on Human factors in computing systems*, Boston, Ma., April 4 - 9 2009.

[34] P. Riehmann, M. Hanfler and B. Froelich, “Interactive Sankey diagrams,” in *InfoVis '05*, 2005.

[35] P. Craig, “Animated interval scatter-plot views for the exploratory analysis of large scale microarray time-course data,” School of Computing, vol. PhD, Edinburgh Napier University, Edinburgh, 2006.

[36] T. Munzner, “H3: Laying out large directed graphs in 3D hyperbolic space,” in *IEEE Symposium on Information Visualization (InfoVis'97)*,

1997.

[37] N. Henry and J.-D. Fekete, “MatLink: enhanced matrix visualization for analyzing social networks,” in *Human-Computer Interaction – INTERACT 2007*, 2007.

[38] R. Keller, C. M. Eckert and P. J. Clarkson, “Matrices or node-link diagrams: which visual representation is better for visualising connectivity models?,” *Information Visualization,* vol. vol. 5, pp. 62-76, 2006.

[39] S. S. Chawathe and H. Garcia-Molina, “Meaningful change detection in structured data,” in *International Conference on Managing of Data ACM SIGMOD '97*, 1997.

[40] S. Melnik, H. Garcia-Molina and E. Rahm, “Similarity Flooding: a versatile graph matching algorithm and its application to schema matching,” in *18th International Conference on Data Engineering (ICDE)*, San Jose, CA, USA, 2002.

[41] L. Wiskott, J.-M. Fellous and N. Krüger, “Face recognition by elastic bunch graph matching,” *IEEE Transactions on Pattern Analysis and Machine Intelligence,* vol. vol. 19, pp. pp. 775-779, 1997.

[42] L. P. Cordella, P. Foggia, C. Sansone and M. Vento, “An improved algorithm for matching large graphs,” in *3rd IAPR-TC15 Workshop on Graph-based Representations*, Italy, 2001.

[43] J. E. Hopcroft and J. K. Wong, “Linear time algorithm for isomorphism of planar graphs (Preliminary Report),” in *Sixth Annual ACM Symposium on Theory of Computing*, Seattle, WA, USA, 1974.

[44] T. Caelli and S. Kosinov, “An eigenspace projection clustering method for inexact graph matching,” *IEEE Transactions on Pattern Analysis and Machine Intelligence,* vol. vol. 26, pp. pp. 515-519, 2004.

[45] B. T. Dai, N. Koudas, D. Srivasta, A. K. Tung and S. Venkatasubraman, “Validating Multi-column Schema Matchings by Type,” in *ICDE 2008*, 2008.

[46] A.-S. Dadzie and A. Burger, “Providing visualisation support for the analysis of anatomy ontology,” *BMC Bioinformatics 2005,* vol. vol. 6, 2005.

[47] W. Li, P. Eades and S.-H. Hong, “Navigating Software Architectures with Constant Visual Complexity,” in *IEEE Symposium on Visual Languages and Human-Centric Computing (VLHCC'05)*, 2005.

[48] L. Chen, “Graph Isomorphism and Identification Matrices: Parallel Algorithms,” *IEEE Transactions on Parallel and Distributed Systems,* vol. vol. 7, pp. 308-319, 1996.

[49] M. Kuramochi and G. Karypis, “Finding Frequent Patterns in a Large Sparse Graph,” *Data Mining and Knowledge Discovery,* vol. Vol. 11, pp. 243-271, 2005.

[50] B. Gallacher, “Matching Structure and Semantics: A survey on graph-based pattern matching,” in *AAAI Fall Symposium on Capturing and Using Patterns for Evidence Detection (AAAI FS '06)*, 2006.

[51] F. Giunchiglia and P. Shvaiko, “Semantic matching,” *The Knowledge Engineering Review,* vol. Vol. 18, no. 3, pp. 265-280, 2003.

[52] J. Y. Hong, J. D'Andries, M. Richman and M. Westfall, “Zoomology: Comparing Two Large Hierarchical Trees.,” in *IEEE InfoVis Poster Compendium*, Seattle, Wa., 19-21 October 2003.

[53] P. Saraiya, P. Lee and C. North, “Visualization of Graphs with Associated Timeseries Data,” in *IEEE InfoVis*, Minneapolis, Minn., October 23-25 2005.

[54] A. Telea, F. Frasincar and G.-J. Houben, “Visualisation of RDF(S)-

based Information," in *Conference on Information Visualization - IV 2003*, London, 2003.

[55] K. Andrews, M. Wohlfahrt and G. Wurzinger, "Visual Graph Comparison," in *13th International Conference on Information Visualisation IV'09*, Barcelona, July 2009.

[56] S. Diehl, C. Goerg and A. Kerren, "Preserving the mental map using foresighted layout," in *Joint Eurographics/IEEE TCVG Symposium on Visualization (VisSym 2001)*, May 2001.

[57] M. Gleicher, D. Albers, R. Walker, I. Jusufi, C. D. Hansen and J. C. Roberts, "Visual comparison for information visualization," *Information Visualisation,* vol. 10, no. 4, pp. 289-309, 2011.

[58] C. Collins and S. Carpendale, "VisLink: Revealing Relationships Amongst Visualizations," *IEEE Transactions of Visualization and Computer Graphics,* vol. vol.13, no. 6, pp. 1192-1199, 2007.

[59] M. T. Goodrich and R. Tamassia, Data Structures and Algorithms in Java, 2nd ed., New York: Wiley & Sons, 2001.

[60] M. S. Marshall, I. Herman and G. Melancon, "An object-oriented design for graph visualization," *Software: Practice and Experience,* vol. 31, no. 8, pp. 739-756, 2001.

[61] J. Barnes and P. Hut, "Error Analysis of A Tree Code," *The Astrophysical Journal Supplement Series,* vol. 70, pp. 389-417, 1989.

[62] D. A. Keim, "Pixel-oriented visualisation techniques for exploring large databases," *Journal of Computational and Graphical Statistics,* vol. vol.5, pp. pp. 58-77, 1996.

[63] D. A. Keim, "Designing Pixel-Oriented Visualization Techniques: Theory and Applications," *IEEE Transactions on Visualization and Computer Graphics,* vol. vol. 6, pp. 59-78, 2000.

[64] F. Beck and S. Diehl, “Visual comparison of software architectures,” in *SOFTVIS '10 Proceedings of the 5th international symposium on Software visualization*, 2010.

[65] H. Sharara, A. Sopan, G. Namata, L. Getoor and L. Singh, “G-Pare: A Visual Analytic tool for Comparative Analysis of Uncertain Graphs,” in *IEEE Symposium on Visual Analytic Science and Technology*, 2011.