A Taxonomy of Force-Directed Placement Techniques

Faruk Bajramovic
Arne Tauber
Ralph Wozelka
Wörister Ferdinand

Contents

1 Introduction

In the 'Silicon Age', the amount of information available at our fingertips has reached an unprecedented scale. The Internet, providing ubiquitous access to information from around the world is justifiedly seen as a development on par with the intruduction of the printing press. However, whereas technology can cope with these staggering amounts of information, to humans it increasingly constitutes a challenge. Presented in form of a table, humans lack the ability to quickly grasp the similarities and dissimilarities of datapoints within a dataset.

However, 'a picture is worth a thousand words'. Our brain has evolved over thousands of years sophisticated techniques that allow us to quickly infer meaning from visual patterns. Thus the transformation of such datasets into geometric representations provides humans with a much easier way of coping with large datasets than e.g. table-based representations.

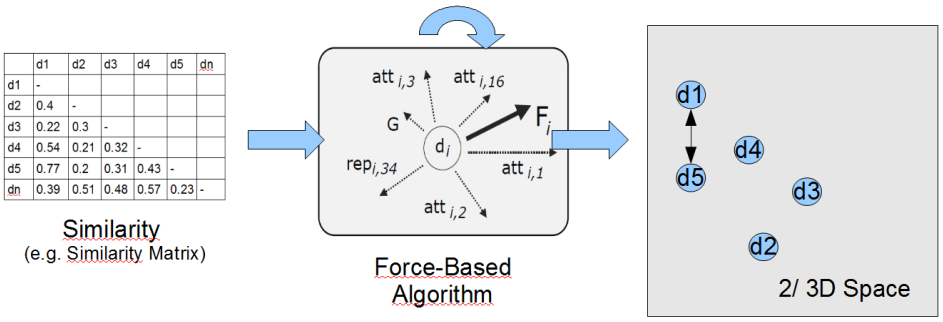


Figure 1: Multimensional Scaling is a class of techniques mapping objects into one-, two- or three dimensional space, based on their (dis)similarity amongst each other. These relationships can be expressed using a matrix structure, referred to a Similarity Matrix. The multidimensional feature space is condensed into a matrix shown on the left side. The matrix' columns and rows represent the records within a high-dimensional dataset, its cells contain the similarity measure between pairs of individual records. Force-Directed Placement can be used to position the individual nodes within the graph.

In this paper we focus on 'force-based' solutions to the problem of visualizing sets of data items in $\{1, 2, 3\}$-dimensional space.

In order to obtain a layout of data items in such low-dimensional spaces, virtual forces between the location of the items are introduced (attractive and/or repelling ones) which are proportional to the distance between items in a layout. These forces generally act as a metric to measure how well the location of items in the layout captures their relationship from within their original representation. Such relationships might stem, e.g., from a graph structure connecting the data items, or from distance calculations between items in high-dimensional spaces.

Such approaches can be subsumed under the category of *Force-Directed Placement Techniques*.

In our survey we identify two two main areas of the field, the visualization of partial graphs, and the visualization of high-dimensional datasets (full graphs).

The former deals with calculating layouts of simple, connected, unidirected graphs. Force-directed algorithms belong to the most flexible algorithms to perform this task (cf. Kobourov (2007)). Good surveys can be found in Battista, Eades, Tamassia, and Tollis (1999) and Brandes (2001).

The latter deals with visualizing data items from high-dimensional feature spaces. Commonly, this problem is tackled by reducing the number of dimensions. This then facilitates the application of layouting techniques suitable for low-dimensional data (such as graph drawing techniques). *Multidimensional Projection* is one method that can be used to for that. It usually maps data from $\mathbb{R}^m$ to $\mathbb{R}^n$ where $n \ll m$ and $n \in \{1, 2, 3\}$. Examples of such datasets are (large) document collections, bibliographies, or e.g. voting behaviour.

This set of techniques can be separated into *Linear* and *Non-Linear Projection Techniques*.

Linear techniques include most prominently *Principal Component Analysis (PCA)* which combines dimensions into a new set of orthogonal basis vectors capturing directions of maximum variation. The lowest dimensions are then used to create the layout. But, in general, linear methods cannot guarantee to capture non-linear structures, such as arbitrarily shaped clusters.

Non-Linear techniques on the other hand are more suitable for this task. However, to achieve high precision is often computationally expensive (cf. Paulovich, Nonato, Minghim, and Levkowitz (2008)).

Such techniques are designed to *minimise the information loss* incurred by the projection to low-dimensional space. Commonly, the cost function involved is based on a metric describing the (dis)similarity of data items in $\mathbb{R}^m$ and their (Euclidian) distance in $\mathbb{R}^n$.

On such non-linear projection technique is *Multidimensional Scaling (MDS)*. For a general discussion see Cox and Cox (2001) or Borg and Groenen (2005).

Below in section 1.1 and 1.2 we shortly discuss Force-Directed Placement and Multidimensional Scaling in further detail.

1.1 Force Directed Placement in Graph Drawing

Force-based graph drawing algorithms are techniques of positioning the nodes of a graph in two- or three dimensional space in an aesthetically pleasing way.

Fundamentally, they simulate repulsive and attractive forces between the graph's nodes. We quote Kobourov (2007):

> *In general, force-directed methods define an objective function which maps each graph layout into a number in* $\mathbb{R}^+$ *representing the energy of the layout. This function is defined in such a way that low energies correspond to layouts in which adjacent nodes are near some pre-specified distance from each other, and in which non-adjacent nodes are well-spaced. A layout for a graph is then calculated by finding a (often local) minimum of this objective function.*

1.2 Multidimensional Scaling

Multimensional Scaling is a class of techniques mapping objects into one-, two- or three dimensional space, based on their (dis)similarity among eachother. The correlation between data points is represented by their geometric proximity. This technique provides a way of quickly achieving an overview of the 'hidden' information within a dataset.

Cox and Cox (2001) consider reconstructing a map from a table of distances between cities a classic example of Multimensional Scaling. Apart from this example, Kruskal and Wish (1978) illustrate Multimensional Scaling as a method that allows condensing data about political candidates into a map, thus providing a spatial representation of the hidden structure within the dataset (partisanship, ideology, etc.). Often, yet not necessarily, Multimensional Scaling relies on a so-called Similarity Matrix (see Figure 1). Its columns and rows represent certain records within a dataset, its cells represent the similarity between individual datarecords.

In this paper we intend to present an overview of MDS methods that are based on Force-Directed Placement. Hence, in that sense, FDP can be considered a subset of Multimensional Scaling.

Quoting Tejada, Minghim, and Nonato (2003) on a formal description of multimensional projection:

> *Formalizing the concept of distance-based multidimensional projections, let* $X = \{x_1, x_2, \ldots, x_n\}$ *be a set of* m*-dimensional data, with* $\delta(x_i, x_j)$ *a dissimi- larity (distance) measure between two* m*-dimensional data instances, and let* $Y = \{y_1, y_2, \ldots, y_n\}$ *be a set of points into a* n*-dimensional space, with* $n = \{1, 2, 3\}$ *and* $d(y_i, y_j)$ *a (Euclidean) distance between two points of the projected space. A multidimensional projection technique can be described as a injective function* $f : X \rightarrow Y$ *that seeks to make* $|\delta(x_i, x_j) - d(f(x_i), f(x_j))|$ *as close to zero as possible,* $\forall x_i, x_j \in X$*.*

If we want apply FDP (e.g.the spring force model of Eades (1984) discussed in section 3.1) as a MDS technique, the forces must be made proportional to the difference between the (dis)similarity between items $\delta(x_i, x_j)$ and their distances $(f(x_i), f(x_j))$ in low-dimensional space $\mathbb{R}^n$, f denoting the multi-dimensional projection step (cf. Paulovich et al. (2008)).

Force-Directed Placement can be used in the context of MDS to calculate the layout of the resulting graph based upon the (dis)similarity inbetween high-dimensional feature vectors.

In the following section, we present two early force-directed drawing algorithms that have infuenced modern algorithms such as those presented in chapters three and four. Chapter three presents an overview of the basic concepts and methods applied in the field of Force-Directed Graph Drawing, whereas the latter chapter four presents improvements upon these earlier contributions. In chapter five, we present a technique particularly suited for large graphs - Multi-Level Algorithms.

In Chapter 6 we focus on the applications of Force-Directed Placement in multidimensional scaling, that is, visualisation of high-dimensional data sets.

1.3 Graphical Taxonomy

In this section you can find a condensed graphical version of our taxonomy in Figure 2 .

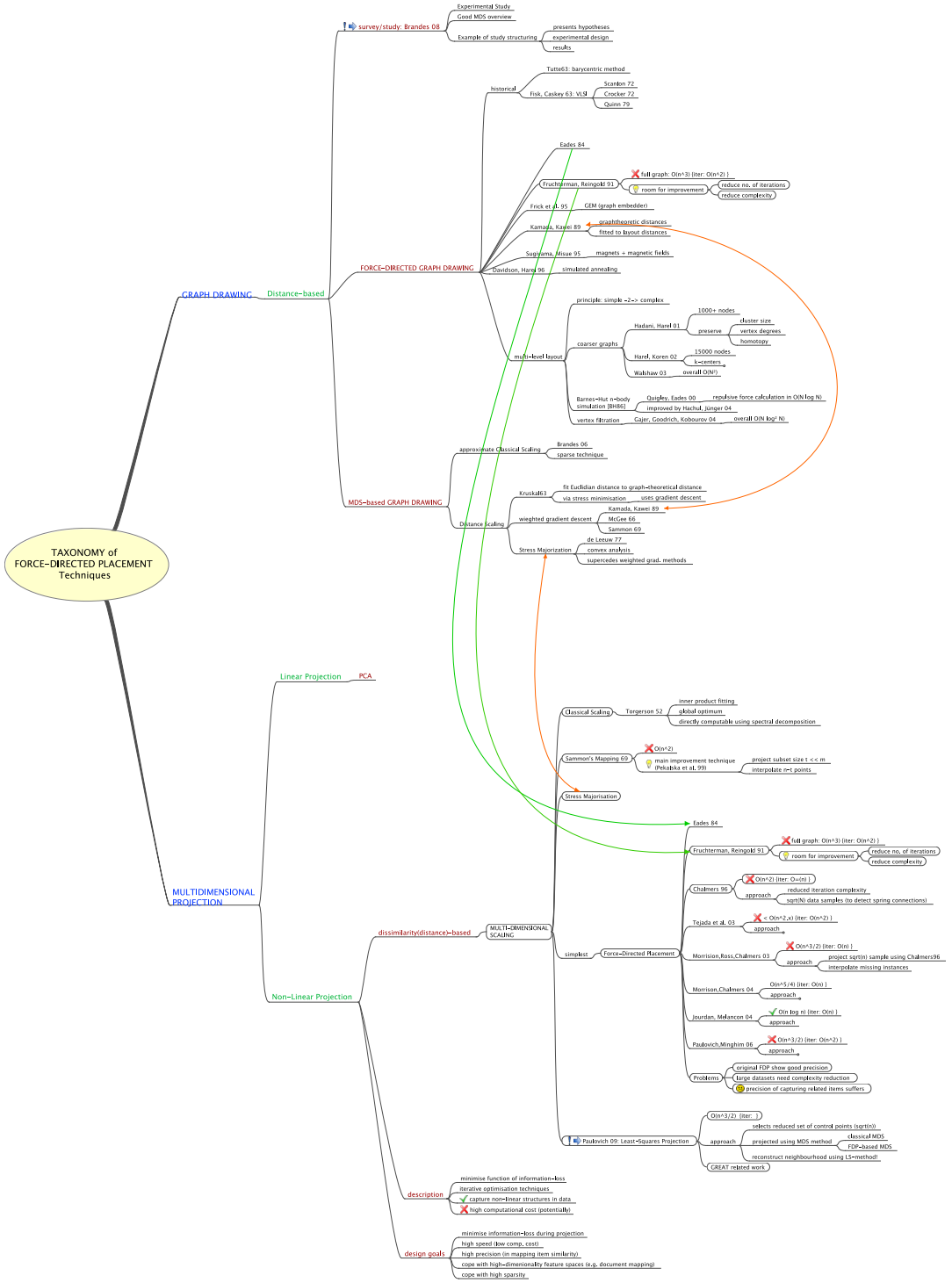


Figure 2: A graphical taxonomy of Force-Directed Placement techniques in graph drawing and general Multidimmensional Scaling and how they relate to each other.

2 First Force-directed Graph Layout Algorithms

Algorithms using force-directed placement have already been described before the ones using spring forces as first introduced by Eades (1984). Tutte's barycentric method can be seen as the first "force-directed"' algorithm. We discuss this algorithm in the first subsection. However, force-directed placement was also applied in the context of VLSI design in the 1960's and 1970's. Fisk and Caskey described in 1967 an automated circuit card etching layout algorithm based on force-directed placement. In 1979 Quinn and Breur described a force-directed component placement procedure for printed circuit boards. We discuss the basic idea behind the latter methods in subsection 2.

2.1 The Barycentric Method

Tutte (1963) described a barycentric method to draw an aesthetic pleasing graph. Historically, this method can be considered as the first force-directed placement algorithm. In case Tutte's algorithm is applied to 3-connected planar graphs, it guarantees that the resulting graph is crossing-free (plane convex drawing of the graph).

Definition of triconnected graph: A connected graph such that deleting any two vertices (and incident edges) results in a graph that is still connected.

Tamassia (2007) describes the idea behind Tutte's algorithm in chapter 5.3 as

> *"...Tutte's algorithm is that if a face of the planar graph is fixed in the plane, then suitable positions for the remaining vertices can be found by solving a system of linear equations, where each vertex position is represented as a convex combination of the positions of its neighbours. This can be considered a force-directed method..."*

Tutte's model uses springs of ideal length zero and no repulsive forces. The force resulting from an edge (u,v) is proportional to the distance between u and v. The force at a vertex v is calculated as follows:

$$F(v) = \sum_{(u,v)\in E} p_u - p_v$$

Where p_u and p_v are the positions of vertices u and v. The time complexity to solve the equations is $O(n^{1,5})$.

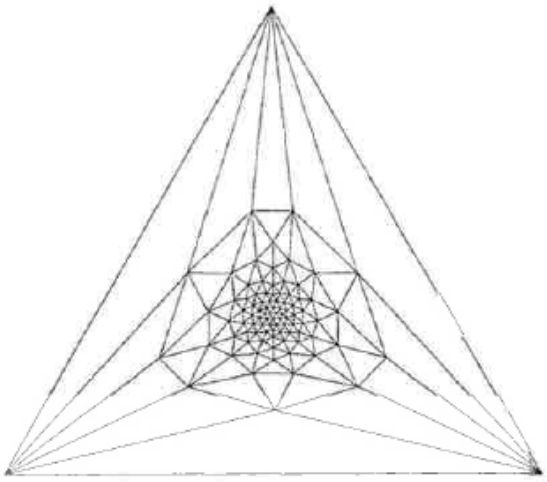


Figure 3: A large planar graph drawn with the barycenter method (Taken from Battista (1999)

2.2 VLSI Layout Algorithms

Fisk, Caskey, and West (1967) described in 1963 a computer program called *ACCEL: Automated Circuit Card Etching Layout*. The aim of ACCEL was to reduce the time needed to design printed circuit boards and was developed jointly by Sandia Corporation and the Thomas Bede Foundation. ACCEL can handle components with up to four leads. Fisk and Caskey describe their method of component layout on the board as "force placement".

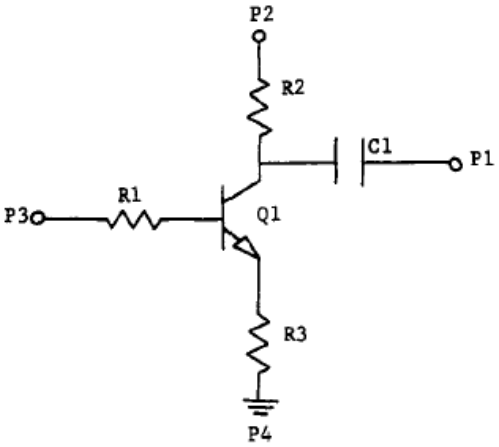


Figure 4: Example of VLSI components to be layouted (Taken from Fisk and Caskey)

They describe the basic concept behind this force-based placement as

follows. Consider the component C_1 in Figure 4. C_1 is connected to R_2, P_1 and Q_1. For an efficient layout, these components should be located together. In order to ensure this, these components are assigned "physical forces of attraction" to be pulled together. Forces are limited to not disturb the layout of other components and are calculated in a way that overlapping of components is prevented.

Futher force-directed placement techniques of VLSI components have been described by Scanlon (1971) and Crocker, McGuffin, and Naylor (1972). Since each of the three mentioned methods has some technical shortcoming, Quinn and Breur (1979) described a force directed component placement procedure for printed circuit boards refining the mentioned techniques to remove these shortcomings. The method of Quinn and Breur consists of two phases: phase 1 is called the "relative location phase" and phase 2 is called the "slot assignment or component overlap resolution phase". Phase 1 deals with determining the correct relative position of a component to each other component. Phase 2 determines the real location of components by taking overlaps into account.

3 Basic force-directed techniques

In this section we discuss some basic force-directed techniques. Eades and Tamassia (1988) introduced a first method using a spring model. Based on this basic idea, other approaches using graph theoretic distances, magnetic fields, simulated annealing or genetic algorithms have been proposed.

3.1 Spring Forces - Eades (1984)

The algorithm of Eades evolved from the VLSI force-directed placement techniques discussed in 2.2. It is designed to handle graphs up to 30 vertices and to produce an "aesthetically" undirected graph. According to Eades and Tamassia (1988) the following criteria are generally considered as being aesthetic:

- Distribute the vertices evenly in the frame.
- Minimize edge crossings.
- Make edge lengths uniform.
- Reflect inherent symmetry.
- Conform to the frame.

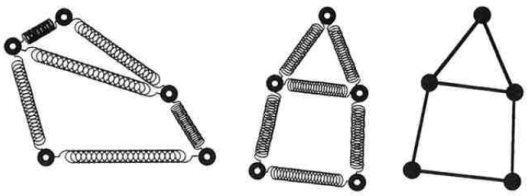


Figure 5: A spring algorithm (Taken from Battista (1999)

We quote Tamassia (2007), who summarizes Eades algorithm as follows:

> *To embed [lay out] a graph we replace the vertices by steel rings and replace each edge with a spring to form a mechanical system. The vertices are placed in some initial layout and let go so that the spring forces on the rings move the system to a minimal energy state. Two practical adjustments are made to this idea: firstly, logarithmic strength springs are used; that is, the force exerted by a spring is:*

$$c_1 * log\left(\frac{d}{c_2}\right)$$

where d is the length of the spring, and c_1 and c_2 are constants. Experience shows that Hookes law (linear) springs are too strong when the vertices are far apart; the logarithmic force solves this problem. Note that the springs exert no force when d = c_2. Secondly, we make nonadjacent vertices repel each other. An inverse square law force,

$$\frac{c_3}{\sqrt{d}}$$

where c_3 is constant and d is the distance between the vertices, is suitable.

From the description above we can see that Eades modeled a graph as a system of rings and springs. However, Eades did not apply Hooke's law to his model. In mechanics and physics Hooke's law of elasticity states that:

$F = -k * x$

where x is the deformation of the elastic body due to the force F, and k is the spring constant. Eades applied his own model to describe the forces in this spring-based system. This is not the only deviation from the physical model. In Eades's model, repulsive force are calculated between all vertices, whereas attractive forces are applied and calculated only between neighboring vertices.

From the efficiency perspective, the time complexity of calculating attractive forces is $O(n)$, however, the calculation of all repulsive forces still remains $O(n^2)$.

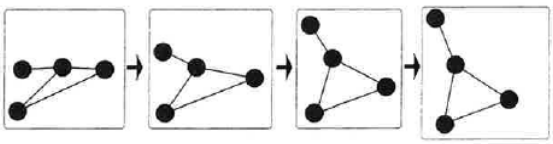


Figure 6: Frames in an animation of a spring algorithm (Taken from Battista (1999)

3.2 Graph Theoretic Distance - Kamada and Kawai (1989)

Like Eades, Kamada and Kawai (1988, 1989) have proposed a force-directed placement algorithm based on a spring model. Whereas Eades uses its own logarithmic function instead of Hooke's law, Kamada and Kawai use a graph theoretic distance approach as quoted below:

We regard the desirable geometric (Euclidean) distance between two vertices in the drawing as the graph theoretic distance between them in the corresponding graph.

Figure 7: Graph drawn using a simple spring algorithm (Taken from Battista (1999)

Their model relies on achieving a good graph layout by minimizing the pair-wise difference between the geometric (Euclidean) and graph theoretic distance of connected vertices, i.e. we have only attractive forces between all adjacent nodes. The "ideal" or graph theoretic distance between two vertices is calculated through a shortest-path algorithm. In contrast to the model of Eades, the model of Kamada and Kawai does not have attractive and repulsive forces in terms of a physical force. Instead we have geometric distances that are smaller or larger than their graph theoretic distances.

The overall energy E of their system is calculated as follows:

$$E = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{1}{2} k_{i,j} \left(|p_i - p_j| - l_{i,j} \right)$$

where p_i is the position of vertex $v_i \in V$. $k_{i,j}$ is the spring constant between p_i and p_j, i.e. the strength of the spring between the two vertices. $l_{i,j}$ is the "ideal" graph theoretic distance between the two vertices.

The model relies on the assumption that the graph-drawing problem can be solved by minimizing the total energy, i.e. minimizing the compression or tension on all springs. By doing so, the vertices would be laid out in a way that the difference between the geometric and graph theoretic distance would be minimal. The minimal energy can be calculated by solving 2n differential equations using the Newton-Raphson method.

Regarding efficiency, the algorithm of Kamada and Kawai is quite expensive. The calculation of the shortest paths can be done in $O(n^3)$ using the Floyd-Warshall algorithm or in $O(|E| \cdot N)$ using the Breath-First search.

3.3 Magnetic Fields - Sugiyama and Misue (1995)

Sugiyama and Misue (1995a, 1995b) extended the spring model by introducing magnetic fields acting on the springs, i.e. the springs are magnetized. This approach allows to control the orientation of the spring with the magnetic field and thus to have more control on the aesthetic look-and-feel of the resulting output.

Springs can be magnetized in in different ways (unidirectional, bidirectional, not at all) and usually there are three types of magnetic fields acting on the springs:

- Parallel field
- Radial field
- Concentric field

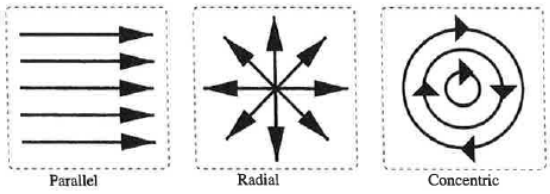


Figure 8: Types of magnetic fields (Taken from Battista (1999))

Figure 8 shows the different types of magnetic fields.

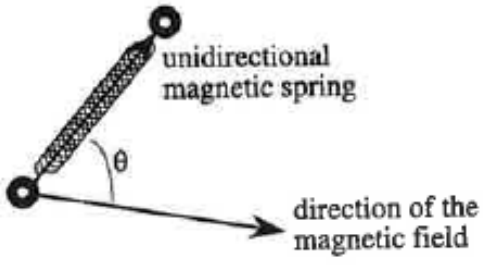


Figure 9: Magnetic spring (Taken from Battista (1999))

Figure 9 shows a drawing using unidirectional magnetic springs where a parallel magnetic field acts on the springs. The magnetic model is able to manage also directional springs (or even mixed models).

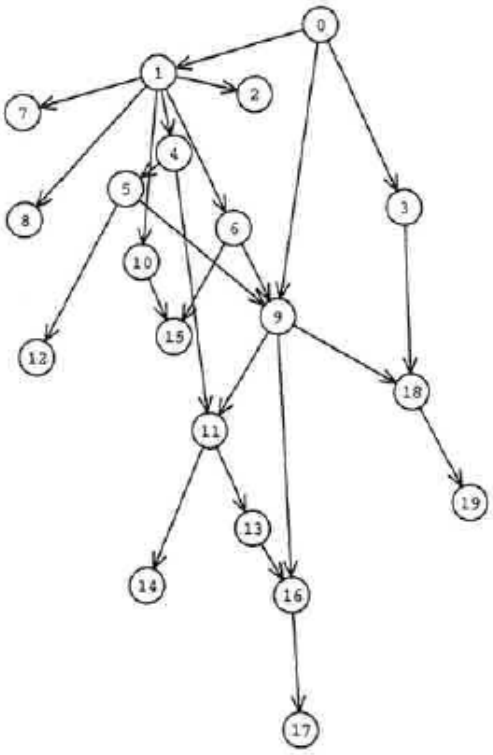


Figure 10: Magnetic spring drawing using a vertical magnetic field and unidirectional magnetic springs (Taken from Battista (1999))

3.4 Simulated Annealing - Davidson and Harel (1996)

Like Kamada and Kawai, Davidson and Harel (1996) tried to reduce the overall energy of a system. Instead of using a graph theoretic approach, Davidson and Harell used the (VLSI) optimization technique called "simulated annealing". The name of this technique comes from metallurgy and describes the process of heating and controlled cooling of a material.

The idea behind simulated annealing is to begin with a randomly starting position and a high temperature T. In each step, the temperature value is decreased (cooling-function) and the current solution is replaced with a random close-by solution. The probability the current solution changes depends on the temperature T. This is the "downhill" move. To avoid the process to get stuck, "uphill" moves are also allowed. The algorithm of Davidson and Harrell searches for local minima of the energy function using

simulated annealing.

Even if the method of Davidson and Harel produces a good outcome in terms of aesthetics, simulated annealing has a poor performance. Even if the inner loop of the algorithm only has a time complexity of $O(n)$, it is very slow and unfeasible for interactive (force-directed) graph drawing.

Other graph drawing approaches using simulated annealing have been proposed or mentioned by Cruz and Twarog (1996), Monien, Friedhelm, and Salmen (1996), Mendonca (1994) and Coleman and Parker (1996).

Tunkelang (1993) uses the same cost function as Davidson and Harel, but has an additional method for finding local minima.

3.5 Genetic Algorithms

Like simulated annealing, genetic algorithms are another common technique for finding nearby optimum solutions. Instead of varying the temperature (uphill, downhill) with simulated annealing, genetic algorithms use other techniques like inheritance, mutation, natural selection and recombination.

Vose (1998) gives a good survey of this topic. The most popular works on genetic algorithms for graph drawing have been proposed by Kosak, Marks, and Shieber (1991) and Branke, Bucher, and Schmeck (1996). Further information on that topic is given by Branke (1996) and Rosete (1997).

Like for simulated annealing, genetic algorithms are computationally intensive and thus are not suited for interactive graph drawing.

4 Improvements

4.1 Fruchterman and Reingold (1991)

Fruchterman and Reingold (1991) proposed a new force-directed placement algorithm, which is based on the concept of Eades. Like Davidson and Harel, it also uses the simulated annealing technique to get a better layout. Their algorithm tries to distribute vertices evenly, achieve uniform edge lengths and reflect symmetry. The main goals of the algorithm are speed and simplicity.

Fruchterman and Reingold compare their model with a molecular (atomic particles) or planetary simulation. They follow the approach of Eades and apply only attractive forces to neighboring vertices, but repulsive forces to all vertices.

The method defines attractive (f_a) and repulsive (f_r) forces as follows:

$$f_a(d) = \frac{d^2}{k}$$

$$f_r(d) = -\frac{k^2}{d}$$

where d is the distance between two vertices and k is the radius of the empty area around a vertex. k is calculated as follows:

$$k = C\sqrt{\frac{area}{numberofvertices}}$$

The constant C is found experimentally.

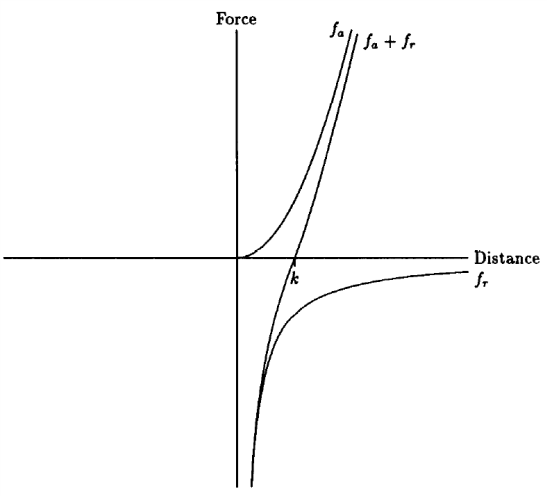


Figure 11: Forces versus distance (Taken from Battista (1999))

Figure 11 shows the mentioned forces versus distance (and their sum). As illustrated in the figure, k is the ideal distance where both attractive and repulsive forces are equal and cancel each other out. Fruchterman and Reingold tried several formulas to calculate attractive and repulsive forces. The ones used by Eades show similar results, but were not so effective, since f_a required a higher amount of computational time.

f_a and f_r used by Eades:

$$f_a(d) = k_a log d$$

$$f_r(d) = \frac{k}{d^2}$$

Their algorithm is similar to the one of Eades and requires the initial configuration to be fully or partly specified. Each iteration has the following main steps:

- Calculate the effect of attractive forces on each vertex
- Calculate the effect of repulsive forces on each vertex
- Limit the total displacement by the temperature

From a time complexity perspective, in each iteration the above algorithm computes $O(|E|)$ attractive and $O(N^2)$ repulsive forces. Fruchterman and Reingold described a variant of their algorithm using grid boxes ("grid-variant algorithm"). The basic idea behind is as follows. The screen is divided into grid squares and in each iteration each vertex is placed in its grid square and repulsive forces are only calculated between it and the vertices in the nearby squares of the grid. This method leads to a time complexity of $O(N)$.

Like the algorithm of Eades, the one of Fruchterman and Reingold can be applied to small graphs less than 40 vertices.

4.2 Frick et al. (1995)

Frick, Ludwig, and Mehldau (1995) proposed a new algorithm called GEM (graph embedder). The algorithm adds several new ideas to the ones discussed above. These are

- the concept of a local temperature (instead of global temperature)
- attraction of vertices towards their barycenter
- detection of oscillations and rotations

Frick et al. are the first ones applying an adaptive cooling schedule for force-directed placement techniques. They state

For each vertex, a local temperature is defined that depends on its old temperature and the likelihood that the vertex oscillates or is part of a rotating subgraph. Local temperatures raise if the algorithm determines that vertex is probably not close to its final destination.

The time complexity of the algorithm is $O(N^3)$. Their measurements show that the Fruchterman and Reingold algorithm is 4 times slower.

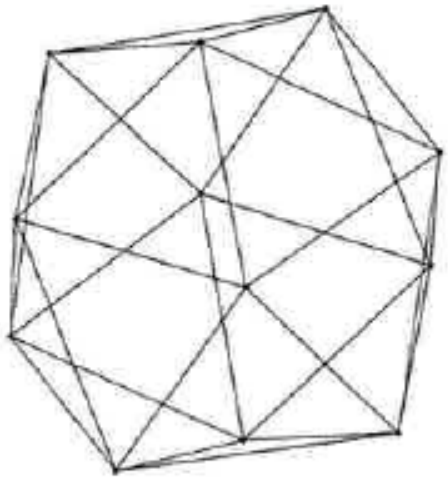


Figure 12: Icosahedron drawn using the GEM algorithm (Taken from Battista (1999))

5 Multi-level Algorithms

The mentioned algorithms above are suited for smaller graphs (1000 nodes or less). Hadany and Harel (1999) introduced a new approach for dealing with large graphs. Their method is called multi-scale, multi-level or multi-dimensional.

The basic principle behind the algorithm is that it operates on several "levels". In a first step it operates on an abstract level and considers just a rough layout of the graph (coarse-scale representation). Then in a next step finer details of the graph are considered. However, the rough layout of the graph must take essential properties and features for visualization into account.

According to Tamassia (2007), the general structure is as follows.

- Perform fine-scale relocations of vertices that yield a locally organized configuration
- Perform coarse-scale relocations (through local relocations in the coarse representations), correcting global disorders not found in stage 1.
- Perform fine-scale relocations that correct local disorders introduced by stage 2.

For step 2 (coarse-scale relocations) they use the energy function of Kamada and Kawai (1989). For step 3 (fine-scale relocations) force-directed calculations of Eades (1984), Fruchterman and Reingold (1991) or Kamada and Kawai (1989) can be used.

In 2000, Harel and Koren (2001) proposed an improvement that was able to handle graphs up to 15.000 vertices. The algorithm used a simpler coarsening process and a faster fine-scale beautification.

Gajer, Goodrich, and Kobourov (2000, 2001) proposed a new approach in the field of multi-level force-directed placement algorithms. Like Harel and Koren they used a simpler coarsening and reduced the quadratic time complexity of former force-directed algorithms. Besides a "filtration" and "neighborhood" technique, they didn't rely on randomized initial placement, but used an "intelligent" method of initial placement based on graph theoretic distances.

Another multi-level approach introduced in 2003 was the one of Walshaw (2001). Instead of using the algorithm of Kamada and Kawai, Walshaw applied the improvement of Fruchterman and Reingold (1991).

6 Multi-dimensional Scaling using FDP

In this section we present the applications of Force-directed Placement to the field of Multi-dimensional Projection, specifically, Multidimensional Scaling. i.e. , we discuss the role of FDP techniques used in graph drawing in the problem of visualizing high-dimensional datasets defined in $\mathbb{R}^m$-space in lower-dimensional space $\mathbb{R}^n$ where $n \ll m$ and $n \in \{1, 2, 3\}$.

When force-directed placement, as introduced by Fruchterman and Reingold (1991) (cf. 4.1), is applied to the general case of *full graphs* instead of simple, connected partial graphs, complexity becomes an issue.

In multidimensional scaling we regard similarity relationships between *all* high-dimensional data items. Hence, the layouting problem of the projected data based on their (dis)similarity essentially equals the problem of calculating a layout for a fully connected graph. In fact, general MDS has a strong connection to graph drawing. Performing MDS on a dataset is essentially equivalent to performing Kamada and Kawai (1989)'s energy-based graph layout on a complete graph whose vertices correspond to points in the dataset and whose edges are weighted by the high-dimensional distance between the corresponding points (Ingram et al., 2009b) (cf. 3.2).

In this case, a neighbourhood is comprised not only of a small subset, but the whole dataset, force calculations become expensive. The complexity of Fruchterman and Reingold (1991) becomes $O(N^3)$ with $O(N^2)$ per iteration. Applying the algorithm to datasets with > 100 nodes is not practical, particularly, with user interaction in mind.

FDP offers two alternatives where there is room for improving complexity: one can reduce the number of iterations necessary to reach the final layout, or reduce the complexity of each iteration.

An important aspect of iterative placement techniques is to decide when to stop the approximation process. Most commonly, the evaluation of a layout is performed using a metric based on the *mechanical stress* of the spring system. This is formulated as the residual sum of all inter-item distances with a normalization term:

$$Stress = \frac{\sum_{i<j} (\delta_{ij} - d_{ij})^2}{\sum_{i<j} (d_{ij})^2}$$

δ_{ij} being the (dis)similarity measure in high-dimensional space and d_{ij} being the Euclidian distance in the layout. The normalization tends to yield more compact layouts (Chalmers, 1996).

In the following sections we present various approaches which tackle the complexity problem.

6.1 Chalmers '96: $O(N^2)$ $[O(N)]$

Chalmers (1996) is an example of reducing complexity of the inner iteration. In their approach they reduce iteration time to linear complexity $O(N)$. The overall complexity becomes $O(N^2)$.

They presented a stochastically-based algorithm which uses data samples to identify nodes connected by spring-forces. Their algorithm is easy to implement, has low overhead, and produces good layouts.

6.2 Morrison et al. '02: $O(N^{\frac{3}{2}})$ $[O(N)]$

Morrison, Ross, and Chalmers (2003) were the first to introduce a *hybrid* approach to lower complexity. Their algorithm relies on stochastic sampling, spring models, and interpolation of data items in low-dimensional space.

The algorithm works as follows:

1. A random set S of size $\sqrt{N}$ data items is sampled
2. The sample is projected using Chalmers (1996)
3. Relative positions of missing items are interpolated

The interpolation process takes each missing data item and places it next to its closest member of the projected random sample considering desired distances. Then they refine the placement by applying the combined forces of yet another random sample from the original subset S to the inserted item (a constant number of times).

Compared to Chalmers (1996) their solution performs significantly faster and produces layouts of superior quality.

6.3 Morrison et al. '04: $O(N^{\frac{5}{4}})$ $[O(N)]$

In 2004 Morrison and Chalmers (2004) proposed a modification of their original hybrid MDS approach presented in Morrison et al. (2003) which improves overall complexity from $O(N^{\frac{3}{2}}))$ to $O(N^{\frac{5}{4}})$.

The new algorithm's structure stays basically the same. However, the original algorithm used computationally expensive nearest-neighbour searches for parent finding during the interpolation stage (cf. 6.2). This method is replaced by a novel approach to parent finding using randomly selected *pivots*. We quote (Morrison & Chalmers, 2004)

> *"...the complexity of this phase has been reduced by treating all high-dimensional relationships as a set of discretized distances to a constant number of randomly selected pivot items."*

Their results document a significant performance improvement as well as comparable layout quality.

A short summary of their algorithm (Morrison & Chalmers, 2004):

To form a layout of N multivariate objects :

1. *Select $\sqrt{N}$ subset of objects [$O(\sqrt{N})$]*
2. *Create 2D layout of subset using Chalmers' (Chalmers, 1996) linear per iteration spring model [$O(N)$]*
3. *Interpolate remaining objects onto the layout [$O(N\sqrt{N})$]*
   - *Find parent in sample for each remaining object [$O(N\sqrt{N})$]*
   - *Use high-dimensional distances to a $N^{\frac{1}{4}}$ sample (of the sample) to position remaining objects [$O(N)$]*
4. *Fine-tune layout with a constant number of iterations of Chalmers' spring model run on the full data set [$O(N)$]*

6.4 Jourdan et al. '04: $O(N \log n)$ $[O(N)]$

Jourdan and Melancon (2004) further refine the Pivot-based hybrid MDS algorithm presented by Morrison and Chalmers (2004) resulting in an even lower overall complexity of $O(NlogN)$, as opposed to Morrison and Chalmers (2004)' $O(N\sqrt{N})$. The performance increase is significant (cf. figure 13 while quality is maintained.

They achieve this by replacing the parent finding strategy introduced by Morrison and Chalmers (2004) by searches based on sorted lists. Hence, the parent finding task can be performed in $O(\log N)$ time.

The Jourdan and Melancon (2004) algorithm is considered the fastest algorithm available.

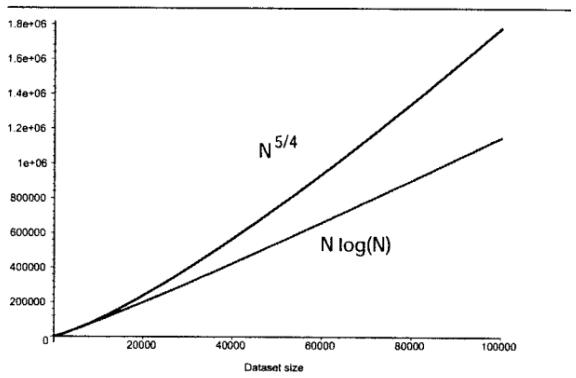


Figure 13: Comparition of $N^{\frac{5}{4}}$ and $N \log N$ curves (from Jourdan and Melancon (2004))

6.5 Ingram et al. '09: $O(N^2)$ GPU-based, massively parallel

Ingram, Munzner, and Olano (2009a) proposed Glimmer, a so called multilevel MDS algorithm exploiting modern graphics processing units (GPU), inspired by the hybrid MDS approaches by Chalmers (1996), Morrison and Chalmers (2004) and Jourdan and Melancon (2004).

They designed a entirely GPU-based algorithm which breaks down the input dataset into a hierarchy of multiple levels of subsets. At each level three operations are performed, namely, *restriction*, *relaxation* and *interpolation*. These operations are performed by a GPU-based parallel implementation of a stochastic force MDS solver (based on the ideas of Chalmers (1996)).

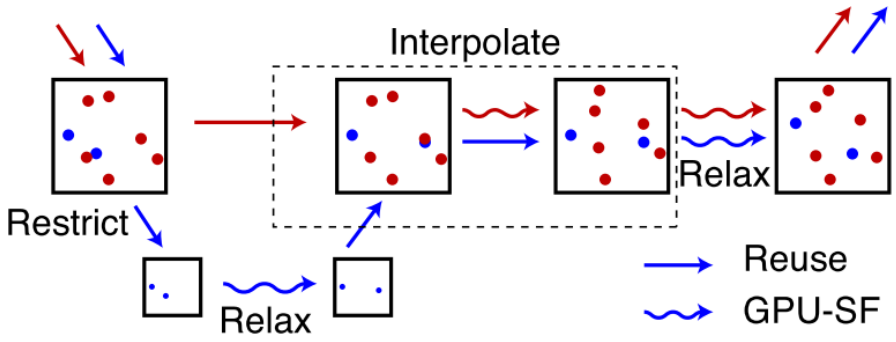


Figure 14: The Glimmer multilevel algorithm. The restriction operator builds the hierarchy by sampling points. GPU-SF (GPU-Stochastic Forces) is used as the relaxation operator at each level, with all points allowed to move, and as the interpolation operator, with only new points allowed to move. Lower levels untwist complex layouts while higher levels converge quickly because of computation at the lower levels. (Taken from Ingram et al. (2009))

The algorithms is less prone to getting stuck in local minima due to its multilevel hierarchical approach in contrast to the above discussed FDP-based MDS variants (cf. also Paulovich et al. (2008) on the matter). Although complexity is $O(N^2)$, the algorithm can compete with the fastest methods available since GPU parallelism significantly improves performance.

7 Visual Comparison of Selected Graph Drawing Techniques

7.1 Different Visualization of the Same Graph Drawn using Different Algorithms

See the figures 15 and 16.

7.2 Comparative Data

See the figures 17 , 18 and 19.

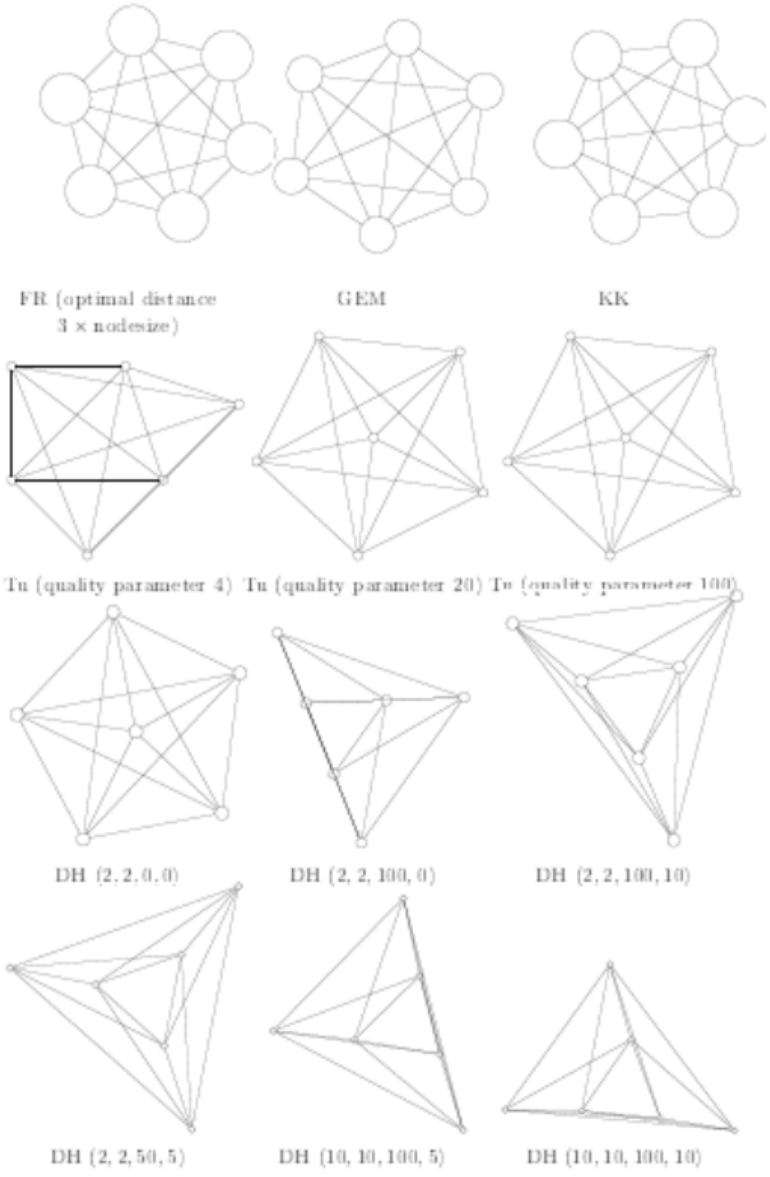


Fig. 5. Several drawings of the K_6

Figure 15: Graphs Drawn by Different Drawing Algorithms (Taken from Brandenburg, Himsolt, Rohrer)

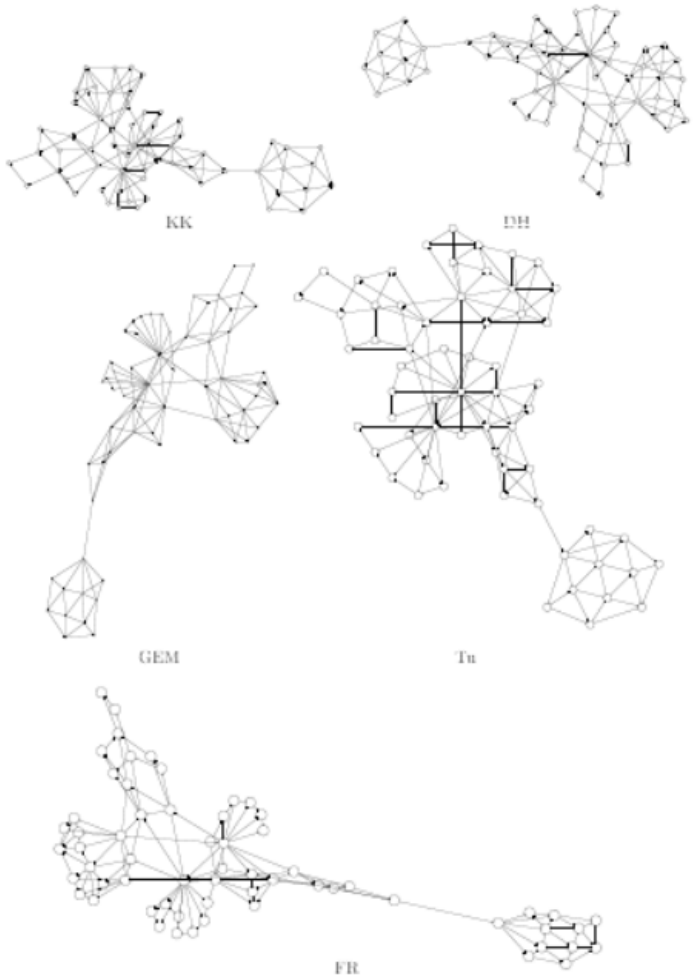


Fig. 6. Graphs from the graph drawing competition drawn with several algorithms

Figure 16: Graphs Drawn by Different Drawing Algorithms with More Nodes (From Brandenburg, Himsolt, Rohrer)

Criteria	Battista et al. (1994)	Eades (1984)	and Kawai (1989)	and Reingold (1991)	and Harel (1996)	NicheWorks (1997)
Symmetric	✓	✓	✓			
Evenly distributed nodes	✓		✓	✓	✓	Clustered
Uniform edge lengths	✓	✓	✓	✓	✓	Weights
Minimized edge crossings	✓		✓	✓	✓	

Figure 17: Comparing Graphs by Criteria (Chen Chaomei, Graph-Drawing Algorithms)

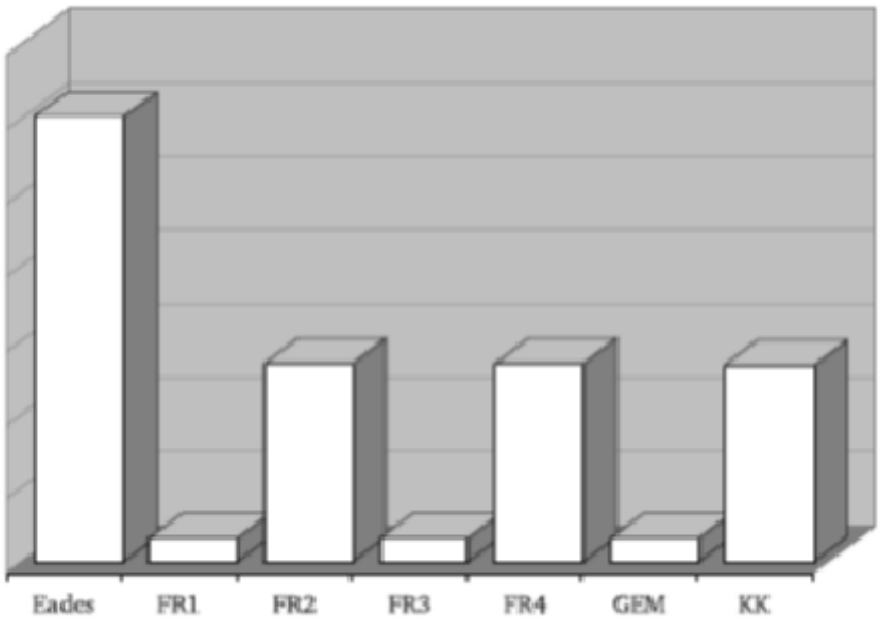
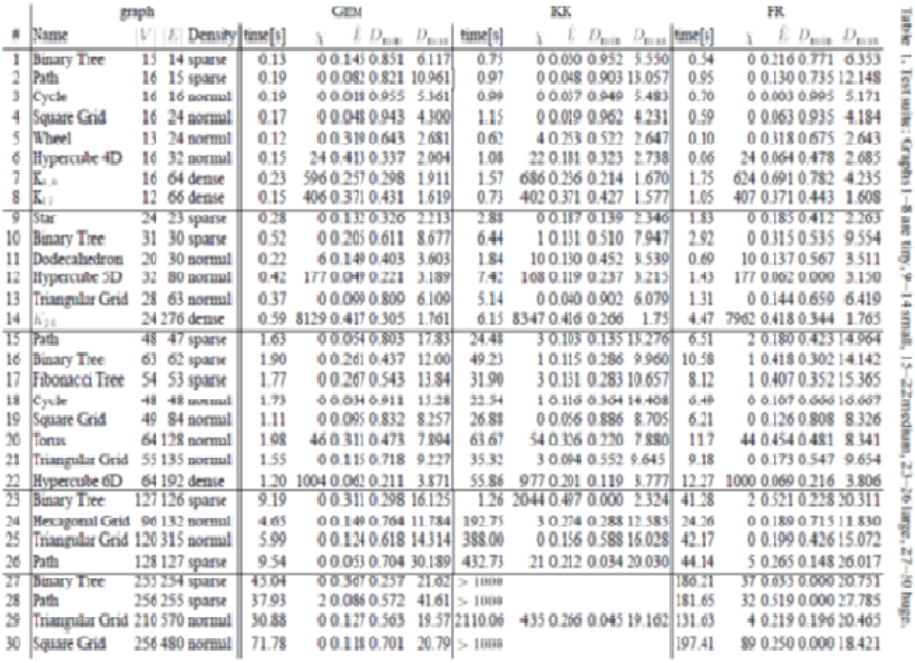


Figure 18: Comparing Graphs to Each Other Regarding Performance; Less is Better; (From Diplomarbeit by Michael Forster Universität Passau



#	graph Name	$\|V\|$	$\|E\|$	Density	GEM time[s]	GEM $\bar{\chi}$	GEM $\bar{L}$	GEM D_{min}	GEM D_{max}	KK time[s]	KK $\bar{\chi}$	KK $\bar{L}$	KK D_{min}	KK D_{max}	FR time[s]	FR $\bar{\chi}$	FR $\bar{L}$	FR D_{min}	FR D_{max}
1	Binary Tree	15	14	sparse	0.13	0	0.143	0.851	6.117	0.75	0	0.030	0.952	5.550	0.54	0	0.216	0.771	6.353
2	Path	16	15	sparse	0.19	0	0.082	0.821	10.961	0.97	0	0.048	0.903	13.057	0.95	0	0.130	0.735	12.148
3	Cycle	16	16	normal	0.19	0	0.018	0.955	5.361	0.99	0	0.037	0.949	5.483	0.70	0	0.003	0.995	5.171
4	Square Grid	16	24	normal	0.17	0	0.048	0.943	4.300	1.15	0	0.019	0.962	4.231	0.59	0	0.063	0.935	4.184
5	Wheel	13	24	normal	0.12	0	0.310	0.643	2.681	0.62	4	0.253	0.522	2.647	0.10	0	0.318	0.675	2.643
6	Hypercube 4D	16	32	normal	0.15	24	0.413	0.337	2.004	1.08	22	0.181	0.323	2.738	0.66	24	0.064	0.478	2.685
7	$K_{8,8}$	16	64	dense	0.23	596	0.257	0.298	1.911	1.57	686	0.256	0.214	1.670	1.75	624	0.691	0.782	4.235
8	K_{12}	12	66	dense	0.15	406	0.371	0.431	1.619	0.73	402	0.371	0.427	1.577	1.05	407	0.371	0.443	1.608
9	Star	24	23	sparse	0.28	0	0.132	0.326	2.213	2.88	0	0.187	0.139	2.346	1.83	0	0.185	0.412	2.263
10	Binary Tree	31	30	sparse	0.52	0	0.205	0.611	8.677	6.44	1	0.131	0.510	7.947	2.92	0	0.315	0.535	9.554
11	Dodecahedron	20	30	normal	0.22	6	0.149	0.403	3.603	1.84	10	0.130	0.452	3.539	0.69	10	0.137	0.567	3.511
12	Hypercube 5D	32	80	normal	0.42	177	0.049	0.221	3.189	7.42	108	0.119	0.237	3.215	1.43	177	0.092	0.090	3.150
13	Triangular Grid	28	63	normal	0.37	0	0.069	0.800	6.109	5.14	0	0.040	0.902	6.079	1.31	0	0.144	0.650	6.419
14	$K_{2,12}$	24	276	dense	0.59	8129	0.417	0.305	1.761	6.13	8347	0.416	0.266	1.75	4.47	7962	0.418	0.344	1.765
15	Path	48	47	sparse	1.63	0	0.054	0.803	17.83	24.48	3	0.103	0.135	13.276	6.51	2	0.180	0.423	14.964
16	Binary Tree	63	62	sparse	1.90	0	0.261	0.437	12.00	49.23	1	0.115	0.286	9.960	10.58	1	0.418	0.302	14.142
17	Fibonacci Tree	54	53	sparse	1.77	0	0.267	0.543	13.84	31.90	3	0.131	0.283	10.657	8.12	1	0.407	0.352	15.365
18	Cycle	48	48	normal	1.73	0	0.034	0.911	13.28	22.54	1	0.116	0.364	14.408	6.49	0	0.167	0.666	16.607
19	Square Grid	49	84	normal	1.11	0	0.095	0.832	8.257	26.88	0	0.056	0.886	8.705	6.21	0	0.126	0.808	8.326
20	Torus	64	128	normal	1.98	46	0.311	0.473	7.894	63.67	54	0.326	0.220	7.880	11.7	44	0.454	0.481	8.341
21	Triangular Grid	55	135	normal	1.55	0	0.115	0.718	9.227	35.32	3	0.094	0.552	9.645	9.18	0	0.173	0.547	9.654
22	Hypercube 6D	64	192	dense	1.20	1004	0.062	0.211	3.871	55.86	977	0.201	0.119	3.777	12.27	1000	0.069	0.216	3.806
23	Binary Tree	127	126	sparse	9.19	0	0.311	0.298	16.125	1.26	2044	0.497	0.000	2.324	41.28	2	0.521	0.228	20.311
24	Hexagonal Grid	96	132	normal	4.65	0	0.149	0.764	11.784	192.75	3	0.274	0.288	12.585	24.26	0	0.189	0.715	11.830
25	Triangular Grid	120	315	normal	5.99	0	0.124	0.618	14.314	388.00	0	0.156	0.588	16.028	42.17	0	0.199	0.426	15.072
26	Path	128	127	sparse	9.54	0	0.053	0.704	30.189	432.73	21	0.212	0.034	20.030	44.14	5	0.265	0.148	26.017
27	Binary Tree	255	254	sparse	45.04	0	0.307	0.257	21.02	> 1000					186.21	37	0.035	0.000	20.751
28	Path	256	255	sparse	37.93	2	0.086	0.572	41.61	> 1000					181.65	32	0.519	0.000	27.785
29	Triangular Grid	210	570	normal	30.88	0	0.127	0.563	19.57	2110.06	435	0.268	0.045	19.162	131.63	4	0.219	0.196	20.465
30	Square Grid	256	480	normal	71.78	0	0.118	0.701	20.79	> 1000					197.41	89	0.250	0.000	18.421

Table 1: Test suite: Graphs 1–8 are tiny, 9–14 small, 15–22 medium, 23–26 large, 27–30 huge.

Figure 19: Detailed Comparison of the Three Different Approaches (Spring, Annealing, Optimal Path; From Frick, Ludwig, Mehldau, Universitaet Karlsruhe)

8 References

Battista, G., Eades, P., Tamassia, R., & Tollis, I. G. (1999). *Graph drawing algorithms for the visualization of graphs*. Prentice Hall.

Borg, I., & Groenen, P. J. F. (2005). *Modern multidimensional scaling* (Second ed.). Springer.

Brandenburg, F.-J., Himsholt, M., & Rohrer, C. (1995). An experimental comparison of force-directed and randomized graph drawing algorithms. *Proceedings of the Symposium on Graph Drawing*.

Brandes, U. (2001). Drawing graphs. In M. Kaufmann & D. Wagner (Eds.), (pp. 71–86). London, UK: Springer-Verlag. Available from http://portal.acm.org/citation.cfm?id=376944.376948

Branke, J. (1996). Drawing graphs using genetic algorithms. *Manuscript*.

Branke, J., Bucher, F., & Schmeck, H. (1996). Using genetic algorithms for drawing undirected graphs. In *The third nordic workshop on genetic algorithms and their applications* (pp. 193–206).

Chalmers, M. (1996, October). A linear iteration time layout algorithm for visualising high-dimensional data. In *Proc. visualization'96* (pp. 127–132). San Francisco, California, USA. Available from http://www.dcs.gla.ac.uk/~matthew/papers/vis96.pdf

Chen, C. (2006). *Information visualization: Beyond the horizon*. Springer, Berlin; Auflage: 2nd ed. 2004. 2nd printing. (31. Mai 2006).

Coleman, M. K., & Parker, D. S. (1996, December). Aesthetics-based graph layout for human consumption. *Softw. Pract. Exper.*, *26*, 1415–1438. Available from http://portal.acm.org/citation.cfm?id=246460.246473

Cox, T., & Cox, M. (2001). *Multidimensional scaling, second edition*. Chapman & Hall/CRC.

Crocker, N., McGuffin, R., & Naylor, R. (1972). Computer - aided placement for high - density chip - interconnection system. In *Electronics letters* (pp. 503–504).

Cruz, I. F., & Twarog, J. P. (1996). 3d graph drawing with simulated annealing. In *Proceedings of the symposium on graph drawing* (pp. 162–165). London, UK: Springer-Verlag. Available from http://portal.acm.org/citation.cfm?id=647547.728601

Davidson, R., & Harel, D. (1996, October). Drawing graphs nicely using simulated annealing. *ACM Trans. Graph.*, *15*, 301–331. Available from http://doi.acm.org/10.1145/234535.234538

Eades, P. (1984). A heuristic for graph drawing. *Congressus Numerantium*, *42*, 149–160. Available from http://www.cs.usyd.edu.au/~peter/old_spring_paper.pdf

Eades, P., & Tamassia, R. (1988). *Algorithms for drawing graphs: An annotated bibliography* (Tech. Rep.). Providence, RI, USA.

Fisk, C. J., Caskey, D. L., & West, L. E. (1967). Accel: Automated circuit card etching layout. *Proceedings of The IEEE*, *55*, 1971–1982.

Forster, M. (1999). *Zeichnen ungerichteter Graphen mit gegebenen Knotengrössen durch ein Springembedder-Verfahren*. Unpublished master's thesis, Universiät Passau, Germany. Available from http://www.michael-forster.de/publications/dipl.pdf

Frick, A., Ludwig, A., & Mehldau, H. (1995). A fast adaptive layout algorithm for undirected graphs. In *Proceedings of the dimacs international workshop on graph drawing* (pp. 388–403). London, UK: Springer-Verlag. Available from http://portal.acm.org/citation.cfm?id=647546.730941

Fruchterman, T. M. J., & Reingold, E. M. (1991). Graph drawing by force-directed placement. *Software - Practice and Experience*, *21*, 1129–1164.

Gajer, P., Goodrich, M. T., & Kobourov, S. G. (2000). A fast multi-dimensional algorithm for drawing large graphs. In *In graph drawing'00 conference proceedings* (pp. 211–221).

Gajer, P., Goodrich, M. T., & Kobourov, S. G. (2001). A multi-dimensional approach to force-directed layouts of large graphs. In *Proceedings of the 8th international symposium on graph drawing* (pp. 211–221). London, UK: Springer-Verlag. Available from http://portal.acm.org/citation.cfm?id=647552.729396

Hadany, R., & Harel, D. (1999). A multi-scale algorithm for drawing graphs nicely. In *Proceedings of the 25th international workshop on graph-theoretic concepts in computer science* (pp. 262–277). London, UK: Springer-Verlag. Available from http://portal.acm.org/citation.cfm?id=647680.732318

Harel, D., & Koren, Y. (2001). A fast multi-scale method for drawing large graphs. In *Proceedings of the 8th international symposium on graph drawing* (pp. 183–196). London, UK: Springer-Verlag. Available from http://portal.acm.org/citation.cfm?id=647552.729397

Ingram, S., Munzner, T., & Olano, M. (2009a). Glimmer: Multilevel mds on the gpu. *IEEE Transactions on Visualization and Computer Graphics*, *15*, 249–261.

Ingram, S., Munzner, T., & Olano, M. (2009b, March). Glimmer: Multilevel MDS on the GPU. *IEEE Transactions on Visualization and Computer Graphics (TVCG)*, *15*(2), 249–261.

Jourdan, F., & Melancon, G. (2004). Multiscale hybrid mds. In *Proceedings of the information visualisation, eighth international conference* (pp. 388–393). Washington, DC, USA: IEEE Computer Society. Available from http://portal.acm.org/citation.cfm?id=1018435.1021647

Kamada, T., & Kawai, S. (1988). Automatic display of network structures for human understanding. *Information Processing Letters*.

Kamada, T., & Kawai, S. (1989, April). An algorithm for drawing general undirected graphs. *Inf. Process. Lett.*, *31*, 7–15. Available from http://dx.doi.org/10.1016/0020-0190(89)90102-6

Kobourov, S. G. (2007). *Handbook of graph drawing and visualization (discrete mathematics and its applications)* (R. Tamassia, Ed.). Chapman & Hall/CRC. Available from http://www.cs.brown.edu/~rt/gdhandbook/chapters/force-directed.pdf

Kosak, C., Marks, J., & Shieber, S. M. (1991). A parallel genetic algorithm for network-diagram layout. In *Icga* (p. 458-465).

Kruskal, J., & Wish, M. (1978). *Multidimensional scaling*. Sage Publications.

Mendonca, X. (1994). A system for drawing conceptual schema diagrams. *PhD Thesis*.

Monien, B., Friedhelm, R., & Salmen, H. (1996). A parallel simulated annealing algorithm for generating 3d layouts of undirected graphs. In *Proceedings of the symposium on graph drawing* (pp. 396–408). London, UK: Springer-Verlag. Available from http://portal.acm.org/citation.cfm?id=647547.728590

Morrison, A., & Chalmers, M. (2004, May). A pivot-based routine for improved parent-finding in hybrid mds. *Information Visualization*, *3*(2), 109–122.

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

Paulovich, F. V., Nonato, L. G., Minghim, R., & Levkowitz, H. (2008, May). Least square projection: A fast high-precision multidimensional projection technique and its application to document mapping. *IEEE Transactions on Visualization and Computer Graphics*, *14*(3), 564–575. Available from http://www.lcad.icmc.usp.br/~nonato/pubs/lsp.pdf

Quinn, N., & Breur, M. (1979). A force directed component placement procedure for printed circuit boards. In *Ieee transactions on circuits and systems* (pp. 377–388).

Rosete, A. (1997). Drawing graphs using genetic algorithms. *Manuscript*.

Scanlon, F. T. (1971). Automated placement of multi-terminal components. In *Proceedings of the 8th design automation workshop* (pp. 143–154). New York, NY, USA: ACM. Available from http://doi.acm.org/10.1145/800158.805068

Sugiyama, K., & Misue, K. (1995a). Graph drawing by magnetic spring model. In *Journal of visual languages and computing* (pp. 217–231).

Sugiyama, K., & Misue, K. (1995b). A simple and unified method for drawing graphs: Magnetic-spring algorithm. In *Proceedings of the dimacs international workshop on graph drawing* (pp. 364–375). London, UK: Springer-Verlag. Available from http://portal.acm.org/citation.cfm?id=647546.730940

Tamassia, R. (2007). *Handbook of graph drawing and visualization (discrete mathematics and its applications)*. Chapman & Hall/CRC.

Tejada, E., Minghim, R., & Nonato, L. G. (2003, December). On improved projection techniques to support visual exploration of multidimensional data sets. *Information Visualization*, *2*, 218–231. Available from http://portal.acm.org/citation.cfm?id=982444.982447

Tunkelang, D. (1993). A layout algorithm for undirected graphs. In *Proceedings of graph drawing '93.*

Tutte, W. T. (1963). How to draw a graph. *Proceedings of The London Mathematical Society*, *s3-13*, 743–767.

Vose, M. D. (1998). *The simple genetic algorithm: Foundations and theory.* Cambridge, MA, USA: MIT Press.

Walshaw, C. (2001). A multilevel algorithm for force-directed graph drawing. In *Proceedings of the 8th international symposium on graph drawing* (pp. 171–182). London, UK: Springer-Verlag. Available from http://portal.acm.org/citation.cfm?id=647552.729414