Minimizing Cross Intersections in Graph Drawing via Linear Splines

Rida Ghafoor Hussain, Matteo Tiezzi, Gabriele Ciravegna, Marco Gori

Minimizing Cross Intersections in Graph Drawing via Linear Splines

Rida Ghafoor Hussain[1(✉)], Matteo Tiezzi[2], Gabriele Ciravegna[3], and Marco Gori[2,3]

[1] University of Florence, Florence, Italy
rida.ghafoor@unifi.it
[2] University of Siena, Siena, Italy
{mtiezzi,marco.gori}@unisi.it
[3] Universitè Cóte d'Azur, Nice, France
gabriele.ciravegna@inria.fr

Abstract. The generation of aesthetically pleasing graph layouts is the main purpose of Graph Drawing techniques. Recent contributions delved into the usage of Gradient-descent (GD) based schemes to optimize differentiable loss functions, built to measure the graph layout adherence to given layout characteristics. However, some properties cannot be easily expressed via differentiable functions. In this direction, the recently proposed Graph Neural Drawer (GND) framework proposes to exploit the representational capability of neural models in order to be able to express differentiable losses, specifically for edge intersection, that can be subsequently optimized via GD. In this paper, we propose to improve graph layout readability leveraging linear splines. We exploit the principles behind GND and use a neural model both to identify crossing edges and to optimize their relative position. We split crossing edges introducing linear splines, and threat the control points as novel "fake" vertices that can be optimized via the underlying layout optimization process. We provide qualitative and quantitative analysis over multiple graphs and optimizing different aesthetic losses, that show how the proposed method is a viable solution.

Keywords: Graph Drawing · Gradient Descent · Graph Neural Drawers · Splines

1 Introduction

The widening interest over structured data and its representation calls to the development of methods and techniques to better visualize and investigate its complex relations and characteristics [8]. Graphs are mathematical structures which model pairwise relations (edges) between entities (nodes). Graph Drawing [5] methods focus on developing algorithmic techniques to generate drawings of graphs via, for instance, the node-link paradigm [10], i.e. entities represented as nodes and their relation expressed by edges linking them.

N. El Gayar et al. (Eds.): ANNPR 2022, LNAI 13739, pp. 28–39, 2023.
https://doi.org/10.1007/978-3-031-20650-4_3

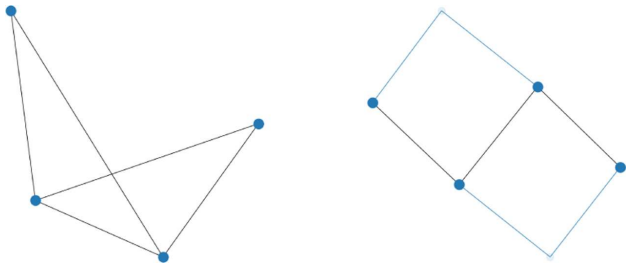

Fig. 1. Reducing the number of arc intersections via linear splines. On the left, a Diamond Graph layout has been obtained via an optimization process that is not able to fully avoid arc intersections. The proposed method introduce linear splines, whose control points (*fake nodes*), plotted with an alpha color transparency, are treated as additional nodes for the underlying optimization process. This approach ease the graph drawing optimization process.

The prominent role of graph drawing is the improvement of graph readability, which is generally evaluated with some aesthetic criteria such as community preservation, crossing angles, edge length variance, number of crossing edges, etc. [2]. The final goal is to devise appropriate coordinates for the node positions, a goal that often requires to explicitly express and combine the aforementioned criteria. Beyond the classic Graph Drawing methods that are based on energy models [15,16] or spring-embedders [7,12], interesting directions are the ones which try to express the graph aesthetics via a differentiable function that can be optimized via Gradient Descent [2,29]. Machine learning applications have been used also to inject the user preferences into the layout optimization process. Such methods exploit evolutionary algorithms (e.g. genetic algorithms) to learn user preferences [3,4] keeping the human interaction in loop – causing however an inherent dependence on the user. Lately, given the increasing successes of Deep Learning models in several research fields, several works applied these architectures into the field of Graph Drawing. Neural networks are capable to learn the layout characteristics from graph drawing techniques [25] or from the layout distribution itself [17]. A recent contribution from Tiezzi et al. [21] introduced the general framework of Graph Neural Drawers (GNDs), where the main intuition is that neural networks (i.e., *Neural Aesthetes*) can be used both to express in a differentiable-manner the cross-intersection among arcs and to use this information as a loss that can be optimized via Graph Neural Networks (GNNs) [20,22,23]. In particular, the pre-trained *Neural Aesthete* exploited in the first step predicts if any two edges cross. Then, since neural networks outputs are differentiable, the well-trained edge crossing predictor serves as a guide to gradient descent steps to move the node towards the direction of non-intersection.

We build on this principles, introducing a novel method to obtain better and more pleasing layouts where the number of intersections is reduced. We propose to use a *Neural Aesthete* to devise if two arcs are crossing, and, if that is the case, we split one of the arc introducing a linear spline. The obtained control points of the spline can be threated as additional nodes of the graph, what we refer to as a *fake vertex*, that is added to the layout optimization process. This idea gives

further freedom to the layout generation dynamics, with the final goal to move the newly obtained segments (i.e., the fake node coordinates) into direction where the number intersection is highly reduced. We depict this intuition in Fig. 1. We prove that this approach can be further extended not only in optimizing the number of intersecting arcs, but also with other relevant layout losses, such as the *stress function* [29]. Our experimental findings show that our proof-of-concept is a viable solution for several graphs. An identifiable decrease in number of cross intersection of arcs is observed, overal several input graphs we tested, after the concept of linear splines is introduced into the optimization process.

The paper is organized as follows. Section 2 introduces some references on the Graph Drawing literature. Section 3 describe the concepts of Neural Aesthete and how to use it to introduce linear splines into the graph layout. Section 4 describes our proof-of-concept and the obtained experimental findings. Conclusions are drawn in Sect. 5.

2 Related Work

Literature provides a large variety of research methods aimed at improving graph readability. Most of these algorithms are designed to optimize a single aesthetic criteria. The main directions, depicted in the relevant survey by Gibson et al. [13] (i.e. force-directed, dimension reduction, multi-level techniques), involve the generation of interesting and aesthetically pleasing layouts by methods which regard a graph as a physical system, with forces acting on nodes with attraction and repulsion dynamics up to a stable equilibrium state [16]. For instance, a classic layout criterion is brought by the minimization of the *stress* function [16], where the node positions are computed in such a way that the actual distance between node pairs gets proportional to their graph theoretical distance. A very effective approach that have been proved to enhance the human understanding of the layouts and graph topologies, consists in reducing the number of cross intersections in edges [19]. However, the computational complexity of the problem is NP-hard, and several authors proposed complex solutions and algorithms to address this problem [1].

Recently, several works moved towards the direction of improving multiple layout criteria at once. Wang et al. [26] propose a revised formulation of stress that can be used to specify ideal edge direction in addition to ideal edge lengths. Devkota et al. [9] also use a stress-based approach to minimize edge crossings and maximize crossing angles. Eades et al. [11] provided a technique to draw large graphs while optimizing different geometric criteria, including the Gabriel graph property. Such approaches are capable to optimize multiple aesthetic criteria, but they cannot adapt naturally and dynamically to handle further optimization goals. Some recent contributions in this specific context presented the advantages of applying gradient based methodologies in graph drawing tasks. Zheng et al. [28] efficiently applied the Sthocastic Gradient Descent (SGD) method to reduce the stress loss, displacing pairs of vertices following the direction of the gradient. The recent framework by Ahmed et al. [2], $(GD)^2$, leverages

Gradient Descent to optimize several readability criteria at once, as long as the criterion can be expressed by smooth functions. Indeed, thanks to the powerful auto-differentiation tools available in modern machine learning frameworks [18], several criteria such as ideal edge lengths, stress, node occlusion, angular resolution and many others can be optimized smoothly and with ease. Finally, we report some early attempts to leverage Deep Learning Models in the Graph Drawing scenario. Wang et al. [26] proposed a graph-based LSTM model able to learn and generalize the coordinates patterns produced by other graph drawing techniques on certain graph layouts. Very recently, in DeepGD [24] a message-passing Graph Neural Network (GNN) is leveraged to process starting positions produced by graph drawing frameworks [6] to construct pleasing layouts that minimize combinations of aesthetic losses (stress loss combined with others) on arbitrary graphs. The Graph Neural Drawer framework [21] focus on the criteria of edge crossing, proving that a Neural Network (regarded as a *Neural Aesthete*) can learn to identify if two arcs are crossing or not. This simple model provides a useful and flexible gradient direction that can be exploited by (Stochastic) Gradient Descent methods. Moreover, the GND framework proved that GNNs, even in the non-attributed graph scenario, if enriched with appropriate node positional features, can be used to process the topology of the input graph with the purpose of mapping the obtained node representation in a 2D layout minimizing several provided loss functions. We built on the Neural Aesthete component of the framework, to introduce the concept of *fake nodes*, the linear splines control points, that can be optimized via the same underlying optimization process of the graph layout generation. Our contribution can be injected into heterogeneous optimization schemes, in principle involving multiple criteria at once. The idea of handling graph arcs as curves can be found in the very recent work by Yu et al. [27], where a reformulation of gradient descent based on a Sobolev-Slobodeckij inner product enables rapid progress toward local minima of loss functions.

3 Method

We denote a graph as G = (V, E), where V = $\{v_1, ..., v_n\}$ is a finite set of N nodes and $E \subseteq V \times V$ is the edge set representing the arcs connecting the nodes. We further represent each vertex $v_i \in V$ with coordinates $p_i : \mathbb{V} \longmapsto \mathbb{R}^2$, mapping the ith node to a bi-dimensional space. The node coordinate matrix is denoted as $\mathbf{P} \in \mathbb{R}^{N \times 2}$. Finally, the neighborhood of node ith is denoted by N_i.

In literature, a number of graph drawing algorithms have been devised, optimizing functions that express aesthetic index with the advantage of graph topology, geometry, visualization and theory concepts in bi-dimensional or tri-dimensional space [5]. The most typical aesthetic criteria take into account the uniformity of vertex allocation [13], the degree of edge intersections, [19], or the angles between crossing or adjacent edges. It is commonly assumed that graph drawing consists of finding the best vertex allocation when given the adjacency matrix. The latter drives the arcs drawing by connecting linked vertices with segments. In this paper, however, we propose to employ non-uniform splines

instead of segments to enhance the readability of the graph. For the sake of simplicity, we restrict our work to linear splines computed by means of *fake vertices* (control points). This allows to improve the readability of graph drawn when employing different GD-based methods, such as the GND or the Stress optimization. More precisely, we propose to introduce splines whenever the GD algorithm is not capable of accomplishing a certain aesthetic criterion while just employing segments. The ratio behind this idea is that the higher degree of freedom available to the optimization algorithm should allow achieving the required aesthetic criterion. The latter is measured by means of the Neural Aesthete, a neural network based model trained to measure the fulfillment of any (even non-differentiable) aesthetic criterion, as explained in Sect. 3.1. In Sect. 3.2, we better clarify how splines are introduced. In Sect. 3.3 we show how the same Neural Aethete provides a loss function that can be employed for Graph Drawing (as shown [21]). Finally, in Sect. 3.4 we show how we can also pair it with other standard GD-based methods, such as Stress optimization.

3.1 Learning Non-differentiable Aesthetic Criteria: The Neural Aesthete

As previously proposed in [21], we employ here the Neural Aesthete, a neural network-based model, capable of learning - and therefore measuring - any aesthetic criteria. Standard GD methods, indeed, are limited by the requirement of employing a differentiable aesthetic criterion to draw a graph. On the contrary, the Neural Aesthete is capable of learning also non-differentiable criteria such as the edge intersections. As done in [21] and without loss of generality, in this paper we train a neural network to measure edge intersection.

To train the Neural Aesthete model, we provide in input a couple of arcs $x = (e_u, e_v)$ and the model return the probability that the arcs are intersecting or not. Each arc is in turn defined as the couple of vertex coordinates $e_u = (p_i, p_j)$, with $e_u \in \mathbf{E}$ and $p_i \in [0, 1]^2$. The model $\nu(\cdot, \cdot, \cdot) : \mathbf{E}^2 \times \mathbb{R}^m \rightarrow R$ is working on the two arcs e_u and e_v and returns the output as

$$y_{e_u,e_v} = \nu(\theta, e_u, e_v) \tag{1}$$

where $\theta \in \mathbb{R}^m$ is the vector which represents the weights of the neural network. The learning of the model is based on optimization of a cross-entropy loss function $L(y_{e_u,e_v}, \hat{y}_{e_u,e_v})$, which is defined as:

$$L(y_{e_u,e_v}, \hat{y}_{e_u,e_v}) = -(\hat{y}_{e_u,e_v} * \log(y_{e_u,e_v}) + (1 - \hat{y}_{e_u,e_v}) * \log(1 - y_{e_u,e_v})), \tag{2}$$

where y_{e_u,e_v} is the target label:

$$y_{e_u,e_v} = \begin{cases} 1, & \text{if } (e_u, e_v) \text{ do intersect} \\ 0, & \text{otherwise} \end{cases} \tag{3}$$

Notice that the label y is computed by solving the system of equations of the lines passing by the vertices of the arcs. The solution of this system, if exists,

however, is either 0 or 1, therefore it cannot be directly optimized through gradient descent. An artificial dataset composed of 100k input-target entries (x, y) is employed for training the Neural Aesthete. The coordinates $p_{i,j}$ of the vertices of the input arcs are randomly chosen in the interval $[0, 1]^2$. The dataset is balanced to have samples for both classes (cross/no cross). The model is implemented as a MultiLayer Perceptron (MLP) composed of 2 hidden layers, each of 100 nodes with ReLu activation functions. The model is trained to minimize the cross entropy loss in terms of target values, taking maximum advantage through Adam optimizer. A dataset of 50k entries is taken to test the generalization capabilities of the model and achieved an accuracy of 97%.

At test time, the learning process yields a model capable to calculate the probability of intersection between any 2 given arcs. The Neural Aesthete is therefore capable to provide a differential smooth function that guides the positioning of node coordinates via gradient descent. More in details, the provided loss function is the cross-entropy towards respecting the given aesthetic critera - i.e., in this case, towards non-intersection. The optimization parameters, however, are not anymore the weights of the neural networks, but the coordinates of the vertices of the intersecting arcs $(e_u, e_v) = (p_i, p_j), (p_h, p_k)$ in input to the model.

3.2 Employing Splines to Improve Graph Readability

The major contribution of this research is the introduction of splines to connect the arcs which are predicted to not satisfy a certain aesthetic criterion. To predict the satisfaction degree, the previously trained Neural Aesthete model predicting edge intersection is employed. To introduce splines, we add a *fake vertex* on one of the intersecting arcs. Consequently, the arc will be optimized by employing more nodes instead of only the two extremities, as it is split and considered as two separated arcs. At each iteration, a batch of random arcs are chosen as input and the trained Neural Aesthete outputs the degree of intersection of the given arcs. Whenever two arcs are predicted as intersecting, a fake node is generated to further help the optimization process and reduce the probability of intersection. Since we want to create a spline, we add two arcs to the adjacency matrix connecting the new node to the previous ones, and we remove the old arc. The newly introduced spline will help the optimization process and by enhancing the freedom of the layout generation process, resulting in more optimized results. The Graph Drawing process is then conducted on all the vertices. For the sake of simplicity, and to avoid degenerated solutions, we avoid introducing splines during the first T iterations. Also, we limit to S_{max} the number of splits for each starting arc in the input graph.

Going into more details, the Neural Aesthete v process a random arc pair (e_u, e_v) picked from the arc list E, to predict their degree of intersection $\hat{y}_{e_u,e_v} = \nu(\theta, e_u, e_v)$. If the two arcs are intersecting each other, the first arc is split with the creation of a *fake vertex* $p_f = \frac{p_i+p_j}{2}$. The arc list E is updated, and the node coordinates of the created vertex are added to the parameters of the optimization process. In case the arcs created when introducing the spline

are randomly chosen, the coordinates of the new vertex is also processed. This is due to the fact that we want to further optimize the spline. This helps in optimizing edges while using more points and possibly by further splitting the arcs.

3.3 Edge Crossing Optimization with Splines

As previously introduced, the same Neural Aesthete can be employed in a gradient descent-based optimization process to display an input graph. The employed loss function $L(\cdot, \cdot)$ is the cross entropy loss introduced in Eq. 2 computed on the randomly chosen arcs e_u, e_v with respect to the non-intersection $y_{e_u//e_v} = 0$.

$$H_{u,v} = L(\hat{y}_{e_u,e_v}, y_{e_u//e_v}) = -\log(1 - \hat{y}_{e_u,e_v}) \tag{4}$$

This smooth and differential loss function increases the utilization of gradient descent methods to optimize edge coordinates (e_u, e_v) with respect to any learned aesthetic criterion. This is process is replicated for all graph edges,

$$H(\mathbf{P}) = \sum_{(e_u,e_v)\in\mathbf{E}} L(\hat{y}_{e_u,e_v}, y_{e_u//e_v}) \tag{5}$$

A possible scheme for graph drawing is then:

$$\mathbf{P}^* = \arg\min_{\mathbf{P}} H(\mathbf{P}) \tag{6}$$

A feasible solution to carry out this is by using optimization methods as gradient descent:

$$\mathbf{P} \longleftarrow \mathbf{P} - \eta \bigtriangledown_{\mathbf{P}} H(\mathbf{P}) \tag{7}$$

where η represents the learning rate.

The remarkable progress given by this framework is the computational efficiency and parallelization capabilities of neural networks, since the prediction of edge crossing can be carried out for multiple arc pairs in parallel. However, the optimization process is greatly facilitated when introducing splines, since both new arcs are moved in the directions where they are no more intersecting, participating in reducing the number of cross intersections and improving the graph layouts.

3.4 Stress Optimization with Splines

The Stress function [?], is one of the empirically proved techniques to be of great importance in drawing aesthetically pleasing graph layouts through pleasing node coordinate selection. The optimized loss function is defined as:

$$STRESS(P) = \sum_{n=1} w_{ij}(\parallel p_i - p_j \parallel - d_{ij})^2, \tag{8}$$

where $p_i, p_j \in [0; 1]^2$ are the coordinates of vertices i and j respectively. The graph theoretic distance d_{ij} is the shortest path between node i and j, and w_{ij}

is a normalization factor balancing the impact of the pairs, and it is defined as $w_{ij} = d_{ij}^{-\alpha}$ with $\alpha \in [0, 1, 2]$[1]. The graph drawing capabilities of stress optimization allows generating pleasing layouts even when employing segments to connect vertices. However, in this work, we also tested the employment of splines along with stress optimization. The steps exposed in Sect. 3.2 to generate splines are performed in this case as well.

We perform stress optimization on the vertices coordinates by minimizing the stress function (see Eq. 8). For each graph, we compute the shortest path d_{ij} among every node pair (i, j) and the loss is calculated based on the shortest path and the distance calculated for each node pair. The graph layouts clearly presents pleasing layouts as a result of the stress optimization. Further, a decreased number of cross intersection of arcs is observed in the graphs after splines are introduced in the model.

4 Experiments

We devised two different experimental settings to prove the effectiveness of our proposal. We compare the graph layouts produced by different combinations of loss functions, optimized by Gradient Descent. We implemented our method exploiting the PyTorch framework and we used the NetworkX [14] python package for the graph visualization utilities.

The main goal of the first experiment is the reduction of the number of arc intersections by injecting our spline-based method into an optimization process guided by the Neural Aesthete. Hence, the Neural Aesthete is exploited both to identify the crossing edges and to move the node coordinates (both the real and *fake* vertices, once they are introduced) towards the direction of non-intersection. In order to prove the efficiency of our method, we tested our approach over 9 heterogeneous graph classes (see Table 1), considering graphs with different characteristics and properties. Overall, we focused on graphs having small sizes, given that previous works highlighted how node-link layouts and cross-intersection based losses are more suitable to graphs with small size [25].

We compare the number of arc intersections produced, at the end of the layout optimization process, by optimizing solely the Edge-crossing (EC) loss provided by the Neural Aesthete, with the case in which we combine this optimization process with the proposed spline-based method (EC+SPLINES). We carried on the optimization via mini batches composed by 2 arc-couples, for an amount of $10k$ iterations (gradient steps), with a learning rate $\eta = 10^{-2}$. The initial node coordinates are randomly picked in the interval $(0, 1)$. We provide in Table 1 a quantitative comparison of these two approaches in the considered graph classes.

[1] We tested our work with stress optimization of graphs using different weighing factor α. However, the best graph layouts were empirically observed with $\alpha = 2$.

Table 1. Cross intersection reduction. We tested the proposed approach in 9 different graph classes (first column). We report the number of arc intersections at the end of the drawing optimization process, when minimizing the edge-crossing loss from the Neural Aesthete only (EC column), and when combining it with the proposed spline-based method (EC+SPLINES column).

Graph Class	EC	EC+SPLINES
Karate	142	118
Circular	25	18
Cube	3	0
Tree	3	2
Diamond	0	0
Bull	0	0
Simple	1	1
Cycle	1	1
Barbell	22	22

Thanks to the adoption of the proposed method, the number of intersection is reduced in most of the considered settings. We remark that the introduction of *fake* vertices and the corresponding arcs hinders, in principle, the optimization process, given that the amount of edges and arc is increased. Nevertheless, in most of the cases, the Neural Aesthete is capable to produce improved layouts at the end of the learning process. We show in Fig. 2 qualitative examples of the produced layouts. First column depicts the node positions produced by the standard EC loss optimization. When injecting the EC+SPLINES loss, the number of intersections is reduced thanks to inherited improved freedom in node position selection.

To further show the general nature of our proposal, we injected the proposed *fake* vertices creation into an optimization process minimizing the stress loss. We tested this method over the same graph classes of the previous experimental setup, with mini batches having the size of half the arc numbers of the input graph. We optimized the loss for an amount of $10k$ iterations (gradient steps). We set the learning rate $\eta = 10^{-1}$. We report in the third and fourth column of Fig. 2 the layouts obtained via optimizing the stress loss solely (STRESS column) and combining the stress loss with the proposed spline-based method (STRESS+SPLINES column). We can see how the obtained graphs still hold the visual characteristics specfi to the stress minimization, while reducing the number of intersecting arcs (evident in Diamond, Circular ladder).

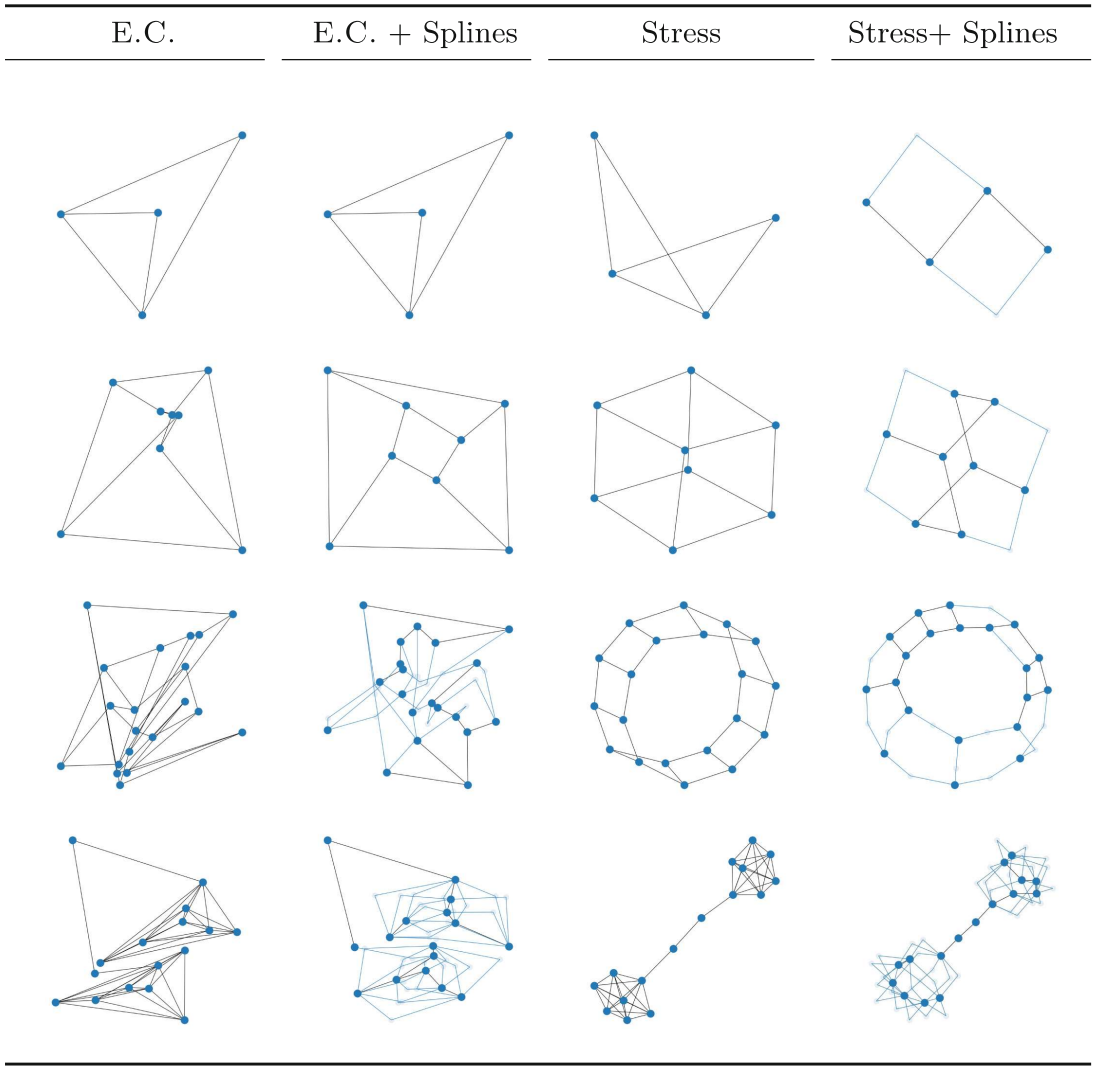

Fig. 2. Qualitative analysis on the obtained graph layouts. Different graph classes (rows: Diamond, Cube, Circular-Ladder and Barbell graphs) are plot using the 4 optimization paradigms (columns) investigated in the experimental experience.

5 Conclusion

We proposed a framework that improves graph layouts leveraging the tool of linear splines. We exploit a Neural Aesthete to devise if arcs are crossing, and if it is the case, we split them into spline-based segments. We show how this approach can be injected into a common Graph Drawing scenario based on gradient Descent optimization, both when optimizing the number of arcs intersections and common aesthetic losses such as the stress function. Future work will be focused on extending the proposed solution into Graph Neural Drawers powered by Graph Neural Networks, in order to speed up the graph drawing pipeline and the optimization processing time.

References

1. Ábrego, B.M., Fernández-Merchant, S., Salazar, G.: The rectilinear crossing number of k_n: closing in (or are we?). In: Pach, J. (ed.) Thirty Essays on Geometric Graph Theory, pp. 5–18. Springer, New York (2013). https://doi.org/10.1007/978-1-4614-0110-0_2
2. Ahmed, R., De Luca, F., Devkota, S., Kobourov, S., Li, M.: Graph drawing via gradient descent, $(gd)^2$ (2020). https://doi.org/10.48550/ARXIV.2008.05584, https://arxiv.org/abs/2008.05584
3. Bach, B., Spritzer, A., Lutton, E., Fekete, J.D.: Interactive random graph generation with evolutionary algorithms. In: Didimo, Walter, Patrignani, Maurizio (eds.) GD 2012. LNCS, vol. 7704, pp. 541–552. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-36763-2_48, https://hal.inria.fr/hal-00720161
4. Barbosa, H.J.C., Barreto, A.M.S.: An interactive genetic algorithm with co-evolution of weights for multiobjective problems. In: Proceedings of the 3rd Annual Conference on Genetic and Evolutionary Computation, pp. 203–210. GECCO'01, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (2001)
5. Battista, G.D., Eades, P., Tamassia, R., Tollis, I.G.: Graph Drawing: Algorithms for the Visualization of Graphs. Prentice Hall PTR, Hoboken (1998)
6. Brandes, U., Pich, C.: Eigensolver methods for progressive multidimensional scaling of large data. In: Kaufmann, M., Wagner, D. (eds.) GD 2006. LNCS, vol. 4372, pp. 42–53. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-70904-6_6
7. Brandes, U., Pich, C.: An experimental study on distance-based graph drawing. In: Tollis, I.G., Patrignani, M. (eds.) GD 2008. LNCS, vol. 5417, pp. 218–229. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-00219-9_21
8. Bronstein, M.M., Bruna, J., Cohen, T., Veličković, P.: Geometric deep learning: Grids, groups, graphs, geodesics, and gauges. arXiv preprint. arXiv:2104.13478 (2021)
9. Devkota, S., Ahmed, R., De Luca, F., Isaacs, K.E., Kobourov, S.: Stress-plus-x (SPX) graph layout. In: Archambault, D., Tóth, C.D. (eds.) GD 2019. LNCS, vol. 11904, pp. 291–304. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-35802-0_23
10. Didimo, W., Liotta, G., Montecchiani, F.: A survey on graph drawing beyond planarity. ACM Comput. Surv. (CSUR) **52**(1), 1–37 (2019)
11. Eades, P., Hong, S.-H., Klein, K., Nguyen, A.: Shape-based quality metrics for large graph visualization. In: Di Giacomo, E., Lubiw, A. (eds.) GD 2015. LNCS, vol. 9411, pp. 502–514. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-27261-0_41
12. Frick, A., Ludwig, A., Mehldau, H.: A fast adaptive layout algorithm for undirected graphs (extended abstract and system demonstration). In: Tamassia, R., Tollis, I.G. (eds.) GD 1994. LNCS, vol. 894, pp. 388–403. Springer, Heidelberg (1995). https://doi.org/10.1007/3-540-58950-3_393
13. Gibson, H., Faith, J., Vickers, P.: A survey of two-dimensional graph layout techniques for information visualisation. Inf. Vis. **12**(3–4), 324–357 (2013)
14. Hagberg, A., Swart, P., S Chult, D.: Exploring network structure, dynamics, and function using networkx. Technical report, Los Alamos National Lab. (LANL), Los Alamos, NM (United States) (2008)
15. Jacomy, M., Venturini, T., Heymann, S., Bastian, M.: Forceatlas2, a continuous graph layout algorithm for handy network visualization designed for the gephi software. PLoS ONE **9**(6), e98679 (2014)

16. Kamada, T., Kawai, S., et al.: An algorithm for drawing general undirected graphs. Inf. Process. Lett. **31**(1), 7–15 (1989)
17. Kwon, O.H., Ma, K.L.: A deep generative model for graph layout. IEEE Trans. Visual Comput. Graphics **26**(1), 665–675 (2019)
18. Paszke, A., et al.: Automatic differentiation in pytorch (2017)
19. Purchase, H.: Which aesthetic has the greatest effect on human understanding? In: DiBattista, G. (ed.) GD 1997. LNCS, vol. 1353, pp. 248–261. Springer, Heidelberg (1997). https://doi.org/10.1007/3-540-63938-1_67
20. Scarselli, F., Gori, M., Tsoi, A.C., Hagenbuchner, M., Monfardini, G.: The graph neural network model. IEEE Trans. Neural Netw. **20**(1), 61–80 (2009)
21. Tiezzi, M., Ciravegna, G., Gori, M.: Graph neural networks for graph drawing. In: IEEE Transactions on Neural Networks and Learning Systems, pp. 1–14 (2022). https://doi.org/10.1109/TNNLS.2022.3184967
22. Tiezzi, M., Marra, G., Melacci, S., Maggini, M.: Deep constraint-based propagation in graph neural networks. In: IEEE Transactions on Pattern Analysis and Machine Intelligence (2021)
23. Tiezzi, M., Marra, G., Melacci, S., Maggini, M., Gori, M.: A lagrangian approach to information propagation in graph neural networks. In: ECAI 2020–24th European Conference on Artificial Intelligence. Frontiers in Artificial Intelligence and Applications, vol. 325, pp. 1539–1546. IOS Press (2020). https://doi.org/10.3233/FAIA200262
24. Wang, X., Yen, K., Hu, Y., Shen, H.: Deepgd: a deep learning framework for graph drawing using GNN. CoRR abs/2106.15347 (2021), https://arxiv.org/abs/2106.15347
25. Wang, Y., Jin, Z., Wang, Q., Cui, W., Ma, T., Qu, H.: Deepdrawing: a deep learning approach to graph drawing. IEEE Trans. Visual Comput. Graph. **26**(1), 676–686 (2019)
26. Wang, Y., et al.: Revisiting stress majorization as a unified framework for interactive constrained graph visualization. IEEE Trans. Visual Comput. Graph. **24**(1), 489–499 (2017)
27. Yu, C., Schumacher, H., Crane, K.: Repulsive curves. ACM Trans. Graph. (TOG) **40**(2), 1–21 (2021)
28. Zheng, J.X., Goodman, D.F.M., Pawar, S.: Graph drawing by weighted constraint relaxation. CoRR abs/1710.04626 (2017), http://arxiv.org/abs/1710.04626
29. Zheng, J.X., Pawar, S., Goodman, D.F.: Graph drawing by stochastic gradient descent. IEEE Trans. Visual Comput. Graph. **25**(9), 2738–2748 (2018)