AutoFDP: Automatic Force-based Model Selection for Multicriteria Graph Drawing

Mingliang Xue, Yifan Wang, Zhi Wang, Lifeng Zhu, Lizhen Cui,
Yueguo Chen, Zhiyu Ding, Oliver Deussen, Yunhai Wang

***Abstract*—Traditional force-based graph layout models are rooted in virtual physics, while criteria-driven techniques position nodes by directly optimizing graph readability criteria. In this paper, we systematically explore the integration of these two approaches, introducing criteria-driven force-based graph layout techniques. We propose a general framework that, based on user-specified readability criteria, such as minimizing edge crossings, automatically constructs a force-based model tailored to generate layouts for a given graph. Models derived from highly similar graphs can be reused to create initial layouts, users can further refine layouts by imposing different criteria on subgraphs. We perform quantitative comparisons between our layout methods and existing techniques across various graphs and present a case study on graph exploration. Our results indicate that our framework generates superior layouts compared to existing techniques and exhibits better generalization capabilities than deep learning-based methods.**

***Index Terms*—Graph Layout, Readability Criteria, Optimization**

I. Introduction

Graphs serve as fundamental data structures across diverse domains for representing relationships among entities. A common approach to visualizing graphs is the node-link diagram, in which nodes are depicted as points in a two-dimensional space and edges as connecting lines. A wide range of layout algorithms has been developed to generate graph drawings that are both visually coherent and structurally informative.

Most widely used graph layout algorithms are based on physical analogies, particularly force-directed models. In these models, edges—or conceptual links between nodes—are modeled as springs generating attractive forces, while all node pairs exert mutual repulsive forces, akin to electrically charged particles. A layout is considered optimal when these forces balance out in a state of equilibrium. The diversity of force formulations has given rise to numerous variants, each designed to emphasize different structural features of graphs. For example, SFDP [1] is effective at revealing clusters, ForceAtlas2 [2] accurately depicts local neighborhood structures, and the Stress Model [3]–[5] preserves the overall structure of a graph. While these methods highlight various aspects of a graph , finding the appropriate model for a specific graph can be quite challenging, even formulating the criteria for its layout in the form of a unified force representation, such as presented in [6].

Recent advancements in optimization and deep learning techniques have triggered criteria-driven graph layout techniques that compute node positions by directly optimizing graph readability criteria [7] such as edge crossing, neighborhood preservation, and node resolution. For example, SGD^2 proposed by Ahmed et al. [8], [9] directly optimizes a large set of criteria provided that each of them can be expressed as a differentiable function. They provide surrogate functions for non-differentiable criteria, such as number of crossings and crossing angles. Although experimental results show that SGD^2 generates better or comparable results to existing force-based methods on small graphs, SGD^2 suffers two major issues: i) being inefficient for optimizing real-world graphs and ii) parameters found by the optimization cannot be reused for other graphs. Similar to SGD^2, which uses these criteria as the loss function, deep learning-based methods such as DeepGD [10] and SmartGD [11] train neural network models by using a set of graphs with various structural characteristics. The obtained models can then be used to generate layouts for other graphs. The evaluation of SmartGD indicates that deep learning-based methods perform comparably or even better than SGD^2 when applied to graphs with similar structures, while also being significantly faster. However, most of the tested graphs are limited to small synthetic graphs with less than hundreds of nodes.

This paper systematically investigates the potential of extending force-based models to directly optimize layout readability, aiming to produce graph drawings that outperform those generated by conventional methods. To this end, we introduce AutoFDP, a unified framework that combines physics-inspired force-based modeling with readability-driven layout optimization. The framework is designed to automatically select the most suitable force-directed model for a given graph, guided by a broad set of readability criteria, including the reduction of edge crossings and the enhancement of crossing angles. Our framework is designed to satisfy the following requirements: (i) constructing a layout model in an unsupervised manner without the training process; (ii) optimizing a model towards

M. Xue and L. Cui are with the School of Software & Joint SDU-NTU Center for Artificial Intelligence Research (C-FAIR), Shandong University, CN. E-mail:{mingliangxue,clz}@sdu.edu.cn

Yifan Wang and Z. Wang are with the Department of Computer Science, Shandong University, CN. M. Xue and Yifan Wang are joint first author. E-mail: {yfwang2001, wangzizi2020}@gmail.com

L. Zhu is with the Department of Instrument Science and Engineering, Southeast University, CN. E-mail: lfzhulf@gmail.com

Z. Ding is an independent researcher. E-mail: achilles.ding.zju@gmail.com

O. Deussen is with Computer and Information Science, University of Konstanz, DE. E-mail: oliver.deussen@uni-konstanz.de

Y. Chen and Yunhai Wang are with the School of Information, Renmin University of China, CN, E-mail:{chenyueguo, wang.yh}@ruc.edu.cn

Yunhai Wang is the corresponding author.

TABLE I: Design objectives of AutoFDP in comparison to prior criteria-driven graph drawing techniques.

Technique	Force-based	W/O Training	Non-differentiable	Reusable
SGD^2	×	✓	×	×
DeepGD	×	×	×	✓
SmartGD	×	×	✓	✓
AutoFDP	✓	✓	✓	✓

all quantitative criteria, even if they are non-differentiable; and (iii) supporting the reuse of a model for other graphs while ensuring high-quality visualization. As far as we know, these requirements have not been adopted in any existing layout technique for exploring node-link diagrams, though some are met in SGD^2 [9] and deep learning-based techniques by Wang et al. [10], [11].

Built on the quotient-based force representation of Xue et al. [6] that unifies almost all existing physics-inspired layout models, our approach automatically searches proper force-based models for all combined force functions that best meet given criteria with a bilevel optimization problem [12]. Our search space covers all force functions parameterized by six continuous variables ($\theta = \{\omega_a, \alpha_a, \beta_a, \omega_r, \alpha_r, \beta_r\}$, see section IV). Since our optimization is based on a subset of force parameters, it is faster and more robust than SGD^2 [9]. In addition, the force-based models obtained from one graph can be reused for other graphs that are other graphs that have a kernel-based similarity greater than or equal to 0.5, and the generated layout can be refined by interactively changing the model's parameters or optimizing subgraphs of interest with different criteria.

In doing so, interactive example-based layout generation of complex graphs becomes feasible. We refer to this scheme as a reuse-and-optimization scheme. Table I compares the design objectives of AutoFDP and existing techniques along three considerations. SGD^2 requires full optimization of each graph, struggles with large-scale or complex criteria, and can only handle simple graphs. DeepGD and SmartGD rely heavily on the similarity of the training data and are only suitable for cases where suitable training data is available. In contrast, AutoFDP is superior in terms of computational efficiency, physical interpretability, and flexibility. Note that our framework is not limited to the quotient-based force representation; it can also be integrated with other physics-inspired models, such as t-force [13].

To illustrate its efficacy, we begin by comparing our approach with both SGD^2 and prevalent force-based layout techniques, adhering to the same parameters established by SGD^2. Our findings indicate that AutoFDP is the only one to consistently achieve reasonable layouts across all types of graphs, while also being significantly faster—up to ten times—than SGD^2. Next, we assess our method against DeepGD and SmartGD, the deep learning-based approaches with publicly available code. Here, our method exhibits comparable performance for graphs that resemble the training dataset in structure, yet it excels with other types of graphs. Finally, through a case study, we demonstrate how our framework facilitates the example-based layout generation of intricate graphs, underscoring its practical utility.

II. Related Work

Related work falls into two categories: graph layout methods and example-based layout generation.

A. Graph Layout Methods

A large number of graph layout methods have been proposed for visualizing graphs as node-link diagrams. Force-directed layout methods generate graph layouts by simulating one of two virtual physical models: the spring-electrical model and the stress model. The spring-electrical model [2], [14] treats nodes as electrically-charged particles that exert repulsive forces between all nodes and the edges of the graph as springs, which exert attractive forces to connected nodes. Using this model, the classical force-directed placement (FDP) algorithm [2] defines attractive forces as proportional to the squared distance between two nodes and repulsive forces reciprocal to their distance. The resultant force is used to move nodes until convergence is reached. To reveal different graph structures, different force variants [15] have been proposed. For example, Hu [1] introduces a repulsive force weaker at long ranges to distribute nodes more evenly. Noack [16] suggests using a constant attractive force and a repulsive force proportional to the inverse of the distance for clearly separating node clusters. To balance local structures and clusters, Jacomy et al. [17] redefine attractive and repulsive forces proportional and inversely proportional to the distance between nodes. However, these methods do not directly optimize the distance between nodes as primary objectives; their preservation emerges indirectly through force equilibrium.

In contrast, stress models [3], [4] assume that every pair of nodes in the graph is connected by springs whose lengths are equal to the shortest path distances in data space. By minimizing the energy of this spring system, the resulting layout better reveals the global structure of a graph than spring-electric models. To reduce its computation cost, some variants evaluate the model by using a selected subset of springs. For example, the Sparse Stress Model [18] aggregates long-range springs using a set of pivot nodes. The Maxent Stress Model [19] restricts the definition of springs to the k-ring neighborhood of a node but uses repulsive forces between all node pairs for uniformly distributing them. It can be seen as a hybrid model, where the repulsive force is represented as an entropy term added to the stress model.

The commonly used stress model can be regarded as a minimization of the stress error, which allows the capture of the global graph structure. However, optimizing for criteria like minimizing the number of edge crossings [7] or maximizing crossing angles [20] is challenging due to their discrete and non-convex nature. These properties make it difficult to apply exact optimization methods efficiently. Therefore, different heuristics [21], [22] have been proposed to achieve them. Bekos et al. [21] introduce a heuristic algorithm for maximizing the minimal crossing angle, while Radermacher et al. [22] present three geometric heuristics for minimizing the number of edge crossings. However, these heuristic methods cannot be used for optimizing other criteria. Recently, Ahmed et al. [8], [9] proposed a general method, SGD^2, for directly optimizing

multiple criteria in a unified manner when each criterion can be expressed as a differentiable function. Yet, its optimization directly moves every individual node and hence its optimization result cannot be explained in terms of force-based models, let alone manipulated by changing force parameters. In contrast, our method automatically finds a specific force-based model for a given graph, where the model can be refined and reused.

Taking readability criteria as the loss function, DeepGD [10] trains a convolutional graph neural network to generate the layout for arbitrary graphs once trained. However, it is designed to optimize differentiable criteria as SGD^2 and requires manually defined surrogate functions for non-differentiable criteria. SmartGD addresses this issue by using Generative Adversarial Networks [23] but requires a collection of pre-constructed high-quality layouts. In contrast, our method enables searches of a customized force-based model for the input graph to meet the given criteria without the training procedure.

B. Example-based Layout Generation

For large graphs, computing a layout and evaluating its aesthetic metrics can be time-consuming, often limiting the feasibility of interactive exploration. To address this challenge, Kwon et al. [24] propose a method that retrieves precomputed layouts for graphs with topologically similar structures, enabling quick visualization of the expected layout and significantly reducing the need for expensive real-time computations. Similarly, Pan et al. [25] introduce an interactive exemplar-based layout fine-tuning technique, where users modify the layout of a local subgraph, with these adjustments automatically transferred to topologically similar subgraphs and integrated into the entire graph.

Building on the above approaches, we introduce a model reuse strategy that leverages precomputed force models from topologically similar graphs to efficiently generate reasonable layouts for a given graph. Once the layout is generated, users can refine the layout of subgraphs to meet different aesthetic criteria by re-running the optimization. In contrast to Fisheye views [26], [27], which often cause significant distortion to the global shape, our reuse-and-optimization exploration scheme provides detailed views of subgraphs while preserving the overall context without distortion.

III. Background

Given a graph $G(V,E)$ with nodes V and edges E, computing a graph layout means to find the positions $\mathbf{X} = \{\mathbf{x}_1, \cdots, \mathbf{x}_{|V|}\}$ of all nodes in 2D space with $\mathbf{x}_i \in R^2$. Using these notations, we first briefly describe the quotient-based force representation which unifies most existing virtual physics-based graph layout methods, and then introduce common criteria [7], which we use to drive the automated selection of layouts.

A. Quotient-based Force Representation

Xue et al. [6] show that most existing graph layout methods can be formulated as a combination of *quotient-based forces* that combine power functions of graph-theoretic distances d_{ij} between two nodes i and j as well as Euclidean distances (in 2D space). Suppose that each node i is subject to a total of l forces, each force is denoted by $F_{i,k}$ and the resultant force by F_i:

$$F_{i,k} = \sum_{(i,j)\in\Omega_k} \omega_k * \frac{||\mathbf{x}_i - \mathbf{x}_j||^{\alpha_k}}{d_{ij}^{\beta_k}} * \mathbf{e}_{k,ij}, \quad F_i = \sum_{k}^{l} F_{i,k} \qquad (1)$$

where Ω_k is the *force range* specifying the nodes that exert forces towards node i, ω_k is a weight with a sign deciding the *force type* (attractive force vs. repulsive force), $\mathbf{e}_{k,ij}$ is a normalized vector of the force direction, α_k and β_k are exponents for graph-theoretic and Euclidean distances between nodes j and i, determining the *force magnitude*. For $\alpha = -1$, the exponential function is replaced by $ln(||\mathbf{x}_i - \mathbf{x}_j||)$,

To ensure the convergence of the layout algorithm, there should be at least one attractive force ($\omega_k > 0$) and one repulsive force ($\omega_k < 0$) and hence $l \geq 2$. The force range Ω_k can be E, V^2, or a user-specified node set P. The exponents $\{\alpha_k, \beta_k\}$ can take many different values, each of which might capture different graph characteristics. For choosing proper $\{\alpha_k, \beta_k\}$, Xue et al. [6] further identify and validate two guidelines:

- **G1:** $\alpha_k \geq 0, \beta_k \geq 0$ if $\omega_k > 0$; and
- **G2:** $\alpha_k \leq 0, \beta_k \leq 0$ if $\omega_k < 0$.

Following these guidelines, users can find $\{\alpha_k, \beta_k\}$ to meet given requirements. However, it is still a time-consuming and trial-and-error process. By using this representation as a proxy for finding the desired layout, our proposed optimization framework can automatically search proper values $\{\alpha_k, \beta_k\}$ that reveal different graph structures (e.g., clusters, tree, and grid) while meeting given criteria (e.g., stress error, crossing number and etc.). Compared to SGD^2, searching this compact parameter space is more efficient while yielding similar or even better layouts, see Subsection V-A.

B. Graph Readability Criteria

Here, we briefly describe nine readability criteria [7] to measure the layout quality and refer to the supplemental material for complete definitions.

- Normalized stress error (SE) [19] measures the squared difference of the shortest path distance between node pairs and their Euclidean distance;
- Ideal Edge Length (IL) computes the variance of the absolute difference between the length of an edge and its ideal length (1.0);
- Neighborhood Preservation (NP) measures the similarity between the k-nearest neighborhoods in the input graph and those defined within the layout;
- Crosslessness (CL) quantifies the amount of edge crossings, normalized by the maximal possible number of edge crossings;
- Crossing Angle (CA) measures the average absolute discrepancy between each crossing angle and the target crossing angle of $\pi/2$;
- Minimum Angle (MA) is defined as the average deviation between the minimum angle $\theta_{min}(i)$ between edges incident to a vertex i and the ideal minimum angle of $2\pi/deg(i)$ for all nodes;

TABLE II: Symbol Description Table

symbol	description
$\mathbf{X}$	Layout for input graph
$\psi(\mathbf{X})$	Cost of layout for given criteria
L_i	Cost for each criterion
$W=\{w_1,\cdots,w_m\}$	Weights for criteria
$\theta=\{\omega_a,\alpha_a,\beta_a,\omega_r,\alpha_r,\beta_r\}$	Parameters for describing the force model
$\mathbf{X}_\theta$	Layout obtained by force model present by θ
$\phi(\mathbf{X},\theta)$	Energy of the force model present by θ for layout $\mathbf{X}$
$\mathbf{x}_i$	Layout coordinates of node i
d_{ij}	graphical distance between node i and j
θ^*	Parameters corresponding to the force model that minimizes the layout cost
λ and ρ	step sizes
$\Delta\theta$	Parameter difference
$\delta\theta$	Parameter perturbations
$\hat{\mathbf{X}}$	Layout perturbations

- Node Resolution (NR) assesses the amount of node overlap defined as the minimum distance between two nodes in a layout [28];
- Aspect Ratio (AR) indicates to what extent the bounding box of a layout approaches a square; and
- Gabriel Graph Property (GP) measures the extent of a graph to be a Gabriel graph [29] by calculating the maximum proportion of edges without nodes in their circles.

For the criterion whose values L are not the smaller the better, we change them to $1-L$ to ensure that the scores consistently show smaller values as being better. In doing so, we take them as minimization objectives.

By default, AutoFDP employs nine criteria and their combinations as optimization objectives. Our method can directly handle non-differentiable criteria such as CA, NP, and CL, eliminating the need for differentiable surrogate functions. As a result, the definitions of these criteria used for optimization objectives differ from those used in SGD^2. Additionally, the definitions of MA and NR used by DeepGD as objectives are also distinct from ours. To facilitate a fair comparison with DeepGD, we implemented their versions of these criteria as optimization objectives in our method.

IV. Our Approach

Given an input graph and a set of weighted criteria, our framework aims to find a force-based layout model. This model dynamically adjusts node positions by optimizing the weighted combination of criteria, thereby generating layouts that emphasize the graph characteristics prioritized by the criteria. Once a suitable model has been identified, it can be used as a proxy for the specified criteria to determine the optimal layout for other similar graphs. In addition, users can interactively refine the layouts of the subgraphs of interest by performing local optimization. In this section, we first describe the optimization scheme for searching model parameters, then elaborate on example-based layout generation. To facilitate reading, we put the symbol description in Table II.

A. Automated Model Search

To circumvent the computational complexity of directly optimizing the given criteria in layout, we introduce a force model where minimizing its system energy efficiently approximates the minimum criteria layout. We assume that there are l quotient-based forces in the model, each having four parameters: force range Ω_k, force weight ω_k, and force exponents (α_k,β_k). For simplicity, we use only two forces ($l=2$) by default: one attractive force ($\omega_a>0$), and one repulsive force ($\omega_r<0$), both forces are exerted on nodes in the range $\Omega_k=V^2$. While other forces (e.g., label collision) can be better supported by other choices of Ω_k and l, such adaptations incur significant efficiency costs, which motivates us to make this restriction. Each model is therefore described by six parameters: $\theta=\{\omega_a,\alpha_a,\beta_a,\omega_r,\alpha_r,\beta_r\}$. We obtain these values by optimizing the given criteria for a given input graph.

Given m weighted criteria, the cost of a layout $\mathbf{X}$ is measured by:

$$\psi(\mathbf{X})=\sum_i^m w_i\mathrm{L}_i(\mathbf{X}), \tag{2}$$

where $\mathrm{L}_i(\mathbf{X})$ is one of the criteria defined by the user and $W=\{w_1,\cdots,w_m\}$ are the weights for all criteria. Our method approximates $\psi(\mathbf{X})$ by optimizing the energy equation for the layout model. Once θ is known, the layout $\mathbf{X}$ can be obtained by finding the state of equilibrium induced by the force model, which corresponds to minimizing the system energy:

$$\mathbf{X}_\theta=\arg\min_{\mathbf{X}}\phi(\mathbf{X},\theta), \tag{3}$$

where $\phi(\mathbf{X},\theta)$ is the energy term of the Taurus model [6], which is the primitive function of the Taurus force function in Equation 1 under the previous assumptions:

$$\phi(\mathbf{X},\theta)=\sum_{i,j\in V}\frac{\omega_a||\mathbf{x}_i-\mathbf{x}_j||^{\alpha_a+1}}{(\alpha_a+1)d_{ij}^{\beta_a}}+\frac{\omega_r||\mathbf{x}_i-\mathbf{x}_j||^{\alpha_r+1}}{(\alpha_r+1)d_{ij}^{\beta_r}}. \tag{4}$$

A desired θ results in a layout $\mathbf{X}$ with a minimal value of cost $\psi(\mathbf{X})$. Therefore, searching θ can be formulated as a unified optimization problem that combines Equations 2 and 4:

$$\theta^*=\arg\min_\theta\psi(\mathbf{X}_\theta). \tag{5}$$

This can be interpreted as seeking a force-based model with proper parameters θ that produces an optimal layout $\mathbf{X}$ in terms of the cost ψ. We tackle this problem by iteratively alternating between updating the layout $\mathbf{X}$ and updating θ to improve layout quality. After initializing θ, we use it to compute $\mathbf{X}^{i+1}$ and then use $\mathbf{X}^{i+1}$ to update θ and repeat this process until reaching convergence. Accordingly, $\mathbf{X}$ and θ are updated by:

$$\mathbf{X}^{i+1}=\mathbf{X}^i-\lambda\frac{\partial\phi(\mathbf{X}^\mathbf{i},\theta^i)}{\partial\mathbf{X}^i}, \tag{6}$$

$$\theta^{i+1}=\theta^i-\rho\frac{\partial\psi(\mathbf{X}^{i+1})}{\partial\theta^i}, \tag{7}$$

where λ and ρ are step sizes. Note that we use the layout results from one iteration rather than the converged layout results as the $\mathbf{X}$ for the bilevel optimization in each iteration. Although we do not achieve the optimal $\mathbf{X}_{\theta^i}$ within each iteration, our iterative optimization ensures an optimal θ when the layout cost $\psi(\mathbf{X})$ converges.

By differentiating ϕ with respect to $\mathbf{X}$, we obtain

$$\frac{\partial \phi(\mathbf{X},\theta)}{\partial \mathbf{X}} = \frac{\omega_a}{2} * \frac{||\mathbf{x}_i - \mathbf{x}_j||^{\alpha_a}}{d_{ij}^{\beta_a}} * \mathbf{e}_{a,ij} + \frac{\omega_r}{2} * \frac{||\mathbf{x}_i - \mathbf{x}_j||^{\alpha_r}}{d_{ij}^{\beta_r}} * \mathbf{e}_{r,ij}, \tag{8}$$

where $\mathbf{e}_{a,ij}$ and $\mathbf{e}_{r,ij}$ are the normalized attractive and repulsive force directions, respectively. We update the position of each node $\mathbf{x}_i$ by using an augmented stochastic gradient descent solver (ASGD) [6] with fast convergence. Following Zheng et al. [30], we exponentially decrease λ from 100 to 0.01, and ρ follows a similar exponential decrement from 0.2 to 0.05.

Yet, some readability criteria do not have analytical partial derivatives and hence we use numerical differentiation to solve the partial derivative of ψ with respect to θ:

$$\frac{\partial \psi(\mathbf{X}^{i+1})}{\partial \theta^i} = \frac{\partial \psi(\mathbf{X}_{\theta^{i+1}})}{\partial \theta^i} \approx \frac{\psi(\mathbf{X}_{\theta^i+\delta\theta}) - \psi(\mathbf{X}_{\theta^i})}{\delta\theta}, \tag{9}$$

where

$$\mathbf{X}_{\theta^i} = \mathbf{X}^i \tag{10}$$

$$\mathbf{X}_{\theta^i+\delta\theta} \approx \hat{\mathbf{X}} = \mathbf{X}^i - \lambda \frac{\partial \phi(\mathbf{X}^i, \theta^i + \delta\theta)}{\partial \mathbf{X}^i}. \tag{11}$$

Layout $\hat{\mathbf{X}}$ is now generated by adding a small perturbation to θ. Considering that θ only contains six parameters, the computational cost of the numerical differentiation is still acceptable. Some criteria like SE have analytical partial derivatives, but we found that numerical derivatives do not show significant differences to analytic ones in our experiments. To reduce the likelihood of being trapped in local optima, we adjust $\delta\theta$ using cooling schedules stemming from simulated annealing [31]. Once the partial derivatives are obtained, θ^{i+1} is updated according to Eq. 7.

Algorithm 1 Pseudocode for force-based model selection

1: Input: graph $G = (V, E)$, cost ψ
2: $\mathbf{X}^0$ = *RandomMatrix*($|V|$, 2)
3: θ^0 = *RandomVector*($|\theta|$), $iter = 0$
4: **repeat**
5: Update $\mathbf{X}^{i+1}$ with Eq. 6
6: Calculate θ^{i+1} with Eq. 7
7: $iter = iter + 1$
8: **until** $|\psi(\mathbf{X}^{i+1}) - \psi(\mathbf{X}^i)|/\psi(\mathbf{X}^i) < \varepsilon$ or $iter > maxiter$
9: **return** θ^i

Algorithm 1 outlines the alternating optimization: after initializing θ according to the guidelines (see Section III), it iteratively updates $\mathbf{X}$ and θ until the convergence criterion ($\varepsilon = 10^{-7}$ in line 8) is satisfied. By choosing an appropriate decay function for $\Delta\theta$, our algorithm converges all the time during our experiments. Applying two different readability criteria to the graphs *dwt1005* and *bus685*, Figure 1a and Figure 1b show the convergence curves generated by using three decay functions over 30 iterations. The exponential decay function performs best for both graphs and gradually converges toward a lower objective value. Figure 1c and Figure 1e show the generated intermediate layouts of the graphs at the 1st, 15th, and 30th iterations by minimizing SE and NP, respectively.

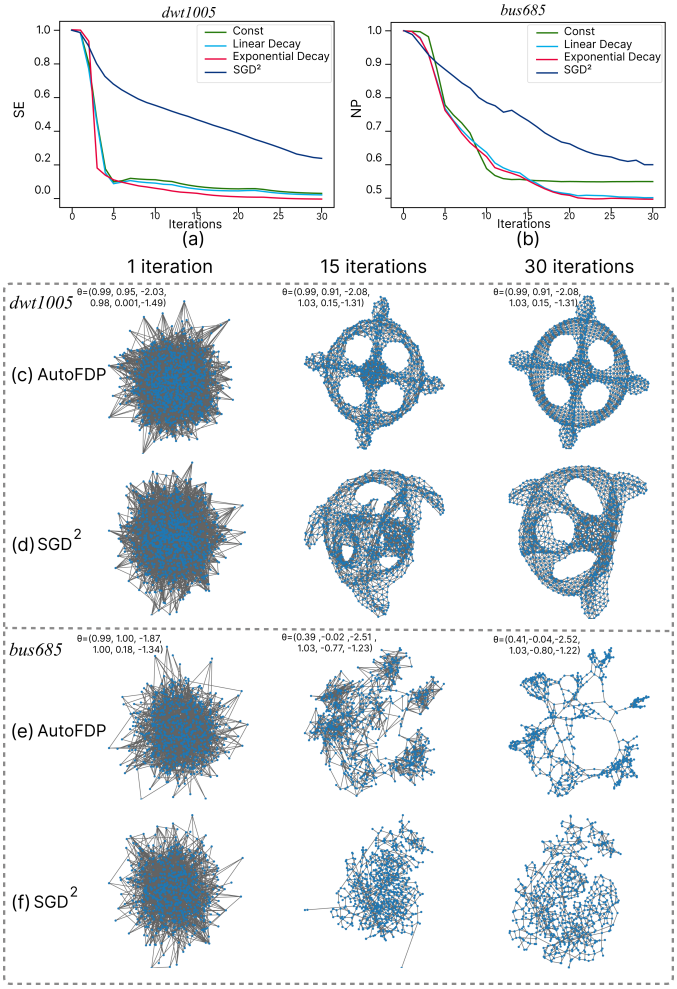


Fig. 1: Convergence curves of AutoFDP with different decay functions and SGD^2 on two graphs (a,b) and intermediate results of the 1st, 15th, and 30th iteration in optimizing SE on the graph *dwt1005* and NP on the graph *bus685* by AutoFDP (c,e) and SGD^2 (d,f).

To verify that this optimization selects reasonable force-based models, we carefully examined the obtained θ in Figure 1c and Figure 1e . The ideal result of minimizing SE Figure 1c is a stress model [4], where $|\alpha_a - \alpha_r|$ are close to $|\beta_a - \beta_r|$ for balancing graph-theoretic distance and Euclidean distance. Looking at θ shown on the top left of the sub-figures, we see that θ at the 15th iteration is reasonable. Although minimizing NP in Figure 1e does not correspond to any existing force-based model, the values of β_a should be gradually increased to reduce the attractive forces between nodes, while for β_r, values should be gradually decreased to enhance the repulsive forces during intermediate iterations. The values of β_a and β_r in Figure 1e fit that requirement. In contrast, SGD^2 does not quickly reach the convergence as AutoFDP (see the curves in Figure 1a and Figure 1b) and the resulting intermediate layouts shown in Figure 1d and Figure 1f cannot reveal the grid and tree-like structures in the two graphs with the same number of iterations.

In summary, our optimization approach efficiently converges to a reasonable layout solution with fewer iterations than conventional methods for the given weighted criteria. More

results are shown in Subsection V-A.

B. Interactive Example-based Layout Generation

Given a graph with around 5K nodes, finding the proper θ with algorithm 1 takes around 2 mins in our experiment, which is too slow for layout generation. On the other hand, even if θ is rapidly obtained, it is a challenge to manually change θ to manipulate the layout to meet different readability criteria, since the influence of θ on the layout is unpredictable. To address this issue, we use the reuse-and-optimization mentioned above scheme for efficient layout generation of large graphs, in which reusing proper pre-computed force models is performed to provide a clear overview of an input graph. Subsequently, the user can specify task-related criteria for subgraphs of interest to refine the layout using local optimization. Figure 2 shows an example.

Model Reuse. We generate a layout for a given graph by reusing a force model obtained from a similar graph using the following three steps: First, for a given graph G^*, we find a similar graph G^s, where G^s is smaller in size and achieves a similarity score of at least 0.5 with G^* based on graph kernel methods [24], [32]. Next, for a set of optimization objectives given by the user, we use the optimization method outlined in algorithm 1 on G^s to pre-compute a set of corresponding force models and layouts. Finally, after comparing the layouts for G^s, the user can apply the most appropriate force model get from the previous step to G^*, resulting in the desired layout of G^* for the user. Figure 2a and Figure 2b illustrate two of the pre-computed layouts of the most similar graph *bus494* to the graph *bcspwr07*, while the user-desired layout of *bcspwr07* in Figure 2c is generated by reusing the force model from Figure 2a. In the process of obtaining Figure 2c, the layout of *bus494* took 1.38s, the model reuse took 5.88s, the CL value of Figure 2c was 0.0242. In contrast, Figure 2d shows the result form directly using AutoFDP[CL] on *bcspwr07*, which takes 12.61s with CL value of 0.0238. Using model reuse not only yields a metric value and visually very similar results, but also takes less time.

Local Optimization. During the exploration of node-link diagrams, users often perform tasks on subgraphs. For example, they might be interested in finding the set of adjacent nodes to a node or exploring the sub-clusters of a large cluster. Although a task-driven fisheye lens [27] might show structures of interest, they inevitably change the context. To address this issue, we propose a context-preserving local optimization technique to improve the readability of subgraph layout.

Given a layout $\mathbf{X}$ of graph G, we allow users to select a subgraph SG with node positions $\mathbf{X}_s$ within a region R and then impose a task-related function ψ_s (e.g., NP for exploring clusters) to SG for adjusting the layout. By initializing SG with $\mathbf{X}$, we first apply algorithm 1 to SG to generate a layout $\mathbf{X}'_s$ and then employ the iterative closest point (ICP) algorithm [33] to align the layout $\mathbf{X}'_s$ to the reference $\mathbf{X}_s$. Since the optimization might generate layouts with flipped structures, we flip all layouts horizontally and vertically to create more variants for $\mathbf{X}'_s$. Next, we use ICP to find an affine transformation including translation, rotation, and scaling to map the variants of $\mathbf{X}'_s$ to $\mathbf{X}_s$. Finally, we choose the variant with the smallest cost for mapping to the reference. Figure 2e shows the refined layouts after performing local optimizations on Figure 2c, the circle structure is shown more clearly in Figure 2e.

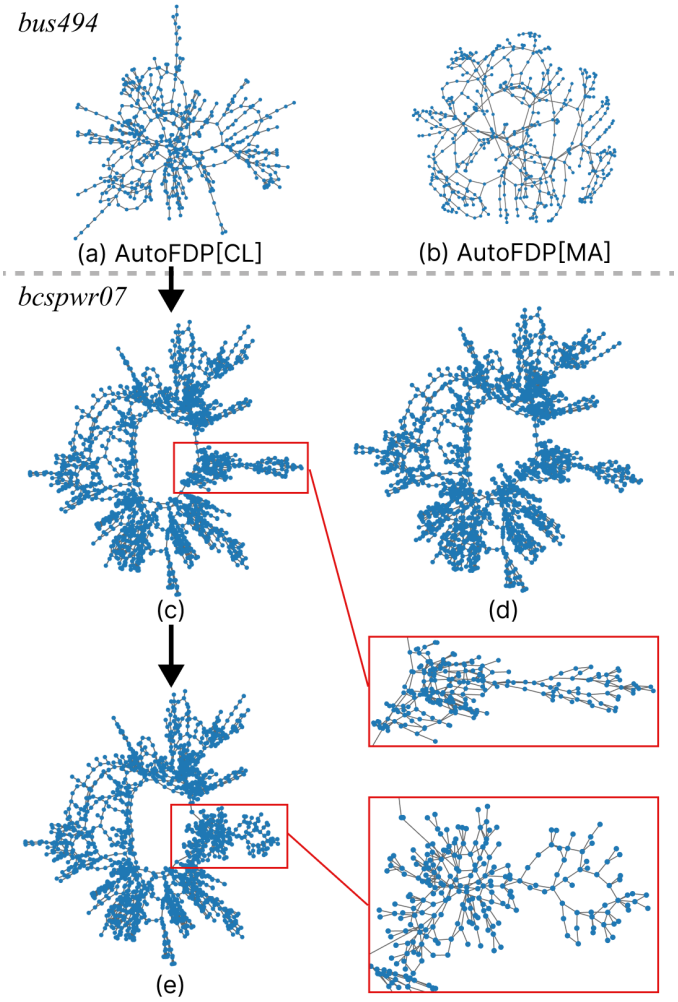


Fig. 2: Example-based layout generation for *bcspwr07* with AutoFDP. First, the most similar graph *bus494* is found, the pre-computed layouts are shown in (a,b); then unlike (d) which optimizes CL directly, the corresponding force model of (a) is reused for generating the layouts (c) of the graph *bcspwr07*; (e) refining the layouts of the subgraphs by applying local optimizations using SE as the criterion.

V. Evaluation

AutoFDP[1], developed in C++ and leveraging the Taurus library [6], is evaluated against traditional and deep learning methods to learn if AutoFDP outperforms existing methods and how deep learning methods perform on real graphs. We limit our deep learning comparison to deepGD and SmartGD due to their availability. Thus, comparisons are classified into non-learning (SGD[2] and standard layouts) and learning-based (deepGD and SmartGD) categories for a balanced analysis, detailed in Subsection V-A and Subsection V-B respectively. We run the test on a machine with AMD Ryzen 3990X (64

[1]https://github.com/IDEASLab-AutoFDP/AutoFDP

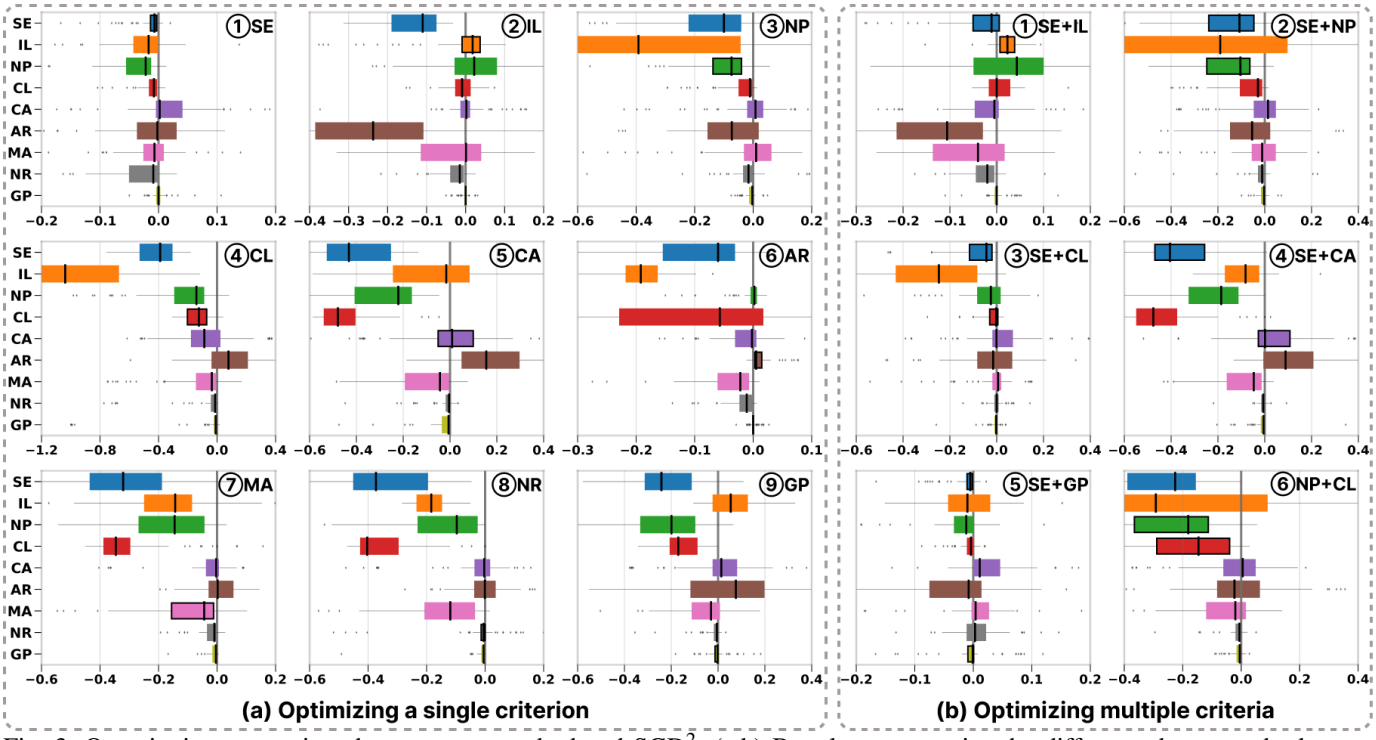


Fig. 3: Quantitative comparison between our method and SGD^2: (a,b) Boxplots summarize the difference between the layouts generated by two methods of optimizing a single criterion (a) and weighted criteria (b). The optimized criterion on each sub-figure is listed on the top right. The weights of different criteria in weighted criteria are equal, all of which are 0.5.

cores at 2.9GHz) processor and 256GB memory, where our experiments use up to 6 threads and 1 GB of memory.

A. *Comparison to non-learning based methods*

We compare our method with SGD^2 by optimizing every single criterion of the nine and six sums of weighted criteria provided by the original implementation of SGD^2. For the classic layout methods, we choose the stress model (SM) with a stochastic gradient descent solver [30], SFDP [1], and Maxent [19], all of them can be formulated within the quotient-based force representation [6] and correspond to spring-electrical models, stress models, and a combination of these two models, respectively. In addition, we include tsNET [34] and DRGraph [35] which are based on dimensionality reduction techniques and designed for efffciently preserving neighborhoods. For all these methods, we use implementations with the default parameters provided by the original authors.

Datasets. For a comprehensive comparison, we incorporate a number of real-world graphs sourced from a widely used resource [36], and also generate a set of graphs with common structures. In all, we use 30 real-world graphs of different applications (see supplementary material) and 45 synthetic graphs with sizes varying from 100 to 5000 nodes and 128 to 19,016 edges. The synthetic graphs were generated in the same way as the synthetic graphs in Taurus [6] and have three different kinds of structures: *grids*, *binary trees*, and *clusters*, with each of them having 15 exemplar graphs. For each grid graph, the ideal desired layout is a uniform grid with a set of uniform-sized squares; the ideal layout for a binary tree should preserve the hierarchy, while for a clustered graph the goal is to maintain the cluster structures.

Measures. We used the nine criteria provided by Ahmed et al. [9] described in Sec. III-B. We calculate the scores of our approach ($M_{AutoFDP}$) and SGD^2 (M_{sgd^2}) for two layouts of a given graph for each quality measure and compute the difference δM:

$$\delta M = M_{AutoFDP} - M_{sgd^2}, \tag{12}$$

where a value around zero indicates that both techniques produce similar results. Since small values for all measures are better, a negative δM is better.

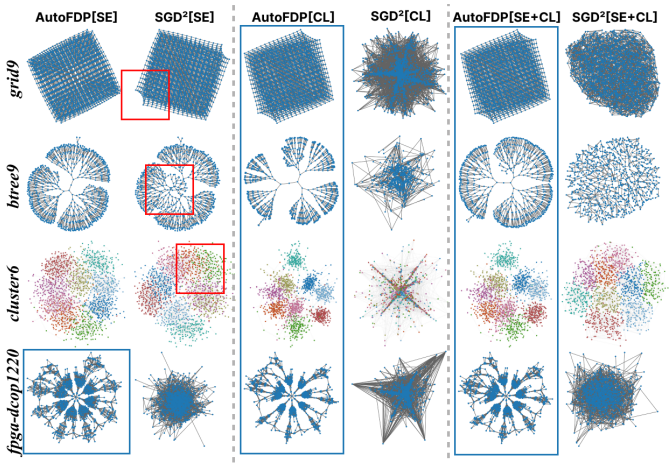


Fig. 4: Example layouts generated by three variant of AutoFDP and the corresponding variant of SGD^2 for the four types of graphs: grid (top row), tree (second row), clustered graph (third row), and real graph (bottom row). In which blue boxes represent well-perform layouts and red boxes represent layouts where structures are not visually represented well.

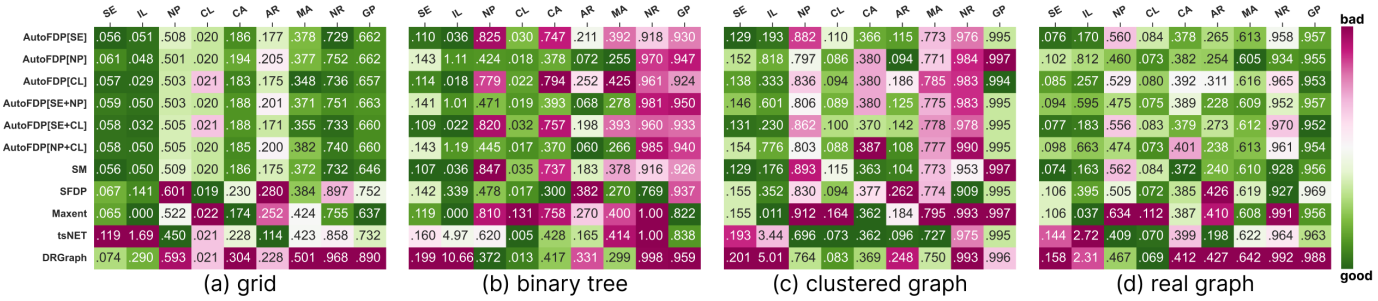


Fig. 5: Heatmaps presenting the scores of nine quality measures for layouts generated by eleven layout methods on four types of graphs. Each row represents a layout method, and each column is a quality measure. All columns are colored relatively with regard to best and worst value.

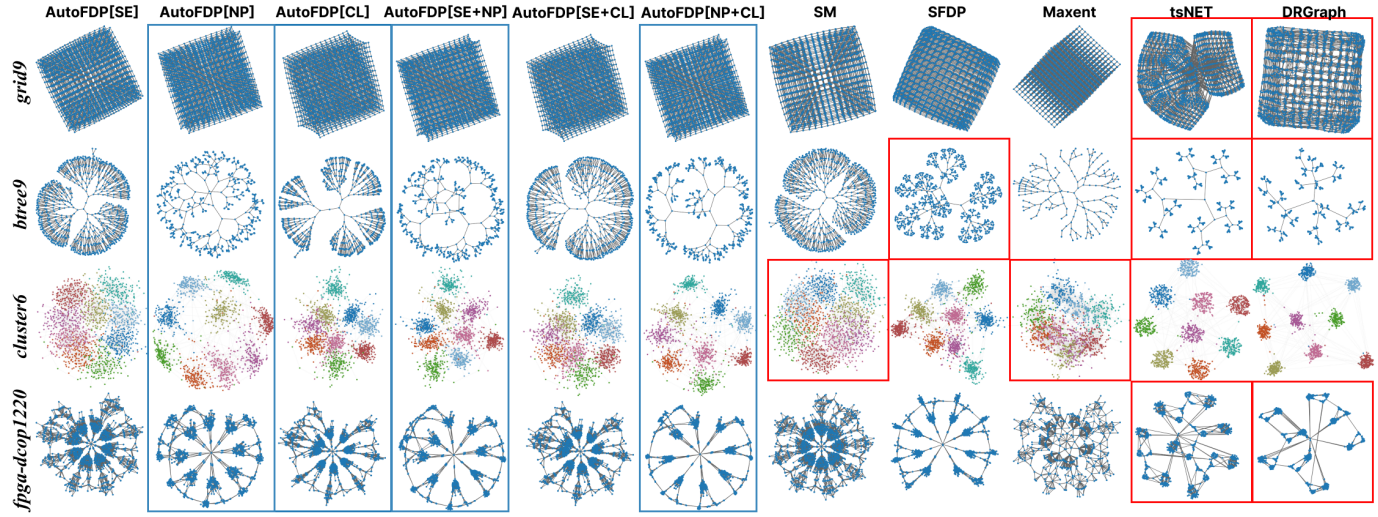


Fig. 6: Example layouts generated by six variant of AutoFDP and five force-based methods for the four types of graphs: grid (top row), tree (second row), clustered graph (third row), and real graph (bottom row). The blue boxes represent well-perform layouts, the red boxes represent layouts where structures are not visually represented well.

Parameters. The efficiency of our method and the quality of the layout are affected by four parameters: the maximum number of iterations, the initial value of θ, the step size ρ and λ. As suggested by Xue et al. [6] we initialize θ with (1,1,1,-1,-1,-1) and set the maximum number of iterations to 50.

Comparing with SGD^2 The boxplots in Figure 3 summarize the differences between our method and SGD^2 in terms of the nine criteria. Each sub-figure in Figure 3a corresponds to the results measured on the layouts produced by optimizing a single criterion as indicated on the top right. To comprehensively evaluate layout quality beyond the specific optimization target, we report performance across multiple criteria. This evaluation reflects the practical requirement that a useful layout must maintain good overall readability, not just excel on a single, user-specified criterion at the expense of others. Taking NP, MA, and NR as the optimization objective, our method yields layouts with better performance on almost all measures. For SE, CL, CA, and AR, our layout results are better or similar for most measures and slightly worse on one measure. For example, our results obtained by optimizing SE and CL (see Figure 3a(1) and Figure 3a(4)) are slightly worse in terms of the CA and AR measures, respectively. However, their visual layouts more effectively reveal grid, tree, clusters, and symmetric structures compared to SGD^2 (as shown in the first four columns of Figure 4). We believe that this is because our quotient-based force models consider global graph structures manifested by the shortest path distances between all node pairs. Yet, our results obtained by optimizing IL and GP show worse performance on two measures, NP and IL, when optimizing IL, and on two measures, AR and IL, when optimizing GP. However, the difference between the two methods is less than 0.1, indicating that both methods perform similarly. After examining the visual layouts, we observed that both methods fail to produce meaningful layouts when optimizing IL. For example, no layout for both methods on binary trees accurately shows the eight branches of the given graph, which can be revealed by optimizing SE. For GP, AutoFDP better reveal graph structures for all kinds of graphs (see the supplemental material). We hypothesize that this is due to IL and GP only considering local structure, which is insufficient to capture the overall graph structure.

For each of the six evaluated weighted criteria—constructed using an average weighting scheme and criterion combinations selected based on superior and compatible pairs from SGD^2—the layouts produced by AutoFDP consistently outperform or closely match those generated by SGD^2 across the majority of evaluation metrics. The exceptions are the layouts optimized for SE and IL (see Figure 3b(1)), which perform slightly worse in NP and IL; the layout optimized

for SE and CA (see Figure 3b(4)), which performs slightly worse in AR. However, in all cases of poorer performance, the differences are less than 0.1. Despite using this simple uniform weighting strategy without any graph-specific tuning, after carefully checking the visual layouts, we are convinced that our method can show the important graph structures, especially for real graphs. For instance, as depicted in the fifth and sixth columns of Figure 4, our method consistently generates layouts that effectively reveal grid, tree, cluster, and symmetric structures using these equally weighted criteria combinations. For a comprehensive view of all the visual layouts, please refer to the supplemental material.

Comparing with Force-based methods. The heatmaps in Figure 5 present the average values of nine measures generated by eleven layout methods for four types of graphs. Each row corresponds to a layout method and each column corresponds to one quality measure, where each cell shows the average measure value with a background color encoding the relative measure on the same column. Since traditional methods cannot optimize different criteria, we did not perform a comprehensive comparison of criteria as we did with SGD^2, but instead selected three important and relevant criteria and their combination: SE (optimized by SM/MAXENT), NP (focused by SFDP/tsNET/DRGraph), and CL (well-behaved in DRGraph).

We can see that AutoFDP[SE] and SM perform similarly and yield similar layouts on all graphs (see the first and seventh columns in Figure 6). They generate the best SE scores on all graphs, while tsNET yields good NP scores, especially for the grid and clustered graphs but performs the worst in SE and IL for all graphs. Maxent performs the best in IL on all graphs since it is designed to explicitly maintain uniform edge lengths, whereas SFDP performs the worst on a few measures on the grid and behaves similarly on the other types of graphs, yielding poor performance in SE and AR. DRGraph performs well on grid in terms of the SE and IL, and it also performs well on other types of graphs in terms of the NP and CL. However, it generally performs poorly on other criteria.

In contrast, our methods create a good balance between optimizing different aspects. For example, AutoFDP[NP] also attempts to preserve neighborhoods as tsNET but also maintains overall structures and uniform edge lengths. Although its yielded IL scores are the third to the last on the binary trees, clustered and real graphs, its scores are much smaller than the ones of tsNET and DRGraph. Likewise, AutoFDP[CL] yields good scores not only in CL but also in SE and IL. The performance of AutoFDP[SE+CL] is between AutoFDP[SE] and AutoFDP[CL] in most measures, especially in SE and CL. The same phenomenon is observed for AutoFDP[SE+NP] and AutoFDP[NP+CL].

This observation is consistent with visual results shown in Figure 6, where the layouts of the tree and clustered graphs yielded by AutoFDP[SE+CL] and AutoFDP[NP+CL] can be treated as the interpolation between the layouts generated by AutoFDP[SE] / AutoFDP[NP] and AutoFDP[CL]. Since large weights can cause certain criteria to have a dominant influence on the optimization process [9] , sometimes the layouts generated by optimizing some combined criteria might be similar to the ones generated by optimizing a specific single criterion.

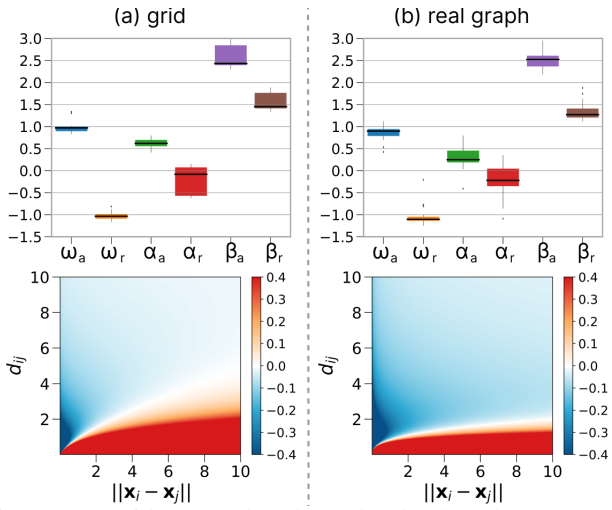


Fig. 7: For grids (a) and real graphs (b), boxplots (top) summarize the parameters obtained by AutoFDP[NP]. Heatmaps (bottom) show the resultant force magnitude for two nodes with varying graph-theoretical and Euclidean distances generated by applying the median values of six parameters to the quotient-based force representation.

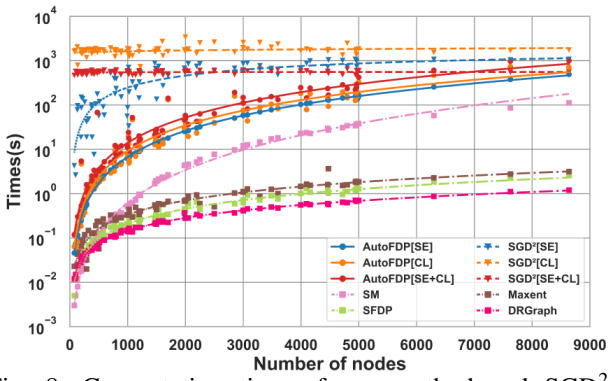


Fig. 8: Computation time of our method and SGD^2 for optimizing three different criteria in comparison to four classical methods.

For example, the layouts generated by AutoFDP[SE+CL] and AutoFDP[CL] on the grid graphs are highly similar.

As shown in Figure 6, SM allows to deliberately show the grid structure, exhibits the global symmetry of the binary tree and the real graph *fpga-dcop1220*, but cannot reveal clusters in the clustered graph. SFDP keeps the global structure of grid, clustered graphs and *fpga-dcop1220*, but fails to show the symmetry of the binary tree. DRGraph and tsNET do not allow to unfold the 3D grid and maintain the global symmetry of the binary tree as well as the real graph *fpga-dcop1220*, although they clearly reveal clusters in the clustered graph. Maxent clearly shows the structure of the 3D grid graph and *fpga-dcop1220* with a uniform node distribution, but cannot separate all clusters.

However, AutoFDP[NP], AutoFDP[CL], AutoFDP[SE+NP] and AutoFDP[SE+CL] can generate reasonable layouts for four different types of graphs. We speculate that the robustness of our methods is due to the force model searched for each graph in terms of the given criteria.

Variation Analysis. To investigate how the obtained force

models vary with different input graphs, we compute the statistics for each of the six parameters $\{\omega_a, \alpha_a, \beta_a, \omega_r, \alpha_r, \beta_r\}$ obtained by optimizing different criteria on 45 synthetic graphs and 30 real graphs. For the ones generated by AutoFDP[SE], their mean values are (1.01,0.99,2.00,-1.02,0.02,0.99) with variances being close to zero. The parameters of the SM are (2, 1, 2, -2, 0, 1), which closely align with these means after normalizing the weights ω_a and ω_r to (1,-1). Such results confirm that our optimization can produce accurate results.

However, unlike SE which can be optimized directly with the stress model, the other criteria like NP cannot be directly represented in a force-based framework and the parameters obtained might be largely different in different graphs. The top row in Figure 7 summarizes the values of these parameters generated by AutoFDP[NP] on the grid and real graphs using boxplots. We did not show the parameters for the tree and clustered graphs, since they are almost the same. We can see that the variations of α_r, β_a, and β_r are the largest on the grid graphs and the variation of α_a is the largest on the real graphs. In other words, AutoFDP[NP] attempts to find different parameters to reveal different graph structures.

To further explore how these parameters influence the forces on different types of graphs, we compute the resultant force for the two nodes with varying graph-theoretic measures d_{ij} and Euclidean distance in 2D space $||\mathbf{x}_i - \mathbf{x}_j||$ by using the median values of these parameters. The bottom row in Figure 7 shows the corresponding heatmaps, which show different effective resultant force ranges on two types of graphs,

where a positive value of the combined force indicates an attractive force between nodes and a negative value represents a repulsive force. For grid graphs, nodes with graph-theoretic distances less than 4 are subject to attractive forces between them at large Euclidean distances, helping to preserve neighborhood structure by compressing the sparse grid layout. In contrast, on real graphs, only nodes with graph-theoretic distances less than 2 may be subject to attractive forces between them, but repulsive forces are applied at all other ranges.

Such different behaviors are beneficial to enhance neighborhood preservation, where large repulsive forces are required to separate the nodes belonging to different k-nearest neighborhood graphs.

Runtime. Figure 8 illustrates the relationship between layout computation time and the number of graph nodes. To provide a comprehensive comparison, we include results of AutoFDP, other C++ based methods (SM, SFDP, Maxent, and DRGraph), and the most related baseline SGD^2. Due to space constraints, only the curves for the selected variants of AutoFDP and SGD^2 are shown here. Direct absolute runtime comparisons between implementations in different languages (C++ vs. Python for SGD^2 / tsNET) should be interpreted with caution, as they can be significantly influenced by language efficiency and optimization levels. The primary purpose here is to show the scaling trends of each method and provide a ballpark estimate of the relative computational cost. The computation time for the shortest path distances is excluded for all methods to ensure a fairer comparison of the core layout computation. The complete set of results, including SGD^2 and tsNET, is provided in the supplemental materials.

The results indicate that AutoFDP's runtime remains consistent across different optimization criteria. Its computation time is approximately one order of magnitude slower than SM, SFDP, Maxent, and DRGraph for graphs with more than 1000 nodes, but with the same scaling trend. This is expected, as AutoFDP jointly optimizes the graph layout model and layout, whereas the classical methods do not enforce specific aesthetic criteria. Despite this cost, AutoFDP demonstrates one order of magnitude speedup over SGD^2 on the graph with less than 3000 nodes. For graphs exceeding 3,000 nodes, since SGD^2 has a maximum computation time limitation, it may appear to be closer to our processing time at first glance. However, when examining the overall trend, our method actually demonstrates superior performance.

In summary, AutoFDP finds force-based layout models that adapt to the structures of different types of graph with less than 1000 nodes in a reasonable amount of time, while existing layout methods lack this capability. While the search process on large graphs has a higher computational cost, the optimized parameters on small graphs can be used to efficiently layout similar graphs. This makes AutoFDP primarily suitable for offline analysis, as well as for graph layout tasks where layout quality is a priority and there are similar small-scale graphs.

B. Comparison to Deep Learning-based Methods

We also conducted a comparative analysis of AutoFDP against deep learning-based techniques, namely DeepGD [10] and SmartGD [11]. For our experiments, we used the original code provided by the authors. Due to the limitation that the authors only offered code to optimize specific criteria, our comparison was confined to those criteria available in the source code. Specifically, the criteria included in the DeepGD source code are SE, SE+MA, SE+NR, SE+IL, and SE+TSNE, where the TSNE criterion measures the divergence between the graph space and the layout space [37]. Conversely, SmartGD only offered SE and CL as criteria. Since the definition of MA and NR used by SGD^2 is different from the one of DeepGD, we implemented the criteria of MA, NR, and TSNE as the optimization objectives for a fair comparison. For each objective, we retrained the model on the Rome dataset using the same configuration described in DeepGD and SmartGD paper, employing a single A100 GPU with 40 GB of memory.

Dataset. To assess the generalization ability of DeepGD and SmartGD, we constructed two datasets for performing independent and identically distributed (IID) and out-of-distribution (OOD) tests. The IID test includes 1000 graphs randomly chosen from the Rome dataset using test data selection code in DeepGD, resembling the structures of the training data, and the OOD one consists of synthesized graphs with varied structures, including 50 instances each of grids, binary trees, and clusters. Since DeepGD was trained on smaller graphs with less than 100 nodes, the synthesized graphs were created with node counts ranging from 10 to 500. Both DeepGD and SmartGD utilize 10,000 graphs selected from the Rome dataset as training dataset. To maintain consistency, we first run AutoFDP on the same training dataset, then take the average of the force model

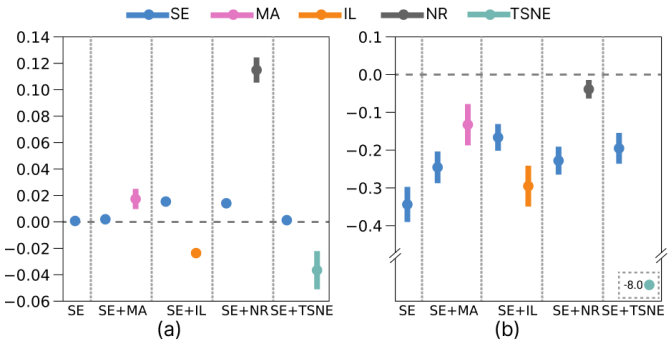


Fig. 9: Mean value and the 95% confidence interval of the difference between AutoFDP and DeepGD on the Rome dataset (a) and synthetic graph dataset (b) – lower values are better. The optimized objective for each sub-figure is provided at the bottom.

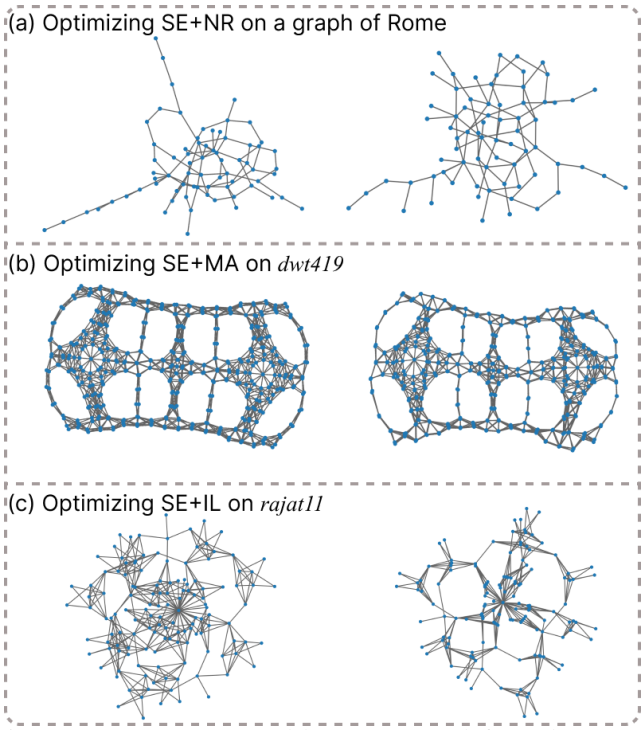


Fig. 10: Layouts generated by AutoFDP (left) and DeepGD (right) for three graphs: a graph of the Rome dataset (a) and two real graphs (b,c): *dwt419* (b) and *rajat11* (c). The optimized objective on each graph is listed at the top.

parameters as a frozen parameter $\bar{\theta}$ for testing on the test datasets.

Measures. As described in Subsection V-A, we employed the criteria provided by Ahmed et al. [9] as the evaluation metrics and calculated the difference δM (see Eq. 12) between the scores obtained by AutoFDP in comparison to those of DeepGD and SmartGD. A negative δM indicates that AutoFDP outperforms DeepGD, and vice versa. Due to limited space, we only present the differences in terms of optimized criteria with confidence interval (CI) in Figure 9 and Figure 11. Please refer to the supplementary material for comprehensive results.

Comparing with DeepGD. Figure 9 shows the differences between AutoFDP and DeepGD. On the Rome dataset (Figure 9a), the mean differences mostly range between [-0.02, 0.02], suggesting that both methods produce similar layouts. The exception is the result for optimizing SE+NR, where our score

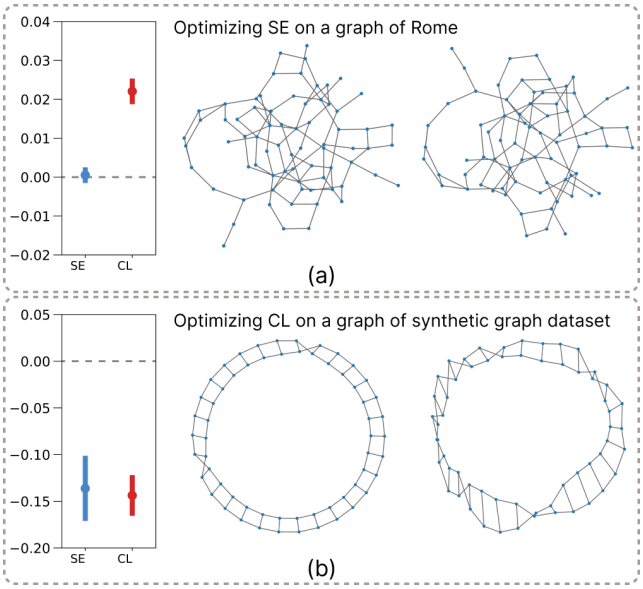


Fig. 11: Comparison of AutoFDP and SmartGD on Rome (a) and synthetic datasets (b). Error bars show the differences for SE and CL optimization. Example layouts by AutoFDP (left) and SmartGD (right), with optimized criteria listed at the top.

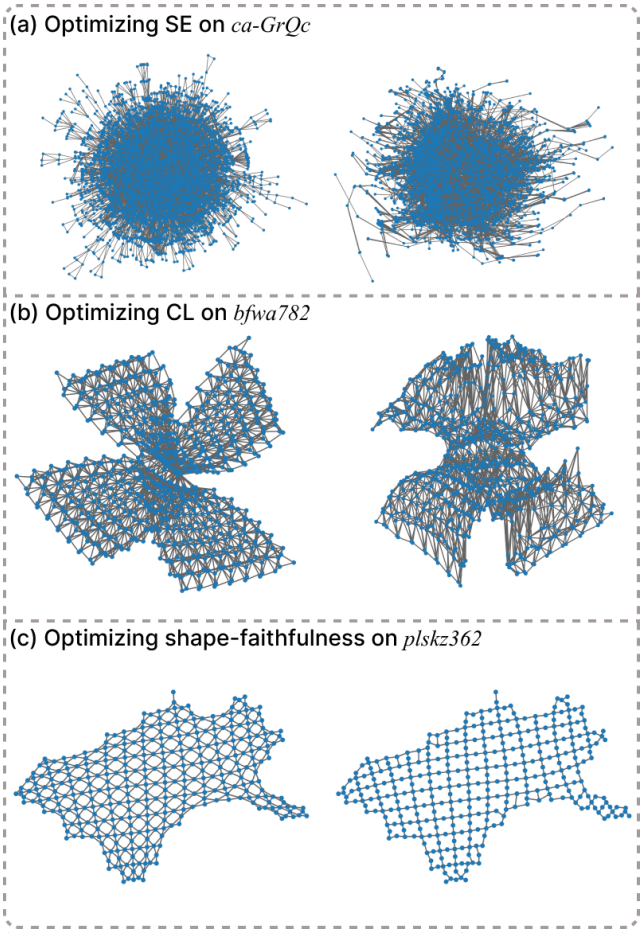


Fig. 12: Layouts generated by AutoFDP (left), SmartGD (right in (a) and (b)), and relative neighborhood graph (right in (c)) for three real graphs: *ca-GrQc* (a), *bfwa782* (b), and *plskz362* (c). The optimized objective on each graph is listed at the top.

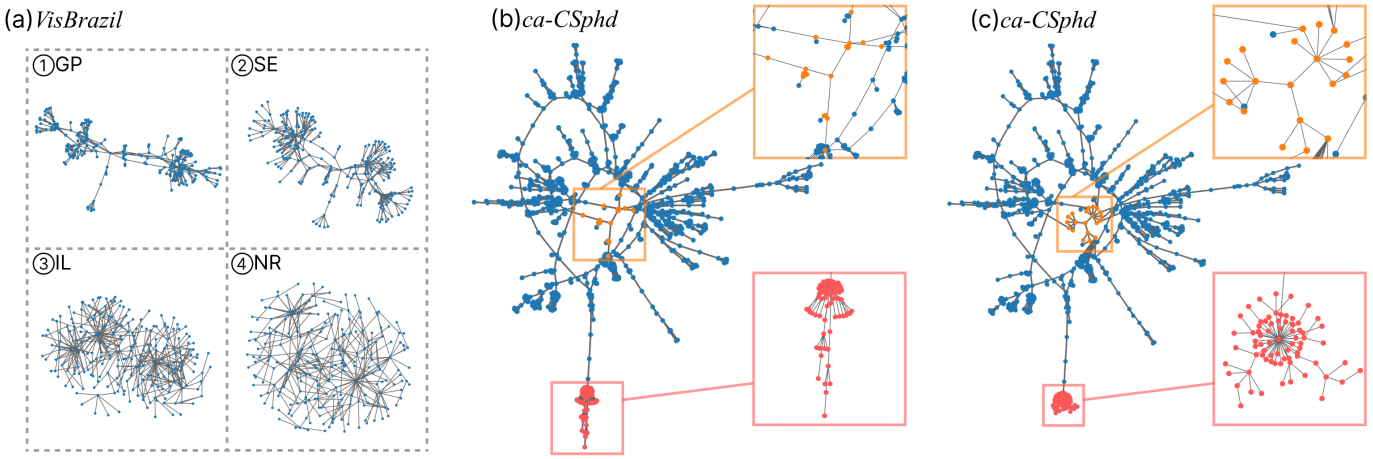


Fig. 13: Generating layout of a new graph with the reuse-and-optimization scheme. (a) Four layouts of the graph *VisBrazil* provided by AutoFDP with four different criteria; (b) A layout generated by reusing the force-based model in (a)(1) to the graph *ca-CSphd*; (c) Refining the layout by applying AutoFDP[SE] and AutoFDP[MA] to the orange and red subgraphs selected in (b), respectively.

is marginally worse than DeepGD's, with mean differences in NR around 0.11. Upon examining the corresponding visual layouts, we observed that AutoFDP generates more compact layouts but effectively reveals the graph structures, similar to DeepGD, see an example in Figure 10a.

Conversely, on the synthetic data (Figure 9b), the mean values of almost all criteria for all optimization objectives are negative, with a mean difference of approximately -0.2, indicating that our method significantly outperforms DeepGD. Upon analyzing the generated visual layouts (refer to the supplementary material), we found that DeepGD results in severe edge crossings and structural distortions for tree structures, noticeable grid distortions for grid graphs, and fails to produce clear clusters for clustered graphs, whereas AutoFDP accurately characterizes all structures. While the OOD comparison with DeepGD is asymmetric by design (training-dependent vs. training-free), it reveals a key strength of optimization-based approaches: On diverse unseen graphs, our method achieves robust performance without requiring training data or fine-tuning. This demonstrates its readiness for real-world deployment where graph types are unpredictable. In contrast, data-driven methods like DeepGD face generalization gaps under distribution shift.

To further compare the layouts generated by AutoFDP and DeepGD on real-world graphs, we conducted a qualitative comparison of 15 real graphs. For additional details, please refer to the supplementary material. As an illustration, we selected two smaller real-world graphs: *dwt419* (a grid graph with 419 nodes and 1,572 edges) and *rajat11* (a collaboration network with 135 nodes and 377 edges). As shown in Figure 10b and Figure 10c, AutoFDP reveals more structural details than DeepGD and offers a more accurate representation of the graph structures. It's worth noting that we limited the data size to 500 nodes for this comparison. However, on larger synthetic and real graphs with more than 1000 nodes, the quality of DeepGD's layouts deteriorates further (see supplementary material). Therefore, we conclude that AutoFDP outperforms DeepGD on real-world graphs.

Comparing with SmartGD. Figure 11 shows the differences between AutoFDP and SmartGD. On the Rome dataset (Figure 11a), the average difference for the optimized SE approximates 0, while the average difference for the optimized CL is approximately 0.02, showing a negligible distinction. The example layouts further showcase that both methods perform similarly on the Rome dataset. In contrast, on the synthetic dataset (Figure 11b), both the Optimize SE and CL exhibit an average difference of approximately -0.15, indicating that AutoFDP significantly outperforms SmartGD. After comparison, we discovered that the results produced by SmartGD are considerably distorted on grid data, and the graph structures are inaccurately represented on tree and clustered graphs (see supplementary material). We speculate that SmartGD also heavily relies on training data and lacks the extensibility of AutoFDP.

Similarly, we compared the differences between AutoFDP and SmartGD on real graphs. Figure 12a and Figure 12b depict the results of AutoFDP and SmartGD, respectively, when optimizing SE and CL on graph *ca-GrQc* and *bfwa782*. Both for optimizing SE and CL, AutoFDP better preserves data structure and evenly distributes nodes. Therefore, AutoFDP also outperforms SmartGD, especially on large graphs (see supplementary material for details).

In addition, SmartGD tested a shape-based metric [29], and we similarly conducted tests using the same metric. The shape-based metrics measure the similarity between the shape graph generated from the layout results and the original graph. Consistent with SmartGD, we used the relative neighborhood graph (RNG) to generate the shape graph. Since SmartGD does not provide the corresponding code, we only show the result of AutoFDP and the result of RNG on graph *plskz362* in Figure 10c. As illustrated, the majority of edges in the RNG layout align with those in our layout, with only a small fraction differing. Hence, our method also exhibits strong performance in optimizing shape-faithfulness. Please refer to the supplementary material for more results.

Runtime. Since AutoFDP involves solving an optimization problem, it is slower than applying a pre-trained DeepGD or

SmartGD model, particularly on large graphs. However, while DeepGD and SmartGD requires hours to days of training time, our method does not have a training phase, offering a potential advantage in terms of overall time efficiency.

In summary, AutoFDP performs similarly to DeepGD and SmartGD on datasets with graph structures akin to their training data but outperforms both DeepGD and SmartGD on other datasets, indicating superior generalizability of AutoFDP. In terms of runtime, our method is slower than both DeepGD and SmartGD, but it does not necessitate the substantial training time that DeepGD requires.

C. Case Study

Here, we demonstrate the effectiveness of our reuse-and-optimization scheme using real data. Given the social network *ca-CSphd* describing a faculty-student relationship, we first find the most similar graph from a set of graphs with pre-computed force-based models. The retrieved graph is *VisBrazil*, which is a collaboration network with tree-like structures. As shown in Figure 13a, our method pre-computes four models by separately optimizing GP, SE, IL, and NR, resulting in four layouts, where details are gradually added from Figure 13a(1) to Figure 13a(4) and only takes only 0.227s for average.

These results indicate that our method can find multiple meaningful layouts that show different aspects of a given graph, providing the user with a suitable reference.

Since the layout in Figure 13a(1) conveys the overall hierarchical structure, the user reuses its force-based model to the graph *ca-CSphd* for rapidly getting an overview, which takes 2.25 s. In contrast, direct use of AutoFDP[GP] on *ca-CSphd* takes 4.54 s and gets a very similar layout with GP value of 0.98, wheras 0.99 for Figure 13b. As shown in Figure 13b, a radial layout reveals the hierarchy, where the influential nodes are placed in the center and the leaf nodes are arranged on the outside. Yet, the hierarchical structure of the subgraphs around the central nodes in Figure 13b is obscured. For example, the user gets the orange subgraph by clicking on the center node and selecting its neighbors within a 3-ring neighborhood. A few adjacent edges in the selected orange subgraph are bundled together (see the zoomed inset in the top right). Some tree branches exhibit heavy visual clutter, such as the red subgraph selected by the user with the lasso tool in Figure 13b. Such behaviors prevent us from exploring the structures in detail.

To show the inner hierarchy within the orange subgraph, the user runs the local optimization with the criterion of SE and obtains a new result shown in Figure 13c. The inset shows the three-level hierarchy that is consistently connected to the other part. Meanwhile, the user applies the local optimization using the MA criterion to the red subgraph, a local tree structure is obtained. As shown in Figure 13c, all adjacent edges of the red subgraph are nearly uniformly spread around the parent nodes.

VI. Conclusion and Future Work

We propose a general graph layout framework AutoFDP that automatically selects a force model that generates a graph layout to meet a given sum of weighted criteria. Built on the virtual physics-based force representation, AutoFDP automatically selects a proper graph layout model that not only adapts to the structures of different types of graphs but can also be reused for other graphs. Our evaluation demonstrates that AutoFDP produces layouts that are comparable or even superior to SGD^2 and deep learning-based methods for most graphs and can handle large graphs as well.

However, our framework has a few limitations that may be addressed in future research. One is that it assumes that forces are exerted on all node pairs, requiring the computation of pairwise shortest path distances between all nodes, which is costly for large graphs. We plan to explore combining the optimization of force ranges with dynamically computing shortest path distances [38] or other physics-inspired force representations [13], [39]. Second, reusing a pre-computed force model of a similar graph might not yield a better layout than directly optimizing the input graph with the same criteria. Therefore, it is crucial to offer guidance on choosing suitable criteria for different graph analysis tasks. Last, current deep learning-based methods directly solve for node positions that often yield poor layouts for large graphs, we will therefore investigate the possibility of integrating virtual physics-based force representations into neural networks [10], [11].

Acknowledgments

The authors like to thank the anonymous reviewers for their valuable input. This work is supported by grants from the NSFC (No.62402284, No.92367202, No.62132017, and No.U2436209), NSF of Shandong province (No.ZR2024QF212 and No.ZQ2022JQ32), National Key R&D Program of China (No.2022ZD0160805), the Beijing Natural Science Foundation (No.L247027), the Fundamental Research Funds for the Central Universities, and the Research Funds of Renmin University of China.

References

[1] Y. Hu, "Efficient high-quality force-directed graph drawing," *Mathematica Journal*, vol. 10, no. 1, pp. 37–71, 2005.

[2] T. M. Fruchterman and E. M. Reingold, "Graph drawing by force-directed placement," *Software: Practice and Experience*, vol. 21, no. 11, pp. 1129–1164, 1991.

[3] T. Kamada, S. Kawai *et al.*, "An algorithm for drawing general undirected graphs," *Information Processing Letters*, vol. 31, no. 1, pp. 7–15, 1989.

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

[5] Y. Wang, Y. Wang, Y. Sun, L. Zhu, K. Lu, C.-W. Fu, M. Sedlmair, O. Deussen, and B. Chen, "Revisiting stress majorization as a unified framework for interactive constrained graph visualization," *IEEE Transactions on Visualization and Computer Graphics*, vol. 24, no. 1, pp. 489–499, 2017.

[6] M. Xue, Z. Wang, F. Zhong, Y. Wang, M. Xu, O. Deussen, and Y. Wang, "Taurus: Towards a unified force representation and universal solver for graph layout," *IEEE Transactions on Visualization and Computer Graphics*, vol. 29, no. 01, pp. 886–895, 2023.

[7] H. C. Purchase, "Metrics for graph drawing aesthetics," *Journal of Visual Languages and Computing*, vol. 13, no. 5, pp. 501–516, 2002.

[8] R. Ahmed, F. D. Luca, S. Devkota, S. Kobourov, and M. Li, "Graph drawing via gradient descent, $(GD)^2$," in *International Symposium on Graph Drawing and Network Visualization*. Springer, 2020, pp. 3–17.

[9] 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*, vol. 28, no. 6, pp. 2388–2399, 2022.

[10] 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*, vol. 41, no. 5, pp. 32–44, 2021.

[11] X. Wang, K. Yen, Y. Hu, and H. Shen, "SmartGD: A GAN-based graph drawing framework for diverse aesthetic goals," *IEEE Transactions on Visualization and Computer Graphics*, vol. 30, no. 8, pp. 5666–5678, 2024.

[12] B. Colson, P. Marcotte, and G. Savard, "An overview of bilevel optimization," *Annals of Operations Research*, vol. 153, pp. 235–256, 2007.

[13] F. Zhong, M. Xue, J. Zhang, F. Zhang, R. Ban, O. Deussen, and Y. Wang, "Force-directed graph layouts revisited: a new force based on the t-distribution," *IEEE Transactions on Visualization and Computer Graphics*, vol. 30, no. 7, pp. 3650–3663, 2024.

[14] P. Eades, "A heuristic for graph drawing," *Congressus Numerantium*, vol. 42, pp. 149–160, 1984.

[15] S. G. Kobourov, *Force-Directed Algorithms*. Handbook of Graph Drawing and Visualization, 2013.

[16] A. Noack, "An energy model for visual graph clustering," in *International Symposium on Graph Drawing*. Springer, 2003, pp. 425–436.

[17] 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*, vol. 9, no. 6, p. e98679, 2014.

[18] M. Ortmann, M. Klimenta, and U. Brandes, "A sparse stress model," in *International Symposium on Graph Drawing and Network Visualization*. Springer, 2016, pp. 18–32.

[19] E. R. Gansner, Y. Hu, and S. North, "A maxent-stress model for graph layout," *IEEE Transactions on Visualization and Computer Graphics*, vol. 19, no. 6, pp. 927–940, 2012.

[20] E. N. Argyriou, M. A. Bekos, and A. Symvonis, "Maximizing the total resolution of graphs," *The Computer Journal*, vol. 56, no. 7, pp. 887–900, 06 2012.

[21] M. A. Bekos, H. Förster, C. Geckeler, L. Holländer, M. Kaufmann, A. M. Spallek, and J. Splett, "A heuristic approach towards drawings of graphs with high crossing resolution," in *Graph Drawing and Network Visualization*. Springer International Publishing, 2018, pp. 271–285.

[22] M. Radermacher, K. Reichard, I. Rutter, and D. Wagner, "A geometric heuristic for rectilinear crossing minimization," in *2018 Proceedings of the Twentieth Workshop on Algorithm Engineering and Experiments*. SIAM, 2018, pp. 129–138.

[23] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," *Advances in Neural Information Processing Systems*, vol. 27, 2014.

[24] 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*, vol. 24, no. 1, pp. 478–488, 2017.

[25] J. Pan, W. Chen, X. Zhao, S. Zhou, W. Zeng, M. Zhu, J. Chen, S. Fu, and Y. Wu, "Exemplar-based layout fine-tuning for node-link diagrams," *IEEE Transactions on Visualization and Computer Graphics*, vol. 27, no. 2, pp. 1655–1665, 2020.

[26] E. R. Gansner, Y. Koren, and S. C. North, "Topological fisheye views for visualizing large graphs," *IEEE Transactions on Visualization and Computer Graphics*, vol. 11, no. 4, pp. 457–468, 2005.

[27] Y. Wang, Y. Wang, H. Zhang, Y. Sun, C.-W. Fu, M. Sedlmair, B. Chen, and O. Deussen, "Structure-aware fisheye views for efficient large graph exploration," *IEEE Transactions on Visualization and Computer Graphics*, vol. 25, no. 1, pp. 566–575, 2018.

[28] M. Chrobak, M. T. Goodrich, and R. Tamassia, "Convex drawings of graphs in two and three dimensions (preliminary version)," in *Proceedings of the Twelfth Annual Symposium on Computational Geometry*, 1996, pp. 319–328.

[29] P. Eades, S.-H. Hong, K. Klein, and A. Nguyen, "Shape-based quality metrics for large graph visualization," in *Graph Drawing and Network Visualization*. Springer International Publishing, 2015, pp. 502–514.

[30] J. X. Zheng, S. Pawar, and D. F. Goodman, "Graph drawing by stochastic gradient descent," *IEEE Transactions on Visualization and Computer Graphics*, vol. 25, no. 9, pp. 2738–2748, 2018.

[31] E. Aarts, "A stochastic approach to combinatorial optimization and neural computing," *Simulated Annealing and Boltzmann Machines*, 1989.

[32] Y. Wang, L. Chen, J. Jo, and Y. Wang, "Joint t-SNE for comparable projections of multiple high-dimensional datasets," *IEEE Transactions on Visualization and Computer Graphics*, vol. 28, no. 1, pp. 623–632, 2021.

[33] D. Chetverikov, D. Svirko, D. Stepanov, and P. Krsek, "The trimmed iterative closest point algorithm," in *2002 International Conference on Pattern Recognition*, vol. 3. IEEE, 2002, pp. 545–548.

[34] J. F. Kruiger, P. E. Rauber, R. M. Martins, A. Kerren, S. Kobourov, and A. C. Telea, "Graph layouts by t-SNE," *Computer Graphics Forum*, vol. 36, no. 3, pp. 283–294, 2017.

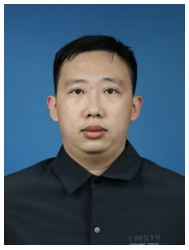
[35] M. Zhu, W. Chen, Y. Hu, Y. Hou, L. Liu, and K. Zhang, "DRGraph: An efficient graph layout algorithm for large-scale graphs by dimensionality reduction," *IEEE Transactions on Visualization and Computer Graphics*, vol. 27, no. 2, pp. 1666–1676, 2020.

[36] S. Di Bartolomeo, E. Puerta, C. Wilson, T. Crnovrsanin, and C. Dunne, "A collection of benchmark datasets for evaluating graph layout algorithms," Under submission to Graph Drawing Posters, 2023. [Online]. Available: https://visdunneright.github.io/gd_benchmark_sets/

[37] L. Van der Maaten and G. Hinton, "Visualizing data using t-SNE," *Journal of Machine Learning Research*, vol. 9, no. 86, pp. 2579–2605, 2008.

[38] L. Roditty and U. Zwick, "On dynamic shortest paths problems," in *Algorithms–ESA 2004: 12th Annual European Symposium*. Springer, 2004, pp. 580–591.

[39] Y. Hu and Y. Koren, "Extending the spring-electrical model to overcome warping effects," in *IEEE Pacific Visualization Symposium*. IEEE, May 2009, pp. 129–136.



Mingliang Xue is a Postdoctoral Researcher at the Joint SDU-NTU Center for Artificial Intelligence Research, Shandong University. His research interests include graph visualization and dimensional reduction.

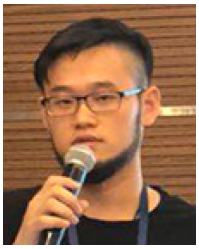
Yifan Wang is a Master's student in the School of Computer Science and Technology, at Shandong University. His research interest is graph visualization.

Zhi Wang is a Master's student in the School of Computer Science and Technology, at Shandong University. His research interest is graph visualization.

Lifeng Zhu is a Professor in the Department of Instrument Science and Technology at Southeast University. He received his doctoral degree in computer science from Peking University in 2012. His research topics are visual computing and human-computer interaction. His interests are in particular shape modeling and its applications in medical science and fabrication.

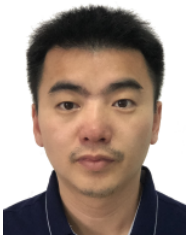
Lizhen Cui (Member, IEEE) is currently a Professor and the Chair of the School of Software Engineering, Shandong University, Jinan, China. From 2013 and 2014, he was a Visiting Scholar with the Georgia Institute of Technology, Atlanta, GA, USA. His current research interests include data science and engineering, intelligent data analysis, service computing, and collaborative computing.

Yueguo Chen received the BS and master's degrees in mechanical engineering and control engineering from Tsinghua University, Beijing, in 2001 and 2004. He received the PhD degree in computer science from the National University of Singapore in 2009. He is currently a professor in School of Information, Renmin University of China. His recent research interests include massive knowledge bases and semantic search.



Zhiyu Ding obtained his Ph.D. from the State Key Lab of CAD&CG , Zhejiang University. His research interests encompass AI, HCI, and visual computing.

Oliver Deussen graduated at Karlsruhe Institute of Technology and is professor at University of Konstanz (Germany) and visiting professor at the Chinese Academy of Science in Shenzhen. He served as Co-Editor in Chief of Computer Graphics Forum and was President of the Eurographics Association. His areas of interest are modeling and rendering of complex biological systems, non-photorealistic rendering as well as Information Visualization.



Yunhai Wang is a Professor in School of Information, Renmin University of China. He serves as the associate editor of *IEEE Transactions on Visualization and Computer Graphics* and *IEEE Computer Graphics and Applications*. His interests include scientific visualization, information visualization, and computer graphics.