UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

ANDRE SUSLIK SPRITZER

MagnetViz: Design and Evaluation of a Physics-based Interaction Technique for Graph Visualization

Thesis presented in partial fulfillment
of the requirements for the degree of
Master of Computer Science

Profa. Dra. Carla Maria Dal Sasso Freitas
Advisor

Porto Alegre, April 2009

CIP – CATALOGING-IN-PUBLICATION

Spritzer, Andre Suslik

MagnetViz: Design and Evaluation of a Physics-based Interaction Technique for Graph Visualization / Andre Suslik Spritzer. – Porto Alegre: PPGC da UFRGS, 2009.

102 f.: il.

Thesis (Master) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR–RS, 2009. Advisor: Carla Maria Dal Sasso Freitas.

1. Graph visualization. 2. Force-directed layouts. 3. Interaction. 4. Information visualization. I. Freitas, Carla Maria Dal Sasso. II. Título.

ACKNOWLEDGEMENTS

First and foremost I would like to thank my advisor Carla Maria Dal Sasso Freitas, with whom I have been working ever since I was an undergraduate student. Her support and advice have been essential for the realization of this work.

I would also like to thank the UFRGS computer graphics group, a group of which I am proud to be a part of. Many essential parts of this work resulted from the numerous discussions (formal or not) and cafeteria conversations with the students and professors of the CG group, most of whom are great friends as well as colleagues. Even though they were not exactly fundamental for this work, the many parties, ‘churrascos’ and other events were also a very welcome part of the past years and a good deal of what made working with this group so great.

I would also like to thank Professor Renata Galante and her students, for providing the dataset and help with the elaboration of a set of questions regarding it for the case study performed in this work. Also essential for this work were the datasets and suggestions given by Professor Jean Daniel Fekete and his Ph.D. student Nathalie Henry, of INRIA-Paris.

I would like as well to thank my friends from Brazil, France, Germany, Paraguay and the U.S.. Although they are not as directly involved with this work, they were no less important for its realization. They have contributed with their support and understanding and although they don’t get me publications, they help me grow as a human being, which ultimately reflects on my work (even though sometimes they do make my work harder by encouraging me to procrastinate - an essential part of research, of course - through funny youtube videos and invitations for parties, happy hours and other events :-).

A special thanks should also be given to some friends who went along with me from graduate (some from even elementary school) to post-graduate school and are now embarking on Ph.D.’s or working on big companies throughout the world. Even though our works were completely unrelated, we’ve all shared some of the same burdens and joys of academic life.

Last, but by no means least, I would like to thank my family. They are also not directly related to this work, but they are the people that have to put up with me on a daily basis and who provide me with unconditional support. They are thus a big part of what made this work possible by helping me become who I am today.

CONTENTS

LIST OF FIGURES

ABSTRACT

This dissertation presents MagnetViz, a technique for the visualization of graphs. While most techniques visualize a static pre-computed graph layout, MagnetViz allows users to dynamically alter the layout of a graph to better satisfy their needs. This is done by building on the physics metaphor of force-directed algorithms to provide users with virtual magnets, which can attract nodes that fulfill a set of criteria associated with them. Criteria can be based on either the topology or semantics of the graph. Through boundary shapes, which are simple geometric shapes that can be placed around magnets, users can also define regions within the scene where the attracted nodes should remain.

Graphs are described in GraphML, a XML-like description language which allows the specification of nodes and edges between nodes as well as attributes associated to nodes and edges. After loading a graph, Magnetviz displays it using a slightly modified version of the classical Fruchterman and Reingold' algorithm, and allows the user to insert magnets. Users can build the criteria associated with the magnets using the attributes of nodes and/or edges, besides the common graphs' topological attributes.

For MagnetViz's evaluation, it was first analyzed how the technique fared in aiding users to perform tasks defined by a graph visualization task taxonomy described in the literature. Then, MagnetViz was tested within a practical context by means of a case study. A co-authorship network was chosen as the target dataset. The MagnetViz prototype was initially used to answer questions relevant to this dataset and then tested by a group of potential users, who had to use it to answer these same questions. After trying the application, subjects answered questionnaires about their opinion on the technique's usability, applicability, relevance and visual results.

While some aspects of the technique should still be refined, results of the evaluation proved MagnetViz to be a valid approach when it comes to interaction with graph visualizations.

Keywords: Graph visualization, force-directed layouts, interaction, information visualization.

MagnetViz: Projeto e Avaliação de uma Técnica de Interação Baseada em Física para Visualização de Grafos

RESUMO

Esta dissertação apresenta MagnetViz, uma técnica para visualização de grafos. Enquanto a maior parte das técnicas visualizam um layout de grafo estático pre-computado, MagnetViz permite que usuários dinamicamente alterem o layout de um grafo de forma a melhor satisfazer suas necessidades. Isso é feito ao construir em cima da metáfora de física de algoritmos dirigidos à força para proporcionar aos usuários imãs virtuais, que podem atrair nodos que satisfazem um conjunto de critérios associados a eles. Critérios podem ser baseados na topologia ou semântica do grafo. Através de *boundary shapes*, que são simples formas geométricas que podem ser colocadas ao redor de imãs, usuários podem também definir regiões na cena onde os nodos atraídos devem permanecer.

Grafos são descritos usando GraphML, uma linguagem baseada em XML, que permite a especificação dos nodos e arestas e de atributos para essas entidades. Após a submissão de um grafo como entrada, MagnetViz o exibe utilizando uma versão modificada do algoritmo clássico de Fruchterman and Rheingold, e permite que usuário, a seguir, insira imãs na cena. Usuários podem construir as condições associadas aos imãs utilizando os atributos dos nodos e arestas, além de atributos topológicos próprios de grafos.

Para a avaliação de MagnetViz, foi primeiro analisado o desempenho da técnica ao ajudar usuários a executarem tarefas definidas por uma taxonomia de tarefas de visualização de grafos encontrada na literatura. Então, MagnetViz foi testada em um contexto prático através de um estudo de caso. Uma rede de co-autorias foi escolhida como conjunto de dados e o protótipo de MagnetViz foi inicialmente usado para responder questões relevantes a esses dados e então testado por um grupo de potenciais usuários, que tinham de usa-lo para responder essas mesmas perguntas. Após testar a aplicação, os sujeiotos receberam questionários sobre usas opiniões quanto a usabilidade, aplicabilidade, relevância e resultados visuais da técnica.

Enquanto alguns aspectos da técnica ainda podem ser melhorados, os resultados da avaliação provaram que MagnetViz é uma abordagem válida para interação com visualizações de grafos.

Palavras-chave: visualização de grafos, layouts baseados em física, interação, visualização de informações.

1 INTRODUCTION

1.1 Motivation

A graph is an abstract structure used to represent information that is comprised of objects and relationships between those objects. Due to its intuitive use, simple and easy to understand definition and power as a modeling and representational tool for relational data, graphs have become the structure of choice for representing information in many fields, such as Biology, Economy, Psychology and Computer Science.

In some applications, graphs are simply the data structure in which the information is internally represented, not explicitly manipulated by the end-user. Others, though, require the user to deal directly with them. This leads to the necessity of systems that allow the user to explore and manipulate the information contained in the graphs.

A user might textually explore the information contained in a graph through tools such as query languages. For a user who is not very computer-savvy, though, this might prove to be too complex and not very intuitive. Therefore, it is interesting for many applications to visualize those graph structures in order to make it easier for the users to read and manipulate the information as required.

The most common approach to visually representing a graph is the use of node-link diagrams. These consist simply of visual objects representing the nodes, which are connected by lines that represent the edges. There is a large community of researchers dedicated to studying algorithms to lay out those nodes and edges in the best possible way.

The problem that the field of graph drawing tries to solve can be very simply stated as how to find the geometric positions of the nodes that are aesthetically more interesting for the better comprehension of the graph and its structure. Its solution, though, has proven to be quite complex.

Different layout algorithms for node-link diagrams of graphs have been created, each favoring certain aesthetic criteria, such as edge crossings, edge bends, graph symmetry, etc., in detriment of others. Graph aesthetics, however, is very subjective. While there are some guidelines as to what might lead to a good layout (that is, a layout that is easily readable by a human) they are far from being hard rules. This leads to techniques that might work perfectly for some applications but not at all for others. A good source on the classical graph drawing algorithms is the book written by Di Battista et al. (1999).

To circumvent the limitations of layout algorithms, many approaches have been experimented with. Some have attempted to deal with the problem by using creative navigation and interactive schemes, building on the knowledge provided by the fields of Information Visualization and Human-Computer Interaction. Others have tried to build 3D visualization techniques, or even created alternative visual representations altogether. Some have also combined all of these approaches into new hybrid techniques. Most graph drawing and visualization techniques take into account only the topological structure of a graph. While they might frequently produce aesthetically pleasing visualizations, they might be leaving out important data in their computation of the layout by ignoring the semantic information contained in the attributes of the nodes and edges. Even though there are some algorithms that produce attribute-based layouts, these are few and mostly not very generically applicable. Therefore, it is usually the case that semantic information is handled by interaction techniques, such as filtering.

Depending on the application, semantic information might be just as important to the user as the topology of the graph. Leaving out this information might result on the user missing out on important data and even relationships that are not explicitly expressed.

1.2 Objective

The goal of this work is to investigate an alternative visualization technique which combines a flexible layout with interaction mechanisms that allow the user to explore the graph. The proposed technique builds on the metaphor of physics-based graph-layout algorithms. It consists of providing the user with virtual magnets that are associated with user-defined criteria. These criteria can be topology or attribute-based, and allow the interactive manipulation of the visualization of a graph in order to make it semantically more interesting and valuable. Magnets can also be used to intuitively perform set operations such as union and intersection, becoming a visual tool for exploring the datasets, and allowing the user to discover new relationships that were previously invisible due to the techniques that focused exclusively on the topology. To illustrate and validate the technique, it is analyzed in terms of a task taxonomy and how it can be used in practice in the context of a graph representing a social network.

1.3 Organization of the Text

The rest of this work is organized as follows. Chapter 2 provides a background on graphs and graph drawing, with emphasis given to force-directed layout algorithms, specifically to the ones employed by the proposed technique. Chapter 3 covers the related work, illustrating techniques for interactive layout reorganization, focus+context interaction and relevant graph visualization techniques and toolkits. The proposed technique is detailed in chapter 4, while its evaluation process and results are shown in chapter 5. Conclusions and the planned future work are described in chapter 6’.

2 BACKGROUND

This chapter provides some background on all the topics necessary to the comprehension of the technique proposed in this work. A brief review of some graph theory concepts is presented and followed by an overview of the field of graph drawing. Force-directed graph layout algorithms are then covered, with the one employed by the technique proposed in this work explained in more detail.

2.1 Basic Concepts

In many fields it is often necessary to represent objects and the relationships between these objects. Graphs provide a very simple, powerful and intuitive way of doing so and are thus vastly used in fields ranging from software engineering, databases and artificial intelligence to medical science, biology, chemistry and many others. In computer science, graphs are one of the most commonly used data structures.

Formally, a graph can be simply defined as a finite set of nodes and a finite multi-set of edges, with the former standing for the objects and the latter for the relationships between those objects. An edge is defined as a pair of nodes. If both nodes of an edge are the same, it is said to be a self-loop. If an edge occurs more than once, it is called a multiple edge. If a graph has no loops and no multiple edges, it is known as a simple graph. Two nodes that are a part of the same edge are adjacent to each other. The neighbors of a node are the nodes adjacent to it and its degree is the number of its neighbors. A graph is said to be non-directed if the edges are unordered pairs of nodes, and is said to be regular if all its nodes have the same degree.

A directed graph (or digraph) is found almost as commonly as regular graphs and is just as simple, with the only difference being that an edge is defined by an ordered pair of nodes. In a digraph, an edge goes from node A to node B, being an outgoing edge of A and an incoming edge of B. Nodes may have no incoming or no outgoing edges, being then known as sources and sinks, respectively.

In a digraph, a path is a sequence of nodes in which there is always an outgoing edge of a node to the one that immediately follows it (except for the last node of the sequence). If the paths form cycles (paths that begin and end with the same node), the graph is known as cyclic. Otherwise, it is acyclic. If there is a path for each pair of nodes, the graph is said to be connected.

An important special case of a graph is the tree, which is used to represent hierarchical structures and is so called due to its resemblance to biological trees. A tree is a graph in which every two nodes are connected by at most one path, thus being connected and acyclic. An important type of tree is the rooted tree, in which each node (with the exception of the root), has one parent node and may have children nodes. If a node has no children, it is a leaf node.

2.2 Graph Drawing

Graph drawing algorithms are of course essential to graph visualization techniques. Here we present a short background on the subject. For a more detailed study of graph drawing, the reader can refer to the book by Di Battista et al. (1999).

The problem that graph drawing techniques attempt to solve can be very simply stated: given a graph, calculate the position of its nodes and the curve to be drawn for each edge (HERMAN et al., 2000). Finding a solution to this problem, though, is not as trivial.

Many layout algorithms have been developed (BATTISTA et al., 1999) and are widely used in graph visualizations. Each algorithm takes into account different aesthetic criteria, such as number of bends in an edge, amount of line crossings and the size of the area occupied by the drawing, and very often will only work on specific types of graphs.

Graph drawing algorithms usually define a rule, or drawing convention, that the generated drawing must adhere to. These conventions might be very complex requirements, but they are usually quite simple, with some of the most often used being the polyline, straight-line and orthogonal drawings (Figure 2.1). Constraints are often placed on the drawing, such as forcing the algorithm to always place a certain vertex at a certain position, clustering (placing a subset of nodes close together according to certain criteria) or drawing a certain subgraph with a predefined shape.

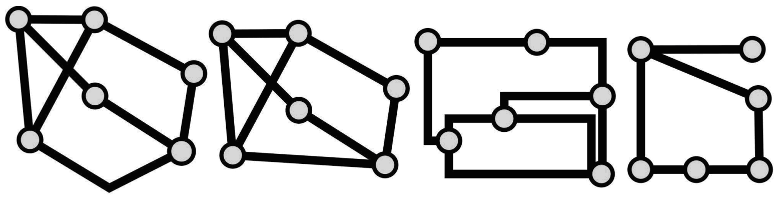


Figure 2.1: Common graph layouts. From left to right: polylines, line segments, orthogonal and grid layout.

It is also desirable that a graph drawing algorithm be predictable, so that two different runs with the same or similar graphs should produce similar visual representations (HERMAN et al., 2000). This way the user can maintain a consistent "mental map" of the drawing.

There is no graph drawing algorithm that could be considered the best - as said before, the effectiveness of graph drawings depends on their readability, and that changes depending on how (and where) the technique is being applied. User studies show that a drawing is more readable if it has no edge-bends, no edge-crossing and small edge-lengths, occupies small screen-space, displays symmetries and satisfies application-specific constraints such as clustering and alignment (BATINI; FURLANI; NARDELLI, 1985; COHEN et al., 1994; PURCHASE; COHEN; JAMES, 1997). It is usually the case that those readability criteria cannot all be satisfied simultaneously, so trade-offs must be made.

There are many different types of graph drawing algorithms, with the optimization-based approach being of particular interest to us. Optimization-based layout algorithms rely on a series of mathematical equations that act as constraints for the drawing and are passed to a "solver", which tries to optimize them. Such constraints might be the minimization of edge crossings and edge bends, for instance. Optimization-based layouts tend to generate organic-looking layouts that can be applied on all types of graphs and are very customizable. They are also interesting for allowing the user to see the graph being built in a real-time animation by rendering the intermediary steps of the layout computation process. The main drawbacks of this type of layout are that it tends to be very computationally expensive (especially with larger graphs) and that it is not deterministic.

For the user to be able to interact wih the graph in real-time it is very important that a graph drawing algorithm be efficient. Due to faster, multi-core CPUs and GPUs, performance is gradually ceasing to be a problem.

Of the optimization-based techniques, the ones that are of particular interest to us are the force-directed algorithms, which we will detail in the following section.

2.3 Force Directed Layouts

The most popular and widely used class of techniques that use the optimization-based approach is the force-directed or spring layout. Their pleasing visual results, relative implementation simplicity and flexibility as to the inclusion of new aesthetic criteria make them a staple of graph visualization in general.

The basic idea of a force-directed algorithm is to treat the graph as a physical system, assigning forces to the nodes and edges, and minimizing its energy until reaching a stable layout. The forces will work to rearrange the positions of the nodes until the system finds itself in a state of mechanical equilibrium.

One interesting property of force-directed algorithms is that most of them support the application of constraints. A position constraint can be established by forcing nodes to remain within a certain region, while other types of constraints can be used if they can be expressed with forces. Examples of this include the use of magnetic fields to impose orientation constraints (SUGIYAMA; MISUE, 1994, 1995) (seen in Figure 2.2) and the utilization of dummy nodes to force groupings.

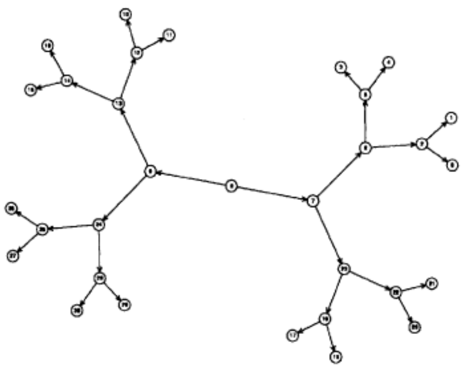


Figure 2.2: Sugiyama and Misue’s magnetic-spring layout (1994; 1995).

For many years, force-directed algorithms have suffered dramatically from a scalability problem: the more nodes and edges we have, the slower it is for the system to converge. Thus, it was only possible to use these algorithms in real-time with smaller graphs due to their high computational cost. However, with the advent of faster, multi-core processors and powerful, programmable graphic processing units (GPUs) this reality is changing fast. Mass-spring algorithms can now deal with hundreds of thousands nodes

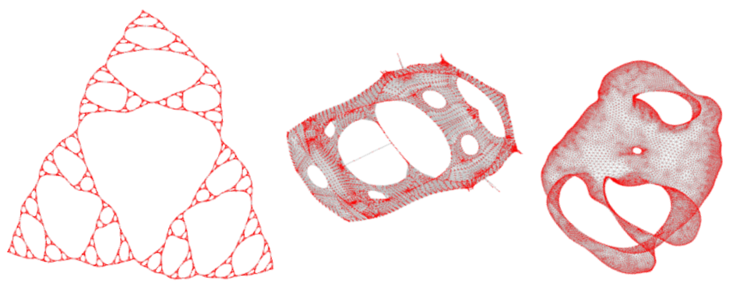


Figure 2.3: Some graph drawings generated by Frishman and Tal's algorithm (2007).

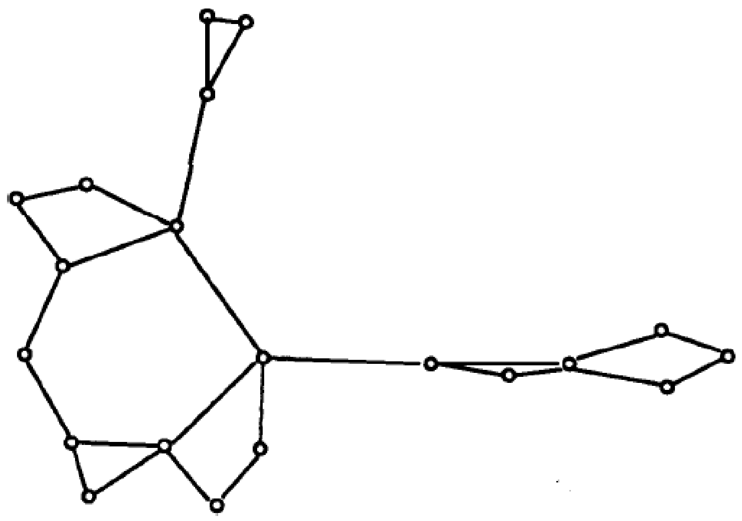


Figure 2.4: Sample layout generated using the Eades spring-embedder technique (1984).

and edges in real-time, and many different applications, such as real-time cloth simulation, are already making use of that (GEORGII; ECHTLER; WESTERMANN, 2005; TEJADA; ERTL, 2005). Recent works on GPU-based force-directed layout include a multi-level graph layout algorithm proposed by Frishman and Tal (2007), seen in Figure 2.3.

Aside from the scalability problem, force-directed algorithms also suffer greatly from a predictability problem. Two different runs of an algorithm over similar (or even the same) input graphs might generate two completely different layouts, which is not very helpful in allowing the user to create and maintain a mental map of the visualization. One approach that has been used to minimize this is to run another layout algorithm first, and afterwards execute the force-directed technique on top of that.

The first force-directed algorithm was proposed by Eades (1984), producing visual results as shown in Figure 2.4. It takes the intuitive approach of treating the graph as a mass-spring system, with nodes being steel rings and edges springs that connect them. Despite the physical metaphor, it does not aim for physical accuracy, not employing Hooke's law for the springs and with forces affecting velocity instead of acceleration. It produces visualizations of uniform edge length and allows representation of graph symmetry. In this technique, nodes act as physical bodies and edges as springs, with nodes exerting a repulsive force and a drag force or some amount of friction being applied to the model.

This algorithm results in an optimization process, since it minimizes the energy within the physical system. This technique is very simple to implement and also generically applicable. Bertault (2000) managed to make a more deterministic version of this technique by preserving edge crossings.

Many other algorithms extended the basic idea presented by Eades. One of those is Kamada and Kawai's (1989), which aims at positioning nodes in a way that their geometric distance is equal to their graph-theoretic distance. It does so by having the simulation assume that between every two nodes there is a spring with rest length equal to the theoretical distance between them.

Another interesting algorithm is Davidson and Harel's (1996), which introduced the idea of using simulated annealing to minimize the system's energy function, which takes into account vertex distribution, edge length and edge crossings. This algorithm, seen in Figure 2.5, can be very time consuming, but can produce better results.

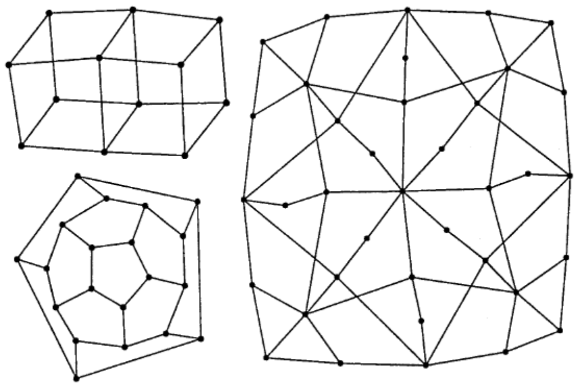


Figure 2.5: Layouts computed with Davidson and Harel's technique (1996)

One of the most important force-directed algorithms is the one proposed by Fruchterman and Reingold (1991). This algorithm consists on calculating on each iteration all the forces that attract and repulse nodes. Nodes connected by edges exert an attraction force between them, while all nodes exert a repulsion force on all others. From the forces, the position displacement a node will suffer during each iteration is calculated and limited by the current value of an attribute (usually used as temperature), which is progressively decreased. This algorithm's straightforward implementation, relatively fast running speed and high flexibility when it comes to the insertion of constraints and other aesthetic criteria make it probably the most popular force-directed layout technique. Its major drawbacks are its high complexity of $O(n^3)$ and its unpredictability. These can be dealt with, though, and numerous improved versions of the algorithm have been proposed. One of these approaches was used in this work and will be explained in detail in the next section. Graph layouts generated with this technique can be seen in figures 2.6 and 2.7 while the algorithm itself can be seen in Algorithm 1.

Another interesting approach is Noack's LinLog energy model (2003), exemplified in Figure 2.8, which attempts to reveal clusters of highly connected nodes. This technique is particularly useful for datasets such as social networks, and was proposed in two variations, the node-repulsion LinLog model (NOACK, 2003, 2004) and the edge-repulsion LinLog model (NOACK, 2005). Both variations produce similar drawings, but the latter avoids dense accumulations of nodes with high degrees for graphs with non-uniform degrees.

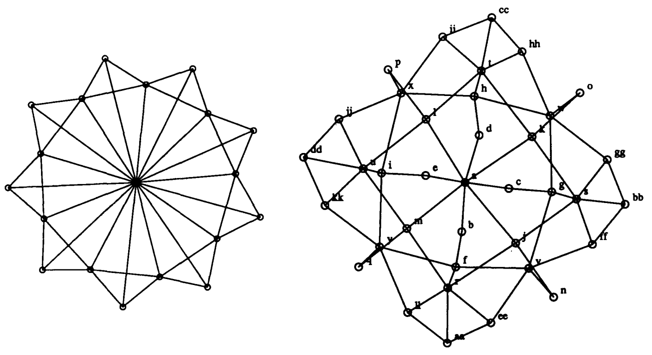


Figure 2.6: Fruchterman-Reingold layout (1991).

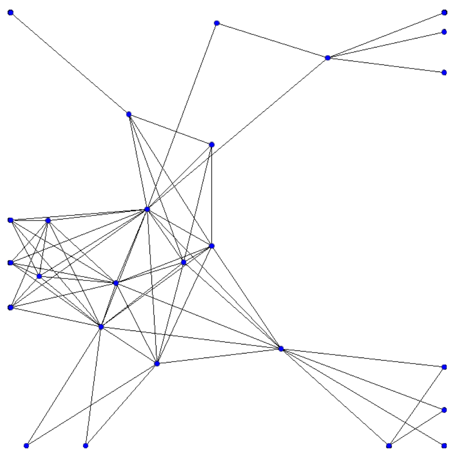
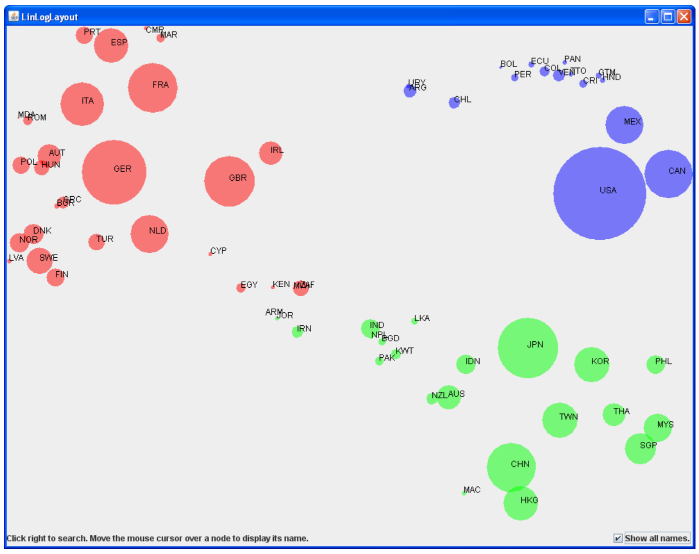


Figure 2.7: Fruchterman-Reingold layout (1991) of the largest component of the co-authorship graph of the AVI conference (26 nodes and 78 edges).



Click right to search. Move the mouse cursor over a node to display its name.

Figure 2.8: Noack's LinLog algorithm (2003; 2004). Image taken from the open source implementation provided by Noack in his website (http://www.informatik.tu-cottbus.de/ an/GD)

Algorithm 1 Fruchterman-Reingold layout algorithm.

```
Require: $C$ {Constant for computation of the optimal distance between nodes.}
Require: $W$ {Width of the workspace}
Require: $H$ {Height of the workspace}
Require: $nodes$ {List of nodes, which are in random positions},
Require: $edges$ {List of edges}
Require: $nIterations$ {Number of iterations to be performed}
 1:
 2: {Calculate optimal distance between nodes}
 3: $k \leftarrow C * \sqrt{(W * H)/nodes.count()}$
 4:
 5: function $attractiveForceStrength(x) \leftarrow x^2 / k$
 6: function $repulsiveForceStrength(x) \leftarrow k^2 / x$
 7:
 8: for $i$ = 1 to $nIterations$ do
 9:    {Calculate attractive forces.}
10:    for each $e$ in $edges$ do
11:       $\Delta \leftarrow e.n$1.pos() - $e.n$2.pos()
12:       $e.n1.force -= ( \Delta / \mid \Delta \mid ) * attractiveForceStrength(\mid \Delta \mid)$
13:       $e.n2.force += ( \Delta / \mid \Delta \mid ) * attractiveForceStrength(\mid \Delta \mid)$
14:    end for
15:
16:    {Calculate repulsive forces.}
17:    for each $n$ in $nodes$ do
18:       for each $m$ in $nodes$ do
19:          if $n \neq m$ then
20:             $\Delta \leftarrow n$.pos() - $m$.pos()
21:             $n.force += ( \Delta / \mid \Delta \mid ) * repulsiveForceStrength(\mid \Delta \mid)$
22:          end if
23:       end for
24:       {Limit maximum displacement to the temperature $t$}
25:       {and then prevent node from leaving the workspace.}
26:       $n.pos += (n.force / \mid n.force \mid) * min(n.force, t)$
27:       $n.pos.x \leftarrow min( W/2, max(-W/2, n.pos.x))$
28:       $n.pos.y \leftarrow min( H/2, max(-H/2, n.pos.y))$
29:    end for
30:    $t \leftarrow cool(t)$ {Reduce temperature.}
31: end for
```

Other force-directed approaches include Frick et al.’s GEM (1995), Gajer and Kouborov’s Grip (2000; 2004), and more recently the DiG-CoLa technique by Dwyer and Koren (2005), which allows the application of force-directed algorithms to directed graphs. Examples of layouts generated with these techniques can be seen in Figures 2.9, 2.10 and 2.11, respectively.

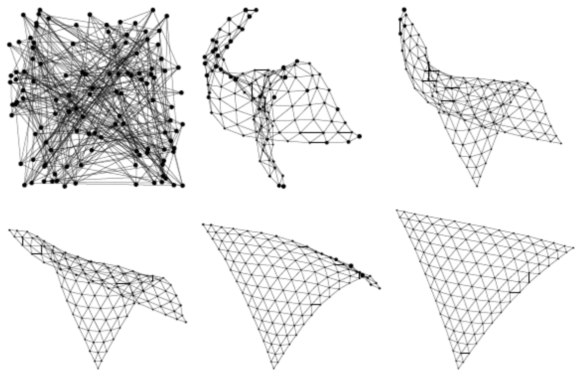


Figure 2.9: Evolution of a layout computed with Frick et al.’s technique (1995)

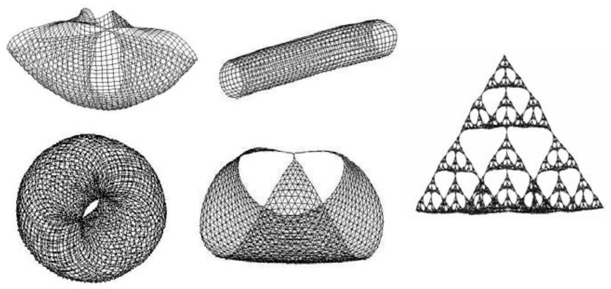


Figure 2.10: Graph layouts computed with Grip (GAJER; KOBOUROV, 2000)

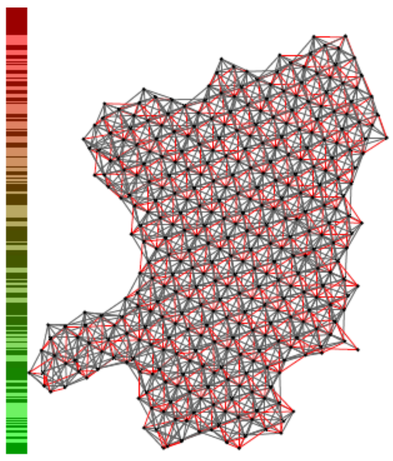


Figure 2.11: Layouts computed with Dwyer and Koren’s DiG-CoLa technique (2005)

2.4 Combining Fruchterman-Reingold's Algorithm With the Barnes and Hut Technique

One approach to reduce the complexity of Fruchterman and Reingold's technique is to combine it with Barnes and Hut's n-body simulation algorithm (1986), as done by Tunkelang (1999). With this, one can reduce the original algorithm's complexity from O(n^3) to O(n^2), making it substantially more scalable.

The n-body problem consists of computing the movement of a given number of bodies that are mutually affected by gravitational forces. The brute force approach for solving this problem consists of going through every body and computing how all other bodies affect it. This is clearly not very scalable, having a complexity of O(n^2). Thus, the more bodies we have, the slower the algorithm runs.

The Barnes-Hut algorithm is able to speed up the computation to O($n \log n$) by grouping together nearby bodies and approximating them as a single body. If a group of bodies is sufficiently far away, their total gravitational effect on a certain body can be approximated with the group's center of mass. This can be applied recursively, with each group of bodies being comprised of smaller groups and the last level of groups containing just one body each.

The data structures that naturally fit this sort of space partitioning problem are the *quadtrees* for 2D simulations, and the *octrees* for 3D (SAMET, 1990). In the case of a 2D n-body simulation, the Barnes-Hut algorithm consists of building a quadtree of bodies and using it to compute how each body will be affected by all the others. The leaves of this quadtree will contain each just one body, while inner nodes will contain quadrants that represent groups of bodies, storing their total mass and center of mass. If the affected body is sufficiently far away from a group of bodies, the group's total mass and center of mass is used to calculate the gravitation force instead of each body's individual location and mass. If the body is close to other bodies, it will naturally be affected by smaller groups, which might eventually represent each body individually depending on their distance. The quadtree can be built by inserting into it all the bodies with algorithm 2, while the repulsive force exerted on a body can be computed using algorithm 3. For 3D the algorithm works analogously, but substituting the quadtree by an octree.

The combination of Fruchterman and Reingold's algorithm with Barnes and Hut is show in Algorithm 4. Every node of the graph becomes a physical body with a mass and at each iteration of the algorithm the tree is rebuilt (since the nodes will have moved a little), and subsequently used to compute the repulsion force that is applied on each node by all the others.

2.5 Final Comments

Even though the proposed technique can work with any layout algorithm that supports the application of forces on nodes, it was decided to use a variation of the Fruchteman-Reingold algorithm (1991). It is one of the most well-known force-directed algorithms, producing organic-looking, clean, layouts that minimize edge crossings, and is also very straightforward to implement. The visual results it produces were thought to be very reasonable for the technique's proof-of-concept. This, along with its relative simplicity and widespread use, as well as the fact that it lends itself very easily to the technique, made this the algorithm of choice for the prototype developed in this work. An approach similar to Tunkelang's (1999) was taken.

While this chapter provided an overview of what is necessary to properly understand the proposed technique and reproduce the implementation of the prototype as done for this work, the following chapter describes other works that are not exactly necessary for the comprehension of this one but that help put it into context by providing an overview of the field of graph visualization.

Algorithm 2 Method for insertion of a body in the Barnes-Hut quadtree.

```
Require: $body$ {Body to be inserted.}
Require: $bhTreeNode$ {Node of Barnes-Hut quadtree where $body$ should be inserted}

if $bhTreeNode$ is not a leaf then
    Recursively insert $body$ into the quadrant of $bhTreeNode$ that contains it.
    {Note that quadrants are also Barnes-Hut quadtrees}

    {Compute $bhTreeNode$'s new total mass and center of mass.}
    for each $q$ in $bhTreeNode$.quadrants() do
        $bhTreeNode.mass$ $+=$ $q$.mass()
        $bhTreeNode.centerMass$ $+=$ $q$.centerMass() $*$ $q$.mass()
    end for
    $bhTreeNode.centerMass$ $/=$ $bhTreeNode$.mass()

else

    if leaf is empty then
        $bhTreeNode.body \leftarrow body$
        $bhTreeNode.mass \leftarrow body$.mass()
        $bhTreeNode.centerMass \leftarrow body$.pos()
    else
        Subdivide $bhTreeNode$ by creating new quadrants in it
        Move $bhTreeNode$'s body to the quadrant that contains it.
        Recursively insert $body$ in the subquadrant that contains it.
        Recalculate $bhTreeNode$'s mass and center of mass as in lines 7-11.
    end if

end if
```

Algorithm 3 Using the Barnes-Hut technique to calculate the repulsive forces exerted on a given body by all the others (*bhGetRepulsiveForce( )*).

```
Require: $body$ {Target body}
Require: $bhTreeNode$ {Barnes-Hut quadtree node. Initially this should be the root}
Require: $\theta$ {User-supplied threshold. Usually a little less than 1}
Require: $G$ {Gravitation constant},
 1:
 2: if $bhTreeNode$ is leaf then
 3:     $\Delta = |\ body.\text{pos}() - bhTreeNode.body.\text{pos}()\ |$
 4:     $force = G * body.\text{mass}() * bhTreeNode.body.\text{mass}()\ /\ \Delta^2$
 5: else
 6:     $\Delta = |\ body.\text{pos}() - bhTreeNode.\text{centerOfMass}()\ |$
 7:     if $bhTreeNode.\text{quadrantLength}()\ / \Delta < \theta$ then
 8:         $force = G * body.\text{mass}() * bhTreeNode.\text{mass}()\ /\ \Delta^2$
 9:     else
10:         {Recursively sum the forces of the four subquadrants, which are also tree nodes}
11:         for each $q$ in $bhTreeNode.\text{quadrants}()$ do
12:             $force \mathrel{+}= bhGetRepulsiveForce(\ body, q\ )$
13:         end for
14:     end if
15: end if
16: return $force$
```

Algorithm 4 Combined Fruchterman-Reingold / Barnes-Hut layout algorithm.

```
Require: $C$ {Constant for computation of the optimal distance between nodes.}
Require: $W$ {Width of the workspace}
Require: $H$ {Height of the workspace}
Require: $nodes$ {List of nodes, which are in random positions},
Require: $edges$ {List of edges}
Require: $nIterations$ {Number of iterations to be performed}

{Calculate optimal distance between nodes}
$k \leftarrow C * \sqrt{(W * H)/nodes.count()}$

function $attractiveForceStrength(x) \leftarrow x^2/\ k$
function $repulsiveForceStrength(x) \leftarrow k^2/\ x$

for $i$ = 1 to $nIterations$ do
    {Calculate attractive forces.}
    for each $e$ in $edges$ do
        $\Delta \leftarrow e.n1$.pos() - $e.n2$.pos()
        $e.n1.force -= (\ \Delta\ /\ \mid \Delta \mid\ ) * e.n1$.mass() $* attractiveForceStrength(\mid \Delta \mid)$
        $e.n2.force += (\ \Delta\ /\ \mid \Delta \mid\ ) * e.n2$.mass() $* attractiveForceStrength(\mid \Delta \mid)$
    end for

    Build $barnesHutTree$. All of the graph's nodes are inserted as the tree's bodies.
    {see algorithm 2 for insertion of a body into a Barnes-Hut quadtree}

    {Calculate repulsive forces.}
    for each $n$ in $nodes$ do
        $bhGetRepulsiveForce(node, barnesHutTree)$
        {Note that $repulsiveForceStrength()$ is used as $bhGetRepulsiveForce()$'s grav-
        ity function, with the current $\mid \Delta \mid$ passed as its parameter (see algorithm 3)}

        {Limit maximum displacement to the temperature $t$}
        {and then prevent node from leaving the workspace.}
        $n.pos += (n.force/\mid n.force\mid) * min(n.force, t)$
        $n.pos.x \leftarrow min(\ W/2, max(-W/2, n.pos.x))$
        $n.pos.y \leftarrow min(\ H/2, max(-H/2, n.pos.y))$
    end for
    $t \leftarrow cool(t)$ {Reduce temperature.}
end for
```

3 RELATED WORK

This chapter describes work related to the fields of graph visualization and human-computer interaction. The works here presented are not necessary for the implementation and comprehension of MagnetViz, but are nonetheless essential to put it into context.

3.1 Graph Visualization

3.1.1 Node-Link Diagrams

The most intuitive (and therefore the most commonly used) approach for visually representing a graph is the node-link diagram. It consists of visual objects that represent the nodes and lines that connect these objects, representing the edges. If the graph is directed, the lines are drawn as arrows that go from the origin node to the target. An example of a graph drawn as a node-link diagram can be seen in Figure 3.1.

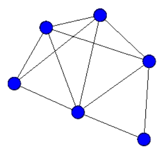
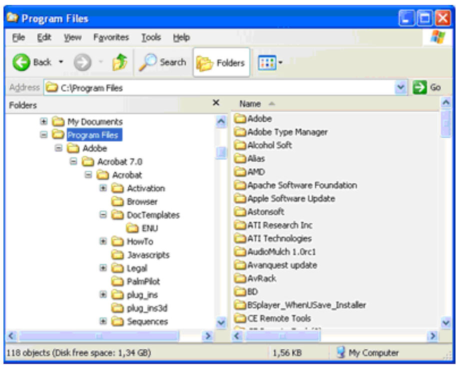


Figure 3.1: Example of a node-link diagram

Node-link diagrams of graphs are drawn using layout algorithms. These algorithms attempt to calculate geometric positions for the nodes in ways that are aesthetically pleasing and that convey useful information to the viewer. There are many different types of layout algorithms, with the force-directed approach that was discussed in Chapter 2 being one of the most widely used. A good source of material about graph visualization is the book written by DiBattista et al. (1999), which covers the most important work of the field.

Many graph layout algorithms are focused on the drawing of trees, which are a special case of graphs. Amongst these, one of the most commonly found is the indentation layout, seen in Figure 3.2, which is typically used to represent file structures. Also well known is the Reingold-Tilford layout (REINGOLD; TILFORD, 1981) (seen in Figure 3.3), which was designed for binary trees and later extended for trees of any degree by Walker (1990). Shiloach's H-tree layout (1976) for binary trees is another prominent algorithm, being later adapted by Eades into the radial layout (1992) for general graphs (seen in Figure 3.4). Another important technique is Carrière et al.'s Balloon Trees layout (1995) (seen in Figure 3.5), which is a rare adaptation of a 3D algorithm (namely, Robertson et al.'s classic 3D Cone Trees algorithm (1991)).



C:\Program Files

```
My Documents
Program Files
  Adobe
    Acrobat 7.0
      Acrobat
        Activation
        Browser
        DocTemplates
          ENU
        HowTo
        Javascripts
        Legal
        PalmPilot
        plug_ins
        plug_ins3d
        Sequences
```

```
Adobe
Adobe Type Manager
Alcohol Soft
Alias
AMD
Apache Software Foundation
Apple Software Update
Astonsoft
ATI Research Inc
ATI Technologies
AudioMulch 1.0rc1
Avanquest update
AvRack
BD
BSplayer_WhenUSave_Installer
CE Remote Tools
```

118 objects (Disk free space: 1,34 GB) 1,56 KB My Computer

Figure 3.2: Indentation layout.

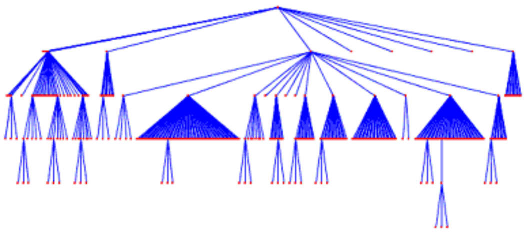


Figure 3.3: The Reingold-Tilford layout (REINGOLD; TILFORD, 1981).

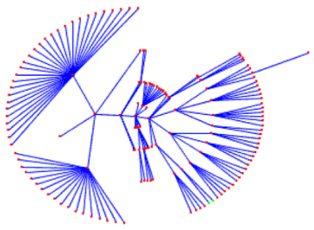


Figure 3.4: The radial layout (EADES, 1992).

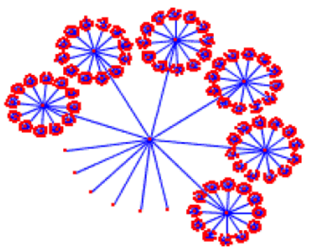


Figure 3.5: The Balloon Trees layout (CARRIERE; KAZMAN, 1995).

Due to the effects of interacting with it, perhaps one of the most interesting tree visualizations is the hyperbolic layout (LAMPING; RAO; PIROLLI, 1995; LAMPING; RAO, 1996), which builds a node-link diagram on a hyperplane and projects the result back into Euclidean geometry. The visual results have a focus+context effect similar to Furnas's Fisheye technique (1981), with nodes far from the center shown in progressively smaller size and lines being projected as curves. An example of a tree drawn with a hyperbolic layout can be seen in Figure 3.6. For more details about focus+context techniques, see Section 3.4.

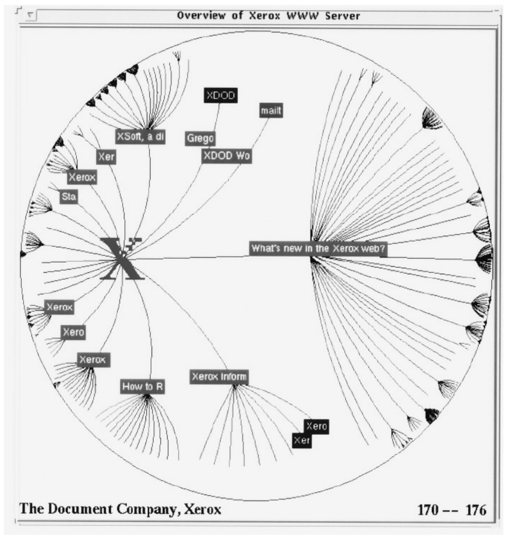


Figure 3.6: Example of a 2D hyperbolic layout (LAMPING; RAO, 1996).

Regarding general graphs, there are several approaches that can be taken. Since tree-drawing algorithms usually have a reasonably low complexity and are relatively easy to implement, one of the most popular approaches is simply drawing the spanning tree of the graph using one of the previously mentioned tree layout algorithms. Another approach is hierarchical layouts (also known as upward layouts), in which most edges start from a source node in a directed graph and flow towards a sink. These layouts are particularly useful for displaying hierarchies, organization charts, scheduling and logistic diagrams, etc.. One of the fundamental techniques of this type is Sugiyama et al.'s "dot" algorithm (1981), in which nodes are assigned to hierarchically organized layers according to a chosen layering technique. Once that is done, nodes are positioned within their layers in a way that minimizes the edge-crossings between two consecutive layers. Figure 3.7 illustrates this algorithm. Techniques that build on this approach include the works by Gansner et al. (1988; 1993).

Some techniques are also designed specifically for graphs, such as the optimization/force-directed, orthogonal, spectral, attribute-based and feature-based layouts. Covered in more detail in Chapter 2, optimization techniques use physics-based algorithms to create organic, non-deterministic layouts.

Orthogonal layouts attempt to reduce the number of edge crossings and the area occupied by the graph drawing by always laying the edges horizontally or vertically, being thus quite useful for VLSI circuit layout design. Important orthogonal layout algorithms include the ones by Tamassia and Tollis (1989) and Biedl and Kant (1994). Other techniques can be found in the survey written by Eiglsperger et al. (2001). An example of a

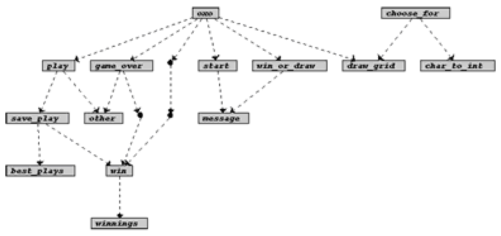


Figure 3.7: Graph drawn with the "dot" layout (SUGIYAMA; TAGAWA; TODA, 1981).

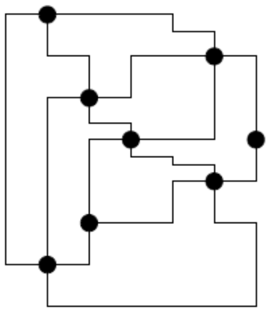


Figure 3.8: Graph drawn with Tamassia and Tollis's orthogonal layout algorithm (1989).

graph drawn with an orthogonal layout can be seen in Figure 3.8.

Spectral layout algorithms employ the eigenvectors of the Laplacian matrix for generating layouts for undirected graphs. These techniques are not widely used, since they do have a very intuitive aesthetic interpretation. However, they provide an exact solution to the layout problem (other approaches generally result in an NP-hard problem that can only be approximated) and can be computed extremely fast. The basic method of spectral graph drawing was described in Godsil and Royle's book (2001), with the algorithms by Pisanski and Shawe-Taylor (1998) and Koren (2003) being important works on the subject. More information on this topic can be found in Ross's survey (2004) and Puppe's book (2008). Examples of graph layouts generated with spectral algorithms can be seen in Figure 3.9.

Attribute-based algorithms take into consideration the information stored in the nodes and edges for the computation of the layout, mostly discarting the topological structure of the graph. Some works that take this approach include Pretorius and van Wijk's two

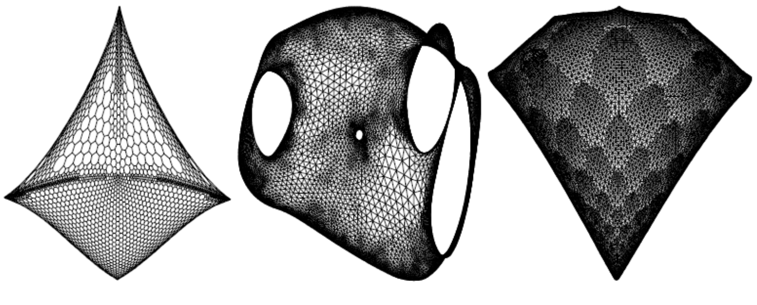


Figure 3.9: Graphs drawn with a spectral layout algorithm (KOREN, 2003).

papers on the visualization of transition graphs (2005; 2006), Wattenberg's PivotGraph (2006) and Aris and Shneiderman's NVSS (Network Visualization by Semantic Substrates) (2006). Some works, such as Archambault et al.'s GrouseFlocks (2008), take a middle-ground approach between topology and attribute-based techniques. GrouseFlocks, seen in Figure 3.10 is a system for the interactive visualizations of a graph's hierarchy space that provides users with operations that help them create and modify their graph hierarchies based on selections, which can be made manually or based on patterns in the attribute data.

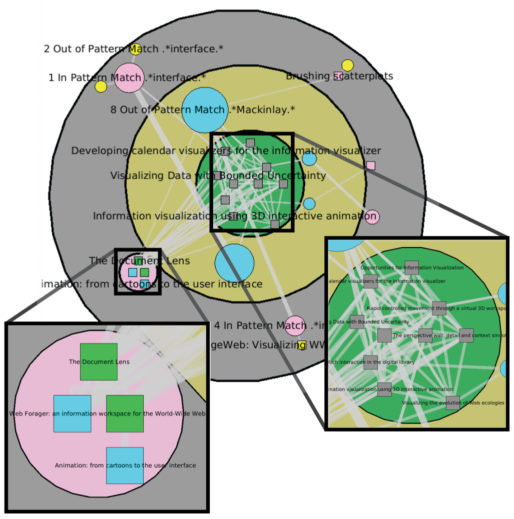

2 Out of Pattern Match .*interface.*
1 In Pattern Match .*interface.*
Brushing Scatterplots
8 Out of Pattern Match .*Mackinlay.*
Developing calendar visualizers for the information visualizer
Visualizing Data with Bounded Uncertainty
Information visualization using 3D interactive animation
The Document Lens
Animation: from cartoons to the user interface
4 In Pattern Match
Web Forager: an information workspace for the World-Wide Web

Figure 3.10: GrouseFlocks (ARCHAMBAULT; MUNZNER; AUBER, 2008).

Feature-based techniques detect some desired trait the graph may have and use it to draw the layout. In this class of algorithms, we find works such as Archambault et al.'s TopoLayout (2007), which is a multi-level algorithm that recursively detects topological features of the graph, such as trees, connected and biconnected components, and uses this information to build a hierarchy. For each topological feature found, an appropriate algorithm is used to compute the layout. TopoLayout can be seen in Figure 3.11.

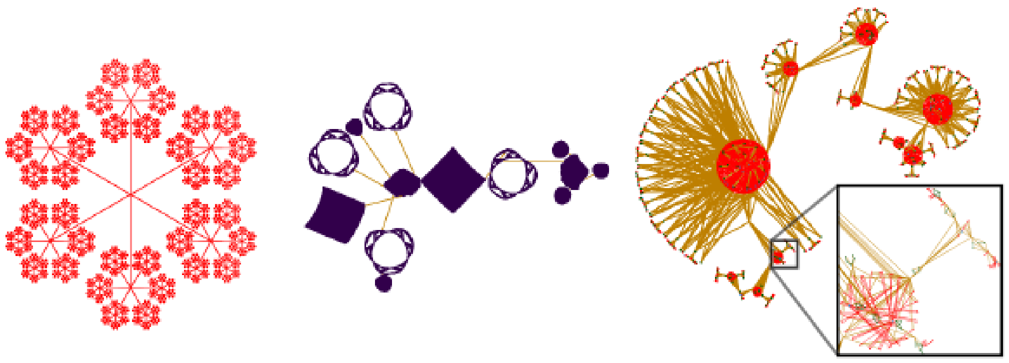


Figure 3.11: Graph layouts generated with Archambault et al.'s TopoLayout (2007).

3.1.2 Alternative Approaches for Large Graphs

A common problem found in most 2D graph layouts is that a large amount of nodes and edges tends to create cluttered visualizations that occupy a large amount of screen space, being thus potentially too big and/or dense to be comprehensible to users. Some authors attempt to solve this problem by creating layout algorithms geared at large/dense graphs, such as the works of van Ham and Wattenberg (2008). Another approach is the use of interaction and navigation techniques, such as selection, picking, drag-and-dropping, zooming, panning, etc., to explore the drawing of a graph. Other approaches include trying alternative visual representations, coordination of different visualizations, 3D visualizations (which can be completely novel or extended from 2D), and new hybrid techniques built from combinations of others.

One example of alternative approach is Shneiderman's very popular Treemaps (1992), which uses nested rectangles to communicate the hierarchy of a tree, easily providing the user with an overview of an entire large tree. This technique is one of the most well known of the field of Information Visualization, having been used in many applications, ranging from visualization of file systems (SHNEIDERMAN, 1992) to stock market data (JUNGMEISTER; TURO, 1992; WATTENBERG, 1999). A directory structure viewed with Treemaps can be seen in Figure 3.12. The image was generated using the demo available in the Treemaps website (http://www.cs.umd.edu/hcil/treemap).

Amongst the techniques that take the approach of using a different visual representation, some of the most prominent are the ones that visualize a graph's adjacency matrix. This approach consists of showing the graph as a table with the nodes as its rows and columns and the edges as filled table cells. Matrix visualizations are not as intuitive as node-link diagrams, but they have proven themselves very promising, since they eliminate all occlusion and scalability problems that are often found in node-link diagrams. When the nodes are ordered following some criteria, the visualization as an adjacency matrix allows a rapid perception of the relationships between nodes, since there are no edge crossings. Ghoniem et al. (2004) wrote an interesting comparison of matrix and

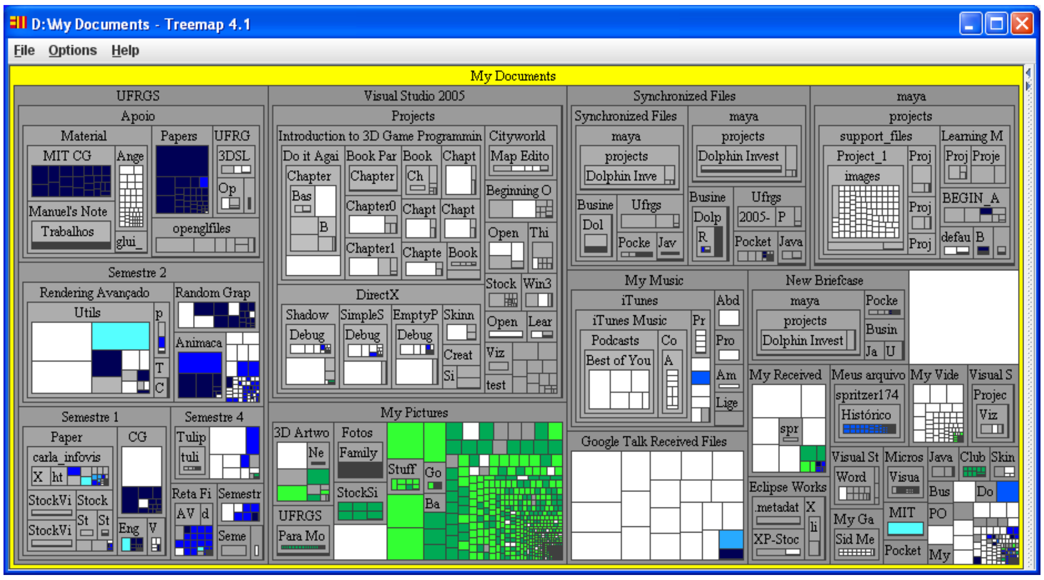


Figure 3.12: A directory structure viewed with Treemaps (SHNEIDERMAN, 1992) using the demo provided in the Treemaps website.

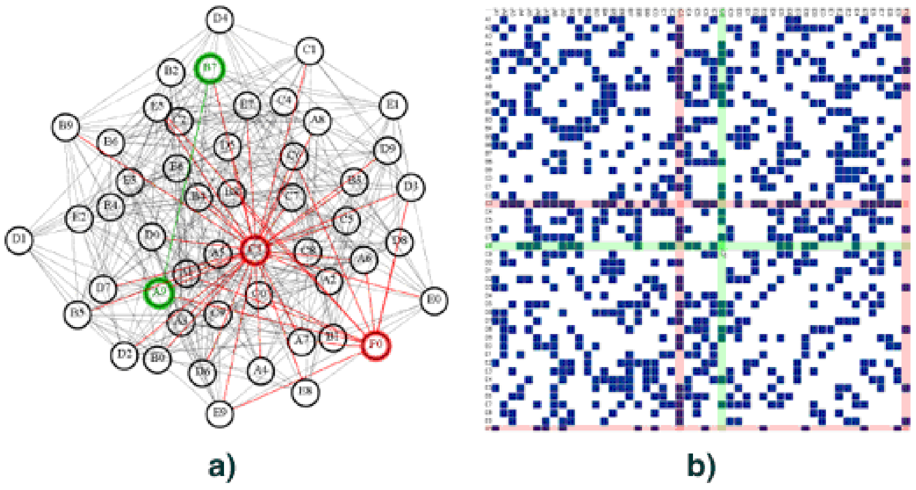


Figure 3.13: Graph visualized as both a node-link diagram and a matrix (GHONIEM; FEKETE; CASTAGLIOLA, 2004).

node-link representations. Applications that use matrix visualization include Henry and Fekete's Matrix Explorer (2006), MatLink (2007) and NodeTrix (2007), the system developed by Abello and Korn for visualizing multidigraphs (2002), Abello and van Ham's Matrix Zoom (2004) and Ghoniem et al.'s VISEXP (2004). Figure 3.13 shows a graph visualized both as a node-link diagram and a matrix.

Each visualization technique has its own different advantages and drawbacks. Therefore, one sometimes taken approach is to use two or more different techniques in a coordinated manner so that one's disadvantages is counterbalanced by the the other's advantages. An example of technique that does this is Henry and Fekete's MatrixExplorer (2006), which uses matrix visualization in conjunction to node-link diagrams in order to help the user with filtering and searching, while also providing the traditional and intuitive view of the graph.

Adjacency matrix visualization and its coordination with other techniques is one of

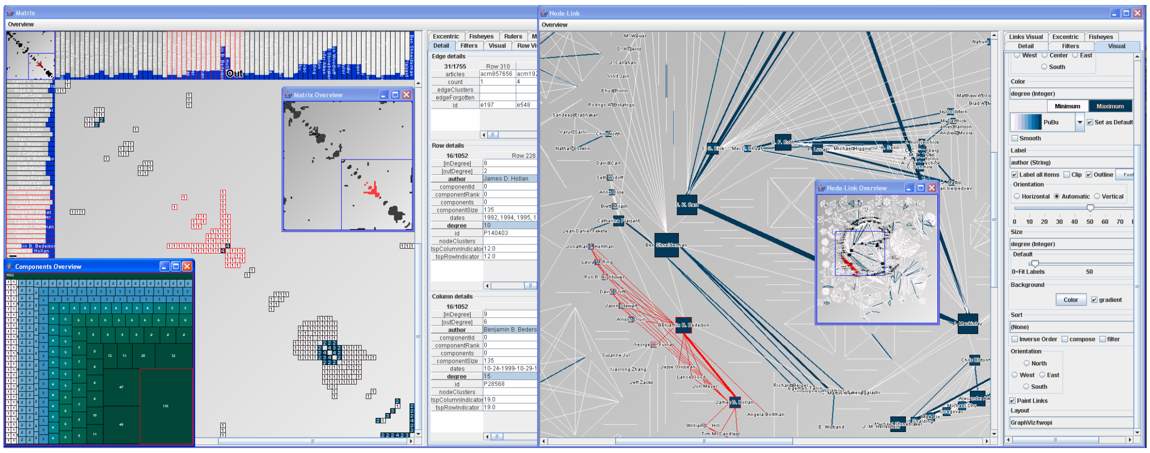


Figure 3.14: MatrixExplorer showing two synchronized representations of the same network: matrix on the left and node-link on the right (HENRY; FEKETE, 2006).

the possible approaches to visualize graphs that might be too big for regular drawing and interaction techniques. Some have also tried what are known as hybrid techniques. These consist of two or more techniques that are effectively combined into new methods, with one's technique's advantages compensating for the other's drawbacks (working similarly to coordinated techniques, albeit in an integrated manner). Amongst these are found Balzer et al.'s Voronoi Treemaps (2005), Zhao et al.'s Elastic Hierarchies (2005), Dwyer et al.'s DiG-CoLa (2005) and Henry et al.'s MatLink (2007) and NodeTrix (2007). These can be seen in Figures 3.15, 3.16, 2.11 and 3.18, respectively.

Other ways of dealing with the problems of large graph visualization is the use of

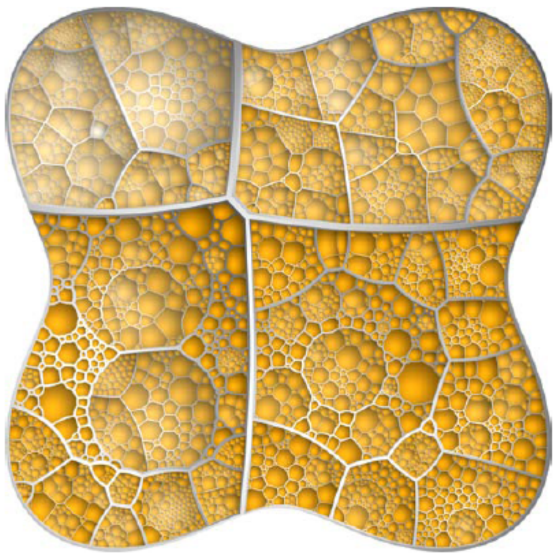


Figure 3.15: Voronoi Treemaps (BALZER; DEUSSEN; LEWERENTZ, 2005).

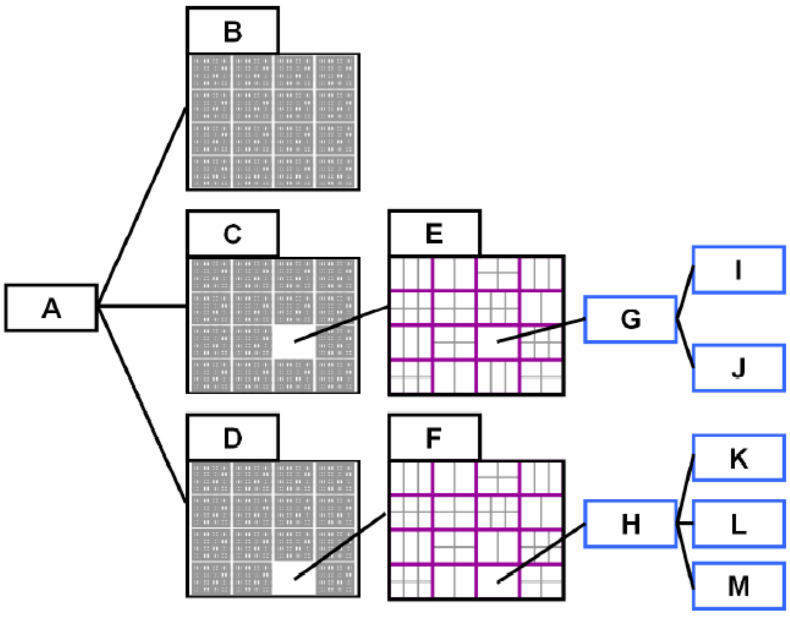


Figure 3.16: Elastic Hierarchies (ZHAO; MCGUFFIN; CHIGNELL, 2005).

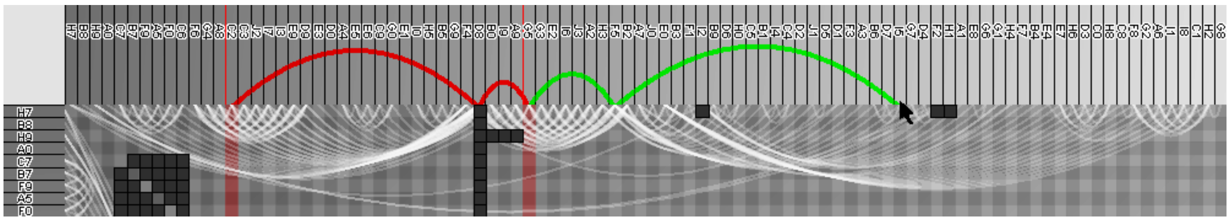


Figure 3.17: MatLink (HENRY; FEKETE, 2007).

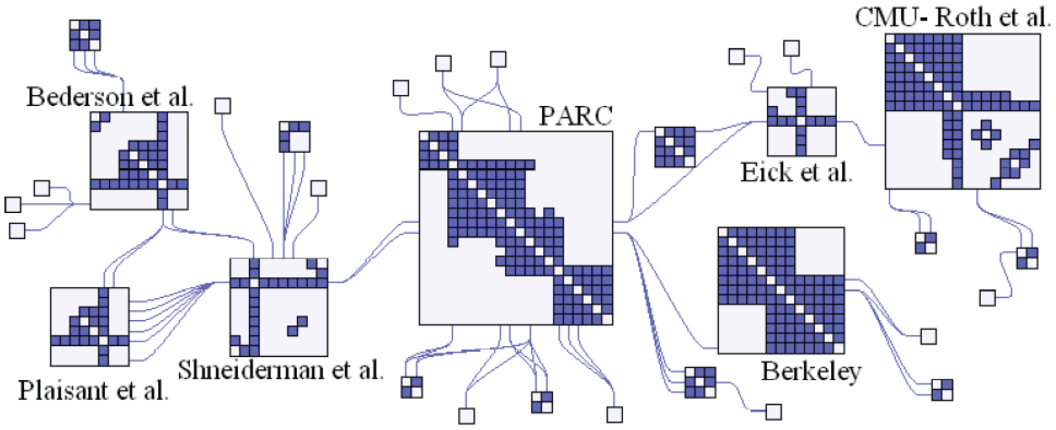


Figure 3.18: Largest component of the InfoVis co-authorship network visualized with NodeTrix (HENRY; FEKETE; MCGUFFIN, 2007).

special navigation and interaction schemes (see Section 3.2) and clustering.

Clustering is the grouping together of related nodes in order to reduce screen clutter. It is usually of two types: structure-based or content-based. Structure-based clustering utilizes the graph structure to group elements, while content-based clustering uses the semantics of the graph (HERMAN et al., 2000). A structure-based clustering technique can be more generally applied since it does not depend on application-specific information, while a content-based technique can generate the best results for its application (possibly even being used in combination with structure-based clustering) but might not be general enough to use in other contexts.

Clustering works by allowing the use of a single on-screen element to represent several, making possible, for example, the use of different levels of detail that can be accessed by zooming in and out of the graph. Clustering also helps the user identify desired relationships that otherwise could not be seen, being frequently employed to help with filtering and searching. Node clustering techniques can be either an inherent part of the visualization, such as in Henry et al.'s MatrixExplorer (2006), MatLink (2007) and NodeTrix (2007) and ASK-GraphView (2006) or independent methods that can be used with another technique. It is important to note that clustering is not exclusive to nodes, with Cui et al.'s work (2008) being an example of edge clustering.

Occasionally it might happen that ideally a node would have to belong to two clusters to better suit the graph (i.e. the case of an author that contributes to two or more different groups of authors). One approach that can be used to deal with this situation is the duplication of nodes, as proposed by Henry et al. (2008).

To deal with the limitations of 2D visualizations, another approach that is occasionally taken is to use 3D. Some 3D visualizations are just natural extensions of 2D counterparts, while other are created completely from scratch. Amongst the first group, are Patrignani et al.'s 3DCube (1998), Rekimoto et al.'s Information Cube (1993), Hendley et al.'s Narcissus (1995) and Parker's NV3D (1998). Amongst the latter are Robertson et al.'s Cone Trees (1991), Munzner's H3 (1997), the SGI File System Navigator (????) and Hyun's Walrus (2008).

The 3D techniques often inherently solve many of the problems faced by 2D methods, but they come with many of their own. While 2D techniques can seize on a large amount of knowledge about 2D interfaces and interaction on well-defined standards, for the 3D case development is largely experimental, counting on just a few guidelines and recommendations that depending on the case might or not be applicable.

3.2 Interaction in Graph Visualization

In Information Visualization, interaction and navigation are as important as the visualization of the data itself, especially when we are dealing with (possibly large) graphs. Finding desired information on a graph is often difficult without the proper navigation and interaction tools.

Techniques for interaction and navigation in 2D graph visualizations have evolved from generally applicable human-computer interaction (HCI) methods, being thus able to rely on all the well known HCI standards. Many different techniques have already been developed, sometimes being an integral part of the visualization technique itself, supporting all users' tasks.

It is important to note that both navigation and interaction present a related, but very different problem from graph drawing, with most researchers focusing either in one or the other. In order to produce an efficient visualization, the two areas should be combined, since they are quite dependent on one another to work properly.

Some of the most important categories of interaction techniques are filtering, search, selection, scrolling/panning and scaling/zooming. Recently, with the surge of interest in visualizing large and complex graphs incremental exploration and navigation has also become essential for many applications. See Herman et al. (2000), for a wider review on navigation and interaction techniques for graph visualizations.

Filtering allows the user to define a set of criteria that will be used by the visualization system to determine what will be displayed on the screen, after filtering the information to a subset that is of interest to the user. Filtering allows for faster finding of desired information by the user due to less visual complexity, being thus very useful to deal with problems generated by visual clutter and occlusion that tend to happen with large datasets. The criteria chosen by the user for the filtering operation might be anything from the graph's structure to a particular attribute of the nodes. North et al.'s (1994), Gansner et al.'s (2000) and Jia et al.'s (2008) works describe the use of filtering in graph visualization applications, with the latter showing how edge filtering can be used to improve visualizations of large graphs (namely social networks).

Searching allows the user to easily find information on the visual representation of the graph by providing some criteria as input to the search mechanism, which usually returns the node that best matches the search parameters by visually focusing on it somehow. The Cone Trees (ROBERTSON; MACKINLAY; CARD, 1991) and Hyperbolic Browser (LAMPING; RAO; PIROLLI, 1995) techniques are some of the first that implemented this feature that is nowadays common to all visualization systems.

Selection provides the user with a mechanism to select certain information from the dataset. When an element is selected, it is usually visually highlighted from the rest of the dataset, with more information often being displayed about it. Selection is one of the most common interaction techniques and is closely related to search mechanisms (which may return a certain element selected). Selection also serves as basis for tools that allow moving and manipulating elements.

When the drawing of the graph occupies more than the size of the screen, it is often useful to apply the classical zoom and pan tools. Zooming allows a visualization to be viewed in more detail and has two distinct variations: geometric and semantic (HERMAN et al., 2000). Geometric zooming simply scales up the drawing while semantic zooming shows more detailed information when approaching an area of the graph, thus requiring some sort of level of detail control.

One approach when dealing with large graphs is the use of incremental exploration and navigation techniques, which display only a small part of the graph at a time, showing other parts as they are needed. This approach allows the use of layout techniques that otherwise might be too expensive for real-time interaction, since it would only be applied to a relatively small subgraph at a time, and avoid screen clutter by showing in detail only a part of the dataset at a time (the context might still be visible, albeit in a limited and/or simplified form, such as cluster nodes). These techniques are useful for exploration of graphs that are not known in their entirety, such as the World Wide Web. Examples of techniques that uses this approach to dealing with large graphs is Archambault et al.'s Grouse (2007) and Rodrigues Jr. et al.'s GMine (2006; 2006; 2008).

3.3 Interactive Layout Reorganization

Most graph visualization techniques usually use interaction and navigation techniques to explore static, pre-computed layouts. As mentioned before, well-known techniques include filtering; fish-eye views; scrolling and panning; zooming and even coordination of two or more visualizations. Very few techniques, though, allow for dynamic interactive reorganization of graph layouts.

Some applications allow for simple layout reorganization by letting the user move around nodes in force-directed layouts, which will cause an alteration in the balance of energy of the force-directed system, thus triggering a repositioning of the nodes, which will move until equilibrium is again reached. Another known technique is to find clusters of nodes and transform them into cluster-nodes that can be expanded and collapsed by the user. Clusters can also be used to perform cluster-based semantic zooming, which allows for a level-of-detail-like approach to the visualization, letting the user incrementally explore the graph by zooming in or out.

Amongst the few techniques that allow for dynamic graph layout reorganization, we find the work of Henry et al., NodeTrix (2007), which is a hybrid of matrix and node-link visualizations. NodeTrix allows the user to turn clusters of nodes present in node-link visualizations into matrices, which are then displayed within the node-link diagram. The layout itself is computed with the previously mentioned LinLog algorithm (NOACK, 2003).

In our work we make use of a magnet metaphor to provide the user with tools for layout manipulation. Some other works have used a similar metaphor, but for purposes that are not necessarily related to graph visualization. Amongst these, we find Fidg't

(PROTOHAUS, ????) and Dust & Magnet (YI et al., 2005). Fidg't is an application developed for the management of social networks that includes an interactive visualization tool that allows users to iteratively explore their networks by creating tag magnets for pictures (from Flickr) or music (from Last.fm), and observing how its nodes are attracted or repelled. Although devoted to a different data domain (multivariate information), the Dust & Magnet information visualization technique also uses a magnet metaphor.

3.4 Focus+Context Interaction Techniques

Usually an intrinsic part of the visualization itself, focus+context interaction techniques allow a detailed view of a desired subset of information while at the same time showing the larger context where that information is located with a reduced level of detail (CARD; MACKINLAY; SHNEIDERMAN, 1999). As the user's focus of interest changes, the highlighted subset may also change correspondingly.

One of the classic focus+context techniques is the fisheye view (Figure 3.19), which shows a larger, detailed view of the desired information subset (possibly centered on the screen) while also displaying the rest of the information surrounding it (the context) in progressively less detail (FURNAS, 1981). The fisheye view is based on mapping the graph to a plane, defining a focus point and applying a degree-of-interest function on every node with their distance from the focus point as parameter.

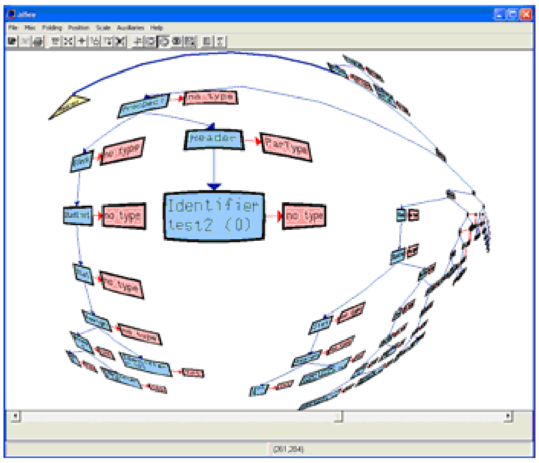


Figure 3.19: Fisheye view as seen in AbsInt's aiSee (see http://www.absint.com/aisee/).

There are many variants of the fisheye view technique, mostly divided into two categories: filtering and distortion (NOIK, 1993). Filtering fisheye views work by using the degree-of-interest function to display only the most interesting or relevant elements and are closer to what Furnas suggested. By far the most commonly found, the distortion techniques work by using a degree-of-interest function to geometrically distort the visualizations in order to obtain a fisheye lens look. There are also techniques with more than one focus point, such as the one by Storey et al. (1997).

The fisheye view can be applied to any visualization independent of its layout algorithm, but that is not the case with all focus+context techniques. The previously mentioned hyperbolic layouts (LAMPING; RAO; PIROLLI, 1995; MUNZNER, 1997) merge the fisheye-like visual distortions with the layout itself. This layout technique was developed with focus+context in mind, so it becomes an integral part of the layout algorithm instead of a separate stage applied later to the drawing.

3.5 Graph Visualization Toolkits

Many toolkits and libraries have been developed to aid the creation of graph visualization software, ranging from tools developed as part of academic research to free GPL-licensed libraries and commercial toolkits, supporting many languages and platforms, from C++ and Win32 to Java and .NET. Most allow the drawing of the graphs using many different algorithms and several are also able to handle interaction and navigation (drag and dropping nodes, zoom and pan, etc.). Many also support graph theory algorithms, with a few providing as well features specifically for network visualization. Table 3.1 provides a list of some toolkits.

It is important to note that while there is a multitude of graph visualization toolkits, the prototype for MagnetViz was implemented from scratch. Since the prototype was just a proof-of-concept, the alterations required in the toolkits' layout implementations to support MagnetViz would not be justified.

3.6 Final Comments

As mentioned in the previous sections, there is really no such a thing as a perfect graph layout. Depending of the user's necessities, a given technique might either apply or not. Even if a technique applies, it might have several limitations. A given layout algorithm, for example, might end up in a visualization that is too dense and cluttered, or a graph might eventually be so big that the users might not know where to start their search for information.

One of the approaches to circumvent all the natural limitations of graph drawing algorithms is to combine them with interaction techniques. A good interaction scheme might make the efficient exploration of a graph possible even if the graph is too big and complex or the layout not exactly ideal. This is the approach taken in the tool developed for this work, which is described in detail in the following chapter.

Toolkit	**Licensing**	**Supports**	**URL / Reference**
GDToolkit	Free for academic use; Commercial	C++	http://www.dia.uniroma3.it/~gdt
GINY	Free, Open-source	Java	http://csbi.sourceforge.net
GoVisual	Commercial	Win32, .NET, Java, Linux	http://www.oreas
Gravisto	Not yet available	Java	http://gravisto.fim.uni-passau.de (BACHMAIER et al., 2005)
GVF (Graph Visualization Framework)	Free, Open-source	Java	http://gvf.sourceforge.net
HyperGraph	Free, Open-source	Java	http://hypergraph.sourceforge.net
ILOG Views GraphLayout	Commercial	C++, Java, .NET	http://www.ilog.com
InfoVis Toolkit	Free for academic use	Java	http://ivtk.sourceforge.net
JGraph	Free; Commercial	Java	http://www.jgraph.com
JGraphT	Free, Open-source	Java	http://jgrapht.sourceforge.net
JUNG	Free, Open-source	Java	http://jung.sourceforge.net
LGL (Large Graph Layout)	Free, Open-source	Java	http://lgl.sourceforge.net
Prefuse	Free, Open-source (BSD)	Java	http://www.prefuse.org
Tom Sawyer Visualization	Commercial	ActiveX, ASP.NET, Java, JSP, MFC	http://www.tomsawyer.com
Tom Sawyer Layout	Commercial	ActiveX, C++, Java, .NET	http://www.tomsawyer.com
Touchgraph	Free; Commercial	Java	http://www.touchgraph.com http://touchgraph.sourceforge.net
Tulip	Free, Open-source (GPL)	C++	http://www.tulip-software.org
yFiles	Commercial	Java	http://www.yworks.com

Table 3.1: Toolkits for graph visualization.

4 MAGNETVIZ

4.1 Overview

The aim of this work is to introduce MagnetViz, a technique that is able to aid users in interactively reorganizing the layout of a graph to better fit their particular needs by giving them tools that allow for the manipulation of graph visualizations based on topological and semantic attributes. This is inherently a focus+context approach, since it lets the users find and have more details about some specific subset of nodes while at the same time maintaining the other nodes on display and reorganizing their disposition in a way that clearly shows their relationship to the ones in focus.

Although Archambault's GrouseFlocks (2008) also takes a middle-ground approach when it comes to favoring semantics or topology, its goals and approach differ from MagnetViz's. GrouseFlocks is geared at visualizing and manipulating graph hierarchies, while MagnetViz is a set of tools for the manipulation of graph layouts. GrouseFlocks does not use traditional graph drawing algorithms, while MagnetViz is designed to complement them, allowing the user to query the graph based on its semantic and topological features.

The technique developed in this work is based on the physics metaphor of the force-directed layouts. It provides the user with virtual magnets that can be inserted into the scene and set to attract nodes and edges that fulfill a set of criteria based on the semantic and/or topological attributes of the graph. Force-directed techniques were chosen as a starting point because they are widely used, are flexible and easily adaptable, and are a natural fit for real-time applications that require the visual representation of the graph to dynamically change according to some criteria.

With the goal of evaluating MagnetViz, a prototype was implemented and subsequently used in a case study, which will be covered in the next chapter. This prototype was built for generic graph visualization, with no specific application in mind. The technique, however, can be easily adapted to work for more specific cases.

The prototype was implemented in C++ using OpenGL for the rendering and the free, open-source version of Nokia/Qt Software's Qt library (INC., ????) for the some of the interface, data structures and handling of XML files. It can open graphs stored in the GraphML format (GRAPHML, ????), which can then be explored using the tools provided by MagnetViz, as well as basic interaction techniques (selection, drag+dropping, zooming and panning). Even though there are many toolkits with graph visualization capabilities, none were used, since MagnetViz requires not only an alteration of the layout algorithm but also in the way it is executed. Since the current implementation is just a proof-of-concept, the work involved in adapting a toolkit to handle MagnetViz would not be worth the effort.

The prototype allows the user to import a graph, manipulate and explore it using the tools of MagnetViz. The graph is displayed using a modified version of Tunkelang's (1999) approach, with the users being allowed to pause and play the physics simulation whenever they desire. Figure 4.1 shows a sample graph layout generated with this algorithm for the co-authorship network of the AVI conference (years 1994 - 2006).

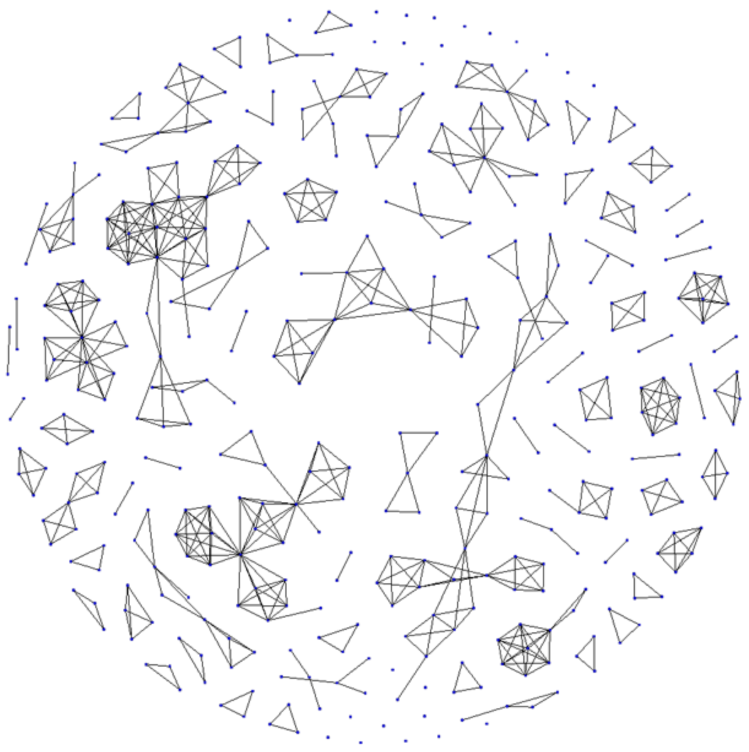


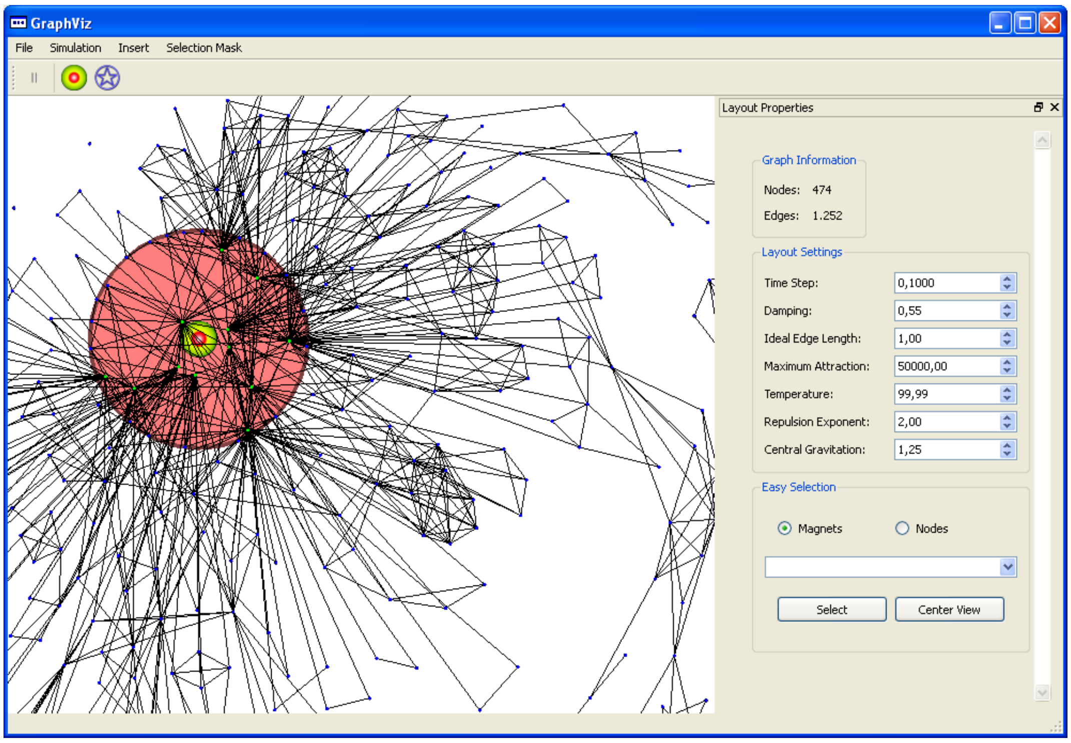
Figure 4.1: A sample graph layout generated by our algorithm from the AVI co-authorship dataset (years 1994-2006).

Layout properties are usually displayed on a panel to the right of the working area whenever no object (nodes, edges, magnets) in the scene is selected. Object properties are displayed on the same panel whenever a single object is selected. In case of selection of a node or edge, their attributes are displayed in this panel as well. Users can select objects by clicking on them and subsequently moving them by drag-and-dropping. The scene can be navigated by panning and zooming. Figure 4.2 shows the user-interface.

The following sections cover all the features that comprise MagnetViz, which include gravity spots, magnets and all the attraction criteria that can be used with them. Details about the data structures used by the prototype to internally represent a graph are then given, followed by a description of the algorithm and its alterable properties. The chapter is then closed with a few final comments.

4.2 Gravity Spots

Gravity spots were designed to help users shape the graph to their liking by attracting nodes to certain regions of the scene. Therefore, a gravity spot basically attracts nodes towards itself, with the user being able to define whether it will affect all nodes of the scene or only the nodes that are within a certain radius of its location. In the latter case, the user can set to display a circle with the desired radius. They can be inserted wherever



Graph Information

Nodes: 474

Edges: 1.252

Layout Settings

Time Step:	0,1000
Damping:	0,55
Ideal Edge Length:	1,00
Maximum Attraction:	50000,00
Temperature:	99,99
Repulsion Exponent:	2,00
Central Gravitation:	1,25

Easy Selection

[✓ Magnets] Nodes

Figure 4.2: User-interface of the MagnetViz prototype.

the user desires by the simple click of a mouse button and subsequently moved about whenever the user finds necessary. The strength of the force they apply on the attracted nodes can be set through their properties panel. Figure 4.3 shows the icon used for a gravity spot and Figure 4.4 shows their properties. Figure 4.5 shows a gravity spot being used to attract nodes towards the top left corner of the workspace.

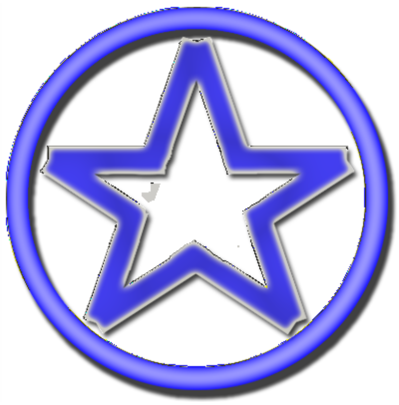
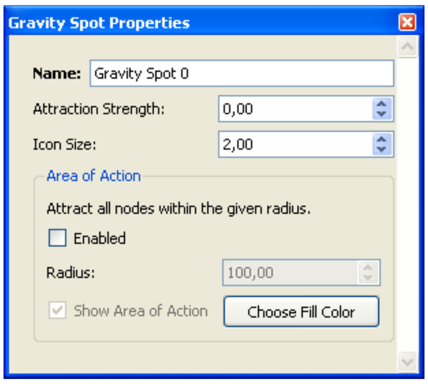


Figure 4.3: Gravity spot icon.

4.3 Magnets

Magnets are tools that can be inserted into the scene to attract all nodes that fulfill a user-defined set of criteria. A magnet works by exerting onto each of these nodes an


Gravity Spot Properties

Name: Gravity Spot 0

Attraction Strength: 0,00

Icon Size: 2,00

Area of Action

Attract all nodes within the given radius.

☐ Enabled

Radius: 100,00

☑ Show Area of Action

Figure 4.4: Gravity spot properties panel.

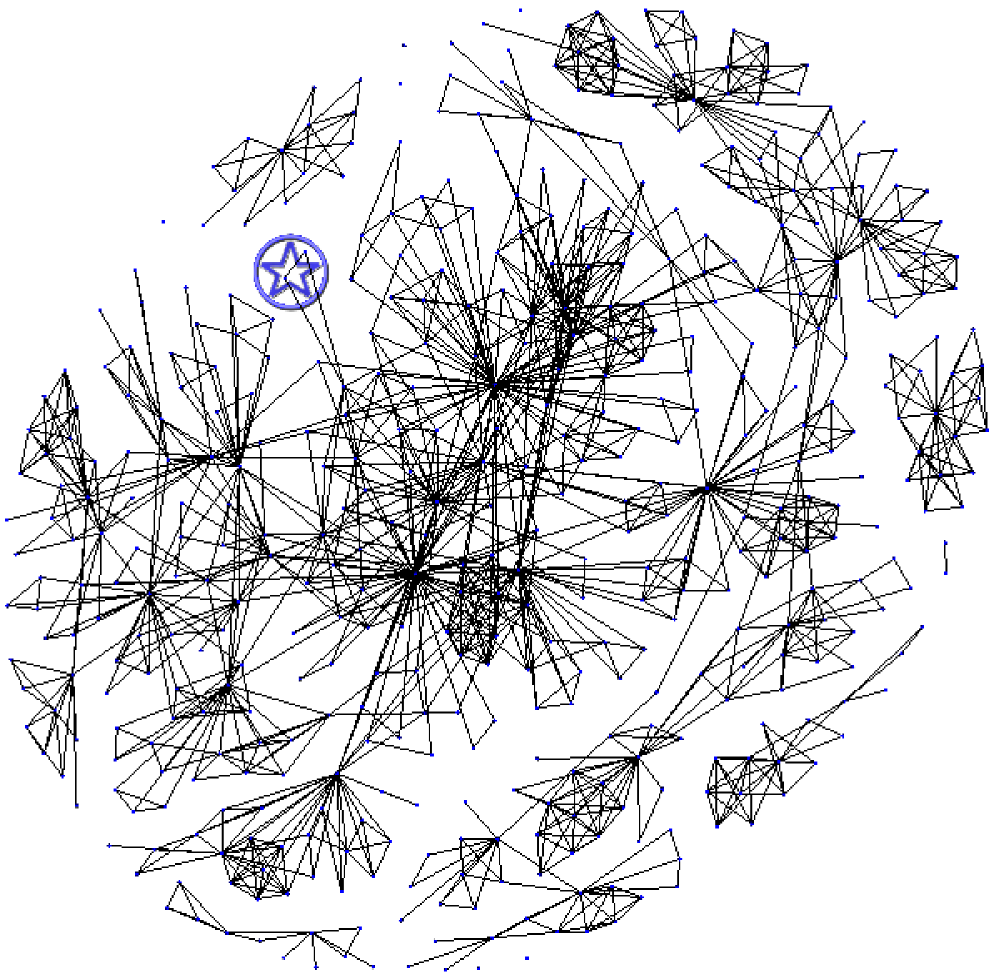


Figure 4.5: Gravity spot in action .

attraction force that will progressively move them towards it, thereby building a cluster of semantically-related nodes around it. When these nodes move, the force-directed layout algorithm ensures that all the other nodes that are connected to them by edges will be pulled along, reorganizing the layout of the graph in the process. Figure 4.6 shows how magnets are visually represented in the prototype we developed while a magnet that attracts all nodes that have a degree of 3 can be seen in Figure 4.7.

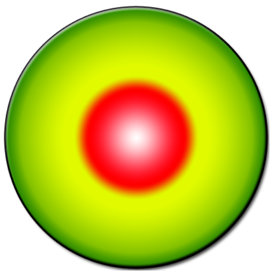


Figure 4.6: Magnet icon.

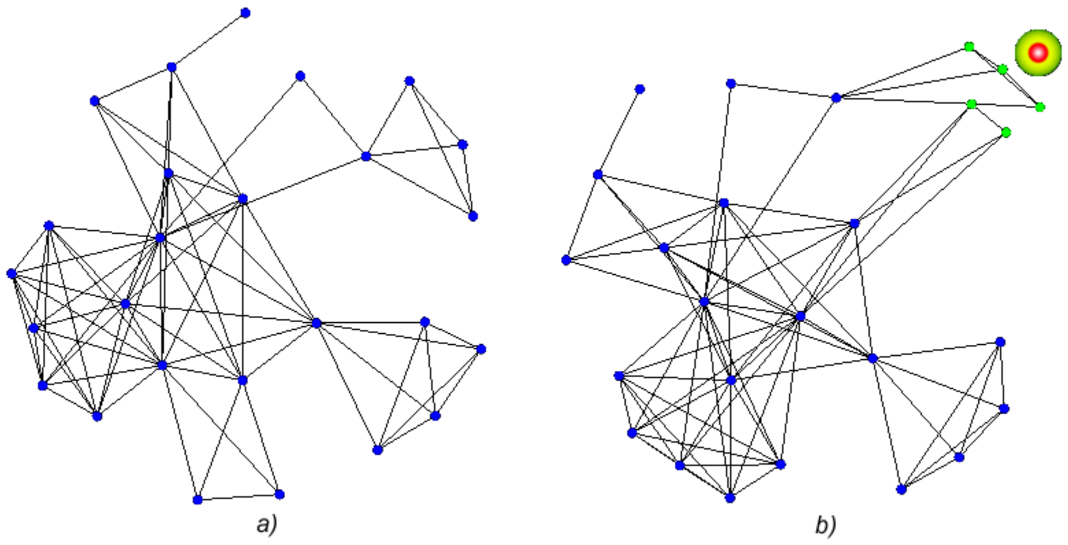


Figure 4.7: a) Largest component of the AVI co-authorship network in its initial layout; b) A magnet in action, attracting all nodes of degree 3.

To define which nodes a magnet should attract, users must assign it a set of attraction criteria. These criteria can be set as requirements or as simple criteria. To be attracted by a magnet, a node should satisfy all of its requirements and at least one of its criteria. If a magnet has no criteria, attraction is based only on the requirements, and vice versa for the case of a magnet having no requirements. These requirements and criteria can be based on the topology of the graph, the attributes of its nodes and edges or even other magnets that have been placed on the scene.

Within the physics metaphor, a magnet works by applying on each of the attracted nodes a force in its direction, with the magnitude of this force being defined by the user. To make sure that it is not overlapped by the attracted nodes and to keep these nodes from being all bundled together too close to each other, a magnet also exerts a repulsion force on all the graph's nodes. This repulsion force is analogous to the one exerted by common nodes, working like a reverse gravity by being inversely proportional to the distance of the node to the magnet and proportional to the magnitude of the magnet's force of attraction, so that it is stronger with nodes that are closer to the magnet and weaker with the ones that are progressively further away from it.

Occasionally it might be hard to see which of the nodes that are close to a magnet are actually attracted by it. To deal with this situation, it is possible to assign a color to all the nodes that a magnet attracts, making them more distinguishable. If desired, the user may also define a region where all the attracted nodes should be. This is done through a boundary shape (Figure 4.8), which is simply a circle that is placed around the magnet. Nodes that are attracted by the magnet are forced to stay within the boundary shape while other nodes are forced out. In the prototype, the boundary shape can have its radius and background color set by the user, making the magnet's attracted nodes very clear to the user.

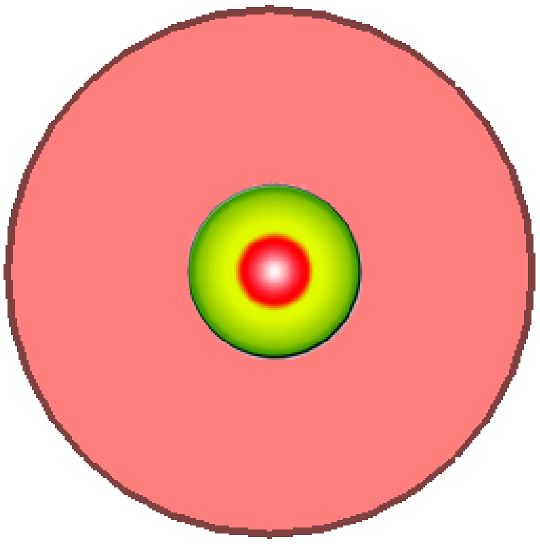


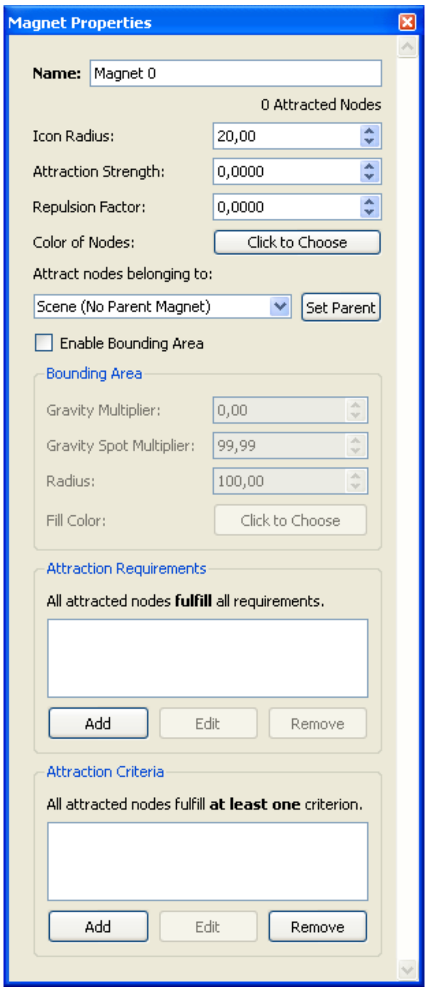
Figure 4.8: Magnet with boundary shape.

Magnets can be placed and moved anywhere on the scene by the user and all of their properties can be altered at any time through the panel displayed when a magnet is selected (Figure 4.9).

When a magnet has a boundary shape, its attracted nodes ignore all forces exerted on them but the magnet's attraction force (which is also doubled) until they are within the bounding region. Once inside, they are affected by all forces normally, but are kept from escaping the region by simple collision detection. The user may also choose to increase or decrease the effect of the central gravity and the gravity spots on the nodes that are within the shape by setting a scalar multiplier on these forces.

A magnet effectively creates sets of related nodes and ensures that they remain near a certain physical region. Occasionally it might be useful to refine this set of nodes into subsets. To allow for that, it is possible to define magnets that act only on the subset of the graph that is already attracted by another magnet. To do this, the user must simply create a magnet and define another one as its parent. It is interesting to note that children magnets might children magnets of their own; creating thus a hierarchy of magnets that might be helpful for incremental exploration of a graph, corresponding to composing complex queries.

Occasionally it might happen that some nodes fulfill the criteria of two or more magnets. Whenever such magnet "intersections" are found, these common nodes are visually



Magnet Properties

Name:	Magnet 0
	0 Attracted Nodes
Icon Radius:	20,00
Attraction Strength:	0,0000
Repulsion Factor:	0,0000
Color of Nodes:	Click to Choose

Attract nodes belonging to:

Scene (No Parent Magnet) | Set Parent

☐ Enable Bounding Area

Bounding Area

Gravity Multiplier:	0,00
Gravity Spot Multiplier:	99,99
Radius:	100,00
Fill Color:	Click to Choose

Attraction Requirements

All attracted nodes **fulfill** all requirements.

Add | Edit | Remove

Attraction Criteria

All attracted nodes fulfill **at least one** criterion.

Add | Edit | Remove

Figure 4.9: Magnet properties panel.

distinguished from regular nodes by having a different icon (as seen in Figure 4.10) and by colored lines that link them to their attracting magnets, with each line being in the same color as the nodes attracted by their corresponding magnets. The user will be also asked if the common nodes should be left free or if they should be bound to any of the magnets (Figure 4.11). If they are left free, they will ignore boundary shapes and will be affected by both magnets. If they are bound to a magnet, they will be physically treated like all the other nodes of this magnet and will ignore the other.

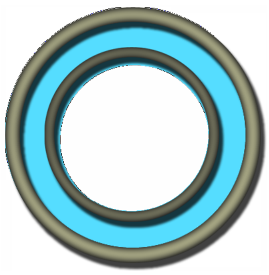


Figure 4.10: Intersection node icon.

4.4 Attraction Criteria

To make a magnet attract a particular set of nodes, users have to assign it criteria. These criteria can be set as requirements that have to be fulfilled by any attracted node or as simple criteria, of which (if there are any) at least one should be fulfilled. These requirements and criteria can be based on the topology of the graph, the attributes of its nodes and edges or even other magnets that have been placed on the scene.

Topology-based criteria use the structure of the graph to attract nodes. In our prototype, the following topology-based criteria were implemented:

- **Degree Criterion.** It can be used to attract all nodes that have as degree a value that is within a user-specified range. Figure 4.12 shows the dialog used to create and edit a criterion of this type.

- **Path Length Criterion** It can be used to attract all nodes that have as path length to another user-specified node a value that is within a user-specified range. Figure 4.13 shows the dialog used to create and edit a criterion of this type and Figure 4.15 shows an example of a magnet that attracts all nodes with path length 1 from a node that been selected in the dialog box.

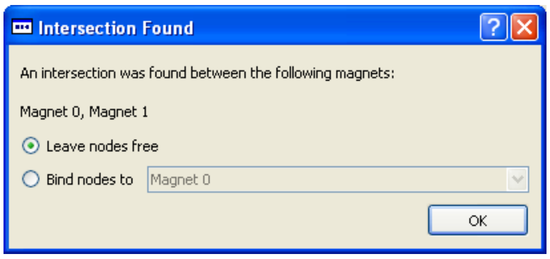
- **Connected Subgraph Criterion** It can be used to attract subgraphs that have an amount of nodes that is within a certain user-specified range. These subgraphs can be maximally connected (connected components) or not. Figure 4.14 shows the dialog used to create and edit a criterion of this type. Figure 4.16 shows a magnet that attracts all connected subgraphs that have from 4 to 15 nodes.

Attribute-based criteria use the semantic properties contained in nodes and edges in order to attract nodes. The following were implemented:

- **Node/Edge Has Attribute.** It can be used to attract all nodes/edges that have a certain attribute. Figure 4.17 shows the dialog used to create and edit a criterion of this type.

- **Node/Edge Has Attribute With Given Numerical Value.** It can be used to attract all nodes/edges that have a certain attribute with a value within a given range. Figure 4.18 shows the dialog used to create and edit a criterion of this type.

- **Node/Edge Has Attribute With Given Substring.** It can be used to attract all nodes/edges that have a certain attribute containing a given substring. Figure 4.19 shows the dialog used to create and edit a criterion of this type. An example can be seen in Figure 4.20.


Intersection Found

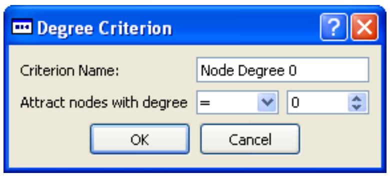
An intersection was found between the following magnets:

Magnet 0, Magnet 1

[✓ Leave nodes free]

Bind nodes to Magnet 0

Figure 4.11: Intersection found dialog.

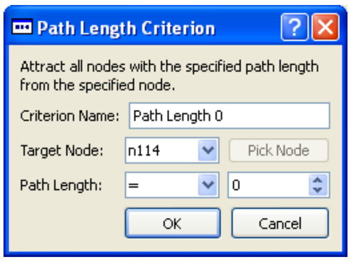


Degree Criterion

Criterion Name: Node Degree 0

Attract nodes with degree = 0

Figure 4.12: Degree criterion properties.



Path Length Criterion

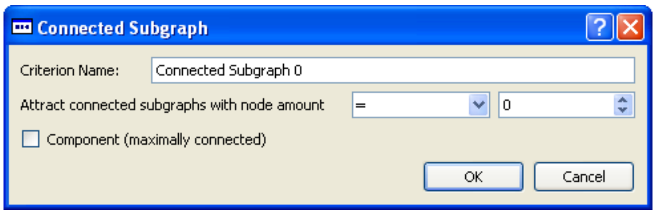
Attract all nodes with the specified path length from the specified node.

Criterion Name: Path Length 0

Target Node: n114

Path Length: = 0

Figure 4.13: Path length criterion properties.



Connected Subgraph

Criterion Name: Connected Subgraph 0

Attract connected subgraphs with node amount = 0

☐ Component (maximally connected)

Figure 4.14: Connected subgraph criterion properties.

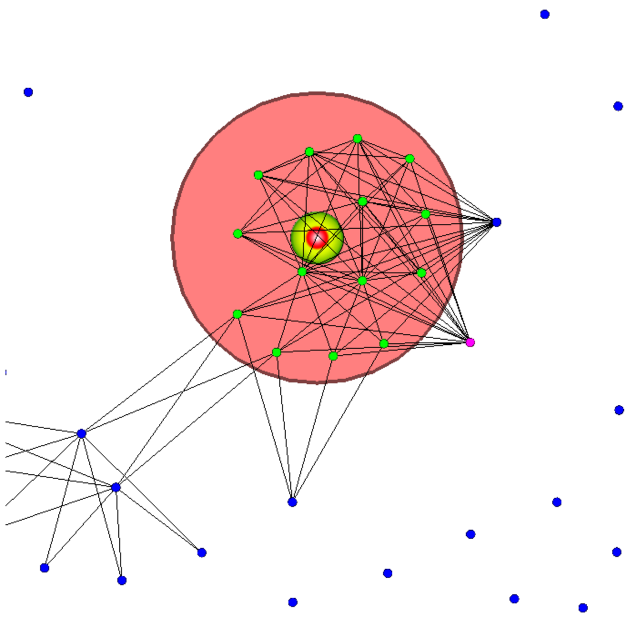


Figure 4.15: Magnet with path length criterion that attracts all nodes with path length 1 from the pink node.

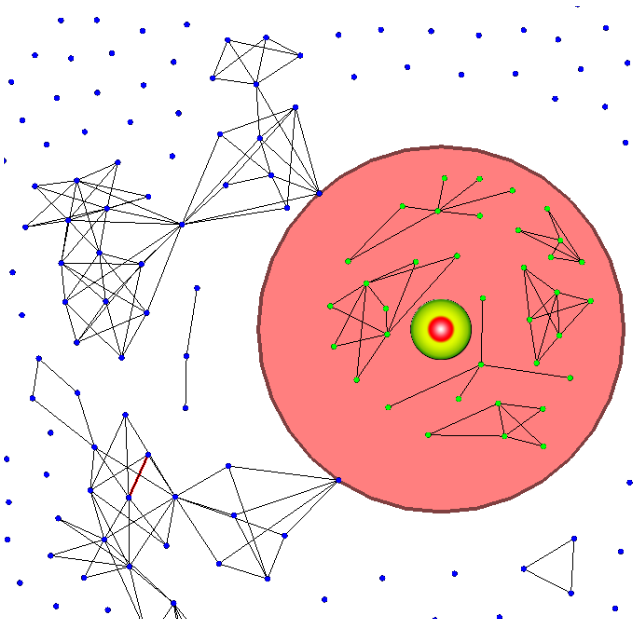
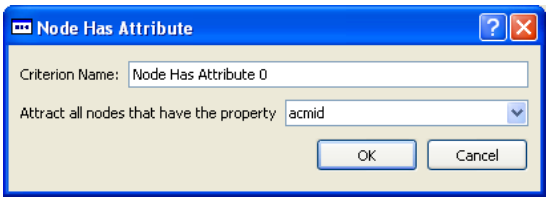


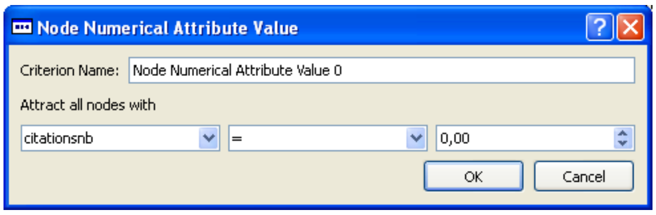
Figure 4.16: Magnet with connected subgraph criterion that attracts all non-unitary groups of less than 20 nodes.


Node Has Attribute

Criterion Name: Node Has Attribute 0

Attract all nodes that have the property acmid

Figure 4.17: Properties of the "Node Has Attribute" criterion.

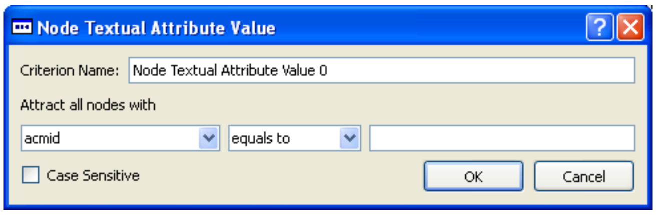

Node Numerical Attribute Value

Criterion Name: Node Numerical Attribute Value 0

Attract all nodes with

citationsnb = 0,00

Figure 4.18: Properties of the "Has Attribute With Given Numerical Value" criterion.


Node Textual Attribute Value

Criterion Name: Node Textual Attribute Value 0

Attract all nodes with

acmid equals to

Case Sensitive

Figure 4.19: Properties of the "Has Attribute With Given Substring" criterion.

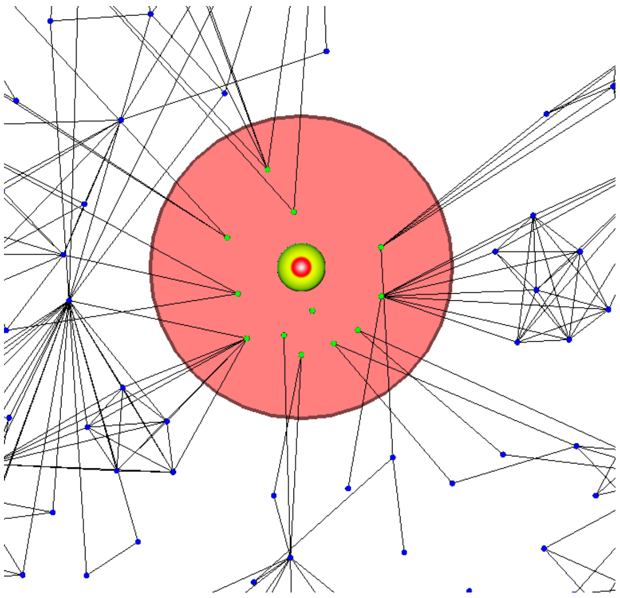
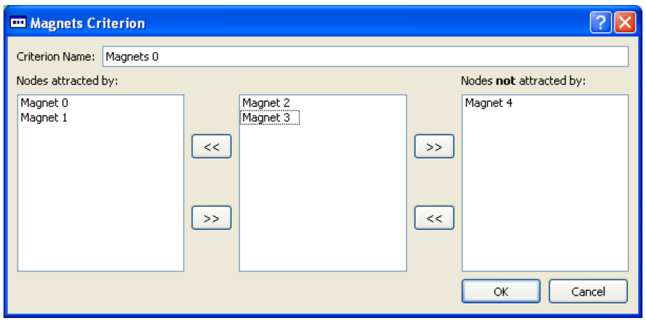


Figure 4.20: Magnet with the "Has Attribute With Given Substring" criterion.

A magnet-based criterion can be used to attract all nodes that are attracted by some magnets and not attracted by others. This criterion's dialog, as shown in Figure 4.21, shows in the center a list of all the scene's magnets. These magnets can be moved to the lists on the left or right using the arrow buttons. The criterion will set the magnet to attract all the nodes that are attracted by the magnets on the list to the left and all the nodes that are not attracted by the magnets on the list to the right. A practical example can be seen in Figure 4.22.



Magnets Criterion

Criterion Name: Magnets 0

Nodes attracted by: Magnet 0, Magnet 1

Magnet 2, Magnet 3

Nodes **not** attracted by: Magnet 4

Figure 4.21: Magnets criterion properties dialog.

4.5 Graph Representation

Graphs are represented as GraphML files. A typical GraphML file is an XML file that begins with a declaration of the type of graph (directed or undirected) and a listing of the properties nodes and edges can have and their types (integer, string, etc.). This is followed by the nodes and edges themselves, with the values they have for each property. Nodes and edges cannot have properties without values. A parser built on top of Qt's XML classes is used to read the GraphML. A sample GraphML can be seen in Figure

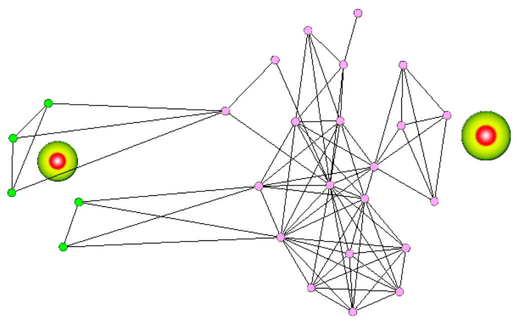


Figure 4.22: Magnet with Magnets criterion (on the right), attracting all nodes not attracted by the one on the left.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<graphml>
    <graph edgedefault="undirected" id="co-authorships">

        <key attr.name="name" attr.type="string" for="node" id="name"/>
        <key attr.name="numpapers" attr.type="int" for="node" id="numpapers"/>
        <key attr.name="numcommonpapers" attr.type="int" for="edge" id="numcommonpapers"/>

        <node id="JDOE">
            <data key="name">John Doe</data>
            <data key="numpapers">5</data>
        </node>
        <node id="JSMITH">
            <data key="name">Jane Smith</data>
            <data key="numpapers">3</data>
        </node>

        <edge id="JDOE_JSMITH" source="JDOE" target="JSMITH">
            <data key="numcommonpapers">2</data>
        </edge>
</graph>
</graphml>
```

Figure 4.23: Sample GraphML file.

4.23.

Internally, graphs are represented by three classes: *Graph*, *Node* and *Edge*. *Graph* contains pointers to nodes and edges. *Node* and *Edge* inherit from a *GLSceneObject* class, which contains properties and functions common to most objects that have to be drawn by the prototype's rendering engine (position, scale, color, visibility, virtual functions for drawing, collision detection and picking routines, etc.). On top of that, *Node* contains lists of pointers to edges and magnets, a list of attributes and values, and a pointer to a binding magnet (if there is one), as well as properties such as mass (always set to 1.0 in the case of this prototype), radius and implementations of the necessary virtual functions. *Edge* contains pointers to its two nodes and a list of attributes and values, and also implementations of required virtual functions.

Magnets and Gravity Spots are respectively represented by the *Magnet* and *GravitySpot* classes, which also inherit from *GLSceneObject*. *Magnet* properties such as radius, strength, radius of bounding area, parent magnet, fill color of bounding area, multipliers for gravity and gravity spot forces, and color of attracted nodes, as well as hash maps for its associated criteria and requirements and a list of attracted nodes. *GravitySpot* is similar, but doesn't contain the criteria and requirements and the bounding area, having instead a property that defines the radius of the region where it can act.

Requirements and criteria are a both instances of the same set of classes, being differentiated simply by the list in which they are placed. These classes inherit from *AttractionCriterion*, which is an abstract class that defines the interface and common elements of all criteria. These common elements are the magnet for which the instance of the criterion was created, and virtual functions to ensure that the criteria will have routines to verify that a node fulfills them and to display their associated properties dialog. Each criterion has its own different properties dialog class, with all of these inheriting from Qt's *QDialog*.

A simplified class diagram of the main data structures of the MagnetViz prototype can be seen in Figure 4.24.

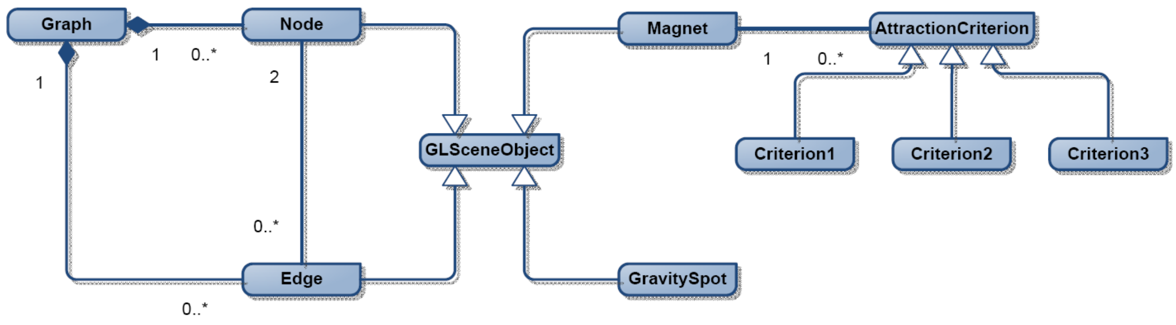


Figure 4.24: Simplified class diagram of the MagnetViz prototype data structures.

4.6 Layout Algorithm

The basic layout used as a basis in our prototype is based on the modified version of the adapted Fruchterman-Reingold algorithm proposed by Tunkelang (1999) described in the related work. This approach was taken because Fruchterman-Reingold is one of the most classic and widely used graph-drawing algorithms, produces natural-looking layouts, is straightforward to implement and lends itself easily to our technique. However, even though this was the one we implemented for our prototype, practically any force-directed algorithm can be used in its place - the only requirement is that the algorithm must allow for the application of forces on the nodes.

Since our technique requires the interactive manipulation of the graph layout, we had to make some modifications in Tunkelang's original approach (1999) in order to support that. The original technique runs the simulation for a fixed number of iterations while gradually decreasing a temperature. Since we deal with real time interaction, we needed the simulation to be constantly running to be able to handle any modifications made on the scene by the user, much like the physics engine of a game. Therefore, aside from the regular algorithm, we created a new version (seen as Algorithm 5) that uses the integration method proposed by Verlet (1967) to run constantly in real time and allow for the utilization of the forces generated by the magnets.

It must be noted that Algorithm 5 is a simplified version of what was actually implemented. The version here depicted does not take into consideration a few details which were omitted for simplicity's sake. The actual algorithm handles cases such as whether a node is bound to a magnet (in case of a magnet intersection), is outside the boundary

Algorithm 5 Simplified MagnetViz layout algorithm.

```
**Require:** $C$ {Constant for computation of the optimal distance between nodes.}
**Require:** $W, H$ {Width and height of the workspace}
**Require:** $nodes$, $edges$ {Lists of nodes and edges}
**Require:** $magnets$ {List of magnets}
**Require:** $gravitySpots$ {List of gravity spots}
**Require:** $repulsionExponent$, $centralGravityStrength$, $damping$, $\Delta t$

{Note: This algorithm is called at every timer event.}

{Calculate optimal distance between nodes}
$k \leftarrow C * \sqrt{(W * H)/nodes.count()}$

**function** $attractiveForceStrength(x) \leftarrow x^2 / \ k$
**function** $repulsiveForceStrength(x) \leftarrow k^{repulsionExponent} / \ x$

{Calculate attractive forces.}
**for each** $e$ **in** $edges$ **do**
    $\Delta \leftarrow e.n1$.pos() - $e.n2$.pos()
    $e.n1.force -= ( \ \Delta \ / \ | \ \Delta \ | \ ) * e.n1$.mass() $* \ attractiveForceStrength(| \ \Delta \ |)$
    $e.n2.force += ( \ \Delta \ / \ | \ \Delta \ | \ ) * e.n2$.mass() $* \ attractiveForceStrength(| \ \Delta \ |)$
**end for**

Build $barnesHutTree$.

{Calculate repulsive forces.}
**for each** $n$ **in** $nodes$ **do**
    Calculate $centralGravityForce$ using $centralGravityStrength$ and $n$'s position
    Calculate $magnetForce$ {Total force generated by $n$'s magnets}
    Calculate $gravitySpotForce$ {Total force generated by $n$'s gravity spots}
    $n.force += centralGravityForce + magnetForce + gravitySpotForce$
    $n.force += bhGetRepulsiveForce(n, barnesHutTree)$

    $newPos = n$.pos() + $((n$.pos()$-$ $n$.oldPos() $+$ $n$.force()$) * \Delta t^2 * damping)$

    Deal with temperature and position constraints (workspace area, boundary shapes)

    $n.pos \leftarrow newPos$
**end for**
```

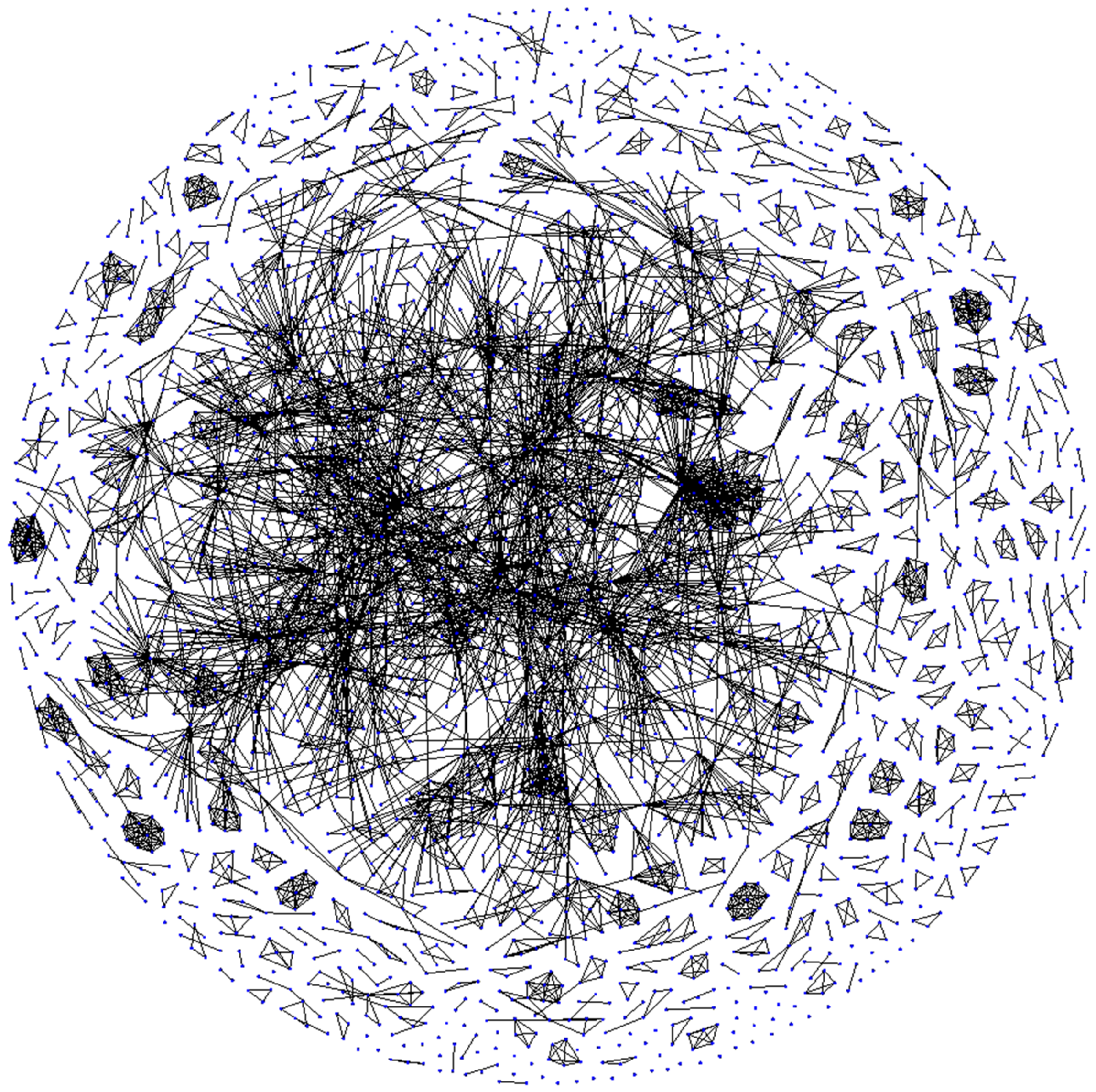


Figure 4.25: CHI co-authorship graph, with 2800 nodes and 5724 edges.

shape of a magnet that attracts it (in which case the only force that affects it is the force of that magnet), should be affected differently by certain specific forces (i.e. if it is inside a boundary shape and values for the multipliers were set), etc..

Another small modification made to the original technique was the addition of a central gravitation force, which is applied on all the nodes to make the visualization more aesthetically pleasing and keep nodes from getting too close to the border. The visual result looks somewhat like constellations in space, as can be seen in Figure 4.1. However, dense graphs (Figure 4.25) are hard to observe in details, thus justifying the interaction technique developed thus far.

Whenever a graph is opened in our prototype, some pre-computation is done to provide an initial layout. Having an initial layout keeps the user from having to wait while the layout converges and gives the dynamic algorithm a starting point from which it can work. In the case of our prototype, we simply run for 200 iterations the adapted Furchterman-Reingold algorithm slightly modified to also apply on each node the central gravitation force with its default value. This number of iterations was chosen arbitrarily.

Many properties of the layout are editable and accessible by the user through its properties panel, which can be seen in Figure 4.26. If the physics simulation is not paused, all user alterations in the properties take effect immediately. The editable properties are:

- **time step**

  Specifies the frequency of timer events. These events are continuously triggered in a specified frequency (i.e. every millisecond). Whenever they occur the physics engine runs its procedures.

- **damping**

  Real value that multiplies the total force of each node to gradually decrease their magnitude and help the system converge.

- **optimal distance value**

  Real value used to compute the optimal distance between nodes.

- **maximum attraction force**

  Limits the total force that can be applied to any node to aid in the convergence of the layout and make it more stable.

- **temperature**

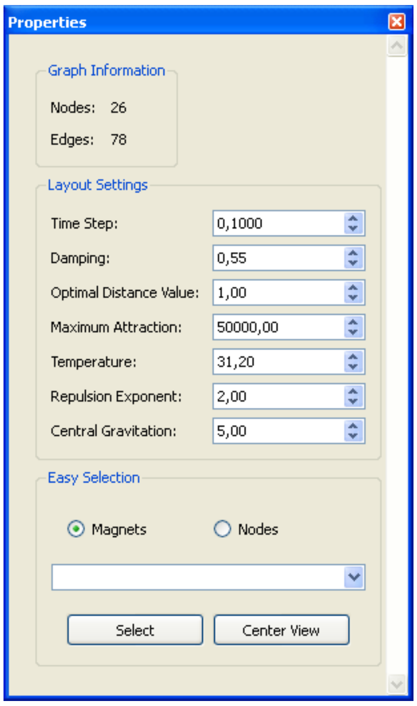
  Limits the movement of the nodes to help the convergence of the system.

- **repulsion exponent**

  Used to control the strength of the repulsion force that exist between nodes.

- **central gravitation**

  Magnitude of the gravitation force that pulls all nodes towards the center of the workspace.


Properties

Graph Information

Nodes:	26
Edges:	78

Layout Settings

Time Step:	0,1000
Damping:	0,55
Optimal Distance Value:	1,00
Maximum Attraction:	50000,00
Temperature:	31,20
Repulsion Exponent:	2,00
Central Gravitation:	5,00

Easy Selection

[✓ Magnets] Nodes

Figure 4.26: Layout properties panel.

The properties panel also includes a box called "Easy Selection", which in the current version of the prototype allows the user to select or center the camera on any node or magnet by its name/id.

4.7 Final Comments

MagnetViz seizes on the physics metaphor of the popular force-directed algorithms to allow users to shape layouts according to their needs. With MagnetViz, users are able to perform queries in a manner similar to query languages, with the results being returned visually instead of textually and displayed in a focus+context approach. Users are able to see the way the graph relates to their query results by the way the layout adapts itself to the modifications inflicted on the physical system by the movement of the nodes attracted by magnets that are inserted into the scene. It is important to note, though, that MagnetViz does not have the same power of expression as traditional query languages and was not designed as a substitute for them, since the options for query composition are limited.

The latest version of the prototype is the fourth major revision. The first version was a simple proof of concept written in C++ and DirectX, which contained limited user interaction. The second version was a more complete implementation using Python and Qt, being practically feature-complete. Due to Python's interpreted nature, though, it suffered from performance issues. This version was used to refine the technique and its related algorithms and then ported to C++ and Qt. Performance gains were achieved, but Qt's built-in rendering engine proved to be a major bottleneck. Therefore, a rendering engine was built from scratch using OpenGL, resulting in the current implementation, which can easily handle graphs with several thousand nodes on a modest computer.

Despite this considerable speed increase, it can still be said that performance is this technique's major drawback, since it is naturally limited by current hardware capabilities. Though current technology is already fast enough to allow for the handling of reasonably large graphs (thousands of nodes and edges) in real time, the high complexity of the physics algorithm impairs its scalability. The bigger a graph is, the more costly it is to compute a layout and interact with it in real time.

In a 2 GHz Athlon 64 computer with 2 MB of DDR2 memory and an ATI 9700 Mobile graphics card, the prototype was able to comfortably handle a graphs with 2800 nodes and 5724 edges (the CHI Co-authorship graph, seen in Figure 4.25). It should be noted that while some performance considerations were taken into account, they were not priorities in the implementation of this prototype and design of the technique's layout algorithm.

5 TOOL EVALUATION

One of the most important and complex aspects of developing a graph visualization technique is its evaluation. While studies have provided some generally applicable guidelines that good techniques tend to follow, there is no hard rule. What works for one application might not necessarily work for another, and this happens in all levels of a technique - from layout to interaction scheme.

Based on empirical studies of human perception, some works described aesthetic properties that increase the readability of a graph (namely, the amount of edge crossings, edge bends, edge length, etc.) (BATINI; FURLANI; NARDELLI, 1985; COHEN et al., 1994; PURCHASE; COHEN; JAMES, 1997). Guidelines also exist for 2D interaction and navigation schemes, being the longtime subject of intensive research in the field of Human-Computer Interaction. While all of these guidelines can be used to build a technique for graph visualization, though, they are by no means a guarantee of the practical usefulness of the technique. With that in mind, to evaluate MagnetViz it was decided to take two different approaches. The first, aiming to see how it fares in answering generic graph-related questions, consists of putting it against a task taxonomy that was proposed specifically for graph visualization applications (LEE et al., 2006). The second approach consisted of applying the technique within a specific context and subsequently performing user tests to evaluate its usefulness in a practical real-world application. The application chosen for the users tests was the visualization of a social network - namely, the co-authorship graph obtained from all the works published in 2007 by the UFRGS professors, students and their external collaborators.

Section 5.1 describes the taxonomy-based evaluation while section 5.2 covers the user tests.

5.1 Taxonomy-based Evaluation

Lee et al. (2006) have proposed a useful task taxonomy for graph visualization in which it is defined a list of tasks that are commonly performed while exploring a graph. To obtain an initial evaluation of the MagnetViz technique, it was measured against this taxonomy on a task by task basis. By doing this it was possible to find out that most of the tasks are inherently covered by the technique, while the ones that are not currently supported can be made so by the adding of new features or the application of a complementary technique.

In their paper, Lee et al. (2006) divide graph visualization tasks in three categories: general low-level tasks, graph-specific tasks and high-level tasks. The low-level tasks are comprised of 10 general, not graph-specific, visual analytic tasks that were originally described by Amar et al. (2005) and three other operations (of which one is exclusive to graphs) proposed by the authors themselves. Graph-specific tasks are further categorized into topology-based, attribute-based, browsing and overview tasks, with most of these being built on combinations of the low-level tasks.

Topology-based, attribute-based and browsing tasks are covered in detail in Subsections 5.1.2, 5.1.3 and 5.1.4. Overview tasks correspond to a compound exploratory task performed in order to quickly get estimated values, such as the size of a graph or subgraph, or patterns that the graph (or parts of it) tends to have. Estimate values might sometimes be more important than exact ones and often a simple look at an overview of the graph might provide the desired information. Often topology tasks can also be covered by simple visual examination of the overview of the graph, especially if the layout algorithm in use allows for easy visual recognition of clusters, connected components, or any other element.

Lee et al. (2006) also mentions high-level tasks, which are not covered by the above tasks. High-level tasks include application-specific tasks, as well as general comparison of two graphs, identification of redundant/duplicate nodes (i.e. two nodes that represent the same author), naming a group of nodes, choosing a node to represent a cluster, the evolution of a graph over time, etc.. Due to their generic nature, overview and high-level tasks are not directly covered by the study here presented.

The following subsections contain listings of low-level, topology-based, attribute-based and browsing tasks (subsections 5.1.1, 5.1.2, 5.1.3 and 5.1.4, respectively), with descriptions of how MagnetViz would (or not) be able to handle them. Sample questions from the co-authorship network domain are used as examples throughout these subsections to illustrate each of the tasks. Subsection 5.1.5 contains a short discussion and the preliminary conclusions obtained with this initial study.

5.1.1 Low-level Tasks

The low-level tasks are comprised of general, not graph-specific, visual analytic tasks that were originally described by Amar et al. (2005) and complemented by Lee et al. (2006). They are listed, described and commented below.

1. **Retrieve Value**

   Given a specific set of cases, find attributes of those cases.

   Example: In what conference was a certain paper published?

   Comment: Though not exactly a part of the technique, this task is supported in the prototype, since the information contained in the attributes of the nodes and edges can be accessed by clicking on them.

2. **Filter**

   Given some concrete conditions on attribute values, find data cases satisfying those conditions.

   Example: Which authors are faculty members?

   Comment: This task is inherently supported by the technique by the simple application of magnets associated with criteria. It is important to note that this is not the same as visually filtering the graph, which would mean omitting all nodes and/or edges that do not meet the desired criteria .

3. **Compute Derived Value**

   Given a set of data cases, compute an agregate numeric representation of those data cases, (e.g. average, median, and count).

   Example: What is the average number of papers of all faculty members of a given institution in a certain year?

   Comment: This task is not supported by MagnetViz.

4. **Find Extremum**

   Find data cases possessing an extreme value of an attribute over its range within the data set.

   Example: What author has the most collaborators?

   Comment: In its current version, the prototype does not directly support this task, but MagnetViz could be made to handle it by providing the user with the option of using maximum and minimum values in the comparisons of the Node/Edge Numerical Attribute Value and Degree criteria. A workaround to accomplish this task in the current version of the prototype is to guess a value that could be near the desired extremum and manually and gradually increase or decrease such value until the extremum is found (see Section 6.2).

5. **Sort**

   Given a set of data cases, rank them according to some ordinal metric.

   Example: Order authors by number of publications.

   Comment: This task is not currently supported by MagnetViz, but can be made so by extending it with a new tool, a Sorting Magnet (see Section 6.2).

6. **Determine Range**

   Given a set of data cases and an attribute of interest, find the span of values within the set.

   Example: Which authors published between 2004 and 2008?

   Comment: This task is inherently supported by MagnetViz. According to Amar et al. (2005) this task is equivalent of enumerating all unique values in a set.

7. **Characterize Distribution**

   Given a set of data cases and a quantitative attribute of interest, characterize the distribution of that attribute's values over the set.

   Example: What is the distribution of the number of publications of authors?

   Comment: This task is not currently supported by MagnetViz, but can be made to partially support it through a Sorting Magnet (see Section 6.2), which would visually organize nodes according to the amount of a given quantitative attribute they have.

8. **Find Anomalies**

   Identify any anomalies within a given a set of data cases with respect to a given relationship or expectation, e.g. statistical outliers.

   Example: Is there a professor who has not published?

   Comment: This task is partially supported, depending as much on the dataset and its semantics and on what is considered an exception by the users as on the features of MagnetViz. While some questions related to anomalies can be answered, it may not be possible to answer others.

9. **Cluster**

   Given a set of data cases, find clusters of similar attribute values.

   Example: Which professors have published in a given conference?

   Comment: Clustering is MagnetViz's specialty, being inherent to the technique, since through magnets users are able to find all nodes that fulfill their criteria.

10. **Correlate**

    Given a set of data cases and two attributes, determine useful relationships between the values of those attributes.

    Example: Is there a correlation between number of publications and whether the author is a professor or a student?

    Comment: This task is mostly supported by MagnetViz, since magnets can be used to explore the dataset in order to find correlations. In the case of the mentioned example, for instance, a magnet can be used to attract all author-nodes that have more than a given number of publications and other magnets parented to this first one can be used to see which and how many of those authors are students/professors.

11. **Find Adjacent Nodes**

    Given a node, find its adjacent nodes.

    Example: Which authors have collaborated with a given author?

    Comment: MagnetViz fully supports this task through the Path Length criterion. Node-link diagrams inherently support this as well, since they show the nodes and the edges that link them. To make adjacent nodes clearer to the user, they are always highlighted when a node is clicked on the prototype.

12. **Scan**

    Quickly review the list of items.

    Example: What authors belong to a certain institution?

    Comment: This task is currently supported by using the appropriate magnets. According to Lee et al. (2006) this task is similar to the "Retrieve Value" task but differs from it in the sense that it requires the visualization of many items at once but not necessarily the retrieval of their exact values.

13. **Set Operation**

    Given multiple sets of nodes, perform set operations on them.

    Example: What authors have collaborated with two given professors?

    Comment: Set operations are inherently supported by MagnetViz, being one of its strong points. Using the magnet-based criterion and magnet intersection along with the topology and semantic criteria, users can find sets of nodes and perform operations on them.

5.1.2 Topology-based Tasks

The topology-based tasks described by Lee et al. (2006) are listed, described and commented below.

1. **Adjacency**

    - *Find the set of nodes adjacent to a node.*

      Example: Which authors have collaborated with a particular author?

      Comment: Using MagnetViz, this task can be accomplished by using a magnet with a Path Length criterion set to attract all nodes with path length of one from the desired node.

    - *How many nodes are adjacent to a node?*

      Example: How many people have collaborated with a certain author?

      Comment: This task can be simply accomplished either by using the magnet as described above and seeing how many nodes have been attracted or by simply clicking on the node in question and seeing what is its degree.

    - *Which node has a maximum number of adjacent nodes?*

      Example: What is the author with the highest number of collaborators?

      Comment: As showed in Subsection 5.1.1, the prototype does not yet provide full support for the Finding Extremum low level task. This task can still be solved using MagnetViz, though, by using a magnet with a degree criterion and incrementally increasing the target value of such criterion until the maximum is found. An extension is already planned to be able to fully support this task (see Section 6.2).

2. **Accessibility (direct or indirect connection)**

    - *Find the set of nodes accessible from a node.*

      Examples: Who are a certain author's collaborators, their own collaborators, and so son?

      Comment: MagnetViz can be used to handle this task through a magnet with a Path Length criterion. This criterion should be set to attract all nodes with path length from the target node equal or lower to the maximum path length found for that node. The prototype does not yet provide this option, but the answer can be found nonetheless by iteratively increasing the criterion's comparison value. An extension is already planned to be able to fully support this task (see Section 6.2).

- *How many nodes are accessible from a node?*

  Example: How many authors can be reached from a certain author's collaborators, their collaborators, and so on?

  Comment: This task can be executed with MagnetViz by using the proper magnet as described above and subsequently verifying how many nodes were attracted.

- *Find the set of nodes accessible from a node where the distance is less than or equal to* n.

  Example: Which authors have collaborated with a certain author's collaborators?

  Comment: This task can be accomplished with MagnetViz by inserting a magnet with a Path Length criterion set to attract all nodes with a distance less or equal to *n* to the target node.

- *How many nodes are accessible from a node where the distance is less than or equal to* n*?*

  Example: How many authors have collaborated with a certain author's collaborators?

  Comment: This task can be simply accomplished with MagnetViz by inserting the magnet as described in the previous task and verifying the number of nodes that are attracted.

3. **Common Connection**

- *Given nodes, find a set of nodes that are connected to all of them.*

  Example: Find all authors who have collaborated with two given authors.

  Comment: There are several ways this task can be accomplished with MagnetViz. Considering the given example, for instance, one simple approach is to use just one magnet with a requirement for each author, resulting in a magnet that has to attract everyone who has collaborated with all the desired authors.

4. **Connectivity**

- *Find the shortest path between two nodes.*

  Example: Which authors comprise the shortest connection between two given authors?

  Comment: One way in which MagnetViz can be used to accomplish this task is by first using a magnet to attract one of the two nodes and then another magnet with a Path Length criterion set to attract all nodes with a distance equal or lesser than 1 from the first node. The shortest path length would be found by iteratively increasing the target value of the path length criterion until an intersection would be found between the two magnets. The nodes that comprise the shortest path would be amongst the nodes attracted by this second magnet.

- *Identify clusters (subgraphs of connected components whose nodes have high connectivity).*

  Example: Find groups of authors who frequently collaborate with each other.

  Comment: Connected subgraphs and connected components can be easily found using magnets with the Connected Subgraph criterion. High density nodes can be found by visually examining the attracted nodes or maybe using a magnet with a Degree criterion parented to the first magnet.

- *Identify connected components (maximal connected subgraphs).*

  Example: Find groups of collaborating authors that are not linked to other groups. In which of these groups every author collaborated with every other author?

  Comment: Connected subgraphs can be found with the Connected Subgraph criterion. Connected components can be found by selecting the correspondent option within the Connected Subgraph criterion dialog. This will ensure that all subgraphs attracted are maximally connected (and thus connected components).

- *Find bridges.*

  Example: By which common publication two distinct groups of frequently collaborating authors are linked?

  Comment: MagnetViz does not support this task, but can help in its execution by allowing the user to somewhat isolate the two distinct groups of nodes through magnets assigned with topology-based criteria such as Degree and Path length or, depending on the dataset, attribute-based criteria. The graph can than be visually inspected for the bridge.

- *Find articulation points.*

  Example: Who is the author through whom two distinct groups of frequently collaborating authors are linked?

  Comment: MagnetViz does not support this task, but can help in its execution by doing as described in the above task.

5.1.3 Attribute-based Tasks

The attribute-based tasks described by Lee et al. (2006) are listed, described and commented below.

1. **On the Nodes**

   - *Find the nodes having a specific attribute value.*

     Example: What authors have more than 5 publications?

     Comment: MagnetViz fully supports this task through the atribute-based criteria.

   - *Review the set of nodes.*

     Example: What authors belong to a certain institution?

     Comment: This task is basically the low level Scan task. See Subsection 5.1.1 for a description of how this can be accomplished with MagnetViz.

2. **On the Edges**

   - *Given a node, find the nodes connected only by certain types of edges.*

     Example: Which authors have collaborated with a certain author in papers published in a given conference?

     Comment: MagnetViz can be used to perform this task by inserting a magnet with two requirements. The first one would be a Path Length requirement that ensures that the attracted nodes have a path length of 1 to the given node. The second would be an Edge Has Attribute With Given Substring/Numerical Value requirement, which would establish the type of edges that the nodes should have to be attracted by the magnet.

   - *Which node is connected by an edge having the largest/smallest value?*

     Example: What author has the most cited publication?

     Comment: This task is not yet fully supported by MagnetViz, since it does not yet allow for comparisons with maximum/minimum values. If it allowed for these operations, though, this task could be executed by inserting a magnet with a Edge Has Attribute With Numerical Value set to attract the maximum/minimum value for the wanted attribute. In the current implementation, this task can be accomplished by inserting this same criterion, but iteratively increasing the compared value until the maximum/minimum is found.

5.1.4 Browsing Tasks

The browsing tasks described by Lee et al. (2006) are listed, described and commented below.

1. **Follow Path**

   - *Follow a given path.*

     Example: Find an author that has published with a certain authors that has published with yet another given author.

     Comment: MagnetViz does not directly support this task, but can help in its execution. One manner in which it can be used is by first creating a magnet that attracts the first node's neighbors and subsequently using other magnets to attract the neighbors of the neighbors, and so on. It is important to note that paths are not yet highlighted in the MagnetViz prototype.

2. **Revisit**

   - *Return to a previously visited node.*

     Example: What other authors have published with the given author of the example of the task above? And with this author's collaborators? And with these collaborators' own collaborators?

     Comment: MagnetViz does not directly support this task, but aids the user through the visual reorganization of the graph provoked by the previous queries. Depending on what magnets the user inserted, the visualization might make it easier to go back to the first node and explore another path in its tree.

5.1.5 Discussion and Preliminary Conclusions

Tables 5.1.5, 5.1.5, 5.1.5 and 5.1.5 summarize the previous subsections, listing each task and indicating whether it is supported by MagnetViz. In these tables, a task can be supported, partially supported, not directly supported or not supported. Tasks that are supported can be fully accomplished with MagnetViz. Partially supported tasks are mostly supported, but might have some variations that are unsupported or require some element beyond MagnetViz. Tasks that are not directly supported by MagnetViz cannot be accomplished with the technique's tools, but these can nonetheless aid in their execution. Tasks that are not supported are simply not aided by MagnetViz in any way.

From Subsection 5.1.1 and Table 5.1.5, it is possible to see that 8 of the low level tasks are fully supported by the MagnetViz technique, while 2 other tasks are partially supported. One of these two partially supported tasks (the Finding Extremum task) can be easily made supported by minor alterations in the MagnetViz prototype, since it is conceptually supported by the technique itself. From Subsection 5.1.1 it can also be verified that tools such as magnet boundary shapes and the ability to operate on magnets themselves (through magnet-based criteria) are natural fits for clustering and set operations.

Of the three low-level tasks that are not supported by the technique, two (namely, the Sort and Characterize Distribution tasks) could be made at least partially supported by some small additions to the prototype and/or technique. These additions would be improving some of the already available criterion types, the implementation of new criteria and the extension of the technique with new tools. Section 6.2 discusses how the current limitations of MagnetViz will be dealt with in the future, providing some details of the planned improvements.

Regarding graph specific tasks, MagnetViz also provides adequate support for the user in carrying out most tasks. Out of a total of 19 tasks, 13 are fully supported, with other 2 asks being partially supported and further 4 being aided by the technique, but not directly supported. None of the graph specific tasks is not aided in any way by the MagnetViz technique.

As can be verified in subsections 5.1.2, 5.1.3 and 5.1.4, the majority of the graph specific tasks can be easily carried out by relying simply on the placement of magnets

	Task	**Level of Support**
1	Retrieve Value	Supported
2	Filter	Supported
3	Compute Derived Value	Not Supported
4	Find Extremum	Partially Supported
5	Sort	Not Supported
6	Determine Range	Supported
7	Characterize Distribution	Not Supported
8	Find Anomalies	Partially Supported
9	Cluster	Supported
10	Correlate	Supported
11	Find Adjacent Nodes	Supported
12	Scan	Supported
13	Set Operations	Supported

Table 5.1: Low-level tasks.

	Subcategory	**Task**	**Level of Support**
1	Adjacency	Find the set of nodes adjacent to a node.	Supported
		How many nodes are adjacent to a node?	Supported
		Which node has a maximum number of adjacent nodes?	Partially Supported
2	Accessibility	Find the set of nodes accessible from a node.	Supported
		How many nodes are accessible from a node?	Supported
		Find the set of nodes accessible from a node where the distance is less than or equal to *n*.	Supported
		How many nodes are accessible from a node where the distance is less than or equal to *n*?	Supported
3	Common Connection	Given nodes, find a set of nodes that are connected to all of them.	Supported
4	Connectivity	Find the shortest path between two nodes.	Supported
		Identify clusters (subgraphs of connected components whose nodes have high connectivity).	Supported
		Identify connected components (maximal connected subgraphs).	Supported
		Find bridges.	Not Directly Supported
		Find articulation points.	Not Directly Supported

Table 5.2: Topology-based tasks.

	Subcategory	Task	Level of Support
1	On the Nodes	Find the nodes having a specific attribute value.	Supported
		Review the set of nodes.	Supported
2	On the Edges	Given a node, find the nodes connected only by certain types of edges.	Supported
		Which node is connected by an edge having the largest/smallest value?	Partially Supported

Table 5.3: Attribute-based tasks.

	Subcategory	Task	Level of Support
1	Follow Path	Follow a given path.	Not Directly Supported
2	Revisit	Return to a previously visited node.	Not Directly Supported

Table 5.4: Browsing tasks.

with the proper combination of topology, attribute and magnet-based criteria, followed by a visual inspection of the layout obtained with the modifications. This makes the technique particularly well suited for topology and attribute-based tasks.

Out of 13 topology-based tasks, it can be verified from Subsection 5.1.2 and Table 5.1.5 that 10 are fully supported, while 1 being partially supported and 2 are not directly supported. From Subsection 5.1.3 and Table 5.1.5 it can also be seen that out of 4 tasks, 3 are fully supported and 1 is partially supported.

Since they can be inserted to make sure that the nodes that fulfill certain criteria are within a certain region, magnets make it easier for the users to find nodes, providing the visualization with some node position predictability. Therefore, browsing and overview tasks are also aided by MagnetViz, though not directly supported.

One interesting aspect of MagnetViz is that it can be used to easily explore graph datasets by building queries through the specification of magnets and their criteria, and performing set operations on them, becoming thus an intuitive and simplified alternative to query languages or filtering operations, which can be too complex for most end-users, who do not necessarily have advanced programming and computer skills.

For the tasks that MagnetViz is unable to cover, a possible solution is simply combining it with other interaction and navigation techniques, such as fish-eye-like visualizations, overview windows, node search, etc.

5.2 Case Study on Social Network Visualization

To evaluate MagnetViz within a practical context, it was decided to perform a case study on social network visualization, due to this field's growing relevance. There has been a great surge in popularity of social networking applications in the past few years, in part due to websites such as Facebook, MySpace, Orkut and Flickr. This has generated a greater interest in the analysis of these networks, which has proved to be necessary in many fields, including social and behavioral sciences, economy and marketing (WASSERMAN; FAUST; IACOBUCCI, 1994). While this can be done using statistical studies and other procedures that do not have a visual output, it is often the case that producing a visual representation of the network might make the same information more accessible and reveal new traits and relationships that were previously undetectable. With all of this in mind, as a case study it was chosen to visualize the co-authorship graph generated from all the works published in 2007 by the professors and students of the Informatics Institute of the Federal University of Rio Grande do Sul (UFRGS) and their external collaborators.

The graph built with the UFRGS dataset contains 474 nodes and 1252 edges. Each node represents an author while each edge represents all the publications between two authors. From the GraphML file, a graph is built where nodes and edges have several attributes. Each node contains:

- author's name;
- category (whether they are faculty members, students or external collaborators).
- degree (number of edges connected to it);
- total number of publications;
- number of publications in conference proceedings;
- number of publications in journals;
- number of publications in books;

Each edge contains:

- the id of their two nodes;
- the years of the two authors' common publications;
- the types of the publications (journal, conference proceedings or book);
- the number of common publications.

A list of query tasks pertinent to this dataset was compiled for this case study by UFRGS database researchers and the prototype was first put to test by being used to answer them. User tests were then performed, with the users having to answer the questions using the prototype and subsequently a questionnaire concerning their opinions about the technique and its usefulness in this particular context.

Subsection 5.2.1 contains a description of known limitations of the prototype and dataset. Subsection 5.2.2 shows the questions and a description of how they can be answered (or not) using the technique, while Subsection 5.2.3 contains a description of the tests performed with users and the results obtained from them.

5.2.1 Limitations of the Prototype and Dataset

It is important to note that while all the questions examined in Subsection 5.2.2 can be answered with the technique itself, the prototype does not support some of them. Since the prototype was used as is, without being adapted in any way for the specific case of co-authorship network visualization, some of these questions would require small adaptations of the prototype software to be answered with the proposed technique. Whenever that is the case, we point out exactly why the generic version of the technique cannot be used and which modifications are needed to better reach a solution.

As of this writing the prototype does not fully support some low-level tasks (including a few that would be conceptually supported by the technique itself). These tasks could have been dealt with by some types of criteria and/or tools, and support for them would have proven useful (and occasionally necessary) to the answering of some of the questions. Whenever this is the case, it is also pointed out. Section 6.2 described the modifications that would allow these to be handled.

It is also worth noting that some potentially interesting questions could not be asked due to limitations of the dataset (i.e. lack of information on conferences and paper titles, dataset containing only information on one year, etc.) and the prototype itself. Supposing, for example, the dataset also contained information from other years and the conference of each paper, and that we wanted to obtain all the authors that have published on certain conferences in a certain year, we could not directly obtain an answer with our current prototype. The prototype does not contain a mechanism for linking a conference to a year - they are just two strings containing conference names and years separated by commas. Therefore, we can obtain all the authors who have published on the desired conferences and that have publications in the desired year, but not necessarily all the authors who have published in those conferences in that specific year.

Chapter 6 addresses some of this limitations by describing the plans for future development of the MagnetViz technique.

5.2.2 Questions Concerning the UFRGS Co-Authorship Network

Below is a listing of the proposed questions and a description of how each can be answered using the technique. It is important to note that some questions might be answered in many different ways aside from the ones described below.

1. *What is the faculty member with the highest amount of collaborators?*

   The number of collaborators of any author is their node's degree on the graph. Therefore, to find out what faculty member has the largest amount of collaborators we can insert on the scene a magnet that attracts the node with the highest possible degree amongst the nodes of faculty members. To do so, we first create a magnet that has a requirement to attract all nodes that have the attribute "category" with the value "faculty" and then create another magnet parented to the first one that has a requirement to attract the node with the highest possible degree. The answer, which can be seen in can be seen in Figure 5.1, is professor Ricardo Augusto da Luz Reis, with 51 collaborators.

   The current implementation lacks a criterion/requirement for the attraction of the node with the highest possible degree, so in the prototype one has to use a regular degree criterion and keep editing it until the largest is found. Still, the highest possible degree criterion is a small modification that will be implemented for a subsequent version of the tool.

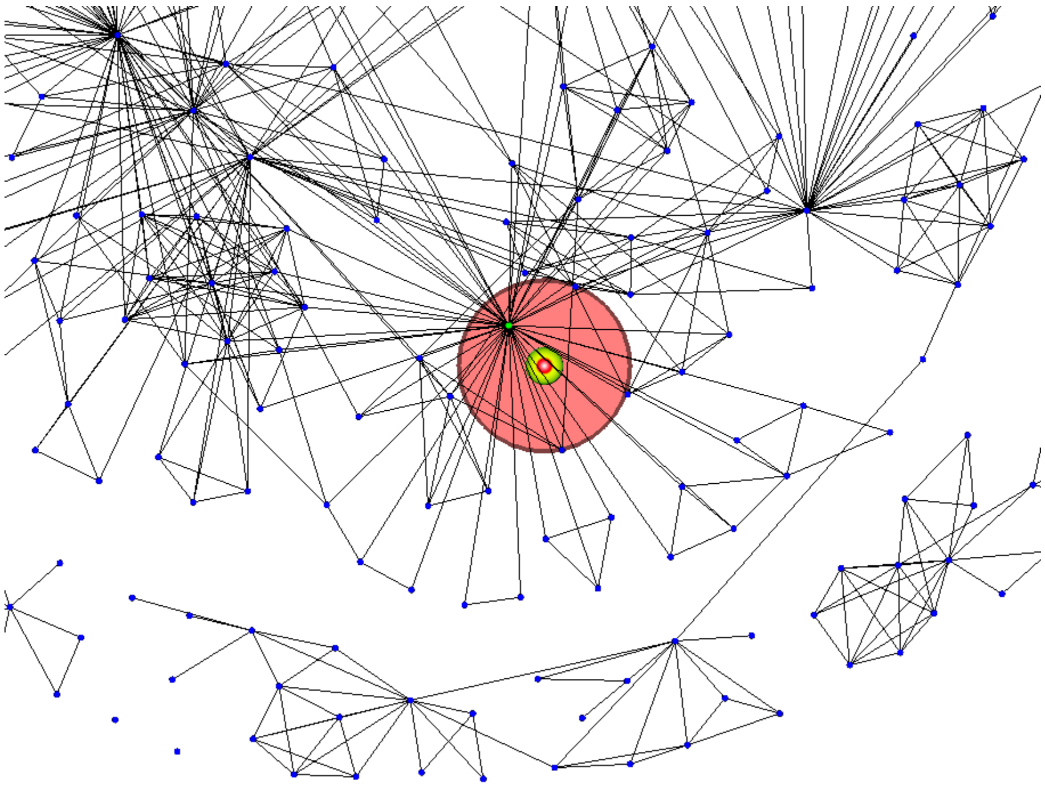


Figure 5.1: Faculty member with the highest amount of collaborators.

2. *Which is the faculty member with the highest amount of external collaborators?*

   To discover who amongst all faculty members has the largest number of external collaborators, we could create a magnet that attracts all faculty members and another one that attracts all external collaborators. We could then create yet another magnet parented to one that attracts faculty members and set it to attract the node with the highest possible amount of edges to nodes that are attracted by the external collaborators magnet.

   The prototype does not include yet a criterion/requirement for the attraction of nodes that have a certain number of edges (within a given range or the highest possible) to nodes that are attracted by a certain magnet. Using this same implementation, the answer to this question could be obtained by changing the dataset to include as a node attribute the number of collaborators of each category.

3. *Which students have worked with both professors Carla Freitas and Luciana Nedel?*

   To answer this question, we can first find out all the collaborators that the two professors have in common and then see, amongst these, who are students. We can do this by creating a magnet that attracts all of the first professor's collaborators by using a requirement that targets all nodes that have path length of 1 to this professor. We then create a similar magnet for the second professor - with the difference that this one is parented to the first's magnet. This way we find all of the first's professor's collaborators that are also collaborators of the second, and have "student" as their "category" attribute.

   In Figure 5.2, Luciana Nedel's collaborators are the green nodes while all of her collaborators who are students and also collaborate with Carla Freitas are inside the boundary shape, being Marta Villamil the answer for the question.

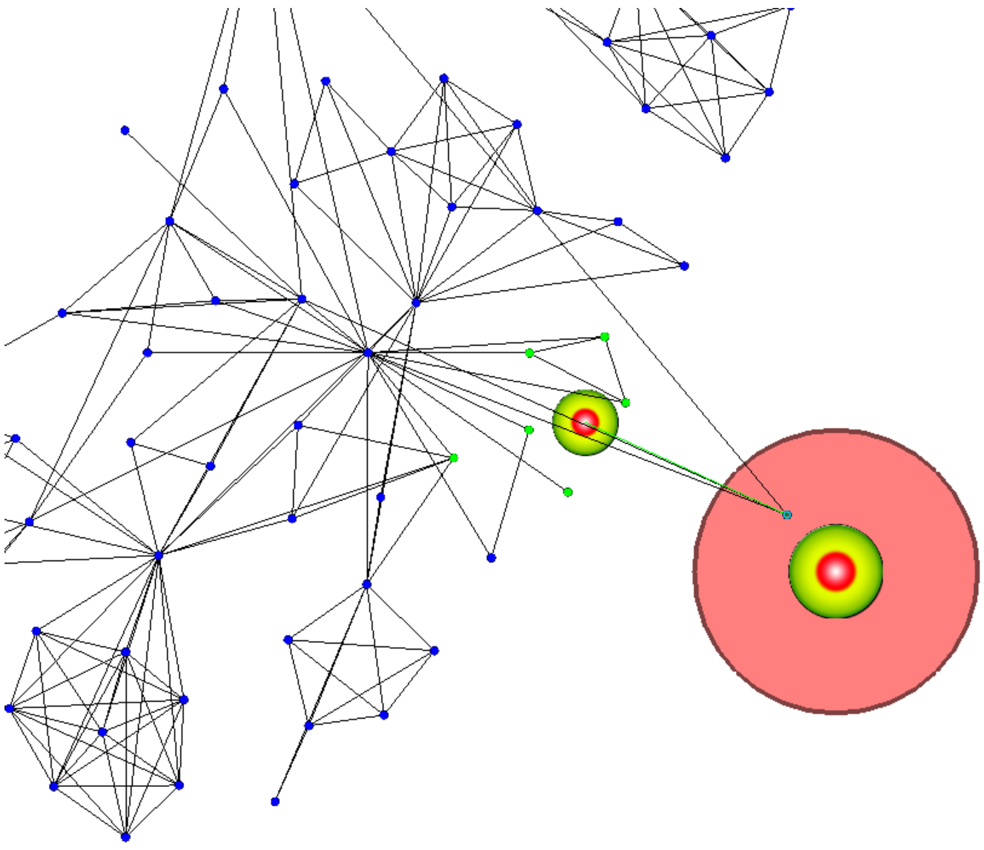


Figure 5.2: Common student collaborators of professors Carla Freitas and Luciana Nedel.

4. *Which is the student with the highest amount of publications in journals?*

   We can find the answer to this question by creating a magnet with two requirements: one to attract all nodes with the "category" attribute being a student and the other to attract the node with the highest value for the attribute "journals." This would result in the single node that represents the student with the most publications in journals.

   Since in the current implementation we do not have an attraction criterion/requirement for the node that has the highest amount of a certain attribute, we can answer this question by finding all the students who have at least one publication in journals and keep refining our search from there by then looking for students who have at least two publications, and so on. The answer to this question is the student Luciano Volcan Agostini, who can be seen in Figure 5.3 as the node inside the boundary shape.

5. *Which are the researchers that have publications in both conference proceedings and journals?*

   This question can be answered by simply inserting a magnet with two requirements, one for all nodes that have the attribute "journals" higher than 0 and the other analogous, but for the attribute "proceedings." The answer to this question can be seen in Figure 5.4 as the nodes inside the boundary shape.

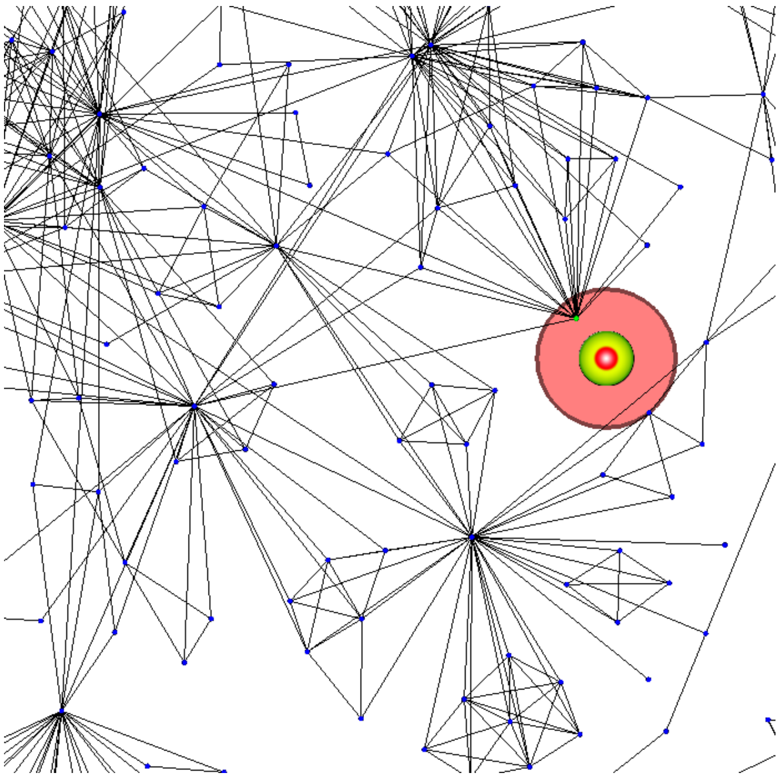


Figure 5.3: Student with the highest amount of publications in journals.

6. *What external collaborators have worked with more than one faculty member?*

   To answer this question using the technique itself, we can begin by creating one magnet with a requirement that attracts all nodes that have the attribute "category" with the value "professor". We could then insert another one with requirements that target the nodes that have the attribute "category" with the value "external" and that share more than one edge with a node attracted by the first magnet.

   In its current state the prototype doesn't support this question. To be able to answer it, it would need a criterion/requirement for the attraction of nodes that have a certain number of edges (within a given range) to nodes attracted by another magnet.

7. *What are the groups of less than 20 authors who have worked only with other members of the same group?*

   We can find the answer to this question by simply creating a magnet that attracts connected subgraphs with less than 20 nodes. The answer can be seen in Figure 5.5.

8. *What faculty members have never worked with any student?*

   To answer this question, we can create a magnet that has a requirement for all nodes with value "student" for the attribute "category" and another one that has two requirements, one for all nodes with value "professor" for the attribute "category" and another for all nodes that have 0 as the number of edges shared with nodes attracted by the other magnet. Even though this question can be answered with the technique, it is not yet supported by the prototype, for the same reason as in question 6.

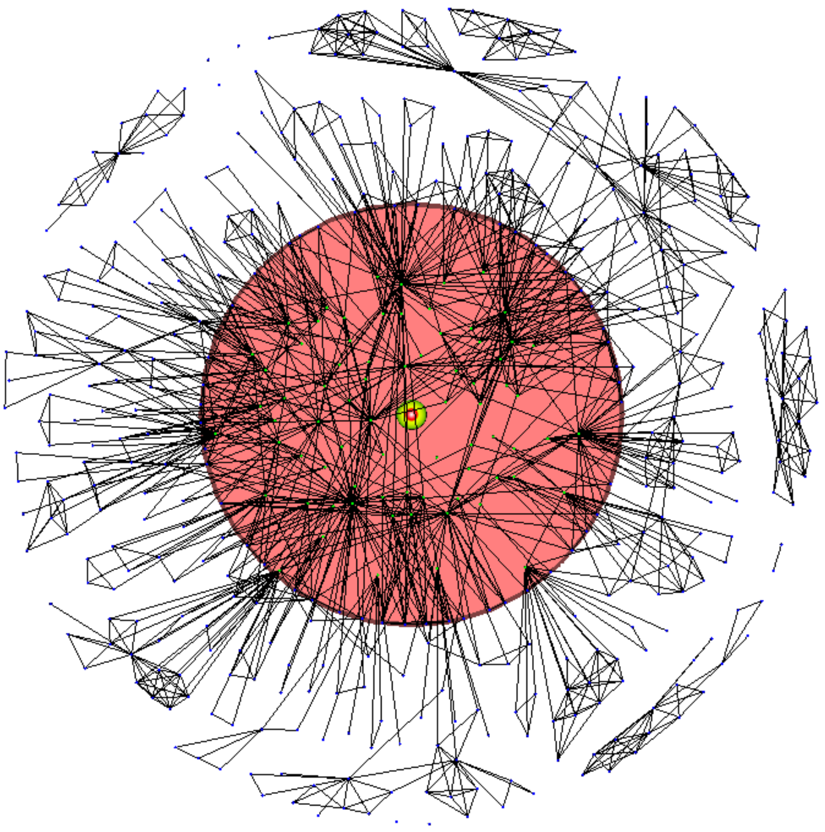


Figure 5.4: Researchers who published in both conference proceedings and jounals.

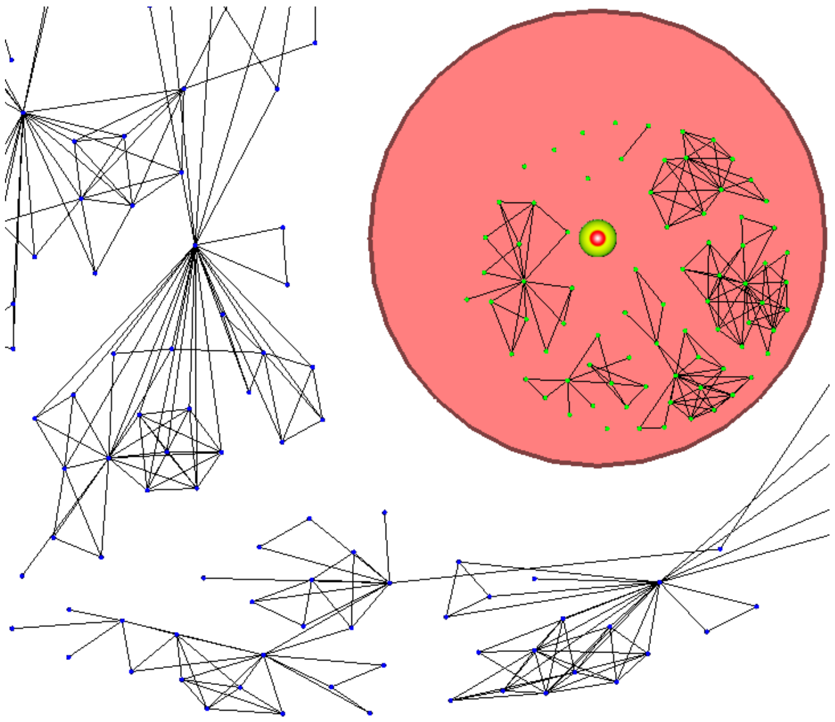


Figure 5.5: Groups of less than 20 authors.

5.2.3 User Tests for Subjective Evaluation

The technique was tested by a group of 14 subjects with ages between 20 and 50. Most alleged having little experience with visualization in general and progressively less with visualization of graphs social networks, with only one claiming to be very experienced with graph and social network visualization. The subjects were all students and faculty members of the Informatics Institute of the UFRGS, therefore being potential users for the technique with the test dataset.

The experiment began with a brief presentation of the technique. To better familiarize the subjects with MagnetViz, they were given simple tasks described step-by-step that guided them through most of the tools and functionalities of the prototype.

After this training session, they were asked to use the prototype to answer the questions described in Subsection 5.2.2 and were subsequently given a questionnaire (Appendix B) about their opinions on several aspects of the technique.

The questionnaire was comprised of 35 positive affirmations and the subjects had to indicate whether and how much they agreed with what was stated by ticking in a corresponding check box. If they had not used the feature referred to by an affirmation, users could tick a box that indicated so.

Table 5.5 shows the affirmations and the average, mode and standard deviation of the level of agreement indicated by the subjects, with the an average value closer to 5 indicating a higher agreement. The table is sorted first by average and then by mode. Note that table 5.5 is only partial, showing only the statements with the highest level of agreement - the full table can be found in Appendix A.

As a technique, MagnetViz is comprised mostly of the magnet metaphor, their configuration through requirements and criteria, and their use for layout manipulation. It can be said, then, that the key features of the technique are magnets, the application of requirements and criteria, and boundary shapes. Based on tables 5.5, A.1, A.2 and A.3 (see Appendix A), it is possible to see that these key elements are all validated by the subjects in terms of their concept and usefulness.

From the grades given by the subjects, it can be observed that the magnet metaphor is clear and mostly useful for graph exploration (affirmations 5, 6, 33, 34 and 35). The subjects were easily able to understand the difference between criteria and requirements (affirmation 13) and found those useful for the exploration of the graph (affirmations 15, 16, 17, 18, 19, 20, 21, 22, 23, 24 and 25). The concept of boundary shapes was also easily understood and found to be useful (affirmations 7 and 8). They also gave a high grade for the usefulness of MagnetViz in the visualization of social networks (affirmation 34). While this user evaluation in effect validates MagnetViz's approach for graph visualization, at least conceptually and within the context of social networks exploration, it is also quite clear that the technique still has some details that need to be better worked out.

As can be seen from the results, in general subjects somewhat disliked the way intersections are handled (affirmations 26, 27, 29), even though they found it to be reasonably useful for the exploration of the graph (affirmation 28). The special icon for intersection nodes received just average marks (affirmation 26) and while most have found the colored lines useful (affirmation 27), there seems to be a reasonable margin for improvement. Also, gravity spots were found by subjects to be of limited utility - users understood what they stood for, but found no use for them (affirmation 12).

There also seems to be some margin for improvement in areas not directly related to the technique, but essential for its application and the overall user-experience, such as layout and interface elements (affirmations 2, 3, 4, 9, 30, 31 and 32). Concerning the

#	Affirmations	Average	Mode	Standard Deviation
5	I clearly understand what a magnet can be used for.	4.92	5.00	0.27
24	The criterion of type "Node Textual Attribute Value" is useful.	4.92	5.00	0.27
23	The criterion of type "Node Numerical Attribute Value" is useful.	4.92	5.00	0.27
21	The criterion of type "Node Degree" is useful.	4.92	5.00	0.27
13	I clearly understand the difference between requirements and criteria.	4.92	5.00	0.27
15	Criteria and requirements are useful for the exploration of the graph.	4.77	5.00	0.43
6	Magnets are useful for the exploration of the graph.	4.62	5.00	0.64
7	I clearly understand what a boundary shape can be used for.	4.54	5.00	0.65
34	The technique is useful for the exploration of social networks.	4.38	5.00	0.96
20	The criterion of type "Magnets" is useful.	4.38	5.00	0.76
8	Boundary shapes are useful for layout reorganization.	4.31	5.00	0.74
25	The criterion of type "Path Length" is useful.	4.31	5.00	1.39
16	The criterion of type "Connected Subgraph" is useful.	4.31	5.00	1.01
35	Manipulating the graph makes it possible to discover situations that were previously unknown.	4.15	4.00	0.86
33	Manipulating the graph makes it possible to explore it and discover relationships between the nodes.	4.08	4.00	0.62
4	The alterable properties of the layout are useful for controlling the physics simulation that determines the layout of the graph.	4.00	5.00	1.04

Table 5.5: Statistics about the subjects' level of agreement with statements related to characteristics of the technique.

layout, the great majority of subjects found the initial layout clear, but thought general traits of nodes and edges could be made clearer. They had the same opinion about the visual results of a query after the layout is modified with the provided tools. Concerning interface elements, while an overwhelming majority of users gave high marks for ease of requirement and criteria insertion and editing, a couple of subjects gave very low grades. The layout properties that can be altered and the function and use of the boundary shape's gravity multiplier were also not very clear for the subjects, though the former were found by the subjects to be important for the control of the physics simulation that determines the layout.

Since the technique was user-tested only within the context of social network visualization, it is only possible to evaluate how the criterion types fared within this specific context and dataset. Also, the questions the subjects had to answer not necessarily required the use of all criterion types. Given that, subjects in general gave high marks for node-related attribute and topology-based criteria (affirmations 21, 23 and 24), with the magnet criterion, which they had to use to perform set operations, (affirmation 20) receiving good grades as well. Topology-based criteria in general, such as path length and connected subgraph, also fared well (affirmations 16 and 25). Edge-related criteria got lower marks, as they were less necessary to answer the given questions (affirmations 17, 18 and 19). Overall the users thought the provided criteria types were mostly adequate for the tasks they were given (affirmation 14), but in the comments they provided after the test they expressed the need to have a few more varieties to make the unanswerable questions possible. Some of these concerns were already expected and solutions for them were already foreseen (see Subsection 5.2.1 and Chapter 6).

Aside from filling the questionnaire, users were asked to give comments and suggestions for the future improvement of the technique. Several subjects pointed out bugs found in the prototype (especially in the selection of objects) and gave suggestions regarding interface elements, such as the way node and edge attributes are shown. One subject pointed out the layout changes too much after a query is executed, confusing the user's mental map of the graph. Such comments are addressed in Chapter 6.

6 CONCLUSIONS AND FUTURE WORK

6.1 Summary and Conclusions

This work presented MagnetViz, a physics-based technique for the interactive manipulation of graph visualizations. The main contribution of this work was the innovative approach taken when dealing with the problem of graph visualization and manipulation. While most techniques are concerned with visualizing a static, previously generated layout, MagnetViz allows users to adapt a layout to better fit their needs by giving them tools to establish regions for nodes that fulfill desired semantic or topological criteria.

The technique was built upon the physics metaphor of the popular force-directed graph layout algorithms. They were extended to support the concept of magnets, which are the key element of MagnetViz. Magnets are inserted into the scene by the user and associated to attraction criteria. When a magnet's attraction force is set, the physics systems re-evaluated the layout, changing it so that this magnet's attracted nodes move closer to it. To make visualization clearer, users can define regions where the attracted nodes should be through the use of boundary shapes. These are simply geometric shapes in which attracted nodes are forces to stay and from which not attracted nodes are made to leave.

MagnetViz was initially presented in detail and subsequently evaluated. This evaluation happened in two stages, with the first consisting of verifying how it fared against a task taxonomy developed for graph visualization applications. While it did not apply directly to all general low-level tasks, it could directly handle most of the higher level, graph-specific tasks, being an especially natural fit for the ones that involve clustering and set operations.

The second stage of the evaluation process consisted of putting MagnetViz into a practical setting by performing a case study. For such it was decided to apply it within the ever more relevant context of social network visualization. The co-authorship graph generated by the 2007 publications of the UFRGS professors and students and their external collaborators was chosen as dataset, and questions were elaborated about this network to be answered through the use of the technique. During tests it was found that two questions were not directly answerable with the technique for two reasons: the prototype was not adapted for this specific context and some planned features that would support these questions were still not implemented.

To further validate the technique, a group of 14 subjects was asked to use MagnetViz to answer the same questions. The subjects subsequently received a questionnaire which contained affirmations about the relevance of the technique itself and the usability of its features which they had to rate according to their level of agreeance with them. In general, users seem to have approved the technique and its usefulness, finding the metaphor intuitive and relevant to social network and graph visualization. From the answers to the questionnaire, though, while it could be seen that there are still some rough spots that need to be worked out (especially when it comes to the developed prototype) it was found that conceptually the technique itself was mostly approved by the users that tested it. The only shortcoming that involved the core technique that users claimed to have experienced was regarding the way magnet intersections was handled. Intersections were found to be dealt in a not very intuitive manner and the visual result of their treatment was not found to be very clear.

The technique presented in this work is by no means adequate to all domains of graph visualization. MagnetViz, however, constitutes an important step, contributing with an innovative approach that can be verifiably useful within the real, practical context of social network analysis. This same approach might also be useful for other areas, such as visualizing software structures (e.g. function call graph) or the Wikipedia (e.g. the graph of citations between articles). It is thus another potentially useful tool in the arsenal of software designers that need to tackle the problem of interactive graph visualization.

While there are strong indications that the basic concept of MagnetViz is valid, there are still aspects that can be improved and the technique itself has potential to be extended in different ways. Section 6.2 details the future work that is already foreseen.

6.2 Future Work

Both the prototype and the MagnetViz technique itself can be improved and extended in many different ways. The planned future work includes both extensions to the technique and improvement of features that were already part of it, as well as continued development and amelioration of the prototype.

Amongst planned extensions to the technique are the following tools:

- **Sorting Magnet**

  The sorting magnet would be a special kind of magnet that would place attracted nodes in orbits around it based on their (or one of their edge's) value for of a certain attribute. The highest the value, the closer to the magnet it stays.

- **Collector**

  The collector would be an improvement over the currently available Gravity Spots. It would be a magnet with boundary shape that attracts all nodes that cross the shape's boundary. These nodes would be then kept inside the boundary, being thus effectively collected by this magnet. An option would be to allow the user to associate requirements and criteria to selectively collect nodes as well, differing from the regular magnet in the sense that only nodes over which the user positions the boundary shape would be effectively attracted.

- **Selection Magnet**

  This would be a special kind of magnet that would not have criteria and requirements associated with it. Instead, the user would be able to manually select the nodes that will be attracted.

- **Probe**

  Aimed at searching the graph for nodes/edges that fulfilled certain criteria, the probe would work as a selective fish-eye tool. Users would associate requirements and criteria to a probe, much like they do with a magnet, and would move it around the workspace to find nodes/edges in desired regions that have the desired properties.

Aside from these tools that would increment the technique, from the user evaluation it was clear that way the manner magnet intersections is handled has to be improved. Another useful amelioration would be to allow a list of exceptions for each magnet, so that users can chose certain nodes and edges to be ignored by the magnet even if they fulfill the criteria and requirements.

To complete the fulfillment of the graph visualization tasks as described in Section 5.1 the following criterion types should be altered in the prototype:

- **Node/Edge Numerical Attribute Value**

  This criterion should be changed to allow for comparisons with the maximum and minimum values of a certain numerical attribute of all the graph's nodes and of the nodes attracted by the current magnet's parent.

- **Node Degree**

  This criterion should be changed to allow for comparisons with the maximum and minimum degrees of all the graph's nodes and of the nodes attracted by the current magnet's parent.

For the two criterion types shown above it would also be useful to allow the user to chose a tolerance for the target value (a range from the target attribute value within which the node's value should be) and to set the nodes with the closest possible value to be attracted if the targeted ones are not found.

Some tasks could not be accomplished in the evaluation and case study because some potentially useful criteria were not available in the prototype. Therefore, the following new types should be added in order to support these tasks:

- **Path Length to Magnet**

  This criterion type would work analogously to the "Path Length" type. Instead of being used to attract the nodes that are within a certain path length from another node, it would attract the nodes that are within a certain path length from their closest connected node that is attracted by a target magnet.

- **Number of Edges to a Magnet's Nodes**

  This criterion type would attract all nodes that have a number of edges to nodes attracted by a specified magnet within a certain user-specified range.

- **Connected Groups**

  This criterion would attract groups of nodes that are highly connected with each other but are not necessarily disconnected from the rest of the graph. This would allow for the easier finding of clusters, bridges and articulation points.

The prototype needs as well to have its interface improved. Certain features should be made clearer and easier to use and some elements should be made more easily accessible, such as the properties of scene objects (magnets, nodes edges, etc.), which could appear in a tooltip whenever the user positions the cursor over one of them. The possibility of showing and hiding node and edge labels would also improve the visualization, as well as node and edge filtering. Other layout techniques should be experimented with, also, and used as basis for potential new algorithms better suited for MagnetViz.

All the mentioned additions and modifications should be thoroughly evaluated, through the graph visualization task taxonomy, case studies and user tests. It is important to put the MagnetViz technique within a practical context to verify its applicability and usefulness for real applications and further refine it.

REFERENCES

ABELLO, J.; VAN HAM, F. Matrix Zoom: a visual interface to semi-external graphs. In: INFOVIS '04: PROCEEDINGS OF THE IEEE SYMPOSIUM ON INFORMATION VISUALIZATION, 2004, Washington, DC, USA. **Proceedings...** IEEE Computer Society, 2004. p.183–190.

ABELLO, J.; VAN HAM, F.; KRISHNAN, N. ASK-GraphView: a large scale graph visualization system. **Visualization and Computer Graphics, IEEE Transactions on**, [S.l.], v.12, n.5, p.669–676, Sept.-Oct. 2006.

ABELLO, J.; KORN, J. MGV: a system for visualizing massive multidigraphs. **IEEE Transactions on Visualization and Computer Graphics**, Piscataway, NJ, USA, v.8, n.1, p.21–38, 2002.

AMAR, R.; EAGAN, J.; STASKO, J. Low-Level Components of Analytic Activity in Information Visualization. In: INFOVIS '05: PROCEEDINGS OF THE PROCEEDINGS OF THE 2005 IEEE SYMPOSIUM ON INFORMATION VISUALIZATION, 2005, Washington, DC, USA. **Proceedings...** IEEE Computer Society, 2005. p.15.

ARCHAMBAULT, D.; MUNZNER, T.; AUBER, D. Grouse: feature-based, steerable graph hierarchy exploration. In: EUROGRAPHICS/ IEEE-VGTC SYMPOSIUM ON VISUALIZATION, 2007. **Proceedings...** Eurographics Association, 2007. p.67–74.

ARCHAMBAULT, D.; MUNZNER, T.; AUBER, D. GrouseFlocks: steerable exploration of graph hierarchy space. **IEEE Transactions on Visualization and Computer Graphics**, Piscataway, NJ, USA, v.14, n.4, p.900–913, 2008.

ARIS, A.; SHNEIDERMAN, M.-B. Network Visualization by Semantic Substrates. **IEEE Transactions on Visualization and Computer Graphics**, Piscataway, NJ, USA, v.12, n.5, p.733–740, 2006.

BACHMAIER, C. . Gravisto: graph visualization toolkit. , [S.l.], v.3383, p.502–503, 2005.

BALZER, M.; DEUSSEN, O.; LEWERENTZ, C. Voronoi treemaps for the visualization of software metrics. In: SOFTVIS '05: PROCEEDINGS OF THE 2005 ACM SYMPOSIUM ON SOFTWARE VISUALIZATION, 2005, New York, NY, USA. **Proceedings...** ACM, 2005. p.165–172.

BARNES, J.; HUT, P. A hierarchical O(N log N) force-calculation algorithm. **Nature**, [S.l.], v.324, n.6096, p.446–449, December 1986.

BATINI, C.; FURLANI, L.; NARDELLI, E. What is a Good Diagram? A Pragmatic Approach. In: FOURTH INTERNATIONAL CONFERENCE ON ENTITY-RELATIONSHIP APPROACH, 1985, Washington, DC, USA. **Proceedings...** IEEE Computer Society, 1985. p.312–319.

BATTISTA, G. D.; EADES, P.; TAMASSIA, R.; TOLLIS, I. G. **Graph Drawing**. Upper Saddle River, NJ, USA: Prentice Hall, 1999.

BERTAULT, F. A force-directed algorithm that preserves edge-crossing properties. **Inf. Process. Lett.**, Amsterdam, The Netherlands, v.74, n.1-2, p.7–13, 2000.

BIEDL, T. C.; KANT, G. A Better Heuristic for Orthogonal Graph Drawings. In: ESA ’94: PROCEEDINGS OF THE SECOND ANNUAL EUROPEAN SYMPOSIUM ON ALGORITHMS, 1994, London, UK. **Proceedings...** Springer-Verlag, 1994. p.24–35.

CARD, S. K.; MACKINLAY, J. D.; SHNEIDERMAN, B. (Ed.). **Readings in information visualization**: using vision to think. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1999.

CARRIERE, J.; KAZMAN, R. Research report: interacting with huge hierarchies: beyond cone trees. In: INFOVIS ’95: PROCEEDINGS OF THE 1995 IEEE SYMPOSIUM ON INFORMATION VISUALIZATION, 1995, Washington, DC, USA. **Proceedings...** IEEE Computer Society, 1995. p.74.

COHEN, R. F.; EADES, P.; LIN, T.; RUSKEY, F. Three-Dimensional Graph Drawing. In: GRAPH DRAWING, 1994. **Proceedings...** Springer, 1994. p.1–11. (Lecture Notes in Computer Science, v.894).

CUI, W. et al. Geometry-Based Edge Clustering for Graph Visualization. In: IEEE TRANSACTIONS ON VISUALIZATION AND COMPUTER GRAPHICS (PROCEEDINGS VISUALIZATION / INFORMATION VISUALIZATION 2008), 2008. **Proceedings...** [S.l.: s.n.], 2008. v.14, n.6.

DAVIDSON, R.; HAREL, D. Drawing graphs nicely using simulated annealing. **ACM Trans. Graph.**, New York, NY, USA, v.15, n.4, p.301–331, 1996.

DWYER, T.; KOREN, Y. DIG-COLA: directed graph layout through constrained energy minimization. In: INFOVIS ’05: PROCEEDINGS OF THE PROCEEDINGS OF THE 2005 IEEE SYMPOSIUM ON INFORMATION VISUALIZATION, 2005, Washington, DC, USA. **Proceedings...** IEEE Computer Society, 2005. p.9.

EADES, P. A. A heuristic for graph drawing. **Congressus Numerantium**, [S.l.], v.42, p.149–160, 1984.

EADES, P. Drawing Free Trees. In: BULLETIN OF THE INSTITUTE FOR COMBINATORICS AND ITS APPLICATIONS, 1992. **Proceedings...** [S.l.: s.n.], 1992. p.10–36.

EIGLSPERGER, M.; FEKETE, S. P.; KLAU, G. W. Orthogonal graph drawing. , London, UK, p.121–171, 2001.

FRICK, A.; LUDWIG, A.; MEHLDAU, H. A Fast Adaptive Layout Algorithm for Undirected Graphs. In: GD ’94: PROCEEDINGS OF THE DIMACS INTERNATIONAL WORKSHOP ON GRAPH DRAWING, 1995, London, UK. **Proceedings...** Springer-Verlag, 1995. p.388–403.

FRISHMAN, Y.; TAL, A. Multi-Level Graph Layout on the GPU. **Visualization and Computer Graphics, IEEE Transactions on**, [S.l.], v.13, n.6, p.1310–1319, Nov.-Dec. 2007.

FRUCHTERMAN, T. M. J.; REINGOLD, E. M. Graph drawing by force-directed placement. **Softw. Pract. Exper.**, New York, NY, USA, v.21, n.11, p.1129–1164, 1991.

FURNAS, G. W. **The Fisheye View**: a new look at structured files. [S.l.]: Bell Laboratories, 1981.

GAJER, P.; GOODRICH, M. T.; KOBOUROV, S. G. A multi-dimensional approach to force-directed layouts of large graphs. **Comput. Geom. Theory Appl.**, Amsterdam, The Netherlands, v.29, n.1, p.3–18, 2004.

GAJER, P.; KOBOUROV, S. G. Grip: graph drawing with intelligent placement. **Journal of Graph Algorithms and Applications**, [S.l.], v.6, p.2002, 2000.

GANSNER, E. R.; KOUTSOFIOS, E.; NORTH, S. C.; VO, K.-P. A Technique for Drawing Directed Graphs. **IEEE Trans. Softw. Eng.**, Piscataway, NJ, USA, v.19, n.3, p.214–230, 1993.

GANSNER, E. R.; NORTH, S. C. An open graph visualization system and its applications to software engineering. **Softw. Pract. Exper.**, New York, NY, USA, v.30, n.11, p.1203–1233, 2000.

GANSNER, E. R.; NORTH, S. C.; VO, K. P. DAG—a program that draws directed graphs. **Softw. Pract. Exper.**, New York, NY, USA, v.18, n.11, p.1047–1062, 1988.

GEORGII, J.; ECHTLER, F.; WESTERMANN, R. Interactive Simulation of Deformable Bodies on GPUs. In: SIMULATION AND VISUALISATION 2005, 2005. **Proceedings. . .** [S.l.: s.n.], 2005. p.247–258.

GHONIEM, M.; FEKETE, J.-D.; CASTAGLIOLA, P. A Comparison of the Readability of Graphs Using Node-Link and Matrix-Based Representations. In: INFOVIS ’04: PROCEEDINGS OF THE IEEE SYMPOSIUM ON INFORMATION VISUALIZATION, 2004, Washington, DC, USA. **Proceedings. . .** IEEE Computer Society, 2004. p.17–24.

GHONIEM, M.; JUSSIEN, N.; FEKETE, J.-D. VISEXP: visualizing constraint solver dynamics using explanations. In: FLAIRS’04: SEVENTEENTH INTERNATIONAL FLORIDA ARTIFICIAL INTELLIGENCE RESEARCH SOCIETY CONFERENCE, (MIAMI BEACH, FL, 2004), AAAI, 2004. **Proceedings. . .** press, 2004.

GODSIL, C.; ROYLE, G. **Algebraic Graph Theory**. New York, NY: Springer-Verlag, 2001.

GRAPHML. [S.l.:s.n.: 200-]. Available in: <http://graphml.graphdrawing.org>. Accessed on: 7 Oct. 2008.

HENDLEY, R. et al. Case study: narcissus: visualising information. **Information Visualization, IEEE Symposium on**, Los Alamitos, CA, USA, v.0, p.90, 1995.

HENRY, N.; BEZERIANOS, A.; FEKETE, J.-D. Improving the Readability of Clustered Social Networks using Node Duplication. In: INFOVIS '08: PROCEEDINGS OF THE 2008 IEEE SYMPOSIUM ON INFORMATION VISUALIZATION (INFOVIS '08), 2008, Washington, DC, USA. **Proceedings...** IEEE Computer Society, 2008.

HENRY, N.; FEKETE, J.-D. MatrixExplorer: a dual-representation system to explore social networks. **IEEE Transactions on Visualization and Computer Graphics**, Piscataway, NJ, USA, v.12, n.5, p.677–684, 2006.

HENRY, N.; FEKETE, J.-D. Matlink: enhanced matrix visualization for analyzing social networks. In: INTERNATIONAL CONFERENCE INTERACT, 2007. **Proceedings...** [S.l.: s.n.], 2007.

HENRY, N.; FEKETE, J.-D.; MCGUFFIN, M. NodeTrix: hybrid representation for analyzing social networks. **ArXiv e-prints**, [S.l.], v.705, May 2007.

HERMAN, I.; SOCIETY, I. C.; MELANÇON, G.; MARSHALL, M. S. Graph visualization and navigation in information visualization: a survey. **IEEE Transactions on Visualization and Computer Graphics**, [S.l.], v.6, p.24–43, 2000.

HYUN, Y. **Walrus**. [accessed on October 7, 2008], website.

J. Q. WALKER, I. A node-positioning algorithm for general trees. **Softw. Pract. Exper.**, New York, NY, USA, v.20, n.7, p.685–705, 1990.

JIA, Y. et al. On the Visualization of Social and other Scale-Free Networks. In: INFOVIS '08: PROCEEDINGS OF THE 2008 IEEE SYMPOSIUM ON INFORMATION VISUALIZATION (INFOVIS '08), 2008, Washington, DC, USA. **Proceedings...** IEEE Computer Society, 2008.

JUNGMEISTER, W.-A.; TURO, D. Adapting Treemaps to Stock Portfolio Visualization. , [S.l.], 1992.

KAMADA, T.; KAWAI, S. An algorithm for drawing general undirected graphs. **Inf. Process. Lett.**, Amsterdam, The Netherlands, v.31, n.1, p.7–15, 1989.

KOREN, Y. On Spectral Graph Drawing. In: IN COCOON 03, VOLUME 2697 OF LNCS, 2003. **Proceedings...** Springer-Verlag, 2003. p.496–508.

LAMPING, J.; RAO, R. The Hyperbolic Browser: a focus+context technique for visualizing large hierarchies. **Journal of Visual Languages Computing**, [S.l.], v.7, n.1, p.33–55, march 1996.

LAMPING, J.; RAO, R.; PIROLLI, P. A focus+context technique based on hyperbolic geometry for visualizing large hierarchies. In: CHI '95: PROCEEDINGS OF THE SIGCHI CONFERENCE ON HUMAN FACTORS IN COMPUTING SYSTEMS, 1995, New York, NY, USA. **Proceedings...** ACM Press/Addison-Wesley Publishing Co., 1995. p.401–408.

LEE, B.; PLAISANT, C.; PARR, C. S.; FEKETE, J.-D.; HENRY, N. Task taxonomy for graph visualization. In: BELIV '06: PROCEEDINGS OF THE 2006 AVI WORKSHOP ON BEYOND TIME AND ERRORS, 2006, New York, NY, USA. **Proceedings...** ACM, 2006. p.1–5.

MUNZNER, T. H3: laying out large directed graphs in 3d hyperbolic space. In: INFOVIS '97: PROCEEDINGS OF THE 1997 IEEE SYMPOSIUM ON INFORMATION VISUALIZATION (INFOVIS '97), 1997, Washington, DC, USA. **Proceedings...** IEEE Computer Society, 1997. p.2.

NOACK, A. **Energy models for drawing clustered small-world graphs**. [S.l.]: Brandenburg University of Technology at Cottbus, 2003.

NOACK, A. An energy model for visual graph clustering. In: INTERNATIONAL SYMPOSIUM ON GRAPH DRAWING (GD 2003), LNCS 2912, 11., 2004. **Proceedings...** Springer-Verlag, 2004. p.425–436.

NOACK, A. Energy-Based Clustering of Graphs with Nonuniform Degrees. In: INTERNATIONAL SYMPOSIUM ON GRAPH DRAWING (GD 2005, 13., 2005. **Proceedings...** Springer-Verlag, 2005. p.309–320.

NOIK, E. G. Layout-independent fisheye views of nested graphs. In: VL'93: IEEE SYMPOSIUM ON VISUAL LANGUAGES, 1993. **Proceedings...** [S.l.: s.n.], 1993. p.336–341.

NORTH, S. C.; KOUTSOFIOS, E. Applications of Graph Visualization. In: GRAPHICS INTERFACE, 1994, Banff, Canada. **Proceedings...** [S.l.: s.n.], 1994.

PARKER, G.; FRANCK, G.; WARE, C. Visualization of Large Nested Graphs in 3D: navigation and interaction. **Journal of Visual Languages and Computing**, [S.l.], v.9, p.299–317, 1998.

PATRIGNANI, M.; VARGIU, F. 3DCube: a tool for three dimensional graph drawing. In: GRAPH DRAWING (PROC. GD '97), VOLUME 1353 OF LECTURE NOTES COMPUT. SCI, 1998. **Proceedings...** Springer, 1998. p.284–290.

PISANSKI, T.; SHAWE-TAYLOR, J. **Characterizing graph drawing with eigenvectors**. [S.l.]: J. Chem. Inf. Comput. Sci, 1998.

PRETORIUS, A. J.; WIJK, J. J. van. Multidimensional Visualization of Transition Systems. In: IV '05: PROCEEDINGS OF THE NINTH INTERNATIONAL CONFERENCE ON INFORMATION VISUALISATION, 2005, Washington, DC, USA. **Proceedings...** IEEE Computer Society, 2005. p.323–328.

PRETORIUS, A. J.; WIJK, J. J. van. Visual Analysis of Multivariate State Transition Graphs. **IEEE Transactions on Visualization and Computer Graphics**, Piscataway, NJ, USA, v.12, n.5, p.685–692, 2006.

PROTOHAUS. **Fidg't**. [S.l.:s.n.: 200-]. Available in: <http://www.fidgt.com>. Accessed on: 2 Nov. 2008.

PUPPE, T. **Spectral Graph Drawing**. Germany: VDM Verlag Dr. Muller Aktiengesellschaft Co. Kg, 2008.

PURCHASE, H. C.; COHEN, R. F.; JAMES, M. I. An experimental study of the basis for graph drawing algorithms. **J. Exp. Algorithmics**, New York, NY, USA, v.2, p.4, 1997.

QT SOFTWARE INC. **Qt 4.4**. [S.l.:s.n.: 200-]. Available in: <http://www.trolltech.com>. Accessed on: 7 Oct. 2008.

REINGOLD, E.; TILFORD, J. Tidier Drawings of Trees. **Software Engineering, IEEE Transactions on**, [S.l.], v.SE-7, n.2, p.223–228, March 1981.

REKIMOTO, J.; GREEN, M. The information cube: using transparency in 3d information visualization. In: IN PROCEEDINGS OF THE THIRD ANNUAL WORKSHOP ON INFORMATION TECHNOLOGIES SYSTEMS (WITS'93), 1993. **Proceedings...** [S.l.: s.n.], 1993. p.125–132.

ROBERTSON, G. G.; MACKINLAY, J. D.; CARD, S. K. Cone Trees: animated 3d visualizations of hierarchical information. In: CHI '91: PROCEEDINGS OF THE SIGCHI CONFERENCE ON HUMAN FACTORS IN COMPUTING SYSTEMS, 1991, New York, NY, USA. **Proceedings...** ACM, 1991. p.189–194.

RODRIGUES JR., J. F. et al. SuperGraph Visualization. In: ISM '06: PROCEEDINGS OF THE EIGHTH IEEE INTERNATIONAL SYMPOSIUM ON MULTIMEDIA, 2006, Washington, DC, USA. **Proceedings...** IEEE Computer Society, 2006. p.227–234.

RODRIGUES JR., J. F. et al. GMine: interactive browsing of large graphs. , [S.l.], 2008.

RODRIGUES JR., J. F.; TONG, H.; TRAINA, A. J. M.; FALOUTSOS, C.; LESKOVEC, J. GMine: a system for scalable, interactive graph visualization and mining. In: VLDB'2006: PROCEEDINGS OF THE 32ND INTERNATIONAL CONFERENCE ON VERY LARGE DATA BASES, 2006. **Proceedings...** VLDB Endowment, 2006. p.1195–1198.

ROSS, E. **Spectral Graph Drawing: a survey**. Burnaby, BC, Canada: Simon Fraser University, 2004.

SAMET, H. **The Design and Analysis of Spatial Data Structures**. Reading, MA, USA: Addison-Wesley, 1990.

SHILOACH, Y. **Arrangements of Planar Graphs on the Planar Lattices**. 1976. Tese (Doutorado em Ciência da Computação) — Rehovot, Israel: Weizmann Institute of Science,.

SHNEIDERMAN, B. Tree visualization with tree-maps: 2-d space-filling approach. **ACM Trans. Graph.**, New York, NY, USA, v.11, n.1, p.92–99, 1992.

SILLICON GRAPHICS INC. **3D File System Navigator**. [S.l.:s.n.: 1992]. Available in: <http://www.sgi.com>. Accessed on: 7 Oct. 2008.

SPRITZER, A. S.; FREITAS, C. M. D. S. Interaction and Navigation in Graph Visualizations. **Revista de Informática Teórica e Aplicada**, [S.l.], v.15, n.1, p.111–137, Mar 2008.

SPRITZER, A. S.; FREITAS, C. M. D. S. A physics-based approach for interactive manipulation of graph visualizations. In: AVI '08: PROCEEDINGS OF THE WORKING CONFERENCE ON ADVANCED VISUAL INTERFACES, 2008, New York, NY, USA. **Proceedings...** ACM, 2008. p.271–278.

SPRITZER, A. S.; VOLQUIND, F. P.; FREITAS, C. M. D. S. Case Study of Co-authorship Networks Using a Tool for Graph Visualization. In: WORKSHOP ON INFORMATION VISUALIZATION AND ANALYSIS IN SOCIAL NETWORKS (WIVA 2008), 2008. **Proceedings...** SBC, 2008.

STOREY, M.-A. D. et al. A. On Integrating Visualization Techniques for Effective Software Exploration. In: INFOVIS '97: PROCEEDINGS OF THE 1997 IEEE SYMPOSIUM ON INFORMATION VISUALIZATION (INFOVIS '97), 1997, Washington, DC, USA. **Proceedings...** IEEE Computer Society, 1997. p.38.

SUGIYAMA, K.; MISUE, K. **Graph Drawing by Magnetic-Spring Model**. [S.l.]: Fujitsu Laboratories, ISIS, 1994.

SUGIYAMA, K.; MISUE, K. A Simple and Unified Method for Drawing Graphs: magnetic-spring algorithm. In: GD '94: PROCEEDINGS OF THE DIMACS INTERNATIONAL WORKSHOP ON GRAPH DRAWING, 1995, London, UK. **Proceedings...** Springer-Verlag, 1995. p.364–375.

SUGIYAMA, K.; TAGAWA, S.; TODA, M. Methods for Visual Understanding of Hierarchical Systems Structures. In: IEEE TRANSACTIONS ON SYSTEMS, MAN AND CYBERNETICS, 1981. **Proceedings...** [S.l.: s.n.], 1981. v.1, n.2, p.109–125.

TAMASSIA, R.; TOLLIS, I. Planar grid embedding in linear time. **Circuits and Systems, IEEE Transactions on**, [S.l.], v.36, n.9, p.1230–1234, Sep 1989.

TEJADA, E.; ERTL, T. Large Steps in GPU-based Deformable Bodies Simulation. **Simulation Practice and Theory. Special Issue on Programmable Graphics Hardware**, [S.l.], v.13, n.9, p.703–715, 2005.

TUNKELANG, D. **A Numerical Optimization Approach to General Graph Drawing**. 1999. Tese (Doutorado em Ciência da Computação) — School of Computer Science, Carnegie Mellon University.

VAN HAM, F.; WATTENBERG, M. Centrality Based Visualization of Small World Graphs. In: EUROVIS '08: PROCEEDINGS OF THE 2008 EUROGRAPHICS/IEEE-VGTC SYMPOSIUM ON VISUALIZATION (EUROVIS'2008), 2008, Washington, DC, USA. **Proceedings...** IEEE Computer Society, 2008.

VERLET, L. Computer "Experiments" on Classical Fluids. I. Thermodynamical Properties of Lennard-Jones Molecules. **Phys. Rev.**, [S.l.], v.159, n.1, p.98, Jul 1967.

WASSERMAN, S.; FAUST, K.; IACOBUCCI, D. **Social Network Analysis** : methods and applications (structural analysis in the social sciences). [S.l.]: Cambridge University Press, 1994.

WATTENBERG, M. Visualizing the stock market. In: CHI '99: CHI '99 EXTENDED ABSTRACTS ON HUMAN FACTORS IN COMPUTING SYSTEMS, 1999, New York, NY, USA. **Proceedings...** ACM, 1999. p.188–189.

WATTENBERG, M. Visual exploration of multivariate graphs. In: CHI '06: PROCEEDINGS OF THE SIGCHI CONFERENCE ON HUMAN FACTORS IN COMPUTING SYSTEMS, 2006, New York, NY, USA. **Proceedings...** ACM, 2006. p.811–819.

YI, J. S.; MELTON, R.; STASKO, J.; JACKO, J. A. Dust & magnet: multivariate information visualization using a magnet metaphor. **Information Visualization**, [S.l.], v.4, n.4, p.239–256, 2005.

ZHAO, S.; MCGUFFIN, M.; CHIGNELL, M. Elastic hierarchies: combining treemaps and node-link diagrams. **Information Visualization, 2005. INFOVIS 2005. IEEE Symposium on**, [S.l.], p.57–64, Oct. 2005.

APPENDIX A USER EVALUATION TABLE

Tables A.1, A.2 and A.3 show the complete tabulation of the answers given by the users in the evaluation questionnaire they filled after testing MagnetViz's prototype, as described in Chapter 5.

#	**Affirmations**	**Average**	**Mode**	**Standard Deviation**
5	I clearly understand what a magnet can be used for.	4.92	5.00	0.27
24	The criterion of type "Node Textual Attribute Value" is useful.	4.92	5.00	0.27
23	The criterion of type "Node Numerical Attribute Value" is useful.	4.92	5.00	0.27
21	The criterion of type "Node Degree" is useful.	4.92	5.00	0.27
13	I clearly understand the difference between requirements and criteria.	4.92	5.00	0.27
15	Criteria and requirements are useful for the exploration of the graph.	4.77	5.00	0.43
6	Magnets are useful for the exploration of the graph.	4.62	5.00	0.64
7	I clearly understand what a boundary shape can be used for.	4.54	5.00	0.65
34	The technique is useful for the exploration of social networks.	4.38	5.00	0.96
20	The criterion of type "Magnets" is useful.	4.38	5.00	0.76
8	Boundary shapes are useful for layout reorganization.	4.31	5.00	0.74

Table A.1: Affirmations and statistics about the subjects' level of agreement (part 1).

#	Affirmations	Average	Mode	Standard Deviation
25	The criterion of type "Path Length" is useful.	4.31	5.00	1.39
16	The criterion of type "Connected Subgraph" is useful.	4.31	5.00	1.01
35	Manipulating the graph makes it possible to discover situations that were previously unknown.	4.15	4.00	0.86
33	Manipulating the graph makes it possible to explore it and discover relationships between the nodes.	4.08	4.00	0.62
4	The alterable properties of the layout are useful for controlling the physics simulation that determines the layout of the graph.	4.00	5.00	1.04
1	The graph's initial layout is clear.	3.85	4.00	0.66
31	In general, magnets are easy to use.	3.82	5.00	1.04
32	It is easy to specify and edit criteria.	3.77	4.00	1.25
14	The criterion types provided in the prototype are adequate for the layout modifications necessary to answer the given questions.	3.69	4.00	1.07
3	It is clear which properties of the layout are alterable.	3.69	3.00	1.01
28	Magnet intersection is interesting for the exploration of the graph.	3.62	4.00	1.65
2	It is possible to clearly observe aspects of the layout and general traits of nodes and edges.	3.58	4.00	0.96
22	The criterion of type "Node Has Attribute" is useful.	3.38	5.00	1.74
18	The criterion of type "Edge Numerical Attribute Value" is useful.	3.38	5.00	2.03

Table A.2: Affirmations and statistics about the subjects' level of agreement (part 2).

#	**Affirmations**	**Average**	**Mode**	**Standard Deviation**
30	The layout obtained after the manipulation allows for a clear view of the query's result.	3.31	4.00	1.28
26	The icon used for the nodes of a magnet intersection allows for their easy identification.	3.31	4.00	1.50
19	The criterion of type "Edge Textual Attribute Value" is useful.	3.23	5.00	2.13
27	The colored lines make perception of magnet intersection easier.	3.23	3.00	1.25
9	I clearly understand what the boundary shape's gravity multiplier can be used for.	2.92	3.00	1.27
17	The criterion of type "Edge Has Attribute" is useful.	2.54	4.00	1.86
10	The gravity multiplier is useful to better control layout modification.	2.54	2.00	1.51
11	I can clearly see the difference between magnets and gravity spots.	2.54	0.00	2.24
29	Magnet intersection is adequately represented in the layout.	2.31	4.00	1.55
12	Gravity spots are useful to control the layout.	1.54	0.00	1.83

Table A.3: Affirmations and statistics about the subjects' level of agreement (part 3).

APPENDIX B USER EVALUATION QUESTIONNAIRE

The list below shows the affirmations contained in the evaluation questionnaire that was handed to users after they experimented with the MagnetViz prototype. The affirmations are shown in the same order as they were in the original questionnaire.

1. The graph's initial layout is clear.
2. It is possible to clearly observe aspects of the layout and general traits of nodes and edges.
3. It is clear which properties of the layout are alterable.
4. The alterable properties of the layout are useful for controlling the physics simulation that determines the layout of the graph.
5. I clearly understand what a magnet can be used for.
6. Magnets are useful for the exploration of the graph.
7. I clearly understand what a boundary shape can be used for.
8. Boundary shapes are useful for layout reorganization.
9. I clearly understand what the boundary shape's gravity multiplier can be used for.
10. The gravity multiplier is useful to better control layout modification.
11. I can clearly see the difference between magnets and gravity spots.
12. Gravity spots are useful to control the layout.
13. I clearly understand the difference between requirements and criteria.
14. The criterion types provided in the prototype are adequate for the layout modifications necessary to answer the given questions.
15. Criteria and requirements are useful for the exploration of the graph.
16. The criterion of type "Connected Subgraph" is useful.
17. The criterion of type "Edge Has Attribute" is useful.
18. The criterion of type "Edge Numerical Attribute Value" is useful.

19. The criterion of type "Edge Textual Attribute Value" is useful.
20. The criterion of type "Magnets" is useful.
21. The criterion of type "Node Degree" is useful.
22. The criterion of type "Node Has Attribute" is useful.
23. The criterion of type "Node Numerical Attribute Value" is useful.
24. The criterion of type "Node Textual Attribute Value" is useful.
25. The criterion of type "Path Length" is useful.
26. The icon used for the nodes of a magnet intersection allows for their easy identification.
27. The colored lines make perception of magnet intersection easier.
28. Magnet intersection is interesting for the exploration of the graph.
29. Magnet intersection is adequately represented in the layout.
30. The layout obtained after the manipulation allows for a clear view of the query's result.
31. In general, magnets are easy to use.
32. It is easy to specify and edit criteria.
33. Manipulating the graph makes it possible to explore it and discover relationships between the nodes.
34. The technique is useful for the exploration of social networks.
35. Manipulating the graph makes it possible to discover situations that were previously unknown.

APPENDIX C   RESUMO E CONTRIBUIÇÕES

Esta dissertação apresenta MagnetViz, uma técnica baseada em física para a manipulação interativa de visualizações de grafos. A principal contribuição deste trabalho é a abordagem utilizada para modificar o layout do grafo. Enquanto a maior parte das técnicas se preocupa em visualizar um *layout* estático previamente gerado, MagnetViz permite que os usuários adaptem interativamente o *layout* usando ferramentas para o estabelecimento de regiões para nodos que preenchem determinados critérios topológicos ou semanticos. Desta forma, consultas aos dados representados pelo grafo são realizadas interativamente e tem efeito no layout seguindo o mesmo algoritmo usado para gerar o layout inicial.

A técnica foi construída sobre a metáfora de física dos conhecidos algoritmos de *layout* dirigidos por forças (C.1a). Eles foram estendidos para suportar o conceito de imãs, que são o elemento chave do MagnetViz. Imãs, ilustrados nas figuras C.1b e C.2 são inseridos na cena pelo usuário e a eles são associados critérios de atração. Critérios podem ser *requerimentos* (todos devem ser satisfeitos por um nodo para que este seja atraído) ou *critérios simples* (se houver critérios, ao menos um deve ser satisfeito). Pode-se definir uma área onde os nodos atraídos devem permanecer através de uma *boundary shape*, que é simplesmente uma forma geométrica (no caso, um círculo) que limita esta área.

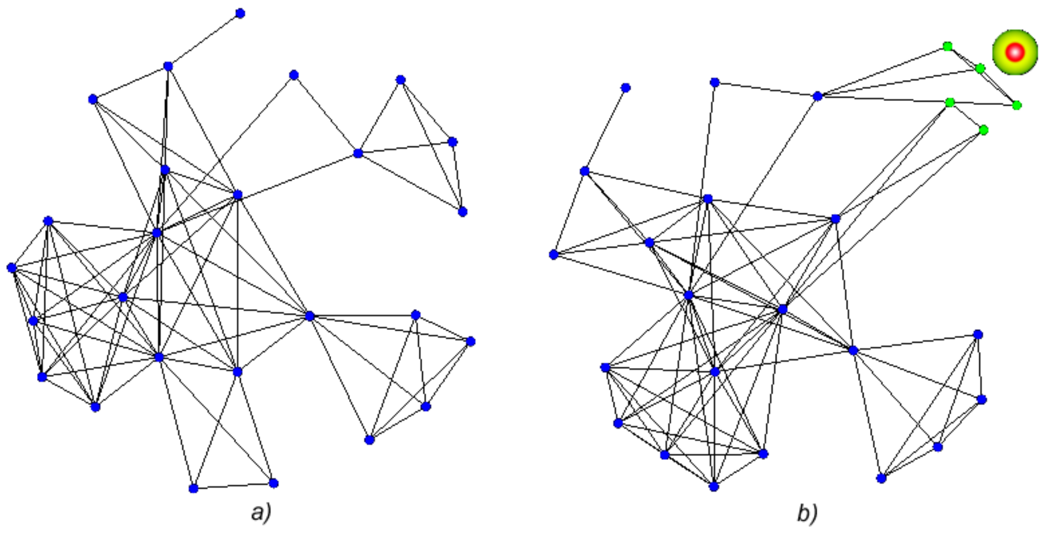


Figure C.1: a) Maior componente do grafo de co-autorias da conferência AVI em seu layout inicial (26 nodos e 78 arestas); b) Um imã em ação, atraindo nodos de grau 3.

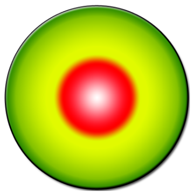


Figure C.2: Um imã.

Quando a força de atração de um imã é configurada, o sistema de física reavalia o *layout*, o alterando de forma que os nodos atraídos por este imã se movam em sua direção. Caso um imã possua uma *boundary shape* (Figura C.3) os nodos atraídos são forçados a ficar dentro dela, enquanto os não-atraídos são forçados a permanecer fora. Vários imãs podem ser inseridos, inclusive sendo possível criar uma hierarquia de imãs. Quando um mesmo nodo e atraído por dois imãs, o usuário é chamado a decidir como deve ser o posicionamento final do mesmo em relação aos imãs o atráem.

O algoritmo utilizado como base para o protótipo desenvolvido para prova de conceito e testes é uma combinação do algoritmo clássico de Fruchterman-Reingold (1991) com a técnica desenvolvida por Barnes e Hut (1986) para simulação de n-corpos, como feito por Tunkelang (1999). Um layout de um grafo mais complexo gerado com este algoritmo pode ser visto na figura C.4.

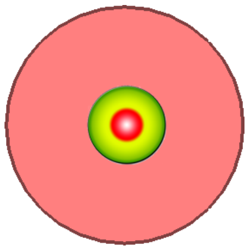


Figure C.3: Um imã com *boundary shape*.

A avaliação da técnica foi realizada em duas etapas, com a primeira consistindo da verificação das possibilidades de uso do MagnetViz em relação à taxonomia de tarefas desenvolvida para aplicações de visualização de grafos que foi proposta por Lee et al. (2006). Através deste estudo foi possível constatar que apesar de não poder ser aplicado diretamente em todas as tarefas como descritas na taxonomia, MagnetViz pôde lidar diretamente com a maior parte delas, sendo especialmente bem apropriado para as de mais alto nível, específicas para grafos, e de natural utilização para as que envolvem *clustering* e operações de conjuntos.

A segunda etapa da avaliação consistiu em colocar MagnetViz em um ambiente prático através de um estudo de caso. Para isso foi decidido aplicar a técnica no cada vez mais relevante contexto de análise e visualização de redes sociais. O grafo de co-autorias gerado pelas publicações de 2007 dos professores e alunos da UFRGS e seus colaboradores externos foi escolhido como conjunto de dados a ser visualizado, e perguntas foram elaboradas sobre essa rede para serem respondidas fazendo uso da técnica. Durante testes foi descoberto que duas questões não podiam ser respondidas diretamente por duas razões: o protótipo não estava adaptado para este contexto específico e algumas funcionalidades

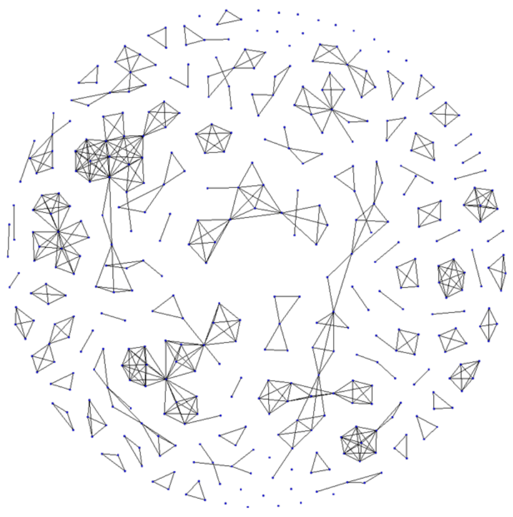


Figure C.4: Exemplo de layout gerado com o algoritmo utilizado (grafo de co-autorias da conferência AVI, com 416 nodos e 628 arestas).

planejadas que seriam necessárias para dar suporte a essas perguntas ainda não foram implementadas.

Para a avaliação experimental da técnica, foi planejado um ensaio de interação com um grupo de 14 pessoas que usaram o MagnetViz para responder as mesmas questões sobre a rede de co-autorias. Os sujeitos subsequentemente receberam um questionário com afirmações sobre a relevância da técnica e a usabilidade de suas funcionalidades. Estas afirmações deveriam ser avaliadas com notas que deveriam ser dadas de acordo com o nível de concordância do usuário com o que estava sendo afirmado.

Em geral, os usuários aprovaram a técnica e sua utilidade, achando a metáfora intuitiva e relevante para a visualização de redes sociais. A partir das respostas do questionário, entretanto, apesar de poder-se concluir que a técnica tem alguns pontos que necessitam ser melhorados (especialmente quanto ao protótipo desenvolvido), foi verificado que conceitualmente a técnica em si foi validada pelos usuários que a testaram. O único ponto deficiente envolvendo a técnica apontado pelos usuários foi sobre a forma como as intersecções de imãs são tratadas. Os usuários acharam que a solução dada para ressaltar os nodos atraídos por mais de um imã não foi muito intuitiva, e que o resultado visual do tratamento aplicado não ficou muito claro.

Com a possibilidade de estabelecer critérios de rearranjo do layout com base nos atributos semânticos dos nodos e arestas, além dos topológicos, a técnica é de certa forma genérica. Entretanto, o protótipo (tal como está hoje) não contém um processador de consultas que permita sua aplicação a qualquer domínio.

MagnetViz, entretanto, contribue com uma abordagem inovadora que pode ser aperfeiçoada para se tornar realmente genérica. A técnica é, assim, outra ferramenta potencialmente útil para *designers* de *software* que necessitam lidar com o problema de visualização de grafos interativa.

O restante deste texto foi organizado expandindo-se as publicações realizadas ao longo do período: o artigo tutorial apresentado no SIBGRAPI 2007 e publicado recentemente (SPRITZER; FREITAS, 2008a) é a base do capítulo sobre conceitos e trabalhos relacionados (capítulos 3 e 4); o artigo publicado na International Working Conference on Advanced Visual Interfaces (SPRITZER; FREITAS, 2008b) foi o ponto de partida do capítulo 5, que descreve a técnica e o protótipo MagnetViz, e de parte do capítulo 6; e o mais recente, publicado no Workshop on Information Visualization and Analysis in Social Networks (SPRITZER; VOLQUIND; FREITAS, 2008), deu origem à seção de avaliação experimental (capítulo 6).