Using Spring Algorithms to Remove Node Overlapping

Wanchun Li, Peter Eades and Nikola Nikolov

National ICT Australia Ltd,
University of Sydney
NSW, 2006, Australia
Email: `lpaul@it.usyd.edu.au, {peter.eades,nikola.nikolov}@nicta.com.au`

Abstract

Cluttered drawings of graphs cannot effectively convey the information of graphs. Two issues might cause node overlapping when one draws a picture of a graph. The first issue occurs when applying a layout algorithm for an abstract graph to a practical application in which nodes are labeled. The second is the changing of a node's size in a dynamic drawing system. This paper presents two algorithms, *DNLS* and *ODNLS*, for removing the two kinds of overlapping. The algorithms are based on the well-known spring embedder model. The outputs of the algorithms provide the features of spring algorithms. Experiments are carried out to compare *DNLS* and *ODNLS* to the Force Scan(FS) algorithm and its variants. The results demonstrate the advantages of *DNLS* and *ODNLS* in terms of some aesthetic criteria.

Keywords: graph drawing, node overlapping, spring algorithm

1 Introduction

Graphs are abstract structures that are used to model relational information. Good pictures of a graph can convey the information rapidly and efficiently. Many successful algorithms have been introduced to automatically generate the layout of a graph(Battista et al. 1998). Most graph layout algorithms treat nodes as points. Such an algorithm is a layout creation algorithm(Eades et al. 1995). A layout generated by a layout creation algorithm might have overlapping nodes when the layout is applied to practical applications, in which nodes are labeled and are drawn as boxes or other symbols. Overlapping might also occur in a dynamic graph drawing system. When a user opens some node for more details, the enlarged node might overlap its neighbour nodes.

Cluttered drawings of graphs cannot effectively convey the information of the graphs. Eades et al. first address the issue of removing node overlapping(Eades et al. 1995). Before giving any specific algorithm, they introduce an important concept, the *mental map*. Three mathematical models are constructed to define the mental map. One of the models is *orthogonal ordering*. Orthogonal ordering models the most basic mental map, which preserves *up*, *down*, *left*, and *right*. Suppose a node $v \in$V has the positions (x'_v,y'_v) and (x_v,y_v) in the drawings of Γ' and Γ respectively. Then Γ' preserves the mental map of Γ if for $\forall u \in$V,

- $x'_v < x'_u \Leftrightarrow x_v < x_u$, and
- $y'_v < y'_u \Leftrightarrow y_v < y_u$, and
- $x'_v = x'_u \Leftrightarrow x_v = x_u$, and
- $y'_v = y'_u \Leftrightarrow y_v = y_u$

They then propose three requirements for a layout adjustment:

1. The adjusted drawing should be compact.
2. Node images should be disjoint.
3. The mental map should be preserved during interaction.

One simple solution is to uniformly scale all edges. The overlapping nodes can be disjoint, and the mental map is preserved. However, the generated layout is usually not compact.

Eades et al. present the *Force Scan(FS)* algorithm for adjusting a cluttered layout (Eades et al. 1995). The algorithm scans in both horizontal and vertical direction to find overlapping nodes, and move the overlapping nodes to new positions. The new positions of overlapping nodes are determined by the *desirable distance*. The *desirable distance* is the shortest distance to move two overlapping nodes apart along the line that connect their centers. The algorithm iterates the scanning and moving process until there is no further overlapping. They treat all nodes as rectangle. If the image of a node is not a rectangle, the bounding box will be considered.

Hayashi et al. prove that it is a NP-hard to transform a given layout of a graph with overlapping rectangular nodes into a minimum-area layout without node overlapping which preserves the orthogonal order(Hayashi et al. 1998). They also propose a heuristic FS' that is an improvement of FS. Given a node v_i, a set of nodes overlapped $\{v_1, v_2, ..., v_n\}$, and the *desirable distances* $\{d_1, d_2, ..., d_n\}$, FS moves in one iteration all the overlapping nodes by the distance $max(d_i)$, $1 \leq$ i $\leq$n, while FS$'$ moves $min(d_i)$. A mathematical proof demonstrates that FS$'$ can generate a smaller area of layout than FS. Huang and Lai introduces another version of FS, the *Force-Transfer*(*FT*)(Huang, & Lai 2003). Instead of moving nodes along the line across the centers of two overlapping nodes, FT moves the overlapped nodes either horizontally or vertically. Given a pair of overlapping nodes, d_h is the distance to horizontally move the nodes to remove the overlapping, and d_v is the distance in the vertical direction. If $d_v < d_h$, the overlapped node is moved vertically, otherwise horizontally. Empirical experiments show that the output of FT has a smaller area than that of FS. All of FS, FS$'$ and FT can preserve the *orthogonal order.*

Lyons et al. propose different requirements of adjustment(Lyons et al. 1998). There are three criteria with different priorities. The first is to keep the adjusted layout in a window with the same size as the window of the cluttered layout. The second is more even distribution of nodes. Their approach weakens the restriction of similarity. Instead of preserving the exact ordering of two layouts, the approach tries to minimise the difference between them. In other words, it tries to make the two layouts "not very different". This is the third criterion, and its priority is the lowest. They present two algorithms, namely, *VDCB* and *GeoForce*. VDCB is based on Voronoi diagram, and GeoForce is an application of the force-directed model. Empirical experiments show that both the algorithms satisfy the goals of even distribution and similarity in corresponding priorities. It is not clear how to keep the size of a layout if there is not enough space for a dense graph or if the labelled node sizes are too big to fit in the fixed area. Gansner and North propose an algorithm inspired by VDCB (Gansner, & North 1998). The algorithm removes the restriction of the fixed size of the window, which is the first goal of VDCB. The algorithm first generates the Voronoi diagram using the node positions in a cluttered layout as sites. Then it moves each node to the centroid of its Voronoi cell. It iterates this process until there is no longer any overlapping. Experiments show that the algorithm has positive results in terms of the area of the drawings.

Marriott et al. view removing overlapping as a constrained optimisation problem(Marriott et al. 1995). Four algorithms are designed based on quadratic programming. Experiments show that it takes the algorithms less time to remove overlapping than FS needs, but more time to find overlapping than FS. They state that the layout generated by the algorithms is better than the output of FS, but no measurement for evaluation is mentioned. This algorithm does not guarantee to preserve the mental map.

All mentioned solutions are post-processing approaches, and are called *layout adjustment algorithms*. Generally, there are three steps to produce an uncluttered layout using these approaches. First, an *initial layout* is generated by a layout creation algorithm. Second, labels of nodes are added to the picture. Finally, the adjustment algorithm cleans the layout if there is overlapping.

Another general approach is to integrate the layout adjustment process and the layout creation process. Force-directed model is used in such methods. Kamps, Kleinz and Read add to the classical spring embedder model (Eades 1984) a "node-node repulsion force" that minimises the overlapping area of two nodes(Kamps, Kleinz & Read 1995). Wang and Miyamoto increase the repulsive force but cancels the attractive force if a pair of nodes are overlapping each other(Wang & Miyamoto 1995). (Harel &Koren 2002) calculates repulsive force using a variant value l_b. l_b is the distance between the boundaries between a pair of nodes. If a pair of nodes are overlapped, l_b is a small constant vale ϵ. The output of such algorithms has the advantages of force-directed graph drawing algorithm, eg. display of symmetry and uniform edge length. A common problem of these algorithms is that they do not guarantee to completely separate all overlapping nodes. Note that there is no need to preserve the mental map, because the process of removing the potential overlapping is merged into the process of layout creation. In other words, there is no "initial layout" at all. However, the lack of a mechanism for preserving the mental map makes these algorithms unsuitable for dynamic drawing systems.

In this paper, we present two algorithms for removing overlapping nodes. One is designed for static drawings of graph; the other is for dynamic drawings of graphs. The two algorithms are based on the spring embedder model. In the classical spring embedder models, the repulsive and attractive force are calculated by taking two distances into account. One is the current distance of two abstract nodes d; the other is the expected balance distance, *natural length* k, between nodes. In such models, k is a constant value for all nodes. The proposed algorithms calculate the forces using a *dynamic natural length* instead of a fixed natural length. The dynamic natural length k' takes into account the sizes of nodes. k' varies and is dynamically calculated according to the pair of nodes involved.

The first algorithm is called *Dynamic Natural Length Spring(DNLS)*. DNLS is a layout creation algorithm which can generate uncluttered layout. It is designed for static graph drawing and does not consider preserving mental map. The other algorithm is based on DNLS but has the ability to preserve the mental map. We name this algorithm *Orthogonal Dynamic Natural Length Spring(ODNLS)*. ODNLS is designed for dynamic graph drawing.

DNLS integrates the layout creation process and the adjustment process, while ODNLS processes a layout generated by another spring algorithm. The uncluttered layout generated by both algorithms has the features of a spring algorithm: display symmetry and uniform edge length.

In this paper we consider nodes with rectangular shape. This is commonly accepted for representation of labeled graph nodes. Our results are also applicable for non-rectangular node shapes if we consider the corresponding rectangles that enclose the node shapes.

Empirical experiments are carried out to compare DNLS and ODNLS with FS and its variants, FS$'$ and FT. The results demonstrate that DNLS/ODNLS improves on the FS and its variants in terms of some aesthetic criteria. The measurements of evaluation are such aesthetic criteria as *area, L1 metric length, aspect ratio, and edge length ratio*. The effectiveness of removing overlapping and running time are also tested.

The rest of the paper is organised as follows. In section 2, we introduce some preliminaries. In section 3, we present our algorithms. In section 4, we describe our experiments and give an analysis of experiments results. In section 5, we conclude and pose some future work.

2 Graphs and Drawings of Graph

2.1 Graphs

Suppose that $G = (V, E)$ is a graph. V is a finite set of nodes, and E is a finite set of edges. Each edge is a pair of vertices of G. Nodes represent entities, and edges represent the relationship between entities. If the pair of nodes of an edge is unordered, G is an undirected graph, otherwise a directed graph. A node v_i is said to be adjacent to a vertex v_i if v_iv_j is an edge of G. If any two nodes are joined by a path, G is a connected graph. A tree T is a connected graph without cycles.

2.2 Drawings of Graphs

A drawing of a graph is a representation of the graph in the Euclidean space. The drawing of an abstract graph usually represents a node as a point and an edge as a curve. Sometimes, interaction with a drawing of a graph is needed for exploring more details or analyzing the relationships between entities. The visual elements of the drawing will change in response to an interaction: some new nodes and edges might be displayed, some nodes might change their size and position, and some nodes and edges might disappear. Such drawings are called *dynamic graph drawing*s.

Some drawings are better than others in conveying the information of a graph. A drawing is sometimes measured by how well they meet the aesthetic criteria:

- minimisation of *crossings*
- minimisation of *area*
- minimisation of *L1 metric length*(see below for details)
- maximisation of *symmetry*
- minimisation of *total edge length*, uniform *edge length*
- *aspect ratio* close to specified value
- minimisation of *bends*
- maximisation of smallest *angular resolution*

Suppose Γ is a drawing of a graph G in a 2D Euclidean space with height h and width w. *L1 metric length* is equal to *max(h, w)*. Aspect ratio is equal to *max(h, w)/min(h, w)*. Angular resolution applies to straight line drawings, and it refers to the smallest angle formed by two edges incident on the same vertex. Note that a drawing of a graph generally can not simultaneously optimise all the criteria.

2.3 Spring Algorithms

In the drawing of undirected graphs there is a problem of "too much freedom". The spring algorithm(Eades 1984) is a successful layout creation algorithm for drawing an undirected graph. The basic idea of the spring algorithm is to treat a graph as a mechanical system, in which nodes are replaced by steel rings and edges by springs connected to the rings. All the springs have the same *natural length k*, and each spring has a *current length d*. Given a pair of rings connected by a spring, if $k > d$, then the spring attracts the rings; if $k < d$, then the spring repulses the rings; if $k = d$, then the rings are stable. The spring forces will attract or repulse the rings until the system reaches the *minimum energy.* This is called the *balanced state.*

The strength of the forces is determined by k and d. Eades calculates the forces using the functions:

$$f_a = C_1 * log(k/C_2)$$

$$f_r = C_3/sqr(d)$$

where f_a is the attractive force, f_r is the repulsive force, and C_1, C_2, C_3 are coefficients.

There are some variants of the spring algorithm that adopt different force models. All of these algorithms are broadly called force-directed algorithms in most literature. Eades and Lin review some of these algorithms and present a general spring system(Eades &Lin 1999).

The input of a spring algorithm is an undirected graph, and the output of the algorithm is a drawing of the embedded graph. The running time of a spring algorithm is $O(n^2)$, where n is the number of the nodes. The layout generated by a spring algorithm has two features: display of symmetry and uniform edge length.

A spring system is not guaranteed to generate a good drawing for all graphs. The forces exerted on some rings might be too strong or too weak, and the spring system cannot arrive at the *balanced state.* One solution is to adjust the coefficients; another is to adopt different force models. Some implementations, however, also set a fixed number of iterations, in order to end the spring forces in a predictable time.

2.4 Cluttered Layouts and Layout Adjustment Algorithms

An algorithm for drawing abstract graphs is a layout creation algorithm. In such a drawing, the images of nodes are concrete points. The output of a layout creation algorithm might have overlapping nodes when labels are added. Such overlapping is called *static drawing overlapping* in this paper. Another kind of overlapping is occurred in dynamic drawing systems. The new nodes or enlarged nodes might overlap their neighbour nodes. This kind of overlapping is called *dynamic drawing overlapping* in this paper.

Cluttered drawings can not convey information effectively. It is essential for the drawings to remove the overlapping. An algorithm that removes the overlapping of a layout is a layout adjustment algorithm.

Note that there are two possible kinds of overlapping in a drawing of a graph. One is the image of a node overlapping the image of another node; another is the drawing of an edge crossing the image of a node. In this paper, we only study the former case: overlapping among images of nodes.

2.5 Notations

In this paper, we denote a graph as G, and a drawing of G as Γ. An edge is referred to as $e(v_1, v_2)$, where v_i and v_j are the two end nodes connected by the edge. $d_{vi,vj}$ is the distance between the center points of v_i and v_j. $d^x_{vi,vj}$ is the distance of projection of the center points of v_i and v_j in the x direction, $d^y_{vi,vj}$ in the y direction. f_a is the attractive spring force in a spring system, and f_r the repulsive spring force.

The images of nodes are rectangular shapes, the sides of which are parallel to the coordinate system axes. A drawing of a node in Γ is refereed to as $v_i(x_i, y_i, w_i, h_i)$, where x_i and y_i are coordinates of the node's center point in the x and y directions, respectively, and w_i is the width, and h_i the height.

3 Algorithms

Most existing spring algorithms use a fixed natural length k to calculate the spring forces. The generated layout is usually a good drawing for an abstract graph. However, there might be overlapping nodes when labels are added. To fix the overlapping problem, we proposed variants of the spring embedder model, namely, *Dynamic Natural Length Spring*(*DNLS*) and *Orthogonal Dynamic Natural Length Spring* (*ODNLS*).

3.1 Dynamic Natural Length Spring(DNLS)

DNLS is designed for solving the problem of static drawing overlapping. DNLS calculates the forces using a *dynamic natural length* k_d that is determined by both k and the sizes of nodes.

Suppose the drawings of a pair of nodes v_1, v_2 are rectangles. The images of two nodes will not overlap each other if $d_{v1,v2} > k'$,
where

$k' = l_1/2 + l_2/2 + m$

l_1 is the diagonal of v_1
l_2 is the diagonal of v_2

m is the *desired margin* between v_1 and v_2

we call k$'$ the *clean distance*. we define the dynamic natural length k_d between the nodes v_1 *and* v_2 as

k_d=*max(k$'$, k)*

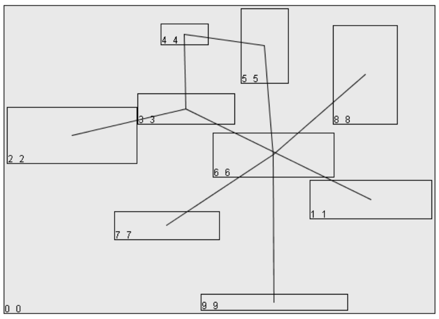
When the spring mechanism system calculates the forces between a pair of nodes, k_d is used to determine the strength of the forces. If there is overlapping between a pair of nodes, then $d_{v1,v2} < k_d$. The spring force will repulse them. If $d_{v1,v2} > k_d$, the nodes are not overlapping but the edge connecting them is longer than necessary. The spring force will attract the two nodes. The spring system will not arrive at the balanced state unless there are no overlapping nodes and the length of every edge is k_d.

DNLS is a layout creation algorithm that can avoid overlapping. The idea of dynamic natural length can be applied in any force model. In our implementation, we use the force models below:

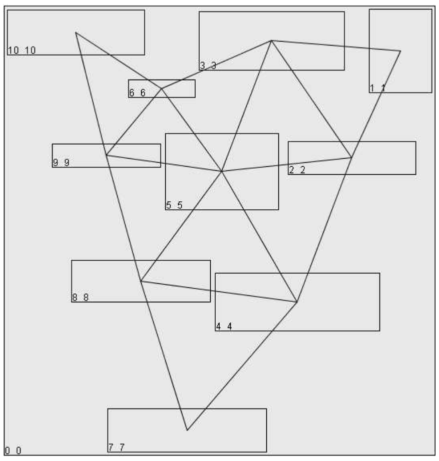
$f_a = d/k_d$

$f_r = (k_d/d)^2$

Figure 1 are two sample layouts generated by DNLS.



(a) Drawing of a graph with 9 Nodes



(b) Drawing of a graph with 10 Nodes

Figure 1: Drawings generated by DNLS

3.2 Orthogonal Dynamic Natural Length Spring(ODNLS)

DNLS is a layout creation algorithm, and does not consider the mental map. However, it is necessary to preserve the mental map in dynamic graph drawing systems. ODNLS is a post-processing approach for solving the problem of *dynamic drawing overlapping*. It preserves the mental map measured by the orthogonal ordering model. The input of ODNLS is a cluttered layout. The output is a clean layout that has the same orthogonal order as the input layout.

ODNLS uses dynamic natural length k_d to calculate the energy of a spring system. An extra force f_e is added to the spring system in order to keep the *orthogonal ordering* the same as that of the input layout. f_e works as follows.

Suppose $v_1(x_1, y_1, w_1, y_1), v_2(x_2, y_2, w_2, h_2)$ are drawings of a pair of nodes in Γ, $x_1 < x_2$, $y_1 < y_2$, k_d is the dynamic natural length, and m the desired margin between the nodes in Γ. Let

p.left=min(x_1, x_2)

p.right=max(x_1, x_2)

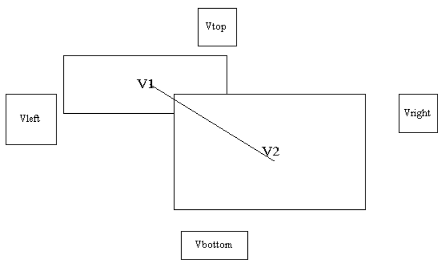
p.top=min(y_1, y_2)

p.bottom=max(y_1, y_2)

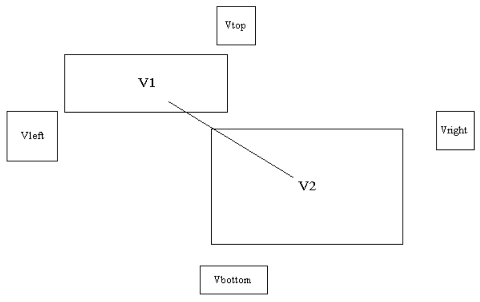
If $d_{v1,v2} < k_d$, f_r will repel v_1 and v_2. Let r_x and r_y refer to the repulsive distances in the x and y directions, respectively. For $\forall$ v(x, y, w, h) $\in \Gamma$, f_e sets:

- $x := x + r_x$, if $x >= p.right$
- $x := x - r_x$, if $x <= p.left$
- $y := y + r_y$, if $y >= p.bottom$
- $y := y - r_y$, if $y <= p.top$

In Figure 2(a), v_1 and v_s are a pair of overlapping nodes. When f_r repulse them, v_{left}, v_{top}, v_{right}, and v_{bottom} are pushed by f_e left, up, right, and down, respectively. The new drawing is in Figure 2(b).



(a)



(b)

Figure 2: f_e repulsive effect

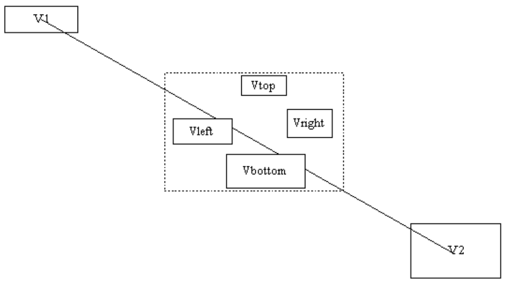
If $d_{v1,v2} > k_d$, f_a will attract v_1 and v_2. Let a_x and a_y refer to the attractive distances in x and y directions, respectively. Suppose a set of nodes V_k that are between v_1, v_2 in x direction, and v_{left} is the leftmost node of V_k, and v_{right} the rightmost. Also suppose a set of nodes V_j that are between v_1, v_2 in y direction, and v_{top} is the uppermost node of V_j, and v_{bottom} the lowermost. f_e will move:

- v_1 to where

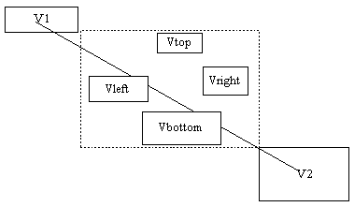
  - $d^x_{v1,vleft} = w_1/2 + w_{left}/2 + m$, and
  - $d^y_{v1,vtop} = h_1/2 + h_{top}/2 + m$

- v_2 to where
  - $d^x_{v2,vright} = w_2/2 + w_{right}/2 + m$, and
  - $d^y_{v2,vbottom} = h_2/2 + h_{bottom}/2 + m$

In Figure 3(a), v_1 and v_2 are attracted by f_a, while f_e will not allow the two nodes moving inside the dashed rectangle. Note that the margins between the dashed rectangle and v_{left}, v_{top}, v_{right}, and v_{bottom} are the desired margin between nodes in the drawing. Figure 3(b) shows the positions of v_1 and v_2 determined by f_e.



(a)



(b)

Figure 3: f_e attractive effect

Now, Let e_{x1}, e_{y1}, e_{x2} and e_{y2} refer to the distance between v_1 and v_2 covered by f_e in the x and y directions, respectively. Then

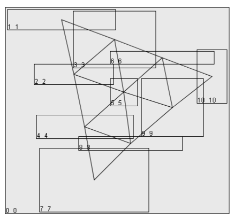
- $dx_1 = min\{0, e_{x1}, a_x\}$
- $dy_1 = min\{0, e_{y1}, a_y\}$
- $dx_2 = min\{0, e_{x2}, a_x\}$
- $dy_2 = min\{0, e_{y2}, a_y\}$

For $\forall v(x, y, w, h) \in \Gamma$, the spring system sets

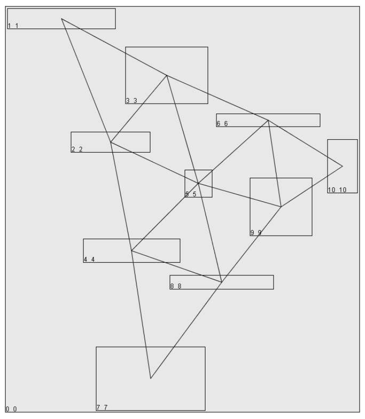
- $x := x + dx_1$, if $x = x1$
- $x := x - dx_2$, if $x = x2$
- $y := y + dy_1$, if $y = y1$
- $y := y - dy_2$, if $y = y2$

By adding f_e to the spring system, ODNLS can also preserve the orthogonal order. Since ODNLS is a variant of spring model as well, the output of ODNLS has the features of the spring algorithms.

Figure 4 is a sample of ODNLS. Figure 4(a) is a cluttered drawing of a graph with 10 nodes. Figure 4(b) is the output of ODNLS.



(a) Cluttered Initial layout(INI)



(b) Output of ODNLS

Figure 4: Sample of ODNLS

4 Experiments

4.1 Overview

DNLS and ODNLS are variants of classical spring models. FS and its variants, FS′ and FT, are actually simplified spring models, which have only repulsive forces but no attractive forces. Our experiments evaluate the outputs of the two types of algorithms.

In the experiment, we first generate a layout for a data set by a spring algorithm. The layout becomes cluttered after labels are added. We call the cluttered layout the *initial layout(INI).* We also generate a layout using DNLS. All of the other algorithms, ODNLS, FS, FS′ and FT, are adjustment algorithms, and each of them produces a clean layout for INI. The experiments will evaluate these layouts.

The measurements for evaluation are area, L1 metric length, aspect ratio, edge ratio, orthogonal ordering, running time, and effectiveness of removing overlapping. Edge ratio refers to the ratio of the longest and the shortest edge length, which is used here to evaluate how even the edges are. We also test the *symmetry* presented in the layouts by visual observation. Note that the edge crossing is not considered in the experiments. The reasons are: 1. the spring algorithms have no mechanism to reduce the edge crossing, 2. we only study the node overlapping and do not take into account the edges.

All nodes are assigned the same size. There are two reasons for this. First, uniform node size is adequate for most practice applications. A simple way of keeping nodes the same size is to adjust font size of the labels. Another common way is to compress or truncate long labels. The second reason and more important reason is that if nodes of different sizes are used for experiments, for example, and the sizes are randomly generated, it might be difficult to analyze of the results. For example, it is hard to determine whether the difference of area of different drawings is caused by the layout algorithms or by the size of nodes. Therefore, we simplify the experiment by the assumption that all nodes have the same size. In addition, nodes are drawn as rectangles, and they have a larger width than height. This is the common shape in most practical applications.

Six sets of data are tested in our experiment. Table 1 gives a description of the data sets. The first column of the table is the *ID* of the data set. The second column is the number of nodes of the data set. The third is the number of overlapping nodes in the *initial layout*. The last column is the drawing pattern presented in the *initial layout*.

Data	Nodes	Overlapping	Pattern
No.1	7	16	closed star
No.2	9	12	tree
No.3	15	20	triangle
No.4	21	35	opened star
No.5	25	37	grid
No.6	44	79	tree

Table 1: Experiment data set

4.2 Analysis

Figure 5(a) - 5(e) show the experimental results for area, L1 metric length, aspect ratio, edge ratio, and running time, respectively. In each figure, there are 6 histogram sets representing the results of the 6 data sets described in Table 1. The results for area and L1 metric length are normalised.

4.2.1 FS and its variants

FS′ and FT are the variants of FS. They can be seen as one algorithm family. FS′ improves the results of area and L1 metric length of FS in all cases. It has similar results for aspect ratio and edge ratio to FS in most cases. FS′ moves the nodes in the same direction as FS does, but more efficiently in the cases where overlapping occurs between more than two nodes. So it can improve the area and L1 Metric length of FS but keep the the aspect ratio and edge ratio.

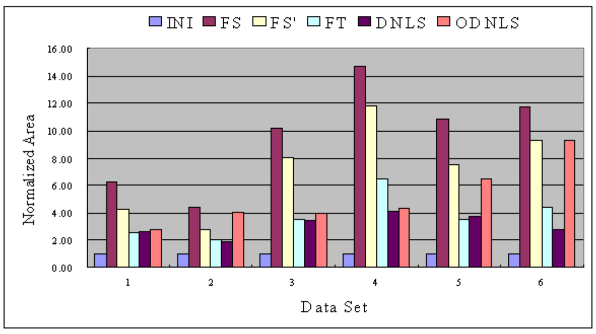
Results show that FT has the best area in this algorithm family. It also has a better L1 metric length than that of FS. However, the aspect ratio of FT is worse than those of FS and FT′ in all cases. The output of FT tends to increase more in the vertical direction than in the horizontal direction. The reason is that FT moves nodes in the direction of which desirable distance is smaller. Because all the nodes have the same size and the height of a node is smaller than the width, there is more possibility of moving in the vertical direction than in horizontal direction. The same reason also causes the large edge ratio in some cases.

The drawings of all the FS family have large edge ratio, that is, the edge length are not uniform. It indicates that the advantages of the INI, symmetry and uniform edge length, are destroyed after overlapping removed.

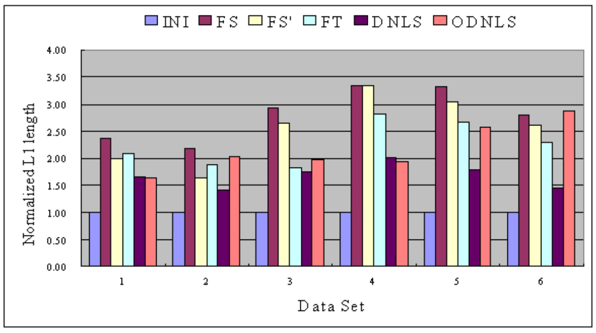
It takes almost the same time to run the three algorithms in all cases.

4.2.2 DNLS

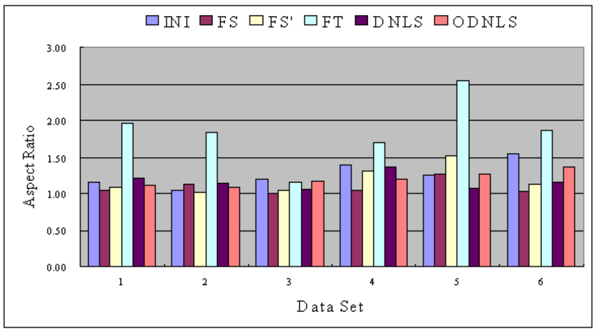
DNLS has the smallest area of data sets 4 and 6. In the other data sets, DNLS has similar area results to FT, which generates the smallest area drawing in the FS family. DNLS removes overlapping by both pushing and pulling the overlapped nodes. It also has the ability to evenly distribute the nodes. On the other side, the FS family has the ability to enlarge the drawing, but no ability to compress it. When they move the overlapping nodes, the FS family have to push



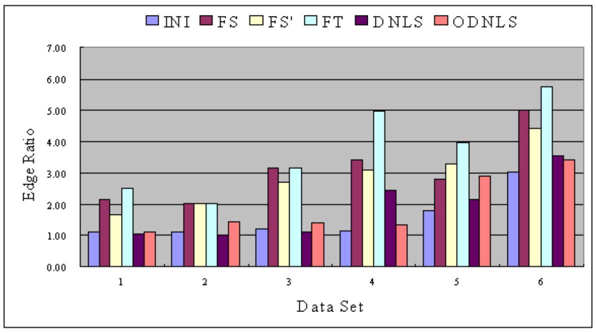
(a) Normalized Area



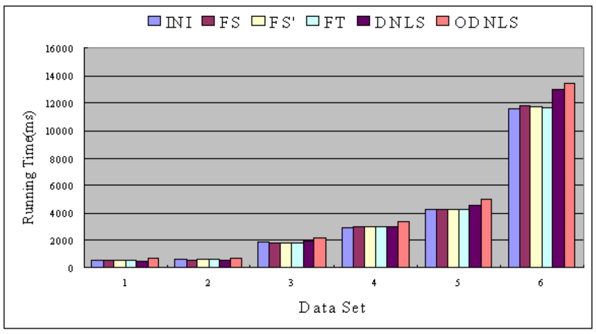
(b) Normalized L1 Metrics Length



(c) Aspect Ratio



(d) Edge Ratio



(e) Running Time

Figure 5: Experiment Results

away some uncluttered nodes in order to preserve the orthogonal ordering. Those uncluttered nodes will not be pulled closer to the overlapping nodes. So the output of DNLS is usually smaller than those of FS family.

In all the cases, DNLS has a better L1 metric length than any member of the FS family. This means that the values of width and height of the layouts generated by DNLS are closer than those of the FS family, that is, the aspect ratio should be closer to 1. The results of aspect ratio support this conclusion.

Basically, DNLS is a spring algorithm. So it should have the advantages of the spring algorithms: display of symmetry and uniform edge length. Uniform edge length is proved by our results for aspect ratio and edge ratio. Figure 5(c) and 5(d) show that DNLS has very similar results to INI, which is generated by a spring algorithm, in terms of aspect ratio and edge ratio in all cases. Visual observation confirms the display of symmetry.

The experimental results show that DNLS has better performance than the FS family in terms of the tested aesthetic criteria.

The results also indicate an interesting feature of DNLS. Because node size is constant in the experiments, the dynamic natural length k$'$ of all pairs of nodes of DNLS is the same. In such cases, DNLS is actually a classical spring algorithm whose natural length k can prevent nodes overlapping. Thus, we can say that, in a static graph drawing system that employees a classical spring algorithm, it is a better solution to produce an uncluttered layout by increasing the natural length of the spring system than by adopting the FS family as a post-processing adjustment.

It takes DNLS less time to draw a graph in data sets 1 and 2 than the FS family. However, the results also show that DNLS is slightly slower in data sets 4, 5, 6. Note that the running time is the sum of the time of creation and the time of adjustment, if a post-processing adjustment is applied. It seems strange that DNLS is slower than the FS family, because DNLS is basically a layout creation spring algorithm, while the FS family is a post-processing of another layout creation spring algorithm. We believe this is the trade-off of computing the dynamic natural length k$'$. DNLS find the spring forces, both attractive and repulsive, by computing k' for each pair of nodes in every iteration. Note that the running time of the spring algorithm is $O(n^2)$. When the number of nodes is small, the computing trade-off is trivial. However, the trade-off will become obvious when the number of nodes increases.

4.2.3 ODNLS

ODNLS has similar results for aspect ratio and edge ratio to INI. This means that ODNLS has some features of a spring algorithm.

ODNLS sometimes costs more space to draw graphs than DNLS. The results for area and L1 metric length are larger in data sets 2, 5, and 6 than that of DNLS, and similar to DNLS in data sets 1, 3, and 4. Compared to the FS family, ODNLS's performance of area in its good cases is similar to FT, but never worse than FS in any case.

Experiments of preserving orthogonal ordering are carried on for ODNLS and FS family. The results show that they all preserve *orthogonal ordering*. Both ODNLS and the FS family increase the area of the drawings, but the FS family destroy the aesthetic beauty.

ODNLS is a post-processing adjustment algorithm. It is slower than any algorithm tested in the experiments. The running time of both the FS family and ODNLS is $O(n^2)$. However, in the best case, the FS family runs in time $O(n)$, while ODNLS in time O(i*(m + $O(n^2)$)), where i is the number of iterations, and m is the number of edges. One way to improve the practical running time of ODNLS is to decrease k. But it might be unsatisfying from the aesthetic point of view if k is too small. In our implementation, *k=20.*

4.2.4 Effectiveness of Removing Overlapping

Table 2 shows the result of removing overlapping by these algorithms. The column 2 to 7 are the results of the algorithms when applied to data sets 1 to 6.

Algorithm	No1	No2	No3	No4	No5	No6
FS	0	0	0	0	0	0
FS'	0	0	0	0	0	0
FT	0	0	0	0	0	0
DNLS	0	0	0	5	6	13
ODNLS	0	0	0	0	0	0

Table 2: The Results for Removing Overlapping

We can see from Table 2 that all the algorithms of the FS family can remove the overlapping completely.

DNLS does not guarantee to remove overlapping. Just as a classical spring algorithm can not find a good drawing for all graphs, DNLS can not remove all overlapping in drawings of all graphs. The most common solution is to adjust the coefficients, to increase iteration time, or to change the force model. But this would still not guarantee to completely remove overlapping. An effective method is to use a FS algorithm as an additional step after DNLS. The extra post-processing adjustment will not much change the aesthetic properties. Comparing to Table **??**, we can see that DNLS has significantly reduced the number of overlapping nodes. Visual observation also shows the overlapping area of DNLS is much smaller than those of FS family. So the number of nodes to be moved and the distance of the motion are relatively small when an algorithm of the FS family cleans the output of DNLS. The appendix contains snapshots of layouts generated for data sets 3 and 4 by all the algorithms tested. The output of DNLS of dataset 4 is processed by FS$'$. The drawing still presents more uniform edges and displays more symmetry than the FS family do.

ODNLS can remove all the overlapping nodes. We believe that this is the effect of the extra force f_e, although no mathematical proof has been given. In a ODNLS system, when f_r repels the overlapping nodes, and f_e prevents the repulsed nodes from creating new overlapping. When f_a attracts the nodes connected by an edge, f_e will not allow the attracted nodes overlapping any nodes in between them. So f_e preserves the *orthogonal ordering*, as well as preventing overlapping.

4.3 Summary of the Experiment

The experiments show that the FS family is guaranteed to remove overlapping, but they usually destroy the aesthetic beauty of the initial layout. Although the running times of DNLS and ODNLS is a little longer than those of the FS family in some cases, the results show that both DNLS and ODNLS have the one of the features of spring algorithms: uniform edge length. We do not do a quantitative analysis of symmetry, but the visual observation confirms that in most cases DNLS and ODNLS present symmetry.

There is an interesting fact about the static drawing overlapping. It is better to adjust the natural length of a classical spring algorithm to avoid overlapping nodes than to clean the generated layout by the algorithms of the FS family.

DNLS is a layout creation algorithm, and does not consider the issue of preserving the mental map. ODNLS and the FS family are post-processing adjustment algorithms, and they can all preserve the mental map according to the orthogonal ordering model.

5 Conclusion and Future Work

5.1 Conclusion

This paper addresses the issue of removing overlapping of drawings of graphs. We present two algorithms, namely, DNLS and ODNLS. Both algorithms are based on the spring embedder model. Instead of a fixed value of the natural length in the spring system, DNLS and ODNLS use the *dynamic natural length k* to compute the attractive forces and repulsive forces exerting on the nodes.

DNLS is designed for static drawing overlapping, and it does not consider the mental map. ODNLS is an algorithm for dynamic drawing overlapping. It adds an extra force f_e to the spring embedder of DNLS in order to preserve the mental map. When a pair of nodes are attracted, f_e will prevent the pair from overlapping the nodes in between them. When a pair of nodes are repulsed, f_e will push in the corresponding direction all the nodes that are not in between the pair. The produced uncluttered layout keeps the same orthogonal ordering as the initial layout. f_e also helps to complete remove overlapping.

Experiments are carried out to compare the outputs of DNLS and ODNLS to those of the FS family. Six sets of data are tested in the experiments. Results show that the FS family guarantee to remove overlapping, but they usually destroy the aesthetic beauty of the initial layout. Both DNLS and ODNLS have a better performance than FS family in terms of aesthetic criteria. They have the features of spring algorithms: display symmetry and uniform edge. The results of uniform edge are quantitative, while the display of symmetry is visually observed. DNLS does not guarantee to remove all overlapping. But a FS family algorithm can be employed as a post-processing of DNLS. The output of the pipeline usually still has the features spring algorithms.

We also found an interesting fact about removing static drawing overlapping. It is better to increase the *natural length* of a classical spring algorithm to avoid overlapping nodes than to clean the generated layout by the algorithms of the FS family.

5.2 Future works

ODNLS and the FS family can be used in dynamic graph drawing systems. In this paper, we only evaluate these algorithms in a static environment. One future work is to evaluate them in a dynamic drawing system.

6 Acknowledgement

We would like to thank Damian Merrick and Tim Dwyer of the University of Sydney for helpful discussions about the spring algorithms. We also thank Tim Dwyer for his help through the writing of the paper.

References

Gansner E. & North S. (1998), Improved Force-Directed Layouts, *in* 'Graph Drawing', LNCS, Montreal, Canada, August 13-15, pp. 364–373.

Hayashi K., Inoue M., Masuzawa T., & Fujiwara H. (1998), A layout Adjustment Problem fo Disjoint Rectangles Preserving Orthogonal Order, *in* 'Graph Drawing', LNCS, Montreal, Canada, August 13-15, pp. 183–197.

Huang X., & Lai W. (2003), Force-Transfer: A New Approach to Removing Overlapping Nodes in Graph Layout, *in* '25th Australian Computer Science Conference', Adelaida, Australia, 2003.

Kamps T., Kleinz J., & Read J. (1995), Constraint-Based Spring-Model Algorithm for Graph Layout, *in* 'Graph Drawing', LNCS, F.J. Brandenburg, Canada, 1996, pp. 349–360.

Generating customized layouts. *in* 'Graph Drawing', LNCS, F.J. Brandenburg, Canada, 1996, pp. 504–515.

Di Battista G., Eades P., Tamassia R.,& Tollis I., (1998), *Graph drawing: algorithms for the visualization of graphs*, Prentice Hall.

Eades P. (1984), 'A Heuristci Graph Drawing', *Congressus Numerantium* **42**,1984.

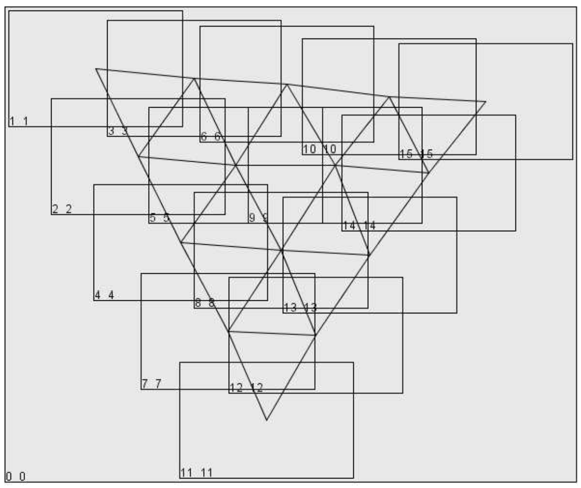
Eades P., Lai W., Misue K., & Sugiyama K. (1995), 'Layout Adjustment and the Mental Map', *Journal of Visual Languages and Computing* **6**, 183–210, 1995.

Eades P., & Lin X. (1999), 'Spring Algorithms and Symmetry', *Theoretical Computer Science* **240-2**, 379–405, 2000.

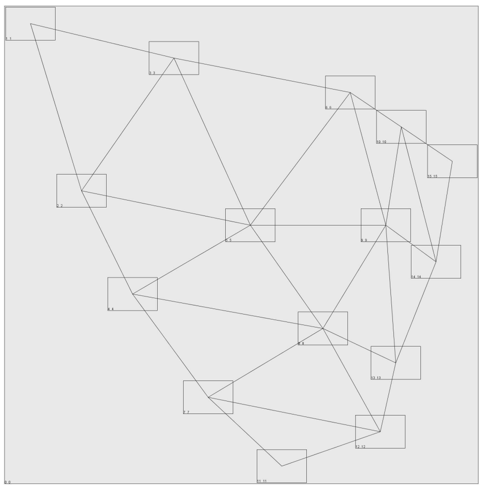
Harel D. &Koren Y. (1999), 'Drawing Graphs with Non-Uniform Vertices', *in* 'Proceedings of Working Conference on Advanced Visual Interfaces (AVI'02), Italy, 2002, pp. 157–166.

Lyons K., Meijer H.,& Rappaport D., (1998), 'Algorithms for Cluter Busting in Anchored Graph Drawing', *J. Graph Algorithms and Applications* **2-1**, 1–24, 1998.

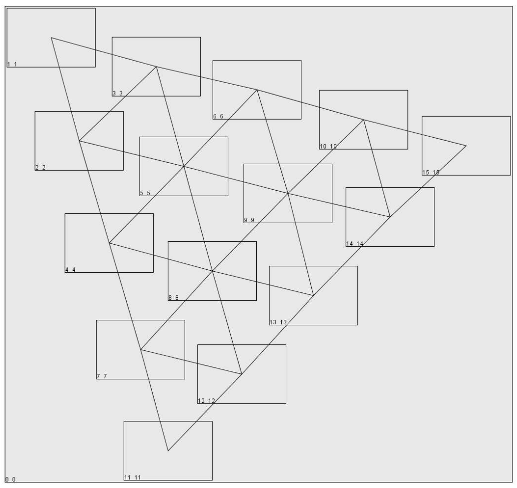
Marriott K., Stuckey P., Tam V., & He W. (1995), 'Removing Node Overlapping in Graph Layout Using Constrained Optimization', *Constraints* **8**, 143–171, 2003.



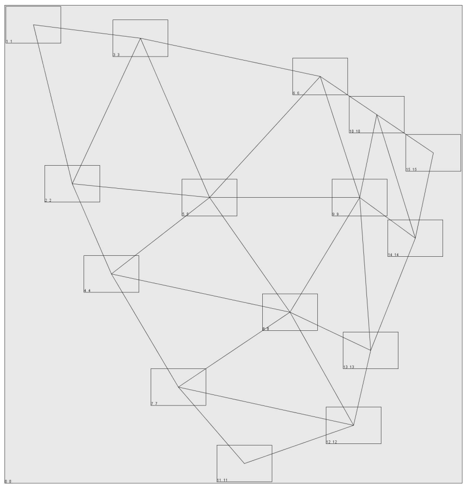
a) INI



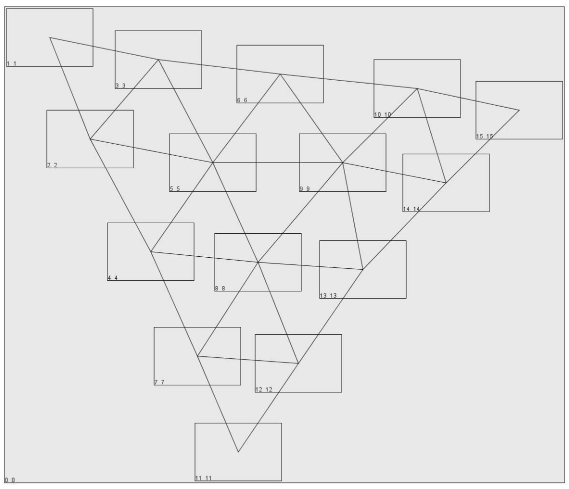
b) FS



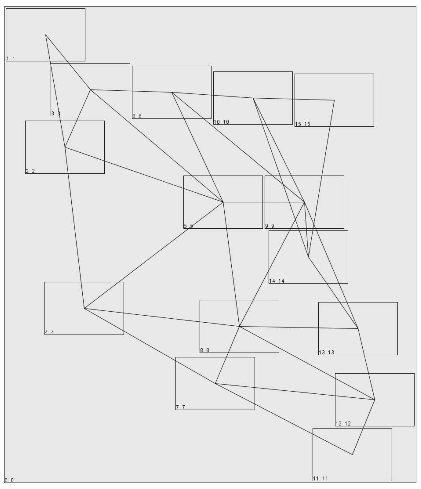
c) DNLS



d) FS'

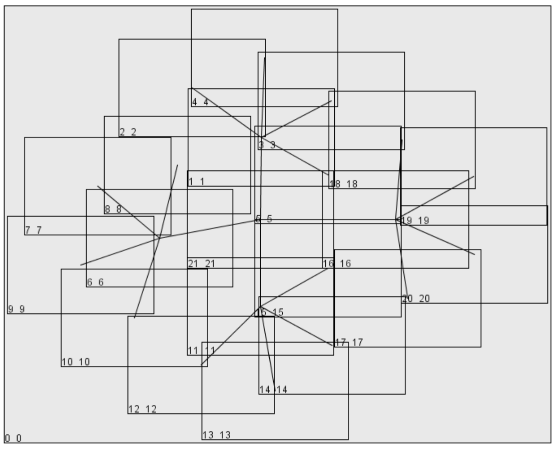


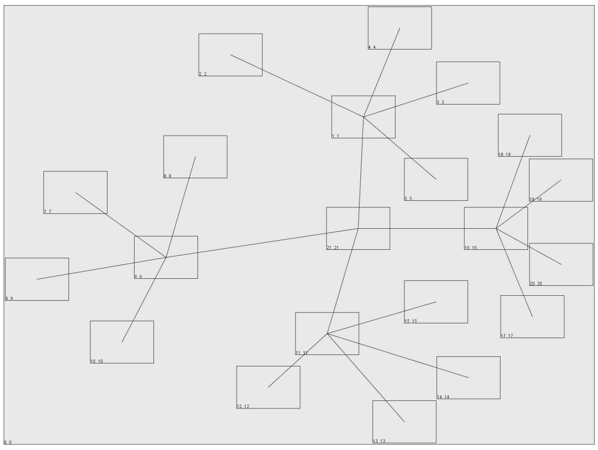
e) ODNLS

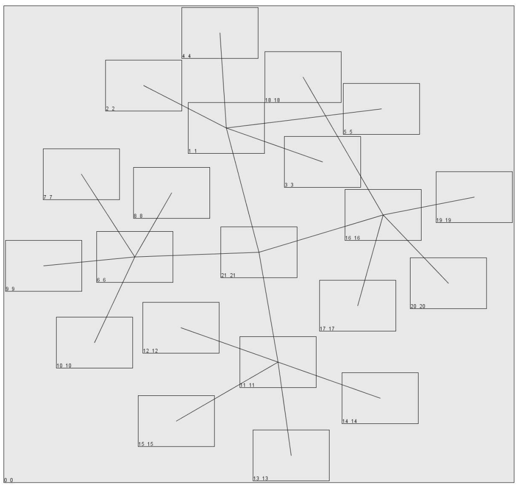


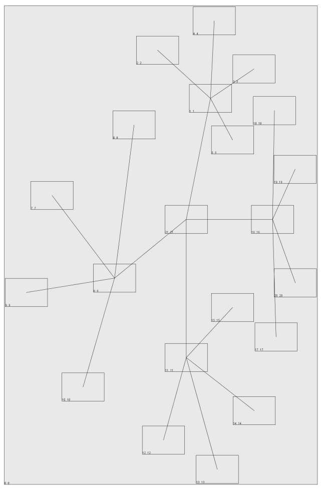
f) FT

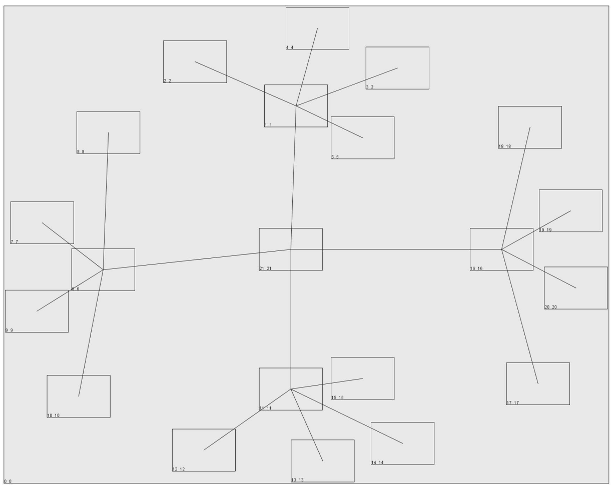
Appendix 1: Snapshots of data set 3

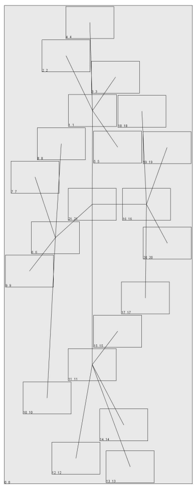

a) INI


b) FS


c) DNLS


d) FS'


e) ODNLS


f) FT

Appendix 2: Snapshots of data set 4