Author: Huang, Xiaodi; Lei, W; Gao, Junbin; Sajeev, A S M
Email address:- jbgao@csu.edu.au
Year:- 2007
Title: A new algorithm for removing node overlapping in graph visualization
Journal International Journal of Information Sciences
Volume: 177
Issue: If applicable
Pages: pp2821-2844
ISSN: 0020-0255
URL: http://www.sciencedirect.com/science?_ob=ArticleURL&_udi=B6V0C-4N43RYB-5&_user=1588505&_coverDate=07%2F15%2F2007&_rdoc=2&_fmt=summary&_orig=browse&_srch=doc-info(%23toc%235643%232007%23998229985%23651254%23FLA%23display%23Volume)&_cdi=5643&_sort=d&_docanchor=&_ct=12&_acct=C000053903&_version=1&_urlVersion=0&_userid=1588505&md5=cd04b0273bc1c71b4132530293498458
Abstract: Techniques for drawing graphs have proven successful in producing good layouts of undirected graphs. When nodes must be labeled however, the problem of overlapping nodes arises, particularly in dynamic graph visualization. Providing a formal description of this problem, this paper presents a new approach called the Force-Transfer algorithm that removes node overlaps. Compared to other methods, our algorithm is usually able to achieve a compact adjusted layout within a reasonable running time.

A New Algorithm for Removing Node Overlapping in Graph Visualization

Xiaodi Huang [a], Wei Lai [b], A. S. M. Sajeev [a], Junbin Gao [c]

[a] Department of Mathematics, Statistics and Computer Science, The University of New England, Australia
[b] School of Information Technology, Swinburne University of Technology, Australia
[C] School of Information Technology, Charles Sturt University, Australia

Abstract: *Techniques for drawing graphs have proven successful in producing good layouts of undirected graphs. When nodes must be labeled, however, the problem of overlapping nodes arises, particularly in the dynamic graph visualization. Providing a formal description of this problem, this paper presents a new approach called the Force-Transfer algorithm that removes node overlaps. Compared to other methods, our algorithm is able to achieve a compact adjusted layout with reasonable running time.*

Keywords: Graph Layout; Overlapping Nodes; Force Transfer; Layout Adjustment

1. Introduction

In the field of graph visualization, nodes in a graph represent objects or entities, which often have distinct labels as their identifiers. These labels in a drawing may be in the form of text, digits, or even images. Unlike most of existing drawing algorithms, nodes should be drawn as rectangles that have enough areas to display labels, rather than as abstract points with almost no size. UML diagrams in CASE tools, for example, are labeled graphs. The problem of node-overlapping may arise while displaying such graphs by using traditional algorithms. The resulting layout contains overlapping nodes, which destroy the layout aesthetics, an underlying purpose of graph layout. This is because most of these algorithms do not take into account the node size. The need thus arises for removing node-overlapping. More importantly, in a dynamic situation where the changes of a graph often happen such as enlargement/shrinkage of sub-graphs and addition/deletion of nodes, its layout should be adjusted accordingly. While eliminating node overlaps, the adjustment of an original layout should be kept to a minimum. The techniques for removing node-overlapping can directly be applied to reposition overlapping windows in multi-window applications, to avoid the overlaps of compound nodes in graph drawing [22，23], as well as to layout information display for small devices such as mobile phones and PDAs [16].

Three typical kinds of approaches to removing node-overlapping have been reported in the literature: uniform scaling [4, 7], constrained optimization [2, 5, 6], and force-based algorithms [1, 4, 12, 14]. Preserving the original structure of a graph, a straightforward approach called uniform scaling avoids node overlaps by uniformly scaling the overlapping layout. The layout may, however, be expanded unnecessarily. The adjusted layout thus tends to be too large. The constrained optimization approach makes use of an objective function, which consists of a quadratic expression about the differences between the initial and adjusted coordinates of nodes. An optimal solution to such a function is then provided, subject to a set of constraints that ensure no node-overlapping [5, 6]. These kinds of approaches can “give better layout than the force scan algorithm, although they are slower” [5]. The force-based algorithm includes cluster busting in an anchored graph drawing [14], as well as the force-scan algorithm (FSA) [1, 4, 12]. The procedure of cluster busting is iterative in that the nodes in a graph are iteratively relocated in accordance with measurable criteria. To improve the distribution of

nodes (cluster busting), these criteria minimize the differences between the original layout and adjusted layout (anchored graph drawing). Also, these algorithms have to run several iterations so as to achieve a better-adjusted layout. Compared to uniform scaling, FSA produces a compact layout, preserving the mental map [12, 21] of an original graph. To make a graph layout as compact as possible, a variant of FSA [4] allows an additional pull force between two nodes. Considering the node size in the original spring embedded algorithm, D. Harel et al. [17] recently proposed a modified spring method that prevents node-overlapping at the beginning of a layout. Other related work includes the SHriMp algorithm [15] where nodes uniformly give up screen space to allow a node of interest to grow. These nodes are appropriately scaled to fit inside the screen.

Using a heuristic method, our new approach called the Force-Transfer Algorithm (FTA) approximates a global, optimal adjustment with minimal local changes.

The major contributions of this paper lie in: (1) formalizing the node-overlapping problem in graph visualization; (2) proposing a new approach to removing node-overlapping; (3) providing a scalable version of FTA; (4) presenting the properties of FTA; and (5) implementing the proposed algorithm in a prototype called PGD (Practical Graph Drawing).

The remainder of this paper is organized as follows. The following section presents a formal description of removing node-overlaps problem. Section 3 describes FTA, followed by presenting its scalable version. The properties of FTA and their proofs are given in Section 5. Section 6 compares FTA to FSA. Section 7 reports the empirical results on a number of graph layouts with overlapping nodes, by applying both FTA and FSA. The potential applications are briefly described in Section 8. Finally, the conclusion is given in Section 9.

2. A Formal Description of the Removing Node -Overlapping Problem

We assume that an overlapping graph $G = (V, E)$ has the set of nodes denoted by $V = \{1, 2, ..., |V|\}$, and the set of edges by $E \subseteq V \times V$. Let (x^0_q, y^0_q) denote the centre of node q (i.e., the index of a node is q) with a rectangular bounding box of width w_q and height h_q, and denote (x^1_q, y^1_q), (x^2_q, y^1_q), (x^2_q, y^2_q) and (x^1_q, y^2_q) as its four corner coordinates (See node q in Figure 1). Also, we suppose that the nodes in G are sorted by their x^1_i and then by y^1_i in the horizontal and vertical directions, respectively, where i=1, 2, …, $|V|$, such as the coordinates of the upper left corner (x^1_q, y^1_q) of node q.

With the above preliminaries, we provide definitions and formalize the node-overlapping problem, starting by identifying all overlapping nodes of a node in a given graph.

Definition 1 (*Neighbor node*). Two nodes i and j in G with the following expression being true:

$$\begin{aligned} &| x_i^0 - x_j^0 | < (w_i + w_j)/2 + d \quad or \\ &| y_i^0 - y_j^0 | < (h_i + h_j)/2 + d \end{aligned} \qquad (1)$$

are called the *Neighbor node* of each other denoted by $N(i, j)$, where the gap d is a minimum, horizontal distance between nodes i and j that makes them un-overlapped.

To simplify notation, the gap d will be omitted in the rest of this paper. It is obvious that a *neighbor* relation is reflective and symmetric.

Definition 2 (*Neighbor nodes*). A set of nodes that overlap with node q directly. We have $NN(q) = \{i \mid N(i,q) \wedge i,q \in V\}$ where $NN(q)$ denotes the *neighbor nodes* of node q.

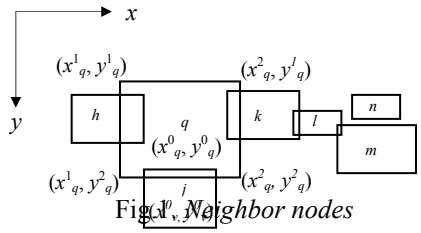


Fig. 1. Neighbor nodes

Considering the example shown in Figure 1, we have $NN(q) = \{h, j, k\}$. For describing our algorithm later, the *left, right, up* and *down neighbor nodes* of node q are defined by:

$$LNN(q) = \{i \mid x_i^1 < x_q^1 \wedge i \in NN(q)\},\ RNN(q) = \{i \mid x_i^1 \geq x_q^1 \wedge i \in NN(q)\}$$

$$UNN(q) = \{i \mid y_i^1 < y_q^1 \wedge i \in NN(q)\},\ \text{and}\ DNN(q) = \{i \mid y_i^1 \geq y_q^1 \wedge i \in NN(q)\}$$

Again, we have $LNN(q) = \{h\}$, $RNN(q) = \{j, k\}$, $UNN(q) = \emptyset$, and $DNN(q) = \{h, j, k\}$ for the graph in Figure 1.The *neighbors* relations on a set of $NN(q)$ is not transitive, or symmetric, but reflective.

Definition 3 (*Transfer neighbor nodes*). A set of nodes that overlap with node q both directly and indirectly; that is, all nodes around q in which every node overlaps with at least one other node.

The *transfer neighbor nodes* of a particular node, denoted by $TNN(q)$, are identified as follows: find the *neighbor nodes* of node q, i.e., $NN(q)$, and then all nodes in $NN(q)$ in turn find their *neighbor nodes* and so on, until a particular node with no *neighbor nodes* is encountered. A $TNN(q)$ is defined recursively in this way:

$$i{=}1: TNN(q)_1 = \left(\bigcup_{j \in NN(q)} NN(j) \right) \cup NN(q) - \{q\}$$

$$i{>}1: TNN(q)_i = \left(\bigcup_{j \in TNN(q)_{i-1}} NN(j) \right) \cup TNN(q)_{i-1} - \{q\}$$

We have $TNN(q) = TNN(q)_i$ where $i \in \{1, \cdots, |V|\}$. There exist $i \in \{2, \cdots, |V|\}$, $k \geq i$, and $k \in \{2, \cdots, |V|\}$ such that the equation $TNN(q)_k = TNN(q)_{k-1}$ always holds true. The $TNN(q)_i$ formed by the i-th iterative expansion is a subset of $TNN(q)$ where $i \in \{1, \cdots, |V| - 1\}$.

Again referring to Figure 1, we have:

$$TNN(q)_1 = NN(h) \cup NN(j) \cup NN(k) \cup \{h, j, k\} - \{q\} = \{h, j, k, l\}$$

$$TNN(q)_2 = NN(h) \cup NN(j) \cup NN(k) \cup NN(l) \cup \{h, j, k, l\} - \{q\} = \{h, j, k, l, m\}$$

The *transfer neighbor nodes* of node q are eventually obtained as $TNN(q) = \{h, j, k, l, m\}$. In a similar way, we are able to define *left, right, up* and *down transfer neighbor nodes*, such as $TLNN(q) = \{i \mid x_i^1 < x_q^1 \wedge i \in TNN(q)\}$, and $TRNN(q) = \{i \mid x_i^1 \geq x_q^1 \wedge i \in TNN(q)\}$. For instance, we have $TLNN(q) = \{h\}$, $TRNN(q) = \{j, k, l, m\}$, $TUNN(q) = \phi$, and $TDNN(q) = \{h, j, k, l, m\}$ in Figure 1.

The use of the following theorem will significantly reduce the computational complexity of the proposed algorithm.

Theorem 1 *Given a graph layout with overlapping nodes and* $i \neq j$, *the following claims hold true.*

1. *If* $TNN(i) \cap NN(j) \neq \phi$, *then* $TNN(i) \supset NN(j)$

2. *The transfer neighbor relation on a set of TNN(q) is an equivalence relation*.

Proof. 1. $TNN(i) \cap NN(j) \neq \phi$ holds, implying at least one node k in both TNN (i) and NN (j). We have $\exists u \in TNN(i)$ and $\exists w \in NN(j)$ such that both N (k, u) and N (k, w) are true. That is, node k acts as a bridge node to form a *neighbor node* with one node in TNN (i) and with another in NN (j). This indicates that these three nodes are *neighbor nodes*. According to Definitions 2 and 3, the claim is immediately correct.

2. We show the *transfer neighbor* relation (R) is reflective, symmetric and transitive. It is obvious that the relation is reflective. For the symmetric property, we show that i R j implies j R i. That is, if $i \in TNN(j)$, then $TNN(i) = TNN(j)$. Without loss of generality, we suppose $TNN(i) \subset TNN(j)$, then rewrite it as TNN (j) = TNN (i) + TNN (k) (1), where $TNN(k) \neq \phi$. It follows that there must be two nodes $u \in TNN(i)$ and $w \in TNN(k)$ such that u and w is a *neighbor node* namely N (u, w). Otherwise they cannot be included in TNN (j). According to the definition of *transfer neighbor nodes* and the fact that N (u, w) is true, TNN (k) can be viewed as part of TNN (i). This is because the neighbor nodes u and w link them together. On the other hand, given $i \in TNN(j)$ there must exist a node $v \in TNN(j)$ such that N (i, v) is true. For the same reason, TNN (j) can also be treated as part of TNN (i). As a result, we have TNN (i) = TNN (j) + TNN (k) +S (2) where S is a set. From (1) and (2) we have $TNN(k) + S = \phi$ which means $TNN(k) = \phi$. This results in a contradiction. It is readily shown that the *transfer neighbor* relation is transitive, meaning that $i \in TNN(j)$ and $j \in TNN(k)$ deduce $i \in TNN(k)$. The claim is therefore correct. □

Our algorithm is a force-based one, sequentially applying the forces within each scan. We need a particular node in a graph to start the scan.

Definition 4 (*Seed node*). A node s or a dummy node in a graph from which to start applying the first force in an adjustment. As a dummy node, i.e., a point without size, it is a referring point used as the benchmark of a scan.

One important issue related to the problem of removing node-overlaps is how to measure the quality of an adjustment. In the following we present three measures to quantify the degree of an adjustment.

A straightforward way is to count the number of nodes that has been repositioned after an adjustment, denoted by $\lambda_1 = n / |V|$, where n is the number of adjusted nodes.

The second measure quantifies the degree of the destroyed mental map after an adjustment. The preservation of orthogonal ordering can be measured by the *Kendall*'s *tau* distance [18, 19] that captures the number of disagreements between two rankings. In the case of graph layout, the rankings reflect the respective sequence of node relative positions in original and adjusted layouts denoted by V and V'. We define a mismatched set $\nu(V, V')$ as follows:

$$\nu(V,V') = \{(i,j) \in V \mid (x_i^1(V) > x_j^1(V) \wedge x_i^1(V') < x_j^1(V')) \vee (x_j^1(V) < x_j^1(V) \wedge x_i^1(V') > x_j^1(V')) \\ \vee (y_i^1(V) > y_j^1(V) \wedge y_i^1(V') < y_j^1(V')) \vee (y_i^1(V) > y_j^1(V) \wedge y_i^1(V') < y_j^1(V'))\}$$

In other words, the mismatched set contains a set of node pairs whose order has been changed in an adjusted layout. An indicator function is thereby given by

$$\tau(i,j)=\begin{cases}1 & if\ (i,j)\in v(V,V') \\ 0 & otherwise\end{cases}$$

The degree of orthogonal ordering changes is defined as

$$\lambda_2=\frac{2}{|V|(|V|-1)}\sum_{(i,j)\in V}\tau(i,j)$$

It is normalized by $|V|$ ($|V|$-1)/2, the maximum number of mismatched node pairs, so that it takes a value in [0, 1].

We are concerned also with the change areas of bounding rectangle after an adjustment. This is measured by $\lambda_3=1-W/W'$, where W denotes the minimal area of bounding rectangle of an original graph layout, and *W'* the area of the adjusted layout.

In fact, a compact adjustment is the primary goal of the removing node-overlaps problem. We therefore introduce another definition to measure the adjustment degree with respect to the adjusted distances.

Definition 5 (*Cost function*). A function used to measure the degree of an adjustment. Its value equals the sum of adjusted distances over all nodes in V after a layout adjustment:

$$f_{cost}(V,V',s)=\sum_{i\in V}\sum_{j\in V}(\Delta x_{ij}+\Delta y_{ij})$$

where Δx_{ij} and Δy_{ij} denote the moved distances of node j in the x and y directions caused by the applied force $\boldsymbol{f}_{ij}$. It is clear that this function is related to the *seed node s*.

By using an aggregation function, the combination of the number of adjusted nodes, orthogonal ordering, and the cost function yields the following measure that quantifies the degree of a layout adjustment.

$$\eta(V,V',s)=f(\lambda_1,\lambda_2,\lambda_3,f_{cost}(V,V',s))$$

As an example, the aggregation function $f(.)$ uses the following simple one:

$$\eta(V,V',s)=w_1\frac{n}{|V|}+w_2\frac{\sum_{(i,j)\in V}\tau(i,j)}{|V|(|V|-1)/2}+w_3(1-\frac{W}{W'})+w_4\sum_{i\in V}\sum_{j\in V}(\Delta x_{ij}+\Delta y_{ij})$$

where $w_1 + w_2 + w_3 + w_4$ is 1. For simplicity, we may assign equal importance to the three measures using equal weight values.

With the above preliminaries, we are ready to formalize the problem of removing node-overlaps.

Definition 6 (*Removing Node Overlap Problem*) (*RNOP*)**.** Given a graph G with an overlapping node layout, an adjustment for removing node overlaps should satisfy the following conditions:

1. $NN(q)=\phi,\ \text{for}\ \forall q\in V$
2. $\min\{(\eta(V,V',s)\}$

In the following, we show that the *RNOP* is *NP*.

Theorem 2 *RNOP* $\in$ *NP*.

Proof. we can solve *RNOP* with a nondeterministic algorithm. For all possible adjustments of a given overlapping graph, we compute the value of $\eta(V,V',s)$ to see whether it satisfies the requirements

given in Definition 6. It is obvious that the time complexity for checking whether or not any two nodes are overlapping is polynomial. □

Furthermore, *RNOP* is NP-complete. Computing the optimal *Kendall tau* aggregation, which serves as one component of our measure of an adjustment degree, is NP complete [19]. Since *RNOP* is an NP-complete problem, it is unlikely to find a good deterministic algorithm with polynomial running time. No complete solution to the problem has been reported in the literature. We resort to a heuristic approximation algorithm. As mentioned earlier, the minimization of adjusted distances is of primary importance in an adjustment. Relaxing other constraints, we present an algorithm that approximately minimizes the *cost function*.

3. Force-Transfer Algorithm

In this section, we present a new algorithm, called the Force-transfer algorithm, based on answering three related, fundamental questions.

The main idea of FTA is to apply a 'force' to one of two overlapping nodes so that this force pushes one node away from another. For all overlapping nodes in a graph, applying forces is conducted one by one within two scans: one left-to-right scan, and another top-to-bottom scan. As a result, some nodes are moved to avoid overlaps in either the horizontal or vertical direction. This scheme raises three issues: (1) how great a force should be; (2) in which way an applied force is transferred to other nodes; and (3) where to start with the first force. The answers that follow to these questions make a distinction between FTA and the widely used FSA. First, a minimum force between two overlapping nodes in FTA is applied so that a local adjustment is optimized. Second, the force is restrictedly transferred to a dynamic sub-graph that is a group of overlapping nodes. It is likely that a node does not overlap with any node in a sub-graph, but does with at least one of them after an adjustment. The nodes in the sub-graph are therefore updated iteratively during the scan. An adjustment will process until all sub-graphs including overlaps no longer exist in the final layout. It should be noted that a node with the initial applied force, called the seed node in FTA, can be any node of a graph.

We take the overlapping layout in Figure 1 as an example to illustrate our algorithm.

- Start by selecting node *h* as the starting point of the *x* and *y* direction scans.
- The force f_{hq} is applied to node *q* in order to separate it from its first overlapping node *h*, and is then transferred to the right nodes. This transferred force makes every node in the sub-graph consisting of nodes *q, j, k, l* and *m* move to the right at a distance that equals its magnitude.
- The node *j*, which overlaps with node *q*, is then processed. The fact that its horizontal overlapping distance is greater than the vertical one leads to postponing the reposition in the *y* scan.
- Next the applied force f_{qk} causes nodes *k, l* and *m* to move to the right. Two following forces f_{kl} and f_{lm} push nodes *l* and *m*, and then node *m* to the right, respectively.
- During this horizontal scan, node *l* may overlap with node *n*. For example, the force f_{kl} pushes node *l* such that it overlaps with node *n*. That is, a sub-graph is dynamically formed including

nodes *h, q, j, k, l, m* and a new member of node *n*. In this case, the subsequent force f_{ln} will make node *n* move to the right.

- The *x* scan ends by applying f_{lm} if the above case does not happen, and then the *y* scan starts where the f_{qj} pushes node *j* down.

Calculation of Forces

In this section, we provide a way of calculating the minimum applied force to separate two overlapping nodes.

According to Definition 1, two nodes *i* and *j* $\in V$ are separated such that the following equation holds true:

$$\begin{aligned} &| x_i^0 - x_j^0 | \geq (w_i + w_j)/2 \quad or \\ &| y_i^0 - y_j^0 | \geq (h_i + h_j)/2 \end{aligned} \qquad (1)$$

Consider a simple case of two overlapping nodes in Figure 2(a). With node *i* remaining unchanged, a force on node *j* can randomly be applied from anywhere to eliminate the overlap. The orthogonal projection of this force, however, must push *j* at a distance at least either Δx_{ij} to the right/left or Δy_{ij} down/up. FTA initially calculates the orthogonally overlapping distances between nodes *i* and *j*:

$$\begin{aligned} \Delta x_{ij} &= \min_{N(i,j):i,j \in V} \{| x_i^2 - x_j^1 |, | x_j^2 - x_i^1 |\} \\ &= (w_i + w_j)/2 - | x_i^0 - x_j^0 | \end{aligned} \qquad (2)$$

$$\begin{aligned} \Delta y_{ij} &= \min_{N(i,j):i,j \in V} \{| y_i^2 - y_j^1 |, | y_j^2 - y_i^1 |\} \\ &= (h_i + h_j)/2 - | y_i^0 - y_j^0 | \end{aligned} \qquad (3)$$

With the minimum magnitude of Δx_{ij} and Δy_{ij} , an applied "force" pushes *j* away from *i:*

$$| f_{ij} | = \min_{N(i,j):i,j \in V} \{\Delta x_{ij}, \Delta y_{ij}\}$$

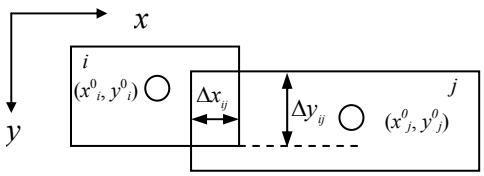
Considering the direction of the force, we rewrite the above equation as:

$$\boldsymbol{f}_{ij} = \begin{cases} (-1)^p \Delta x_{ij} \boldsymbol{i} & if \ \ (\Delta x_{ij} \leq \Delta y_{ij}) \wedge N(i,j) \\ (-1)^p \Delta y_{ij} \boldsymbol{j} & if \ \ (\Delta x_{ij} > \Delta y_{ij}) \wedge N(i,j) \\ 0 & otherwise \end{cases} \qquad (4)$$

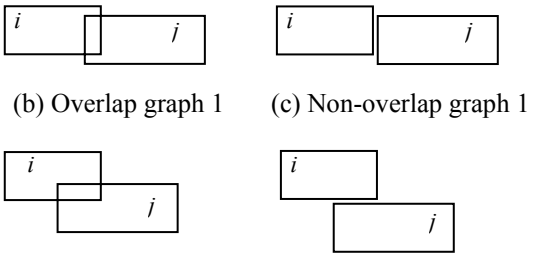
where

$$p = \begin{cases} 1 & if \ \ (x_i^1 < x_s^1 \wedge x_j^1 < x_s^1) \vee (y_i^1 < y_s^1 \wedge y_j^1 < y_s^1) \\ 0 & if \ \ (x_i^1 \geq x_s^1 \wedge x_j^1 \geq x_s^1) \vee (y_i^1 \geq y_s^1 \wedge y_j^1 \geq y_s^1) \end{cases}$$

and ***i*** and ***j*** are unit vectors in the *x* and *y* directions respectively. For the graph shown in Figure 2 (b), the applied force is chosen such that it has $f_{ij}^x = \Delta x_{ij}$ and $f_{ij}^y = 0$ as its *x* and *y* projections. This is because of different overlapping distances in the two directions ($\Delta x_{ij} < \Delta y_{ij}$). In contrast, the fact that the graph in Figure 2(d) has $\Delta x_{ij} > \Delta y_{ij}$ leads to using $f_{ij}^y = \Delta y_{ij}$ and $f_{ij}^x = 0$ instead.



(a) Overlap distances



(b) Overlap graph 1 (c) Non-overlap graph 1

(d) Overlap graph 2 (e) Non-overlap graphs 2

Fig.2. Overlap calculation

By applying the above forces f_{ij}^{x} and f_{ij}^{y} to the respective graphs in Figures 2 (b) and (d), the resulting layouts are obtained as shown in Figures 2 (c) and (e). Sequentially computing the orthogonal overlap distance of every pair of neighbor nodes in a graph, the four scans start from the *seed node* to those nodes located at its right, left, above, and below directions.

Transfer of Forces

As stated before, our approach sequentially conducts four orthogonal scans beginning with the *seed node*. Using the right scan as an example, we look at the way in which the force applied to a node is transferred to other nodes.

It is likely that an applied force on a particular node affects other subsequent nodes in the scan direction. In other words, each affected node is also moved by the same distance as does this node, receiving the equivalent force. After an adjustment, the moved distance of a node is determined by its own applied force, if any, as well as the transferred forces from other nodes. The propagation of an applied force on a node is, however, conditional on the overlapping pattern.

A group of nodes overlapping with each other, as mentioned before, is thought of as the *transfer neighbor nodes* of a node using Definition 3. The applied force on a node is sequentially transferred to all nodes in the *right transfer neighbor nodes* of this node. Formally, given i and $j \in TRNN\ (q)$ and $i < j$, the x projection of an applied force f_{ij}^{x} between nodes i and j creates an equivalent force on other nodes to the right, namely $k \in TRNN\ (q)$ and $k \geq j$. A node k obtains the aggregate forces that are individually applied to its preceding nodes included in $TRNN(q)$. After removing the overlaps among nodes between nodes i and k in $TRNN(q)$, the total force on the node k is given by $F_{k}^{x} = \sum_{i=m<n}^{k} f_{mn}^{x}$. It is easy to see that the rightmost node within a group of overlapping nodes moves the greatest distance.

A transferred force relocates all subsequent nodes within the set of $TRNN\ (q)$, thereby allowing some other nodes in a graph to turn into new members in some cases. In other words, the $TRNN\ (q)$ is dynamic in that it accepts as its new member a node that is not previously included. This happens in a

situation where the gap between two nodes is reduced to a certain extent, while nodes in *TRNN* (*q*) are being repositioned.

Determination of Seed Node

From the previous description, we know that the *seed node* plays an important role in our approach. A *seed node* is chosen in such a way that it minimizes the value of the *cost function*. As mentioned before, *RNOP* can be viewed as an optimization problem. This means the minimization of the objective function $f_{cost}(V,V',s)$, subject to removing all the overlapping in a graph. Many methods may be employed to find the solution such as Simulated Annealing [8], Genetic algorithm [9], and Tabu Search [10, 11]. In order to approximate a global optimization, these algorithms escape from being trapped in local minima by different ways. Applying those approaches to *RNOP*, an optimal point may be found as the centre of a *seed node*. In the case of the point not being inside any node in *V*, it is treated as a dummy node. Determining this optimal point is, however, a time-consuming process by means of these approaches. Our experiments have shown that in most cases, the choice of the leftmost node as the *seed node* is able to produce a better layout, except for symmetric overlapping layouts in which the centre nodes of the graphs should be used instead.

Despite the fact that there are many optimal methods for finding an optimal *seed node*, it is too slow to find it. In fact, it is not necessary to determine an accurate position of the *seed node*, since our approach provides only an approximate minimization of an adjustment. In the following we present a heuristics that seeks a *seed node* from an overlapping graph layout. The basic idea of this approach is to estimate the central location of all overlapping nodes according to their distribution.

We impose a window with a minimal size that covers all overlapping nodes of a graph layout. Given that the coordinates of the window centre is (x_0, y_0), we have the *seed node* (x_s, y_s) as follows.

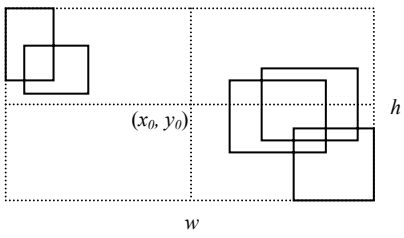


Fig.3. Determination of seed node

$$x_s = \frac{\sum_{N(i,j):\Delta x_{ij} \le \Delta y_{ij}} (-1)^p (x_i^1 + x_j^1)}{2n}, \text{ and } y_s = \frac{\sum_{N(i,j):\Delta x_{ij} > \Delta y_{ij}} (-1)^p (y_i^1 + y_j^1)}{2n}$$

where $p = \begin{cases} 1 & if\ x_i^1 \le x_0 (w/2)\ or\ y_i^1 \le y_0 (h/2) \\ 0 & if\ x_i^1 > x_0\ or\ y_i^1 > y_0 \end{cases}$, $j = \min\{j' \mid j' \in RNN(i)\}$, and n is the number of pairs of overlapping nodes.

The coordinates of overlapping nodes on the left of the window centre are negative whereas those on the right are positive in the above equation. As an example shown in Figure 3, the coordinates of first

two nodes are negative whilst the rest three ones are positive in the calculation. Also note that *n* equals 3.

Another useful variant of FTA utilizes multiple *seed nodes* rather than one *seed node* within a graph. In general, a given graph layout varies in the distribution of overlapping-node locations. Based on this fact, we assign the respective *seed node* in several areas surrounded by densely overlapping nodes. We will give this detailed variant of FTA in Section 4.

Pseudo-Code of Force-Transfer Algorithm

In the following pseudo-code, given $G = (V, E, s)$, let x^1_i denote the upper left corner x coordinates of nodes, where $i \in \{1,\cdots,|V|\}$, and s the *seed node*. The horizontal transfer is carried out in two directions, namely from the *seed node* to the right nodes, and then from the *seed node* to the left nodes. In the following we use the former as an example to show the algorithm.

Right Horizontal Transfer:

Input: *A graph G layout and a seed node s*

Output: *An adjusted graph layout where all node overlaps on the right side of the seed node are removed if their vertical overlapping distances exceed their horizontal ones*

$i \leftarrow s$

Find_ RNN (*G, s*)

Find_TRNN(*G, s*)

while $i <|V|$ {*right horizontal scan*}

if $TRNN(i) \neq \phi$ **then**

$$j = \min_{j' \in RNN(i)} \{j'\}$$

$$f^x_{ij} = x^2_i - x^1_j$$

$$f^y_{ij} = \min\{|\ y^2_i - y^1_j\ |, |\ y^1_i - y^2_j\ |\}$$

$$\delta \leftarrow \min\{|\ f^x_{ij}\ |, |\ f^y_{ij}\ |\}$$

if $\delta = f^x_{ij}$ **then**

// *Right horizontal transfer*

for *each* $j \in TRNN(i)$ *do*

$$x^0_j = x^0_j + \delta + d$$

end for

end if

end if

//*update TRNN*

$i \leftarrow i+1$

if $N(i, q)$ *is true* **and** $q \in V - TRNN(i)$ **then**

```
            TRNN(i)= TRNN(i)+{q}
        endif
    end {Right horizontal scan}
    // Find the right neighbor nodes
    Find_RNN (G, s) {
        for i=s to |V|-1
            RNN(i)={i}
            for j=s+1 to |V|
                    if N(i, j) is true  then
                            RNN (i) = RNN (i) + {j}
                    endif
            endfor
        endfor
    }
    // Find the right transfer neighbor nodes
    Find_TRNN(G, s) {
        i=1
        do
            for j=i+1 to |V|
                    if  TRNN(i) ∩ RNN( j) ≠ ϕ  then
                        //According to Theorem 1
                        TRNN(i) = TRNN(i) ∪ RNN( j)
                    end if
            end for
            i=i+1
            if  i ∈ TNN(i −1)  then
                        //According to Theorem 1
                        TRNN(i) = TRNN(i −1)
                else if  RNN(i) ≠ ϕ
                        then TRNN(i)= RNN(i)
                    end if
            end if
        while i ≤| V | −1
    }
```

The main challenge in determining the *TNNs* of all nodes for the above algorithm is that the number of *TNNs* may increase exponentially with the number of nodes in a given graph, particularly for large graph, making exhaustive approaches infeasible. To overcome this difficulty, we use Theorem 1, and a simple and powerful principle: the n-$TNN(q)$ comes only from $(n-1)$-$TNN(q)$, where n is the cardinality of a *TNN* set. Our approach has three-step procedure: (1) find all pairs of two overlapping nodes which

have a neighbor relation; that is, 1-*TNN* ; (2) recursively generate *n*-*TNN* from the (*n*-1)-*TNN* sets; and (3) iteratively execute Step (2) until no larger *TNN* can be formed.

Consider the example in Figure 1. We first find 1-*TNN*(*q*) as $\{h, q\}$, $\{q, j\}$,$\{q, k\}$, $\{k, l\}$,$\{l, m\}$, and then generate all *TNNs* by following the sequence$\{h, q\}$, $\{q, j\}$, $\{q, k\}$, $\{k, l\}$,$\{l, m\}$→ $\{h, q, j\}$, $\{j, q, k\}$, $\{q, k, l\}$, $\{k, l, m\}$ →$\{h, q, j, k\}$, $\{j, q, k, l\}$,$\{q, k, l, m\}$→$\{h, q, j, k, l\}$, $\{q, j, k, l, m\}$→$\{h, q, j, k, l, m\}$.

FTA consists of the above horizontal scan followed by a similar vertical scan. Overall, the force transfer starts from the *seed node*, and proceeds to other nodes in four orthogonal directions. The main FTA algorithm is shown as follows:

FTA algorithm

Input: *A graph G layout*

Output: *An adjusted layout without overlapping nodes*

Sort all the nodes according to their x^1_i *coordinates*

Find a node or use the leftmost node as the seed node s

Right Horizontal Transfer (G, s)

Left Horizontal Transfer (G, s)

Up Vertical Transfer (G, s)

Down Vertical Transfer (G, s)

---

4. Scalability of FTA

The above version of FTA cannot effectively handle a graph with a large number of overlapping nodes. This problem has become particularly important in the case of a large, dynamic graph adjustment. In the following we provide new versions of FTA. The idea is to apply FTA to a sequence of graphs at different abstract levels, respectively. These abstract graphs have much smaller number of overlapping nodes than the original one. This approach starts with grouping a sub-graph of overlapping nodes, namely, *transfer neighbor nodes*, into a virtual node. The original overlapping graph can then be thought of as a new, next abstract level graph, where the nodes consist of virtual nodes and the rest of ungrouped, original nodes.

Consider, for example, an original graph with three groups of overlapping nodes as shown in Figure 4(a). We impose three windows over these groups as big virtual nodes. The FTA is applied to removing the node overlaps within the virtual nodes respectively, the sizes of which are expanded accordingly. The three virtual nodes shown in Figure 4(b) then form the first abstract level of a new overlapping graph. In this way, we repeat this procedure until all overlaps have been removed.

Fig.4. Determination of seed node

The multiple abstract level version of FTA has still suffered the problem of slow running in the case of a *TTN* with a large number of overlapping nodes. Suppose that all the nodes have iteratively overlapped with each other in a large graph. In other words, a large graph includes only one *TTN*. We develop a grid version of FTA, in which a grid window is imposed over such a graph with densely overlapping nodes. After removing overlaps within each grid cell using FTA, all grid cells are composed of the next abstract level of overlapping graphs. In fact, the multiple abstract level version of FTA can be regarded as the adaptive grid version of FTA.

Based on the above ideas, the procedure for the grid version of FTA is given below.

A grid FTA

***Input**: a graph $G=(V, E)$ with overlapping nodes*

***Output**: an adjusted graph without overlapping nodes*

$l=0$, $V^0=V$

while *$|V^l| >$ the threshold* **do**

For all $q \in V^l$

if *$t < |TNN(q)| < t_1$* **then**

Impose a minimal window to include all adjusted nodes in TNN(q)

Apply FTA to TNN(q)

end if

if *$|TNN(q)| \geq t_1$* **then**

Impose a grid window over all overlapping nodes

Apply FTA to removing node-overlaps in each grid cell

The imposed windows are regarded as virtual nodes in the next abstract level of a graph; that is, V^{l+1}

end if

$l=l+1$

end while

Remove all rest overlaps

Note that all already adjusted nodes may still be relocated together with the relocation of its virtual node during the adjustment of the next abstract level of a graph.

In our experiments, the first threshold of the above algorithm is 30, while other thresholds are set to t =10~20 and t_1= 30~40. As for the size of a grid cell of the window, we provide an experimental principle: the number of overlapping nodes covered by each grid should not exceed 40.

5. Properties of Force-Transfer Algorithm

In order to show the correctness and some properties of our algorithm, we provide three theorems and their proofs in this section.

Theorem 2 *Given a graph layout with overlapping nodes, FTA ensures that the adjusted graph layout has no overlapping nodes*.

Proof. To prove that FTA works correctly, we need to show $NN(i) = \phi$ or $NN(j) = \phi$ where any two nodes *i* and *j* are on the same side of the *seed node s*. Suppose that for an adjustment, $x_i^0, x_j^0, x_i^{0'}$ and $x_j^{0'}$ denote the original and new *x* centre coordinates of nodes *i* and *j* respectively, and $x_j^0 \geq x_i^0$, then we have

$$\begin{aligned} x_j^{0'} - x_i^{0'} &= x_j^0 + \sum_{n=s+1}^{j}\sum_{m=s}^{n}\delta_{mn}^x - (x_i^0 + \sum_{n=s+1}^{i}\sum_{m=s}^{n}\delta_{mn}^x) \\ &= x_j^0 - x_i^0 + \sum_{n=s+1}^{j-i}\sum_{m=s}^{n}\delta_{mn}^x \\ &\geq x_j^0 - x_i^0 + \delta_{mn}^x \\ &= x_j^0 - x_i^0 - |x_i^0 - x_j^0| + \frac{w_i + w_j}{2} \\ &= \frac{w_i + w_j}{2} \end{aligned}$$

For $y_j^0 \geq y_i^0$, it can similarly be shown that $y_j^{0'} - y_i^{0'} \geq \frac{h_i + h_j}{2}$.

That is, the applied force on two random nodes separates them in the *x* or *y* direction with a distance of either $x_j^{0'} - x_i^{0'} \geq \frac{w_i + w_j}{2}$ or $y_j^{0'} - y_i^{0'} \geq \frac{h_i + h_j}{2}$. If nodes *i* and *j* locate at different sides of the *seed node s* such as the left and right sides, we are also able to give a similar proof. In conclusion, we have $NS(i) = \phi$ or $NS(j) = \phi$ after an adjustment. □

Theorem 3 *FTA is a polynomial-time* $\rho(|V|)$ *approximation algorithm, where* $\rho(|V|) = (|V|-1)^2$, *and* $|V|$ *is the number of nodes*.

Proof. We have already shown that FTA runs in polynomial time.

We consider the worst case in which all the right nodes of node *i* are iteratively overlapping with each other. In other words, the applied force between nodes *i* and *j* (*j* > *i*) in FTA is transferred to all the nodes to the right of node *i*, namely its *transfer right neighbor nodes* denoted as *TRNN* (*i*).

Without loss of generality, suppose that among number of |*TRNN*(*i*)| nodes within *TRNN*(*i*), there are *m* pairs of overlapping nodes *i* and *j* having $\Delta x_{ij} \leq \Delta y_{ij}$, where *m* is a positive integer. Also suppose that the *seed node* is the leftmost node in *V,* i.e., $s = v_1$.

For the removal of the first node-overlapping *j* of node *i*, and if we have $\Delta x_{ij} \leq \Delta y_{ij}$, the optimal adjustment at least pushes node *j* by the distance of Δx_{ij}. In contrast, within FTA each node in *TRNN*(*i*) has to be moved by a distance Δx_{ij} to the right. The adjusted distances for all nodes due to relocating node *j* thus amount to Δx_{ij} |*TRNN*(*i*)|. Based on the above assumption, *m* pairs of overlapping nodes in

$TRNN(i)$ lead to the total adjustment distances of $m\,\Delta x_{ij}$ in the optimal adjustment, and $m\,\Delta x_{ij}\,|TRNN(i)|$ in FTA. It follows that the rest of the nodes $|TRNN(i)|$- m should be repositioned in the y direction. Consequently, we have the lower bound of the optimal cost function as $f^*(v_1) \geq \sum_{i=1}^{|V|-1} \{ m\Delta x_{ij} + (|TRNN(i)| - m)\,\Delta y_{ij} \}$

where $j = \min_{j' \in RNN(i)} \{ j' \}$.

The adjustment cost of FTA is as follows

$$\begin{aligned} f^{FTA}(v_1) &= \sum_{i=1}^{|V|-1} \{ m\,|TRNN(i)|\,\Delta x_{ij} + (|TRNN(i)| - m)\,\Delta y_{ij} \} \\ &\leq \sum_{i=1}^{|V|-1} \{ m(|V| - 1)\Delta x_{ij} + (|TRNN(i)| - m)\,\Delta y_{ij} \} \\ &= (|V| - 1)^2 \sum_{i=1}^{|V|-1} \{ m\Delta x_{ij} + (|TRNN(i)| - m)\,\Delta y_{ij} \} \\ &= (|V| - 1)^2 f^*(v_1) \end{aligned}$$

An intuitive explanation for the above proof is as follows: one of each pair of overlapping nodes has been moved at a distance of Δx_{ij} (or Δy_{ij}). In FTA, all the nodes on the right of this node will be relocated the same distances. In the worst case, this amounts to $(|V| - 1)\Delta x_{ij}$ (or $(|V| - 1)\Delta y_{ij}$). It is easy to reach the conclusion for all $(|V| - 1)$ nodes. □

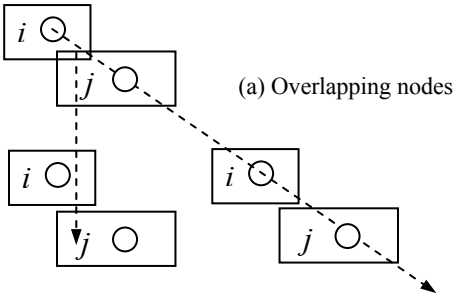
Theorem 4 *The time complexity of FTA is* $O(n^2)$.

Proof. As previously stated, the algorithm contains four scans. Here we look at the right scan again. The right scan starts by obtaining both the *right neighbor nodes* and *transfer right neighbor nodes* of each node on the right side of the *seed node.* Obviously, the running time of the former is bounded by $O(n^2)$ since there are two loops in the pseudo-code presented in Section 2.4. The worst-case time complexity of the latter is also $O(n^2)$. The computational complexity of the *transfer right neighbor nodes* of each node is greatly reduced by using Theorem 1. The *while* statement in the right scan and the updating of the *transfer right neighbor nodes* of the next adjusted node require $O(n^2)$ computational time at most. As such, the time complexity of FTA is $O(n^2)$ in the worst case. □

FTA cannot make sure that the adjusted layout preserves the "orthogonal ordering" mental map [12] of an original graph layout. The forces are only restrictedly transferred between neighbor nodes, thereby maintaining the relative positions among them. They, however, cannot ensure that the relative positions are preserved between neighbored and non-neighbored nodes. For this reason, a slightly varied FTA is provided to keep the mental map of "orthogonal ordering". This is accomplished by considering all nodes in a graph equivalently. More precisely, a transferred force on a node moves an identical distance for all successive nodes of this node in the scan direction, regardless of whether they are neighbor nodes or not. As a consequence, by preserving the "orthogonal ordering", this variant of FTA produces a less compact graph layout than the original FTA. Nevertheless, the resulting layout is still more compact than that by FSA due to the reason presented as Lemma 1 in Section 6. Note that we will not compare this variant of FTA to FSA in Section 7.

6. Comparison with Force-Scan Algorithm

In this section, we compare FTA to FSA. We start with a brief introduction of FSA, and then give a proof of the fact that FTA is superior to FSA.



(b) A force in FTA (c) A force in FSA

Fig.5. the forces in FSA and FTA

Similar to FTA, FSA uses a force to separate nodes *i* and *j* as shown in Fig. 5. However, they differ in the choice and transfer of the force. FSA uses the force that is along the line connected the centre of two overlapping nodes as shown. Moreover, a force in FTA is equal to the minimum of two orthogonal projections of the corresponding force in FSA. Additionally, this force is transferred only to the nodes within the *transfer neighbor nodes* of a particular node from which to be applied. This is in contrast to the fact that the force on the same node in FSA is unconditionally transferred to all nodes to the right of this node.

The horizontal scan of FSA is shown as follows [7, 12].

$i \leftarrow 1$

while $i<|V|$ **do**

$\quad$ *Suppose that* $x_i = x_{i+1} = \cdots = x_k$

$\quad \delta \leftarrow \max_{i \le m \le k \le j \le |V|} f_{mj}^x$

$\quad$ **for** $i \leftarrow k+1$ *to* $|V|$ **do** $x_j \leftarrow x_j + 1$

$\quad i \leftarrow k+1$

In the following we show that FTA outperforms FSA.

Lemma 1 *Given a graph layout with overlapping nodes, an applied force in FTA is not greater than the corresponding one in FSA, namely* $|\boldsymbol{f}_{ij}^{FTA}| \le |\boldsymbol{f}_{ij}^{FSA}|$.

Proof. We give the formulae for computing an applied force used to separate two overlapping nodes *i* and *j* in both algorithms. Note that for briefly, in the following we omit the parameters *G*, *G'* and s of the *f* function in Definition 5. From (2), (3) and (4), in FTA we have:

$$\delta_{ij}^x = f_{ij}^x = \begin{cases} (w_i + w_j)/2 - |x_i^0 - x_j^0| & \text{if } \Delta x_{ij} \le \Delta y_{ij} \wedge N(i,j) \\ 0 & \text{otherwise} \end{cases}$$

and

$$\delta_{ij}^y = f_{ij}^y = \begin{cases} (h_i + h_j)/2 - |y_i^0 - y_j^0| & \text{if } \Delta x_{ij} > \Delta y_{ij} \wedge N(i,j) \\ 0 & \text{otherwise} \end{cases}$$

(5)

In contrast, in FSA we have [7, 13]:

$$\delta_{ij}^{x} = \begin{cases} (w_i + w_j)/2 - |x_i^0 - x_j^0| & if \quad \Delta x_{ij} \le \Delta y_{ij} \wedge N(i,j) \\ \phi_{ij}^{x} & if \quad \Delta x_{ij} > \Delta y_{ij} \wedge N(i,j) \wedge y_i^0 \ne y_j^0 \\ 0 & otherwise \end{cases}$$

$$\delta_{ij}^{y} = \begin{cases} (h_i + h_j)/2 - |y_i^0 - y_j^0| & if \quad \Delta x_{ij} > \Delta y_{ij} \wedge N(i,j) \\ \phi_{ij}^{y} & if \quad \Delta x_{ij} \le \Delta y_{ij} \wedge N(i,j) \wedge x_i^0 \ne x_j^0 \\ 0 & otherwise \end{cases}$$

(6)

where

$$\phi_{ij}^{x} = (1 + \frac{h_i + h_j}{2|y_i^0 - y_j^0|}) \, | \, x_i^0 - x_j^0 \, |$$

and

$$\phi_{ij}^{y} = (1 + \frac{w_i + w_j}{2|x_i^0 - x_j^0|}) \, | \, y_i^0 - y_j^0 \, |$$

Comparing (5) to (6), it is clear that for removing two overlapping nodes *i* and *j,* an applied force δ_{ij}^{x} (or δ_{ij}^{y}) in FTA is not greater than $\sqrt{(\delta_{ij}^{x})^2 + (\phi_{ij}^{y})^2}$ (or $\sqrt{(\delta_{ij}^{y})^2 + (\phi_{ij}^{x})^2}$) in FSA. This means that the force in FTA is the minimum projection of the corresponding force in FSA. Only in the case of two overlapping nodes with the same *x* or *y* centre coordinates that the two forces in FTA and FSA are equal. □

Lemma 2 *Given that an applied force is transferable, the sum of the transferred forces on the nodes in FTA is not greater than those in FSA.*

Proof. Suppose that all nodes in a graph are sorted in increasing order by the *x* coordinates of their upper left corners, and that the *seed node* is the leftmost node in *V,* i.e., $s = v_1$. In general, the sum of adjusted distances in FTA and FSA is given by

$$f(v_1) = \sum_{i=1}^{|V|-1} (\sum_{j=i+1}^{|V|} \Delta x_{ij} + \sum_{j=i+1}^{|V|} \Delta y_{ij}) \qquad (7)$$

where Δx_{ij} and Δy_{ij} are the orthogonally moved distances of node *j* caused by separating it from its left overlapping node *i*. Similar to the proof in Theorem 3 , we have

$$f^{FTA}(v_1) = \sum_{i=1}^{|V|-1} \{ m | TRNN(i) | \Delta x_{ij} + (|TRNN(i)| - m) \Delta y_{ij} \} \qquad (8)$$

where $j = \min_{j' \in RNN(i)} \{ j' \}$. That is, node *j* is the first node on the right that overlaps with node *i*.

In contrast, all nodes to the right of node *i* within FSA, which have |*V*|-*i* nodes in total, move at both Δx_{ij} (or Δy_{ij}) and ϕ_{ij}^{y} (or ϕ_{ij}^{x}) in the *x* and *y* directions. The entire adjustment for relocating node *j* accumulates to (|*V*|-*i*) Δx_{ij} and (|*V*|-*i*) Δy_{ij}. According to (6) and (7), we have:

$$f^{FSA}(v_i) = \sum_{i=1}^{|V|-1} \{ m(|V| - i) \Delta x_{ij} + m(|V| - i) \phi_{ij}^{y}$$

$$+(|TRNN(i)|-m)(|V|-i)\Delta x_{ij} \quad (9)$$
$$+(|TRNN(i)|-m)(|V|-i)\phi_{ij}^{x}\}$$

where $j = \min_{j' \in RNN(i)} \{j'\}$.

It is obvious that $|TRNN(i)| \leq |V|-i$ holds. Comparing (8) to (9), we have $f^{FTA}(v_i) \leq f^{FSA}(v_i)$. □ Intuitively, staring from the same node with the different applied forces, FTA transfers the smaller force only to a group of overlapping nodes, while FSA transfers the bigger one to all the nodes on the right of this node.

Theorem 5 *Given a graph layout with overlapping nodes, the adjusted graph layout by FTA is more compact than one by FSA.*

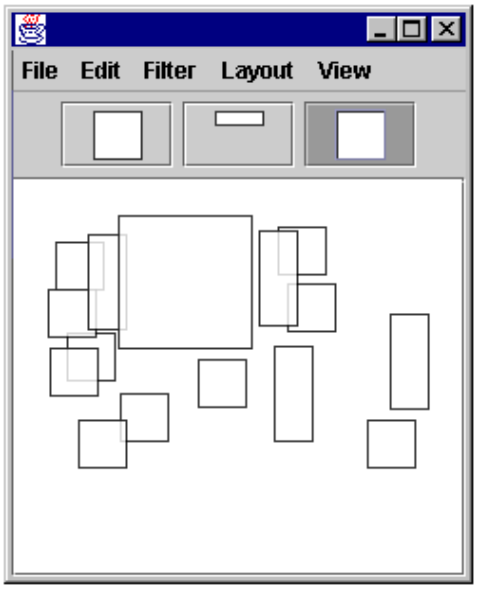
Proof. We use the *cost function* of Definition 5 to show the above result. The smaller the value of the *cost function*, the more compact the layout. In other words, we need to show $f_{cost}^{FTA}(s) \leq f_{cost}^{FSA}(s)$. The *cost function* is determined by both the force applied on a node and the number of nodes affected by this transferred force. From Lemmas 1 and 2, it is easy to see the result.

In fact, we have $f_{cost}^{FTA}(s) < f_{cost}^{FSA}(v_1)$ in most cases. Only in special cases such as two overlapping nodes with the same *x* or *y* centre coordinates does the equation $f_{cost}^{FTA}(s) = f_{cost}^{FSA}(s)$ hold.

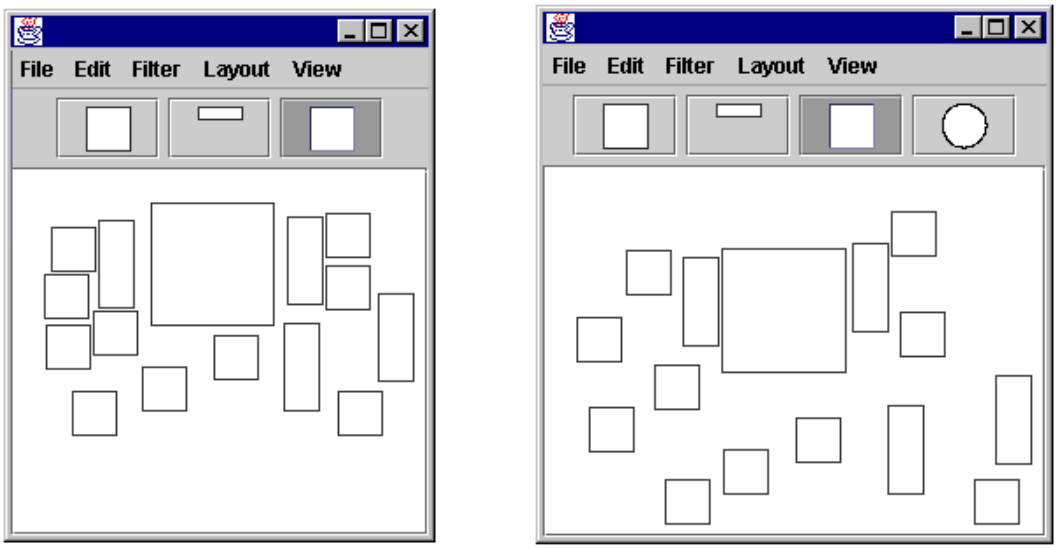
7. Experimental Evaluation

In this section, we report the experimental results using FTA and FSA.

Both FTA and FSA have been implemented in the prototype called PGD using the Java programming language. In the following we compare FTA to FSA by applying them to a set of arbitrary overlapping graphs, some of which are similar to those in [5]. First of all, a typical example using FTA and FSA is shown in Figure 6.



(a) An overlapping graph

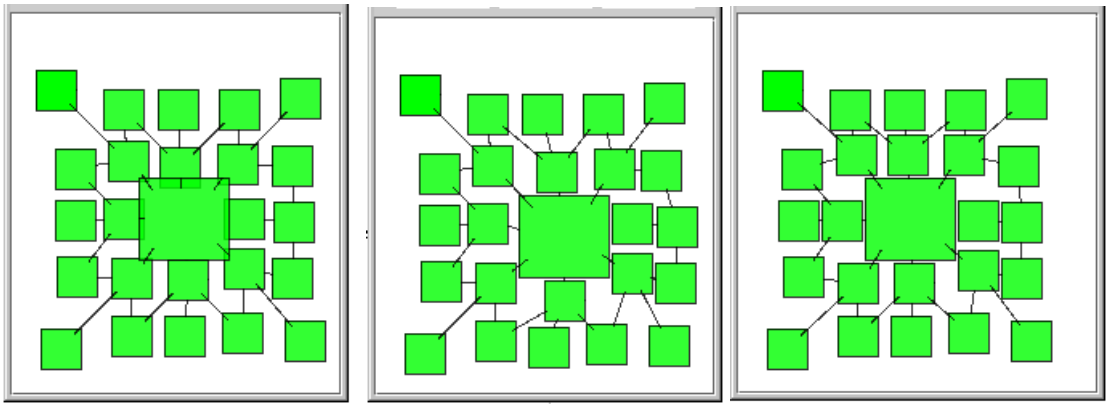


(b) Layout adjusted by FTA (c) Layout adjusted by FSA

Fig.6. An example of an overlapping graph adjusted by FTA and FSA

Figure 7 illustrates different resulting layouts for the same graph by FTA with various the *seed nodes*. In particular, Figure 7 (b) uses the leftmost node while Figure 7 (c) selects the centre node as the *seed node*s. In order to find the best candidate for a *seed node*, we should select the one that minimizes the values of the *cost function*.

Within our prototype system, either users are alternatively allowed to select the *seed node* for an adjustment, or the system automatically detects it using the heuristic presented in Section 3.3.

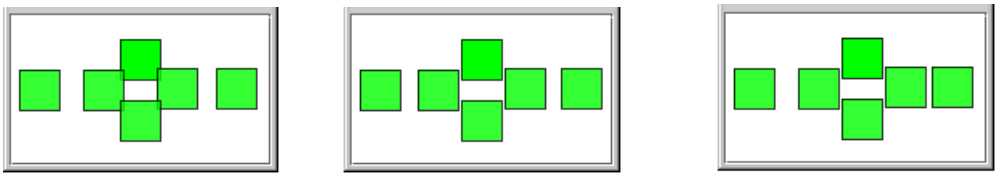


(a) Graph 1 (b) Layout by FTA with $s = v_1$ (c) Layout by FTA with s = centre node

Fig.7. Initial and adjustment layouts for Graph 1

When selecting the central position as the dummy *seed node* for the graph in Figure 8 (a), for example, Figure 8 (b) shows the layout result by FTA. This layout is better than that shown in Figure 8 (c) also by FTA but with the leftmost node as the *seed node*. This is because only the two nodes in

Figure 8 (b) had been moved to the left or right. In contrast, three nodes in Figure 8 (c) had been repositioned to the right. More importantly, Figure 8 (b) preserves the symmetry of the original layout.



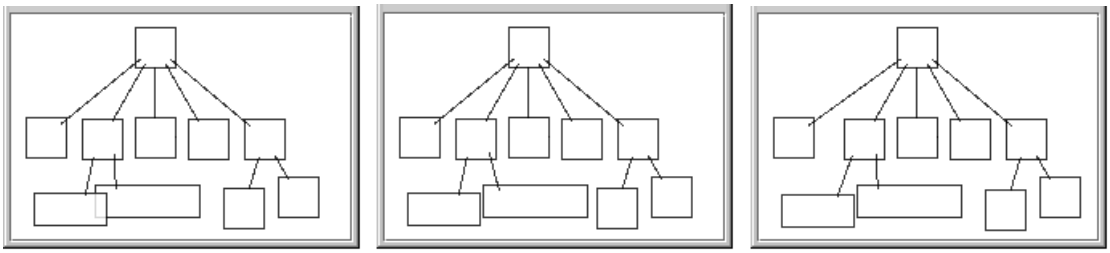
(a) Graph 2 (b) (c)

(b) Layout by FTA with a virtual node in the centre

(c) Layout by FTA with $s = v_1$

Fig.8. Initial and adjustment layouts for Graph 2

Figure 9 (b) shows that an applied force stops transferring from one node to another by using FTA. The reason for this is that they are not *transfer neighbor nodes*. Two overlapping nodes are located at the bottom level of Graph 3, as shown in Figure 9(a). When removing the overlap, the applied force on the second node cannot pass on to other following nodes on the right direction. It is for this reason that makes the layout more compact by FTA than by FSA in this instance.

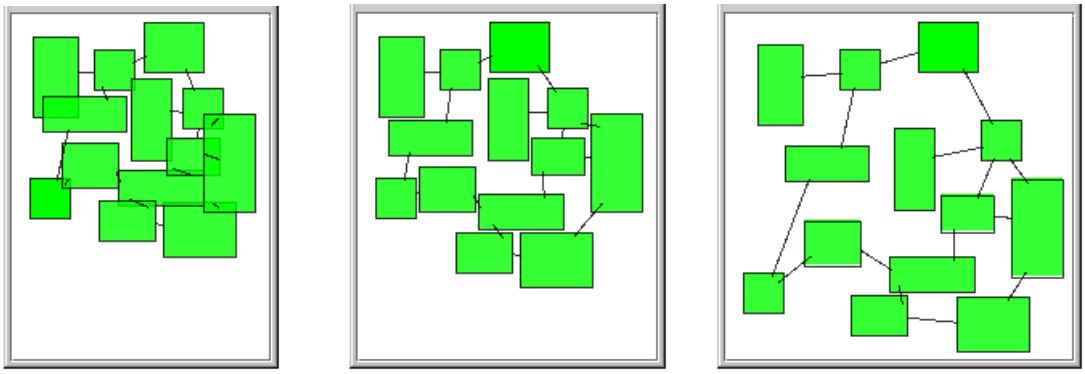


(a) Graph 3 (b) Layout with FTA (c) Layout with FSA

Fig.9. Initial and adjustment layouts for Graph 3

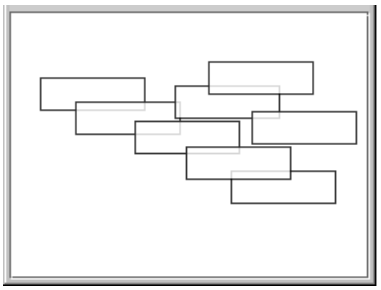
Figures 10, 11 and 12 show three adjustment examples by applying both algorithms to Graphs 4, 5 and 6, respectively.

These layouts adjusted by FTA are obviously more compact than those by FSA. Furthermore, their occupying areas by FTA are smaller than those by FSA.

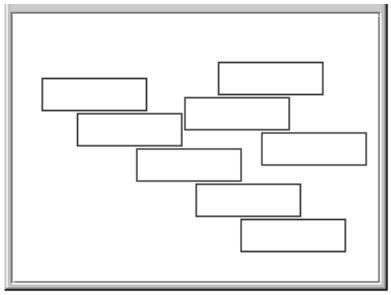


(a) Graph 4 (b) Layout with FTA (c) Layout with FSA

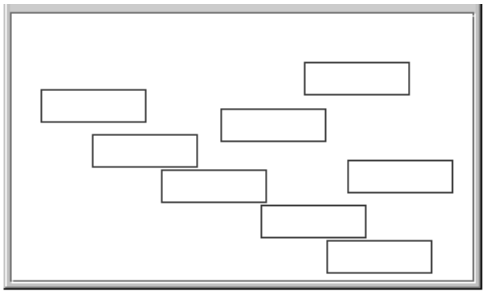
Fig.10. Initial and adjustment layouts for Graph 4



(a) Graph 5

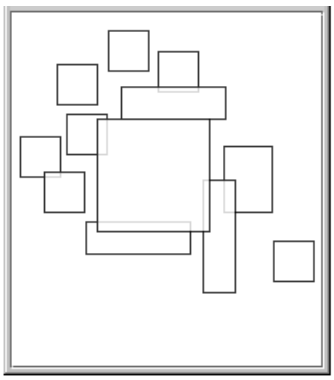


(b) Layout with FTA

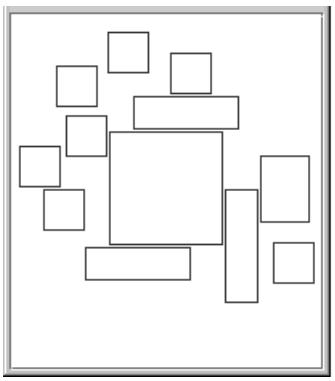


(c) Layout with FSA

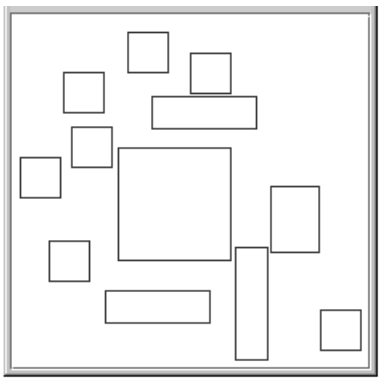
Fig.11. Initial and adjustment layouts for Graph 5



(a) Graph 6



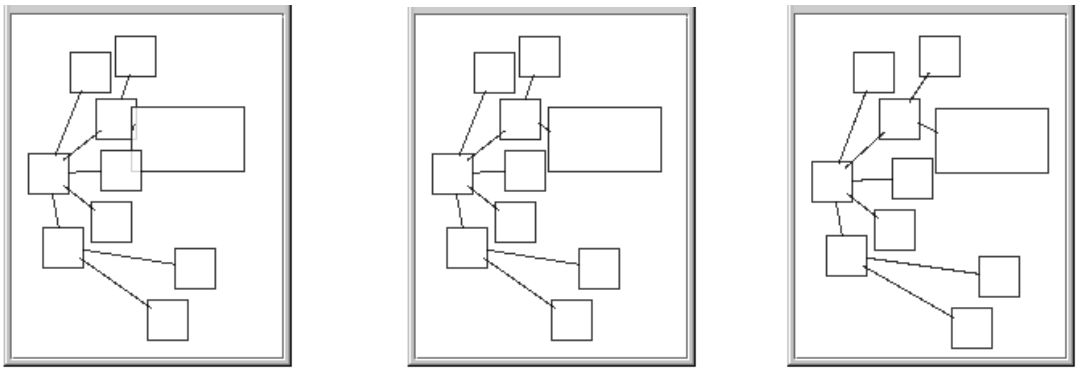
(b) Layout with FTA



(c) Layout with FSA

Fig.12. Initial and adjustment layouts for Graph 6

The biggest node in Graph 7 shown in Figure 13(a) overlaps with two other nodes so that FTA moves only this node as shown in Figure13 (b). FSA, however, repositions all the nodes to the right of the first node that is overlapping with the biggest one.

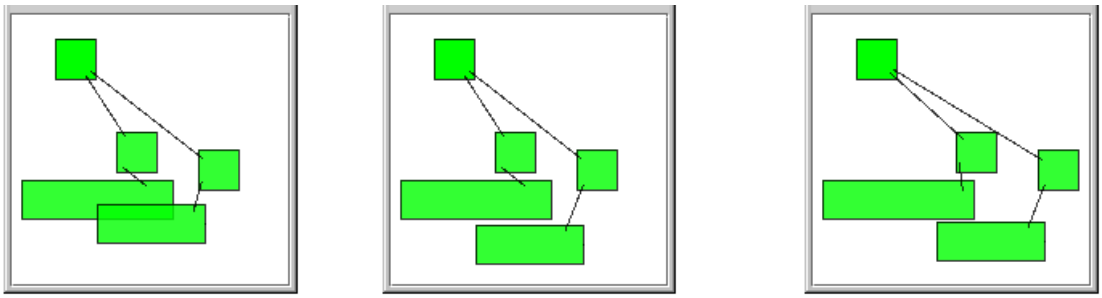


(a) Graph 7 (b) Layout with FTA (c) Layout with FSA

Fig.13. Initial and adjustment layouts for Graph 7

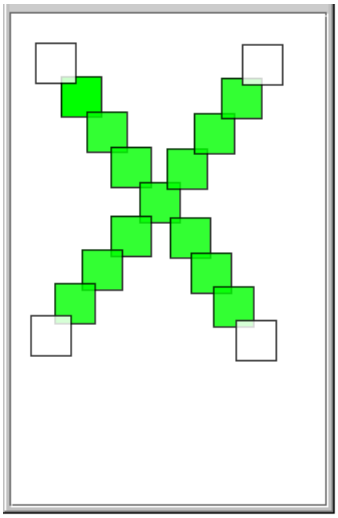
Graph 8 in Figure 14 exemplifies that a layout adjustment may introduce edge-node intersections using FSA, whereas FTA does not produce edge-node intersections for this graph. In some cases, FTA, however, may also generate edge-node intersections. Fortunately, this problem is readily solved by the approach in [1].

Graph 9 in Figure15 is an X-shaped graph with the symmetry about both the *x*- and *y*-axis. Both the layouts by FTA and FSA are good, but the gaps between the nodes in Figure 15(c) are a little bigger than those in Figure 15(b).

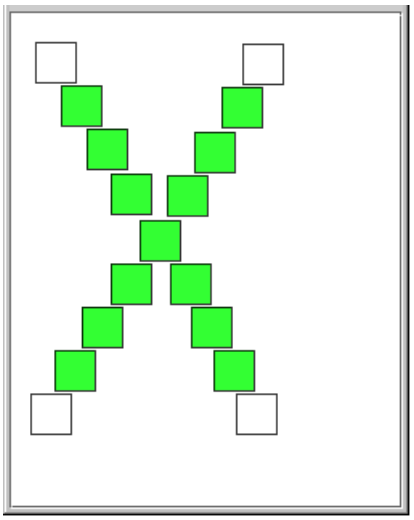


(a) Graph 8 (b) Layout with FTA (c) Layout with FSA

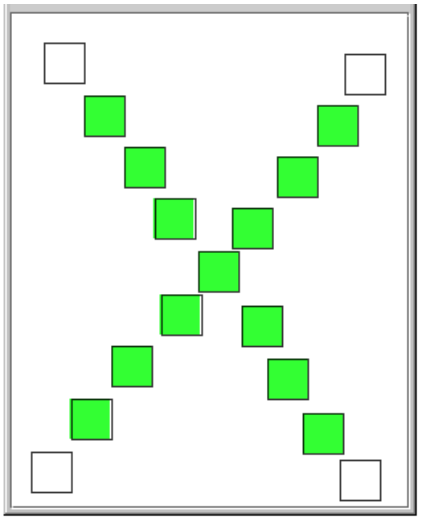
Fig.14. Initial and adjustment layouts for Graph 8



(a) Graph 9



(b) Layout with FTA



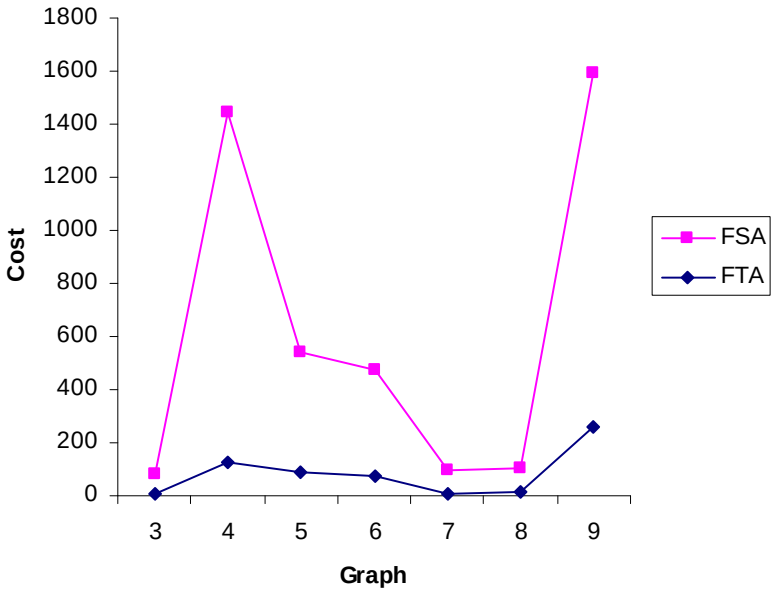
(c) Layout with FSA

Fig.15. Initial and adjustment layouts for Graph 9

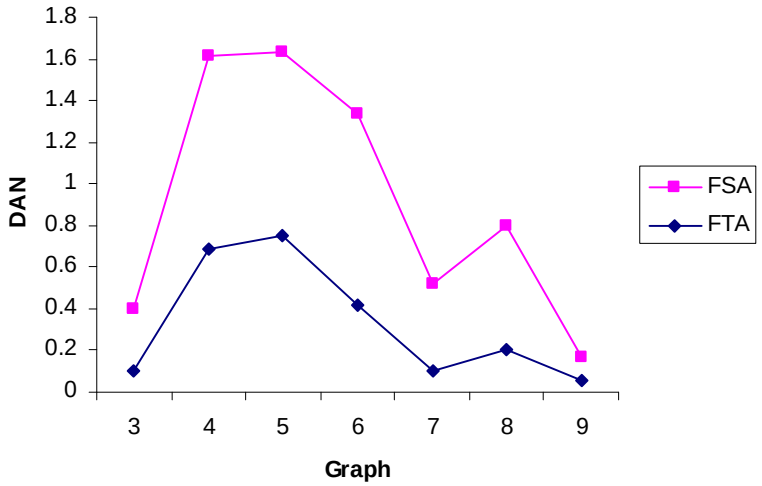
In summary, we conclude by the above examples that the layouts adjusted by FTA are more compact than those by FSA. Furthermore, we also precisely measure the qualities of adjusted layouts using the *cost function* and others measures previously defined. The primary measures such as the adjustment costs and degree of adjusted nodes are reported in Table 1 for the previous example layouts. We also illustrate them in Figure 16. Note that the leftmost nodes (v_1) of those graphs were chosen as the *seed nodes* for the calculations of the *cost functions*.

Graph(# nodes)		3(10)	4(13)	5(8)	6(12)	7(10)	8(5)	9(18)
FTA	Cost function	7	127	86	76	4	18	258
	degree of nodes adjusted	0.10	0.69	0.75	0.42	0.10	0.20	0.06
FSA	Cost function	74	1318	457	399	89	85	1335
	degree of nodes adjusted	0.30	0.92	0.88	0.92	0.42	0.60	0.11

Table 1. Value of Cost Function $f(v_1)$



(a) Adjustment cost for the graphs



(b) Degree of adjusted nodes (DAN)

Fig.16. Comparison of values of Cost Function and the degree of adjusted nodes for Graph 3 -Graph 9

More comparison experiment results are reported in Figs. 17 and 18, with respect to the adjustment cost and running time.

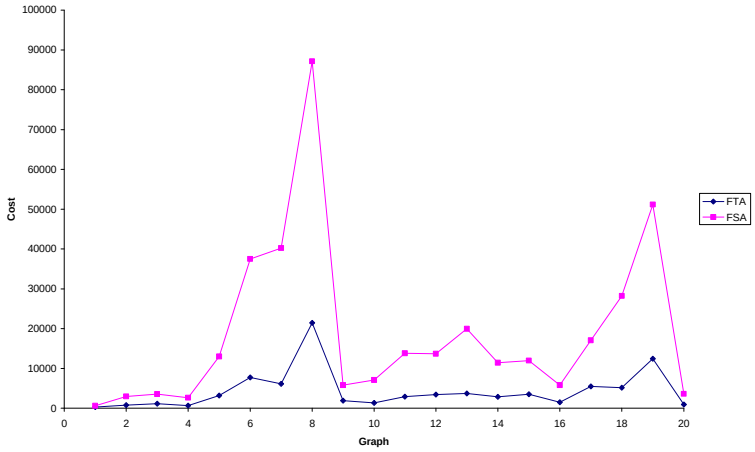


Fig.17. Comparison of values of Cost Function for more graphs

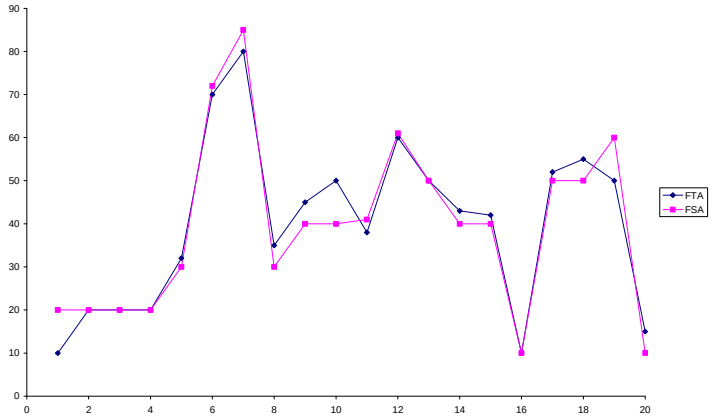
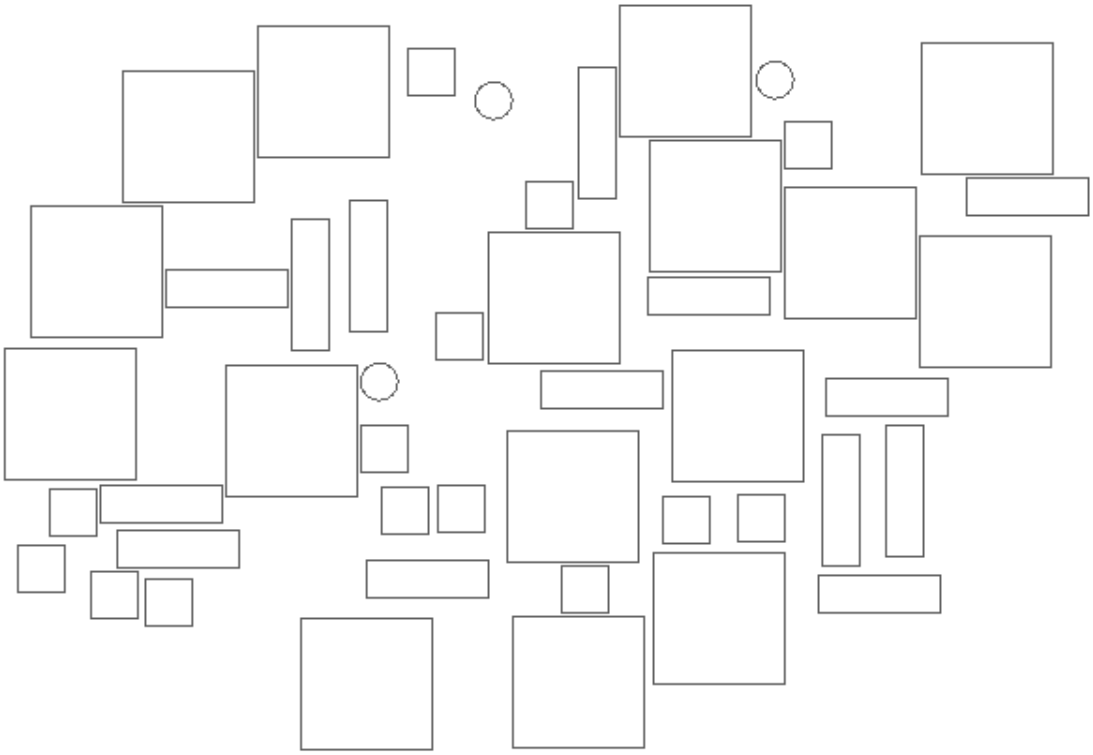


Fig.18. Comparison of the running times for the graphs in Fig. 16

The more complex example layouts are shown in Figs 19 and 20. As the authors of [5] pointed out, the algorithms are ranked in increasing order of running time as uniform scaling, force-scan, and constrained optimization. Uniform scaling and force-scan run considerably faster than the constrained optimization. The time complexity of FTA is the same as that of FSA. Interestingly, the lagrangian method [5] uses the same minimum force as that in FTA to separate two overlapping nodes. As opposed to an iterative adjustment, FTA, however, uses four scans to eliminate all overlaps, in order to reduce the running time. FTA not only runs faster, but also makes the adjusted layout compact.

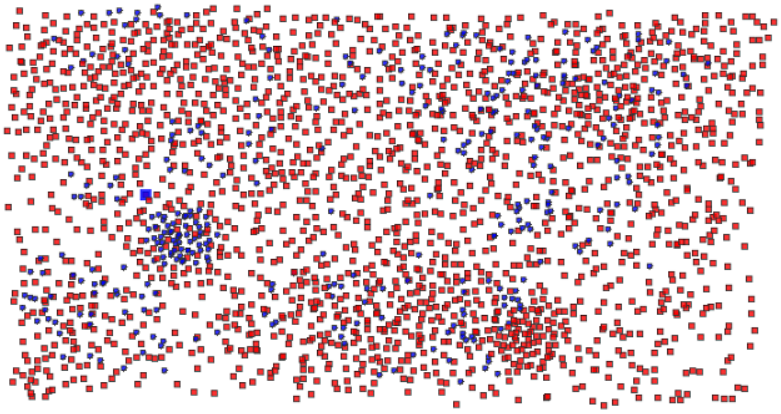
(a) A graph with many overlapping nodes



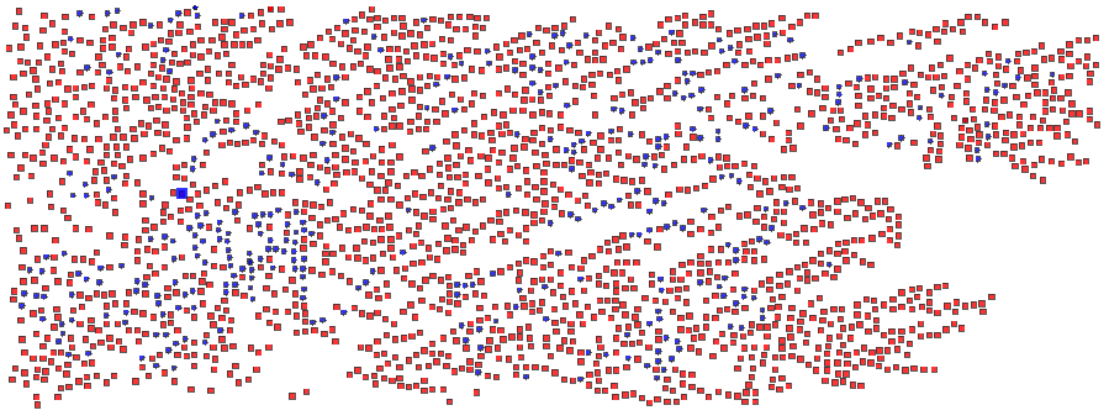
(b) Layout by FTA

(c) Layout by FSA

Fig.19. A comparison example of FTA and FSA



(a) A graph with many overlapping nodes



(b) Layout by FTA

Fig.20. An example of overlapping layout adjusted by FTA

8. Applications

Current graph visualization systems usually employ static layout algorithms. Although the static layout is useful in many applications [24], the dynamic layout is essential in the following situations. Our algorithm is capable of being applied to all these situations effectively.

Incremental navigation of large graphs: There is considerable interest in visualizing large graphs, such as computer networks and the WWW. Consider, for example, the incremental navigation of a Web graph, users' mental map of the layout should be preserved after changing the graph views from one to another. Otherwise the users have to spend extra cognitive efforts being familiar with those big changes. This requires that the alteration of an existing layout should be made as little as possible.

Viewing dynamic data: Graphs are applied to visualize dynamic systems in many settings. Dynamic graph visualization could be an appropriate way of showing the history of the data. Relative stability is necessary for such visualizations to reveal what is actually changing in the structure of the underlying data.

Interactive graph editing: Graph visualization systems usually support interactive editing. Our dynamic adjustment algorithm could improve the interface usability of such systems.

9. Conclusion

Graphs where the nodes include their labels are often used in applications. A typical example of such graphs is UML diagrams in CASE tools. In order to make the label of each node readable, it requires that drawings of these graphs should have no node-overlapping. This is also crucial in a dynamic environment of graph visualization. In this paper, we have presented the Force-Transfer algorithm that removes node-overlaps. The proposed approach employs a heuristic method to approximate a global optimal adjustment with the local minimal movement. Scanning from particular nodes called the seed nodes in a graph, this approach orthogonally transfers the minimum forces to only those nodes that

recursively overlap with the nodes from where the forces start. We compare the Force-Transfer to the Force-Scan algorithm, demonstrating that our approach is able to generate better results. It is concluded that FTA achieves an adjusted layout with a good trade-off between a compact layout and running time. Our approach can be applied not only to adjust graph layouts, but also to position non-overlapping objects such as windows and labels, by specifying a minimum gap between the objects.

References

[1] W. Lai and P. Eades, Removing edge-node intersections in drawings of graphs, Information Processing Letters, 81(2002) 105-110.

[2] W. He and K. Marriott, Constrained graph layout, Constraints, 3 (4) (1998) 289-314.

[3] G. D. Battista, P. Eades, R. Amassia and I. Tollis, Graph drawing: Algorithms for the Visualization of Graphs, Prentice Hall, 1998.

[4] P. Eades and W. Lai, Algorithms for disjoint node images, in: Proc. 15th Australian Computer Science Conference, Hobart, 1992, pp.253-265.

[5] K. Marriott, P. Stuckey, V. Tam and W. He, Removing node overlapping in graph layout using constrained optimization, Constraints, (2003)1-31.

[6] W. He and K. Marriott, Removing node overlapping using constrained optimization, in Proc. 21st Australian Computer Science Conference, Perth, 1998, pp.169-180.

[7] W. Lai, Building interactive diagram applications. PhD thesis, University of Newcastle, 1993.

[8] R. Davidson and D. Harel, Drawing graphs nicely using simulated annealing, ACM Transactions on Graphics, 15(4) (1996) 301-331.

[9] B. P. Buckles and F. E. Petry, Genetic algorithms, Computer Society Press, Los Alamitos, CA., 1992.

[10] F. Glover, Tabu search, Part I, ORSA Journal on Computing, 1(3) (1989) 190-206.

[11] F. Glover, Tabu search, Part II, ORSA Journal on Computing, 2(1) (1989) 4-32.

[12] P. Eades, W. Lai, K. Misue and K. Sugiyama, Layout adjustment and the mental map, Journal of Visual Languages Computer, 6 (1995)183-210.

[13] H. Kunihiko, I. Michilo, M. Toshimitsu and F. Hideo, A layout adjustment problem for disjoint rectangles preserving orthogonal order, Systems and Computers in Japan, 33(2) (2002) 31-42.

[14] K. A. Lyons, H. Meijer, and D. Rappaport, Algorithms for cluster busting in anchored graph drawing, Journal of Graph Algorithms and Applications, 2(1) (1998) 1–24.

[15] M. D. Story and A. H. Müller, Graph layout adjustment strategies, in: Symposium on Graph Drawing, GD'95, LNCS 1027, Springer-Verlag, 1995, pp. 487-499.

[16] M-F. Natasa and S. Ralph, SmartView: Flexible viewing of Web page contents, in: Proc. 11th International World Wide Web Conference, Hawaii, USA, 2002.

[17] D. Harel and Y. Koren, Drawing graphs with non-uniform Vertices, in: Proc. Working Conference on Advanced Visual Interfaces (AVI'02), ACM Press, 2002, pp.157-166.

[18] M.G. Kendall, Rank correlation methods, Griffin, London, UK, 1970.

[19] C. Dwork, R. Kumar, M. Naor and D. Sivakumar, Rank aggregation methods for the Web, in: Proc. 10th International World Wide Web Conference, Hong Kong, 2001.

[20] X.D. Huang Filtering, clustering and dynamic layout for graph visualization, Ph.D. thesis, Swinburne University of Technology, 2004.

[21] P. Eades, W. Lai, K. Misue, and K.Sugiyama, Preserving the mental map of a diagram. Technical report IIAS-RR-91-16E, International Institute for Advanced Study of Social Information Science, Fujitsu Laboratories Ltd, 1991.

[22] X. Huang, W. Lai, Clustering graphs for visualization via node similarities, Journal of Visual Languages and Computing, 17 (2006) 225-253.

[23] U. Dogrusoz， Two-dimensional packing algorithms for layout of disconnected graphs， Information Sciences 143 (2002) 147-158.

[24] B. Genc, U. Dogrusoz, layout algorithm for signaling pathways, Information Sciences, 176 (2006) 135-149.