Revisiting Stress Majorization as a Unified Framework for Interactive Constrained Graph Visualization

Yunhai Wang, Yanyan Wang, Yinqi Sun, Lifeng Zhu, Kecheng Lu
Chi-Wing Fu, Michael Sedlmair, Oliver Deussen, and Baoquan Chen

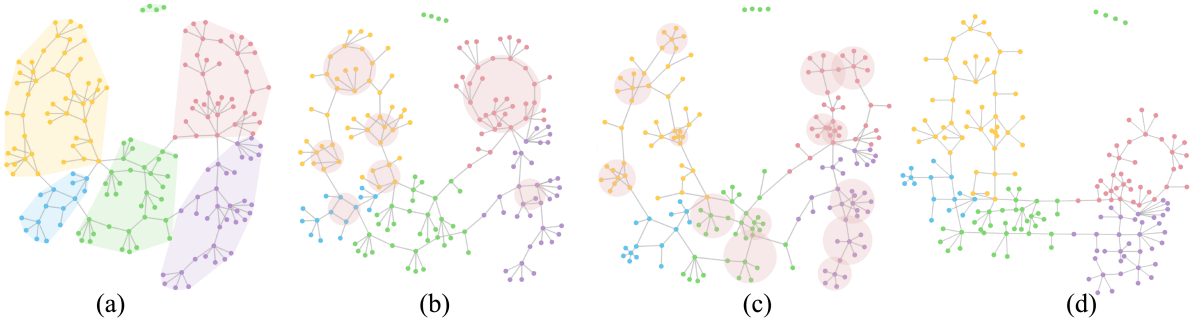


Fig. 1. Our unified framework for constrained graph visualization allows us to create graph layouts with various constraints: a) cluster non-overlap (CN); b) CN + circle constraint (CC); c) CN + star constraint (SC); and d) CN + CC + SC + edge direction constraint.

Abstract—We present an improved stress majorization method that incorporates various constraints, including directional constraints without the necessity of solving a constraint optimization problem. This is achieved by reformulating the stress function to impose constraints on both the edge vectors and lengths instead of just on the edge lengths (node distances). This is a unified framework for both constrained and unconstrained graph visualizations, where we can model most existing layout constraints, as well as develop new ones such as the star shapes and cluster separation constraints within stress majorization. This improvement also allows us to parallelize computation with an efficient GPU conjugant gradient solver, which yields fast and stable solutions, even for large graphs. As a result, we allow the constraint-based exploration of large graphs with 10K nodes – an approach which previous methods cannot support.

Index Terms—Graph visualization, stress majorization, constraints

1 Introduction

Graphs are fundamental representations for a wide variety of data, such as transport networks, social networks, and molecular interactions. A very popular method for automatically laying out nodes and edges of a graph to create a planar visualization, while maintaining its structure, is the *stress model* [28]. This model aims to minimize the sum of squared distance differences between pairs of nodes in the graph when laying out nodes in the visualization. Only satisfying such distance constraints, however, might not generate a user-desired layout for an effective visualization, since various *application-specific layout requirements* [13] including non-overlapping of nodes and directed edges between nodes should be maintained. Ideally, the mental map of the user should be preserved during the evolution of a graph layout [1, 34].

This motivated the development of constrained graph layout methods that enable users to define *user-specific constraints* [25, 39]. Often, this is achieved by adapting a traditional graph layout optimization to a constrained case. Taking the stress function as the objective, *stress majorization* handles such constraints by projecting the node positions resulted from an unconstrained optimization to feasible positions that satisfy the constraints [13, 17]. Later, Dwyer et al. [11] proposed a fast scalable method with an unconstrained graph layout as input. The method's applicability is demonstrated for interactive graph navigation [15] and diagram creation [16].

Despite their benefits, existing constrained graph layout methods have three major limitations that hamper their applicability. First, they attempt to satisfy hard constraints, which could lead to high stress values that are not preferred by users [14]. Second, they fail to support several aesthetic criteria [36, 37], e.g., minimizing the edge crossings and maximizing the layout symmetry. These criteria are useful for effective examination of local structures, but they generally require a dynamic update of the orientation constraints, which is not supported by existing methods. Finally, due to high computational costs, existing methods work well only for selected small sub-graphs [15], but not for larger graphs beyond hundreds of nodes. The main reason for this is that users often need a large amount of constraints to achieve a high-quality layout for larger graphs; this requires a rather expensive constrained optimization, which cannot be solved efficiently by existing methods.

The above issues motivate us to revisit an unconstrained stress model, which can be efficiently solved by stress majorization. In short, we *revisit the mathematical model of stress majorization* and *reformulate it in terms of a vector form*, where we minimize the sum of squared differences between *the vectors* that connect pairs of nodes in the planar visualization, referred to as *edge vectors*. By default, the magnitude of an edge vector is the ideal distance[1] between the associated pair

- *Y.H. Wang, B. Chen, Y.Y. Wang, Y. Sun, and K. Lu are with Shandong University. Email: {wang.yh, baoquan}@sdu.edu.cn, {yanyanwang93, sunyinqi0508, lukecheng0407}@gmail.com.*
- *L. Zhu is with Southeast University. E-mail: lfzhulf@gmail.com.*
- *C.-W. Fu is with the Chinese University of Hong Kong. E-mail: cwfu@cse.cuhk.edu.hk.*
- *M. Sedlmair is with University of Vienna, Austria. E-mail: michael.sedlmair@univie.ac.at.*
- *O. Deussen is with Konstanz University and VCC SIAT, China. E-mail: oliver.deussen@uni-konstanz.de.*
- *Y.H. Wang and Y.Y. Wang are joint first authors.*

Manuscript received xx xxx. 201x; accepted xx xxx. 201x. Date of Publication xx xxx. 201x; date of current version xx xxx. 201x. For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org. Digital Object Identifier: xx.xxxx/TVCG.201x.xxxxxxx

[1]The ideal distance between two nodes in a graph is generally defined as the

of nodes. This reformulation introduces auxiliary variables *edge directions*, which provide us with an alternative interpretation of stress majorization and enable us to further model various kinds of user constraints.

Based on our reformulation, we show that stress majorization can be developed into a *unified framework* for *both* constrained and unconstrained graph visualizations. Hence, compared to previous constrained graph layout methods [11, 13, 17], our framework can handle any constraint that can be modeled using edge vectors. This capability allows us to model almost all existing layout constraints and potentially to create new ones. These constraints can be categorized into three classes: i) *direct constraints* explicitly specify the edge vectors; ii) *metrics-based constraints* require an update of the edge vectors based on the applied aesthetic criteria such as minimizing the edge crossings and overlapping between clusters/nodes; and iii) *shape-based constraints* enforce a local structure in the layout to follow a certain shape by means of shape matching. These constraints are helpful for merging of small graph layouts [51] and incremental layouts for dynamic graph visualization [8]. Moreover, we can combine multiple constraints with different edge weights to relieve the conflicts between them.

Furthermore, our new formulation enables us to accelerate stress majorization using the GPU. This allows users to interactively explore constrained layouts, even for very large graphs with 10K nodes and more, and to examine sub-graphs of interest with various constraints. Users can start from an unconstrained layout, select a sub-graph by using a lasso, and explore its structures by imposing constraints to the layout. Once this is done, the sub-graph evolves according to the constraints while the rest of the graph is kept as stable as possible. The sub-graph with constraints can be shown in a detailed view to hide the influence of the unselected structures.

In summary, the main contributions of this paper are:

- we revisit stress majorization and show that it can be reformulated into a unified framework for both constrained and unconstrained graph visualizations;
- we devise various constraints, including the direct constraints, metrics-based constraints and shape-based constraints, to effectively explore the structures in sub-graphs of interest; and
- we present an interactive constrained graph exploration interface supported by GPU-accelerated stress majorization; our interface enables users to interactively explore graph layouts of 10K nodes with various constraints.

2 Related Work

2.1 Constrained Graph Layout

Many graph layout algorithms have been proposed in the past, see [44], among which the force-directed approach is the most popular. This approach is built upon a spring-electrical model [18, 21]; it treats edges as springs and nodes as charged particles, and simulates repulsive forces between all pairs of nodes and attractive forces between adjacent nodes. While this approach has proven to be scalable [49] and can yield reasonable layouts, it may not preserve edge length in the visualization. In contrast, the stress model by Kamada and Kawai [28] assumes that the springs connect all pairs of nodes in the graph with an ideal spring length, which equals to an ideal edge length, thus producing layouts of comparatively high quality. The stress model is rooted in multidimensional scaling [29] that can be efficiently solved with stress majorization [22]. Although both models might produce aesthetically-pleasing graph layouts, they cannot generate customized layouts that reflect the semantics of the graph or the preferences of a user.

To address these issues, constrained graph layout [3] was proposed to introduce layout constraints for node placement. Early works focus on drawing directed graphs and constrain the layout to show the graph hierarchy. Most approaches, e.g., [43] achieve this by computing x and y coordinates of the objects in separate stages with different objectives. Typically, the y-coordinate is computed by dividing the y-axis into a finite number of layers and assigning one layer to each node with a goal of minimizing edge lengths, while the x-axis placement is adjusted for maintaining additional aesthetic considerations, e.g., minimizing edge crossings. The corresponding optimization problems are NP-hard, so heuristic methods have been designed [24, 27] to solve the problem quickly. Carmel et al. [5] design an alternative approach that associates nodes with continuous y-coordinates that are computed by an energy minimization. Such an axis separation strategy is algorithmically appealing but makes it hard to control the layout aesthetics. To overcome the issue, Dwyer et al. [12] propose Dig-CoLa that computes all axes simultaneously by integrating constraints into stress majorization.

To enhance the functionality, various constraints are introduced to improve the force-directed placements. Ryall et al. [39] model constraints as springs, which are satisfied using a spring-electrical simulation model with different stiffness values. Wang and Miyamoto [50] as well as Huang and Eades [26] improve force-directed placement by discouraging node overlapping and encouraging a compact drawing of clusters. Recently, Simonetto et al. [40] incorporate planar constraints into an improved force-directed algorithm to prevent edge crossing. All these methods consider soft constraints, where weights are adjustable. Our method takes a similar direction and models the constraints as springs, enabling us to fuse multiple constraints with different weights. The main difference is that we incorporate the springs into the stress model and solve the optimization with stress majorization, which guarantees a monotonously converging stress function.

To impose hard constraints, He and Marriot [25] add separation constraints to the stress model using quadratic programming techniques. Due to the inefficiency of the solver, however, they demonstrate their method only with very small graphs. Dwyer et al. [13] extend stress majorization to satisfy separation constraints by modifying the quadratic goal function in each iteration. Such quadratic programs can be solved quickly with an iterative gradient projection algorithm. Later, Dywer et al. [17] extend the constraint graph layout to allow a preservation of the topology by using a new stress function that minimizes the total path length of routed edges rather than just the straight edges. To satisfy non-linear constraints such as circular sub-graphs, Dwyer [11] further extend the constraints (inequalities or equalities) over either x- or y-position variables to the ones over Euclidean space. Hence, the new constraints can be solved by fast and scalable approaches that combine state-of-the-art force-directed layout approaches with simple constraint relaxation schemes. Similar to Dwyer et al. [13], our method is also based on stress majorization, but we reformulate stress majorization and show that the constrained graph layout can be computed by an unconstrained stress majorization, which can be efficiently solved.

2.2 Interactive Graph Exploration

A variety of interaction techniques have been designed for large graph exploration; see Von et al. [48] for an overview. Among them, we focus on those for exploring topological structures. In general, most of these techniques [19, 41, 46] provide the user with i) an overview of the entire graph, and ii) a detailed view of a small sub-graph around a focal node. Often the overview layout is produced by using a scalable force-directed method, while the detailed view is simply a zoomed version of the overview. In other words, if the sub-graph of interest is not clearly recognizable in the overview, it may not be comprehensible in the detailed view. To resolve this issue, Dwyer et al. [15] suggest to use a constrained graph layout algorithm for the detailed view as well. The structures of interest, however, might not be synchronized in both views, especially for the clusters. Moreover, the authors demonstrate their system only with graphs of less than a thousand nodes due to the slow speed of the constrained layout algorithm [17]. Our efficient unconstrained stress majorization is capable of providing a high-quality layout in the overview as well as in the detailed view.

To prevent clutter in node-link diagrams, focus+context interaction [45] is often used for graph exploration. Lamping et al. [30] present an hyperbolic browser that visualizes large tree graphs with a user-selected focus on details, while preserving the context of the entire graph. By pre-computing a hierarchy of coarsened graphs, the topological fisheye approach [23] achieves similar visualizations for general

shortest-path distance between them.

graphs. Van Ham and Perer [47] propose a general framework for using different degree of interest (DOI) functions and use the corresponding metrics to direct the user to explore interesting sub-graphs within a context. Our solution also involves focus+context visualization, where the selected area can be shown clearly with user-specified constraints, while the rest of the structures are kept as stable as possible. Our method does not introduce as much distortions as fisheye views [45], and maintains the stress constraints of the entire graph during the whole procedure.

3 Revisiting Stress Majorization

The main idea of our method is to reformulate the stress model in a way that allows us to define a unified framework for solving *both* constrained and unconstrained graph layout problems. To this end, we treat each edge as a vector, we say *edge vector*, whose magnitude and direction can be given as a constraint. In traditional stress models, only edge lengths are used as constraints. Below, we first review the stress model and stress majorization, and then present our reformulation.

Given a graph $G(V,E)$, where V is a set of n nodes and E a set of m edges, the stress model proposed by Kamada and Kawai [28] places nodes in 2D space by minimizing the difference between the resulting pairwise Euclidean distances and their corresponding target distances:

$$S(\mathbf{X}) = \sum_{i<j} w_{ij}(\parallel \mathbf{x}_i - \mathbf{x}_j \parallel - d_{ij})^2 \,, \tag{1}$$

where $\mathbf{x}_i$ is the position of the i-th node in the 2D visualization space; $\mathbf{X} = \{\mathbf{x}_1, \cdots, \mathbf{x}_n\}^T$ is an $n \times 2$ matrix for all the nodes; d_{ij} gives the target distance between nodes i and j; and $w_{ij} = d_{ij}^{-2}$ is used as a normalization constant. By minimizing the stress function $S(\mathbf{X})$, the layout in the planar visualization minimizes the stress error. To minimize Eq. (1), Gansner et al. [22] propose *majorization*, which offers distinctive advantages over the original implementation [28]; most importantly, it guarantees a monotonic decrease of the stress.

3.1 Reformulation in Vector Form

The core idea of stress majorization [22] is to use the upper bound of Eq. (1) rather than solving it directly. By expanding Eq. (1), one can obtain three terms:

$$S(\mathbf{X}) = \sum_{i<j} w_{ij} d_{ij}^2 + \sum_{i<j} w_{ij} \parallel \mathbf{x}_i - \mathbf{x}_j \parallel^2 - 2\sum_{i<j} w_{ij} d_{ij} \parallel \mathbf{x}_i - \mathbf{x}_j \parallel, \tag{2}$$

where the first term is a constant and the second term is a quadratic sum. By using the Cauchy-Schwarz inequality $\parallel \mathbf{x} \parallel \parallel \mathbf{y} \parallel \geq \mathbf{x}^T\mathbf{y}$ and dividing both sides by $-\parallel \mathbf{z}_i - \mathbf{z}_j \parallel$, one can bound the third term by:

$$-\sum_{i<j} w_{ij} d_{ij} \parallel \mathbf{x}_i - \mathbf{x}_j \parallel \;\leq\; -\sum_{i<j} w_{ij} d_{ij} \frac{(\mathbf{x}_i - \mathbf{x}_j)^T(\mathbf{z}_i - \mathbf{z}_j)}{\parallel \mathbf{z}_i - \mathbf{z}_j \parallel} \,. \tag{3}$$

Here, the method introduces the auxiliary variable $\mathbf{Z} = \{\mathbf{z}_1, \cdots, \mathbf{z}_n\}^T$, which is an $n \times 2$ matrix; the equality holds when $\mathbf{X} = \mathbf{Z}$. Based on this inequality, the stress function is bounded by

$$S(\mathbf{x}) \leq \sum_{i<j} w_{ij} d_{ij}^2 + \sum_{i<j} w_{ij} \parallel \mathbf{x}_i - \mathbf{x}_j \parallel^2 - 2\sum_{i<j} w_{ij} d_{ij} (\mathbf{x}_i - \mathbf{x}_j)^T \frac{\mathbf{z}_i - \mathbf{z}_j}{\parallel \mathbf{z}_i - \mathbf{z}_j \parallel} \,. \tag{4}$$

Putting $\mathbf{d}_{ij} = d_{ij} \frac{\mathbf{z}_i - \mathbf{z}_j}{\|\mathbf{z}_i - \mathbf{z}_j\|}$, we rewrite Eq. (4) as

$$\begin{aligned} S(\mathbf{x}) &\leq \sum_{i<j} w_{ij} \parallel \mathbf{d}_{ij} \parallel^2 + \sum_{i<j} w_{ij} \parallel \mathbf{x}_i - \mathbf{x}_j \parallel^2 - 2\sum_{i<j} w_{ij} (\mathbf{x}_i - \mathbf{x}_j)^T \mathbf{d}_{ij} \\ &= \sum_{i<j} w_{ij} ||\mathbf{x}_i - \mathbf{x}_j - \mathbf{d}_{ij}||^2 . \end{aligned} \tag{5}$$

Compared to Eq. (1), the reformulated quadratic term in Eq. (5) now takes a vector form, in which we minimize the sum of squared differences between the vectors that connect pairs of nodes, whose magnitudes are the target distances. By using the above-mentioned *edge vectors* (denoted by $\mathbf{d}_{ij}$), we allow the user to explicitly control not only *edge lengths* but also *edge directions* at the same time.

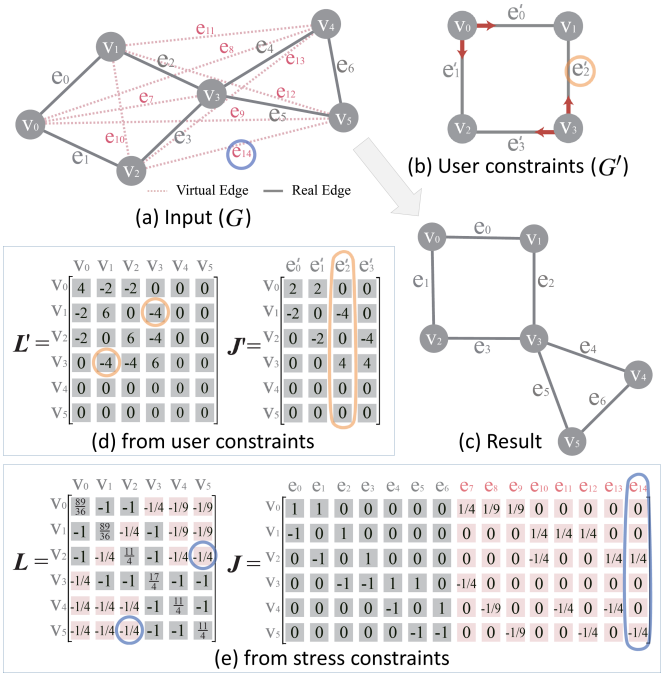


Fig. 2. An illustrative example shows how we optimize a given graph G (from (a) to (c)) by stress constraints together with user constraints (in this example, edge orientations) defined on a constraint graph G' (b), where four directed edge constraints $\{e'_0, e'_1, e'_2, e'_3\}$ (red arrows) are enforced on the nodes $\{v_0, v_1, v_2, v_3\}$ in (a). Note that real and virtual edges are drawn as solid grey lines and dashed red lines, respectively, and in (a), the lengths of real edges are all 1 and we set the target distance between nodes as the shortest path distance between nodes in the graph, so d_{34}=1, d_{14}=2, and d_{04}=3, while w_{34}=1, w_{14}=1/4, and w_{04}=1/9. (d) $\mathbf{L}'$ and $\mathbf{J}'$ (defined by weights in user constraints); and (e) $\mathbf{L}$ and $\mathbf{J}$ (defined by w_{ij}). The orange and blue circles in (d) and (e) reveal the relationship between $\mathbf{L}'$ and $\mathbf{J}'$, and $\mathbf{L}$ and $\mathbf{J}$, respectively.

3.2 Optimization Process

Our reformulation provides an alternative interpretation of the optimization process of stress majorization. For convenience, we rewrite Eq. (5) in matrix form [33]:

$$\mathrm{Tr}(\mathbf{X}^T\mathbf{L}\mathbf{X}) - 2\mathrm{Tr}(\mathbf{X}^T\mathbf{J}\mathbf{D}) + C, \tag{6}$$

where $\mathbf{L}$ is the $n \times n$ weighted Laplacian matrix:

$$\mathbf{L}_{ij} = \begin{cases} -w_{ij} & i \neq j \\ \sum_{k \neq i} w_{ik} & i = j \end{cases} \,; \tag{7}$$

$\mathbf{D} \in \mathbb{R}^{s \times 2}$ (with $s = n(n-1)/2$) is a matrix that includes all the edge vectors between every pair of nodes in the visualization; C is a constant; and $\mathbf{J} \in \mathbb{R}^{n \times s}$ is a matrix that stores the weights of all the edge vectors. The k-th column in $\mathbf{J}$ corresponds to the k-th edge vector, which is defined as

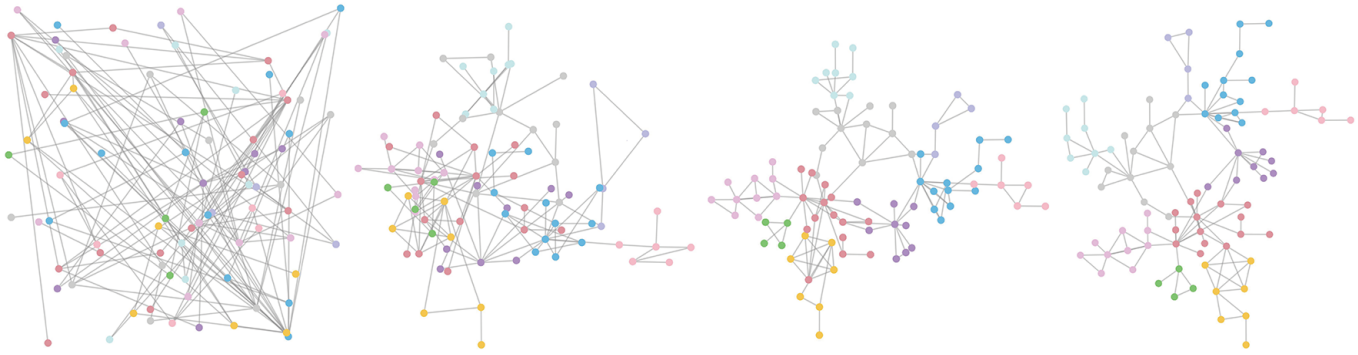
$$\mathbf{J}_{ik} = \begin{cases} w_{ij} & \text{if i is the source node of the } k\text{-th edge vector} \\ -w_{ij} & \text{if i is the target node of the } k\text{-th edge vector} \\ 0 & \text{otherwise.} \end{cases} \tag{8}$$

Both, $\mathbf{J}$ and $\mathbf{L}$ are defined by the weights of the edge vectors, see the blue circles highlighted in Figure 2(a,e) for their relationship. Note that $G(V,E)$ is an undirected graph, and thus we consider each edge only once in $\mathbf{J}$ and $\mathbf{L}$.

By differentiating Eq. (6) with respect to $\mathbf{X}$ and setting the derivative to zero, we obtain:

$$\mathbf{L}\mathbf{X} \;=\; \mathbf{J}\mathbf{D} \,, \tag{9}$$

which is equivalent to the linear system solved in the original stress majorization [22] but has auxiliary variables, i.e., edge vector directions.



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

Fig. 3. Illustration of the convergence of our method. We show intermediate results of a graph optimized with multiple constraints: edge directions, cluster non-overlap, circles as well as star shapes: (a) random initialization; (b) results after 5 iterations; (c) results after 15 iterations; and (d) results after 30 iterations.

This enables us to apply a block coordinate descent method [42] by alternating between finding the optimal vector directions $\mathbf{D}$ (D-step) and finding the node positions $\mathbf{X}$ (P-step):

D-step: Update $\mathbf{D}$ using the current configuration $\mathbf{X}$; and

P-step: Fix $\mathbf{D}$ and compute $\mathbf{X}$ by solving Eq. (9).

This solving process provides a new interpretation of stress majorization. In the P-step, $\mathbf{X}$ can be computed by solving the linear system in the same way as the original stress majorization, while in the D-step, $\mathbf{D}$ can be updated by using the solution of the previous step:

$$\mathbf{d}_{ij} = d_{ij} \frac{\mathbf{x}_i(t) - \mathbf{x}_j(t)}{\| \mathbf{x}_i(t) - \mathbf{x}_j(t) \|}. \tag{10}$$

This indicates that we do not impose any edge direction to any edge where the edge length d_{ij} is already constrained. This way, we are able to *decouple* distance (or length) and directional constraints.

3.3 Incorporating User Constraints

In the original formulation (cf. Eq. (1)), stress energy $S(\mathbf{X})$ can be regarded as a set of distance constraints between node pairs; we might call them *stress constraints*. In contrast, our formulation with edge vectors $\mathbf{d}_{ij}$ encodes both distances and directions between node pairs: our proposition is that we can adopt the same framework to model stress constraints as well as to allow users to directly or indirectly specify *constraints on the edge vectors* to control a layout. To differentiate from $\mathbf{d}_{ij}$, which denotes the edge vectors of stress constraints, we define $\mathbf{d}'_{ij}$ to denote the target edge vectors for user-specified constraints, or user constraints. In Section 4, we will show that various constraints can be modeled using $\mathbf{d}'_{ij}$, and this is a unified and flexible way of modeling various kinds of graph layout constraints.

To extend the current formulation to include user constraints with $\mathbf{d}'_{ij}$, we form a directed (constraint) graph $G'(V',E')$, where $V' \subseteq V$ and E' may contain real as well as virtual edges in G; see Figure 2(a, b) for an example G and G'. Combining user constraints with stress constraints, we obtain:

$$\sum_{i<j} w_{ij} \| \mathbf{x}_i - \mathbf{x}_j - \mathbf{d}_{ij} \|^2 + \sum_{(i,j)\in E'} v_{ij} \| \mathbf{x}'_i - \mathbf{x}'_j - \mathbf{d}'_{ij} \|^2 , \tag{11}$$

where v_{ij} is the user-specified weight of target edge vector $\mathbf{d}'_{ij}$ in G'. Note that in Eq. (11), the first term considers all the real and virtual edges in G, while the second term considers only the edges in the constraint graph (G'). By rewriting it into the matrix form, we get

$$\begin{aligned} &\mathrm{Tr}(\mathbf{X}^T\mathbf{L}\mathbf{X}) - 2\mathrm{Tr}(\mathbf{X}^T\mathbf{J}\mathbf{D}) + \mathrm{Tr}(\mathbf{X}^T\mathbf{L}'\mathbf{X}) - 2\mathrm{Tr}(\mathbf{X}^T\mathbf{J}'\mathbf{D}') + C' \\ &= \mathrm{Tr}(\mathbf{X}^T(\mathbf{L}+\mathbf{L}')\mathbf{X}) - 2\mathrm{Tr}(\mathbf{X}^T(\mathbf{J}\mathbf{D}+\mathbf{J}'\mathbf{D}')) + C' , \end{aligned} \tag{12}$$

where C' is a constant, $\mathbf{D}'$ consists of all the target edge vectors ($\mathbf{d}'_{ij}$), and $\mathbf{L}'$ and $\mathbf{J}'$ are defined by the user-specified weights (v_{ij}) and constructed according to Eq. (14) and Eq. (8), respectively. For the example constraint graph G' shown in Figure 2(b), its V' is a subset of nodes of G; see Figure 2(a), and Figure 2(d) shows the corresponding $\mathbf{L}'$ and $\mathbf{J}'$, where the four edges of G are constrained by the four target edge vectors $\{e'_0, e'_1, e'_2, e'_3\}$ with weights $\{2,2,4,4\}$.

Next, to minimize $F^{\mathbf{Z}}_{new}(\mathbf{X})$, we differentiate Eq. (12) with respect to $\mathbf{X}$ and setting the derivative to zero; thus, we obtain:

$$(\mathbf{L}+\mathbf{L}')\mathbf{X} = \mathbf{J}\mathbf{D}+\mathbf{J}'\mathbf{D}' = \begin{bmatrix} \mathbf{J} & \mathbf{J}' \end{bmatrix} \begin{bmatrix} \mathbf{D} \\ \mathbf{D}' \end{bmatrix} , \tag{13}$$

which is still a linear system that can be solved by alternating between D- and P-steps. Compared to Eq. (9), the P-step can be solved in the same way, but the D-step now involves new variables $\mathbf{D}'$. If a constraint explicitly defines the direction and length of variable $\mathbf{d}'_{ij}$, such as the separation constraint used by Dwyer et al. [13], the update of $\mathbf{D}'$ is straightforward, otherwise we should satisfy the constraint with the least layout changes. By doing so, Eq. (13) is able to handle conflicts between different constraints. An example is shown in Figure 1(d), which demonstrates that our method can properly resolve conflicts between the circle and edge direction constraints; see Section 4.

3.4 Solving the Linear System

For computing $\mathbf{X}$ in the P-step (Eq. (13)), Gansner et al. [22] recommend to use either a conjugate gradient (CG) solver, or Cholesky factorization. Since we aim for interactive performance, we use the CG solver due to its iterative nature. Once the user adds/deletes a constraint during the interaction, we take the result from the previous iteration of the CG solver as the input for the updated Eq. (12). This leads to a very efficient update of the results in comparison to a costly re-factorization of the system matrix in the case of using Cholesky factorization.

Figure 3 illustrates the fast convergence of the process by showing intermediate results of a graph, where the result of the 15th iteration almost satisfies all the constraints. For interactive exploration of large graphs, however, the CPU version is still not fast enough. Therefore, we developed a GPU implementation of our system based on the GPU-based linear algebra library ViennaCL [38].

4 User Constraints on Graphs

Having reformulated stress majorization, we next show how to adapt our framework to model various kinds of constraints in a unified fashion, including some of the constraints proposed in previous works [11–13], e.g., edge orientations, circularity, and non-overlapping clusters. Besides existing constraints, our framework also allows us to define new constraints to improve the graph layout quality.

In our framework, we categorize user constraints into three classes: *direct*, *metrics-based*, and *shape-based* constraints. For direct constraints, we directly specify target edge vectors ($\mathbf{d}'_{ij}$) for the related

edges in graph. For metrics-based constraints, the target edge vectors of the related nodes are computed based on the applied aesthetic metrics, while for shape-based constraints, we constrain the layout to follow certain reference shapes. In the following, we describe each class of constraints and explain how we update $\mathbf{d}'_{ij}$ in the D-step based on the corresponding layout configuration $\mathbf{X}$, which is $\{\mathbf{x}_1, \cdots, \mathbf{x}_n\}^T$.

4.1 Direct Constraints

Direct constraints explicitly specify the target edge vectors, so the update of $\mathbf{d}'_{ij}$ is straightforward. By representing the target edge vector between nodes i and j with a unit vector $\mathbf{u}_{ij}$ and length l_{ij} , we set $\mathbf{d}'_{ij} = l_{ij} * \mathbf{u}_{ij}$. The edge length constraint, edge orientation constraint, and temporal coherence constraint all belong to this class.

Edge Length Constraint. This is like a stress constraint that follows the original stress majorization. After the user employs a lasso to select edges in the graph, our method constrains the $\mathbf{d}'_{ij}$ of each selected edge to take a certain user-specified length l_{ij} like an ideal distance [11]; hence, we set these $\mathbf{d}'_{ij}$ by a simple scaling: $l_{ij} * (\mathbf{x}_i - \mathbf{x}_j) / \parallel \mathbf{x}_i - \mathbf{x}_j \parallel$.

Edge Direction Constraint. This constraint is similar to the previous length constraint, but here we constrain the $\mathbf{d}'_{ij}$ of each selected edge to follow a certain user-specified direction $\mathbf{u}_{ij}$, i.e., we set these $\mathbf{d}'_{ij}$ as $||\mathbf{x}_i - \mathbf{x}_j|| \mathbf{u}_{ij}$.

Temporal Coherence Constraint. To enable the creation of a coherent time-varying graph layout, we take the graph layout from the previous time step as a reference layout and map the graph layout of the current time step to it. Suppose the layout configuration at time $t-1$ is $\mathbf{X}(t-1)$, we set the target edge vector $\mathbf{d}'_{ij}(t)$ at time t to be $\mathbf{d}'_{ij}(t-1)$ for all node pairs that exist at both times. Since we also keep the original unconstrained stress model through the stress constraints in Eq. (13), nodes that only exist in the graph at time t can also be properly positioned. Figure 4 illustrates temporal coherence, showing that nodes (A, C, D, E, and F) existed in both graphs have similar structures.

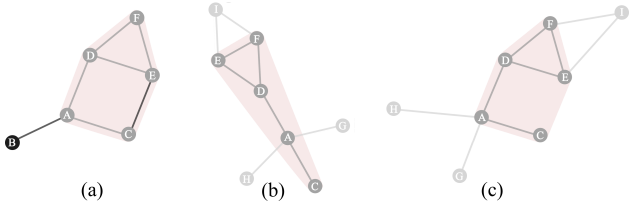


Fig. 4. Enforcing temporal coherence for a dynamic graph visualization: (a) layout at time $t-1$; (b) layout at time t without the temporal coherence constraint; and (c) layout at time t after enforcing the temporal coherence constraint. A sub-graph existed at both times (nodes A, C, D, E, & F) is highlighted with a pink background. Nodes and edges that exist at both times are shown in gray, elements that existed only in (a) are shown in black, while elements that existed only in (b,c) are shown in light gray.

4.2 Metrics-based Constraints

Metrics-based constraints aim to optimize a layout to meet certain aesthetic criteria, such as reducing cluster overlap and minimizing edge crossings. In this class, the target edge vector to be constrained is not directly provided but needed to be determined based on the metric.

Non-overlap Constraint. Clusters are prominent structures that always catch viewer's attention. Hence, if multiple clusters overlap in a graph layout, we try to separate them to improve the readability of their structures. A method to detect overlapping clusters can be found in [11], which refers to minimal penetration depth (mpd) for collision detection in rigid-body simulations [10]. Since cluster boundaries are usually convex, we estimate the vector **mpd** between two overlapping clusters by finding the shortest vector to move a cluster out of the other; see the grey arrow (**mpd**) in Figure 5(a) for an example.

To formulate a non-overlap constraint, we make use of the **mpd** to modify $\mathbf{d}'_{ij}$. Given two overlapping clusters C_a and C_b, for each node i in C_a and each node j in C_b, we add **mpd** to the associated target edge vector between them: $\mathbf{d}'_{ij} = (\mathbf{x}_i - \mathbf{x}_j) + \mathbf{mpd}$. Hence, the constraint will push the nodes of the two clusters away from each other by **mpd**; (Figure 5(a)). Furthermore, if we take each node as a cluster, non-overlap for node boundaries (cf. [11]) can be considered as a special case of this non-overlap constraint.

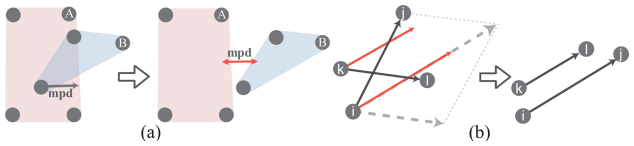


Fig. 5. Metrics-based constraints: (a) we push two overlapping clusters away from each other by taking **mpd** into the target edge vectors between nodes in the two clusters; and (b) the weighted average of two crossing edge vectors $\overrightarrow{ij}$ and $\overrightarrow{kl}$ is taken as the direction component of the target edge vectors (shown in red) to avoid the crossing.

Minimizing Edge Crossings. Excessive edge crossings prevent us from seeing local graph structures. However, it is not always possible to avoid all the crossings, cf. [27] and current methods can mainly deal with crossings in small graphs. In this work, we do not aim to solve the general edge crossing problem. Instead we show that using our unified framework with the edge vectors, we are able to model a simple constraint to resolve edge crossings in small graphs.

Given a small sub-graph of interest selected by lasso, we first identify all pairs of intersecting edges inside, and then locate the node with the largest degree (say node i) among the nodes in all the intersecting edges. After that, we find a pair of intersecting edges that involve node i, e.g., the edge between nodes i and j, and another between nodes k and l (see Figure 5(b), and compute their target edge vectors by:

$$\rho = l_{ij} \frac{\mathbf{x}_i - \mathbf{x}_j}{\parallel \mathbf{x}_i - \mathbf{x}_j \parallel} + l_{lk} \frac{\mathbf{x}_l - \mathbf{x}_k}{\parallel \mathbf{x}_l - \mathbf{x}_k \parallel} \; ;$$
$$\mathbf{d}'_{ij} = l_{ij} \frac{\rho}{\parallel \rho \parallel} \quad \text{and} \quad \mathbf{d}'_{lk} = l_{lk} \frac{\rho}{\parallel \rho \parallel} \; . \tag{14}$$

4.3 Shape-based Constraints

Based on our unified framework, it is possible to constrain the local structure of a graph, since the framework enables us to constrain edge directions in addition to edge lengths. Hence, we can formulate shape-based constraints in our framework to optimize the local structure of a sub-graph to follow certain reference shapes, so that the local structure around the sub-graph can be revealed more clearly.

Unlike direct and metrics-based constraints, the nodes to be updated are not directly provided but need to be determined based on the reference shape. Moreover, the reference shape has different forms and is provided in a different way for different constraints in this constraint class. Below, we first present the part of our method common to all the shape-based constraints and then give details to individual constraint.

Given a reference shape and a set of nodes $\{\mathbf{p}_i\}_{i=1}^{n_p}$ belonging to a sub-graph of interest, our goal is to re-arrange $\mathbf{p}_i$ to follow the reference shape. Hence, we first uniformly discretize the reference shape into n_q points, say $\{\mathbf{q}_j\}_{j=1}^{n_q}$, where $n_q \geq n_p$. Second, to help preserve the mental map of users, so that the graph layout receives the least amount of changes, we adopt the iterative closest point (ICP) matching model, and define an affine transformation $\mathbf{M}$ (with translation, rotation, and scaling) to transform $\{\mathbf{p}_i\}$. In each ICP iteration, we determine for each $\mathbf{p}_i$ a point on the reference shape that is the closest to $\mathbf{M}\mathbf{p}_i$, say $\mathbf{q}_i$, and then compute $\mathbf{M}$ that minimizes the following objective:

$$\sum_{i}^{n_p} \omega_i \parallel \mathbf{M}\mathbf{p}_i - \mathbf{q}_i \parallel^2 , \tag{15}$$

where ω_i is the weight of node $\mathbf{p}_i$. To emphasize the nodes that connect with many, we set ω_i as the degree of node $\mathbf{p}_i$, so that the major structure is more clearly revealed. We solve this matching problem by using the iterative closest point (ICP) algorithm [2], which is widely employed in point set registration. Once $\mathbf{M}$ is computed, we determine pairs of $\mathbf{p}_i$

and $\mathbf{p}_j$ whose mapped points $\mathbf{Mp}_i$ and $\mathbf{Mp}_j$ are adjacent to each other in the reference shape, and set the target edge vector ($\mathbf{d}'_{ij}$) between them as $\mathbf{M}(\mathbf{p}_i - \mathbf{p}_j)$ to follow the reference shape.

Circle Constraint. Circular structures are salient features in graph layouts, particularly for cases like biological pathways and transport networks. After the user selects a sub-graph of interest by lasso, our method looks for the largest cycle in the sub-graph and aims to move the nodes in the cycle to fit a circle (reference shape). Figure 6(a) shows an illustrative example, where we achieve a stable circular layout with six edge vector constraints. In contrast, Dwyer [11] requires a combination of more separation constraints to create a satisfactory layout, including also constraints between non-adjacent nodes; see Figure 6(b).

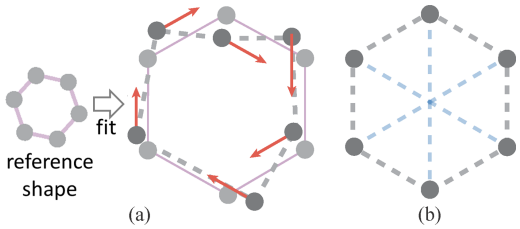


Fig. 6. Comparison the constraints to create a circular layout: (a) our method and (b) Dwyer [11]. Our method fits a target circle to a selected sub-graph and constrains only six target edge vectors (red arrows), while Dwyer [11] requires more constraints, including the separation between adjacent (grey) and non-adjacent (blue) nodes.

Star Constraint. Purchase [36] introduced an aesthetically-pleasing metric that spreads out edges around a node by maximizing the minimum angle in-between outgoing edges from the node. In this work, we develop a more efficient metric, the star constraint, which can be modeled in our unified framework by referencing a local substructure to a star shape. To do so, after the user selects a sub-graph, we find the nodes with the largest degree for the user to select. Then, we pick as the reference shape a star shape whose edge count matches the degree of the user-selected node (see Figure 7), and modifies the target edge vectors accordingly. Since ICP matches shapes with a rotation, the star shape can have varying orientation when mapped to the nodes with the least layout changes.

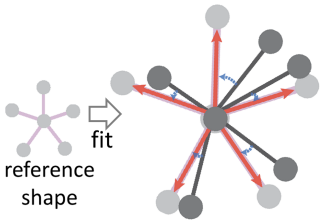


Fig. 7. Fit a star shape to a sub-graph (black) by manipulating the black edges in the sub-graph to follow the red target edge vectors.

Symmetry Constraint. Symmetry maximization is another way to enhance a graph's readability, especially by means of reflective symmetry, see [36]. Taking a symmetry axis and a sub-graph of interest (say G_s) specified by the user, we can regard the mirror image of G_s as a reference shape and apply it to the graph's substructure on the other side of the symmetry axis; see Figure 8(a). After determining the correspondence with ICP, each node i in G_s associates with a corresponding node i' in G'_s, which is a mirrored sub-graph. In G_s, for each pair of connecting nodes i and j, whose length is l_{ij} and unit vector is $\mathbf{u}_{ij}$, we can apply l_{ij} and $\mathbf{u'}_{ij}$ (which is the mirror of $\mathbf{u}_{ij}$ about the symmetry axis) to set the target edge vector between nodes i' and j' in G'_s, i.e., $\mathbf{d}'_{i'j'} = l_{ij} * \mathbf{u}'_{ij}$. Figure 8 shows an example, where the hexagon boxed by dashed lines is mapped to the sub-graph shown on the right hand side according to the vertical symmetry axis in the middle.

5 Results and Evaluation

We implemented and tested our method on a computer with an Intel Core i7 processor with 16GB memory using C++. Moreover, we developed a GPU implementation that runs on the NVidia GTX1080 graphics card with 8GB video memory using CUDA. The code is

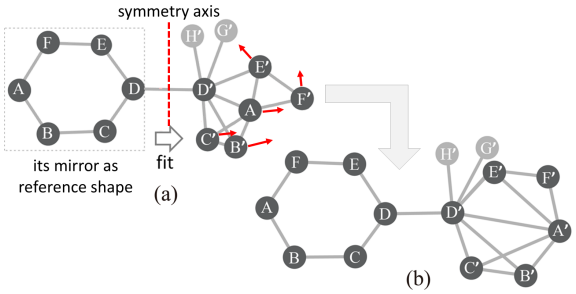


Fig. 8. Symmetry Constraint. (a) User specifies a symmetry axis (red dashed) and a sub-graph of interest (boxed); (b) we take the mirrored sub-graph as a reference shape to symmetrize the structure on the right.

available for download on GitHub[2]. To demonstrate that our method is able to produce fast and stable constrained graph layouts, we evaluated three main aspects. First, we quantitatively compare the quality of our results with two state-of-the-art constrained methods. Second, we demonstrate its usefulness in different applications and qualitatively compare it with related methods. Last, we present a constraint-based graph exploration interface that enables users to interactively explore a large graph of approx. 10K nodes and 90K edges.

5.1 Comparison with Previous Methods

We compare our approach with two closely related constrained graph layout methods: an incremental procedure for layouts with separation constraints (IPSep) [13] and a scalable and versatile constrained graph layout (SV) [11]. Both methods are closely related to this work, since both enforce separation constraints between selected pairs of nodes; both of them, however, cannot explicitly constrain edge directions. The main difference between them is that IPSep defines the separation in horizontal/vertical axis, while SV defines the Euclidean distance in an arbitrary direction. Namely, both methods support edge length and direction constraints, and SV supports even more, e.g., a circle constraint and non-overlap of convex cluster constraint, because of its flexibility. On the other hand, SV is faster than IPSep but its quality might be inferior to IPSep, because it incorporates constraints into the force-directed layout algorithm [31] instead of the stress model.

Accordingly, we designed two experiments: i) we compare our method with IPSep and SV for the downward constraint (Section 5.1.1), and ii) we compare our method with SV for the circle constraint and non-overlap constraint (Section 5.1.2). The first experiment involves a direct constraint, while the second involves a metrics-based constraint (non-overlap) and a shape-based constraint (circle). We use the code that was provided by the authors to perform the comparison. Following Dwyer et al. [13], we measure the layout quality using stress error (SE) as defined in Eq. (1). Moreover, we compute the degree of constraint satisfaction, which has different definitions for different constraints; we will outline the exact definitions below.

5.1.1 Comparison: Direct Constraints

In this comparison, we use two datasets: *Bus1138* with 1138 nodes and *US Power Grid* with 4941 nodes [9]. The former dataset was used to test downward constraints by IPSep and SV before. To measure the constraint satisfaction degree, we define vertical edges (VE) [4] :

$$V(\mathbf{X}) = \sum_{(i,j)\in E} \| < \frac{\mathbf{x}_i - \mathbf{x}_j}{||\mathbf{x}_i - \mathbf{x}_j||}, [1,0] > \| \,, \tag{16}$$

which measures the sum of absolute inner products to be between the edge vector and the positive X-axis. Like SE, a small $V(\mathbf{X})$ indicates that the constraints are better satisfied. Like Dwyer et al. [13], we also count the number of edge crossings (#EC).

Results. Table 1 lists the metric values for layouts of two different data sets generated by unconstrained layout (UC), IPSep, SV, and two versions of our method with different weights ($v_{ij} = 2$ and $v_{ij} = 4$).

[2] https://github.com/yanyan.wang/vector_stress_majorization

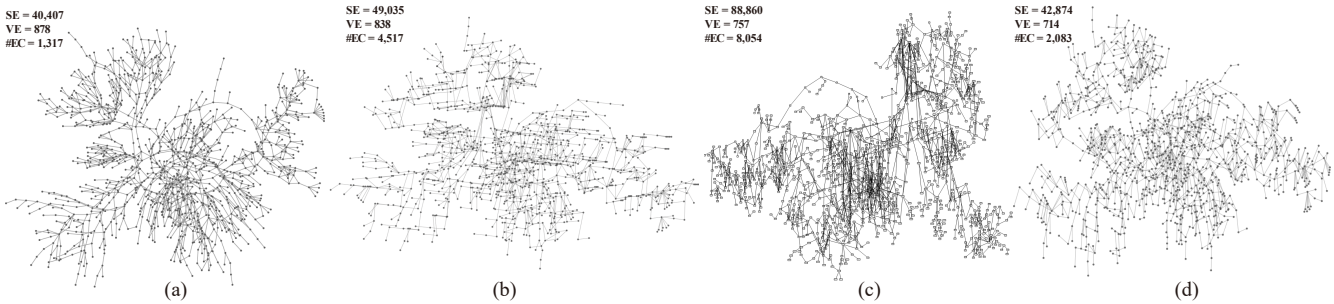


Fig. 9. Comparison of the layout of the *Bus1138* graph data with downward constraints generated by different methods: (a) unconstrained stress majorization (UC); (b) IPSep; (c) SV; and (d) our method with weight v_{ij} set to be 4.

Table 1. Comparing two versions of our method (with different weights, Ours-I: $v_{ij} = 2$ and Ours-II: $v_{ij} = 4$) with unconstrained layout (UC), IPSep, and SV. The comparison considers all real edges in two different graphs.

Graphs / Methods	*Bus1138*			*US Power Grid*		
	SE	VE	#EC	SE	VE	#EC
UC	**40,407**	878	**1,317**	**687,963**	3,907	13,462
Ours-I	42,874	714	2,083	698,140	3,457	13,599
Ours-II	46,241	**560**	2,484	714,519	**2,786**	**13,071**
IPSep	49,035	838	4,517	817,994	2,789	41,560
SV	88860	757	8,542	1,858,103	4,849	116,568

Table 2. Comparing our method (weight $v_{ij} = 3.5$ for all specified constraints) with unconstrained layout (UC) and SV on two graphs.

Graphs / Methods	*Bcspwr10*			*Psse1*	
	SE	#MN	ME	SE	#MN
UC	**6.71e5**	1,109	18.14	**1.42e7**	50,146
Ours	9.95e5	**134**	**1.95**	1.75e7	**13,378**
SV	1.14e11	831	14.82	1.92e10	49,679

We can see that our method produces the best quality, with stress values close to UC. Since the downward constraint changes the edge direction, it introduces more edge crossings for the *Bus1138* graph, but our method still produces fewer edge crossings than IPSep and SV, and it even leads to the smallest number of edge crossings for the *US Power Grid* data set. Regarding the constraint satisfaction degree VE, our method performs the best for the *Bus1138* graph, while it produces a VE similar to IPSep for the the *US Power Grid* graph. The results also confirm that SV is worse than IPSep and our method. Since the weight v_{ij} balances the stress constraint and user constraints, our version I with smaller weight leads to smaller vertical energy, while our version II with larger weight leads to smaller stress value.

Figure 9(a-d) presents the resulting layouts of the *Bus1138* data set generated by an unconstrained stress majorization (UC), IPSep, SV, and our method (with $v_{ij} = 4$), respectively. Comparing with UC shown in Figure 9(a), we can see that our method clearly displays the major structure with the least amount of edge crossings (see Figure 9(d)), while SV leads to visual clutter (see Figure 9(c)). Due to space limit, we only show the results of the *Bus1138* data set; for the results of the *US Power Grid* graph, please refer to the supplemental material.

5.1.2 Comparison: Metrics- and Shape-based Constraints

To study the quality of layouts generated with the non-overlap and circle constraints, we compare our method with SV [11], since IPSep cannot support these constraint classes. We use two large graphs: *Bcspwr10* (5300 nodes) and *Psse1* (14318 nodes), and generate clusters in them using the Snap library [32]. Moreover, we use two additional metrics to quantify the non-overlap and circle constraints: i) the number of mis-placed nodes (#MN) and ii) the shape matching error (ME). Mis-placed nodes refer to nodes located within the convex hull of some other classes, while for ME, we evaluate Eq. (15) for each circle and compute the average weighed by the circle area. For *Psse1*, since it does not have circle information, we only consider non-overlap constraints.

Results. Table 2 lists the metric values for layouts of *Bcspwr10* and *Psse1* generated by unconstrained layout (UC), SV, and our method (with $v_{ij} = 3.5$) for two graphs. Similar to the downward constraint results, the stress values produced by our method are quite close to those of UC, meaning that its layout is similar to that generated by the original stress majorization, while SV produces much larger stress errors (SE). Meanwhile, both the number of mis-placed nodes and shape matching energy for our method are much lower than others.

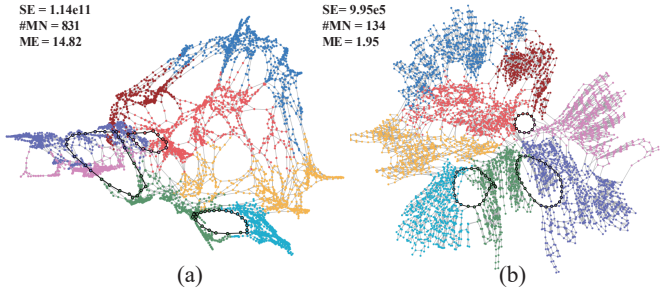


Fig. 10. Comparison of the layout of *Bcspwr10* with cluster non-overlap and circle constraints generated by SV (a) and our method with v_{ij} set to 3.5 (b).

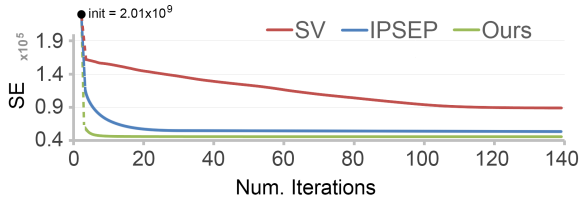


Fig. 11. Convergence of different constrained layout methods for the *Bus1138* graph. We plot stress error vs. number of iterations. Our method and IPSep both converge in less than ten iterations but our method has smaller stress errors, whereas SV requires 100 iterations to converge with still having a large stress error.

Figure 10(a,b) presents the layouts of the *Bcspwr10* graph data generated by SV and our method, respectively. We can see from the figures that our method almost separates all clusters. Furthermore, the three sub-graphs of interest are well arranged in a circular fashion, especially for the smallest one. In contrast, SV introduces more mis-placed nodes (#MN) and the three sub-graphs cannot really evolve to circular shapes. The layouts of the *Psse1* graph data show similar results, and can be found in the supplemental material.

5.1.3 Performance.

Table 3 summarizes the time taken to generate graph layouts of varying complexity, from 1138 to 14318 nodes. For *Bus1138* and *US Power Grid*, all edges are enforced with the downward constraints, so the number of constraints equals the edge count (m). In contrast, the non-overlap constraints for *Bcspwr10* and *Psse1* involve many node pairs among different clusters, so the number of constraints becomes really large. SV is faster than our CPU implementation but slower than our

Table 3. Statistics of our results: the data set size (n and m), number of constraints (c), and computational time (in seconds) for creating the layouts using the CPU and GPU versions of our method, IPSep [13], and SV [11]. DC: Downward Constraint; CN: Cluster Non-overlap Constraint; CC: Circle Constraint; ED: Edge Direction Constraint; and SC: Star Constraint.

Data sets	# nodes (n)	# edges (m)	# constraints (c)	constraint type	Ours(CPU)	Ours(GPU)	SV	IPSep
Bus1138	1138	1458	1458	DC	0.83	0.51	0.43	32.49
USPowerGrid	4941	6594	6594	DC	8.43	1.79	0.53	320.07
Facebook4039	4039	88234	6947528	CN+ED+CC+SC	11.84	2.36	–	–
Bcspwr10	5300	8271	6805586	CN+CC	13.60	3.93	11.89	–
Psse1	14318	57366	79297968	CN	261.2	122.753	150.77	–

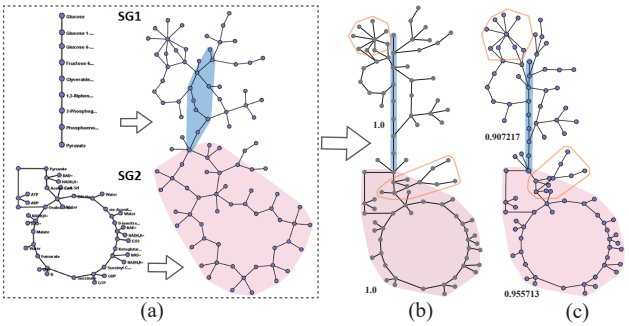


Fig. 12. Comparing our method with Yuan et al. [51] for merging graphs. (a) highlighting the subgraphs shown in pink and blue regions on a graph layout with reference shapes SG1 and SG2; (b) The result produced by our method; and (c) the result produced by Yuan et al. [51]. The numbers shown in (c) indicate the similarity scores between the sub-graph and its reference shape, the structures highlighted in the orange boxes can be more clearly revealed in (b) than in (c).

Fig. 13. The user interface of our constraint-based graph exploration system includes a button menu for constraints selection (top-right, boxed in purple), the overview of the entire graph (boxed in yellow), detail view (boxed in red), constraints weight controlling view (boxed in green), and detail information view (boxed in blue).

GPU implementation for large graph data.

We also compared the convergence of the stress error of the three methods, as summarized in Figure 11. Both our method and IPSep, converge to similar stress errors and run more rapidly than SV, while our method produces the smallest stress error.

In summary, we see that our method runs at comparable speed for small graphs like SV [11], but it is still able to produce higher quality results, as shown in Table 1. For large graphs with non-overlap constraints, the cost of transferring $\mathbf{L}'$ and $\mathbf{J}'$ is too expensive, so that the GPU performance is not as good as we expected; however, it still improves over the CPU version and runs faster than SV, since SV requires a larger amount of constraints than our method. For the graph *Psse1*, SV and our method run out of system memory. On the other hand, during the interactive exploration, only the newly added constraints needed to be updated for $\mathbf{L}'$ and $\mathbf{J}'$, so the performance becomes acceptable for graph exploration.

5.2 Comparison with Applications

Here we show that our direct constraints can be used for merging small graph layouts and incremental layouts for dynamic graph visualization, since target edge vectors of specific edges have been explicitly given, so we can directly treat them as constraints in our framework. Moreover, for each application we compare to previous related works.

Merging Subgraphs. Yuan et al. [51] propose to combine many user's input sub-graphs into a consistent layout that maintains the topological information of the individual input layouts. They developed an algorithm, Laplacian constrained distance embedding, that attempts to preserve the Euclidean distance between the nodes of the input sub-graph layouts and the nodes of an initial layout of the whole graph. Since the nodes correspondence between the input sub-graph and the whole graph is already known, the edge of the input sub-graph layout can be taken to form the target edge vectors of the associated nodes, and thus our method can also be used for this purpose.

As shown in Fig 12(a), the user requires the subgraph shown in the blue region to be aligned with a straight line SG1, and the subgraph in the pink region to be matched up with the reference shape SG2. Our method automatically creates the constrained graph layout shown in Figure 12(b), where we measure the layout similarity in terms of the procrustes statistic [7]. Such measure is 1.0 between the sub-graphs in our case, while those generated by Yuan et al. [51] are 0.95 and 0.91. Moreover, we can see that the sub-structures highlighted by the orange boxes can be shown more clearly with fewer edge crossings and node overlapping. We assume the reason is that their method preserves the Euclidean distance between nodes of a given unconstrained graph layout rather than the graph distance, while our method consistently combines the stress (distance) constraints and user constraints together.

Visualizing Dynamic Graphs. Our temporal coherence constraint naturally supports time-varying visualizations of dynamic graphs. To demonstrate the effectiveness of our method, we first compare it with the Laplacian-based dynamic graph layout algorithm [6], which has been demonstrated to perform better than the online dynamic graph drawing algorithm [20]. Figure 14 shows the results, where our layout is more stable for the Newcomb's fraternity data [35], For example, the shape of the green and pink sub-graphs in our method (a) are almost kept constant throughout the whole sequence. Using the state-of-the-art approach (b), these areas re-shape substantially at T1 and T3. We further apply our method to large dynamic graphs, the results of which can be found in the supplemental material.

5.3 Constraint-based Graph Exploration

In this section, we demonstrate the usefulness of our method in exploring large graphs. Figure 13 shows a screenshot of our interface, which consists of five main components: constraint selection view (upper-left), main view (left), detail view (upper-right), weight controlling view (middle-right), and finally a detailed information view providing additional information about sub-graph of interest. Once the user selects a constraint type, the associated constraints will be imposed to the entire graph or to the subgraph of interest selected by a lasso, while the sub-graph of interest can be highlighted in the detail view. The user can also assign different weights to individual constraints.

Using this setup, users can interactively enforce constraints to explore clusters, paths, circles, and nodes. The clusters can be separated

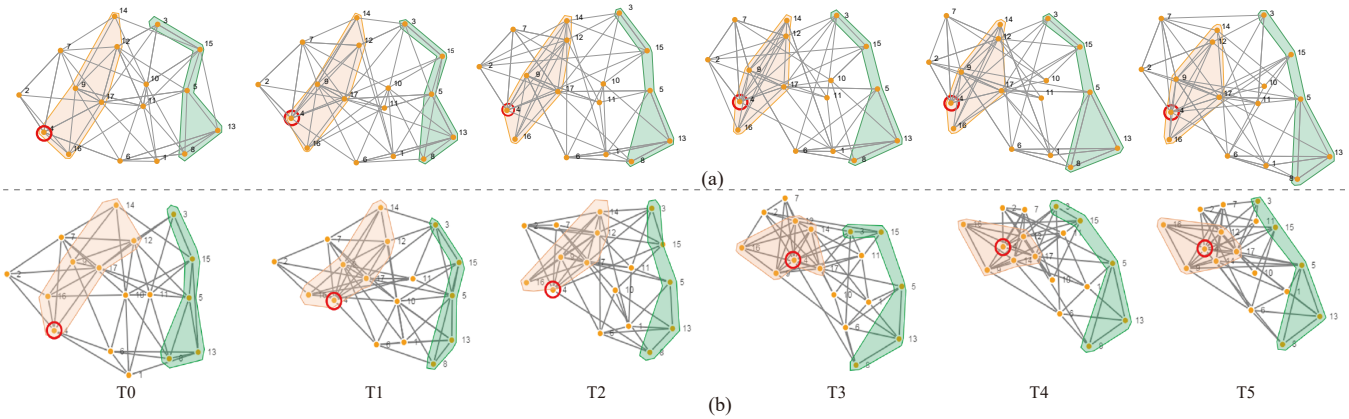


Fig. 14. Comparison of the layout results generated by (a) our method and (b) the Laplacian-based dynamic graph layouting algorithm [6]. The sub-graphs shown in pink and green regions keep the shape during the whole sequence in (a), but change substantially in (b). Node 4 is highlighted with a red circle, revealing that its position is kept stable in our method but changes in the other method.

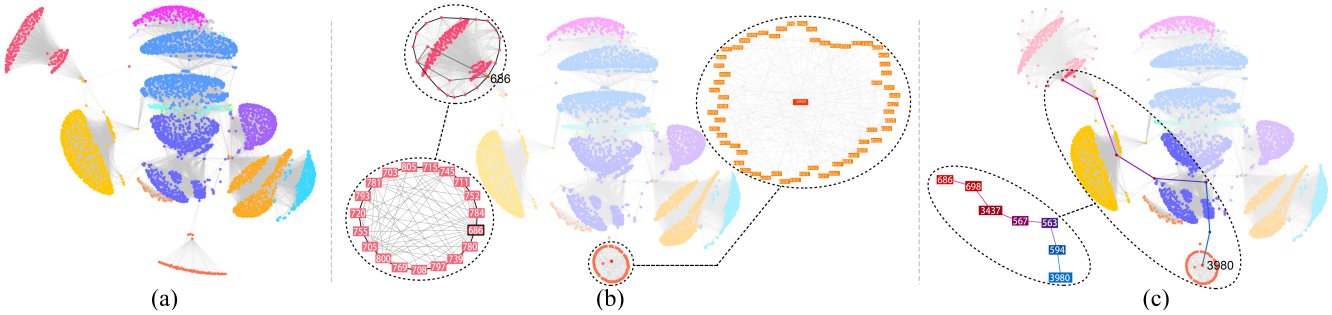


Fig. 15. Exploration of circles in the Ego-network. (a) results generated by using cluster non-overlap constraints, where 16 communities are separated; (b) results generated by enforcing circle and star shapes to two selected nodes: 686 and 3980, the highlighted sub-graphs are shown in the detail views. (c) results generated by enforcing path constraints to these two nodes, the detailed node label information is shown in the detail view.

using the non-overlap constraint, while circles and nodes can be highlighted by enforcing circular and star-like shapes. To help the users to intuitively perceive a path, we introduce a path constraint that orients edges using the weighted average of all edge vectors along a path. This allows us to straighten the path while imposing only small changes to the whole graph layout.

Exploration of Ego-Network. Here, we use the data set *ego-Facebook* [32] to demonstrate the effectiveness of our method for graph exploration. This graph has 4039 nodes, 88234 edges, and 4037 different, hand-labeled circles, which represent a categorization of Facebook friends. Since each circle is not only densely-connected but also might be nested in large ones, it is very hard to see their structures.

To quickly get an overview of this ego-network, we suggest that the user explores the relationship between different communities by enforcing cluster non-overlap constraints. Figure 15(a) shows the result, where 16 communities are separated, and different communities have different structures. To compare the different communities, the user selects the central nodes (pink and orange) and applies circle and star shape constraints to them. Next, the user obtains the results shown in Figure 15(b), where the pink node is involved in two nested circles, while the orange node has very dense connections. It appears that these two nodes have very different structures. To investigate how far they are from each other, we enforce a path constraint to them, the corresponding path is shown in Figure 15(c). We see that there are five nodes with different labels between them. The user can further explore the nodes on this path, for instance, to understand the structure of the other communities.

6 Conclusions and Future Work

In this work, we revisit stress majorization and reformulate it in terms of a vector form. By this means, we develop a unified framework that enables us to model various forms of constraints: direct constraints, metric-based constraints, and shape-based constraints. These constraints can all be solved efficiently by a GPU-accelerated stress majorization method. Through a set of quantitative comparisons with state-of-the-art constrained layout methods, we show that our approach outperforms others in producing graph layouts with lower stress while satisfying the constraints. Meanwhile, we demonstrated the usefulness of our method with the applications of merging user's input sub-graph layout and temporal coherent dynamic graph visualization. Finally, we developed a system for interactive exploration of large graphs using overview+detail with constraint based layouts, and demonstrated its effectiveness by using a large social network data.

Our formulation of stress majorization opens various new opportunities for graph visualization. First, we may develop more constraints, especially shape-based constraints, where the user can fit a specific shape to a given graph layout for aesthetics and data analysis. Second, the weights of the constraints are currently manually set by users, so finding a way to automatically set proper weights to fuse multiple constraints is part of our future work. Third, our current GPU implementation is still not sufficiently fast to support interactive exploration of very large graphs. In the future, we plan to further improve its performance by optimizing the data transfer between CPU and GPU. Last, we plan to explore different applications of our improved stress majorization, such as constrained multidimensional scaling (MDS) for high-dimensional data visualization [7].

Acknowledgments

The authors would like to thank Zeyu Wang and Tong Ge for their help in making the video, and the anonymous reviewers for the valuable comments. This work is supported by the grants of NSFC-Guangdong Joint Fund (U1501255), NSFC (61379091), the National Key Research & Development Plan of China (2016YFB1001404), National Foreign

1000 Talent Plan (WQ201344000169), Leading Talents of Guangdong Program (00201509), Shandong Provincial Natural Science Foundation (11150005201602) and the Fundamental Research Funds of Shandong University.

REFERENCES

[1] D. Archambault, H. Purchase, and B. Pinaud. Animation, small multiples, and the effect of mental map preservation in dynamic graphs. *IEEE Trans. Vis. & Comp. Graphics*, 17(4):539–552, 2011.

[2] P. Besl and N. Mckay. A method for registration of 3-D shapes. *IEEE Trans. Pat. Ana. & Mach. Int.*, 14(2):239–256, 1992.

[3] K.-F. Böhringer and F. N. Paulisch. Using constraints to achieve stability in automatic graph layout algorithms. In *ACM CHI*, pages 43–51, 1990.

[4] S. Bridgeman and R. Tamassia. Difference metrics for interactive orthogonal graph drawing algorithms. In *International Symposium on Graph Drawing*, pages 57–71, 1998.

[5] L. Carmel, D. Harel, and Y. Koren. Combining hierarchy and energy for drawing directed graphs. *IEEE Trans. Vis. & Comp. Graphics*, 10(1):46–57, 2004.

[6] L. Che, J. Liang, X. Yuan, J. Shen, J. Xu, and Y. Li. Laplacian-based dynamic graph visualization. In *Proceedings of the IEEE Pacific Visualization Symposium*, volume 00, pages 69–73, 2015.

[7] T. F. Cox and M. A. Cox. *Multidimensional scaling*. CRC press, 2000.

[8] T. Crnovrsanin, J. Chu, and K.-L. Ma. An incremental layout method for visualizing online dynamic graphs. In *International Symposium on Graph Drawing and Network Visualization*, pages 16–29, 2015.

[9] T. A. Davis and Y. Hu. The University of Florida sparse matrix collection. *ACM Transactions on Mathematical Software*, 38(1):1, 2011.

[10] D. Dobkin, J. Hershberger, D. Kirkpatrick, and S. Suri. Computing the intersection-depth of polyhedra. *Algorithmica*, 9(6):518–533, 1993.

[11] T. Dwyer. Scalable, versatile and simple constrained graph layout. *Computer Graphics Forum*, 28(3):991–998, 2009.

[12] T. Dwyer and Y. Koren. Dig-CoLa: directed graph layout through constrained energy minimization. In *Proceedings of the IEEE Information Visualization Symposium*, pages 65–72, 2005.

[13] T. Dwyer, Y. Koren, and K. Marriott. IPSep-CoLa: An incremental procedure for separation constraint layout of graphs. *IEEE Trans. Vis. & Comp. Graphics*, 12(5):821–828, 2006.

[14] T. Dwyer, B. Lee, D. Fisher, K. I. Quinn, P. Isenberg, G. Robertson, and C. North. A comparison of user-generated and automatic graph layouts. *IEEE Trans. Vis. & Comp. Graphics*, 15(6):961–968, 2009.

[15] T. Dwyer, K. Marriott, F. Schreiber, P. Stuckey, M. Woodward, and M. Wybrow. Exploration of networks using overview+detail with constraint-based cooperative layout. *IEEE Trans. Vis. & Comp. Graphics*, 14(6):1293–1300, 2008.

[16] T. Dwyer, K. Marriott, and M. Wybrow. Dunnart: A constraint-based network diagram authoring tool. In *International Symposium on Graph Drawing*, pages 420–431, 2008.

[17] T. Dwyer, K. Marriott, and M. Wybrow. Topology preserving constrained graph layout. In *International Symposium on Graph Drawing*, pages 230–241, 2008.

[18] P. Eades. A heuristic for graph drawing. *Congressus numerantium*, 42:146–160, 1984.

[19] P. Eades, R. F. Cohen, and M. L. Huang. Online animated graph drawing for web navigation. In *International Symposium on Graph Drawing*, pages 330–335, 1997.

[20] Y. Frishman and A. Tal. Online dynamic graph drawing. *IEEE Trans. Vis. & Comp. Graphics*, 14(4):727–740, 2008.

[21] T. M. Fruchterman and E. M. Reingold. Graph drawing by force-directed placement. *Software: Practice and experience*, 21(11):1129–1164, 1991.

[22] E. R. Gansner, Y. Koren, and S. North. Graph drawing by stress majorization. In *International Symposium on Graph Drawing*, pages 239–250, 2004.

[23] E. R. Gansner, Y. Koren, and S. C. North. Topological fisheye views for visualizing large graphs. *IEEE Trans. Vis. & Comp. Graphics*, 11(4):457–468, 2005.

[24] E. R. Gansner, E. Koutsofios, S. C. North, and K.-P. Vo. A technique for drawing directed graphs. *IEEE Trans. on Software Engineering*, 19(3):214–230, 1993.

[25] W. He and K. Marriott. Constrained graph layout. In *International Symposium on Graph Drawing*, pages 217–232, 1996.

[26] M. L. Huang and P. Eades. A fully animated interactive system for clustering and navigating huge graphs. In *International Symposium on Graph Drawing*, pages 374–383, 1998.

[27] M. Jünger and P. Mutzel. Exact and heuristic algorithms for 2-layer straightline crossing minimization. In *International Symposium on Graph Drawing*, pages 337–348, 1995.

[28] T. Kamada and S. Kawai. An algorithm for drawing general undirected graphs. *Information processing letters*, 31(1):7–15, 1989.

[29] J. B. Kruskal. Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis. *Psychometrika*, 29(1):1–27, 1964.

[30] J. Lamping, R. Rao, and P. Pirolli. A focus+context technique based on hyperbolic geometry for visualizing large hierarchies. In *ACM CHI*, pages 401–408, 1995.

[31] U. Lauther. Multipole-based force approximation revisited – a simple but fast implementation using a dynamized enclosing-circle-enhanced kd-tree. In *International Symposium on Graph Drawing*, pages 20–29, 2006.

[32] J. Leskovec and R. Sosič. Snap: A general-purpose network analysis and graph-mining library. *ACM Trans. on Intelligent Systems and Technology*, 8(1):1, 2016.

[33] C. D. Meyer. *Matrix analysis and applied linear algebra*, volume 2. Siam, 2000.

[34] K. Misue, P. Eades, W. Lai, and K. Sugiyama. Layout adjustment and the mental map. *Journal of Visual Languages & Computing*, 6(2):183–210, 1995.

[35] T. M. Newcomb. The acquaintance process: Looking mainly backward. *Journal of Personality and Social Psychology*, 36(10):1075, 1978.

[36] H. C. Purchase. Metrics for graph drawing aesthetics. *Journal of Visual Languages & Computing*, 13(5):501–516, 2002.

[37] H. C. Purchase, C. Pilcher, and B. Plimmer. Graph drawing aesthetics created by users, not algorithms. *IEEE Trans. Vis. & Comp. Graphics*, 18(1):81–92, 2012.

[38] K. Rupp, F. Rudolf, and J. Weinbub. ViennaCL - a high level linear algebra library for GPUs and multi-core CPUs. In *Intl. Workshop on GPUs and Scientific Applications*, pages 51–56, 2010.

[39] K. Ryall, J. Marks, and S. Shieber. An interactive constraint-based system for drawing graphs. In *Proceedings of the 10th annual ACM symposium on User interface software and technology*, pages 97–104, 1997.

[40] P. Simonetto, D. Archambault, D. Auber, and R. Bourqui. ImPrEd: An improved force-directed algorithm that prevents nodes from crossing edges. *Computer Graphics Forum*, 30(3):1071–1080, 2011.

[41] M. E. Smoot, K. Ono, J. Ruscheinski, P.-L. Wang, and T. Ideker. Cytoscape 2.8: new features for data integration and network visualization. *Bioinformatics*, 27(3):431–432, 2011.

[42] O. Sorkine and M. Alexa. As-rigid-as-possible surface modeling. In *Symp. on Geom. Proc.*, volume 4, 2007.

[43] K. Sugiyama, S. Tagawa, and M. Toda. Methods for visual understanding of hierarchical system structures. *IEEE Trans. on Systems, Man, and Cybernetics*, 11(2):109–125, 1981.

[44] R. Tamassia. *Handbook of graph drawing and visualization*. CRC press, 2013.

[45] C. Tominski, J. Abello, F. Van Ham, and H. Schumann. Fisheye tree views and lenses for graph visualization. In *International Conference on Information Visualization*, pages 17–24, 2006.

[46] Touchgraph Navigator. TouchGraph, LLC, 2009. `www.touchgraph.com`.

[47] F. Van Ham and A. Perer. Search, show context, expand on demand: supporting large graph exploration with degree-of-interest. *IEEE Trans. Vis. & Comp. Graphics*, 15(6), 2009.

[48] T. Von Landesberger, A. Kuijper, T. Schreck, J. Kohlhammer, J. J. van Wijk, J.-D. Fekete, and D. W. Fellner. Visual analysis of large graphs: state-of-the-art and future research challenges. *Computer Graphics Forum*, 30(6):1719–1749, 2011.

[49] C. Walshaw. A multilevel algorithm for force-directed graph-drawing. *Journal of Graph Algorithms and Applications*, 7(3):253–285, 2006.

[50] X. Wang and I. Miyamoto. Generating customized layouts. In *International Symposium on Graph Drawing*, pages 504–515, 1995.

[51] X. Yuan, L. Che, Y. Hu, and X. Zhang. Intelligent graph layout using many users' input. *IEEE Trans. Vis. & Comp. Graphics*, 18(12):2699–2708, 2012.