DiffGraph: Controllable Graph Layout Generation with Diffusion Models

Guozheng Li, Xudong Zhou, Haotian Mi, Jinyuan Liang, Min Lu, Yi Chen, Takayuki Itoh, and Chi Harold Liu

Abstract—Graph visualization is an essential way to understand the relationships within graph-structured data, and node-link diagrams are widely adopted because of their intuitive representation. However, generating satisfactory layouts remains challenging, because multiple valid layouts may exist for a single graph, and their usefulness often depends on users' analytical goals. Conventional layout algorithms (e.g., force-directed methods) typically require users to tune low-level parameters to achieve desirable results, which can be time-consuming and non-intuitive for users. Recent deep learning-based methods largely operate automatically but provide limited support for expressing user intentions. We propose DiffGraph, a controllable graph layout generation method with diffusion models. DiffGraph leverages a denoising diffusion probabilistic model, with graph neural networks serving as its core architecture. Instead of requiring users to tune low-level algorithmic parameters, DiffGraph allows users to provide high-level guidance, such as relative node positions or desired global layout properties. This design enables users to incorporate domain knowledge, highlight regions of interest, or enforce specific structural patterns, thereby supporting diverse analytical needs. Qualitative and quantitative evaluations show that DiffGraph can generate high-quality and diverse layouts while effectively incorporating user-defined constraints. A user study further demonstrates that participants generally understood the constraint mechanism and perceived the generated layouts as consistent with their intentions, while an extended usage scenario illustrates how DiffGraph can support iterative and practical graph exploration.

Index Terms—Graph, layout, constraint, diffusion model, graph neural network

1 Introduction

Graph visualization enables users to intuitively understand and explore the underlying connections and structures. Among various available graph visualizations, node-link diagrams are popular for their effectiveness in revealing the graph topology [18]. Over the years, many algorithms have been developed for the layout of node-link diagrams [7,11,13,34]. These algorithms typically involve different design principles and parameter settings, often producing different layouts for the same graph. As a result, achieving a desired visual result often requires users to select appropriate algorithms and carefully adjust their parameters based on data characteristics and analytical tasks. However, this trial-and-error approach can be time-consuming and particularly difficult for users who lack expertise in graph drawing algorithms.

Recently, there has been a growing interest in leveraging deep learning techniques for graph layout generation. Unlike conventional methods that require users to manually adjust parameters, deep learning approaches learn a direct mapping from graph structures to visual representations, thus reducing the need for explicit parameter tuning [63]. Existing deep learning approaches can be broadly categorized into supervised and unsupervised models. Supervised models [63] learn layout patterns from ground-truth layout data sets and can generalize these patterns to unseen graphs. In contrast, unsupervised models [20,21,58,61,62] optimize predefined aesthetic criteria, allowing them to generate visually appealing layouts without requiring ground-truth data. Despite these advancements, these methods often operate in a fully automatic manner, making it difficult to adapt the visualization to specific analytical tasks or personal preferences. To support the exploration and selection of various graph layouts, researchers have proposed methods that generate multiple potential layouts for the same graph [41,42], allowing users to browse and select from a range of candidates. However, this selection-based paradigm remains inefficient, as users need to inspect many candidates without explicit mechanisms to guide the generation process toward their specific intentions.

Therefore, we seek a method that satisfies user expectations for graph layouts while reducing the need for laborious parameter tuning. The core challenge lies in the vast space of potential graph layouts, arising from the diversity of available layout algorithms, each with several adjustable parameters. Additionally, translating subjective user preferences into explicit layout parameters remains inherently ill-posed.

Motivated by the ability of diffusion models to capture diverse layout distributions through iterative denoising, while naturally enabling the incorporation of user-defined constraints, we propose DiffGraph, a controllable graph layout generation method based on the diffusion model [28,55,56]. The core architecture of DiffGraph is built on denoising diffusion probabilistic models (DDPMs) [28,53,66], with graph neural networks (GNNs) [51,65] as its backbone. Furthermore, we introduce a dedicated node feature enhancement method and a graph feature augmentation technique to effectively encode layout-related characteristics. DiffGraph is trained on a diverse set of graph layout samples to learn various layout distributions. During the inference stage, DiffGraph supports the constrained generation mode, allowing users to provide intuitive guidance, such as specifying relative positions of selected nodes [64,69] or the desired global layout properties.

We evaluate the performance of DiffGraph through quantitative and qualitative experiments. The quantitative experiment results show that DiffGraph is capable of capturing key layout characteristics from the training data and generating satisfactory layouts for unseen graphs. Moreover, DiffGraph also effectively incorporates user-specified local and global constraints, enabling the generation of diverse layouts aligned with different analytical intentions. A user study further demonstrates the understandability, perceived satisfaction, and analytical usefulness of the constraint mechanism, and a usage scenario illustrating how DiffGraph supports an iterative graph analysis workflow through global steering and local refinement.

In summary, the key contributions of this paper are as follows:

- We propose DiffGraph, a diffusion model for controllable graph layout generation, allowing users to guide the layout process by specifying relative node positions as local constraint or layout metric as global constraint.
- We conduct both quantitative and qualitative experiments to demonstrate the effectiveness of the graph layout method.

- *Guozheng Li, Xudong Zhou, Haotian Mi, Jinyuan Liang, and Chi Harold Liu are with Beijing Institute of Technology.*
- *Min Lu is with The Hong Kong University of Science and Technology (Guangzhou).*
- *Yi Chen (corresponding author) is with China Food Flavor and Nutrition Health Innovation Center and Beijing Key Laboratory of Big Data Technology for Food Safety, Beijing Technology and Business University.*
- *Takayuki Itoh is with Ochanomizu University.*

Manuscript received xx xxx. 202x; accepted xx xxx. 202x. Date of Publication xx xxx. 202x; date of current version xx xxx. 202x. For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org. Digital Object Identifier: xx.xxxx/TVCG.202x.xxxxxxx

2 Related Work

2.1 Graph Layout Methods

Existing studies have proposed several rule-based graph layout algorithms, which can be divided into three main categories [6, 19]: force-directed, dimension-reduction, and multi-level methods: (1) **Force-directed methods** treat a graph as a physical system, where nodes and edges are modeled as physical objects to create aesthetically pleasing layouts. Key approaches include spring-embedded methods [13, 33] that apply physical forces, energy-based techniques [16, 34] that minimize system energy, and probabilistic methods [2, 7, 40, 71] that utilize randomness to explore layout possibilities. (2) **Dimensionality reduction methods** [5, 27, 39] aim to transform high-dimensional information, such as the length of the shortest path between nodes, into a two-dimensional space using specialized algorithms designed for dimensionality reduction. The algorithms include PivotMDS [5], utilizing multidimensional scaling, HDE [27], employing principal component analysis, and tsNET [39], leveraging stochastic neighbor embedding. (3) **Multi-level methods** [3, 14, 15, 59, 73] were proposed to speed up graph visualization by simplifying graphs into coarser representations and then generating layouts.

Furthermore, recent studies have attempted to use deep learning techniques to compute graph layout, employing both supervised and unsupervised models. Supervised models, such as DeepDrawing [63], utilize graph-based LSTMs to learn coordinate patterns from ground-truth data, generating layouts that closely resemble training examples. In contrast, unsupervised approaches optimize aesthetic criteria without relying on labeled data. DeepGD [61] and $(\text{DNN})^2$ [20, 21] employ Graph Convolutional Networks (GCNs) to minimize aesthetic losses, while Neural Aesthete [58] introduces differentiable losses to handle non-differentiable objectives. Furthermore, SmartGD [62] pioneers the use of Generative Adversarial Networks (GANs) [22] to optimize quantitative aesthetic goals. These studies demonstrate that deep learning approaches can effectively capture graph layout characteristics and generate high-quality graph layouts.

Despite advances, these methods often offer limited user interactivity, which can restrict their ability to align layouts with user intentions and accommodate diverse requirements. Moreover, finding the satisfactory layout remains challenging due to varying user preferences and task-specific demands. To support the exploration and selection of various graph layouts, researchers have proposed methods that generate multiple potential layouts for the same graph [41, 42]. These methods collect a variety of layouts and provide intuitive interaction mechanisms that allow users to explore multiple layouts for a single graph. However, this selection-based layout exploration model remains inefficient in mapping user intents to appropriate visual representations.

2.2 Constrained Graph Layout Generation

Some studies [4, 35, 50, 57] have explored constrained graph layout methods to incorporate user preferences into automated layout processes, with the aim of aligning the resulting layouts with these preferences. Typically, these methods use rule-based algorithms, which are adjusted to meet specific user requirements. Specifically, Böhringer and Paulisch introduce structural stability to manage constraints on node positioning imposed by users or application programs. Based on force-directed layout methods, researchers also develop techniques to guide nodes toward positions that adhere to the defined constraints.

Furthermore, techniques such as preventing node overlap and promoting compact cluster drawings have been integrated to enhance force-directed placement [60]. For the direct graph layout constraint, DIG-COLA [10] organizes graph nodes into different layers and integrates constraints into the stress majorization process to ensure layer separation. Additionally, intelligent graph layout [69] combines multiple subgraphs within a large graph while retaining the topological information of each subgraph through Laplacian-constrained distance embedding. By reformulating stress majorization into a vector form, a recent study [64] presents a unified framework that encodes diverse layout constraints as edge vectors, enabling the computation of constrained graph layouts through an unconstrained optimization process.

These constraint-based methods integrate user intents into automated graph layout processes, offering essential support for customized visualization tasks. However, these rule-based approaches often rely on pre-defined mathematical formulations tailored to specific aesthetic criteria or constraints. This reliance on rigid, hand-crafted objective functions inherently restricts the diversity of achievable layouts, as discussed in Section 2.1, where distinct methods are bound to generate structurally different outputs. Consequently, it is challenging to identify a single layout method applicable across various scenarios. Moreover, different constraint approaches offer unique advantages, yet rule-based methods often rely on fixed subgraph structures, hindering the incorporation of new constraints. To overcome this limitation, we employ a diffusion model framework that learns layout features from diverse methods, thereby transcending the constraints of specific mathematical formulations. This approach enables the flexible integration of heterogeneous constraints, encompassing both user-desired local subgraph structural requirements and global aesthetic criteria.

3 Background

3.1 Denoising Diffusion Probabilistic Models

Diffusion models [28, 55, 56] have emerged as a powerful class of deep-generative models with remarkable performances in various applications, including image synthesis [49], video generation [29], and molecule design [67]. DDPMs [28, 53] represent a notable subset of diffusion models, which involve two Markov chains [67]: a forward chain to add noise to ground truth data and a reverse chain to restore data from noise. The forward chain typically employs predefined parameters to transform the original data distribution into a simpler prior distribution, often a standard Gaussian distribution. In contrast, the reverse Markov chain reverses the transformations using transition kernels parameterized by deep neural networks. DDPMs are distinguished by two essential aspects: expressiveness and guidance capacity. Expressiveness denotes the model's ability to capture diverse data distributions and produce high-quality output that aligns with the underlying distribution. Guidance capacity refers to the utilization of prompts or conditions to guide the model to generate outputs that reflect the intent of the user. Given these characteristics, DDPMs are well-suited for our goals of generating graph layouts of both high quality and diversity, and at the same time, accommodating specific constraints.

3.2 Graph Neural Networks

Graph Neural Networks (GNNs) adapt deep learning techniques to the combinatorial properties of graphs, enabling the processing of graph-structured data [72]. They enhance information exchange among nodes through a message-passing mechanism, enriching node representations with contextual and relational details. This mechanism aggregates information from neighboring nodes or the entire graph and propagates it to each node. GNNs have achieved state-of-the-art performance in various graph-related tasks, such as graph classification [68], which establishes mappings between graphs and labels. Similarly, graph drawing can be conceptualized as a structure-to-layout mapping function [63], making GNNs a suitable choice for this task. Given that graph layouts should remain invariant under rotations and translations in the two-dimensional space, ensuring rotational and translational invariance is critical. E(n)-Equivariant Graph Neural Networks (EGNNs) [30, 51, 66] address this requirement by exhibiting equivariance to rotations, translations, reflections, and permutations. Inspired by EGNNs, we implement an equivariant convolutional layer for graph layout generation tasks.

4 DiffGraph Method

4.1 Task Specification

Preliminaries. The graph layout generation task can be defined as the following. A graph G can be represented as $G = (V, E)$, where V is the set of its nodes, and E is the set of its edges, with G being an undirected, connected graph. N and M indicate the number of nodes and edges. The graph layout is denoted as $X \in \mathbb{R}^{N\times 2}$, where X_i represents the two-dimensional position vector for node i. We denote the node positions

of the generated layout as X^{gen}. The general layout generation task is to build the mapping relation between G and X^{gen}.

Constraint Mode. DiffGraph offers two types of constraints to guide the graph layout generation process: local constraints and global constraints. As their names suggest, local constraints adjust the layout's local structure, while global constraints regulate its overall appearance. Our approach implements both constraint types to address the needs of various tasks. For local constraints, users can express their layout expectations by indicating the relative positions of specific nodes, denoted X^{fix}, with an empty value indicating the generation of an unconstrained layout. For global constraints, users can specify a desired global layout metric, denoted ω, such as the variance of the edge length and the number of edge crossings, which reflect the global characteristics of a graph layout. Traditionally, metrics are computed only after a layout is generated. The key innovation is learning to map metrics to layouts, enabling users to specify desired metric values and generate layouts that satisfy these constraints.

Problem Definition. As noted in Section 2.1, the layouts of the same graph can vary significantly. Numerous algorithms have been developed to address diverse objectives and contexts. In addition, certain layouts come not only from algorithmic processes, but also from user adjustments [36]. Thus, the controllable graph layout generation task defined here seeks to produce X^{gen} by integrating user expectations, expressed through local constraint X^{fix} and global constraint ω for a given graph G. This task entails predicting the coordinates of each node in a two-dimensional space, accounting for the structure of the graph, the range of possible layouts, and specified constraints. Two primary challenges arise: first, the ability to generate diverse layouts for a given graph, and second, the integration of specified constraints (the relative positions of specific nodes X^{fix} and the global metric features ω) into the layout generation process.

To address these challenges, we adopt data-driven deep learning approaches. Unlike rule-based methods, data-driven techniques can produce diverse outputs when trained on varied data and supported by expressive models. Consequently, we employ DDPMs, detailed in Section 3.1, due to their proven expressiveness in generative tasks.

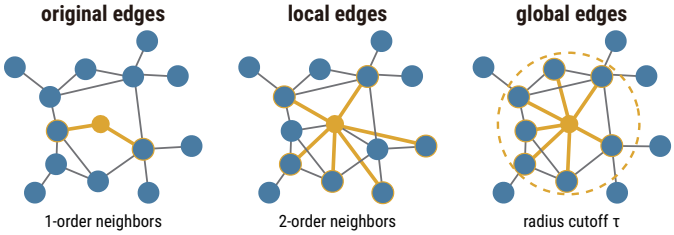

Figure 1: Three types of edges in graph feature augmentation. We select the yellow node as an example. The left graph highlights the original edges connected with the node. The middle graph highlights its two-order local edges, which connect it to neighboring nodes within two hops. The right graph highlights the global edges of the graph, which connect the node to other closed nodes based on a predefined distance threshold.

4.2 Model Inputs

To improve model performance, we incorporate feature augmentation into input processing.

Node Feature Enhancement. Enhancing node features is critical to effective layout generation. The encoding method should effectively capture the graph's topological properties. We evaluated various node encoding techniques, including node2vec [24], random walk positional encoding [9], and Laplacian eigenvector positional encoding [8], to enrich node positional features. The Laplacian positional encoding technique [8] embeds graphs in Euclidean space, integrating positional and structural information to create a local coordinate system while preserving the global graph structure. Previous work [58] has demonstrated its promising performance in graph layout tasks. Therefore, DiffGraph adopts Laplacian positional encoding to compute the positional features of nodes. Furthermore, we evaluated other node embedding techniques (see Section 5.3) to provide a complete comparison.

In our experiments, we use the k smallest non-trivial eigenvectors as the initial node features, where k is a hyperparameter. These eigenvectors encode continuous coordinates of neighboring nodes, enabling effective capture of structural information within their receptive fields during message passing. Additionally, for layout generation with global constraints, we calculate several layout metrics for each graph, as described in Section 5.1.3, and concatenate them with node embeddings throughout the graph to improve the representation of global features.

Graph Feature Augmentation. In addition to adopting an effective node feature enhancement method, we propose a graph feature augmentation technique specifically designed for graph layout tasks, further enhancing performance. Graph feature enhancement involves establishing additional node connections that directly affect message propagation efficiency in GNNs. These connections are classified into three types: original edges, local edges, and global edges, as shown in Figure 1. *Original edges* preserve the inherent topology of the raw graph, reflecting its fundamental structure. *Local edges* are extensions of the original graph topology. They connect each node to its m-order neighbors, with m determined through hyperparameter tuning through grid search. These edges broaden the perceptual range of the nodes, enabling the model to efficiently generate coherent local structures.

Original and local edges primarily support information exchange among topologically proximate nodes, which is often sufficient for general graph representations but inadequate for layout tasks. To address this, we introduce *global edges*, which dynamically connect nodes within a radius cutoff (τ) calibrated to the layout scale. These global edges allow the information to propagate beyond topologically connected neighbors, enabling the graph to capture broader relationships among positionally neighboring nodes.

4.3 Model Architecture

DiffGraph employs the architecture of a diffusion model, consisting of two processes: the forward process and the reverse process. We define a graph layout as a system composed of multiple nodes and edges in a two-dimensional space. The forward process transitions the system from an equilibrium (well-organized) state to a chaotic (noisy) one, while the reverse process refines it back to equilibrium. We denote the ground truth layout as X_0 and the sequence of perturbed layouts at diffusion step t as X_t (for $t = 1, \cdots, T$). The diffusion model identifies that the graph layout distribution at each timestep t follows a Gaussian distribution. At $t = T$, the layout X_T is maximally chaotic, adhering to a standard normal distribution $\mathcal{N}(0,1)$, while at $t = 0$, it is maximally clear. The objective of DiffGraph is to learn the reverse process, recovering the distribution of X_0 from X_T step by step, allowing for the sampling of high quality layouts from the distribution of X_0.

Forward Process: Gradually Adding Noise to the Layout. The forward process is about gradually adding noise to the clear layout X_0 over a series of steps. At each step t, Gaussian noise sampled from $\mathcal{N}(0,1)$ is added to the positions of the nodes, systematically altering the layout. The process is structured as a Markov chain, with each layout X_t depending solely on the previous one X_{t-1}:

$$q(X_t|X_{t-1}, G) = \mathcal{N}(X_t; \sqrt{1-\beta_t}X_{t-1}, \beta_t I) \tag{1}$$

To simplify this process, a closed-form expression allows for direct sampling from X_0 to X_t:

$$q(X_t|X_0, G) = \mathcal{N}(X_t; \sqrt{\bar{\alpha}_t}X_0, (1-\bar{\alpha}_t)I) \tag{2}$$

where $\bar{\alpha}_t = \prod_{i=1}^{t}(1-\beta_i)$.

Reverse Process: Learning to Recover from Chaos. The reverse process is central to DiffGraph, with the aim of reconstructing X_0 from the fully perturbed X_T through iterative refinement. This process, also a Markov chain, proceeds as follows:

$$q(X_{t-1}|X_t, G) = \mathcal{N}(X_{t-1}; \mu_\theta(X_t, G, t), \sigma_t^2 I) \tag{3}$$

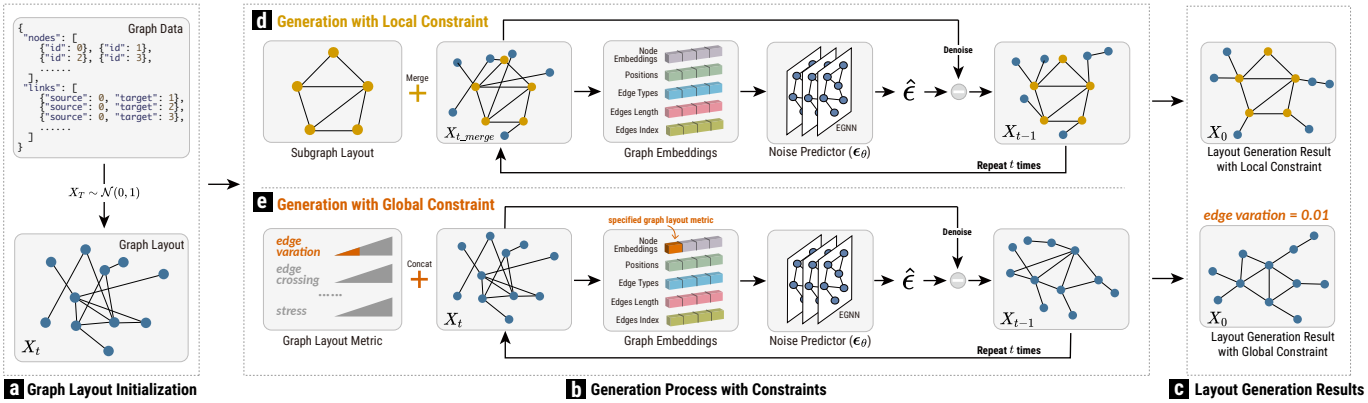

Figure 2: The constrained graph layout generation process in DiffGraph. It begins by sampling a noisy layout $X_T \sim \mathcal{N}(0,1)$ (a). After integrating local constraints (d) and global constraints (e), we embed the graph and employ a trained noise predictor to estimate the noise term $\hat{\epsilon}$ present in the current layout. Following the procedure outlined in Seq. 3, we denoise the input layout to enhance its aesthetic quality. (Notably, to effectively align the given constraints with model generation, each timestep involves resampling n times, detailed in Section 4.4.) This denoising process iterates T times, ultimately generating X_0 as the final layout (c).

where μ_θ can be expressed as:

$$\mu_\theta(X_t, G, t) = \frac{1}{\sqrt{\alpha_t}}(X_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\varepsilon_\theta(X_t, t)) \tag{4}$$

Here, ε_θ is the noise predictor, which estimates the noise in layout X_t to facilitate iterative refinement. During training, ε_θ is optimized by comparing its predictions to the actual noise added.

Structure of Noise Predictor. The noise predictor takes a graph layout X_t as input and predicts the noise $\hat{\varepsilon}$ present in the layout. To achieve this, the noise predictor employs GNNs, where node positions can be refined via the message passing mechanism (detailed in Section 3), to obtain a new layout, denoted $X_{predict}$. The predicted layout $X_{predict}$ is then subtracted from the input layout X_t to obtain $\hat{\varepsilon}$.

It is essential that the noise predictor ε_θ guarantees rotational and translational invariance. To achieve this, the noise predictor ε_θ utilizes EGNNs to predict the positions of the nodes. The noise predictor ε_θ is composed of several layers of EGNN blocks. After the multi-layer EGNN processing, the predicted layout $X_{predict}$ is finally obtained.

The equivariant convolutional layers of EGNN used in DiffGraph are as follows:

$$\text{Node positions: } x_i^l = x_i^{l-1} + \sum_{j \in \mathcal{N}(i)} \frac{x_i^{l-1} - x_j^{l-1}}{d_{ij}^{l-1}} \phi_x(h_i^l, h_j^l, m_{ij}^l) \tag{5}$$

$$\text{Node features: } h_i^l = \phi_h(h_i^{l-1}, \sum_{j \in \mathcal{N}(i)} m_{ij}^l) \tag{6}$$

$$\text{Edge messages: } m_{ij}^l = \phi_e(h_i^{l-1}, h_j^{l-1}, d_{ij}^{l-1}, e_{ij}) \tag{7}$$

Here, the equivariant convolutional layers process node embeddings, positions, edge types, and edge connections, updating them based on relative distances and relationships. l is the index of layers. h^l represents the node embedding with initial values obtained from the positional encoding of Laplacian eigenvectors (as detailed in Section 4.2). x^l corresponds to the two-dimensional coordinates of the node, while $d_{ij} = \|x_i - x_j\|$ quantifies the Euclidean distance between the i^{th} and j^{th} nodes, and m_{ij} indicates the embedding of the edge between these two nodes. e_{ij} represents the types of edges, which include original, local, and global edges. $\mathcal{N}(i)$ denotes the neighborhood of the i^{th} node. Furthermore, ϕ_x is a fully connected neural network that is tasked with projecting the updated endpoint embeddings, along with edge embeddings, into a two-dimensional space. ϕ_h updates node embeddings by leveraging information from neighboring nodes using the GIN [65] architecture. ϕ_e is also a fully connected neural network to aggregate the endpoint embeddings and the node distances for each edge.

The update of the node positions is based on direction vectors $x_i - x_j$ weighted by scalar functions (ϕ_x), to ensure predicted noise rotates correspondingly with input. This equivariance guarantees geometric consistency in ε_θ, a crucial property for graph layout generation.

Furthermore, to ease the workload of the noise predictor, DiffGraph ensures that the overall layout position remains relatively stable within the two-dimensional space. To this end, each layout is preprocessed by moving its Center of Mass (CoM) to the origin. The CoM represents the weighted average of all node positions. This centers the layout, so the model focuses on the relative arrangement of nodes rather than their absolute positions.

Training Process. During training, DiffGraph randomly selects a timestep t and generates a perturbed layout X_t by adding Gaussian noise ε to the ground truth layout X_0 (Eq. 2). The perturbed layout X_t is then fed into the noise predictor ε_θ to train it to predict the layout noise, using the noise ε as the supervision signal. As long as ε_θ accurately predicts noise ε in X_t, DiffGraph can recover the noisy layout X_t to a high-quality graph layout X_0 step by step through the reverse process defined in Eq. 3 and Eq. 4.

The above describes the typical training procedure for tasks with local constraints. In the training process for tasks with global constraints, the global feature ω of each graph is processed by an MLP and concatenated with the node embeddings, treated as part of the node embeddings; aside from this modification, the training process is consistent with that of the unconstrained tasks. Detailed definitions of these tasks are provided in Section 4.4.

4.4 Model Inference

With a well-trained noise predictor ε_θ, DiffGraph generates layouts for a given graph, starting from a randomly initialized layout and progressively denoising it to a well-structured graph layout.

The **unconstrained generation** in DiffGraph follows the reverse process used during training. It starts by sampling a noisy layout X_T for the given graph G from a standard Gaussian distribution $X_T \sim \mathcal{N}(0,1)$. The learned noise predictor ε_θ is then used to iteratively estimate and remove noise from X_T, producing $X_{T-1}, X_{T-2}, \ldots, X_0$ according to the reverse Markov chain defined by Eq. 3 and Eq. 4. Similarly to the training process, we reset the CoM of the layout to zero at each time step to remove arbitrary translations. The resulting X_0 represents a well-organized graph layout for the given graph, reflecting the layout features learned from the training set.

The **constrained generation** process extends the same reverse diffusion procedure by incorporating user-specified constraints, as illustrated in Figure 2. Similar to unconstrained generation, constrained generation starts from a randomly sampled noisy layout $X_T \sim \mathcal{N}(0, I)$, and the CoM of the layout is reset to zero at each timestep. The key difference

is that local and global constraints are injected into the reverse diffusion process to guide the generated layout toward user-specified layout intentions. Local constraints are incorporated through a resampling-and-replacement strategy during denoising, whereas global constraints are incorporated as conditional features in the node representations. Overall, these constraints jointly restrict the feasible solution space, and the model generates layouts consistent with user requirements by sampling within this constrained space.

- **Local constraints.** Local constraints specify the desired relative arrangement of a selected set of nodes. These constraints are used to preserve local structures during the generation of the layout. Because graph layouts are invariant to translation and rotation, directly imposing absolute two-dimensional coordinates may conflict with the coordinate frame of the generated layout. Therefore, we incorporate local constraints using a resampling strategy inspired by diffusion-based inpainting [43, 52]. At each reverse timestep t, DiffGraph first performs a denoising step to obtain a layout X_{t-1}. The local constraint is then aligned to the current layout by matching its center to the center of the corresponding constrained nodes, and the positions of the constrained nodes in X_{t-1} are replaced with the aligned local constraint positions. To better harmonize the constrained nodes with the remaining unconstrained nodes, we repeat this procedure with R resampling iterations at each timestep. Specifically, after replacement, the layout is diffused back from X_{t-1} to X_t according to $q(X_t \mid X_{t-1}) = \mathcal{N}(\sqrt{1-\beta_t}X_{t-1}, \beta_t I)$, and then denoised again. Here, R denotes the number of resampling iterations and is independent of the total number of diffusion timesteps T. This resampling-and-replacement process allows the model to adjust the surrounding layout while keeping the constrained local structure consistent with the user specification, as shown in Figure 2(d). Through training on datasets with diverse local substructures, the model learns the correspondence between local structural patterns and plausible global layouts. During inference, user-specified local constraints serve as conditional guidance, steering generation toward globally coherent layouts while preserving the given local structures.

- **Global constraints.** Global constraints specify desired layout-level properties, such as stress, edge crossing. These constraints are used to steer the overall visual characteristics of the generated layout. To incorporate global constraints, each metric value is log-normalized when necessary and scaled to the range $[0, 1]$. The normalized metric vector is then processed by a multilayer perceptron and subsequently concatenated with the embeddings of node coordinates initialized from randomly sampled Gaussian noise, as shown in Figure 2(e). The same global constraint representation is broadcast to all nodes, enabling each node update in the EGNN to be conditioned on the desired graph-level layout property. Except for this conditional embedding, the training and inference procedures are identical to the unconstrained generation process. This design enables the model to jointly learn layout structures and their associated global metrics, capturing the statistical dependency between layout configurations and their distributions. During inference, conditioning the reverse diffusion process on a target metric guides generation toward layouts that satisfy the desired global metrics, enabling controllable layout synthesis aligned with high-level design objectives. In this sense, global constraints act as soft guidance rather than hard optimization objectives. Therefore, the generated layouts are expected to respond to the specified metrics in a statistically consistent manner, although exact satisfaction of a target metric is not guaranteed.

DiffGraph does not rely on an externally provided initial layout as the starting point of inference. The inference process always starts from randomly sampled Gaussian noise $X_T \sim \mathcal{N}(0, I)$. Therefore, the generated layout is not affected by the coordinate position of the initial layout. When an initial layout is used at the beginning, it only serves as a convenient medium for users to select nodes or specify local/global constraints; its node positions are not directly used as the initial state of generation. Consequently, different initial layouts that encode the same constraints lead to the same conditional generation setting.

5 Evaluation

5.1 Experiment Setup

5.1.1 Dataset

To ensure that DiffGraph can learn various features of graph layout, as outlined in Section 4.1, we constructed a dataset that includes a variety of graph layout methods.

Graphs. In this work, we use the Rome dataset[1] and the SuiteSparse Matrix Collection [38]. Although benchmarking in graph drawing remains non-standardized [23], many studies rely on the Rome dataset [20, 21, 23, 58, 62]. The Rome dataset contains 11,534 undirected graphs that are broadly representative and lack distinctive features, making them suitable for modeling typical real-world networks. To further enhance the effectiveness and generalizability of our method, we also include data from the SuiteSparse Matrix Collection [38], which comprises multi-node, dense graphs. Based on these two datasets, we augmented the graphs by synthesizing 10 structurally similar instances for each graph through random node pruning (0%–40%) or simple substructure enhancement.

Layouts. To generate various layouts for each graph, we applied five layout algorithms: Kamada-Kawai (KK) [34], SGD [71], ForceAtlas2 (FA2) [33], Fruchterman-Reingold (FR) [25], SFDP [31], SmartGD [62]. These algorithms were chosen for their ability to produce varied layouts across a diverse parameter values, as well as their reliable, publicly available implementations. We generated over 300,000 layouts using the methods and parameter ranges detailed in Table 1, averaging more than 20 unique layouts per graph. We splits the dataset with a train/val/test ratio of roughly 8:1:1.

Table 1: The employed layout methods and parameter ranges.

Method	Parameter Ranges	Implementation
KK [34]	• default parameters	[26]
SGD [71]	• eps = [0.0005, 0.01], • tmax = [30, 300]	[70]
FA2 [33]	• iterations = [100, 1000], • gravity = [1.0, 6.0], • scaling ratio = [1.0, 5.0]	[32]
FR [25]	• r = [1.0, 3.0], • weight = None	[47]
SFDP [31]	• C = (0.0, 5.0], • p = (0.0, 5.0]	[47]

5.1.2 Model Configuration

The DiffGraph model is implemented using PyTorch [45] and PyTorch Geometric [12]. The noise predictor is composed of two layers of EGNN blocks. We configure the Laplacian positional encoding dimension k to three and set the hidden dimension to 256. The number of timesteps T is set to 500. For graph feature augmentation, we incorporate local edges linking nodes to their neighbors within three hops (m=3) and global edges connecting nodes within a physical neighborhood defined by a radius of three (τ=3).

For the training phase, the model employs stochastic gradient descent with a batch size of 64 on a single NVIDIA Titan RTX GPU with 24GB of memory. We use the AdamW optimizer with a learning rate of 5e-4 and a weight decay of 0.01. During constrained generation, the model performs resampling twice at each timestep. A single model was used to learn multiple potential layouts concurrently for each graph, with constraints applied to guide the generation process. The average training time for the model is about 10 hours.

5.1.3 Evaluation Metrics

We used the following metrics to evaluate layout results from multiple perspectives. The detailed calculations for the following metric can be found in the Appendix A.

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

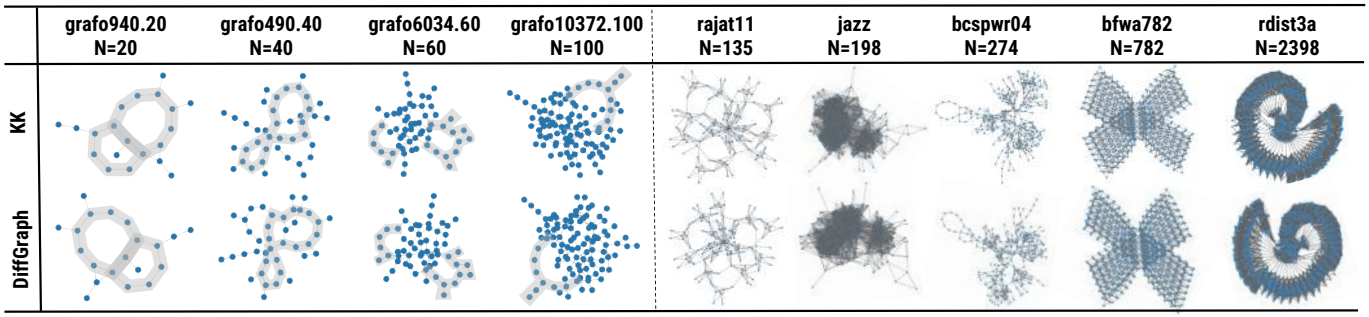

Figure 3: Qualitative comparison of the layouts generated by the DiffGraph model with Kamada-Kawai(KK) methods' layouts. N is the number of graph nodes. The grafo series comes from the Rome dataset, and other graphs are from SuiteSparse Matrix Collection [38].

Stress (m_s) [58, 62] quantifies the difference between the geometric distances of the nodes and their corresponding graph-theoretical distances, normalized by the node count.

Edge Crossing (m_e) [20, 63] quantifies the ratio between the actual number of edge crossings in a graph layout and the maximum possible number of edge crossings.

Edge Angle (m_a) measures the sum of acute crossing angles between edges in the layout.

Edge Variation (m_v) [41, 48] is used to measure the variation of edge lengths within a graph layout.

Neighbors Preservation (m_n) [1, 20] is used to assess whether structurally close nodes also exhibit proximity in the layout by computing the overlap rate of a node's k-hop neighborhood and its nearest nodes.

Minimum Angle (a_{min}) [1, 48] measures the average deviation of the angle of the adjacent incident edge from the ideal minimum angle. It is calculated as the ratio between the minimum incident of the node and the optimal angle, which is determined by the degree of the node.

Root-Mean-Square Error (RMSE) [41] is to quantify the difference between model-generated layouts and ground truth.

Coefficient of Determination (R^2) [41] evaluates the fitness of the model to the observed data. Values closer to 1 indicate better suitability.

5.2 Graph Layout Generation

5.2.1 Unconstrained Layout Generation

We began by assessing the model's unconstrained generation capability. Figure 3 presents a comparative analysis of DiffGraph against the Kamada-Kawai(KK) algorithm. In particular, the key features of the layout captured by DiffGraph are highlighted using the gray background in Figure 3. Although the layouts generated by DiffGraph may exhibit minor rotational differences from the ground truth due to rotational and translational invariance, these differences do not detract from overall layout quality. These results indicate that DiffGraph can learn the layout characteristics of the training samples and generate layouts of reasonable quality for unseen graphs. To further evaluate the robustness of the generative process, we additionally assess the stability under fixed random seeds and the diversity across different random seeds in the appendix.

5.2.2 Controlled Layout Generation with Local Constraints

Local constraints allow users to express their preferences by specifying the relative positions of certain nodes. These nodes may form a subgraph or consist of multiple disconnected components. During the generation process, DiffGraph incorporates these local constraints into the model output to produce layouts that adhere to them, as detailed in Section 4.4. These constraints encompassed shapes such as squares, circles, lines, or highly specific structures, with multiple constraints applied in a single graph. The results shown in Fig. 4 demonstrate that DiffGraph adheres consistently to the constraints and produces diverse layouts for a given graph under varying constraints.

To evaluate the effectiveness of our method, we conducted qualitative and quantitative evaluations against the previous method, ICGV [64], as shown in Figure 5 and Table 2. Applying the same local constraints to the same graph, we calculated the percentage difference (PD) between the layout results of DiffGraph ($Metric(C)$) and ICGV ($Metric(I)$) in the corresponding metrics, as shown in Table 2.

$$PD = \frac{Metric(C) - Metric(I)}{Metric(I)} \times 100\% \tag{8}$$

Negative values indicate that DiffGraph outperforms ICGV, while positive values indicate the opposite. Overall, our method performs better than ICGV in most metrics, demonstrating that the diffusion model can effectively incorporate local constraint features during the denoising process. We also observe that PD exhibits greater fluctuations for graphs with fewer nodes, although these differences diminish as the number of nodes increases.

In addition, to provide a more comprehensive assessment of the model's behavior, we analyze a set of extreme constraints in the appendix, referred to as structurally inconsistent constraints, that may adversely affect layout quality.

Table 2: Quantitative comparison of methods DiffGraph and ICGV. We impose the same local constraints on each graph and calculate the difference ratio of their metrics. Negative values indicate that DiffGraph performs better than ICGV. Rome denotes the aggregate metric difference calculated from 1,000 test graphs selected from the Rome dataset. Other graphs are sourced from the SuiteSparse Matrix Collection [38].

Graph	m_s	m_e	m_a	m_v	m_n
Rome	-6.6%	-28.7%	-32.6%	-7.5%	-5.7%
G1	-5.6%	-12.5%	-15.2%	-5.4%	-7.2%
lp_share2b	-3.8%	-2.6%	-4.3%	-1.4%	-3.2%
rajat11	-2.1%	-1.4%	-2.0%	0.8%	1.5%
bcspwr04	1.1%	-3.5%	-2.2%	1.8%	-2.5%

5.2.3 Controlled Layout Generation with Global Constraints

Global constraints enable users to specify specific metrics of the graph layout, thereby influencing the overall layout results. To maintain consistency with the metrics referenced earlier, we adopted the five metrics to evaluate graph layouts outlined in Section 5.1.3: *edge crossing*, *edge variation*, *minimum angle*, *neighbors preservation*, and *stress*. To improve user comprehension of the strengths of these metrics, their value ranges were normalized to the interval $[0, 1]$. We calculated the corresponding metric values for all layouts in our training set and inputted them into the model as described in Section 4.2. During the inference process, we can specify the desired metric value as the global constraint, which guides the layout generation process.

When evaluating the effectiveness of our approach, we note that comparing the absolute difference between specified global constraint values and computed metric values is impractical due to normalization. Instead, we assessed their positive correlation to validate our method: whether an increase in the global constraint corresponds to an increase in the computed metric value of the inferred layout. To quantitatively

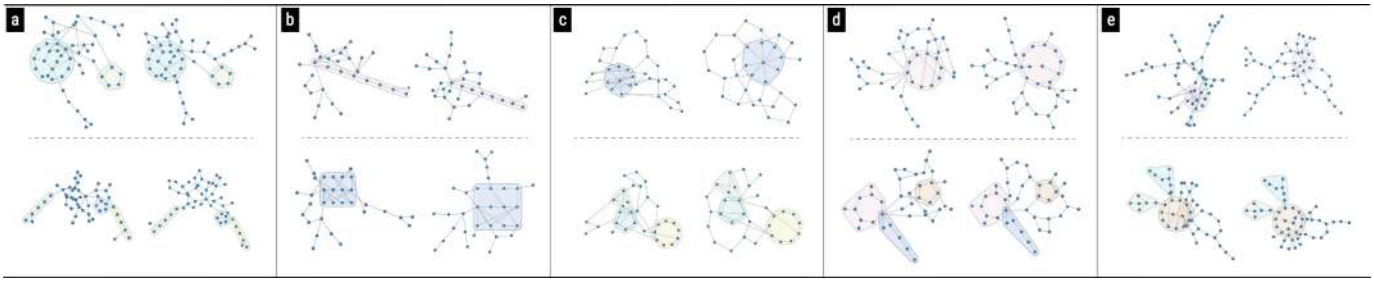

Figure 4: Th figure shows possible results obtained by applying two different local constraints to the same graph. In each group, the left side shows the local constraint specified on the original graph, and the right side shows the layout generated under that constraint. Each subfigure illustrates how different local constraints lead to different visualization results for the same graph.

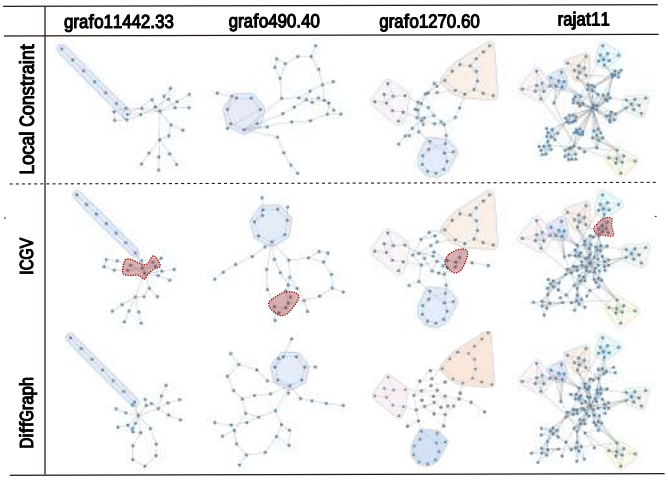

Figure 5: Qualitative comparison between DiffGraph and ICGV. The grafo series comes from the Rome dataset. Graph rajat11 is taken from SuiteSparse Matrix Collection [38]. In the ICGV layout results, red dotted marks highlight the edge crossings in the graph layout results.

measure this correlation, we employed the Pearson Correlation Coefficient (P_c) as an indicator, which is used to measure the strength of the linear correlation between two groups of variables. The closer P_c is to one, the more linearly correlated the two variables are.

In our experiments, we randomly selected 200 graphs from the test set, ensuring an even distribution of node counts. For each metric, we set global constraint values ω as [0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0], to guide DiffGraph in generating 11 layouts for each graph. The actual metric values of the generated layouts were then calculated as Y to calculate P_c. The results in Figure 6 display node count on the horizontal axis and P_c on the vertical axis. Across the five metrics, the coefficient P_c approaches 1, indicating a positive correlation between the global constraints specified and the actual metric values of the generated layouts. Notably, as node count increases, P_c tends to approach 1 more closely, possibly because graphs with fewer nodes exhibit limited layout variation, which limits metric differentiation. In contrast, the number of nodes expands the layout variation space, allowing DiffGraph to better generate layouts consistent with global constraints. In general, this experiment demonstrates that DiffGraph adheres effectively to global constraint guidance, producing layouts with a positive correlation to the specified constraints.

In addition, we illustrate the effect of varying the *edge variation* global constraint on the generated layouts, as shown in Figure 7. As the *edge variation* metric increases, the edge lengths of the graph become noticeably more uneven. Specifically, lower *edge variation* values produce more uniform and aesthetically pleasing layouts, while higher values result in more pronounced primary structures with contracted edge formations. Moreover, we show the generation results under several additional sets of global constraints in the appendix.

5.3 Ablation Study

We conducted an ablation study and trained different models on Rome graphs. Each model generated 1000 graph layouts, followed by a quantitative evaluation of each model's generation performance using the aforementioned metrics. Table 3 presents the results of the similarities based on aesthetic metric. In this table, *GCN*, *GIN*, and *GAT* indicate that we used different common graph neural networks as the backbone network in DiffGraph rather than the EGNNs. Given that common GNNs do not account for rotation and translation invariance, their effectiveness is significantly diminished, emphasizing the necessity of EGNNs in graph layout generation tasks. The entry labeled *No Global Edges* indicates that we did not incorporate global edges during graph feature augmentation (see Section 4.2), and other configurations remain consistent with DiffGraph. Nevertheless, the model experienced a notable decline in performance without global edges, emphasizing the significance of dynamically adding them based on the nodes' actual positions. *RandomWalkPE* refers to using the random walk positional encoding to initialize the features of the nodes in DiffGraph, in contrast to the Laplacian eigenvector positional encoding. These two encoding methods do not show significant variance in performance. However, Laplacian encoding exhibits slightly superior overall effectiveness, with only minor discrepancies observed in the edge variation and neighbor preservation metrics.

5.4 Usage Scenario

We present a usage scenario to demonstrate how DiffGraph can support practical graph analysis workflows. After obtaining a trained model, analysts can use global and local constraints to progressively refine graph layouts according to their evolving analysis goals, as illustrated in Figure 8. Rather than treating graph layout as a one-shot optimization result, this scenario presents graph exploration as an iterative sensemaking process in which analysts inspect an initial overview, identify structures of interest, formulate layout intentions as constraints, and generate alternative layouts for further analysis.

The usage scenario is based on the Les Misérables character co-occurrence network [37], which represents relationships among major characters in the novel. Figure 8(a) shows the initial layout generated. This layout provides a general overview of the network, allowing the analyst to identify highly connected characters and approximate community structure. However, some character groups remain visually entangled, making it difficult to distinguish communities.

To support community-level exploration, the analyst first applies a global constraint by setting a high *neighbor preservation* value. This constraint encourages structurally close nodes to remain spatially close in the generated layout. As shown in Figure 8(b), the resulting layout makes densely connected character groups more visually coherent. This helps the analyst distinguish major communities in the overall network. In this step, the global constraint serves as a high-level steering mechanism for transforming an ambiguous overview into a layout that better supports community inspection.

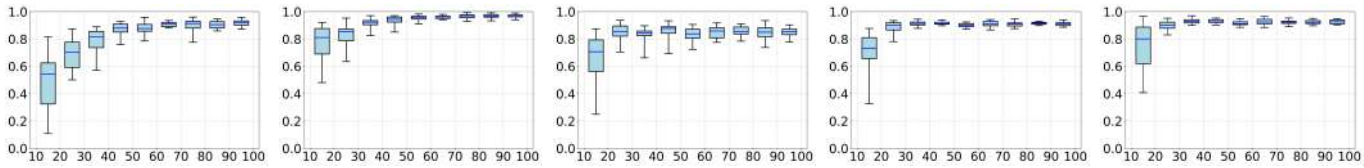

Figure 6: This figure presents box plots that illustrate the Pearson correlation coefficient between the predefined global constraint values and the actual layout metrics. The x-axis is the number of nodes. The y-axis is the Pearson correlation coefficient P_c.

Table 3: The quantitative evaluation of aesthetic metrics-based similarity on the Rome graphs among six methods. The best performance is shown in bold, while the second best is underlined.

Methods	Edge Crossing		Edge Variation		Minimum Angle		Neighbors Preservaion		Stress	
	RMSE ↓	R^2 ↑	RMSE ↓	R^2 ↑	RMSE ↓	R^2 ↑	RMSE ↓	R^2 ↑	RMSE ↓	R^2 ↑
GCN	0.2147	-1.2E+3	0.0164	-25.1467	0.3136	-11.4517	0.4479	-5.5374	45.9871	-6.7E+5
GIN	0.1777	-8.1E+2	0.0176	-28.9209	0.3109	-11.3560	0.4363	-5.3508	8.9E+28	-2.5E+60
GAT	0.2161	-1.2E+3	0.0196	-36.2213	0.3139	-11.4707	0.4519	-5.6555	45.0462	-6.4E+5
No Global Edges	0.0787	-1.5E+2	0.0129	-15.0486	0.1027	-0.3347	0.1314	0.4377	0.0744	-0.7453
RandomWalkPE	0.0038	0.6308	**0.0027**	**0.2882**	0.0243	0.9251	0.0473	**0.9390**	0.0636	-0.2787
DiffGraph	**0.0034**	**0.7097**	**0.0027**	0.2683	**0.0242**	**0.9261**	**0.0467**	0.9290	**0.0428**	**0.4231**

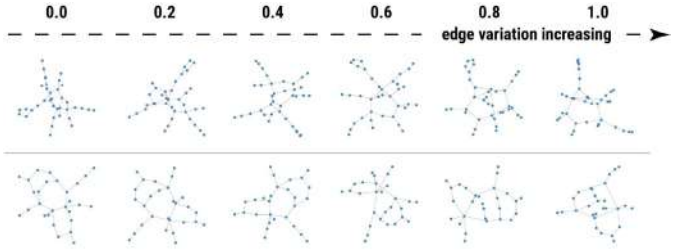

Figure 7: This figure presents graph layouts generated by DiffGraph, constrained globally by the edge variation metric and assessed across a range of values. Each row corresponds to the same graph.

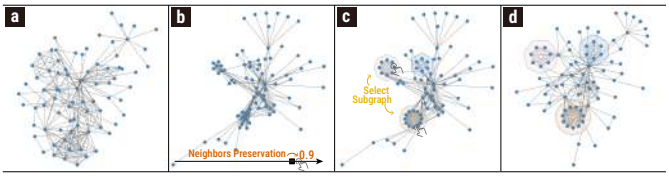

Figure 8: A usage scenario for DiffGraph on the Les Misérables character co-occurrence network. (a) original graph layout, (b) adding global constraints, (c) adding local constraints, and (d) generating the final layout.

After obtaining a clearer community-level layout, the analyst focuses on a selected group of characters for detailed inspection. For example, the analyst may select communities containing several central characters and examine how these groups connect to other nodes. A local constraint is then applied to arrange the selected nodes into a compact and interpretable structure, so that their internal relationships and external links can be inspected. As shown in Figure 8(c), the constrained layout preserves the analyst-specified local arrangement while maintaining the surrounding graph context. This allows the analyst to inspect whether the selected characters form a cohesive group and whether some nodes serve as bridges to other communities.

Finally, by combining global and local constraints, DiffGraph generates a layout that supports both overview and detail-oriented analysis, as shown in Figure 8(d). The analyst can simultaneously observe the community-level organization of the network and inspect a selected subgraph in a more controlled and interpretable form. This workflow demonstrates that the constraints in DiffGraph are not merely numerical controls over layout metrics. Instead, they can be used to express practical analytical intentions, such as emphasizing communities, preserving meaningful subgraph structures, comparing alternative organizations of selected nodes, and inspecting bridge relationships between groups.

This scenario illustrates how DiffGraph can be integrated into a realistic graph analysis workflow. Graph exploration is often an iterative process in which analysts move between overview, focus, comparison, and refinement. A fixed automatically generated layout may provide a useful starting point, but it often cannot accommodate the analyst's changing interests during exploration. DiffGraph supports this process by allowing analysts to use global constraints to steer high-level layout properties, such as community separation or neighborhood preservation, and use local constraints to refine specific regions of interest. Analysts can then generate and compare alternative layouts under different constraints to examine whether visual patterns remain consistent.

5.5 User Study

We implemented an interactive prototype system based on DiffGraph and conducted a user study to examine whether users could understand the proposed constraint mechanism and whether the generated layouts were perceived as satisfying meaningful analytical intentions. We recruited 10 participants, including graduate students and researchers with experience in graph visualizations.

The study used representative publicly available graphs with recognizable structural patterns, including the Les Misérables character co-occurrence network and the Political Books co-purchasing network[2]. These graphs were selected because they contain interpretable communities or meaningful node relationships, allowing participants to formulate layout intentions. At the beginning of the study, participants were shown the initial layout of each graph and were briefly introduced to the meanings of local and global constraints. They were then asked to freely explore the graph using the DiffGraph prototype system, with the general goal of identifying meaningful graph structures, such as communities and bridge nodes.

During the exploration process, participants could use both types of constraints supported by DiffGraph. Participants were encouraged to iteratively adjust constraints, inspect the generated layouts, and compare alternative results during exploration. This process allowed them to use global constraints for high-level layout steering and local constraints for detailed inspection of specific subgraphs.

[2]https://networks.skewed.de/net/polbooks

After the exploration task, participants rated the constraint-guided layout generation process using a five-point Likert scale. Overall, participants gave positive ratings for constraint understandability ($\mu = 4.40, \sigma = 0.52$), expressiveness of layout intentions ($\mu = 4.20, \sigma = 0.79$), perceived constraint satisfaction ($\mu = 4.50, \sigma = 0.53$), layout readability ($\mu = 4.60, \sigma = 0.52$), and usefulness for exploratory graph analysis ($\mu = 4.50, \sigma = 0.53$). In the post-study interview, participants reported that local constraints helped them inspect selected subgraphs, while global constraints were useful for steering community separation, edge uniformity, and visual clutter. Some participants also noted that global metric-based constraints were less intuitive than local constraints. These findings complement the quantitative evaluation by showing that DiffGraph's constraints were perceived as meaningful and interpretable in exploratory graph analysis.

6 Discussion and Future Work

DiffGraph is designed to address the controllable graph layout generation task with deep learning. Our experimental results provide preliminary evidence that the model supports constrained generation by incorporating layout expectations as local or global constraints. To our knowledge, this work indicates an initial exploration of applying diffusion models to graph layout. Although the findings are encouraging, further investigation and more comprehensive evaluations are needed to fully establish the effectiveness of this approach.

Supporting Realistic Graph Analysis Workflows. DiffGraph can support realistic graph analysis workflows by enabling iterative, human-guided layout exploration. In practice, analysts often need to inspect graph structures at different scales and refine visual representations according to evolving analytical goals. By allowing users to specify both global layout properties and local node arrangements, DiffGraph turns layout generation from a one-shot automatic process into a controllable exploration process. Analysts can use global constraints to steer overall visual characteristics, such as community separation or neighborhood preservation, and then apply local constraints to focus on regions of interest while maintaining the surrounding graph context. This workflow helps users examine alternative visual organizations and investigate network structures from multiple perspectives. More broadly, DiffGraph illustrates how analytical intent can be incorporated directly into graph layout generation, supporting a human-in-the-loop approach to graph visualization.

Inference Time. DiffGraph uses diffusion models, recognized for their remarkable expressiveness, enabling effective learning of the data distribution of the training set and the production of high-quality graph layouts. On the Rome dataset, the inference time of DiffGraph depends primarily on the number of denoising steps, while the size of the graph has a relatively limited impact on the inference time, since GPU memory can accommodate the entire graph for parallel processing. We evaluated the impact of denoising steps (200-5000) on inference time, finding a range of less than one second to over 15 seconds. In particular, a setting of 200 denoising steps yields satisfactory graph layouts. More specifically, the overall average time for 200 denoising steps is about 0.8 seconds. Further details on graph layout generation across different denoising steps are available in the Appendix. In addition, several techniques [44, 54] have been developed to improve the inference efficiency of diffusion models. For example, DDIM [54] can accelerate sampling and reduce inference time. Moreover, the denoising process in DiffGraph could be further optimized, for example, employing larger batch sizes. In the future, we plan to further explore the utilization of diffusion models in different graph layout scenarios.

Comparison with the Subgraph Matching Algorithm. DiffGraph can incorporate the constraints provided in the layout generation process and produce multiple layouts for a given graph, as demonstrated in Figure 4. This constrained generation process differs significantly from dataset-based subgraph matching. First, the subgraph matching approach cannot cover all graph structures because it requires generating graph layouts based on different layout algorithms before the matching process. In contrast, DiffGraph can capture the graph and layout features of a sample set and then apply them to generate layouts for new graphs. Second, when handling a sizable dataset, the subgraph matching approach for layout retrieval becomes impractical due to its time-intensive nature. In comparison, DiffGraph implicitly learns the layout knowledge of graphs, ensuring that expanding the dataset does not increase the time required for the generation of the layout.

Additional Approaches for Constrained Graph Layout Generation. In this study, our approach integrates local and global constraints to steer the generation process toward layouts aligned with user expectations. The flexibility of the diffusion framework allows us to handle multiple types of generation tasks—general generation, local constraints, and global constraints—within a single unified process. Although this method effectively reflects user preferences, it does not suggest that our constraint-based approach is flawless. For example, regarding global constraints, we utilized the evaluation metrics of Section 5.1.3 in our experiments (see Section 5.2.3) to ensure consistency and mitigate reader confusion. While these metrics are widely adopted for layout assessment, they may not adequately capture visual expressiveness. In future work, we plan to develop more intuitive metrics, such as the uniformity of node distribution in layouts or the flatness of the layout structure, to more effectively represent visual characteristics of graph layouts. As evident from the design in Section 4.2, DiffGraph is well-suited to accommodate new metrics.

Moreover, unlike traditional rule-based constrained layout generation methods, DiffGraph is not tailored to a specific constraint technique, enabling it to support diverse constraint specifications without requiring substantial modifications to its core framework. The ability of DiffGraph to perform constrained generation arises from its inherent expressiveness, allowing the generation of multiple layouts for a given graph. Constraints act as a guiding mechanism for the pre-trained model, ensuring that the generated layouts align with user expectations. In our approach, both local and global constraints are implemented within the consistent main structure of DiffGraph, with only minor differences in input specifications. In future work, we aim to enhance DiffGraph by integrating additional constraint methods, making the constrained generation process more intuitive. These extensions include cluster shape constraints for controlling the geometric organization of node groups and community boundary constraints for regulating inter-group separation and interaction patterns, thereby improving layout interpretability. We also plan to support sketch-based constraints and natural language instructions, allowing users to specify layout intent in a more natural and intuitive manner.

Scalability. Our evaluation focuses on graphs with up to thousands of nodes. Scaling DiffGraph to substantially larger graphs is challenging because diffusion-based generation requires multiple denoising steps, and each step performs GNNs updates over nodes and edges. Consequently, both inference time and memory consumption increase with graph size, which may limit direct application to graphs with hundreds of thousands or millions of nodes. In addition, training models for large-scale graph layout generation would require substantially larger and more diverse layout datasets, which are currently difficult to obtain. The lack of publicly available large-scale graph layout benchmarks with reliable reference layouts also makes rigorous quantitative evaluation challenging. Future work could explore multilevel, sampling-based, or subgraph-based diffusion architectures to improve scalability.

7 Conclusion

This paper presents DiffGraph, a controllable method for generating graph layout based on diffusion models. The core architecture of DiffGraph uses DDPMs with GNNs. In addition, DiffGraph incorporates a node feature and a graph feature enhancement technique to capture graph layout characteristics more effectively. By incorporating user-specified layout expectations as local and global constraints, DiffGraph allows constrained generation within the layout generation process. We evaluated DiffGraph through both qualitative and quantitative experiments. The results indicate that DiffGraph is capable of capturing the characteristics of the layout of the training set and generating layouts of comparable quality. Furthermore, DiffGraph can produce different layouts for a given graph under varying constraints. These findings suggest that DiffGraph is feasible for graph layout generation, and diffusion models may offer a promising direction for further exploration.

Acknowledgments

This work was supported by the NSFC (62302038, 62572054, U23B2009), China Railway Group Co. Ltd. Science and Technology Research and Development Program (P2025J001), Open Fund of the China Food Flavor and Nutrition Health Innovation Center and the National Key Laboratory of Data Space Technology and System.

References

[1] R. Ahmed, F. De Luca, S. Devkota, S. Kobourov, and M. Li. Graph drawing via gradient descent,$(GD)^2$. In *Proc. Symp. Graph Drawing (GD)*, pp. 3–17, 2020. doi: 10.1007/978-3-030-68766-3_1 6, 12

[2] R. Ahmed, F. De Luca, S. Devkota, S. Kobourov, and M. Li. Multicriteria scalable graph drawing via stochastic gradient descent, $(SGD)^2$. *IEEE Transactions on Visualization and Computer Graphics*, 28(6):2388–2399, 2022. doi: 10.1109/TVCG.2022.3141538 2

[3] D. Archambault, T. Munzner, and D. Auber. TopoLayout: Multilevel graph layout by topological features. *IEEE Transactions on Visualization and Computer Graphics*, 13(2):305–317, 2007. doi: 10.1109/TVCG.2007.46 2

[4] K.-F. Böhringer and F. N. Paulisch. Using constraints to achieve stability in automatic graph layouts. In *Proc. ACM Conf. Human Factors in Computing Systems (CHI)*, pp. 43–51, 1990. doi: 10.1145/97243.97250 2

[5] U. Brandes and C. Pich. Eigensolver methods for progressive multidimensional scaling of large data. In *Proc. Symp. Graph Drawing (GD)*, pp. 42–53, 2006. doi: 10.1007/978-3-540-70904-6_6 2

[6] S.-H. Cheong and Y.-W. Si. Force-directed algorithms for schematic drawings and placement: A survey. *Proc. Int. Conf. Information Visualisation (IV)*, 19(1):65–91, 2020. doi: 10.1177/1473871618821740 2

[7] R. Davidson and D. Harel. Drawing graphs nicely using simulated annealing. *ACM Transactions on Graphics*, 15(4):301–331, 1996. doi: 10.1145/240701.240704 1, 2

[8] V. P. Dwivedi, C. K. Joshi, T. Laurent, Y. Bengio, and X. Bresson. Benchmarking graph neural networks. *CoRR*, abs/2003.00982, 2020. doi: 10.48550/arXiv.2003.00982 3

[9] V. P. Dwivedi, A. T. Luu, T. Laurent, Y. Bengio, and X. Bresson. Graph neural networks with learnable structural and positional representations. *CoRR*, abs/2110.07875, 2021. doi: 10.48550/arXiv.2110.07875 3

[10] T. Dwyer and Y. Koren. DIG-COLA: directed graph layout through constrained energy minimization. In *Proc. IEEE Symp. Information Visualization (InfoVis)*, pp. 65–72, 2005. doi: 10.1109/INFVIS.2005.1532130 2

[11] P. Eades. A heuristic for graph drawing. *Congressus Numerantium*, 42:149–160, 1984. 1, 12

[12] M. Fey and J. E. Lenssen. Fast graph representation learning with pytorch geometric. *CoRR*, abs/1903.02428, 2019. doi: 10.48550/arXiv.1903.02428 5

[13] T. M. Fruchterman and E. M. Reingold. Graph drawing by force-directed placement. *Software: Practice and experience*, 21(11):1129–1164, 1991. doi: 10.1002/spe.4380211102 1, 2

[14] P. Gajer, M. T. Goodrich, and S. G. Kobourov. A multi-dimensional approach to force-directed layouts of large graphs. In *Proc. Symp. Graph Drawing (GD)*, pp. 211–221, 2000. doi: 10.1007/3-540-44541-2_20 2

[15] E. R. Gansner, Y. Hu, S. North, and C. Scheidegger. Multilevel agglomerative edge bundling for visualizing large graphs. In *Proc. IEEE Pacific Visualization Symposium (PacificVis)*, pp. 187–194, 2011. doi: 10.1109/PACIFICVIS.2011.5742389 2

[16] E. R. Gansner, Y. Koren, and S. North. Graph drawing by stress majorization. In *Proc. Symp. Graph Drawing (GD)*, pp. 239–250, 2005. doi: 10.1007/978-3-540-31843-9_23 2

[17] E. R. Gansner and S. C. North. An open graph visualization system and its applications to software engineering. *Software: practice and experience*, 30(11):1203–1233, 2000. doi: 10.1002/1097-024X(200009)30:11<1203::AID-SPE334>3.0.CO;2-G 12

[18] M. Ghoniem, J.-D. Fekete, and P. Castagliola. A comparison of the readability of graphs using node-link and matrix-based representations. In *Proc. IEEE Symp. Information Visualization (InfoVis)*, pp. 17–24. Ieee, 2004. doi: 10.1109/INFVIS.2004.1 1

[19] H. Gibson, J. Faith, and P. Vickers. A survey of two-dimensional graph layout techniques for information visualisation. *Proc. Int. Conf. Information Visualisation (IV)*, 12(3-4):324–357, 2013. doi: 10.1177/1473871613487082 2

[20] L. Giovannangeli, F. Lalanne, D. Auber, and R. Bourqui. Toward efficient deep learning for graph drawing (DL4GD). *IEEE Transactions on Visualization and Computer Graphics*, pp. 1–16, 2022. doi: 10.1109/TVCG.2022.3222186 1, 2, 5, 6, 12

[21] L. Giovannangeli, F. Lalanne, D. Auber, R. Giot, and R. Bourqui. Deep neural network for drawing networks,$(DNN)^2$. In *Proc. Symp. Graph Drawing (GD)*, pp. 375–390, 2021. doi: 10.48550/arXiv.2108.10822 1, 2, 5

[22] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair et al. Generative adversarial nets. *Advances in Neural Information Processing Systems*, 27, 2014. 2

[23] F. Grötschla, J. Mathys, R. Veres, and R. Wattenhofer. Core-gd: A hierarchical framework for scalable graph visualization with gnns. *arXiv preprint arXiv:2402.06706*, 2024. doi: 10.48550/arXiv.2402.06706 5

[24] A. Grover and J. Leskovec. node2vec: Scalable feature learning for networks. In *Proc. ACM Conf. Knowledge Discovery and Data Mining (SIGKDD)*, pp. 855–864, 2016. doi: 10.1145/2939672.2939754 3

[25] S. Hachul and M. Jünger. Drawing large graphs with a potential-field-based multilevel algorithm. In *Proc. Symp. Graph Drawing (GD)*, pp. 285–295, 2004. doi: 10.1007/978-3-540-31843-9_29 5

[26] A. Hagberg, P. Swart, and D. S Chult. Exploring network structure, dynamics, and function using networkx. Technical report, Los Alamos National Lab.(LANL), Los Alamos, NM (United States), 2008. doi: 10.25080/TCWV9851 5

[27] D. Harel and Y. Koren. Graph drawing by high-dimensional embedding. In *Proc. Symp. Graph Drawing (GD)*, pp. 207–219, 2002. doi: 10.1007/3-540-45848-4_18 2

[28] J. Ho, A. Jain, and P. Abbeel. Denoising diffusion probabilistic models. In *Proc. Conf. Neural Information Processing Systems (NeurIPS)*, 2020. 1, 2

[29] J. Ho, T. Salimans, A. Gritsenko, W. Chan, M. Norouzi, and D. J. Fleet. Video diffusion models. *arXiv:2204.03458*, 2022. doi: 10.48550/arXiv.2204.03458 2

[30] E. Hoogeboom, V. G. Satorras, C. Vignac, and M. Welling. Equivariant diffusion for molecule generation in 3d. In *Proc. Int. Conf. Machine Learning (ICML)*, pp. 8867–8887, 2022. 2

[31] Y. Hu. Efficient, high-quality force-directed graph drawing. *Mathematica journal*, 10(1):37–71, 2005. 5, 12

[32] M. Jacomy. Fastest gephi's forceatlas2 graph layout algorithm implemented for python and networkx. `https://github.com/bhargavchippada/forceatlas2`, 2017. 5

[33] 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):1–12, 2014. doi: 10.1371/journal.pone.0098679 2, 5, 12

[34] T. Kamada and S. Kawai. An algorithm for drawing general undirected graphs. *Inf. Process. Lett.*, 31(1):7–15, 1989. doi: 10.1016/0020-0190(89)90102-6 1, 2, 5, 12

[35] T. Kamps, J. Kleinz, and J. Read. Constraint-based spring-model algorithm for graph layout. In *Proc. Symp. Graph Drawing (GD)*, pp. 349–360, 1995. doi: 10.1007/3-540-60477-8_30 2

[36] S. Kieffer, T. Dwyer, K. Marriott, and M. Wybrow. HOLA: Human-like orthogonal network layout. *IEEE Transactions on Visualization and Computer Graphics*, 22(1):349–358, 2016. doi: 10.1109/TVCG.2015.2467451 3

[37] D. E. Knuth. *The Stanford GraphBase: a platform for combinatorial computing*, vol. 1. AcM Press New York, 1993. 7

[38] S. P. Kolodziej, M. Aznaveh, M. Bullock, J. David, T. A. Davis, M. Henderson et al. The suitesparse matrix collection website interface. *Journal of Open Source Software*, 4(35):1244, 2019. doi: 10.21105/joss.01244 5, 6, 7

[39] 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*, pp. 283–294, 2017. doi: 10.1111/cgf.13246 2

[40] M. Kudelka, P. Krömer, M. Radvanský, Z. Horak, and V. Snásel. Efficient visualization of social networks based on modified sammon's mapping. *Swarm Evol. Comput.*, 25:63–71, 2015. doi: 10.1016/j.swevo.2015.08.006 2

[41] 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, 2017. doi: 10.1109/TVCG.2017.2743858 1, 2, 6, 12

[42] O.-H. Kwon and K.-L. Ma. A deep generative model for graph layout. *IEEE Transactions on Visualization and Computer Graphics*, 26(1):665–675, 2019. doi: 10.1109/TVCG.2019.2934396 1, 2

[43] A. Lugmayr, M. Danelljan, A. Romero, F. Yu, R. Timofte, and L. Van Gool. Repaint: Inpainting using denoising diffusion probabilistic models. In *Proc. IEEE / CVF Conf. Computer Vision and Pattern Recognition (CVPR)*, pp. 11461–11471, 2022. doi: 10.1109/CVPR52688.2022.01117 5

[44] A. Q. Nichol and P. Dhariwal. Improved denoising diffusion probabilistic models. In *Proc. Int. Conf. Machine Learning (ICML)*, pp. 8162–8171, 2021. doi: 10.5555/3454287.3455037 9

[45] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan et al. PyTorch: An imperative style, high-performance deep learning library. In *Proc. Conf. Neural Information Processing Systems (NeurIPS)*, pp. 8024–8035, 2019. doi: 10.5555/3454287.3455008 5

[46] K. Pearson. Notes on regression and inheritance in the case of two parents proceedings of the royal society of london, vol. 58, 1895. doi: 10.1098/rspl.1895.0041 12

[47] T. P. Peixoto. The graph-tool python library. `https://graph-tool.skewed.de/`, 2017. 5

[48] H. C. Purchase. Metrics for graph drawing aesthetics. *Journal of Visual Languages and Computing*, 13(5):501–516, 2002. doi: 10.1016/S1045-926X(02)90232-6 6, 12

[49] R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer. High-resolution image synthesis with latent diffusion models. In *Proc. IEEE / CVF Conf. Computer Vision and Pattern Recognition (CVPR)*, pp. 10684–10695, 2022. doi: 10.1109/CVPR52688.2022.01043 2

[50] K. Ryall, J. Marks, and S. Shieber. An interactive constraint-based system for drawing graphs. In *Proc. ACM Conf. Symposium on User Interface Software and Technology (UIST)*, pp. 97–104, 1997. doi: 10.1145/263407.263521 2

[51] V. G. Satorras, E. Hoogeboom, and M. Welling. E (n) equivariant graph neural networks. In *Proc. Int. Conf. Machine Learning (ICML)*, pp. 9323–9332, 2021. doi: 10.48550/arXiv.2102.09844 1, 2

[52] A. Schneuing, Y. Du, C. Harris, A. R. Jamasb, I. Igashov, W. Du et al. Structure-based drug design with equivariant diffusion models. *CoRR*, abs/2210.13695, 2022. doi: 10.48550/arXiv.2210.13695 5

[53] J. Sohl-Dickstein, E. Weiss, N. Maheswaranathan, and S. Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. In *Proc. Int. Conf. Machine Learning (ICML)*, pp. 2256–2265, 2015. 1, 2

[54] J. Song, C. Meng, and S. Ermon. Denoising diffusion implicit models. *CoRR*, abs/2010.02502, 2020. doi: 10.48550/arXiv.2010.02502 9

[55] Y. Song and S. Ermon. Generative modeling by estimating gradients of the data distribution. In *Proc. Conf. Neural Information Processing Systems (NeurIPS)*, pp. 11895–11907, 2019. 1, 2

[56] Y. Song, J. Sohl-Dickstein, D. P. Kingma, A. Kumar, S. Ermon, and B. Poole. Score-based generative modeling through stochastic differential equations. *CoRR*, abs/2011.13456, 2020. doi: 10.48550/arXiv.2011.13456 1, 2

[57] R. Tamassia. Constraints in graph drawing algorithms. *Constraints An Int. J.*, 3(1):87–120, 1998. doi: 10.1023/A:1009760732249 2

[58] M. Tiezzi, G. Ciravegna, and M. Gori. Graph neural networks for graph drawing. *IEEE Transactions on Neural Networks and Learning Systems*, 35(4):4668–4681, 2022. doi: 10.1109/TNNLS.2022.3194008 1, 2, 3, 5, 6, 12

[59] C. Walshaw. A multilevel algorithm for force-directed graph drawing. In *Proc. Symp. Graph Drawing (GD)*, pp. 171–182, 2001. doi: 10.1007/3-540-45848-4_14 2

[60] X. Wang and I. Miyamoto. Generating customized layouts. In *Proc. Symp. Graph Drawing (GD)*, pp. 504–515, 1996. doi: 10.1007/3-540-62495-3_48 2

[61] X. Wang, K. Yen, Y. Hu, and H.-W. Shen. DeepGD: A deep learning framework for graph drawing using gnn. *IEEE Computer Graphics and Applications*, 41(5):32–44, 2021. doi: 10.1109/MCG.2021.3056822 1, 2, 12

[62] X. Wang, K. Yen, Y. Hu, and H.-W. Shen. SmartGD: A gan-based graph drawing framework for diverse aesthetic goals. *arXiv preprint arXiv:2206.06434*, 2022. doi: 10.48550/arXiv.2206.06434 1, 2, 5, 6, 12

[63] 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*, 26(1):676–686, 2019. doi: 10.1109/TVCG.2018.2864411 1, 2, 6, 12

[64] Y. Wang, Y. Wang, Y. Sun, L. Zhu, K. Lu, C.-W. Fu et al. Revisiting stress majorization as a unified framework for interactive constrained graph visualization. *IEEE Transactions on Visualization and Computer Graphics*, 24(1):489–499, 2017. doi: 10.1109/TVCG.2017.2745919 1, 2, 6

[65] K. Xu, W. Hu, J. Leskovec, and S. Jegelka. How powerful are graph neural networks? *CoRR*, abs/1810.00826, 2018. doi: 10.48550/arXiv.1810.00826 1, 4

[66] M. Xu, L. Yu, Y. Song, C. Shi, S. Ermon, and J. Tang. GeoDiff: a geometric diffusion model for molecular conformation generation. *CoRR*, abs/2203.02923, 2022. 1, 2

[67] L. Yang, Z. Zhang, Y. Song, S. Hong, R. Xu, Y. Zhao et al. Diffusion models: A comprehensive survey of methods and applications. *CoRR*, abs/2209.00796, 2022. doi: 10.48550/arXiv.2209.00796 2

[68] Z. Ying, J. You, C. Morris, X. Ren, W. L. Hamilton, and J. Leskovec. Hierarchical graph representation learning with differentiable pooling. In *Proc. Conf. Neural Information Processing Systems (NeurIPS)*, pp. 4805–4815, 2018. doi: 10.48550/arXiv.1806.08804 2

[69] X. Yuan, L. Che, Y. Hu, and X. Zhang. Intelligent graph layout using many users' input. *IEEE Transactions on Visualization and Computer Graphics*, 18(12):2699–2708, 2012. doi: 10.1109/TVCG.2012.236 1, 2

[70] J. Zheng and S. Gigante. Stress-based graph drawing by stochastic gradient descent. `https://github.com/jxz12/s_gd2`, 2017. 5

[71] J. X. Zheng, S. Pawar, and D. F. Goodman. Graph drawing by stochastic gradient descent. *IEEE Transactions on Visualization and Computer Graphics*, 25(9):2738–2748, 2018. doi: 10.1109/TVCG.2018.2852698 2, 5, 12

[72] J. Zhou, G. Cui, S. Hu, Z. Zhang, C. Yang, Z. Liu et al. Graph neural networks: A review of methods and applications. *AI Open*, 1:57–81, 2020. doi: 10.1016/j.aiopen.2020.06.001 2

[73] M. Zhu, W. Chen, Y. Hu, Y. Hou, L. Liu, and K. Zhang. DRGraph: An efficient graph layout algorithm for large-scale graphs by dimensionality reduction. *IEEE Transactions on Visualization and Computer Graphics*, 27(2):1666–1676, 2020. doi: 10.1109/TVCG.2020.3030447 2

A Evaluation Metrics Computation

We used the following metrics to evaluate layout results from multiple perspectives. The computations of these metrics are shown below.

Stress (m_s) [58, 62] quantifies the difference between the geometric distances of the nodes and their corresponding graph-theoretical distances, normalized by the node count. The calculation is as follows:

$$m_s = \frac{1}{N} \sum_{i,j \in V} w_{ij} (||X_i - X_j|| - \delta_{ij})^2, \quad w_{ij} = \delta_{ij}^{-2} \tag{9}$$

where X_i represents the position of node i in the layout and δ_{ij} denotes the graph-theoretical distance between nodes i and j.

Edge Crossing (m_e) [20, 63] quantifies the ratio between the actual number of edge crossings (denoted e) in a graph layout and the maximum possible number of edge crossings (denoted e_{max}). It can be calculated as follows:

$$\begin{aligned} m_e &= \begin{cases} 1 - \frac{e}{e_{max}}, & \text{if } e_{max} > 0 \\ 1, & \text{otherwise} \end{cases} \\ e_{max} &= \frac{|E|(|E|-1)}{2} - \frac{1}{2} \sum_{v \in V} (\deg(v)(\deg(v)-1)) \end{aligned} \tag{10}$$

Edge Angle (m_a) measures the sum of acute crossing angles between edges in the layout The calculation is as follows:

$$m_{\text{xa}} = \sum_{(\vec{u},\vec{v}) \in \text{crossings(G)}} \left| \arcsin < \frac{\vec{u}}{\|\vec{u}\|}, \frac{\vec{v}}{\|\vec{v}\|} > \right| \tag{11}$$

Edge Variation (m_v) [41, 48] is used to measure the variation of edge lengths within a graph layout. It can be calculated as follows:

$$m_v = \frac{1}{\sqrt{|E|-1}} \sqrt{\frac{\sum_{e \in E} (l_e - \bar{l})^2}{|E|\bar{l}|^2}} \tag{12}$$

Neighbors Preservation (m_n) [1, 20] is used to assess whether structurally close nodes also exhibit proximity in the layout by computing the overlap rate of a node's k-hop neighborhood N and its nearest $|N|$ nodes M in the layout.

$$m_n = 1 - \frac{1}{|V|} \sum_{v \in V} \frac{|N_v \bigcap M_v|}{|N_v \bigcup M_v|} \tag{13}$$

Minimum Angle (a_{min}) [1, 48] measures the average deviation of the angle of the adjacent incident edge from the ideal minimum angle. It is calculated as the ratio between the minimum incident of the node and the optimal angle, which is determined by the degree of the node.

$$m_a = 1 - \frac{1}{N} \sum_{i=i}^{N} \frac{|\theta_i - \theta_{i\,min}|}{\theta_i}, \quad \theta_i = \frac{360^\circ}{\deg(v_i)} \tag{14}$$

Root-Mean-Square Error (RMSE) [41] is to quantify the difference between model-generated layouts and ground truth. It can be defined as follows.

$$\text{RMSE}(Y, \hat{Y}) = \sqrt{\frac{1}{n} \sum_i (y_i - \hat{y}_i)^2} \tag{15}$$

Here, y_i and $\hat{y}_i$ represent the aesthetic metric value of a ground truth layout and a model-generated layout. Y and $\hat{Y}$ denote the set of ground truth layouts n and model-generated layouts, respectively. The smaller the RMSE value, the higher the similarity.

Coefficient of Determination (R^2) [41] evaluates the fitness of the model to the observed data. A value closer to 1 indicates a better suitability. It can be calculated as follows, where $\bar{y_i}$ is the mean of y_i.

$$R^2 = 1 - \frac{\sum_i (y_i - \hat{y}_i)^2}{\sum_i (y_i - \bar{y_i})^2} \tag{16}$$

Pearson Correlation Coefficient (P_c) [46] is a statistic used to measure the strength of linear correlation between two groups of variables $[X, Y]$. Its value range is [-1,1], where: P_c = 1, fully positive linear correlation, that is, Y strictly increases with the increase of X; P_c = -1, completely negative linear correlation, that is, Y decreases strictly with the increase of X; P_c = 0, no linear correlation. Its calculation is as follows:

$$\begin{aligned} P_c &= \frac{Cov(X,Y)}{\sigma X \cdot \sigma Y} \\ Cov(X,Y) &= \frac{1}{n} \sum_{i=1}^{n} (X_i - \bar{X})(Y_i - \bar{Y}) \\ \sigma X &= \sqrt{\frac{1}{n} \sum_{i=1}^{n} (X_i - \bar{X})^2}, \quad \sigma Y = \sqrt{\frac{1}{n} \sum_{i=1}^{n} (Y_i - \bar{Y})^2} \end{aligned} \tag{17}$$

B Additional Results

B.1 General Layout Generation

To demonstrate the effectiveness of our model, we compared DiffGraph with several baseline methods, as well as DeepGD and SmartGD. The results are shown in Table 4. We use two metrics: the number of edge crossings (Cross) and Stress as optimization metrics to train SmartGD respectively. SmartGD [62] uses a self-challenging mechanism that allows it to outperform many baseline methods in the metric which the model optimizes. For example, Stress *Stress*(m_s) in SmartGD (Stress) outperforms many other baseline methods, but it does not necessarily outperform the baseline methods in other metrics. Similarly, the *Edge Crossing* (m_e) and *Edge Angle*(m_a) in SmartGD (Cross) outperform other methods, but other metrics are relatively poor. In summary, while DiffGraph may not outperform specific optimization methods on a single metric, it performs exceptionally well on all five metrics.

Table 4: Comparison of DiffGraph with baseline methods.

Method	$m_s \downarrow$	$m_e \downarrow$	$m_a \downarrow$	$m_v \downarrow$	$m_n \downarrow$
FA2 [33]	794.46	48.28	24.49	0.319	0.511
Spring [11]	638.88	57.74	31.18	0.122	0.640
Neato [17]	514.56	57.21	29.11	0.124	0.660
SFDP [31]	578.27	51.24	26.69	0.105	0.618
SGD [71]	487.78	54.28	27.22	0.116	0.647
KK [34]	505.22	55.84	28.10	0.122	0.656
DeepGD(Stress) [61]	469.53	55.19	27.86	0.115	0.645
SmartGD(Cross) [62]	675.73	33.33	16.04	0.289	0.520
SmartGD(Stress) [62]	465.59	56.23	28.37	0.119	0.646
DiffGraph	483.40	38.58	18.45	0.125	0.530

B.2 Failure Cases of Local Constraint Guidance

During the systematic evaluation of a broader range of local constraints, we paid particular attention to extreme constraint scenarios, such as forcibly reducing the geometric distance between topologically distant nodes or artificially enlarging the spatial separation between topologically adjacent nodes. In this work, we term such constraints **structurally inconsistent constraints**, as they contradict the underlying topological structure of the graph and tend to impair layout quality. Experimental results indicate that the introduction of such incompatible constraints often leads to a substantial degradation in overall layout quality, manifested by blurred structural semantics and a noticeable increase in edge crossings, as shown in Figure 9.

To better understand this phenomenon, we identify two primary factors that contribute to the deterioration of the layout performance.

Topology–Geometry Inconsistency. Local constraints may conflict with the intrinsic relationship between graph topology and spatial layout. For example, forcing topologically distant nodes to be spatially close, or separating strongly connected nodes, can violate the structural fidelity that graph layouts are expected to preserve. Such conflicts weaken the correspondence between topology and geometry and may produce layouts that no longer faithfully reflect the underlying graph structure.

Disruption of Local Topological Ordering. Some constraints may also disturb the inherent ordering of local subgraphs. For example, compressing a grid-like structure into a near-linear arrangement, or reversing the order of nodes that follow a circular pattern, can distort local directional or structural information. These cases suggest that local constraints should be specified in a way that respects the original topological relationships within the selected subgraph.

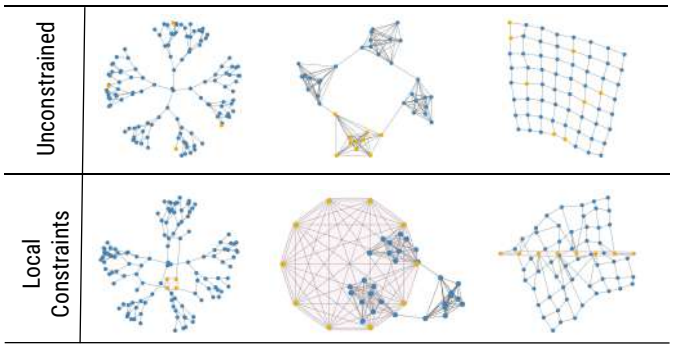

Figure 9: Examples of some extreme local constraint generation examples.

B.3 Controlled Layout Generation with Global Constraints

In the main text, we highlighted the effect of varying the edge variation global constraint on the generated layouts. Here, we present the generation results under several additional sets of global constraints, as shown in Figure 10.

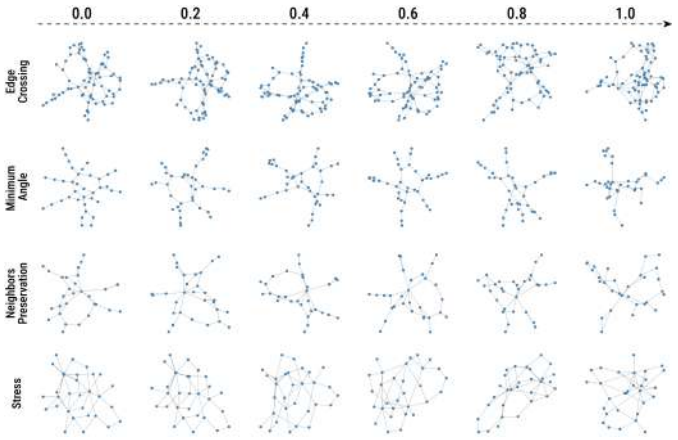

Figure 10: This figure shows the layout results of DiffGraph under the four global constraints *Edge Crossing*, *Minimum Angle*, *Neighbors Preservation*, and *Stress*.

C Sensitivity Analysis

C.1 Inference Time

DiffGraph is implemented based on diffusion models, recognized for remarkable expressiveness. Diffusion models transition data from disorder to equilibrium state by performing denoising steps iteratively.

First, we evaluated the inference time of DiffGraph at different denoising steps (200-5000) based on the Rome dataset. The results are depicted in Figure 11, with a single NVIDIA Titan RTX GPU of 24GB memory. We observed that as the number of denoising steps increases, the inference time of DiffGraph also increases correspondingly. However, within each denoising step, the inference time remains almost unchanged despite the increase in the number of nodes. This indicates that in the Rome dataset with a small number of nodes, the inference time of DiffGraph is primarily determined by the number of denoising steps rather than the size of the graph. Furthermore, we present the graph layout results of each denoising step, as shown in Figure 13 (5000 steps), Figure 14 (3000 steps), Figure 15 (1000 steps), Figure 16 (500 steps), Figure 17 (200 steps). By comparing these results, we found no significant differences in the layout effectiveness among different denoising steps. The model with 200 denoising steps can still generate satisfactory layouts, with the inference time below 1 second.

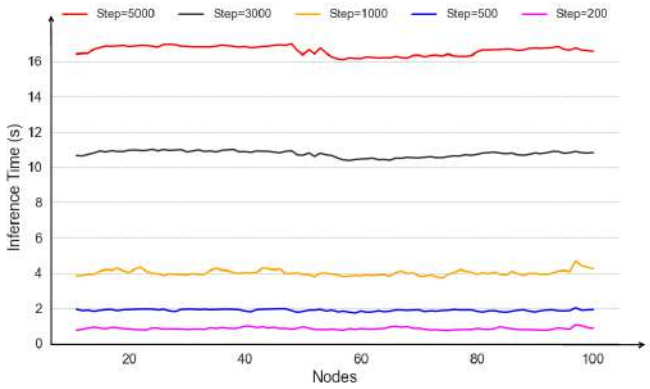

Figure 11: The inference time of DiffGraph in different denoising steps and node count.

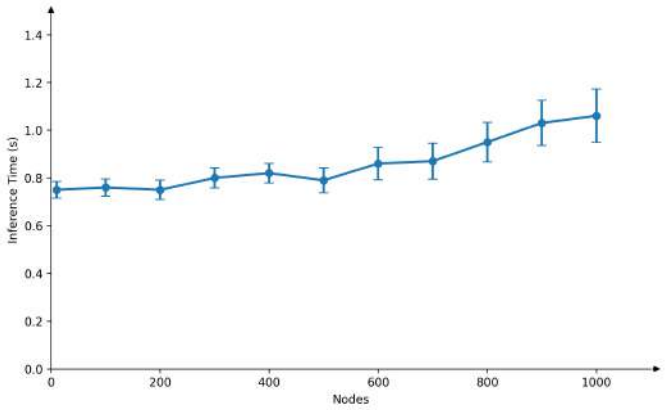

Figure 12: Inference latency of DiffGraph (with 200 denoising steps) as the number of nodes increases.

Additionally, we evaluated the inference time of the model with 200 denoising steps on graphs containing a larger number of nodes. As shown in Figure 12, the increase in inference latency grows relatively slowly with respect to node count. This mild scaling behavior is attributable to GPU memory capacity. When the entire graph fits within the available video memory and can be processed in a single forward pass without batching or memory swapping, computational overhead remains low, leading to sub-linear growth in runtime.

C.2 Stability Evaluation

To systematically evaluate the stability and diversity of the proposed layout generation method, we conducted a seed sensitivity analysis on 100 graphs randomly sampled from the evaluation set. For each graph, a total of 200 layouts were generated: 100 layouts using a fixed random seed to assess reproducibility, and another 100 layouts using different random seeds to examine the diversity of generated solutions. In total, 20,000 layouts were analyzed for each layout mode to quantify the impact of random seed variations on the consistency and stability of the resulting layouts.

In principle, under an identical random seed, the model is expected to produce highly consistent layouts, demonstrating its determinism and

reproducibility. Conversely, when different random seeds are employed, the model should generate structurally diverse yet high-quality layouts, thereby exhibiting its ability to explore multiple plausible solutions within the layout space while maintaining robustness and stable layout quality.

To quantify the structural consistency among layouts produced under different initializations, we introduce the Relative Distance Matrix Difference (RDMD). Given a layout L with node coordinates $\{\mathbf{x}_i\}_{i=1}^{N}$, we first compute its pairwise Euclidean distance matrix

$$D^{(L)} = [d_{ij}]_{N\times N},$$

where

$$d_{ij} = \|\mathbf{x}_i - \mathbf{x}_j\|_2.$$

To eliminate the influence of scale variation, the distance matrix is normalized as

$$\tilde{D}^{(L)} = \frac{D^{(L)}}{\max(D^{(L)})}.$$

The RDMD between two layouts L_a and L_b is then defined as

$$\mathrm{RDMD}(L_a, L_b) = \frac{\left\|\tilde{D}^{(L_a)} - \tilde{D}^{(L_b)}\right\|_F}{\left\|\tilde{D}^{(L_a)}\right\|_F},$$

where $\|\cdot\|_F$ denotes the Frobenius norm.

Since pairwise distances are invariant to translation and rotation, and the normalization removes scaling effects, RDMD measures only differences in the relative spatial organization of nodes. Therefore, it provides a direct assessment of whether different random initializations lead to substantially different layout structures.

For each graph, we compute RDMD for all pairs of generated layouts and report the average value over the entire benchmark. A lower RDMD value indicates that the generated layouts preserve the relative geometric relationships among nodes consistently across multiple runs, reflecting strong reproducibility and stability of the proposed method. In contrast, a higher RDMD value suggests substantial structural differences among generated layouts, indicating a greater degree of layout diversity. This demonstrates the model's capability to produce multiple plausible solutions and facilitates users' exploration of the solution space from different perspectives, thereby supporting interactive analysis and informed layout selection. The final results are shown in Table 5.

Table 5: RDMD for different layout modes.

Mode	Fixed Seed		Different Seeds	
	RDMD-Mean	RDMD-Std	RDMD-Mean	RDMD-Std
General Layout	2.30e-7	7.95e-8	0.150	0.059
Local Constraints	3.11e-7	8.52e-8	0.144	0.048
Global Constraints	5.82e-7	9.45e-8	0.158	0.062

In addition, we employ the Coefficient of Variation (CV) to quantify the variability of generated layouts under different random seeds with respect to a given layout metric. It is defined as follows:

$$CV = \frac{\sigma}{\mu} \times 100\%$$

where σ and μ denote the standard deviation and mean value of the metric across multiple generated layouts, respectively. As a dimensionless measure of relative dispersion, the coefficient of variation effectively eliminates the influence of measurement scales and mean magnitudes, enabling a more objective comparison of performance fluctuations across different metrics. A lower CV value indicates that layouts generated with different random seeds exhibit greater consistency and stability with respect to the evaluated metric.

Under the fixed-seed setting, the resulting RDMD scores consistently remain close to zero, indicating that the proposed model produces highly consistent layouts when initialized with the same random seed.

Table 6: Coefficient of variation in metrics for different seed layouts.

Mode	m_s	m_e	m_a	m_v	m_n
General Layout	5.3%	4.8%	3.9%	5.5%	3.5%
Local Constraints	5.6%	5.0%	3.7%	5.2%	3.7%
Global Constraints	6.3%	5.3%	4.5%	5.9%	4.3%

This demonstrates strong stability and reproducibility of the generation process. In contrast, under different random seed settings, the RDMD scores increase substantially, indicating that the generated layouts exhibit significant geometric diversity. Meanwhile, the relatively low CV values suggest that, despite noticeable variations in layout configurations, the generated results remain highly consistent with respect to key evaluation metrics. This observation demonstrates that the proposed method is capable of producing diverse layout solutions while preserving stable layout quality and structural validity. Taken together, these results demonstrate that the proposed method effectively balances generation stability and solution-space diversity, achieving both reliable reproducibility and rich exploratory capability in graph layout generation.

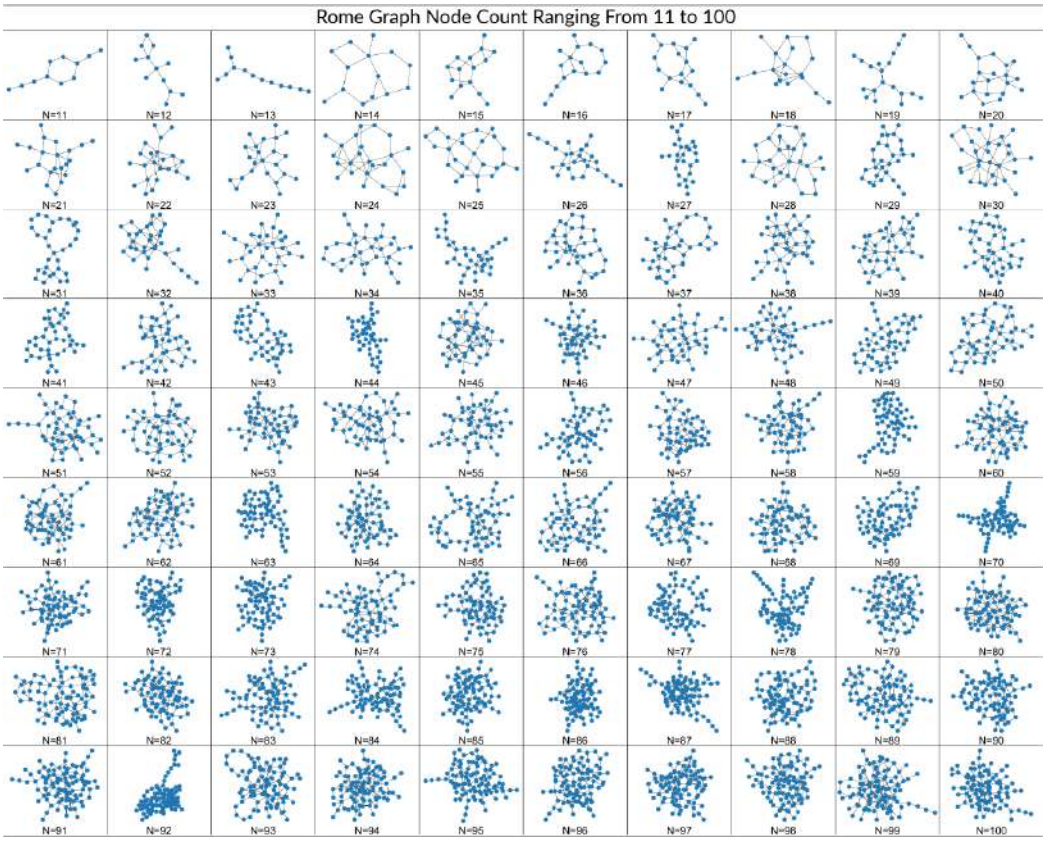

Figure 13: The Rome graph layouts generated by DiffGraph in 5000 denoising steps.

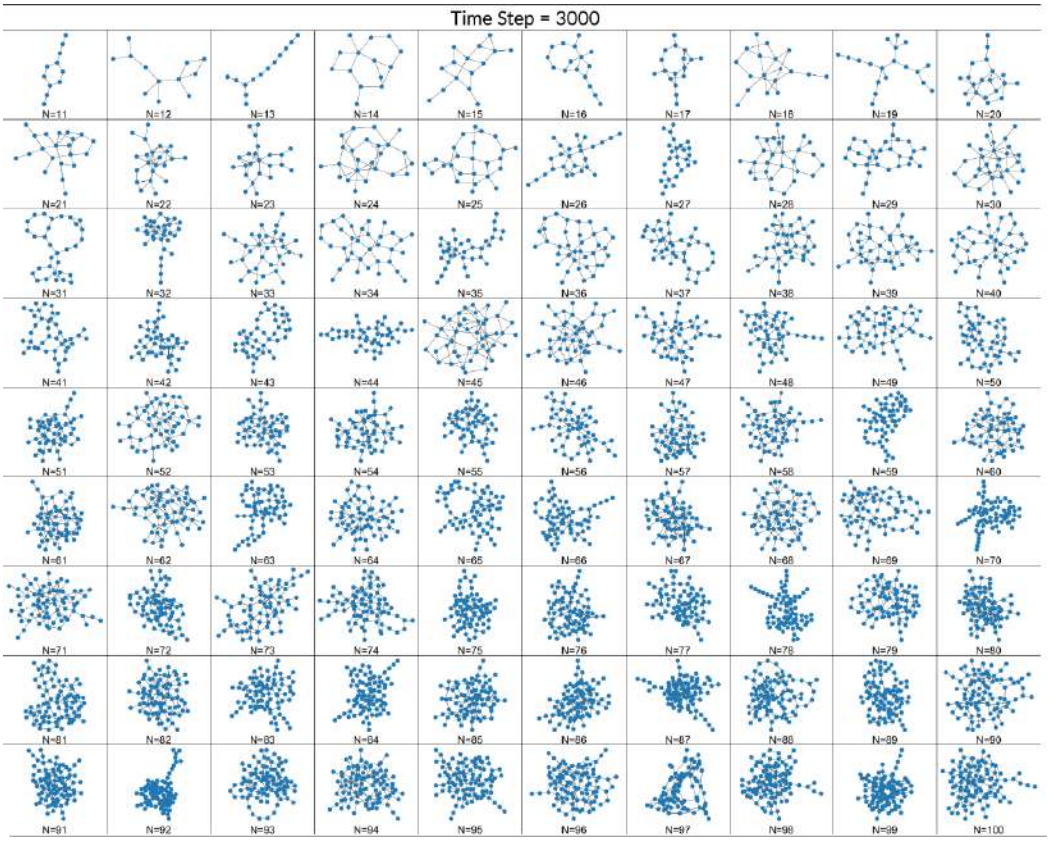

Figure 14: The Rome graph layouts generated by DiffGraph in 3000 denoising steps.

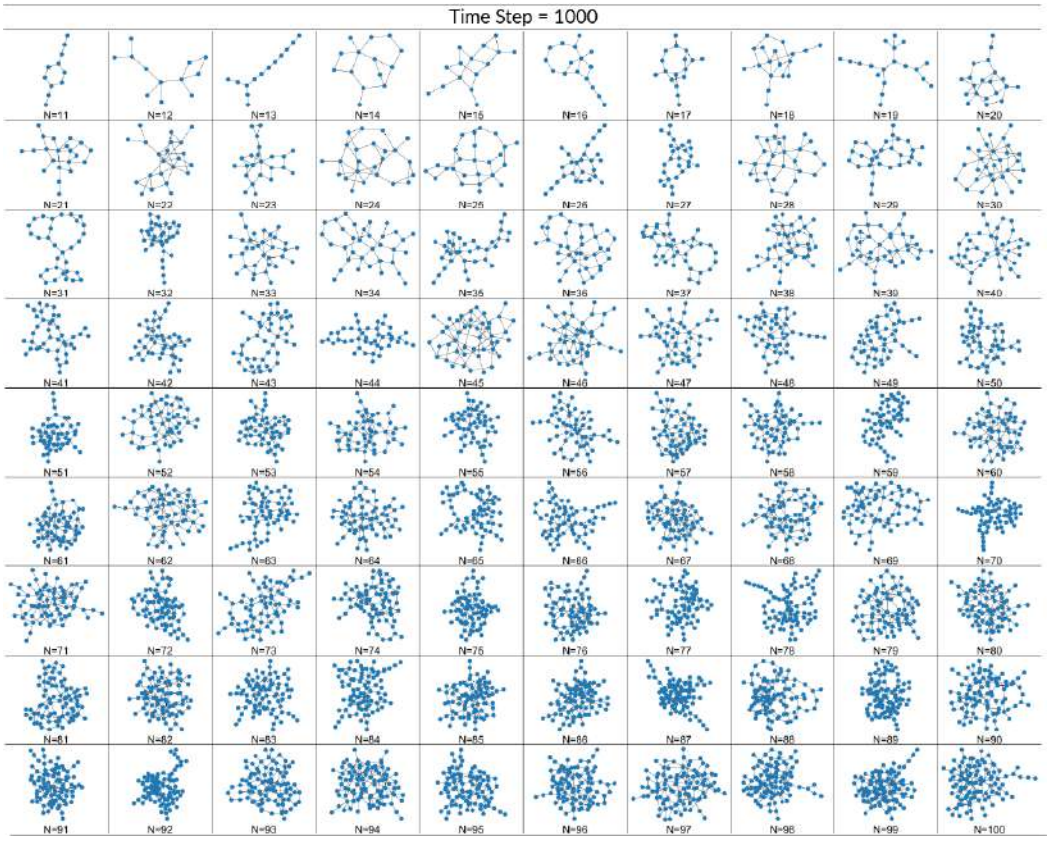

Figure 15: The Rome graph layouts generated by DiffGraph in 1000 denoising steps.

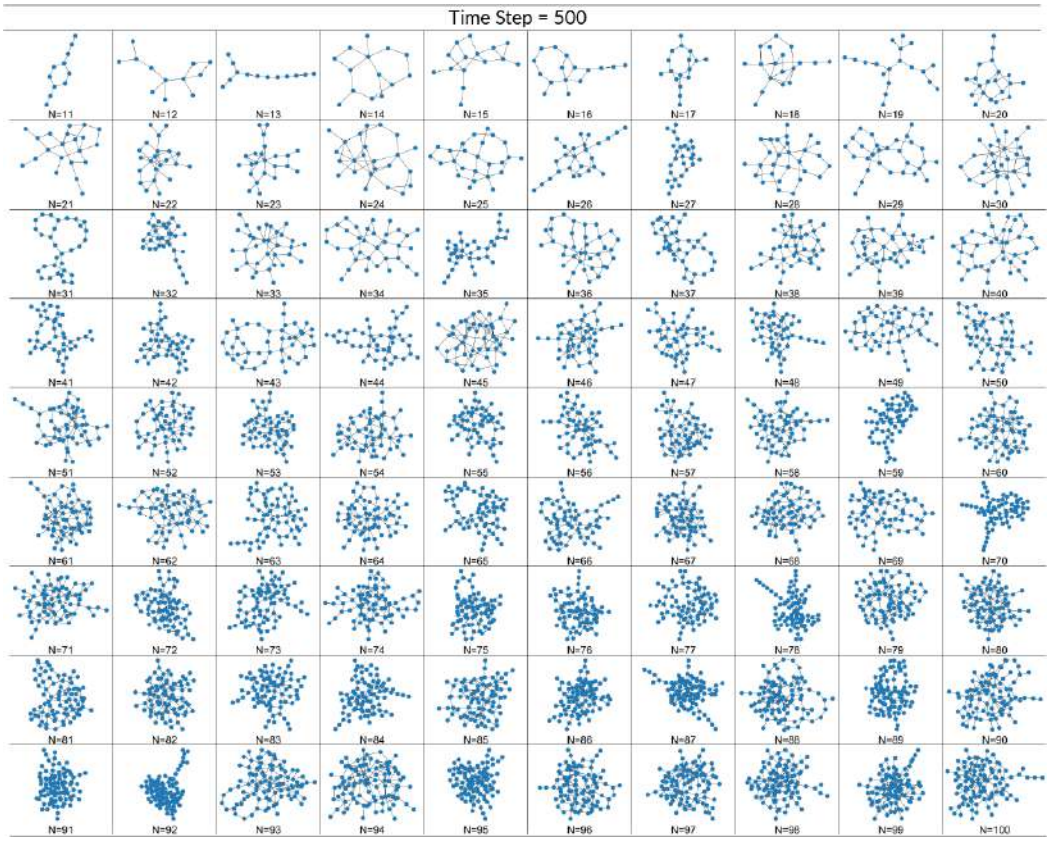

Figure 16: The Rome graph layouts generated by DiffGraph in 500 denoising steps.

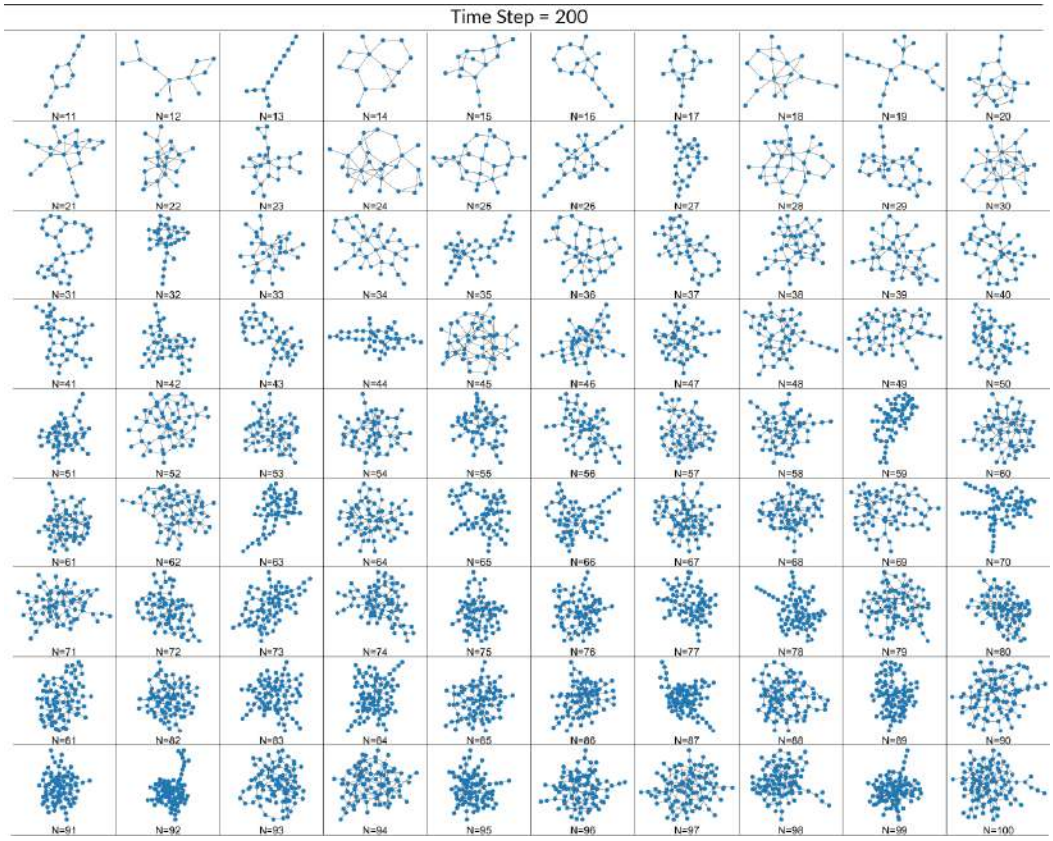

Figure 17: The Rome graph layouts generated by DiffGraph in 200 denoising steps.