Journal of Visual Languages and Computing 44 (2018) 72–88

Contents lists available at ScienceDirect

Journal of Visual Languages and Computing

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

View Points

Neighborhood beautification: Graph layout through message passing

Severino F. Galán[a,*], Ole J. Mengshoel[b]

[a] Artificial Intelligence Dept., UNED, C/ Juan del Rosal, 16, Madrid 28040, Spain
[b] Carnegie Mellon University, NASA Research Park, Moffett Field, CA 94035, USA

ARTICLE INFO

Article history:
Received 18 March 2016
Revised 15 June 2016
Accepted 29 November 2017
Available online 1 December 2017

Keywords:
Graph drawing
Aesthetic graph layout
Neighborhood interaction
Message passing

ABSTRACT

Graph layout algorithms are used to compute aesthetic and useful visualizations of graphs. In general, for graphs with up to a few hundred nodes, force-directed layout algorithms produce good layouts. Unfortunately, for larger graphs, they often get stuck at local minima and have high computational complexity. In this paper, we introduce a novel message passing technique for graph layout. The key idea of our message passing technique is that an aesthetic layout can be obtained if each node independently suggests aesthetic placements of its neighbors. In other words, every node sends messages to its neighbors, indicating new and better positions for them. As a result, the new technique, which we call *Neighborhood Beautification*, provides a new perspective that turns out to give a useful trade-off between the excellent layout quality reached by force-directed methods and the fast runtime achieved by algebraic methods. Neighborhood Beautification reduces, in many cases, the computational cost of force-directed algorithms, since only interactions between neighboring nodes are considered. Experimentally, we show that Neighborhood Beautification produces high-quality layouts for grid-like graphs but is outperformed by force-directed algorithms in the case of more complex graphs.

1. Introduction

Graphs are widely used in artificial intelligence, computer engineering, computer science, mathematics, and statistics to model objects and connections between them. They have been successfully applied in areas like automata theory, automatic planning, databases, electrical power networks, machine learning, network security, probabilistic graphical models, social networks, software engineering, and VLSI technology [26]. In order to easily understand and manipulate the data represented by graphs, aesthetic and useful visualizations, including layouts, of them are needed. Many graph layout methods have been developed [12,35,49]. Objectives that have been considered in developing such methods include [12,47]: minimization of edge crossings, uniformity of edge lengths, even distribution of nodes, and maximization of symmetries.

The present work deals with the problem of graph layout in the sense of nicely drawing undirected graphs whose edges are straight lines. This problem reduces to appropriately positioning nodes in a two dimensional plane. A popular approach to this problem is the force-directed technique, where the graph is associated with a system of interacting physical objects. This approach defines the energy of the system such that, when a minimum is achieved, the layout tends to be nice and useful. Given initial positions for the interacting physical objects, the system evolves by reducing its energy. Variants of this approach differ in how the energy is defined [11,14,20,37]. Force-directed graph layout can be combined with other methods for initializing the position of the nodes, for example, treemap-based methods [44].

Although force-directed methods are intuitive, easy to implement, and produce acceptable results for graphs of medium size (up to a few hundred nodes), they also have important disadvantages:

- The number of iterations needed to produce a good layout can be quite high. For example, since graphs are usually initialized with random positions for their nodes, in the case of large graphs the convergence to a global minimum is often too slow or even hard to achieve due to the presence of many local but non-global minima. Therefore, layout of large graphs requires the use of techniques in addition to force-directed methods. The multi-level technique [30,31,43,55] (see Section 2.2), which works regardless of the initial positions of the nodes and only requires the graph structure, is a convenient option for large graph drawing. Using this technique, a sequence of graphs $(G^0, G^1, \ldots, G^{l-1}, G^l)$ is first generated, so that G^0 is the original graph and G^{k+1} is obtained by grouping nodes of G^k. The coars-

* Corresponding author.
E-mail addresses: seve@dia.uned.es (S.F. Galán), ole.mengshoel@sv.cmu.edu (O.J. Mengshoel).

https://doi.org/10.1016/j.jvlc.2017.11.008
1045-926X/

ening continues until a graph G^l with only a small number of nodes is reached. Since the optimized layout for the coarsest graph can be more easily obtained, then two additional consecutive phases (placement and refinement) are performed iteratively. In the placement phase, the optimized layout of a coarser graph G^{k+1} is transformed to a finer layout G^k. In the refinement phase (or single-level layout), typically a force-directed method is used to improve the layout resulting from the placement phase.

- The computational complexity of each iteration of a force-directed method is usually $\mathcal{O}(M+N^2)$, where M and N are the number of edges and nodes in the graph respectively. This complexity derives from the fact that every node changes its position at each iteration, and this change depends on the attractive and repulsive forces exerted by the rest of the nodes. Whereas the attractive (mechanical) forces take place between neighboring nodes and are calculated in $\mathcal{O}(M)$ time for the whole graph, the repulsive (electrical) forces occur between any pair of nodes and are determined in $\mathcal{O}(N^2)$ time. Since the quadratic complexity of repulsive forces makes force-directed methods impractical for large graphs, these forces can be approximated by grouping sufficiently distant nodes, which reduces the complexity to $\mathcal{O}(N \cdot \log N)$ [4,28,34,48,52]. Alternatively, Fruchterman and Reingold [20] use a grid variant of their force-directed algorithm that ignores the repulsive forces between distant nodes. Although this variant calculates the repulsive forces in $\mathcal{O}(N)$ time for sparse graphs with uniform node distribution over the grid squares, for general graphs this calculation may become more expensive.

This work presents a novel message passing technique for graph layout named *Neighborhood Beautification*, which provides a new perspective on graph layout similar to how message passing has provided new perspectives on areas such as decoding [42] and compressive sensing [13]. The underlying idea of this novel technique is that a globally aesthetic layout of a graph can be achieved if every set formed by a node and its neighbors is independently laid out in an aesthetic way. An iteration of the neighborhood beautification method consists of three message passing phases that address the following aesthetic criteria in turn: minimize edge crossings, maximize edge length uniformity, and maximize angular resolution. In these three iterated phases, messages are passed from every node in the graph to each of its neighbors. A message contains information about the desired position for the receiving node from the viewpoint of the sending node under one of the three aesthetic criteria. Neighborhood Beautification can be applied as a single-level graph layout method, similar to force-directed algorithms, or as a graph layout method for the refinement phase of the multi-level technique.

The usefulness of a graph layout method is usually established by taking into account both its computational cost (in terms of time and memory) and the quality of the resulting layouts. The Neighborhood Beautification method takes $\mathcal{O}(M+N \cdot \Delta \cdot \log \Delta)$ time for each iteration in the worst case[1] (see Section 3.4), since only interactions between neighboring nodes are considered; therefore, the high complexity associated with the calculation of repulsive forces in force-directed algorithms is reduced in many cases. In the experiments of this paper, we find that although the Neighborhood Beautification method in general needs more iterations than force-directed algorithms, it usually offers shorter computation time. As far as the layout quality is concerned, Neighborhood Beautification provides promising results for grid-like graphs compared to force-directed algorithms, although for complex graphs the quality of layouts produced by force-directed algorithms is superior in general. In this work, we extensively evaluate the performance of the Neighborhood Beautification method in terms of running time and layout quality.

Since Neighborhood Beautification relies on message passing between neighbors, a distributed implementation [2,25] could be constructed that is enabled by the graph structure. Yet, in this work we do not implement a distributed version of Neighborhood Beautification, but leave it to future work. Further, Neighborhood Beautification is implemented in this work as a sequential algorithm. As mentioned in Section 5, an interesting future research direction is its implementation in the context of parallel computing [33,45,50].

The rest of this paper is organized as follows. Section 2 reviews several widely used methods for single-level and multi-level layout of graphs. Section 3 explains our Neighborhood Beautification method in detail. Section 4 describes a series of experiments for evaluating the Neighborhood Beautification method. Finally, Section 5 summarizes the main results of our work and enumerates future research directions.

2. Related work

We consider graphs whose nodes have unconstrained positions and whose edges are straight lines. Existing algorithms for aesthetically laying out such graphs can broadly be classified as single-level or multi-level. Single-level methods exclusively operate on the original graph to be laid out, whereas multi-level methods use additional auxiliary graphs obtained from transforming the original graph. Single-level graph layout can be carried out in several ways: force-directed [11,14,20,37], incrementally [10,51], or algebraically [8,32,39,40]. Although the algebraic methods are significantly faster than the force-directed methods, they frequently produce graph layouts of inferior quality compared to graph layouts obtained using force-directed methods [27, Section 1.3.3].

Due to their importance and widespread use, this section reviews two force-directed approaches: (1) the spring-embedder model, proposed by Eades [14] and later modified by Fruchterman and Reingold [20], and (2) Kamada and Kawai's model [37]. We also review the multi-level technique for graph layout [30,31,55], an approach successfully used for large graphs, and where a force-directed method is used as a subroutine.

2.1. Force-directed graph layout

To tackle the graph layout problem, force-directed algorithms are inspired by a physical analogy. These algorithms consider forces acting on the nodes and iteratively transform the graph layout from its initial configuration (usually with random positions for the nodes) until a configuration is reached where the net force acting on each node is null. Equivalently, they associate a potential energy with a layout and aim to find a layout whose energy is locally optimal. Kobourov has written an excellent and detailed survey of different force-directed algorithms for graph layout [38].

2.1.1. The spring embedder model

The spring embedder model was defined by Eades [14] as a system of electrically charged rings connected by springs. The mechanical forces provoke the attraction between every pair of neighboring nodes, whereas the electrical forces make every pair of non-neighboring nodes repel each other.

The attractive force exerted by a spring is modeled by the following formula:

$$F_{\text{att}} = c_1 \cdot \log\left(\frac{d}{c_2}\right), \tag{1}$$

[1] The symbol Δ denotes the maximum degree of the graph.

where c_1 and c_2 are constants, and d is the length of the spring. The repulsive force acting between two non-neighboring nodes is defined as follows:

$$F_{\text{rep}} = \frac{c_3}{d^2}, \tag{2}$$

where c_3 is a constant and d is the distance between the nodes. Whereas attractive forces mean that neighboring nodes are drawn close to each other, repulsive forces imply that nodes are spread well out in the drawing area [7, Section 4.1].

The spring embedder model iteratively updates the positions of the nodes. At each iteration, every node is moved proportionally to the net force acting on it as indicated by these formulas:

$$\begin{cases} x'(u) = x(u) + c_4 \cdot F_x(u) \\ y'(u) = y(u) + c_4 \cdot F_y(u) \end{cases},$$

where $(x(u), y(u))$ are the Cartesian coordinates of node u, c_4 is a constant, and $F(u)$ is the net force acting on node u resulting from considering both attractive and repulsive forces. The time complexity of the calculation of forces in each iteration of this model is $\mathcal{O}(M + N^2)$. Here, the quadratic term N^2, associated with the calculation of repulsive forces, makes this model most useful for graphs with up to a few hundred nodes.

Fruchterman and Reingold [20] modified the spring embedder model of Eades by introducing new and computationally more efficient formulas for Eqs. (1) and (2), without changing the qualitative character of the force model. Additionally, in order to reduce the $\mathcal{O}(N^2)$ time complexity associated with the calculation of the repulsive forces in the Eades model, they introduced a grid variant of their algorithm. This grid variant ignores repulsive forces between distant nodes by dividing the space into squares, so that each node is only affected by repulsive forces from nodes in its own and neighboring squares. In this way, the repulsive forces for the whole graph are calculated in $\mathcal{O}(N)$ time for sparse graphs with a uniform node distribution over the grid squares. Unfortunately, this calculation may become more expensive when this condition is not met in general graphs.

Other more recent methods of reducing the quadratic complexity associated with the calculation of repulsive forces in the Eades model are inspired by the N-body problem studied in physics. These methods [28,34,48,52] approximate the repulsive forces by applying two main phases: firstly, the drawing space is partitioned by creating a quadtree of rectangles and, secondly, the quadtree is traversed to approximate the net force acting on each node. Nodes inside a distant enough rectangle are grouped together and treated as a supernode located at the center of mass of the grouped nodes. In this way, the repulsive forces in the graph are calculated in $\mathcal{O}(N \cdot \log N)$ time.

Another variant of the spring embedder model is the GEM algorithm [17], which improves the graph layout convergence by detecting rotations and oscillations of the nodes. However, the running time for each iteration of GEM is $\mathcal{O}(M + N^2)$.

2.1.2. Kamada and Kawai's model

The model of Kamada and Kawai [37] is based on the idea that a graph layout is aesthetic when the pairwise geometric distances between the nodes in the layout match the pairwise graph-theoretic distances. In this model, every pair of nodes is connected by a spring that obeys Hooke's law. The natural length of each spring is proportional to the graph-theoretic distance between the two nodes connected by the spring. In this way, the total energy of the system is defined as follows:

$$E = c \cdot \sum_{u,v \in V} \left(\frac{\ell(u,v)}{d_{\min}(u,v)} - L \right)^2, \tag{3}$$

where c is a constant, V is the set of nodes in the graph, $\ell(u, v)$ is the current length of the spring connecting nodes u and v, L is the desired length of a single edge in the layout, and $d_{\min}(u, v)$ is the shortest path distance between nodes u and v. The Newton–Raphson method is used to iteratively minimize the energy E of the graph layout.

Kamada and Kawai's model requires a preprocessing step. It calculates the shortest path distance $d_{\min}(u, v)$ between every pair of nodes in the original graph. The computational complexity of this model is dominated by the preprocessing step, and thus the algorithm has a time complexity of $\mathcal{O}(N \cdot M)$ and a space complexity of $\mathcal{O}(N^2)$. This quickly becomes an obstacle to lay out large graphs.

Stress majorization [24] is a variant of Kamada and Kawai's model. Instead of using the Newton–Raphson method to iteratively minimize the energy function, techniques from the field of multidimensional scaling are employed to improve computation speed. In multidimensional scaling, the energy function defined in Eq. (3) is known as the stress function. Under stress majorization, the stress function is usually defined such that $c = 1$ and $L = 1$ in Eq. (3). Nonetheless, stress majorization needs to carry out the same preprocessing step as Kamada and Kawai's model in order to calculate the all-pairs shortest paths for the nodes in the graph.

The maximal entropy stress model [23] avoids the preprocessing step by only considering springs between neighboring nodes and resolving the remaining degrees of freedom via maximization of the entropy of the layout. The sparse stress model [46] stabilizes the sparse stress function restricted to 1-neighborhoods with aggregated long-range influences inspired by the use of Barnes & Hut approximation [4].

2.2. The multi-level technique for graph layout

The multi-level technique [30,31,55] has become the standard method for graph layout of general large graphs. This technique operates regardless of the initial positions of the nodes and only requires the graph structure. Multi-level graph layout utilizes auxiliary graphs (one for each level) whose nodes represent subsets of nodes in the original graph, although some authors have used node filtrations [21,22] as an alternative way to create the auxiliary graphs.

The multi-level technique for graph layout consists of three phases:

1. **Coarsening:** A sequence of graphs $(G^0, G^1, \ldots, G^{l-1}, G^l)$ is generated, so that G^0 is the original graph and G^{k+1} is obtained by grouping nodes of G^k for $0 \le k \le l-1$. The coarsening continues until a graph G^l with only a small number of nodes is reached. (For example, Walshaw typically coarsens down to two nodes in [55].) In order for this process to be efficient, the coarsening phase is designed so that $l = \mathcal{O}(\log(N))$. Once the optimized layout for the coarsest graph has been obtained, then two additional phases (placement and refinement) are performed iteratively. Specifically, the sequence of placement followed by refinement is repeated l times.
2. **Placement:** The optimized layout of a coarser graph G^{k+1} is transformed to a finer layout G^k. Typically, given a supernode in G^{k+1}, its nodes in G^k are assigned random positions near that of the supernode.
3. **Refinement (or single-level layout):** A force-directed method (see Section 2.1) is often used to improve the layout obtained after the placement phase. Only a few iterations of the force-directed method are usually needed in this phase in order to optimize the layout for the current level.

Algorithm: Neighborhood Beautification (NB)

Input:

G	An undirected graph with initial positions for its nodes
iterations	Number of executions for the three consecutive NB phases
$k_1 \in (0, 1)$	Constant used in the first phase (minimization of edge crossings)
$k_2 \in (0, 1]$	Constant used in the second phase (minimization of edge length differences)
$k_3 \in (0, 1]$	Constant used in the third phase (maximization of angular resolution)

Output:

A graph layout for G

```
1:  begin
2:      repeat iterations times:
3:
4:          ; Phase 1: Minimization of Edge Crossings
5:          MinimizeEdgeCrossings(G, k1)
6:
7:          ; Phase 2: Minimization of Edge Length Differences
8:          MinimizeEdgeLengthDifferences(G, k2)
9:
10:         ; Phase 3: Maximization of Angular Resolution
11:         MaximizeAngularResolution(G, k3)
12:
13:         Conform the current graph layout to the drawing frame (optional)
14:     end-repeat
15:     Conform the current graph layout to the drawing frame
16: end-begin
```

Fig. 1. Pseudocode for the NB algorithm.

3. Neighborhood beautification graph layout by message passing

The present work introduces a novel approach to graph layout named *Neighborhood Beautification* (NB). This approach performs message passing between neighboring nodes in the graph. Every node sends a message to each of its neighbors with a recommended position for the neighbor. Therefore, the messages passed by a sending node u to its neighboring nodes $N(u)$ reflect node u's preferences about local beauty in the graph layout. The set of messages received by a node, along with its current position, determine its next position in the graph layout.

Fig. 1 shows the top-level steps of the NB algorithm. Normally, an initial layout is provided with random positions for the nodes, and this layout is progressively improved after each iteration of NB. In each iteration, three phases are carried out in the following order: (1) *minimization of edge crossings*, (2) *minimization of edge length differences*, and (3) *maximization of angular resolution*. Each of these phases operates by visiting all the nodes in the graph, sending messages from each visited node to its neighbors, and processing a node's incoming messages.

There are two ways of organizing message passing in each NB phase:

1. In synchronous NB, the nodes of the graph are visited in an arbitrary order that is fixed between iterations. Each visited node sends its outgoing messages and, once the last message has been sent in the graph, every node processes its incoming messages. For a node u, its new position is determined by the barycenter of the set formed by u's current position along with the recommended positions of its incoming messages.
2. In asynchronous NB, the nodes of the graph are visited in a random order, which varies from iteration to iteration. Each visited node sends its outgoing messages, which are immediately processed by its neighbors as they arrive: The new position for each neighbor is the recommended position in its incoming message.[2]

In this work, we use synchronous message passing for each of the three NB phases, since we found in experiments that it produces better results. In the rest of the present section, we first discuss each of the three NB phases in turn. Section 3.1 discusses how NB minimizes edge crossings, which helps to unfold and untangle the current layout. Section 3.2 explains how NB keeps edge lengths uniform, which facilitates nodes to be uniformly distributed in the layout. Section 3.3 describes how NB maximizes angular resolution, which improves the global orientation of the layout. Finally, an analysis is included in Section 3.4.

3.1. Phase 1: minimization of edge crossings

The first phase of NB aims to reduce the number of edge crossings in a graph layout by making edges shorter. The reduction of the number of crossings is a side effect of shortening edges, while other approaches [5,6] augment force-directed techniques with explicit (and computationally more costly) calculations and intelligent removal of crossings. The reduction can be achieved locally if every node sends a message to each of its neighbors recommending the following translation in the plane:

$$m_1^{u,v} = \left(x'(v), y'(v)\right)$$

[2] Asynchronous NB is similar to Kamada and Kawai's method (see Section 2.1.2) in that the layout is updated by considering one node at a time. However, there are important differences. In Kamada and Kawai's method, the current node is moved so that the global layout energy is minimized. In asynchronous NB, on the other hand, the randomly chosen current node locally sends messages that move its neighbors according to a specific aesthetic criterion.

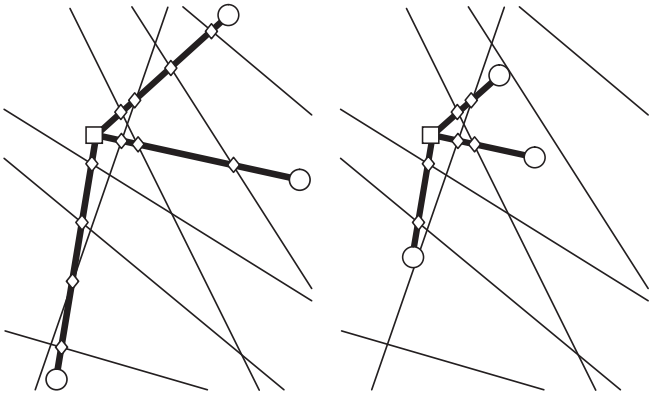

Fig. 2. Graph layout before one of the nodes (square) sends messages to its neighbors (circles) in Phase 1 of NB (left) and the resulting graph layout after these messages have been processed by the neighbors with $k_1 = 0.5$ in Phase 1 of NB (right). Note how the number of edge crossings (rhombuses) has been reduced.

such that

$$\begin{cases} x'(v) = x(v) + k_1 \cdot (x(u) - x(v)) \\ y'(v) = y(v) + k_1 \cdot (y(u) - y(v)) \end{cases},$$

where u is the sending node, v is the receiving node, $m_1^{u,v}$ is the Phase-1 message sent from u to v containing the new recommended coordinates for v from the viewpoint of u, and $k_1 \in (0, 1)$ is a constant. Normally, values of $k_1 \approx 1$ produce edge crossing minimization in a fewer number of NB iterations. Note that, when $k_1 \approx 1$, this phase operates in a similar way to Tutte's barycentric method [53], which places each free node at the barycenter of its neighbors. A difference is that we include the position of the receiving node in the calculation of the barycenter.

The idea behind message $m_1^{u,v}$ is that the neighbor v should approach the sending node u proportionally to a constant k_1. In this way, as a consequence of promoting shorter edges, a decrease is expected in the number of crossings resulting from the intersection of, on the one hand, the edges having the sending node as an end node and, on the other hand, the rest of the edges in the graph layout. For example, consider the two graph layouts in Fig. 2, where the following elements have been depicted: the sending node (as a square), the receiving nodes (as circles), the edges connecting the sending node and the receiving nodes (as highlighted lines), and the intersections of the highlighted edges with the rest of the edges (as rhombuses). The left panel shows the graph layout prior to message passing from the sending node. The right panel illustrates the resulting graph layout after the receiving nodes have processed their incoming messages with $k_1 = 0.5$. Whereas eleven crossings appear in the left graph layout, only six of them remain in the right graph layout.

3.2. Phase 2: minimization of edge length differences

The second phase of NB is designed to make the edge lengths in the graph layout more uniform. Keeping edge lengths approximately uniform is a generally accepted aesthetics for obtaining attractive drawings of undirected graphs [12]. Similar to Phase 1, the smoothing of edge lengths can be done locally if every node sends a message to each of its neighbors recommending a translation in the plane.

We say *desired edge length* to denote the desired length for the edges in the current layout. (The "current layout" is the one resulting from Phase 1.) The desired edge length can be defined in several ways:

1. It can be established locally for each sending node and its neighbors in the graph. For example, it can be calculated as the length to the nearest (or, alternatively, farthest) neighbor. We discarded a local definition of the desired edge length because we found that it produces severe distortions in the graph layout.
2. It can be set up globally in the graph.[3] An option that produces satisfactory layout results (regardless of the particular graph at hand) consists of defining the desired edge length as the longest edge length present in the current layout. In this way, in contrast to Phase 1, the receiving node is separated from the sending node so that the new length between them approaches the desired edge length. This is the option that we use in the experimental part of this work (see Section 4). We experimentally found that other variants of this option, like setting the desired edge length to the shortest or to the mean edge length in the current layout, produce final layouts of poorer quality, mainly in terms of uniformity of edge length.

Let u be a sending node and v a receiving node. A message passed in this phase is defined as follows:

$$m_2^{u,v} = \left(x'(v), y'(v)\right)$$

such that

$$\begin{cases} x'(v) = x(v) + k_2 \cdot (d_{\text{desired}} - d(u,v)) \cdot \frac{x(v)-x(u)}{d(u,v)} \\ y'(v) = y(v) + k_2 \cdot (d_{\text{desired}} - d(u,v)) \cdot \frac{y(v)-y(u)}{d(u,v)} \end{cases},$$

where $m_2^{u,v}$ is the Phase-2 message sent from u to v containing the new recommended coordinates for v from the viewpoint of u, $k_2 \in (0, 1]$ is a constant, d_{desired} is the desired edge length, and $d(u, v)$ is the Euclidean distance from u to v.

When $k_2 = 1$, the goal of the messages passed by a sending node u to its neighbors is to place the neighbors at a distance from the sending node equal to the desired edge length, without changing the directions of the neighbors relative to the sending node. The greater k_2 is, the more uniformity is achieved after this phase. This phase is similar to Kamada and Kawai's spring model (see Section 2.1.2), with the difference that we place springs only between neighboring nodes and all of them have the same natural length equal to d_{desired}. For example, consider the two graph layouts in Fig. 3, where the sending node is depicted as a square and the receiving nodes are depicted as circles. The left panel shows the graph layout prior to message passing from the sending node. The right panel shows the graph layout resulting from those incoming messages being processed by the receiving nodes for $k_2 = 1$ and for a desired edge length globally set to the longest edge length in the current graph layout. Fig. 3 shows that uniform edge lengths are achieved in the neighborhood of the sending node.

It is important to note that, in Phases 1 and 2 of NB, the two messages sent in opposite directions through each edge contain symmetric values and, therefore, just one of the two messages needs to be calculated. In other words, if node v is translated (δ_x, δ_y) under message $m_i^{u,v}$ then u would be translated $(-\delta_x, -\delta_y)$ under message $m_i^{v,u}$ for $i \in \{1, 2\}$.

3.3. Phase 3: maximization of angular resolution

Let u be a node whose neighbors are denoted by $N(u)$ and whose degree is denoted by $\deg(u) = |N(u)|$. Phase 3 of NB aims

[3] An efficient option is to allow the user to enter the value of the desired edge length, which will remain unchanged throughout the execution of the NB algorithm. This option has the disadvantage that the final layout quality will depend on the specific desired edge length employed, and the user will need to manually tune this parameter for each particular graph.

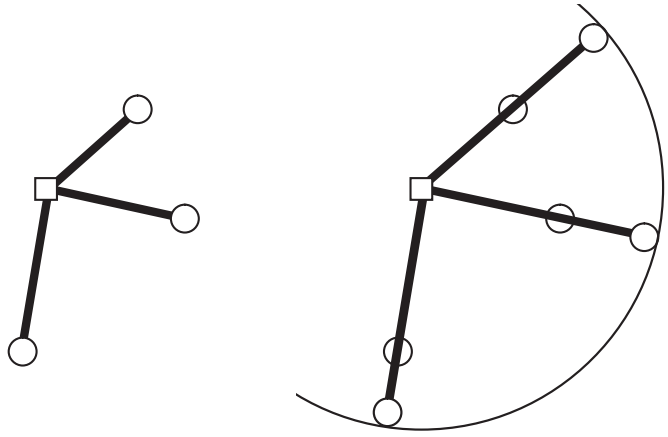

Fig. 3. Graph layout before one of the nodes (square) sends messages to its neighbors (circles) in Phase 2 of NB (left) and the resulting graph layout after these messages have been processed by the neighbors with $k_2 = 1$ in Phase 2 of NB (right).

to place $N(u)$ such that the angle between the edges from u to any pair of consecutive neighbors is equal to $360^o/\deg(u)$. This aesthetics criterion has received several names in the literature, for instance, maximizing the minimum angle between edges leaving a node [47] or maximizing the uniformity of the angles around each node [54]. Several force-directed graph layout methods implement this criterion [9,15,36,41]. Eades et al. [15] construct graph drawings with large crossing angles between non-adjacent edges. (Note that NB only operates with angles between adjacent edges.) Lin and Yen [41] define a repulsive force between adjacent edges, but it has the disadvantage of taking as input a reasonable graph layout computed by another force-directed method. The work of Chernobelskiy et al. [9] is applied in the context of Lombardi-style graph drawings. In a Lombardi drawing of a graph, nodes are represented as points, edges are represented as circular arcs between their endpoints, and every node has perfect angular resolution. Finally, Huang et al. [36] create a sine force in order to increase the angular resolution of nodes. The sine force makes the angle between two neighboring edges approach the optimal angle $360^o/\deg(u)$.

When $\deg(u) < 2$, no message is sent from node u. Intuitively, this is because the angular resolution is not adjustable in this case. Given a sending node u, with $\deg(u) \geq 2$, and an ordering of its neighbors $\left(v_1, v_2, \ldots, v_{\deg(u)-1}, v_{\deg(u)}\right)$ by angle with respect to u, the messages passed in this phase are defined as follows:

$$m_3^{u,v_i} = \left(x'(v_i), y'(v_i)\right)$$

such that

$$\begin{cases} \left(x'(v_i), y'(v_i)\right) = \texttt{rotate}\left(v_i, k_3 \cdot \left(\texttt{angle}(v_i, v_{i+1}) - \frac{360^o}{\deg(u)}\right)\right) & \text{if } \texttt{angle}(v_i, v_{i+1}) > \frac{360^o}{\deg(u)} \\ \left(x'(v_i), y'(v_i)\right) = (x(v_i), y(v_i)) & \text{otherwise} \end{cases}$$

Here, m_3^{u,v_i} is the Phase-3 message sent from u to v_i (with $i \in \{1, \ldots, \deg(u)\}$) containing the new recommended coordinates for v_i from the viewpoint of u, $\texttt{angle}(v_i, v_{i+1})$ is the angle between edges $\langle u, v_i\rangle$ and $\langle u, v_{i+1}\rangle$, $\texttt{rotate}(v_i, \alpha)$ computes the new coordinates for v_i after being rotated an angle α around u, and $k_3 \in (0, 1]$ is a constant. Note that moving a node means reducing its angle with the next node; since the goal is to progressively make angles more uniform to the value $360^o/\deg(u)$, the only way to achieve that goal is to only move nodes with angle to the next node greater than $360^o/\deg(u)$. If rotating a node v_i towards v_{i+1} causes the angle between v_{i-1} and v_i to become greater than $360^o/\deg(u)$, the node v_{i-1} will be rotated towards v_i in the next iteration of NB.

Messages in Phase 3 operate under the following conditions:

- Any coordinate system is valid for this phase. For example, 0^o could represent North and angles could increase clockwise, or 0^o could represent East and angles could increase counterclockwise.
- The ordering of the neighbors v_i, defined earlier in this section, can be done either by increasing angle or by decreasing angle. If angles are assumed to increase clockwise, this gives rise respectively to two types of rotations: clockwise and counterclockwise. We experimentally found that Phase 3 is more effective if the type of rotation is selected uniformly at random, over "clockwise" and "counterclockwise", every time a sending node u executes this phase.
- Message m_3^{u,v_i} is sent before message $m_3^{u,v_{i+1}}$.
- Message m_3^{u,v_i} with $i = \deg(u)$ uses $v_{i+1} = v_1$. Note that v_1 has already been rotated under message m_3^{u,v_1}.

Although the goal of the messages passed by a sending node to its neighbors is to make angles (with respect to the sending node) between the neighboring nodes more uniform, in general this goal can only be achieved after several iterations of NB. One iteration of Phase 3 increases the angle uniformity of the neighboring nodes but, even if $k_3 = 1$, it is unlikely that complete uniformity is obtained after just one iteration. For example, consider the two graph layouts in Fig. 4, where the sending node is depicted as a square and the receiving nodes are depicted as circles. The left graph layout constitutes the state prior to message passing from the sending node. The messages are sent in the order (v_1, v_2, v_3). The right graph layout depicts the resulting state after all of the incoming messages have been processed by the receiving nodes for $k_3 = 1$. Fig. 4 reflects that just one of the receiving nodes, v_2, was rotated. The rest of the neighboring nodes might be rotated in subsequent iterations of NB, however.

Once message passing and processing has taken place in all of the nodes in Phase 3, and before the next NB iteration is started in Phase 1, an optional scaling of the resulting graph layout can be carried out so that it conforms to the drawing frame. If this scaling is not applied, the graph layout will typically become smaller after each iteration; in such a case, it will be necessary to apply the scaling after the final NB iteration in order to obtain a proper visualization of the final graph layout.

3.4. Discussion and analysis

In the three NB phases, the idea behind the messages sent by a node u to its neighbors is that these messages contain information about suggested new locations representing appealing locations for the neighbors from the viewpoint of u. Therefore, NB promotes a local concept of good layout (abbreviated as "beauty") by making node u interact only with its immediate neighborhood. Thus, in Fig. 2 only the crossings caused by u's edges are for simplicity considered, and the rest of incident edges to u's neighbors are ignored. After the translations depicted in Fig. 2, it is possible that u's neighbors connected to other nodes increase the number of crossings for their incident edges. However, as the experimental evaluation of this work has demonstrated, in general the global effect after an iteration of Phase 1 is a reduction of the total number of crossings in the layout.

Phase 2 stretches edges as shown in Fig. 3, which could reintroduce crossings in the layout. Nonetheless, the goal of Phase 1 is precisely to keep that from happening in Phase 2 by making k_1 as close to unity as possible.

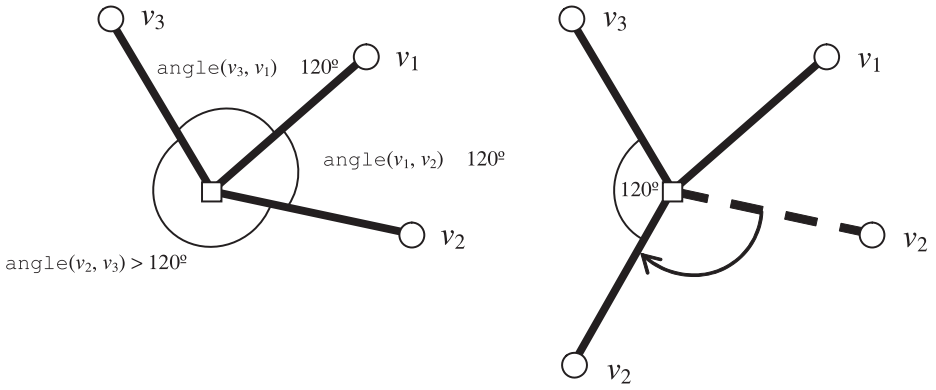

Fig. 4. Graph layout before one of the nodes (square) sends messages to its neighbors (circles) in Phase 3 of NB (left) and the resulting graph layout after these messages have been processed by the neighbors with $k_3 = 1$ and order (v_1, v_2, v_3) in Phase 3 of NB (right).

Although Phase 1 shortens edges and Phase 2 lengthens them, they are not necessarily opposing phases but instead typically have complementary effects. Phase 1 uniformly reduces edge lengths throughout the layout, whereas Phase 2 extends edge lengths depending on how similar they are to the largest edge length in the layout. In this way, edges whose lengths are close to that of the longest edge are greatly shortened in Phase 1 while left almost unchanged in Phase 2. This is confirmed by the fact that the best results in Section 4 are obtained when $k_1 \approx 1$ for most of the graphs, while k_2 is much more graph dependent and adopts values throughout the whole real interval $(0, 1]$.

If Phase 2 is considered in isolation, it could be argued that modifying the target edge length would only mean a global scaling of the current layout. In other words, if a layout with target length equal to unity is scaled by a factor k, then this would be the same as applying a target length equal to k. However, even though scaling by k could have that effect locally if just one node and its neighbors are considered, the result is quite different globally after all the messages have been sent, as explained in Section 3.2 (point 2). Note that when nodes receive more than one message, setting up a long target length can even "destroy" the current layout. In general, specifying a desired edge length in Phase 2 has an influence on the final layout quality.

The reason why only too large angles are decreased but too small angles are not enlarged in Phase 3 (see Fig. 4) is that the ordering of edges by angle needs to be preserved in this phase.[4] When an edge is rotated, that can only be guaranteed if angles between consecutive edges are decreased and not enlarged when approaching the ideal angle $360^o/\deg(u)$, where u denotes the current node.

The computational complexity of an iteration of the first and second NB phases is $\mathcal{O}(M)$, since simple operations are carried out for each pair of neighboring nodes. However, the third NB phase contributes with additional complexity derived from the ordering of angles. In this phase, each node u orders its neighbors by angle, which takes $\mathcal{O}(\deg(u) \cdot \log \deg(u))$. In the worst case of a regular graph where all degrees are the same, the overall complexity of angle ordering in the third NB phase is $\mathcal{O}(N \cdot \Delta \cdot \log \Delta)$, where Δ denotes the maximum degree of the graph. Thus, the computational complexity for an iteration of NB is $\mathcal{O}(M + N \cdot \Delta \cdot \log \Delta)$ in the worst case. In comparison to the complexity for an iteration of the best force-directed algorithms, which is $\mathcal{O}(M + N \cdot \log N)$ as shown in Section 2.1, note that a reduction is obtained by NB when large graphs with low Δ are considered; nonetheless, as the graph gets more connected and $\Delta \approx N$, the complexity of NB becomes higher in the worst case. Interestingly, the angle orderings are typically more and more similar as NB progresses. This can be used to improve the efficiency of the ordering process. In this way, like in Phases 1 and 2, the complexity of angle ordering becomes $\mathcal{O}(M)$ as the execution of NB progresses.

[4] Since Phase 3 acts locally in a neighborhood, it is impossible to determine the optimum edge ordering for the neighborhood with respect to the global graph. Therefore, in order to avoid unexpected disruptive changes in the layout, Phase 3 preserves the current edge ordering when maximizing angular resolution.

4. Experimental evaluation

There are two main aspects to be considered when the performance of a graph layout algorithm is evaluated: (1) the quality of the layouts obtained by the algorithm and (2) the computational resources needed to compute layouts. We explore these two aspects in the following two sections. The experiments were carried out on an Intel Core i5 processor (2.67 GHz) with 8Gb of memory and running Windows 7.

4.1. Graph layout quality

We experimentally evaluate the quality of the graph layouts computed by NB by applying it, first, as a single-level method for graph layout and, second, as the method used in the refinement phase of the multi-level approach. As a single-level method, we execute a number of NB iterations on a set of graphs of small or medium size which are initially assigned random positions for their nodes. In the context of the refinement stage of the multi-level approach, NB is executed for a given number of iterations on a set of large graphs in which only their structure (nodes and edges) is required and the initial positions of the nodes are not needed. We implemented the NB algorithm within NetLogo [56], an agent-based modeling and programming environment, due to the fact that it is particularly well suited for modeling and inspecting complex systems developing over time.

Figs. 5 and 6 contain twelve graphs (widely used in the graph layout literature) to which NB has been applied as a single-level method. Graphs 5 a and b are two types of grids, which are successfully unfolded by means of NB. For graph 5c, most of the previous graph layout methods produce a non-planar layout (see [20]); however, NB (like [11]) is able to generate a layout that is free of edge crossings. Graph 5 d represents a view from a regular dodecahedron with twelve pentagonal faces. Graphs 5 e and f, which appear in [11] and [41] respectively, constitute two layouts for trees. Finally, the six graphs in Fig. 6 are selected because they have a higher number of nodes than the six graphs in Fig. 5. The NB parameters used for obtaining the graph layouts depicted in Figs. 5 and 6 are summarized in Table 1. The parameter values for k_1, k_2, and k_3 were selected via manual fine-tuning, such that those values efficiently leading to a good layout are preferred. NB was executed for the minimum number of iterations producing

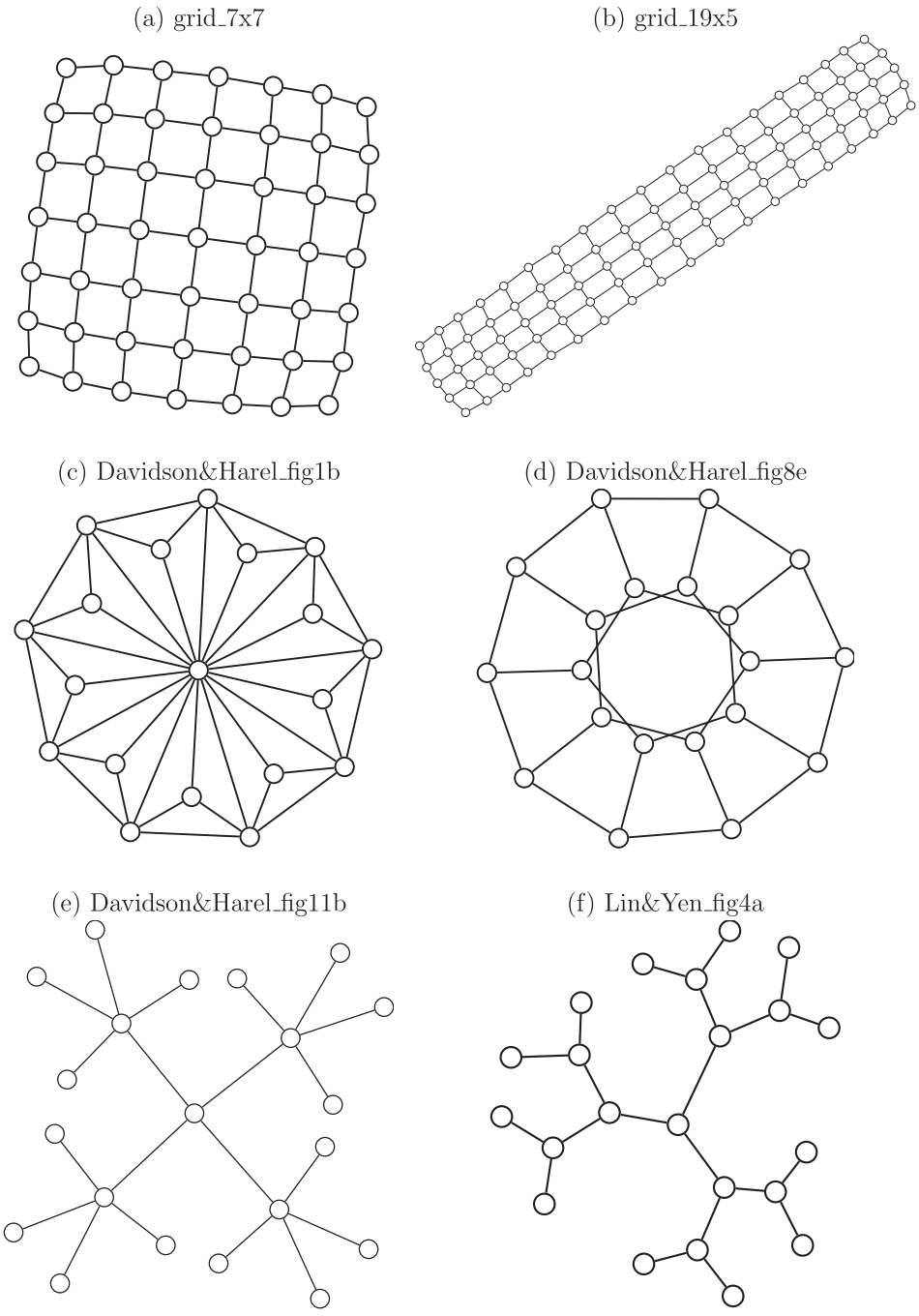

Fig. 5. Layouts for six simple graphs computed by NB used as a **single-level** method. The graphs are initially assigned random positions for their nodes. Additional information about the graphs and NB parameters is in Table 1.

Table 1
NB parameters used for the graph layouts depicted in Figs. 5 and 6.

Graph	Name	Nodes	Edges	k_1	k_2	k_3	NB iterations
5a	grid_7x7	49	84	0.999999	1	0.1	200
5b	grid_19x5	95	166	0.999999	1	0.1	400
5c	Davidson&Harel_fig1b	19	45	0.5	0.5	0.1	200
5d	Davidson&Harel_fig8e	20	30	0.999999	1	0.1	50
5e	Davidson&Harel_fig11b	21	20	0.999999	0.8	0.4	200
5f	Lin&Yen_fig4a	22	21	0.5	0.5	1	150
6a	grid_20x20_singlefolded	399	760	0.999999	0.2	0.1	1000
6b	grid_20x20_doublefolded	397	760	0.999999	0.1	0.1	1000
6c	cylinder_rnd_10x10	97	178	0.999999	0.2	0.2	100
6d	cylinder_rnd_32x32	985	1866	0.999999	0.2	0.2	1000
6e	Sierpinski_4	123	243	0.999999	0.5	0.05	2000
6f	flower_1	210	3057	0.999999	0.5	0.05	16,000

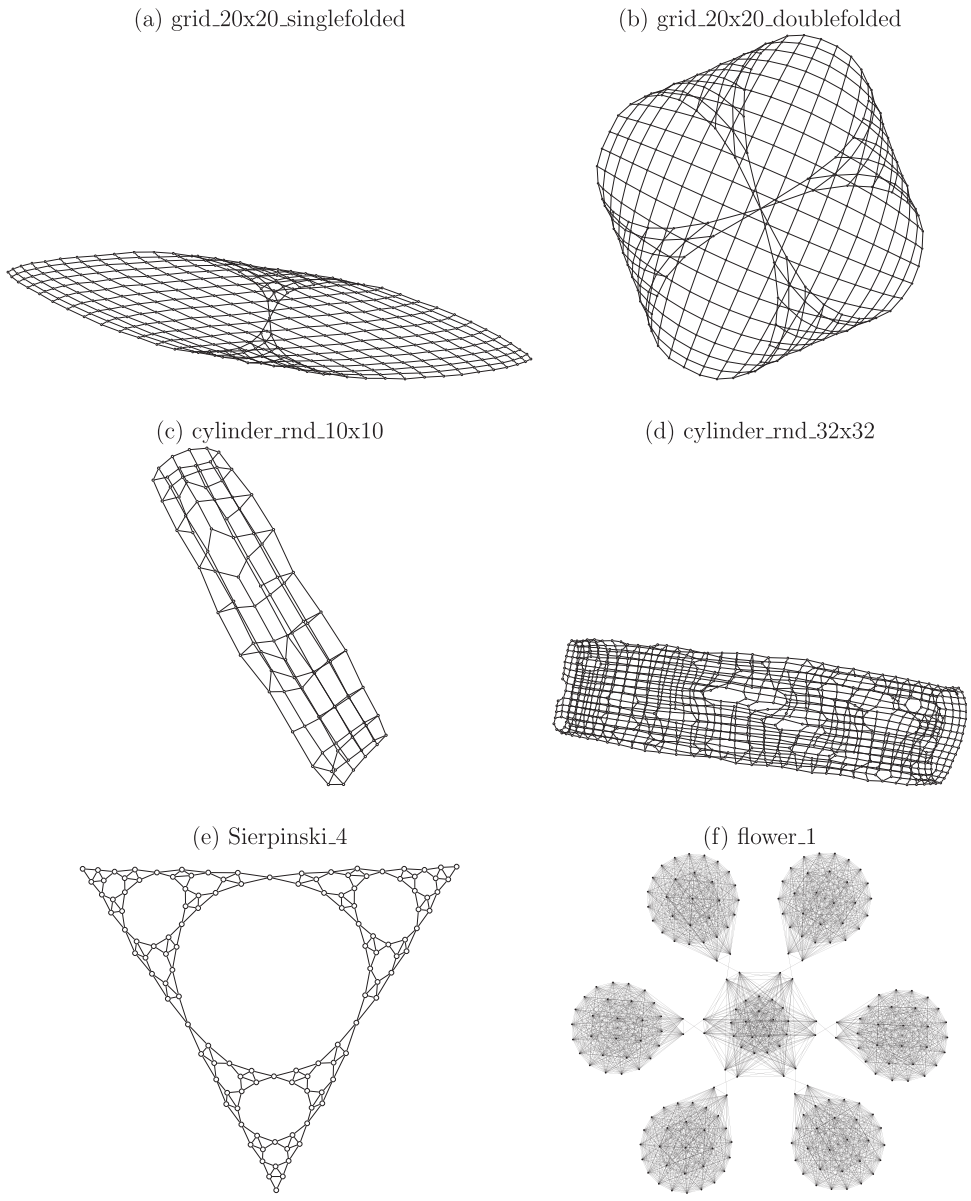

Fig. 6. Layouts for six more complex graphs computed by NB used as a **single-level** method. The graphs are initially assigned random positions for their nodes. Additional information about the graphs and NB parameters is in Table 1.

good layout results in each case. ("Good" was visually determined by comparing the current graph layout with the layout obtained at convergence, after a high number of iterations.) The direction of rotation (clockwise or counterclockwise) was randomly selected as explained in Section 3.3. The optional scaling of the layout after each iteration was carried out for the twelve graphs.

Figs. 7 and 8 contain twelve large graph layouts obtained by NB using the multi-level approach. We use a simple coarsening method in which every node is grouped with one randomly chosen neighbor so that clusters for the next level are formed by at most two nodes of the current level. Regarding the placement method, we employ the typical technique that assigns each node a random position near its supernode. While Fig. 7 contains six grid-like layouts, Fig. 8 depicts six layouts with complex structures. The corresponding graphs are available from the University of Florida Sparse Matrix Collection.[5] The layouts shown in Figs. 7 and 8 were obtained by using the NB parameters shown in Table 2. The parameter values for k_1, k_2, and k_3 were selected via manual fine-tuning, such that those values efficiently leading to a good layout are preferred. The refinement stage of the multi-level approach was executed for the minimum number of NB iterations producing good layout results in each case. The direction of rotation (clockwise or counterclockwise) was randomly selected as explained in Section 3.3. The optional scaling of the layout resulting after each iteration was carried out for the twelve graphs. Compared to other layouts obtained for the same graphs by state-of-the-art methods, the grid-like layouts in Fig. 7 are of very good quality, while the complex layouts in Fig. 8 are slightly less aesthetic.

4.2. Running time evaluation

Due to efficiency reasons and in order to compare the running times of NB with those of other state-of-the-art layout methods,

[5] http://www.cise.ufl.edu/research/sparse/matrices

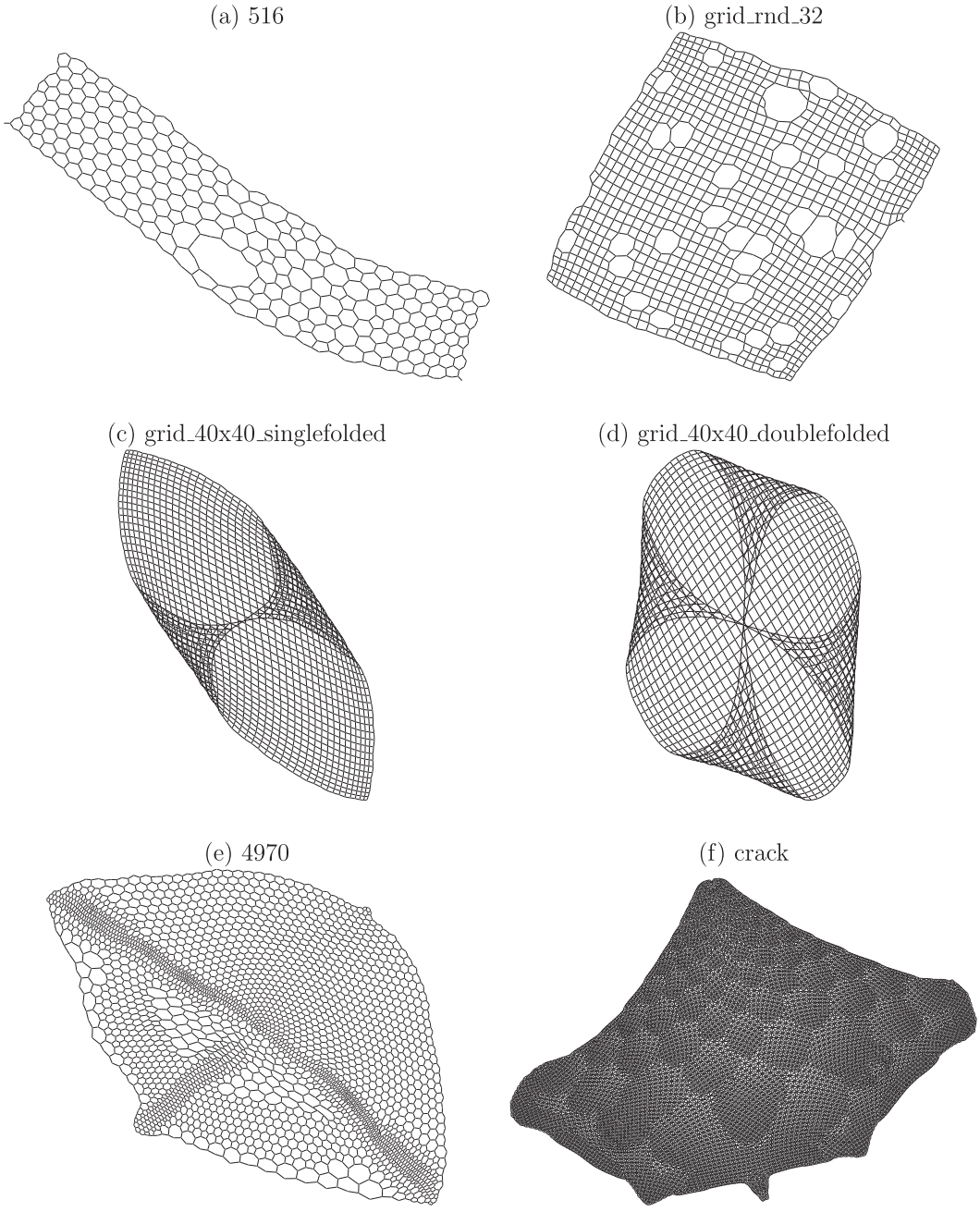

Fig. 7. Layouts for six grid-like large graphs computed by NB using the **multi-level** approach. Additional information about the graphs and NB parameters is in Table 2.

Table 2
NB parameters used for the graph layouts depicted in Figs. 7 and 8.

Graph	Name	Nodes	Edges	Levels	k_1	k_2	k_3	NB iterations for each refinement stage
7a	516	516	729	10	0.999999	0.5	0.5	70
7b	grid_rnd_32	985	1834	12	0.999999	0.25	0.5	50
7c	grid_40x40_singlefolded	1599	3120	12	0.999999	0.2	0.2	50
7d	grid_40x40_doublefolded	1597	3120	12	0.999999	0.2	0.2	100
7e	4970	4970	7400	14	0.999999	0.1	0.75	30
7f	crack	10,240	30,380	18	0.999999	0.5	0.05	50
8a	Sierpinski_6	1095	2187	12	0.999999	0.25	0.025	30
8b	Sierpinski_8	9843	19,683	16	0.999999	0.05	0.4	100
8c	data	2851	15,093	13	0.999999	0.2	0.4	100
8d	add32	4960	9462	25	0.999999	0.03	0.5	5
8e	4elt	15,606	45,878	19	0.999999	0.01	0.8	10
8f	cti	16,840	48,232	14	0.999999	0.1	0.01	100

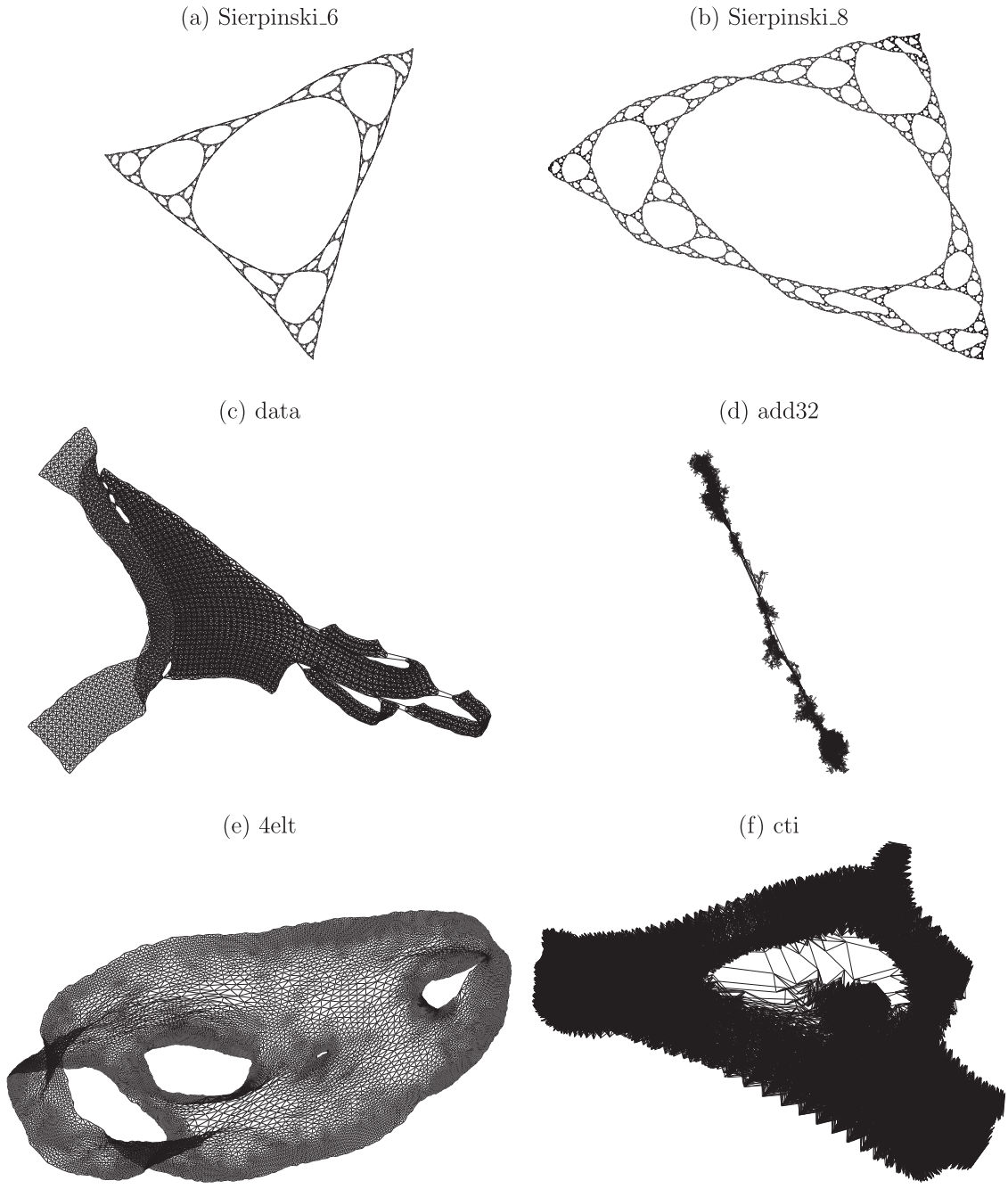

Fig. 8. Layouts for six complex large graphs computed by NB under the **multi-level** approach. Additional information about the graphs and NB parameters is in Table 2.

we implemented NB in OGDF,[6] a publicly available C++ class library for the automatic layout of graphs. The following layout algorithms from the literature, as implemented in C++ in OGDF, were used in comparative experiments:

1. **FR**: The original algorithm of Fruchterman and Reingold [20] that calculates exact repulsive forces (see Section 2.1.1).
2. **FRg**: The grid-variant algorithm of Fruchterman and Reingold [20] that uses approximate repulsive forces (see Section 2.1.1).
3. **GEM**: The algorithm of Frick, Ludwig, and Mehldau [17] that detects rotations and oscillations of the nodes (see Section 2.1.1).
4. **FM^{3}s**: The fast multipole multi-level method of Hachul and Jünger [28], used as a single-level method, that approximates the repulsive forces through techniques inspired by the *N*-body problem (see Section 2.1.1).
5. **KK**: The algorithm of Kamada and Kawai [37] that is based on calculating the pairwise geometric distances between nodes (see Section 2.1.2).
6. **SM**: The algorithm of Gansner, Koren, and North [24] that is based on the stress majorization technique and constitutes a variant of the algorithm by Kamada and Kawai (see Section 2.1.2).

We considered fourteen graphs (see Table 3) with different characteristics as described in [27]. Graphs #1 through #3 and graphs #6 through #11 are instances of several types of grid-like

[6] http://www.ogdf.net

Table 3
Running times in seconds (rounded to two decimal places) for NB, used as a single-level algorithm, and six other layout algorithms when applied to fourteen graphs with different characteristics. The best running time for each graph is highlighted in bold. The graphs are ordered by decreasing number of nodes.

Graph	Name	Nodes	Edges	NB	FR	FRg	GEM	FM^{3}s	KK	SM
#1	grid_40x40	1600	3120	269.99	1250.87	**195.03**	1039.87	709.54	367.34	573.09
#2	grid_40x40_singlefolded	1599	3120	260.21	1177.59	**199.18**	749.19	705.35	664.56	674.20
#3	grid_40x40_doublefolded	1597	3120	326.12	1327.42	**223.70**	800.93	723.38	811.50	402.55
#4	ug_380	1104	3231	**3.65**	4.42	12.16	9.76	14.12	35.94	7.86
#5	Sierpinski_6	1095	2187	351.07	487.76	**220.77**	404.11	748.99	349.16	234.50
#6	cylinder_rnd_32x32	985	1866	**37.77**	344.25	82.25	299.71	276.08	103.39	92.72
#7	grid_rnd_32	985	1834	126.94	306.08	144.92	272.58	259.54	**123.07**	167.18
#8	516	516	729	**37.19**	95.79	62.04	81.55	106.28	123.13	69.21
#9	grid_20x20	400	760	12.88	49.35	21.14	38.65	27.39	13.42	**11.99**
#10	grid_20x20_singlefolded	399	760	**12.17**	39.22	22.74	31.57	28.20	15.77	14.90
#11	grid_20x20_doublefolded	397	760	**10.52**	46.52	25.30	34.14	32.84	23.08	13.22
#12	flower_1	210	3057	48.09	**0.18**	16.51	2.36	4.68	2.95	2.17
#13	Sierpinski_4	123	243	1.13	5.64	5.62	1.42	4.60	1.13	**1.06**
#14	spider_A	100	160	0.65	**0.31**	12.89	1.17	3.07	0.99	2.32

graphs. Graph #4 has a high maximum degree whose value is 856. Graph #12 is formed by seven complete subgraphs with 30 nodes. Graphs #5 and #13 are Sierpinski graphs. Finally, graph #14 is a spider graph.

In order to measure the running times reported in Table 3, we generated initial layouts for each graph by assigning random positions to its nodes in the drawing frame. The generation of random initial positions for the nodes was carried out with different seeds for each new generated graph layout.

Every layout algorithm uses a number of parameters that need to be set up before execution, so that an aesthetic layout is computed in a short time. For each layout algorithm, we initially employed the default parameters included in OGDF. When these parameters were insufficient to produce an aesthetic layout, we manually tuned the parameters in order to improve the quality of the layout. Since this manual tuning process is a time-consuming task, we only generated five drawings for each entry in Table 3, where the mean running times for them are reported.[7]

The results in Table 3 demonstrate the consistently strong performance achieved by NB, which is always among the two best methods in terms of running time, except in the cases of flower_1 and Sierpinski_6. While FRg achieves good running-time performance, it produces layouts of poor quality due to the fact that it uses an inaccurate approximation of the repulsive forces. Fig. 9 shows four examples of the FRg layouts obtained in our experiments, whose quality is inferior to that of the corresponding four layouts for NB illustrated in Fig. 6a, b, e, and f. The quality of the layouts obtained through NB is much better than that produced by FRg, and is comparable to that of the rest of the methods included in Table 3.

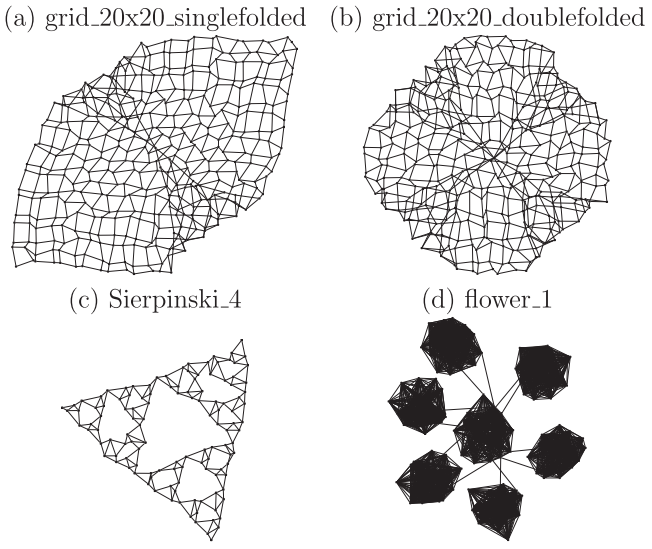

Fig. 9. Layouts computed by FRg for four graphs.

4.3. Quantitative evaluation

In this section, the evaluation carried out in the preceding two sections is extended to larger graphs. Instead of visual inspection, we carry out a quantitative evaluation of graph layout quality by using three measures: *relative edge-crossing number, normalized standard deviation of the edge length*, and *angular resolution*. The relative edge-crossing number [29], denoted as ρ_1, represents the mean number of crossings per edge. The normalized standard deviation of the edge length [29], denoted as ρ_2, is defined as follows:

$$\rho_2 = \sqrt{\sum_{e \in E} \frac{(l_\Gamma(e) - l_\Gamma^{av})^2}{M \cdot (l_\Gamma^{av})^2}},$$

where E is the set of M edges, $l_\Gamma(e)$ is the length of edge e in layout Γ, and l_Γ^{av} is the mean edge length in layout Γ. Finally, the angular resolution [16,36], denoted as ρ_3, is the average over the nodes of the difference between the smallest angle and the optimal angle ($360^o/\deg(u)$ at node u). Note that the smaller the values of the three measures, the higher the quality of the graph layout.

We compare NB used as a multi-level algorithm with FM^3 (an outstanding state-of-the-art multi-level algorithm which was used as single level in Section 4.2) and PivotMDS [9] (an algebraic algorithm). The default parameter values included in OGDF were employed for FM^3 and PivotMDS. In the case of NB, we employed the following parameter values: $k_1 = 0.999999$, $k_2 = 0.03$, $k_3 = 0.5$, and 200 iterations per level, since in Section 4.1 these k_i values were appropriate for complex graphs like add32 (see Table 2).

We consider graphs of the following types in this section:

- Complex graphs generated with the Barabási–Albert model [3], which expands graphs through the addition of new nodes such that each new node is linked according to a probabilistic rule named preferential attachment. The random scale-free graphs

[7] The high number of parameters that were used in the set of experiments reported in Table 3 can be calculated as follows: 7 algorithms × around 5 parameters per algorithm × 14 graphs × 5 random initializations per graph = around 2450 parameters.

Table 4
Experimental results for the relative edge-crossing number (ρ_1). The best value of ρ_1 for each graph is highlighted in bold.

Name	Nodes	Edges	ρ_1^{PivotMDS}	$\rho_1^{\text{FM}^3}$	ρ_1^{NB}
BarabasiAlbert1000	1000	999	0.730	**0.072**	3.498
BarabasiAlbert5000	5000	4999	0.723	**0.217**	12.282
BarabasiAlbert10000	10000	9999	1.585	**0.308**	29.225
BarabasiAlbert15000	15000	14999	1.089	**0.391**	34.325
BarabasiAlbert20000	20000	19999	1.054	**0.473**	52.118
BarabasiAlbert25000	25000	24999	1.878	**0.506**	82.331
BarabasiAlbert30000	30000	29999	1.107	**0.559**	79.693
BarabasiAlbert35000	35000	34999	1.574	**0.593**	74.795
BarabasiAlbert40000	40000	39999	1.400	**0.749**	83.237
BarabasiAlbert45000	45000	44999	1.190	**1.011**	108.495
BarabasiAlbert50000	50000	49999	1.464	**1.177**	139.410
ego-Facebook	4039	88234	**607.772**	912.846	792.546
email-Enron	36692	183831	–	2289.258	**2266.903**
loc-Brightkite	58228	214078	–	**2157.611**	2242.809
p2p-Gnutella04	10876	39994	1719.878	**1485.615**	3873.022
p2p-Gnutella05	8842	31837	**1220.879**	1227.381	1440.326
p2p-Gnutella06	8717	31525	1370.663	**1131.227**	2162.801
fe_sphere	16386	49152	**1.057**	1.120	1.059
G57	5000	10000	1.335	0.803	**0.732**
grid2	3296	6432	0.210	0.001	**0.000**
netz4504	1961	2578	0.111	0.025	**0.000**
rajat03	7602	12551	1.106	**0.594**	0.819
rbd3200l	3200	7840	**0.389**	0.415	0.400
saylr4	3564	9376	2.327	2.345	**2.303**

Table 5
Experimental results for the normalized standard deviation of the edge length (ρ_2). The best value of ρ_2 for each graph is highlighted in bold.

Name	Nodes	Edges	ρ_2^{PivotMDS}	$\rho_2^{\text{FM}^3}$	ρ_2^{NB}
BarabasiAlbert1000	1000	999	**0.279**	0.400	0.999
BarabasiAlbert5000	5000	4999	**0.482**	0.566	0.662
BarabasiAlbert10000	10000	9999	**0.458**	0.608	0.518
BarabasiAlbert15000	15000	14999	**0.398**	0.603	0.456
BarabasiAlbert20000	20000	19999	**0.454**	0.682	0.466
BarabasiAlbert25000	25000	24999	1.062	0.675	**0.458**
BarabasiAlbert30000	30000	29999	0.528	0.696	**0.453**
BarabasiAlbert35000	35000	34999	0.454	0.774	**0.447**
BarabasiAlbert40000	40000	39999	0.451	0.716	**0.418**
BarabasiAlbert45000	45000	44999	**0.155**	0.790	0.425
BarabasiAlbert50000	50000	49999	**0.142**	0.891	0.466
ego-Facebook	4039	88234	2.352	0.700	**0.661**
email-Enron	36692	183831	–	**0.577**	3.164
loc-Brightkite	58228	214078	–	**0.513**	3.138
p2p-Gnutella04	10876	39994	0.540	**0.389**	0.556
p2p-Gnutella05	8842	31837	0.580	**0.383**	0.598
p2p-Gnutella06	8717	31525	0.542	**0.387**	0.502
fe_sphere	16386	49152	0.246	**0.196**	0.244
G57	5000	10000	0.937	0.443	**0.434**
grid2	3296	6432	0.350	**0.236**	0.385
netz4504	1961	2578	0.360	**0.191**	0.392
rajat03	7602	12551	0.678	**0.607**	0.760
rbd3200l	3200	7840	0.490	**0.330**	0.497
saylr4	3564	9376	0.570	**0.373**	0.774

generated by this model are similar to social networks or citation networks, for example. We denote as BarabasiAlbert*N* a graph with *N* nodes generated with the Barabási-Albert model.
- Complex real-world graphs taken from the Stanford Large Network Dataset Collection.[8] This is a collection of large networks from domains such as social networks, computer networks, or communication networks, among others.
- Grid-like large graphs taken from the University of Florida Sparse Matrix Collection. We have selected graphs from the domains of computational fluid dynamics, circuit simulation, and 2D/3D problems.

Tables 4 through 7 contain the results obtained for the relative edge-crossing number (ρ_1), the normalized standard deviation of the edge length (ρ_2), the angular resolution (ρ_3), and the running time (t) for PivotMDS, FM^3, and NB over the set of complex and grid-like graphs described in this section. The two groups of graphs (complex versus grid-like) are separated by a double horizontal line in the four tables. An entry "-" in the PivotMDS column means that no output was produced by OGDF because the input was a disconnected graph, and PivotMDS only works for connected graphs.

Regarding the number of edge crossings, the results for ρ_1 in Table 4 show that NB behaves worse than PivotMDS and FM^3 for the complex graphs. However, the opposite is found for the grid-like graphs, where NB obtains the best results for ρ_1 in general. As far as the uniformity of the edge lengths is concerned, the results for ρ_2 in Table 5 indicate that no algorithm outperforms the other two. For the last quality measure, angular resolution, the results for ρ_3 in Table 6 show that NB is superior for grid-like algorithms, while there is no clear pattern for the complex graphs. Finally, the running times in Table 7 demonstrate that NB lies between PivotMDS and FM^3 in terms of computational efficiency. Nonetheless, due to the influence of the angle ordering operation in NB, the running time increases in a graph like egoFacebook which has a high M/N rate. In summary, NB offers important advantages for grid-like graphs but is outperformed by other state-of-the-art algorithms when applied to complex graphs.

Table 6
Experimental results for the angular resolution (ρ_3). The best value of ρ_3 for each graph is highlighted in bold. (The entry * for ego-Facebook and PivotMDS has been removed since almost half of the edges turned out to be of length zero.)

Name	Nodes	Edges	ρ_3^{PivotMDS}	$\rho_3^{\text{FM}^3}$	ρ_3^{NB}
BarabasiAlbert1000	1000	999	42.191	40.817	**35.736**
BarabasiAlbert5000	5000	4999	**41.976**	42.139	43.537
BarabasiAlbert10000	10000	9999	43.443	43.769	**41.100**
BarabasiAlbert15000	15000	14999	**43.604**	44.297	43.650
BarabasiAlbert20000	20000	19999	43.499	44.581	**42.842**
BarabasiAlbert25000	25000	24999	**42.951**	43.617	42.982
BarabasiAlbert30000	30000	29999	43.957	44.782	**43.187**
BarabasiAlbert35000	35000	34999	**42.620**	43.685	43.533
BarabasiAlbert40000	40000	39999	**42.951**	44.357	43.834
BarabasiAlbert45000	45000	44999	**42.552**	44.463	42.860
BarabasiAlbert50000	50000	49999	43.389	45.183	**43.052**
ego-Facebook	4039	88234	*	23.079	**21.252**
email-Enron	36692	183831	–	64.896	**57.818**
loc-Brightkite	58228	214078	–	60.871	**49.925**
p2p-Gnutella04	10876	39994	**48.043**	59.282	56.351
p2p-Gnutella05	8842	31837	**47.941**	59.329	52.212
p2p-Gnutella06	8717	31525	**49.569**	59.990	55.294
fe_sphere	16386	49152	27.739	**18.516**	19.292
G57	5000	10000	57.057	33.400	**15.227**
grid2	3296	6432	35.265	18.578	**10.253**
netz4504	1961	2578	22.675	14.985	**8.867**
rajat03	7602	12551	86.062	84.649	**63.141**
rbd3200l	3200	7840	**39.829**	48.388	40.098
saylr4	3564	9376	53.715	53.504	**45.661**

To conclude this section, we consider in Fig. 10 the layouts for four graphs as produced by PivotMDS, FM^3, and NB. The two upper rows correspond to examples of complex graphs (BarabasiAlbert1000 and p2p-Gnutella05), while the two lower rows are examples of grid-like graphs (grid2 and netz4504). Intuitively, NB produces nice layouts with no edge crossings for the two grid-like graphs, and less nice layouts for the other two graphs.

[8] https://snap.stanford.edu/data/

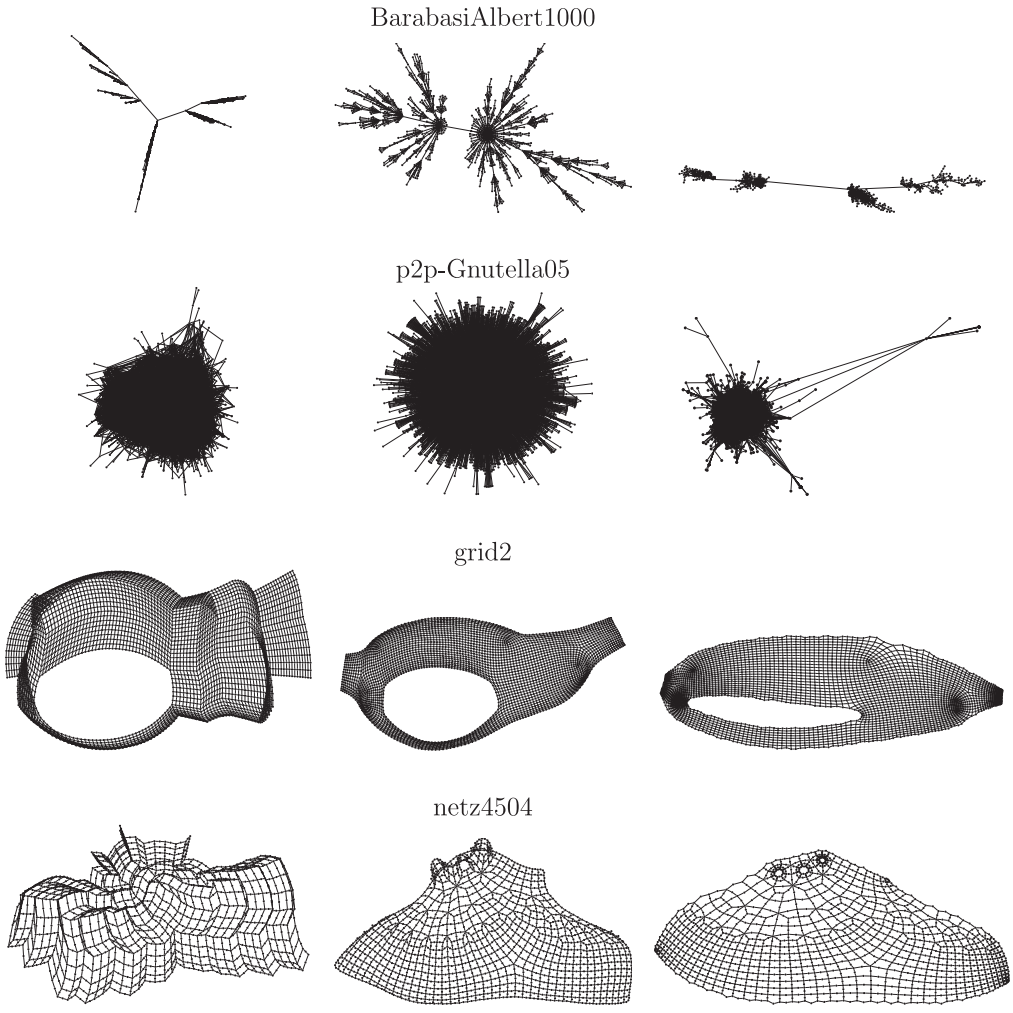

Fig. 10. Four examples of the graph layouts produced by PivotMDS (left-hand column), FM^3 (center column), and NB used as a multi-level algorithm (right-hand column).

Table 7
Experimental results for the running time (t). The best running time for each graph is highlighted in bold.

Name	Nodes	Edges	t^{PivotMDS}	t^{FM^3}	t^{NB}
BarabasiAlbert1000	1000	999	**2.153**	34.320	7.941
BarabasiAlbert5000	5000	4999	**10.421**	182.270	39.187
BarabasiAlbert10000	10000	9999	**20.951**	351.937	79.450
BarabasiAlbert15000	15000	14999	**31.683**	565.517	121.711
BarabasiAlbert20000	20000	19999	**42.042**	689.046	161.616
BarabasiAlbert25000	25000	24999	**53.055**	901.526	206.529
BarabasiAlbert30000	30000	29999	**64.147**	1036.010	251.316
BarabasiAlbert35000	35000	34999	**75.130**	1216.040	293.062
BarabasiAlbert40000	40000	39999	**86.174**	1409.490	334.137
BarabasiAlbert45000	45000	44999	**98.108**	1615.710	375.711
BarabasiAlbert50000	50000	49999	**109.169**	1698.360	423.463
ego-Facebook	4039	88234	**13.291**	89.372	262.658
email-Enron	36692	183831	–	912.823	**881.215**
loc-Brightkite	58228	214078	–	2270.020	**1582.980**
p2p-Gnutella04	10876	39994	**26.832**	356.008	245.826
p2p-Gnutella05	8842	31837	**27.830**	276.089	190.523
p2p-Gnutella06	8717	31525	**21.216**	274.779	188.464
fe_sphere	16386	49152	**54.647**	377.130	198.338
G57	5000	10000	**12.636**	104.083	49.140
grid2	3296	6432	**6.958**	73.772	31.528
netz4504	1961	2578	**4.056**	47.877	16.302
rajat03	7602	12551	**17.067**	273.687	79.528
rbd3200l	3200	7840	**21.965**	81.198	34.398
saylr4	3564	9376	**7.737**	77.704	40.981

4.4. Discussion of experiments

In order to get a deeper understanding of how NB works, it is interesting to study the effects of applying each of NB's three phases. For example, consider the grid_7x7 graph depicted in Fig. 5a, which was generated from an initial random layout after 200 NB iterations with constants $k_1 = 0.999999$, $k_2 = 1$, and $k_3 = 0.1$ as indicated in Table 1. If we repeat the experiment on the random layout in Fig. 11a by applying only Phase 1 (with $k_1 = 0.999999$), we obtain the layout shown in Fig. 11b. Note that the application of Phase 1 alone has eliminated the crossings of the initial random layout of Fig. 11a. Although Phase 1 also produced more uniform edge lengths and angles, further improvement is clearly needed. If we repeat the experiment for the layout in Fig. 11b as initial layout by applying only Phase 1 (with $k_1 = 0.999999$) and Phase 2 (with $k_2 = 1$), the layout depicted in Fig. 11c is obtained. Now, the edge lengths are more uniform in the layout, but the edge angles are inconsistent. Finally, if we repeat the experiment for the layout in Fig. 11c as initial layout by applying Phase 1 (with $k_1 = 0.999999$), Phase 2 (with $k_2 = 1$), and Phase 3 (with $k_3 = 0.1$), then the layout in Fig. 11d is generated. In this layout, edge angles are nearly uniform, and an aesthetic graph layout has been produced. Consequently, all of the three phases turn out to be necessary to produce the good layout shown in Fig. 11d.

In general, as long as the three phases are executed together within each iteration of the NB algorithm (see Fig. 1), we observed

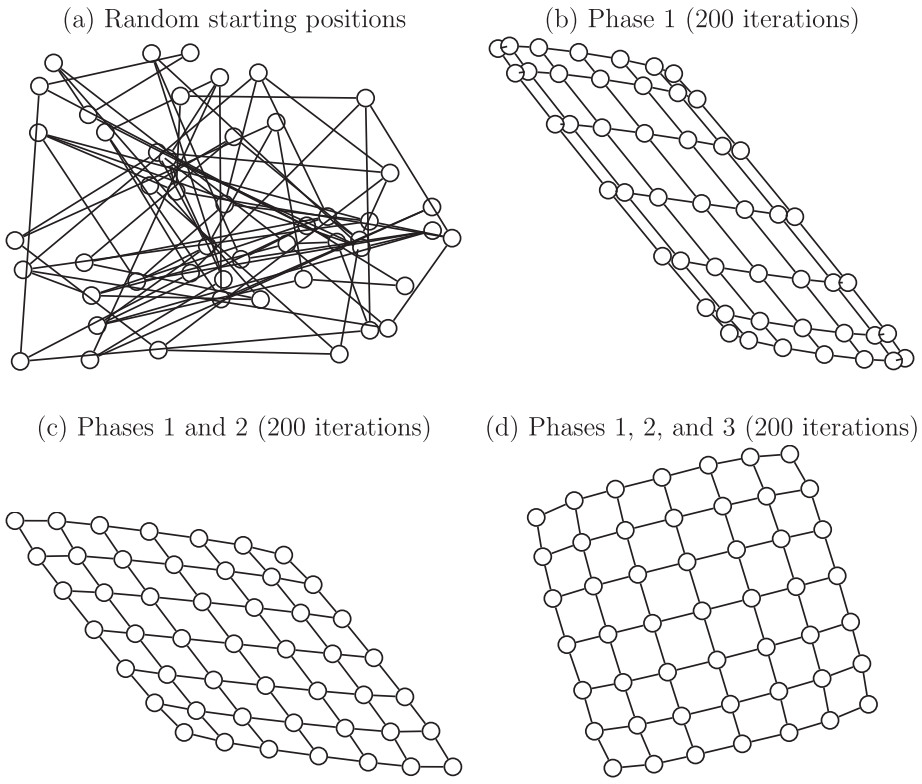

Fig. 11. A grid_7 × 7 graph layout: (a) with random starting positions for the nodes, (b) after the execution of 200 NB iterations with Phase 1 on the random graph layout, (c) after the execution of 200 NB iterations with Phases 1 and 2 on the graph layout in (b), and (d) after the execution of 200 NB iterations with Phases 1, 2, and 3 on the graph layout in (c).

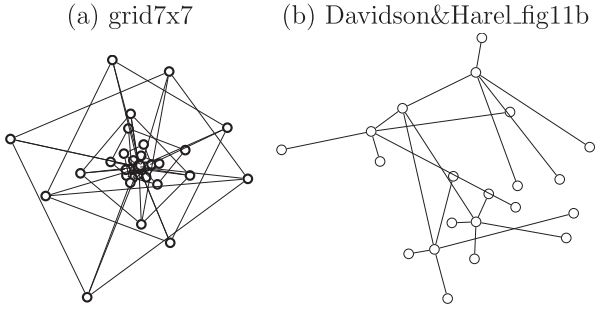

Fig. 12. A grid_7 × 7 graph layout (left) and a Davidson&Harel_fig11b graph layout (right) after the execution of 200 NB iterations with only Phase 1, then 200 NB iterations with only Phase 2, and then 200 NB iterations with only Phase 3.

empirically that the order of the three phases does not significantly change the quality of the layouts produced. However, if the three phases are iterated separately, the layouts become much poorer, as illustrated in Fig. 12a for a grid_7 × 7 graph with random starting positions for its nodes. Note that Figs. 12a and 11d correspond to two different layouts of the same graph. Another example of poor behavior with separate NB phases is included in Fig. 12b for the Davidson&Harel_fig11b graph (see Fig. 5e).

In all of the single-level experiments conducted in this section, we observed that the layouts progressively converge from their initial state (with random node positions) to their final state (shown in Figs. 5 and 6) with no significant occurrence of iterations producing a sudden worsening of the graph layout quality. The convergence speed is influenced by the graph at hand and by the values assigned to the NB parameters.

Our experiments suggest that NB produces high-quality results for grid-like layouts, whereas it achieves somewhat less aesthetic layouts for certain complex graphs, for instance, those in Fig. 8c through f. The reason behind this is that the local interactions defined by the three NB phases match the neighborhood structure for the inner nodes of a grid layout. The running times of NB are impacted in a similar way. In our experiments, NB running times are very competitive for grid-like graphs and become somewhat less competitive for certain complex graphs. An extreme case of this is the flower_1 graph in Table 3.

As stated in Section 2, algebraic layout algorithms are significantly faster than force-directed layout algorithms, and we do not claim that NB is faster than algebraic layout algorithms. In general, our experiments show that NB occupies an intermediate position between algebraic and force-directed methods in terms of running time. Regarding final graph layout quality, as mentioned in Section 2, algebraic layout algorithms often yield graph layouts of inferior quality compared to those produced by force-directed methods. Some examples of layouts obtained from algebraic methods can be found in Figures 7.15 through 7.17 and Figures 7.19 through 7.23 of [[27], Section 7.7]. In these figures, the algebraic methods ACE and HDE cannot clearly visualize the structure of many challenging graphs. In related work [1], the ACE and HDE algorithms did not perform well on any of the evaluation datasets with the exception of the crack graph, and appeared to only work well on graphs with a mesh-like structure. Other specific examples of layouts produced by algebraic methods are shown in [32,39,40], which in general are of lower quality than the ones obtained by force-directed algorithms. In terms of layout quality, similarly to running time, NB tends to be in an intermediate position between algebraic and force-directed methods.

NB may seem very similar to Kamada and Kawai's graph layout method at first glance. However, there are key differences as mentioned in Section 3 and as reflected in Table 3. For instance, Table 3 shows that Kamada and Kawai's method is faster than NB

only in 3 of 14 graphs. On the other hand, the method of Kamada and Kawai often produces graph layouts of better quality than that of the graph layouts computed by NB.

Currently, NB needs an expert user to fine-tune the parameters and determine the number of iterations by visual inspection. There does not seem to be a clear default setting (see Tables 1–3).

5. Conclusion and future research

In this work, we have developed and evaluated a novel message-oriented graph layout method, Neighborhood Beautification (NB), that adopts a message-passing perspective. This perspective is different from but complementary to the algebraic and force-directed perspectives of existing state-of-the-art graph layout algorithms. The new NB method is based on the idea of neighborhood beautification, in which each node tries to improve the layout of its immediate neighbors through message passing. The objective of the messages sent by a node is to aesthetically place its neighbors in the graph.

In the case of grid-like graphs, NB yields graph layouts of comparable quality to those obtained by the best force-directed algorithms and offers important advantages regarding running time for most of the tested graphs. Although NB is slower than algebraic algorithms, it achieves better layout quality, especially when NB is integrated with multi-level algorithms for large graphs. In other words, NB gives a nice trade-off between layout quality and runtime. Another advantage of NB is that it can be used either as a single-level layout method or within a multi-level approach.

The present work opens up the following research directions:

- An interesting topic for future research is to tune the NB parameters automatically.
- Since NB relies on message passing between neighbors, an efficient distributed implementation [2,25] could be developed that is enabled by the graph structure and has the advantage of avoiding excessive communication.
- The use of parallel computing in the context of NB could be investigated in order to improve its computation time for larger graphs [33,45,50].
- This article deals with static graph drawing. However, dynamic graph drawing [18,19] aims to produce aesthetic and useful views of graphs whose structure may change over time. An important aesthetic criterion that needs to be met in dynamic graph drawing is to preserve the user's mental map of the layout. The decentralized nature of NB contributes to its robustness against local changes in the graph structure; as a consequence, NB could be directly employed for dynamic graph drawing in order to maintain the stability of the layouts in an efficient manner, thus preserving the mental map.

References

[1] D. Archambault, T. Munzner, D. Auber, Topolayout: multi-level graph layout by topological features, IEEE Trans. Vis. Comput. Graph. 13(2) (2007) 305–317.
[2] A. Arleo, W. Didimo, G. Liotta, F. Montecchiani, A million edge drawing for a fistful of dollars, in: E. di Giacomo, A. Lubiw (Eds.), Graph Drawing: 23rd International Symposium (GD 2015), Springer, 2015, pp. 44–51.
[3] A.-L. Barabási, R. Albert, Emergence of scaling in random networks, Science 286(5439) (1999) 509–512.
[4] J. Barnes, P. Hut, A hierarchical O(N logN) force-calculation algorithm, Nature 324 (1986) 446–449.
[5] A. Barsky, T. Munzner, J. Gardy, R. Kincaid, Cerebral: visualizing multiple experimental conditions on a graph with biological context, IEEE Trans. Vis. Comput. Graph. 14(6) (2008) 1253–1260.
[6] F. Bertault, A force-directed algorithm that preserves edge-crossing properties, Inf. Process. Lett. 74 (1–2) (2000) 7–13.
[7] U. Brandes, Drawing on Physical Analogies, in: M. Kaufmann, D. Wagner (Eds.), Drawing Graphs, Springer, 2001, pp. 71–86.
[8] U. Brandes, C. Pich, Eigensolver methods for progressive multidimensional scaling of large data, in: M. Kaufmann, D. Wagner (Eds.), Graph Drawing: 14th International Symposium (GD 2006), Springer, 2007, pp. 42–53.
[9] R. Chernobelskiy, K. Cunningham, M.T. Goodrich, S.G. Kobourov, L. Trott, Force-directed Lombardi-style graph drawing, in: W. Didimo, M. Patrignani (Eds.), Graph Drawing: 20th International Symposium (GD 2012), Springer, 2012, pp. 320–331.
[10] J.D. Cohen, Drawing graphs to convey proximity: an incremental arrangement method, ACM Trans. Comput.-Hum. Interact. 4(3) (1997) 197–229.
[11] R. Davidson, D. Harel, Drawing graphs nicely using simulated annealing, ACM Trans. Graph. 15(4) (1996) 301–331.
[12] G. di Battista, P. Eades, R. Tamassia, I.G. Tollis, Graph Drawing: Algorithms for the Visualization of Graphs, Prentice-Hall, 1999.
[13] D.L. Donoho, A. Maleki, A. Montanari, Message-passing algorithms for compressed sensing, Proc. Natl. Acad. Sci. 16(45) (2009) 18914–18919.
[14] P. Eades, A heuristic for graph drawing, Congressus Numerant. 42 (1984) 149–160.
[15] P. Eades, W. Huang, S.-H. Hong, A force-directed method for large crossing angle graph drawing, 2010, arXiv:1012.4559.
[16] B. Finkel, R. Tamassia, Curvilinear graph drawing using the force-directed method, in: J. Pach (Ed.), Graph Drawing: 12th International Symposium (GD 2004), Springer, 2005, pp. 448–453.
[17] A. Frick, A. Ludwig, H. Mehldau, A fast adaptive layout algorithm for undirected graphs, in: R. Tamassia, I.G. Tollis (Eds.), Graph Drawing: 2nd International Symposium (GD 1994), Springer, 1994, pp. 388–403.
[18] C. Friedrich, P. Eades, Graph drawing in motion, J. Graph Algorithms Appl. 6(3) (2002) 353–370.
[19] Y. Frishman, A. Tal, Online dynamic graph drawing, IEEE Trans. Vis. Comput. Graph. 14(4) (2008) 727–740.
[20] T. Fruchterman, E. Reingold, Graph drawing by force-directed placement, Software 21(11) (1991) 1129–1164.
[21] P. Gajer, M.T. Goodrich, S.G. Kobourov, A fast multi-dimensional algorithm for drawing large graphs, Comput. Geom. Theory Appl. 29(1) (2004) 3–18.
[22] P. Gajer, S.G. Kobourov, GRIP: graph drawing with intelligent placement, J. Graph. Algorithms Appl. 6(3) (2002) 203–224.
[23] E.R. Gansner, Y. Hu, S. North, A maxent-stress model for graph layout, IEEE Trans. Vis. Comput. Graph. 19(6) (2013) 927–940.
[24] E.R. Gansner, Y. Koren, S. North, Graph drawing by stress majorization, in: J. Pach (Ed.), Graph Drawing: 12th International Symposium (GD 2004), Springer, 2005, pp. 239–250.
[25] C. Gotsman, Y. Koren, Distributed graph layout for sensor networks, J. Graph Algorithms Appl. 9(3) (2005) 327–346.
[26] , Handbook of Graph Theory, in: J.L. Gross, J. Yellen, P. Zhang (Eds.), Chapman and Hall/CRC, 2013.
[27] S. Hachul, A Potential-Field-Based Multilevel Algorithm for Drawing Large Graphs, Faculty of Mathematics and Natural Sciences, University of Cologne, Cologne, Germany, 2005 Ph.D. thesis.
[28] S. Hachul, M. Jünger, Drawing large graphs with a potential-field-based multilevel algorithm, in: J. Pach (Ed.), Graph Drawing: 12th International Symposium (GD 2004), Springer, 2005, pp. 285–295.
[29] S. Hachul, M. Jünger, Large-graph layout algorithms at work: an experimental study, J. Graph Algorithms Appl. 11(2) (2007) 345–369.
[30] R. Hadany, D. Harel, A multi-scale algorithm for drawing graphs nicely, Discrete Appl. Math. 113(1) (2001) 3–21.
[31] D. Harel, Y. Koren, A fast multi-scale method for drawing large graphs, J. Graph Algorithms Appl. 6(3) (2002) 179–202.
[32] D. Harel, Y. Koren, Graph drawing by high-dimensional embedding, J. Graph. Algorithms Appl. 8(2) (2004) 195–214.
[33] A. Hinge, D. Auber, Distributed graph layout with Spark, in: E. Banissi, T.G. Wyeld (Eds.), Proceedings of the 19th International Conference on Information Visualization, IEEE, 2015, pp. 271–276.
[34] Y. Hu, Efficient, high-quality force-directed graph drawing, Math. J. 10(1) (2006) 37–71.
[35] Y. Hu, L. Shi, Visualizing large graphs, Wiley Interdiscip. Rev. Comput. Stat. 7(2) (2015) 115–136.
[36] W. Huang, P. Eades, S.-H. Hong, C.-C. Lin, Improving multiple aesthetics produces better graph drawings, J. Vis. Lang. Comput. 24(4) (2013) 262–272.
[37] T. Kamada, S. Kawai, An algorithm for drawing general undirected graphs, Inf. Process. Lett. 31 (1989) 7–15.
[38] S.G. Kobourov, Force-directed drawing algorithms, in: R. Tamassia (Ed.), Handbook of Graph Drawing and Visualization, CRC Press, 2013, pp. 383–408.
[39] Y. Koren, Graph drawing by subspace optimization, in: O. Deussen, C. Hansen, D.A. Keim, D. Saupe (Eds.), Proceedings of the 6th Joint Eurographics - IEEE TCVG Symposium on Visualization (VisSym'04), Eurographics Association, 2004, pp. 65–74.
[40] Y. Koren, L. Carmel, D. Harel, Drawing huge graphs by algebraic multigrid optimization, Multiscale Model. Simul. 1(4) (2003) 645–673.
[41] C.-C. Lin, H.-C. Yen, A new force-directed graph drawing method based on edge-edge repulsion, J. Vis. Lang. Comput. 23(1) (2012) 29–42.
[42] R.J. McEliece, D.J.C. Mackay, J.-F. Cheng, Turbo decoding as an instance of Pearl's belief propagation algorithm, IEEE J. Sel. Areas Commun. 16(2) (1998) 140–152.
[43] H. Meyerhenke, M. Nöllenburg, C. Schulz, Drawing large graphs by multilevel maxent-stress optimization, in: E. di Giacomo, A. Lubiw (Eds.), Graph Drawing: 23rd International Symposium (GD 2015), Springer, 2015, pp. 30–43.
[44] C. Muelder, K.-L. Ma, A treemap based method for rapid layout of large graphs, in: I. Fujishiro, H. Li, K.-L. Ma (Eds.), 2008 IEEE Pacific Visualization Symposium (PacificVis 2008), IEEE, 2008, pp. 231–238.

[45] C. Mueller, D. Gregor, A. Lumsdaine, Distributed force-directed graph layout and visualization, in: A. Heirich, B. Raffin, L.P. dos Santos (Eds.), Proceedings of the 6th Eurographics Symposium on Parallel Graphics and Visualization (EGPGV'06), Eurographics Association, 2006, pp. 83–90.
[46] M. Ortmann, M. Klimenta, U. Brandes, A sparse stress model, in: Y. Hu, M. Nöllenburg (Eds.), Graph Drawing: 24th International Symposium (GD 2016), Springer, 2016, pp. 18–32.
[47] H.C. Purchase, Metrics for graph drawing aesthetics, J. Vis. Lang. Comput. 13 (2002) 501–516.
[48] A. Quigley, P. Eades, FADE: Graph drawing, clustering, and visual abstraction, in: J. Marks (Ed.), Graph Drawing: 8th International Symposium (GD 2000), Springer, 2001, pp. 197–210.
[49] R. Tamassia (Ed.), Handbook of Graph Drawing and Visualization, CRC Press, 2013.
[50] A. Tikhonova, K.-L. Ma, A scalable parallel force-directed graph layout algorithm, in: J. Favre, K.-L. Ma, D. Weiskopf (Eds.), Proceedings of the 8th Eurographics Symposium on Parallel Graphics and Visualization (EGPGV'08), Eurographics Association, 2008, pp. 25–32.
[51] D. Tunkelang, A practical approach to drawing undirected graphs, Technical Report CMU-CS-94-161, School of Computer Science, Carnegie Mellon University, 1994.
[52] D. Tunkelang, A Numerical Approach to General Graph Drawing, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, 1999 Ph.D. thesis.
[53] W.T. Tutte, How to draw a graph, Proc. London Math. Soc. 13(1) (1963) 743–767.
[54] D. Vrajitoru, Hybrid multiobjective optimization genetic algorithms for graph drawing, in: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2007), ACM Press, 2007, p. 912.
[55] C. Walshaw, A multilevel algorithm for force-directed graph drawing, J. Graph Algorithms Appl. 7(3) (2003) 253–285.
[56] U. Wilensky, NetLogo, (http://ccl.northwestern.edu/netlogo/). Center for Connected Learning and Computer Science, Northwestern University, Evanston, IL, 1999.