UNICON: A UNIform CONstraint Based Graph Layout Framework

Jiacheng Yu[2,3*] Yifan Hu[6†] Xiaoru Yuan[1,2,4,5‡]

1) Key Laboratory of Machine Perception (Minstry of Education), and School of AI, Peking University, Beijing, China
2) National Engineering Laboratory for Big Data Analysis and Application, Peking University, Beijing, China
3) Center for Data Science, Academy for Advanced Interdisciplinary Studies, Peking University, Beijing, China
4) National Institute of Health Data Science at PKU, Health Data Visualization and Visual Analytics Research Center, Beijing, China
5) Beijing Engineering Technology Research Center of Virtual Simulation and Visualization, Peking University, Beijing, China
6) Yahoo Research, New York, United States

ABSTRACT

We propose UNICON, a UNIform CONstraint based graph layout framework that supports both soft and hard constraints. We extend the stress model to accommodate soft constraints by incorporating them in the objective functions, optimized by stochastic gradient descent. For hard constraints, such as inequalities or equalities in the layout space, we utilize a gradient projection method to satisfy them. A visualization prototype system is implemented based on this framework for the user to interactively add or remove constraints to generate the desired layouts. We demonstrate the efficiency, quality, and flexibility of the framework and the system on a number of datasets with a wide range of user-defined constraints.

Keywords: Graph visualization, stress model, constraints, stochastic gradient descent

1 INTRODUCTION

Graphs are used to represent a set of entities and their relationships, such as social networks, citation networks, and traffic connections. Graph drawing algorithms are used to generate highly readable graph layouts. Kamada and Kawai [29] introduced a simple and effective algorithm based on stress model to place nodes in a graph according to the graph-theoretic distance between them to obtain an aesthetic visual representation of the graph. However, aesthetics encompass multiple aspects that are often difficult to satisfy with one model. The generated layouts may not be user-satisfying when only considering these distance constraints. Careful considerations of application-specific layout constraints [11], such as directed edges, fixed position, non-overlap of clusters, etc., generate the user's desired layout.

Various user-defined constraints and constrained graph layout methods have been proposed in the past decades. Wang et al. [46] revisited the stress majorization model and reformulated it with edge vectors. Wang's framework can only handle constraints that edge vectors can model. They used the conjugate gradient method to solve the problem faster than the previous methods. However, Zheng et al. [48] used stochastic gradient descent to solve the optimization problem, moving a single pair of vertices at a time, converging faster than the conjugate gradient method, and generating constrained layouts more quickly. With the gradient descent method, Ahmed et al. [1] modeled the aesthetic criteria as objective functions and optimized them to generate graph layouts with high readability. However, these existing methods support only a few constraints in a specific domain.

*e-mail: jiachengyu@pku.edu.cn
†e-mail: yifanh@gmail.com
‡e-mail: xiaoru.yuan@pku.edu.cn (corresponding author)

To define and model constraints efficiently and quickly, we propose UNICON, a uniform constraint based graph layout framework. We consider that constraints can be divided into global constraints and local constraints. For local constraints, they are usually defined on one node (position constraint), two nodes (distance constraint and direction constraint), or three nodes (angle constraint). These are the most fundamental cases of constraints, and most other constraints can be transformed into them. Thus, we define these as the basic constraints and other constraints as advanced constraints. For example, the constraints defined in the stress model [29] are applied globally over the graph, but it can be translated into a distance constraint between each pair of nodes in the graph.

In UNICON, we further divide constraints into soft and hard constraints. The soft constraints are approximately satisfied, while the hard constraints are strictly satisfied. We model each soft basic constraint as an objective function and optimize the objective functions with stochastic gradient descent. For the soft advanced constraints proposed by the users, we first transform them into soft basic constraints so that they can be modeled and solved directly and quickly using UNICON. For hard constraints, such as equalities or inequalities in the layout space, we can not model them using the objective function. Therefore, we use the gradient projection approach to satisfy the hard constraints by moving the nodes as little as possible. The direction and distance of movement are defined by the projection induced by the constraint. And the gradient projection approach is integrated into the flow of stochastic gradient descent. For a user-given constraint, we first transform it into basic constraints, model them as objective functions or projections of constraints, and then add them to a list of constraints. In the optimization step, one constraint at a time is randomly selected for optimization by gradient descent or projection. Based on this stochastic gradient descent and projection approach, our framework can efficiently handle both soft and hard constraints to generate high-quality graph layouts that meet different user expectations. We further design an interactive system prototype to demonstrate the effectiveness of UNICON.

The main contributions of our work are summarized as follows. First, we established a taxonomy for constraints in constrained graph visualization based on the user-defined constraints proposed in the previous works. Second, we proposed UNICON, a uniform constraint based graph layout framework that supports any soft constraints that can be modeled by objective functions as well as hard constraints that can be modeled by equalities or inequalities. Third, we built an efficient toolkit of the constrained graph layout algorithm based on the proposed framework and an interactive system prototype for creating and editing constraints to generate the layout that the user expects.

2 RELATED WORK

In this section, we summarize the related works of this paper, which include graph layout methods and constrained graph visualization.

2.1 Graph Layout Methods

Many graph layout methods [43] have been proposed to generate high-quality representations of node-link diagrams. Gibson et al. [20] categorized the graph layout methods into three approaches: force-directed methods [16, 17, 29], dimension reduction methods [3, 23, 32] and multi-level methods [22, 26]. With the emergence and development of deep learning techniques, applications of deep learning in graph layout appeared [34, 45].

Eades [16] first introduced the spring-electrical model, which is the basis for all force-directed methods. Fruchterman and Reingold [17] were inspired by Eades' work and then proposed the spring-embedder algorithm. Further, Kamada and Kawai [29] treated the graph layout problem as an optimization problem of stress model, and the conjugate gradient method [18] was used to solve the problem faster. Zheng et al. [48] used the stochastic gradient descent method to move one pair of nodes at a time, which converges faster and more consistently than the conjugate gradient method. Dimensionality reduction methods first embed the nodes into high-dimensional vectors and then project them onto a two-dimensional plane. These methods include multidimensional scaling [3], linear dimension reduction [23] and t-SNE [32]. Multi-level methods [22,26] were proposed for high computational scalability while maintaining high quality when dealing with large graphs. Kwon and Ma [34] used an encoder-decoder framework with graph neural networks to systematically visualize a graph in diverse layouts. Our framework extends the stress model [29] to accommodate constraints by incorporating them in the objective functions.

2.2 Constrained Graph Visualization

Based on the above graph layout methods, especially the force-directed layout methods, many works [2, 30, 44] further took into account user-defined constraints and integrated constraint solvers to generate user-desired graph layouts.

Earlier works mainly focused on drawing hierarchical layouts. Sugiyama et al. [42] considered vertical and horizontal positions separately, first determining which layer each node is on, i.e., the discrete space on the y-axis, and then deciding the horizontal positions of nodes to minimize edges crossings. The quadratic programming method [24] was applied to solve the constraints, and some heuristic methods [19, 28] were proposed to reduce the computational cost. Later works [4, 10] considered nodes with continuous y-coordinates and formed hierarchical constraints in the form of energy functions.

In order to satisfy various user considerations, different constraints have been proposed to generate user-desired layouts. Existing layout algorithms often assume that nodes are ideal points, but when nodes have a certain radius, they may cause node overlapping problems. Wang and Miyamoto [44] discouraged node overlapping by setting the attractive force between nodes to zero and increasing the repulsive force when two nodes overlap. Marriott [35] modeled node overlapping constraints as objective functions and used constrained optimization to eliminate node overlapping in graph layout. ImPrEd [39] proposed by Simonetto et al. improved the force-directed algorithm to prevent nodes from crossing edges. Huang and Eades [27] improved the clustering effect of the force-directed layout by adding a virtual spring between the nodes and the virtual center of its cluster. Ko and Yen [31] later formulated clustered drawing using the stress model. Dwyer et al. [11] extended force-directed layout algorithms to support separation constraints, including directed edges, alignment or distribution, fixed position, and other constraints mentioned above.

Dwyer et al. [14] took an initial feasible layout and improved it while retaining its topology. Yuan et al. [47] merged many sub-graphs of the large graph while maintaining the topological information of each sub-graph by Laplacian constrained distance embedding. The topology-preserving method [5] was applied to dynamic graph visualization to preserve a coherent mental map of the changing graphs. High-Dimensional Layout Stitching [41] was applied to connect small layout patches to generate the final layout for dynamic views of large static graphs. By revisiting the stress model, Wang et al. [46] reconstructed the model with edge vectors and proposed a uniform framework that allows modeling constraints as edge vectors and solving the problems by conjugate gradient descent. Readability metrics [8,33,38] can also serve as constraints. Ahmed et al. [1] used gradient descent to optimize readability criteria modeled as objective functions. Devkota et al. [7] modeled edge crossings, crossing angle, and constraints for preserving upwardness as objective functions and used gradient descent to optimize stress plus those terms. However, these existing methods support only certain types of constraints. For example, Wang et al. [46] only handled constraints that edge vectors can model.

The above methods can handle soft constraints that can be translated into objective functions but can not handle hard constraints. Dwyer et al. [9] extended the constraints to inequalities or equalities in the Euclidean space, which are also known as hard constraints, which can not be satisfied by the previous layout method. Gradient projection method [12] was proposed to handle these hard constraints by moving the minimum distance of nodes to satisfy the inequalities or equalities after each iteration of the layout method. Procrustes projection method [15] extended the gradient projection to deal with non-linear constraints. Our method uses the similar method of Dwyer et al. [9] to deal with hard constraints. However, we use stochastic gradient descent to optimize the stress functions and objective functions for modeling user-specific constraints, making our framework more flexible and efficient.

For user-friendly specification of constraints, Guchev and Gena [21] presented a simple and intuitive gesture-friendly interface with sketching techniques for user-guided refinement of the force-directed graph layout. For domain-specific graph data, high-level constraints [25] are proposed to reduce specification effort. A Crowd sourcing system [40] provided design guidelines and editing tools to help novice workers perform like experts in biological network visualizations. For large graphs, exemplar-based fine-tuning techniques [37] took in user modifications on sub-graphs and transferred the constraints to other similar substructures.

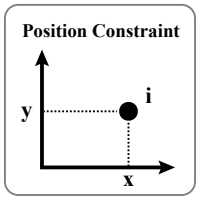
3 User-defined Constraints

In UNICON, users can define various constraints to achieve desirable layouts. These constraints can be defined on the graph at the global level or the local level. At the global level, constraints are applied to all nodes and/or edges in the graph. For example, a user wants the layout to look like a subway diagram, which requires all edges in the graph to satisfy certain conditions. The local level means the constraints are relevant to only a subset of nodes in the graph.

We define and illustrate some basic constraints and explain how advanced constraints can be decomposed to basic constraints. Depending on whether the constraints could be optimized by incorporating them as objective functions or must be satisfied as inequalities or equalities, we classify the constraints into soft constraints and hard constraints.

3.1 Basic Constraints

The constraints defined at the global level could be converted into constraints on the local level. Particularly, constraints that only involve one node, two nodes, or three nodes are more basic and serve as the cornerstone of advanced constraints. We define basic constraints as follows:



Position Constraint is defined on one node at a time, but it is common to specify the positions of multiple nodes. In the case of soft position constraint, the desired position of node i is (x, y), or it can be defined on the

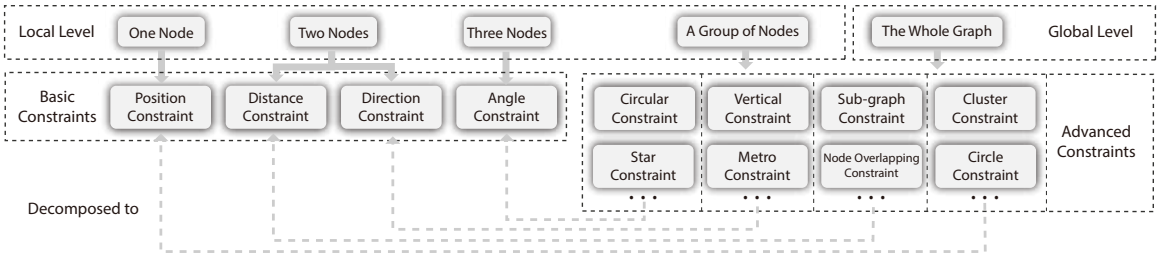
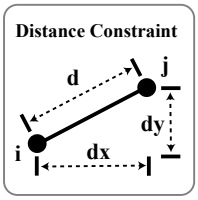


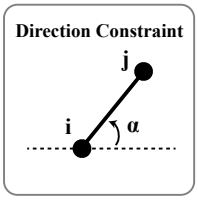
Figure 1: Taxonomy of Constraints. The user-defined constraint are first defined into global level and local level. The constraints on the local level are further divided according to the number of nodes they are defined on. The advanced constraints can be decomposed to basic constraints.

x-axis or y-axis, respectively. In the case of hard position constraint, it can be defined as either a fixed position constraint or a boundary constraint. The fixed position constraint indicates the node must be at position (x, y). The boundary constraint indicates that the position of node i on the x-axis (or y-axis) must be greater or less than x (or y), or node's position must be on a circle of radius r centered on the position.

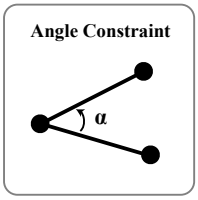
Distance Constraint is defined on two nodes which do not necessarily have to be connected by an edge. In the case of soft distance constraint, the ideal distance between two nodes i, j is d_{ij}. In the case of hard distance constraint, it can be defined as either a fixed distance constraint or a distance boundary constraint. The fixed distance constraint indicates that the distance between node i and j must be d_{ij}. The distance boundary constraint indicates that the distance between node i and j must be greater or less than d_{ij}. A distance constraint can also be defined on the x-axis or y-axis.



Direction Constraint is defined on two nodes. In the case of soft direction constraint, the ideal angle of the edge between node i and j with respect to the x-axis is α. In the case of hard direction constraint, it can be defined as either a fixed direction constraint or a direction boundary constraint. The fixed direction constraint indicates the angle of the edge between node i and j with respect to the x-axis must be equal to α. The direction boundary constraint indicates that the angle of the edge between node i and j with respect to the x-axis must be greater or less than α.



Angle Constraint is defined on three nodes. These three nodes are connected by two edges which form an angle. In the case of soft direction constraint, the ideal angle is α. In the case of hard angle constraint, it can be defined as either a fixed angle constraint or a angle boundary constraint. The fixed angle constraint means the angle must be α. The angle boundary constraint indicates that the angle must be greater or less than α.



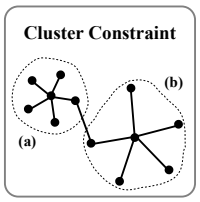
3.2 Advanced Constraints

In addition to the basic constraints defined above, many other constraints are commonly used to satisfy various application scenarios. These advanced constraints are more complex than the basic constraints. However, they can be expressed in terms of basic constraints, some can even be decomposed in different ways. We illustrate some advanced constraints and explain how they can be converted to different basic constraints.

3.2.1 Position-based Constraints

Some advanced constraints are relative to node positions, and we can convert them to position constraints.

Cluster Constraint. Node spread [8] is a heuristic to measure the node dispersion, modeling the distance of each node from the center of its cluster. A smaller node spread indicates compact drawing within the cluster. So for each node in the cluster, we want it to be close to the center of its cluster. Therefore, we can model cluster constraint by using the position constraint on each node with its cluster's pseudo center position.



Circle Constraint. The circle is an aesthetically pleasing shape. The circle constraint forms the nodes over a circle. That is, each node only needs to be at the same distance from the center position. That is a position constraint, where the center position and the radius of the circle are defined by the user.

3.2.2 Distance-based Constraints

Some advanced constraints can be converted to distance constraints. The following three distance-based constraints are commonly used in constrained graph layout.

Sub-graph Constraint. Maintaining the topology of the given initial layout helps to preserve the mental map of dynamic graph visualization [5], as well as merging sub-graphs of many users [47]. The sub-graph constraint aims to preserve the topology of the given sub-graph by maintaining the relative distance between each pair of nodes in the sub-graph. Thus, it can be represented in terms of distance constraints. It can be a soft or hard constraint defined by user specifications for various application scenarios. When using distance constraints to represent sub-graph constraint, the sub-graph is free to rotate.

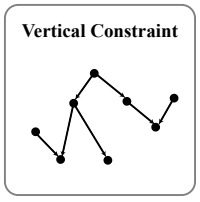
Edge Length Variation Constraint. Uniform edge length is an effective criterion when measuring the quality of a graph layout [33]. Edge length variation constraint wants a uniform edge length. We can add a distance constraint for each edge in the graph with an ideal distance equal to the average edge length of all edges to represent this constraint.

Node Overlapping Constraint. Miyamoto [44] discouraged node overlapping by adjusting attractive force and repulsive force when two nodes overlap. However, node overlapping constraints can be illustrated as hard distance constraints between two nodes. That is, the distance between them must be greater than the sum of their radii.

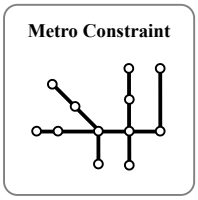
3.2.3 Direction-based Constraints

Some advanced constraints are composed of many direction constraints. Here we give a few examples.

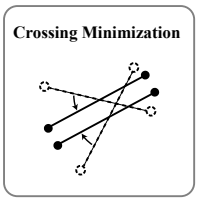
Vertical Constraint. When drawing a directed graph in flow-style, we ensure that the start node of each directed edge is above its end node [13]. This vertical constraint can be converted to direction constraint ensuring that the edge is downward pointing. According to the definition in Sect. 3.1, that is α equals $-\frac{\pi}{2}$.



Metro Constraint. Nöllenburg and Wolff [36] applied a design rule for metro maps that all edges should be restricted to the four octilinear orientations horizontal, vertical, and $\pm 45°$-diagonal. The metro constraint conforms to the design rule. We build metro constraints with direction constraints on each edge, determining the direction of them based on the location of the stations they are connected to. Thus, the metro constraint could be converted to hard direction constraints.



Crossing Minimization. It is important to minimize edge crossings to generate a high quality layout. We can use direction constraints to reduce edge crossing. For a detected crossing, as shown in the dash lines, we can rotate the edges to prevent them from crossing each other. The target direction is set to make the two edges parallel to each other.

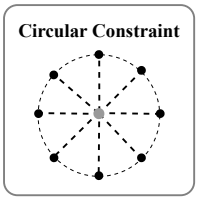


Sub-graph Constraint. When decomposing the sub-graph constraint, we can additionally use direction constraints to preserve the rotation of the sub-graph, while the distance constraints help preserve the shape of the sub-graph. It is a case of direction constraint that require both the direction and the distance, and we will further discuss it in Section 4.2.3.

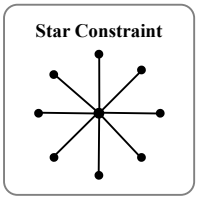
3.2.4 Angle-based Constraints

Some advanced constraints can be converted to angle constraints. Those angle-based constraints are usually designed to maintain a certain symmetry of the graph.

Circular Constraint. Circular structures are often found in biological pathways, where nodes are uniformly distributed on a circle. To form a uniformly distributed circle, the circular constraint is converted to angle constraint between the adjacent edges and the center of the nodes. For the selected n nodes, the angle is set to $\frac{2\pi}{n}$.



Star Constraint. Purchase [38] proposes to maximize the minimum angle from one center node to generate a pleasing layout. Ahmed [1] model the angular resolution by an energy function by minimizing the angular energy, the group of nodes forms like a star. However, we can convert the star constraint to angle constraint between each pair of the nodes on the periphery and the center node. For $n+1$ nodes selected, the angle is set to $\frac{2\pi}{n}$.



3.2.5 Other Constraints

Some advanced constraints like maximizing crossing-angle and preventing edges from passing through node may not be able to be converted to the basic constraints. On the other hand, if some constraints can be modeled by derivative objective functions, they could be solved by stochastic gradient descent in our proposed framework.

4 The Uniform Framework

We propose UNICON, a uniform framework (Fig. 2) to generate the user-desired graph layout with user-defined constraints. Based on the stress model, we incorporate each soft constraint as an objective function, using stochastic gradient descent for optimization. For hard constraints, we utilize the gradient projection method to satisfy them. In this way, UNICON is capable of dealing with both soft and hard constraints.

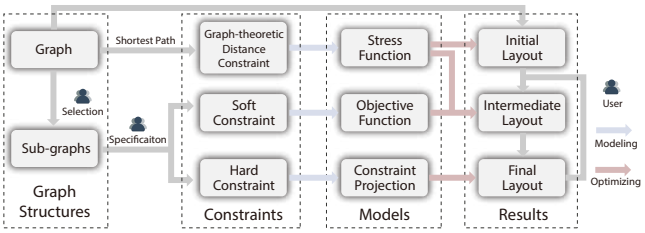


Figure 2: The uniform constraint based graph layout framework. For a graph, the graph-theoretic distance is first calculated and modeled as a stress function to generate an initial layout. The user can select sub-graphs from the graph and specify soft or hard constraints. The soft constraints are modeled as objective functions, while the hard constraints are converted into projections. The initial layout is further optimized using stochastic gradient descent of the objective function with gradient projection of hard constraints to generate the final layout after certain iterations.

4.1 Stress Model and Stochastic Gradient Descent

Our framework is based on the stress model proposed by Kamada and Kawai [29]. For a given Graph $G = (V, E)$, where V denotes a set of n nodes and E denotes a set of m edges. We place the nodes on a 2D plane, and X denotes the coordinates of each node. Each pair of node i and j has an ideal distance d_{ij}, which is the default graph-theoretical distance, which can be calculated by the shortest path algorithm. The stress function of the system is

$$Stress(\mathbf{X}) = \sum_{i<j} w_{ij} \left(\|\mathbf{X}_i - \mathbf{X}_j\| - d_{ij}\right)^2 \tag{1}$$

where $\|\ldots\|$ is L_2-norm of the vector, and $w_{ij} = d_{ij}^{-2}$ to produce the best drawings in common scenarios. By minimizing the stress function, we generate a initial layout for the given graph, which serves as the base layout for adding constraints later.

To solve the optimization problem, we apply Zheng et al.'s stochastic gradient descent method [48] in which one pair of nodes is moved at a time. We rewrite the stress function Equation 1 as

$$Stress(\mathbf{X}) = \sum_{i<j} E_{ij}(\mathbf{X}) \tag{2}$$

$$E_{ij}(\mathbf{X}) = w_{ij}(\|\mathbf{X}_i - \mathbf{X}_j\| - d_{ij})^2 \tag{3}$$

The full gradient of a stress term E_{ij} is $\partial E_{ij}/\partial \mathbf{X}$ is

$$\frac{\partial E_{ij}}{\partial \mathbf{X}_k} = \begin{cases} 4w_{ij}\mathbf{r}, & \text{if } k = i \\ -4w_{ij}\mathbf{r}, & \text{if } k = j \\ 0, & \text{otherwise} \end{cases} \tag{4}$$

where

$$\mathbf{r} = \frac{\|\mathbf{X}_i - \mathbf{X}_j\| - d_{ij}}{2} \frac{\mathbf{X}_i - \mathbf{X}_j}{\|\mathbf{X}_i - \mathbf{X}_j\|} \tag{5}$$

We apply stochastic gradient descent to minimize stress by iteratively selecting E_{ij} and moving the nodes according to the gradient, with a hard upper limit proposed by Zheng et al. [48].

$$\Delta\mathbf{X}_i = -\Delta\mathbf{X}_j = -\mu\mathbf{r}$$

$$\mu = \min\{w_{ij}\eta, 1\}$$

where η is a step size with an exponential annealing.

Our framework extends the stress model and stochastic gradient descent method to incorporate various soft constraints.

4.2 Constrained Stochastic Gradient Descent

In our framework, to take into account user-defined constraints, we model the soft constraints as objective functions. We first model the

four basic soft constraints and then show how to model the advanced constraints by the set of objective functions of the basic constraints.

As shown in Algorithm 1, for each constraint defined by the user or the default constraint derived from the stress model, we push them into a list of constraints for further optimization. Then we randomly shuffle the constraint list and pick the constraint from the constraint list. For each soft constraint, we compute the gradient of its objective function and perform the gradient descent. For hard constraints, we apply gradient projection to satisfy them, which will be introduced in the Sect. 4.3.

Algorithm 1 Constrained Stochastic Gradient Descent with Projection

Input:
Graph $G = (V, E)$;
List of user-defined soft constraints C_Soft;
List of user-defined hard constraints C_Hard;
Output:
2-D layout $\mathbf{X}$ with n nodes;

$X \leftarrow RandomMatrix(n,k)$
$d_{ij} \leftarrow ShortestPaths(G)$
for each $i, j : i < j$ **do**
  $C\_Soft.push(Distance\_Constraint(i, j, d_{ij}))$
end for
for η in *step size annealing schedule* **do**
  for *constraint* in C_Soft in random order **do**
    Gradient Descent(*constraint*)
  end for
  for *constraint* in C_Hard **do**
    Projection(*constraint*)
  end for
end for

4.2.1 Position Constraint

For soft position constraint, suppose the ideal position of node i is $\mathbf{P}_i$. We model the energy function based on the distance between $\mathbf{X}_i$, that is, the position of node i and the ideal position $\mathbf{P}_i$:

$$E_i = w_i \|\mathbf{X}_i - \mathbf{P}_i\|^2 \tag{6}$$

where w_i is a user-defined coefficient. The gradient is calculated as follows:

$$\frac{\partial E_i}{\partial \mathbf{X}_i} = 2w_i(\mathbf{X}_i - \mathbf{P}_i) \tag{7}$$

4.2.2 Distance Constraint

As mentioned in the previous Sect. 3.1, for soft distance constraint, the ideal distance between node i and j is d_{ij}. If we do not consider distance on the x-axis or y-axis separately, the objective function is defined the same as in the stress model:

$$E_{ij} = w_{ij}(\|\mathbf{X}_i - \mathbf{X}_j\| - d_{ij})^2 \tag{8}$$

The gradient of E_{ij} is calculated as follows:

$$\frac{\partial E_{ij}}{\partial \mathbf{X}_k} = \begin{cases} 4w_{ij}\mathbf{r}, & \text{if } k = i \\ -4w_{ij}\mathbf{r}, & \text{if } k = j \end{cases} \tag{9}$$

where $\mathbf{r}$ is the same in Equation 5.

However, suppose we consider the distance on the x-axis or y-axis separately . In that case, the objective function is rewritten as follows based on the difference between the distance between two points and the ideal distance on the x-axis (dx_{ij}) or on the y-axis.

$$Ex_{ij} = w_{ij}(|x_i - x_j| - dx_{ij})^2 \tag{10}$$

Here we only explain the case on x-axis, the case on y-axis is similar. The gradient is calculated as follows:

$$\frac{\partial Ex_{ij}}{\partial x_k} = \begin{cases} 4w_{ij}r, & \text{if } k = i \\ -4w_{ij}r, & \text{if } k = j \end{cases} \tag{11}$$

where

$$r = \frac{|x_i - x_j| - dx_{ij}}{2} \frac{x_i - x_j}{|x_i - x_j|} \tag{12}$$

To obtain the desired graph layout, we need to carefully consider the coefficients w_{ij} in front of the objective function. Here, we set the default w_{ij} based on the average weight of node i and node j computed in the initial step on graph-theoretic distance. However, the users can adjust the weight of w_{ij} to achieve their desired layout.

4.2.3 Direction Constraint

For a direction constraint, it can be divided into two cases according to the actual requirements. One is to require the direction and the distance, that is, and the other is to require only the direction. In the first case, the ideal distance d_{ij} is considered, the objective function is defined as follows, simultaneously ensuring the satisfaction of both direction constraint and distance constraint:

$$E_{ij} = w_{ij}\left[(x_i - x_j - d_{ij}\sin\alpha)^2 + (y_i - y_j - d_{ij}\cos\alpha)^2\right] \tag{13}$$

The gradient is calculated as follow:

$$\frac{\partial E_{ij}}{\partial x_k} = \begin{cases} 4w_{ij}r, & \text{if } k = i \\ -4w_{ij}r, & \text{if } k = j \end{cases} \tag{14}$$

where

$$r = \frac{x_i - x_j - d_{ij}\sin\alpha}{2} \tag{15}$$

If d_{ij} is not specified or we do not consider ideal distance, we can calculate d'_{ij} at each iteration, and the corresponding objective function is defined as follows:

$$E_{ij} = w_{ij}\left[(x_i - x_j - d'_{ij}\sin\alpha)^2 + (y_i - y_j - d'_{ij}\cos\alpha)^2\right] \tag{16}$$

where

$$d'_{ij} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} \tag{17}$$

Otherwise, the objective function can also be defined on the angle of the edge between the two nodes with respect to the x-axis, which we denote as θ:

$$E_{ij} = w_{ij}(\theta - \alpha)^2 \tag{18}$$

The gradient of the angle is calculated as:

$$\frac{\partial E_{ij}}{\partial \theta} = 2w_{ij}(\theta - \alpha) \tag{19}$$

We then rotate the edge from its midpoint according to the gradient. This movement allows moving the nodes as short a distance as possible to get the desired angle while keeping the distance between the two points constant. We will illustrate the rationale in a subsequent Sect. 4.3.

4.2.4 Angle Constraint

Suppose the current angle of the two edges is θ, we use an objective function based on the angle instead of distance.

$$E_{ijk} = w_{ijk}(\theta - \alpha)^2 \tag{20}$$

The gradient is calculated as follows:

$$\frac{\partial E_{ijk}}{\partial \theta} = 2w_{ijk}(\theta - \alpha) \tag{21}$$

By calculating the gradient, we rotate the direction of two edges from the angle bisector. Note that the coefficient w_{ijk} can be adjusted by the user to generate their desired layout.

4.3 Gradient Projection

For hard constraints, we apply gradient projection to satisfy them. After each iteration of stochastic gradient descent, we project the nodes according to the hard constraints. That is, the nodes are moved at the shortest distance to satisfy the constraint. Dwyer et al. [9] use gradient projection to deal with Euclidean distance constraint. Here we define the projection problem and then introduce its solution for each basic constraint.

Denote $\mathscr{D}$ the hard constraint on position $\mathbf{X}_i$, $P(\cdot)$ is the projection operator, and itself is an optimization problem defined as follows:

$$P_{\mathscr{D}}(\mathbf{X}_i) = \arg\min_{\mathbf{X}_i \in \mathscr{D}} \frac{1}{2}\|\mathbf{X}_i - \mathbf{X}_{i0}\|^2 \tag{22}$$

where $\mathbf{X}_{i0}$ is the result of the position of the node after each iteration of stochastic gradient descent. This optimization problem finds a point $\mathbf{X}_i$ on the constraint boundary that is closest to $\mathbf{X}_{i0}$. The solution to this problem is thus a projection of the node that satisfies the constraint. It is hard to find a general solution to an arbitrary $\mathscr{D}$. However, we can find particular solutions in the case of four basic constraints. We are going to introduce variants of projection operation for each basic constraint and the simplified solutions to the problems.

4.3.1 Position Constraint

For a hard position constraint, if the given constraint is a fixed position constraint on position $\mathbf{P_i}$, there is no doubt that the solution is that the position of the node i is moved to $\mathbf{P_i}$. In the case of given fixed P_x or P_y, it is apparent that we should move the node to the specified position in the direction parallel to the y-axis or x-axis, which is the shortest path to move the node to the given position.

The linear boundary position constraint can be expressed in the form of inequalities:

$$\mathbf{A_i X_i} + b_i \geq 0 \tag{23}$$

where $\mathbf{A_i}$ and b_i are parameters to define the boundary. On a two-dimensional plane, we can fit this inequality with different parameters to define various position boundary constraints as shown in Fig. 3. The node must be on one side of the line, and the special case is that the line is parallel to the x-axis or y-axis. However, the special cases are more commonly used.

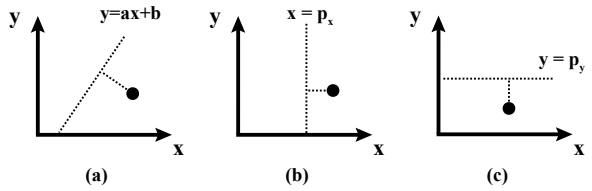


Figure 3: Projection of position boundary constraint. (a) generalized situation; (b) constraint on the x-axis; (c) constraint on the y-axis.

From Fig. 3, we can conclude that the solution to this projection optimization problem is to move the node along the vertical line to the boundary line if the constraint is not satisfied.

4.3.2 Distance Constraint

For a hard distance constraint, the projection operator is associated with node i and j, and it can be expressed as:

$$P_{\mathscr{D}}(\mathbf{X}_i, \mathbf{X}_j) = \arg\min_{\mathbf{X}_i, \mathbf{X}_j \in \mathscr{D}} \frac{1}{2}\left(w_i\|\mathbf{X}_i - \mathbf{X}_{i0}\|_2^2 + w_j\|\mathbf{X}_j - \mathbf{X}_{j0}\|_2^2\right) \tag{24}$$

where w_i and w_j are the weight of two nodes. The fixed distance constraint can be written as:

$$\|\mathbf{X_i} - \mathbf{X_j}\| = d_{ij} \tag{25}$$

Essentially, this optimization problem finds two points $\mathbf{X}_i$, $\mathbf{X}_j$ on the constraint boundary that are closest to $\mathbf{X}_{i0}$ and $\mathbf{X}_{j0}$, respectively. Thus this represents a projection of $\mathbf{X}_{i0}$ and $\mathbf{X}_{j0}$ to the nearest constraint surface.

As shown in Fig. 4, we can solve the problem by moving the two nodes in the direction of the edge between them until the distance between them is equal to d_{ij}. In the situation of equal weight, that is when w_i equals w_j. The solution is to move the two nodes an equal distance in the opposite direction. The vector of movement of nodes i and j are $\mathbf{r}_i$ and $\mathbf{r}_j$:

$$\mathbf{r}_i = -\mathbf{r}_j = \frac{\|\mathbf{X}_i - \mathbf{X}_j\| - d_{ij}}{2} \frac{\mathbf{X}_i - \mathbf{X}_j}{\|\mathbf{X}_i - \mathbf{X}_j\|} \tag{26}$$

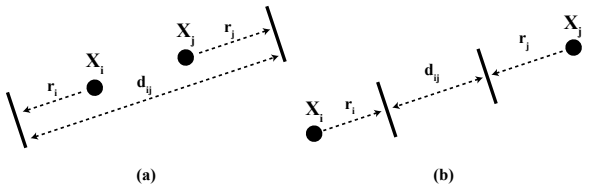


Figure 4: Projection of hard distance constraint. (a) the distance between two nodes is less than the given distance; (b) the distance between two nodes is greater than the given distance

The distance boundary constraints are in the form of

$$\|\mathbf{X_i} - \mathbf{X_j}\|(\leq, \geq)d_{ij} \tag{27}$$

If the inequality is already satisfied before the projection, we need to do nothing. Otherwise, we should move the nodes in the same way as the fixed distance constraint till the inequality is satisfied.

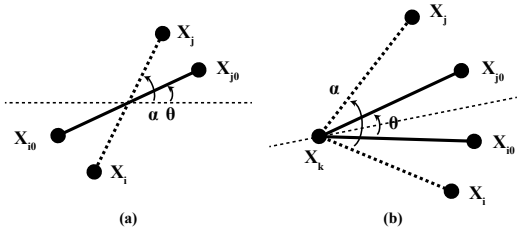


Figure 5: (a) Projection of hard direction constraint. Rotate the edge on the axis of its midpoint to satisfy the hard direction constraint. (b) Projection of hard angle constraint. Rotate the two edges on the axis of the vertex from their bisector to satisfy the hard angle constraint.

4.3.3 Direction Constraint

For a hard direction constraint, the projection operator is defined as Equation 24. However, the constraint here is different. In a fixed direction constraint, the angle of the edge between node i and j with respect to the x-axis must be α. When moving the two nodes, the distance between them should be preserved.

As shown in Fig. 5 (b), in the situation of equal weight, we derive the solution to the problem. That is to rotate the two nodes on the axis of the midpoint of the edge. For direction boundary constraint, we do a minimum angular rotation of the edge to meet requirements.

4.3.4 Angle Constraint

To solve the projection problem of angle constraint, we need to consider the angle-based advanced constraints. In the star constraint, we do not expect the center node to move. So in the projection of the decomposed hard angle constraint, we consider not moving the angle's vertex. Thus, the projection operation is the same as Equation 24 with a different constraint. For fixed angle constraint, the angle is to be equal to α.

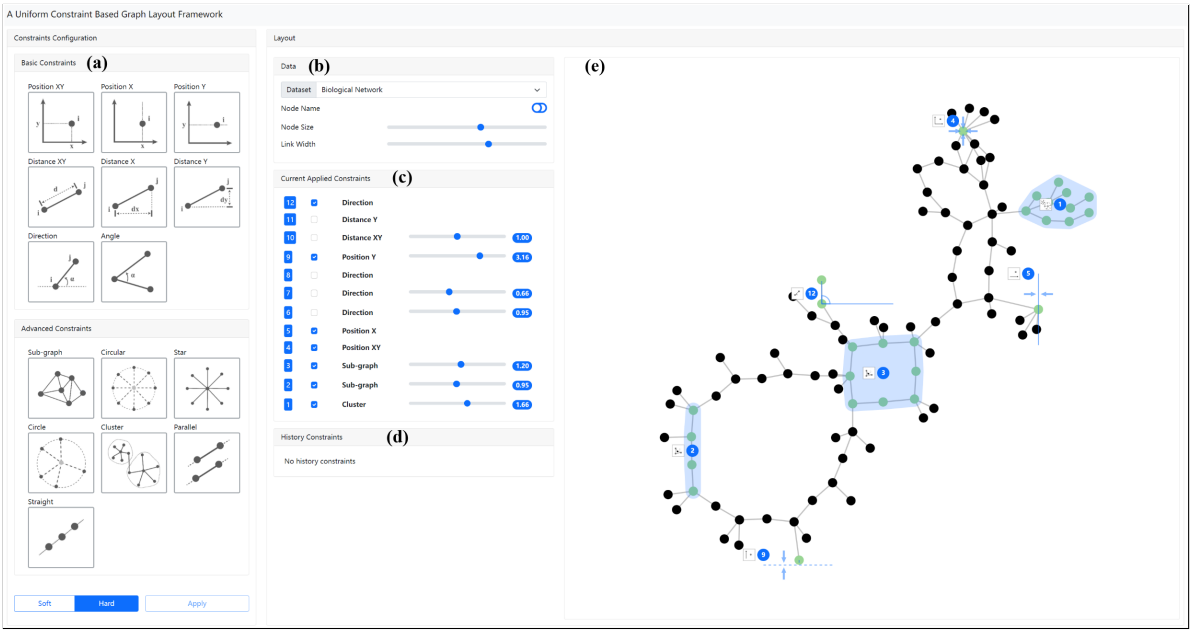


Figure 6: Interface Overview. (a) Constraint Configuration Panel; (b) Data Manipulation Panel; (c) Constraint Manipulation Panel; (d) History Constraint Panel; (e) Graph Layout View.

As shown in Fig. 5 (b), in the situation of equal weight, we derive the solution to the problem. That is to rotate node i and j around the vertex of the angle, by the same angle from its bisector. We do a minimum angular rotation of the edges for angle boundary constraint to meet requirements.

5 Interactive Prototype System

To demonstrate the strength of our framework, we implement an interactive system for constrained graph visualization based on the uniform framework. Users can efficiently select nodes in the graph and apply desired constraints with a few clicks to instruct the system or abandon constraints if not satisfied. The system can save and load user-defined constraints for convenient exploration.

The user interface in Fig. 6 consists of five parts: graph layout view, data manipulation panel, constraint configuration panel, constraint manipulation panel , and history constraint panel. We briefly describe the supported user interactions below.

In the graph layout view, the layout of the graph is displayed. The user is able to drag and drop nodes. The positions of nodes and links are updated in real time according to the user's interaction. Thus the user can easily adjust them to the ideal position. The user can select or deselect a node by clicking it. Then, the user is allowed to perform further operations on the selected nodes. In the data manipulation panel, the user is able to choose and load the dataset through the drop-down selection box. The "Node Name" switch below allows the user to toggle the display of node names. With the "Node Size" and "Link Width" drag bars, the user is able to adjust the size of nodes and the width of links in the graph layout view. The constraint configuration panel contains the function buttons to define constraints. The buttons are used to select constraints, and illustrations are provided on the button to help the user understand the constraints to select. In the bottom toolbar, the user is able to switch between soft constraint and hard constraint and apply the defined constraint. The constraint manipulation panel displays the applied constraints, which allows the user to make further modifications to their defined constraints. The history constraint panel can record the history of user defined constraints.

After the data of the graph is loaded, the rendered graph will be displayed in the graph layout view. The user will be able to click to select one or several nodes in the graph layout view, and the selected nodes will be displayed in red. Then the program will highlight the constraints that are allowed to be added according to the number of selected nodes. For example, the position constraint requires the user to select at least one node, and the angle constraint requires the user to select exactly three nodes. In order to add constraints, the user needs to drag the selected nodes in the graph to the desired position and select a constraint rule by clicking the corresponding button in the constraint configuration panel. After clicking the "Apply" button, a new constraint will be created based on the nodes and constraint button selected by the user. The graph will then be refreshed, and the relevant nodes will be displayed in green as shown in Fig. 6 (c). We designed different highlights for different constraints. For example, we have straight lines and arrows highlighting position constraints and convex hulls highlighting sub-graph constraints. Each constraint that has been added will be displayed in the form of a list in the constraint manipulation panel. By checking the checkbox, the user can easily enable or disable a constraint. The name of the constraint is editable. Thus the user can customize it for easy management. A drag bar is displayed to adjust the weight for soft constraints.

6 Case Study

We implemented a JavaScript version of our algorithm for building the interactive prototype system. In the previous Sect. 4, we illustrate how our framework deals with soft and hard constraints. Here, we apply different constraints for different graphs and observe the change in graph layout to confirm the satisfaction of the constraints. To demonstrate the capability of our method to deal with soft and hard constraints, we evaluated from three aspects. First, we used several synthesized dataset to verify the effect of our framework on soft constraints and hard constraint. Then we tested the sub-graph constraints for merging graphs and compared with Yuan et al. [47] on a biochemical metabolic pathway dataset. We also tested our framework on a dataset Bus1138 [6] with 1138 nodes and compared results with previous works, an incremental procedure for layouts with separation constraints (IPSep) [11], a scalable, versatile and simple constrained graph layout (SV) [9] , and Wang et al. [46] using edge vectors. Finally, we evaluated the metro constraint on the dataset of the Beijing subway map.

6.1 Distance Constraints

For distance constraints, we conducted experiments on a dataset used by [44]. As shown in Fig. 7, we apply soft or hard constraints to four parts of the graph in different colors. For edges in blue and edges in green, we apply soft distance constraints. For each edge in blue, we add distance constraints that are smaller than its original ideal distances. While for each edge in green, we set a larger distance constraint than its original ideal distance. We can observe that the shape of these two parts becomes smaller or larger respectively after applying the constraints.

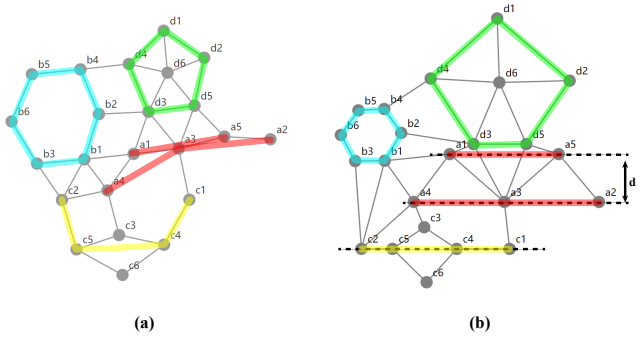


Figure 7: Applying distance constraint. (a) unconstrained layout; (b) layout with both soft and hard distance constraints.

For nodes connected by red and yellow lines, we apply hard distance constraints that can be written as follows:

$$\begin{aligned} a2.y &= a3.y = a4.y \\ a1.y &= a5.y = a2.y + d \\ c1.y &= c2.y = c4.y = c5.y \end{aligned} \tag{28}$$

That is the distances among nodes $a2, a3, a4$ on the y-axis should be zero, and so are those among nodes $c1, c2, c4, c5$. In addition, nodes $a1$ and $a5$ should be position at a distance of d above node $a2$. In the constrained layout (Fig. 7 (b)), those nodes that need to be on the same level are placed strictly along a horizontal line, and the distance between node $a1$ and $a2$ on the y-axis is realized at d.

6.2 Star Constraints

For the graph shown in Fig. 8, there are two star-like shapes highlighted in blue and green. However, in the unconstrained layout (Fig. 8 (a)), the star shape is not clearly observable. To see the shape of the star more clearly, we apply the star constraint to each of the two stars. First, we apply a soft constraint to the graph, the constrained layout in shown in Fig. 8 (b). We can observe that the star shape is more pronounced than the unconstrained layout. We further test the hard star constraint. As shown in Fig. 8 (c). It can be seen that the angles in the star are exactly equal.

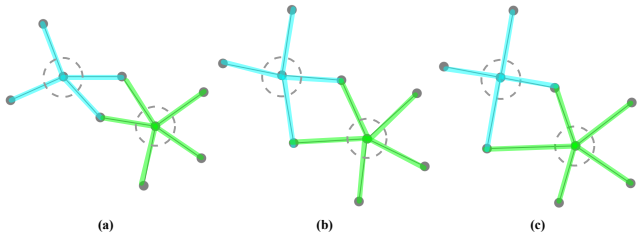


Figure 8: Applying star constraints. (a) unconstrained layout; (b) soft star constraint; (c) hard star constraint.

6.3 Sub-graph Constraints

Here, we merge sub-graphs while preserving their topology using sub-graph constraints. And we compare our method with Yuan et al. [47] on a biochemical metabolic pathway dataset. As shown in Fig. 9(a) is the desired layouts of sub-graphs, in Fig. 9(b) is the unconstrained layout of our method. Yuan et al. [47] used Laplacian constrained distance embedding to preserve the topological information of the input sub-graph. In UNICON, we can use either soft or hard sub-graph constraints, the results are shown in Fig. 9(c) and Fig. 9(d). We use the similarity scores to measure how well the sub-graph structures are preserved in the merged layout. And for the two sub-graphs, the results of UNICON using hard sub-graph constraints are both 1.00, better than 0.91 and 0.96 in Yuan et al. [47]. This indicates the hard constraints can better preserve topology when merging sub-graphs.

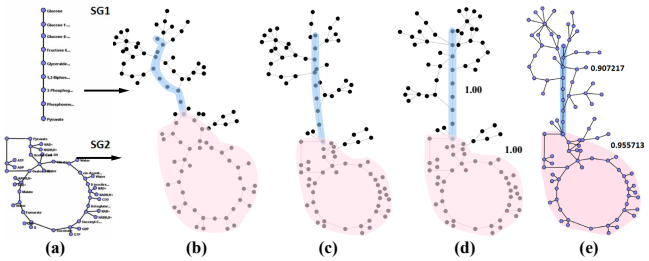


Figure 9: Merging sub-graphs compared with Yuan et al. [47]. (a) Given sub-graphs; (b) Our method without any constraint; (c) Our method with soft sub-graph constraint; (d) Our method with hard sub-graph constraint; (e) Yuan's method.

6.4 Vertical Constraints

We ran our algorithm on data Bus1138 [6], which contains 1138 nodes and 1458 edges. As shown in Fig. 10, first, no constraints are applied to the graph. Then, we apply constraints to achieve vertical constraints in two ways in UNICON. The first is to translate the vertical constraint into a distance constraint on the x-axis, which means that the distance in x-axis between two adjacent nodes would ideally be zero. The second is to transform it into a direction constraint, where the direction of the directed edge is the vertically downward direction. Here we utilize vertical edges (VE) [46] to measure the constraint satisfaction degree.

$$V(\mathbf{X}) = \sum_{(i,j)\in E} \left| \left\langle \frac{\mathbf{X}_i - \mathbf{X}_j}{\|\mathbf{X}_i - \mathbf{X}_j\|}, \vec{x} \right\rangle \right| \tag{29}$$

where $\vec{x} = (1,0)$ is the direction vector along the x-axis. This measures the sum of absolute inner products between the edge direction and the positive x-axis. The smaller the VE is, the better the layout satisfies the vertical constraint. VE measure is a real number, and here we rounded our results to compare with the prior works. IPsep has a VE number of 714, SV of 757, and Wang's method of 714. Our method using the direction constraint results in a VE number of 714. And with the distance constraint, our method results in the smaller VE number of 703. This indicates our approach using distance constraints better satisfies the vertical constraints.

6.5 Metro Constraints

We first convert metro constraint to hard direction constraint, according to the original location of stations. We give a list of available directions, in this case horizontal, vertical, or $\pm 45°$. The algorithm automatically chooses the direction of each edge from the given list, which is closest to its original direction based on the positions of its connecting subway stations. The result is shown in Fig. 11 (a),

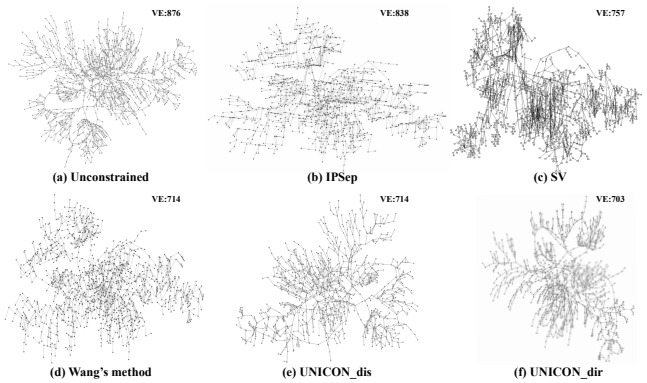


Figure 10: Comparison of the layout of the Bus1138 graph data with vertical constraints generated by different methods: (a) unconstrained layout; (b) IPSep [11]; (c) SV [9]; (d) Wang et al. [46]; (e) UNICON with distance constraint; and (d) UNICON with direction constraint. Here VE is the metric defined in (29), smaller = better. We ran our algorithms for several times and the results with a median VE were chosen.

we can see that almost all the edges are satisfied with the direction constraint.

However, if we zoom in on the metro map, there are still some flaws. At an interchange station, if two subways pass along a similar direction, for example, along the near horizontal. We notice that in this situation, the two subway lines will overlap. However, this is not a problem with our framework but rather a lack of thoughtfulness in constructing the constraints. We can avoid this situation by adjusting the direction of the subway line when specifying the constraint.

We can further add edge length variation constraint on the graph, which makes the lengths of edges as uniform as possible as shown in Fig. 11 (b). For edges that are originally extremely long or short, this constraint will force them to a medium length. However, since the constraint is soft, and there are existing constraints on the length of each edge, the lengths of edges are not perfectly uniform.

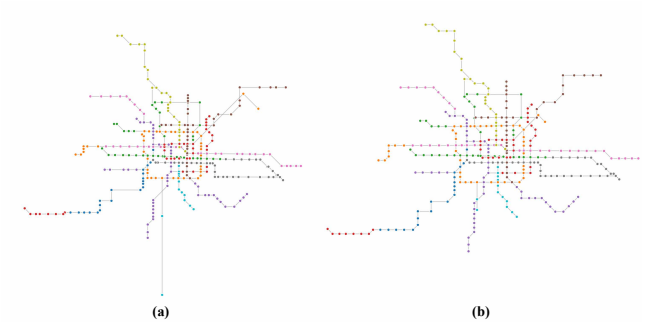


Figure 11: The map of Beijing subway stations with 340 nodes and 381 edges. (a) Metro constraints applied. The direction of the edges is chosen automatically from horizontal, vertical, or $\pm 45°$. (b) Additional edge length variation constraint applied.

7 Discussion and Future Work

Our framework is able to handle both soft and hard constraints by modeling them with objective functions or by gradient projection methods. While Ahmed et al. [1] and Devkota et al. [7] can not handle hard constraints. In our proposed taxonomy, we divided the constraints into basic and advanced constraints, and we introduced how to convert advanced constraints to basic constraints. With the position constraint and angle constraint, our framework is able to handle constraints on a single node and angles while Wang et al. [46] could not by using edge vectors. However, there are still some advanced constraints that can not be translated into basic constraints. For example, constraints on maximizing crossing-angle and preventing edges from passing through nodes, etc. We will try to decompose them into basic constraints in the future. Compared with Dwyer et al. [9], we use stochastic gradient descent in optimization instead of steepest descent. This makes our framework more flexible and efficient, and one can add any differentiable objective functions to generate their desired layouts.

For soft constraints, because of the stochastic gradient descent approach used, our framework is order invariant. For any constraints defined by the user, they are pushed into a constraint list which will be used as input to the gradient descent algorithm. For hard constraints, our framework uses a projection approach to satisfy them. But the order of iterations can have a significant impact on the satisfaction of these constraints, especially in the case of random order of constraint selection for projection. In every iteration, each hard constraint will be projected only once, and hard constraints selected later will tend to be satisfied, while those selected earlier are easily destroyed by later projections. This situation can be more obvious in the case of constraint conflicts and may even lead to non-convergence. However, the current design of our framework is based on the case where the user is given reasonable constraints. We plan to design a conflict detection algorithm to handle conflicting constraints. Given a priority list of hard constraints, in the case that multiple hard constraints cannot be satisfied at the same time, the hard constraint with the higher priority is guaranteed first.

The interactive prototype system we implemented is designed to help users specify the desired constraints efficiently. Currently, a number of constraints can be specified interactively, and we have implemented a set of constraints that the prototype system can not support at the moment. In future work, we will develop more interaction forms to help users quickly specify constraints that are not currently available, including boundary constraints and other advanced constraints.

Currently, our toolkit is implemented using the JavaScript programming language. This facilitates the building of the interactive prototype system without the need to consider the communication between the front-end and back-end. However, due to the speed of the JavaScript, this can bring some efficiency problems for the constrained layout of very large graphs. If we were to consider very large graphs and multi-person collaboration, we would choose to use a python back-end with C++ as the underlying code to improve the efficiency of the framework.

8 Conclusion

In this paper, we propose UNICON, a uniform constraint based graph layout framework. UNICON accommodates soft constraints by incorporating them in the objective functions based on the stress model, and optimizes them with stochastic gradient descent. It handles hard constraints, such as inequalities or equalities in the layout space, by gradient projection. An interactive constrained graph layout system is implemented based on UNICON. With this system, the user can easily add or remove constraints to generate the desired layouts. We have tested the efficacy of UNICON in dealing with soft constraints as well as in satisfying hard constraints and have demonstrated the ability of this approach to generate pleasing graph layouts on a number of application cases.

Acknowledgments

The authors thank Shuqiao Zhang at Peking University for participating in implementation of the prototype system. The authors thank the anonymous reviewers for their valuable comments. This work is supported by NSFC No. 61872013.

REFERENCES

[1] A. R. Ahmed, F. D. Luca, S. Devkota, S. Kobourov, and M. Li. Graph drawing via gradient descent, $(GD)^2$. In *Proc. Intl. Symp. on Graph Drawing and Network Visualization*, pp. 3–17, 2020.

[2] K.-F. Böhringer and F. N. Paulisch. Using constraints to achieve stability in automatic graph layout algorithms. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pp. 43–51, 1990.

[3] U. Brandes and C. Pich. Eigensolver methods for progressive multidimensional scaling of large data. In *Proc. Intl. Symp. on Graph Drawing*, pp. 42–53, 2006.

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

[5] 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*, pp. 69–73, 2015.

[6] T. A. Davis and Y. Hu. The university of florida sparse matrix collection. *ACM Transactions on Mathematical Software*, 38(1).

[7] S. Devkota, A. R. Ahmed, F. D. Luca, K. Isaacs, and S. G. Kobourov. Stress-Plus-X (SPX) graph layout. In *Proc. Intl. Symp. on Graph Drawing and Network Visualization*, pp. 291–304, 2019.

[8] C. Dunne, S. I. Ross, B. Shneiderman, and M. Martino. Readability metric feedback for aiding node-link visualization designers. *IBM Journal of Research and Development*, 59(2/3):14:1–14:16, 2015.

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

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

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

[12] T. Dwyer, Y. Koren, and K. Marriott. Constrained graph layout by stress majorization and gradient projection. *Discrete Mathematics*, 309(7):1895–1908, 2009.

[13] T. Dwyer, K. Marriott, and M. Wybrow. Dunnart: A constraint-based network diagram authoring tool. In *Proc. Intl. Symp. on Graph Drawing*, pp. 420–431, 2008.

[14] T. Dwyer, K. Marriott, and M. Wybrow. Topology preserving constrained graph layout. In *Proc. Intl. Symp. on Graph Drawing*, pp. 230–241, 2008.

[15] T. Dwyer and G. Robertson. Layout with circular and other non-linear constraints using procrustes projection. In *Proc. Intl. Symp. on Graph Drawing*, pp. 393–404, 2009.

[16] P. Eades. A heuristic for graph drawing. *Congressus Numerantium*, 42:149–160, 1984.

[17] T. M. J. Fruchterman and E. M. Reingold. Graph drawing by force-directed placement. *Softw. Pract. Exper.*, 21(11):1129–1164, 1991.

[18] E. R. Gansner, Y. Koren, and S. North. Graph drawing by stress majorization. In *Proc. Intl. Symp. on Graph Drawing*, pp. 239–250, 2004.

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

[20] H. Gibson, J. Faith, and P. Vickers. A survey of two-dimensional graph layout techniques for information visualisation. *Information Visualization*, 12:324–357, 2013.

[21] V. Guchev and C. Gena. Sketch-based interactions for untangling of force-directed graphs. In *Proceedings of International Conference Information Visualisation*, pp. 288–291, 2017.

[22] S. Hachul and M. Jünger. Drawing large graphs with a potential-field-based multilevel algorithm. In *Proc. Intl. Symp. on Graph Drawing*, pp. 285–295, 2004.

[23] D. Harel and Y. Koren. Graph drawing by high-dimensional embedding. In *Proc. Intl. Symp. on Graph Drawing*, pp. 207–219, 2002.

[24] W. He and K. Marriott. Constrained graph layout. In *Proc. Intl. Symp. on Graph Drawing*, pp. 217–232, 1997.

[25] J. Hoffswell, A. Borning, and J. Heer. SetCoLa: High-level constraints for graph layout. *Computer Graphics Forum*, 37, 2018.

[26] Y. Hu. Efficient, high-quality force-directed graph drawing. *The Mathematica journal*, 10:37–71, 2006.

[27] M. L. Huang and P. Eades. A fully animated interactive system for clustering and navigating huge graphs. In *Proc. Intl. Symp. on Graph Drawing*, pp. 374–383, 1998.

[28] M. Jünger and P. Mutzel. Exact and heuristic algorithms for 2-layer straightline crossing minimization. In *Proc. Intl. Symp. on Graph Drawing*, pp. 337–348, 1996.

[29] T. Kamada and S. Kawai. An algorithm for drawing general undirected graphs. *Information Processing Letters*, 31:7–15, 1989.

[30] T. Kamps, J. Kleinz, and J. Read. Constraint-based spring-model algorithm for graph layout. In *Proc. Intl. Symp. on Graph Drawing*, pp. 349–360, 1995.

[31] Y. Ko and H. Yen. Drawing clustered graphs using stress majorization and force-directed placements. In *Proceedings of International Conference Information Visualisation*, pp. 69–74, 2016.

[32] J. F. Kruiger, P. E. Rauber, R. M. Martins, A. Kerren, S. Kobourov, and A. C. Telea. Graph layouts by t-SNE. *Computer Graphics Forum*, 36(3):283–294, 2017.

[33] O. Kwon, T. Crnovrsanin, and K. Ma. What would a graph look like in this layout? a machine learning approach to large graph visualization. *IEEE Trans. Vis. Comput. Graphics*, 24(1):478–488, 2018.

[34] O. Kwon and K. Ma. A deep generative model for graph layout. *IEEE Trans. Vis. Comput. Graphics*, 26(1):665–675, 2020.

[35] K. Marriott, P. J. Stuckey, V. Tam, and W. He. Removing node overlapping in graph layout using constrained optimization. *Constraints*, 8:143–171, 1998.

[36] M. Nöllenburg and A. Wolff. Drawing and labeling high-quality metro maps by mixed-integer programming. *IEEE Trans. Vis. Comput. Graphics*, 17(5):626–641, 2011.

[37] J. Pan, W. Chen, X. Zhao, S. Zhou, W. Zeng, M. Zhu, J. Chen, S. Fu, and Y. Wu. Exemplar-based layout fine-tuning for node-link diagrams. *IEEE Trans. Vis. Comput. Graphics*, 27(2):1655–1665, 2021.

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

[39] 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.

[40] D. P. Singh, L. Lisle, T. M. Murali, and K. Luther. CrowdLayout: Crowdsourced design and evaluation of biological network visualizations. In *Proceedings of CHI Conference on Human Factors in Computing Systems*, pp. 1–14, 2018.

[41] M. Steiger, H. Lücke-Tieke, T. May, A. Kuijper, and J. Kohlhammer. Deterministic local layouts through high-dimensional layout stitching. In *Human-Computer Interaction. Theories, Methods, and Tools*, pp. 643–651, 2014.

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

[43] R. Tamassia. *Handbook of Graph Drawing and Visualization*. Chapman & Hall/CRC, 1st ed., 2016.

[44] X. Wang and I. Miyamoto. Generating customized layouts. In *Proc. Intl. Symp. on Graph Drawing*, pp. 504–515, 1995.

[45] Y. Wang, Z. Jin, Q. Wang, W. Cui, T. Ma, and H. Qu. DeepDrawing: A deep learning approach to graph drawing. *IEEE Trans. Vis. Comput. Graphics*, 26(1):676–686, 2020.

[46] Y. Wang, Y. Wang, Y. Sun, L. Zhu, K. Lu, C. Fu, M. Sedlmair, O. Deussen, and B. Chen. Revisiting stress majorization as a unified framework for interactive constrained graph visualization. *IEEE Trans. Vis. Comput. Graphics*, 24(1):489–499, 2018.

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

[48] J. X. Zheng, S. Pawar, and D. F. M. Goodman. Graph drawing by stochastic gradient descent. *IEEE Trans. Vis. Comput. Graphics*, 25(9):2738–2748, 2019.