DeepDrawing: A Deep Learning Approach to Graph Drawing

Yong Wang, Zhihua Jin, Qianwen Wang, Weiwei Cui, Tengfei Ma and Huamin Qu

Abstract—Node-link diagrams are widely used to facilitate network explorations. However, when using a graph drawing technique to visualize networks, users often need to tune different algorithm-specific parameters iteratively by comparing the corresponding drawing results in order to achieve a desired visual effect. This trial and error process is often tedious and time-consuming, especially for non-expert users. Inspired by the powerful data modelling and prediction capabilities of deep learning techniques, we explore the possibility of applying deep learning techniques to graph drawing. Specifically, we propose using a graph-LSTM-based approach to directly map network structures to graph drawings. Given a set of layout examples as the training dataset, we train the proposed graph-LSTM-based model to capture their layout characteristics. Then, the trained model is used to generate graph drawings in a similar style for new networks. We evaluated the proposed approach on two special types of layouts (i.e., grid layouts and star layouts) and two general types of layouts (i.e., ForceAtlas2 and PivotMDS) in both qualitative and quantitative ways. The results provide support for the effectiveness of our approach. We also conducted a time cost assessment on the drawings of small graphs with 20 to 50 nodes. We further report the lessons we learned and discuss the limitations and future work.

Index Terms—Graph Drawing, Deep Learning, LSTM, Procrustes Analysis

1 Introduction

Node-link diagrams are widely used to visualize networks in various areas, such as bioinformatics, finance, and social networks analysis. Many graph drawing techniques have been proposed in the past five decades [4, 30, 41, 49] to achieve desired visual properties of node-link diagrams, such as fewer edge crossings, less node occlusion, and better community preservation, to support an easy interpretation of the underlying network structures.

Graph drawing methods are often based on different underlying principles: from spring-embedder algorithms [24], to energy-based approaches [47,48,63], to dimension-reduction based techniques [9,27]. When users employ a specific graph drawing algorithm, they usually need to understand its basic mechanism and tune its various parameters to achieve the desired visual properties for different graphs, though some default parameters are often provided by the developers. Such trial-and-error process requires time and is a non-trivial challenge for less experienced users without a background in graph drawing. Since the algorithm-specific parameters and the corresponding drawings often depend on the input graph structure, we consider the question whether a machine learning approach can be used instead to generate the graph drawings.

One possible choice is using the graph structure information to directly predict the graph drawings with certain visual properties, where graph drawing is considered as a structure-to-layout mapping function. A recent work [54] speeds up the graph drawing process by showing a pre-computed drawing of a graph that has a similar graphlet frequency to the input graph. However, such an approximation needs to extract a hand-crafted feature (i.e., graphlet frequency) and cannot guarantee that it will definitely generate an accurate drawing for the input graph, since graphs with a similar graphlet frequency can have totally-different topology structures. On the other hand, deep learning techniques have shown a powerful capability for modelling the training data and making predictions for relevant data inputs, where no hand-crafted features are needed. Deep learning techniques have been successfully used in various applications such as computer vision and natural language processing fields [32,56]. Inspired by these successes, we are exploring the possibility of applying deep learning techniques to the problem of graph drawing in this paper.

However, we are not aware of any prior work on using deep learning for graph drawing, and there is still a significant gap in how to achieve this. This gap is mainly due to three aspects: model architecture, loss function design, and training data.

Model architecture: Graphs represent topological relationships between different entities; this makes graphs intrinsically different from typical datasets that are often used in deep learning, such as images, videos and texts, which are all Euclidean data. Therefore, it remains unclear whether deep learning techniques can be used for graph drawing and how to adapt existing techniques for the graph drawing problem. Some recent work on Graph Convolutional Neural Networks (GCN) [15, 51] has adapted the CNN framework for graph data, but they are mainly applied to node classification and link prediction tasks, which is different from ours. Since there is no prior work on this problem, our first step is identifying a deep neural network model that can be used to predict graph drawings. At the same time, it is also necessary to identify a transformation that can convert a graph structure into a data structure that can be processed by deep learning models.

Loss function design: One key part of using deep learning techniques is designing an appropriate loss function to guide the model training. For typical deep learning tasks such as classification, the loss function can be easily defined by counting incorrect predictions. However, it is much more complicated for graph drawing. For example, how can we define whether the prediction of a node position is "correct" or "incorrect"? Since a graph drawing may have significantly-different visual appearances after linear transformations, like translation, rotation and scaling, it is also critical for us to design a loss function that is invariant to those transformations.

Training data: High-quality training datasets are critical for using deep learning techniques and many benchmark datasets have been published in different applications, e.g., ImageNet dataset [1] for image classification, MNIST dataset [2] for digits recognition. However, there are no available benchmark datasets with clear drawing labels for graph drawing tasks.

In this paper, we propose a graph-LSTM-based approach to directly generate graph drawing results based on the topology structures of input graphs. We transform the graph topology information into a sequence of adjacency vectors using Breadth First Search (BFS), where each adjacency vector encodes the connection information between each node and its adjacent nodes in the sequence. In addition, we propose a Pro-

- *Y. Wang, Z. Jin, Q. Wang and H. Qu are with the Hong Kong University of Science and Technology (HKUST). E-mail: {ywangct, zjinak, qwangbb, huamin}@cse.ust.hk. Z. Jin is also affiliated with Zhejiang University.*
- *W. Cui is with Microsoft Research Asia. E-mail: weiwei.cui@microsoft.com.*
- *T. Ma is with IBM T. J. Watson Research Center. E-mail: tengfei.ma1@ibm.com.*

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

[1] http://www.image-net.org/

[2] http://yann.lecun.com/exdb/mnist/

crustes Statistics based loss function, which is essentially invariant to translation, rotation and scaling of graph drawing, to assess the learning quality and guide the model training. Furthermore, we generate three graph datasets (including grid graphs, star graphs and general graphs with clear communities), which are further drawn by both two regular drawings (i.e., grid layout and star layout) and two general drawings (i.e., a force-directed graph drawing [47] and a dimension-reduction-based drawing [9]). We carefully choose the parameters of drawing algorithms to generate drawing results with certain desired visual properties for these graphs (e.g., better preservation of community structure and less node occlusion). As a proof of concept, these drawings are treated as the ground-truth labels. The graphs and the corresponding drawings are used to train and test the proposed approach.

We investigated the effectiveness of the proposed deep learning approach through both qualitative comparisons and quantitative metric evaluations, where the drawings of our approach are compared with the ground truth drawings (drawn by ForceAtlas2 [47] and PivotMDS [9]) and the drawings by a 4-layer bidirectional LSTM model. In summary, the primary contributions of this work include:

- A novel graph-LSTM-based approach for graph drawing, which, to the best of our knowledge, is the first time that deep learning has been applied to graph drawing.
- Qualitative and quantitative evaluations on three synthetic graph datasets (i.e., grid graphs, star graphs and clustered graphs with 20–50 nodes) and four types of drawings (i.e., grid layout, star layout, ForceAtlas2 and PivotMDS), which provides support for the effectiveness and efficiency of our approach in generating graph drawings similar to the training data.
- A detailed summary of the lessons we learned in the development of the proposed approach, which, we hope, will assist in future research on using deep learning for graph visualization.

2 Related Work

This section summarizes the related work of this paper, which mainly consists of three parts: graph drawing, graph neural networks, and machine learning approaches to graph drawing.

2.1 Graph Drawing

One of the central problems in graph visualization is the design of the algorithms for graph layout. Since Tutte [72, 73] proposed his barycenter method for graph drawing more than fifty years ago, the information visualization community has proposed many graph drawing techniques. These algorithms can be found in various books [4, 49, 71] and surveys [16, 30, 41, 74, 79].

Typically, graph drawing algorithms generate only one drawing for a graph, though some work [6] also proposes producing multiple drawings for the same graph. According to the survey by Gibson *et al.* [30], the existing graph drawing algorithms can be categorized into three types: force-directed layouts, dimension reduction based layouts, and computational improvements like multi-level techniques. Force-directed graph layout approaches regard a graph as a physical system, where nodes are attracted and repelled in order to achieve desirable graph drawing aesthetics. Eades [22] proposed a spring-electrical-based graph drawing approach, where nodes and edges are modeled as steel rings and springs, respectively. The final graph drawing result is the stable state when the forces on each node reach an equilibrium. This kind of modelling is the start of all force-directed techniques and has inspired many follow-up algorithms, like the spring-embedder algorithm by Fruchterman and Reingold [25], the graph-embedder (GEM) algorithm [24], and the energy-based approaches [47, 48, 63, 78]. Dimension reduction based methods focus on retaining the information of high-dimensional space in the projected 2D plane, especially the graph-theoretic distance between a pair of nodes. Various dimension reduction techniques have been used for graph drawing, including multidimensional scaling (MDS) [9, 27], linear dimension reduction [38], self-organising maps (SOM) [7, 8] and t-SNE [53]. The last category of algorithms mainly aims to improve the efficiency of force-directed algorithms for drawing very large graphs. These approaches often follow a multi-level paradigm: optimizing the graph drawing in a coarser graph representation and further propagate the layout result back to the original graph [26, 34, 37, 44].

Different from prior studies, this paper explores the possibility of using deep neural networks for graph drawing.

2.2 Graph Neural Networks

Existing deep neural networks mainly focus on regular Euclidean data (e.g., images and text), which cannot be directly applied to non-Euclidean data, like graphs. To address this issue, a number of graph neural networks (GNN) have been proposed by extending existing deep neural networks, e.g., convolutional neural networks (CNNs) and recurrent neural networks (RNNs), to the graph domain [82].

The GNNs that are derived from CNNs can be categorized into spectral approaches and non-spectral approaches [82]. Spectral approaches apply convolution to the spectral representation of graphs [9, 12, 15, 40, 51]. For example, Bruna et al. [12] conducted convolution in the Fourier domain using eigen decomposition of the graph Laplacian. Defferrard et al. [15] approximated the spectral convolution using Chebyshev polynomials and reduced the computational cost. Since spectral convolution depends on the input graph, spectral approaches are usually applied in the learning problem within a single graph. Instead of defining convolution operations in the spectral field, non-spectral approaches operate convolution directly on the graph [36, 60, 62]. The key challenge of non-spectral approaches is how to define the neighborhood of a node as the receptive field and various methods have been proposed, including adaptive weight matrices [21], uniformly sampling [36], and transition matrices [1]. A closely related research direction explores using RNNs for graph-structured data [57, 65, 70, 82]. For example, Li et al. [57] modified the Gate Recurrent Units (GRU) and proposed a gated GNN to learn node representations. Tai et al. [70] proposed two types of tree-LSTM, generalizing the basic LSTM to tree-structure typologies, to predict the semantic relatedness of sentences. Peng et al. [65] extended tree-LSTM by distinguishing different edge types in the graph and applied the model to the relation extraction problem in the Natural Language Processing (NLP) field. You et al. [80] developed an RNN-based method for modeling complex distributions over multiple graphs and further generating graphs.

However, the idea of applying GNNs to graph drawing has been rarely explored, even though it is a fundamental research direction in the visualization community.

2.3 Machine Learning Approaches to Graph Drawing

According to the survey by Santos Vieira *et al.* [17], there have been only a few studies about applying machine learning techniques to graph drawing. These techniques can be roughly classified into two categories: the approaches that learn from human interaction and those without using human interaction. The first group of techniques assume that the choices of aesthetic criteria and their importance depend on the users' subjective preferences. Therefore, these approaches keep humans in the loop and use evolutionary algorithms (e.g., genetic algorithms) to learn user preferences [2, 3, 58, 68, 69]. However, these approaches are inherently dependant on user interactions. The second category focuses on using traditional neural-network-based algorithms to optimize the aesthetic criteria of a graph layout [13, 76] or to draw graphs in both 2D and 3D space [59]. However, these early studies are essentially categorized as traditional graph drawing methods, where algorithm-specific parameters are still needed.

Recently, Kwon *et al.* [54] proposed a machine learning approach that provides users with a quick preview of the graph drawing and it uses graphlet frequency to compute the similarities among different graph structures. However, as Kwon *et al.* pointed out in their paper, similar graphlet frequencies do not necessarily lead to similar drawings. Deep learning techniques have recently been applied to multidimensional projections [23]. However, the networks cannot be directly used for graph drawing, since the designs of the model input and training loss function for graph drawing are significantly different from those of multidimensional projection. Also, neural-network-based approaches have been proposed to evaluate graph drawing results [35, 52].

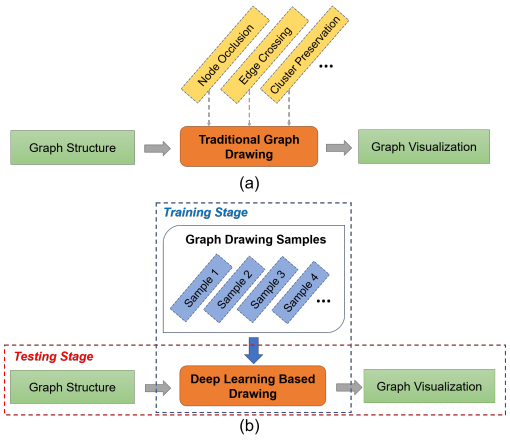

Fig. 1. The workflow of graph drawing algorithms: (a) traditional graph drawing algorithms, (b) the proposed deep learning based approach.

This paper expands on these earlier findings and makes new contributions to the field by focusing on applying deep learning techniques to direct graph drawing instead of using a similar preview, and the trained model can be reusable for new graphs.

3 Problem Statement

A graph $G = (V, E)$ consists of a set of n nodes $V = \{v_1, v_2, ..., v_n\}$ and an edge set $E \subseteq V \times V$. The graphs can be classified into directed and un-directed graphs, depending on whether the edges are directed or not. For graph drawing, the edge directions are often ignored, since they can easily be visualized by adding an arrow to each link. In this paper, we focus on the visualization of unweighted and undirected graphs. The graph drawing problem is finding a set of coordinates $C = \{c_v | v \in V\}$ for the given graph $G = \{V, E\}$ [44]. In this paper, we only consider 2D drawings, which means $p_v \in R^2$. Also, we assume that the edges in the graph drawings are straight-lines, instead of arcs or curves.

As discussed in Section 2.1, there have been many graph drawing algorithms that are proposed for optimizing aesthetic criteria like minimizing edge crossings, avoiding node occlusions, and preserving the node community structures. These criteria are formulated as objective functions and integrated into the design of *traditional graph drawing algorithms* (Fig. 1(a)). When using a drawing algorithm to visualize a specific graph, users also need to tune the algorithm parameters through trial and error to achieve a suitable graph drawing result.

In this paper, we formalize graph drawing as a learning problem and propose a novel and generalizable deep-learning-based approach to graph drawing (Fig. 1(b)). Given a set of graph drawing examples with desirable aesthetic properties and their structures, the deep learning model is trained to learn the mapping and corresponding algorithm-specific parameters for determining the desirable graph drawings (*Training Stage* in Fig. 1(b)). Once the deep learning model is successfully trained, when given a new graph, it can automatically analyze the graph structure and directly generate a layout that carries the common visual properties of the drawing examples (*Testing Stage* in Fig. 1(b)). We use the term *"graph drawing style"* to refer to the common visual properties (e.g., the characteristics regarding edge crossings, community preservation and node occlusion) that are shared by the training graph drawings. The deep learning model learns one specific drawing style from a certain training dataset. In real applications, the deep learning model can be pre-trained by experts. Then, the well-trained model can be directly used by different users to visualize graphs, especially graph drawing novices.

4 Background: Long Short-Term Memory Networks

As will be introduced in Section 5, we propose a graph-LSTM-based approach for graph drawing, where the foundation of our model is the typical LSTM architecture. This section introduces the basic concepts and other related background of LSTM.

LSTM architecture is a popular variant of Recurrent Neural Networks (RNNs). It can learn long-distance dependencies of the input sequence and can avoid the gradient exploding and vanishing problems of traditional RNN models [28, 42]. The main reason for this is that LSTM models introduce a *memory cell*, which can preserve the state over a long time range. An LSTM memory cell consists of an input gate, output gate, and forget gate. It takes sequential data $\{x_0, ..., x_T\}$ as inputs and maintains a time-variant *cell state* vector c_t and *hidden state* vector h_t, where $x_t \in \mathbb{R}^m$, $c_t \in \mathbb{R}^n$ and $h_t \in \mathbb{R}^n$.

The transitions functions of LSTM are as follows:

$$i_t = \sigma\left(W^{(i)}x_t + U^{(i)}h_{t-1} + b^{(i)}\right) \quad (1)$$

$$o_t = \sigma\left(W^{(o)}x_t + U^{(o)}h_{t-1} + b^{(o)}\right) \quad (2)$$

$$\tilde{c}_t = \tanh\left(W^{(c)}x_t + U^{(c)}h_{t-1} + b^{(c)}\right) \quad (3)$$

$$f_t = \sigma\left(W^{(f)}x_t + U^{(f)}h_{t-1} + b^{(f)}\right) \quad (4)$$

$$c_t = i_t \odot \tilde{c}_t + f_t \odot c_{t-1} \quad (5)$$

$$h_t = o_t \odot \tanh(c_t) \quad (6)$$

where x_t, h_t and c_t represent the input feature vector, hidden state, and cell state of the current time step t, respectively. W's and U's are the weighted matrices for the input and hidden state, and b's are the bias vectors. σ, tanh and $\odot$ are the sigmoid function, the hyperbolic tangent function, and the pointwise multiplication operation, respectively. Basically, the input gate i_t controls how much the information is updated at each time step; the output gate o_t controls how much of the internal state information flows out of the cell. The forget gate f_t is a key design of LSTM; it enables LSTM models to forget the previous cell state that has become irrelevant to a certain degree. Due to the design of these gates, LSTM models can learn and represent long-distance correlations within sequential input data [42].

5 *DeepDrawing*

We propose a deep learning based approach, called *DeepDrawing*, for graph drawing. Our model is built on the widely-used LSTM framework [42]. This section introduces *DeepDrawing* from the perspectives of model architecture, input design, and loss function.

5.1 Graph-LSTM-based Architecture

When applying deep learning techniques to graph drawing, a fundamental requirement is to learn a certain graph drawing style from multiple graphs of various sizes. As discussed in Section 2.2, many graph neural networks, like spectral approaches [9, 12, 15, 40], mainly focus on learning from a single graph or fixed-size graphs. Thus, these models do not easily generalize graphs with different sizes and structures [10]. On the other hand, RNN-based graph neural networks are intrinsically applicable to graphs with variable sizes, since RNN cells can be recurrently used. Also, a recent study [80] has shown that RNNs are capable of modelling the structure information of multiple graphs. Inspired by these models, we focus on RNN-based approaches for graph drawing in this paper.

Among the RNN-based approaches, vanilla RNNs have proved to be difficult to train, as they suffer from gradient vanishing or explosion [5, 64] problems. On the contrary, as introduced in Section 4, LSTM models introduce a series of gates to avoid amplifying or suppressing the gradients, making them better at capturing long-distance dependencies.

In this paper, we propose a graph-LSTM-based approach for graph drawing. The commonly-used LSTM architectures are often linearly chained, as described in Section 4. One of their major limitations is that they can only explicitly model sequential data. However, for graph drawing, the input is essentially the graph/network structure, which is usually not linearly-chained. The layout position of a node in the graph drawing depends on all the other nodes that are directly or indirectly connected to it. When using a general LSTM model, such kind of dependency information can still be weakened or lost, especially for the LSTM cells that are far from each other. Inspired by the recent

work in natural language processing field [65, 70], we propose adding direct connections between different LSTM cells to explicitly model the topological structure of input networks. Such direct connections are termed "skip connections" in the deep learning field [32]. Then we can use the linear chain between adjacent LSTM cells to propagate the overall state of prior graph nodes to the subsequent nodes along the chain.

Although our model architecture is similar to prior studies [65, 70], it targets at different problems. Unlike natural language processing (NLP) problems, where the input text is already sequential data and the input feature vector can be directly gained through word embedding, it is necessary to carefully design the architecture and input feature vector to model the graph topology information when using deep learning for graph drawing. To the best of our knowledge, our model is the first deep learning architecture proposed for graph drawing tasks. Fig. 2 provides an overview of the proposed model architecture. The input graph is transformed into a sequence of nodes. Each LSTM cell takes the feature vector of one node as input and generates the output status of each node. The green arrows between LSTM cells represent the ***real edges*** in the graph structure, while the dotted yellow arrows are the ***"fake" edges*** between adjacent nodes in the BFS-ordered node sequence to propagate the summary state of previous nodes to subsequent unprocessed nodes. The detailed transition equations of our model are as follows:

$$i_t = \sigma\left(W^{(i)}x_t + U^{(i)}h_{t-1} + \sum_{k\in P(t)} \tilde{U}^{(i)}h_k + b^{(i)}\right) \tag{7}$$

$$o_t = \sigma\left(W^{(o)}x_t + U^{(o)}h_{t-1} + \sum_{k\in P(t)} \tilde{U}^{(o)}h_k + b^{(o)}\right) \tag{8}$$

$$\tilde{c}_t = \tanh\left(W^{(c)}x_t + U^{(c)}h_{t-1} + \sum_{k\in P(t)} \tilde{U}^{(c)}h_k + b^{(c)}\right) \tag{9}$$

$$f_{t,t-1} = \sigma\left(W^{(f)}x_t + U^{(f)}h_{t-1} + b^{(f)}\right) \tag{10}$$

$$f_{tk} = \sigma\left(W^{(f)}x_t + \tilde{U}^{(f)}h_k + b^{(f)}\right), k \in P(t) \tag{11}$$

$$c_t = i_t \odot \tilde{c}_t + f_{t,t-1} \odot c_{t-1} + \sum_{k\in P(t)} f_{tk} \odot c_k \tag{12}$$

$$h_t = o_t \odot \tanh(c_t) \tag{13}$$

where $P(t)$ denotes the prior nodes that have *real* edges linked to Node t. Like the standard LSTM model (Equations 1-6), the proposed model also considers the hidden state of the immediate predecessor node $(t-1)$ in the recurrent terms (i.e., $U^{(i)}$, $U^{(o)}$, $U^{(c)}$ and $U^{(f)}$), which correspond to the *fake edges* discussed above. However, when comparing the transition functions of both models (Equations 1–6 and Equations 7–13), it is easy to find the main difference: our model further considers the *real edges* in the architecture and integrates the states of the remotely-connected predecessors (i.e., $\tilde{U}^{(i)}$, $\tilde{U}^{(o)}$, $\tilde{U}^{(c)}$ and $\tilde{U}^{(f)}$) into the current node. Thus, it can directly reflect the actual graph structure and well model the influence of former nodes on subsequent nodes along the node sequence in the graph drawing.

On the other hand, graphs are not sequential data. When attempting to draw a graph using this approach, all nodes (both those before and those after in the linear layout) should be taken into consideration, i.e., *the latter nodes in the sequence can also influence the positions of the former nodes during the actual graph drawing*. To better model this mutual influence, we further introduce a backward propagation to the proposed graph-LSTM-based model by simply reversing the link direction in the forward propagation (Fig. 2(b)). Then, we combine the outputs of each LSTM cell in both forward and backward propagations into a concatenated feature vector, which is further input into a fully-connected layer to generate the final 2D coordinate of each node.

5.2 Model Input

When applying the LSTM model to graph drawing, it is crucial to find a suitable way to input the graph structure information into the LSTM model. More specifically, we need to determine the *feature vector* for each node, where the graph structure information of each node should be properly encoded. This feature vector will be further input into the LSTM model. Also, it is necessary to transform the original graph to a sequence of nodes (i.e., *node ordering*) that can be processed by the LSTM model, which is another key point for applying LSTM into graph drawing.

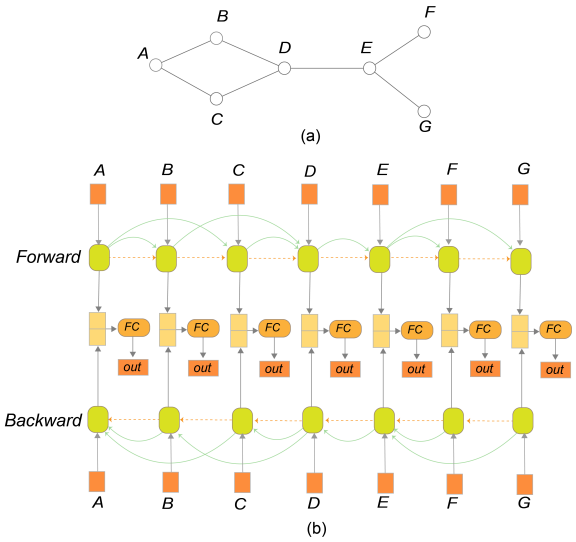

Fig. 2. An illustration of the proposed graph-LSTM-based model architecture: (a) an example graph input, (b) the proposed model to process the input graph. The graph nodes are sorted using BFS, with each node represented by an adjacency vector encoding its connections with predecessor nodes. The dotted yellow arrows ("fake" edges) propagate the prior nodes' overall influence on the drawing of subsequent nodes, and the curved green arrows (real edges of graphs) explicitly reflect the actual graph structure, enhancing the graph drawing details. The information of both forward and backward rounds is considered for generating the final 2D node layouts.

Node Feature Vector: When LSTM models are applied to NLP tasks, word embedding techniques are often used to transform words into fixed-length feature vectors; these vectors can be further input into LSTM models. Considering that many node embedding techniques have been proposed, like node2vec [33], DeepWalk [66] and SDNE [75], it is natural to use node embedding techniques to encode graph structure information and further input it into the proposed model. However, these node embedding techniques are mainly used for a single graph and have been proved incapable to be generalized to multiple graphs [39]. Our initial experiments during the design of the proposed method also confirmed this observation.

Taking into account that adjacency information between nodes is the essential information in a graph, we propose using a fixed-length adjacency vector as the feature vector of each node directly; it will be further input into the proposed model. The adjacency vector of each node encodes the connectivity between the current node and its prior k nodes, where k is empirically set as a fixed number.

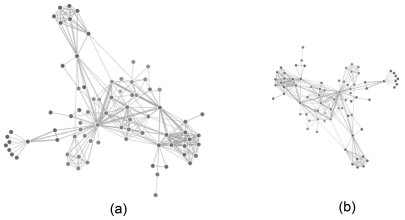

Fig. 3. The same graph drawing under transformations may look different: (a) the original graph drawing, (b) the same graph drawing that has been translated, rotated by 180 degrees and further scaled.

Node Ordering: Graphs can be represented by adjacency matrices and are permutation-invariant in terms of node ordering. Therefore, one naive way to model a graph as a sequence of nodes is to use random order. However, for a specific graph, the total number of such kind of random node orderings is $O(n!)$, where n is the total number of nodes.

Inspired by the recent study [80], we propose using breadth-first-search (BFS) to generate node ordering for a specific graph. The major advantage of BFS ordering is that we need to train the proposed model on only all the possible BFS orderings, instead of exhaustively going through all possible node permutations, which can reduce the model searching space and benefit the model training. In addition, for a specific node in a node sequence sorted by BFS, there is an upper bound for the possible connections between this node and the nodes before it along the BFS sequence [80]. More specifically, let $(v_1, v_2, ..., v_{i-1}, v_i, v_{i+1}, ..., v_n)$ be a BFS ordering of nodes, the furthest node before v_i that is possible to link to v_i is v_{i-M} and all the nodes before v_{i-M} are impossible to link to v_i, where M is the maximum number of nodes of each level in a BFS sorting. Due to the existence of this upper bound, we can set the length of the node feature vector to a fixed length smaller than the node number without losing much graph structure information.

To further reduce the model's searching space, we also use node degree to sort the nodes at the same depth level of a BFS sorting. During the training stage of the model, we randomly choose the starting node of the BFS sequence, which can augment the training dataset and improve model generalizability.

5.3 Loss Function

The design of loss function is another crucial part of applying deep learning techniques to graph drawing, since the loss function will guide the neural network to learn what a desirable graph drawing should be based on the training dataset. Specifically, the graph drawings in the training dataset are regarded as the ground-truth drawings, and the purpose of the loss function is to guide the proposed model to generate graph layouts as ***"similar"*** to the corresponding training data as possible.

However, it is challenging to propose an appropriate loss function for comparing the similarity between two drawings of the same graph. For example, a specific graph drawing may look very different after a series of operations from the set of translation, rotation and scaling, as shown in Fig. 3. Therefore, the loss function should be invariant to such kind of transformations. Motivated by these requirements, we propose conducting Procrustes Analysis [14, 18] to assess the similarity between two drawings of the same graph. Procrustes Analysis is a widely-used technique in statistics and shape comparison [31] and it has also been used in graph drawing [45]. For two graph drawings of the same graph with n nodes, Procrustes Analysis will explicitly translate, scale and rotate the drawings to align them. Suppose the corresponding coordinates of all the n nodes in the two drawings are $C = [c_1, ..., c_n]^T$ and $\bar{C} = [\bar{c}_1, ..., \bar{c}_n]^T$, where $c_i = (x_i, y_i)$ and $\bar{c}_i = (\bar{x}_i, \bar{y}_i)$, the ***Procrustes Statistic*** will be calculated to indicate the shape difference between them as follows:

$$R^2 = 1 - \frac{(tr(C^T \bar{C} \bar{C}^T C)^{1/2})^2}{tr(C^T C) tr(\bar{C}^T \bar{C})} \tag{14}$$

where $0 \leq R^2 \leq 1$. It is essentially the squared sum of the distances between C and $\bar{C}$ after a series of best possible transformations. $R^2 = 0$ denotes both graph drawings are exactly the same, while $R^2 = 1$ indicates that the two graph drawings are totally different and cannot be matched by any transformations.

6 Evaluation

We thoroughly assess *DeepDrawing* through both qualitative and quantitative evaluations. This section introduces the detailed experiment settings and evaluation results.

6.1 Experiment Setup

The experiment settings include graph generation, drawing dataset generation, baseline method selection and other implementation details.

6.1.1 Graph Generation

Since there are no public graph drawing datasets, where the layout position of each node is labeled, we need to generate graphs and further properly draw the graphs to gain drawing datasets for training and evaluating *DeepDrawing*. For graph generation, we adopt the classical method of synthetic graph generation proposed by Lancichinetti et al. [55], since it can generate realistic benchmark graphs with various community structures, which is helpful for verifying whether community-related visual properties are learned by *DeepDrawing*. The implementation by the authors[3] is used. We can specify various parameters, including node number, average node degree, community number and community overlap coefficient, to control the structures of generated graphs. In this paper, we mainly generated graphs with the node number evenly ranging from 20 to 50, as prior studies [29, 46] have shown that node-link diagrams are more suitable for graphs with dozens of nodes in terms of visual perception. We randomly split the whole dataset into training, validation and testing datasets. Table 1 shows the detailed statistics of the generated graphs used in this paper.

Apart from general graphs, we also generate grid and star graphs, which have simple and regular topological structures, to extensively evaluate the effectiveness of *DeepDrawing*.

To avoid contaminating the training data, we carefully checked and guaranteed that no validation and testing graphs have exactly the same topology structure with any training graphs by using pynauty [4], a public library for checking graph isomorphism.

6.1.2 Drawing Dataset Generation

We used different types of graph drawing methods to draw the graphs. As discussed in Section 2.1, there are mainly three types of graph drawing methods. There are many force-directed and dimension-reduction-based graph layout approaches. We chose ForceAtlas2 [47] and PivotMDS [9] respectively, since both algorithms are typical and widely-used algorithms for each type. Also, ForceAtlas2 can preserve the communities and PivotMDS is deterministic and fast. The multi-level drawing techniques are not used, as they mainly target at the acceleration of large graph visualization (Section 2.1), which is not the focus of this paper.

All the graphs are drawn on a canvas with a size of 800×800, which is big enough for rendering the graphs used in this paper. When drawing the graphs using the above two methods, we manually tune the algorithm-specific parameters according to the node number, community structure and edge density of the input graphs, in order to achieve desirable visual properties such as better clustering, fewer edge crossings and less node occlusion. Also, due to the random point initialization of ForceAtlas2, its layout results are not deterministic. Given a graph with the same algorithm-specific parameters, the graph drawing result may be different, which can confuse the deep learning model. Therefore, considering that PivotMDS is deterministic, we follow the method by Kruiger et al. [53] and initialize the node positions with PivotMDS instead of using default randomized initial 2D positions, guaranteeing that the same input graph is mapped to the same layout. By drawing the generated graphs using ForceAtlas2 and PivotMDS, we gained two drawing datasets with different layout styles.

For the grid and star graphs, apart from using ForceAtlas2 and PivotMDS, we also visualized grid graphs as perfect grid layout and star graphs as perfect star layout, gaining three drawing datasets for grid and star graphs.

6.1.3 Baseline Method Selection

To better evaluate the effectiveness of our approach, it is necessary to compare it with other general models that have a similar architecture. Initially, we compared unidirectional LSTM and bidirectional LSTM models, and also tested them with $1 \sim 4$ layers. Among all these general LSTM models, our initial results show that a 4-layer bidirectional LSTM (Bi-LSTM) model has the best graph drawing performance in terms of the Procrustes Statistic based similarity with the ground truth

[3] https://github.com/eXascaleInfolab/LFR-Benchmark_UndirWeightOvp
[4] https://web.cs.dal.ca/~peter/software/pynauty/html

Table 1. Statistics of the generated graphs.

Graph Type	#Node	#Edge	Node Degree	Training	Validation	Testing	Total	#Community
Grid Graphs	[100, 576]	[180, 1104]	[2,4]	72	24	24	120	-
Star Graphs	[10, 209]	[9, 208]	[1,208]	120	40	40	200	-
General Graphs	[20, 50]	[23, 178]	[1,10]	26000	3000	3000	32000	[2, 12]

Table 2. The configurations of the baseline model and our model.

Model	Hidden Size	Layers	Direction	# Parameters
Baseline	256	4	bidirectional	5.33M
Ours	256	1	bidirectional	1.12M

drawings. This performance comparison result is also consistent with Google Brain's prior study [11]. Therefore, a 4-layer bidirectional LSTM model is chosen as the baseline method for comparison in the subsequent evaluations.

6.1.4 Implementation and Model Configuration

PivotMDS and ForceAtlas2 are implemented in Python based on Tulip [5] and Gephi [6]. The proposed graph-LSTM-based model is implemented with PyTorch [7] and PyG Library [8]. The LSTM model implementation integrated in PyTorch is used for the baseline model. The machine we used for model training and all the subsequent experiments has 48 Intel Xeon(R) CPU processors (E5-2650 v4, 2.20GHz), and 4 NVIDIA Titan X (Pascal) GPUs.

6.2 Model and Training Configurations

The detailed configurations for both our model and the baseline model are shown in Table 2. We use the Adam optimizer [50] for the model training. The learning rate and batch size for training both models are set to 0.0015 and 128, respectively. The size of training, validation and testing graphs are shown in Table 1.

For each graph drawing dataset in Section 6.1.2, we train an individual model (the proposed graph-LSTM-based model or the baseline LSTM model) to learn the graph drawing style, with the corresponding algorithm-specific drawing parameters encoded in the model as well. The size of the input adjacency vector of each node for both the grid and general graphs is empirically set as 35. The star graphs are an extreme case, where the surrounding nodes are only connected to the center node. Therefore, its input adjacency vector size is set as the maximum number of prior nodes, i.e., 208 (Table 1). When the training loss converges, the corresponding model is used for generating graph drawings in the subsequent qualitative and quantitative evaluations.

6.3 Qualitative Evaluation

We first trained *DeepDrawing* on the drawing datasets of grid and star graphs, where the graphs are drawn with three different drawing styles: perfect regular layouts (i.e., grid layout or star layout), ForceAtlas2 and PivotMDS. We further compared *DeepDrawing* with the baseline model on the drawing datasets of general graphs with two drawing styles (i.e., ForceAtlas2 and PivotMDS).

6.3.1 Grid and Star Graphs

Fig. 4 shows the graph drawing results of *DeepDrawing* on grid graphs, where the drawings by *DeepDrawing* and the corresponding ground truth drawings are aligned by the Procrustes Analysis. The results demonstrate the excellent performance of *DeepDrawing* in generating three different styles of graph drawings for grid graphs. All the graph drawing styles in the ground truth layouts are well preserved by *DeepDrawing*. For example, the generated perfect grid layouts make all the nodes evenly distributed, but the results of *DeepDrawing* trained on ForceAtlas2 drawings have a sparse distribution of nodes in the center and a dense distribution of nodes in the four corners, and results of *DeepDrawing* trained on PivotMDS drawings tend to make the grid contours curved. All the generated graph drawings by our approach are consistent with the ground-truth drawings.

Fig. 5 shows the drawing results of *DeepDrawing* on the star graphs. The drawing styles of perfect star layout, ForceAtlas2 and PivotMDS are different on star graphs, but *DeepDrawing* is also able to learn these drawing styles and generates graph drawings that are very similar to the ground truth, further confirming the effectiveness of *DeepDrawing*.

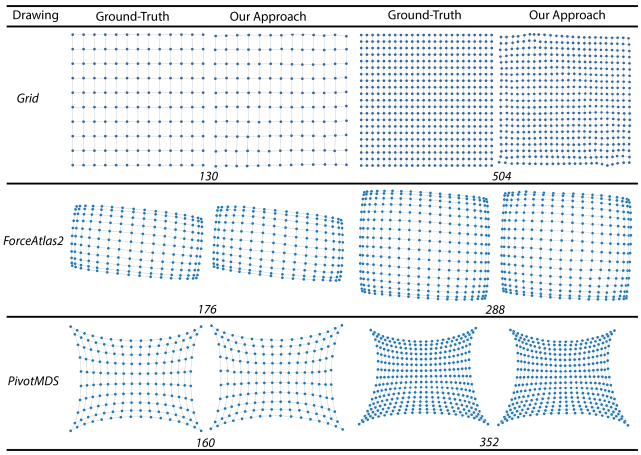

Fig. 4. Qualitative evaluation results on grid graphs: the ground truth and the graph drawing generated by our approach are compared side by side. Each row shows the results of a specific drawing style (i.e., perfect grid layout, ForceAtlas2 and PivotMDS) and the number of graph nodes is shown in the bottom.

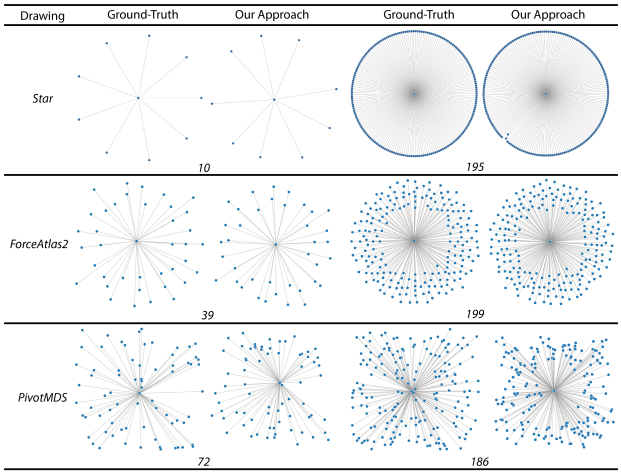

Fig. 5. Qualitative evaluation results on star graphs: the ground truth and the graph drawing generated by our approach are compared side by side. Each row shows the results of a specific drawing style (i.e., perfect star layout, ForceAtlas2 and PivotMDS) and the number of graph nodes is shown in the bottom.

6.3.2 General Graphs

Fig. 6 shows the drawing results on the drawing dataset rendered by ForceAtlas2. The graph drawings generated by both the baseline

[5] http://tulip.labri.fr/Documentation/current/tulip-python/html/index.html
[6] https://github.com/bhargavchippada/forceatlas2
[7] https://pytorch.org/
[8] https://rusty1s.github.io/pytorch_geometric

method and our approach, as well as the ground-truth drawings by ForceAtlas2, are presented; these drawings cover different number of nodes and communities. The ground-truth graph drawings (Column 2 of Fig. 6) can well visualize the community structures. When comparing the drawing results of the baseline model (Column 3 of Fig. 6) with the ground-truth graph drawings, it is easy to see that the baseline model is able to preserve the overall community structures, especially when there are fewer nodes and communities (Rows 1 and 2 of Fig. 6). However, when there are more nodes and communities, the drawings generated by the baseline model deviate from the ground-truth drawings, and different communities may overlap with each other (Rows 3-6). On the contrary, our approach (Column 4 of Fig. 6) better preserves the community structures across different number of nodes and communities than the baseline method. When the drawing results by our approach are further compared to the corresponding ground-truth drawings, it is easy to see that our results reflect the visual properties of the ground-truth drawings and the overall node layouts in both drawings are similar.

Fig. 7 shows the drawing results on the drawing dataset rendered by PivotMDS, where the ground-truth drawings by PivotMDS, the drawings generated by the baseline model and the drawings generated from our approach are compared side by side. Like the drawings of ForceAtlas2, the graph drawings of PivotMDS can also reflect the overall community structure of the graphs. But there are also two major differences between these drawings in terms of the graph drawing style. One difference is that the edge length in the drawings of PivotMDS is more uniform than that of ForceAtlas2. Specifically, when compared with ForceAtlas2, the length of edges within communities is more similar to that of edges between different communities in PivotMDS layouts. The other difference is that some communities with tight connections can overlap each other in PivotMDS drawings, while the drawings of ForceAtlas2 often do not have community overlapping (Fig. 6). For example, the light blue community and dark blue community overlap with each other in Row 4 of Fig. 7, and the darker green community, light green community and darker blue community are also mixed in Row 6 of Fig. 7, due to the tight connections between those communities. Such visual properties are well preserved by our approach. As shown in Column 4 of Fig. 7, the community structure can be clearly recognized and the overlapping communities in the ground-truth drawings are also reflected (Rows 4 and 6). The overall layouts by our approach are quite similar to the ground-truth. However, the baseline model (Column 3 of Fig. 7) cannot preserve these visual properties (Rows 4-6), though it can reflect the community structures to some extent.

6.4 Quantitative Evaluation

We quantitatively evaluate the graph drawing results of both our approach and the baseline method by comparing their similarity with the ground-truth drawings. This comparison is conducted from two perspectives: the Procrustes Statistic-based similarity and the aesthetic metrics-based similarity. The time costs of the graph drawing algorithms running on CPU and GPU are also reported. This section considers only general graphs, since the qualitative evaluation has already shown that our approach achieves good performance on grid and star graphs. All the subsequent experiments are conducted on the testing set of the general graph dataset (Table 1).

6.4.1 Procrustes Statistic-based Similarity

The Procrustes Statistic (Equation 14), which was used as the loss function for the model training, is further used to evaluate whether the models effectively learn a specific graph drawing style or not. We analyzed the Procrustes Statistic-based similarity of all the testing graphs. We first ran Shapiro-Wilk test to check its normality, which indicates the results are not always normal. Thus, we further ran a Friedman test with a Nemenyi-Damico-Wolfe-Dunn for post-hoc analysis to determine the statistical significance (the statistical level $\alpha = 0.05$).

Fig. 8 shows the Procrustes Statistic-based similarity results, where the results on both ForceAtlas2 and PivotMDS drawing datasets are reported. Compared with the baseline approach, our approach achieves

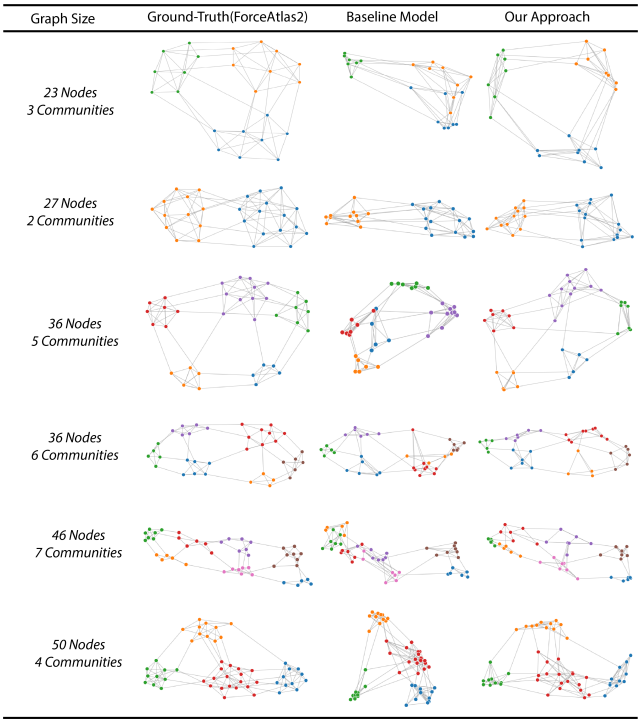

Fig. 6. Qualitative evaluation on general graphs drawn by ForceAtlas2. For the same graph, the ground truth drawing, the drawing by the baseline model and the drawing by our approach are compared in each row. Different colors indicate different communities.

a significantly better ($p < 0.05$) Procrustes Statistic-based similarity on both ForceAtlas2 (0.19 vs. 0.23) and PivotMDS (0.21 vs. 0.34) drawing dataset. This also provides support for the effectiveness of our approach in learning different graph drawing styles.

6.4.2 Aesthetic Metrics-based Similarity

Similar to the prior work [54], we also evaluate the effectiveness of our approach by comparing the aesthetic metric similarity between the drawings generated by our approach and the ground-truth drawings. The ***Root-Mean-Square Error (RMSE)*** is used to measure the aesthetic similarity between them. Given n graphs, suppose the aesthetic metric values of the generated drawings are $\tilde{A} = \{\tilde{a}_1, ..., \tilde{a}_n\}$ and those of the ground-truth drawings are $A = \{a_1, ..., a_n\}$, then the similarity is defined as follows:

$$\mathrm{RMSE}(A, \tilde{A}) = \sqrt{\frac{1}{n}\sum_i (\tilde{a}_i - a_i)^2}. \tag{15}$$

Smaller RMSE scores correspond to higher similarity to the ground-truth graph drawing.

Various aesthetic metrics have been proposed to assess the aesthetic quality of different graph drawings [19, 67, 77]. In this paper, three aesthetic metrics are considered: edge crossings, node occlusions and community overlapping, since they are widely-used aesthetic criteria for evaluating how well the underlying graph topology has been realized in a drawing [20, 67, 77]. Also, they have a normalized form and thus can be used to compare graphs of different sizes.

Edge crossings (A_{ec}): We use the edge crossing metric introduced by Purchase [67], which is defined as the ratio of the number of edge crossings in a drawing over the upper bound of the possible crossings.

Node occlusions (A_{no}): We choose the global metric of node occlusion introduced by Dune et al. [19], i.e., the ratio of the union area of all the node representations over their total area if drawn independently.

Community overlapping (A_{co}): We employ the global version of the autocorrelation-based metric introduced by Wang et al. [77]. For a specific node, this metric considers both the Euclidean distance between the node and its surrounding nodes and also whether they belong to the

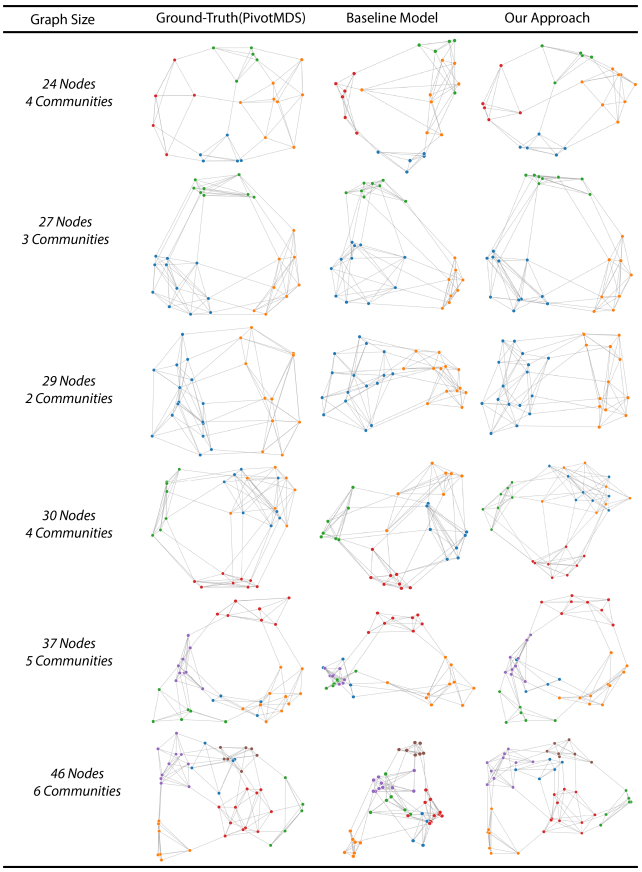

Fig. 7. Qualitative evaluation on general graphs drawn by PivotMDS. For the same graph, the ground truth drawing, the drawing generated by the baseline model and the drawing generated by our approach are compared in each row. Different colors indicate different communities.

same community. Therefore, this metric can clearly reflect the degree of overlapping between different communities.

Table 3 shows the results of the aesthetic metric-based similarities. For node occlusions, our approach is similar or slightly better than the baseline method on both the ForceAtlas2 and PivotMDS drawing datasets. However, our approach has a better performance than the baseline method in terms of edge crossing similarity and community overlapping similarity. Since community structure can reflect the global graph structure, a better similarity of community structure indicates the better preservation of a certain graph drawing style. Overall, the aesthetic metrics-based similarity comparisons confirm the effectiveness of our approach in preserving the original graph drawing style, which is also consistent with our observations in Section 6.3.

Table 3. The RMSEs of aesthetic metrics-based similarity evaluated on the drawing datasets visualized by both ForceAtlas2 and PivotMDS.

Aesthetic Metrics	ForceAtlas2		PivotMDS	
	Baseline	Ours	Baseline	Ours
$RMSE(A_{ec})$	0.0169	**0.0125**	0.0191	**0.0134**
$RMSE(A_{no})$	0.0316	0.0310	0.0799	0.0794
$RMSE(A_{co})$	0.0138	**0.0125**	0.0171	**0.0131**

6.4.3 Time Cost

We evaluate the time cost of the proposed approach in comparison with both the original graph drawing techniques (i.e., ForceAtlas2 and PivotMDS) and the baseline model. The time costs on both CPU and GPU are tested. For CPU mode, each graph is repeatedly drawn 10 times. Their average is regarded as the actual time cost of drawing a graph. The iteration number for ForceAtlas2 is empirically set as

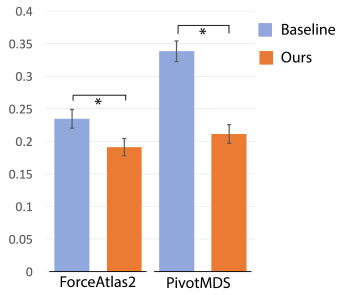

Fig. 8. The results of Procrustes Statistic-based similarity. The baseline method and our approach are evaluated on both ForceAtlas2 and PivotMDS graph drawing datasets. The error bars are 95% confidence intervals and significant differences are marked with a line between them ($*$: $p < 0.05$).

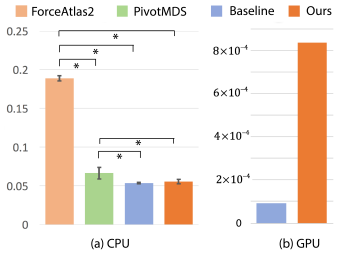

Fig. 9. A comparison of the average running time (second) for drawing each graph using different methods. (a) The average running time on CPU, where the error bars are 95% confidence intervals and the graph drawing techniques with significant difference are marked with a line between them ($*$: $p < 0.05$), (b) the average running time of the baseline method and our approach on GPU.

700 steps, since the graph drawing results after 700 steps are relatively stable for the given graphs, though it is indicated that it can achieve a slightly better drawing quality with more steps [47]. For GPU mode, we only compared the time cost of our approach with that of the baseline model to provide a quick understanding of how fast the deep learning-based approaches can achieve on GPUs. Since the major advantages of GPU-based programming lie in its parallel computation, the testing graphs are input into the model all together to generate the drawings and the corresponding average time cost for each graph is calculated accordingly.

Fig. 9 shows the average time cost of using the graph drawing methods to draw a graph on both CPU and GPU. The CPU time costs do not follow a normal distribution according to a Shapiro-Wilk test. Thus, we ran a non-parametric Friedman test with a Nemenyi-Damico-Wolfe-Dunn for post-hoc analysis to determine the statistical significance ($\alpha = 0.05$). Fig. 9(a) shows that our approach and the baseline approach have a similar CPU time cost (0.055 vs. 0.054 second) and both of them are significantly faster than the original graph drawing methods on CPU. Specifically, the time cost of our approach is only 29% of ForceAtlas2 (0.189 second) and 82% of PivotMDS (0.067 second). Fig. 9(b) indicates that our approach (8.3×10^{-4} second) is slower than the baseline method (0.93×10^{-4} second) on GPU, though its training parameter number is only about 20% of the baseline model (Table 2). One major reason for this is that we directly used the implementation integrated in PyTorch for the baseline LSTM model, whose underlying implementation is based on C/C++ and has been extensively optimized with NVIDIA cuDNN, but the implementation of *DeepDrawing* is Python without optimization using NVIDIA cuDNN. However, the speed of both deep learning-based methods is increased by two orders of magnitude when running on GPU than that of the techniques running on CPU.

For the model training, Fig. 10 shows the training validation loss curves. It is easy to observe that *DeepDrawing* needs fewer epochs to converge and has lower and smoother training loss than the baseline model. One possible explanation for this is that the architecture of *DeepDrawing* explicitly considers the input graph structure.

7 Discussions

In this section, we first report the lessons we learned when working on this paper. Then, we further discuss the limitations and usage issues of the proposed approach for graph drawing.

7.1 Lessons

We learned many lessons from the model design and training while working on this paper.

Node Feature Vector For determining the adjacency vector size, a naive choice is to set it as the maximum node number of all the graphs, which, however, is at the cost of model complexity and efficiency. We quickly evaluated its influence and found that there is no significant degradation of drawing performance for general graphs, with the decreasing of node size in a certain range. This is probably because the architecture of *DeepDrawing* explicitly encodes the graph structure, and there is an upper bound of the distance between connected nodes in a BFS sequence of a graph [80].

Model Hyper-parameters The choice of model hyper-parameters (e.g., hidden size, layer numbers) of *DeepDrawing* is crucial for deep learning models. With the increase of these hyper-parameters, the learning capability of the model can often increase, but it is also at the cost of increasing overfitting risk and training time and decreasing model efficiency. For *DeepDrawing*, we conducted control experiments to balance these factors to determine the suitable hyper-parameters. For example, we found that increasing the layer number to 2 or 3 brings little improvement in the drawing performance, but changing the model to be unidirectional results in significant performance degeneration.

Loss Function Design For the loss function, we also considered other possible choices, for example, the sum of the edge length difference between the predicted drawing and the ground truth. This kind of loss function can delineate the similarity of different drawings under rotation and translation. However, it cannot capture the similarity between two drawings under scaling and the loss value seems to be dominated by long edges. After a series of careful designs and comparisons, the Procrustes Statistic was finally chosen to guide the model training.

Drawing Complexity, Training Size and Overfitting As shown in Table 1, we used a small training dataset for grid and star graphs (72 for grid graphs and 120 for star graphs), the graph drawing performance is good and no overfitting was observed. The major reason for this is that the graph drawings for grid and star graphs have fixed patterns with fewer variations, which makes it easier for the deep learning model to learn from them. On the contrary, the drawings of general graphs are complex. Initially, we used about 8000 graphs with a fixed order of input nodes for training on general graphs, which results in an overfitted model. This is solved by randomly selecting the starting node of the input BFS-ordering node sequence, which essentially augmented the training data, and further increased the training data size.

7.2 Limitations and Usage Issues

Our evaluations above have shown that *DeepDrawing* can learn from the training graph drawings (i.e., grid layout, star layout, ForceAtlas2 and PivotMDS) and further generate drawings for new graph inputs. However, the proposed method has limitations and next we discuss some of them.

Failure Cases and Limited Evaluations According to our empirical observations, two factors can affect the performance of *DeepDrawing* in generating drawings with a drawing style similar to the training drawing: *graph structure similarity* and *node ordering*. When the input graph has significantly-different graph structures or the nodes are not sorted by BFS, it can result in a decrease of drawing performance of *DeepDrawing* or even generate messy drawings. Also, as the first step of using deep learning for graph drawing, *DeepDrawing* currently focuses on small graphs (i.e., clustered graphs with 20–50 nodes). Testing whether the proposed approach generalizes to large graphs requires further exploration.

Model Interpretability and Interactivity *DeepDrawing* is a deep learning based approach. Like most of deep learning based methods, it also has the interpretability issues, which are an active research topic in both visualization and machine learning [43, 61, 81]. Specifically, in the case of *DeepDrawing*, it is not clear what graph layout aspects are learned by the neural network. Also, once the model training is done, *DeepDrawing* can directly generate drawings with a drawing style similar to the training data for input graphs, where no user interaction is needed. This is both an advantage and disadvantage. It can benefit novice users without a background in graph drawing, but may be a disadvantage for expert users who want to interactively explore more graph properties by themselves.

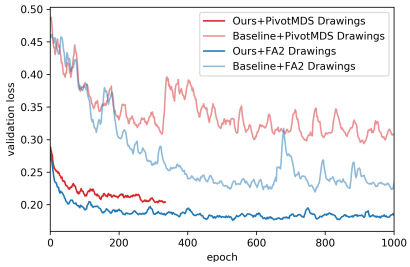

Fig. 10. The loss curves of training the baseline and our model on ForceAtalas2 and PivotMDS drawing datasets. The training of our model on PivotMDS drawings is stopped at 330 epochs due to its convergence.

Usage Issues With a trained model for a specific type of graph and a specific drawing style, *DeepDrawing* can be used to generate similar drawings of similar graphs. However, many sample graphs and the drawings of these graphs are needed, as well as expert interaction with the model (e.g., hyper-parameter tuning), and the time to train the model. In order to generate different types of drawings, the model needs to be retrained on new graphs and new graph drawings.

8 Conclusion

In this paper, we propose *DeepDrawing*, a novel graph-LSTM-based approach for graph drawing, where the graph drawing is formalized as a learning and prediction problem. Given a graph drawing dataset, *DeepDrawing* is trained to learn a graph drawing style and can further generate graph drawings with similar characteristics. We carefully designed the proposed approach in terms of model architecture, model input and training loss. We conducted both qualitative and quantitative evaluations on three types of graphs (i.e., grid graphs, star graphs and general graphs with good community structures) and four types of drawings (i.e., grid layout, star layout, ForceAtlas2 and PivotMDS). The results show that *DeepDrawing* can generate similar drawings for the three types of graphs and its speed is fast on the testing graphs with 20–50 nodes, which provides support for the effectiveness and efficiency of *DeepDrawing* for graph drawing. Also, it is observed that *DeepDrawing* can better preserve the original graph drawing style than the general LSTM-based method, confirming the advantage of our model architecture.

In future work, we plan to optimize the current implementation of *DeepDrawing* (e.g., accelerations with PyTorch C/C++ extensions and NVIDIA cuDNN) and further evaluate its performance on additional types of graphs. Also, it would be interesting to explore how a deep learning approach can benefit the visualization of large graphs with thousands of nodes. For example, given our comparison results of time cost on small graphs with 20 to 50 nodes, deep learning based approaches may also be able to improve the efficiency of large graph drawing. Furthermore, since dynamic graph visualization often depends on the temporal correlation between adjacent time stamps, it is also promising to investigate whether deep learning techniques can be extended to dynamic graph visualization. We hope this work can inspire more research on using deep learning techniques for graph drawing as well as general information visualization.

Acknowledgments

The authors wish to thank Yao Ming, Qiaomu Shen and Daniel Archambault for the constructive discussions. The authors also thank the anonymous reviewers for their valuable comments. This work is partially supported by a grant from MSRA (code: MRA19EG02).

REFERENCES

[1] J. Atwood and D. Towsley. Diffusion-convolutional neural networks. In *Proceedings of Advances in Neural Information Processing Systems*, pp. 1993–2001, 2016.

[2] B. Bach, A. Spritzer, E. Lutton, and J.-D. Fekete. Interactive random graph generation with evolutionary algorithms. In *International Symposium on Graph Drawing*, pp. 541–552. Springer, 2012.

[3] H. J. Barbosa and A. Barreto. An interactive genetic algorithm with co-evolution of weights for multiobjective problems. In *Proceedings of the 3rd Annual Conference on Genetic and Evolutionary Computation*, pp. 203–210. Morgan Kaufmann Publishers Inc., 2001.

[4] G. D. Battista, P. Eades, R. Tamassia, and I. G. Tollis. *Graph drawing: algorithms for the visualization of graphs*. Prentice Hall PTR, 1998.

[5] Y. Bengio, P. Simard, P. Frasconi, et al. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166, 1994.

[6] T. Biedl, J. Marks, K. Ryall, and S. Whitesides. Graph multidrawing: Finding nice drawings without defining nice. In *International Symposium on Graph Drawing*, pp. 347–355. Springer, 1998.

[7] E. Bonabeau. Graph multidimensional scaling with self-organizing maps. *Information Sciences*, 143(1-4):159–180, 2002.

[8] E. Bonabeau and F. Hénaux. Self-organizing maps for drawing large graphs. *Information Processing Letters*, 67(4):177–184, 1998.

[9] U. Brandes and C. Pich. Eigensolver methods for progressive multidimensional scaling of large data. In *International Symposium on Graph Drawing*, pp. 42–53. Springer, 2006.

[10] X. Bresson and T. Laurent. An experimental study of neural networks for variable graphs. 2018.

[11] D. Britz, A. Goldie, M.-T. Luong, and Q. Le. Massive exploration of neural machine translation architectures. *arXiv preprint arXiv:1703.03906*, 2017.

[12] J. Bruna, W. Zaremba, A. Szlam, and Y. Lecun. Spectral networks and locally connected networks on graphs. In *Proceedings of International Conference on Learning Representations*, 2014.

[13] A. Cimikowski and P. Shope. A neural-network algorithm for a graph layout problem. *IEEE Transactions on Neural Networks*, 7(2):341–345, 1996.

[14] T. F. Cox and M. A. Cox. *Multidimensional scaling*. Chapman and hall/CRC, 2000.

[15] M. Defferrard, X. Bresson, and P. Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *Proceedings of Advances in Neural Information Processing Systems*, pp. 3844–3852, 2016.

[16] G. Di Battista, P. Eades, R. Tamassia, and I. G. Tollis. Algorithms for drawing graphs: an annotated bibliography. *Computational Geometry-Theory and Application*, 4(5):235–282, 1994.

[17] R. dos Santos Vieira, H. A. D. do Nascimento, and W. B. da Silva. The application of machine learning to problems in graph drawing a literature review. In *Proceedings of International Conference on Information, Process, and Knowledge Management*, pp. 112–118, 2015.

[18] I. L. Dryden. Shape analysis. *Wiley StatsRef: Statistics Reference Online*, 2014.

[19] 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, 2015.

[20] C. Dunne and B. Shneiderman. Improving graph drawing readability by incorporating readability metrics: A software tool for network analysts. *University of Maryland, HCIL Tech Report HCIL-2009-13*, 2009.

[21] D. K. Duvenaud, D. Maclaurin, J. Iparraguirre, R. Bombarell, T. Hirzel, A. Aspuru-Guzik, and R. P. Adams. Convolutional networks on graphs for learning molecular fingerprints. In *Proceedings of Advances in Neural Information Processing Systems*, pp. 2224–2232, 2015.

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

[23] M. Espadoto, N. S. Hirata, and A. C. Telea. Deep learning multidimensional projections. *arXiv preprint arXiv:1902.07958*, 2019.

[24] A. Frick, A. Ludwig, and H. Mehldau. A fast adaptive layout algorithm for undirected graphs (extended abstract and system demonstration). In *International Symposium on Graph Drawing*, pp. 388–403. Springer, 1994.

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

[26] P. Gajer, M. T. Goodrich, and S. G. Kobourov. A multi-dimensional approach to force-directed layouts of large graphs. In *International Symposium on Graph Drawing*, pp. 211–221. Springer, 2000.

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

[28] F. A. Gers, J. Schmidhuber, and F. A. Cummins. Learning to forget: Continual prediction with lstm. *Neural Computation*, 12:2451–2471, 2000.

[29] M. Ghoniem, J.-D. Fekete, and P. Castagliola. A comparison of the readability of graphs using node-link and matrix-based representations. In *IEEE Symposium on Information Visualization*, pp. 17–24, 2004.

[30] H. Gibson, J. Faith, and P. Vickers. A survey of two-dimensional graph layout techniques for information visualisation. *Information Visualization*, 12(3-4):324–357, 2013.

[31] C. Goodall. Procrustes methods in the statistical analysis of shape. *Journal of the Royal Statistical Society: Series B (Methodological)*, 53(2):285–321, 1991.

[32] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio. *Deep learning*, vol. 1. MIT Press Cambridge, 2016.

[33] A. Grover and J. Leskovec. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 855–864. ACM, 2016.

[34] S. Hachul and M. Jünger. Drawing large graphs with a potential-field-based multilevel algorithm. In *International Symposium on Graph Drawing*, pp. 285–295. Springer, 2004.

[35] H. Haleem, Y. Wang, A. Puri, S. Wadhwa, and H. Qu. Evaluating the readability of force directed graph layouts: A deep learning approach. *arXiv preprint arXiv:1808.00703*, 2018.

[36] W. Hamilton, Z. Ying, and J. Leskovec. Inductive representation learning on large graphs. In *Proceedings of Advances in Neural Information Processing Systems*, pp. 1024–1034, 2017.

[37] D. Harel and Y. Koren. A fast multi-scale method for drawing large graphs. In *International Symposium on Graph Drawing*, pp. 183–196. Springer, 2000.

[38] D. Harel and Y. Koren. Graph drawing by high-dimensional embedding. In *International Symposium on Graph Drawing*, pp. 207–219. Springer, 2002.

[39] M. Heimann and D. Koutra. On generalizing neural node embedding methods to multi-network problems. In *KDD MLG Workshop*, 2017.

[40] M. Henaff, J. Bruna, and Y. LeCun. Deep convolutional networks on graph-structured data. *arXiv preprint arXiv:1506.05163*, 2015.

[41] I. Herman, G. Melançon, and M. S. Marshall. Graph visualization and navigation in information visualization: A survey. *IEEE Transactions on Visualization and Computer Graphics*, 6(1):24–43, 2000.

[42] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

[43] F. M. Hohman, M. Kahng, R. Pienta, and D. H. Chau. Visual analytics in deep learning: An interrogative survey for the next frontiers. *IEEE Transactions on Visualization and Computer Graphics*, 2018.

[44] Y. Hu. Efficient, high-quality force-directed graph drawing. *Mathematica Journal*, 10(1):37–71, 2005.

[45] Y. Hu, S. G. Kobourov, and S. Veeramoni. Embedding, clustering and coloring for dynamic maps. In *2012 IEEE Pacific Visualization Symposium*, pp. 33–40. IEEE, 2012.

[46] W. Huang, P. Eades, and S.-H. Hong. Measuring effectiveness of graph visualizations: A cognitive load perspective. *Information Visualization*, 8(3):139–152, 2009.

[47] M. Jacomy, T. Venturini, S. Heymann, and M. Bastian. Forceatlas2, a continuous graph layout algorithm for handy network visualization designed for the gephi software. *PloS one*, 9(6):e98679, 2014.

[48] T. Kamada, S. Kawai, et al. An algorithm for drawing general undirected graphs. *Information Processing Letters*, 31(1):7–15, 1989.

[49] M. Kaufmann and D. Wagner. *Drawing graphs: methods and models*, vol. 2025. Springer, 2003.

[50] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[51] T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[52] M. Klammler, T. Mchedlidze, and A. Pak. Aesthetic discrimination of graph layouts. In *International Symposium on Graph Drawing and Network Visualization*, pp. 169–184. Springer, 2018.

[53] J. F. Kruiger, P. E. Rauber, R. M. Martins, A. Kerren, S. Kobourov, and A. C. Telea. Graph layouts by t-sne. In *Computer Graphics Forum*, vol. 36, pp. 283–294. Wiley Online Library, 2017.

[54] O.-H. Kwon, T. Crnovrsanin, and K.-L. Ma. What would a graph look like in this layout? a machine learning approach to large graph visualization. *IEEE Transactions on Visualization and Computer Graphics*, 24(1):478–488, 2018.

[55] A. Lancichinetti, S. Fortunato, and F. Radicchi. Benchmark graphs for testing community detection algorithms. *Physical Review. E, Statistical, Nonlinear, and Soft Matter Physics*, 78:046110, 11 2008.

[56] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521(7553):436, 2015.

[57] Y. Li, D. Tarlow, M. Brockschmidt, and R. Zemel. Gated graph sequence neural networks. *arXiv preprint arXiv:1511.05493*, 2015.

[58] T. Masui. Evolutionary learning of graph layout constraints from examples. In *Proceedings of the 7th annual ACM Symposium on User Interface Software and Technology*, pp. 103–108. ACM, 1994.

[59] B. Meyer. Self-organizing graphsa neural network perspective of graph layout. In *International Symposium on Graph Drawing*, pp. 246–262. Springer, 1998.

[60] F. Monti, D. Boscaini, J. Masci, E. Rodolà, J. Svoboda, and M. M. Bronstein. Geometric deep learning on graphs and manifolds using mixture model cnns. In *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 5425–5434. IEEE, 2017.

[61] W. J. Murdoch, C. Singh, K. Kumbier, R. Abbasi-Asl, and B. Yu. Interpretable machine learning: definitions, methods, and applications. *arXiv preprint arXiv:1901.04592*, 2019.

[62] M. Niepert, M. Ahmed, and K. Kutzkov. Learning convolutional neural networks for graphs. In *Proceedings of International Conference on Machine Learning*, pp. 2014–2023, 2016.

[63] A. Noack. Energy models for graph clustering. *J. Graph Algorithms Appl.*, 11(2):453–480, 2007.

[64] R. Pascanu, T. Mikolov, and Y. Bengio. On the difficulty of training recurrent neural networks. In *Proceedings of International Conference on Machine Learning*, pp. 1310–1318, 2013.

[65] N. Peng, H. Poon, C. Quirk, K. Toutanova, and W.-t. Yih. Cross-sentence n-ary relation extraction with graph lstms. *arXiv preprint arXiv:1708.03743*, 2017.

[66] B. Perozzi, R. Al-Rfou, and S. Skiena. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 701–710. ACM, 2014.

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

[68] A. Rosete-Suarez, M. Sebag, and A. Ochoa-Rodriguez. A study of evolutionary graph drawing. 1999.

[69] M. Spönemann, B. Duderstadt, and R. von Hanxleden. Evolutionary meta layout of graphs. In *International Conference on Theory and Application of Diagrams*, pp. 16–30. Springer, 2014.

[70] K. S. Tai, R. Socher, and C. D. Manning. Improved semantic representations from tree-structured long short-term memory networks. In *Proceedings of the 7th International Joint Conference on Natural Language Processing*, vol. 1, pp. 1556–1566, 2015.

[71] R. Tamassia. *Handbook of graph drawing and visualization*. Chapman and Hall/CRC, 2013.

[72] W. T. Tutte. Convex representations of graphs. In *Proceedings of the London Mathematical Society*, vol. 3, pp. 304–320. Wiley Online Library, 1960.

[73] W. T. Tutte. How to draw a graph. *Proceedings of the London Mathematical Society*, 3(1):743–767, 1963.

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

[75] D. Wang, P. Cui, and W. Zhu. Structural deep network embedding. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 1225–1234. ACM, 2016.

[76] R.-L. Wang and Okazaki. Artificial neural network for minimum crossing number problem. In *Proceedings of 2005 International Conference on Machine Learning and Cybernetics.*, vol. 7, pp. 4201–4204. IEEE, 2005.

[77] Y. Wang, Q. Shen, D. Archambault, Z. Zhou, M. Zhu, S. Yang, and H. Qu. Ambiguityvis: Visualization of ambiguity in graph layouts. *IEEE Transactions on Visualization and Computer Graphics*, 22(1):359–368, 2016.

[78] Y. Wang, Y. Wang, Y. Sun, L. Zhu, K. Lu, C.-W. Fu, M. Sedlmair, O. Deussen, and B. Chen. Revisiting stress majorization as a unified framework for interactive constrained graph visualization. *IEEE transactions on visualization and computer graphics*, 24(1):489–499, 2017.

[79] V. Yoghourdjian, D. Archambault, S. Diehl, T. Dwyer, K. Klein, H. C. Purchase, and H.-Y. Wu. Exploring the limits of complexity: A survey of empirical studies on graph visualisation. *Visual Informatics*, 2(4):264–282, 2018.

[80] J. You, R. Ying, X. Ren, W. L. Hamilton, and J. Leskovec. Graphrnn: a deep generative model for graphs. In *Proceedings of the 35th International Conference on Machine Learning*, 2018.

[81] Q. Zhang and S. Zhu. Visual interpretability for deep learning: a survey. *Frontiers of Information Technology & Electronic Engineering*, 19(1):27–39, 2018.

[82] J. Zhou, G. Cui, Z. Zhang, C. Yang, Z. Liu, and M. Sun. Graph neural networks: A review of methods and applications. *arXiv preprint arXiv:1812.08434*, 2018.