DeepGD: A Deep Learning Framework for Graph Drawing Using GNN

Xiaoqi Wang
The Ohio State University

Kevin Yen
Yahoo! Research

Yifan Hu
Yahoo! Research

Han-Wei Shen
The Ohio State University

***Abstract*—In the past decades, many graph drawing techniques have been proposed for generating aesthetically pleasing graph layouts. However, it remains a challenging task since different layout methods tend to highlight different characteristics of the graphs. Recently, studies on deep learning based graph drawing algorithm have emerged but they are often not generalizable to arbitrary graphs without re-training. In this paper, we propose a Convolutional Graph Neural Network based deep learning framework, DeepGD, which can draw arbitrary graphs once trained. It attempts to generate layouts by compromising among multiple pre-specified aesthetics considering a good graph layout usually complies with multiple aesthetics simultaneously. In order to balance the trade-off, we propose two adaptive training strategies which adjust the weight factor of each aesthetic dynamically during training. The quantitative and qualitative assessment of DeepGD demonstrates that it is capable of drawing arbitrary graphs effectively, while being flexible at accommodating different aesthetic criteria.**

THE INTRODUCTION A graph is a mathematical structure in which nodes represent entities and edges indicate the relationships among the entities. Graphs are widely used to represent many different types of data such as transactions, transportation networks, social relationship, etc. The most popular way to visualize graphs is to use node-link diagrams, where the topological structure of the graph can be directly visualized.

However, graph drawing is a challenging task. Different layout methods tend to highlight different characteristics of a graph. Therefore, choosing the most appropriate layout method usually requires in-depth knowledge about layout methods as well as the graph being drawn. This leads to a question: what's the most appropriate layout method and how should we evaluate it?

The goodness of a graph layout can be evaluated by aesthetic metrics such as edge length variation, stress, minimum angle, etc. Different metrics focus on different visual properties and cater for different human preferences. Nevertheless, there is no single consensus on which metric is the best, and some of the metrics are even contradictory to

each other. Therefore, a method that considers multiple aesthetic metrics simultaneously and makes sensible compromises is more likely to generate a visually pleasing graph layout [1]. Furthermore, we are ultimately interested in learning human preference automatically. Towards that goal, we need a general framework that is able to optimize an arbitrary objective function that represents the human preference.

In recent years, new techniques have been proposed to utilize machine learning algorithms for graph layouts. However, some of these algorithms require special training data for each graph drawing task so that the model needs to be re-trained with new training data. For example, if it is trained to draw a star graph, the model cannot properly draw a tree graph without re-training on new data. To relax this constraint, other layout methods involve human interaction to collect data and to learn the graph drawing per human preference. In short, among the machine learning based graph drawing algorithms proposed so far, it is not possible to train a general graph drawing model that directly optimizes multiple aesthetic criteria, as specified by a cost function.

In this paper, we propose *DeepGD*, a deep learning framework for graph drawing, which generates graph layouts complying with multiple aesthetic metrics simultaneously. DeepGD can be applied easily to optimize most of the commonly agreed aesthetic metrics. Also, DeepGD only needs to be trained once and can subsequently be applied to draw arbitrary types of graphs. In terms of the methodology, a Convolutional Graph Neural Network (ConvGNN) is used to produce the position of nodes such that the desired aesthetic metrics are optimized in the resulting layout. To accomplish this goal, a multi-objective loss function is designed in a way that each of the aesthetic aspects is represented by an individual component in the composite loss function. Additionally, two adaptive training strategies are proposed to automatically adjust the weight factors corresponding to each loss component such that the human preference on aesthetics trade-off is reflected in the weight factors.

The effectiveness and efficiency of the proposed deep learning framework is evaluated both qualitatively and quantitatively against baseline including the stress majorization algorithm [2] and PivotMDS [3]. The results from our extensive experiments show that our work can generate aesthetically pleasing graphs layout by compromising between multiple aesthetic metrics. In addition, the robust performance of our model shows that it has the ability to capture the latent patterns and relationships in graph data, instead of merely memorizing the training samples.

The primary contributions of this work are:

- A novel ConvGNN-based framework for graph drawing capable of incorporating multiple aesthetic metrics simultaneously.
- A flexible model design which makes DeepGD applicable to arbitrary types of graphs once the model is trained.
- Two adaptive training strategies for graph drawing that adjust the weight factors dynamically to balance the trade-off among aesthetics.
- A comprehensive experimental study that optimizes minimum angle, edge length variation, stress energy, t-SNE, and node occlusion loss functions, either on their own or in combination.

It is worth mentioning that scaling up for large graphs is not the focus of this paper. Rather, our work confirms that our proposed deep learning framework can produce aesthetically pleasing layouts that optimize *arbitrary objective functions* for small *general graphs*. Further discussion is given in the discussion section.

Related Work

Our work is related to three fields: graph visualization, graph neural networks, and machine learning approaches for graph visualization. The previous work in each field is discussed and summarized in this section.

Graph Visualization

There have been a multitude of graph visualization techniques proposed in the past five decades. The force-directed graph layout models the graph as a physical system in which adjacent nodes are pulled by the attractive force and all other nodes are pushed away by the repulsive force following energy minimization principles (e.g., [2]).

To evaluate the quality of a graph layout, there are some commonly agreed aesthetic metrics, and each of which highlights different visual properties.

However, aesthetics metrics often conflict with each other [4]. As a result, most graph drawing algorithms aim to satisfy only one or two aesthetic criteria. It is widely recognized that a balance of aesthetics yields the best graph layout [5]. Hence, Huang et al. [1] propose a force-directed approach to generate layout with the best compromise between the spring force, crossing angle force, and incident angle force. However, this force-directed approach lacks the flexibility of accommodating arbitrary combination of aesthetics. In this paper, we propose a general deep learning framework for graph drawing that is highly flexible over the choice of the target aesthetics and is capable of improving them simultaneously with adaptively adjusted weights.

Graph Neural Network

Graph Neural Network (GNN) is designed to adapt deep learning to the combinatorial nature of graphs. Convolutional Graph Neural Network (ConvGNNs), as one type of GNN, is employed in our work. Encouraged by the success of Convolutional Neural Networks (CNN) in image data, the convolution operation is extended to graph data in Convolutional Graph Neural Networks (ConvGNN). In general, ConvGNNs are designed to generate node embeddings by aggregating information from their neighboring nodes.

ConvGNNs can be divided into two main streams: spectral-based approaches and spatial-based approaches. In this work, a spatial-based convolutional layer [6] is employed as the building block of our model architecture. The graph convolution layer aggregates neighbors' information for each node while considering the characteristics of the connection between the node and its neighbors. In our case, each message passed from a node's neighbor is weighted according to the direction and magnitude of the difference between two node embeddings.

Machine Learning Approaches for Graph Visualization

The applications of machine learning for graph drawing problem can be classified into three categories: graph drawing with human interaction, graph drawing without human interaction, and the evaluation of graph drawing.

The earliest work [7] in the first category proposed an interactive system which automatically adjusts the objective function and the parameters of a simulated annealing graph drawing method obtained from the user preference. Since then, many other approaches aimed at adjusting the fitness function of genetic algorithms by collecting information from human feedback. Specifically, Spönemann et al. [8] collects human feedback by designing a slider for user to indicate the desired aesthetic criteria and a canvas to select the better layout from a collection of layouts.

The approaches in second category focus on generating graph layouts based upon the graph structure per se or the result from other traditional graph drawing methods. For example, Wang et al. [9] proposed an LSTM-based neural network which learns the layout characteristics from the results of other graph drawing techniques, and draws a graph in a similar drawing style as the specific targeting technique. The limitation is that this approach requires collecting new training data and model re-training for different targeting techniques and for different types of graphs. Kwon and Ma [10] designed a deep generative model, which systematically draws a specific graph in diverse layouts. For each graph, a two-dimensional latent layout space is constructed that allows users to navigate and explore various layouts. Nevertheless, a model trained on one graph is only applicable to generating layouts for that specific graph. DeepGD also falls into this category. *However, our approach is more flexible because once the desired aesthetic criteria are specified and the model is trained, it can be applied to arbitrary types of graphs.* Furthermore, even though the model needs to be re-trained for different aesthetic criteria, the same training data set can be reused regardless of the chosen criteria.

The last category is evaluation of graph layouts with machine learning approaches. Klammler et al. [11] uses a Siamese neural network to identify a more aesthetically pleasing layout from a pair of layouts. A different evaluation approach is proposed by Haleem et al. [12] who designs a CNN-based model to predict various aesthetic metrics for a graph layout without knowing the node and edge coordinates.

Background

This background section introduces some preliminary knowledge and basic concepts about graph drawing and ConvGNN. We represent a graph as $G = (V, E)$, where V and E is the set of vertices and edges. The layout is represented as $\mathbf{x} : V \to \mathbb{R}^2$. The graph theoretic distance between nodes u and v is denoted as d_{uv}.

Graph Drawing

We describe two popular graph drawing algorithms including stress majorization [2] and t-SNE [13] in this section. Both algorithms serves as the bases to define our loss function.

Stress Majorization Stress majorization formulates a graph as a physical system in which there exists a spring between each pair of nodes. The stress energy for G is computed as

$$L_{\text{stress}} = \sum_{\substack{u,v \in V \\ u \neq v}} w_{uv} \left(\|\mathbf{x}_u - \mathbf{x}_v\| - d_{uv} \right)^2, \quad (1)$$

where the weighting factor w_{uv} is typically set to $1/d_{uv}^2$.

Gasner et al. [2] propose a stress majorization algorithm which minimizes the stress by a majorization-based optimization approach. This approach iteratively updates position of nodes as follows.

$$\mathbf{p}_u \leftarrow \sum_{v \neq u} w_{uv} \left(\mathbf{x}_v + d_{uv} \frac{\mathbf{x}_u - \mathbf{x}_v}{\|\mathbf{x}_u - \mathbf{x}_v\|} \right) \Big/ \sum_{v \neq u} w_{uv}. \quad (2)$$

tsNET The t-distributed stochastic neighbor embedding (t-SNE) is a dimensionality reduction algorithm widely used for visualizing various types of data. Kruiger et al. [13] adapted t-SNE into the context of graph visualization and proposed a dimensionality reduction based graph drawing approach called tsNET.

tsNET aims at minimizing the divergence between the graph space and the layout space. Let $d_{ij} \in \mathbb{N}$ denotes the graph theoretic distance between node i and j in a graph of size N. Then, the graph space similarity p_{ij} between node i and j is computed from a normalized Gaussian distribution as follows,

$$p_{ij} = p_{ji} = \frac{p_{j|i} + p_{i|j}}{2N} \quad (3)$$

$$\text{where } p_{j|i} = \frac{\exp(-\frac{d_{ij}^2}{2\sigma_i^2})}{\sum_{\substack{k \\ k \neq i}} \exp(-\frac{d_{ik}^2}{2\sigma_i^2})} \quad (4)$$

and σ_i is a hyper parameter representing the Gaussian standard deviation for node i.

Similarly, in the layout space, the similarity q_{ij} between node i and j is derived from a t-distribution as follows,

$$q_{ij} = q_{ji} = \frac{(1 + ||\mathbf{x}_i - \mathbf{x}_j||^2)^{-1}}{\sum_{\substack{k,l \\ k \neq l}} (1 + ||\mathbf{x}_k - \mathbf{x}_l||^2)^{-1}}. \quad (5)$$

As a result, the optimal layout can be obtained by minimizing the KL-divergence between p_{ij} and q_{ij}, namely

$$L_{\text{t-SNE}} = \sum_{\substack{i,j \\ i \neq j}} p_{ij} \log \frac{p_{ij}}{q_{ij}}. \quad (6)$$

Convolutional Graph Neural Network

The proposed deep learning framework is built upon a Convolutional Graph Neural Network (ConvGNN) which utilizes the convolution operation to aggregate messages from neighboring nodes. Specifically, the spatial-based convolutional network [6] lays the foundation of our model architecture, and is described as follows.

Given a graph with N node and D node features, a ConvGNN takes a node feature matrix $\widetilde{\mathbf{X}} \in \mathbb{R}^{N \times D}$ and an adjacency matrix $\mathbf{A} \in \mathbb{R}^{N \times N}$ as inputs. A hidden layer i with output feature length F^i in the ConvGNN can thus be written as,

$$\begin{cases} \mathbf{H}^0 & = \widetilde{\mathbf{X}}, \\ \mathbf{H}^i & = f(\mathbf{H}^{i-1}, \mathbf{A}), \end{cases} \quad (7)$$

where $\mathbf{H}^i \in \mathbb{R}^{N \times F^i}$ is the hidden node representation after i^{th} layer. The function f is the propagation rule of choice, also known as the the message aggregation function, which determines how the messages are passed between connected nodes.

In general, as shown in Figure 1, all propagation rules follow a basic idea – the hidden representation of each node in the current layer is aggregated from the hidden representation of its neighbors in the previous layer. In this case, after i layers, each node can gather information from nodes that are i graph-theoretic distance away.

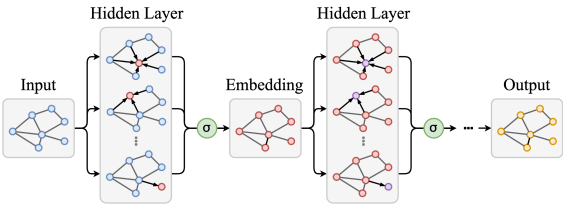

Figure 1: A multi-layer ConvGNN. In each hidden layer, arrow represents the direction of message flow and the change of node color indicates that the hidden node representation is updated by the aggregator function.

At the end of training, ConvGNN learns a set of aggregator functions that define the way of combining feature information from a node's local neighborhood [14]. During the inference phase, the network utilizes learned aggregators to generate meaningful embeddings for unseen graphs.

Methodology

We propose a general deep learning framework for generating graph layouts complying with multiple aesthetics simultaneously. More importantly, the proposed deep learning framework can be easily generalized to adopt most of the existing aesthetics. Our approach is described in this section from the perspectives of training data, model architecture, loss function, and training strategy.

Training Data and Preprocessing

As a data-driven approach, the performance of ConvGNN is impacted by the quality of the training data. A training dataset that is diverse in variety can make the model robust to a variety of structural characteristics in the graphs, while a homogeneous training data can inject bias into the model. Rome graphs[1], as a widely used and publicly available benchmark data set, meets our expectations. Rome contains 11,534 undirected graphs each of which consists of 10 to 100 nodes with significantly different graph structures. Therefore, our model can handle many unseen graphs regardless of their structure, thanks to a large number of graphs with different structures seen by the model during training.

[1]http://www.graphdrawing.org/data.html

Since our deep learning framework is based on ConvGNN in which the edges between nodes in the input graphs defines the flow of message passing, we made two modifications to the Rome graphs in order to facilitate the message flow within the graph. Firstly, we add virtual edges for any pairs of unconnected nodes. As mentioned in Section Convolutional Graph Neural Network, every node will gather information from connected nodes during convolution operation. Thus, adding virtual edges allows information to propagate through longer distances quickly even in a shallow neural network [6], as messages can be directly passed between any pair of nodes. In the mean time, the original graph structure is still retained by encoding the real edge information as edge features. Secondly, even though Rome graphs are undirected, the ConvGNN assumes its input graphs are directed. During propagation, the message flow direction is determined by the edge direction. To avoid information asymmetry, we add duplicated edges in the opposite direction of the original edges for the connected nodes. After applying these two modifications, the Rome graphs all become complete graphs without self-loop such that there exist two edges between any pairs of nodes but in opposite directions.

The node features of graphs can also affect the learned aggregator functions. The input node features $\widetilde{\mathbf{X}}$ in (7) is defined as the initial node position, which allows the model to find high-quality layouts easier given a reasonable starting point as a hint. We employ two initialization strategies. The first strategy is randomly sampling $\widetilde{\mathbf{X}} \in \mathbb{R}^{N\times 2}$ from a uniform distribution in $[0,1]^2$. The second strategy is initializing $\widetilde{\mathbf{X}}$ as PivotMDS [3] layouts.

The edge features play a key role of message passing in our model. The propagation rule [6] in our work aggregates messages from each node's neighbors based on the information passed from the connecting edges. Accordingly, we encode the information about the original graph structure in an edge feature matrix $\mathbf{Z} \in \mathbb{R}^{N(N-1)\times 1}$ because the structural information of the original graphs is not represented in the adjacency matrix $\mathbf{A}$. For an edge between node u and node v, the edge attribute is specified as the graph-theoretic distance d_{uv}.

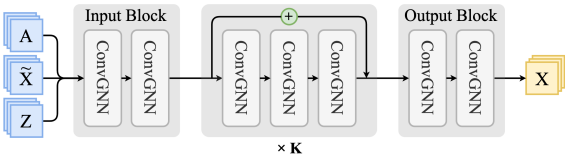

Figure 2: The model architecture of DeepGD.

Model Architecture

The model we propose is a ConvGNN-based deep learning framework which can generate graph layouts complying with multiple aesthetic criteria simultaneously. The high-level idea is that with the graph structural characteristic captured by the convolution operation, the model predicts the node positions such that the resulted layout follows the aesthetic criteria specified by the loss function.

The input to our model includes an adjacency matrix $\mathbf{A}$, a node feature matrix $\widetilde{\mathbf{X}}$, and an edge feature matrix $\mathbf{Z}$. Overall, the model is composed of an input block, a sequence of residual blocks, and an output block (see Figure 2). To be specific, the input block processes and transforms the input data; a sequence of residual blocks is the key component for generating meaningful hidden node representations; and the output block is responsible for projecting the hidden node representation generated by the last residual block to two-dimensional space.

Propagation Rule As stated in Section Convolutional Graph Neural Network, the propagation rule defines the way of aggregating messages from local neighborhood of a node. Since the original graph structure information is only maintained in the edge feature matrix $\mathbf{Z}$, our aggregator function should take into account the corresponding edge feature while aggregating messages. Therefore, we incorporate an edge feature network into the aggregator function such that the message passed between nodes also depends on their corresponding edge connection. This type of aggregation function is originally proposed by Gilmer et al. [6].

Our propagation rule is described in Algorithm 1. For line 2, $\mathbf{m}_v$ carries the messages aggregated from v's neighbors in a complete graph G. Specifically, the message $\mathbf{h}_u$ from each neighbor u is weighted by a transformation matrix which is generated by a learned edge feature network ϕ

Algorithm 1: Message Aggregation

Input: Graph $G(V,E)$; node embeddings $\mathbf{h} = \{\mathbf{h}_v | v \in V\}$; edge features $\mathbf{e} = \{\mathbf{e}_{uv} | (u,v) \in E\}$; weight matrix $\mathbf{W}$; edge feature network ϕ

Output: Updated node embeddings $\{\tilde{\mathbf{h}}_v | v \in V\}$

1 **for** $v \in V$ **do**
2 $\quad \mathbf{m}_v \leftarrow \frac{1}{|\mathcal{N}(v)|} \sum_{u \in \mathcal{N}(v)} \phi(\mathbf{e}_{uv}) \cdot \mathbf{h}_u$
3 $\quad \tilde{\mathbf{h}}_v \leftarrow \mathbf{W} \cdot \mathbf{h}_v + \mathbf{m}_v$
4 **end**

according to the corresponding edge information. As a result, this allows the model to judge the importance of the message from a specific node u even if u is not the close neighbor of v in the original graph. After weighting the messages from all neighbors, the message is aggregated and normalized by the number of neighbors. Then, the aggregated message $\mathbf{m}_v$ is used to update the hidden node embedding $\tilde{\mathbf{h}}_v$ on line 3. The entire message aggregation function represents the mathematical operation of one ConvGNN layer.

Edge Feature Network In order to take the edge information into consideration during message aggregation, we design an edge feature network ϕ to process the edge information. We train a separate edge feature network for each ConvGNN layer because the edge information should be treated differently depending on the depth of that ConvGNN layer. The edge feature network is shared among all nodes within a single ConvGNN layer.

Specifically, our edge feature network ϕ comprises of two dense layers. It predicts a projection matrix $\mathbf{T}_{uv} \in (-1,1)^{F^i \times F^{i+1}}$ based on the edge feature vector $\mathbf{z}_{uv}$, where F^i represents the node feature length in layer i. Thus, the projection matrix $\mathbf{T}_{uv}$ serves as a weighting factor for neighbor's message. Additionally, *tanh* is used as the output activation function to confine each element in $\mathbf{T}_{uv}$ inside the range between -1 and 1.

Residual Blocks The backbone of our model is composed by a sequence of residual blocks which is the key component for generating hidden node embedding. Since the graph convolution

layer at a shallower depth of the network can capture low-level features and the layer at a deeper depth can learn a higher-level node representation, the residual connection can help to combine different levels of information together.

Specifically, each residual block contains 3 hidden layers, and the input node representation of $(k-1)^{th}$ residual block will be directly added to the input node representation of k^{th} residual block (see Figure 2). Thus, the node representation at different levels can directly pass through the model pipeline without hindrance. Moreover, skip connections facilitate the back-propagation of gradients so that vanishing gradient problem is alleviated.

Inspired by the stress majorization approach [2], we add two additional input edge features for each residual block. In the stress majorization algorithm (see (2)), node positions are iteratively updated by taking into account the directions of edges between each pair of node from last iteration. The reason being, the direction of edges during iteration is an important information for minimizing stress. Therefore, in addition to the original edge feature $\mathbf{d}_{uv}$, we add the direction of edges $\frac{\mathbf{h}_u-\mathbf{h}_v}{\|\mathbf{h}_u-\mathbf{h}_v\|}$ and the Euclidean distance $\|\mathbf{h}_u-\mathbf{h}_v\|$ between each pair of nodes as two additional edge features for each residual block. These two additional edge features for block k^{th} are calculated according to the hidden node representation output from residual block $(k-1)^{th}$. In this case, if we regard each residual block as one iteration in (2), each block attempts to minimize the loss function by taking the direction and edge length from last block into consideration. The entire model architecture including the residual blocks with additional edge features is described in Algorithm 2.

Loss Function

Our loss function design largely depends on the desired aesthetics. That is, the desired aesthetics are specified in the loss function such that minimizing the loss function will thus optimize the aesthetic metrics. This enables our deep learning framework to generalize to most of the known aesthetic metrics.

For the loss function, we have explored to optimize stress in (1), t-SNE in (6), and three other commonly agreed aesthetic metrics including minimum angle, edge length variation and node occlusion.

Minimum Angle

The minimum angle is the sharpest angle formed by any two edges that meet at a common vertex of the drawing [15]. If a node v's minimum angle is maximized, all incident edges at node v will form the same angle around node v. Therefore, maximizing the minimum angle can help to generate aesthetically pleasing layouts. Our loss for maximizing minimum angle is computed as

$$L_{\text{angle}} = \sum_{v\in V} \sum_{\theta_v^{(i)}\in \text{angles}(v)} \left| \frac{2\pi}{\deg(v)} - \theta_v^{(i)} \right|, \qquad (8)$$

where angle(v) is the incident angles over node v and deg(v) denotes the node degree of v.

Edge Length Variation

The edge length variation is the standard deviation of edge length for all edges in a graph [12]. If it is minimized for a graph, the length of all edges in the graph tends to be equal. From the aesthetic perspective, a graph layout with a smaller edge length variation is always preferable. The loss function for minimizing edge length variation is defined as,

$$L_{\text{edge}} = \frac{1}{|E|} \sum_{(u,v)\in E} \frac{(l_{uv}-\bar{l})^2}{\bar{l}^2}, \qquad \begin{cases} l_{uv} = \|\mathbf{x}_u - \mathbf{x}_v\| \\ \bar{l} = 1 \end{cases} \qquad (9)$$

where l_{uv} denotes the edge length between u and v, and $\bar{l}$ is the expected edge length.

Node Occlusion

Node occlusion or node overlapping measures how densely the nodes are clustered. The global structure of graph is clearer with smaller node occlusion. Inspired by Haleem et al. [12], who defines node occlusion as the total pairs of nodes closer than a threshold, we design a smooth version of node occlusion by replacing hard threshold function with a exponential function as follows,

$$L_{\text{node occlusion}} = \sum_{\substack{u,v\in V \\ u\neq v}} e^{-\|\mathbf{x}_u-\mathbf{x}_v\|}. \qquad (10)$$

Multi-objective Training Strategy

In order to consider multiple aesthetic criteria at the same time, we compute a weighted sum of loss components derived from corresponding aesthetic metrics as our multi-objective loss function.

Algorithm 2: DeepGD

Input: Graph $G(V,E)$; initial node features $\tilde{\mathbf{x}} = \{\tilde{\mathbf{x}}_v | v \in V\}$; edge features (graph theoretical distances) $\mathbf{d} = \{d_{uv} | (u,v) \in E\}$; total number of blocks B; number of layers in each block L; weight matrices $\mathbf{W}^{(b,l)}$; edge feature network $\phi^{(b,l)}$, $\forall b \in \{1,...,B\}, \forall l \in \{1,...,L\}$.

Output: Node positions $\mathbf{x} = \{\mathbf{x}_v | v \in V\}$

$\mathbf{h}^{(1,0)} \leftarrow \tilde{\mathbf{x}}$
$\mathbf{h}^{(1,1)} \leftarrow \text{ReLU}(\text{Message Aggregation}(G, \mathbf{h}^{(1,0)}, \mathbf{d}, \mathbf{W}^{(1,1)}, \phi^{(1,1)}))$
$\mathbf{h}^{(1,2)} \leftarrow \text{ReLU}(\text{Message Aggregation}(G, \mathbf{h}^{(1,1)}, \mathbf{d}, \mathbf{W}^{(1,2)}, \phi^{(1,2)}))$
$\mathbf{h}^{(2,0)} \leftarrow \mathbf{h}^{(1,2)}$
for $b \leftarrow 2...B-1$ **do**
  $\mathbf{e}^{(b)} \leftarrow \left\{ \left(d_{uv}, \text{direction}(\mathbf{h}_u^{(b,0)}, \mathbf{h}_v^{(b,0)}), \text{distance}(\mathbf{h}_u^{(b,0)}, \mathbf{h}_v^{(b,0)}) \right) \right\}$
  for $l \leftarrow 1...L$ **do**
    $\mathbf{h}^{(b,l)} \leftarrow \text{ReLU}(\text{Message Aggregation}(G, \mathbf{h}^{(b,l-1)}, \mathbf{e}^{(b)}, \mathbf{W}^{(b,l)}, \phi^{(b,l)}))$
  end
  $\mathbf{h}^{(b+1,0)} \leftarrow \mathbf{h}^{(b,L)} + \mathbf{h}^{(b,0)}$
end
$\mathbf{h}^{(B,1)} \leftarrow \text{ReLU}(\text{Message Aggregation}(G, \mathbf{h}^{(B,0)}, \mathbf{d}, \mathbf{W}^{(B,1)}, \phi^{(B,1)}))$
$\mathbf{h}^{(B,2)} \leftarrow \text{Message Aggregation}(G, \mathbf{h}^{(B,1)}, \mathbf{d}, \mathbf{W}^{(B,2)}, \phi^{(B,2)})$
$\mathbf{x} \leftarrow \mathbf{h}^{(B,2)}$

The multi-objective loss function for epoch t is defined as

$$L^{(t)} = \sum_{k=1}^{n} \alpha_k^{(t)} L_k^{(t)}, \tag{11}$$

where $\alpha_k^{(t)}$ (s.t. $\sum_k \alpha_k^{(t)} = 1$) represents the weight factor for k^{th} component and $L_k^{(t)}$ is the average loss value for the k^{th} component at epoch t.

Different combinations of weight factors will result in different optimization results. Intuitively, the weight factors should be specified based on human preferences. In other words, if α_1 is greater than α_2, more emphasize is put on optimizing the first loss component. However, the weight factors do not always directly reflect the human preference of each loss component during optimization, because different loss components may have different numerical scales. So, one challenge is how to determine the weight factor α for each aesthetic while considering both the human preference and the difference in the numerical scale.

We discuss two multi-objective training strategies to help find a compromise between multiple aesthetics in the following.

Adaptive Weight Adaptive weight is to adaptively adjust α with respect to the numerical scale of corresponding loss component for each epoch. The weight factor for k^{th} components at epoch t is defined as

$$\alpha_k^{(t)} = \frac{\frac{\gamma_k}{L_k^{(t-1)}}}{\sum_{l=1}^{n} \frac{\gamma_l}{L_l^{(t-1)}}}, \tag{12}$$

where γ_k represents the importance factor for k^{th} component and $L_k^{(t-1)}$ the average loss value for k^{th} component in epoch $t-1$.

The importance factor basically indicates the human preference for each loss component without considering the difference between numerical scale for multiple loss components. The basic idea is that, we normalize each component by its numerical scale $L_k^{(t)} / L_k^{(t-1)}$ and then multiply it with their user-specified human preference γ_k. In this case, the weight factor $\alpha_k^{(t)}$ will be decreased if the numerical scale of k^{th} component is greater than that of other components, and vice versa. Thus, the weight factor for each epoch takes into account both the human preference specified as importance factor and the numerical scale of all loss components from previous epoch.

SoftAdapt by Importance The SoftAdapt by Importance strategy is derived from the SoftAdapt technique proposed by Heydari et al. [16]. SoftAdapt adaptively sets the weight factor for each loss component according to their recent rate of descent. Since the original SoftAdapt does not consider different numerical scales for loss components, we improve it by taking into account the relative scale of each loss components and incorporating the concept of importance factor γ_k. Specifically, the weight factor $\alpha_k^{(t)}$ for k^{th} loss component during epoch t is computed as,

$$\alpha_k^{(t)} = \frac{\frac{\gamma_k}{L_k^{(t-1)}} \exp(\beta^* s_k^{(t)})}{\sum_{l=1}^{n} \frac{\gamma_l}{L_l^{(t-1)}} \exp(\beta^* s_l^{(t)})}, \tag{13}$$

$$s_k^{(t)} = \underset{k\in\{1\ldots n\}}{\text{normalize}_{L1}} \left(\frac{\tilde{L}_k^{(t)} - \tilde{L}_k^{(t-1)}}{\tilde{L}_k^{(t-1)}} \right), \tag{14}$$

$$\tilde{L}^{(t)} = \begin{cases} L^{(1)} & t = 1 \\ \tau \cdot \tilde{L}^{(t-1)} + (1-\tau) \cdot L^{(t)} & t > 1, \end{cases} \tag{15}$$

where β is the sensitivity factor that controls how responsive the weight adjustment is to the rate of descent; $\tilde{L}_k$ is the smoothed version of L_k after exponential smoothing with smoothing factor τ; and $s_k^{(t)}$ is the descending rate of k^{th} loss component at epoch t.

Evaluation

We conducted extensive experiments to evaluate our approach quantitatively and qualitatively. This section describes the experimental details and evaluation results.

Experimental Setup

The proposed deep learning framework is implemented with PyTorch and PyTorch Geometric library[2]. Besides, in all of our experiments, the models were trained on a single Tesla V100 GPU with memory of 16 GB.

As mentioned in Section Training Data and Preprocessing, our experiments are conducted over Rome dataset, which contains 11,534 undirected graphs each consisting of 10 to 100 nodes. We excluded two disconnected graphs from the dataset. Among the remaining graphs, we randomly selected 10,000 graphs as training examples, 1,000 graphs as testing examples, and 532 graphs as validation examples. Validation data serve the purpose of hyperparameter tuning. We only report the model performance on testing data in this paper.

Regarding model architecture, we conducted a series of experiments/ablation studies to investigate how does each component contribute to the model performance. Those experiments explored the effect of removing residual connection, removing all the virtual edges, removing different edge features, different numbers of neurons for each residual block, and different numbers of hidden layers in the edge feature network. Given the word limits, our model configuration and detailed experimental results for comparing different model architecture are presented in the *supplemental material.*

Quantitative Evaluation

In this section, we quantitatively assess the performance of DeepGD. For comparison, PivotMDS [3] and the stress majorization [2] algorithm *neato* implemented in Graphviz[3] are chosen as baseline methods.

In addition to Graphviz and PivotMDS, inspired by Tsitsulin et al. [17] who propose to use t-SNE to project high dimensional node embedding to 2D for graph visualization, we implemented GNN+tSNE and GNN+UMAP. These two baselines used t-SNE and UMAP, respectively, to project the latent node embedding generated by the last hidden layer of DeepGD onto 2-D space.

To evaluate the relative difference, we computed Symmetric Percent Change (SPC) with respect to Graphviz as follows.

$$\text{SPC} = 100\% \times \frac{1}{N_t} \sum_{i=0}^{N_t} \frac{D_i - G_i}{max(D_i, G_i)}, \tag{16}$$

where D_i and G_i denotes the same evaluation metric computed on two layouts generated by a certain model (i.e., DeepGD or PivotMDS) and Graphviz respectively, for the i^{th} test graph; N_t is the total number of test graphs. SPC holds a nice property that it ranges from -100% to 100%, thus the value of SPC can be interpreted as how many percent G_i outperforms D_i.

Stress Optimization Since minimizing stress is known to be an overall effective approach to

[2]https://github.com/rusty1s/PyTorch_geometric

[3]https://graphviz.org/

improve many aesthetic aspects of graph layouts, we first explored the effectiveness and efficiency of optimizing stress only. In other words, the loss function of DeepGD in this subsection contains only one component which corresponds to stress.

Table 1: To assess the DeepGD with stress only, average stress is computed over 1000 test graphs; and the stress SPC (smaller is better) represents the relative difference in stress compared to Graphviz.

Models	Avg. Stress	Stress SPC w.r.t. Graphviz
DeepGD + Random Init	**246.57**	1.10%
DeepGD + PivotMDS Init	**239.73**	**-5.43%**
Graphviz [2]	251.93	0.00%
PivotMDS [3]	372.06	36.53%
GNN + t-SNE [18]	483.01	56.98%
GNN + UMAP [19]	379.88	41.40%

As shown in Table 1, DeepGD obtains better stress than Graphviz on average, no matter what the initialization strategy is. For stress SPC, DeepGD initialized with PivotMDS outperforms Graphviz by 5.43% whereas DeepGD with random initialization achieves a comparable performance as Graphviz. We observe that DeepGD initialized randomly achieves a positive stress SPC but with lower stress than Graphviz. The potential reason is that DeepGD usually outperforms Graphviz for large graphs but obtains higher stress than Graphviz for drawing small graphs. Besides, if we compare the three alternative baselines (PivotMDS, GNN+t-SNE, and GNN+UMAP) with Graphviz, their performance are much worse. In particular, projecting the output of the last hidden layer using t-SNE or UMAP is inferior to DeepGD, showing the importance of end-to-end training by including the final nonlinear layer. In conclusion, regarding stress, DeepGD significantly outperforms PivotMDS, GNN+ t-SNE, and GNN+ UMAP and is 5.43% better than Graphviz on average.

To evaluate the stability and robustness of DeepGD, we performed 11-fold cross validation over 11,000 Rome graphs. With random initialization, DeepGD achieves stress SPC of 1.10% on average. Additionally, the median stress SPC for each folds are only slightly above zero, which indicates that DeepGD with stress performs as good as Graphviz regardless of the outliers. We can conclude that DeepGD is able to consistently perform well even with the variation in the training data.

Optimization with Two Aesthetics We assess the model capability of optimizing and compromising between two aesthetics metrics in this section. Since stress can improve the overall aesthetic quality, we conducted experiments for combining stress and one other aesthetic metrics including t-SNE, edge length variation, minimum angle, and node occlusion respectively. The following three models are trained by adaptive weight strategy mentioned in Section Adaptive Weight with our choice of importance factors.

Stress + Minimum Angle Loss
In this DeepGD model, the stress and minimum angle loss are optimized at the same time. From the first row of Table 2, DeepGD with both initialization strategies outperforms Graphviz by at least 17% considering minimum angle loss. However, the stress SPC of DeepGD with random initialization and PivotMDS initialization increases by 16.49% and 9.67%, respectively, compared to the DeepGD with stress only in first two rows of Table 1. The potential reason is that minimum angle and stress are conflicting criterion so that there is an unavoidable trade-off between them.

Stress + Edge Length Variation
We also conducted experiment to optimize stress and edge length variation simultaneously. In the second row of Table 2, the edge length variation SPC of -20.28% and -32.92% shows that DeepGD can draw a graph with much more uniform edge length than Graphviz on average. It is interesting to see that DeepGD with both initialization strategies still obtains a reasonably good stress, even though it has to compromise between stress and edge length variation. This is because stress will be minimized when the layout distance between each node pairs equals to their graph theoretic distance. Therefore, minimizing edge length variation could also help with minimizing stress.

Stress + Node Occlusion
From the third row of Table 2, we observe that the performances of DeepGD with two initialization strategies are slightly different. With random initialization, DeepGD obtained 3.01% higher node occlusion loss than Graphviz on average. Given that Graphviz only optimizes stress, it indicates

Table 2: The quantitative evaluation of DeepGD with multiple aesthetics. Each row represents one DeepGD model with our choice of aesthetics, which are weighted linear combinations of different loss components (stress, occlusion etc.). Negative SPC indicates that DeepGD outperforms Graphviz with certain percentage regarding that specific metric.

Importance Weighting Factors of Loss Components					**Metric SPC w.r.t. Graphviz**									
					Random Initialization					**PivotMDS Initialization**				
Stress	Angle	Edge	Occlusion	t-SNE	Stress	Angle	Edge	Occlusion	t-SNE	Stress	Angle	Edge	Occlusion	t-SNE
0.6	0.4				17.59%	**-17.10%**	–	–	–	4.24%	**-22.66%**	–	–	–
0.8		0.2			4.88%	–	**-20.28%**	–	–	4.64%	–	**-32.92%**	–	–
0.6			0.4		0.72%	–	–	3.01%	–	**-4.67%**	–	–	**-2.70%**	–
0.7				0.3	1.88%	–	–	–	**-5.18%**	**-3.84%**	–	–	–	**-12.09%**
0.5	0.1		0.1	0.3	4.19%	**-0.60%**	–	0.76%	**-7.29%**	**-1.53%**	**-7.36%**	–	**-2.09%**	**-14.48%**

that optimizing stress can help to avoid node occlusion as well. With PivotMDS initialization, we outperform Graphviz regards both stress and node occlusion by 4.67% and 2.70%, respectively. Overall, this experimental result again proves that DeepGD has the capability and flexibility of optimizing most of aesthetics.

Stress + t-SNE

To optimize stress and t-SNE simultaneously, the loss function in this DeepGD model is the weighted average of stress and t-SNE (see Section Multi-objective Training Strategy). The quantitative measurement shown in the fourth row of Table 2 indicates that DeepGD with PivotMDS outperforms Graphviz regarding both t-SNE and stress. Also, with random initialization, we achieve better t-SNE and comparable stress than Graphviz. This result again shows that PivotMDS initialization indeed can help to improve the performance.

Optimization with Four Aesthetics To assess the model's capability of optimizing more than two aesthetics, we conducted experiments for optimizing stress, t-SNE, minimum angle, and node occlusion simultaneously. The quantitative evaluation is presented in the fifth row of Table 2. By initializing DeepGD with PivotMDS, we achieved outstandingly better results than Graphviz from four different aesthetic perspectives. This result clearly demonstrates that DeepGD indeed can draw arbitrary graphs by balancing among multiple aesthetics with the help of adaptive importance training strategy, even though t-SNE and minimum angle loss somehow contradicts with stress. More importantly, since a layout method that considers multiple aesthetics simultaneously is more likely to generate a visually pleasing graph layout [1], DeepGD might be aesthetically more attractive to human.

Qualitative Evaluation

For qualitative evaluation shown in Table 3, we only present the results for DeepGD with PivotMDS initialization due to the significantly better result comparing to DeepGD with random initialization. Among all 10 methods, PivotMDS, GNN+t-SNE, and GNN+UMAP each have obviously identifiable weaknesses that make them not as visually pleasing as the rest, which is consistent with the results shown in Table 1. In contrast, DeepGD generated layouts on all sample graphs are at least comparably good to Graphviz. We also observe that, for DeepGD models, the specific loss components involved indeed improve the corresponding visual aspects of the resulting layouts.

Multi-Objective Training Strategy

Different aesthetic metrics measure different visual properties, and optimizing one metric sometimes cause others to worsen. For example, if we try to optimize stress and minimum angle simultaneously, the minimum angle metric will be worsened when we put more effort in optimizing stress. Hence, in this multi-objective settings, our goal is to find the Pareto optimal between two aesthetics such that no change can be made to improve both aesthetics. The question we need to answer in this section is which training strategies can help us get closer to the Pareto optimal.

For the two multi-objective training strategies proposed in Section Multi-objective Training Strategy, their effectiveness is assessed by

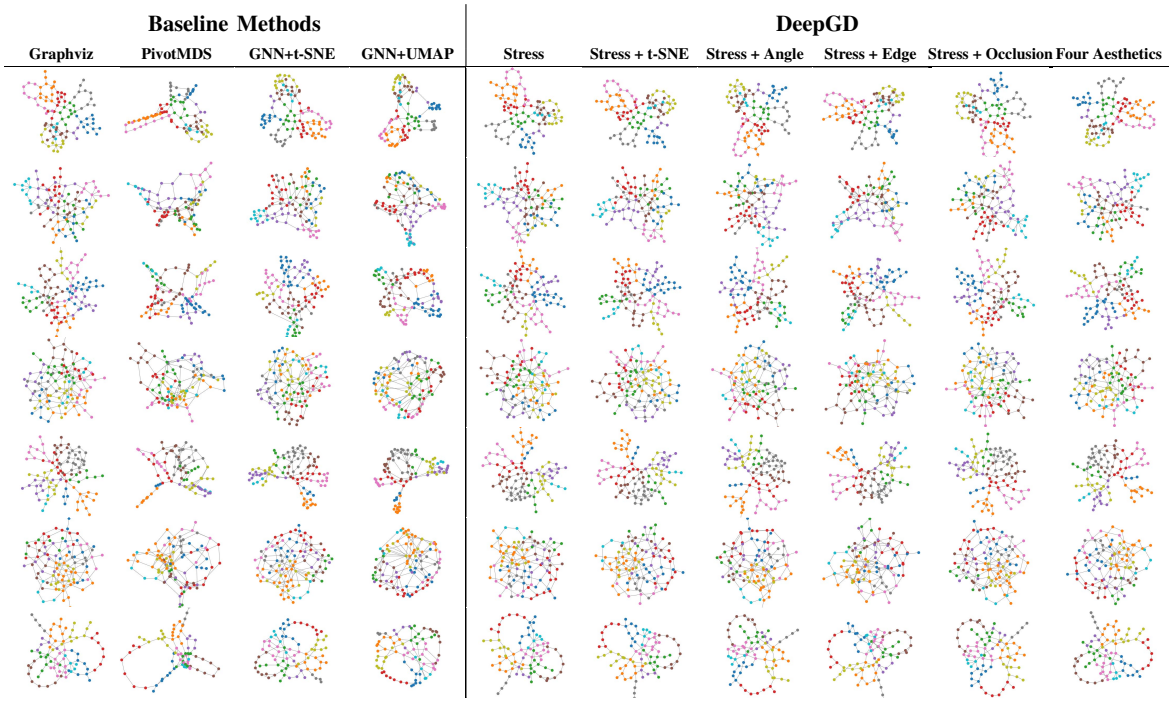

Figure 3: The qualitative evaluation of DeepGD with PivotMDS initialization. Each row corresponds to one test graph and each column represents one method. The colors of the nodes represent their community within the graph.

comparing against the simplest training strategy of setting the fixed weight factor for each epoch. Taking stress and minimum angle as an example, the Pareto frontier lines are drawn for these three training strategies in Figure 4.

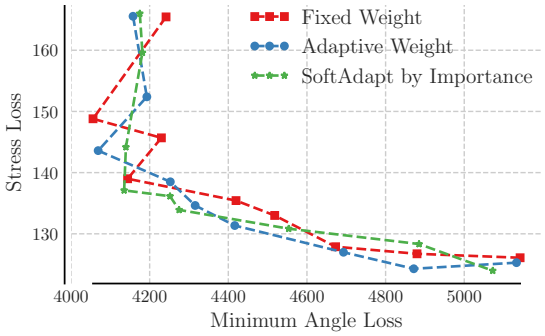

Figure 4: The Pareto frontier for different training strategies. The points on each line represents one model with different importance/weight factors. The connections indicate that the starting point of that connection has smaller stress weight/importance than its ending point.

As our optimization goal is to minimize both the minimum angle loss and the stress loss, the Pareto optimal should locate at the bottom left of the Pareto frontier plot. It is obvious that, the adaptive weight and SoftAdapt by importance is closer to the Pareto optimal for most of the points, in comparison with the fixed weight strategy. In conclusion, in this multi-objective optimization problem, we are more likely to achieve the Pareto optimal with adaptive weight and SoftAdapt by importance.

Computation Time

For training, it takes 220 seconds on average for each epoch. According to our observations, for all models we trained, the validation loss converges after around 400 epochs. After training, the average testing time per graph is 0.049 seconds. In a word, once trained, DeepGD can generate layouts in real-time for any modest sized graphs to optimize the desired aesthetics.

Scalability

Even though the main focus of this paper is to draw small graphs ($\leq$ 100 nodes), we still estimated the model capability on large graphs from

SuiteSparse Matrix Collection[4] with thousands of nodes. Using 16GB of GPU memory, DeepGD can draw graphs with at most 4000 nodes. For 8 large graphs with 3500-4000 nodes, DeepGD can draw them in 28.14 seconds on average.

In terms of the visualization quality, the performance of DeepGD cannot be guaranteed for large graphs because DeepGD is only trained on small graphs. Specifically, we observed that DeepGD performs significantly better than Graphviz for some large graphs (see Figure 5) but may fail to outperform Graphviz for some other large graphs, regardless of the graph size.

We note that DeepGD was not trained on large graphs, due to the limitation of long training time on these graphs. Nevertheless, we believe it is possible to scale DeepGD to very large graphs by considering only the original edges in the graph plus a sparse subset of node pairs that are not neighbors. We are currently working along this direction.

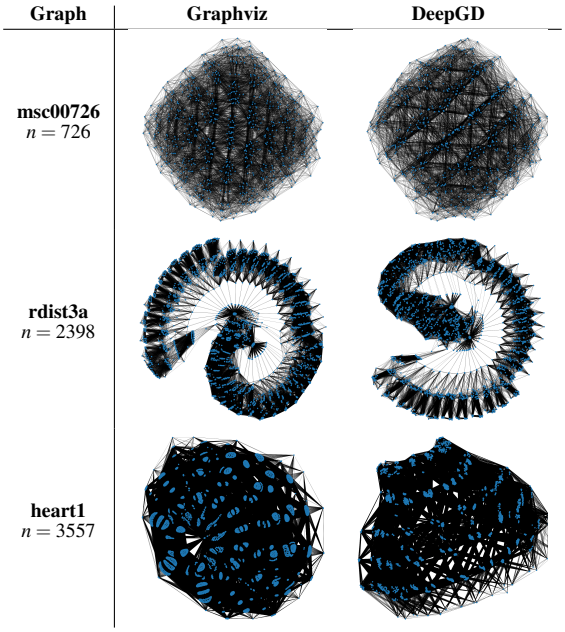

Figure 5: The qualitative comparison between Graphviz and DeepGD over three large graphs, where n denotes the number of nodes in the graph.

Discussions

The evaluation section substantiates that the DeepGD framework can generate visually pleasing layouts for unseen graphs by optimizing certain aesthetic metrics simultaneously.

Choice of Importance Factor

In multi-objective settings, the importance factor is specified by the user and is supposed to reflect human preferences on different loss components. However, it's still difficult to quantify the abstract human preference into a single number, even though the difference in numerical scale is already automatically taken into account by the adaptive weight training strategy. We plan to explore models that can learn human preference automatically.

Training Data Representativity

A common issue of deep learning models is that the representativity of training data constrains the model capability of generalizing to unseen data. If the model does not see a specific type of graph during training, it is challenging for the model to draw that type of graph during inference. For example, we observed that DeepGD cannot draw star graph well. We suspect the reason is that Rome does not contain graphs with extremely large node degree. Therefore, we conducted an experiment of training DeepGD with Rome and North[5] graph dataset together. Since North data includes many graphs with large node degree, DeepGD trained by both Rome and North can draw the star graph properly. In a word, if the training data is not representative enough, the generalization of model could be affected.

Conclusions

We propose a novel ConvGNN-based framework, DeepGD, which can generate graph layouts such that any desired combination of aesthetics can be complied with, as long as the aesthetics can be expressed as differentiable cost functions. To balance among multiple aesthetics, two adaptive training strategies for graph drawing are proposed to dynamically adjust the weight factors. It is worth mentioning that optimizing any aesthetic metrics only requires a redefined loss function without any algorithmic change. Compared to other deep learning based graph drawing algorithms, DeepGD only needs to be trained once and can subsequently be applied to draw arbitrary types of graphs.

[4]https://sparse.tamu.edu/

[5]http://www.graphdrawing.org/data.html

We explored the effectiveness and efficiency of DeepGD by optimizing stress, minimum angle, edge length variation, t-SNE, and node occlusion. The quantitative and qualitative evaluation demonstrate that DeepGD outperforms the baseline models on all of the five aesthetic metrics we experimented, especially for the DeepGD initialized by PivotMDS with four aesthetics.

REFERENCES

1. W. Huang, P. Eades, S.-H. Hong, and C.-C. Lin, "Improving multiple aesthetics produces better graph drawings," *Journal of Visual Languages & Computing*, vol. 24, p. 262–272, 08 2013.
2. E. R. Gansner, Y. Koren, and S. North, "Graph drawing by stress majorization," *Graph Drawing Lecture Notes in Computer Science*, p. 239–250, 2005.
3. U. Brandes and C. Pich, "Eigensolver methods for progressive multidimensional scaling of large data," *LNCS*, vol. 4372, 09 2006.
4. H. Nascimento, P. Eades, and C. Mendonça, "A multi-agent approach using a-teams for graph drawing." *Proceedings of the 9th International Conference on Intelligent Systems*, pp. 39–42, 01 2000.
5. W. Didimo, G. Liotta, and S. Romeo, "Topology-driven force-directed algorithms," *Proc. of GD 2010*, vol. 6502, pp. 165–176, 09 2010.
6. J. Gilmer, S. Schoenholz, P. Riley, O. Vinyals, and G. Dahl, "Neural message passing for quantum chemistry," 04 2017.
7. C. F. X. M. Neto and P. Eades, "Learning aesthetics for visualization," *Anais do XX Semin´ario Integrado de Software e Hardware*, p. 76–88, 1993.
8. M. Spönemann, B. Duderstadt, and R. von Hanxleden, "Evolutionary meta layout of graphs," vol. 8578, pp. 16–30, 07 2014.
9. Y. Wang, Z. Jin, Q. Wang, W. Cui, T. Ma, and H. Qu, "Deepdrawing: A deep learning approach to graph drawing," *IEEE Transactions on Visualization and Computer Graphics*, vol. PP, pp. 1–1, 08 2019.
10. O.-H. Kwon and K.-L. Ma, "A deep generative model for graph layout," *IEEE Transactions on Visualization and Computer Graphics*, vol. 26, 01 2020.
11. T. Mchedlidze, A. Pak, and M. Klammler, "Aesthetic discrimination of graph layouts," *Journal of Graph Algorithms and Applications*, vol. 23, pp. 525–552, 01 2019.
12. H. Haleem, Y. Wang, A. Puri, S. Wadhwa, and H. Qu, "Evaluating the readability of force directed graph layouts: A deep learning approach," *IEEE computer graphics and applications*, vol. 39, pp. 40–53, 07 2019.
13. 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*, vol. 36, no. 3, p. 283–294, 2017.
14. W. Hamilton, R. Ying, and J. Leskovec, "Inductive representation learning on large graphs," 06 2017.
15. H. Purchase, "Metrics for graph drawing aesthetics," *Journal of Visual Languages & Computing*, vol. 13, pp. 501–516, 10 2002.
16. A. A. Heydari, C. A. Thompson, and A. Mehmood, "Softadapt: Techniques for adaptive loss weighting of neural networks with multi-part loss functions," *CoRR*, vol. abs/1912.12355, 2019. [Online]. Available: http://arxiv.org/abs/1912.12355
17. A. Tsitsulin, D. Mottin, P. Karras, and E. Müller, "Verse: Versatile graph embeddings from similarity measures," 03 2018.
18. L. Van Der Maaten and G. Hinton, "Viualizing data using t-sne," *Journal of Machine Learning Research*, vol. 9, pp. 2579–2605, 11 2008.
19. L. McInnes, J. Healy, and J. Melville, "UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction," *ArXiv e-prints*, Feb. 2018.

Xiaoqi Wang is a Ph.D. student in computer science at the Ohio State University. She received a Bachelor of Science in data analytics from the Ohio State University and a Master of Science in data science from Columbia University. Her research interests include visualization and machine learning. Contact her at wang.5502@osu.edu.

Kevin Yen is a Research Engineer at Yahoo Research. He works on various system design and development that involves applying Natural Language Processing, Computer Vision, and machine learning to products and services. Contact him at kevinyen@verizonmedia.com.

Yifan Hu is a Senior Director of Research at Yahoo Research. Prior to joining Yahoo, he worked at ATT Labs, Wolfram Research, and Daresbury Laboratory. He received his B.S. and M.S. in applied mathematics from Shanghai Jiao-Tong University, and Ph.D. in optimization from Loughborough University. His research interests include data mining, machine learning and visualization. He is a co-author of a number of best papers, including the 2017 ICDM 10-year highest impact award paper on recommender systems. He is the author of a number of functions in Mathematica and contributes to the open source software Graphviz. Contact him at yifanh@gmail.com.

Han-Wei Shen is a Full Professor at The Ohio State University. He is currently an Associate Editor in Chief for IEEE Transactions on Visualization and Computer Graphics, and a member of IEEE Visualization Academy. His primary research interests are scientific visualization and computer graphics. Professor Shen is a winner of National Science Foundation's CAREER award and US Department of Energy's Early Career Principal Investigator Award. He received his BS degree from Department of Computer Science and Information Engineering at National Taiwan University in 1988, the MS degree in computer science from the State University of New York at Stony Brook in 1992, and the PhD degree in computer science from the University of Utah in 1998. From 1996 to 1999, he was a research scientist at NASA Ames Research Center at Mountain View, California. Contact him at shen.94@osu.edu.