Journal of Visual Languages and Computing 20 (2009) 385–402

Contents lists available at ScienceDirect

Journal of Visual Languages and Computing

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

Drawing graphs with nonuniform nodes using potential fields ☆

Chun-Cheng Lin [a,b], Hsu-Chun Yen [a,c,*,1], Jen-Hui Chuang [d]

[a] Department of Electrical Engineering, National Taiwan University, Taipei 106, Taiwan, ROC
[b] Department of Computer Science and Information Engineering, National Kaohsiung University of Applied Sciences, Kaohsiung 807, Taiwan, ROC
[c] Department of Computer Science, Kainan University, Taoyuan 338, Taiwan, ROC
[d] Department of Computer Science, National Chiao-Tung University, Hsinchu 300, Taiwan, ROC

ARTICLE INFO

Article history:
Received 1 March 2008
Received in revised form
9 December 2008
Accepted 24 April 2009

Keywords:
Graph drawing
Potential field

ABSTRACT

Graphs with nonuniform nodes arise naturally in many real-world applications. Although graph drawing has been a very active research in the computer science community during the past decade, most of the graph drawing algorithms developed thus far have been designed for graphs whose nodes are represented as single points. As a result, it is of importance to develop drawing methods for graphs whose nodes are of different sizes and shapes, in order to meet the need of real-world applications. To this end, a *potential field* approach, coupled with an idea commonly found in force-directed methods, is proposed in this paper for drawing graphs with nonuniform nodes in 2-D and 3-D. In our framework, nonuniform nodes are uniformly charged, while edges are modelled by springs. Using certain techniques developed in the field of potential-based *path planning*, we are able to find analytically tractable procedures for computing the repulsive force and torque of a node in the potential field induced by the remaining nodes. The crucial feature of our approach is that the *rotation* of every nonuniform node and the *multiple edges* between two nonuniform nodes are taken into account. In comparison with the existing algorithms found in the literature, our experimental results suggest this new approach to be promising, as drawings of good quality for a variety of moderate-sized graphs in 2-D and 3-D can be produced reasonably efficiently. That is, our approach is suitable for moderate-sized interactive graphs or larger-sized static graphs. Furthermore, to illustrate the usefulness of our new drawing method for graphs with zero-sized nodes, we give an application to the visualization of hierarchical clustered graphs, for which our method offers a very efficient solution.

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 (see, e.g., [1]). In visualizing graphs associated with real-world entities, it is common to annotate each node with some labels (such as name, attribute), and a good way to display such information (in the form of text or icon) is to fill the information inside the drawing of the node. As a result, it is of importance and interest to be able to cope with nodes of different sizes and shapes in graph drawing.

Among the various graph drawing techniques reported in the literature, the so-called *force-directed methods* (see, e.g., [2–11]) have received much attention and have

☆ A preliminary version of this work was presented at the 11th International Symposium on Graph Drawing, September 21–24, 2003, Perugia, Italy.

* Corresponding author at: Department of Electrical Engineering, National Taiwan University, Taipei 106, Taiwan, ROC.
E-mail address: yen@cc.ee.ntu.edu.tw (H.-C. Yen).

[1] Research supported in part by NSC Grant 97-2221-E-002-094-MY3.

1045-926X/$ - see front matter © 2009 Elsevier Ltd. All rights reserved.
doi:10.1016/j.jvlc.2009.04.006

become very popular for drawing 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. According to the experimental and theoretical results of [12,13], graph drawing methods based on the force-directed strategy usually enjoy the merit of producing graph layouts with a high degree of symmetry and uniformity of the edge length.

Unfortunately, most of the conventional force-directed graph drawing algorithms only deal with graphs with zero-sized nodes. A naive extension of conventional drawing algorithms by plugging in objects of nonzero size for point nodes in the drawing is unlikely to produce a high quality drawing. In view of this, it becomes apparent that a new line of research has to be carried out in hope of narrowing the gap between the conventional force-directed algorithms and graphs consisting of nodes of various sizes and shapes that are frequently encountered in the real world. About the previous work, Harel and Koren [14] extended the conventional force-directed methods to develop three new algorithms to draw graphs with either elliptic or rectangular nodes. Huang and Lai [15] proposed a force-directed graph drawing algorithm for graphs with rectangular nodes in which they guaranteed that there is no overlapping between any pair of rectangular nodes.

In this paper, we present a new approach, combining the force-directed strategy as well as the theory of *potential fields*, for drawing graphs with nonuniform nodes. Although there exist other conventions (e.g., like hierarchical-layered drawing [16] and orthogonal drawing [17]) for drawing graphs with nonuniform nodes, we for the first time take node rotations into account when implementing our force-directed-based drawing algorithm. The concept of potential fields has already found appli-cations in a variety of areas in computer science and engineering, such as path planning [18,19], among others. In our setting, each edge is modelled by a spring (like in conventional force-directed methods) and the boundary of every nonuniform node is uniformly charged. We consider both 2-D and 3-D drawings of nonuniform nodes, each of which is represented by a polygon (in 2-D) or polyhedron (in 3-D). Other shapes of objects can be approximated by their enclosing polygons (or polyhedrons). For a given nonuniform node, the forces on the node come from two sources: one resulting from being connected by springs, and the other is caused by being present in the potential field induced by the remaining nodes. From the efficiency viewpoint, it turns out that computing the repulsive force and torque of a node in the potential field induced by the remaining nodes is analytically tractable, as our subsequent discussion indicates.

In comparison with [14], our approach differs in the following. First, we deal with graphs in both 2-D and 3-D, whereas in [14], only 2-D graphs with either elliptical or rectangular nodes are investigated. Second and perhaps a more significant difference is that when measuring the movement of a nonuniform node, *rotation* is taken into account rather naturally in our framework, while the algorithm in [14] lacks this capability. In the process of reaching equilibrium, the degree of inclination of a nonuniform node can be adjusted while moving from one position to another. By doing so, the final drawing has a tendency to display a high degree of symmetry or occupy a small area, which will be shown by our experimental results. Consider various drawings of a graph with nonuniform nodes shown in Fig. 1, which can be classified into four styles, depending on where the end points of the drawing edges are emanated from and whether the orientation of each nonuniform node can be modified. In Figs. 1(a) and (b), every edge connects two corners between a pair of nonuniform nodes; in Figs. 1(c) and (d), each edge connects the centers of two nonuniform nodes. Figs. 1(b) and (d) allow each nonuniform node with angle of inclination, but Figs. 1(a) and (c) do not. The algorithm in [14] can only produce the drawing depicted in Fig. 1(c). In contrast, our approach can produce all four drawing styles simply by altering the values of some parameters in our drawing algorithm. In view of this example, it is clear that *rotation* plays a crucial role in producing a nicer drawing for a graph with nonuniform nodes. Another feature of our approach is that *multiple edges* can be dealt with in a natural way. By allowing an edge to come out from a corner or the interior of a nonuniform node rather than from its center, our approach can produce drawings of graphs with multiple edges. Note that Figs. 1(a) and (b) allow multiple edges between two nonuniform nodes.

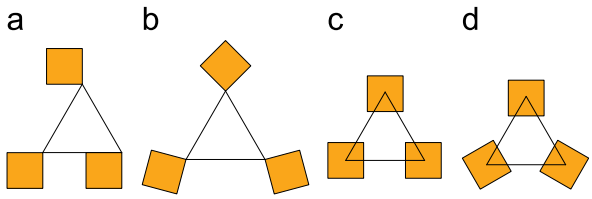

Fig. 1. Various drawings of a graph with three square-shaped nodes and three edges. (a) Style $\mathbb{S}_1$. (b) Style $\mathbb{S}_2$. (c) Style $\mathbb{S}_3$. (d) Style $\mathbb{S}_4$.

Aside from the fact that natural objects tend to be of nonzero size, another motivation behind the need of handling graphs with nonuniform nodes has to do with drawing *clustered graphs*, which are frequently used for in the visualization of huge graphs. Assuming that each cluster of a clustered graph has already been drawn nicely, our idea (motivated by [20]) is to view the convex hull (polygon) of each nicely drawn cluster as a nonuniform node first. The next step is to produce a nice drawing for the graph with its constituent clusters represented by nonuniform nodes. In the final phase of the drawing procedure, the details of each cluster are then plugged into the corresponding nonuniform node. The drawing approach by abstracting out the details of each cluster runs very efficiently, making the method suitable for graph drawing in dynamic scenarios (see, e.g., [21]).

We have developed a prototype system to compare our new graph drawing method with other existing approaches empirically. Our experimental results look

promising as our subsequent discussion indicates. However, our method is slower than the previous methods for moderate-sized graphs without rotations of nonuniform nodes, and hence our method is suitable for moderate-sized interactive graphs or larger-sized static graphs (note that graph drawing algorithms may be designed for the static or dynamic scenarios, e.g., Biedl and Kaufmann [22] presented algorithms for generating orthogonal drawings in the static and dynamic scenarios). Therefore, finding a more efficient algorithm for drawing graphs with nonuniform nodes remains open. In addition, note that some applications do not allow rotations of nonuniform nodes. Therefore, we give examples applied to UML class diagrams by using our approach in this paper.

The rest of this paper is organized as follows. Section 2 gives some preliminaries. In Section 3, the theory behind potential fields in graph drawing is developed in depth. Section 4 gives some experimental results in 2-D and 3-D, and Section 5 gives an application. Finally, a conclusion is given in Section 6.

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 nodes 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 node 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 $D(v)$.

A *nonuniform node* refers to a *polygon* in 2-D or a *polyhedron* in 3-D. Points of zero size are regarded as degenerated nonuniform nodes, and to draw other types of nonuniform nodes (such as circles and ovals in 2-D or balls in 3-D), we simply use polygons or polyhedra to approximate such nonuniform nodes. Note that nonuniform nodes may be of different shapes and different sizes. A *graph with nonuniform nodes* (GNN) is a pair $G = (N, E)$ such that N is the set of nonuniform nodes and E is the set of edges. For satisfying different applications, we can classify drawings of a GNN into four categories, as shown in Fig. 1, depending on where the end points of the drawing edges are emanated from and whether the orientation of every nonuniform node can be altered. Through the rest of this paper, we denote $\mathbb{S}_1$–$\mathbb{S}_4$ as the drawing styles of Figs. 1(a)–(d), respectively, in which:

- $\mathbb{S}_1$: every drawing edge connects two corners between a pair of nonuniform nodes, and nonuniform nodes are not allowed to be rotated.
- $\mathbb{S}_2$: every drawing edge connects two corners between a pair of nonuniform nodes, and nonuniform nodes are allowed to be rotated.
- $\mathbb{S}_3$: every drawing edge connects the centers of two nonuniform nodes, and nonuniform nodes are not allowed to be rotated.
- $\mathbb{S}_4$: every drawing edge connects the centers of two nonuniform nodes, and nonuniform nodes are allowed to be rotated.

It should be noticed that $\mathbb{S}_1$ and $\mathbb{S}_2$ allow multiple edges between two nonuniform nodes. Here, we assume that the ports to which the edges are connected are given.

A *hierarchical clustered graph* $C = (G, H)$ is a graph with its nodes clustered recursively in a way that each leaf (resp. internal node) of the rooted tree H called *inclusion tree* corresponds to a node (resp. a subset of nodes called "cluster") of G. In the drawing of a hierarchical clustered graph, every node of G is drawn as a point, every edge of G is drawn as a simple curve, and every internal node of H is drawn as a simple closed region bounding its descendents. A *c-planar* drawing [23], i.e., the drawing of a hierarchical clustered graph without any crossing between edge pairs or edge/region pairs, is of importance in the fields of graph drawing or information visualization. Although there exist polynomial time algorithms for drawing restricted types of *c*-planar graphs [24], whether *c*-planar drawings can be generated efficiently in the general case remains a challenging unanswered problem. Dogrusöz et al. [21] (resp., [20]) applied force-directed strategy (resp., layered-drawing strategy) to producing the straight-line drawings of hierarchical clustered graphs where every internal node of the inclusion tree is drawn as a rectangular box.

In order to draw hierarchical clustered graphs, Eades and Huang [25] devised a force-directed model where the edges between two nodes that belong to the same cluster are replaced by stronger springs; the edges between two nodes that belong to different clusters are replaced by weaker springs. By this setting, the nodes of the same cluster are drawn closer; the nodes of different clusters are drawn farther. Hence, the relationship of clustering can be observed in visualization even if we do not draw the simple closed regions of the internal nodes of the inclusion tree. In this paper, we propose an approach to drawing clustered graphs. Although this approach cannot avoid crossings between edge pairs and edge/region pairs, it may run more efficiently than the force-directed strategy in [25] applied to clustered graphs. In this paper, we show that our approach gives nicer quality than theirs.

2.2. Defining the problem

The drawing problem considered in this paper is addressed as follows. Suppose we begin with an initial drawing of a GNN, one is required to produce a nice drawing of the GNN with respect to the following aesthetic criteria: no overlapping between any two nonuniform nodes, no overlapping between any pair of nonuniform node and edge, symmetry, uniform edge length, and minimizing the size of the drawing. In addition, the efficiency of the drawing algorithm is also factored into our design.

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 nodes is divided into two sets, a set of fixed nodes and a set of free nodes. By nailing down the fixed nodes as a strictly convex polygon and then placing each free node at the barycenter of its neighbor in each iteration, the model can yield a nice drawing.

Furthermore, since the introduction of the simple force-directed method by Eades in [2] (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:

$$f_a(\mathbf{d}_{uv}) = (c_a * \log(|\mathbf{d}_{uv}|/l))\mathbf{d}_{uv}/|\mathbf{d}_{uv}|,$$
$$f_r(\mathbf{d}_{uv}) = -(c_r/|\mathbf{d}_{uv}|^2)\mathbf{d}_{uv}/|\mathbf{d}_{uv}|, \quad (1)$$

where c_a and c_r are scaling constants, l is the given spring natural length, and $\mathbf{d}_{uv}$ is the vector from node u to node v. Besides above considering repulsive forces between every node–node pair, the models of considering repulsive forces between every node–edge pair [3], and every edge–edge pair [4] also were developed to preserve the edge crossing property and handle the zero angular resolution problem, respectively.

Subsequently, a number of variations of spring algorithms have been proposed to improve the performance as well as the drawing quality. Notable examples include [5–9]. The algorithm in [5] 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 node initially has a temperature value and the value decreases by computing some cooling function at the end of each iteration. Until all nodes cool down to some constant value, the algorithm stops to attain a nice drawing. In a similar work, the approach of GEM [6] makes use of the history of the moving trajectory of all nodes to compute the temperature.

As the algorithm in [7] indicates, the simulated annealing [28] approach also plays a constructive role in graph drawing. The basic idea is as follows. Given a evaluation function consisting a set of criteria, e.g., number of edge crossings, the temperature of every node decreases at the end of each iteration, but the temperature of a node may increase when the return value of the evaluation function for the new position of the node is worse than that regarding the original position of the node. The algorithm terminates when the temperature of every node is below some predefined value. In addition, Kamada and Kawai [8] 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 [9] 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 nodes). Recently, the multi-scale approaches [10,11] make them successful with much larger graphs (over 10,000 nodes).

2.4. Previous work on graph drawings with nonuniform nodes

Harel and Koren [14] extended the conventional force-directed methods to develop three new algorithms to draw graphs with either elliptic or rectangular nodes. Rectangular nodes can be viewed as the special case of nonuniform nodes (polygons) in 2-D, so the force-directed strategy in [14] for drawing graphs with rectangular nodes is related to our work, and will be compared in this paper. The spring and repulsive force formula of their approach are calculated according to $f_a(u,v) = l_b(u,v)^2/Len$ and $f_r(u,v) = Len^2/\max(l_b(u,v),\varepsilon)$, respectively, where $l_b(u,v)$ is the shortest distance between the boundaries of rectangular nodes u and v, and Len and ε are positive constants.

Other work concerning the force-directed methods for drawing graphs with rectangular nodes includes [15,29,30], which mainly focus on only the aesthetic criterion that removes the overlaps between any pairs of nodes, and hence, the results of our paper are not compared with those work.

3. Force-directed method using potential fields

To draw GNNs, our force-directed approach is based upon the idea of replacing each edge by a spring and assuming that the border of every nonuniform node is uniformly charged. In this setting, the repulsive forces among nonuniform nodes avoid crossings between nonuniform nodes, and the spring forces pull nonuniform nodes closer. A nice drawing would be generated when the corresponding model reaches an equilibrium between the repulsive forces due to the charged nonuniform nodes and attractive forces due to springs.

In each iteration of our algorithm, each nonuniform node moves and rotates, respectively, according to the force (consisting of repulsive and spring forces) and the torque (consisting of repulsive and spring torque) acting at that node. In what follows, since we use Eq. (1) as the spring force formula, we mainly focus on deriving the repulsive forces as well as the repulsive and spring

torques. Note that the *torque* acting at each nonuniform node is a measure to cause that node to rotate counterclockwise about an axis, called *pivot point* of that node, i.e., the rotation center of the node.

In what follows, we only consider the style $\mathbb{S}_2$ (the remaining styles are similar and simpler) and elaborate on the foundations of the theory behind the use of potential fields in graph drawing. The reader is referred to [18,19] for more about potential fields as well as some of the detailed derivations of formulas involved in our subsequent discussion.

3.1. 2-D model

Consider two polygons (representing two nonuniform nodes) A and B in 2-D connected by two edges $\overline{a_1b_2}$ and $\overline{a_3b_2}$ as shown in Fig. 2(a). Polygon A has vertices $a_1,\ldots,a_6$, and polygon B has b_1, b_2, and b_3 along their boundaries. Each polygon is associated with a pivot point exerted by repulsive and attractive forces, and shifts and rotations are carried out with respect to this pivot point. For instance, in Fig. 2 e_a and e_b are the pivot points of polygons A and B, respectively. In xy-plane, for any two points of unit charge and of distance r apart, the *Newtonian potential* is defined as $V = 1/r$ and the Newtonian potential of a point q with respect to a uniformly charged line segment $\overline{a_1a_2}$ is $V = \int_{a_1}^{a_2} 1/r(x)\,dx$ where $r(x)$ is the distance between q and x. Fig. 3 shows uniformly charged edges along with the equi-potential contours.

Consider two line segments $\overline{a_1a_2}$ and $\overline{b_1b_2}$ on the new coordinate system (uv-plane), placing a_1 on the original point and $\overline{a_1a_2}$ on the positive u-axis (v-axis is perpendicular to u-axis), after coordinate transformation as shown in Fig. 2(b). $Q(\mu,\eta)$ is some point on $\overline{b_1b_2}$, and $v = au + b$ (where a and b are constants) is the line equation representing $\overline{b_1b_2}$. d is the length of $\overline{a_1a_2}$, and u_1 and u_2 are the projection points of the end points of $\overline{b_1b_2}$. After a sequence of computations of the above Newtonian potential function such as negative gradient, integral, coordinate transformation, and integral, the repulsive force on $\overline{b_1b_2}$ due to $\overline{a_1a_2}$, i.e., the repulsive force between two line segments, along the u-axis and the v-axis can be respectively expressed as

$$f_{r_u} = f_{r_u}(u_2) - f_{r_u}(u_1), \tag{2}$$

$$f_{r_v} = f_{r_v}(u_2) - f_{r_v}(u_1), \tag{3}$$

where

$$f_{r_u}(u) = \log\frac{f_1'(u)/2 + \sqrt{1+a^2}f_1^{1/2}(u)}{f_2'(u)/2 + \sqrt{1+a^2}f_2^{1/2}(u)},$$

$$\begin{aligned} f_{r_v}(u) = \Bigg(&\frac{1}{a}\log\frac{f_1'(u)/2 + \sqrt{1+a^2}f_1^{1/2}(u)}{f_2'(u)/2 + \sqrt{1+a^2}f_2^{1/2}(u)} \\ &+ \frac{\sqrt{1+a^2}(b+d)}{a(ad+b)} \\ &\times \log\frac{2(ad+b)(d-u) + 2(ad+b)f_1^{1/2}(u)}{au+b} \\ &+ \frac{\sqrt{1+a^2}}{a}\log\frac{-2bu + 2bf_2^{1/2}(u)}{au+b}\Bigg), \end{aligned}$$

where $f_1(u) = (au+b)^2 + (u-d)^2$ and $f_2(u) = (au+b)^2 + u^2$. And the repulsive torque on $\overline{b_1b_2}$ due to $\overline{a_1a_2}$, i.e., the repulsive torque between two line segments, with respect to the pivot point $e_b = (e_b^u, e_b^v)$ can be expressed as $\tau_r^{e_b} = \tau_r^{e_b}(u_2) - \tau_r^{e_b}(u_1)$ where

$$\begin{aligned} \tau_r^{e_b}(u) = \Bigg(&b - e_b^v - \frac{ba^2+b^2}{1+a^2} - \frac{e_b^u + b}{a}\Bigg) \\ &\times \log\frac{f_1'(u)/2 + \sqrt{1+a^2}f_1^{1/2}(u)}{f_2'(u)/2 + \sqrt{1+a^2}f_2^{1/2}(u)} \\ &- \left(\frac{b^2}{a^2} + e_b^u d + \frac{be_b^u + bd}{a}\right)\frac{\sqrt{1+a^2}}{b+da} \\ &\times \log\frac{2(ad+b)(d-u) + 2(ad+b)f_1^{1/2}(u)}{au+b} \\ &+ \left(\frac{b}{a} + e_b^u\right)\frac{\sqrt{1+a^2}}{a}\log\frac{-2bu + 2bf_2^{1/2}(u)}{au+b} \\ &+ \frac{\sqrt{1+a^2}}{a}(f_1^{1/2}(u) - f_2^{1/2}(u)), \end{aligned} \tag{4}$$

whose direction is $i_z = i_u \times i_v$, in which i_u and i_v are the unit vectors of u-axis and v-axis, respectively.

In addition, the torques due to the spring forces should be considered. For example, in Fig. 2, suppose the attractive force on the point b_2 due to the spring edge $\overline{b_2a_1}$ is equal to $f_a(\overrightarrow{b_2a_1})$, from Eq. (1), and B rotates with respect to the pivot point e_b. The torque with respect to point e_b due to the attractive force $f_a(\overrightarrow{b_2a_1})$ from spring $\overline{b_2a_1}$, on point b_2, is equal to the cross product

$$(b_2 - e_b) \times f_a(\overrightarrow{b_2a_1}). \tag{5}$$

Note that it is inappropriate to choose the center of shape as the pivot point of a polygon because the forces and torques due to springs are computed by the spring

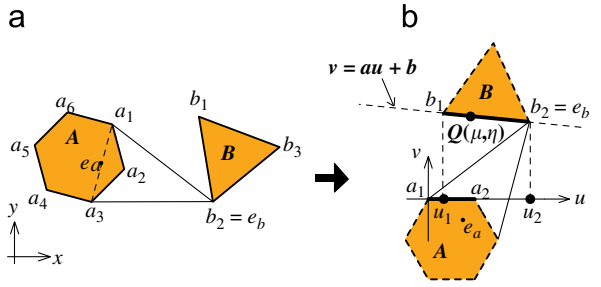

Fig. 2. Coordinate transformation of a 2-D graph with two nonuniform nodes A and B where e_a and e_b are the pivot points. (a) The original coordinate system (xy-plane). (b) The new coordinate system (uv -plane) after the transformation.

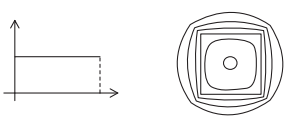

Fig. 3. Potential contours due to uniformly charged square boundary (left: charge distribution; right: equi-potential contours).

lengths and the springs are not connected to the centers of the shape. Instead, the mean of the coordinates of the vertices connected by springs is selected as the pivot point. Consider Fig. 2 for instance. The polygon A has vertices a_1 and a_3 connected by springs, and the midpoint of a_1 and a_3 is the pivot point of A. The polygon B only has a single vertex b_2 connected by a spring, and hence b_2 is the pivot point of B.

In view of the above derivations, the force between two polygons can be derived. For example, in Fig. 2, the total repulsive force on polygon B due to polygon A is equal to

$$\mathbf{F}_r = \sum_{i\in\mathscr{B}(B)} \sum_{j\in\mathscr{B}(A)} f_r^{ij}, \tag{6}$$

where $\mathscr{B}(X)$ is the set of border lines of polygon X, and $f_r^{ij} = (f_{r_u}^{ij}, f_{r_v}^{ij})$ is the repulsive force on border line segment i due to j in which $f_{r_u}^{ij}$ (resp., $f_{r_v}^{ij}$) is the component along the u-axis (resp., v-axis) computed by Eq. (2) (resp., Eq. (3)).

On the other hand, the total spring force on polygon B due to all of the springs is equal to

$$\mathbf{F}_a = \sum_{p\in\mathscr{P}(B)} \sum_{(p,q)\in E} f_a(\overrightarrow{pq}), \tag{7}$$

where $\mathscr{P}(B)$ is the set of vertices of B, E is the edge set, p is a vertex of B, q is a vertex of some polygon, and f_a is the spring force defined in Eq. (1).

In summary, the total force applied to polygon B due to the remaining polygons and springs is equal to the sum of the repulsive force (Eq. (6)) and the attractive force (Eq. (7)) as follows:

$$\mathbf{F} = \sum_{N-\{B\}} (\mathbf{F}_r + \mathbf{F}_a). \tag{8}$$

Similar results can be obtained for the total torque $\mathbf{T}$ with respect to the pivot point e_b including repulsive and spring torques.

3.2. 3-D model

Now we turn our attention to drawing nonuniform nodes in 3-D in the framework of potential fields.

In Fig. 4(a), consider two polyhedra (representing nonuniform nodes in 3-D) A and B, where q is a *sample point* of B, and S is a plane surface of A surrounded by border lines C_1–C_4 and ∂S denotes the boundary of S. Fig. 4(b) only illustrates q and S where $\hat{\mathbf{u}}_i$ and $\hat{\mathbf{l}}_i$ are along the (outward) normal and tangential directions of ∂S, and $\hat{\mathbf{n}}$ is its surface normal, where $\hat{\mathbf{n}} = \hat{\mathbf{u}}_i \times \hat{\mathbf{l}}_i$. Consider a generalized potential function (i.e., higher-order potential function) of the kind used in [18] for reasoning about motion planning.

According to the derivation from [18], the repulsive force at point q due to S can be expressed in the following analytical from (see [18] for more details):

$$\begin{aligned} F_r &= (F_{r_x}, F_{r_y}, F_{r_z}) \\ &= -\sum_i \nabla_i [\Phi_{3,i}(x_i = l_i^+, y_i, z) \\ &\quad - \Phi_{3,i}(x_i = l^-, y_i, z)] + \frac{\alpha}{z^2} i_z, \end{aligned} \tag{9}$$

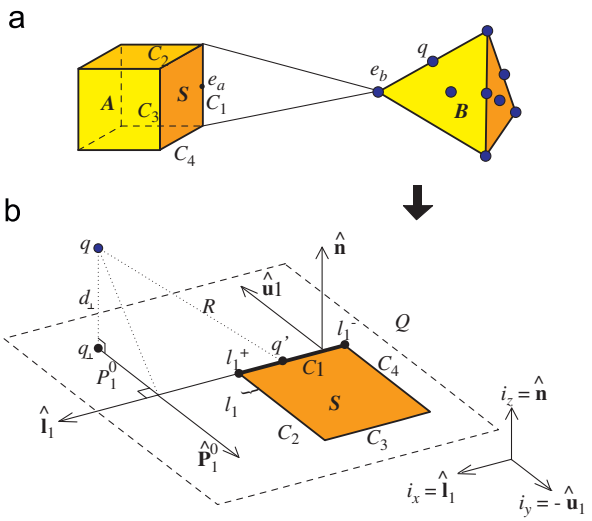

Fig. 4. A 3-D graph consisting of two polyhedra. (a) Uniformly selected sample points on the surfaces of the right polyhedron B. (b) Geometric quantities associated with a sample point q from B and a surface S of A, where Q is the plane containing S.

where, for each C_i, the triple x_i, y_i, and $z = d_\perp > 0$ are measured along $\hat{\mathbf{l}}_i$, $-\hat{\mathbf{u}}_i$, and $\hat{\mathbf{n}}$, respectively, with the origin located at the projection of q on C_i; the gradient ∇_i is determined by the coordinate system in which C_i resides, and i_z is the unit vector of z, i.e., $\hat{\mathbf{n}}$. And then Eq. (9) can be resolved as follows:

$$\begin{aligned} \frac{\partial \Phi_3}{\partial x} &= \frac{y}{(x^2+y^2)\sqrt{x^2+y^2+z^2}}, \\ \frac{\partial \Phi_3}{\partial y} &= \frac{-x(x^2+2y^2+z^2)}{(x^2+y^2)(y^2+z^2)\sqrt{x^2+y^2+z^2}}, \\ \frac{\partial \Phi_3}{\partial z} &= -\frac{\tan^{-1}\dfrac{xz}{y\sqrt{x^2+y^2+z^2}}}{z^2} \\ &\quad + \frac{xy(x^2+y^2)}{z(x^2+y^2)(y^2+z^2)\sqrt{x^2+y^2+z^2}}. \end{aligned}$$

In 3-D, the repulsive force between two polyhedra A and B is the sum of the repulsive forces of every possible pair of polygonal surfaces in A and B. However, the repulsive force between two polygonal surfaces is a complicated four-order integral. In order to reduce the complexity of computing repulsive forces between A and B, we uniformly select *sample points* on the surfaces of each polyhedron. Hence the repulsive force on B due to A is the sum of the repulsive forces at each sample point q due to every polygonal surface of B. See Fig. 4(a) for an illustrating example. The complexity of computing the forces between two polyhedra can be reduced because the forces at a single point due to polygons is a two-order integral.

Assume that $\mathscr{SP}(X)$ is the set of sample points of polyhedron X and $\mathscr{S}(X)$ is set of polygonal surfaces of polyhedron X. The repulsive force at a sample point p_i on B due to A is the sum of the repulsive forces at p_i due to n (all) polygonal surfaces on A. Thus, the repulsive force $\mathbf{F}_r$,

standing for the force at B due to A, is the sum of the repulsive forces for all sample points on B, as follows:

$$\mathbf{F}_r = \sum_{i \in \mathscr{S}\mathscr{P}(B)} \sum_{j \in \mathscr{S}(A)} F_r^{ij}, \tag{10}$$

where the repulsive force at p_i due to the j-th polygonal surface of A is denoted as $F_r^{ij} = (F_{r_x}^{ij}, F_{r_y}^{ij}, F_{r_z}^{ij})$, which is computed according to Eq. (9).

On the other hand, the total attractive force on polyhedron B due to all of the springs is equal to

$$\mathbf{F}_a = \sum_{p \in \mathscr{P}(B)} \sum_{(p,q) \in E} f_a(\overrightarrow{pq}), \tag{11}$$

where p is a vertex of B, and $f_a = (f_{a_x}, f_{a_y}, f_{a_z})$ in which f_{a_x}, f_{a_y}, f_{a_z} are the projections of the spring force defined in (1) along x-, y-, and z-axes, respectively.

Thus, the total force on polyhedron B due to all springs and the other polyhedra except for B can be expressed as the sum of (10) and (11), as follows:

$$\mathbf{F} = \sum_{N-\{B\}} (\mathbf{F}_r + \mathbf{F}_a). \tag{12}$$

The repulsive torque (13) is the cross product of the repulsive force F_r^i at sample point p_i and distance vector X_i which is the distance from the reference point e_b to p_i. So the total repulsive torque (14) is the sum of torques due to all of the sample points.

$$T_r^i = X_i \times F_r^i, \tag{13}$$

$$\mathbf{T}_r = \sum_{i \in \mathscr{S}\mathscr{P}(B)} T_{r_x}^i. \tag{14}$$

The total attractive torque $\mathbf{T}_a$ can be derived similarly. Consequently, the total torque (15) on polyhedron B due to all springs and the other polyhedra except for B can be expressed as

$$\mathbf{T} = \mathbf{T}_r + \mathbf{T}_a. \tag{15}$$

Note that an important aspect of our approach lies in the formulas derived in our potential field model being analytically tractable, making our algorithm computationally tractable.

4. Implementation and experimental results

Based on the theory of potential fields detailed in the previous section, in this section we propose our algorithm, detail the implementation of our algorithm, and develop a prototype system for drawing GNNs in 2-D and 3-D and give some experimental results.

4.1. Algorithm

Our drawing algorithm, based upon the theory of potential fields, is presented in Algorithm 1, which has a structure similar to those found in conventional force-directed methods. The algorithm operates iteratively to generate a continuous process from the initial drawing to the final drawing, as shown in Fig. 7. It is worthy of pointing out that if the drawing of each iteration is rendered, the animated process allows the user to predict the dynamics of the drawing, which meets the requirement in information visualization.

Algorithm 1. POTENTIAL_SPRING (*graph G*)

```
1:  assign the initial locations of nonuniform nodes of G according to a
    given (or random) initial drawing
2:  while converged ≠ 1 or the maximal number of iterations is reached
    do
3:    converged ← 1
4:    oldPosn ← newPosn
5:    for each nonuniform node N_i in G do
6:      Calculate the total force F_i of node N_i, consisting of:
        (a) the repulsive force due to the other nonuniform nodes
    located within the neighborhood of node N_i (the circle in 2-D (ball in
    3-D) with center N_i and a small radius γ) according to spring force
    formula (1), and
        (b) the spring force due to its adjacent springs according to
    repulsive force formulas (2) and (3) in 2-D or (9) in 3-D;
7:      Calculate its total torque T_i according to 2-D formulas (4) and
    (5) or 3-D formulas (13) and (14);
8:      Save F_i and T_i;
9:    endfor
10:   for each nonuniform node N_i in G do
11:     Simultaneously move and rotate every nonuniform node N_i
    according to min(c_1 × F_i, t_1) and min(c_2 × T_i, t_2), respectively, and
    then save new positions to newPosn;
12:     if ‖c_1 F_i‖ > ε_1 or ‖c_2 T_i‖ > ε_2 then
13:       converged ← 0
14:     end if
15:   end for
16: end while
```

In Algorithm 1, c_1 and c_2 (resp., t_1 and t_2) are constants used to control the magnitudes (resp., upper bounds) of movement and rotation of every nonuniform node, *NewPos* and *OldPos* are data structures used to record the new and old positions of vertices of all nonuniform nodes in N respectively, and ε_1 (resp., ε_2) in line 12 is the tolerance of convergence for force (resp., torque) which is usually a very small positive number. In addition, by using the flag *converged*, the while loop in lines 2–16, especially lines 2–4 and 12–14, stops when the drawing remains unchanged between two iterations, i.e., the algorithm converges, or the maximal number of iterations is reached. There exist two for loops in the rest of the main loop, i.e., in each iteration, the first (lines 5–9) is to compute all the forces and torques of each nonuniform node according to the derivations in previous sections, and the second (lines 10–15) is to move and rotate each nonuniform node and then render the graph. To avoid a drastic change in position, we can assign upper bounds to the movement and rotation for each nonuniform node.

The way of rotating a nonuniform node in line 11 is by taking the pivot point as the center of rotation and rotating with an angle with degree $c_2 \times \mathbf{T}_i$ counterclockwise.

About the computation of repulsive forces in line 6(a) of Algorithm 1, we consider only the *local repulsive forces* (derived from the approach in [5]) to speed up the algorithm. Like most of the force-directed approaches to drawing huge graphs [10,11], the idea is to compute the *local* repulsive force of each nonuniform node N_i, i.e., only to compute the repulsive force of N_i due to the other nonuniform nodes placed within the *neighborhood* of N_i

(which is defined as the circle in 2-D (sphere in 3-D) with center N_i and a small radius γ) instead of all the other nonuniform nodes.

As for the time complexity, since line 6(a) in Algorithm 1 needs to enumerate all pairs of nonuniform nodes, one can easily check that each iteration of Algorithm 1 takes time $O(|N|^2 + |E|)$ in the worse case. Line 6(a) in Algorithm 1 may take time $O(|N|)$ if nonuniform nodes are uniformly distributed in the 2-D plane (note that the situation often occurs in the drawings generated by force-directed methods). In this case, each iteration of Algorithm 1 taking time $O(|N| + |E|)$ may be executed efficiently. In order to evaluate the total number of iterations experimentally, we execute our algorithms on a number of GNNs with moderate sizes ($|N| \leq 100$), which are generated randomly, under our setting of parameters. The experimental evaluation shows the total number of iterations to be at most about 1000.

4.2. Implementation

The analysis on the convergence and the adjustments of parameters in force-directed methods has been discussed a lot in previous work (see, e.g., [7,11,13]). On theoretical aspect, Eades and Lin [13] 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 and considers additional torque formulas, but does not have too much modification on the optimized procedure of conventional force-directed algorithms. Therefore, like [13], 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_2, ε_1, ε_2 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 node must be exerted by nonzero spring and repulsive forces. That is, if the total force acted at a node is nonzero, then the node moves either inward into or outward from the other nodes of the graph. Since parameters c_1 and c_2 (resp., t_1 and t_2) control the magnitudes (resp., upper bounds) of movement and rotation, the ranges in which nodes move and rotate are bounded. That is, if those parameters are set smaller, then the movement and rotation ranges of nodes are smaller. Note that parameters ε_1 and ε_2 control the tolerance of convergence, so under larger ε_1 and ε_2 nodes do not move and rotate in smaller movement and rotation ranges. As a result, smaller c_1, c_2, t_1, t_2, as well as larger ε_1, ε_2 narrow the movement and rotation ranges of nodes, so our algorithm can achieve convergence if the movement and rotation ranges are set appropriately.

In fact, given certain settingof 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 (resp., rotation) magnitude parameter c_1 (resp., c_2) makes nodes move farther (resp., be rotated faster) so that the final drawing may be generated faster and hence the total running time may be shorter. Note that t_1 and t_2 control the upper bounds of the movement and rotation, respectively. However, if some nodes move back and forth between two consecutive drawings (resp., rotates very fast during a series of drawings) so that the convergent positions (resp., convergent inclination degrees) of these nodes cannot be determined, then it implies that the value of c_1 (resp., c_2) should be decremented.
2. Bigger force (resp., torque) convergence tolerance ε_1 (resp., ε_2) can make the algorithm achieve the convergence of forces (resp., torque) faster. However, if we observe that a further iteration executed at the final drawing can obtain a drawing with better placements of nodes (resp., better inclination degrees of nodes), then it implies that the value of ε_1 (resp., ε_2) should be decremented.
3. Smaller neighborhood radius γ can make the algorithm run faster because only the local repulsive forces are computed. However, if we observe that the final drawing looks bad, i.e., we encounter a local minimal problem, then it implies that the value of γ should be incremented.

Intuitively, our approach with repulsive forces between nonuniform nodes handles the problem of node–node overlapping,[2] except for the case when the pivot point of a small-size node falls inside a large node. If there is no overlapping in the initial drawing, this pathological case would occur only when the small-size node moves so fast such that it rushes into the big-size node, and hence the problem can be solved by setting c_1 smaller. In future work, we may apply an alternative strategy in which each iteration checks out this case and adds a repulsive force between the two nodes.

Aside from the problem of node–node overlapping, the problem of node–edge overlapping also needs to be taken into account. In [14], it was proven that a drawing is free from edge–node overlapping if the distance between any two nonuniform nodes is at least half of the maximal edge length. If we also consider uniform edge lengths, it is required that the ratio between the maximal edge length and the minimal edge length is less than two. Hence, in practice edges rarely intersect with nodes if the deviation between the edge lengths is not too large. Moreover, a post-processing strategy may be applied to remove node–edge overlapping, e.g, adding repulsive forces

[2] Without considering edges, the approach in [15] produces the drawing of style $\mathbb{S}_3$ without the overlapping between the rectangular nodes.

between nodes and edges [7] or drawing edges as curves [29], bypassing the intersected nodes.

In the future work, we may consider the following. The repulsive forces make our approach computationally expensive, and so we may apply an extension of the conventional spring algorithms (viewing each nonuniform node as a zero-size point) as the preprocessing procedure of our approach may be a good strategy of reducing the total executing time as well as 'jumping' out local minimal problems initially. Especially, if all the nodes of the input graph are rectangles, applying the algorithm in [14] as the preprocessing procedure of our approach is a good starting point. Obviously, we can imagine that [14] is used to produce an initial drawing with a nice embedding, and then our approach only takes a slight amount of additional effort to adjust the initial drawing by means of rotations for achieving a smaller area or a higher degree of symmetry.

Note that conventional force-directed methods cannot guarantee nice drawings for all kinds of graphs because they in general do not necessarily reach minimal configurations, i.e., the repulsive forces among some nodes might be too weak or too strong. Fortunately, some local minimal problems can be overcome by adjusting the coefficients of the models. Our model for drawing GNNs can also handle the local minimal problem in the same way.

4.3. 2-D experimental results

In what follows, we present some experimental results for a variety of graphs in 2-D and give some discussion. Recall that there exist four styles $\mathbb{S}_1$–$\mathbb{S}_4$ for drawing GNNs. In this subsection, we only consider the drawings of style $\mathbb{S}_2$ or $\mathbb{S}_4$ because the other styles can be formed from these by rotation.

In practice, setting different spring magnitudes for the same GNN may produce a variety of nice drawings. Technically, spring magnitude is controlled by scaling constant c_a or spring natural length l in Eq. (1), and we say that spring is *stronger* (resp. *weaker*) if c_a is set larger (resp. smaller) or l is set shorter (resp. longer). In Fig. 5, all the various drawings of style $\mathbb{S}_2$ display symmetrically regardless of the setting of spring magnitudes.

Figs. 6 and 7 display the continuous process from initial drawing to final drawing of styles $\mathbb{S}_2$ and $\mathbb{S}_4$, respectively. Note that Fig. 6 allows multiple edges, and hence is easy to manipulate nonuniform nodes arbitrarily. By observing the continuous processes of many instances, we find that springs can be set very strong such that our approach can output a nice drawing with a compact area as well as avoiding node–node overlapping although there may exist some node–node overlapping in the process. However, setting springs too strongly may result in an unbalanced and divergent state because of lacking a compromise between spring and repulsive forces. On the other hand, setting springs weaker may not only lead to a local minimal problem but also produce a final drawing with larger area. Therefore, the strength of springs might as well be adjusted case by case, and there exist a tradeoff between spring magnitude and drawing area.

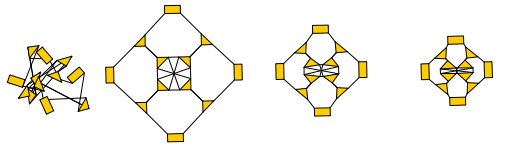

Fig. 5. A variety of drawings of style $\mathbb{S}_2$ for the same GNN with respect to different kinds of springs. (Average running time per iteration: 0.00452 s.)

Note that it is more difficult to draw a GNN with more edges. Consider Fig. 8 as an example. Fig. 8(a) illustrates an initial drawing of a hypercube-structure GNN, where each nonuniform node has degree four. Consider the drawings produced by our approach with weaker and stronger springs in Figs. 8(b) and (c), respectively. Our approach with weaker springs can produce a larger-sized nice drawing, as shown in Fig. 8(b). If a smaller-sized drawing is required, since each nonuniform node occupies a nonzero size, the crossings among edges are easily overlapped by nonuniform nodes, as shown in Fig. 8(c), so that the degrees of those nonuniform nodes may not be easily recognized by users. Note that the case is not serious for graphs with zero-sized nodes, as show in Fig. 8(d). As a result, our approach may be more suitable for GNNs with fewer edges.

Fig. 9 gives some 2-D drawings for various graphs with rectangular nodes, where the graphs of (a)–(d) comes from the analogies of the examples in [14] in which we initially give every nonuniform node a certain angle of inclination, and the experimental statistics is given in Table 1. For comparison, five drawings for each graph are given in Fig. 9. That is, the *initial* drawing displays graphs by randomly assigning the location and the inclination degree of every nonuniform node, while the *classical*, *HK*, *our* drawings with stronger and weaker springs display graphs by, respectively, using the naive extension of [2],[3][14], and Algorithm 1 with respect to the initial drawing. The criteria of displaying smaller drawing area and more symmetries may be conflicting with each other, and hence in Fig. 9 our approaches using stronger springs and weaker springs are applied to, respectively, display smaller drawing area and more symmetries. Note that we only give the classical and HK drawings with stronger springs in Fig. 9 because those with weaker springs occupy larger area and cannot generate any symmetry. It is obvious from Fig. 9 to see that the rotations of nonuniform nodes play an important role in the drawing area and symmetries.

In Table 1, the term 'area' measures the rectangular region bounding the graph drawing, and the area value is calculated by viewing the smallest (square) node of graph as one unit. The degree of uniform edge length is measured by the term 'standard deviation of edge length'

[3] The classical spring algorithm is carried out by first assuming all nodes to be of zero size. Once the drawing using a spring algorithm is done, each of the point node is replaced by its actual structure (a small or a large ball).

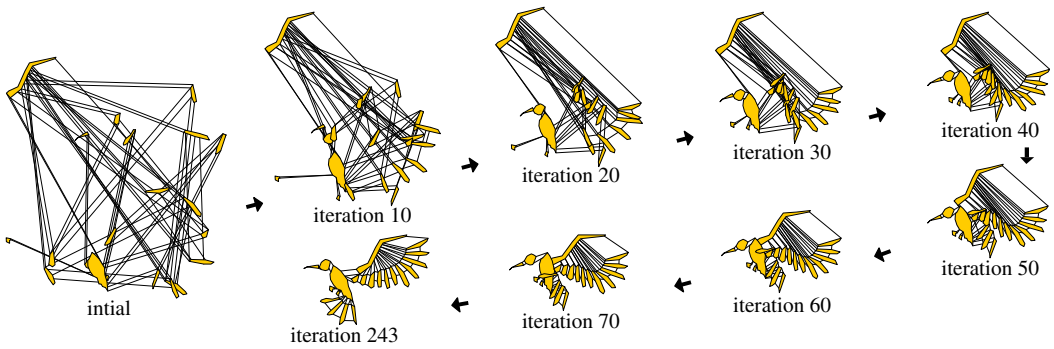

Fig. 6. Continuous process from initial drawing to final drawing of style $\mathbb{S}_2$ with multiple edges for a bird structure. (Average running time per iteration: 0.127572 s.) Note that in this case each spring has different strength for yielding good drawing.

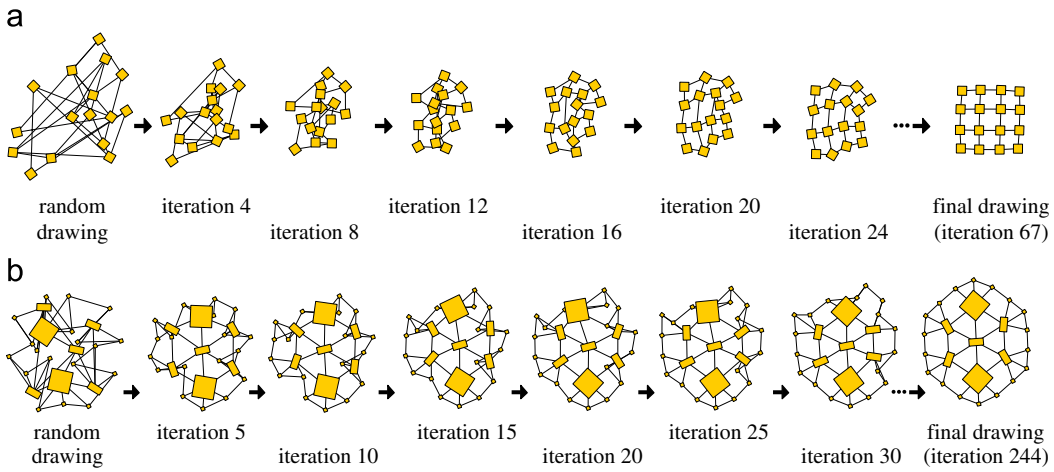

Fig. 7. Continuous process from initial drawing to final drawing. (a) A simple case makes us easy to realize how to follow the dynamic drawing. (b) Bigger nodes walk along a collision-free path.

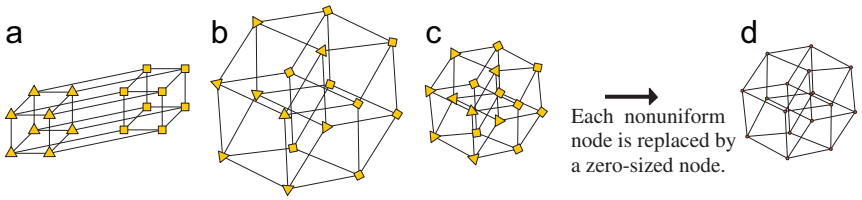

Fig. 8. Different drawings of a hypercube. (a) Initial; (b) our (weaker spring) and (c) our (stronger spring).

[14], and obviously a drawing with more uniform edge length has lower deviation. The symmetries of generic 2-D graph drawings involve *reflectional* and *rotational symmetries*, and the relationship between force-directed methods and symmetries has been formally defined in [13]. Nevertheless, the symmetries of the drawings for GNNs has a little difference. A drawing for GNNs has *reflectional symmetry* if it can be folded in half along a reflection axis and the two halves line up with each other, while the drawing has *k-rotational symmetry* if it can be rotated around degrees $2\pi/k$ and still look the same. Table 1 records the number of reflection axes and k in the columns 'reflection axes' and 'k-rotational symmetries' respectively. That is, the higher the two values are, and the more symmetric the drawing appears. In addition, Figs. 9(b) and (d) reveal that the drawings of some subgraphs in the drawings generated by our approach display a high degree of symmetries.

In what follows, we observe and analyze the entries in Table 1 in greater detail. As for running time, the total running time and total iterations[4] depend on many complicated factors, e.g., movement magnitude parameter c_1, rotation magnitude parameter c_2, force convergence tolerance ε_1, torque convergence tolerance ε_2, nature length of spring, and so on, and hence in Table 1 'average

[4] After trying many examples of moderate-size graphs, we find that our approach can generate nice drawing by taking about two or three times of the number of iterations of HK approach.

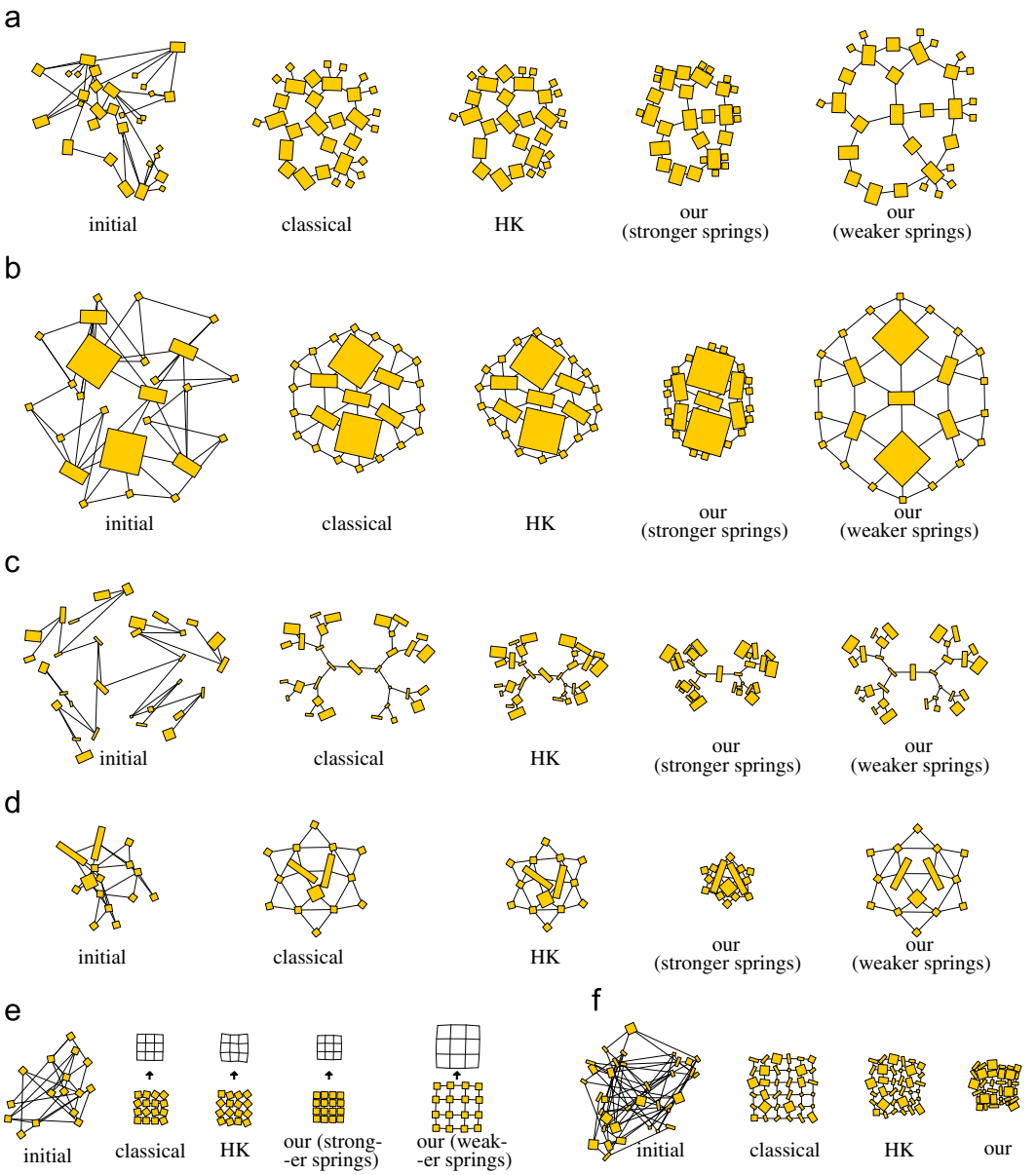

Fig. 9. 2-D drawing of styles $\mathbb{S}_4$ for various graphs with rectangular nodes. (a) HK_A. (b) HK_B. (c) HK_C (The size of each drawing is 40% of the original.). (d) HK_D. (e) Mesh 4×4. (f) Mesh 6×6 (The size of each drawing in (f) is 30% of the original.).

time per iteration' and 'number of iterations' are measured under the setting $c_1 = 0.1$, $c_2 = 0.01$, $t_1 = 10$, $t_2 = \pi/2$, $\varepsilon_1 = 0.1$, $\varepsilon_2 = \pi/1800 = 0.1^\circ$, and $\gamma = 40$ and considering that all the three approaches obtain the drawings with the same total edge length. Note that, in order to speed up the convergence of our algorithm, the force and torque convergence tolerances (i.e., ε_1 and ε_2) are set as large values as possible such that in visualization the final drawing is hardly modified by subsequent iterations. We apply the same idea to setting the force tolerances used by the classical and HK drawings, which are naturally different from ε_1.

As for the standard deviation of the edge length, Table 1 seems to indicate that our approach using weaker springs is the best. However, a closer examination reveals that the measure of the remaining methods can be reduced by adjusting the parameter of stretching the nature lengths of springs, which has a tendency to maintain the uniformity of the edge length. Therefore, it appears that there is no clear winner in producing drawings with the most uniform edge lengths. As for the drawing area, the HK drawing and our drawing using stronger springs (in Fig. 9) perform better, as the numerical values in Table 1 suggest. Obviously, our approach has the capability of producing the drawing with the smallest area because the rotations of nonuniform nodes are considered in our procedure. As for symmetries, both Fig. 9 and Table 1 suggest that our

Table 1
Statistics on the experimental results.

Graph name	$\|N\|$	Method	Standard deviation of edge length $\left(\frac{StdDev}{AvgLen}\right)$		Area	Reflect. axes	k-Rotational symmetries	Ave. time per iter. (s)	Number of iter.[a]	Total run. time (s)
Spring type			Stronger	Weaker	Stronger	Weaker	Weaker			
HK_A	31	Classical	0.3950	–	386.50	–	–	0.00019	984	0.187
		HK	0.4121	–	349.98	–	–	0.00109	1047	1.141
		Our	0.4746	0.3817	305.28	–	–	0.05371	208	11.172
HK_B	25	Classical	0.4152	–	475.99	0	0	0.00011	282	0.031
		HK	0.4989	–	405.94	0	0	0.00077	366	0.281
		Our	0.6366	0.3768	249.31	2	2	0.05558	244	13.562
HK_C	31	Classical	0.2697	–	1012.32	–	–	0.00020	1181	0.234
		HK	0.3454	–	603.20	–	–	0.00112	460	0.515
		Our	0.4813	0.2950	500.31	–	–	0.05263	293	15.422
HK_D	15	Classical	0.2702	–	283.26	0	0	0.00005	87	0.004
		HK	0.3001	–	181.00	0	0	0.00031	562	0.172
		Our	0.4754	0.2764	70.68	1	0	0.01895	357	6.766
Mesh 4×4	16	Classical	0.3170	–	28.94	0	0	0.00007	222	0.016
		HK	0.2341	–	32.15	0	0	0.00030	256	0.078
		Our	0.3584	0.2961	22.66	4	4	0.02379	67	1.594
Mesh6 $\times$ 6	36	Classical	0.3649	–	80.10	–	–	0.00025	557	0.141
		HK	0.4269	–	63.59	–	–	0.00157	1390	2.188
		Our	0.4737	–	44.61	–	–	0.08758	529	46.328

Running time is measured on an Athlon 64 X2 Dual Core 1.99 GHz PC with 1.25 GB memory under setting $c_1 = 0.1, c_2 = 0.01, t_1 = 10, t_2 = \pi/2, \varepsilon_1 = 0.1, \varepsilon_2 = \pi/1800 = 0.1^\circ, \gamma = 40$.

[a] The number of iterations is measured when all the three approaches obtain the drawings with the same total edge length.

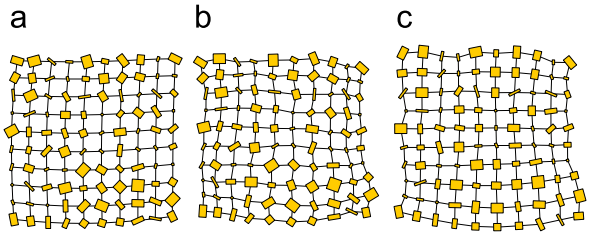

Fig. 10. The drawing of a mesh GNN with 10×10 nodes produced by our approach. (a) Classical. (b) Our. (c) HK.

Table 2
Statistics of the experimental result for the mesh GNN shown in Fig. 10.

Mesh 10×10 ($\|N\| = 100$)			
Method	Average time per iterations (s)	Number of iterations	Total running time (s)
Classical	0.00200	2267	4.531
HK	0.01187	1316	15.625
Algorithm 1[a]	0.20775	753	156.438

[a] Applying the classical approach as the preprocessing procedure of our approach (i.e., applying the output of the classical approach as the input of our approach) costs only about 95.406 s.

approach using weaker springs enjoys the merit of a high degree of symmetries.

In order to estimate the number of nodes that our algorithms can handle, Figs. 10(a)–(c) give the drawings of a mesh GNN with 100 nodes produced by the classical approach, the HK approach, and Algorithm 1, respectively, and their experimental statistics is given in Table 2. In view of Tables 1 and 2, our algorithm can still be considered tractable for handling moderate-sized graphs, although it runs slower than the others, as expected. On average, for the GNNs in Table 1, our algorithm runs 263–505 times slower than the classical approach, whereas from Table 2 our algorithm runs about 34.5 times slower than the classical approach for a 100-node mesh GNN. It implies that, as far as a larger-sized GNN is concerned, the effect of computing only the local repulsive forces in Algorithm 1 on the reduction of time complexity is getting more obvious, and the gap of time complexity between Algorithm 1 and the classical approach is getting narrower. Although Algorithm 1 still run slower than the classical approach by a large multiple, Algorithm 1 takes about 156.438 s ($\approx$ 2.6 min) to handle such a 100-node GNN. This leads us to a conclusion that our algorithm is mostly suitable for either moderate-sized GNNs or the larger-sized GNNs in the static scenario. Note that applying the classical approach as the preprocessing procedure of our approach may reduce the total running time (in the example, this strategy costs only about 95.406 s $\approx$ 1.6 min). It is of interest to see whether the multi-scale technique [10,11] could be applied and

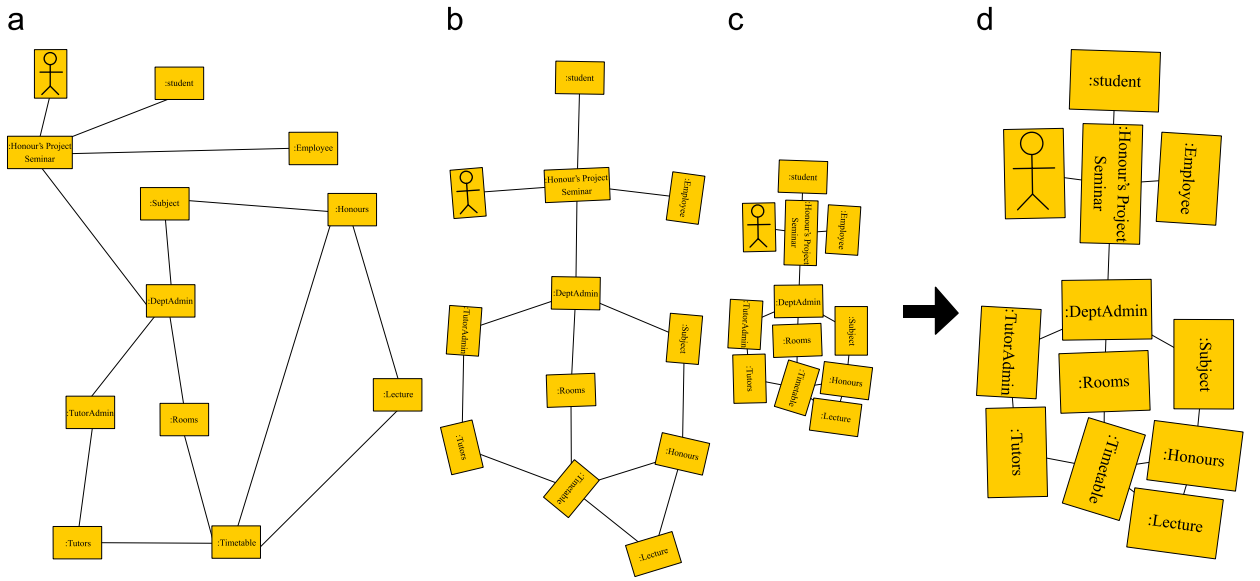

Fig. 11. Different drawings of a UML class diagram, where (a) is derived from [32]. Note that the original drawing in [32] includes texts associated with edges, but the texts are omitted in (a) because the edge labeling is not concerned in our work. (a) original; (b) our (weaker springs); (c) our (stronger springs) and (d) Zooming in (c) until the height is the same as (a).

incorporated into our algorithm to improve the time complexity.

In summary, allowing the rotation of each nonuniform node has the advantage that weaker springs make our final drawing more symmetrical, whereas strong springs make our final drawing with a smaller size. In graph drawing, symmetry is a much admired property [31]. A more symmetrical drawing can help users observe the structure and the properties of the graph clearly in visualization. On the other hand, because the display screen is limited, it is crucial to generate a drawing with a smaller area, especially, in the case when every node has a nonzero size. When wezoom in a smaller-sized drawing with tiny attributes (texts or icons) associated with nonuniform nodes, the size of the attributes are enlarged simultaneously and hence can be displayed clearly.

Consider Fig. 11 for an example. The *unified modeling language (UML)* is a standardized specification language for object modeling, and it is often used to describe the static view of an application [33]: the main constitutions are classes and their relationships. In the graphical notations of UML, classes are represented by nonuniform nodes and their relationships are represented by arcs. For example, Fig. 11(a) gives a drawing of a UML class diagram derived from [32], depicting the procedure followed for organizing honors students' seminars. Fig. 11(b) (resp., Fig. 11(c)) redraws Fig. 11(a) by our approach with weaker (resp., stronger) springs, which gives a nearly symmetrical drawing (resp. a smaller-sized drawing). As compared to the drawing in Fig. 11(a), the drawing in Fig. 11(b) reveals the information that the UML class diagram is a symmetrical graph if we delete the node labeled by ':Lecture' (the bottommost node in Fig. 11(b)). As for the advantage of a small-sized drawing, Fig. 11(d) gives the drawing after zooming in the small-sized drawing in Fig. 11(c) until the drawing height is the same as Fig. 11(a). By doing this, although some text inside nonuniform nodes in Fig. 11(d) is oblique, they still can be observed more clearly than those in Fig. 11(a), under the same display screen. If text labels are too oblique to be observed, we may further restrict every nonuniform node to rotating under a range of inclination angles.

For graphs with not only rectangular nodes, we consider an example shown in Fig. 12. Fig. 12(a) is a diagram derived from [34], which applies UML notations and a special type of high-level Petri-nets [35] to the visual modeling of object-oriented distributed system. Note that Fig. 12(a) is drawn manually. By representing each object by a polygon and each net by straight line segments, we obtain a straight-line drawing of a GNN in Fig. 12(b), where multiple edges are allowed. Fig. 12(c) redraws Fig. 12(b) by our approach. We observe that there are three clusters of nonuniform nodes which are divided by two dashed curves in Figs. 12(b) and (c), respectively. Note that the way to divide the GNN into three clusters in Fig. 12(c) is different from that in Fig. 12(b). It is easy to see that the clusters in Fig. 12(c) are more reasonable (e.g., according to the connectivity of node 'processtask', the node should belong to the largest cluster, but Fig. 12(b) is not the case). Therefore, like other force-directed methods, our approach has the advantage that the nodes with highly connectivity are drawn closely.

4.4. 3-D experimental results

Now we turn our attention to drawing GNNs in 3-D. Figs. 13 shows some experimental results of our algorithm on *pyramid*, *cube*, *mesh*, *flower*, and *sphere*. As one can easily see, each of the final drawings displays a high

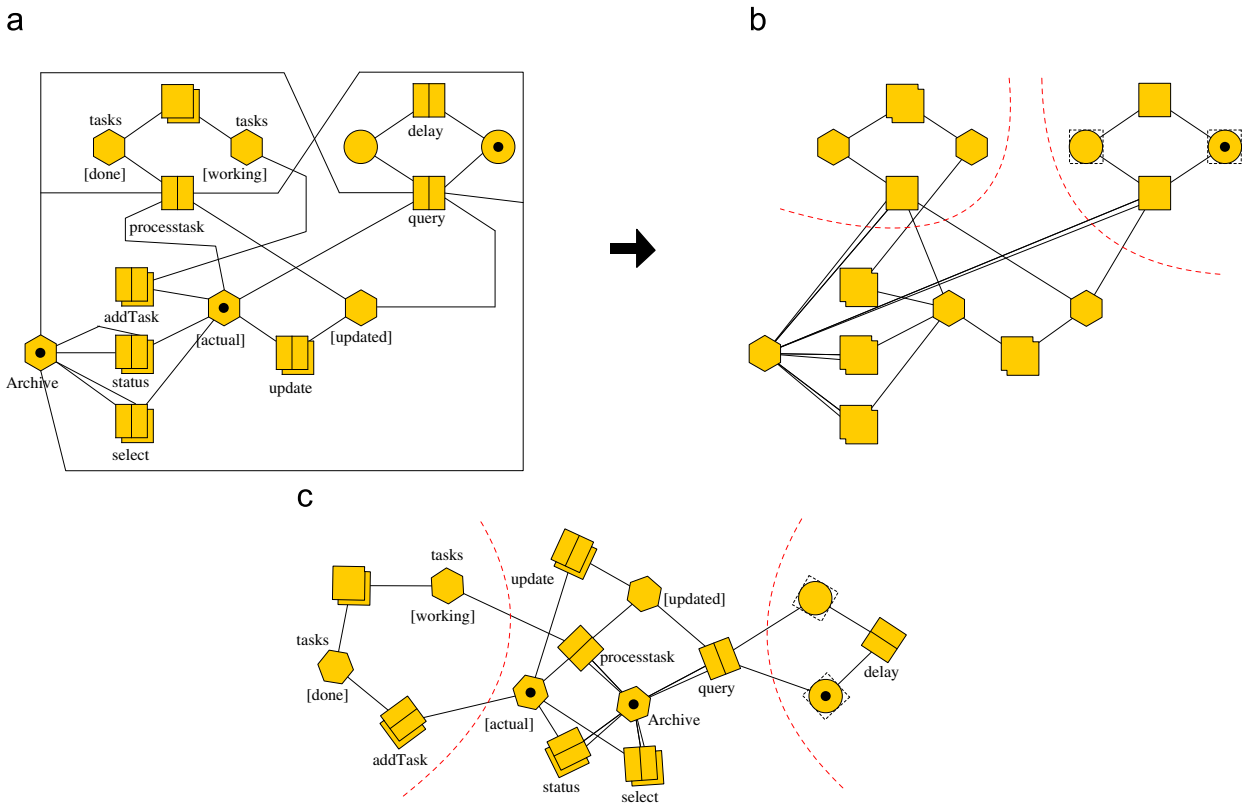

Fig. 12. (a) A diagram derived from [34]. (b) A straight-line drawing of (a). (c) A drawing of the diagram produced by our approach. Note that two dashed lines divide the diagram into three clusters in (b) and (c), respectively.

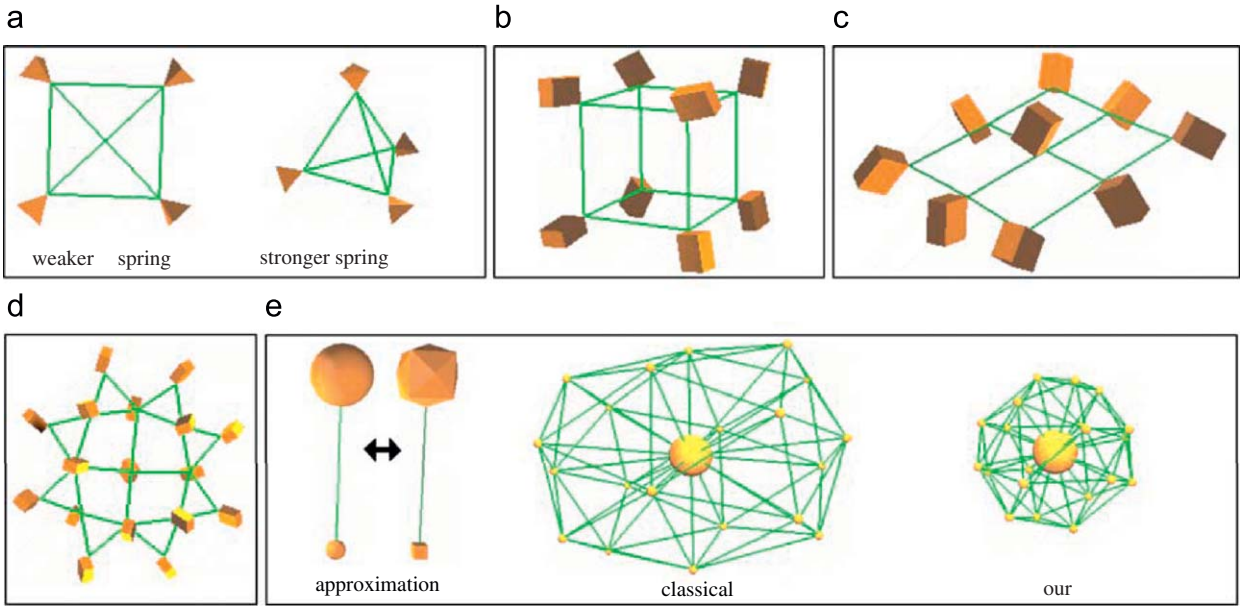

Fig. 13. 3-D graph drawing. (a) For the same structure, using different springs show two different layouts. (b) Cube structure. (c) Mesh structure. (d) Flower structure. (e) Sphere structure (our approach uses polyhedra to approximate spherical nodes). (Respectively, average running time per iteration (s): 0.00015, 0.00453, 0.00593, 0.02203, 0.02375).

degree of symmetry and is reasonably 'nice'. In Fig. 13(e), we compare our approach with a naive extension of the classical spring algorithm [2] using an example of a 3-D sphere structure. In this graph, a large ball is surrounded by a number of small balls. Our potential fields approach produces a nicer drawing as Fig. 13(e) indicates. This is

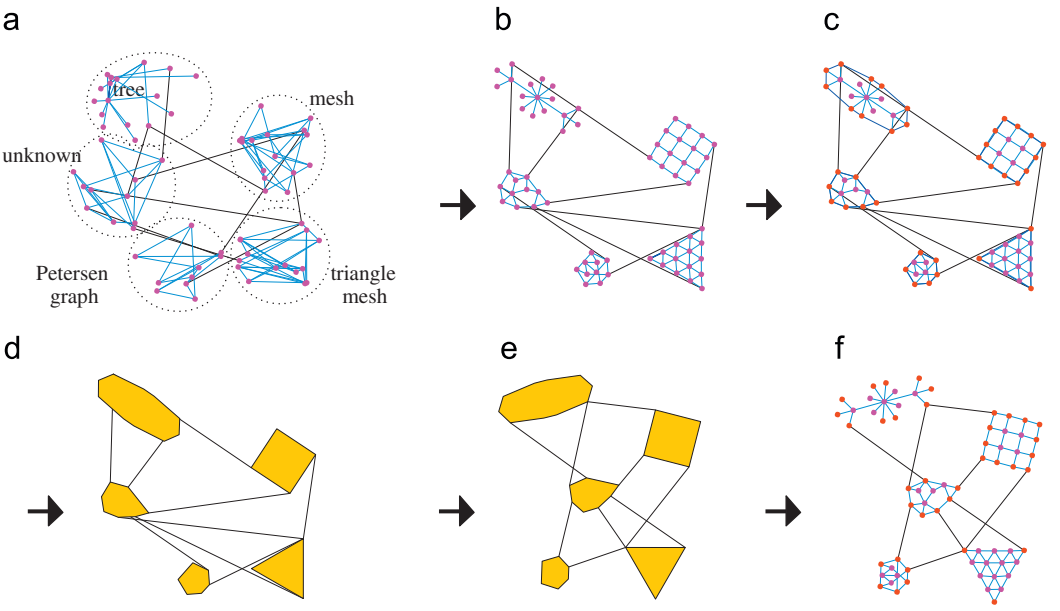

Fig. 14. Drawing a clustered graph. (a) Cluster partition. (b) Drawing individual clusters using the conventional force-directed method. (c) Convex hulls of clusters. (d) Drawing while treating clusters as nonuniform nodes. (e) Output of our algorithm (taking (d) as the input). (f) Final drawing by plugging in details of clusters.

because our approach can detect the large ball and consider it to draw the GNN.

It should be noticed that the nonuniform nodes (polyhedra) in each graph of Fig. 13 do not display symmetrically. The reason is that our 3-D approach uses sample points to approximate each polyhedron for reducing the running time, and hence it may easily result in local minimal problems between polyhedra.

In the future, it is of interest to further compare our 3D approach with other 3D methods, e.g., GEOMI [36] or WilmaScope [37].

5. Applications

Aside from the fact that natural objects tend to be of nonzero size, another motivation behind the need of handling GNNs has to do with drawing clustered graphs. Consider the scenario of drawing a graph with millions of nodes. Applying a drawing algorithm to the huge graph directly suffers from a number of drawbacks. Disregarding the inefficiency of the method, the outcome of the drawing often fails to reflect the structure of the graph, for related nodes may not be close to each other as one normally prefers. As a result, a number of strategies have been developed targeting at keeping related nodes in a cluster. For instance, the EH method [25] suggests the use of three types of springs: 'weaker' springs for connecting inter-cluster nodes, 'stronger' springs for intra-cluster ones, and 'virtual springs' for gathering adjacent and nonadjacent intra-cluster nodes. Using this idea, it becomes easier to 'navigate' through clustered graphs with huge numbers of nodes. However, with nodes added into (or deleted form) the graph, the EH algorithm has to be applied to all the nodes in the new graph, even to clusters that already exist in the original graph. This leaves us to wonder, instead of re-drawing an existing cluster, whether it is possible to inherit the drawing of the cluster from its original graph. By treating each cluster as a nonuniform node, one would naturally expect that drawing graphs with nonuniform nodes might find another application in the study of clustered graphs.

Consider Fig. 14 for example. Instead of applying a drawing algorithm (such as a force-directed method) to the entire graph, it might be beneficial to take advantage of the nature of the graph being clustered. By using some technique, we partition the graph into some clusters, e.g., the graph of Fig. 14(a) is divided into clusters of tree, mesh, triangle mesh, and Petersen graph, and the remaining nodes form a cluster called 'unknown'. Knowing the constituent clusters, a good starting point to draw such a clustered graph, perhaps, is to draw each individual cluster separately by means of, for instance, the classical spring algorithm (Fig. 14(b)). Once this is done, the details of each cluster can be abstracted out by regarding its convex hull as a *nonuniform* node. See Figs. 14(c) and (d). The next step is to apply our approach to producing a layout of the GNN such as Fig. 14(e) shows. Finally, a nice drawing of the original clustered graph is obtained by restoring the details of all the clusters as Fig. 14(f) illustrates. Now suppose new nodes are added to or deleted from Fig. 14(a). Instead of running the drawing algorithm on the new (possibly huge) graph all over again, our approach allows us to keep the internal drawings of those unaffected (due to insertion/deletion of nodes) clusters intact, while the redrawing need only be applied to a much smaller graph, giving rise to a much better performance.

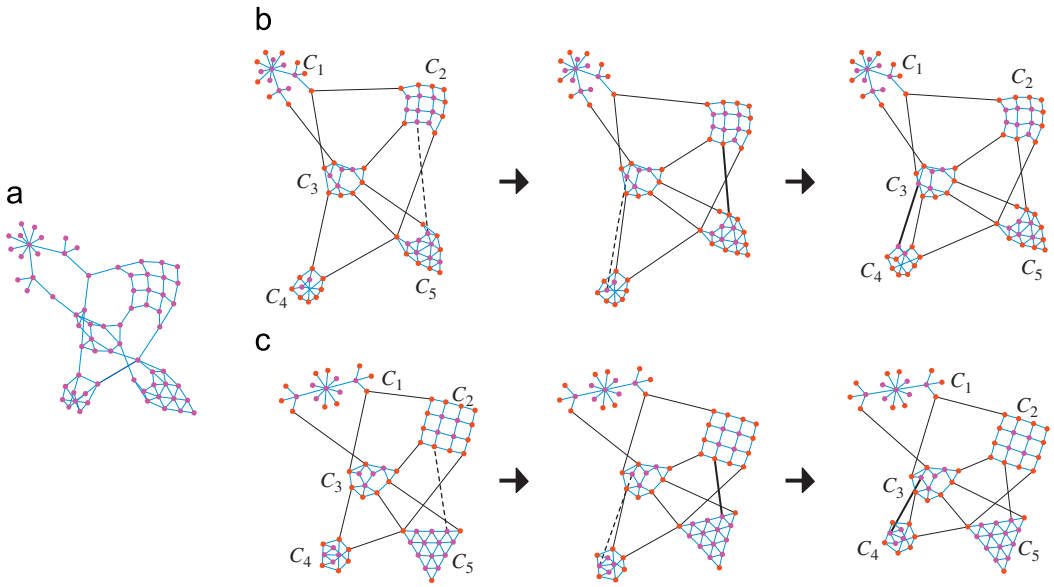

Fig. 15. Comparison of drawing the clustered graph in Fig. 14 (a) classical (without clustering); (b) EH (c) our.

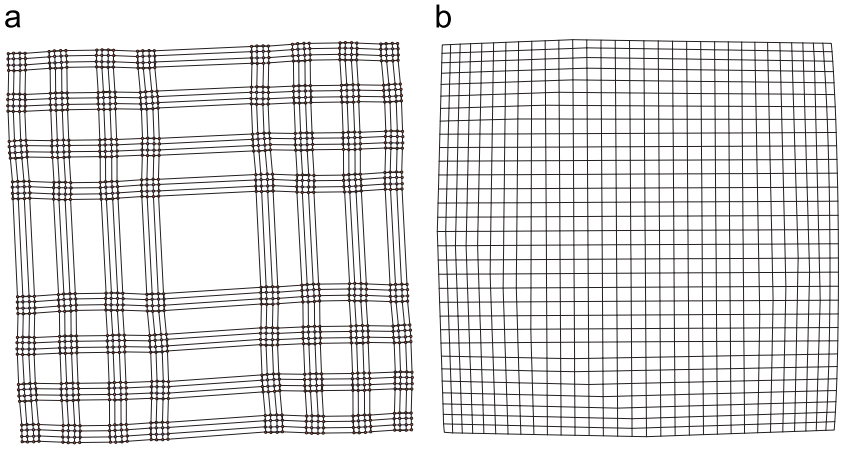

Fig. 16. Drawing a 1024-node mesh graph by (a) using our approach which considers the graph as a hierarchical clustered graph with four-level inclusion tree (total running time: 73.297 s); (b) using classical spring embedder [2] (total running time: 1477.250 s).

With respect to hierarchical clustered graphs, the superiority of our approach is twofold. First, by taking advantage of the cluster structure, our method requires a much lesser amount of computation than the EH method as pointed out in the above discussion. Furthermore, the drawing quality of our approach is also better than the EH method with respect to the example shown in Fig. 15. To see this, the drawings of clusters C_2 and C_5 are somewhat distorted by using the EH method; on the other hand, these two clusters are nicely displayed by ours. (See the left-most drawings of Figs. 15(b) and (c).) Now if we add two edges to the graph (see the middle drawings of Figs. 15(b) and (c)), and re-draw the graph using the two methods, we end up with the right-most drawings of Figs. 15(b) and (c). Comparing the left-most drawing with the right-most drawing in Fig. 15(b), the embedding (see C_3 and C_4 clusters) and the contour (see C_2 cluster) have changed. Our approach, on the other hand, does not have this problem as Fig. 15(c) shows.

Note that our approach, however, may have a disadvantage as follows. The inter-cluster edge incident to a node that is not drawn on the boundary of a cluster (polygon) can induce edge crossings (see the thick edge in the right-most of Fig. 15(c)). This is because the drawings of clusters which are generated in earlier stages and not changed afterwards are not aware of whether the nodes drawn inside the cluster drawings are connected by inter-cluster edges in later stages.

Our approach to drawing hierarchical clustered graphs would also work well in the visualization of social networks [38]. Their comparison is remained as the future work.

Although the graph used in Figs. 14 and 15 is a hierarchical clustered graph with a 3-level inclusion tree,

our approach can be easily applied to more complicated hierarchical clustered graphs. Fig. 16(a) gives an example of the drawing of a 4-level hierarchical clustered graph. In comparison with Fig. 16(b) which is generated byusing the classical embedder [2] (regardless of its cluster structure), our approach performs about 13 times more efficient than the classical approach. Fig. 16(a) in fact displays the cluster structure of the hierarchical clustered graph. What makes our approach run faster than the classical one is that our drawing method need not re-draw the clusters at the lower levels when processing the clusters at the higher levels. As the size of the graph increases, the classical approach is likely to run even slower and often suffers from a more serious local minimal problem. In view of the above, our approach also has similar merits as so-called multi-scale approach [10,11], which runs efficiently because in the multi-scale approach each node only considers the local forces instead of the global forces. The main difference between our approach and the multi-scale approach is that the main purpose of the multi-scale approach is to draw generic graphs instead of hierarchical clustered graphs, and hence, the multi-scale approach cannot display the cluster structure of a clustered graph.

6. Conclusion

A potential-based approach, coupled with a force-directed method, has been proposed and implemented for drawing graphs with nodes of different sizes and shapes. Some applications, including drawing clustered graphs, have also been given to demonstrate the usefulness of our approach. A unique feature of our approach is that in the process of reaching equilibrium, the degree of inclination of a nonuniform node can be adjusted while moving from one position to another. By doing so, the final drawing has a tendency to display a high degree of symmetry as well as to fit in a compact area. An equally important aspect of our approach is that the formulas derived in our potential field model are analytically tractable, making our algorithm computationally efficient. In comparison with some existing algorithms, our experimental results look promising. A line of future improvement includes the reduction of time complexity and the investigation of a more general model for potential fields in 3-D, for our current treatment of a 3-D node is based on approximating its faces using sample points. It would be of interest to find solutions to overcome the local minimal problems for drawing graphs with nonuniform nodes, to apply the multi-scale technique to handle graphs with huge nonuniform nodes, and to produce dynamic drawings of graphs while preserving the mental map. In addition, it would also be of interest to find a wider variety of instances for our approach such that users may decide whether our approach suits their practical needs.

Acknowledgments

The authors thank the anonymous referees for comments that improved the content as well as the presentation of this paper.

References

[1] K. Kaufmann, D. Wagner (Eds.), Drawing graphs: methods and models, in: Lecture Notes in Computer Science, vol. 2025, Springer, Berlin, 2001.
[2] P. Eades, A heuristic for graph drawing, Congress Numerantium 42 (1984) 149–160.
[3] F. Bertault, A force-directed algorithm that preserves edge crossing properties, in: Graph Drawing 1999, Lecture Notes in Computer Science, vol. 1731, Springer, Berlin, 1999, pp. 351–358.
[4] C.-C. Lin, H.-C. Yen, A new force-directed graph drawing method based on edge–edge repulsion, in: IV 2005, IEEE CS Press, 2005, pp. 329–334.
[5] T. Fruchterman, E. Reingold, Graph drawing by force-directed placement, Software-Practice and Experience 21 (1991) 1129–1164.
[6] A. Frick, A. Ludwig, H. Mehldau, A fast adaptive layout algorithm for undirected graphs, in: Graph Drawing '94, Lecture Notes in Computer Science, vol. 894, Springer, Berlin, 1995, pp. 388–403.
[7] R. Davidson, D. Harel, Drawing graphs nicely using simulated annealing, ACM Transactions on Graphics 15 (1996) 301–331.
[8] T. Kamada, S. Kawai, An algorithm for drawing general undirected graphs, Information Processing Letters 31 (1989) 7–15.
[9] K. Sugiyama, K. Misue, Graph drawing by the magnetic spring model, Journal of Visual Languages and Computing 6 (1995) 217–231.
[10] S. Hachul, M. Jünger, Drawing large graphs with a potential-field-based multilevel algorithm, in: Graph Drawing 2004, Lecture Notes in Computer Science, vol. 3383, Springer, Berlin, 2004, pp. 285–295.
[11] D. Harel, Y. Koren, A fast multi-scale method for drawing large graphs, Journal of Graph Algorithms and Applications 6 (3) (2002) 179–202.
[12] F.J. Brandenburg, M. Himsolt, C. Rohrer, An experimental comparison of force-directed and randomized graph drawing algorithms, in: Graph Drawing '95, Lecture Notes in Computer Science, vol. 1027, Springer, Berlin, 1995, pp. 76–87.
[13] P. Eades, X. Lin, Spring algorithms and symmetry, Theoretical Computer Science 240 (2) (2000) 379–405.
[14] D. Harel, Y. Koren, Drawing graphs with nonuniform vertices, in: AVI 2002, ACM Press, New York, 2002, pp. 157–166.
[15] X. Huang, W. Lai, Force-transfer: a new approach to removing overlapping nodes in graph layout, in: 25th Australasian Computer Science Conference, CRPIT, vol. 16, Australian Computer Society, 2003, pp. 349–358.
[16] H. Eichelberger, SugiBib, in: Graph Drawing 2001, Lecture Notes in Computer Science, vol. 2265, Springer, Berlin, 2002, pp. 467–468.
[17] C. Gutwenger, M. Jünger, K. Klein, J. Kupke, S. Leipert, P. Mutzel, A new approach for visualizing UML class diagrams, in: SOFTVIS 2003, ACM Press, New York, 2003, pp. 179–188.
[18] J.-H. Chuang, Potential-based modeling of three-dimensional workspace for obstacle avoidance, IEEE Transactions on Robotics and Automation 14 (5) (1998) 778–785.
[19] 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—Part B: Cybernetics 28 (5) (1998) 729–736.
[20] F.B.M. Miller, An algorithm for drawing compound graphs, in: Graph Drawing '99, Lecture Notes in Computer Science, vol. 1731, Springer, Berlin, 1999, pp. 197–204.
[21] U. Dogrusöz, E. Giral, A. Cetintas, A. Civril, E. Demir, A compound graph layout algorithm for biological pathways, in: Graph Drawing 2004, Lecture Notes in Computer Science, vol. 3383, Springer, Berlin, 2004, pp. 442–447.
[22] T.C. Biedl, M. Kaufmann, Area-efficient static and incremental graph drawings, in: ESA'97, Lecture Notes in Computer Science, vol. 1284, Springer, Berlin, 1997, pp. 37–52.
[23] Q. Feng, R. Cohen, P. Eades, Planarity for clustered graphs, in: ESA'95, Lecture Notes in Computer Science, vol. 979, Springer, Berlin, 1995, pp. 213–226.
[24] Q. Feng, Algorithms for drawing clustered graphs, Ph.D. Thesis, University of Newcastle, Australia, 1997.
[25] P. Eades, M.L. Huang, Navigating clustered graphs using force-directed methods, Journal of Graph Algorithms and Applications 4 (3) (2000) 157–181.
[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] S. Kirkpatrick, C.D. Gelatt, M.P. Vecchi, Optimization by simulated annealing, Science 220 (1983) 671–680.
[29] E. Gansner, S. North, Improved force-directed layouts, in: Graph Drawing '98, Lecture Notes in Computer Science, vol. 1547, Springer, Berlin, 1998, pp. 364–373.
[30] T. Dwyer, K. Marriott, P.J. Stuckey, Fast node overlap removal, in: Graph Drawing 2005, Lecture Notes in Computer Science, vol. 3843, Springer, Berlin, 2005, pp. 153–164.
[31] H. Purchase, Which aesthetics has the greatest effect on human understanding, in: Graph Drawing '97, Lecture Notes in -Computer Science, vol. 1353, Springer, Berlin, 1997, pp. 248–261.
[32] H. Purchase, J.-A. Allder, D. Carrington, Graph layout aesthetics in UML diagrams: user preferences, Journal of Graph Algorithms and Applications 6 (3) (2002) 255–279.
[33] J. Rumbaugh, J. Jacobson, G. Booch, The Unified Modeling Language Reference Manual, Addison-Wesley, Essex, UK, 1998.
[34] H. Giese, G. Wirtz, Visual modeling of object-oriented distributed systems, Journal of Visual Languages and Computing 12 (2001) 183–202.
[35] W. Brauer, W. Reisig, G. Rozenberg, Petri nets: central models (part I)/applications (part II), in: Lecture Notes in Computer Science, vols. 254/255, Springer, Berlin, 1987.
[36] A. Ahmed, et al., GEOME: GEOmetry for maximum insight, in: Graph Drawing 2005, Lecture Notes in Computer Science, vol. 3843, Springer, Berlin, 2005, pp. 468–479.
[37] T. Dwyer, Extending the WilmaScope 3D graph visualization system—software demonstration, in: Proceedings of Asia Pacific Symposium on Information Visualisation 2005 (APVIS2005), CRPIT, vol. 45, 2005, pp. 35–42.
[38] N. Henry, J.-D. Fekete, M.J. McGuffin, NodeTrix: a hybrid visualization of social networks, IEEE Transactions on Visualization and Computer Graphics 13 (6) (2007).