Journal of Visual Languages and Computing 23 (2012) 29–42

Contents lists available at SciVerse ScienceDirect

Journal of Visual Languages and Computing

journal homepage: www.elsevier.com/locate/jvlc

A new force-directed graph drawing method based on edge–edge repulsion[☆], [☆☆]

Chun-Cheng Lin[a], Hsu-Chun Yen[b,*]

[a] Department of Industrial Engineering and Management, National Chiao Tung University, Hsinchu 300, Taiwan, ROC
[b] Department of Electrical Engineering, National Taiwan University, Taipei 106, Taiwan, ROC

ARTICLE INFO

Article history:
Received 14 September 2009
Received in revised form
28 April 2010
Accepted 6 December 2011
Available online 13 December 2011

Keywords:
Force-directed method
Potential field
Angular resolution

ABSTRACT

The conventional force-directed methods for drawing undirected graphs are based on either *vertex–vertex* repulsion or *vertex–edge* repulsion. In this paper, we propose a new force-directed method based on *edge–edge* repulsion to draw graphs. In our framework, edges are modelled as charged springs, and a final drawing can be generated by adjusting positions of vertices according to spring forces and the repulsive forces, derived from *potential fields*, among edges. Different from the previous methods, our new framework has the advantage of overcoming the problem of *zero angular resolution*, guaranteeing the absence of any overlapping of edges incident to the common vertex. Given graph layouts probably generated by previous algorithms as the inputs to our algorithm, experimental results reveal that our approach produces promising drawings not only preserving the original properties of a high degree of symmetry and uniform edge length, but also preventing zero angular resolution and usually having larger average angular resolution. However, it should be noted that exhibiting a higher degree of symmetry and larger average angular resolution does not come without a price, as the new approach might result in the increase in undesirable overlapping of vertices as some of our experimental results indicate. To ease the problem of node overlapping, we also consider a hybrid approach which takes into account both edge–edge and vertex–vertex repulsive forces in drawing a graph.

1. Introduction

As graphs are known to be one of the most important abstract models in various scientific and engineering areas, *graph drawing* (or *information visualization* in a broader sense) has naturally emerged as a fast growing research topic in computer science. Among various graph drawing techniques reported in the literature, the so-called *force-directed methods* (see, e.g., [1,3,4,7,11,13,15,18,25,28]) have received much attention and have become very popular for drawing general, undirected graphs. In such a framework, a graph is viewed as a system of particles with forces acting between the particles, and then a good configuration or drawing of the particles could be generated with locally minimal energy, i.e., the sum of the forces on each particle is zero.

Generally speaking, the notions of repulsions in the setting of conventional force-directed methods fall into two categories, namely, *vertex–vertex repulsion* and *vertex–edge repulsion*. First, Eades [4] presented a *vertex–vertex repulsion* model in which vertices are replaced with charged steel rings and edges with springs to form a mechanical system. The equilibrium configuration, in which the sum of repulsive forces due to rings and attractive forces due

[☆] A preliminary version of this work was presented at the Ninth International Conference on Information Visualisation, July 6–8, 2005, London, UK.
[☆☆] This paper has been recommended for acceptance by Shi Kho Chang.
[*] Corresponding author. Tel.: +886 2 33663581.
E-mail address: yen@cc.ee.ntu.edu.tw (H.-C. Yen).

1045-926X/$ - see front matter © 2011 Elsevier Ltd. All rights reserved.
doi:10.1016/j.jvlc.2011.12.001

to springs is zero, normally results in a good drawing. Frunchterman and Reingold [11] subsequently presented an effective modification of the model.

In subsequent studies, *vertex–edge repulsion* models have been proposed to prevent a vertex from being placed too close to an edge, overcoming a potential shortcoming as a result of using the vertex–vertex repulsion model. Davidson and Harel [3] used the paradigm of simulated annealing, suited for combinatorial optimization problems, to draw graphs. Their method tries to find an optimal configuration according to a cost function inclusive of a measure for the distance between each pair of vertex and edge. This measure penalizes the vertex and edge that are too close to each other. In addition, Bertault [1] presented a force-directed method based on vertex–edge repulsion to ensure that two edges cross in the final drawing if and only if they cross in the initial layout as well.

Aesthetic criteria specify graphic structures and properties of drawing, such as minimizing number of edge crossings or bends, minimizing area, and so on (e.g., see [24]), but the problem of simultaneously optimizing those criteria is, in many cases, NP-hard. Among important aesthetic criteria, *angular resolution* refers to the smallest angle formed by two neighboring edges incident to the common vertex in a straight-line drawing, and constructing straight-line drawings of huge graphs with large angular resolution is very important in visualization applications and, in addition, the design of optical communications networks [6].

Given a drawing of graph G, an angle formed by the two adjacent edges incident to a common vertex v is called an angle incident to vertex v. With respect to a vertex v, the *angular resolution* is the smallest angle incident to vertex v. The *angular resolution* of a drawing of G is defined as the minimum angular resolution among all vertices in G. The *average angular resolution* of a drawing of G is defined as the average of the angular resolutions of all vertices in G. The (average) angular resolution of a graph drawing is in the range of $[0^\circ, 180^\circ]$. It is worthy of pointing out the fundamental difference between angular resolution and average angular resolution. The angular resolution only concerns the minimum angle in the drawing, while the average angular resolution deals with angular resolutions of all the vertices.

The main aesthetic criteria concerned in this paper are symmetry, uniform edge length, and maximization of (average) angular resolution. Theoretical and experimental results (e.g., see [5,4]) have suggested that force-directed methods usually enjoy the merit of producing graph layouts with a high degree of symmetry and uniform edge length. However, their graph layouts may have the problem of *zero angular resolution*, i.e., there exist at least two of the edges incident to the common vertex overlapping, resulting in a bad drawing with edge–edge and vertex–edge crossings simultaneously. Note that the simulated annealing graph drawing method [3] additionally considering a term of punishing zero angular resolution can be applied, but it is not efficient and only can be applied to moderate-size graphs.

In this paper, a new force-directed method using the concept of *edge–edge* repulsion based upon the theory of *potential fields* is presented to draw graphs without zero angular resolution. The concept of potential fields has already found applications in a variety of areas in computer science and engineering, such as path planning [2] and drawing of graph with nonzero-sized vertices [22], among others. In our setting, the repulsive forces applied to an edge are caused by the edge being present in the potential field induced by its neighboring edges. Although Chuang and Ahuja [2] have derived analytically formulas of repulsive forces between two charged edges, respectively, they are unnecessarily complicated to implement practically. Therefore, as we shall see later in our derivation, the formulas of our edge–edge repulsion are very simple and can be implemented easily.

Given a reasonably nice graph layout probably generated by one of those force-directed methods reported in the literature as the input of our algorithm, the experimental results reveal that our approach is capable of producing a drawing not only preserving a high degree of symmetry and uniform edge length but also preventing zero angular resolution and usually having larger average angular resolution. Like many aesthetic criteria which are often in conflict with one another, exhibiting a higher degree of symmetry and larger average angular resolution does not come without a price, as the new approach might result in the increase in undesirable overlapping of vertices as some of our experimental results indicate. To ease such a problem we also propose a hybrid approach which takes into account both edge–edge and vertex–vertex repulsive forces in drawing a graph.

Conventional force-directed methods cannot guarantee the generation of nice drawings for all kinds of graphs because they might not be able to reach global minimal configurations, i.e., the repulsive forces among some vertices (rings) might be too weak or too strong. Fortunately, some local minimal problems can be resolved by adjusting parameters of the force models. Our approach, based on the model of edge–edge repulsion, might not be able to escape from the occurrence of a local minimal solution in general, because we only consider the repulsive force between neighboring edges locally. As a result, taking as an input a reasonably nice drawing is very helpful for our algorithm to converge to a solution corresponding to a drawing meeting the aesthetic criteria for which our design is targeted.

The rest of this paper is organized as follows. Section 2 gives some preliminaries. In Section 3, our approach is introduced in depth. Section 4 gives a variety of experimental results. Finally, a conclusion is given in Section 5.

2. Preliminaries

In this section, we give formal definitions for the problem at hand, as well as the necessary background in force-directed methods.

2.1. Basic definitions

A graph is a pair $G=(V,E)$ where V is the set of vertices and $E \subseteq V \times V$ is the set of edges. A *drawing* of a graph G on the plane is a mapping D from V to $\mathbb{R}^2$, where $\mathbb{R}$ is the set of real numbers. That is, each vertex v is placed at point $D(v)$ on the plane, and each edge (u,v) is displayed as a straight-line segment connecting $D(u)$ and $D(v)$.

2.2. Defining the problem

The drawing problem considered in this paper is addressed as follows. Suppose we begin with an initial "reasonably nice" drawing of a graph, one is required to produce a "nicer" drawing of the graph with respect to the following aesthetic criteria: nonzero angular resolution, maximization of (average) angular resolution, symmetry, uniform edge length, and the minimization of the whole drawing size. In addition, the efficiency of the drawing algorithm is also factored into our design. Note that our initial input drawing is a "reasonably nice" drawing, which may be generated by a pervious force-directed method or drawn manually. That is, our drawing method can be regarded as a fine-tuning postprocessing stage of the existing force-directed methods.

2.3. Previous work on force-directed methods

The graph drawing algorithm of Tutte [26,27] described by pure mathematics can be regarded as the earliest *force-directed method*. In the Tutte model, the set of vertices is divided into two sets, a set of fixed vertices and a set of free vertices. By nailing down the fixed vertices as a strictly convex polygon and then placing each free vertex at the barycenter of its neighbor in each iteration, the model can yield a nice drawing.

Subsequently, since the introduction of the simple force-directed method by Eades in [4] (a.k.a., *spring algorithm*), there has been a number of variants of force-directed approaches reported in the literature. Generally speaking, such modifications fall into the following two categories. One has to do with altering the repulsive force and the spring force models, while the other attempts to manipulate the local minima problem resulting from the equilibrium between attractive and repulsive forces.

The model introduced by Eades uses logarithmic strength springs in place of Hooke's law for spring forces f_a, and the inverse square law for repulsive forces f_r as follows:

$$f_a(\mathbf{d}_{uv}) = \left(c_1 \cdot \log\left(\frac{|\mathbf{d}_{uv}|}{c_2}\right)\right)\frac{\mathbf{d}_{uv}}{|\mathbf{d}_{uv}|}, \tag{1}$$

$$f_r(\mathbf{d}_{uv}) = -\left(\frac{c_r}{|\mathbf{d}_{uv}|^2}\right)\frac{\mathbf{d}_{uv}}{|\mathbf{d}_{uv}|}, \tag{2}$$

where c_1 and c_r are scaling constants for spring forces and repulsive forces, respectively, c_2 is the given spring natural length, and $\mathbf{d}_{uv}$ is the vector from vertex u to vertex v. Besides above considering repulsive forces between every vertex–vertex pair, the model of considering repulsive forces between every vertex–edge pair [1] also was developed to preserve the edge crossing property. Subsequently, a number of variations of spring algorithms have been proposed to improve the performance as well as the drawing quality. Notable examples include [3,7,11,18,25]. The algorithm in [11] uses the following as alternate spring and repulsive force formulas: $f_a(\mathbf{d}_{uv}) = (|\mathbf{d}_{uv}|^2/k)\mathbf{d}_{uv}/|\mathbf{d}_{uv}|$ and $f_r(\mathbf{d}_{uv}) = -(k^2/|\mathbf{d}_{uv}|)\mathbf{d}_{uv}/|\mathbf{d}_{uv}|$, respectively, where k is a constant. As it turns out, this change makes the algorithm more efficient than the original spring algorithm. In addition, a parameter called *temperature* is used to terminate the algorithm. Every vertex initially has a temperature value and the value decreases by computing some cooling function at the end of each iteration. Until all vertices cool down to some constant value, the algorithm stops to attain a nice drawing. In a similar work, the approach of GEM [7] makes use of the history of the moving trajectory of all vertices to compute the temperature.

As the algorithm in [3] indicates, the simulated annealing [19] approach also plays a constructive role in graph drawing. The basic idea is as follows. Given an evaluation function consisting of a set of criteria, e.g., number of edge crossings, the temperature of every vertex decreases at the end of each iteration, but the temperature of a vertex may increase when the return value of the evaluation function for the new position of the vertex is worse than that regarding the original position of the vertex. The algorithm terminates when the temperature of every vertex is below some predefined value. In addition, Kamada and Kawai [18] have used a potential formula to replace the force formula, and the optimization procedure tries to minimize the total energy of the system. Sugiyama and Misue [25] have considered the force-directed method based on magnetic forces. Their method replaces some or all of the edges of a graph by magnetized springs, and gives a global magnetic field that acts on the springs. It gives three basic types of magnetic fields (i.e., parallel, radial, and concentric) to control the orientation of the edges, and hence can generate drawings with different aesthetic criteria.

In the past, the force-directed techniques only can handle moderate-sized graphs (about 50 vertices). Recently, the multi-scale approaches [13,15,28] make them successful with much larger graphs (over 10,000 vertices), and an overall experimental comparison is given in [14]. An implementation of the multi-scale force-directed drawing approaches on GPU can be found in [8]. In addition, the force-directed methods in the multi-dimensional Euclidean space and Non-Euclidean space can be found in [12] and [20], respectively.

In the literature, there have existed a lot of applications on force-directed methods, e.g., the visualization of distributed mobile object environments [9], online dynamic graph drawing [10,17], drawings of metro maps [16], drawings of time-varying graphs [21], and so on. It is worthy of pointing out that if the drawing of each iteration of force-directed methods is rendered, the animated process allows the user to predict the dynamics of the drawing, which meets the requirement in information visualization [23].

3. Our force-directed graph drawing method

Similar to other force-directed methods, our method involves two parts, i.e., the force-directed model and the optimal algorithm introduced in Section 3.1 and Section 3.2, respectively.

3.1. Model of edge–edge repulsion

Our force-directed method with edge–edge repulsion is based upon the idea of replacing edges by charged springs, as opposed to charging vertices as was done in [4]. The closer two adjacent charged edges are, the stronger the repulsive force between them becomes. Intuitively, larger repulsive forces should make the included angle between two neighboring edges wider. In our design, in addition to stretching the included angle, the repulsive forces also contribute to increasing the lengths of the two edges. Thus the positions acted by repulsive forces are set at the end points of the incident edges as Fig. 1 explains. On the other hand, spring forces pull vertices closer when spring lengths are longer than their natural spring lengths. Finally, a drawing without zero angular resolution is generated when the corresponding model reaches an equilibrium between those repulsive forces and spring forces. With a given embedding, two edges are said to be *neighboring edges* if they share a common endpoint, and one is the successor of the other along a clockwise or counter-clockwise rotation. (See $\overline{AB}$ and $\overline{AC}$ in Fig. 1.)

In what follows, the formulas for capturing spring forces and repulsive forces are described. The formula of a spring force is based upon the classical spring embedder model [4], which uses Eq. (1) as the spring force formula.

The formulas of the repulsive force due to two charged edges can be derived from the theory of *potential fields*. The reader is referred to [2] for more about potential fields as well as some of the detailed derivations of exact formulas. However, those formulas derived in [2] appear to be a bit complicated and consequently require special care when implementing such a method. From a practical viewpoint, such a complication may not be needed for the purpose of drawing graphs. Therefore, by observing some characteristics of edge–edge repulsion and experimental results of [2], we are able to derive a simplified version of repulsive forces. Experiments based on the model reveal encouraging and promising results, as reported in the next section.

The key in our edge–edge repulsion model is to express the repulsive forces between two neighboring edges solely in terms of the lengths of the two edges and the included angle between the two edges. To better explain what this means, consider Fig. 1(b) as an example. It is easy to observe that the magnitudes of the repulsive force due to the two edges $\overline{AB}$ and $\overline{AC}$ are

1. *Positively* correlated with the lengths of $\overline{AB}$ and $\overline{AC}$;
2. *Negatively* correlated with the angle between $\overline{AB}$ and $\overline{AC}$.

Figs. 2(a) and (b) are curves, based upon the implementation of formulas derived in [2], displaying how the magnitude of the repulsive force between $\overline{AB}$ and $\overline{AC}$ (see Fig. 1(b)) is related to the length (Fig. 2(a)) and the included angle (Fig. 2(b)) of two uniformly charged edges. As Fig. 2(a) indicates, the relationship between the magnitude of force and the total length of edges is nondecreasing and concave. Intuitionally, the magnitude should approach to zero as edge lengths approach to zero, and flatten out as edge lengths approach to infinity. It can be captured by an *arctangent* function on $(0,\infty)$, and thus the component $|f|_e$ of magnitude of the repulsive force due to the two edge lengths can be simplified as follows:

$$|f|_e = c_3\left(\tan^{-1}\left(\frac{|\overline{AB}|}{c_4}\right)+\tan^{-1}\left(\frac{|\overline{AC}|}{c_4}\right)\right), \tag{3}$$

where $|\overline{AB}|$ and $|\overline{AC}|$ are the lengths of $\overline{AB}$ and $\overline{AC}$, respectively; c_3 and c_4 are constants to control the height of the approaching horizontal line and the scale of the horizontal axis, respectively.

Fig. 2(b) shows the relationship between the angle included by $\overline{AB}$ and $\overline{AC}$ and the magnitude of force. It turns out that the curve is positive, nonincreasing and convex. The magnitude approaches to infinity as the included angle approaches to zero. On the other hand, the magnitude slowly flattens out as the included angle grows. Such a behavior can be captured by a *cotangent* function on $(0,\pi/2]$, and hence the component $|f|_\theta$ of magnitude of the repulsive force due to the included angle can be set as follows:

$$|f|_\theta = c_5 \cot\left(\frac{\theta}{2}\right), \tag{4}$$

where c_5 is a constant to control the scale of the vertical axis, and θ is the angle included by $\overline{AB}$ and $\overline{AC}$. Note that Fig. 2(b) shows that the magnitude value at π is not zero, which, in fact, should be regarded as the contribution to edge lengths. In addition, for avoiding the appearance of values near infinity, $|f|_\theta$ is set to some fixed value when θ is below certain small cutoff value.

Therefore, the total magnitude $|f|$ can be computed as the sum of Eqs. (4) and (5) in the following:

$$|f| = |f|_e + |f|_\theta. \tag{5}$$

In what follows, to compute the orientation of repulsive force, consider an angle included by two edges $\overline{AB}$ and $\overline{AC}$ as shown in Fig. 3. There exist two angles included by the two edges, and the angle with degree smaller than π is denoted as θ. In the process of computing orientation, however, we need to determine which one is θ, and which

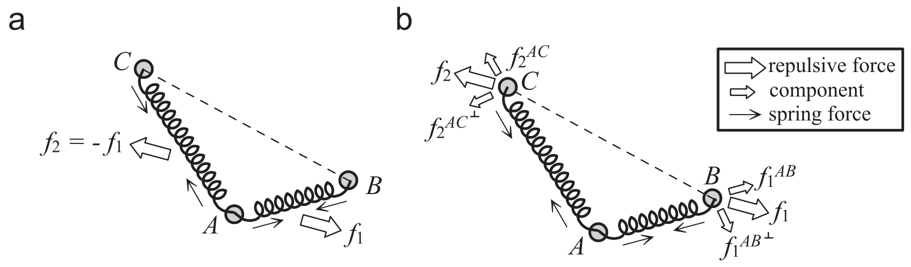

Fig. 1. A graph with three vertices A, B, and C and two edges modelled by charged springs. (a) The force model where f_1 and f_2 are repulsive forces acting on $\overline{AB}$ and $\overline{AC}$, respectively. (b) The positions acted by repulsive forces f_1 and f_2 should be set at the end points B and C of incident edges of vertex A.

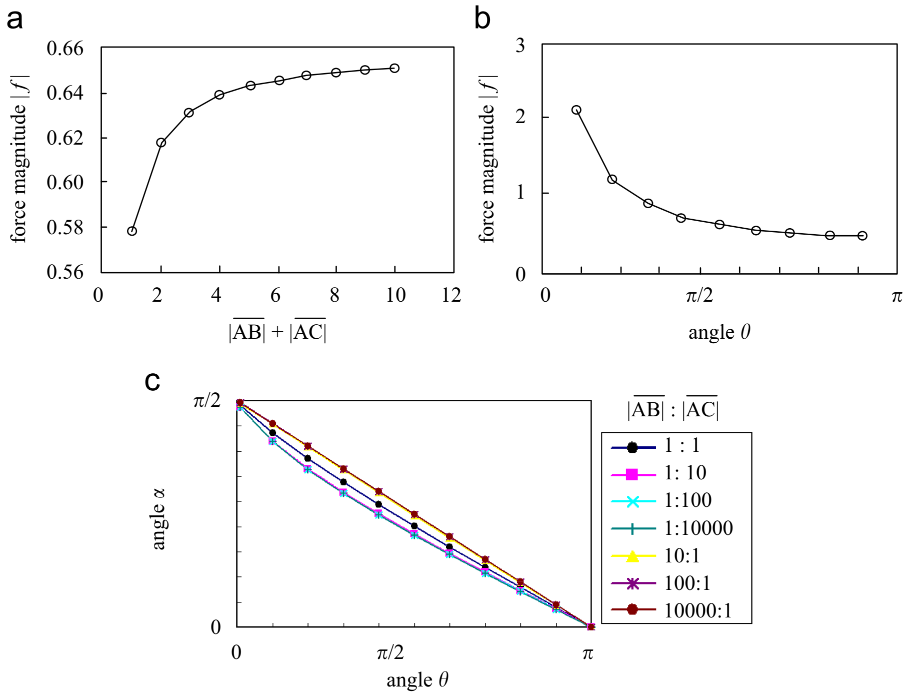

Fig. 2. Curves displaying the relationships between the magnitude of force and (a) the sum of edge lengths (the tendency between the magnitude of force and the length of each edge is similar), and (b) the included angle of two uniformly charged edges. (c) The experimental results designed for measuring the orientation of force. Consider a variety of $|\overline{AC}| : |\overline{AB}|$ to plot α versus θ, where α denotes the acute angle included by f_1 and AB.

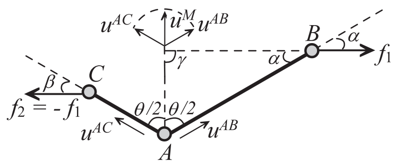

Fig. 3. Illustration of orientation of the repulsive force due to two edges $\overline{AB}$ and $\overline{AC}$ with an included angle θ.

edge is its right or left included edge. Assume that u^{AB} and u^{AC} are unit vectors of $\overrightarrow{AB}$ and $\overrightarrow{AC}$, respectively, and u^M is the unit vector of the sum of them. Based upon u^M and one of u^{AB} and u^{AC}, Proposition 1 by using the formula of cross product below allows us to judge which is the right or left included edge of the smaller included angle θ, and then assign the right (resp., left) edge denoted as $\overline{AB}$ (resp., $\overline{AC}$), and its corresponding unit vector as u^{AB} (resp., u^{AC}) accordingly.

Proposition 1. *Assume that* $u^{AC} = (a,b)$ *and* $u^M = (c,d)$, $\overline{AC}$ *is the left edge of the included angle* θ *if and only if* $(c\cdot b - a\cdot d) > 0$.

Because the angle included by u^{AB} and u^M is equal to $\theta/2$, $\theta/2$ can be uniquely computed by the inner product formula of cosine function as follows:

$$\frac{\theta}{2} = \cos^{-1}\left(\frac{u^M \cdot u^{AB}}{|u^M||u^{AB}|}\right) = \cos^{-1}(u^M \cdot u^{AB}),$$

where $|u^M|$ and $|u^{AB}|$ are lengths of the unit vectors u^M and u^{AB}, respectively, and both equal one.

f_1 and f_2 denote the repulsive forces acting at vertices B and C, respectively. α denotes the acute angle included by f_1 and $\overrightarrow{AB}$. In view of the curve shown in Fig. 2(c), it is interesting to observe that θ and α appear to be correlated with each other according to the following simple linear equation:

$$\alpha + \frac{\theta}{2} = \frac{\pi}{2},$$

where α equals $\pi/2$ when θ is zero, and α equals zero when θ is π. That is, α is the complementary angle of $\theta/2$, and hence, the orientation u_{f_1} of f_1 is perpendicular to u_M, i.e., $\gamma = \pi/2$ in Fig. 3.

Therefore, u_{f_1} can be computed by rotating u^M clockwise with degree of $\pi/2$ as follows:

$$u_{f_1} = \frac{f_1}{|f|} = R\left(\frac{-\pi}{2}\right)\cdot u^M = \begin{pmatrix} \cos\left(\frac{-\pi}{2}\right) & -\sin\left(\frac{-\pi}{2}\right) \\ \sin\left(\frac{-\pi}{2}\right) & \cos\left(\frac{-\pi}{2}\right) \end{pmatrix} \cdot u^M$$
$$= \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix} \cdot u^M, \quad (6)$$

where $R(-\pi/2)$ is the rotation matrix of $-\pi/2$.

Therefore, f_1 is a vector with magnitude $|f|$ in Eq. (5) and orientation u_{f_1} in Eq. (6) computed as follows, and f_2 equals $-f_1$:

$$f_1 = |f| u_{f_1}. \quad (7)$$

Note that the repulsive force in Eq. (7) has the advantage that f_1 is determined only by three parameters $\overline{AB}$, $\overline{AC}$, and θ, facilitating a simple implementation of our drawing algorithm based upon edge–edge repulsion.

3.2. Graph drawing algorithm

Our drawing algorithm is sketched in Algorithm 1. The algorithm mainly includes three parts: for each vertex, first, Lines 6–15 compute the repulsive forces due to each pair of neighboring edges incident to the vertex; second, Lines 16–20 compute the spring force of each edge; third, Lines 21 draws the graph according to certain scale of the forces acting at each vertex.

Algorithm 1 *EERepulsion (a nice drawing of graph $G=(V,E)$).*

```
Input: a reasonably nice drawing of $G$.
Output: a nice drawing without zero angular resolution.
Require: $tmpForce[|V|]$ stores temporary forces of all vertices;
newPos and oldPos record the new and old positions of all vertices, respectively.
1:  assign initial locations of vertices in $V$
2:  while $converged \neq 1$ or the maximum iteration number is achieved do
3:    $converged \leftarrow 1$
4:    $oldPosn \leftarrow newPosn$
5:    initialize $tmpForce[|V|]$ as zeros matrix
6:    for each vertex $v$ in $V$ do
7:      if the degree of vertex $v$ is at least two then
8:        determine the neighboring order of adjacency edges of vertex $v$ by using outer product
9:        for each pair $(e_i,e_j)$ where edge $e_i=(v,v_i)$ and edge $e_j=(v,v_j)$ are neighboring edges incident to $v$, and $e_i$ is the right edge of their included angle with smaller degree do
10:         calculate the repulsive force $f_1$ at edge $e_i$ due to edge $e_j$ according to Eq. (7)
11:         $tmpForce[v_i] \leftarrow tmpForce[v_i]+f_1$
12:         $tmpForce[v_j] \leftarrow tmpForce[v_j]-f_1$
13:       end for
14:     end if
15:   end for
16:   for each edge $e=(v_i,v_j)$ in $E$ do
17:     calculate the spring force $f_a$ of edge $e$ according to Eq. (1)
18:     $tmpForce[v_i] \leftarrow tmpForce[v_i]+f_a$
19:     $tmpForce[v_j] \leftarrow tmpForce[v_j]-f_a$
20:   end for
21:   simultaneously move each vertex according to $min(c_6 \cdot tmpForce[|V|],t)$ where $c_6$ (resp., $t$) is a constant to control the magnitude (resp., upper bound) of movement, and then save new positions to newPosn
22:   if $\|newPosn-oldPosn\| > \epsilon$
23:     $converged \leftarrow 0$
24:   end if
25: end while
```

In Algorithm 1, parameter c_6 (resp., t) are used to control the magnitude (resp., upper bound) of movement of every vertex, *NewPos* and *OldPos* are data structures used to record the new and old positions of all vertices, respectively, and ϵ in line 22 is the tolerance of convergence for force which is usually a very small positive number. Remind that parameters c_1 and c_2 in Eq. (1) (logarithmic spring force formula) are used to control the force magnitudes and natural lengths of springs, respectively; parameters c_3 and c_4 in Eq. (3) (the magnitude of repulsive force due to two edge lengths) are used to control the height of the approaching horizontal line and the scale of the horizontal axis of the arctangent magnitude function, respectively; parameter c_5 in Eq. (4) (the magnitude of repulsive force due to the included angle) is used to control the scale of the repulsive force magnitude due to the included angle. In addition, the flag *converged* is used to control the convergence of the algorithm. Note that the algorithm can reach convergence if parameters c_1–c_6 and ϵ are given appropriately.

Assuming that Δ is the maximum degree of vertices, the algorithm in each iteration takes time complexity $O(|V|\Delta \log \Delta)$ to compute repulsive forces between each pair of neighboring edges, $O(|E|)$ to compute spring forces, and $O(|V|)$ to draw graph; hence, the time complexity of the algorithm is $O(|V|\Delta \log \Delta+|E|)$. In order to evaluate the total number of iterations experimentally, we execute our algorithm on a number of graphs ($|V| \leq 10,000$), which have reasonably nice drawings initially. The experimental evaluation shows that more than 1000 iterations of our algorithm do not have much improvement on the quality of the final drawing.

It should be noticed that the input of our algorithm must be restricted to a reasonably nice graph layout because our approach based on the model of edge–edge repulsion that only locally considers repulsive force of each pair of neighboring edges does not avoid the occurrence of any kind of local minimal problems.

4. Implementation and experimental results

Based on the formulas and algorithm detailed in the previous sections, we develop a prototype system for drawing undirected graphs. The implementation is in C++ using OpenGL library, and runs on an Intel Core 2 Duo E8400 3.00 GHz PC with memory of size 2.00 GB running Windows XP. In this section, we detail the implementation of our algorithm, and give some experimental results for some simple graphs and then huge graphs.

4.1. Implementation

The analysis on the convergence and the adjustments of parameters in force-directed methods has been discussed a lot in previous works (e.g., see [3,5,15]). On theoretical aspect, Eades and Lin [5] have shown that the general framework of force-directed methods can lead to a stable drawing in which many symmetries are displayed. In fact, our force-directed approach only modifies the force formulas, but does not have too much modification on the optimized procedure of conventional force-directed algorithms. Therefore, like [5], our approach also can be shown to lead to a stable drawing, under appropriate setting of parameters.

In Algorithm 1, it should be noticed that the setting of c_1–c_6 and ϵ influences not only the running time but also the convergence of our approach and the quality of the final drawing. In the following, we briefly explain how to set those parameters in Algorithm 1 to achieve convergence. W.l.o.g., we consider that the input graph is connected; thus, each vertex must be exerted by nonzero spring and repulsive forces. That is, if the total force acted at a vertex is nonzero, then the vertex moves either inward into or outward from the other vertices of the graph. Since parameters c_6 (resp., t) control the magnitude (resp., upper bound) of movement, the ranges in which vertices move are bounded. That is, if those parameters are set smaller, then the movement range of vertices is smaller. Note that parameter ϵ controls the tolerance of convergence, so under larger ϵ vertices do not move in smaller movement range. As a result, smaller c_6, t, as well as larger ϵ narrow the movement range of vertices, so our algorithm can achieve convergence if the movement range are set appropriately.

In fact, given certain setting of parameters, the user can easily verify whether the parameter setting achieves convergence or not by observing the differences of the drawings generated by some consecutive iterations of Algorithm 1. If the algorithm under a given parameter setting is divergent, the user can judge the divergence type to adjust the parameters so that the algorithm becomes convergent. The possible divergence types are stated as follows:

1. Bigger movement magnitude parameter c_6 makes vertices move farther so that the final drawing may be generated faster and hence the total running time may be shorter. Note that t controls the upper bounds of the movement. However, if some vertices move back and forth between two consecutive drawings so that the convergent positions of these vertices cannot be determined, then it implies that the value of c_6 should be decremented.
2. Bigger force convergence tolerance ϵ can make the algorithm achieve the convergence of forces faster. However, if we observe that a further iteration executed at the final drawing can obtain a drawing with better placements of vertices, then it implies that the value of ϵ should be decremented.
3. As for the setting of parameters c_1–c_5, the quality of the final drawing can be modified by adjusting those parameters according to their implication in the design of force formulas. Once one of those parameters is needed to be increased to improve the drawing quality, the value might be set too large, which makes the drawing divergent. In this situation, parameter c_6 should be decremented appropriately.

4.2. Experimental results for simple graphs

Some of the experimental results for a variety of simple graphs using the classical method [4] and our method are presented in Figs. 4 and 5, and their corresponding statistics

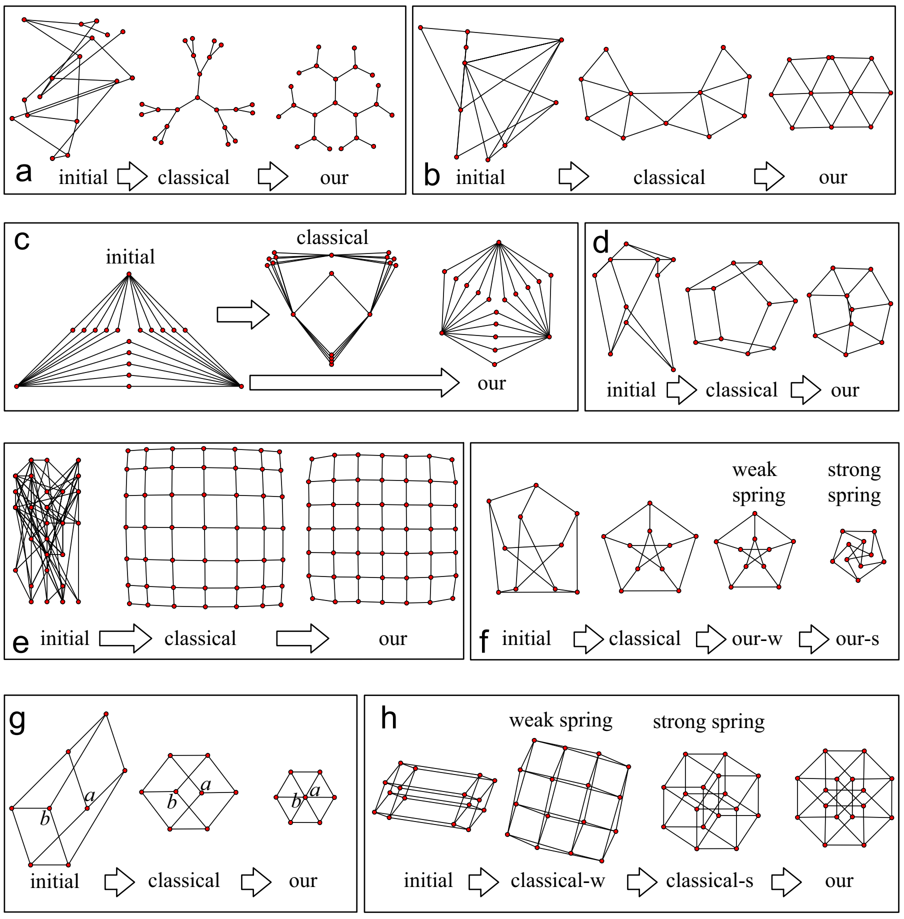

Fig. 4. Experimental results on some simple graphs (a) *tree_22*: a 22-vertex tree, (b) *tri_mesh_9*: a 9-triangle mesh, (c) *onion*, (d) *pentagon*, (e) *mesh_49*: a 49-vertex mesh, (f) *Petersen*: the Petersen graph, (g) *hypercube_3*: a 3-hypercube and (h) *hypercube_4*: a 4-hypercube.

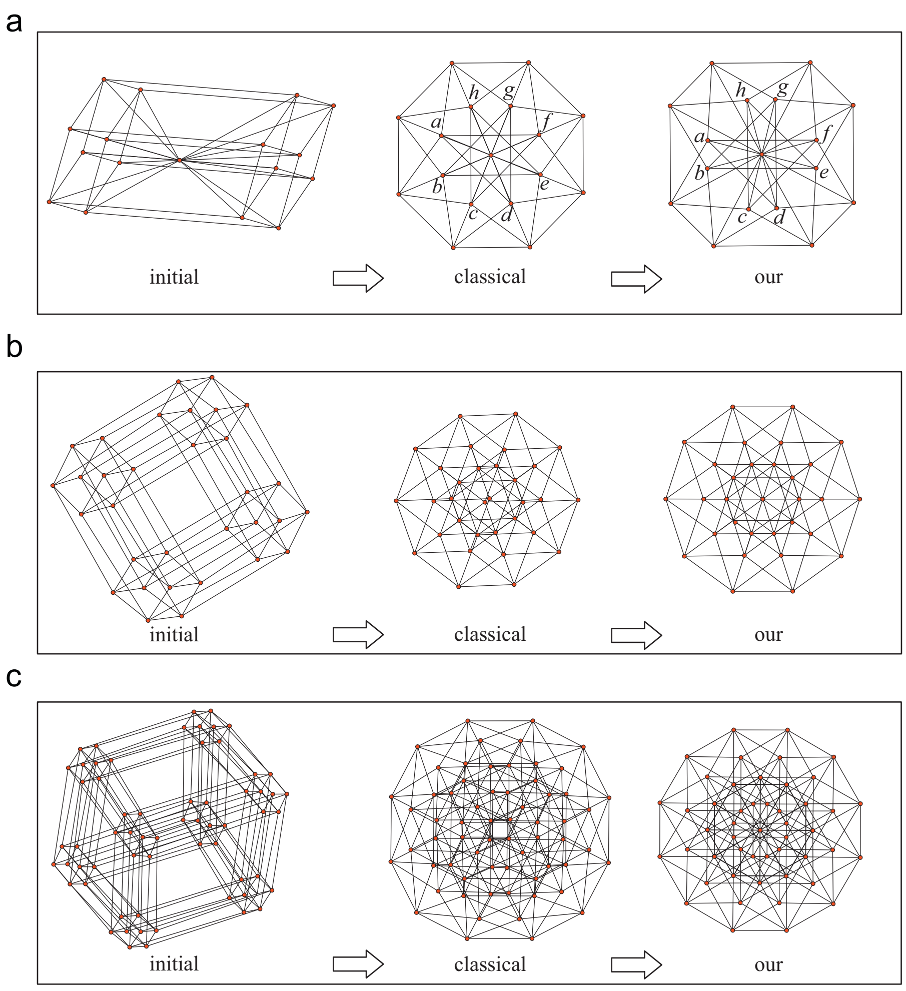

Fig. 5. Experimental results on some simple hypercube graphs (a) *hypercube_4_plus*: a 4-hypercube structure with a redundancy, (b) *hypercube_5*: a 5-hypercube structure and (c) *hypercube_6*: a 6-hypercube structure.

are shown in Table 1. From Table 1, those graphs are simple (with at most 64 vertices); both the classical and our methods under our setting of parameters are executed very efficiently (the total running time for each graph using either method is less than 0.5 s); the number of iterations for each graph using our method is less than 1000.

For convenience, the "arrow" sign in each case in Figs. 4 and 5 means that the concerned method takes the source of the "arrow" sign as the input, and produces the final drawing at the sink of the "arrow" sign. That is, the classical method takes the initial drawing as the input (see each case in Figs. 4 and 5), while our method takes either the drawing generated by the classical method (see each case in Figs. 4 and 5 except for Fig. 4(c)) or the initial drawing (see Fig. 4(c)) as the input. In addition, for explanatory convenience, the drawing produced by our (resp., the classical) method is said our (resp., the classical) drawing in the rest of this paper.

We can observe from each case in Figs. 4 and 5 that our method usually preserves the original properties of a high degree of symmetry and uniform edge length. As shown in Fig. 4(a), our method has the capacity of guaranteeing the drawing of a 22-vertex tree with evenly angular resolution, but the classical method may not. As shown in Figs. 4(b) and (c), given stronger springs, our method may reach the situation of the drawing with the largest angular resolution, but the classical method may not.

As shown in Figs. 4(d) and (e), although the classical method may already produce a nice drawing, our method may further improve its average angular resolution (the information can be found in Table 1). Note that in Fig. 4(e), the classical drawing has larger angular resolution but smaller average angular resolution than our drawing, in which almost all the angles incident to each vertex perform well except for those incident to the vertices on the boundary of the drawing. Hence, it is concluded that our method may lack the capacity of drawing the vertices on the boundary well. We put this as the future work.

A special case is shown in Fig. 4(f), in which the drawing using our method given stronger springs (ours) results in the central star spinning, and hence appears more compact and has more uniform edge length, though the (average) angular resolution is smaller. That is, we observe in this case that there is a trade-off between the area and the angular resolution.

Table 1
Statistics on the experimental results of simple graphs.

Graph name	$\|V\|$	$\|E\|$	Method	$\frac{StdDev}{AvgLen}$	*AngResl*	*AvgAngResl*	Number of iterations	Total running time (s)
tree_22 (Fig. 4(a))	22	21	Classical	0.12	24.93	45.67	1234	0.172
			Our	0.11	113.10	117.07	378	0.016
tri_mesh_9 (Fig. 4(b))	11	19	Classical	0.15	40.43	58.82	269	$< 10^3$
			Our	0.01	59.23	60.08	366	0.016
onion (Fig. 4(c))	18	30	Classical	0.06	0.13	63.06	345	0.031
			Our	0.05	11.38	124.24	673	0.046
pentagon (Fig. 4(d))	10	30	Classical	0.29	20.45	55.71	72	$< 10^3$
			Our	0.10	49.85	70.06	36	$< 10^3$
mesh_49 (Fig. 4(e))	49	84	Classical	0.21	80.88	86.90	513	0.359
			Our	0.02	78.17	88.63	265	0.046
Petersen (Fig. 4(f))	10	30	Classical	0.29	35.79	44.66	76	$< 10^3$
			Our-w	0.29	35.98	44.99	22	$< 10^3$
			Our-s	0.14	25.16	30.58	120	$< 10^3$
cube (Fig. 4(g))	8	24	Classical	0.13	48.40	62.10	83	$< 10^3$
			Our	0.00	60.00	75.00	35	$< 10^3$
hypercube_4 (Fig. 4(h))	16	64	Classical-w	0.35	2.28	24.60	163	0.016
			Classical-s	0.00	33.30	39.17	505	0.032
			Our	0.02	43.14	44.07	25	$< 10^3$
hypercube_4_plus (Fig. 5(a))	17	80	Classical	0.20	0.09	25.00	117	0.078
			Our	0.23	14.24	23.27	397	0.046
hypercube_5 (Fig. 5(b))	32	160	Classical	0.00	32.80	35.78	610	0.187
			Our	0.01	35.34	38.04	295	0.062
hypercube_6 (Fig. 4(c))	64	384	Classical	0.01	19.98	21.66	380	0.469
			Our	0.01	29.30	29.70	314	0.125

Figs. 4(g)–(h) and 5(a)–(c) draw the η-hypercubes[1] with $\eta = 2-6$. Because the model of edge–edge repulsion does not restrict at least two vertices coinciding, our method may produce drawings with more symmetries. Especially in Figs. 4(g), 5(b) and (c), although our drawings are improper (noting that a drawing is *improper* if there exist at least two vertices overlapping), our method has center symmetry with clear displays, while the classical method only has axial symmetry with somewhat confused displays. As a consequence, it can be observed from the above that the drawings with more symmetries may be in conflict with the aesthetic criterion of avoiding vertex–vertex overlapping.

In practice, the improper drawings may be prevented by applying a hybrid model of edge–edge repulsion and vertex–vertex repulsion, which simultaneously considers three kinds of forces: the edge–edge repulsive force f_1 for each pair of neighboring edges according to Eq. (7), the spring force f_a of each edge (spring) according to Eq. (1), and, different from the original model of edge–edge repulsion, the vertex–vertex repulsive force f_r for each pair of vertices (rings) according to Eq. (2). That is, f_r is added to the model of edge–edge repulsion, which only considers f_1 and f_a originally. By taking the improper drawings generated by our approach as the input of the algorithm based on the hybrid model, vertex–vertex overlapping would be avoided due to the vertex–vertex repulsive forces between vertices, and the output drawings may inherit a high degree of the merits of those improper input drawings, under appropriate setting of parameters, in which the parameter used to control the magnitude of vertex–vertex repulsive forces can be assigned a small value as long as there is no vertex–vertex overlapping.

In our experimental results above, Figs. 5(b) and (c) are the only two improper drawings. Hence, the hybrid model of edge–edge repulsion and vertex–vertex repulsion is used for the postprocessing of those drawings. The experimental results for Figs. 5(b) and (c) using the hybrid model under different strengths of the scaling parameter c_r of vertex–vertex repulsive force f_r in Eq. (2) are given in Figs. 6(a) and (b), respectively. We observe from those drawings that vertex–vertex overlapping is avoided if $c_r \neq 0$ (i.e., the vertex–vertex repulsion is acted), and a high degree of symmetries is preserved in compared to our original drawings. It is reasonably easy to see that a large c_r value has the effect of keeping vertices apart.

For evaluating the drawing quality in a quantitative way, the measures that are mainly concerned include the normalized standard deviation of edge lengths (*StdDev/AvgLen*), the angular resolution of the drawing (*AngResl*) and the average angular resolution of the drawing (*AvgAngResl*), as shown in Table 1.

Observing *StdDev/AvgLen* in Table 1, our method seems to have equal or more uniform edge length than

[1] Introduction to hypercubes: http://mathworld.wolfram.com/Hypercube.html

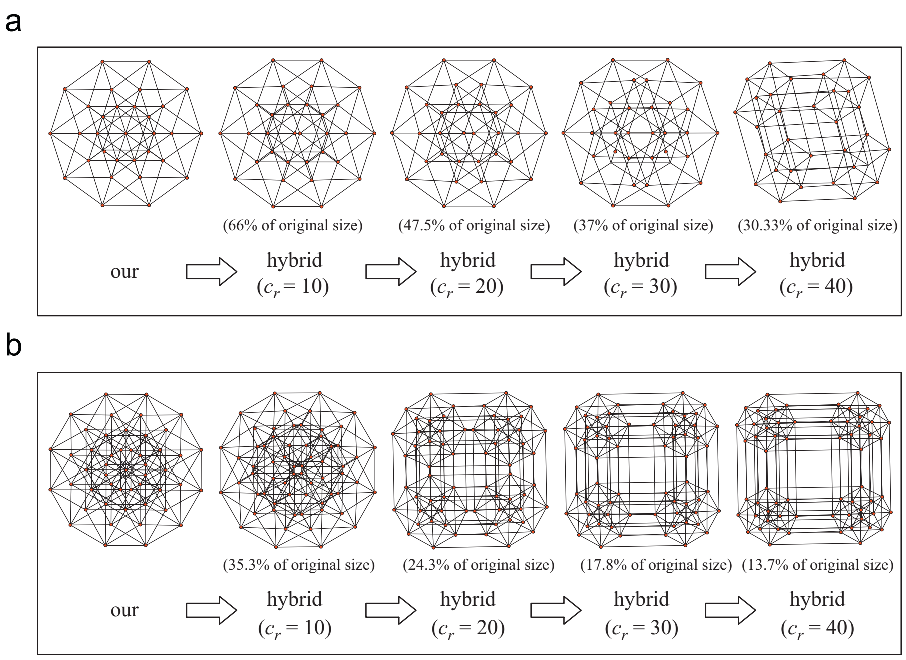

Fig. 6. Experimental results on some simple hypercube graphs using hybrid model with edge–edge and vertex–vertex repulsion (a) hypercube_5, and (b) hypercube_6.

the classical method, except for graph *hypercube_4_plus* (Fig. 5(a)) and *hypercube_5* (Fig. 5(b)). Observing *AngResl* in Table 1, the classical method may have the problem of almost zero or few angular resolution (e.g., see Figs. 4(c) and 5(a)), while our method normally has larger angular resolution than the classical, except for graph *mesh_49* (Fig. 4(e)). Observing *AvgAngResl* in Table 1, our method normally has larger average angular resolution than the classical, except for graph *hypergraph_4_plus* (Fig. 5(a)).

Since we knew that the drawings produced by the classical method have a high degree of symmetry and uniform edge length, our method taking the classical drawings as the input (which can be viewed as a fine-tuning postprocessing stage of the classical method) with not costing too much running time also inherits the same merits, according to the experimental results compared to the classical. In addition, from the experimental results, our method also has the merit of producing drawings with larger (average) angular resolution. As a result, it is concluded that our method can prevent zero angular resolution, and normally has the capacity of producing drawings with a high degree of symmetry, uniform edge length and larger (average) angular resolution.

4.3. *Experimental results for huge graphs*

Some experimental results for huge graphs (with about 10,000 vertices) are given in this subsection. The test graphs include: graph *rnd_graph_100* (Fig. 7) is a random grid graph that is produced by first establishing a 100×100 regular square grid graph and then randomly removing 3% of the vertices and their adjacent edges; graph *sierpinski_08* (Figs. 8 and 9) is a graph associated with the *Sierpinski triangle* after 8 iterations; graph *crack* (Fig. 10) is taken from real-world applications from C. Walshaw's graph partition archive.[2]

Hachul and Jünger [14] have given an overall experimental comparison for some multi-level force-directed methods for huge graphs, among which the *Fast Multipole Multilevel Method* (FM^3) [13] obtained comparable or better results in reasonable time [14]. Note that the implementation of the FM^3 used here comes from the OGDF library,[3] which has implemented a variety of graph drawing algorithms. Therefore, in the following, our method takes the drawings generated by FM^3 as the input to produce our drawings. The experimental results using the FM^3 and our method are presented in Figs. 7–10, and, and their corresponding statistics are shown in Table 2.

Since those huge graphs are more complicated, it is not easy to find an appropriate setting of parameters to make our algorithm convergent for force at a good placement of vertices. Hence, our method for huge graphs stops until the given maximal number of iterations is achieved. After an experimental evaluation for a variety of huge graphs, we find that the final drawings do not have a lot of modification after executing 1000 iterations of our method. Hence, most of our experimental results in this subsection are evaluated by executing 1000 iterations of our method. As for the running time, drawing a graph with about 10,000 vertices executed by 1000 iterations of our method takes about one minute (see Table 2).

In Fig. 7, for graph *rnd_grid_100*, the drawing produced by the FM^3 is given in (a); the drawings produced by our method after 1000 and 10,000 iterations are given in (b)

[2] http://staffweb.cms.gre.ac.uk/~c.walshaw/partition/
[3] http://www.ogdf.net/

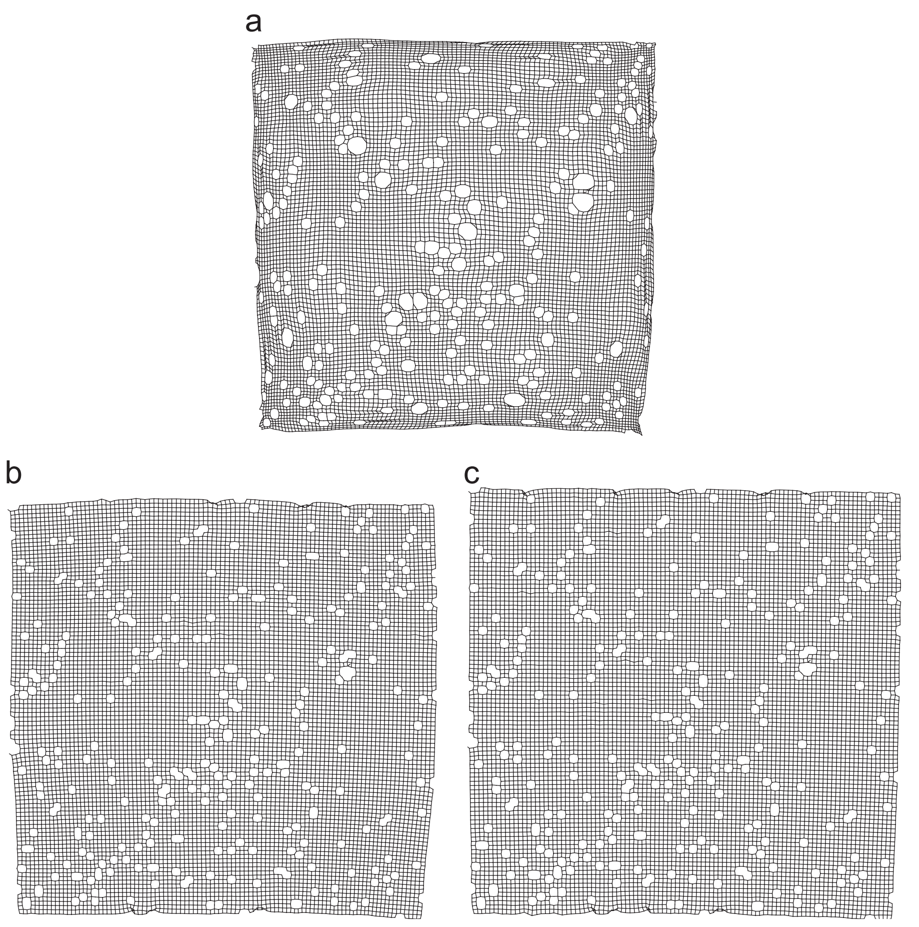

Fig. 7. Drawings of *rnd_grid_100*: (a) FM^3; (b) our (inputting (a) after 1000 iterations); and (c) our (inputting (a) after 10,000 iterations).

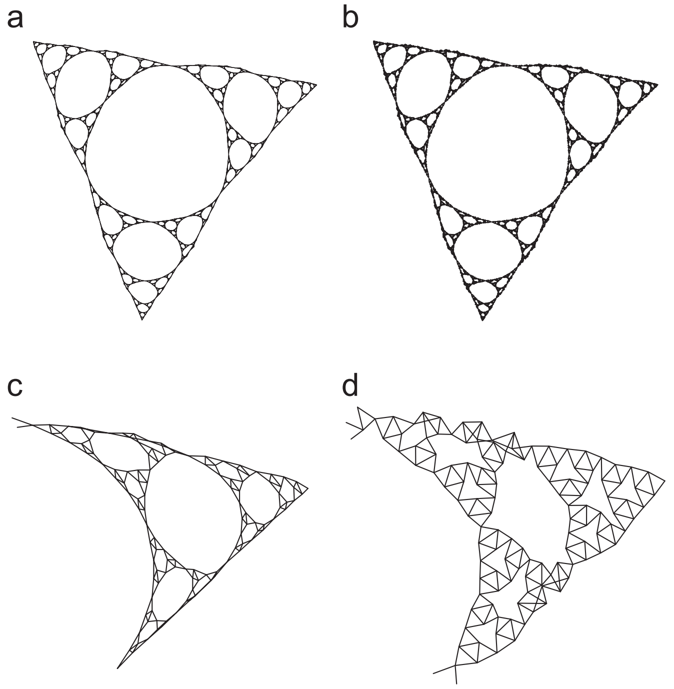

Fig. 8. Drawings of *sierpinski_08*: (a) FM^3; (b) our (inputting (a)); (c) local view of (a); and (d) local view of (b).

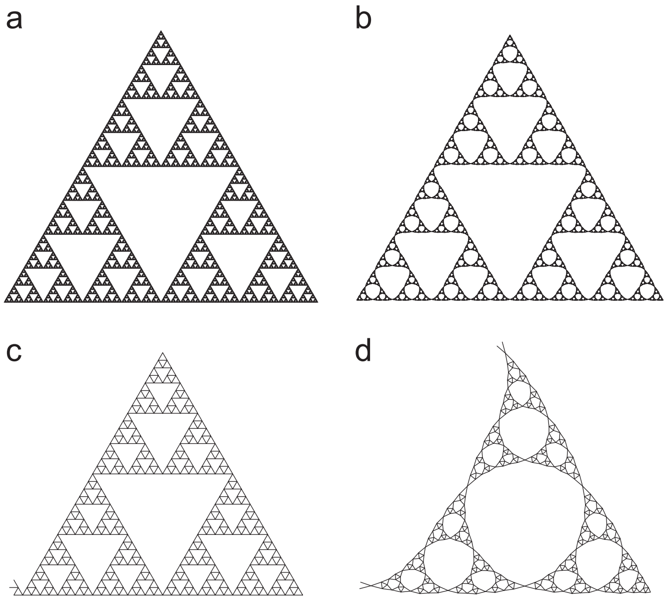

Fig. 9. Drawings of *sierpinski_08*: (a) manual; (b) our (inputting (a)); (c) local view of (a); and (d) local view of (b).

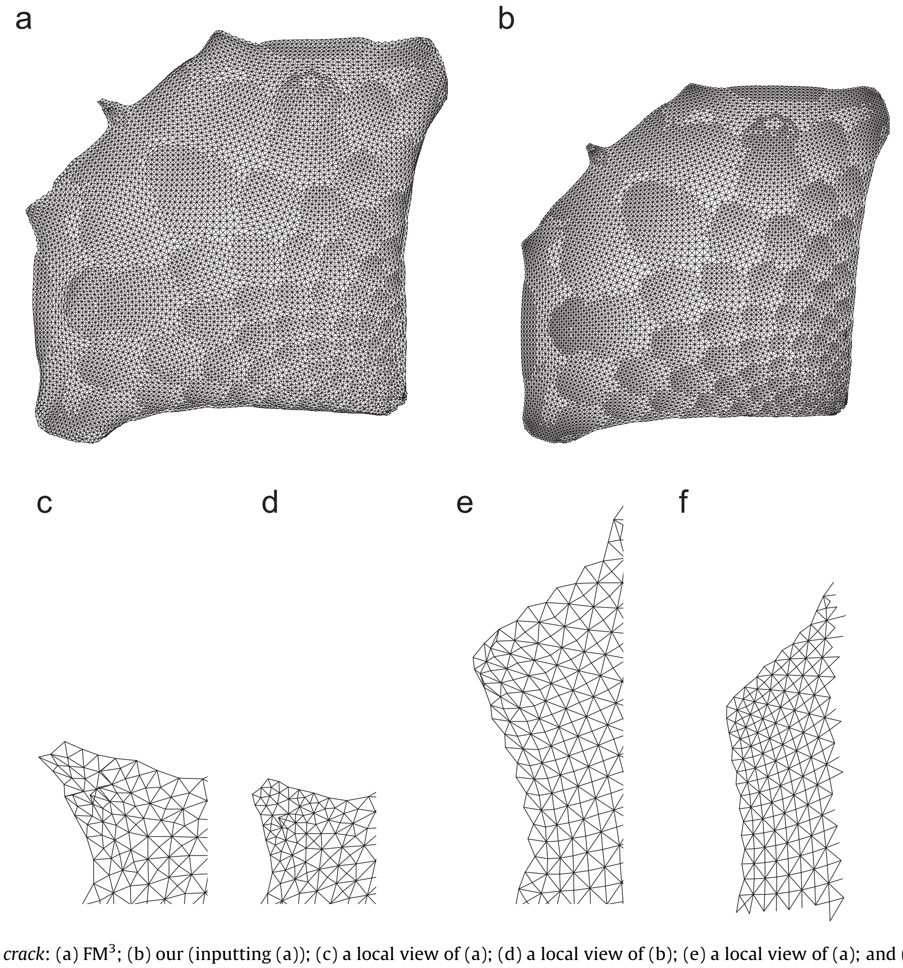

Fig. 10. Drawings of *crack*: (a) FM^3; (b) our (inputting (a)); (c) a local view of (a); (d) a local view of (b); (e) a local view of (a); and (f) a local view of (b).

Table 2
Statistics on the experimental results of huge graphs.

Graph name	$\|V\|$	$\|E\|$	Method	$\frac{StdDev}{AvgLen}$	*AngResl*	*AvgAngResl*	Number of iterations	Total running time (s)
rnd_grid_100 (Fig. 7)	9700	18,640	FM^3	0.17	6.00	79.81	–	24.938
			Our	0.05	17.91	88.64	1000	43.531
			Our	0.04	17.42	88.87	10,000	448.019
sierpinski_08 (Fig. 8)	9843	19,683	FM^3	0.41	0.00	25.32	–	16.750
			Our	0.25	7.67	44.38	1000	45.843
			Our	0.20	11.20	47.55	4150	190.891
sierpinski_08 (Fig. 9)	9843	19,683	Manual	0.01	59.97	59.98	–	–
			Our	0.07	44.65	57.17	1000	45.875
crack (Fig. 10)	10,240	30,380	FM^3	0.25	0.13	53.20	–	23.016
			Our	0.29	12.60	56.33	1000	68.406

and (c), respectively. Obviously, the drawings in (b) and (c) give clearer visualization than (a), which has some distortion on the boundary of the drawing. Since graph *rnd_grid_100* is a grid graph originally, our method produces a neat unfold drawing in (b) or (c), so that it is helpful for users to realize the original grid structure, though there are still a few edge crossings on the drawing boundary. As mentioned in the previous subsection, we put the local minimal problem for the vertices on the drawing boundary as the future work. As for the comparison between drawings (b) and (c), the contour of drawing (c) is more rectangular, but takes more running time than drawing (b). But, from Table 2, the aesthetic measures do not differ a lot between the two drawings. Therefore, from the quantitative point of view, we consider that it should be enough to execute 1000 iterations of our method for huge graphs with at most 10,000 vertices.

In Fig. 8, for graph *sierpinski_08*, the drawings produced by the FM^3 and our method are given in (a) and (b), respectively; the local views of (a) and (b) are given in (c) and (d), respectively. It seems that there are no much difference between (a) and (b) from the visualization, except the vertices look denser in (b). In fact, from the local views in (c) and (d), our method makes each triangle look clear than the FM^3. From the quantitative point of view, we observe from Table 2 (see the row of Fig. 8, especially, noting the drawing produced by FM^3 has the zero angular resolution problem) that our method obviously improves the drawing from the three aesthetic measures.

From Figs. 8(c) and (d), we observe that there still exist some edge crossings. That is, our method still cannot handle the local minimal problem of the original drawing in Fig. 8(c). Hence, we wonder whether our method has the capacity of not producing more edge crossings. Consider Fig. 9(a), in which each triangle of graph *sierpinski_08* is manually drawn as a right triangle, and hence there is no edge crossing in the drawing. Fig. 9(b) shows the drawing using our method taking Fig. 9(a) as the input. We observe from the local views shown in Figs. 9(c) and (d) that our method may not produce more edge crossings.

In Fig. 10, for graph *crack*, the drawings produced by the FM^3 and our method are given in (a) and (b), respectively; the local views for two parts of graph *crack* using the FM^3 and our method are given in (c)–(f). From the visualization, it is not easy to say which method is better if observing only (a) and (b). But, if the local views (c) and (d) as well as (e) and (f) are referred, we observe that our drawing has a high degree of symmetry (see the comparison between (c) and (d)), and has a more smooth drawing boundary (see the comparison between (c) and (d) as well as (e) and (f)).

5. Conclusion

Different from the conventions of force-directed methods, a new force-directed method based on edge–edge repulsion for generating a straight-line drawing not only preserving the original properties of a high degree of symmetry and uniform edge length but also without zero angular resolution has been proposed and implemented. In addition, the drawings generated by our method usually have larger average angular resolution. The simplified formulas of edge–edge repulsion can be derived from the theory of potential fields.

A line of future work is to overcome the local minimal problem for the vertices on the drawing boundary (which also poses difficulties for the conventional force-directed methods) by multilevel techniques or using optimal heuristics, such as simulated annealing, genetic algorithm, etc. It is also of importance and interest to provide more experimental results on graphs of huge size and theoretical results on the power of the model of edge–edge repulsion.

Acknowledgments

The authors thank the anonymous referees for comments that improved the content as well as the presentation of this paper. The first author (Chun-Cheng Lin) was supported in part by NSC 98-2218-E-009-026-MY3. The second author (Hsu-Chun Yen) was supported in part by Institute for Information Industry, Taiwan, under the project “Next Generation Telematics System and Innovative Applications/Services Technologies”, and NSC100-2221-E-002-132-MY3.

References

[1] F. Bertault, A force-directed algorithm that preserves edge-crossing properties, Information Processing Letters 74 (1–2) (2000) 7–13.

[2] J.-H. Chuang, N. Ahuja, An analytically tractable potential field model of free space and its application in obstacle avoidance, IEEE Transactions on System, Man, and Cybernetics 28 (5) (1998) 729–736.
[3] R. Davidson, D. Harel, Drawing graphs nicely using simulated annealing, ACM Transactions on Graphics 15 (1996) 301–331.
[4] P. Eades, A heuristic for graph drawing, Congress Numerantium 42 (1984) 149–160.
[5] P. Eades, X. Lin, Spring algorithms and symmetry, Theoretical Computer Science 240 (2) (2000) 379–405.
[6] M. Formann, J. Haralambides, M. Kaufmann, F.T. Leighton, A. Simvonis, E. Welzl, G. Woeginger, Drawing graphs in the plane with high resolution, SIAM Journal on Computing 22 (5) (1993) 1035–1052.
[7] A. Frick, A. Ludwig, H. Mehldau, A fast adaptive layout algorithm for undirected graphs, Graph Drawing '94, Lecture Notes in Computer Science, vol. 894, Springer, 1995, pp. 388–403.
[8] Y. Frishman, A. Tal, Multi-level graph layout on the GPU, IEEE Transactions on Visualization and Computer Graphics 13 (6) (2007) 1310–1317.
[9] Y. Frishman, A. Tal, MOVIS: a system for visualizing distributed mobile object environments, Journal of Visual Languages and Computing 19 (3) (2008) 303–320.
[10] Y. Frishman, A. Tal, Online dynamic graph drawing, IEEE Transactions on Visualization and Computer Graphics 14 (4) (2008) 727–740.
[11] T. Fruchterman, E. Reingold, Graph drawing by force-directed placement, Software—Practice and Experience 21 (1991) 1129–1164.
[12] P. Gajer, M.T. Goodrich, S.G. Kobourov, A multi-dimensional approach to force-directed layouts of large graphs, Computational Geometry: Theory and Applications 29 (1) (2004) 3–18.
[13] S. Hachul, M. Jünger, Drawing large graphs with a potential-field-based multilevel algorithm, Graph Drawing 2004, Lecture Notes in Computer Science, vol. 3383, Springer, 2005, pp. 285–295.
[14] S. Hachul, M. Jünger, Large-graph layout algorithms at work: an experimental study, Journal of Graph Algorithms and Applications 11 (2) (2007) 345–369.
[15] D. Harel, Y. Koren, A fast multi-scale method for drawing large graphs, Journal of Graph Algorithms and Applications 6 (3) (2002) 179–202.
[16] S.-H. Hong, D. Merrick, H.A. do Nascimento, Automatic visualisation of metro maps, Journal of Visual Languages and Computing 19 (3) (2008) 303–320.
[17] M. Huang, P. Eades, J. Wang, On-line animated visualization of huge graphs using a modified spring algorithm, Journal of Visual Languages and Computing 9 (6) (1998) 345–623.
[18] T. Kamada, S. Kawai, An algorithm for drawing general undirected graphs, Information Processing Letters 31 (1989) 7–15.
[19] S. Kirkpatrick, C.D. Gelatt, M.P. Vecchi, Optimization by simulated annealing, Science 220 (1983) 671–680.
[20] S. Kobourov, K. Wampler, Non-Euclidean spring embedders, IEEE Transactions on Visualization and Computer Graphics 11 (6) (2005) 757–767.
[21] G. Kumar, M. Garland, Visual exploration of complex time-varying graphs, IEEE Transactions on Visualization and Computer Graphics 12 (5) (2006) 805–812.
[22] C.-C. Lin, H.-C. Yen, J.-H. Chuang, Drawing graphs with nonuniform nodes using potential fields, Journal of Visual Languages and Computing 20 (6) (2009) 385–402.
[23] K. Misue, P. Eades, W. Lai, K. Sugiyama, Layout adjustment and the mental map, Journal of Visual Languages and Computing 6 (2) (1995) 183–210.
[24] H.C. Purchase, Metrics for graph drawing aesthetics, Journal of Visual Languages and Computing 13 (5) (2002) 501–516.
[25] K. Sugiyama, K. Misue, Graph drawing by the magnetic spring model, Journal of Visual Languages and Computing 6 (1995) 217–231.
[26] W.T. Tutte, Convex representations of graphs, Proceedings of the London Mathematical Society 10 (1960) 304–320.
[27] W.T. Tutte, How to draw a graph, Proceedings of the London Mathematical Society 13 (1963) 743–768.
[28] C. Walshaw, A multilevel algorithm for force-directed graph-drawing, Journal of Graph Algorithms and Applications 7 (3) (2003) 253–285.