GRAPH DRAWING WITH SPRING ALGORITHM

SiXin Cheng

A Major Report

in

The Department

of

Computer Science

Presented in Partial Fulfillment of the Requirements
for the Degree of Master of Science at
Concordia University
Montreal, Quebec, Canada

July 2002

0-612-72930-3

ABSTRACT

Graph Drawing with Spring Algorithm

SiXin Cheng

Graph drawing research investigates the automatic methods for visual representation of graph. The spring algorithm draws graph layout by simulating physical model. It is easy to understand and implement, and it generally produces a pleasing layout. A spring algorithm with new termination method is introduced in this report. The spring algorithm is applicable to undirected, connected simple graphs. It can be used to produce graph layout with uniform length of edge and evenly distributed vertices. The algorithm starts from random initial configuration. Then, it iteratively minimizes the force acting on a vertex until it reaches the upper bound of the force. Finally, the algorithm stops when the remaining force is under the lower bound of the force.

Acknowledgements

The author is extremely thankful to his supervisor Dr. Peter Grogono for his professional supervision. intellectual guidance. and patience. through this study. It was a pleasure working with him.

This report is dedicated to the author's wife ChunHua Zhang. father and mother. sister. sister in law for their help. lover and encouragement to achieve excellence. Without them this would not have been possible.

1. Introduction

A picture is worth of a thousand words

1.1. Information visualization

The visualization of abstract, complex conceptual, and large structures is the most important component of support tools for many applications in science and engineering. The representation exploits human visual processing to reduce the cognitive load of many tasks that require understanding of overall global structure or local detail contents.

Graph made up by vertices and edges are abstraction that can model objects and some interesting information among those objects. It is very important in computer science domain. It has been applied in most segments of software industry to represented information. In addition to Object Oriented Analysis Design, modeling networks of chemical reactions, etc., the animation of sorting algorithm and neural-network simulation are efficient ways in training and education. Graph has rich theoretical support is another reason that make it useful.

Information visualization involves drawing an easy to read and understand layout in either 2D or 3D space. In information visualization applications, the usefulness of a layout depends on its readability, that is, the capability of conveying the meaning of the diagram quickly and clearly. Various *graphic standards* have been proposed for drawing graphs, depending on the application. Usually, the vertices are represented by circles, and the edges are represented by a simple open curve between the vertices. Both vertices and edges can be stereotypes. A *straight-line drawing* maps each edge into a straight-line segment. A *polyline* drawing represents the edge by a polygonal chain. Figure 1 represents a class diagram.

The quality of a layout is a subject matter, but several aesthetics are generally accepted. A fundamental and classical aesthetic is the minimization of crossings between edges. Symmetry is another most important aesthetics for delivering an easily understandable graph layout. Minimizing the number of bends in the edges and minimizing variance edge length are specific aesthetics for certain approaches, such as orthogonal drawing.

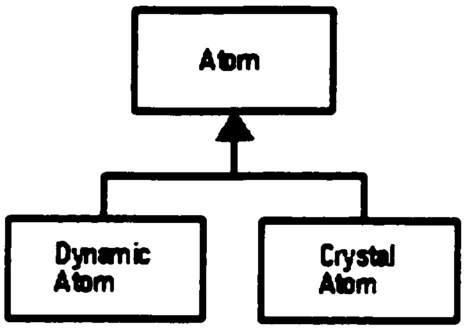


Figure 1. A class diagram

There are three approaches in graph drawing. Orthogonal, hierarchy and Force-directed approach. Spring algorithm, which falls into the category of Force-directed approach, is easy to understand and implement. It gives graph layout with high qualities such as uniform edge length, evenly distributed nodes [1] and symmetry [2].

1.2. Contributions

The major contribution of this report is to apply spring algorithm to draw quality graph layout in three-dimensions space and introduce a new termination method that uses upper and lower bound of estimate the quality and terminating the algorithm. The algorithm is able to generate quality graph layout with more than 40 vertices and 60 edges within 10 seconds. Other contributions include parameter analysis and studies and minimizing the numbers of iteration.

1.3. Outline of the report

The organization of this report is as follows: in Chapter 2, we review the three approaches on the graph drawing. Chapter 3 describes the various spring algorithms and presents the spring algorithm used in this report. Chapter 4 covers the problem and the solution for the spring algorithm. Chapter 5 presents the experimental results of the graph layout created by the spring algorithm. Finally, Chapter 6 presents the conclusion of the report and the future work on searching solutions on large graph drawing.

2. Background

The purpose of the graph drawing research field is to investigate automatic methods for visual representation of graph.

2.1. Graph

An undirected graph is a pair of $G = (V, E)$, where V is a finite and non-empty set of elements call vertices (or node) of G, and E is a finite set of elements called edge (or arcs) of G. An edge $e \in E$ is an unordered pair $\{u, v\}$ of vertices of G; vertices u and v are called end-vertices of e, or simpler vertices of e.

An edge of graph G is self-loop if its end vertices coincide. A graph G contains multiple edges if it has two or more edges with the same end-vertices. A graph G is simple is it has neither self-loop nor multiple edges.

A path p between two vertices v_0 to v_k, $k \geq 1$, of a graph G is an alternating sequence v_0, e_0, v_1, e_1, ..., v_k, e_k of vertices and edges of G, where $e_i = (v_{i-1}, v_i)$, $(I = 1, \ldots, v_k)$. A connected graph is a graph where any two vertices are joined by a path. A sub-graph G' of a graph $G = (V, E)$ is a graph $G = (V', E')$ such that $V' \subseteq V$ and $E' \subseteq E$.

2.2. Three approaches in Graph Drawing

Topology-shape approach, hierarchical approach, and the force-directed approach are the most commonly approaches to 2D graph layout. In the first two approaches, potential conflicts between drawing aesthetics are handled by decomposing the drawing procedure into a sequence of steps. Each step produces a refinement of drawing while satisfying one of more of the drawing aesthetics. In the force-oriented approach, the satisfaction of the drawing aesthetics is transferred to optimizing forces in a physical model.

2.2.1. Topology-shape approach

Topology-shape approach constructs orthogonal drawing. An orthogonal drawing is a polyline drawing that maps each edge into a chain of horizontal and vertical segments. Many applications, such as circuit layout, software engineering case tools use or orthogonal drawing. An orthogonal drawing is shown in the following figure:

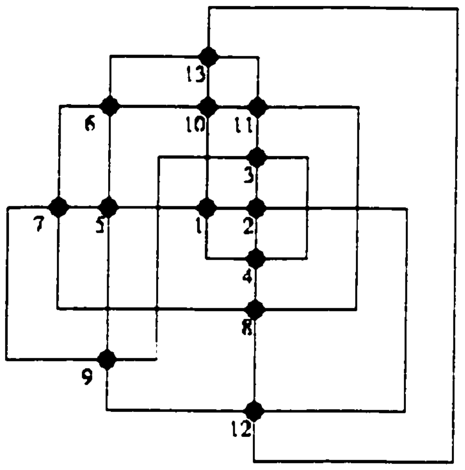


Figure 2.1: An orthogonal drawing

Topology and shape are properties of orthogonal drawings which establish equivalence classes in orthogonal drawings of the same graph: two orthogonal drawings have the same topology if one can derived from other by continuous deformation which does not alter the sequence of edges around closed loops in the drawing, and two orthogonal drawings have the same shape if they have the same topology, and are related by changes of edges in length.

There are planar orthogonal drawing and non-planar orthogonal drawing. A planar orthogonal drawing of a planar graph requires minimizing number of bends, or minimizing area. Tamassia uses network flow techniques to give a polynomial-time algorithm for minimizing bends when the graph and fixed embedding are given as input [3]. Lower bounds for planar orthogonal drawings of graphs are described in [4]. On the other hand, a linear-time heuristic algorithm for bend minimization is presented in [5]. A non-planar orthogonal drawing requires minimization of drawing area and low number of bends. Investigation on non-planar orthogonal drawing are documented in [6] and [7].

2.2.2. The hierarchical approach

The hierarchical approach is particularly suited to acyclic directed graphs. The vertices of an acyclic directed graph can always be distributed among a sequence of ordered layers so that all edges can be drawn with a downward (or upward) component. The arrangement generally gives a much clearer display of the relational information represented by the graph with arbitrary edge orientations.

The basic approach is to partition the graph into vertex-disjointed sets, which are assigned to different layers in the drawing. The vertex partition is chosen so that all edges in the drawing have the same orientation when the layers are arranged as parallel equal spaced line on the page. The four aesthetic constraints that hierarchical layout algorithm should satisfy are:

- Edges should be oriented in the same general orientation.
- Vertices should be assigned to same layers corresponding to parallel, equal-spaced lines.
- Edge crossing should be avoided, and

- Edges should be as straight as possible.

An example of hierarchical drawing style for directed graph is in figure 3:

2.2.3. Force-directed approach

Force-directed approach generates a mathematical description of a physical model of the graph. The drawing aesthetics are indirectly expressed as properties of. and constraints on the physical model. At the state of equilibrium. the model approximately satisfies the readability and understandability. The force-oriented approach has a venerable place in the graph layout. It is easy to understand and implement. and it generally produces pleasing layout. However. this approach typically takes a long time to converge to equilibrium.

Eades presented the first spring embedder in [1]. Since then several authors have presented variations of the original technique. The most recent techniques have appeared in [4] and [8]. Another technique that seems to produce nice drawings is simulated annealing.

2.2.3.1. Spring and Electrical Force

Spring and Electrical Force is simplest and most popular force-directed method. Edges of the graph are springs that connect the nodes. and nodes are charged particles. The force defined on a node is defined as follows:

$$F(v) = \sum_{(u,v)\in E} f_a(d_{uv}) * \left(\frac{u-v}{d_{uv}}\right) - \sum_{(u,v)\in V^2} f_r(d_{uv}) * \left(\frac{u-v}{d_{uv}}\right) \quad (2.1)$$

Where d_{uv} is the distance between u and v, and f_a(the attractive factor) and f_r(the repulsive factor) are based on the Hooke's law and electrical repulsive respectively which are defined as follows

$$f_a(d_{uv}) = k_{uv}^{(1)}(d_{uv} - l_{uv}) \tag{2.2}$$

$$f_r(d_{uv}) = \frac{k_{uv}^{(2)}}{d_{uv}^2} \tag{2.3}$$

where l_{uv} is the natural length of the spring between u and v, $k_{uv}^{(1)}$ is the stiffness of the spring between u and v, and $k_{uv}^{(2)}$ indicates the strength of the repulsive force between u and v.

The spring force between u and v makes length of the edge equal to l_{uv}. The repulsive electrical forces keep nodes that are not connected directly far from each other.

2.2.3.2. Barycenter Method

Similar to the "Spring and Electrical Force" method, Barycenter Method introduced by Tutte [9], does not have electrical charge, the stiffness of each spring is set to one, and the length of each spring is set to zero. Therefore, the force acting on the vertex v is given by

$$F(v) = \sum_{(u,v)\in E}(u - v) \tag{2.4}$$

Equation 2.4 results in an undesirable equilibrium state where all the nodes are located at the origin. To solve this, a set of at least three vertices is fixed at the corners of a convex polygon. The rest of the vertices are free to move. The algorithm is an iterative. In each iteration. a new location is computed for each free vertex using equation 2.4. The algorithm terminates when equilibrium reached.

2.3. 3D Graph Layout

Most research in graph layout to date has been directed towards the problem of drawing graphs in two dimensions. An indication of the interest in 2D graph layout during the last decade is more than 300 papers in Annotated Bibliography on Graph drawing [10]. Although the most research attention has focused on 2D layout. there are good reasons that making use of 3D for graph layout can offer benefits in visualizing information. Three-dimension display adds more freedom in placing vertices compare to 2D graph layout. In addition. some information may be more naturally represented in three-dimension. For examples in Figure 4. the same graph displays in two different ways. Figure 1.4(b) presented the graph in three dimensions. The with symmetric and equal length edges graph is a better presentation of a cube comparing to 2D drawing in the Figure 1.4(a).

Experimental studies in [11] and [12] indicated that the error rate in identifying connectivity between vertices using 3D picture of a random graph is roughly linearly proportional to the number of vertices. Compare to the 2D picture. the error rate is reduced by a factor of about three times by displaying the graph in 3D. Also. graphs with three times number of vertices can be displayed at about the same error rate by using 3D.

Viewpoint rotations form a set of transformations for understanding a structure in 3D. It is a simple way for user-controlled placement of a designated vertex. Finally. display devices and hardware supporting 3D graphical display are becoming more common.

2.4. Summary

This chapter presents the three approaches for graph drawing. It lists some reasons why drawing graph in 3D is attractive. In next chapter, we first give the literature review on spring algorithm. Then we present the spring algorithm using in this report.

3. The 3D Spring Algorithm

The spring algorithm has an important place in the relatively short history of the graph drawing. It is easy to understand and implement, and it generally produce a pleasing layout, which display uniform length of edge and the symmetries of the graph [2]. Although spring embedder algorithms introduced by [1] in 1984 is proposed for the 2D graph layout of general undirected graphs, various spring algorithm has been introduced and applied to the 3D graph layout [2], [13] and [14].

The spring algorithm draws graph layout by simulating a physical model. The physical model is mapped to the input graph, and the drawing of the graph layout becomes a problem of finding equilibrium of the physical model.

Discussed in chapter 2, the aesthetics of the graph layout can be implemented as mechanical constraints. It is suggested in [15] that constraining the vertices to lie on simple 3D surfaces can improve the readability of the graph layout.

The outline for this chapter is as follows. In section 3.1, we present the literature review on various spring algorithms. In Section 3.2, we introduce an extended spring algorithm, which is an extension of Eades [1] on drawing 3D graph layout.

3.1. Background

This section gives a literature reviews on various spring algorithms.

3.1.1. Eades' Spring Algorithm

Eades introduce the first spring algorithm for graph drawing with electrical forces [1]. The algorithm is proposed for 2D graph layout. In this algorithm, Eades used logarithmic strength springs in place of Hooke's law, and inverse square law for repulsive forces. In addition, there is repulsive force only for non-adjacent vertices. The following is Eades's algorithm detail as stated in [1].

```
Algorithm SPRING(G:graph)
      Place vertices of G in Random locations;
            Repeat M times
                  Calculate the force on each vertex;
                  Move the vertex C4*(force on vertex)
            End \\ Repeat
      Draw graph
End SPRING
```

Where $C4 = 0.1$ and $M = 100$. This algorithm is the base for subsequent spring algorithm. The major drawbacks of this algorithm are the high cost of computing the attractive and repulsive factors, and the absence of repulsive forces between adjacent nodes, which can result in concentration of a number of adjacent nodes in a small area. Further more, this algorithm uses simple stop methods for stopping and for choosing the amount of displacement of nodes, which are non-adaptable to the graph and node qualities [16].

3.1.2. FR Algorithm

Fruchterman and Reingold extended Eades' work by introducing new attractive, repulsive factors, a temperature and a cooling schedule in an algorithm that is known as FR [13]. In this algorithm, the force acting on a vertex v is computed using Equation 2.1. However, FR uses the following attractive and repulsive factors, which can produce results similar to Eades' layouts but are more efficient to compute

$$f_a(d_{uv}) = \frac{d_{uv}^2}{k} \qquad (3.1)$$

$$f_r(d_{uv}) = \frac{-k^2}{d_{uv}} \qquad (3.2)$$

Within each iteration of FR, forces on all nodes are computed, and then all the nodes are moved at once. The algorithm limits the maximum displacement to a temperature t where t is computed by a cooling function for all nodes at the end of each iteration. Starting from an initial value, the temperature decrease in an inverse linear fashion. It is notice that a better cooling schedule can dramatically change resulting layouts and reduce the number of iterations required to find the graph layout. To speed up their algorithm, they use grid-variant algorithm where drawing area is divided into grids. The computation of attractive forces remains the same, but for each vertex the repulsive force is only computed between the vertex and the nodes in its close-by cells of the grid. This method has not been very popular mainly due to its dependency on the distribution of nodes within the grid, the overhead of placing nodes in grid cells in each iteration, and the compromised layout quality that it produces in some cases.

FR terminates after a maximum number of iterations is exhausted. Since the number of the iterations for different types of graphs is uncertain, this stopping scheme may terminate too early or too late depending on the graph. Graphlet [17] extends FR by introducing new terminating conditions: the maximum overall force magnitude of all nodes falls below 3 or if the maximum number of iterations is reached. The experimental results in [16] show Graphlet's method of stopping works only for the graph with less than 50 nodes. In addition, FR makes use of a drawing frame that limits the nodes' displacements. However, most algorithms including the Graphlet version of FR do not add this extra restriction on the displacement of nodes, and use rescaling to draw the final layout in a frame of a fixed size.

3.1.3. GEM Algorithm

GEM [15], a short form for graph embedder, is a Spring algorithm based on Eades' algorithm, and is the first one that uses the history of a node's movement to choose the temperature for the current displacement of the node. Unlike FR, this algorithm moves the nodes one at a time in each iteration. The following is an outline of this algorithm:

```
Algorithm GEM(G:graph)
    WHILE T-global > T-min and #round < R-max DO
        Find a random permutation of vertices;
        FOR each vertex v from the random permutation
            v's impulse =
                Attraction to center of gravity +
                Random disturbance +
                Repulsive forces + Attractive forces;
            Update v's position limited by its individual
            temperature;
            Update v's temperature;
        END FOR
    END WHILE
END GEM
```

where T-min is the minimum global temperature, R-max is the maximum number of rounds, and T-global is defined as the average of the local temperatures of all vertices. The local temperature for each vertex is computed based on the old local temperature and the likelihood that the vertex is oscillating or rotating. GEM determines whether a node is oscillating, rotating, or moving towards its final destination based on the directions of movements in the previous displacements of the node. In case of oscillation or rotation, the local temperature of the node is decreased, and otherwise is raised. The experimental results in [16] show that for all of our test graphs GEM fails to stop before R-max is reached. In addition starting from random graph layouts, GEM finds drawings with many edge crossings for large planar graphs. We must note that GEM introduces an initial layout algorithm named insert that inserts the vertices one-by-one in an initial round. The layouts of some types of graphs such as binary trees and grids generated by GEM may benefit from an initial layout produced by this insert algorithm.

3.1.4. Modeling Graph Theoretic Distances: KK

Kamada and Kawai introduced an algorithm based on Graph-Euclidean and graph-theoretic distances (the shortest distance) between pairs of vertices. The algorithm has been referred to as the KK algorithm [18]. In this algorithm, the graph nodes are considered as particles that are all connected by springs whose ideal lengths are equal to the graph-theoretic distances between their two end-point particles multiplied by the desirable length of one edge. The goal of the KK algorithm is to find a balanced spring system. The energy of KK's spring system is defined by

$$E = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{1}{2} k_{v_i v_j} (d_{v_i v_j} - l_{v_i v_j})^2 \qquad (3.3)$$

where $l_{v_i v_j}$ corresponds to the desirable distance between v_i and v_j, and $k_{v_i v_j}$ is the strength of the spring between these two vertices. KK computes a local minimum of E and it solves the minimization problem using the Newton-Raphson method. The graph theoretic distance between the nodes v_i and v_j in a graph G is the length of a shortest path between these two nodes in the graph. As a result, they define E_i, the energy of a node v_i, using the partial derivatives of Equation 2.7 by x_i and y_i. The following is an outline of this algorithm

```
Algorithm KK(G:graph)
WHILE $MAX(E_i$ for $1<i<n) > \varepsilon$) DO
    Let $v_m$ be the particle satisfying $E_m = MAX(E_i$ for $1<i<n)$
    WHILE $E_m > \varepsilon$ DO
        Compute displacement of $v_m$ by solving a system of
        partial differential equations
        Displace $v_m$
    END  Inner WHILE
END  Outer WHILE
END  KK
```

where n is the number of nodes. The major drawback of this method is its high computational cost due to the work in solving a system of partial differential equations in the inner while-loop of the algorithm. Graphlet [1] version of KK uses a maximum of $10 \times n$ iterations of the inner loop. Although, this modification speeds up the algorithm and generates good quality layouts for small graphs, for large graphs it generates layout of planar graphs with many edge crossings.

3.1.5. Tunkelang Algorithm

Tunkelang has presented two different algorithms for graph drawing [19], [20]. In this section we outline these two approaches.

A Practical Approach to Drawing Undirected Graphs

The first algorithm introduced by Tunkelang [19], referred as Tu, is an incremental algorithm. This algorithm has three stages. In the first stage, a permutation of the nodes of a given graph is constructed from a minimal height breadth-first spanning tree of the graph. In the second stage, for each node in the order computed in the first stage, the area surrounding the already positioned neighbors of the node is sampled and examined for an approximate best position, and the node is placed at this position. In the second stage, when the local optimization procedure improves a vertex's position, it recursively performs the process on the already placed adjacent nodes of the vertex. Finally, in the last stage, the algorithm performs the local optimization process at every node for fine-tuning.

This algorithm generates layouts that are different from those of FR, KK, and GEM. Specifically, it does not capture graph symmetries. In addition, the algorithm takes a quality parameter where a large value results in better quality of graph layouts. However as reported by Himsolt et al. [8], this quality parameter is estimated to have an exponential effect on the running time of the algorithm.

Efficient Computation of Forces and The Optimization Process

The second graph drawing algorithm produced by Tunkelang [20] is based on the FR algorithm. This algorithm differs from FR in its computation of repulsive forces, in its optimization process, and in stopping the algorithm. Similar to the "grid variant" in FR, this algorithm approximates the computation of repulsive forces on a node. Here, this is done using a Barnes-Hut tree-code [8]. FR's optimization process uses force laws that in effect compute the negative gradient of an implicit objective function. However, this algorithm uses the conjugate gradient method on a non-quadratic objective function using an approximate line search. In addition. this algorithm uses the average of the square of the displacement distances of nodes for stopping the algorithm. It terminates when this value is less than 0.01.

We must note that there are not enough details in [20]. particularly. in explaining the optimization process. However the time measurements by Tunkelang on square grids. complete binary trees. and hyper-cubes show that the optimization process speeds up the algorithm in comparison with FR. On the other hand. the overhead of Barnes-Hut approximation of the repulsive forces slows down the algorithm for graphs with approximately less than 150 nodes. and speeds up the algorithm for graphs larger than that. In addition. he reports that the method of stopping the algorithm is a conservative one and in many cases the algorithm could be stopped much sooner

3.1.6. Simulated Annealing: DH

Davidson and Harel [21] use the simulated annealing approach [22] to graph drawing. Their algorithm is often referred to as DH. In the context of simulated annealing for graph drawing. a configuration is the assignment of unique grid points to nodes of a given graph. The algorithm tries to find an optimal configuration according to a cost function. In DH. the rule for generating a new configuration is by moving one node to a randomly picked point on a circle around the vertex with radius T. only if the new configuration is a better one according to a cost function described below. Radius T is initially large and decreases in each round of the algorithm. This decreasing radius (i.e. the decreasing neighborhood in simulated annealing terms) makes this algorithm different from the simulated annealing process in which a constant rule for generating new configurations is used. For a layout L of graph G, the cost function of DH as defined in [21] and [23] is

$$f(L) = \sum_{i=1}^{5} \lambda_i f_i(L) \qquad (3.4)$$

where λ_i is the weight of $f_i(L)$, and

$f_1(L) = \sum_{(u,v)\in V} (1/d_{uv}^2)$ where d_{uv} is the distance between u and v. This is to prevent nodes from being too close to one another.

$f_2(L) = \sum_{u\in V} ((1/r_u^2) + (1/l_u^2) + (1/t_u^2) + (1/b_u^2))$ where r_u, l_u, t_u and b_u are the distances of u from the right, left, top, and bottom borders of the drawing respectively. This is to prevent nodes from positioning too close to the borderlines.

$f_3(L) = \sum_{(u,v)\in E} d_{uv}^2$ This component is to avoid edges from being too long.

$f_4(L) =$ The number of edge crossings in L.

$f_5(L) = \sum_{u\in V, e\in E} (1/g(u,e)^2)$, where $g(u,v)$ is the minimal distance between vertex u and edge e. This penalizes the node and edges that are too close to one another.

The weights on the components of the cost function of Equation 3.8 can be adjusted to obtain drawings with different aesthetics. For instance, a high value of λ_4 results in drawings with few crossings. The computational intensity of the simulated annealing approach to graph drawing makes this algorithm not suitable for interactive applications.

3.2. Spring Algorithm in the Report

The objective of spring algorithm is to find high quality graph layout. The quality is closely related to the initial configuration, parameter setting and terminating algorithm. In Section 3.1, we present a literature review on spring algorithms. In this Section, we give an introduction on the spring algorithm designed and implemented in this report. The introduction includes the initialization, formulating physical model, iterating and movement, and terminating method using upper and lower bound forces.

3.2.1. Input graph data

The input data of the spring algorithm is a graph G with vertex set V and edge set E. The graphs are not required to have any properties beyond connectedness. The vertices $v \in V$ can have different weight, or the edges $e \in E$ can be of different length. The weight property can be used to adjust the length of the directed linked edge. In spring algorithm, graph with weight in vertex and variant length of edges can be modeled using springs of different length.

In this report, the input graphs have uniform length of edge, and the vertices have equal weight. But the algorithm can be extended to drawing the graph with non-uniform length of edge, and variant weight of vertices.

3.2.2. Formulating the mechanical Model

Hinge and springs

The spring algorithm uses the input graph data to define a mechanical system where frictionless hinges are connected by springs. A hinge represents a vertex in the graph.

There are two types of springs in the system. "Type I" spring and "Type II" spring. A "Type I" spring places between a pair of vertices that are connected, and a "Type II" spring places between vertices that are not connected. "Type I" spring has specified natural, unstressed length. It can generate either attractive or repulsive force on the end-vertices. "Type II" spring can generate only repulsive force.

Aesthetics

Spring algorithm is used to find approximately equilibrium state of the mechanical system, and the relative positions of vertices are used to draw the graph layout. Eades [1] pointed out that at the equilibrium, "Type I" springs, representing edges, tend to equalize in length, and "Type II" springs tend to separate vertices that are not connected by an edge, thus approximately satisfying two important drawing aesthetics:

1) All edges of the graph should be drawn with the same length, and
2) Vertices, which do not have an edge between them, should not be drawn close together.

A further valuable property of the algorithm is that symmetries in the graph tend to be displayed in equilibrium configuration [2].

Force Law

The precise form of the force law has impact on the final layout. The classical ideal force linear spring law may be too strong in attraction region, and Eades has suggested a logarithmic law as the following:

$$F_I(s) = \begin{cases} K_c(L-s) & linear \\ K_c \ln\left(\frac{L}{s}\right) & \log arithmic \end{cases} \tag{3.5}$$

Where

s is the spring length

L is the unstrained spring length

$F_I(s)$ "Type I" force is magnitude of the force developed when the spring has length s. and

K_c is a constant

A repulsive force is also applied between all vertices not directly connected by an edge. This is implemented as "type II" a spring. computed using the following equation:

$$F_{II}(s) = K_{rr}\left(\frac{L}{s}\right)^2 \tag{3.6}$$

where K_{rr} is a constant.

Force Minimization

An equilibrium configuration of the mechanical system can be characterized by two equivalent conditions: at equilibrium, the forces acting on each vertex exactly cancel, and the potential energy stored in the springs is stationary with respect to the vertex positions. These conditions suggest the two main techniques for finding equilibrium, given an arbitrary initial configuration. The two techniques are "force-minimization" method and "energy-minimization" method in [15]. The "force-minimization" method tries to find configuration in which total force on each vertex is zero. The "energy-minimization" method seeks a configuration minimizing the stored spring energy or some generalization of it.

"Force-minimization" methods proceed by initially placing the vertices in an arbitrary configuration and then numerically simulating the relaxation of the mechanical system towards equilibrium. The residual force acting on each vertex is used to compute an incremental modification of the vertex positions. In this report, the "force-minimization" method is used to find the equilibrium configuration.

3.2.3. Mapping input to Mechanical Model

Given a graph G with vertex set V and edge set E, a mechanical model is defined by introducing and ideal hinge to represent each vertex $v \in V$. The position of a hinge in the model is taken to be the position of the corresponding vertex in the layout.

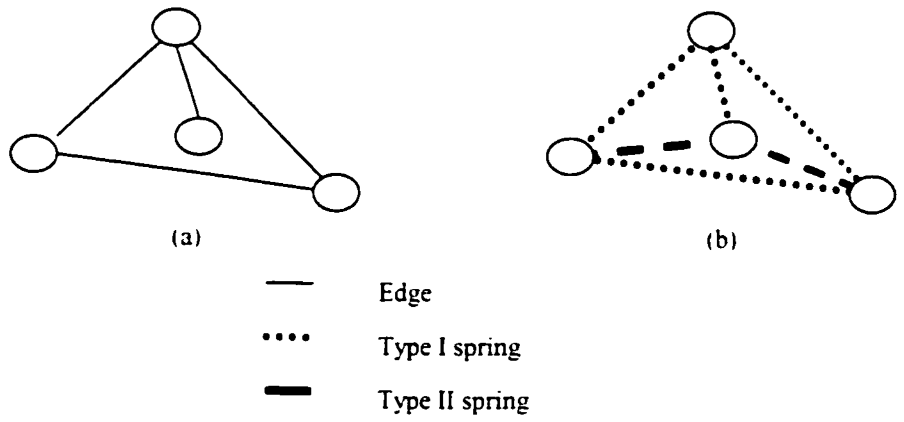


Figure 3.1: (a) Input graph, and (b) spring model

The edges $e \in E$ are represented by springs. These springs – "type I" are usually identical for all edges and have some natural, unstrained length. The intention is to apply an attractive force when spring length is greater than the natural spring length, and a repulsive force when the spring length is smaller than the natural spring length. The spring "force law" is the relationship between the magnitude of the force and the spring length. At the natural length of the spring, the force is zero. Figure 3.1 shows an input graph and its corresponding virtual mechanical model.

Initialization

The initialization of the spring algorithm randomly assigns a coordinator to every vertex in 3D space. In this report, the initialization process starts from a defined spring length L. The logical 3D space is cube L_s^3, and its edge length is defined as follow:

$$L_s = n \times L \qquad (3.7)$$

where n is number of vertices. After defining the 3D space, vertex coordinator is randomly generated one by one, but the coordinator is bounded by $(0,0,0) \sim (L_s, L_s, L_s)$.

This initialization proposes nothing on drawing graph within a predefined 3D space, but to defines only the logical coordinator of vertex. However, limiting the logical coordinator tends to give a good initial configuration without incurring a lot of extra computation.

After tuning by the spring algorithm, the forces may settle down a vertex or more outside the boundary logical 3D space. These are actually coordinators that map to the displaying screen.

Attractive vs. Repulsive forces

At the equilibrium, "Type I" force acts as attractive force, and "Type II" force is repulsive force. Attractive forces settle the edges to the spring length. Repulsive forces separate the every pair of vertices at distance at one or more spring lengths.

Attractive force must be significant compare to the repulsive force. It tends to be zero at the natural spring length. If a small among attractive force cannot subtract repulsive forces, edge length cannot be close to the spring length. Repulsive force must not be too weak compare to attractive force. The vertices may concentrate to a small area, if repulsive force is ignorable weak. The relationship between attraction and repulsion determines by the two constants k_c in Equation (3.5) and k_u in Equation 3.10. The experimental results in chapter 5 indicate that the $k_c : k_u = 1000 : 1$ leads to quality graph layout.

Iteration and Node Movement

In FR algorithm, the forces on all vertices are computed, and then the vertices are move at once. But GEM algorithm uses the history of the vertex movement to choose the temperature for the displacement of the vertex. It moves the vertices one at a time in each iteration. In this report, the forces on the vertices are computed first similar to FR, but only those nodes that are not at local equilibrium are moving in each iteration.

Depending on the total force acting on a vertex, it can be in one of the three states, coarse state, tuning state, or local equilibrium. At coarse state, the force acting on the vertex is strong, and the vertex stays far distance to the local equilibrium. At tuning state, the vertex is very close to the local equilibrium. Small step movement leads the vertex towards the local equilibrium in number of iterations, while large step may force the vertex back to coarse state. If a vertex is at its local equilibrium, it does not move in next iteration. But it may active again, depending on the change of the force acting on the node.

The three states quantified by predefined forces. The algorithm with the value of forces using in this report is as the following:

```
IF $F(v) > 0.05$ THEN
        $P_{curr} = P_{curr} - L_l$
ELSE IF $F(v) > 0.0001$ THEN
        $P_{curr} = P_{curr} - L_s$
END IF
```

Where the total force acting on a vertex is computed as follow

$$F(v) = \sum_{(u,v)\in E} F_a - \sum_{(u,v)\notin E} F_r \qquad (3.8)$$

Equation (3.8) indicates that only non-connected pair of vertices generates repulsive force.

Termination method with Upper and Lower Bound forces

The termination method is an import part of the spring algorithm. Since it gives the final configuration for drawing the graph. Eades' spring algorithm and FR algorithm are terminated by a predefined number of iterations. Although an extra terminating condition has been introduced, all tests reported by [16] indicate that GEM terminates by number of iterations. The drawback is that number of iteration to generate quality layout is unpredictable for different size of input. So this report presents quality oriented terminating method that stops the algorithm by upper and lower bound forces.

The force acting on each vertex and the overall force acting on the system are very important factors for graph layout quality. The local force affects the equal edge aesthetic, while the overall force affect the distribution of vertices and the symmetry aesthetic. The upper bound defines the local equilibrium force acting on vertex. Once the remaining local force on a vertex decreases under the upper bound. we stop moving this vertex. the effect shift more on those vertices with high remaining local force. The lower bound defines the smallest overall force. The algorithm terminates once after the overall force decrease under the lower bound.

The upper and lower bound forces achieve the quality of the graph layout by forcing the local force of each vertex closer to lower bound. As the experiment results shown in Chapter 5. the maximum and average difference between the edge length and the natural spring length are very small.

Spring algorithm

The detail algorithm uses in this report is as following:

```
INITIAL($v_i$ for $1 < i < n$)
ALGORITHM_SPRING($G$)
WHILE (TRUE)
    FOR EACH $v_i$ : $1 < i < n$
        COMPUTE $(\sum_i F)$
    END FOR
    IF $NOT$ EQUILIBRIUM ($G$)
        FOR EACH $v_i$ : $1 < i < n$
            MOVE_VERTEX( $v_i$ );
        END FOR
    Else
        BREAK;
        END IF
END WHILE
SHOW_LAYOUT($G$)
```

3.3. Summary

This chapter gives a literature review on the spring algorithm. It introduces the spring algorithm designed in this report. In next chapter. we discuss the problem on solving graph layout problem using spring algorithm. and also discuss the impact of parameter on the readability of final layout as well as simplifying computation.

4. Implementation

This chapter has three Sections. Section 1 discusses the MFC dialog-based application that implements the spring algorithm. Section 2 talks about the quality measurement of the graph layout. It also discuss how upper and local bound of the force guardant the preserving of the quality. Finally in Section 3, we discuss how to reduce the computation through the step function.

4.1. MFC dialog-based application

The 3D software is a MFC dialog-based application. The dialog shows all the user-accessed controls, and a view window. The view window is a CWnd-derived class that encapsulates all the graphing functionality. It displays the 3D graph layout generated by spring algorithm. The user controls allow user to switch between the initial graph configuration and the final configuration by spring algorithm. The user can rotate the view from any angle with the mouse. Also, the use can set up parameters used for spring algorithm and the size of the vertex to be displayed.

The input graph data is stored in a database, and loaded by the application program through Microsoft Jet Engine. The database has two tables, "Edge" and "Node" table. The basic data in these tables are vertices and edges. But the weight of the edge and vertex color can be added to "Edge" and "Node" table, respectively. There is an interface between the database and the application. The interface defines a load-data function. By using this interface, the application is easier to deal the changes, such as adding proprieties to the graph. replacing data format. such as interchange between file and database.

The application and the spring algorithm are communicated through implementation of strategy pattern. The input to the spring algorithm is a graph represent by adjacency matrix. By using strategy pattern. the coupling between the application and the spring algorithm has been removed. In other word. the application provides a platform that can be used to test various spring algorithms and other 3D algorithms. The only requirement for those algorithms is to accept graph data represented by adjacency matrix.

4.2. Vertex Quality

Each graph-drawing algorithm aims to optimizing certain aesthetic criteria. The quality of the graph layout is an evaluation of the extent to which the drawing meets objective aesthetic criteria. The possible measure for the graph layout can be the uniform of the edge or the force acting on the vertex.

At the equilibrium. the spring algorithm configures the vertices to the position where the force acting on the vertex is approximately zero. When all the edges are at the natural spring length. the attractive force in the system is zero. The remaining force acting on the system is the repulsive force. To be able to balance with repulsive force. some edge has to settle down with length $L_i = L + \Delta L$, where $1 \leq i < n$, n is the number of vertices. The variant length ΔL gives the attractive force that balances the repulsive force. thus leads the system to the equilibrium.

In this paper the quality of the graph layout is measured using the maximum and average difference length of the edge to the natural spring length. The maximum difference L_{max} and the average difference L_{avg} are defined as the follow:

$$L_{max} = Max(\Delta L_i) \qquad (4.1)$$

$$L_{avg} = \sum_{i=1}^{n} \Delta L_i \Big/ n \qquad (4.2)$$

where

$$\Delta L_i = |L_i - L|$$

Here L_{max} and L_{avg} is used to measure the quality of the final layout.

4.3. Graph Layout Qualities

In this report, the desired aesthetics are the uniform edge length and the distribution of the vertices. The distribution that vertex should be arrange in a way that non-adjacent vertices are far from each other. This can be measured using the distance between non-adjacent vertices or the remaining force in the system.

The distance between non-adjacent vertices can be a static measurement for the final graph layout. While the remaining force in the system implemented in this paper can be used as a dynamic measurement.

4.4. Measuring quality with upper and lower bound of force

In the execution of a spring algorithm, the criterion to measure the quality of the layout configuration can be used for:

- Controlling the quantity of the step for one movement in each iteration

- Activating or stopping the local movements of the a vertex once in each iteration
- Stopping the algorithm

The spring algorithm starts form a random configuration of the vertices. Measuring the quality of a graph layout allows for automatic selection of the best graph layout among the number of different layout of the same graph. This is the case where the graph layout is choosing by some cost functions, for example, the temperature in Simulation anneal.

In this report, the two dynamic measures using in each iteration are "lower bound force" F_l and "upper bound force" F_u. F_u is the local equilibrium defined for measuring a vertex quality. It is called upper bound force because one a vertex reached the local equilibrium, the vertex will no be tuned further. F_l is global equilibrium defined for measuring the system quality. It is called lower bound force because the algorithm will not terminate until the total force in the system is under this value.

The lower and upper bound make it possible that algorithm tuning more on the non-local equilibrium vertex and save the computation on tuning the equilibrium one. The bounds lead small variant remaining forces on the vertices, thus generate layout with the uniform edge.

4.5. Reducing Computation using step function

Spring algorithm starts form random initial and ends to the equilibrium of the mechanical system. The algorithm is an iteration process. After each iteration the total forces acting on every vertex decreases, due to the replacement of the vertex moves toward the direction of the force. The running time of the algorithm depends on the total number of iterations and the computations within each iteration.

The time (T_f) spent on compute the force on every vertex is a constant. The Total time on the on force computation is $n_i \times T_f$, where n_i is the total number of iterations. So decreasing n_i decreases the running time of the algorithm. In one iteration, the time T_r, stands for replacement time, is the time to find out the new coordinator for a vertex. The total time in one iteration is $n_v \times T_r$, where n_v are vertices that have not reached it equilibrium. So the more the vertices are equilibrium the less the running time spends for an iteration. The total running time is defined as the follow:

$$T_{total} = n_i \times (T_f + n_v \times T_r) \qquad (4.3)$$

Equation 4.3 indicates that n_i is the key factor to speedup the algorithm.

4.5.1. Minimizing total number of iteration

There are two approaches to minimize the total number of iteration, tuning initial layout or replacing vertices efficient. Tuning initial layout is a process that tries to replace those springs that are much longer than the natural length without computing any forces on vertex. Graph traversal can be used to implement the initial tuning. The algorithm shall be simple and computational efficient. The turned layout has most of edges closer to the spring length. It is easier to converge once the vertices have been well distributed in 3D space.

Replacing vertices efficient requires selecting efficient steps. The important factors for defining the step are the force acting on the vertex, the spring length, historical forces and corresponding steps are very useful data too. The rule of the thumb is that when the force is large, a large step is used. Otherwise a smaller step is taken.

Historical force and its corresponding step can be used to calculate next step. The step is calculated based on a predefined initial step or the last step, the change of the force and the remaining force. The following equation can be used:

$$d_{i+1} = f \times \frac{|F_{i+1} - F_i|}{F_i} \times D \qquad (4.4)$$

Where

d_{i+1} is the next step

F_i is the force in last iteration

F_{i+1} is the remaining force

D is either a predefined step or it is last step d_i, and

f is a factor, $0 < f < 1$

The steps are automatically defined and controlled by the algorithm. User does not need to adjust number of parameters because of the changing in the input graph.

In this report, the algorithm uses a small step and a large step rather than using the historical force and step. The small step is $d_s = 0.00005L$, while the large step is $d_l = 0.005L$. The force $F_s = 0.05$ is used to select a step. If $F(v_i) > F_s$, the next step is a large step, otherwise the next step is smaller step.

The large step is designed to make every vertex ready to start tuning as soon as possible. The small step is introduced to tuning the vertex as closer as possible toward the equilibrium to produce high quality graph layout.

4.6. Summary

In this chapter. first we discuss the vertex quality – Maximum and average difference of the edge to the spring length. graph layout quality – Remaining force. Then. we introduce the new termination of the spring algorithm using the lower bound force and upper bound force. Finally. we analyze how the step function can be used to minimize the computation. In chapter 5. we run 4 sets of experiment that demonstrate the different aspects of the spring algorithm and the multi-view angle advantage of the 3D graph drawing.

5. Experimental Results

5.1. Outline of the Experiments

There are four sets of experiments in report. The first set experiment is to test how different initial layouts affect spring algorithm, when we are using the same configure of the algorithm. The second one is to test capability of the algorithm on processing more than 20 vertices. The layout is displayed and the quality values are collected. The third test demonstrates that the multiple view angles enhance the readability of the 3D graph. Also the algorithm can draw quality graph with more than 40 vertices and 60 edges within 10 seconds. Finally, the last test is to find the better range of the repulsive factor given the attractive factor of 2.0.

The quality measurements for the experiments are the maximum and average differences of the spring length to the natural spring length. Most of the parameters are the same in all of the tests, except the upper bound force – global equilibrium that depends on the number of the vertices.

5.2. Configuration

5.2.1. Hardware Configuration

All of our experiments are performed on IBM compatible PC with single AMD 800 processor and 256 megabytes of RAM.

5.2.2. Software Configuration

The operating system that is used in our experiments is Window 2000. The application is developed using Microsoft Visual Studio. Visual C++ 6.0.

5.3. Experiment Data

There are four sets of input data used in the experiment. Table 5.1 shows the number of the vertices and edges in the put graphs for our experiments.

	Number of vertices	Number of edges
Set 1	7	10
Set 2	21	27
Set 3	24	40
Set 4	41	61

Table 5.1: The statistic of 4 sets of test data

5.4. Initial configuration

The design of this experiment is to show the affection of the various initial configurations on the spring algorithm. This test uses the set 1 as input data. The natural spring length is $L = 10$. The attractive factor is $k_s = 2.0$ and the repulsive factor is $k_n = 0.002$. The upper bound force is the local equilibrium force $F_l = 0.0001$ and the upper bound is the global equilibrium force $F_g = 0.001$. The smaller step is 0.00005L and the large step is 0.005L. If the force acting on the vertex is greater than $F_l = 0.01$, the next step is the large one, otherwise using smaller step.

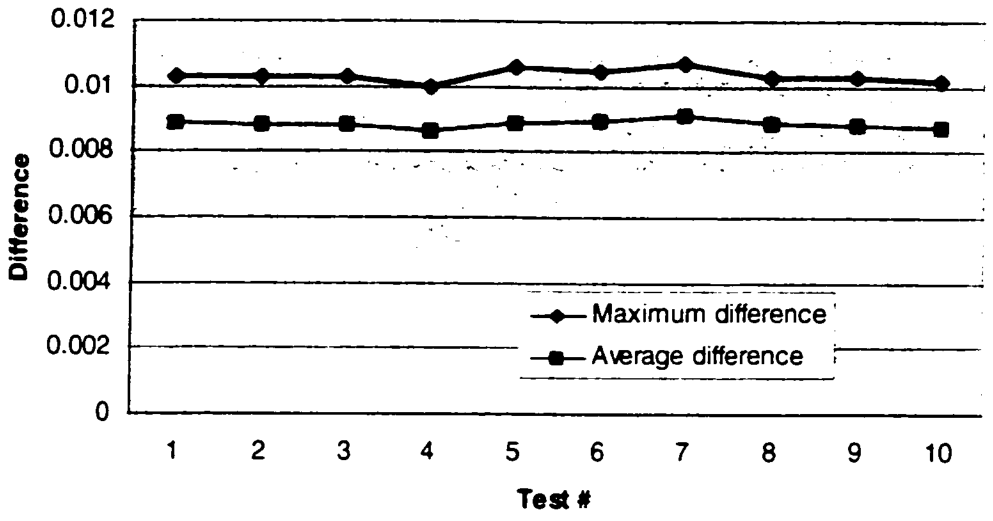


Chart 6.1: The maximum and average difference of the length to the natural spring length

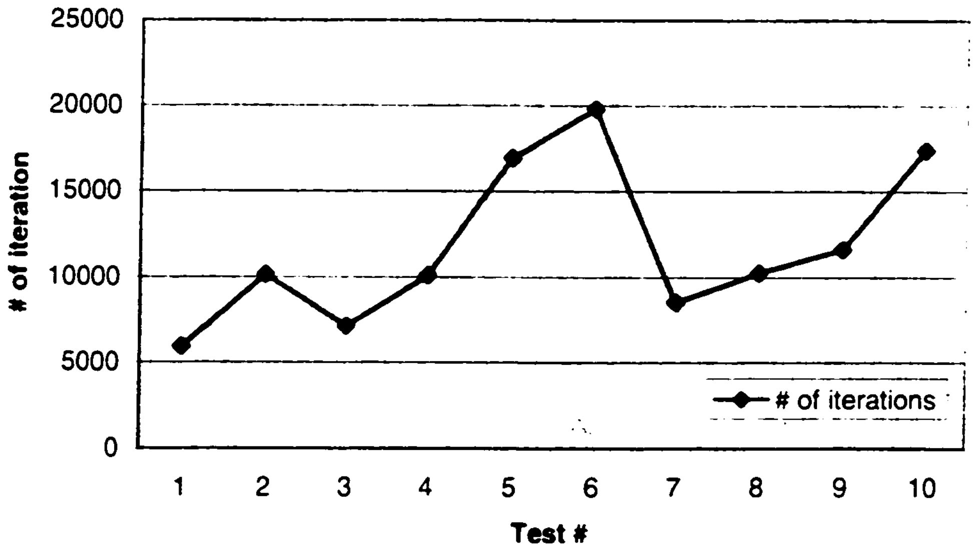
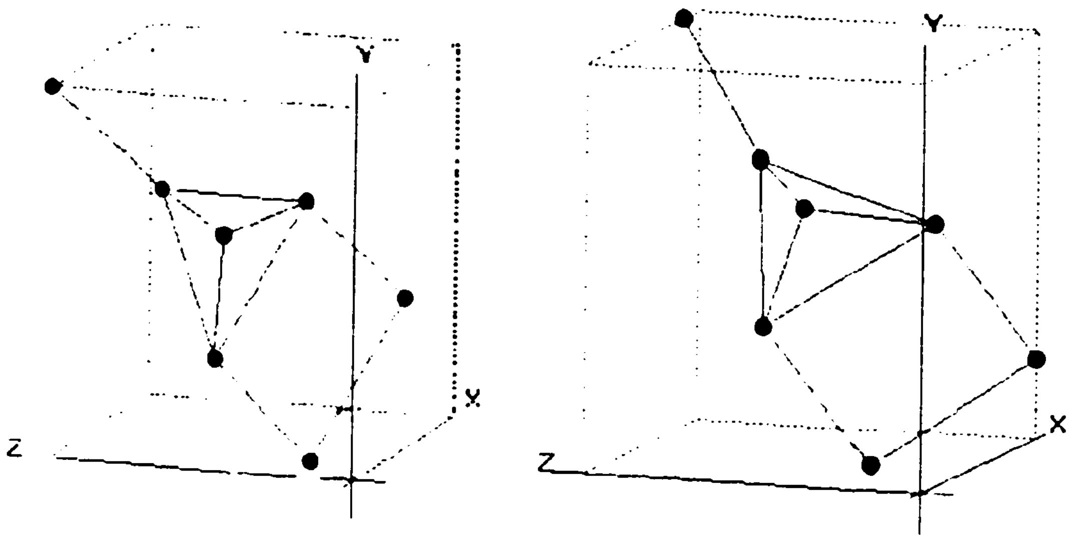


Chart 6.2: The number of iteration on different test

The average edge difference to the natural spring length, the max variant of the spring length and the number of the iteration are the collected from 10 tests. The results are plotted in chart 6.1 and 6.2. The X-Axis in both charts is the test number from test 1 to the test 10. Each of these tests starts from the different initial configuration. The top curve in chart 6.1 is the maximum difference of the spring length compare to the natural spring length. The bottom curve in chart 6.1 displays the average difference. The curve in chart 6.2 is number of the iteration used for each tests. The Figure 6.1 (a) and (b) show two equal quality graph layouts.



(a) *(b)*

Figure 6.1 (a) and (b): two final layouts from the different initial configuration

The two layouts in Figure 6.1 (a) and (b) are just a shift and little bit rotation of one graph to another. In chart 6.1, the maximum difference shows very small difference. The maximum variant is test 7 and test 4, $(T7 - T4)/L \approx 0.0001$, in other word, the maximum difference is less than 0.01 percent of the spring length. The curve of average difference is approximately straight line. In chart 6.2, the number of the iteration of different test varies dramatically. The number of the iteration of the test 6 is almost the 4 times as the test 1.

The conclusion of this test is that the different initializations affect almost nothing on the final layout quality. But it has strong influence on the number of iterations. As we know the iteration number is directly link to the running time of the algorithm for certain size of the input data.

5.5. Graph layout with more than 20 vertices

This experiment run two tests with set 2 and set 3 as input data. This test intends to show the capability of the algorithm in processing more than 20 vertices of both sparse and dense graph.

The natural spring length, the attractive factor, the repulsive factor and the upper bound force etc.. are the same as it is in Section 4. But the upper bound differs to the one used in Section 4 due to the increasing number of vertices. The upper bound value for the two tests are the same $F_\varepsilon = 0.003$.

Figure 6.2 contains 21 vertices and 27 edges. The Maximum spring length has 0.467991 longer than the natural spring length. The average difference is 0.261766. Although both values are little bit large, but the small difference between average and maximum difference indicates that the uniform edge length is preserved. Figure 6.3 contains 24 vertices and 40 edges. Set 3 is a dense graph compare to set 2. The Maximum difference spring length is 0.578694. The average difference is 0.432412. So it has similar quality as the graph layout in Figure 6.2. The number of iteration is 24156 and 10163 with respect Figure 6.2 and 6.3. The running time is less than 2 second for both.

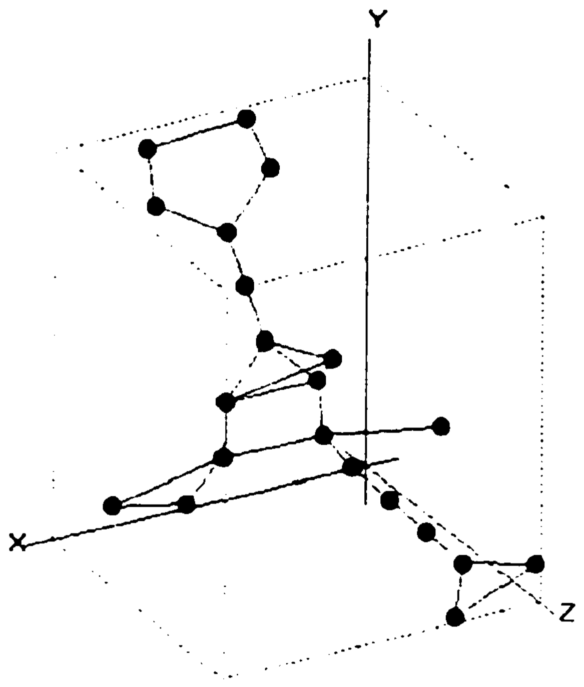


Figure 6.2: Graph layout with 21 vertices and 27 edges

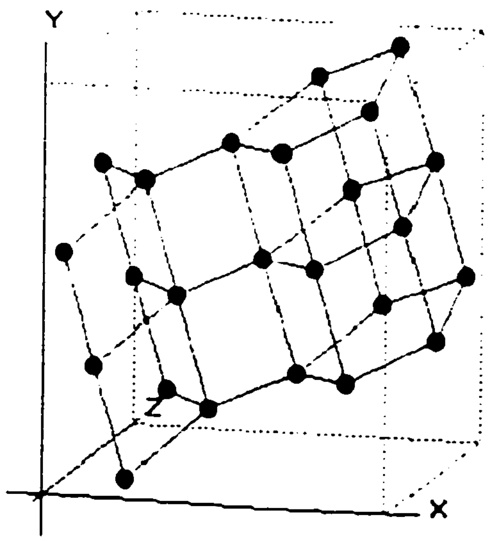


Figure 6.3: Graph layout with 24 vertices and 40 edges

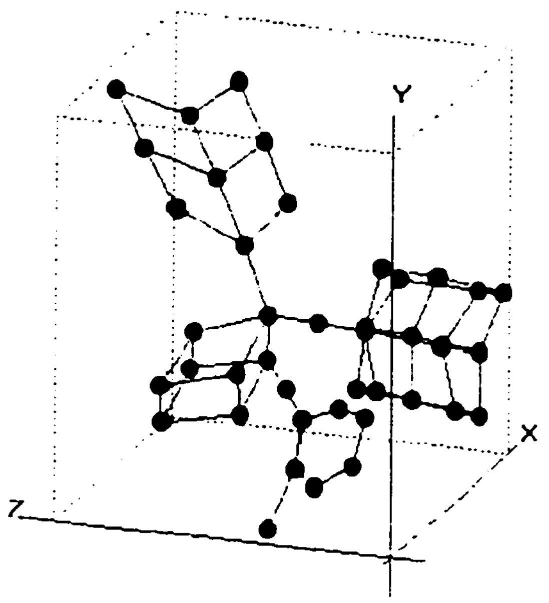
5.6. The advantage of 3D graph drawing

One of the advantages of 3D drawing is that it given batch of view points to facility the understanding of the graph layout. This experiment intends to show how the rotation function increases the readability of the graph layout. Set 4 is input data. It has 40 vertices and 61 edges. One layout is generated with all the parameter the same as it used in Section 5. upper bound value is $F_g = 0.005$. because there are more vertices in the graph. The maximum difference of the spring length is 1.61134. The average difference is 0.510877. The simulation runs for 25718 iterations and the running time is 10 seconds.

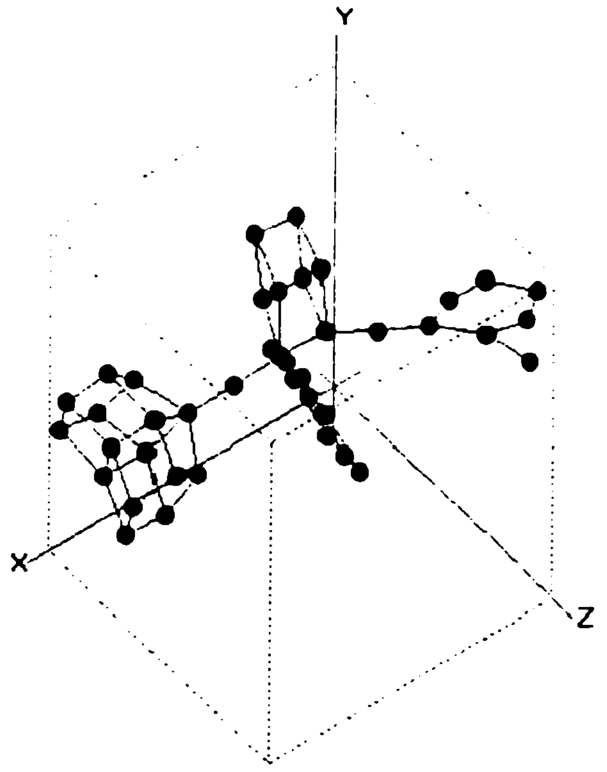
In Figure 6.4. the two graph layouts are viewed from the different angles. From the Figure 6.4 (a) one can easily get the idea what are the shapes of the three cluster in the left. But it is difficult to understand the largest cluster in the right. By rotating the view angle to the Figure 6.4 (b). one can get more information or understand the shape of the largest cluster completely.

5.7. Attractive factor K_c and Repulsive factor K_n

The attractive force and the repulsive force scales by the attractive factor K_c and the repulsive factor K_n. At equilibrium. the spring length must be slightly longer than the spring length. This length difference generates attractive force that balances the repulsive force from the non-adjacent vertices.



(a)



(b)

Figure 6.4: One graph layout display from different view angles.

The length different should not affect the uniform spring length. This means that an edge with large number of neighbors sitting around should not be too longer compare to the edge with fewer neighbors. To satisfy these, K_u should be smaller enough compare to K_s. On the other hand, vertices that are not connected directly should be far from each other. This indicates that the repulsive forces should be enough to carry away the vertices from the strong attraction.

This experiment designs to find best range of K_u when $K_s = 2.0$. Set 2 data is the test data. We execute five tests. The $K_u = \{0.2, 0.02, 0.002, 0.0002, 0.00002\}$. Chart 6.3 shows the edge quality of the five tests. Test 1 used $K_s : K_u = 10:1$. The result indicates that the force is too large, and one of the edge is 35 percent longer than it natural length at equilibrium. The number in chart 6.3 implies that test 2 to test 5 have the uniform edge length property. The graph layout is drawn in Figure 6.5. Figure 6.5 (a), (b) and (c) show that the vertices are well separated. But in Figure 6.5 (d) you observe that some vertices edge in between are closer to each other. This indicates that $K_s : K_u = 100000:1$ cannot give enough repulsive force.

This experiment designs to find best range of K_u when $K_s = 2.0$. Set 2 data is the test data. We execute five tests. The $K_u = \{0.2, 0.02, 0.002, 0.0002, 0.00002\}$. Chart 6.3 shows the edge quality of the five tests. Test 1 used $K_s : K_u = 10:1$. The result indicates that the force is too large, and one of the edge is 35 percent longer than it natural length at equilibrium. The number in chart 6.3 implies that test 2 to test 5 have the

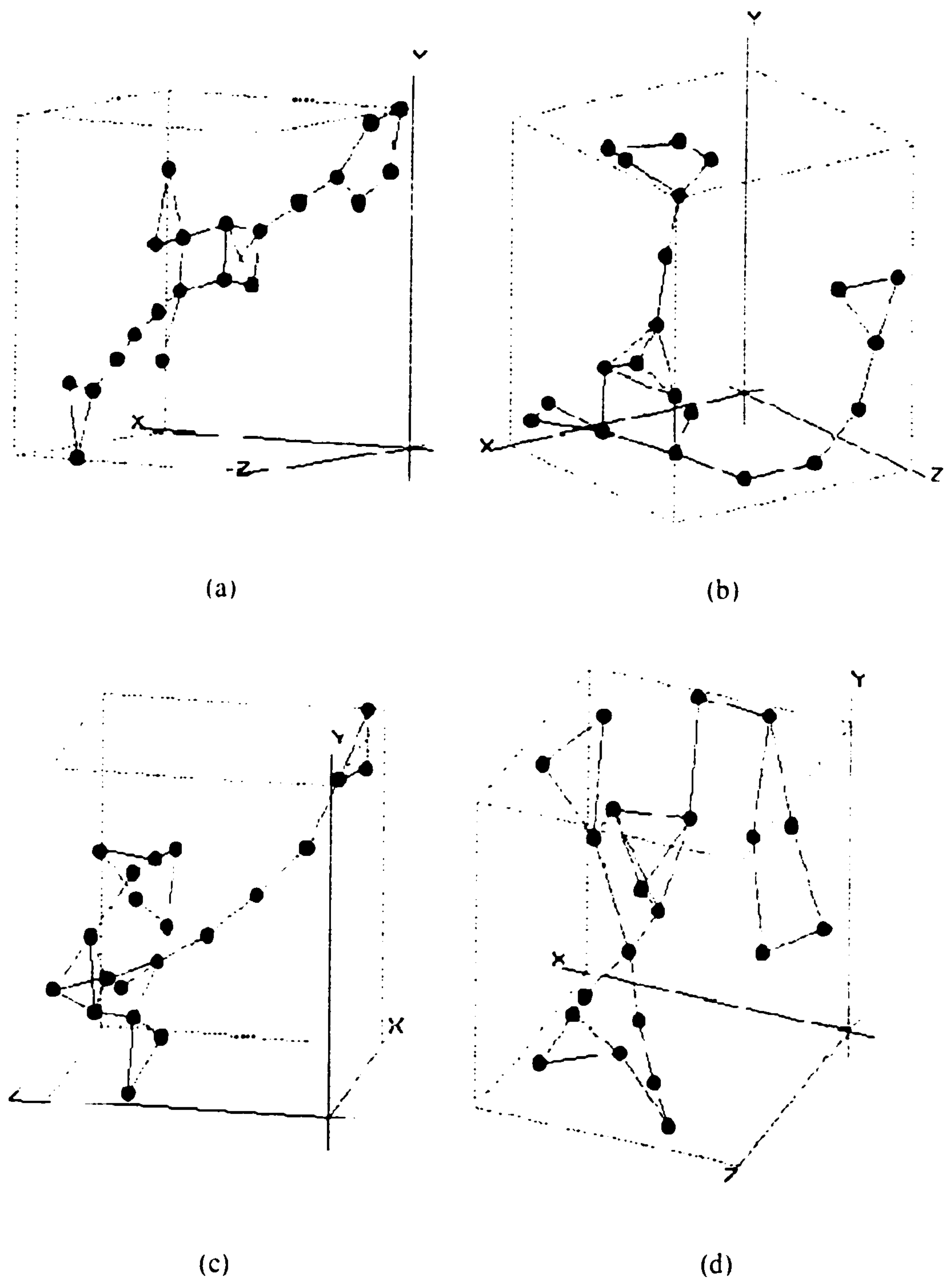


Figure 6.5: graph layout with the different repulsive factors

uniform edge length property. The graph layout is drawn in Figure 6.5. Figure 6.5 (a), (b) and (c) show that the vertices are well separated. But in Figure 6.5 (d) you observe that some vertices edge in between are closer to each other. This indicates that $K_c : K_u = 100000:1$ cannot give enough repulsive force.

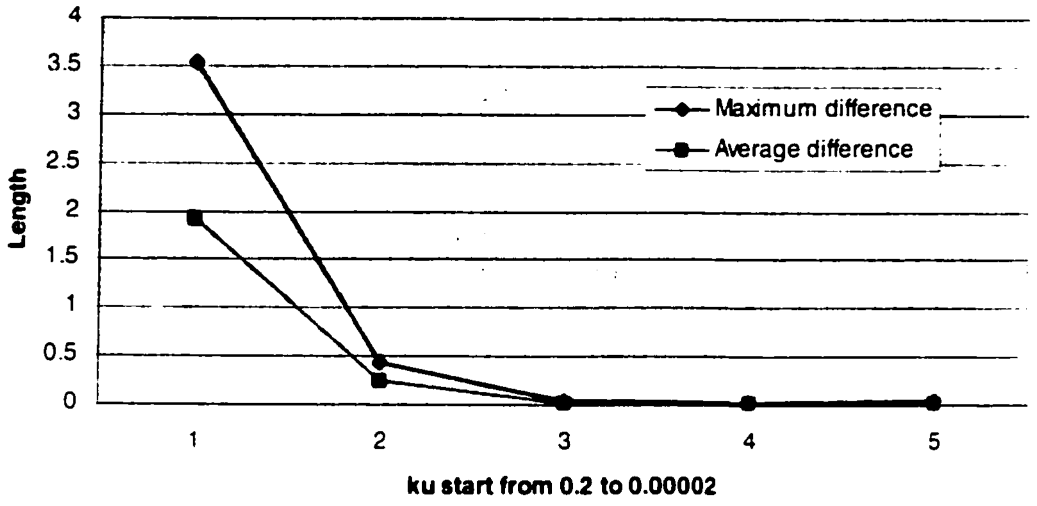


Chart 6.3: The maximum and average difference of the length to the natural spring length

As a conclusion, the range of the $K_c : K_u = 10000:1$ to $100:1$. Among the layout in Figure 6.5, Figure 6.5 (b) shows that $K_c : K_u = 1000:1$ can be the best choose.

5.8. Summary

From the four experiments in this chapter, we observed that the initial layout has greater impact on the total number of iteration, but has no impact on the layout quality with the same parameters. The result indicates that improve initial layout is a way to speed up the algorithm. We can also see that the algorithm is able to draw graph layout with more than 40 vertices and 60 edges within 10 seconds. And the lower bound of the force and the upper bound force enable the algorithm to generate quality layout. Finally, among the parameters used in spring algorithm, $K_u : K_c$ is important, since it affect both edge length and the distribution of the vertices in the final graph layout.

6. Conclusions and Future Works

6.1. Contributions

In this report, we apply spring algorithm to draw graph layout in 3D space and. introduce a new terminating method that uses the upper and lower bound force to terminate the algorithm. A simple step function is designed to minimize the total number of iteration and the quality measurement to automatically detect the finding of a quality layout. The step function defines coarse-tuning and fine-tuning phases for every vertex in the graph. The step function is one of the important parts of the spring algorithm to dealing with the large graph. The quality measurements are local equilibrium force and global equilibrium force. In addition to above, we developed a dialog-based application that include implementing spring algorithm and OpenGL 3D graph. The application is implemented with interface that decouples the data model, algorithm and user control, so that it is flexible to adopt in doing testing of other 3D algorithm.

6.2. Conclusion

Spring algorithm is very important for graph drawing. It is easy to understand and implement. It is applicable to undirected graph, and can be used for generating 3D graph layout. Experiments show some advantages of 3D graph layout compare to 2D layout. The graph without edge crossing is easier to understand. The rotation feature enhances the readability by providing different view angles. It is very help for identifying the shape in case of edge crossing.

Observations from different experiments show that upper bound and lower bound forces ensure the uniform edge and evenly distributed aesthetics. Also the designed spring algorithm can produce high graph layout that has 40 vertices and 60 edges with in seconds.

6.3. Future works

The spring algorithm is easy to implement and understanding. However, it typically takes a long time to converge. In additional to the tuning algorithm, the most important works remain to be done is finding new approaches to solve very large graph with spring algorithm. The possible approaches are

- Appling parallel design patterns, for example, divide and conquer, master and slaves, etc., to solve large graph. In this approach, a graph is divided into small graphs, and every sub-graph may be divided further. Each small graph is computed using spring algorithm, and then merging back to form the final graph layout. The parallel system and parallel algorithm is necessary for this approach.
- Appling pre-computed simple graph patterns to improve the initial configuration. The spring algorithm tends to produce the layout with edges that equal to natural spring length. Also, it has been proved that the better initial configuration, the faster the final graph layout converges.

  A pre-computed pattern defines the logical coordinators of a simple graph, such as an equilateral triangle or a square. After defining the spring length, and obtaining coordinator of any vertices of the simple graph, the coordinators of the rest vertices in the simple graph are computable. This leads to initial graph with most of the edges already settle down at the spring length, and has high level symmetric. Pre-computed patterns are reusable, and will benefit spring algorithms, which depends on the initial configuration.

Reference:

[1] P. Eades. A heuristic for graph drawing. *Congress Numerantium*. 42:149-160. 1984

[2] Xuemin Lin and P. Eades. Spring algorithms and Symmetry. To appear

[3] Tamassia. R.. 1987. On Embedding a Graph in the Grid with the Minimum Number of Bends. *SIAMJ. Computing*. vol 16. no. 3. pp. 421-444

[4] Tamassia R. and Toillis I.G. Editors. 1995. *Graph Drawing: Proceeding of GD '94. Lecture Note in Computer Science 894*. Springer-Verlag

[5] Tamassia. R. and Toillis I.G. Vitter. J.S.. 1991. Lower Bounds for Plannar Orthogonal Drawings of Graphs. *Information Processing Letters*. vol. 39. pp 35-40

[6]. Biedl. T and Kant. G. 1994. A Better Heuristic for Orthogonal Graph Drawings *Proceeding of ESA '94. Lecture Note in Computer Science 855*. Springer-Verlag. pp 124-135

[7]. Papakostas. A.. and Tollis. I.G.. 1997. A Pairing Technique for Area-Efficient Orthogonal Drawings. *Graph Drawing: Proceedings of GD '96. Lecture Note in Computer Science. North. S.. Editor*. Springer-Verlag

[8] F. J. Brandenburg. M. Himsolt. and C. Rohrer. An experimental comparison of force-directed and randomized graph drawing algorithms. *Graph Drawing: Proceedings of GD '95*. Pages 76-87. Springer-Verlag. 1996.

[9] W.T. Tutte. Convex representations of graphs. In *Proceedings London Mathematical Society*. volume 10. pages 304-320, 1960

[10] Di Battista. G.. Eades. P.. Tamassia. R.. and Tollis. I.G.. 1994. Algorithms for Drawing Graphs: an Annotated Bibliography. *Computational Geometry: Theory and Applications*. vol. 4 no. 5. pp. 235-282.

[11] Colin Ware. David Hui. and Glenn Franck. Visualizing O-O software in three dimensions. In *Proceedings of CASCON '93*. Toronto. Canada. 1993.

[12] Colin Ware and Glenn Franck. Visualizing information nets in three dimensions. In *TOG info. Net. Vis. '94*. Toronto. Canada. Auguest 15. 1994.

[13] T. Fruchterman and E. Reingold. Graph drawing by force-directed placement. *Software – Practice and Experience*. 21(11):1129-1164. 1991.

[14] B. Monien. F. Ramme, and H. Salmen. A parallel simulated annealing algorithm for generating 3D layouts of undirected graphs. *Graph Drawing: Proceedings of '95. volumn 1027 of lecture Notes in Computer Science*. page 396-408. Berlin. 1996. Springer-Verlag.

[15] Diethelm Ironi Ostry. "Some Three-Dimensional Graph Drawing Algorithms". Master Thesis. Department of computer Science and Software Engineering. University of Newcastle

[16] Lila Behzadi. "An Improved Spring-based Graph Embedding Algorithm and LayoutShow: a Java Environment for Graph Drawing". Master Thesis. York University.

[17] M. Himsolt. The Graphlet system. *Graph Drawing: Proceedings of '96*. page 233-240. Springer-Verlag. 1997.

[18] T. Kamada and S. Kawai. An algorithm for drawing general undirected graphs. *Information processing letters*. 31:7-15. 1989.

[19] D. Tunlelang. A practical approach to drawing undirected graphs. Technical report, Carnegie Mellon University, School of Computer Science1994.

[20] D. Tunlelang. JIGGLE:Java interactive graph layout environment. *In proceedings of graph Drawing '98*. pages 413-422. Springer-Verlag. 1999.

[21] R. Davidson and D. Harel. Drawing graphs nicely using simulated annealing. *ACM Transactions on graphics*. 15:301-331. 1996

[22] S. Kirkpartrick. C. D. Gelatt. and M. P. Vecchi. Optimization by simulation annealing. *Science*. 220:671-680. 1983.

[23] G. D. Battista. P. Eades. R. Tamassia. and I. G. Tollis. *Graph Drawing Algorithms for the Visualization of Graph*. Prentice Hall. New Jersey. 1999