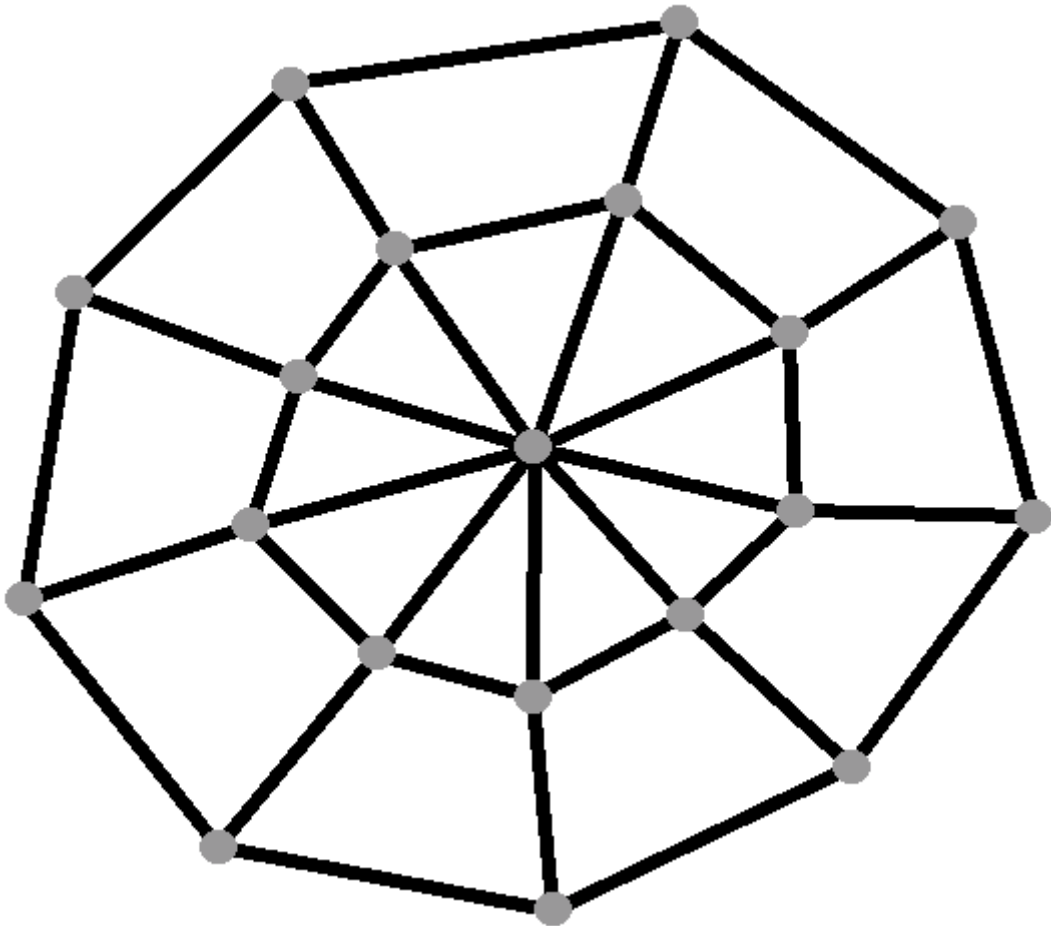


Graham Cormode
King's

"Springs and Sound Layouts"

Graph Drawing using Force Directed Placement

Computer Science Tripos Part II — 1998



This page unintentionally left blank

“Springs and Sound Layouts”

Graph Drawing using Force Directed Placement

Computer Science Tripos Part II — 1998

Approximate Word Count: 10,900

Project Originator: John Naylor

Project Supervisor: Calum Grant

Original aims of the project

The aim of the project was to investigate the problem of graph drawing: taking graph data and finding a way to display it in a pleasing manner. This was to be done using force directed placement methods, which use physical analogies to define forces whose equilibrium it is hoped will give a satisfactory layout. Different formulations of forces would be investigated, in terms of their speed of convergence, and the quality of their final results.

Work Completed

A program was written which implemented three different iterative algorithms, as well as four heuristics to give starting configurations. It allowed quantitative comparison of the algorithms by scoring layouts using an energy function. In addition, the system allowed investigation of other questions about the convergence of force-directed methods, and other tweaks to improve the final result. This meant that it was possible to draw some strong conclusions about the alternative layout methods.

Special Difficulties Faced

None.

by the same author

THE ALTERNATIVE MAIDSTONIAN

RED DRAGON PIE (editor)

THE VALENTINE'S DAY MASCARA

DIARIES, 1994-98

Table of Contents

1. Introduction.....	2
2. Preparation	4
2.1 Research into current theory	4
2.2 Examination of existing implementations.....	5
2.3 Analysis of the problem.....	6
2.4 Choice of language	7
3. Implementation.....	8
3.1 Choice of equations	8
3.2 Friction, cooling and termination	10
3.3 Quantitative analysis.....	10
3.4 Effect of starting position.....	11
3.5 Tools Used	11
3.6 Further implementation details	12
3.7 Unevaluated code.....	15
3.8 Realisation of the project.....	15
4. Evaluation.....	17
4.1 Performance of the system	17
4.2 Comparing the methods quantitatively (2.3.1)	19
4.3 Examining the effect of the starting configuration (2.3.2)	21
4.4 Working with a subset of the graph (2.3.3)	24
4.5 Final Observations.....	24
5. Conclusions	27
Bibliography.....	28
Appendix 1 — Geometric Algorithms.....	30
Appendix 2 — Centre Layout.....	32
Appendix 3 — Code Samples	33
Appendix 4 — The Social Network.....	34
Appendix 5 — The London Underground.....	35
Project Proposal	i

1. Introduction

The problem of graph drawing is of relevance to computer science for a number of reasons. It touches on many aspects of the subject, from algorithms and complexity, through graphics and data visualisation to less obviously related areas, such as VLSI. As well as being a problem encountered whenever there is graph data that needs to be displayed, it is closely related to the problems of circuit layout and map labelling. With the number of variations and approaches to the problem, it is perhaps no wonder that there is an entire annual symposium dedicated to graph drawing [GD98].

The problem of graph drawing can be summarised as “given a vertex list, V , and edge list, E , produce an aesthetic layout”. Such a brief description hides a great deal of detail, notably in the word “aesthetic”, which is not often found used in relation to Computer Science. Some aspects can be dealt with relatively easily — a layout is defined as a mapping of vertices to two-dimensional space, with information regarding how the linking edges are to be drawn; in other words, what we might regard as a drawing of the graph. This leaves the freedom to decide whether the edges should be drawn as unconstrained paths, Bezier curves, straight lines, or rectilinear lines. Likewise, whether the nodes should be placed freely, constrained to lie within a certain area or only at grid intersections is not specified.

How to interpret the word “aesthetic” is perhaps the most difficult. We might imagine that to adjudicate between two alternate layouts of a graph we could call on a human judge, in a manner reminiscent of the Turing test for supposed artificial intelligence; however, during the course of execution it is not reasonable to call on a human oracle, and instead a machine-evaluable function of “aesthetic” can be defined. The problem may then be considered as being to minimise this function.

Some simple criteria for achieving an aesthetic layout are to create an even distribution of nodes over the area, to make all edge lengths approximately equal, and to reduce the number of edge crossings. Of these criteria, the last is the most problematic. The problem of finding the minimum number of crossings in a graph (the crossing number) is NP-Complete [GJ79]. Hence any algorithm that claimed to minimise the number of crossings would, as a sub-problem, have to certificate that the crossing number of the graph had been correctly found, and should be viewed sceptically if it claimed to run in sub-exponential time.

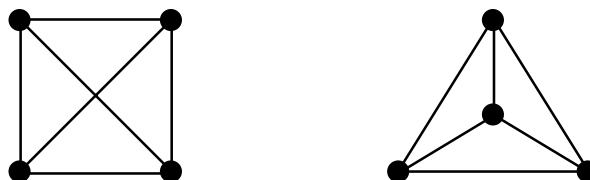


Figure 1: Two drawings of the same graph

It has been asserted that the best approach to “aesthetic” is to try to expose as many symmetries of the graph and its sub-graphs as possible in a drawing of it [KK89]. For example, the complete graph with four vertices can be drawn as in Figure 1. While the second drawing minimises edge crossings, it fails to fully illustrate the fourfold complete symmetry that is clearly visible in the first drawing.

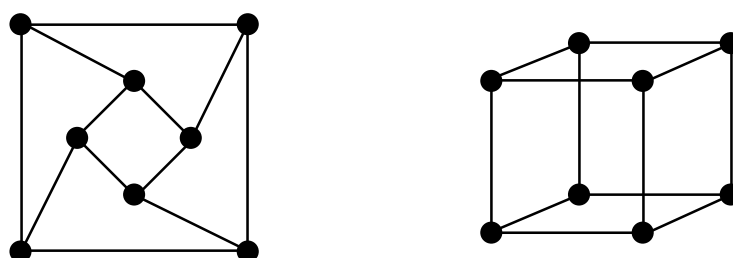


Figure 2: Two drawings of the cube graph

Even symmetry can be a hindrance; consider for example the two drawings of the graph in *Figure 2*. The first drawing shows some of the symmetry of the graph, while the second drawing is not as symmetrical. Yet since we interpret the second drawing as a representation of a cube, the symmetries of the graph are clearer. This is perhaps an anomalous result, since we would not expect general graphs to have such specific familiar, resonant representations.

A more direct way to examine the quality of a drawing is to ask if it is “simple” [HR90]. The criteria for this are that

1.1.1 Any two distinct edges have at most one crossing.

1.1.2 Any two edges which share a node do not cross.

1.1.3 No three edges cross at a common point.

A layout which satisfies these is not difficult to obtain, and so instead we will pursue the additional criteria suggested earlier, *viz.*

1.1.4 The nodes are evenly distributed

1.1.5 Edge lengths are approximately equal

1.1.6 An attempt is made to avoid edge crossings

The additional criteria are more subjective, but a layout which satisfies all six criteria will be described as a “sound layout”.

There are a number of broad approaches to graph drawing, fundamentally different in their methodology. Some only consider special cases: drawing trees, directed acyclic graphs or planar graphs are all areas that have been tackled successfully [BETT94]. These can lead to general algorithms: it is possible to check for planarity efficiently [Eve79], so one method repeatedly removes arbitrary edges from a graph until it is planar, finds a planar drawing of the reduced graph (using for example [HS98]), then adds back the removed edges. With graph drawing, it is frequently found that similar problems are known to differ in difficulty: for example, to find the maximum planar sub-graph rather than randomly testing sub-graphs is known to be in the complexity class NP-hard.

The methods investigated in this project are quite different in approach. They consider the vertices and edges as physical objects, and apply a physical simulation to evolve an aesthetic layout from a (usually random) starting layout. The equations used are designed to embody some of the aesthetic considerations suggested above: for example, we might assign a repulsive force between nodes (the analogy here is with electrically-charged particles), which we hope would prevent nodes clustering together.

These approaches, collected together under the banner of “Force Directed Placement”, still show a variety of approaches. The first to be suggested by Eades in 1984 [Ead84] considered each edge as a spring with a constant natural length, and nodes as bodies which repelled each other. Kamada and Kawai [KK89] gave a slightly simpler approach, where each node pair was connected by a spring whose natural length was proportional to the shortest-path graph distance between them.

Borrowing further ideas from physics, we note that where we have a force, there is a corresponding energy, and so we can convert our forces into energies, and consider ways to minimise the “energy” of a layout. This was documented by Davidson and Harel [DH89]. Once in the energy paradigm, we have the freedom to introduce energy “penalties” for which there is no corresponding force. The price that we pay for such freedom is that the algorithms to find a layout are significantly more expensive. Part of the purpose of this project is to investigate whether this is a price worth paying.

2. Preparation

2.1 Research into current theory

The first few weeks of the project involved investigating the available literature on graph drawing, starting from the papers suggested by the project originator, as well as searching for papers on-line. Three papers formed the basis of the approaches that I decided to implement.

2.1.1 P Eades, A heuristic for graph drawing, 1984 [Ead84]

This paper describes the first time that force directed placement was used to do graph layout. As was suggested in the introduction, force directed placement is generally applicable to many layout problems, of which graph drawing is perhaps one of the simplest. Eades' approach was to treat the current configuration of the graph as a physical system, and by calculating and applying forces on the nodes, simulate the action of the system iteratively. He derived the forces to be used by consideration of the goals of the system.

Firstly, an even distribution of nodes over the area of the drawing was required. This he achieved by associating a repulsive force between nodes, like that between positively-charged particles. In the absence of other forces, this would tend to spread the nodes out without limit; nodes may be confined to a finite area to prevent this. By taking an analogy with electrostatics, the force is defined to be proportional to the inverse square of the distance between the nodes. Secondly, Eades defined an additional force between nodes linked by an edge in the graph, as that of a spring whose natural length is a constant (related to the number of nodes and the area available). The force, again by analogy to real springs, is taken to be proportional to the extension or compression of the spring.

Although Force Directed Placement has been introduced by use of physical analogy, there is no justification to insist on a direct correspondence with the physical counterpart. The constants involved need bear no relation to any real physical constants, but are instead selected by experiment; the force equations do not need to take the same form as their real-world counterparts; and we may choose to "cap" the values of forces so that, for example, two nodes that are repelled by an inverse square law that start extremely close to one another are not sent ridiculously far apart.

The simulation of the system is to start from some arbitrary positioning of the vertices, with zero velocity, and calculate all forces, apply them, reposition the nodes according to the velocities, and iterate. Such a system could easily oscillate indefinitely, so in addition to this, a friction component is used to ensure that the process will eventually halt. This can be modelled by scaling the velocities by a constant damping factor, initially 1.0, which decays as the iterations continue. Overall, the process considers each pair of nodes once per iteration, for a fixed number of iterations, giving a time complexity of $O(V^2)$. The method makes no pronouncement on the value of the constants of proportionality for the two forces, or the friction, but suitable values can be found empirically.

2.1.2 T Kamada and S Kawai, An algorithm for drawing general undirected graphs, 1989 [KK89]

This paper again considers springs as an analogy for edges, but this time uses a single equation to describe the force. It considers a spring connecting each node pair whose natural length is proportional to the shortest-path distance between those nodes in the graph — in other words aiming to equate the geometric distance with the graph-theoretic distance. This makes rather more use of the information available about the graph, but at the cost of having to calculate the shortest path table which, at $O(V^3)$ using Warshall's algorithm, outweighs the $O(V^2)$ cost of the layout algorithm.

Although they describe their method as being for undirected graphs, it is easy to see how it could be used for directed graphs, by treating them as being undirected, and adding arrows to the drawing to indicate the direction. The issue of weighted graphs also becomes apparent: we could either preserve the weights, or consider all edges to have unit length, before finding the shortest paths. Keeping the weights would give a drawing more in keeping with the input data, but we cannot guarantee that the

weights fit the triangle inequality, which could cause the method to converge on a locally collinear solution.

2.1.3 R Davidson and D Harel, Drawing Graphs Nicely Using Simulated Annealing, 1989 [DH89]

The work of Davidson and Harel takes the energy approach to graph layout further. They note that once in the energy paradigm, the link with forces can be weakened, and arbitrary energy “penalties” can be awarded for undesirable features of a layout. For example, a fixed penalty can be given for each edge crossing in the drawing, something for which there is no equivalent within the force approach. Although it is theoretically possible to find a global minimum for our equations (this would correspond to an optimal solution in the terms of the model), it is not practical to do so — the problem is NP-hard. Instead, the energy minimisation approach is usually implemented by selecting a node with high energy and moving it so as to reduce its energy contribution.

Even this is not easy: in only the simplest of models is it feasible to solve the equations to deterministically find the position with the lowest energy for a single node while the rest are held stable. When the function is no longer smooth, but can jump (for example, as a node comes close to an edge, then crosses it), we cannot employ a gradient descent approach. In the presence of such discontinuities, the only way to proceed is to use a much cruder method to reduce the energy of the system. We consider the effect of moving a node to a random new position on the global energy; if it would result in an overall decrease in energy, then the move is taken. This process is iterated many times, with some bound on how long it should continue.

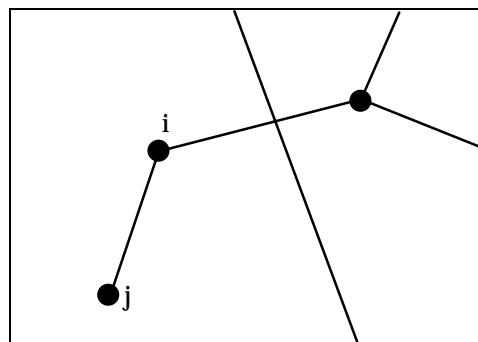


Figure 3: Fragment of a graph drawing. Moving node i to the other side of the dividing edge would not reduce the number of crossings in the drawing, since the edge ij will still cross it.

Since only a single node is moved at a time, it is possible for a local energy minimum to be found which is still much greater than the global minimum; to escape the minimum would require two or more nodes to be repositioned simultaneously (see Figure 3). Here, another idea borrowed from physics can be employed. The technique of *simulated annealing* (after annealing, the physical process whereby crystals are formed by cooling very slowly) allocates a probability to a node being moved so as to increase the global energy of the system; when a new position is selected for a node, if the move reduces the energy it is always taken, but if it would increase the energy, a random test is made to see whether to take it. The scaling factor of this probability, and the size of the random jumps to make are tied together into a single variable for the “temperature” of the system, which decays over the course of the iterations so as time progresses, the probability of taking so-called “uphill” moves reduces.

Such an approach is significantly more expensive than other methods, since each move can require $O(E^2)$ or more work (depending on the nature of the energy equations, and how costly they are to re-evaluate after moving a single node), and we might consider all V nodes before decreasing the temperature. In a dense graph, where $E=O(V^2)$, there is an overall cost of potentially $O(V^5)$.

2.2 Examination of existing implementations

Whilst investigating the problem, it seemed worthwhile to examine various approaches that have been implemented. The Java Development Kit (JDK) contains a sample applet which uses a force relaxation approach to graph drawing [JDK97]. The code is quite short, and continually applies the forces, so

that when the user interacts with the graph, the system responds while the user is still dragging a node. However, it is quite limited, requiring the graph data to be supplied as parameters to the applet, and the small examples given include “hints” to the system about what importance to give to certain nodes.

Graphlet (formerly known as GraphEd) [Grap97] is a full application in the public domain which runs using the TCL/TK windowing interface. It implements several force related methods, some methods tailored for specific kinds of graph (tree and directed-acyclic), and is written in a mixture of C++ and TCL scripts. The results are fast, and it allows convenient editing and input of graphs either via mouse or in a proprietary graph format. It is somewhat fragile though, and does not implement any energy based methods, and the number of configurable constants for some methods (which have several phases) is offputtingly high.

2.3 Analysis of the problem

After surveying the theory and existing implementations, it was necessary to formalise exactly what would be implemented, and what goals were to be aimed at. An important step was to lay down a specification of exactly what the system should be expected to do, and what kind of graph should be considered. The project proposal included some early suggestions of directions to pursue. Following more detailed reading of papers on the subject, the following questions arose which it would be desirable to use the system to answer:

2.3.1 Is there a quantitative way to compare the different layout methods?

2.3.2 All methods described above require an initial arrangement. Does the choice of starting arrangement have an effect on the convergence of the layout?

2.3.3 Is there a way to apply the algorithms to a subset of the nodes, for example if one particular area is confused, while keeping the layout for the rest of the graph?

The primary purpose of the system would be to allow comparison of the various algorithms, in as flexible way as possible. The intent was not to create a fully featured package for graph layout, but rather one which was sufficiently usable to examine the above questions. This approach affected the features that would be implemented: for example, to compare the algorithms it would be desirable to be able to start them from the same initial position, which means that a save and load layout feature was needed. On the other hand, whilst the ability to produce PostScript output of the results might improve the quality of the output for this dissertation, it was not deemed to be of sufficient value to implement.

To return to the question of what data should be permitted, I chose to implement a system to draw undirected, unweighted, non-reflexive, labelled graphs using straight lines. Each of these requires some justification:

undirected — to go from an undirected to directed drawing is simply a matter of adding appropriate arrows to the edges; though simple, I felt that this could distract from the main problem.

unweighted — edge weight information could have been provided, and different methods could have chosen whether or not to take account of this extra data. However, I believe that this again is subsidiary to the main problem, and that it would be more complicated to quantitatively compare methods that took account of edge weights.

non-reflexive — this in part relates to the decision to use straight lines for edges: it is impossible to draw a reflexive edge using a single straight line. I would also include under this heading the tacit decision to disallow multiple edges: two nodes are either connected by an edge, or not. Both decisions reflect the general sentiment to attack the core problem of graph layout, and not to become bogged down with details which can occur in the more general case.

labelled — it is interesting to note that if we take a graph layout as fixed, and try to label the nodes so as to minimise the number of labels crossing one another, or crossing edges, then this problem is in the complexity class NP-hard [CMS95]: it is as difficult as the original problem of graph layout. It might be possible to combine the stage of label placement with that of graph layout, but this starts to resemble the closely related problem of circuit layout. I decided to display labels on nodes, but not to be concerned about the aesthetics of this aspect of the layout, and place labels in a fixed position relative to their node. Edges will not be labelled.

straight-lines — this is perhaps the most significant simplification of the problem. The language I used to implement the system allows the use of straight lines or Bezier curves for edges; these could easily have been adapted to allow polylines (lines with ‘dummy’ nodes along them, allowing bends in the lines) or rectilinear lines. The main reason for choosing straight lines was that the methods described above virtually assume the edges to be lines, if the edges are considered at all. A further motivation is that should we wish to test for intersection of two edges, or find the perpendicular distance between a node and an edge, the geometry is significantly simplified (and hence faster) for direct straight lines.

If we look again at the criteria for making a simple drawing, we find that the first two (1.1.1, two edges cross at most once, and 1.1.2, edges sharing a vertex do not cross) cannot be violated if we use straight lines to draw edges. A final post-hoc quasi-justification is that any planar graph can be drawn with no edge crossings using straight lines [HR90]; this result does not extend to non-planar graphs, but we can comfort ourselves in the knowledge that by using straight lines we probably aren’t causing ourselves any significant penalty compared to the unconstrained case.

2.4 Choice of language

From this analysis it was possible to design the system in terms of functionality, and how this would be achieved. While it is perhaps desirable to separate the design from the details of implementation, some factors, especially the choice of language, can have influence over the design. I chose to use Modula-3 to implement the system, mainly for the wealth of library code available, especially a module called “GraphVBT”, which contains routines to allow the display of graph data in a windowing system. The object-oriented nature of the language meant that it would be easy to take the node, edge and graph objects already defined and derive new objects from them with the additional functionality required to support graph drawing. Other factors influencing my choice included my familiarity with the language, its support for modular code, local availability and support, and my personal preference for nested procedures and garbage collected storage. Since the intention was to be able to compare methods without being overly concerned for high performance, Modula-3’s run-time checking was preferable to the more streamlined code of a language like C++.

3. Implementation

The aim of the implementation stage was to produce an interactive system that allowed the questions posed in the analysis stage to be answered. This required a substantial amount of planning to be done before coding began. I have picked out what I consider to be the salient details of the implementation, trying to give enough of the particulars that a moderately competent programmer could implement a similar program without understanding in detail the action of the algorithms being investigated.

3.1 Choice of equations

Three major iterative methods were defined, based closely on those suggested in the papers of Eades, Kamada and Kawai, and Davidson and Harel (discussed in 2.1). To make the methods as comparable as possible, I decided to try to minimise the number of user-alterable parameters without over-constraining the flexibility of the system. What follows is a my formulation of the methods, put into a consistent notation.

3.1.1 Eades

The first model, based on that proposed by Eades, applies two forces between pairs of nodes i and j . Let \mathbf{r}_i be the position of the i 'th node. All forces are calculated in the direction $(\mathbf{r}_i - \mathbf{r}_j)$. For each $i, j, i \neq j$

$$\text{Vertex Force, } \mathbf{VF}_{i,j} = \frac{c_{\text{vertex}}}{|\mathbf{r}_i - \mathbf{r}_j|^2}$$

l , the natural length of the edge is defined as $(\sqrt{\text{width of drawing area} \cdot \sqrt{(\frac{1}{2}|V|)}})$. Then

$$\text{Edge Force, } \mathbf{EF}_{i,j} = \begin{cases} c_{\text{edge}} \cdot (|\mathbf{r}_i - \mathbf{r}_j| - l) & \text{if } \{i,j\} \in E \\ 0 & \text{otherwise} \end{cases}$$

We calculate the total force on a node i in the as

$$\mathbf{F}_i = \sum_j \mathbf{F}_{i,j} = \sum_j (\mathbf{VF}_{i,j} + \mathbf{EF}_{i,j})$$

3.1.2 Kamada

The second model uses a single equation based on $d_{i,j}$, the graph theoretic distance between nodes i and j . The force is again in the direction $(\mathbf{r}_i - \mathbf{r}_j)$

$$\mathbf{F}_i = \sum_{j, i \neq j} c_{\text{kamada}} \cdot \frac{(|\mathbf{r}_i - \mathbf{r}_j| - d_{i,j} \cdot l)}{d_{i,j}^2}$$

We shall use $c_{\text{kamada}} = c_{\text{edge}}$, and we justify this by observing that in the case that $(i,j) \in E$ then

$$\begin{aligned} \mathbf{F}_{i,j} &= c_{\text{kamada}} \cdot (|\mathbf{r}_i - \mathbf{r}_j| - l) / l^2 \\ &= c_{\text{kamada}} \cdot (|\mathbf{r}_i - \mathbf{r}_j| - l) \\ &= \frac{c_{\text{kamada}}}{c_{\text{edge}}} \cdot \mathbf{EF}_{i,j} \end{aligned}$$

which is equal to the edge force in the Eades case if $c_{\text{kamada}} = c_{\text{edge}}$

3.1.3 Annealing

The energy equivalent of the force equations is found by integrating the forces from Eades' method with respect to distance, so we obtain:

$$\begin{aligned} \text{Vertex Energy for } i \neq j, VE_{i,j} &= \int \mathbf{VF}_{i,j} \, d\mathbf{r} = \frac{C_{\text{vertex}}}{|\mathbf{r}_i - \mathbf{r}_j|} \\ \text{Edge Energy, } EE_{i,j} &= \int \mathbf{EF}_{i,j} \, d\mathbf{r} = \frac{1}{2} C_{\text{edge}} \cdot (|\mathbf{r}_i - \mathbf{r}_j| - l)^2 && \text{if } \{i,j\} \in E \\ &= 0 && \text{otherwise} \end{aligned}$$

In addition to this, we would like to include a penalty for edge crossings. We could simply implement this as

$$\begin{aligned} \text{Edge-Edge Energy, } EEE_{m,n} &= C_{\text{crossing}} && \text{if edge}_m \text{ crosses edge}_n \\ &= 0 && \text{otherwise} \end{aligned}$$

However, simulated annealing works better if the energy functions are continuous as a parameter (such as the position of a node) is varied. Also, two edges which nearly cross (say if a node is very close to another edge) is almost as bad as a crossing; we would wish to penalise this almost as much. This leads to the following equations which use $p_{m,n}$, the perpendicular distance between edge_m and edge_n (defined in Appendix 1) and $p_{\text{threshold}}$, a suitable minimum distance below which we consider the edges to be crossing:

$$\begin{aligned} EEE_{m,n} &= \frac{C_{\text{edges}}}{p_{m,n}} && \text{if } p_{m,n} > p_{\text{threshold}} \\ EEE_{m,n} &= C_{\text{crossing}} = \frac{C_{\text{edges}}}{p_{\text{threshold}}} && \text{if } p_{m,n} \leq p_{\text{threshold}} \text{ or if edge}_m \text{ crosses edge}_n. \end{aligned}$$

This causes the penalty for edge proximity to fall off smoothly as the edges move further apart (once the threshold has been exceeded).

The total energy of a layout is then

$$E = \sum_i \sum_j (VE_{i,j} + EE_{i,j}) + \sum_m \sum_n EEE_{m,n}$$

Although the equations have meaning and impact on convergence for the force based methods (they scale the velocity changes, and hence the size of the steps taken towards equilibrium), since simulated annealing seeks to minimise energy, the individual values of the constants should have no effect on the results, only their ratio.

In summary then, these three methods require three constants between them. By setting any of the constants to zero, we will be able to observe the effect of omitting the corresponding equation.

3.2 Friction, cooling and termination

To ensure termination each method uses some factor, drawn from physical analogy but with no strong grounding in physics. The force methods use a notion of friction. A frictionless system would have

$$\mathbf{v}'_i = \mathbf{v}_i + \sum_j \mathbf{F}_{i,j} \text{ — the velocity is incremented by the sum of the forces on that node (we set the weight of all nodes to unity)}$$

Our frictional system instead scales the new velocity by a factor k :

$$\mathbf{v}'_i = k \cdot (\mathbf{v}_i + \sum_j \mathbf{F}_{i,j})$$

where k is initially 1, and decays over time to zero. When k is sufficiently small (say 0.1) we may conclude that no more significant alteration to the layout will happen, and halt the process. In fact, we could also test to see whether any of the movements at each iteration was significant and if not, terminate the process at this stage.

Simulated annealing has the related concept of the current temperature of the system. It is used in two ways. Firstly, the position to investigate moving a new node to is defined by a radius r , of

$$r = 0.4 * T * (1 + R) * \text{width}$$

where R is a random real number drawn uniformly from the range 0..1.

Secondly, it is used to determine the probability of taking a move to a higher energy position:

$$p(E_{\text{new}}) = \frac{c_{\text{uphill}} \cdot \exp(-c_{\text{scale}} \cdot (E_{\text{new}} - E_{\text{old}}))}{(T \cdot E_{\text{old}})}$$

c_{uphill} is another user alterable constant. c_{scale} is a scaling factor, calculated to give a 10% chance of a 10% increase in system energy at unit temperature. There is no convenient test for termination; we must wait till the temperature is sufficiently low that we expect no further significant activity.

Since all methods require a factor that is initially 1 and falls off over the course of the algorithm, we unify friction and temperature to give a single notion of temperature, T . We still require a cooling scheme, which is periodically to reduce T geometrically.

$$T_{n+1} = c_{\text{cool}} \cdot T_n$$

where c_{cool} is another user alterable constant.

Finally, we observe that T , whilst defined globally, is only applied to a single node at a time; we can then replace the global T with a T_i for each node. If all T_i are initially 1 and are decreased in accordance with the cooling scheme then the effect is the same as using a single global temperature; however, this now gives the freedom to set the temperature of certain nodes to zero, meaning they are effectively rooted, and alter the temperature of others to observe the effect of only giving freedom to a subset of the nodes. This will help answer question 2.3.3.

3.3 Quantitative analysis

The crucial part of designing a quantitative way to examine the methods to answer 2.3.1 was the realisation that the energy equations used in methods like simulated annealing can be used to give an objective score for the current layout. Being derived from aesthetic considerations, a lower score, indicates a better result with respect to the aims of keeping nodes well spaced, edges an even length, and pairs of edges not too close. The caveats to this are that the score depends on the constants used,

and for a given graph, the minimum score cannot be determined analytically. Although these energy equations were motivated by the energy methods, they can be equally applied to force methods, and by using the same scoring functions and constants, we can compare the performance of different methods on the same problem.

This can be achieved by comparing the energy of the final layout converged on by a particular procedure, as well as taking readings of the energy as a layout procedure is in progress, and plotting these over time. Readings should be taken at regular intervals, say every second or tenth of a second, although it is important to be aware that unless the reading is instantaneous then it will affect performance of the procedure being observed.

3.4 Effect of starting position

Question 2.3.2 asks what effect the starting configuration has on the eventual layout. To answer this, we need to have some ways to generate starting configurations. The following methods were implemented:

3.4.1 Random Configuration

In the papers studied, starting configurations were usually found by placing each node at a random location within the drawing area. It is perhaps prudent to ensure that nodes do not coincide, and perhaps are a sensible minimum distance apart.

3.4.2 Circular configuration

One canonical way to arrange nodes is to space them evenly about the circumference of the circle with a radius perhaps $2/5$ of the width of the drawing area, so that the i 'th node of n is placed at $2\pi i / n$ radians about the circumference. For very small graphs (<10 nodes) this may give an adequate final drawing.

3.4.3 Breadth-first arrangement

A slightly more involved approach would be to pick a node and perform a breadth-first search from it, and place it at the left hand end of the drawing area. All nodes at a depth of 1 are placed in the first column, at a depth of 2 in the second column and so on, scaled so that the columns fill the drawing area. Care is needed to handle graphs that are not connected. Since ordering the nodes in any given column to minimise the number of edge crossings is a potentially exponential operation, the nodes should be arranged in an arbitrary order.

3.4.4 Divide and conquer approach

The traditional computer science approach of “divide and conquer” can be applied to graph layout, albeit crudely. We can define the centre of a graph (or sub-graph) quite easily, as (one of) the node(s) with the lowest shortest path distance to reach any other node. So we can find the centre of the current (sub)graph, place it at the centre of the area, divide the remaining nodes into two subsets, and repeat the process on each subset partitioned into one or other of the two halves of the drawing area. See Appendix 2 for more details of this approach.

3.5 Tools Used

The system was implemented in Modula-3 v3.6 on Thor. By far the most important tool used was the GraphVBT library, which allows a graph to be created in a window by specifying the nodes, edges and location of the nodes. There is a great deal of flexibility with regard to the size, shape and colour of the nodes and edges; suitable defaults for these were hard-coded in the program, or calculated from other values. The graph, and each of its edges and nodes are represented as objects within the Modula-3 language. It was hence quite natural to inherit from these to create enhanced objects with additional functionality for layout.

One major deficiency of the library is that it has no provision for allowing the user to interact with the graph: it does not respond to mouse clicks. By overriding the “mouse” interactor method of the window, it was possible to allow the closest node to the pointer to be picked up and dragged to a new position, and, depending on the combination of mouse buttons pressed and the locality of the pointer, add new nodes or edges. The program was designed and written to accommodate the deletion of nodes and edges, but code was never written to implement this.

As with any tool, the gains to be had are not without some drawbacks. One feature of GraphVBT is that there is no control over the placement of node labels — they are placed centrally over the node they label. I could have replaced this with a method to place labels at any position relative to the node location, but chose not to, and to some extent this influenced the decision not to make label placement part of the requirements (section 2.3).

The GraphVBT interface uses two internal co-ordinate representations of the graph, one for the integer window co-ordinates, and another for the internal real graph representation, and carries out the conversion between the two. When I started implementing, I stored the node positions a third way, using integer co-ordinates. It soon became apparent that there was no benefit to be gained from this approach, as any speed-up from using an integer representation was lost by the need to constantly change back to reals, for interaction with the GraphVBT representation, and for distance calculations. Such an unholy trinity of representations could not last, and so the real GraphVBT co-ordinates were used as the absolute position of the nodes. To change the code I had written from integer to real co-ordinates was the work of an hour or two.

3.6 Further implementation details

Graph, edge and node objects were derived from those defined in GraphVBT. Little additional code was required for nodes and edges, only overriding the default code to ensure that correct alterations to the additional data-structures were made. The nodes were stored in a canonical quadtree representation to enable efficient locality searching, which deserves some explanation.

3.6.1 Quadtree

The quadtree is initially empty, with a rectangle defining the area which it covers. It consists of branch records, where four pointers point to the next level, one for each quadrant of the current rectangle, or else a leaf, which contains a single node. *Figure 4* illustrates why the representation is canonical: there is just one way of partitioning up the area by quartering rectangles until each node is in a rectangle on its own.

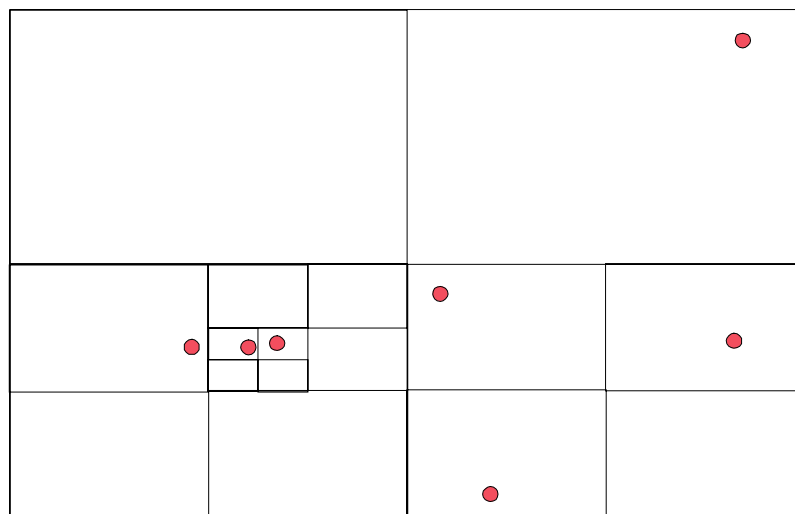


Figure 4: How a quadtree partitions up the space

The following pseudo-code describes the operations on a quadtree:

```
AddNode(node, quadtree) =
  repeat
    if quadtree is an occupied leaf then convert leaf to branch
    find which quadrant of quadtree.rectangle the node is in
    if quadrant is empty then create a new leaf and place node in it
    else quadtree ← quadtree.quadrant
  until node has been placed

RemoveNode(quadtreepointer) =
  // quadtreepointer points to the node's position in the tree
  parent ← quadtreepointer.parent
  delete quadtreepointer from parent

FindWithinRectangle(rectangle, quadtree) =
  if quadtree points to a leaf, then output the node if node ∈ rectangle
  else for each quadrant of quadtree
    if rectangle intersects the quadrant rectangle then
      clip rectangle to quadrant rectangle
      FindWithinRectangle(clippedRectangle, quadrant)
```

AddNode has cost proportional to the depth of the tree which, provided there is an even distribution of nodes, should be $O(\log_4(V))$. RemoveNode has constant cost; hence the overriding cost of a move, which is a compound delete and add, is just that of add. FindWithinRectangle also depends on how many nodes there are within the rectangle, giving a cost of $O(\log(V) + N)$, N being the number of nodes found.

The quadtree representation demonstrates its efficiency when it can be used to search for nodes within a rectangle or an area composed of (preferably non-overlapping) rectangles. This is good when we are attempting to improve efficiency by only considering nodes within a certain distance of a specified node when calculating forces or energies (as the contribution of distant nodes, being related to the inverse square of the distance, soon becomes negligible). It is not applicable when evaluating edge proximity energy, which is the over-riding cost when it comes to applying the energy method. A more involved data-structure might have coped better with this, at the cost of a good deal more design and debugging.

3.6.2 Other Algorithms Used

Certain other algorithms of a computational geometry nature were needed. These are explained in full detail in Appendix 1, but the outlines are sketched here.

Intersection of two line segments — we need to be able to test whether two edges, which are line segments, intersect. A fast reject case is if the two bounding rectangles do not intersect; otherwise, it is still possible the two edges do not intersect. We argue that the lines intersect if and only if the two endpoints of one edge fall on different sides of the infinite line implied by the other edge, and the same test with the roles of the edges interchanged.

Perpendicular distance between two line segments — we define the perpendicular distance between two edges to be the shortest of the perpendicular distances between one endpoint and the other line segment. This in turn is defined as the shortest of the distances between the point in question and the two endpoints if the point does not lie in the area swept out by the line segment perpendicular to its direction. Otherwise it is the perpendicular distance between the point and the infinite extension of the line segment. This is illustrated by the diagram in Appendix 1.

All-pairs shortest paths — the Kamada and Kawai algorithm requires knowledge of the all-pairs shortest paths matrix. Warshall's algorithm achieves this in time $O(V^3)$, which in complexity terms overrides the $O(V^2)$ cost of the layout algorithm. While faster ($O(V^2 \log V)$) algorithms exist, I decided to use the simpler Warshall's, since the actual running time for graphs of moderate size (100 nodes) and larger is still not significant. Also note that we only need to calculate the matrix once for a graph, provided that it does not change.

3.6.3 The Graph Object

The object used to represent graphs was derived from the GraphVBT object. In fact, a two-stage inheritance was used. Firstly a Graph object was defined, which augmented its parent with routines to add a node or edge, to load and save a layout, and to perform some counts of the number of nodes, edges, and crossings in the drawing. In addition, it overrides the default routines to allow mouse interaction with the graph in order to allow user manipulation and addition of nodes and edges.

A further object, “Energy”, inherits from a Graph and adds the capability to evaluate the energy of a graph configuration, based on the three constants (defined in section 3.1), as well as methods to get the value of the constants from the user interface, and to allow nodes to be started with different initial temperatures. The idea of this two-layer descent was to allow clarity of code: procedures that only needed to work with a basic graph would be associated with a Graph object; those that examined the energy properties would take an Energy object.

I initially planned for these objects to be ‘virtual’, and never instantiated; instead, an object would be defined for each layout method used which would provide its own layout procedure. This proved problematic: since it was required that different methods be applied successively, it meant that the object type would have to be changed. The only way to achieve this would be to copy all of the graph information into a new object of the appropriate type every time a different layout method was to be used. This felt clumsy and inelegant: the solution adopted was to add a “layout” procedure as a variable of the Graph object, which was set when a layout algorithm was chosen, and called by a wrapper method within the object.

3.6.4 Applying Layout Procedures

The mechanism for applying the layout algorithms is as follows:

```
Get the constants to be used from the user interface
Set up other values, such as the natural edge length and initial temperature of
nodes
Call the graph.layout() method
Display the results
```

For the ‘simple’ layout methods (random, circular, etc.) the layout procedure is a straightforward implementation of the algorithm. For the force methods (Eades and Kamada), the implementation is as follows:

```
Initialise the node velocities to zero
repeat
  calculate the forces and apply the new velocities
  if time since last sample is more than the sampling period, take an energy reading
  if the number of iterations mod ~20 is zero, then
    reduce the temperature
    *display the current configuration
until the convergence criteria are met
```

The line marked * is to give some visual feedback and can give quite a pleasant animation effect as the graph can be seen to unravel itself into a clearer drawing. It is omitted for testing purposes, as the time to update can affect the timing results.

The energy method (simulated annealing) has a slightly different implementation:

```
repeat
  repeat
    choose a node
    repeat ~20 times
      evaluate energy of moving node to randomly chosen position
      decide whether to move node
    until all nodes have been tried
  reduce the temperatures
```

```
*display the current configuration
until all nodes are cold (T<0.1 say)
```

Again, the asterisked line can be omitted to improve the accuracy of timing results, at the expense of giving user feedback. Some care is required: to calculate the energy of any position can require up to $O(V^2)$ work to calculate the energy between all pairs of nodes. When we are only moving a single node at a time, we can do better than this, since the energy between all pairs of nodes, except those pairs involving the node which is being moved, remains the same; only V new node-pair calculations need be done to evaluate the node energy of a new position, between the node moved and all other nodes. By keeping a matrix of node pair energies, we can update the total energies per node by calculating

$$E_i' = E_i - E_{i,j} + E_{i,j}' \text{ where } j \text{ is the node which has moved}$$

That is, the new node energy is equal to the old node energy, less the energy from the old node position, plus the energy from the new node position. A similar approach can reduce the cost of calculating the energy from edge pairs.

3.7 Unevaluated code

Since the evaluation section of this report seeks to answer the questions posed in 2.3, there is a certain amount of code that was written to facilitate answering these questions, which however does not figure within the evaluation, and merits mention in this chapter. One such tranche of code implements an on-the-fly plot of the energy of the graph layout over time as the layout is created. This gives an immediate quantitative analysis of the success of the current method. For the purpose of the evaluation, it was required to plot several averaged results on a single graph, and so raw values were taken from the system, and processed using a spreadsheet package.

Load and save routines were required to compare layout procedure performance from the same initial configuration. In addition it was useful to be able to input graph connectivity information without any layout information, and to use a reasonable display method (such as random or circular) to display it. The file format chosen was a simple text file (to allow user-created input), which is structured as follows:

```
n, number of nodes
#
optional node numbers and corresponding labels (need not label all nodes)
#
edge information encoded as pairs i,j where i & j are both in the range 1..n
[#
optional node locations for nodes 1..n]
EOF
```

A certain amount of debugging code was generated, for example to list the contents of the quadtree and information about the graph. Most of this was removed on satisfaction that the code seemed to perform correctly after some testing, but certain parts were left in, in case they should be later needed. Some slight variants of the layout algorithms were tried: a version of Eades was implemented which approximated the full version by only considering node pairs within a fixed radius of each other. It was moderately faster, but not significantly so, and so I decided not to investigate further.

3.8 Realisation of the project

In the project proposal, a timetable for the implementation was set out. Initially these were closely adhered to, and a notebook was kept detailing the progress and sketch designs and ideas. Programming the system began in mid November, and by the end of the Christmas vacation I had a system which implemented the user interface, save and load facilities, and the two force methods, slightly ahead of schedule. Around about this time, the written log lapsed, and was superseded by a typed log, recording the progress of the programming.

The list of potential extensions to the basic system given in the project proposal proved to be somewhat ambitious, and with the belief that there would not be room in this write-up to do justice to too many different investigations into aspects of graph drawing, I decided to omit extending the methods into three-dimensions (suggested in work package for 28 January to 11 February). The main deviations from the original timetable in the project proposal were to attempt sections in a different order. I deemed the system to be complete a few days after the proposed end, at the start of March.

The system was implemented in a structured modular fashion, divided into sixteen major commented files — half of these dealt with different layout methods, six with the different object types that were defined, one dealt with plotting the energy, and another defined the actual program, linking the interface to the appropriate procedures and methods, allowing the layout functions to be called. Further details of the implementation can be found in Appendix 3.

4. Evaluation

The evaluation of this project falls into two parts; first a qualitative overview of the system and some observations on the performance of the algorithms that were implemented, and second the quantitative results of using the system to investigate the questions posed in the analysis (2.3) and interpretation of these results.

4.1 Performance of the system

A number of basic graphs were created to test the system, and the results of testing using these gave encouragement that the system was working correctly.

4.1.1 The Cube: a small, structured graph

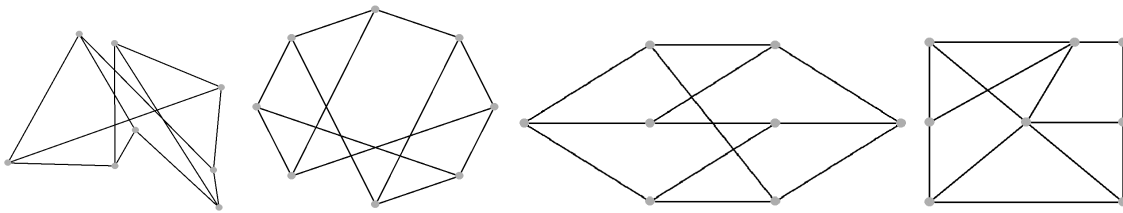


Figure 5: Four drawings of the cube graph. Left to right: Random, Circular, Breadth-first, Divide-and-conquer

Figure 5 illustrates the four different initial layout algorithms that were implemented (described in 3.4), applied to the same graph, the cube graph (the graph which describes the connectivity of a wire-frame cube). All executed instantaneously. It is easy to check that these are all drawings of the same graph, although the underlying structure of the graph is not particularly clear from any of them.

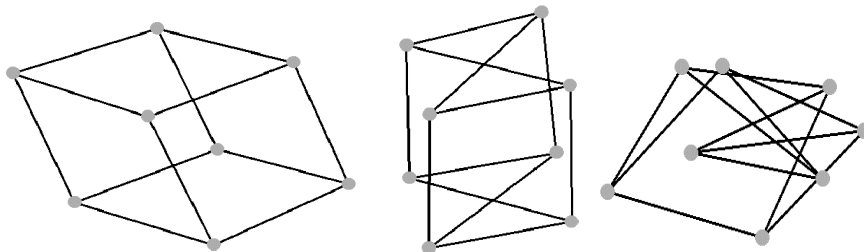


Figure 6: Result of running Eades' Algorithm on the cube graph from different initial configurations

Constants for the iterative methods were determined empirically, using a single node pair, and adjusting the values until an appropriate response was found. The results of running the first of the iterative methods on the cube graph (starting from a random layout) led to three types of solution (shown in Figure 6) — most of the time, it converged upon a drawing closely resembling a cube drawn in projection, with the edge lengths approximately equal; sometimes it would give a result which resembled the cuboid drawing but with a “twist” in it; occasionally it would finish with a drawing which was uneven and asymmetric.

The algorithm of Kamada and Kawai (referred to as Kamada) was equally fast, and almost invariably converged on the drawing of the cube graph illustrated in Figure 7, subject to rotation and reflection. The energy of the final configuration was quantitatively less than those found by Eades' method on average, and this was reflected in the generally more aesthetic layouts that the Kamada method found.

Simulated annealing (abbreviated to Annealing) took longer than the other two iterative methods, and the results resembled those found by them, but appeared cruder. This is to be expected due to the nature of the method: rather than the position of the nodes being directed by forces, they are found by testing the energy contributions from positioning the nodes at different random locations. However,

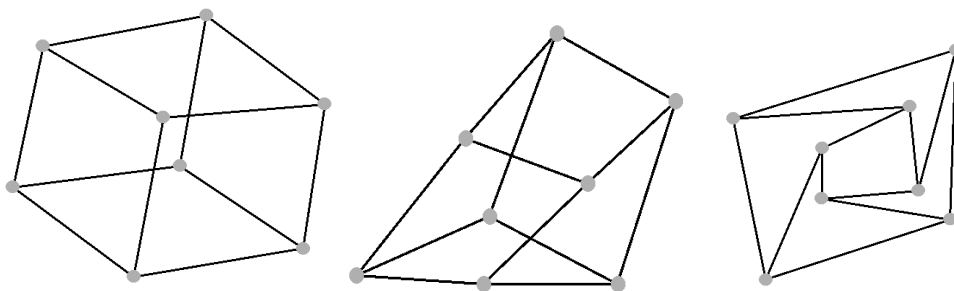


Figure 7: Effect of i) Kamada and Kawai's method and simulated annealing with low- (ii) and high- (iii) edge crossing penalty.

the energy of the final position was slightly less than that found by the Kamada algorithm in the majority of tests. The effect of changing the constants is graphically demonstrated in *Figure 7*, where the middle drawing is the effect of Annealing with a low penalty for edge crossings: the perspective-like drawing is the low-energy solution for this system; the right hand drawing shows the effect of increasing the edge-crossing penalty by a factor of ten: in this case, it is the planar layout which has the lowest energy, and hence is found by the algorithm. These two ways of drawing the cube graph were used in the introduction (*Figure 2*) to illustrate conflicting goals in graph drawing: it is interesting that a single method will find either drawing depending on the value of a single parameter.

4.1.2 The Web: a larger example

Having seen that the iterative methods are working as intended on a small example, to distinguish between them a larger example, a web graph was chosen. This was designed as two concentric nonagons with a single point in the centre, connected like a spider's web. *Figure 8* shows the effect of the different methods, again starting from an initial random placement. Eades' algorithm succeeds in revealing some of the structure, but still ends with a layout which looks as if it has been folded over itself, with many edge crossings. Kamada quickly and consistently arrives at the middle drawing, revealing the structure as it was designed. Annealing took significantly longer, a matter of about ten seconds instead of one or two, and found a planar drawing which, again due to the more probabilistic nature of the algorithm, displayed less symmetry than the middle drawing.

We would now like to start to examine the energy 'scores' for these layouts, to see if they correspond to the aesthetic assessment. The energies are calculated based on the same constants used to create the layouts, and have no direct interpretation; they give an objective scale on which to compare the layouts, with a lower energy being considered 'better'. Random initial configurations of the web graph have an energy of around 80,000 on average; Eades typically reduces this to around 35,000. The layout converged on by Kamada has an energy of 29,200, while Annealing yields around 29,500. We see that from a relatively high initial energy, the methods all succeed at significantly reducing the energy, but there is a relatively small energy difference between a poor layout (Eades) and a very clear one (Kamada), and an even lesser difference between an acceptable layout (Annealing) and one which shows good symmetry (Kamada).

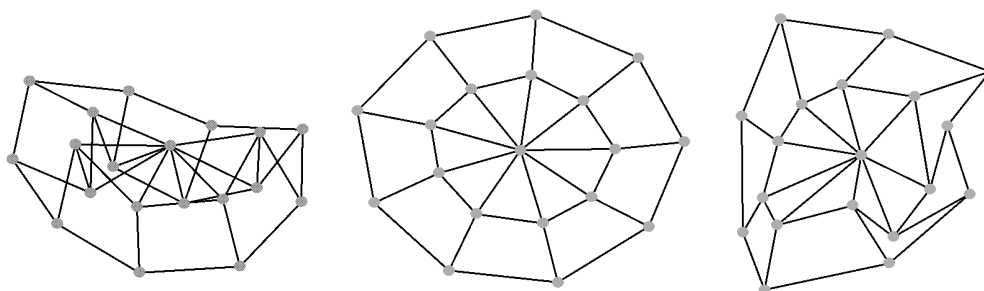


Figure 8: Effect of running i) Eades' ii) Kamada and Kawai iii) simulated annealing on the web graph

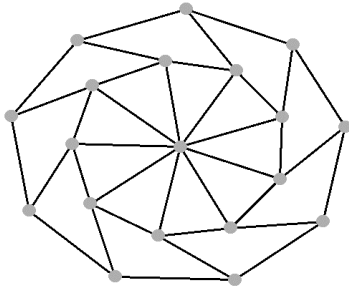


Figure 9: Result of applying Eades after Kamada on the web

The consistency with which Kamada and Annealing found the planar drawing suggests that the graph has a single energy minimum, which the two methods are approaching. In fact the lowest energy configuration that was found is shown in *Figure 9*, which was found by applying Eades to the layout found using Kamada; it has an energy of 29,100. This makes sense: although Kamada is good at finding a layout which takes account of the whole of the graph data, it does not reflect the energy equations which are derived from the Eades forces; hence by applying Eades to the approximate minimum found by Kamada, it is improved slightly in terms of energy. This is the sort of effect that needs to be investigated in more detail to answer question 2.3.2.

4.2 Comparing the methods quantitatively (2.3.1)

The evaluation so far has been quite anecdotal, intended to give confidence that the algorithms are behaving correctly within their own parameters, and noting certain points of interest. We now take a more quantitative and thorough approach to answer the questions that were posed in section 2.3. It has already been implied that the energy is an appropriate way to score a layout (it was originally advanced as such in the implementation section, 3.3). To compare the methods, each was run on the same initial random layout of randomly created graphs (see [Pal85] for details). The graphs were created with a fixed number of nodes, with $O(V)$ edges added for sparse graphs, and $O(V \log V)$ edges added for the dense case. Each method was run three times and the results averaged.

Figure 10 shows that, for sparse graphs, there is not much to choose between the methods. Above fifty nodes, Annealing quite consistently achieves the lowest energy, with Kamada not significantly worse than this, and Eades not much worse again, perhaps to be expected as graphs with relatively few edges have less potential for edge-crossings, and Eades can succeed with respect to spreading out nodes and keeping what few edges there are close to the desired length. The price we have to pay for the performance of Annealing is the time taken to find the layout: with no way to detect convergence and terminate, and the more complex iteration cost it is perhaps no surprise that it takes up to twenty times as long as the other two methods (*Figure 11*).

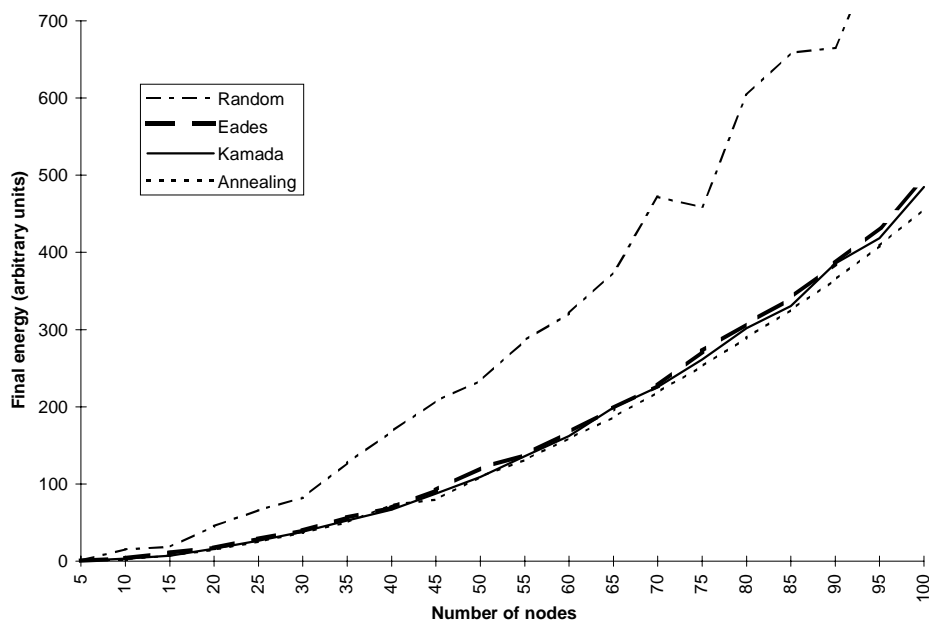


Figure 10: Comparison of the layout methods on sparse graphs

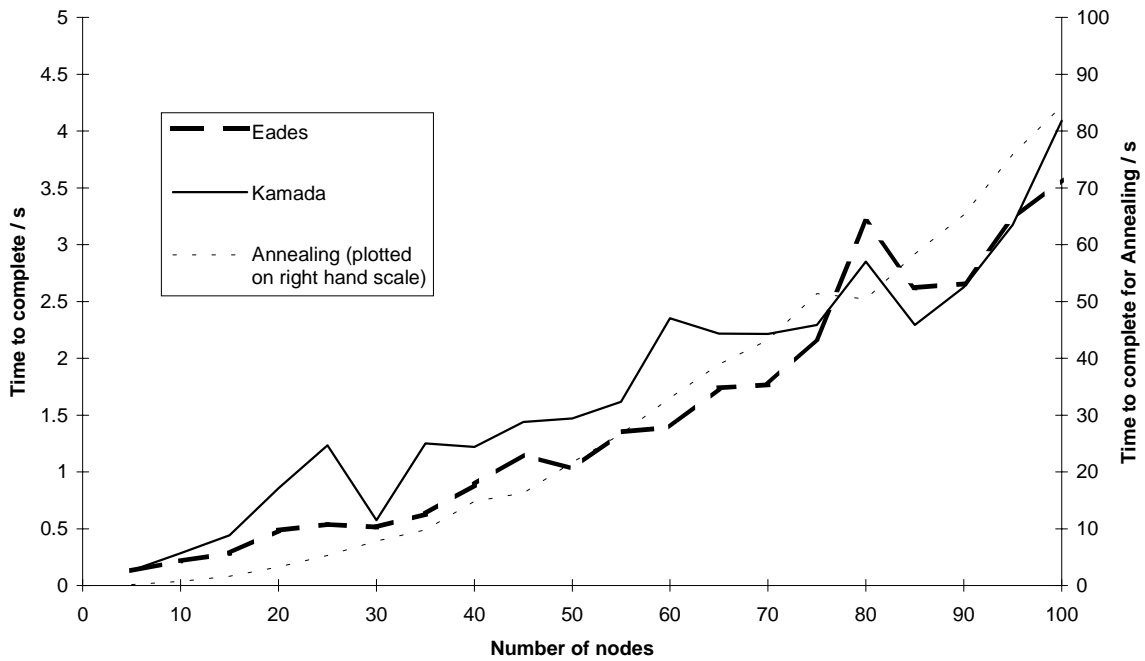


Figure 11: Time comparison for sparse graphs

Looking at dense graphs, the difference between the methods is more visible, though still slight. Figure 12 shows exactly the same ordering as before, with Annealing doing best; in this case Eades outperforms Random layout as before, but not by as clear a margin, as the penalty for edge-crossings becomes significant, and the improvement in energy comes mainly from the evening of edge lengths and spreading out of nodes. Again, the improvement from Annealing comes at great price: in the worst case, it takes thirty times as long as the force-directed methods. In conclusion, for these general graphs of moderate size, we have seen that Annealing performs the best with regard to minimising the energy of the layout, and Kamada does a good job, equalling Annealing for small sized graphs (those with fewer than fifty edges); however, the time to perform simulated annealing is measured in minutes, whereas the force-directed methods take only seconds.

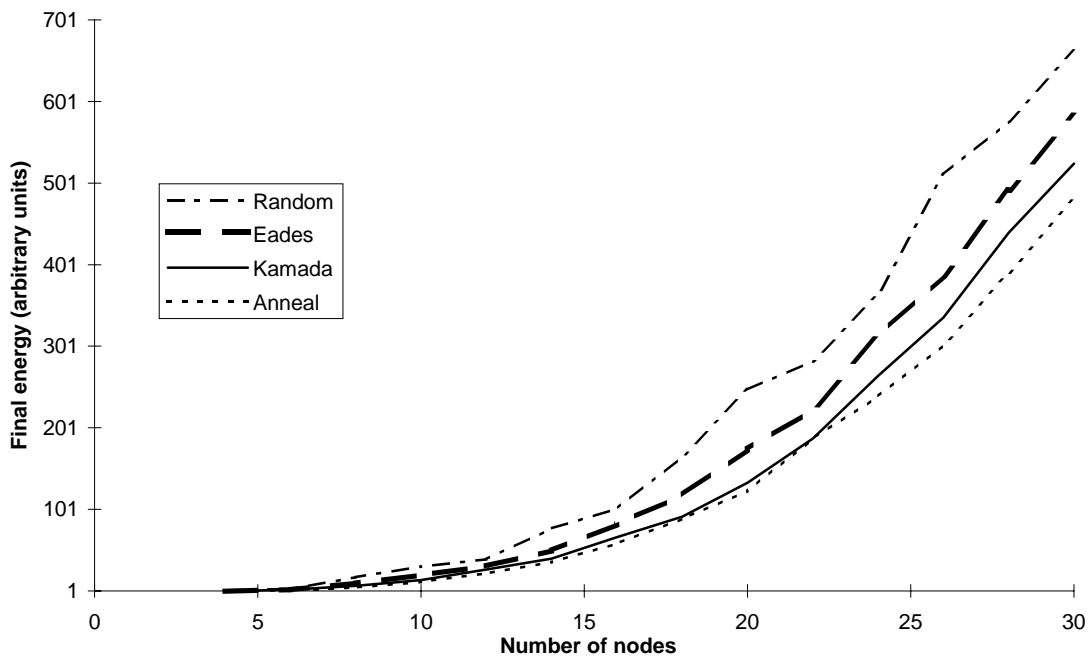


Figure 12: Comparison of the different methods on dense graphs

4.3 Examining the effect of the starting configuration (2.3.2)

To answer this, I chose to examine the effect of using different initial configurations on the final energy of three particular graphs, meant to resemble ‘real’ data that it would be reasonable to want to display. The data was pre-processed using one of the seven layout methods — the four heuristics (Random, Circular, Breadth-First or Divide-and-Conquer) and the three iterative methods (Eades, Kamada, Annealing) — and this initial arrangement was saved. Then each of the three iterative methods was executed on the initial configuration three times, and the final energy averaged across the three executions.

4.3.1 Large and unstructured: a social network

The first example tested was a social network with 143 nodes, generated by recording the ocular interactions of members of my college over the past few years; the results are illustrated in *Figure 13*. In answer to the question “does the starting configuration make a difference”, the answer must be a qualified ‘yes’. Eades achieves a lower energy if working from a position generated by an iterative method — around 870 instead of 888 starting from a random configuration. Likewise, Kamada achieves its best result when run twice, and Annealing achieves the best score, 809, when executed on a layout found using Kamada (these two layouts are illustrated in Appendix 4). But these results should be approached carefully: the differences though significant are not great, and we should also be wary of the spread of values — it turns out that the better results cluster quite tightly round the mean (the range for Annealing after Kamada being 808-810), while the spread for Eades on a Random layout is 880-900.

Another point to note is that Eades seems to undo some of the good work done in preparing the configuration, raising the score from the low 800’s from Annealing and Kamada to about 870, although this is still better than from any of the heuristic methods alone. Keeping an eye on the timings (not illustrated), and we note that, as before, the force-directed methods take no more than six seconds of CPU time, while Annealing consumes nearly three minutes in all cases. When looking at the energies achieved, it is reasonable to ask whether the small difference in energies found by running Annealing is worth the minutes of waiting.

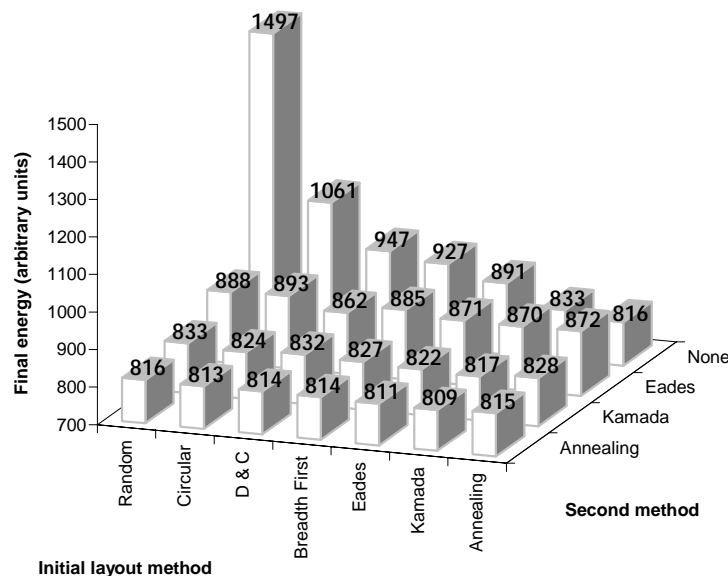


Figure 13: Effect of starting configuration for a large graph (social network)

4.3.2 Medium-sized and planar: the London Underground

The second example that was used was again drawn from everyday life, being a slightly modified version of the London Tube Map. Most nodes with degree 2 were removed leaving 55 stations, thus capturing the connectivity of the network. It was not obvious that this would be planar, but a planar drawing was found by Kamada during testing. The same methodology was used as above, and the results are shown in *Figure 14*. Here we see that the results are exceedingly close, with Kamada just outperforming Annealing on average but not in every particular case. Eades is not that far behind, and again shows an improvement when working from a layout which has been pre-processed. In all cases, the best performance of each method is seen when the initial layout method was Kamada, as in the previous example, suggesting that it is good practice to run Kamada as a pre-processing phase to bring the graph layout into the neighbourhood of an energy minimum, and then use a method of choice to home in on the minimum. Since the CPU time used by Kamada never exceeded two seconds on this example, it adds only a negligible amount of delay if used as a pre-processing step.

It is diverting to compare the drawing generated by the layout algorithms to that which the data was derived from; this is done in Appendix 5.

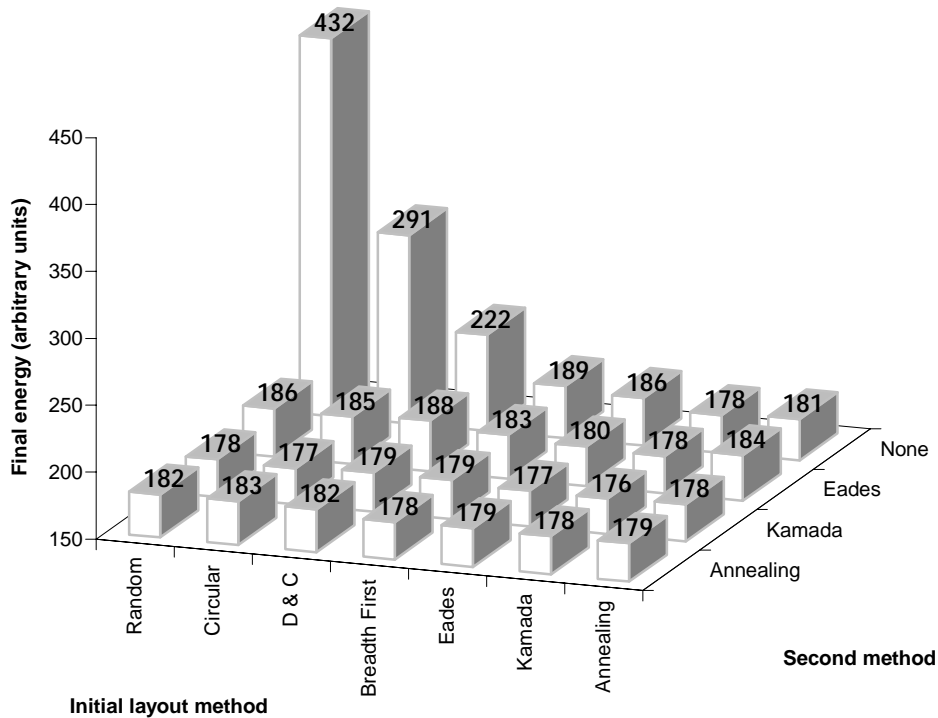


Figure 14: Effect of starting position for a moderate sized graph (tube map)

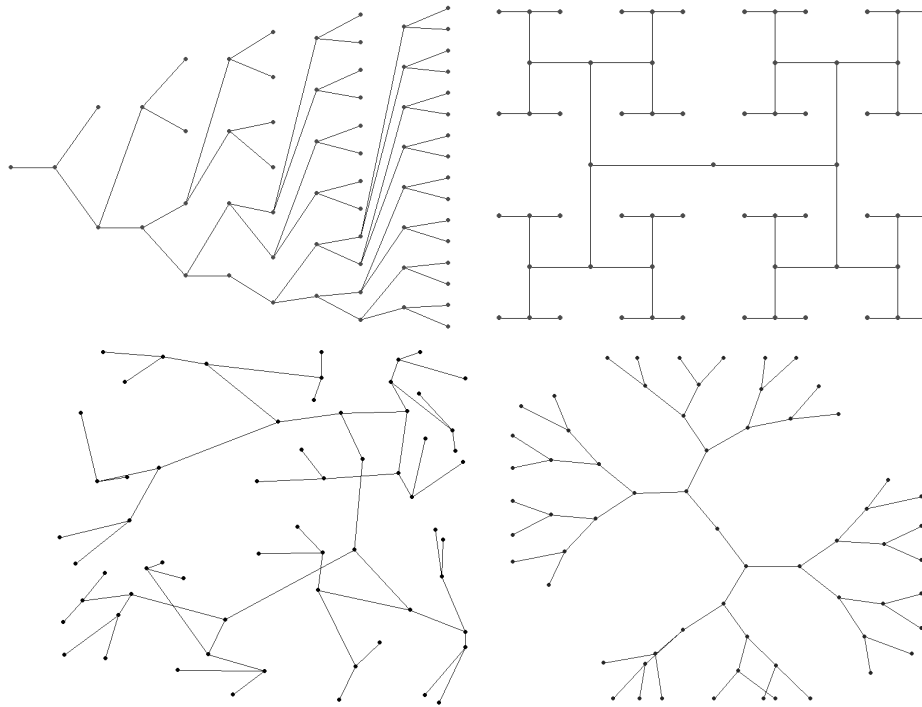


Figure 15: Binary tree laid out using (clockwise from top left) Breadth First; Divide-and-Conquer; Kamada; Annealing

4.3.3 Sparse and structured: the binary tree

The final example examined was a medium sized (63 nodes) binary tree; methods exist to lay out tree data in optimal fashion, so it would be interesting to see how well the force-directed methods fared compared to these. It turns out that the heuristic methods do a good job on this data (Figure 15), with the Breadth First and Divide and Conquer heuristics giving very clear and regular drawings of the graph. It is here we see the limit of the energy approach to scoring layouts: the Breadth First method is scored at 109, while Annealing, which produces an irregular and non-planar drawing is scored at 106. This is because the energy equations are based on an objective approximation of aesthetic criteria, which, while adequate for general graphs, fail to reward layouts which work well visually but do not follow their strict prescription of what a good drawing is. The scores for other combinations (Figure 16) are too close to make any conclusive observations about, other than that two applications of Kamada again achieves one of the best scores.

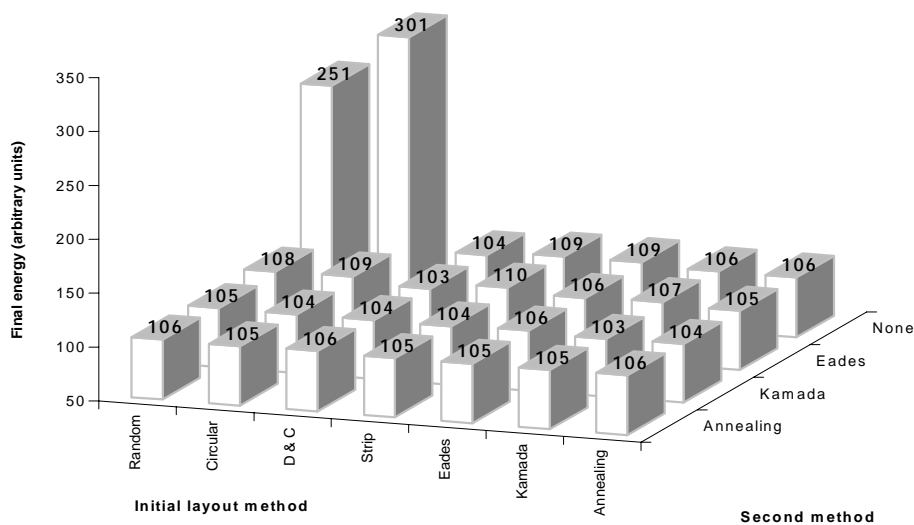


Figure 16: Effect of starting configuration for a sparse, structured graph (binary tree)

4.4 Working with a subset of the graph (2.3.3)

Again, this is a question which the system was designed to answer. Since Annealing is the only method which takes a punitive amount of time to complete, the effect of using local temperatures rather than global temperatures for nodes was investigated for Annealing. The social network used in 4.3.1 was laid out using Kamada, and the nodes which appeared misplaced were selected, and their temperature set to 1.0. The temperature of the remaining nodes was set to zero, then Annealing was run. As a comparison, Kamada and Annealing were run from the same starting configuration but considering all nodes.

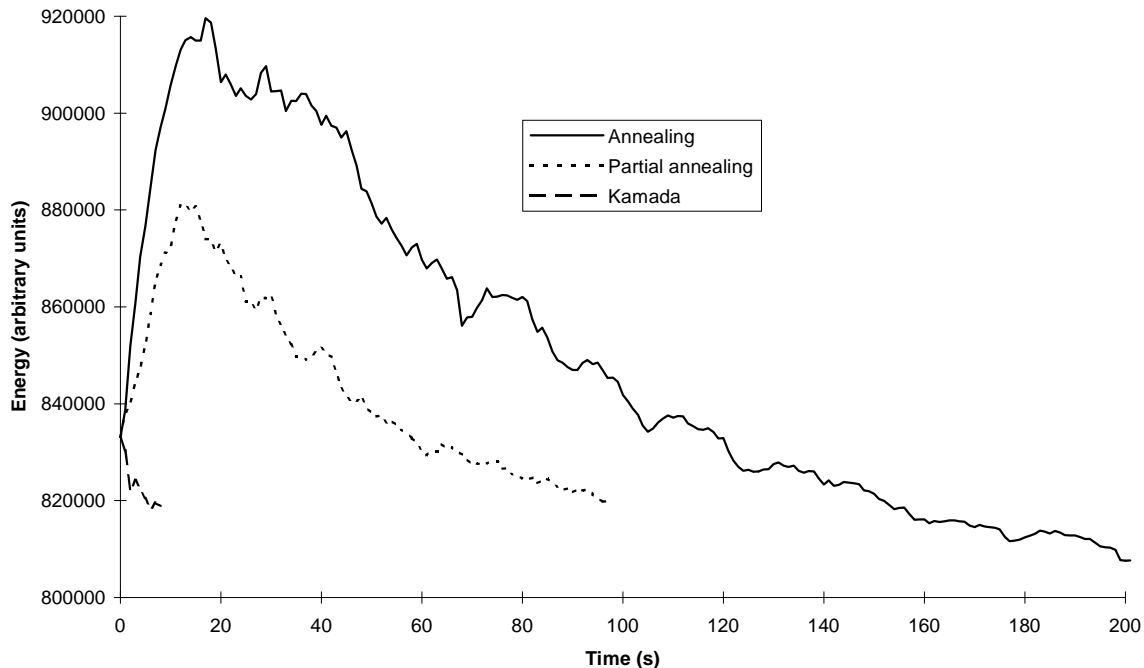


Figure 17: Effect of local versus global temperature

The results (Figure 17) are not very promising: although using local temperatures to control only those nodes of interest results in a decrease of global energy, and only takes about half as long, the decrease is not as great as that found by using Annealing globally, and not even as low as that found by running Kamada again, which completes in significantly less time. In conclusion, using local temperatures sits uncomfortably as a concept, requiring human judgement to pick which nodes to operate on, and achieves neither greatest energy minimisation nor greatest speed.

4.5 Final Observations

The focus of the evaluation has been in evaluating the performance of the different algorithms, and so to some extent the actual performance of the system that was implemented has been apparently overlooked. This is largely due to the implication that the ability to perform such an evaluation of the methods would not be possible without an adequate system within which to test them. The illustrations of the layouts found are actual screen shots of the system after experiments. The first section of this evaluation in particular (4.1), whilst phrased in terms of examining the output of the layout methods, also implicitly tests that they are correctly implemented within the system by seeing whether they produce the kind of results that we expect.

While we would not expect a program of this scale to be entirely free of bugs, we have seen that simple examples perform correctly, and that larger tests give good results — far better than could be expected from any unstructured process. The methods are such that any residual errors are not immediately obvious: correcting some minor bugs during implementation did not noticeably improve the results. Perhaps it would be better to cast this evaluation as a comparison of the algorithms *as implemented*, rather than of the algorithms themselves, since there was some scope for interpretation and variation from the descriptions given in the original papers. But however you wish to look at it, the system implemented three different iterative graph layout methods which were evaluated above, and found to work more than adequately in the cases tested.

5. Conclusions

A system which allowed graphs to be entered, laid out using heuristic methods and different force inspired methods or by the user was implemented successfully. The problem of how to test whether a layout was aesthetic was tackled by advancing a scoring method based on the criteria for a layout to be 'sound' (1.1.4 to 1.1.6), and this corresponded with what was judged to be the better arrangement for general graphs.

This energy approach allowed the performance of the three iterative methods to be compared objectively and quantitatively. Throughout the testing, simulated annealing achieved the best score, but took many times longer than the force-directed methods, and the margin of improvement was slim. The exception to this were special cases such as trees, for which methods handling these cases specifically can give optimal results. For a package designed to allow graph data to be arranged for output purposes, simulated annealing should be implemented along with a range of other approaches, but for an application where display of graph data is incidental to the purpose of the system, the designer would be better off implementing the faster and simpler method of Kamada and Kawai, perhaps performing two phases to improve on a single run solution.

There are many possible extensions to this work. The only extension suggested in the project proposal which was not implemented, to use a third dimension in the generation of a two-dimensional layout, would be an obvious candidate. Extending the algorithms to operate in a full three-dimensional environment would be worth investigating from a data visualisation perspective. The algorithms implemented are by no means the only conceivable force-directed methods, and for that matter, the same forces could have been formulated in a number of different ways (for example, inverse-square relations could equally have been formulated as reciprocal or exponentially decaying with distance). It would also be possible to apply the scoring of a layout by energy to drawings found by using other methods away from the force-directed paradigm, such as planarisation [BETT94].

Were I to redo the project with hindsight, there is little that I could usefully have done differently. The use of local temperatures rather than global turned out to have little impact, but this in itself is worthwhile result. If there was more room in this dissertation and more time, I think I could have usefully investigated the effect of the same methods on the individual components of the energy score (node density, edge length and edge-crossings), and filled in the few gaps, such as the facility to delete nodes and edges, which would turn what is currently a tool to investigate the performance of graph layout algorithms into a fully-featured graph layout application.

Bibliography

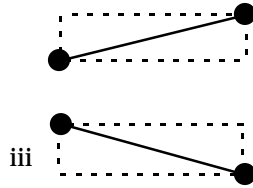
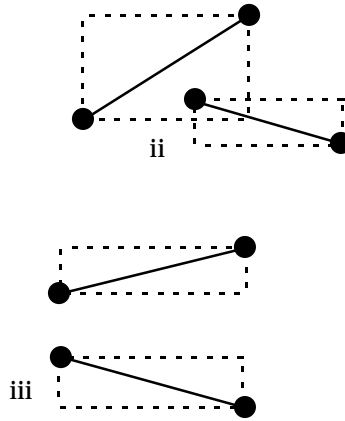
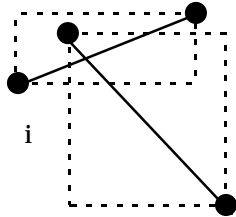
- [BETT94] Battista, Eades, Tamassia, Tollis
Algorithms for Drawing Graphs: an Annotated Bibliography
<ftp://wilma.cs.brown.edu/pub/papers/compgeo/gdbiblio.ps.Z> 1994
Good investigation into the various methods of drawing graphs and special cases (trees, DAGs, planar graphs) with approaches to their optimal solutions. Not brilliant on force and energy methods.
- [CLR90] Cormen, Leiserson, Rivest
Introduction to Algorithms
McGraw-Hill, 1990
Chapters 23-26: Graph Algorithms : alternative presentation of DFS, BFS, MST, all-pairs shortest paths, with more solid implementations.
- [CMS95] J Christensen, J Marks, S Sheiber
An Empirical Study of Algorithms for Point-Feature Label Placement
ACM Transactions on Graphics, July 1995
Vol 14, No 3, CLL
An introduction to the NP-Hard problem of taking an graph drawing, and positioning node labels to minimise crossings. Arc labelling is perhaps even worse (too much freedom!)
- [DH89] Davidson and Harel
Drawing graphs nicely
Technical Report CS 89-14 Weizmann Institute of Science, 1989
Revised: ACM Transactions on Graphics Vol 15 No. 4, Oct 1996
Outlines their approach to the general problem using simulated annealing. Also discusses other methods: Eades (1984) Spring Method - attraction and repulsion calculated, and translated into a (sudden, simultaneous, ie discrete) movement of nodes. Repeat till “stable”. Other approaches vary the attraction and repulsion forces, and use subtly different ways of going to the next configuration.
- [Ead84] Paul Eades,
A Heuristic For Graph Drawing
Congressus Numerantium vol 42, pp 149-160, 1984
The first application of force directed placement to graph drawing.
- [Eve79] Shimon Even
Graph Algorithms
Computer Science Press 1979
Chapters 7,8: Develops a theory of graph planarity (Kuratowski’s theorem) into two linear time algorithms for testing planarity (these do not lead directly to a method for drawing a planar graph with no crossings). Some observations on planarity: if a graph does not satisfy $E < 3V - 6$ then it is not planar.
- [Gar86] Martin Gardner
Knotted Doughnuts & other mathematical entertainments
W.H. Freeman, 1986
Chapter 11: Crossing Numbers. A popular mathematical introduction to the subject of crossing numbers. Rectilinear crossing number (drawn using straight lines) is \geq crossing number. $RC=C$ for complete graphs 1-7, 9, but higher for K_8 . $K_{3,3}$ has crossing number 1 (“utilities puzzle”, Dudeney : three houses, linked to electricity, gas, water). Finding C and RC is NP-C (Garey & Johnson, 1983).
- [GD98] <http://gd98.cs.mcgill.ca/>
Details of the annual graph drawing symposium and its associated graph drawing competition.

- [GJ79] Michael Garey, David Johnson
Computers & Intractability: A guide to the theory of NP-Completeness
W.H. Freeman, 1979
p286 Treated as open problems: can a graph G be embedded on a surface of genus K such that no two edges cross one another? and Does G have a crossing number K or less (embedding in the plane)?
Noted that $K=0$ means is G planar, solved in polynomial time (Hopcroft & Tarjan, 1974), likewise for $K=1$ (Filotti, 1978). p339, noted that CROSSING NUMBER settled as NP-C (Garey & Johnson).
- [Gra97] <http://www.uni-passau.de/~himsolt/Graphlet>
An experimental system for graph drawing (applies various force methods), runs under Windows, Macintosh, X Windows, using the TCL/TK extensions. Quite fun to play with, but seems to have UI problems under 95 (ie when I use it).
- [HR90] Hartsfield and Ringel
Pearls in Graph Theory
Academic Press, 1990
Chapter 8: Drawings of Graphs. Examines Planarity, Crossing Number, Multigraphs and maps.
Chapter 9, measurements of closeness to planarity, definitions of a simple drawing.
- [HS98] D Harel and M Sanders
An algorithm for straight line drawing of planar graphs
Algorithmica vol. 20 no. 2, February 1998
- [JDK97] Graph Drawing Demonstration
Java Development Kit v1.1
<http://www.cl.cam.ac.uk/javadoc/jdk1.1/demo/GraphLayout>
Simple force directed method implemented in Java; four examples provided, quite fun to play with.
- [KK89] Kamada and Kawai
An Algorithm for drawing general undirected graphs
Information Processing Letters Vol 31, Number 1, 1989
A fully explicit description of their method, which uses point-by-point solution of Force equations (so a global minimum is not guaranteed, but progress from a configuration is entirely deterministic). The force is composed of the square of the difference between the current separation and the scaled ideal (shortest path) separation - this is simple enough to be solved to give rules for particle movement, and the highest energy point is moved iteratively.
- [Pal85] Edgar M Palmer
Graphical Evolution
John Wiley and Sons, 1985
Presents a number of probabilistic models for evolving "random" graphs, and analyses them for probability of exhibiting certain features.
- [Sed88] Robert Sedgwick
Algorithms
Addison Wesley, 1988
Chapters 29-31: Graph Algorithms, and Chapter 24, 27: Geometric Methods.
- [Tam98] Roberto Tamassia
Links to information on graph drawing
<http://www.cs.brown.edu/people/rt/gd.html>
- [TC97] Roberto Tamassia, Isabel Cruz
A graph drawing Tutorial
<http://www.cs.brown.edu/people/rt/papers/gd-tutorial/gd-constraints.pdf>
p66-72 Introduction to force based methods. Presented in the form of OHP sheets.

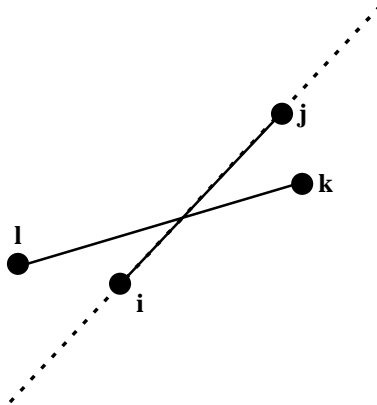
Appendix 1 — Geometric Algorithms

The non-trivial geometric algorithms required by the system involve edges. We need to be able to test whether a pair of edges cross, and if not, find how far apart they are (some of this material is based on [Sed88] and [CLR90]).

Edge Crossing



Firstly, we implement a “fast reject” case, which uses the fact that if a pair of edges cross, then their bounding rectangles intersect, and hence if the rectangles do not intersect then the edges cannot cross. This has the additional benefit of eliminating a case which proves tricky in the general case, if two edges are collinear but do not cross. The three cases are illustrated *left*: i) boxes overlap, edges cross ii) boxes overlap but edges do not cross iii) boxes do not overlap, edges do not cross.



Once it has been determined that the edges might intersect, we test for intersection by use of the fact that if the edges intersect then both points of one edge must lie on opposite sides of the line formed by extending the other edge (and vice-versa).

This is satisfied if $0^\circ < \mathbf{ijk} < 180^\circ$ and $180^\circ < \mathbf{ijl} < 360^\circ$

or if $180^\circ < \mathbf{ijk} < 360^\circ$ and $0^\circ < \mathbf{ijl} < 180^\circ$ (measured clockwise)

We can find these conveniently using a dot-product test if we

rotate the line vector 90° , by using the transformation matrix $T \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$

Then our criteria can be written

$\mathbf{T(j - i) \cdot (k - i) \leq 0}$ and $\mathbf{T(j - i) \cdot (l - i) \geq 0}$

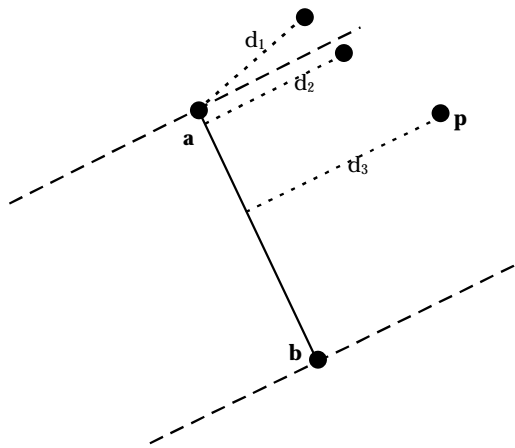
or $\mathbf{T(j - i) \cdot (k - i) \geq 0}$ and $\mathbf{T(j - i) \cdot (l - i) \leq 0}$

which can be simplified to

$$(\mathbf{T(j - i) \cdot (k - i)}) ((\mathbf{T(j - i) \cdot (j - i)}) \leq 0$$

Perpendicular distance between edges

If two edges do not cross, then we need to be able to find the perpendicular distance between them. This is found in terms of the perpendicular distance between a vertex from one edge and the other edge.



The perpendicular distance between a point and an edge is defined as the perpendicular distance between the point and the infinite extension of the edge if the point falls between the two parallel lines perpendicular to the edge passing through its vertices (as in d_2 and d_3 , *left*), or the distance between the point and the nearer of the edge's vertices (as in d_1 , *left*).

To test whether a point p lies inside the dotted lines, we require that the angles \mathbf{pab} and \mathbf{pba} are acute. The conditions can be written

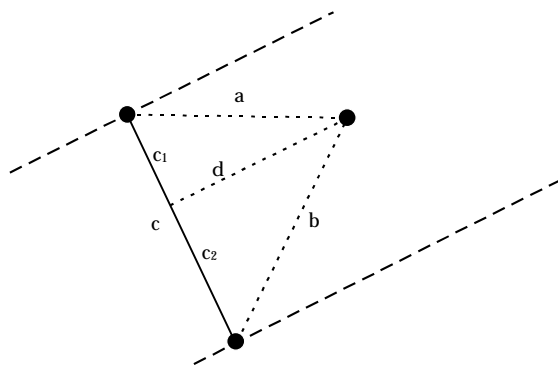
i) $0^\circ < \mathbf{pab} < 90^\circ$ and $0^\circ < \mathbf{abp} < 90^\circ$ (measured clockwise)

or ii) $270^\circ < \mathbf{pab} < 360^\circ$ and $270^\circ < \mathbf{abp} < 360^\circ$

The conditions on \mathbf{pab} can be written $(\mathbf{p} - \mathbf{a}) \cdot (\mathbf{b} - \mathbf{a}) > 0$

and the conditions on \mathbf{abp} are expressed $(\mathbf{p} - \mathbf{b}) \cdot (\mathbf{b} - \mathbf{a}) < 0$

Hence our test simplifies to "p lies within the lines if $((\mathbf{p} - \mathbf{a}) \cdot (\mathbf{b} - \mathbf{a}))((\mathbf{p} - \mathbf{b}) \cdot (\mathbf{b} - \mathbf{a})) < 0$ "



If the p lies outside the lines, then the perpendicular distance is $\text{Min}(|\mathbf{p} - \mathbf{a}|, |\mathbf{p} - \mathbf{b}|)$

If it lies inside the lines, then the distance d is calculated using simple geometry:

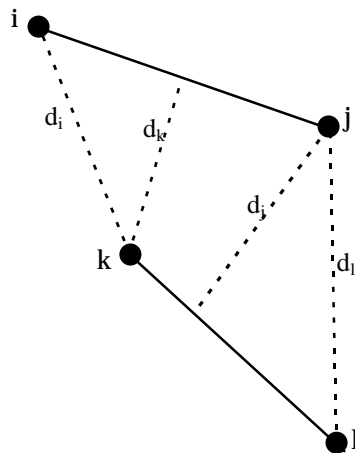
$$c_1^2 + d^2 = a^2$$

$$c_2^2 + d^2 = b^2$$

$$c_1 + c_2 = c$$

These equations can be solved to give

$$2d^2 = a^2 + b^2 - \frac{1}{2}c^2 - \frac{(a^2 - b^2)^2}{2c^2}$$



Now that we have a notion of the perpendicular distance between an edge and a point, we can find the shortest distance between a pair of edges. This is defined as the shortest of the four perpendicular distances between a point from one edge to the other edge. So, using the terminology of the diagram *left* we find

$$d_i = \text{Mindist}(i, \text{Edge}(k,l))$$

$$d_j = \text{Mindist}(j, \text{Edge}(k,l))$$

$$d_k = \text{Mindist}(k, \text{Edge}(i,j))$$

$$d_l = \text{Mindist}(l, \text{Edge}(i,j))$$

And so the perpendicular distance between the edges is

$$\text{Min}(d_i, d_j, d_k, d_l).$$

Appendix 2 — Centre Layout

The attempt to create a divide-and-conquer algorithm to layout a graph heuristically resulted in the following algorithm. Firstly, the definition of the “centre” of a graph is needed.

The centre of a sub-graph $V' \subseteq V$ is defined as follows:

```
Create the graph  $G' = (V', E')$  where  $E' = \{ (v,w) \mid (v,w) \in E \wedge v \in V' \wedge w \in V' \}$ 
construct the all-pairs shortest path matrix,  $M$  for the graph  $G'$ 
Find the vector  $L$ , such that  $L(v \in V') = \text{Max}(M(v, w))$ 
Find  $v$  such that  $L(v) = \text{Min}(L(w))$  :  $v$  is the centre of the subgraph.
```

We can now develop an algorithm:

```
Start with the whole graph and the whole drawing area
Find the centre of the current (sub-)graph
Place the centre in the centre of the drawing area
Divide the vertices into two sets, and divide the area into two
Recursively place one set of vertices in one half of the area, and the other in the
other.
```

To divide the vertices into two sets, we perform two alternating breadth first searches. It operates as follows:

```
Initialise two lists with the current centre node
Then alternate:
    Take the head of the list, consider nodes adjacent to it
    If a node is  $\in V'$  and it has not been marked then
        mark the node; add it to the list; break
```

For such a division to be effective we would like the centre to have at least two edges on it. Fortunately this is easy to prove. Assume that we have a connected graph (which we would expect from the way in which the breadth first search is performed: we will not have any nodes in our graphs that could not be reached from the starting node). Hence if our centre does not have at least two edges incident on it then it will have exactly one. We can label each node v with the value $L(v)$. For w to be the centre, we must have $L(w) \leq L(x)$, where x is the node adjacent to w . But all paths which go from w must go through x , so every path from w is one step longer than that from x , and hence $L(w) = L(x) + 1$ ie $L(w) > L(x)$; this is a contradiction, and so we conclude that the centre of any non-trivial (>2 nodes) graph has at least two edges incident to it.

Appendix 3 — Code Samples

The system produced consisted of 16 modules written in Modula-3, with their corresponding interface definitions, totalling around three and a half thousand lines of code, along with two .fv files which defined the user interface.

The project briefing booklet asserts that “assessors like to see some sample code”. In accordance with this suggestion, I include the “Kamada.m3” module to satisfy this. The tranche below implements the method of Kamada and Kawai in a force directed fashion:

```

MODULE Kamada;
IMPORT Graph, Energy, Node, R2, RefList,
Tick, IO;

PROCEDURE ApplyForces(self : Energy.T;
max : REAL) : BOOLEAN =
  VAR np, npc : RefList.T;
      v0, v1 : Node.T;
      vector, deltav : R2.T;
      dv : REF ARRAY OF R2.T;
      significant : BOOLEAN;
      factor : REAL;
  BEGIN
    (* just do a single iteration here *)
    (* F = 1 / dij^2 *)
    np:=self.vertices;
    dv:=NEW(REF ARRAY OF R2.T,
self.noderecs+1);
    FOR i:=0 TO self.noderecs DO
      dv[i]:=R2.Origin;
    END;

    WHILE np#NIL DO
      v0:=np.head;
      npc:=np.tail;
      WHILE npc#NIL DO
        v1:=npc.head;
        vector:=R2.Sub(v0.pos, v1.pos);
        IF self.adjacency[v0.id, v1.id]=0
THEN factor:=max
        ELSE
factor:=FLOAT(self.adjacency[v0.id,
v1.id]);
          END;
          (* factor is the graph theoretic
distance between points *)
          deltav:=R2.Scale((self.ce /
(factor*factor) ) *
(R2.Length(vector) - factor *
self.l),
R2.Direction(vector));
          dv[v0.id]:=R2.Sub(dv[v0.id],
deltav);
          dv[v1.id]:=R2.Add(dv[v1.id],
deltav); (* update the velocities *)

          npc:=npc.tail;
        END;
      np:=np.tail;
    END;

    np:=self.vertices;
    significant:=FALSE;
    WHILE np#NIL DO
      v0:=np.head;
      v0.velocity:=R2.Scale(v0.T,
R2.Add(v0.velocity, dv[v0.id]));
      IF R2.SumSq(v0.velocity)>4.0 THEN
(* apply the velocities *)
        v0.moveto(R2.Add(v0.pos,
v0.velocity));
          significant:=TRUE;
        END;
      np:=np.tail;
    END;
  END ApplyForces;

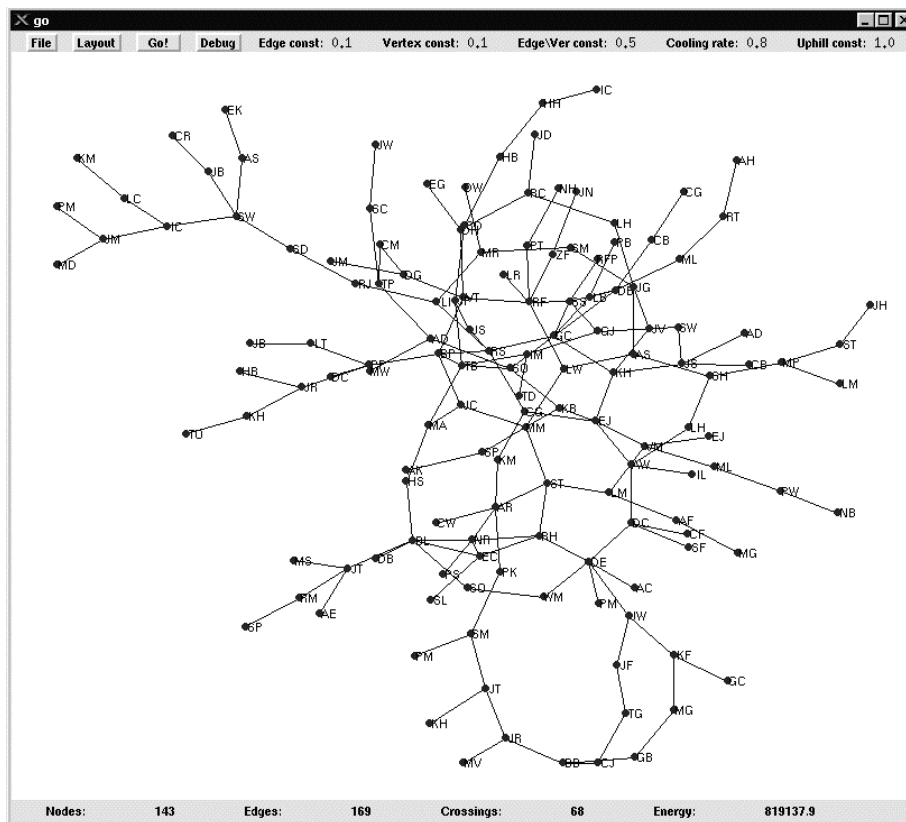
PROCEDURE Apply(g : Graph.T) =
  VAR
    np : RefList.T;
    v0 : Node.T;
    self : Energy.T;
    n, max :=0;
    hottest:=1.0;
    start, now, test : Tick.T;
  BEGIN
    self:=NARROW(g, Energy.T);
    self.calcpaths();
    max:=0;
    FOR i:=1 TO self.noderecs DO
      FOR j:=i TO self.noderecs DO
        IF g.adjacency[i,j]>max THEN
          max:=g.adjacency[i,j];
        END;
      END;
    END;
    (* find the longest path within the
graph *)
    WHILE np#NIL DO
      v0:=np.head;
      v0.velocity:=R2.Origin;
      np:=np.tail;
    END; (* zero the velocities *)

    TRY
      test:=Tick.FromSeconds(self.period);
    EXCEPT Tick.Overflow =>
      IO.PutErr("Internal timer error");
    END;
    start:=Tick.Now();
    REPEAT
      INC(n);
      now:=Tick.Now();
      WHILE (now-start)>test DO
        self.newreading();
        start:=start+test;
      END;
      IF (n MOD 5) = 0 AND (n>20) THEN
        self.updateVBT();
        IF (n MOD 25) = 0 THEN
          hottest:=self.cool();
          END; (* cool the system *)
        END;
      UNTIL (hottest < 0.1) OR NOT
ApplyForces(g, FLOAT(max)); (* actually do
it *)
    END Apply;

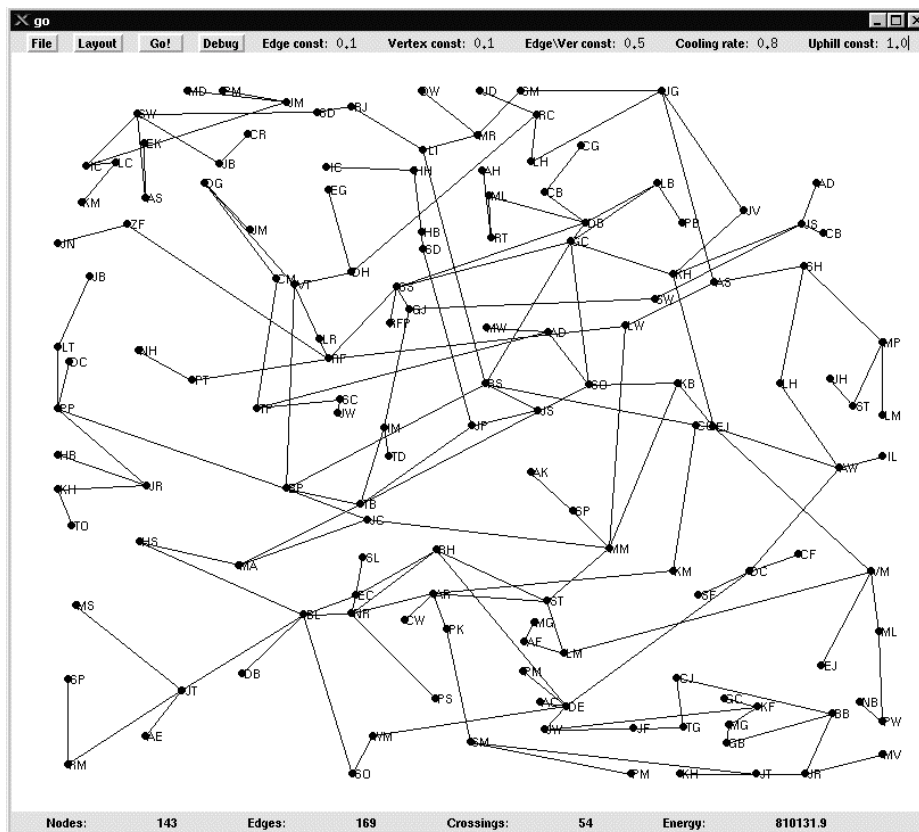
  BEGIN
  END Kamada.

```

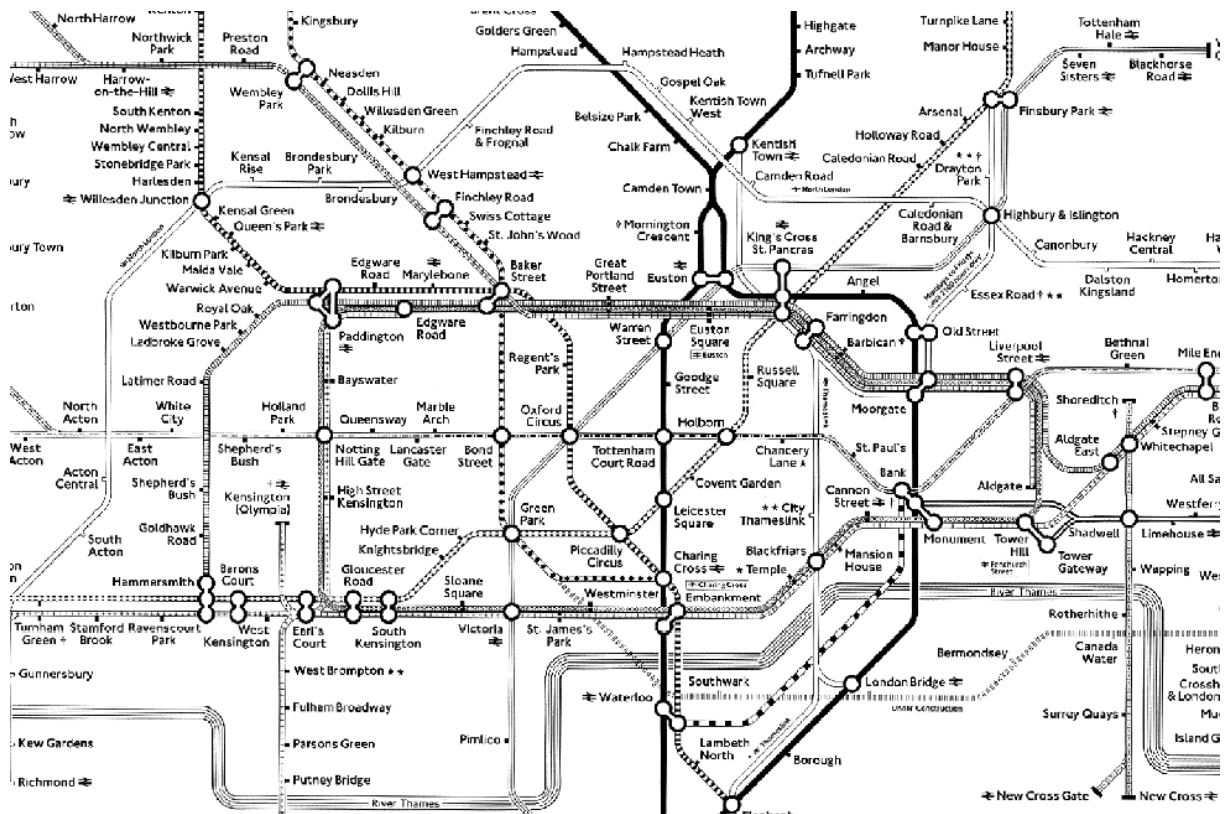
Appendix 4 — The Social Network



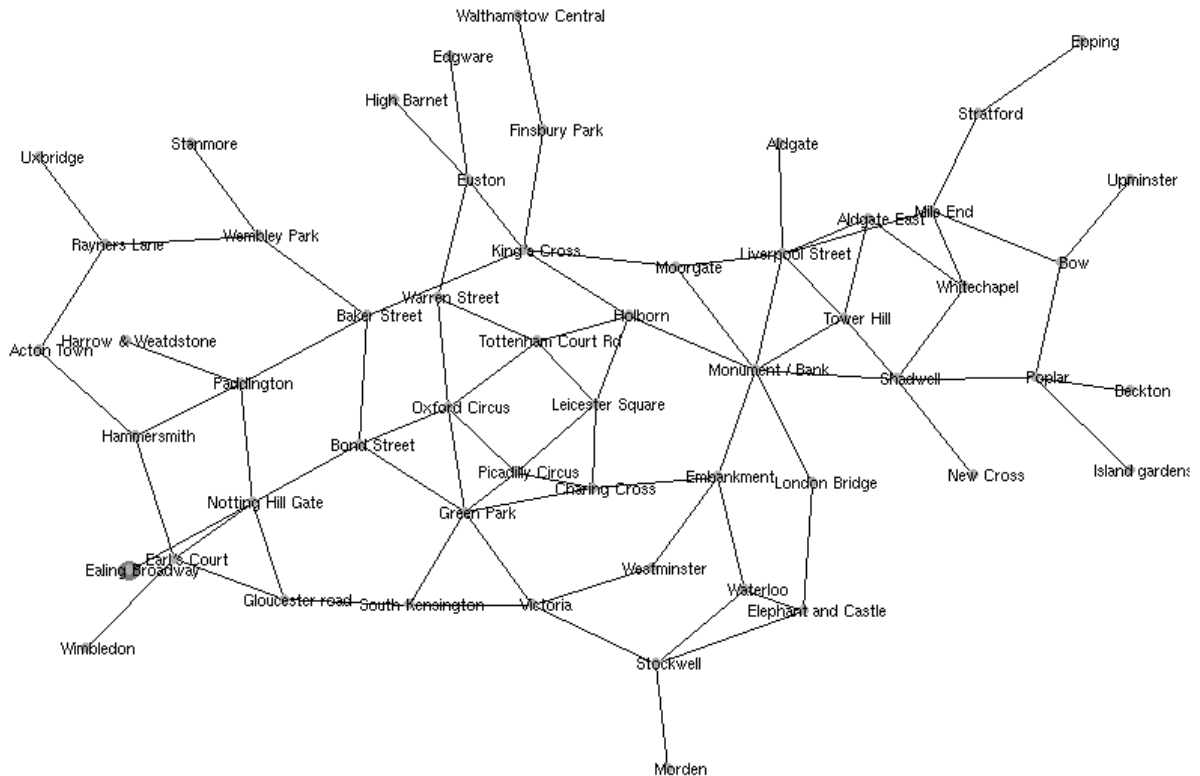
The above layout is the result of performing two iterations of Kamada on the network; below is the result of performing Annealing on the above layout. Annealing gives a less regular drawing, but makes better use of the area and has fewer edge crossings, and hence is awarded the better score.



Appendix 5 — The London Underground



There is a quite good correlation between the two drawings (after the lower diagram has been adjusted for orientation). The central area bounded by Notting Hill Gate, Victoria, London Bridge and King's Cross is especially close to the traditional drawing; perhaps this is due more to the fact that this area of the map has stations placed evenly in a planar fashion, which the algorithm rediscovers.



London Underground Connectivity Information, laid out using the method of Kamada and Kawai

Graham Cormode
King's
grc21

Computer Science Tripos Part II — Project Proposal

Drawing graphs using physical heuristics

21 October 1997

Originator John Naylor

Special Resources None

Supervisor Calum Grant

Signature

Director of Studies Ken Moody

Signature

Overseers Neil Dodgson and Frank King

Signatures

1 Background and general description

The problem of graph drawing - transforming a list of nodes and edges into a graphic representation of the information - is an area of continuing research (with an annual symposium). It is perhaps no surprise that many of the problems under the heading of graph drawing are NP-complete or NP-hard. Yet techniques have been developed which make good attempts at displaying graphs, and sub-optimal solutions (for which more lines cross than the theoretical minimum, say) are still visually acceptable.

One approach is somewhat removed from the algorithmic methods usually taught at degree level. This seeks to find a physical analogy, modelling nodes as magnets which repel each other, connected to non-magnetic springs representing the edges. A step-by-step calculation and application of the forces results in a steady-state solution, which translates into a clear graph representation. If, instead of considering the forces, one analyses the energy of the system and moves nodes so as to reduce the energy, different results can be observed. Allowing a small chance that the energy of the system can increase is known as simulated annealing.

The choice of equations to calculate the forces and energies is vital for any such system. Node density, variance of edge lengths, number of edge crossings, all these affect the "prettiness" of the graph and so are incorporated into models in various ways by different researchers. The core of the project will be to create a system capable of accepting a specified set of force or energy equations, and applying them to a graph to compare the convergence and aesthetics of different models.

There are then a number of extensions and variations which I wish to examine the effect of. Force directed methods are usually carried out in two dimensions, since a 2D drawing is required. I wish to test whether allowing freedom in the third dimension, but reducing this freedom with the iterations of the algorithms, affects the results. These methods use a random starting configuration to improve on; would a more ordered start improve things? Lastly, once a configuration has "set", the user can judge which areas of the graph are well drawn; it would be convenient if they could selectively "reanimate" those areas which are not, without manually moving single nodes into better positions. Other extensions may suggest themselves during research, and certain of those mentioned may reveal themselves to be impractical; it is in the nature of such a project that I will have to be flexible to such discoveries.

2 Note of resources required

No special resources will be required: it is my intention that the work be carried out on Thor, and neither the code nor sample graph descriptions should take up significant amounts of space.

3 Starting Point

Part of the preparatory work will be to look at any existing libraries or systems which deal with graph data and algorithms, and any use made of these will be documented later, though from the literature and my analysis of the problem it would be quite feasible to work almost entirely from scratch. I shall also have to formalise my use of RCS and back-up methods.

4 Plan of work

This is split into ten approximately fortnight-long periods:

24 October to 5 November

Research into the problem — chasing up references. Initial planning of which extensions to concentrate on, with reference to any areas suggested in the material. Also prepare a back-up system & RCS library; look into other areas (such as learning PostScript or other relevant languages). Deliverables: a brief survey of the literature (should be useful when writing up) & back-up scheme.

5 November to 19 November

Detailed planning of project implementation. Choose appropriate algorithms for basic tasks (detecting line crossings, finding all-pairs shortest paths). Ensure system will be flexible enough to accept a number of different physical models, as well as be easy to link in the extensions described above. Deliverables: a document detailing the design and explaining the reasoning behind the design decisions (parts of which will be required in the write-up).

19 November to 1 December

Skeletal implementation of the system, including input / output (save current layout: this will be needed to compare convergence times. Postscript output?), user interface, drawing and scaling of processed results, random plot of data (to test drawing & needed as a starting point for some methods). Deliverables: tested system which implements this, notes on problems / late decisions.

1 December to 14 January

Allow user manipulation of nodes. Implement force method, allowing equations for forces (on each node from other nodes, on each node from edges) to be specified based on a number of variables. Deliverables: a system which meets this specification, or explanation of any problems, and a start of the progress report.

14 January to 28 January

Implementation of simulated annealing (consider the monotonic energy approach as a special case). Allow customisation of energy equations & temperature reduction in same way as with forces. Include information on speed of convergence. Test. Result: core system & progress report.

28 January to 11 February

Extension 1: Extend both methods into 3D with decreasing depth over time. Allow selective reheating of mangled areas. Some slack in case of problems / delays, else start next stage early.

11 February to 25 February

Extension 2: Implement various fast first layout heuristics (to be used in place of random layout) and investigate their relative effect on convergence. Start to draw work together into a single report. Deliverables: finished, stable system.

25 February to 11 March

Begin to write up to the standard format. Carry out structured investigations using the system to answer the questions posed in this project proposal. Deliverables: near final drafts of the Introduction & Preparation parts of the dissertation (though these should have long been in progress).

11 March to 29 April

Complete the evaluation. Write up details of the evaluation and implementation, illustrating with appropriate samples and examples. Write a conclusion. Deliverables: a first draft of the dissertation.

29 April to 13 May

Revise the dissertation draft, and pass it on to supervisor. Leave plenty of time for printing and binding. Deliverables: finished dissertation, as early as possible.

Fall back position

If the extensions all turn out to be impractical, or the core project requires far more effort than estimated, a “bare minimum” would be the implementation of the core, and a comparison of the performance of the two methods would be an adequate, if not especially rewarding, project.