Big Data Research 10 (2017) 21–32

Contents lists available at ScienceDirect

Big Data Research

www.elsevier.com/locate/bdr

Towards Visualizing Big Data with Large-Scale Edge Constraint Graph Drawing

Ariyawat Chonbodeechalermroong, Rattikorn Hewett *

Department of Computer Science Texas Tech University, Lubbock, TX, USA

ARTICLE INFO

Article history:
Received 22 August 2016
Received in revised form 1 August 2017
Accepted 7 October 2017
Available online 23 October 2017

Keywords:
Large graphs
Force-directed
Constraint enforcement methods

ABSTRACT

Visualization plays an important role in enabling understanding of big data. Graphs are crucial tools for visual analytics of big data networks such as social, biological, traffic and security networks. Graph drawing has been intensively researched to enhance aesthetic features (i.e., layouts, symmetry, cross-free edges). Early physic-inspired techniques have focused on synthetic abstract graphs whose weights/distances of the edges are often ignored or assumed equal. Although recent approaches have been extended to sophisticated realistic networks, most are not designed to address very large-scale weighted graphs, which are important for visual analyses. The difficulty lies in the fact that the drawing process, governed by these physical properties, oscillates in large graphs and conflicts with specified distances leading to poor visual results. Our research attempts to alleviate these obstacles. This paper presents a simple graph visualization technique that aims to efficiently draw aesthetically pleasing large-scale straight-line weighted edge graphs. Our approach uses relevant physic-inspired techniques to promote aesthetic graphs and proposes a *weak constraint-based approach* to handle large-scale computing and competing goals to satisfy both weight requirements and aesthetic properties. The paper describes the approach along with experiments on both synthetic and real large-scale weighted graphs including that of over 10,000 nodes and comparisons with state-of-the-art approaches. The results obtained show enhanced and promising outcomes toward a general-purpose graph drawing technique for both big synthetic and real network data analytics.

1. Introduction

Visual analytics enable big data understanding and extraction of useful insights. Many real world data, such as paper citations, social, biological and traffic networks, predominantly use graphs as basic structures for visualization. Graph drawing is essential and has been studied intensively [1–10]. Traditionally, graph drawing algorithms have focused on aesthetic properties of synthetic and abstract graphs, where the aesthetically pleasing criteria are based on the graph symmetry and layout to maximize readability and the number of cross-free edges [9].

Early physic-inspired graph drawing techniques are force-directed algorithms [1,7] that model straight-line undirected edge graphs after a spring or electrical system where edges mimic springs. The spring layout method relies on spring forces, behaving similar to those in Hooke's law, that attract or repulse adjacent nodes till they are stable. Unconnected nodes repel each other as electric charged particles do and this nature can evolve to reduce narrow layout and edge crossing. The force-directed method iteratively minimizes the system energy by moving the nodes along the direction of the force. The minimal energy state corresponds to a well-configured layout. Unfortunately, these traditional techniques only perform well for small graphs of no more than a few hundred nodes [9]. One of the major issues for scaling these algorithms is due to many local minima (thus, oscillation and non-converged drawing) in large graphs that lead to unreadable drawings.

To cope with the scalability issue, the force-directed approaches have been extended to many multi-scale (or multi-level) layout approaches [4–7,10] that draw a graph from coarse-scale simple structures to fine-scale complex structures. The techniques involve graph coarsening [5–7,9] including vertex filtrations [4]. The multi-scale approach produces large graphs of over 10,000 nodes in a reasonable time [9]. One of the state-of-the-art well-performing multi-level algorithms is the algorithm by Hu [7]. Hu's algorithm combines the multi-level approach with a method similar to a well-studied n-body problem in physics to reduce repulsive force calculations [7,9]. The algorithm is widely used and has been im-

* Corresponding author.
E-mail addresses: ariyawat.chonbodeechalermroong@ttu.edu (A. Chonbodeechalermroong), Rattikorn.hewett@ttu.edu (R. Hewett).

https://doi.org/10.1016/j.bdr.2017.10.001
2214-5796/

plemented on graph drawing engines such as GraphViz [11] and Gephi [12].

Typical force-directed graph drawing performs well on synthesized abstract graphs due to their symmetric nature, however, this is not always the case for graphs representing real data. Moreover, most algorithms either do not deal with the weights/distances of the graph edges or assume them to be equal. However, in practice, weights/distances representing strengths of the relationships between nodes can be crucial for extracting useful insights. For example, edge length in communication network graphs can have an important implication on cost and resource requirements. Recent force-directed algorithms designed to address these issues include ForceAtlas2 [13] that handles weighted edges for real relational networks such as web graphs and social networks. ForceAtlas2 applies a power-law distribution of degree to a modified computation of repulsive forces to rely on the degree of the nodes. Using a strong electrical force, ForceAtlas2 allows visualization of important clusters. ForceAtlas2 has been widely used and implemented in an interactive visualization tool, Gephi. Although recent approaches have been extended to sophisticated real networks, most are not designed to address very large-scale weighted graphs, which are crucial for visual analyses. The difficulty lies in the fact that the drawing process governed by these physical properties oscillates in large graphs or conflicts with specified distances leading to poor visual results. Our research attempts to alleviate these obstacles. This paper presents a simple graph visualization technique that aims to efficiently draw aesthetically pleasing large-scale straight-line weighted edge graphs. The proposed approach is based on a constraint-based approach to cloth simulation [14], commonly used in computer graphic and game development to provide efficient realistic simulation of particles with given distance constraints [14,15]. We introduce a *weak constraint-based approach* that relaxes rigid distance preference requirements (within acceptable bounds) in order to improve graph readability. The weak constraints allow flexibility in trading the distance precision with the layout quality. Using the weak constraint enforcement with appropriate physic-inspired techniques, our approach yields graphs that balance the weight precision with aesthetic quality. Note that the distance is typically assumed to be inversely proportional to the weight. This is because an edge with a high weight represents a spring with a strong attracting force that pulls the edge distance short.

The rest of the paper is organized as follows. Section 2 describes related work followed by our proposed algorithm with some illustrations in Section 3. Section 4 presents experimental results to compare our approach with two state-of-the-art approaches, namely Hu's multi-level algorithm and ForceAtlas2. Section 5 gives a summary and concludes the paper.

2. Related work

Force-directed methods are among the most well-studied and flexible techniques for drawing graphs with straight-line edges [9]. Classic physic-inspired algorithms mimic spring and electrical systems [1,3,9]. The basic idea is to replace edges with springs, start with a random initial graph layout and then let go the system so that the spring forces, governing by Hookes Law rules, will reconfigure the graph. The force-directed approach maps the graph layout to the energy of the physical system in such a way that low energy corresponds to an aesthetic layout (e.g., equal distance edge, symmetry and well-spaced for non-adjacent nodes) [9]. The goal is to find the graph layout whose energy is minimized. Ideally, the graph drawing is evolving until it converges to a stable state of minimum energy.

Eades' algorithm [1] makes two modifications. One is to use the logarithm strength forces for far away nodes because Hookes Law (linear) springs are too strong. The other is to make non-adjacent nodes repel each other where the repulsive force is inversely proportional to the square root of the edge distance. Fruchterman and Reingold [3] introduced their algorithm using attractive spring forces between adjacent nodes but an electrical repulsive force between all pairs of nodes. The algorithm re-defines new attractive and repulsive forces in such a way that a *high* graph density gives a *low* attractive force but a very *high* repulsive force resulting in "even vertex distribution" so that the layout is well-spaced. The algorithm also adds a notion of "temperature" to control the displacement of the nodes in the same fashion as that of simulated annealing [16] in order to avoid local minima issue in finding the optimal graph layout. This local minima issue can cause oscillation and prevents the algorithm from converging. The problem is more severe when the graphs are large and complex. In fact, this is the main issue with the force-directed algorithms. They perform well for small graphs of a few hundred nodes [9].

Recent ForceAtlas2 [13] is a widely used force-directed algorithm that is designed to efficiently draw graphs with weighted edges on real networks. ForceAtlas2 takes the degree of nodes into account while repulsive forces are being computed. This helps improve node distribution and space layouts. It respects physical laws and provides a mechanism to balance tradeoffs between speed and precision. Since the high speed of node movement leads to high oscillation and inaccurate layouts, ForceAtlas2 estimates appropriate speeds to balance with acceptable accuracies. Although our approach uses physical properties (e.g., Newton's law of movement), the algorithm is driven by constraints (on distances) as opposed to forces. However, it is similar to Fruchterman and Reingold's algorithm in that both approaches use the notion of temperature to alleviate the local minima issue. Our approach is similar to ForceAtlas2 in aiming to balance speeds to reduce oscillation. While ForceAtlas2 approximates appropriate speeds by adapting between local and global speeds, our approach employs Verlet Integration [17] to integrate Newton's law of motion into estimating the trajectories of the node movement.

A *graph theoretic distance approach* introduced by Kamada and Kawai [8] defines a good layout to be when the geometric (Euclidean) distances between nodes are the same as their corresponding graph (theoretic) distances that are computed from the "All-Pairs-Shortest-Path". The approach sets up the spring system whose goal is to minimize the energy, which is equivalent to minimizing the difference between the geometric and graph distances. The approach has been used in several graph drawing approaches. The major issue of this approach is its computational cost. The "All-Pairs-Shortest-Path" computation is expensive requiring $O(|V|^3)$ time using the Floud–Warshall algorithm [10]. The approach improves graph drawing of large graphs but, it still does not scale well for graphs of thousand nodes.

A large number of force-directed multi-level algorithms [4–7, 10] have been studied to cope with scalability issue. The idea is to draw a graph from coarse-scale simple structures to fine-scale complex structures. The technique is referred to as *graph coarsening*. Nodes in a fine-scale level are relocated to capture local organized groups and then are relocated in a coarse-scale level that abstracts essential but simpler layouts. The fine-scale relocations are then performed to verify and correct, local disorders. In [5], the coarse-grain relocation applies energy function of [8] while the fine-grain relocation uses traditional force-directed formulae of [1,3]. Harel and Koren [6] introduced an algorithm using a k-center approximation in the coarsening process to find a subset of a set of nodes of size k where the maximum distance between nodes of the two sets is minimized. Gajer's approach simplifies graph coarsening by using *vertex filtration* [4]. By restricting the number of nodes considered in relocating any node in the filtration of $O(\log|V|)$ levels, the approach applies formulae in [8] to reduce

the overall running time of $O(|V|^3)$ to $O(|V|\log^2|V|)$. Combining the multi-level approach with a well-studied method similar to that of an n-body problem in physics to treat long range nodes as a super-node using octree data structures, Hu's algorithm [7] reduces repulsive force calculations from $O(|V|^2)$ to $O(|V|\log|V|)$ [9]. The algorithm has been implemented on graph drawing engines such as GraphViz [11] and Gephi [12]. The multi-scale approach has improved graph drawing scalability from hundreds of nodes to large graphs of over thousands of nodes in a reasonable time [9]. Our approach is not multi-level but relies on a modified edge constraint-based model after the cloth simulation [14] to improve efficiency. Our approach and the cloth simulation are similar in that, both apply a physical law of motion and simulate the graph layouts based on the distance (weight) constraints. However, our approach and the cloth simulation model are designed for different purposes. The cloth simulation model deals with constant edge distances and includes other physical properties, such as wind and gravity that are not relevant to our purpose.

3. Our approach

Our approach aims to efficiently draw aesthetically pleasing large-scale straight-line weighted edge graphs. The approach is based on a constraint-based technique used in cloth simulation [14], in computer graphics and physic engines, and in game development, where we can simulate cloths of over ten thousand nodes in real-time. In cloth simulation, a cloth model contains physical particles connecting as a three-dimensional cloth network represented by a graph $G(V, E)$ of a given edge length. An advantage of the constraint-based technique is that it takes $\Theta(|E|)$ time to enforce the edge distance constraints (which typically are of similar length). However, in realistic physical systems, *not* all of the specified distance constraints can be realized (e.g., a square with dimensions of the same length as its diagonals). Our approach aims to alleviate this issue.

The basic idea is to use physic-inspired properties (e.g., electrical repulsive forces, Newton's movement laws, and temperature of metal annealing process) along with a weak constraint-based approach to evolve the large graph layout to an aesthetically pleasing one that "sufficiently" satisfies the edge constraints. The sufficient constraint satisfaction is supposed to provide flexibility to relax the precisions of the edge constraints without significant negative impacts on the quality of the layout. The approach provides a framework that explicitly interplays between the flexibility of the precisions of the edge constraint satisfaction, the layout quality and the execution efficiency.

The proposed approach differs from most existing force-directed approaches because of the two key elements: (1) a *weak edge constraint-based* approach and (2) an integration of physics of motion [17] using a *modified Verlet Integration*. The former is to satisfy the edge weight requirements in big graph drawing by avoiding a common issue of local minima in force-directed approaches when applied to large-scale graphs. The latter integrates the Newtonian movement of the graph nodes during the graph drawing process that allows computation in larger time steps while maintains sufficient numerical stability and precision. We will describe these specific features below. To follow the sequence of our algorithm, we will start with the physic-inspired features followed by the proposed weak constraint-based approach and an overall algorithm with illustrated examples on small graphs.

3.1. Force & movement model

3.1.1. Integration of Newton's law of motion

Graph layouts evolve through movement of nodes and edges until it reaches a layout that satisfies the drawing criteria. Newton's laws govern anything that moves in the world. They are crucial for the study of dynamics of many systems including particle-based physical systems. In practice, many numerical methods for integrating Newton's equations of motions are used for estimating trajectories of particles in simulation models, computer graphics and physics games [14,15,17–19]. Our graph drawing approach employs the integrator called *Verlet Integration* [17]. To introduce its concept, recall the well-known Newton's equations of motion:

$$d\boldsymbol{x}/dt = \boldsymbol{v} \tag{1}$$

$$d\boldsymbol{v}/dt = \boldsymbol{a} = \boldsymbol{F}(t)/m, \tag{2}$$

where $\boldsymbol{F}(t)$ is the force on the particle of mass m at position $\boldsymbol{x}$ with velocity $\boldsymbol{v}$ and acceleration $\boldsymbol{a}$ at time t. (Note bold fonts indicate vectors.)

Next, to integrate these equations into our graph drawing system, consider a single node movement in a discrete timeline of $t_0, t_1, \ldots$, with a constant time step $\Delta_t = t_{n+1} - t_n$. At time t_n, a graph node is at $\boldsymbol{x}_n = \boldsymbol{x}(t_n)$ with velocity $\boldsymbol{v}_n = \boldsymbol{v}(t_n)$ and acceleration $\boldsymbol{a}_n = \boldsymbol{a}(t_n)$ and force $\boldsymbol{F}_n = \boldsymbol{F}(t_n)$. Using (1) and (2), we can estimate the next node position and velocity at time t_{n+1} by

$$\boldsymbol{x}_{n+1} = \boldsymbol{x}_n + \Delta_t \boldsymbol{v}_{n+1} \tag{3}$$

$$\boldsymbol{v}_{n+1} = \boldsymbol{v}_n + \Delta_t \boldsymbol{a}_n = \boldsymbol{v}_n + \Delta_t \boldsymbol{F}_n/m \tag{4}$$

The above numerical integration is referred to as *Euler method*. It is clear that Euler method is simple and inexpensive to compute. However, because $\boldsymbol{x}$ and $\boldsymbol{v}$ are stored separately, they can get out of sync (mostly due to round-off errors) and become numerically unstable for large time steps with strong forces [15]. This problem can be avoided by taking a small time step. Unfortunately, this increases the computational cost.

One way to handle this is to correct the velocities after each time step by

$$\boldsymbol{v}_n = (\boldsymbol{x}_n - \boldsymbol{x}_{n-1})/\Delta_t \tag{5}$$

Now changes in positions directly influence velocities of the next time step (as opposed to estimating $\boldsymbol{v}_{n+1}$ from $\boldsymbol{v}_n$ as in (4) that amplifies round-off errors). Thus, using (5) helps increase the stability of the numerical integration. This allows us to be able to use larger time steps to increase speed. Substituting (5) in (4) gives:

$$\boldsymbol{v}_{n+1} = (\boldsymbol{x}_n - \boldsymbol{x}_{n-1})/\Delta_t + \Delta_t \boldsymbol{a}_n \tag{6}$$

Substituting (6) in (3) yields a basic *Verlet Integration* as:

$$\boldsymbol{x}_{n+1} = \boldsymbol{x}_n + (\boldsymbol{x}_n - \boldsymbol{x}_{n-1}) + \Delta_t^2 \boldsymbol{a}_n \tag{7}$$

Unlike the Euler method, the Verlet integrator computes the trajectories directly from the positions and not the velocities of the particle. The Verlet integration is area preserving in that the area bounded by adjacent trajectories is preserved [19]. Therefore, nodes can never move without bound like in the Euler algorithm. This provides some global stability. Next section refines the basic Verlet Integration for our proposed drawing algorithm in Section 3.3.

3.1.2. Modified Verlet integration

3.1.2.1. Adding a damping factor The first modification to the Verlet Integration is by adding a damping factor $\xi \in [0, 1]$ to $\boldsymbol{v}_n$ in equation (5) and follow the same calculation in previous subsection to obtain:

$$\boldsymbol{x}_{n+1} = \boldsymbol{x}_n + \xi(\boldsymbol{x}_n - \boldsymbol{x}_{n-1}) + \Delta_t^2 \boldsymbol{a}_n \tag{8}$$

Intuitively, in our graph drawing context, each node (particle) uses two-dimensional vectors to keep information of its current position, previous position and current acceleration for estimating its next position.

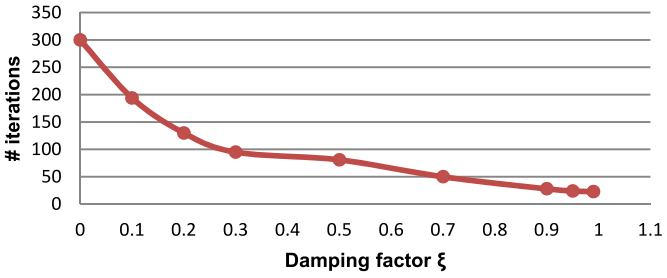


Fig. 3.1. Effects of ξ on convergence of a simple mesh graph size 30×30.

In the first Newton's equation of motion (1), velocities of a particle are maintained at the same rate and if there is no interference the particle will continue moving. However, this is not the case in reality, e.g., frictions or resistant forces when moving in fluid. The damping factor represents this effect to slow down $(\boldsymbol{x}_n - \boldsymbol{x}_{n-1})$, which is an approximation of current velocity. In fact, the velocity reduces exponentially by the factor of ξ for each time step.

The damping factor can have an impact on how long it takes for the graph drawing to converge (i.e., when the layout no longer changes). As shown in Fig. 3.1, the higher ξ is, the larger distance the particle moves and the faster the graph converges. When ξ is one, it converges the fastest as a regular Verlet integration. The damping factor in the Verlet integration disallows nodes to change their direction rapidly and consequently helps reduce oscillation. Thus, the choice of ξ is determined by experiments to balance between oscillation and convergence time.

3.1.2.2. Adding a temperature factor The second modification to the Verlet Integration is a notion of "temperature" to control the displacement of the nodes. Other force-directed methods use the temperature as in simulated annealing [16] when searching for graph layout that optimizes the energy. By mimicking a metal annealing process where metals are heated and then allowed to cool down slowly, simulated annealing schedules the temperature to control the search, in order to avoid a local minima problem. In our approach, we use the temperature control for the displacement of the nodes to help guarantee that the node will eventually stop moving over time. The modified Verlet Integration becomes:

$$\boldsymbol{x}_{n+1} = \boldsymbol{x}_n + T\left(\xi(\boldsymbol{x}_n - \boldsymbol{x}_{n-1}) + \Delta_t^2 \boldsymbol{a}_n\right) \tag{9}$$

where T represents a temperature. As shown in formula (9), the damping factor reduces the speed (approximated by $(\boldsymbol{x}_n - \boldsymbol{x}_{n-1})$) in the next time step resulting in the node inertia but the node displacement can still be influenced by net forces (from $\boldsymbol{a}_n$, which is $\boldsymbol{F}_n/m$). We initialize T to be one. Each iteration of graph drawing, T is reduced to be $T \cdot C$, where C is a cooling factor. Instead of using a cooling schedule as in simulated annealing, we use a constant cooling rate $C \in [0, 1]$ to serve our purpose. T will grow smaller making the node displacement eventually negligible.

3.1.3. Electrical repulsive forces

We define repulsive forces similarly to that of the Fruchterman and Reingold force-directed model [3]. As shown in Equation (10), for any node u and v, the repulsive force $\boldsymbol{f_r}$ is inversely proportional to the distance between them.

$$\boldsymbol{f_r} = K(v.\boldsymbol{x} - u.\boldsymbol{x})/\|v.\boldsymbol{x} - u.\boldsymbol{x}\|^2 \tag{10}$$

Here parameter K represents the strength degree of the repulsive force. Unlike the Fruchterman and Reingold model where K depends on the density of the graph, we keep K constant for simplicity. As shown in the equation, the farther the two nodes are away from each other, the weaker the repulsive force becomes, as it should be. This makes sense for unconnected (non-adjacent) nodes. For adjacent nodes, the distance between them will be later adapted to satisfy the specified distance by our constraint-based mechanism. Thus, in general, repulsive forces serve to make the graph layout more readable by increasing sparseness within bounds and consequently reducing edge crossings and chaotic layouts.

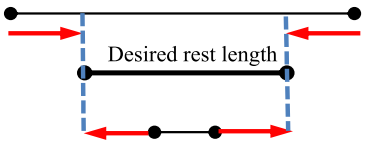


Fig. 3.2. Adjusting the edge distance to satisfy the edge constraint.

3.2. A weak constraint-based approach

The basic idea of the constraint-based approach to drawing grid distances of equal length in cloth simulation [14] is by iteratively moving each two connected nodes closer to satisfy a desired distance between them (called "rest length"). If the current edge distance between them is larger than the rest of the length, they will move toward each other, otherwise they will move away from each other, as shown in the top and bottom parts of Fig. 3.2, respectively. In either case, each of the end nodes will move in opposite directions with equal distance of half of the gap difference between them to satisfy the rest length.

Formally, given any two connected nodes u and v and its rest length r. D is a ratio of the gap difference between the actual and the desired distances. We move u to u' and v to v' to satisfy the rest length constraint as follows:

$$u'.\boldsymbol{x} = u.\boldsymbol{x} + 1/2D(v.\boldsymbol{x} - u.\boldsymbol{x}) \tag{11}$$

$$v'.\boldsymbol{x} = v.\boldsymbol{x} - 1/2D(v.\boldsymbol{x} - u.\boldsymbol{x}) \tag{12}$$

where $D = (1 - r/\|v.\boldsymbol{x} - u.\boldsymbol{x}\|)$.

Our *weak* constraint-based approach is similar to the above, but relaxes the distance constraints. Instead of moving the end nodes of each connected edge to have the resulting edge length satisfied its corresponding exact rest length r, the resulting edge can be of any length between $r \cdot e$ and r, where e is a proximity ratio constant $\in [0, 1]$. The smaller the e is, the weaker the constraint becomes since the interval length is larger and thus it is easier to satisfy the constraint. If e is one, the weak constraint-based approach will give the same effect as the original constraint-based approach.

The main advantage of the weak constraint-based approach is to allow more flexible constraint satisfaction. This is crucial and useful for rendering realistic systems where satisfying all rest length constraints is likely to be infeasible. To implement the weak-constraint concept, we use the same formulae (11) and (12) to move the end nodes, but D is computed as follows:

$$D = \begin{cases} (1 - r/\|v.\boldsymbol{x} - u.\boldsymbol{x}\|) & \text{if } \|v.\boldsymbol{x} - u.\boldsymbol{x}\| > r \\ (1 - r \cdot e/\|v.\boldsymbol{x} - u.\boldsymbol{x}\|) & \text{if } \|v.\boldsymbol{x} - u.\boldsymbol{x}\| < r \cdot e \end{cases} \tag{13}$$

Intuitively, when current edge distance already satisfies the interval constraint, there is no need to adjust the locations of connected nodes and Formula (13) is not applicable in such a case. Otherwise, Formula (13) computes the gap ratio based on whether the current edge distance is lower than the minimum bound or higher than the maximum bound of the interval constraint $[r \cdot e, r]$. For example, if $r = 6$, $e = 0.8$ and the current edge distance is 2, we would move the two end nodes to satisfy the minimum constraint, i.e., 4.8. To do this, D is computed, by (13), to be $(1 - 4.8/2) = -2.8$. By (11) and (12), the end node has a displacement of -1.4 and 1.4, respectively. Thus, the two end nodes

Graph Drawing (Note: Recall that bolded variables are vectors)
Input: V = a set of nodes and E = a set of edges between nodes in V with corresponding *weights*
Output: An graph $G(V, E)$ layout with high proximity to given weighted edge constraints
Begin
1 *For* each u in V *do* /* Initialize node positions, previous positions and accelerations
2 initialize $u.\boldsymbol{pos}$ randomly;
3 $u.\boldsymbol{ppos} = u.\boldsymbol{pos}$; $\boldsymbol{a} = \mathbf{o}$
4 *End For*;
5 Initialize: temperature T; Set: time step Δ_t, repulsive force strength K, cooling rate C, damping factor ξ and precision control parameter Γ;
6 *Repeat*
7 *For* each u and v in V where $u \neq v$ *do* /* add electrical repulsive forces
8 $\boldsymbol{df} = v.\boldsymbol{pos} - u.\boldsymbol{pos}$;
9 $\boldsymbol{PushForce} = K * \boldsymbol{df} / \|\boldsymbol{df}\|^2$;
10 $u.\boldsymbol{a} = u.\boldsymbol{a} - \boldsymbol{PushForce}$;
11 $v.\boldsymbol{a} = v.\boldsymbol{a} + \boldsymbol{PushForce}$;
12 *End For*;
13 *For* each u in V *do* /* move nodes by the modified Verlet Integration
14 $\boldsymbol{tmp} = u.\boldsymbol{pos}$;
15 $u.\boldsymbol{pos} = u.\boldsymbol{pos} + T * (\xi\,(u.\boldsymbol{pos} - u.\boldsymbol{ppos}) + \Delta^2_t\, u.\boldsymbol{a})$;
16 $u.\boldsymbol{a} = \mathbf{0}$;
17 $u.\boldsymbol{ppos} = \boldsymbol{tmp}$
18 *End For*;
19 *For* $i = 1$ to Γ *do* /* apply weak weighted edge constraints
20 For each edge (with end nodes u, v, and rest length constraint r) in E *do*
21 *EnforceWeakConstraint*(u, v, r, T)
22 *End For*
23 *End For*;
24 $T = T * C$;
25 *Until* no change in the graph layout;
End.

Fig. 3.3. Weak constraint-based graph drawing algorithm.

Procedure EnforceWeakConstraint(p, q, r, T)
Input: connected nodes p and q with a corresponding edge constraint r (rest length or weight/distance) and temperature T
Output: distance between p and q lies in $[r \cdot e, r]$, (i.e., satisfies the weak constraint), where e is a proximity ratio in [0, 1]

1 *Begin*
2 Set the proximity ratio e ;
3 $\boldsymbol{df} = q.\boldsymbol{pos} - p.\boldsymbol{pos}$;
4 *distance* = $\|\boldsymbol{df}\|$;
5 *If distance* $> r$ *or distance* $< r * e$ *then* /* when weak constraint is not met
6 *begin* $D = 1 - r/distance$;
7 *if distance* $< r * e$ *then*
8 $D = 1 - r * e / distance$;
9 $p.\boldsymbol{pos} = p.\boldsymbol{pos} + \frac{1}{2}\, D * T * \boldsymbol{df}$;
10 $q.\boldsymbol{pos} = q.\boldsymbol{pos} - \frac{1}{2}\, D * T * \boldsymbol{df}$;
11 *End if*
12 *End.*

Fig. 3.4. Weak (edge) constraint enforcement procedure.

move further apart (in opposite direction) of equal distance giving the resulting edge distance of $(2 + 1.4 + 1.4)$ as desired. In the same scenario, if the current distance is 10, the two end nodes will both have a displacement of 2 to satisfy the rest length constraint of 6.

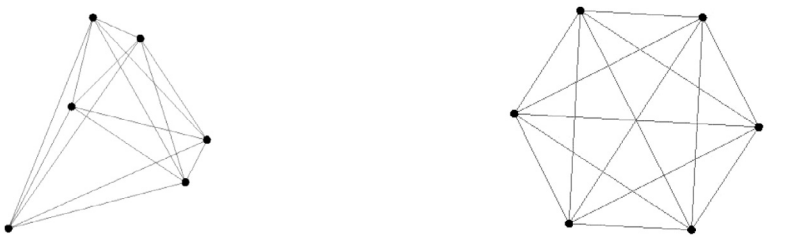
Note that the weak constraint allows the distance to be smaller or equal to the rest length but not bigger. The reason is because our approach, unlike the other force-directed algorithms, does not employ spring or electrical systems to create "attractive force" to pull nodes closer. Instead it relies on the constraint mechanism to adjust the edge distance between the nodes. Thus, the maximum bound for the constraint should be the rest length itself. On the other hand, the edge distance that is shorter than its rest length is adjustable by repulsive forces during the next iteration of the graph drawing (more details later).

3.3. The algorithm and illustrated examples

Combining the above features, we obtain an overall algorithm as shown in Figs. 3.3 and 3.4.

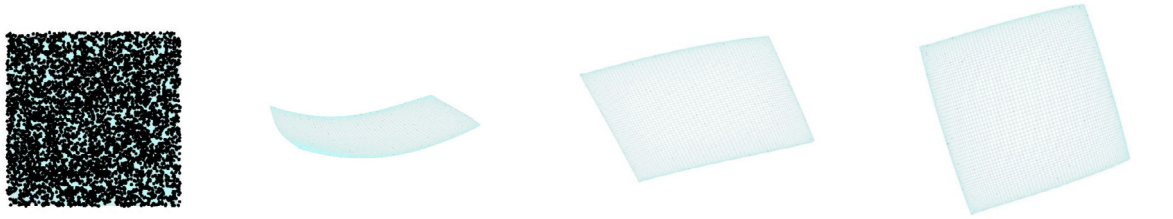
As shown in Fig. 3.3, the algorithm starts by initializing node positions to random locations along with its previous positions and accelerations, and other parameters (Lines 1–5). In each round of iteration, nodes are moved by the influences of electrical repulsive forces (Lines 7–12, as in formula (10)), the modified Verlet Integration (Lines 13–18, as in formula (9)), and connected nodes are adjusted to satisfy the weak edge constraints (Lines 19–23).

The enforcement of weak edge constraints shown in Fig. 3.4 takes the two connected nodes and checks if they satisfy the weak constraint (see also output specification). If they do not (Line 5) then they are to be moved to an equal distance in opposite direction (Lines 9–10) to make the new distance between them be



a) hard constraints (iteration 1000) b) weak constraints (iteration 290)

Fig. 3.5. A small example illustrating the impact of weak edge constraints.



a) initial graph b) iteration 10 c) iteration 20 d) final layout at iteration 290

Fig. 3.6. Evolution of graph drawing of a small mesh with dimensions 70 × 70.

the bound of the weak constraint. If the current distance is smaller (larger) than the lower (upper) bound of the weak constraint, the new distance will be the lower (upper) bound of the weak constraint, as shown in Lines 7–8 (Lines 5–6). These new positions are computed based on the formula (13) but the temperature factor is added to be consistent with that of the Verlet integration (Line 15 of Fig. 3.3) for the same purpose, i.e., to help guarantee that the graph drawing will eventually stop over time.

Note that after the distance between each connected nodes are adjusted to satisfy the weak constraint, they may be moved apart again in the next iteration due to electrical repulsive force. These mechanisms can be thought of as an elastic adjustment that interacts between physical and constraint systems until the algorithm stops (or converges) when no further change in the graph layout occurs (Line 25).

Ideally, these node movements repeat until each of the edge distances satisfies its corresponding weak constraint. However, in practice because of the elasticity of the node movement, a parameter Γ is used to control the computational cost and the *precision* of the graph in terms of the degree of the constraint satisfaction in the graph (Line 19 of Fig. 3.3). It represents the number of times all the constraints are used once to update the edges. The higher the degree is, the closer the edges are to satisfy the weak constraints. If Γ is large enough, the graph drawing will repeat until no graph layout changes, in which case we obtain the optimal weak edge constraint satisfaction. However, if Γ is too small, the computation will stop before the graph layout evolves to maximally satisfy the edge constraints. Thus, the use of Γ allows a tradesoff between precision of the graph (with respect to edge constraint satisfaction) and the time.

In all of our illustrations and experiments we set the parameters as follows. For physic related parameters, namely damping factor $\xi = 0.95$, time step $\Delta_t = 0.5$, strength of repulsive force $K = 10$ and cooling rate $C = 1$ (for synthetic graphs) and $C = 0.99$ (for realistic graphs) in order for the synthetic data to render the best possible graph layout. For weak constraints, we set constraint satisfaction degree $\Gamma = 20$ and proximity ratio $e = 0.3$. The damping factor and the time step are common default values in force directed algorithms. The rest of the parameters are empirically determined by experiments to systematically search for appropriate values.

3.3.1. Illustrated examples

Below illustrates results of the algorithm on small graphs and impacts of some parameters on them.

Example 1 *(Constraint-based versus weak constraint-based).* We ran an experiment on a complete graph of six nodes with edge constraints of length one. Fig. 3.5 shows the results obtained when we set the proximity ratio, $e = 1$ (hard constraints) and $e = 0.3$ (weak constraints) in a) and b), respectively.

For the hard constraint, the drawing keeps oscillating and does not converge. The result obtained was captured when we forced the algorithm to stop after 1000 iterations. Although *not* all edge constraints are satisfied (since the constraints are not realistically realizable), for the weak constraints in b), the algorithm halts after 290 iterations and produces a more symmetric and aesthetic result than that of a).

Example 2 *(Evolution of a simple mesh layouts).* This example shows how the resulting graph layouts of a synthetic mesh with 4900 nodes (70 rows and 70 columns) are evolved in various stages.

As shown in Fig. 3.6, the layouts change dramatically at earlier stages (e.g., iterations 10 and 20). The graph stops after 290 iterations but in fact the final layout looks similar to that obtained after 100 iterations.

4. Comparisons with state-of-the-art approaches

This section compares our approach with Hu's algorithm [7] and ForeAtlas2 [13], two popular state-of-the-art graph drawing techniques available in the widely used Gephi tool. All the three approaches use physical properties and aim to address drawing of aesthetic (i.e., symmetry, space layout, cross-free edges) and efficient layouts of Big graphs. Recall that Hu's algorithm combines multi-level approach with octree data structures to improve graph

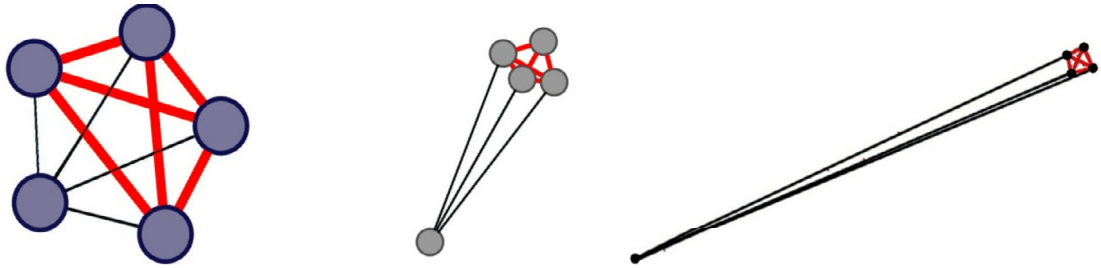


Fig. 4.1. A five-node complete graph with black (red) edges of distance 10 (0.5). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Table 1
Compare time and memory.

Graph $G(V, E)$	Size (i.e., $\|E\|$)	Order (i.e., $\|V\|$)	Density (avg. degree)	Memory (Megabytes)		Running Time (Seconds)		
				Hu (Extra)	Ours	Hu	ForceAtlas2	Ours
1. Small Cloth	760	400	3.8	69 (9)	22.9	4.2	25	**2.3**
2. Small Tree	156	155	1.9	60 (9)	26.9	35	40.9	**5.3**
3. Small Knotted Mesh	4,904	2,500	3.9	83 (23)	25.7	**20.4**	60.8	56.6
4. Multiple Rings	14,240	4,800	5.9	162 (101)	24.2	38.4	61.6	**34.3**
5. Medium Tree	3,905	3,906	1.9	68 (11)	26.3	70.2	120	**54.3**
6. Large Tree	19,607	19,698	1.9	164 (42)	30.8	**350**	>3600	3600
7. Large Cloth	19,800	10,000	3.9	164 (98)	29.6	**96.5**	181.1	161.1
8. Large Knotted Mesh	19,804	10,000	3.9	182 (79)	28.9	**104.9**	179	162.7
9. Collaboration	914	379	4.8	50 (35)	23	6.7	30.3	**6.6**
10. Small Social Network	1,693	150	22.6	49 (19)	23	4.7	6	**2.9**
11. Social Network	26,749	1,034	51.7	282 (220)	29	40.6	10	**9.5**
12. Association Rules	65,611	601	218.3	384 (278)	29	21.1	5.9	**1**
13. Yelp in PA	100,000	541	455.3	434 (277)	32.2	28.3	13.9	**7.3**
14. Yelp in PA and NC	500,000	1,094	904.8	OOM	61	OOM	OOM	**23.6**

drawing scalability. ForceAtlas2, known for its rendering of networks of real data, uses a mechanism to balance between the graph drawing precision and speed along with degrees of nodes to effectively visualize weighted straight edges on realistic large-scale networks. Our approach is not multi-level but relies on the interplays between repulsive forces and edge weak constraints to drive the algorithm. Our comparisons are illustrated in three aspects: general concepts, complexity and resource usage, and aesthetic quality. These will be described in subsections below.

4.1. General concepts

We illustrate how each of the approaches draws graph layouts using a small example of a complete graph of five nodes, as shown in Fig. 4.1. The graph has a complete sub-graph of four nodes that are connected with edges of weight 2 (shown in red) and the rest of the edges have weight 0.1 (shown in black). For visualization, weights are commonly used interchangeably as an inverse of the distance to maintain linearity between them although one can use any function to relate the two. For our illustration, we can think of edge weight constraints of 2 and 0.1 are the same as edge distance constraints of 0.5 and 10. Thus, we expect to see the red edge length approximately 0.5 and the black edge length of about 10.

As shown in Fig. 4.1 a), although Hu's approach displays weight differences through the edge thickness, it does not maintain the information that the red weight is 20 times of the black one. As shown in the figure, Hu's algorithm does not differentiate the edge distances of 0.5 and 10. In fact, the distances between all nodes seem to be about the same in this example. This puts Hu's approach at the disadvantage when distances are crucial information to display.

On the other hand, ForceAtlas2 and our approach show similar layouts in that they differentiate strength of the distance and thus the resulting graphs have two clusters where one has four nodes and the other is an isolated node. However, as shown in Fig. 4.1 b), ForceAtlas2 does not maintain the integrity of the given constraints whereas our approach does. In particular, the black edge length should be 20 times of those of the red edges. This is the case in our approach.

4.2. Complexity and resource usage

Hu's algorithm reduces repulsive force calculation from $O(|V|^2)$ to $O(|V|\log|V|)$ by using a combination of a multi-level approach and octree data structures. Thus, adding attractive force calculation, Hu's overall algorithm takes $O(|V|\log|V| + |E|)$ [7]. Similarly, ForceAtlas2 takes $O(|V|^2 + |E|)$ but has an option to use the octree mode to improve the bound to be the same as that of Hu's algorithm. Our approach requires $O(|V|^2)$ to compute repulsive force and $O(|E|)$ to enforce the edge constraint. Thus, the time complexity of all the three algorithms are essentially $O(|V|^2 + |E|)$ where improvement can be made by using octree data structures. Although the theoretical upper bounds are the same, in practice, actual running times of these approaches may differ by a constant factor of the bound, as a result of other parameter controls (e.g., degrees of nodes, satisfaction constraints) or the fact that the graph inputs may not be the worst case. We ran experiments to compare actual running times and memory usage of the three approaches: Hu's Multi-level approach, ForceAtlas2, and our weak constraint-based approach.

The experiments were run on Intel(R) Core(TM) i7-4720HQ CPU @ 2.60 GHz (8 CPUs) with 2.6 GHz, and 4096 MB RAM on Windows 8.1 OS. Hu's approach and ForceAtlas2 were implemented on the Gephi tool, while ours was implemented on *OpenGL* [20]. Although the results are obtained from the same machine on different environments, they provide sufficient estimated comparisons of these approaches on various scales and characteristics of the graph inputs (e.g., dense, synthetic, etc.). Table 1 shows results in two

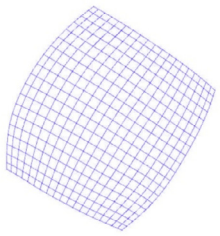
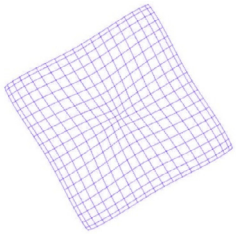
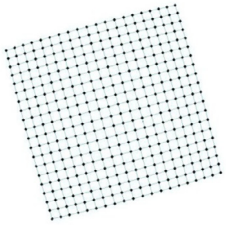
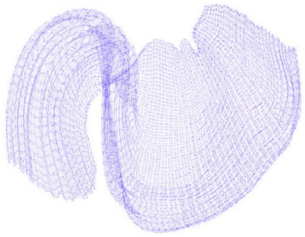
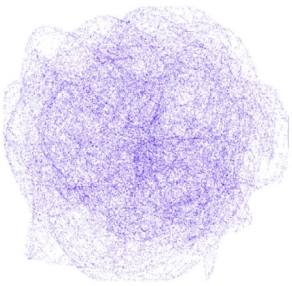
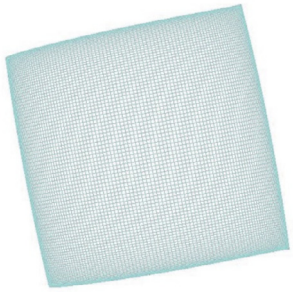
Hu's Approach	ForceAtlas2	Our Weak Constraint Approach	Comments
			**Simple Mesh of size 20x20** (Graph 1 of Table 1) All layouts are symmetry. Our algorithm converges fastest.
			**Simple Mesh of size 100x100** (Graph 7 of Table 1) Only our algorithm terminates with symmetric layout in the least number of iterations.

Fig. 4.2. Simple mesh layouts.

parts: memory usage and running time over 14 graphs, eight of which are of size over 10 thousand including two of size over 100 thousands. The top seven graphs are synthetic and the rest are real data. The density is estimated by the average degrees (i.e., a ratio of $2|E|$ over $|V|$ for a graph $G(V, E)$).

As shown in Table 1, on memory usage, Hu's algorithm requires additional memory to store multi-level layout information, as indicated in the parenthesis of the Hu (Extra) column. We are not able to obtain memory usage of ForceAtlas2 to compare. It is clear that our approach uses significantly less memory than that of Hu's. On Graph 13 of size 500,000, both Hu's algorithm and ForceAtlas2 are out of memory (OOM), while our approach uses 61 megabyte memory and generates graph in 23.6 seconds.

On running time, as shown in Table 1, the bold letter indicates the best running time of the three in each graph experiment. Our approach outperforms Hu's approach on 10 out of 14 graphs. Hu's algorithm performs better than ours in Graphs 3 (Small Knotted Mesh), 6 (Large Tree), 7 (Large Cloth) and 8 (Large Knotted Mesh), all of which are synthetic graphs. Comparing with ForceAtlas2, our approach does better in all cases. As graph scales increase, especially from graphs 11–14, the running times become more different. Although these results are from different platforms, based on the number of iterations, our approach requires a few iterations to correct the coarse layout whereas ForceAtlas2 requires many more and is likely to be trapped in local minima as most force-directed algorithms are in large-scale graphs. Overall running time experiments show that our approach can handle Big graph well for both synthetic and real data graphs. Next we need to look at the quality of the resulting graph layouts.

4.3. Aesthetic quality

This section shows the graph layout results obtained from the three approaches in synthetic and real data graphs.

4.3.1. Synthetic graphs

Synthetic graphs tend to be symmetric. This should reflect by good graph drawing algorithms. Our experiments compare four families of synthetic graphs: *simple mesh, knotted mesh, multiple rings* and *perfect k-ary trees*. Although ForceAtlas2 was not designed for mesh graphs, we include their results for complete comparisons.

Fig. 4.2 shows how each approach scales on the same structure, particularly, a simple mesh from 400 nodes to 10,000 nodes. The resulting graphs from Hu's approach and ForceAtlas2 are not symmetric. In fact, it is not clear if ForceAtlas2 ran into local minima problem. Our approach appears to scale better than the other two approaches both in terms of the aesthetic quality and the running time (shown in Table 1).

Fig. 4.3 shows results for knotted mesh structure with 2500 nodes in the upper row and 10,000 nodes in the bottom row. The results are similar to that of simple mesh structure. Hu's approach performs slightly better than ForceAtlas2. This could be because ForceAtlas2 was designed to follow physic laws that do not necessary hold for abstract graphs. Overall our approach renders good readable layout when we increase the size of the graphs.

Fig. 4.4 shows layouts of Multiple ring structure from a small scale of 800 nodes to that of 4800 nodes. In a small scale, ForceAtlast2 and our approach produce layouts as expected. The layout from Hu's approach is not as symmetrical. For a larger graph size, both layouts from Hu's approach and ForceAtlas2 are not as symmetrical as ours. In fact, ours looks similar to Hu's smaller graph. Our layouts are most aesthetic (symmetry) for both scales.

Fig. 4.5 shows the graphs of the tree structure. All performs very well for the 5-ary tree depth 3 of 156 nodes. However, the layouts are somewhat messy for the 5-ary tree depth 5 of 3,905 nodes and 7-ary tree depth 5 of 19,698 nodes. Ours is most symmetric but the layout could have improved if the branches are better spread.

4.3.2. Real data graphs

This section illustrates applications to real data. We consider two applications below.

4.3.2.1. Collaboration network We obtained a publicly available data set of a collaboration network [21]. In particular, we experimented on the largest component network of 379 scientists who collaborated on network theory and experiments. Nodes represent authors and edges represent co-authorships. The given edge weights represent the degrees of frequency and contribution in the co-authorship. (See [22] for more details.)

Fig. 4.6 shows the results obtained from the three approaches. All layouts are presentable and have similar structures. However, recalling our illustrated example of Fig. 4.1 that distinguishes the

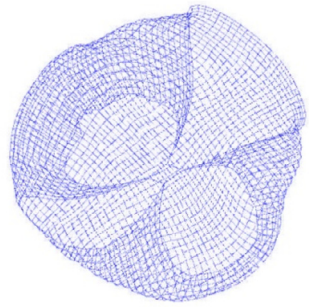
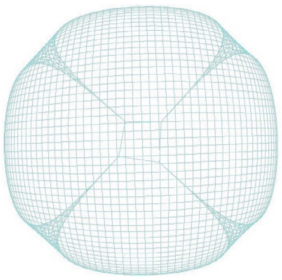
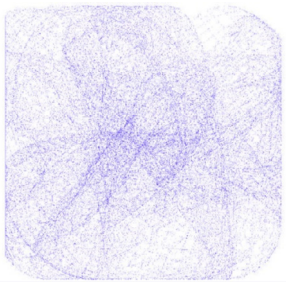
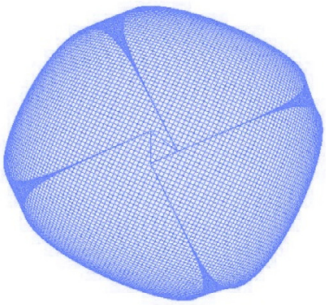
Hu's Approach	ForceAtlas2	Our Weak Constraint Approach	Comments
			**Knotted Mesh of size 50x50** (Graph 3 of Table 1) Based on Table 1, Hu's approach has the best running time but our graph layout is more aesthetic (symmetry).
			**Knotted Mesh of size 100x100** (Graph 8 of Table 1) Based on Table 1, Hu's approach has the best running time. However, the graph layout from our approach is much more readable than that of Hu's.

Fig. 4.3. Knotted mesh layouts.

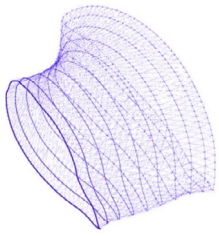
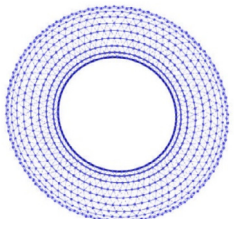
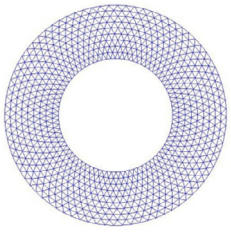
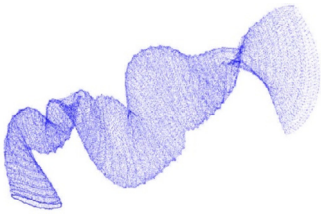
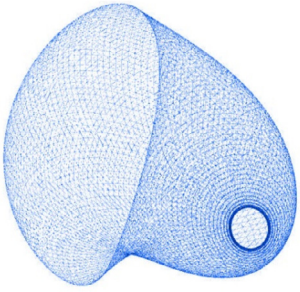
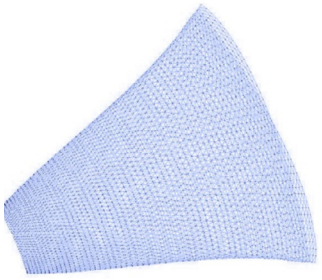
Hu's Approach	ForceAtlas2	Our Weak Constraint Approach	Comments
			**Multiple Rings of size 800 nodes** (10 rings with 80 segments of perimeters per ring)
			**Multiple Rings of size 4800 nodes** (60 rings with 80 segments of perimeters per ring) (Graph 4 of Table 1)

Fig. 4.4. Multi-ring layouts.

concepts of the three approaches, we can see that Hu's layout a) and ForceAtlas2 b) are very similar because edge weights are not constrained. Ours in c) is very different because the weight is much more important. Thus, as shown in Fig. 4.6 c) our network is more elaborated in showing distance/weight realistic ratios.

4.3.2.2. Yelp's business data In this example, we apply our technique to visualize business in the Yelp dataset from publicly available Yelp Dataset Challenge (http://www.yelp.com/dataset_challenge). The data set includes 61,184 businesses. Each business has associated fields such as *business id*, *state*, and *attributes* (e.g., *HappyHour, AcceptCreditCard, GoodForGroup*, *OutdoorSeating*, and *PriceRange*). There are 38 attributes where each business has at most 31 attributes. Different businesses can have different number of attributes as well as different attribute values. Each attribute of each business has one associated business attribute value. For our study, we consider 541 businesses, all of which are located in Pennsylvania (PA).

To visualize the similarity of businesses in the Yelp's data using the three graph drawing approaches, we view the data as a complete graph where each node represents a business and each edge weight represents the degree of similarity between the two businesses end points. The similarity of business u and business v is computed by the ratio of the number of common attributes between u and v with the same corresponding attribute values to the number of distinct attributes in u or v or both. For example, u has 10 attributes and v has 15 attributes, where both have eight common attributes with five attributes that have the

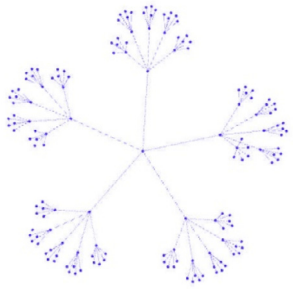
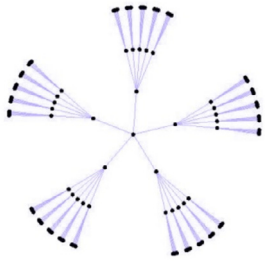
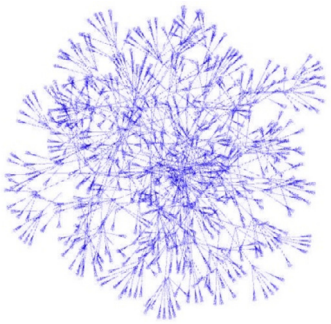
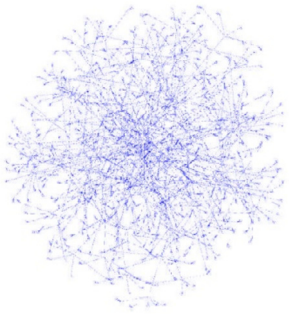
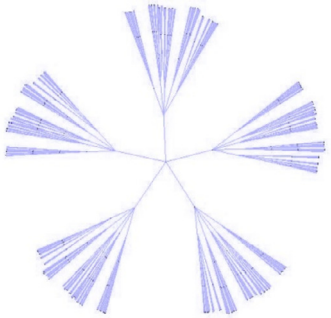
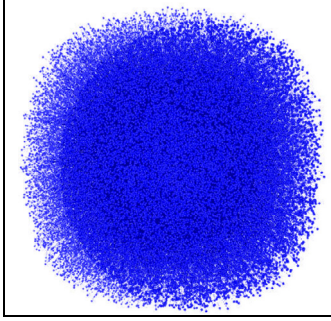
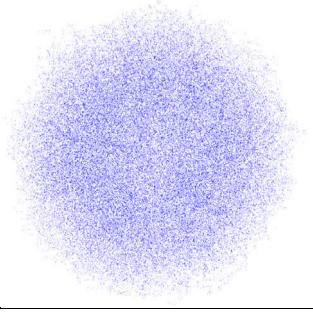
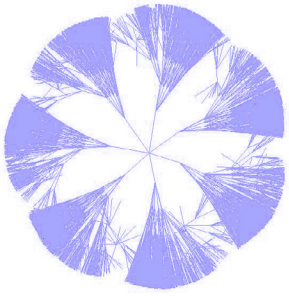
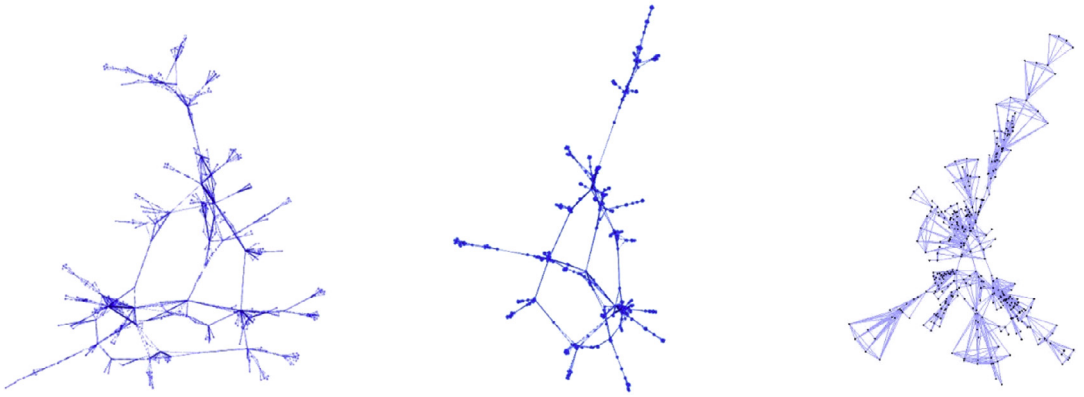
Hu's Approach	ForceAtlas2	Our Weak Constraint Approach	Comments
			**Perfect 5-ary tree of depth 3:** 156 nodes (Graph 2, Table 1)
			**Perfect 5-ary tree of depth 5:** 3905 nodes (Graph 5, Table 1)
			**Perfect 7-ary tree of depth 5:** 19,698 nodes (Graph 6, Table 1)

Fig. 4.5. Perfect n-ary tree layouts.

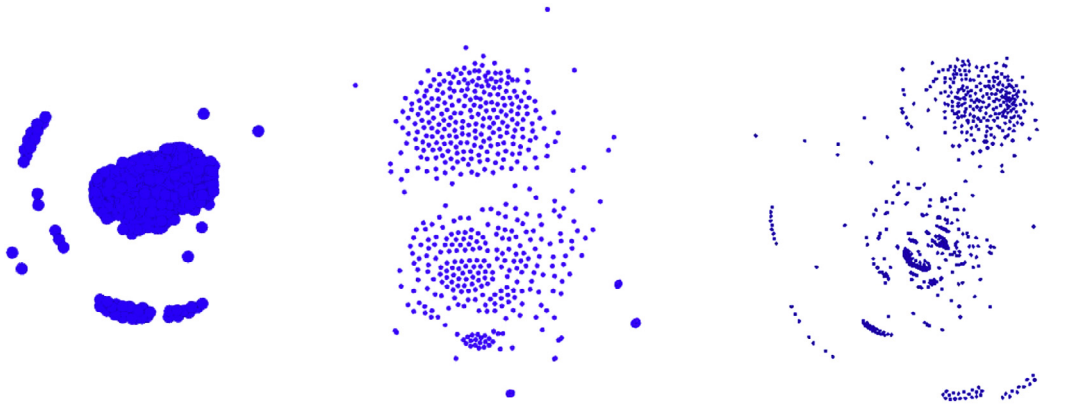


a) Hu's Approach b) ForceAtlas2 c) Our Weak Constraint Approach

Fig. 4.6. Co-authorship collaboration network.

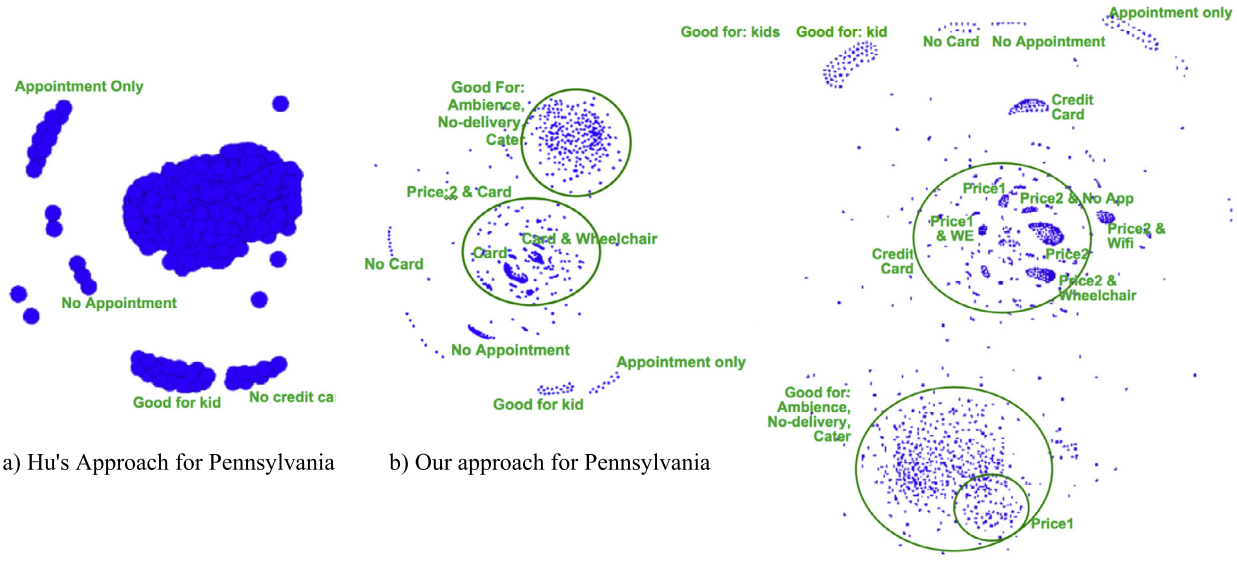
same attribute values. Thus, the similarity between u and v is $5/(10 + 15 - 8) = 5/17$. For any edge between businesses u and v, we define the weight of the edge to be the similarity between u and v. Thus, the higher similarity they have, the closer they are to each other. By omitting the drawing of the straight line edges, the layouts obtained give clusters of "similar" businesses as shown in Fig. 4.7.

Fig. 4.7 shows that the layout from Hu's approach in a) is quite different from those of ForceAtlas2 and ours. The main difference is that Hu's result has one largest cluster with four small clusters surrounding it whereas the results from ForceAtlas2 and especially our approach both have two large central clusters surrounded by several small clusters. This is because Hu's approach does not distinguish edge weights by their lengths. Thus, the resulting graph tends to be denser than others. Comparing to ForceAtlas2, our approach maintains similar structures of business clusters but scatters them more than those of ForceAtlas2. This is because ours try to keep the integrity of the relative weights/distances in the



a) Hu's Approach b) ForceAtlas2 c) Our Weak Constraint Approach

Fig. 4.7. Visualizing Yelp's similar businesses in Pennsylvania.



a) Hu's Approach for Pennsylvania b) Our approach for Pennsylvania

c) Ours for Pennsylvania and North Carolina

Fig. 4.8. Categories of Yelp's similar businesses.

overall graph layout. To verify if the resulting graphs reflect meaningful clusters, we manually observed the labels of these clusters as shown in Fig. 4.8.

As shown in Fig. 4.8, we compare clusters of Yelp's business in Pennsylvania produced by Hu's approach in a) and ours in b). Our clusters are more refined than those of Hu's. For example, in b) we capture several clusters in the center such as businesses that accept cards, or are good for ambience atmosphere with cater but no delivery services that Hu's approach is not able to capture. Fig. 4.8 c) shows the results obtained by our approach when we extend the Yelp's data to include businesses in North Carolina making up a total of 1094 businesses. The visual analysis of this case shows some new categories such as businesses with Price 2 combined with wi-fi when we include businesses in North Carolina. Unfortunately, the Gephi tool ran out of memory due to the high number of edges. Thus, we are not able to compare results with Hu's approach or ForceAtlas2 in this case. This could be another evidence that supports the power of our approach to scale.

5. Conclusions and future work

This paper presents a simple graph visualization technique that aims to efficiently draw aesthetically pleasing large-scale straight-line weighted edge graphs. The proposed approach is based on a constraint-based approach to cloth simulation, commonly known in computer graphic and game development to provide efficient realistic simulation of particles with given distance constraints. We introduce a *weak constraint-based approach* that relaxes rigid distance preference requirements (within acceptable bounds) in order to improve graph readability. The weak constraints allow flexibility in trading the distance precision with the layout quality. Using the weak constraint enforcement with appropriate physic-inspired techniques, our approach yields graphs that balance the weight precision with aesthetic quality as well as enhancing the scalability for Big graphs.

We compare our approach with existing techniques. We have experimented on over 14 synthetic and real data sets, most of which are shown in this paper. The results show that in many cases, our algorithm provides more aesthetic and informative graphs than those of the two popular state-of-the-art approaches and that our algorithm scales to big graphs with over 10,000 nodes quite well. In fact, based on the running time results obtained on different platforms, our approach outperformed the other two in ten out of 14 data sets, some of which have significant running time differences. Thus, we anticipate that our approach is likely

to, on the average, outperform the two state-of-the-art approaches when they run on the same platform as well.

In general, the results obtained show enhanced and promising outcomes toward a general-purpose graph drawing technique for big data analytics. One obvious limitation of the quality of the graph readability is due to the use of straight-line edges especially when graphs contain excessive number of edges. In such cases, we can use the approach for clustering by applying weights to similarity measures and omit drawing the edges as illustrated in the Yelp's data example. Alternatively, an extended approach to non-straight edges would be beneficial.

References

[1] P. Eades, A heuristic for graph drawing, Congr. Numer. 42 (1984) 149–160.
[2] P. Eades, N.C. Wormald, Fixed edge-length graph drawing is NP-hard, Discrete Appl. Math. 28 (2) (Aug. 1990) 111–134.
[3] T.M.J. Fruchterman, E.M. Reingold, Graph drawing by force-directed placement, Softw. Pract. Exp. 21 (11) (Nov. 1991) 1129–1164.
[4] P. Gajer, M.T. Goodrich, S.G. Kobourov, A multi-dimensional approach to force-directed layouts of large graphs, in: J. Marks (Ed.), Graph Drawing, Springer, Berlin, Heidelberg, 2000, pp. 211–221.
[5] R. Hadany, D. Harel, A multi-scale algorithm for drawing graphs nicely, Discrete Appl. Math. 113 (1) (Sep. 2001) 3–21.
[6] D. Harel, Y. Koren, A fast multi-scale method for drawing large graphs, in: Proceedings of the 8th International Symposium on Graph Drawing, London, UK, 2001, pp. 183–196.
[7] Y. Hu, Efficient, high-quality force-directed graph drawing, Math. J. 10 (1) (2005) 37–71.
[8] T. Kamada, S. Kawai, An algorithm for drawing general undirected graphs, Inf. Process. Lett. 31 (1) (Apr. 1989) 7–15.
[9] S.G. Kobourov, Spring Embedders and Force Directed Graph Drawing Algorithms, arXiv:1201.3011 [cs], Jan. 2012.
[10] C. Walshaw, A multilevel algorithm for force-directed graph drawing, in: J. Marks (Ed.), Graph Drawing, Springer, Berlin, Heidelberg, 2000, pp. 171–182.
[11] J. Ellson, E. Gansner, L. Koutsofios, S.C. North, G. Woodhull, Graphviz—open source graph drawing tools, in: P. Mutzel, M. Jünger, S. Leipert (Eds.), Graph Drawing, Springer, Berlin, Heidelberg, 2001, pp. 483–484.
[12] M. Bastian, S. Heymann, M. Jacomy, Gephi: an open source software for exploring and manipulating networks, in: Third International AAAI Conference on Weblogs and Social Media, 2009.
[13] M. Jacomy, T. Venturini, S. Heymann, M. Bastian, ForceAtlas2, a continuous graph layout algorithm for handy network visualization designed for the Gephi software, PLoS ONE 9 (6) (Jun. 2014) e98679.
[14] C. Zeller, Cloth simulation on the GPU, in: ACM SIGGRAPH 2005 Sketches, New York, NY, USA, 2005.
[15] A. Fuhrmann, C. Groß, V. Luckas, Interactive animation of cloth including self collision detection, in: Proceedings of Conference on Computer Graphics, Visualization and Computer Vision, WSCG, 2003.
[16] S. Kirkpatrick, C.D. Gelatt, M.P. Vecchi, Optimization by simulated annealing, Science 220 (4598) (1983) 671–680.
[17] L. Verlet, Computer "experiments" on classical fluids. I. Thermodynamical properties of Lennard-Jones molecules, Phys. Rev. 159 (1) (1967) 98.
[18] G. Wallner, Simulating, animating and rendering clothes, 1999. Available at URL, https://www.cg.tuwien.ac.at/courses/Seminar/SS2007/Cloth-Wallner.pdf, 2016.
[19] P. Young, The leapfrog method and other "symplectic" algorithms for integrating Newton's law of motion, Physics 115 (242) (2014), University of California at Santa Cruz. Available at URL, http://young.physics.ucsc.edu/115/leapfrog.pdf, 2016.
[20] D. Shreiner, Bill The Khronos OpenGL ARB Working Group, OpenGL Programming Guide: the Official Guide to Learning OpenGL, Versions 3.0 and 3.1, Pearson Education, 2009.
[21] M. Newman, NetScience, Available at URL, http://vlado.fmf.uni-lj.si/pub/networks/data/collab/netscience.htm, 2006, Pajek datasets, 2016.
[22] M. Newman, Scientific collaboration networks. II. Shortest paths, weighted networks, and centrality, Phys. Rev. E 64 (1) (2001) 016132.