Graph Neural Networks for Graph Drawing

Matteo Tiezzi[1], Gabriele Ciravegna[2,3], and Marco Gori[1,3]

[1] SAILAB, University of Siena, Siena, Italy
https://sailab.diism.unisi.it/
[2] Politecnico di Torino, Torino, Italy
[3] Université Cote d'Azur, Inria, 3IA, I3S, CNRS, Nice, France

Abstract. Graph Drawing techniques have been developed in the last few years with the purpose of producing aesthetically pleasing node-link layouts. Recently, the employment of differentiable loss functions has paved the road to the massive usage of Gradient Descent and related optimization algorithms. In this paper, we propose the novel framework of *Graph Neural Drawers* (GND), Graph Neural Networks (GNNs) whose learning process can be driven by any provided loss function, such as the ones commonly employed in Graph Drawing. We prove that this mechanism can be guided by loss functions computed by means of Feed-forward Neural Networks, on the basis of supervision hints that express beauty properties. We provide a proof-of-concept by constructing a loss function for the edge-crossing, and provide quantitative and qualitative comparisons among different GNN models working under the proposed framework. This method has been published in the IEEE Transactions on Neural Networks and Learning Systems journal, and it is available at doi.org/10.1109/TNNLS.2022.3184967.

Keywords: Graph Drawing · Graph Representation Learning · Graph Neural Drawers · Graph Neural Networks.

1 Introduction

Visualizing complex relations and interaction patterns among entities is a crucial task, given the increasing interest in structured data representations[13]. The Graph Drawing [6] literature aims at developing algorithmic techniques to construct drawings of graphs – i.e. mathematical structures capable to efficiently represent the aforementioned relational concepts with nodes and edges connecting them – for example via the node-link paradigm [49,55,22]. The readability of graph layouts can be evaluated following some aesthetic criteria such as the number of crossing edges, minimum crossing angles, edge length variance, etc. [3]. The final goal is to find suitable coordinates for the node positions, and this often requires to explicitly express and combine these criteria through complicated mathematical formulations [17]. Moreover, effective approaches such as energy-based models [33,35] or spring-embedders [25,11] require hands-on expertise and trial and error processes to achieve certain desired visual properties. Additionally, such methods define loss or energy functions that must be

optimized for each new graph to be drawn, often requiring to adapt algorithm-specific parameters. Lately, two interesting directions have emerged in the Graph Drawing community. The former one leverages the power of Gradient Descent to explore the manifold given by pre-defined loss functions or combinations of them. Stochastic Gradient Descent (SGD) can be used to move sub-samples of vertices couples to minimize spring-embedder losses [68], substituting complicated techniques such as Majorization [20]. This approach has been extended to arbitrary optimization goals and solved via Gradient Descent if the corresponding criterion can be expressed via smooth functions [3]. The latter novel direction consists in the exploitation of Deep Learning models. Indeed, Neural networks are capable to learn the layout characteristics from plots produced by other graph drawing techniques [10,60], as well as the underlying distribution of data [41]. Very recently, the node positions produced by graph drawing frameworks [10] have been used as an input to Graph Neural Networks (GNNs) [51,62] to produce pleasing layout that minimize combinations of aesthetic losses [59].

We propose a framework, *Graph Neural Drawers* (GND), which embraces both the aforementioned directions. We borrow the representational capability and computational efficiency of neural networks to prove that (1) differentiable loss functions guiding the common Graph Drawing pipeline can be provided directly by a neural network, a Neural Aesthete, even when the required aesthetic criteria cannot be directly optimized. We propose a proof-of-concept where we focus on the criteria of edge crossing, proving that a Neural Aesthete can learn to identify if two arcs are crossing or not and provide a differentiable loss function towards non-intersection that can be exploited by (Stochastic) Gradient Descent methods. Moreover, (2) we prove 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. We compare various commonly used GNN models [38,58,63], proving the flexibility of the proposed framework. In particular, GND is capable to draw graphs (*i*) from supervised coordinates, i.e. emulating Graph Drawing Packages, (*ii*) minimizing common aesthetic loss functions and, additionally, (*iii*) by descending towards the gradient direction provided by the Neural Aesthete.

2 Graph Drawing Algorithms

Graph drawing algorithms typically optimize functions that somehow express a sort of beauty index to derive a graphical visualization of the graph at hand in a bidimensional or tridimensional space [6,21,65]. Amongst others, typical beauty indexes are those of measuring the degree of edge crossings [48], avoiding small angles between adjacent or crossing edges or measures to express a degree of uniform allocation of the vertexes [29,3]. All these requirements inherently assume that the graph drawing only consists of the allocation of the vertexes in the layout space, since the adjacent matrix of the graph can drive the drawing of the arcs as segments. Without loss of generality, also in this work we restrict

our objective to the vertex coordinates optimization, but the basis ideas can be extended also to the case of appropriate arc drawing.

We denote a graph by $G = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = \{v_1, \ldots, v_N\}$ is a finite set of N *nodes* and $E \subseteq V \times V$ collects the *arcs* connecting them. The neighborhood of node v_i is denoted by $\mathcal{N}_i$. We denote the coordinates of each vertex with $p_i : \mathcal{V} \mapsto \mathbb{R}^2$, for a node i mapped to a bi-dimensional space. We denote with $P \in \mathbb{R}^{N \times 2}$ the matrix of the node coordinates.

One of the techniques that empirically proved to be very effective for an aesthetically pleasing node coordinates selection is the *Stress* function [35],

$$\text{STRESS}(P) = \sum_{i<j} w_{ij} \big(||p_i - p_j|| - d_{ij} \big)^2 \tag{1}$$

where p_i, p_j are the coordinates of vertices i and j, respectively, d_{ij} is the graph theoretic distance (or shortest-path) between node i and j, and w_{ij} is a weighting factor leveraged to balance the influence of certain pairs given their theoretical distance. Usually, it is defined as $w_{ij} = d_{ij}^{-\alpha}$ with $\alpha \in [0, 1, 2]$. The optimization of this function is generally carried out leveraging complicated resolution methods (i.e., 2D Newton-Raphson method, Stress Majorization, etc.) that hinder its efficiency.

Recently, Gradient Descent methods were employed to produce graph layouts [68] by minimizing the Stress function, and noticeably, Ahmed et al. [3] proposed a similar approach employing auto-differentiation tools. The advantage of this solution is that, as long as aesthetic criteria are characterized by smooth differentiable functions, it is possible to undergo an iterative optimization process on the node coordinates following, at each variable update step, the gradient of the criteria.

Clearly, the definition of aesthetic criteria as smooth functions could be hard to express. For instance, while we can easily count the number of arc intersections, devising a smooth function that may drive a continuous optimization of this problem is not trivial [52,3]. Indeed, finding the intersection of two lines, is as simple as solving the following equation system:

$$\begin{cases} a_1 x + b_1 y + c_1 = 0, \\ a_2 x + b_2 y + c_2 = 0 \end{cases} \tag{2}$$

By employing the classic Cramer's rule we can see that there is an intersection only in case of a non-negative determinant of the coefficient matrix $A, Det(A) = a_1 b_2 - a_2 b_1 \neq 0$. Clearly, the previous formula cannot be employed as a loss function in an optimization problem since it does not provide gradients.

3 The Neural Aesthete

To tackle this issue and provide a scoring function optimizable via gradient descent, we propose the *Neural Aesthete*, a neural network that learns beauty from examples with the perspective of generalizing to unseen data. The obtained

modelled function that is expressed by the Neural Aesthete is smooth and differentiable by definition and offers a fundamental heuristic for graph drawing. As a proof-of-concept, we focus on edge crossing. In this case, we define the Neural Aesthete as a machine which processes two arcs as inputs and returns the information on whether or not they intersect one each other. Each arc is identified by the coordinates of the corresponding pair of vertices, $e_u = (p_i, p_j)$ for $e_u \in \mathcal{E}$. Hence, the Neural Aesthete $\nu(\cdot,\cdot,\cdot) : \mathcal{E}^2 \times \mathbb{R}^m \rightarrow \mathbb{R}$ operates on the concatenation of two arcs, e_u and e_v and returns:

$$y_{e_u,e_v} = \nu\big(\theta, e_u, e_v\big), \tag{3}$$

where $\theta \in \mathbb{R}^m$ is the vector which represents the weights of the neural network. The Neural Aesthete is learned by optimizing a cross-entropy loss function $L(y_{e_u,e_v}, \hat{y}_{e_u,e_v})$ over the arcs $(e_u, e_v) \in \mathcal{E}$, which is defined as:

$$L(y_{e_u,e_v}, \hat{y}_{e_u,e_v}) = -\big(\hat{y}_{e_u,e_v} \cdot log(y_{e_u,e_v}) + (1 - \hat{y}_{e_u,e_v}) \cdot log(1 - y_{e_u,e_v})\big), \tag{4}$$

where $\hat{y}_{e_u,e_v}$ is the target and it is $\hat{y}_{e_u,e_v} = 1$ only if (e_u, e_v) intersect[4]. Notice that the learning process from a finite set of supervised examples yields weights that allows us to estimate the probability of intersection of any two arcs. Basically, the learned output of the neural network can be regarded as a degree of intersection between any arc couple. Once learned, this characteristic of the Neural Aesthete comes in handy for the computation of the gradient of a loss function for Graph Drawing. In general, we want to move the extreme nodes defining the two arcs towards the direction of *non-intersection*.

Hence, for the Graph Drawing task, the Neural Aesthete is able to process an unseen edge couple (e_u, e_v) randomly picked from the edge list $\mathcal{E}$, and to predict their degree of intersection y_{e_u,e_v}. We define the loss function $L(\cdot,\cdot)$ on this edge pair as the cross-entropy with respect to the target *no-intersection*, $\hat{y}_{e_u//e_v} = 0$

$$H_{u,v} = L(y_{e_u,e_v}, \hat{y}_{e_u//e_v}) = -log(1 - y_{e_u,e_v}) \tag{5}$$

This smooth and differential loss function foster the utilization of Gradient Descent methods to optimize the problem variables, i.e. the arc node coordinates (e_u, e_v). This same procedure can be replicated to all the graph edges. Some qualitative results obtained when optimizing over all the graph edges the Neural Aesthete Loss (alone and combined to the Stress loss) are reported in Appendix C.1.

4 GNNs for Graph Drawing

The increasing adoption of GNNs in several research fields and practical tasks [34,67] opens the road to an even wider spectrum of problems. DeepDrawing [60] employs a graph-based LSTM model to learn node layout styles from Graph Drawing frameworks. DeepGD [59] is a concurrent work in which a Message

[4] The intersection of e_u, e_v is automatically computed, e.g., by solving Eq. 2.

Passing Neural Network (MPNN) processes node positions to develop pleasing layouts that minimize combinations of aesthetic losses. Starting node positions, however, needs to be initialized by standard graph drawing frameworks [10]; in case a random initialization is employed, network performances deteriorate. Approaches like [62], instead, introduce additional virtual connections in the graph topology, thus increasing the computational burden for the model (given that the computational complexity of the GNN propagation is linear in the number of edges). Further analysis of the related work on graph drawing as well as a detailed report on several GNNs-based solutions proposed in the literature is reported in Appendix A.

In this paper, instead, we propose an approach to Graph Drawing, GNDs, that directly leverages the computational efficiency of GNNs and, thanks to informative nodal features (Laplacian eigenvectors, see the following), is general enough to be applied to several learning tasks.

Problem Formulation We formulate the problem as a *node-focused* regression task, in which for each vertex belonging to the input graph we want to infer its coordinates in a bi-dimensional plane, conditioned on the graph topology and the target layout/loss function. Furthermore, in the GND framework, we propose to employ GNNs to learn to draw by themselves following the guidelines prescribed by Neural Aesthetes (see Section 4.3). In order to be able to properly solve the Graph Drawing task via GNDs, a crucial role is played by the expressive power of the GNN model and the nodal features which are used. In fact, in line with the aforementioned regression task, each node state must be uniquely identified to be afterwards mapped to a different 2D position in the graph layout. This problem is inherently connected with recent studies on the representational capabilities of GNNs (see Appendix A and [66]). Standard MP-GNNs have been proved to be less powerful than the 1-WL test [45], both due to the lack of expressive power of the used aggregation mechanisms and to the existence of symmetries inside the graph. For instance, local isomorphic neighborhoods create indiscernible unfolding of the GNN computational structure. Hence, the GNN embeds isomorphic nodes to the same point in the high dimensional space of the states, hindering the Graph Drawing task. Some approaches address this problem proposing novel and more powerful architectures (WL-GNNs) that, however, tend to penalise the computational efficiency of the GNNs [45]. Moreover, given the fact that we focus on the task of drawing non-attributed graphs, it is even more important to enrich the nodes with powerful features able to identify both the position of nodes inside the graph (often referred to as Positional Encodings (PEs)[23]) and the neighboring structure.

To address this issue, we keep standard GNN architectures and leverage Positional features defined as the Laplacian eigenvectors [8] of the input graph, as introduced recently in GNNs [23]. Laplacian eigenvectors embed the graphs into the Euclidean space through a spectral technique, are unique and distance-preserving (far away nodes on the graph have large PE distance). Indeed, they can be considered hybrid positional-structural encodings, as they both define a local coordinate system and preserve the global graph structure.

Formally, they are defined via the factorization of the graph Laplacian matrix:

$$L = \mathrm{I} - D^{-1/2} A D^{-1/2} = U^T \Lambda U, \tag{6}$$

where I is the $N \times N$ identity matrix, D is the node degree matrix, A is the adjacency matrix and Λ and U correspond respectively to the eigenvalues and eigenvectors. As proposed in [23], we use the k smallest non-trivial eigenvectors to generate a k-dimensional feature vector for each node, where k is chosen by grid-search. Noticeably, given that the smallest eigenvectors provide smooth encoding coordinates of neighboring nodes, during the message exchange each node receives and implicit feedback on its own positional-structural characteristics from all the nodes with which it is communicating. This process foster the regression task on the node coordinates, which receives useful information from their respective neighborhood. We believe that this is a crucial component of the model pipeline.

Experimental setup We test the capabilities of the proposed framework comparing the performances of three commonly used GNN models (see Table 3). Given the fact that the outputs of GND are node coordinates, we can impose on such predictions heterogeneous loss functions that can be optimized via BackPropagation. In the proposed experiments, we test the GND performances on the loss functions defined as the following:

1. distance with respect to ground truth node coordinates belonging to certain layouts, produced by Graph Drawing packages (Section 4.1);
2. aesthetic loss functions (e.g. Stress) (Section 4.2);
3. loss functions provided by Neural Aesthetes (Section 4.3).

We assume to work with solely the graph topology, hence the node are not characterized by additional features. For further details regarding the experimental settings, see Appendix D.

4.1 GNNs learn to draw from ground-truth examples

The first experimental goal is focused on the task of learning to draw graph layouts given ground-truths node positions produced by Graph Drawing frameworks. Among several packages, we chose NetworkX [32] for its completeness and ease of integration with other development tools. This framework provides several utilities to plot graph appearances. We choose two different classical layouts. The first is the Kamada-Kawai node layout [35] computed by optimizing the Stress function. In few words, this force-directed method models the layout dynamic as springs between all pairs of vertices, with an ideal length equal to their graph-theoretic distance. The latter is the Spectral layout, which leverages the unnormalized Laplacian $\hat{L}$ and its eigenvalues to build cartesian coordinates for the nodes [7], formally $\hat{L} = D - A = \hat{U}^T \hat{\Lambda} \hat{U}$, where $\hat{\Lambda}$ and $\hat{U}$ correspond to the eigenvalues and eigenvectors, respectively, and using the first two non-trivial eigenvectors ($k = 2$) as the actual node coordinates. This layout highlights clusters of nodes in the graph.

Each training graph is enriched by Positional Encodings defined as k-dimensional Laplacian Eigenvectors and is processed by each of the tested GNN models to predict the node coordinates. Hence, we need a loss function capable to discern if the generated layout is similar to the corresponding ground truth. Furthermore, trained models should generalize the notion of graph layout beyond a simple one-to-one mapping. For these reasons, we leverage the Procrustes Statistic [60] as a loss function since it measures the shape difference among graph layouts independently of affine transformations such as translations, rotations and scaling. Given a graph composed of N nodes, the predicted node coordinates $P = (p_1, ..., p_N)$ and the ground-truth positions $\hat{P} = (\hat{p}_1, .., \hat{p}_N)$, the Procrustes Statistic similarity is defined as the squared sum of the distances between P and $\hat{P}$ after a series of possible affine transformations [60]. Formally:

$$R^2 = 1 - \frac{\left(\mathrm{Tr}\,(P^T \hat{P} \hat{P}^T P)^{\frac{1}{2}}\right)^2}{\mathrm{Tr}\,(P^T P)\mathrm{Tr}\,(\hat{P}^T \hat{P})} \tag{7}$$

where $\mathrm{Tr}(\cdot)$ denotes the trace operator and the obtained metric R^2 assumes values in the interval $[0, 1]$, the lower the better. We will use the Procrustes Statistic-based similarity both as the loss function to guide the model training, and to evaluate its generalization capability on the test set.

We tested the proposed framework comparing the test performances obtained by the three different GNN models (see Table 3 in Appendix), GCN, GAT, GIN. Also, we devised several competitors in order to assess the performances of the proposed approach. Given that Laplacian PEs available at node-level are powerful descriptors of the neighboring graph structure, we leverage a Multilayer Perceptron (MLP) as a baseline. This neural predictor learns a mapping to the node coordinates, solely exploiting the available local information. We compare the performances obtained by GNNs with Laplacian PEs against those achieved by the three corresponding variants of rGNNs (GNN employing randomized input features, see Appendix B for further details), which we denote with rGCN, rGAT, rGIN. For a fair comparison, we searched in the same hyperparameter space for all the baseline and competitors.

In Figure 1, we report a qualitative evaluation obtained by the best performing models for each different GNN architecture on two randomly picked graphs from the test set of the Rome dataset. More precisely, we report the evaluation in the case of Kamada-Kawai layout supervisions (first column depicts the Ground Truth (GT)). See Appendix E for additional results and considerations on the Spectral layout. The results show the good performances of the GND framework in generating two heterogeneous styles of graph layouts, learning from different ground truth node coordinates.

In order to give a more comprehensive analysis, we report in Table 1 a quantitative comparison among the global Procrustes Statistic similarity values obtained on the test set by the best models, for both the Rome dataset and on the Sparse dataset a synthetic, randomly generated dataset with sparse edge connections (further details on both datasets are reported in Appendix D). We report the average score and its standard deviation over three runs with different

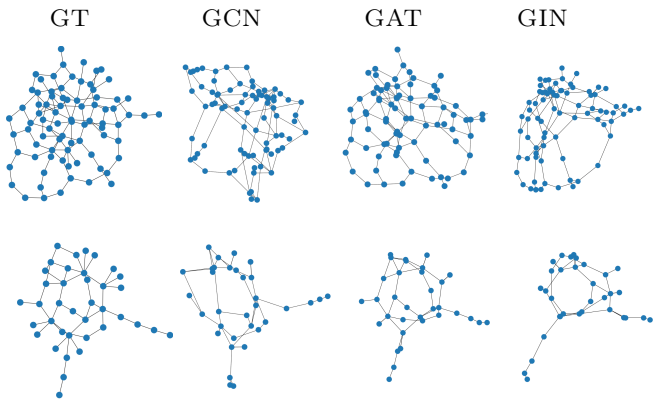

Fig. 1. Kamada-Kawai layout. Qualitative example of the predicted node coordinates for the Rome dataset. Each row depicts the Ground-Truth positions (GT), the graph layout produced by GCN, GAT, GIN model, on two randomly picked test graphs.

Table 1. Procrustes Statistic similarity (defined in Eq. 7) on the test split of the Rome and Sparse dataset. We compare three GNN models with two graph layouts generation, Kamada-Kawai and Spectral. We report the average valued and its standard deviation over three runs with different weights initialization.

Model	Rome		Sparse	
	Kamada	Spectral	Kamada	Spectral
MLP	0.291 ± 0.000	0.144 ± 0.000	0.282 ± 0.001	0.131 ± 0.000
rGCN	0.612 ± 0.009	0.527 ± 0.008	0.592 ± 0.006	0.532 ± 0.009
rGAT	0.475 ± 0.008	0.437 ± 0.008	0.465 ± 0.015	0.425 ± 0.017
rGIN	0.685 ± 0.001	0.590 ± 0.003	0.675 ± 0.010	0.603 ± 0.030
GCN	0.241 ± 0.001	0.078 ± 0.002	0.228 ± 0.000	0.060 ± 0.000
GAT	**0.186** ± 0.000	**0.057** ± 0.000	**0.177** ± 0.001	**0.045** ± 0.001
GIN	0.240 ± 0.002	0.076 ± 0.003	0.229 ± 0.003	0.059 ± 0.001

seeds. The strength of the Laplacian PE is validated by the decent performances yielded by the MLP baseline. Conversely, the random features characterizing the rGNNs are not sufficient to solve this node regression task. Some additional structural information is required in order to jointly represent the node position and its surroundings. Indeed, all the models exploiting the proposed solution outperform the competitors. Instead, the improved performances of GNNs employing Laplacian PE with respect to MLP are due to the fact that in GNNs nodes states receive implicit feedback on their own position during the message passing steps. The proposed GAT model with Laplacian PE achieves the best performances in all the settings. We believe that the attention mechanism plays a crucial role in the task of distinguishing the right propagation patterns, along-

side the fact that the multi-head attention mechanism provides a bigger number of learnable parameters with respect to the competitors.

4.2 GNNs learn to draw minimizing aesthetic loss functions

In Section 4.1, GNDs explicitly minimize the distances with respect to certain ground-truth node positions, hence learning to draw directly from data according to certain layouts. In this second experimental setting, instead, we build GNNs capable to draw at inference time respecting certain aesthetic criteria which are implicitly learnt during training following an unsupervised learning strategy. We defined our framework in such a way that powerful PE features are mapped to 2D coordinates. Given a smooth and differentiable loss function defined on such output, we can leverage the BP algorithm in order to learn to minimize heterogeneous criteria. We investigate the case in which the GNN models minimize the *Stress function* (see Eq. 1) on the predicted node coordinates. Only during the training phase, for each graph, we compute the shortest-path d_{ij} among every node couple (i, j). At inference time, the GND framework process the graph topology (the adjacency matrix) and the node features, directly predicting the node coordinates, without the need of any further information.

We use the same experimental setup, competitors and hyper-parameters selection grids of Section 4.1. However, according to a preliminary run of the models which achieved poor performances, we varied the hidden state dimension grid to $\{100, 200, 300\}$. This means that this task need a bigger representational capability with respect to the previous one, which is coherent with the complex implicit nature of the learning problem. We set the Stress normalization factor to $w_{ij} = \frac{1}{d_{ij}}$ (hence, $\alpha = 1$) and compute the averaged Stress functionFor this experiment, we use the stress value obtained on the validation split as the metric to select the best performing model. For comparison, we report the stress loss values obtained by three State-of-the-art Graph Drawing methods. We also compared against Neato[5], which leverage the stress majorization [27] algorithm to effectively minimize the stress, PivotMDS [10], which employs a deterministic dimension reduction approach, and ForceAtlas2 [33], which generates graph layouts through a force-directed method.

We report in Figure 2 some qualitative examples of the graph layouts produced by the best selected GNN models on test samples (the same graphs selected for Figure 1), following the aforementioned setting. Noticeably, all three models succeed in producing a layout that adheres to the typical characteristics of graphs obtained via Stress minimization. In particular, for reference on the drawing style, the layouts of these same graphs generated via the Kamada-Kawai algorithm (that also minimize stress) are depicted in the first column of Figure 1. Comparing the graph layout produced by the various GNN models and the aforementioned ones from Kamada-Kawai, also in this case it is easy to see from a qualitative analysis that the GAT model is the best performing one.

[5] Implementation available through Graphviz, https://graphviz.com

Table 2. Average Stress loss value obtained on the Training set and Test set by the best selected models, for each dataset. We report the mean and standard deviation obtained over three runs initialized with different fixed seeds. We do not report standard deviations for Neato and PivotMDS, being deterministic algorithms.

Model	Rome		Sparse	
	train loss	test loss	train loss	test loss
ForceAtl.2	27.44 ± 0.01	26.82 ± 0.02	23.31 ± 0.01	22.69 ± 0.02
Neato	4.35	4.34	5.61	5.66
Piv.MDS	16.40	16.65	28.93	29.27
MLP	1.07 ± 0.01	1.06 ± 0.03	1.16 ± 0.02	1.18 ± 0.00
rGCN	1.25 ± 0.01	1.24 ± 0.04	1.70 ±0.02	1.72 ±0.01
rGAT	0.98 ± 0.02	0.97 ± 0.00	1.34 ± 0.01	1.36 ± 0.01
rGIN	1.11 ± 0.03	1.49 ± 0.04	1.49 ±0.07	1.89 ± 0.02
GCN	0.51 ± 0.02	0.53 ± 0.03	0.56 ±0.05	0.61 ±0.02
GAT	**0.33** ± 0.02	**0.34** ± **0.00**	**0.30** ± **0.02**	**0.33** ± **0.01**
GIN	0.49 ± 0.04	0.88 ± 0.05	0.28 ±0.05	0.81 ± 0.01

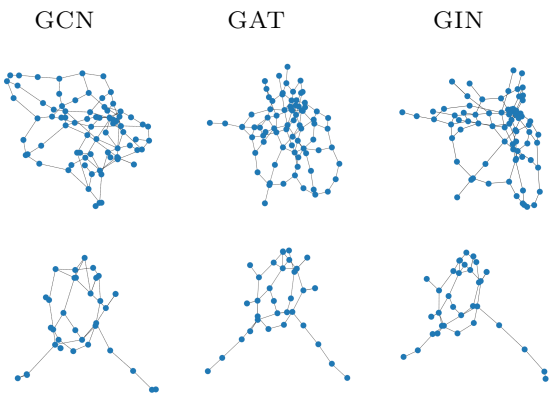

Fig. 2. Stress minimization on the Rome dataset. Qualitative example of the graph layout produced by three GNN models on the test graphs of the Rome dataset.

A quantitative comparison is reported in Table 2, over both the Rome and Sparse datasets. For comparison, we report the stress values for each competitor and dataset, both at training time and test time, averaged over three runs initialized with different seeds. Once again, GAT performs the best. The metrics obtained by the GIN model highlight an overfit of the training data, given the selected grid parameters. All GND models obtain better stress than the compared SOTA drawing packages, with Neato being the best performing one among them in terms of stress minimization, as expected. Similar conclusion with respect to the previous experiment can be drawn regarding the results obtained by rGNNs and MLP. Indeed, these results show how learning to minimize stress requires

both positional and structural knowledge, and that the message passing process foster the discriminative capability of the learned node states, with respect to solely exploiting local information.

4.3 GNNs learn to Draw from Neural Aesthetes

In the previous Section, we showed that GNDs are capable to learn to minimize a differentiable smooth function that implicitly guides the node coordinates positioning. In a similar way, the Neural Aesthete introduced in Section 3 provide a smooth differentiable function that can be leveraged to find a good gradient descent direction for the learning parameters. In this Section, we mix the two proposals in order to build a Graph Neural Drawer that learns to generate graph layouts thanks to the gradients provided by the edge-crossing Neural Aesthete, and, eventually, to optimize the combination of several aesthetic losses.

At each learning epoch, GND minimizes the loss function $H(P)$ defined in Eq. 10, over the whole edge list $\mathcal{E}$. The loss function can be computed as follows: the GNN model process the graph and predicts node-wise coordinates. Given such predicted node positions and the input graph adjacency matrix, the Neural Aesthete (trained beforehand as explained in Appendix C.1) processes couples of arcs and output their degree-of-intersection. The overall loss function can then be composed by the contribution given by each of the considered arc-couples, as in Eq. 5. We restrict our analysis to the Rome dataset and exploiting a GAT model with 2 hidden layers, a hidden size of node state of 25, PE dimension $k = 10$, learning rate $\eta = 10^{-2}$. We compare the graph layout generated by this model when employing three different loss functions: (i) stress loss, (ii) Neural-Aesthete edge-crossing based loss $H(P)$, (iii) a combination of the two losses with a weighing factor $\lambda = 0.5$ acting on the Neural Aesthete loss, i.e.: $\textsc{Loss}(\mathrm{P}) = \textsc{Stress}(P) + \lambda H(P)$.

We report in Figure 3 some qualitative results on two test graphs (one for each row). We compare the layout obtained optimizing the stress function (first column, see Section 4.2), the edge-crossing Neural Aesthete (second column) and the combination of the two losses. The styles of the generated layout are recognizable with respect to the plain optimization of the Neural Aesthete with Gradient Descent (see Figure 4), meaning that the GND framework is able to fit the loss provided by the Neural Aesthete and to generalize it to unseen graphs. Noticeable, the introduction of the combined loss functions (third column in Figure 3) helps in better differentiating the nodes in the graph with respect to the case of solely optimizing stress. The Neural Aesthete guided layouts (second and third column) tend to avoid edge intersections, as expected. This opens the road to further studies in this direction, leveraging the generality of the Neural Aesthetes approach and the representation capability of GNNs.

4.4 Computational Complexity

The proposed framework leverages the same computational structure of the underlying GNN model, which we can generally describe, for each parameter up-

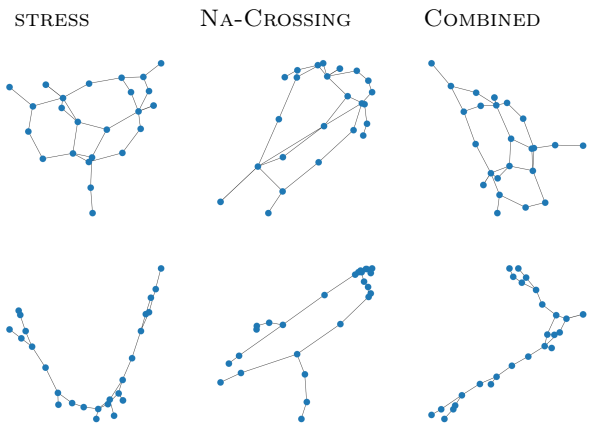

Fig. 3. LEARNING FROM THE NEURAL AESTHETE. We report the layouts obtained on two randomly picked test graphs from the Rome dataset. Left-to-right: Graph layout generated by optimizing the stress loss function, the edge-crossing Neural Aesthete based loss (denoted with NA-Crossing), and the combination of the two losses.

date, as linear with respect to the edge number $\mathcal{O}\big(T(|\mathcal{V}| + |\mathcal{E}|)\big)$, where T is the number of iterations/layers, $|\mathcal{V}|$ the number of nodes and $|\mathcal{E}|$ the number of edges. Through our approach, there is not any increase in the computation related to the graph topology or the edge connection patterns. At inference time, the only additional requirement is the computation of the Laplacian PEs, requiring $\mathcal{O}(\mathcal{E}^{3/2})$, with $\mathcal{E}$ being the number of edges, that however can be improved with the Nystrom method [24,23].

This is confirmed by the experiments that we conducted on bigger graphs (and reported in Appendix F) that show that our method is extremely more efficient than all SOTA method we compared against (as shown in Fig. 7), while still being able to draw aesthetically pleasing graphs (as shown in Fig. 8).

5 Conclusion

In this paper we proposed a general framework to emphasize the role of implicit learning criteria based on loss functions that enforce classic aesthetic measures. *Graph Neural Drawers* open the doors towards the construction of a novel machine learning-based drawing scheme where the *Neural Aesthete* drives the learning of a GNN towards the optimization of beauty indexes. While we have adopted the *Neural Aesthetes* only from learning to minimize arc intersections, the same idea can be used for nearly any beauty index. We show that our framework is effective also for drawing unlabelled graphs. In particular, we rely on the adoption of Laplacian Eigenvector-based positional features [23] for attaching information to the vertexes, which leads to very promising results.

References

1. Abboud, R., Ceylan, I.I., Grohe, M., Lukasiewicz, T.: The surprising power of graph neural networks with random node initialization. In: IJCAI (2021)
2. Ábrego, B.M., Fernández-Merchant, S., Salazar, G.: The rectilinear crossing number of k n: Closing in (or are we?). In: Thirty essays on geometric graph theory, pp. 5–18. Springer (2013)
3. Ahmed, R., De Luca, F., Devkota, S., Kobourov, S., Li, M.: Graph drawing via gradient descent, GD^2. arXiv preprint arXiv:2008.05584 (2020)
4. Bacciu, D., Errica, F., Micheli, A., Podda, M.: A gentle introduction to deep learning for graphs. Neural Networks **129**, 203–221 (2020)
5. Batagelj, V., Brandes, U.: Efficient generation of large random networks. Physical Review E **71**(3), 036113 (2005)
6. Battista, G.D., Eades, P., Tamassia, R., Tollis, I.G.: Graph drawing: algorithms for the visualization of graphs. Prentice Hall PTR (1998)
7. Beckman, B.: Theory of spectral graph layout. Tech. rep., Technical Report MSR-TR-94-04, Microsoft Research (1994)
8. Belkin, M., Niyogi, P.: Laplacian eigenmaps for dimensionality reduction and data representation. Neural computation **15**(6), 1373–1396 (2003)
9. Bodnar, C., Frasca, F., Otter, N., Wang, Y.G., Liò, P., Montúfar, G., Bronstein, M.: Weisfeiler and lehman go cellular: Cw networks. arXiv preprint arXiv:2106.12575 (2021)
10. Brandes, U., Pich, C.: Eigensolver methods for progressive multidimensional scaling of large data. In: GD 2006. pp. 42–53. Springer (2006)
11. Brandes, U., Pich, C.: An experimental study on distance-based graph drawing. In: GD 2008. pp. 218–229. Springer (2008)
12. Bresson, X., Laurent, T.: Residual gated graph convnets. CoRR **abs/1711.07553** (2017), http://arxiv.org/abs/1711.07553
13. Bronstein, M.M., Bruna, J., Cohen, T., Veličković, P.: Geometric deep learning: Grids, groups, graphs, geodesics, and gauges. arXiv preprint arXiv:2104.13478 (2021)
14. Bronstein, M.M., Bruna, J., LeCun, Y., Szlam, A., Vandergheynst, P.: Geometric deep learning: Going beyond euclidean data. IEEE Signal Processing Magazine **34**(4), 18–42 (2017)
15. Chrobak, M., Goodrich, M.T., Tamassia, R.: Convex drawings of graphs in two and three dimensions (preliminary version). In: Proceedings of the twelfth annual symposium on Computational geometry. pp. 319–328 (1996)
16. Corso, G., Cavalleri, L., Beaini, D., Liò, P., Veličković, P.: Principal neighbourhood aggregation for graph nets. In: NeurIPS. vol. 33, pp. 13260–13271 (2020)
17. Cox, M.A., Cox, T.F.: Multidimensional scaling. In: Handbook of data visualization, pp. 315–347. Springer (2008)
18. Cui, H., Lu, Z., Li, P., Yang, C.: On positional and structural node features for graph neural networks on non-attributed graphs. arXiv preprint arXiv:2107.01495 (2021)
19. Dai, H., Kozareva, Z., Dai, B., Smola, A.J., Song, L.: Learning steady-states of iterative algorithms over graphs. In: ICML. vol. 80, pp. 1114–1122 (2018)
20. De Leeuw, J.: Convergence of the majorization method for multidimensional scaling. Journal of classification **5**(2), 163–180 (1988)
21. Di Battista, G., Eades, P., Tamassia, R., Tollis, I.G.: Algorithms for drawing graphs: an annotated bibliography. Computational Geometry **4**(5), 235–282 (1994)

22. Didimo, W., Liotta, G., Montecchiani, F.: A survey on graph drawing beyond planarity. ACM CSUR **52**(1), 1–37 (2019)
23. Dwivedi, V.P., Joshi, C.K., Laurent, T., Bengio, Y., Bresson, X.: Benchmarking graph neural networks. arXiv preprint arXiv:2003.00982 (2020)
24. Fowlkes, C., Belongie, S., Chung, F., Malik, J.: Spectral grouping using the nystrom method. IEEE TPAMI **26**(2), 214–225 (2004)
25. Frick, A., Ludwig, A., Mehldau, H.: A fast adaptive layout algorithm for undirected graphs (extended abstract and system demonstration). In: GD '94. pp. 388–403. Springer (1994)
26. Fruchterman, T.M., Reingold, E.M.: Graph drawing by force-directed placement. Software: Practice and experience **21**(11), 1129–1164 (1991)
27. Gansner, E.R., Koren, Y., North, S.: Graph drawing by stress majorization. In: GD2004. pp. 239–250. Springer (2004)
28. Ghoniem, M., Fekete, J.D., Castagliola, P.: A comparison of the readability of graphs using node-link and matrix-based representations. In: IEEE INFOVIS. pp. 17–24. Ieee (2004)
29. Gibson, H., Faith, J., Vickers, P.: A survey of two-dimensional graph layout techniques for information visualisation. Information visualization **12**(3-4), 324–357 (2013)
30. Gilmer, J., Schoenholz, S., Riley, P., Vinyals, O., George, D.: Message passing neural networks. In: Machine Learning Meets Quantum Physics, pp. 199–214. Springer (2020)
31. Gilmer, J., Schoenholz, S.S., Riley, P.F., Vinyals, O., Dahl, G.E.: Neural message passing for quantum chemistry. In: ICML. pp. 1263–1272. PMLR (2017)
32. Hagberg, A., Swart, P., S Chult, D.: Exploring network structure, dynamics, and function using networkx. Tech. rep., Los Alamos National Lab.(LANL), Los Alamos, NM (United States) (2008)
33. 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)
34. Jumper, J., Evans, R., Pritzel, A., Green, T., Figurnov, M., Ronneberger, O., Tunyasuvunakool, K., Bates, R., Žídek, A., Potapenko, A., et al.: Highly accurate protein structure prediction with alphafold. Nature pp. 1–11 (2021)
35. Kamada, T., Kawai, S., et al.: An algorithm for drawing general undirected graphs. Information processing letters **31**(1), 7–15 (1989)
36. Kaufmann, M., Spallek, A.M., Splett, J.: A heuristic approach towards drawings of graphs with high crossing resolution. In: GD 2018. vol. 11282, p. 271. Springer (2018)
37. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. In: ICLR (2015), http://arxiv.org/abs/1412.6980
38. Kipf, T.N., Welling, M.: Semi-supervised classification with graph convolutional networks (2017)
39. Kipf, T.N., Welling, M.: Semi-supervised classification with graph convolutional networks. In: ICLR (2017), https://openreview.net/forum?id=SJU4ayYgl
40. Kipf, T.N., et al.: Deep learning with graph-structured representations (2020)
41. Kwon, O.H., Ma, K.L.: A deep generative model for graph layout. IEEE TVCG **26**(1), 665–675 (2019)
42. Li, Y., Tarlow, D., Brockschmidt, M., Zemel, R.S.: Gated graph sequence neural networks. In: ICLR (2016), https://arxiv.org/abs/1511.05493
43. Maron, H., Ben-Hamu, H., Shamir, N., Lipman, Y.: Invariant and equivariant graph networks. In: ICLR (2019), https://openreview.net/forum?id=Syx72jC9tm

44. Monti, F., Boscaini, D., Masci, J., Rodola, E., Svoboda, J., Bronstein, M.M.: Geometric deep learning on graphs and manifolds using mixture model cnns. In: IEEE CVPR. pp. 5115–5124 (2017)
45. Morris, C., Ritzert, M., Fey, M., Hamilton, W.L., Lenssen, J.E., Rattan, G., Grohe, M.: Weisfeiler and leman go neural: Higher-order graph neural networks. In: AAAI Conference on Artificial Intelligence. vol. 33, pp. 4602–4609 (2019)
46. Murphy, R., Srinivasan, B., Rao, V., Ribeiro, B.: Relational pooling for graph representations. In: ICML. pp. 4663–4673 (2019)
47. Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., Lerer, A.: Automatic differentiation in pytorch (2017)
48. Purchase, H.: Which aesthetic has the greatest effect on human understanding? In: GD '97. pp. 248–261. Springer (1997)
49. Saket, B., Simonetto, P., Kobourov, S., Börner, K.: Node, node-link, and node-link-group diagrams: An evaluation. IEEE TVCG **20**(12), 2231–2240 (2014)
50. Sato, R., Yamada, M., Kashima, H.: Random features strengthen graph neural networks. In: SIAM SDM 2021. pp. 333–341. SIAM (2021)
51. Scarselli, F., Gori, M., Tsoi, A.C., Hagenbuchner, M., Monfardini, G.: The graph neural network model. IEEE Transactions on Neural Networks **20**(1), 61–80 (2009)
52. Shabbeer, A., Ozcaglar, C., Gonzalez, M., Bennett, K.: Optimal embedding of heterogeneous graph data with edge crossing constraints. In: Presented at NIPS Workshop on Challenges of Data Visualization. p. 1 (2010)
53. Srinivasan, B., Ribeiro, B.: On the equivalence between node embeddings and structural graph representations. ICLR (2020)
54. Staudt, C.L., Sazonovs, A., Meyerhenke, H.: Networkit: A tool suite for large-scale complex network analysis. Network Science **4**(4), 508–530 (2016)
55. Tamassia, R.: Handbook of graph drawing and visualization. CRC press (2013)
56. Tiezzi, M., Marra, G., Melacci, S., Maggini, M.: Deep constraint-based propagation in graph neural networks. IEEE TPAMI (2021)
57. Tiezzi, M., Marra, G., Melacci, S., Maggini, M., Gori, M.: A lagrangian approach to information propagation in graph neural networks. In: ECAI. Frontiers in Artificial Intelligence and Applications, vol. 325, pp. 1539–1546. IOS Press (2020). https://doi.org/10.3233/FAIA200262
58. Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., Bengio, Y.: Graph attention networks. In: ICLR (2018)
59. Wang, X., Yen, K., Hu, Y., Shen, H.W.: Deepgd: A deep learning framework for graph drawing using gnn. IEEE Computer Graphics and Applications (2021)
60. Wang, Y., Jin, Z., Wang, Q., Cui, W., Ma, T., Qu, H.: Deepdrawing: A deep learning approach to graph drawing. IEEE TVCG **26**(1), 676–686 (2019)
61. Weisfeiler, B., Lehman, A.A.: A reduction of a graph to a canonical form and an algebra arising during this reduction. Nauchno-Technicheskaya Informatsia **2**(9), 12–16 (1968)
62. Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., Yu, P.S.: A comprehensive survey on graph neural networks. IEEE Transactions on Neural Networks and Learning Systems **32**(1), 4–24 (2021)
63. Xu, K., Hu, W., Leskovec, J., Jegelka, S.: How powerful are graph neural networks? (2019)
64. Xu, K., Hu, W., Leskovec, J., Jegelka, S.: How powerful are graph neural networks? In: ICLR (2019), https://openreview.net/forum?id=ryGs6iA5Km
65. Yoghourdjian, V., Archambault, D., Diehl, S., Dwyer, T., Klein, K., Purchase, H.C., Wu, H.Y.: Exploring the limits of complexity: A survey of empirical studies on graph visualisation. Visual Informatics **2**(4), 264–282 (2018)

66. You, J., Ying, R., Leskovec, J.: Position-aware graph neural networks. ICML (2019)
67. Zhao, L., Song, Y., Zhang, C., Liu, Y., Wang, P., Lin, T., Deng, M., Li, H.: T-gcn: A temporal graph convolutional network for traffic prediction. IEEE T-ITS **21**(9), 3848–3858 (2019)
68. Zheng, J.X., Pawar, S., Goodman, D.F.: Graph drawing by stochastic gradient descent. IEEE TVCG **25**(9), 2738–2748 (2018)

A Related Work

There exists a large variety of methods in literature to improve graph readability. A straight-forward approach, that has been proved to be effective in improving the human understanding of the graph topology, consists in minimizing the number of crossing edges [48]. However, the computational complexity of the problem is NP-hard, and several authors proposed complex solutions and algorithms to address this problem [2]. In [52], authors employ an Expectation-Maximization algorithm based on the decision boundary surface built by a Support Vector Machine. The underlying idea is that two edges do not cross if there is a line separating the node coordinates. Further aesthetic metrics have been explored, such as the minimization of node occlusions [15], neighborhood preservation, the maximization of crossing edges angular width [36] and many more [29,3]. Given the graph drawing categorization depicted in surveys [29] (i.e. force-directed, dimension reduction, multi-level techniques), interesting and aesthetically pleasing layouts are produced by methods regarding a graph as a physical system, with forces acting on nodes with attraction and repulsion dynamics up to a stable equilibrium state [35]. Force-directed techniques inspired many subsequent works, from spring-embedders [26] to energy-based approaches [33]. The main idea is to obtain the final layout of the graph minimizing the *Stress* function (see Eq. 1). The forces characterizing this formulation can be thought of as springs connecting pairs of nodes. This very popular formulation, exploited for graph layout in the seminal work by Kamada and Kawai [35], was optimized with the localized 2D Newton-Raphson method. Further studies employed various complicated optimization techniques, such as the *Stress Majorization* approach which produces graph layout through an iterative resolution of simpler functions, as proposed by Gasner et al. [27]. In this particular context, some recent contributions highlighted the advantages of using gradient-based methods to solve graph drawing tasks. The SGD method was successfully applied to efficiently minimize the Stress function in Zheng et al. [68], displacing pairs of vertices following the direction of the gradient, computed in closed form. A recent framework, $(GD)^2$, leverages Gradient Descent to optimize several readability criteria at once [3], 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 [47], several criteria such as ideal edge lengths, Stress Majorization, node occlusion, angular resolution and many others can be easily optimized. We build our first contribution upon these ideas, proving that neural networks can be used to learn decomposed single criteria (i.e., edge crossing) approximating

smooth functions, with the purpose of providing a useful descent direction to optimize the graph layout.

Deep Learning has been successfully applied to data belonging to the non-Euclidean domain, e.g. graphs, in particular thanks to GNNs [14,62]. The seminal work by Scarselli et al. [51] proposes a model based on an information diffusion process involving the whole graph, fostered by the iterative application of an *aggregation function* among neighboring nodes up to an equilibrium point. The simplification of this computationally expensive mechanism was the goal of several works which leverage alternative recurrent neural models [42] or constrained fixed-point formulations. This problem was solved via Reinforcement Learning algorithms as done in Stochastic Steady-state Embedding (SSE)[19] or cast to the Lagrangian framework and optimized by a gradient descent-ascent approach, like in Lagrangian-Propagation GNNs (LP-GNNs) [57], even with the advantage of multiple layers of feature extraction [56]. The iterative nature of the aforementioned models inspired their classification into the umbrella of RecGNNs in recent surveys [62,4].

In addition to RecGNNs, several other flavours of GNN models have been proposed, such as the ConvGNNs [40] or Attentional GNNs [58,44,12]. All such models fit into the powerful concept of message exchange, the foundation on which is built the very general framework of Message Passing Neural Networks (MPNNs)[31,30].

Recent works analyze the expressive capabilities of GNNs and their aggregation functions, following the seminal work on graph isomorphism by Xu et al. [64]. The model proposed by the authors, Graph Isomorphism Network (GIN), leverage an injective aggregation function with the same representational power of the Weisfeiler-Leman (WL) Test [61]. Subsequent works (sometimes denoted with the term WL-GNNs) try to capture higher-order graph properties [45,43,16,9]. Bearing in mind that we deal with the non-attributed graph scenario, i.e., graphs lacking node features, we point out the importance of the nodal feature choice. Several recent works investigated this problem [18,53,46]. We borrow the highly expressive Laplacian Eigenvector-based positional features described by Dwivedi et al. [23].

In literature, several efforts have been made to develop powerful informative features for Graph Representation . Recently, it has been shown that the usage of random nodal features theoretically strengthens the representational capability of GNNs [50,1]. Indeed, setting random initial node embedding (i.e., different random values when processing the same input graph) enable GNNs to better distinguish local substructures, to learn distributed randomized algorithms and to solve matching problems with nearly optimal approximation ratios. Formally, the node features can be considered as random variables sampled from a probability distribution μ with support $D \subseteq \mathbb{R}^s$, $l_i \sim \mu, \quad \forall i \in \mathcal{V}$, where μ can be instantiated as the Uniform distribution. The main intuition is that the underlying message passing process combines such high-dimensional and discriminative nodal features, fostering the detection of fixed substructures inside the graph [50]. These approaches, which hereinafter we refer to as rGNNs, proved that

classification tasks can be tackled in a novel way, with a paradigm shift from the importance of task-relevant information (the features values) to the relevance of the relationship among node values. However, the peculiar regression task addressed in this work requires both positional and structural knowledge, which is essential to identify neighboring nodes.

There have been some early attempts in applying Deep Learning models and GNNs to the Graph Drawing scenario. Wang et al. [60] proposed a graph-based LSTM model able to learn and generalize the coordinates patterns produced by other graph drawing techniques. However, this approach is limited by the fact that the model drawing ability is highly dependent on the training data, such that processing different graph classes or layout styles requires re-collecting and re-training procedures. We prove that our approach is more general, given that we are able to learn both drawing styles from graph drawing techniques and to draw by minimizing aesthetic losses. Another very recent work, DeepGD [59], consists in a message-passing GNN which process starting positions produced by graph drawing frameworks [10], to construct pleasing layouts that minimize combinations of aesthetic losses (Stress loss combined with others). Both DeepDraw and DeepGD share the common need of transforming the graph topology into a more complicated one: DeepDrawing [60] introduces skip connections (*fake edges*) among nodes in order to process the graph via a bidirectional-LSTM; DeepGD converts the input graph to a complete one, so that each node couples is directly connected, and requires to explicitly provide the shortest path between each node couple as an edge feature. The introduction of additional edges into the learning problem increase the computational complexity of the problem, hindering the model ability to scale to bigger graphs. More precisely, in the DeepGD framework the computational complexity grows quadratically in the number of nodes $O(N^2)$. Conversely, we show that the GNN are capable of producing aesthetically pleasing layouts without inserting additional edges, by simply leveraging powerful positional-structural features. Additionally, we introduce a novel neural-based mechanism, the *Neural Aesthete*, capable to express differentiable aesthetic losses delivering flexible gradient direction also for non-differentiable goals. We show that this mechanism can be exploited by Gradient-descent based graph drawing techniques and by the proposed GND framework. Finally, GNN-based Encoder-Decoder architectures can learn a generative model of the underlying distribution of data from a collection of graph layout examples [41].

B Graph Neural Networks

In this appendix, we better define some notation. We denote with l_i all the input information (initial set of features) eventually attached to each node i in a graph G. The same holds for an arc connecting two nodes i and j, whose feature, if available, is denoted with $l_{(i,j)}$. Each node i has an associated hidden representation (or state) $x_i \in \mathbb{R}^s$, which in recent models is initialized with the initial features, $x_i = l_i$ (but it is not necessarily the case in RecGNN models [56]).

Many GNN models can be efficiently described under the powerful umbrella of Message Passing Neural Networks (MPNNs) [31], where the node state x_i is iteratively updated at each iteration t, through an aggregation of the exchanged information among neighboring nodes $\mathcal{N}_i$, undergoing a message passing process. Formally,

$$x_{(i,j)}^{(t-1)} = \texttt{MSG}^{(t)}\left(x_i^{(t-1)}, x_j^{(t-1)}, l_{(i,j)}\right) \tag{8}$$

$$x_i^{(t)} = \texttt{AGG}^{(t)}\left(x_i^{(t-1)}, \sum_{j \in \mathcal{N}_i} x_{(i,j)}^{(t-1)}, l_i\right) \tag{9}$$

where $x_{(i,j)}^{(t)}$ represent explicitly the message exchanged by two nodes, computed by a learnable map $\texttt{MSG}^t(\cdot)$.[6] Afterwards, $\texttt{AGG}^t(\cdot)$ aggregates the incoming messages from the neighborhood, eventually processing also local node information such as the node hidden state x_i and its features l_i. The messaging and aggregation functions $\texttt{MSG}^{(t)}(\cdot)$, $\texttt{AGG}^{(t)}(\cdot)$ are typically implemented via Multi-Layer Perceptrons (MLPs) learned from data. Apart from RecGNN, other GNN models leverage a different set of learnable parameters for each iteration step. Hence, the propagation process of such models can be described as the outcome of a multi-layer model, in which, for example, the node hidden representation at layer t, $x_i^{(t)}$, is provided as input to the next layer, $t+1$. Therefore an ℓ-step message passing scheme can be seen as an ℓ-layered model.

This convenient framework is capable to describe several GNN models [62]. In this work, we focus our analysis on three commonly used GNN model from literature (i.e., GCN [38], GAT [58], GIN [64]) whose implementation is given in Table 3, characterized by different kinds of aggregation mechanisms (degree-norm, attention-based, injective/sum, respectively). Following the Table notation, in GCN $c_{u,v}$ denotes a normalization constant depending on node degrees; in GAT $\alpha_{u,v}^{(t-1)}$ is a learned attention coefficient which introduces anisotropy in the neighbor aggregation, σ denotes a non-linearity and W, W_0, W_1 are learnable weight matrices; in GIN ϵ is a learnable parameter (which is usually set to zero).

C Drawing Graphs with the help of the Neural Aesthete

As anticipated in 3, a major contribution of this paper is that of also introducing the notion of *Neural Aesthete*.

Its smooth and differential loss function (see Eq. 5, foster the utilization of Gradient Descent methods to optimize the problem variables, i.e. the arc node coordinates (e_u, e_v).When considering all the graph edges:

$$H(P) = \sum_{(e_u, e_v) \in \mathcal{E}} L(y_{e_u, e_v}, \hat{y}_{e_u//e_v}) \tag{10}$$

[6] Notice that in the case in which arc features $l_{(i,j)}$ are not available they are removed from the problem formulation.

Table 3. Common implementations of GNN aggregation mechanisms. See the main text and the referenced papers for further details on the formulations.

Method: Funct.	Reference	Implementation
GCN[39]: Mean	Kipf and Welling [39]	$\sigma\Big(c_v W^{(t)} {x_v}^{(t-1)} + \sum_{u \in \mathcal{N}_v} c_{u,v} W^{(t)} {x_u}^{(t-1)}\Big)$
GAT[58]: Att.	Veličković et al. [58]	$\sigma\big(\sum_{u \in \mathcal{N}_v} \alpha_{u,v}^{(t-1)} W^{(t)} x_u^{(t-1)}\big)$
GIN[64]: Sum	Xu et al. [64]	$\text{MLP}^{(t)}\big((1+\epsilon){x_v}^{(t-1)} + \sum_{u \in \mathcal{N}_v} {x_u}^{(t-1)}\big)$

Overall, a possible graph drawing scheme is the one which returns: $P^\star = \arg\min_{\mathcal{P}} H(P)$. This can be carried out by classic optimization methods. For instance, a viable solution is by gradient descent as follows: $P \leftarrow P - \eta \nabla_P H(P)$, where η specifies the learning rate.

It is worth mentioning that, overall, this approach leverages the computational efficiency and parallelization capabilities of neural networks. Hence, the prediction of the edge-crossing degree can be carried out for many edge couples in parallel. Moreover, this same approach can be conveniently combined with other aesthetic criteria, for instance coming from other Neural Aesthetes or from classical loss function (e.g., Stress). For example, we could consider

$$E = H(P) + \lambda_A A(\cdot) + \lambda_B B(\cdot) \tag{11}$$

where $A(\cdot)$ and $B(\cdot)$ denotes other aesthetic criteria characterized by smooth differentiable functions.

C.1 Example: Neural Aesthete on small-sized Random Graphs

We provide a qualitative proof-of-concept example for the aforementioned Neural Aesthete for edge-crossing in Figure 4.

We built an artificial dataset composed of 100K entries to train the Neural Aesthete. Each entry of the dataset is formed by an input-target couple $(x, \hat{y})$. The input pattern x corresponds to the Neural Aesthete arcs input positions[7] as defined in Section 2, whose node coordinates are randomly picked inside the interval $[0, 1]$. The corresponding target $\hat{y}$ is defined by Eq. 2.

We balanced the dataset composition in order to have a comparable number of samples between the two classes (cross/no-cross). We trained a Neural Aesthete implemented as a Multi-Layer Perceptron (MLP) with two hidden layers of 100 nodes each and ReLu activation functions, minimizing the cross-entropy loss function with respect to the targets and leveraging the Adam optimizer [37]. We tested the generalization capabilities of the learned model on a test dataset composed of 50K entries, achieving a test accuracy of 97%[8]. Hence, the learned

[7] Which are defined as $x := [e_u, e_v] = [p_i, p_j, p_h, p_k]$, given two arcs $e_u = (p_i, p_j)$ and $e_v = (p_h, p_k)$.

[8] For comparison, a Decision Tree model, trained on the same dataset, only reaches a test accuracy of 78.7%.

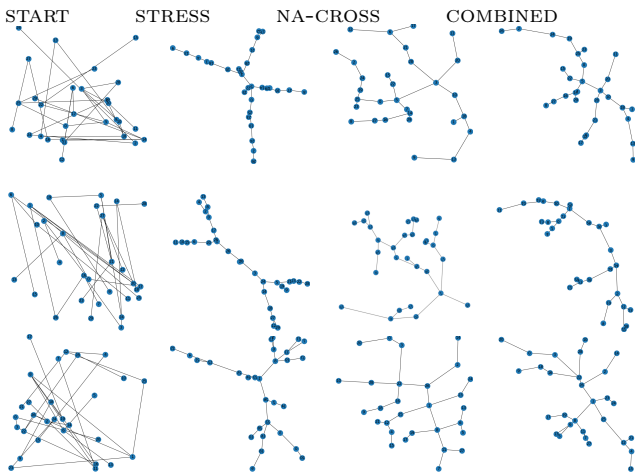

Fig. 4. Neural Aesthete for edge crossing. Left-to right, Graph layouts with starting random node coordinates (START), optimized by minimizing stress function with Gradient Descent (STRESS), optimized by Gradient Descent applied on the Neural Aesthete for edge-crossing loss (NA-CROSS), optimized by alternating stress loss and Neural Aesthete loss in subsequent iterations (COMBINED).

model constitutes the Neural Aesthete for the task of edge crossing. Given an unseen input composed of a couple of arcs, the learned model outputs a probability distribution representing a degree of intersection. Following the common pipeline of Graph Drawing methods with Gradient Descent, the Neural Aesthete output represent a differentiable function that provides an admissible descent direction for the problem parameters P.

To test the capability of the proposed solution, we leverage an artificial dataset of random graphs with a limited number of nodes ($N \in [20, 40]$). We generated Erdős-Rényi graphs with the method presented in [5] for efficiently creating sparse [9] random networks, implemented in NetworkX [32]. We selected the connected component of the generated graph having the biggest size (max node number).

Figure 4 reports a qualitative example of the proposed method in three graphs from the aforementioned dataset. To generate the graph layout, we carried on an optimization process on mini-batches of 10 arc-couples for an amount of 2K iterations (gradient steps). The first column depicts the starting random positions of the nodes; second column reports the graph layout obtained with an in-house implementation of the Stress function (see Eq. 1), optimized via Gradient Descent as done in [3]; third column contains the results obtained by optimizing the loss provided by the proposed Neural Aesthete for edge-crossing; fourth column reports the layouts obtained alternating the optimization of the Stress function

[9] The probability of edge creation has been set to $p = 0.01$.

and edge-crossing in subsequent update steps. It is noticeable to see how the solution provided by our approach is capable to avoid any arc intersection in these simple graphs. Moreover, the fact that the Neural Aesthete output represents a form of degree-of-intersection, seems to provide a good gradient direction that easily moves the arcs into a recognizable angle pattern, even when combined with other criteria (fourth column). The proposed proof-of-concept proves that Neural Aesthetes represent a feasible, general and efficient solution for Graph Drawing. In the following, we prove that this same approach can be used to guide the training process of different kind of Deep Neural Models.

D Experimental Details

We employed two different graph drawing datasets with different peculiarities. We chose to address small-size graphs (≤ 100 nodes) to assure the graph layout readability, since prior works highlighted node-link layouts are more suitable to small-size graphs [60,28]. The former one is the ROME dataset[10], a Graph Drawing benchmarking dataset containing 11534 undirected graphs with heterogeneous structures and connection patterns. We preprocessed the dataset removing three disconnected graphs[11]. Each graph contains a number of nodes between 10 and 100. Some samples of the dataset are reported in the first column of Figures 1 and 6, drawn with different layouts (see the following).

We built a second dataset, which we refer to as SPARSE, with the same technique described in Appendix C.1. We generated 10K Erdős-Rényi graphs following the method presented in [5] for efficient sparse random networks and implemented in NetworkX [32]. We randomly picked the probability of edge creation in the interval $(0.01, 0.05)$ and the number of nodes from 20 to 100. To improve the sparsity and readability, we discarded all the created graphs having both more than 60 nodes and more than 120 edges. Afterwards, we selected the connected component of the generated graph having the biggest size.

In order to carry out the training process and afterwards evaluate the obtained performances, we split each of the datasets into three sets, (i.e. training, validation, test) with a ratio of (75%, 10%, 15%).

All GNN models considered are characterized by the ReLU non-linearity. The GAT model is composed by four attention heads. The ϵ variable in the GIN aggregation process is set to 0, as suggested in [63]. We leverage the PyTorch implementation of the models provided by the Deep Graph Library (DGL)[12].

We searched for the best hyper-parameters selecting the models with the lowest validation error obtained during training, in the following grid of values: size of node hidden states x_i in $\{10, 25, 50\}$; learning rate η in $\{10^{-4}, 10^{-3}, 10^{-2}\}$; the number of GNN layers in $\{2, 3, 5\}$; PE dimension k in $\{5, 8\}$ (20 is added

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

[11] Stress-based Graph drawing techniques cannot take into account disconnected graphs. However, one can easily draw each connected component separately and then plot them side by side.

[12] https://www.dgl.ai/

to the grid in the case of the SPARSE dataset, given its greater node number lowerbound); drop-out rate in $\{0.0, 0.1\}$. We considered 100 epochs of training with an early stopping strategy given by a patience on the validation loss of 20 epochs. For each epoch, we sampled non-overlapping mini-batches composed by β graphs, until all the training data were considered. We searched for the best mini-batch size β in $\{32, 64, 128\}$.

We employed two different graph drawing datasets with different peculiarities. We chose to address small-size graphs (≤ 100 nodes) to assure the graph layout readability, since prior works highlighted node-link layouts are more suitable to small-size graphs [60,28]. The former one is the ROME dataset[13], a Graph Drawing benchmarking dataset containing 11534 undirected graphs with heterogeneous structures and connection patterns. We preprocessed the dataset removing three disconnected graphs[14]. Each graph contains a number of nodes between 10 and 100. Some samples of the dataset are reported in the first column of Figures 1 and 6, drawn with different layouts (see the following).

We built a second dataset, which we refer to as SPARSE, with the same technique described in Appendix C.1. We generated 10K Erdős-Rényi graphs following the method presented in [5] for efficient sparse random networks and implemented in NetworkX [32]. We randomly picked the probability of edge creation in the interval $(0.01, 0.05)$ and the number of nodes from 20 to 100. To improve the sparsity and readability, we discarded all the created graphs having both more than 60 nodes and more than 120 edges. Afterwards, we selected the connected component of the generated graph having the biggest size (max node number). We report in Figure 5 a visual description of the datasets composition.

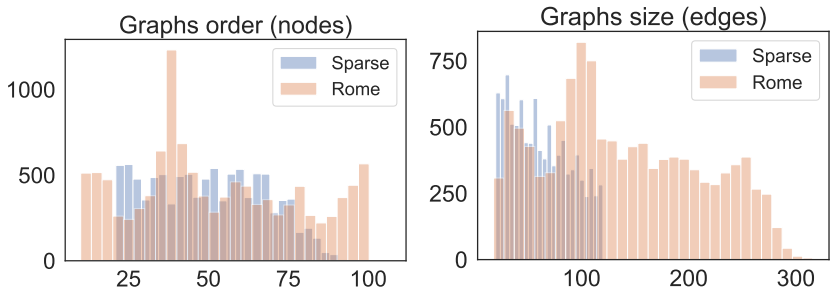

Fig. 5. Datasets composition statistics. On the left, the histogram of the graph order (number of nodes for each graph $|\mathcal{V}|$) for both the analyzed datasets. On the right, the histogram of the graphs sizes (number of edges $|\mathcal{E}|$). The SPARSE dataset is characterized by a sparse connection pattern.

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

[14] Stress-based Graph drawing techniques cannot take into account disconnected graphs. However, one can easily draw each connected component separately and then plot them side by side.

Furthermore, in order to provide a qualitative proof-of-concept of our claims, we built a synthetic dataset, which we refer to as SmallSparse, composed of 1000 graphs, with the same procedure as for the Sparse dataset. It only differs for the graph size, ranging in the interval $[10, 30]$ with the purpose of facilitating the task of edge-crossing avoidance.[15] We split the dataset in training, validation, test partitions with ratio (75%, 10%, 15%) and early stop the learning on the validation loss with a patience of 20 epochs.

E Additional Experiments

Figure 6 shows a qualitative evaluation obtained by the best performing models for each different GNN architecture on two randomly picked graphs from the test set of the Rome dataset, in the case of the Spectral layouts.

In general, the Spectral layout is easier to be learned by the models. This can be due to the fact that the Laplacian PE represent an optimal feature for this task, given the common spectral approach. Even from a qualitative perspective generated layouts are almost identical to the Ground Truth.

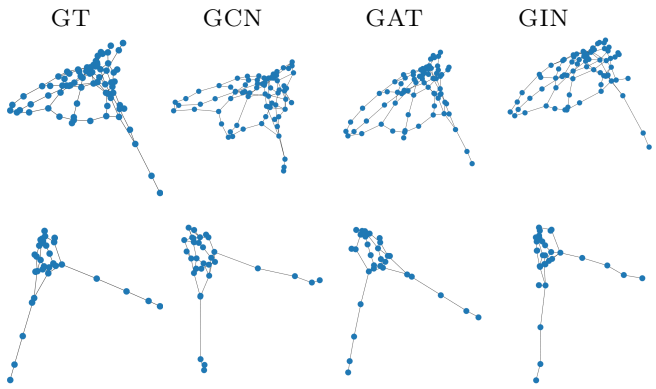

Fig. 6. Spectral layout. Predicted node coordinates for the Rome dataset. Each row depicts the Ground-Truth positions (GT), the graph layout produced by GCN, GAT, GIN model, left-to-right. We report the predictions on two different test graphs.

F Scaling to bigger graphs

Common Graph Drawing techniques based on multidimensional scaling [17] or SGD [68] require ad-hoc iterative optimization processes for each graph to be

[15] We built a dataset characterized by a sparse topology from the edge point of view. The task of edge-crossing avoidance is inherently connected with the graph size.

drawn. Additionally, dealing with large scale graphs – both in terms of number of nodes and number of involved edges – decreases the time efficiency of these approaches. Conversely, once a GND has been learned, the graph layout generation consist solely in the extraction of Laplacian PE followed by a forward pass on the chosen GNN backbone. In this Section, we prove the ability of GND to scale to real-world graphs, providing quantitative results in terms of computational times and a qualitative analysis on the obtained graph layouts, with respect to SOTA Graph Drawing techniques. We employed the best performing GAT model trained to minimize the stress loss on the Rome dataset (Section 4.2). We test the model inference performances on bigger scale graphs from the SuiteSparse Matrix Collection[16]. We report in Figure 7 the computational times required by the different techniques to generate graph layouts of different scale, from the `dwt_n` graph family. We analyze both the correlation on graph order (left – varying number of nodes) and size (right – varying number of edges). We compare the GND execution times against those of the NetworkX-GraphViz implementation of `neato` and `sfdp`, the latter being a multilevel force-directed algorithm that efficiently layouts large graphs. We also tested the Fruchterman-Reingold force-directed algorithm implemented in NetworkX (denoted with FR) and the PivotMDS implementation from the NetworKit `C++` framework [54]. The tests where performed in a Linux environment equipped with an Intel(R) Core(TM) i9-10900X CPU @ 3.70GHz, 128 GB of RAM and an NVIDIA GeForce RTX 3090 GPU (24 GB).

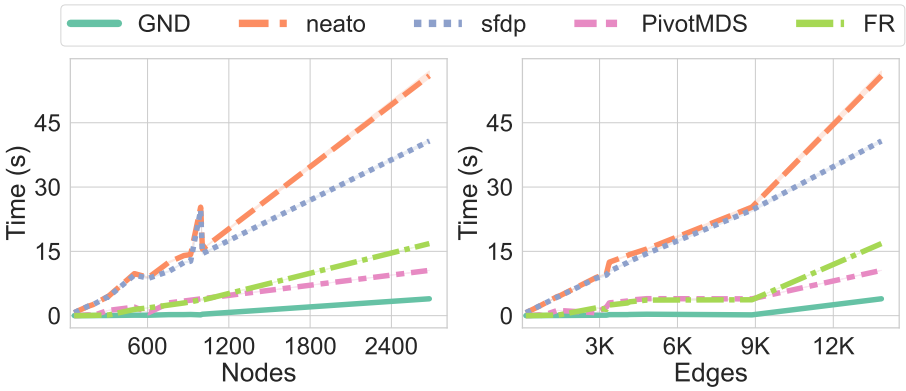

Fig. 7. Computational time comparison on `dwt_n` graphs. Left: correlation between the number of nodes in the graph and the layout generation timings of the analyzed Graph Drawing methods. Right: correlation between number of edges available in the graphs and the corresponding layout generation timings.

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

We report the average execution times over three runs (we omit the variances due to their negligible values). These results confirm the advantages of the proposed approach. While all the competitors require expensive optimization process that increase their impact with bigger graph scale, the fast inference step carried on by GNDs assures small timings even with big graphs. Computing Laplacian PE is scalable and does not hinder the time efficiency of the proposed method. To asses the quality of the generated layouts, we report in Figure 8 a comparison among the ones yielded by GND the framework, `sfdp` and PivotMDS on several graphs from the SuiteSparse collection (we report the graph name, its order $|\mathcal{V}|$ and size $|\mathcal{E}|$). While we remark that in this experiment we exploited a GND model trained on a smaller scale dataset (i.e., Rome), the performances show a significant ability of the model to generalize the learned laws (e.g., the stress minimization in this case) to unseen graphs, even when dealing with diverging characteristics. However, we also remark that graphs having very diverse structures from the training distribution may be not correctly plotted. The causes of such performances drop are twofold. First, the intrinsic dependance of neural models on the inductive biases learned during the training process leads to an inability to generalize to unseen graph topologies. On the other hand, the limitations of Laplacian PE to discriminate certain graph simmetries or structures [23] may be further compounded with larger scale datasets, which is an active area of research [18].

SuiteSparse Matrix Collection

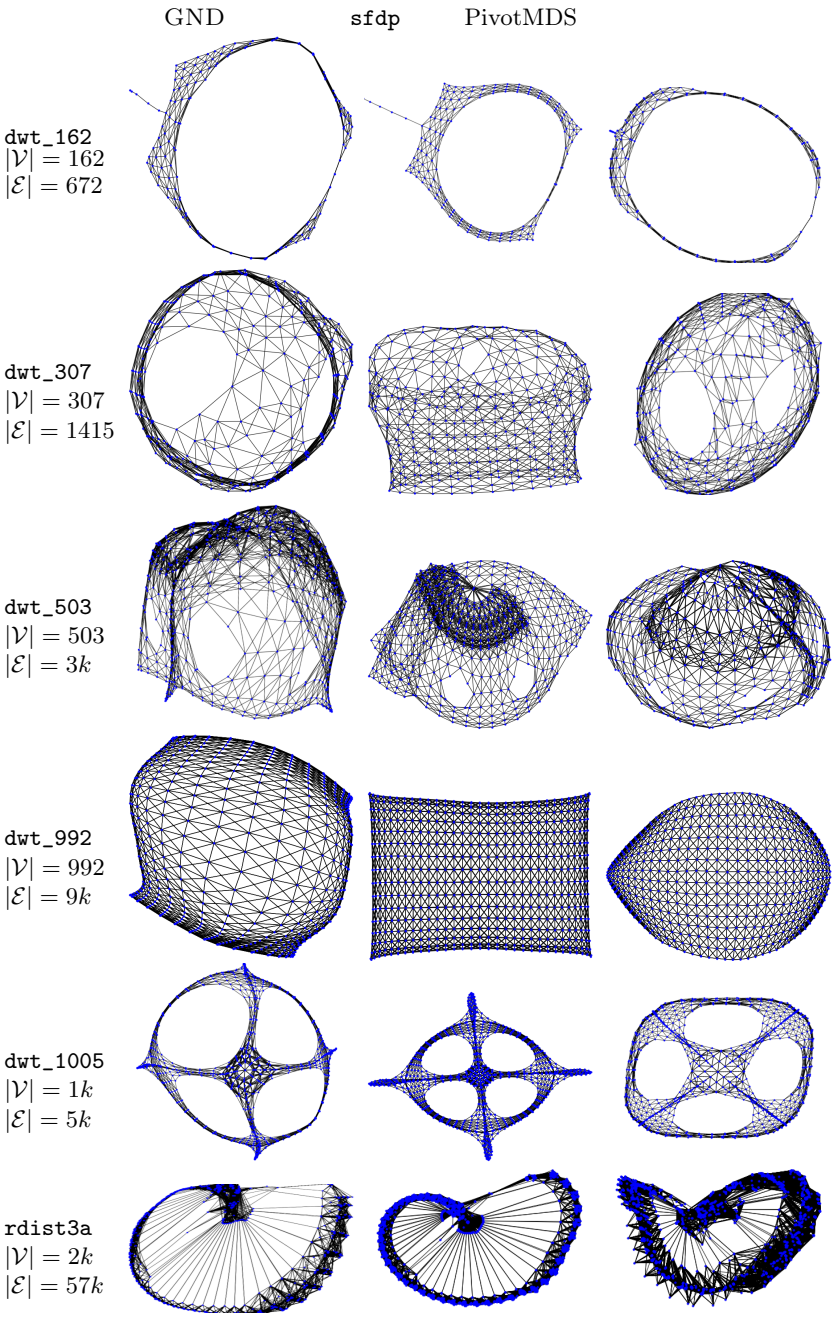

Fig. 8. Large scale graphs from the SuiteSparse Matrix collection. Left to right: layouts produced by a GAT-based GND (trained to minimize stress on the Rome dataset), layout produced by the `sfdp` algorithm for large scale graphs and outcome of the PivotMDS method. We report for each row the name of the graph from the dataset collection, its order ($|\mathcal{V}|$) and size ($|\mathcal{E}|$).