ISSN 1000-9825, CODEN RUXUEW
Journal of Software, Vol.19, No.8, August 2008, pp.1920−1932
DOI: 10.3724/SP.J.1001.2008.01920

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

一种用于Marching-Graph图形绘制的快速收敛布局算法*

全 武[1+], 黄茂林[2]

[1](School of Information Technologies, University of Sydney, NSW 2006, Australia)

[2](Faculty of Information Technology, University of Technology, Sydney, NSW 2007, Australia)

Fast Convergence Layout Algorithm for Drawing Graphs in Marching-Graph

QUAN Wu[1+], HUANG Mao-Lin[2]

[1](School of Information Technologies, University of Sydney, NSW 2006, Australia)

[2](Faculty of Information Technology, University of Technology, Sydney, NSW 2007, Australia)

+ Corresponding author: E-mail: kevin.wuquan@gmail.com

Quan W, Huang ML. Fast convergence layout algorithm for drawing graphs in Marching-Graph. *Journal of Software*, 2008,19(8):1920−1932. http://www.jos.org.cn/1000-9825/19/1920.htm

Abstract: Marching-Graph is a new visualization that integrates the graph metaphor and the spatial metaphor into a single visualization. It provides users with highly interactive maps for accessing the logical structures of information that has the geographical attributes. Instead of presenting known facts onto maps, it provides a mechanism for users to visually analyze and seek unknown knowledge through effective human-map interaction and navigation across different spaces. However, the traditional force-directed layout algorithms are very slow in reaching an equilibrium configuration of forces. They usually spend tens of seconds making the layout of a graph converge. Thus, those force-directed layout algorithms can not satisfy the requirement for drawing a sequence of graphs rapidly, while the users are quickly marching through the geographic regions. This paper proposes a fast convergence layout method that speeds up the interaction time while users are progressively exploring a sequence of graphs through a series of force-directed layouts in Marching-Graph. It essentially combines a radial tree drawing method and a force-directed graph drawing method to achieve the fast convergence of energy minimization.

Key words: graph and network visualization; graph drawing; information analytic; information visualization system; visual design

摘 要: Marching-Graph 是一种将图形隐喻技术和空间隐喻技术集成为一体的新的可视化方法.它为用户提供了高度可交互性地图,使用户可访问那些具有地理属性的信息的逻辑结构.它通过有效的人图交互和跨空间浏览为用户提供了一种可视分析和挖掘未知信息的机制,而不是将已知的信息呈现在地图上.然而,传统的力导向布局算法在达到力量均衡配置方面非常慢.为使一个图形布局收敛,它们通常需花费几十秒的时间.因此,当用户快速行进于地理区间时,那些力导向布局算法就不能满足快速绘制一系列图形的要求.提出了一种快速收敛布局方法,当用户在Marching-Graph 中通过力导向布局逐步探究一系列图形时,它可以加速交互时间.通过结合辐射树绘图技术和力导向图形绘制方法来取得能量最小化的快速收敛.

关键词: 图形和网络可视化;图形绘制;信息分析;信息可视化系统;可视设计

* Received 2008-01-08; Accepted 2008-04-18

中图法分类号: TP391 文献标识码: A

1 Introduction

Many spatially-related information systems such as *Geographic Information Systems* (*GIS*) provide predefined functions allowing the automatic generation of graphic maps to be used for viewing, exploring and manipulating the data sets which contain spatial attributes. These maps usually employ interactive graphic techniques to enable their interactive capacities. We call such maps the “spatial frame of references”. Displaying a familiar geometric map as the background, on which spatial related data items are positioned, aims at amplifying the human cognition process. In the design of user interfaces, while most existing spatial data mining methods consider the use of a *spatial metaphor* (the graphic map) to amplify the cognition process in navigating the geographic data, they do not care about the human factors involved in viewing and exploring mined data structures (such as a variety of output patterns). In most cases, they usually use icon positioning, colour-coding and simple geometric symbols to visualize the statistical results that associate with particular geographic regions. Alternatively, only a textual interface is provided for users to exploit the underlying patterns. However, in many cases, the output patterns are complex, and sometimes include multiple types of relationships. Much of these data is of multivariate nature, featuring from two to hundreds of attributes. Therefore, a simple statistical or textual output cannot satisfy the need to unveil the potential patterns. Hence there is an urgent need to find new efficient user interfaces that are human-centered.

Visual approaches to data exploration and analysis have arisen in many different areas and now been effectively applied. The database community has coined the phrase “visual data mining” to describe their efforts, while the terms “exploratory visual analysis” or “exploratory data analysis” are more common within the statistics community. During the above visual exploratory process, graph visualization has been used as one of the primary vehicles for the viewing, navigation and manipulation tasks through visual transformations.

Graph visualization uses a *graph metaphor* to map the extracted patterns onto one or more graphs, and then applies the graphic rendering techniques to converting graphs into the visual structures for display. Graph visualization concentrates on the automatic positioning of vertices and edges on the screen. A good layout of graph allows users to understand data more easily. Moreover, using an interactive graphics will facilitate users to explore the output patterns and discover the underlying information. Typical existing applications of visual spatial data mining[1−3] combine the information visualization with the thematic maps for the exploration of both geographic locations and analytical patterns.

However, most visual spatial data mining methods divide the spatial metaphor and the graph metaphor into different visual objects (such as windows, widgets) for display (see Fig.1). The divided displaying approaches do not explicitly represent the correlation between any two visual metaphors, which would possibly cause the following problems:

- Lack of the overall context of visualization: If the entire visualization is separated into several pieces, then the visualization will lose its completeness. This will greatly reduce the readability and understandability of the overall visual structure. Shifting the focus across different pieces of visualization will cost users extra cognitive effort.
- Inconvenience of cross navigation: It is inconvenient for users to navigate across different visual metaphors that are located in different pieces of visualization.

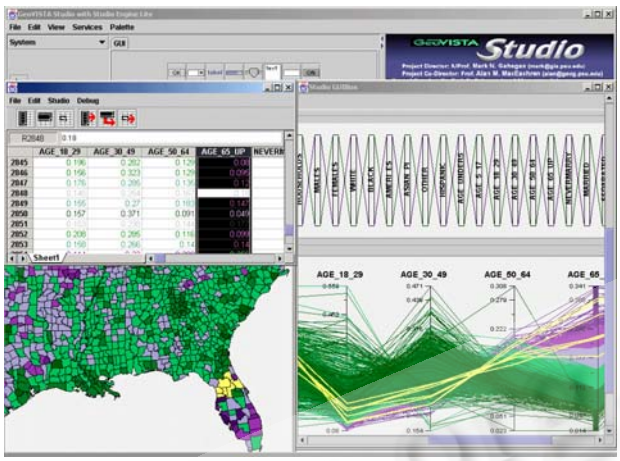


Fig.1 A visualization screen collected from GroVISTA that divides the spatial visual metaphor and other analytical visualizations into different windows

To solve the problems mentioned above, Kreuseler and Schumann proposed a new approach called Marching Sphere[4] in 2002, which integrated the spatial metaphor and the graph metaphor into a single visualization. In order to implement the combination of both embodiments of graphs and spatial frame of reference, Marching Sphere renders the two-dimensional map onto a virtual 3D scene. It represents the complex relational information structures by using 3D graphs that are just located above the areas where the interactive map is displayed. This approach achieves the unique mapping between graph-structured information and spatial references. While Marching Sphere is a good solution to combining two visual metaphors in visualizing complex information, it has the following weaknesses: 1) the implementation of 3D graphics is computationally expensive on desktop PCs; 2) it is practically difficult to navigate across 2D and 3D visual metaphors.

Therefore, we proposed a new fused visual metaphor, called *Marching-Graph*[5] in 2005, which replaced 3D spheres with 2D graphs, for the exploration of spatially related data. We drew the abstract data structure (as a graph) in a 2D circular geometric area above a focused geospatial region linking to that area. At any instant of time, we only showed the detailed layout of one particular graph above a geospatial area of interest to present the relational data associated with this area. It allowed the user to "march" on the whole geospatial map region by region, through the opening and displaying of a series of graphs. Moreover, it provided a variety of 2D interactive methods (include zooming, filtering, highlighting, coloring etc.) allowing users to interactively read and interpret the spatially referenced data through the rich graphical representations (see Fig.2). Since both of the spatial metaphor and the graph metaphor were represented in 2D graphical spaces, this approach was much easier and straightforward to be implemented and used than the *Marching Sphere* approach.

Force-directed layout algorithms[6–9] are very popular in graph visualization due to their flexibility, ease of implementation, and the aesthetically pleasant drawings they produce. These algorithms always seek an equilibrium configuration of forces to reach a locally minimized energy of layout. This is usually achieved by using a simple "follow your nose" algorithm that smoothly moves all nodes from their initial positions to the final positions through iterations towards an equilibrium configuration of forces. In general, the "follow your nose" algorithm works as follows, with nodes initially placed at random locations of a geometrical plane. At each iteration (that could be implemented as an animation loop), the total force applied on each node is computed, and each node is moved in the direction of the force by a small amount, thus maintaining the human mental map.

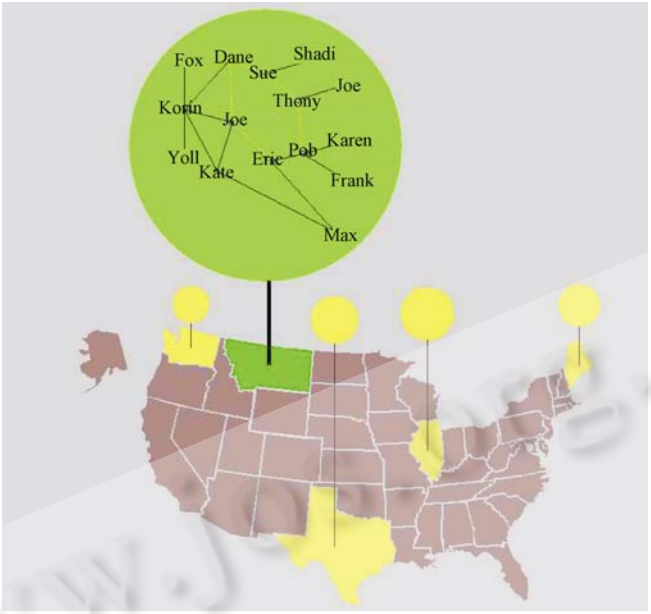


Fig.2 An illustration of the *Marching-Graph* that consists of several graphs with one opened in a green circular region and others closed

However, as a tradeoff of such animated positioning, most of force-directed layout algorithms are slow in reaching an equilibrium configuration of forces. In other words, the convergence time spent in moving nodes from their initial positions to their final positions is quite long. It usually takes tens of seconds to complete an equilibrium configuration of forces to reach the final layout.

In many real world applications when users want to use graph visualization to build their interactive user interfaces which involve the changes of navigational views during the exploration of data, the convergence time of force-directed drawing algorithms becomes a crucial issue affecting the efficiency (or the speed) of the navigation. For example, in *Marching-Graph*, a sequence of graphs $G_1,G_2,\ldots,G_n$ associated with the corresponding geographical regions need to be drawn and displayed when users "marching" through the map (see Figs.2 and 3). This requires the layouts $D(G_1),D(G_2),\ldots,D(G_n)$ to be computed as quickly as possible, to respond to the users' interaction in real-time. If we use a force-directed algorithm to draw a sequence of graphs, then each time when opening a new graph, the convergence time spent in moving vertices of that new graph from their randomly generated

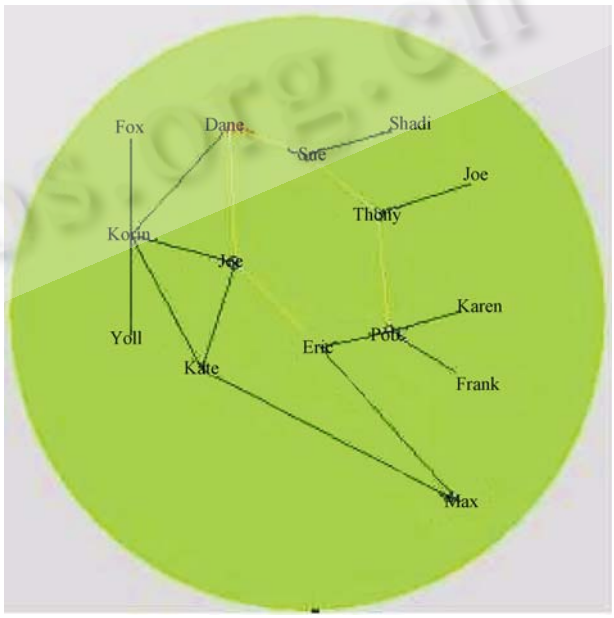


Fig.3 A graph layout produced by force-directed algorithm in *Marching-Graph*

positions to their final positions is significant. This will greatly slow down the real-time response during navigation.

In this paper, we propose a fast convergence method for drawing force-directed layout graphs. This algorithm is effective in speeding up the interaction time when users explore a series of graphs $G_1,G_2,\ldots,G_n$ by displaying a sequence of corresponding force-directed layouts $D(G_1),D(G_2),\ldots,D(G_n)$. We essentially combine a radial tree drawing method and a force-directed graph drawing method to achieve the fast convergence of energy minimization. The transformation of the radial drawing of a graph G to the final force-directed layout will not only greatly reduce the convergence time of the drawing, but also preserve the user's mental map when moving from one view to another.

2 Method

Assuming a connected and tree-like graph $G=\{V,E\}$, G contains a backbone tree $T'=\{r,V',E'\}$, where r is the root of T'; $V'=V$; and $E'\subseteq E$ is a set of tree edges. As a tree-like graph, G consists of a small set of non-tree edges E'', where $E''=E-E'$. Thus, graph G could be approximately defined as $G=\{T',E''\}$, where $E''\subset E$ is a non-tree edge set. The tree-like graphs widely exist in social networks which are examined in *Marching-Graph*. For example, the graphs with 129 and 160 vertices, which are used in the following experimental evaluation, are a friendship network and a network of websites respectively. We first extract the backbone tree $T'=\{r,V',E'\}$ from G. This can be done through a breadth-first search.

Since the radial drawing algorithm is straightforward and computationally inexpensive, we then use a radial drawing algorithm to draw T' as the backbone layout of graph G. Another reason that we choose the radial drawing method to create the initial layout of graphs is this method satisfies the domain requirements of our initial application: *Marching-Graph*. *Marching-Graph* uses a series of circular areas to display graphs. Therefore, the pattern of a series of concentric rings used in radial drawings is very close to the expected final layout of graphs in the *Marching-Graph* application.

Once the backbone layout of G is calculated, we simply draw all edges in E'' into the layout as a set of straight-lines linking between each pair of start-node and end-node, without changing any node positions predefined in the backbone layout.

In the final step, we apply a force-directed drawing algorithm to the above layout to replace the initial radial drawing of graph G. We aim to smooth the transition between two layouts when switching from the radial drawing to the final force-directed drawing of G.

After the switch, we could incrementally explore the graph G through the change of views by using an on-line animated force-directed graph drawing method[10]. The pipeline of this fast convergence layout algorithm is diagrammed in Fig.4. We now discuss these steps one by one in details.

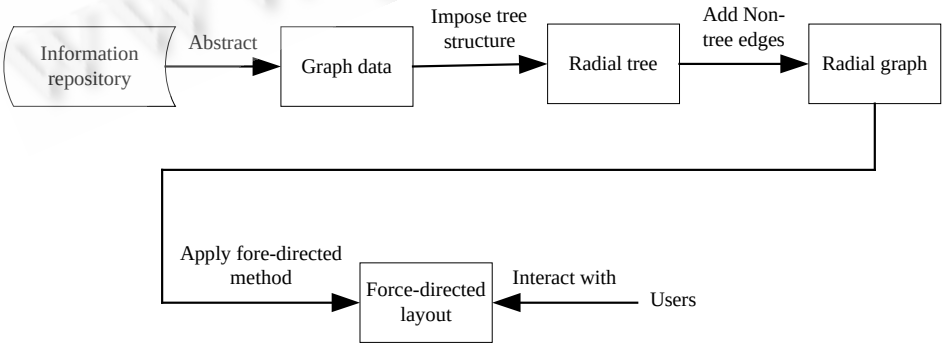


Fig.4 Pipeline of fast convergence layout algorithm

2.1 Data abstraction

The first step of visualization abstracts a graph G from a given spatial referenced information repository. The data will be stored in XML files in the XGMML-format. XGMML (eXtensible Graph Markup and Modelling Language) is an XML application based GML standard that is commonly used for graph description. This modelling language uses tags to describe *vertices* and *edges* of a graph. Using XGMML to describe graphs enables graph data to be exchanged between different authoring and browsing tools.

In many cases, the information repository we want to visualize could store mined data which may contain complex relational structures and from which we might abstract more than one graph. Furthermore, in most cases this information repository could be dynamic so that the content of the repository may update from time to time corresponding to the data mining process. Therefore, the use of an online visualization method as a part of the visual data process is more appropriate.

2.2 Imposing a tree structure

We assume that the given graph $G=(V,E)$ is connected and has a set of vertices $V=\{v_1,v_2,\ldots,v_n\}$ and a set of edges $E\subseteq V\times V$. To use the existing radial tree drawing algorithm to create the backbone layout of G, we have to extract a tree $T'=\{r,V',E'\}$ from G. This can be done through a breadth-first search to establish a new edge set E' in which each edge e_i is in tree T'.

In order to extract tree structures from arbitrary graphs, we use a series of adjacent link lists *AdjLinkedLists* to record each pair of connected vertices in graph G. We then choose a vertex $v_i\in V$ as the root r. Note that we usually choose a vertex v_i which has the highest degree as the root. Starting from this root vertex, we apply a breadth-first search algorithm to these *AdjLinkedLists*. After the search, we obtain a series of parent-child lists *ChildLinkedLists* which record the parent-child relationships of all vertices in tree $T'\subseteq G$ where *ChildLinkedLists*$\subseteq$*AdjLinkedLists*.

Then the parent-child relationships are recorded in the *ChildLinkedLists* which is the data structure of T'. The vertex neighboring relationships recorded in the *AdjLinkedLists* but not in the *ChildLinkedLists* are non-tree edges. The breadth-first search by using adjacent link lists can be finished in time $O(V+E)$.

For example, Fig.5 shows an instance of applying the above algorithm to a small graph G rooted at vertex "A". We can see that after the breadth-first search a tree T' is extracted from the graph G, where edges ($B1$, $B2$), ($B5$, $C1$), ($B2$, $C3$), ($B4$, $C5$), ($C7$, $C8$), coloured in light gray, are collected in the edge set E" as non-tree edges while the

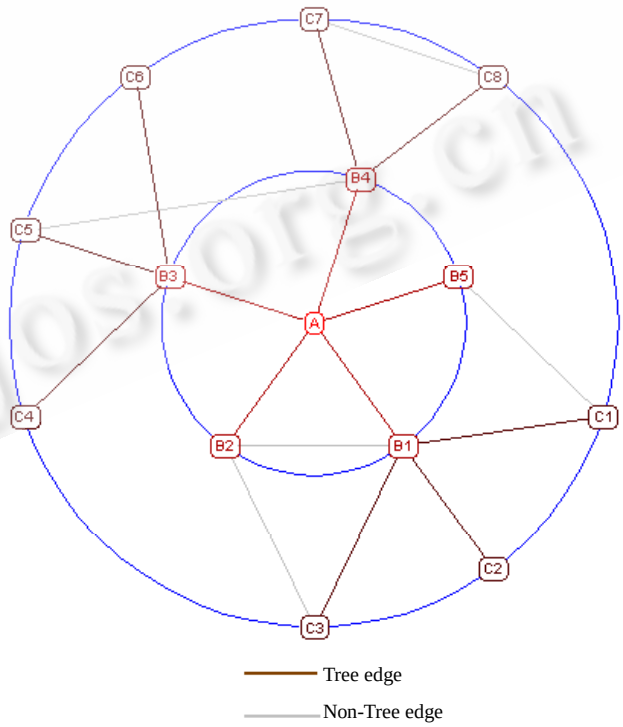


Fig.5 An example of the radial drawing of G rooted at vertex "A"

others are collected into E' as tree edges.

2.3 Radial layout of tree T'

Once the implicit tree structure is extracted, we can now apply the traditional radial tree drawing method to the tree T'. We place each vertex $v_i \in V$ on one of the sequential concentric circles $C_1, C_2, \ldots, C_k$ centred in the display medium, depending on the graph-theoretic distance $d(r, v_i)$. The algorithm for radial drawing is adapted from[11,12] and stated below:

```
Algorithm DrawSubTree (v, r_i, α_1, α_2) {
    // the point with polar coordinates
    ρ(v) = poloar(r_i, (α_1+α_2)/2);
    s=(α_2−α_1)/ w(v);
    α=α_1;

    Iterator itr = v.getChildren();
    While(itr.hasnext()) {
        //get vertex v's child u
        Node u = (Node) itr.next();
        DrawSubTree(u, ρ+δ, α, α+s*w(u));
        α = α + s*w(u);
    }
}
```

In the above algorithm, v is the root of a sub-tree of T' that is drawn in the current recursion; r_i is the radius of circle C_i on which the vertex v lies. $\rho(v)$ is the location of vertex v; α_1 and α_2 are the start and end angles respectively, which compose the annulus wedge in which the vertices of sub-tree rooted at v locate; $w(v)$ represents the number of leaves in the sub-tree rooted at vertex v. The δ is a constant denoting the distance between two consecutive concentric rings. We call the procedure *DrawSubTree*(r,0,0,2π) to draw the entire radial tree T' on the plane. The overall running time of this algorithm is $O(V)$. Figure 5 shows the example of such drawings. This radial drawing of T' is used as the backbone layout of G.

2.4 The radial drawing of complete G

Once the backbone layout of G is drawn, we could add edges in E'' to the layout without changing any vertex positions predefined in the backbone layout. In the example shown in Fig.5, we could simply add 5 non-tree edges ($B1$, $B2$), ($B5$, $C1$), ($B2$, $C3$), ($B4$, $C5$) and ($C7$, $C8$) to the layout.

As assumed at the beginning, we mainly deal with tree-like graphs that contain only a small number of non-tree edges. Therefore, the addition of several non-tree edges will not affect the quality of the layout significantly, in terms of the aesthetic rules and readability.

2.5 Switching to force-directed drawing

While the radial layout algorithm is good for drawing small static graphs in *Marching-Graph* application, it is unsuited for incremental drawing of large (or dynamic) graphs. For example, Yee, *et al*. proposed an animated exploration of radial layout graphs[13], which linearly interpolates the polar coordinates of the vertices during the transition of views. As a result of using that technique, the layout changes greatly when moving from an old focus vertex to a new one during the navigation of a graph. Thus, the broken connection between two layouts will

significantly increase the cognition overload for the user to understand the changes between views.

We therefore hand over the initial radial layout to a force-directed layout, whose effectiveness in view of transformation through its own animation loops and incremental explorations has been demonstrated in our initial experiments[10].

In the final step, we apply a force-directed drawing algorithm to the above layout to replace the initial radial drawing. Although, an improved force-directed layout algorithm has a running time equivalent to $O(V^2 \log V)$. The use of a radial drawing as the initial layout of the force-directed drawing algorithm can greatly reduce the number of iterations to make the drawing to convergent status. However, simply applying a force-directed method to a radial graph drawing could cause a highly disoriented rearrangement. To overcome the disorientation for the viewer, we enforce the following constraints when switching to the force-directed layout:

1) We use the length of each edge produced in the initial radial drawing as the resting length of its spring in the force-directed drawing.
2) Considering the repulsion force between two vertices as the inverse of their squared separation, the major repulsion force exerted on a vertex v comes from the other vertices that are close to v. In order to prevent the vertices from separating too far due to the repulsion force, we make the coefficient of each spring proportional to the total descendant amount of next two hops of the two vertices connected by that spring. With the exception of the most outer fringe vertices, we apply a constant spring coefficient to keep those vertices from spreading out of the display area.
3) A tree-like graph usually contains a small number of non-tree edges. In order to minimize the layout alteration between the final force-directed drawing and the original radial drawing, we waive the spring forces applied on non-tree edges. These non-tree edges without spring forces may create edge crossings. However, this influence can be ignored, as the number of non-tree edges is small.

By applying these constraints, the radial layout can be smoothly transformed to the force-directed layout. These not only facilitate to reduce the convergence time to reach the energy minimization in force-directed drawings, but also preserve the user's mental maps. We can see from Fig.6 that after switching from the radial drawing of the example graph shown in Fig.5 to the force-directed drawing of the same graph, the layout transition is minimized and the change between the two layouts is minimal.

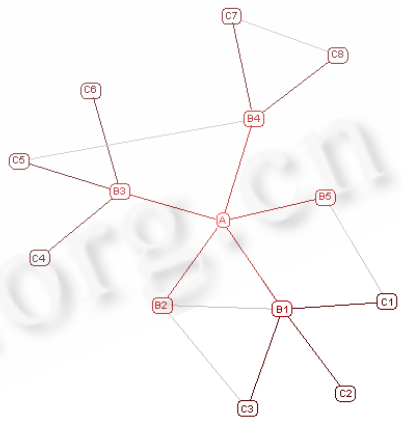


Fig.6 The final force-directed layout of the same graph shown in Fig.5 after being transformed from its original radial drawing. The change of layout is minimal

3 An Experimental Evaluation

In this section, we evaluate our new fast convergence layout algorithm by comparing the experimental results with other traditional force-directed methods, in which the initial positions of vertices are usually defined either randomly or at the centre of the display medium.

We evaluate our method against three measurements in terms of convergence time: *time consumption, average vertex position variance*, and *average vertex travel length*. This experimental evaluation was carried out with four

data sets consisting of 24, 65, 129 and 160 vertices. These four data sets have 28, 73, 161, 212 edges respectively. The final layouts of these data sets generated by our algorithm are presented in Fig.7.

In our experiment, each data set is drawn by three force-directed methods.

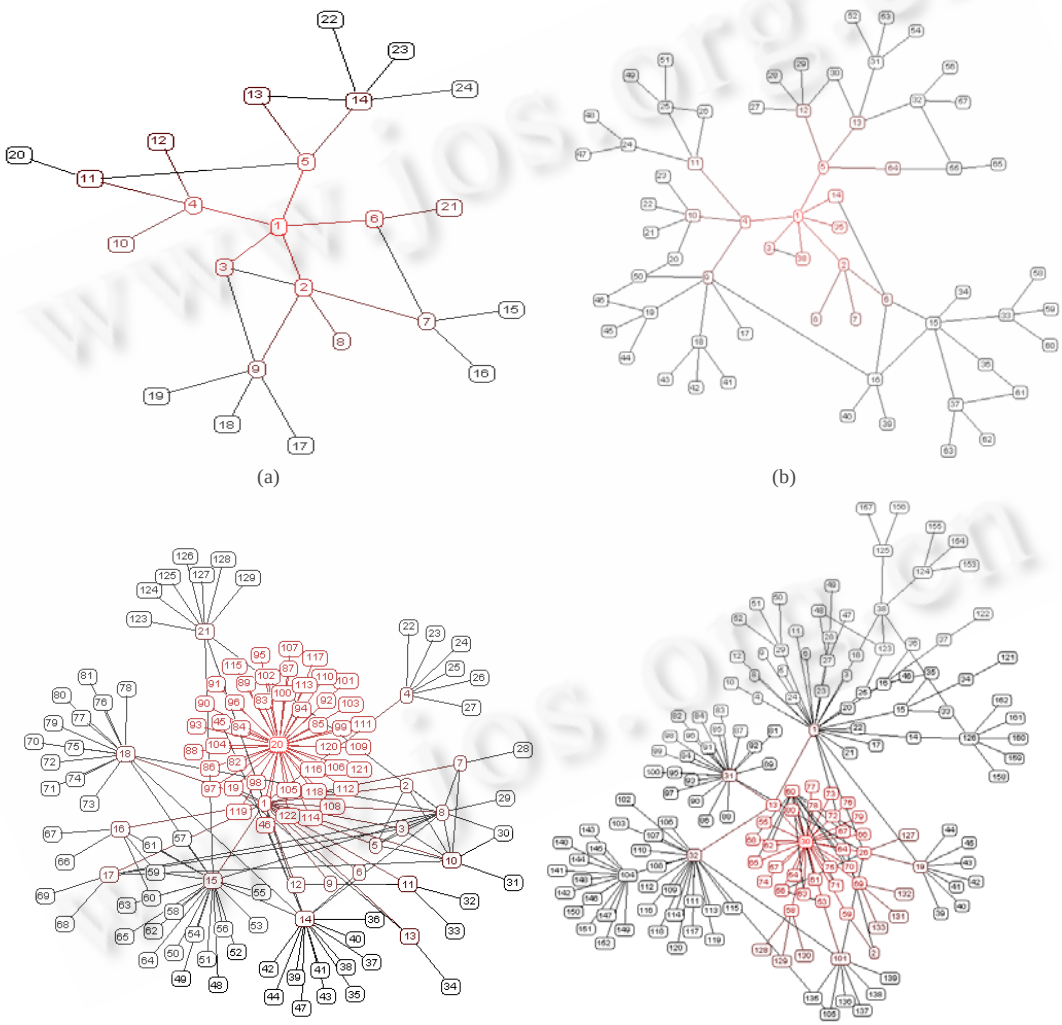
a) The fast convergence layout algorithm (***FC*** method),

b) The force-directed drawing method that places the initial position of vertices randomly (***FR*** method)

c) The force-directed drawing method that places the initial position of vertices at the origin (centre) of the display coordinates (***FO*** method)

The forces used in all the force-directed methods are the same. The total force exerted on vertex v is:

$$f(v)=\sum_{u\in N(v)} f_{uv}+\sum_{u\in V,u\neq v} g_{uv} \tag{1}$$



(a) (b) (c) (d)

Fig.7 The final force-directed layouts of four data sets consisting of 24 (28), 65 (73), 129 (161) and 160 (212) vertices (edges) generated by our algorithm

where f_{uv} is the force exerted on v by the spring between v and $u\in N(v)$ which is the direct neighbourhood set of v;

g_{uv} is the n-body force exerted on v by one of the other vertices $u \in V$. We use a quad-tree based implementation of the Barnes-Hut algorithm[14] for this n-body force calculation efficiently.

We used an IBM ThinkPad T30 Laptop (CPU 1.80GHz, Memory 256 MB) to run this experiment. The display resolution was configured to 1024×768 pixels. All the graphs were drawn in a 700×700 pixels square region.

Time Consumption *Time Consumption* or *TC* is the run-time spent in drawing a graph from its initial state to the end state, which is crucial to measuring the efficiency of force-directed graph drawing. Fast graph drawing is essential for exploring through a series of data sets with real-time response. Therefore, our objective in this research is to layout graphs as fast as possible. The proposed *FC* method attempts to speed up the convergence of traditional force-directed drawing methods, to suit the force-directed methods for interactive graph drawings. Since the convergence procedure of force-directed algorithms involves an infinite loop, we need to define a constant to form a convergence condition that will terminate the drawing. Given a graph G, which consists of n vertices, the average force AF acting on every vertex is calculated by the following formula:

$$AF = \frac{1}{n}\sum_{i=1}^{n}\left|f(v_i)\right| \tag{2}$$

We assume that when AF is less than a small force constant, the drawing of the graph reaches almost to the final layout. Therefore, we terminate the convergence procedure when $AF<1E-5f$ becomes true. We can then use the system clock to easily count the consumed time, $TC=t_1-t_0$, where t_0 is the beginning time of drawing and t_1 is the time at which $AF<1E-5f$ stands.

The total time consumed for a *FC* run R_{fc} consists of two parts

$$TC(R_{fc})=TC(R_{rd})+TC(R_{fd}) \tag{3}$$

where $TC(R_{rd})$ is the time spent on calculating the initial radial drawing and $TC(R_{fd})$ is that on running the force-directed drawing. Since neither the *FR* and the *FO* methods involves radial drawings, the accumulated time for a *FR* run (or a *FO* run) is equal to $TC(R_{fd})$.

Due to the unpredictability of the outcomes of force-directed drawing, the convergence time of every run varies. To accurately evaluate these methods in terms of their convergence time, we conduct ten runs $R^1,R^2,\ldots,R^{10}$ for each method and take the average time as the vector to measure these methods. The average *TC* of each method is thus:

$$TC_{average} = \frac{1}{10}\sum_{i=1}^{10} TC_i \tag{4}$$

The outcome of comparisons is illustrated in Fig.9(a). We can see that the overall performance of our *FC* method is superior over the other two methods, as *FC* improves significantly in convergence of energy minimization. For example, to draw *Graph2* with 65 vertices, *FC* takes only 2.07 seconds. In contrast, *FR* and *FO* take 8.37 and 7.88 seconds respectively, both about four times as long as the time spent by our new method. *FC* outperforms *FR* and *FO* even more for drawing large graphs. When drawing *Graph4* that consists of 160 vertices, *FC* takes only 4.86 seconds, less than a quarter of time spent by the other two methods.

Average Vertex Position Variance *Vertex Position Variance* or *VPV* indicates the position change of a vertex between its end position and its start position in the drawing. As shown in Fig.8(a), a $VPV(v_i)$ of vertex v_i is defined as the geometric distance between the end position $P_e(v_i)$ and the start position $P_s(v_i)$. So, the average value of vertex position variance *AVPV* of G can be expressed below:

$$AVPV(G) = \frac{1}{n}\sum_{i=1}^{n}\left|P_e(v_i)-P_s(v_i)\right| \tag{5}$$

The larger the value of $AVPV(G)$ is, the more greatly the mental map of the graph G changes. Therefore, to

effectively preserve the mental map of layout of G after the view transformation, we should minimize the value of $AVPV(G)$.

The comparisons are illustrated in Fig.9(b). We can see that all $AVPV$ values of four graphs generated by FC method are greatly reduced in comparison with the other two methods. For example, the convergence procedure of FC produces a low $AVPV(Graph1)$ value of 17.86, in contrast to the values of 267.44 and 106.73, produced by the other two methods. The result shows that the FC method could preserve the mental map much better, and hence require less cognition overload for identifying view changes.

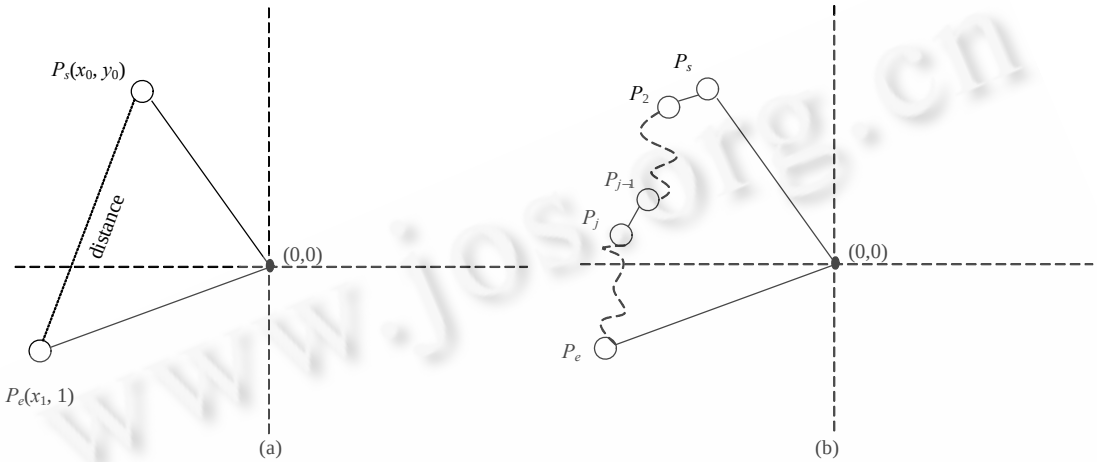


Fig.8 The picture (a) defines the VPV that equals to the distance between P_s and P_e. The picture (b) defines the VTL that equals to the length of total travel path $P_s,P_2,\ldots,P_e$

Average Vertex Travel Length *Vertex Travel Length* or *VTL* is the length of the entire path a vertex travelled from its start position to its end position. *VTL* consists of a series of turning points $P_s,P_2,\ldots,P_{j-1},P_j,\ldots,P_e$, see Fig.8(b). This measurement is important in determining the convergence time. A vertex travel length *VTL* is calculated by accumulating each path segment (P_{j-1},P_j) travelled in each animation loop j, see Fig.8(b). Suppose that m iterations are required for the convergence, the *VTL* of a vertex v_i is:

$$VTL(v_i)=\sum_{j=1}^{m}\left|(P_j-P_{j-1})\right| \tag{6}$$

Thus, the *Average Vertex Travel Length AVTL* of G can be calculated by the following formula:

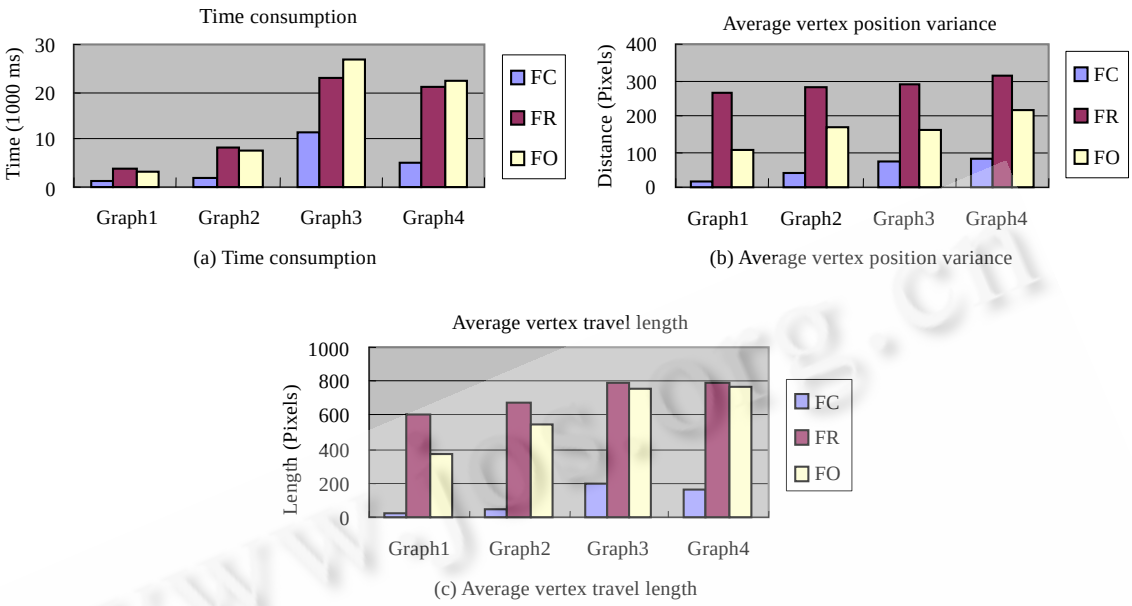
$$AVTL(G)=\frac{1}{n}\sum_{i=1}^{n}VTL(v_i) \tag{7}$$

A large value of $AVTL(G)$ implies that a particular layout method needs long time to reach the convergence when drawing G and the average vertex position adjustment of each iteration is great as well.

Figure 9(c) shows that the overall *AVTL* value in our *FC* method is very small in contrast to the other two methods. For example, for the largest one of the four graphs, the convergence procedure of *FC* produces a low $AVTL(Graph4)$ value of 158.88. In contrast, the other two methods produce the values of 796.14 and 768.06. This result indicates that our *FC* method could greatly shorten the convergence time in drawing force-directed graphs.

Furthermore, we found that the three measurements are comparatively consistent among ten runs of our *FC* method for each of the four data sets. In contrast, the runs of the *FR* and *FO* methods vary greatly. For example, we can see from the Fig.10 that the *AVTL* curves produced by our *FC* method for each data set are nearly level with tiny variance. However, the values generated by the other two methods fluctuate greatly. This implicates that the layouts produced by the *FC* method are more predictable and consistent, which can considerably relief the cognition

overload for users.

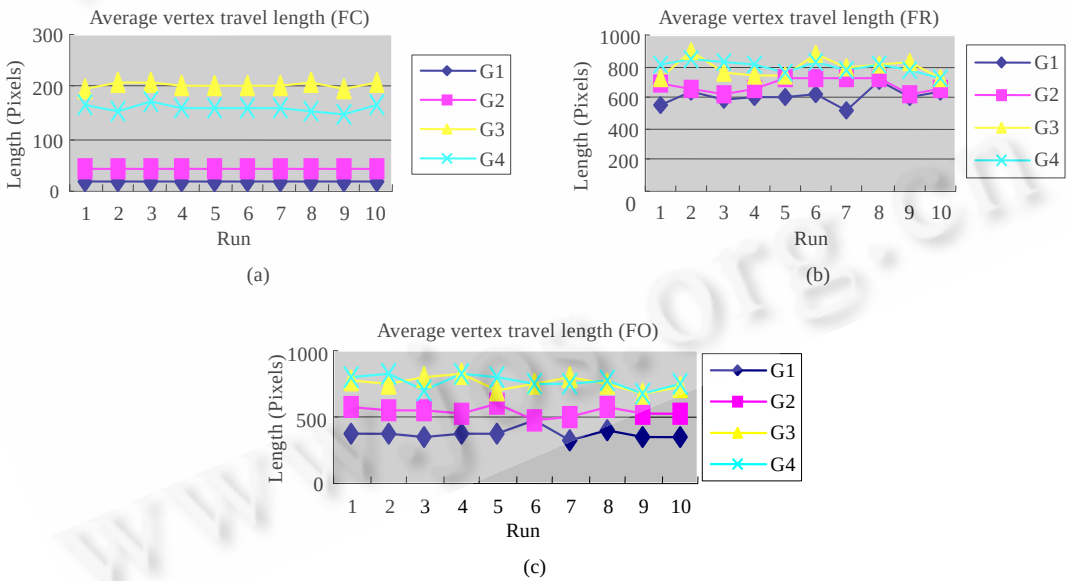


(a) Time consumption

(b) Average vertex position variance

(c) Average vertex travel length

Fig.9 A summary of the experimental results of four data sets



(a)

(b)

(c)

Fig.10 Ten runs of *AVTL* using three methods for each of the four data sets

4 Conclusions

We have presented a *Fast Convergence* (*FC*) graph drawing method for force-directed graph layouts. Particularly, we integrated the radial drawing algorithm into the traditional force-directed algorithm to produce fast convergence and predictable force-directed graph layouts. We believe that *FC* is an effective way for drawing a series of graphs $G_1,G_2,\ldots,G_n$ in a variety of applications, where real-time responses are required. The *FC* method is

originally designed to satisfy the real-time interaction requirements in a spatially referenced information visualization called *Marching-Graph* that allows users to "march" through a series of graphs $G_1,G_2,\ldots,G_n$ that are related to particular geographic regions on a map. The "marching" requires fast drawings of the graphs. The efficiency of navigation relies largely on the convergence time (or *TC*) spent in drawing those graphs. This method can also be applied to other visualization applications that require the navigation of a series of graphs in real-time.

To evaluate the *FC* method, we have conducted an experimental comparison between our method and two traditional force-directed drawing methods by running four data sets consisting of 24, 65, 129 and 160 data items. The comparisons show that the FC method achieves a significant improvement in terms of the convergence time spent for energy minimization and the preservation of the mental map.

In the next phase, we plan to carry out more formal evaluation experiments to investigate the scalability and applicability of the *FC* method. Furthermore, we will continuously investigate other alternative methods for speeding up the convergence procedures in force-directed graph drawing algorithms.

References:

[1] Fuchs G, Schumann H. Visualizing abstract data on maps. In: Proc. of the Information Visualization, 8th Int'l Conf. on Information Visualization (IV 2004). London, 2004. 139−144.

[2] Keim DA, Noth SC, Panse C, Schneidewind J. Visualizing geographic information: VisualPoints vs CartoDraw. Information Visualization, 2003,2:58−67.

[3] Andrienko GL, Andrienko NV. Interactive maps for visual data exploration. Int'l Journal for Geographical Information Science, 1999,13(4):355−374.

[4] Kreuseler M, Schumann H. A flexible approach for visual data mining. IEEE Trans. on Visualization and Computer Graphics, 2002, 8(1):39−51.

[5] Quan W, Huang ML. Dynamic visualization of spatially referenced information. In: Proc. of the Int'l Symp. on Visual Computing. LNCS 3804, 2005. 642−646.

[6] Davidson R, Harel D. Drawing graphs nicely using simulated annealing. ACM Trans. on Graphics, 1996,15(4):301−331.

[7] Eades P. A heuristic for graph drawing. Congressus Numerantium, 1984,42:149−160.

[8] Frick A, Ludwig A, Mehldau H. A fast adaptive layout algorithm for undirected graphs. In: Proc. of the Symp. Graph Drawing GD'93. 1994. 389−403.

[9] Fruchterman T, Reingold E. Graph drawing by force-directed placement. Software-Practice & Experience, 1991,21:1129−1164.

[10] Huang ML, Eades P, Wang JH. On-line animated visualization of huge graphs using a modified spring algorithm. Journal of Visual Languages and Computing, 1998,9:623−645.

[11] Battista GD, Eades P, Tamassia R, Tollis IG. Graph Drawing: Algorithms for The Visualization Of Graphs. Englewood Cliffs: Prentice-Hall, 1999.

[12] Eades P. Drawing free trees. In: Bulletin of the Institute of Combinatorics and Its Applications. 1992. 10−36.

[13] Yee KP, Fisher D, Dhamija R, Hearst MA. Animated exploration of dynamic graphs with radial layout. In: Proc. of the InfoVis'01. 2001. 43−50.

[14] Barnes JE, Hut P. A hierarchical $O(N \log N)$ force-calculation algorithm. Nature, 1986,324(6270):446−449.

QUAN Wu is a postdoctoral fellow at the School of Information Technologies, University of Sydney, Australia. His research interests are information visualization, graph drawing and mobile ad hoc network.

HUANG Mao-Lin is a senior lecturer and doctoral supervisor at the Faculty of Information Technology, University of Technology, Sydney, Australia. His researches areas are graph drawing and information visualization.