NNP-NET: Accelerating t-SNE Graph Drawing for Large Static and Dynamic Graphs by Neural Networks

Ilan Hartskeerl, Tamara Mchedlidze, Simon van Wageningen, Peter Vangorp, Alexandru Telea

***Abstract*—Among recent graph drawing (GD) methods, tsNET creates high quality layouts but suffers from a very high runtime due to its underlying reliance on the t-SNE projection technique. We address this problem by presenting NNP-NET, a method that adapts NNP, a projection technique that can project high-dimensional datasets linearly in the data size, to handle both unweighted and weighted graphs, with layout quality being very close to the ground-truth tsNET. We also exploit NNP's built-in out-of-sample ability to enable NNP-NET to project time-dependent (dynamic) graphs while striking a good balance between layout stability and good layout quality. We show experiments that outline how NNP-NET can handle very large graphs – up to 50 million nodes and 108 million edges faster than all other comparable methods we are aware of while also yielding good quality metric values.**

***Index Terms*—Information visualization, Visualization techniques and methodologies, Dimensionality reduction**

I. Introduction

Multiple techniques have been proposed to create straight-line drawings of graphs, including classical approaches such as force-directed methods [1], [2], spectral methods [3], [4] and recently machine learning (ML) approaches [5]–[7]. Within the ML class, *dimensionality reduction* (DR) methods address GD by encoding the graph structure in a high-dimensional space and project this data to the 2D space [8]–[10]. In the GD field, drawing very large graphs, having millions of nodes, is one of the long-standing open challenges [11]–[13].

DR methods have several advantages when used for GD. High-dimensional data can encode many complementary aspects of graphs such as path lengths, connectivity, and node or edge attributes (weights). Different such methods optimize for different quality aspects of the high-dimensional data to be projected which leads to different GD styles. Last but not least, DR methods are thoroughly tested and optimized for speed and quality in the ML field.

Within this class of methods, tsNET [14] uses the t-SNE projection technique [15], known to create high-quality embeddings, to create graph layouts. Yet, tsNET has quadratic time and space complexity in the graph node count, making it unsuitable for graphs with over a few thousand nodes. While the time complexity can be reduced by GPU-based t-SNE implementations [16], [17], and multilevel schemes and sparse matrices [18] can reduce space complexity, a more radical approach is to entirely *replace* t-SNE. A good candidate here is NNP [19] – a deep-learning projection method that can imitate any DR algorithm, has linear time complexity in input size, is parameter-free and robust to small input changes, and has out-of-sample ability. Yet, NNP training requires a high-dimensional dataset rather than a distance matrix.

A recent method, NNP-NET [20], leveraged NNP for graph drawing by proposing a scheme to reduce graphs to high-dimensional datasets by using the Pivot MDS (PMDS) [3] projection. Such datasets, selected from the input graph by a sampling scheme, can then be used to train NNP, after which the trained model embeds the entire graph. The method was shown to produce high-quality graph drawings on a wide selection of input graphs and to scale well computationally up to graphs having millions of nodes and edges. However, NNP-NET did not exploit the key *out-of-sample* property of NNP. Simply put, this allows NNP to project (slightly) changed datasets in a stable manner, *i.e.*, by keeping unchanged points in roughly the same locations.

In this paper[1], we extend NNP-NET to use this property for drawing time-dependent (dynamic) graphs and also evaluate it on larger datasets. The key added-value points of the NNP-NET method are as follows:

Scalability: NNP-NET has a theoretical time and space complexity linear in the input graph size. It handles graphs of over 1M nodes in similar and often lower time than DRGraph, the only t-SNE based method we are aware of that scales to such sizes, while yielding similar quality metric values. For larger graphs (8M to 50M nodes), NNP-NET becomes 5 to 10 times faster than DRGraph in our testing.

Quality: NNP-NET produces layouts that visually look similar, and have similar quality metrics, to those created by state-of-the art GD methods, including the ground-truth tsNET method it accelerates.

Genericity: NNP-NET can handle any type of graph, including edge-weighted graphs.

Time-dependence: NNP-NET can draw both static and dynamic (time-dependent) graphs. For the latter, the produced layouts change smoothly with no unneeded discontinuities as the input graph topology changes.

Stability: NNP-NET inherits NNP's robustness to noise, as demonstrated by our experiments.

Ease of use: NNP-NET can be used out-of-the-box without tuning any parameters to draw both static and dynamic graphs.

[1]This paper is an extended version of the paper [20] which originally appeared as part of Proc. Graph Drawing 2025.

Replicability: We make our method (full source code, datasets, metrics) openly available via GitHub (https://github.com/IlanHartskeerl/NNP-NET).

The structure of this paper is as follows. Section II presents relevant related work. Section III details our method. Section IV compares NNP-NET with state-of-the-art GD methods on a wide range of graphs and using various quality metrics. Section V presents the results of NNP-NET for the drawing of dynamic graphs. Section VI discusses NNP-NET's strengths and limitations. Finally, Sec. VII concludes the paper.

II. Background and Related Work

Given a graph $G = (V, E)$ with nodes $V = \{v_i\}_{i=1}^{N}$ and edges $E = \{(v_i, v_j) \in V \times V\}$, a straight-line graph drawing algorithm bijectively maps nodes of V to points $P = \{\mathbf{p}_i\}_{i=1}^{N} \subset \mathbb{R}^m$, where typically $m = 2$. Optional edge weights $w_{ij} \in \mathbb{R}^+$ can control P by influencing the target edge length $\|\mathbf{p}_i - \mathbf{p}_j\|$. We set $w_{ij} = 1$ in case weights are not given.

A. Graph Layout Methods

Hundreds of GD methods have been proposed in the past decades [21], [22]. We next focus on methods that aim to satisfy the contributions outlined in Sec. 1.

Force directed methods: Such methods create a layout by modeling the graph as a physical system where attraction forces describe graph edges and repulsion forces aim to reduce drawing visual clutter [2], [8], [23], [24]. Edge weights can be easily incorporated in the attraction factor. Classical force directed methods have a time complexity of $O(N^2)$, which is slow for large graphs. SFDP [25] and the fast multipole multilevel method (FM^3 [26]) use multilevel schemes to reduce this time complexity: A set of increasingly smaller graphs $\{G^i\}_1^K$ is created from the input graph G; the smallest graph G^K is then laid out using a suitable technique, after which its drawing is used to build the drawing for G^{K-1}, and next for all G^i until G. This yields a time complexity of $O(N \log(N))$, significantly better than the original $O(N^2)$ complexity, yet still below the theoretical linear $O(N)$ we aim at.

Dimensionality reduction methods: Dimensionality reduction (DR) and GD have significant overlap, where DR algorithms can be adapted to a GD context [27]. DR based methods encode the graph adjacency information E into a distance matrix and then use classical DR techniques (discussed next in Sec. II-B) to create the drawing. For example, one can encode the shortest-path distances between all node pairs in the distance matrix and then reduce this to a 2D embedding using classical multidimensional scaling (MDS [28]). However, this approach has a time and space complexity of $O(N^2)$. PMDS [3] improves by using a smaller matrix of distances from all nodes to a small set of so-called *pivots*. This makes PMDS fast but the quality of the produced drawings falls behind other GD methods. tsNET [14] follows a similar idea but replaces MDS by t-SNE [15], one of the highest-quality existing DR methods [29]. To increase quality and robustness, t-SNE is started from a set of 2D node positions computed using PMDS – a variation called tsNET*. While tsNET(*) layouts are of high quality, the method has a $O(N^2)$ space and time complexity. DRGraph [18] improves upon tsNET to achieve a theoretical linear time complexity. A sparse similarity matrix, where only the k nearest neighbors of each node are present, is used. To accelerate t-SNE, a negative sampling technique [30] is used to approximate the gradient of the cost function. A multilevel layout scheme is used to reduce the number of t-SNE iterations. Yet, DRGraph cannot handle weighted graphs and has many complex parameters which make its practical use challenging.

Machine learning methods: SGD^2 [31] improves upon the earlier method GD^2 [32] by using stochastic gradient descent like its predecessor [33]. SGD^2 [31] directly optimizes for quality metrics which allows one to control the desired properties of the resulting drawing. Time complexity depends on the metric optimized for, which in the case for stress is $O(N^2)$. Stress-Plus-X [34] takes a similar approach, optimizing for a specific set of metrics (stress, edge crossings, minimum angle and upwardness). Other recent approaches exploit advances in deep learning to support graph drawing. Yet, several challenges remain here. DNN^2 [6], [35] and DeepGD [36], two recent methods in this area, were only trained to lay out graphs of about 100 nodes due to long training times. DeepDrawing [37] uses a Long Short Term Memory (LSTM) network in order to imitate a ground truth layout. This means that it is only able to create a layout for a graph that it is trained on, making it hard to use. DeepDrawing was only tested on graphs with about 100 nodes. SmartGD [38] optimizes for different (combinations of) quality metrics just like SGD^2 [31], but is untested on larger graphs. Graph Neural Drawers (GND) [7] uses graph neural networks and can handle graphs of about 10K nodes. GND has a time complexity of $O(|E|^{\frac{3}{2}})$. CoReGD [5] also uses graph neural networks and can handle graphs of roughly 25K nodes. Yet, its complexity of $O(N \log(N))$ is above our linear target.

B. Dimensionality Reduction

Given a dataset $\mathbf{X} = \{\mathbf{x}_i\} \subset \mathbb{R}^n$, dimensionality reduction (DR) methods, also called projections, $M : \mathbf{X} \to \mathbf{Y}$, create a dataset $\mathbf{Y} = \{\mathbf{y}_i\} \subset \mathbb{R}^m$, $m \ll n$ which preserves the so-called *data structure* of $\mathbf{X}$, such that points close (resp. far) in $\mathbb{R}^n$ are mapped to points close (resp. far) in $\mathbb{R}^m$. DR methods accept as input either $\mathbf{X}$ or a matrix of the pairwise distances between its data points. Many DR methods exist, each of which preserves different data structure aspects and, as such, produce different drawings of the same dataset, differing significantly in terms of quality, speed, robustness, out-of-sample ability, and ease of use [29].

NNP (Neural Network Projection) [19] is a meta-approach that aims to solve the scalability and lack of out-of-sample ability for any other projection technique. Given a subset of $\mathbf{X}' \subset \mathbf{X}$ (a few thousand points) and its projection $\mathbf{X}'$ computed by any user-chosen method, NNP learns the mapping $X' \to M(\mathbf{X}')$ by a simple neural network (3 fully connected hidden layers, sizes 256, 512, 256). This mapping is used to project the entire dataset $\mathbf{X}$. NNP creates projections of quality close to the ground truth in time $O(|\mathbf{X}|)$; is robust to small changes in the input data [39]; can imitate any user-chosen projection;

handles datasets of any dimensionality n; and has no free user set parameters. As such, NNP is an ideal candidate to replace t-SNE in tsNET for graph drawing. Yet, a key obstacle exists: NNP needs a high-dimensional *dataset* $\mathbf{X}$ as input, not a distance *matrix* computed on $\mathbf{X}$, making it not directly applicable to graphs. We describe next how we overcome this limitation.

C. Dynamic Graph Drawing

Let $G(t) = \{G_t\}, 0 \leq t \ \leq T$ be a sequence of T graphs, where G_{t+1} is obtained from G_t by removing a (typically small) subset of the nodes and/or edges of G_t and adding extra nodes and/or edges. We next call a graph $G_t \in G(t)$ a *frame* of $G(t)$. A drawing of the sequence $G(t)$ is a corresponding sequence $P(t) = \{P_t\}, 0 \leq t \ \leq T$, where P_t is the set of 2D points that represent the drawing of G_t. A good drawing of $G(t)$ obeys the desirable properties of static graph drawing mentioned earlier; additionally, it aims to visually encode the *dynamics* of changes in the graph: Small (respectively large) changes from G_t to G_{t+1} should manifest themselves as small (respectively large) changes from P_t to P_{t+1}.

Many dynamic graph drawing approaches exist [40]. For instance, one can visualize each graph in the sequence in a static drawing; visualize solely the change between sequences or create animations to highlight changes. Each solution attempts to visualize smooth changes between graph sequences but as the size of graphs increases this becomes much more challenging. Some large dynamic graph drawing approaches simplify the problem by focusing on preserving clusters of nodes [41]. Overall, however, visualizing dynamic graphs with millions of nodes/edges has been underexplored in dynamic graph drawing [40].

Concerning high-dimensional data, only a few projection methods have been proposed to handle such data when it is time-dependent [42]. Close to our focus, t-SNE cannot properly handle dynamic data due to its stochastic nature (for more details, we refer to [42], [43]. Parametric t-SNE can do this but this method is significantly more complex, has additional parameters to set/tweak, and is significantly slower than t-SNE [44]. Dynamic t-SNE [43] aims to alleviate the above but still depends on the inherently slow t-SNE method. On the other hand, the fast, parameter-free, and simple NNP method [19] has not been extended to handle time-dependent data.

III. Method

As outlined earlier, we aim to replace the t-SNE step of tsNET with NNP. For this, we must solve two problems: Given a graph G, how to (1) reduce G to a high-dimensional dataset, as needed by NNP; (2) extract a representative subset of G to train NNP on. We address both these problems next. An overview of the NNP-NET pipeline is shown in Fig. 1. Pseudo code for the pipeline can be found in Appendix E.

A. Mapping the Graph to High-dimensional Data

NNP requires a dataset $\mathbf{X} = \{x_i\} \subset \mathbb{R}^n$ as input, where n is a free-to-choose parameter. Creating $\mathbf{X}$ from G in theoretical linear time in relation to the graph size, one of our key goals, rules out using G's full distance matrix. We propose two methods to create the embedding $\mathbf{E} = \{e(v)\}_{v \in V}$: (1) *distance-to-pivot* computes a smaller matrix A than $\mathbf{E}$ in the same way as PMDS [3] computes its pivot matrix (repeatedly taking the node with the highest minimum distance to all already chosen nodes); and (2) *projecting* v to $\mathbb{R}^n$ dimensions via PMDS using $p = 250$ pivots. In both cases, weights are taken into account when calculating the graph theoretical distances by using Dijkstra instead of the faster BFS. Comparison of the two approaches shows that method (2) yields better layout quality albeit at a lower speed (see Appendix A). Another advantage of (2) is that PMDS can be freely swapped with any other projection technique. Note also that, while PMDS cannot project data to 2D accurately in general, we only use it as an *intermediate* mapping (to $n \gg 2$ dimensions); the final 2D layout is created by NNP. Setting $n = 50$ balances well between quality and computing time (see Appendix B).

B. Extracting Suitable Training Data

To train NNP, we need to extract a small but representative subgraph $G' = (V', E'), G' \subset G$ of size $S = |V'|$. Following various tests (Appendix C), we set the number of nodes to $S = 10000$. Many multilevel GD methods can extract such a subgraph G', *e.g.*, SFDP [25] and FM^3 [26]. These multilevel layout techniques differ in that (1) they reduce G until it is not useful to reduce any further, while we reduce to a *fixed* size S; (2) they care about all intermediate levels; we only care about the final G'.

To consistently reduce G to S nodes, we use a multilevel scheme similar to DRGraph [18], *i.e.*, *coarsen* G into a sequence $G, G^1, G^2, \ldots, G'$ of increasingly smaller graphs. Each iteration, we repeatedly choose a center node $\mathbf{c} \in G^i$ from all not yet clustered nodes that has the lowest neighbor count. All not-yet-clustered direct neighbors of $\mathbf{c}$ are added to $\mathbf{c}$'s cluster. The iteration ends when all nodes of G^i are clustered. Finally, we create G^{i+1} using the cluster centers $\mathbf{c}$ as nodes, and connecting nodes corresponding to adjacent clusters. Iterations continue until we reach the target size S.

Not all graphs can be reduced to the target size S by this coarsening method. To handle this, we use a stopping heuristic that exits coarsening when $S^i > 0.95S^{i-1}$, where S^i denotes the size of G^i. We then use a slower backup method to create G' by selecting S such points from G^i. This is done using *pivot-points* in a max-min approach, like PMDS [3]– choose a first node randomly, then add nodes in order of highest minimum distance to all already chosen nodes until we get S nodes. This gives good results but has a complexity of $O(NS)$. Appendix D compares the pivot-points and coarsening methods showing that the latter yields better quality-*vs*-speed.

After constructing the subgraph G', we lay out G' using tsNET* and train NNP to mimic the produced layout when given its corresponding embedding.

C. Smoothing

Training NNP on the subgraph G' gives good results in terms of training error (see next Sec. IV). Yet, the produced layouts

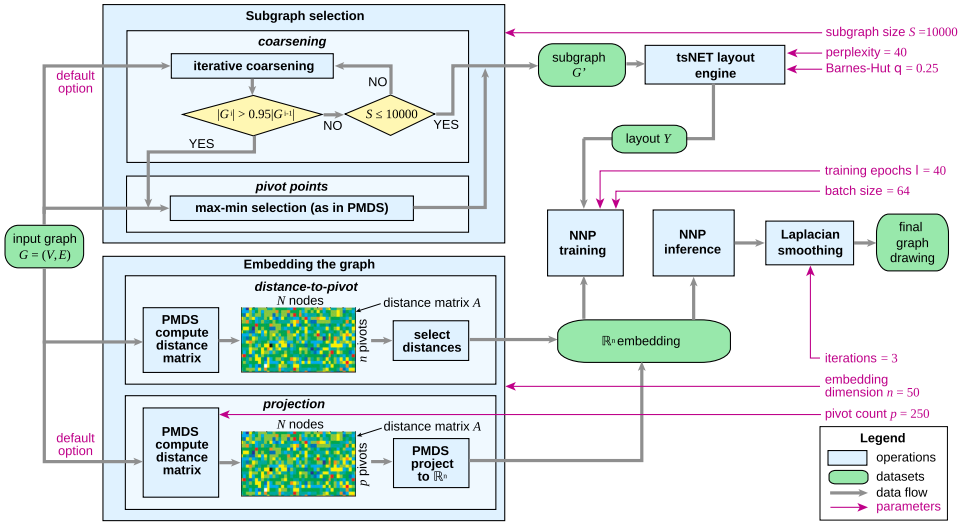


Fig. 1. NNP-NET pipeline. For details, see Sec. III.

contain some amount of high-frequency noise, visible in, *e.g.*, graphs with a grid-like structure (Fig. 2 left). We remove such noise using three Laplacian smoothing iterations (Fig. 2 right) on each node v_i via

$$v_i = \frac{1}{\sum_{j \in V_i} 1/w_{ij}} \sum_{j \in V_i} \frac{v_j}{w_{ij}}, \tag{1}$$

where V_i are all direct neighbors of v_i.

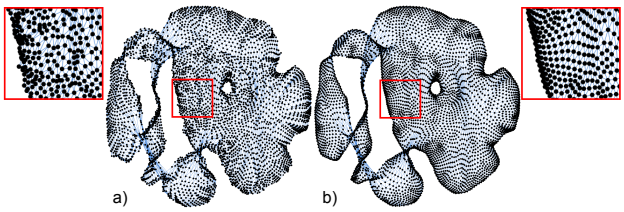


Fig. 2. (a) NNP-NET drawing of the 3elt graph. (b) Effect of 2 Laplacian smoothing passes.

D. Complexity

Table I shows the theoretical time complexity of all steps of our method. For creating the $\mathbb{R}^n$ embedding via PMDS (Sec. III-A), both the dimension count n and pivot count p are constants, yielding linear complexity in the number of nodes N of the input graph G to draw. Note that the term p^2N accounts for PMDS' complexity. For creating the subgraph G' (Sec. III-B), if our coarsening successfully reaches the target size S, the cost reduces to $O(iN)$; the factor SN accounts for using the pivot fallback. Moreover, when using coarsening, since G^k is reduced by at least a factor of 5% at each coarsening step, i is upper bounded by $\sum_{k=0}^{\infty}(0.95)^k = 20$, so the cost term $O(iN)$ is indeed linear in N. Generating ground truth by running tsNET* on G' does not depend on the input size N. Training also does not depend on N and is linear in the number of training epochs λ, which is set to a constant value. Inference is linear in N. Altogether, the end-to-end theoretical time complexity of NNP-NET is linear in its input size N. Separately, we note that NNP-NET's space complexity is also linear in N.

However, we note that our settings $S = 10000$ and $p = 250$ lead to large constant factors for the linear N terms (Tab. I, topmost two rows). As such, in practice, our effective speed can be lower than other linear methods, *e.g.*, PMDS (see Tab. IX and related text). However, as we will show, our method compensates this by an increased layout quality. Note also that relaxing the settings for S and p (discussed in the Appendices) can decrease these costs at the expense of a somewhat decreased layout quality.

Separately, as explained in Sec. III-A, for weighted graphs, we use Dijkstra (which is $O(N \log N)$) instead of BFS (which is $O(N)$) to compute graph theoretical distances. That is, for weighted graphs, our method has a theoretical time complexity of $O(N \log N)$.

TABLE I
THEORETICAL TIME COMPLEXITY OF ALL STEPS OF NNP-NET.

Algorithm step	Time complexity	Parameters
Create the $\mathbb{R}^n$ embedding of G (Sec. III-A)	$O(npN + p^2N)$	n dimensions, p pivots
Create subgraph G' of size S (Sec. III-B)	$O(iN + SN)$	i iterations, S nodes
Create ground truth by running tsNET* on G'	$O(S \log(S))$	none
NNP training	$O(\lambda nS)$	λ training epochs
NNP inference to lay out G	$O(nN)$	none
Laplacian smoothing (Sec. III-C)	$O(N)$	none

E. Drawing dynamic graphs

Given a dynamic graph $G(t)$ as defined in Sec. II-C, NNP's out-of-sample ability implies that the model can be trained only once and then used to create the drawings of all frames t simply from the embeddings of each such frame. Large changes between frames, given by added or deleted nodes and/or edges, will manifest themselves as large layout changes. Conversely, parts of the graph which do not change should appear stable in the layout. However, a problem arises here as the PMDS embedding used by NNP-NET is *not stable* across time-steps.

So in order to make NNP-NET stable, we need to make PMDS stable.

To obtain stability, the PMDS embedding for each time-step needs to be created using the same pivot points. We achieve this as follows. The first time we create an embedding, the pivot points are chosen the same as for the static drawing case, except that we mandate that the chosen nodes must exist in all frames. We next use this set of pivot points to embed all subsequent frames. This implies that our graphs G_t must have at least p common points across all the T frames where $p = 250$ is our used pivot count.

While the above heuristic creates a stable pivot-set, the obtained embedding can still exhibit flipping and scaling problems: The structure of the graph is stable, but flipped or scaled along one or multiple axis. To remove such issues, we use Procrustes alignment [45] applied on the eigenvectors that are computed within the PMDS pipeline. Specifically, for all frames, we align the frame-specific eigenvectors with the corresponding ones computed for the training of NNP.

Finally, we must decide how to select a training set for NNP from the entire dynamic graph $G(t)$. For this, we chose to use the *summary graph*, a combination of all nodes and edges that are in at least a time-step. This is done to ensure that no information of prior or later time-steps is excluded in the training step. NNP is trained on the summary graph using the same NNP-NET pipeline as per the static case (Sec. III-A and Sec. III-B). This trained NNP is then used to project all frames t. The eigenvectors computed as part of PMDS for the embedding of this summary graph are used as input for the Procrustes alignment for all time-steps.

IV. Results on Static Graphs

A. Experimental Setup

Implementation: We implement NNP-NET in C++ (code publicly available [46]). We use PMDS from OGDF [47], modified to allow for $n > 3$ output dimensions. We implement tsNET(*) [14] to use both exact t-SNE and the tree-based Barnes-Hut t-SNE approximation [48] which reduces projection cost to $O(N \log(N))$ from $O(N^2)$; we also parallelized tsNET(*) to use multiple CPU cores. We implement NNP with TensorFlow [49] using the CPU as the GPU proved slower in our tests due to the small size of this neural network.

Datasets and techniques. Table II shows the graphs used in our testing, all coming from the SparseSuite collection [50]. Most graphs used are on the larger side since a key goal we have is to speed up tsNET (Sec. I). We compare NNP-NET with SFDP [25], FM^3 [26], PMDS [3], DRGraph [18], all state-of-the-art GD methods that aim to handle large graphs; and with tsNET [14] given we aim to mimic this method with reduced execution time.

Parameter values: FM^3 uses the OGDF implementation with default parameters. SFDP uses the GraphViz implementation [51]. For DRGraph [18], we use the authors' implementation with their suggested parameter values. Tab. III lists all NNP-NET parameter values. We compute the tsNET* ground truth on the full graph G instead of G' when $S > N$. All tests are run on a PC with an Intel i7-11370H CPU (4 cores) and 16 GB of RAM.

TABLE II
Graph datasets used in the evaluation. All graphs come from SparseSuite [50].

Graph dataset	$\|V\|$	$\|E\|$	Weights
dwt_1005	1005	3808	
sierpinski3d	2050	6144	
MISKnowledgeMap	2427	28511	✓
3elt	4720	13722	
optdigits_10NN	5620	39825	✓
fe_4elt2	11143	32818	
bcsstk36	23052	1143140	
k49_norm_10NN	38547	309079	✓
fe_bcsstk32	44609	985046	
m_t1	97578	9753570	
ship_003	121728	3777036	
fe_ocean	143437	819186	
ok2010	269118	1274148	✓
web-NotreDame	325729	1497134	
coPapersCiteseer	434102	16036720	
gsm_106857	589446	21758924	
tx2010	914231	4456272	✓
com-Youtube	1134890	5975248	
Flan_1565	1564794	59485419	
com-LiveJournal	3997962	34681189	

TABLE III
Parameter values used for NNP-NET.

Parameter	Value
subgraph size $S = \|G'\|$	10000
n (embedding size)	50
Laplacian smoothing passes	3
t-SNE perplexity	40
θ (Barnes-Hut approximation)	0.25
p (PMDS pivot points)	250
batch size (NNP training)	64
epochs λ (NNP training)	40

B. Quality Metrics

We assess the quality of graph drawings using neighborhood preservation, stress, and Shepard diagrams, in line with [14], [18]. Metrics like edge length deviation [52], crossing number [53], and node-edge occlusion [54], though frequently used in GD literature, are not very informative for very large graphs.

Neighborhood preservation measures how well neighborhoods in G are preserved in the drawing $\mathbf{Y}$ [55]. Let $N_G(v_i, r_G) = \{v_j \in V | d_{ij} \leq r_G\}$ be the set of nodes v_j with a *graph-theoretic distance* d_{ij} of at most r_G from v_i; we set $r_G = 2$ following [14]. Let $N_Y(\mathbf{p}_i, k_i)$ be the set of nodes whose drawings $\mathbf{p}_j$ are the k_i-nearest-neighbors of $\mathbf{p}_i$ in the 2D layout space, where $k_i = |N_G(y_i, r_G)|$. Neighborhood preservation

$$\nu = \frac{1}{|V|} \sum_i \frac{|N_G(v_i, r_G) \cap N_Y(\mathbf{p}_i, k_i)|}{|N_G(v_i, r_G) \cup N_Y(\mathbf{p}_i, k_i)|} \in [0, 1] \qquad (2)$$

measures how similar $N_G(v_i, r_G)$ and $N_Y(\mathbf{p}_i, k_i)$ are (higher values are better).

Normalized stress measures how well the graph-theoretical distances d_{ij} between all node pairs are preserved by Euclidean distances in the graph drawing by

$$\sigma = \min_a \frac{1}{|V|^2} \sum_{i \neq j} \frac{(d_{ij} - a\|\mathbf{p}_i - \mathbf{p}_j\|)^2}{{d_{ij}}^2} \in [0, 1], \qquad (3)$$

with lower values indicating better distance preservation. The factor a scales the drawing in order to minimize stress for a fair comparison. Following [5], we compute it using

$$a = \frac{\sum_{i \neq j} \|\mathbf{p}_i - \mathbf{p}_j\| / d_{ij}}{\sum_{i \neq j} \|\mathbf{p}_i - \mathbf{p}_j\|^2 / d_{ij}^2}. \quad (4)$$

Table IV shows neighborhood preservation ν for the graphs listed in Tab. II. Results for the largest graph *com-LiveJournal* are missing as it took too long to evaluate Eqn. 2. Other missing values tell that the respective GD method could not complete on the respective graph due to running out of RAM or took longer than 2 hours to compute. The NNP-NET results are very close to tsNET* for most graphs, with NNP-NET doing worse for *e.g. MISKnowledgeMap* and *k49_norm_10NN* but much better for *fe_bcsstk32*. NNP-NET results are consistently better than PMDS except for *com_YouTube*. When compared with DRGraph, results are more mixed, with no clear winner. However, as discussed further in Tab. X and related text, as graph sizes increase, NNP-NET surpasses DRGraph in terms of neighborhood preservation. Separately, we see that PMDS (which aims at optimizing for stress only) scores very poor on neighborhood preservation – a point which we discuss further.

Table V shows stress σ for all generated drawings. NNP-NET yields very similar values to tsNET* (both versions). Similarly for ν, we do not see a clear winner between NNP-NET and DRGraph. Yet, DRGraph tends to yield lower stress for smaller graphs; NNP-NET performs better for larger graphs. This tendency only increases with graph size, see further Tab. X. PMDS, SFDP and FM^3 yield lower σ values than NNP-NET – not surprising since these methods optimize for stress while NNP-NET (like t-SNE) optimizes for neighborhood preservation. However, as well-known in the GD literature, stress does not fully capture the fact that a graph drawing is of high quality. Stress-optimizing methods like the ones mentioned above (a) deliver quite poor neighborhood preservation (see Tab. IV and (b) have difficulties in disentangling complex graphs – see, for example, the results of PMDS for *sierpinski3d*, *3elt*, and *optdigits_10NN* in Tab. VII. Even worse, PMDS almost fully collapses the layouts of some graphs, see *e.g. ok2010*, *Flan_1565*, and *tx2010* in Tab. VII.

TABLE IV
NEIGHBORHOOD PRESERVATION ν PER GRAPH AND LAYOUT METHOD.

	Neighborhood preservation per graph.						
Graph	SFDP	FM^3	PMDS	tsNET* exact	tsNET* approx	DRGraph	NNP-NET
dwt_1005	0.5	0.53	0.47	0.62	0.59	0.5	0.53
sierpinski3d	0.51	0.51	0.2	0.55	0.53	0.52	0.52
MISKnowledgeMap	0.17	0.16	0.13	0.4	0.41	0.33	0.24
3elt	0.62	0.65	0.36	0.66	0.63	0.63	0.63
optdigits_10NN	0.52	0.5	0.35	0.61	0.63	0.62	0.61
fe_4elt2	0.47	0.52	0.25	0.6	0.59	0.56	0.59
bcsstk36	0.45	0.41	0.3	-	0.51	0.49	0.44
k49_norm_10NN	0.048	0.045	0.034	-	0.18	0.12	0.093
fe_bcsstk32	0.32	0.38	0.21	-	0.24	0.41	0.37
m_t1	0.3	0.34	0.21	-	0.35	0.37	0.32
ship_003	0.21	0.21	0.17	-	0.2	0.24	0.24
fe_ocean	0.11	0.12	0.09	-	-	0.12	0.09
ok2010	0.48	0.46	0.27	-	-	0.32	0.44
web-NotreDame	0.38	0.31	0.33	-	-	0.46	0.39
coPapersCiteseer	-	0.079	0.058	-	-	0.17	0.091
gsm_106857	-	0.17	0.12	-	-	0.24	0.21
tx2010	0.42	0.39	0.26	-	-	0.21	0.36
com-Youtube	0.015	-	0.092	-	-	0.055	0.069
Flan_1565	-	-	0.09	-	-	0.2	0.21

Shepard diagrams refine distance-preservation insights captured by stress. These diagrams are scatterplots that show how distances between point-pairs correlate over two different spaces [18], [56], [57], with graph distance on the x-axis. Figure VI shows these diagrams for our tested graphs and methods. Scatterplots close to the main diagonal indicate cases where the layout preserves graph distances well, *e.g.*, for *dwt_1005* and *ship_003*, for all methods. Plots falling largely under this diagonal indicate layouts where the layout cannot 'unclutter' the graph. We see how this occurs for some of the very large graphs *e.g. com-YouTube* and *com-LiveJournal*, unsurprising, given the difficulty to lay out such complex structures. Plots falling above the diagonal indicate layouts where points are placed too far away from each other, or, in other words, too long edges drawn in the layout, see *e.g. fe_bccstk32* (tsNET* approx). Overall, we see that SFDP, FM^3, and PMDS preserve distances better than the other methods, which is expected given their design. However, as discussed earlier (and also shown by the actual drawings in Tab. VII) such methods yield poorer quality in terms of disentangling some of the more complex graphs. The remaining methods – which all are based on tsNET, so favor neighborhood preservation, have quite similar patterns of distance preservation on the smaller graphs. For the larger graphs, which only DRGraph and NNP-NET can handle, we observe either similar patterns or a tendency for DRGraph to underestimate 2D distances more than NNP-NET, *i.e.*, clutter nodes – see *e.g. tx_2010*, *CoPapers-CiteSeer*, and *ok_2010*. Overall, we conclude that NNP-NET performs at least as well as the other t-SNE based methods with respect to distance preservation.

TABLE V
STRESS σ PER GRAPH AND LAYOUT METHOD.

	Stress (lower is better)						
Graph	SFDP	FM^3	PMDS	tsNET* exact	tsNET* approx	DRGraph	NNP-NET
dwt_1005	0.029	0.026	0.029	0.038	0.036	0.03	0.029
sierpinski3d	0.079	0.078	0.104	0.077	0.086	0.08	0.125
MISKnowledgeMap	0.149	0.171	0.159	0.188	0.186	0.191	0.199
3elt	0.057	0.062	0.056	0.08	0.079	0.059	0.073
optdigits_10NN	0.159	0.144	0.125	0.142	0.14	0.172	0.139
fe_4elt2	0.05	0.069	0.064	0.095	0.098	0.068	0.097
bcsstk36	0.069	0.071	0.068	-	0.11	0.074	0.088
k49_norm_10NN	0.162	0.153	0.155	-	0.168	0.173	0.165
fe_bcsstk32	0.138	0.093	0.098	-	0.187	0.175	0.179
m_t1	0.095	0.079	0.074	-	0.078	0.103	0.147
ship_003	0.069	0.086	0.036	-	0.059	0.042	0.079
fe_ocean	0.04	0.036	0.045	-	-	0.106	0.3
ok2010	0.116	0.123	0.084	-	-	0.316	0.121
web-NotreDame	0.226	0.178	0.319	-	-	0.492	0.314
coPapersCiteseer	-	0.185	0.249	-	-	0.245	0.215
gsm_106857	-	0.089	0.102	-	-	0.131	0.106
tx2010	0.13	0.159	0.058	-	-	0.705	0.137
com-Youtube	0.179	-	0.275	-	-	0.367	0.427
Flan_1565	-	-	0.093	-	-	0.093	0.089
com-LiveJournal	-	-	0.206	-	-	0.305	0.27

C. Visual Comparison

Table VII shows the drawings created by all tested methods on all graphs, with edges drawn half-transparent to limit visual clutter. As in Tables IV and V, missing entries indicate methods that fail due to memory constraints or took too long to complete. We see that exact tsNET* cannot handle graphs larger than *fe_4elt2* (11143 nodes) due to its high runtime cost. Approximate tsNET* scales a bit better given its faster Barnes-Hut t-SNE implementation but hits a limit beyond *ship_003* (121728 nodes). NNP-NET can handle all graphs. For the smaller graphs, NNP-NET yields drawings that are visually very similar to the ground-truth ones from approximate tsNET*. Interestingly,

TABLE VI
SHEPARD DIGRAMS SHOWING DISTANCE PRESERVATION FOR ALL TESTED GRAPHS AND METHODS.

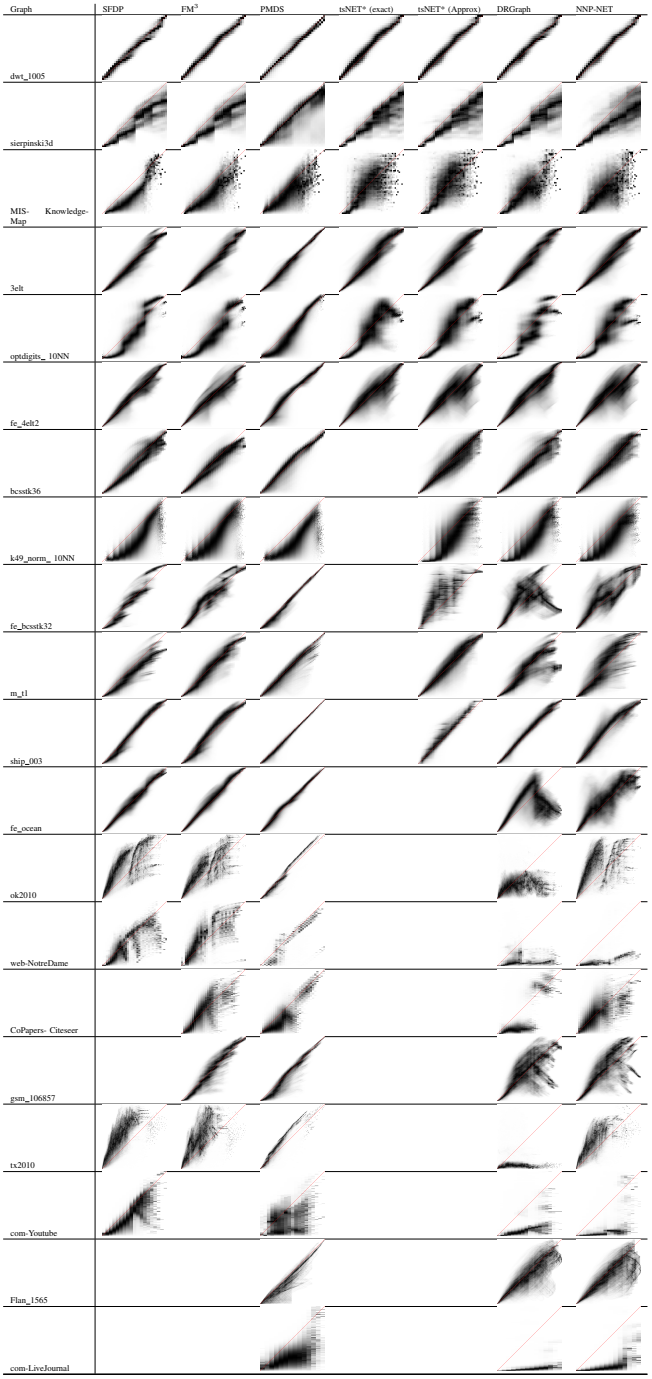
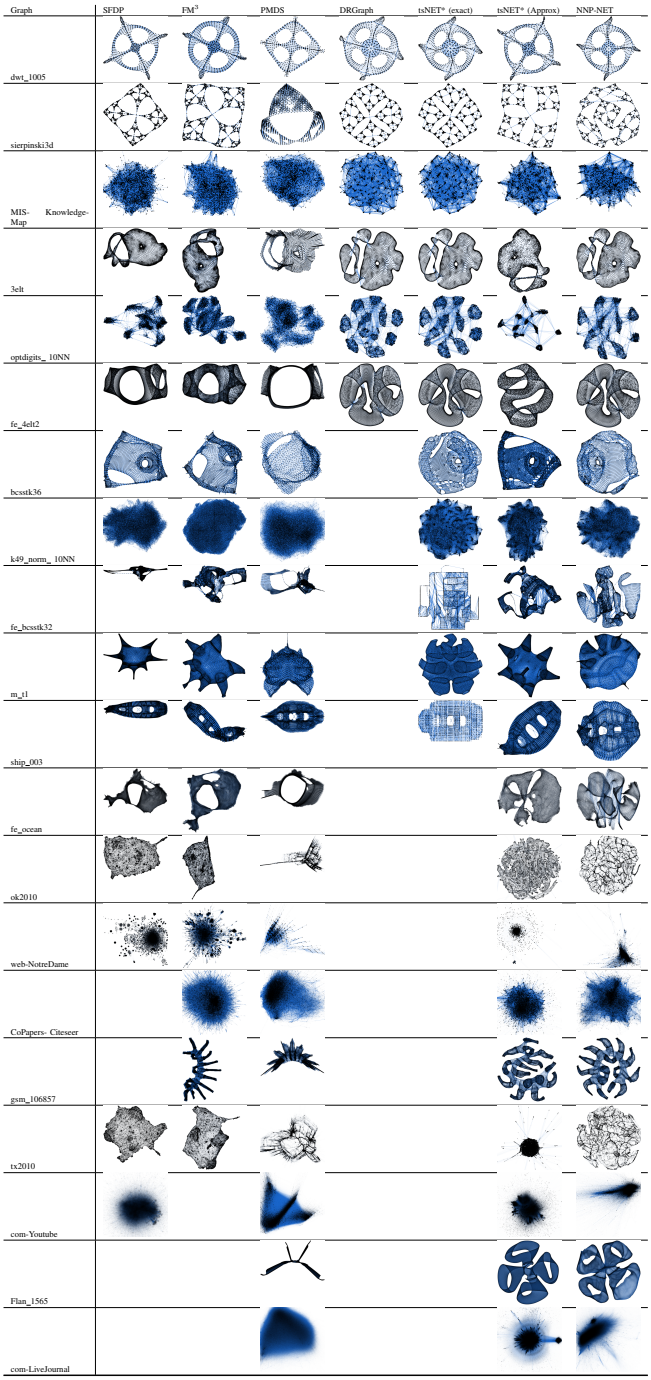


TABLE VII
DRAWINGS CREATED BY THE TESTED METHODS ON VARIOUS GRAPHS.



for *fe_bcsstk32* and *ship_003*, approximate tsNET* shows unexpected behavior where all nodes snap to a grid structure. NNP-NET does not copy this behavior but yields more plausible drawings. For larger graphs (*bcsstk36* and larger), NNP-NET creates different drawings from the other tested methods that can handle such dataset sizes, *i.e.*, SFDP, FM^3, PMDS, and DRGraph. NNP-NET spreads the drawing better over the 2D space. We argue that this is desirable as it allows one to better disentangle the depicted structures.

D. Drawing Weighted Graphs

NNP-NET uses the edge weights of the graph if they are provided. To show their effect, we ran NNP-NET on a set of weighted graphs with and without using the weights (Fig. 3). We see that weights affect indeed the obtained drawings (as they should), most visibly for *ok2010* and *tx2010*, where the unweighted drawings show more clutter and the weighted ones a more structured, network-of-corridors-like, one. Table VIII shows that both stress and neighborhood preservation improved when including weights.

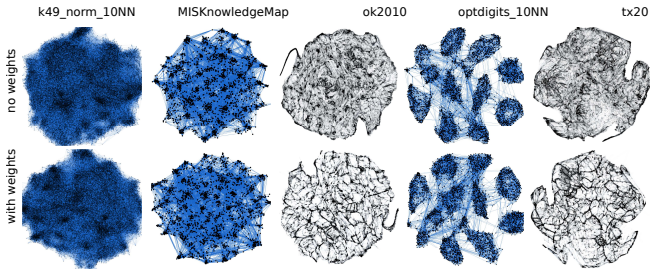


Fig. 3. Comparison of NNP-NET drawings with and without using weights (Sec. IV-D).

TABLE VIII
QUALITY METRICS FOR NNP-NET DRAWING WHEN USING *vs* NOT USING EDGE WEIGHTS (SEC. IV-D).

Graph	Stress σ		Neighborhood preservation ν	
	weights	no weights	weights	no weights
k49_norm_10NN	**0.171**	0.172	**0.081**	0.077
MISKnowledge-Map	0.185	**0.183**	**0.376**	0.364
ok2010	**0.118**	0.122	**0.433**	0.382
optdigits_10NN	**0.139**	0.143	**0.613**	0.611
tx2010	**0.138**	0.144	**0.340**	0.322

E. Execution Time

Table IX shows the execution times for all methods tested with missing values as already explained. NNP-NET takes significantly longer on smaller graphs due to the dominating high constant cost of computing the ground truth. For larger graphs, NNP-NET is slower than PMDS. However, as already discussed, NNP-NET yields significantly better neighborhood preservation (Tab. IV) and can disentangle graphs for which PMDS produces visually poor layouts (Tab. VII). Compared to DRGraph, we see a pattern reversal, with NNP-NET becoming faster for *Flan_1565* and *com-LiveJournal*. To better examine scalability, Fig. 4 shows the time each step of NNP-NET takes *vs* the node count N (both axes use a logarithmic scale). For ease of comparison, the bottom image in the figure shows all the above timings using the same y scale. The following trends are visible:

Embedding creation is roughly linear with N. The spiking outliers in the plot correspond to *weighted* graphs which require Dijkstra's algorithm (which is $O(N \log N)$ instead of the much faster Breadth-First Search (BFS, which is $O(N)$) to compute shortest paths.

Subgraph creation follows the same linear trend in N. The three outliers indicate graphs where our coarsening method could not reach the desired node count $S = 10^4$ and had to use the much slower pivot method (Sec. III-B).

Ground truth creation is roughly constant for $N > 10^4$ nodes, a value matching S.

Training and inference are roughly linear up to about 10^4 nodes, mainly due to the training cost. After this point, training time becomes constant. The low curve slope for larger N values tells that inference costs relatively very little and scales very well with N.

We further compare NNP-NET's scalability with DRGraph, its strongest competitor *vs* ability to handle very large graphs with good quality values. For this, we took the following steps. (1) We ran NNP-NET using BFS for the weighted graphs when calculating graph distances, *i.e.*, ignoring weights – a fair comparison since DRGraph does not use weights. (2) When using the slow pivot method for subgraph creation, we accounted a time of 1 second, equal to the largest cost for all cases where this slow fallback was not needed.

Figure 5 compares the cost of DRGraph (blue) with NNP-NET ran normally (orange) and with the theoretical cost of NNP-NET where we would not need the slow pivot method (steps 1 and 2 above; green). DRGraph shows a linear time complexity. In contrast, NNP-NET starts with a much higher execution time. For over roughly $N = 10^4$ nodes, its cost appears almost constant (green curve) if we eliminate the

TABLE IX
EXECUTION TIME IN SECONDS FOR ALL TESTED METHODS, ALL GRAPHS (SEC. IV-E). BOLD VALUES INDICATE FASTEST METHOD PER GRAPH.

Graph	Execution time per method						
	SFDP	FM^3	PMDS	tsNET* $_{exact}$	tsNET* $_{approx}$	DRGraph	NNP-NET
dwt_1005	0.34	0.12	**0.02**	1.44	2.61	0.06	10.13
sierpinski3d	0.75	0.12	1.37	6.52	5.42	**0.11**	11.38
MISKnowledgeMap	1.23	0.76	1.20	40.42	13.71	**0.13**	24.27
3elt	1.71	0.20	**0.07**	110.70	18.92	0.27	21.14
optdigits_10NN	2.59	0.58	2.38	155.46	59.68	**0.30**	36.88
fe_4elt2	6.06	0.22	**0.15**	931.18	107.97	0.59	58.96
bcsstk36	10.89	2.28	**0.53**	-	288.91	1.49	56.96
k49_norm_10NN	37.28	**2.14**	37.13	-	1718.79	2.66	156.79
fe_bcsstk32	28.25	3.07	**2.47**	-	572.96	3.64	86.68
m_t1	77.61	14.22	**4.04**	-	1133.24	8.21	87.81
ship_003	94.44	14.24	**5.85**	-	1456.06	9.61	66.40
fe_ocean	117.74	**1.80**	2.28	-	-	14.39	94.24
ok2010	260.64	**4.13**	44.83	-	-	23.98	94.18
web-NotreDame	270.48	10.26	**6.13**	-	-	25.49	166.56
coPapersCiteseer	-	99.34	**23.69**	-	-	45.49	98.20
gsm_106857	-	57.28	**35.35**	-	-	55.86	160.41
tx2010	1165.55	**12.95**	200.21	-	-	83.18	254.17
com-Youtube	2209.56	-	**23.47**	-	-	132.47	1293.10
Flan_1565	-	-	**55.92**	-	-	171.44	124.12
com-LiveJournal	-	-	**220.14**	-	-	618.30	394.85

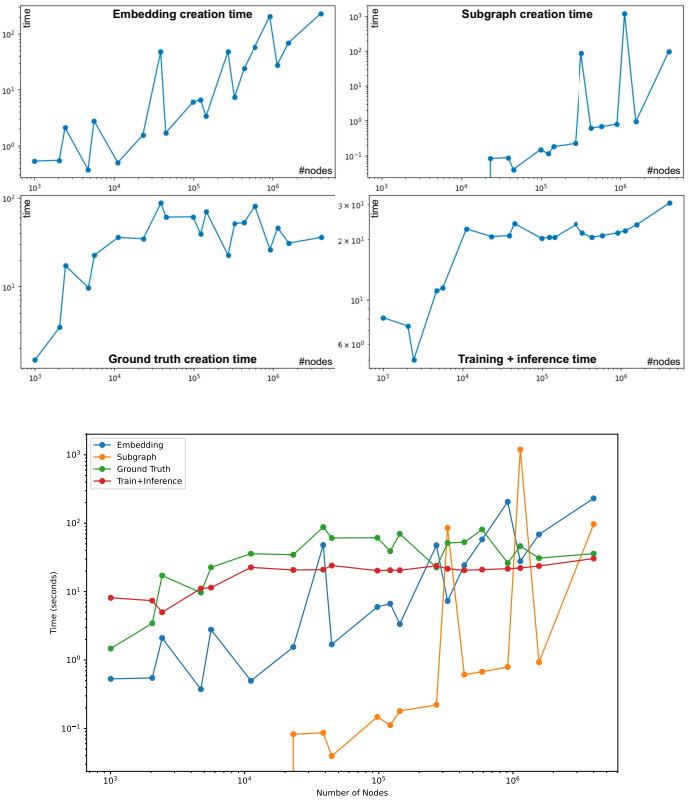


Fig. 4. Execution time of each component of NNP-NET (Sec. IV-E).

effects due to (1) and (2). Actual timings show the same trend, albeit with a bit more noise (orange curve). As Tab. IX also showed, we see how NNP-NET overtakes DRGraph at about $N = 10^6$ nodes.

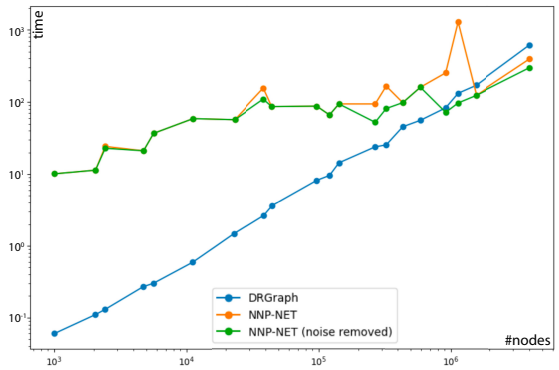


Fig. 5. Execution time comparison between DRGraph, NNP-NET, and theoretical execution time of NNP-NET with reduced noise (see Sec. IV-E).

We further show that NNP-NET is faster, and also yields better quality, than DRGraph on very large graphs by running both methods on the datasets in Tab. X. To our knowledge, the sizes of these surpass most benchmark graphs used in all GD papers we are aware of. We also tried PMDS on these graphs – the method failed to run producing degenerate layouts. To handle such sizes, we used a machine with 1024 GB of RAM and two AMD 7313 processors (16 cores each). Both are able to take advantage of multiple cores. Results show very high stress for DRGraph (which is undesired) while NNP-NET scores better stress. NNP-NET also performed better for neighborhood preservation on all graphs except *asia_osm*. Overall, we thus see that, as graph sizes increase, NNP-NET overtakes DRGraph in both quality metrics (compare Tab. X with Tabs. V and IV.)

Speedwise, NNP-NET is 5 to 10 times faster than DRGraph thereby confirming the 'takeover' trend observed in Fig. 5. We also note that DRGraph failed to handle several nodes in such large graphs (see entries marked (*) in Tab. X. We also see that NNP-NET yields layouts which look visually more informative of the graph structure than DRGraph (Tab. XI).

TABLE X
VERY LARGE GRAPHS USED TO TEST COMPUTATIONAL SCALABILITY (TOP). FOR NNP-NET AND DRGRAPH: STRESS AND NEIGHBORHOOD PRESERVATION VALUES (MIDDLE) AND EXECUTION TIMES (BOTTOM). ALL GRAPHS COME FROM SPARSESUITE [50]. THE DRGRAPH RESULTS MARKED (*) HAD SOME NODES WITH POSITION (-NAN, -NAN). WE PLACED SUCH NODES AT POSITION $(0, 0)$ IN ORDER TO BE ABLE TO COMPUTE QUALITY METRICS.

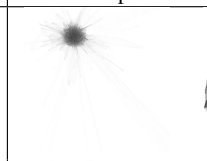
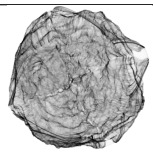
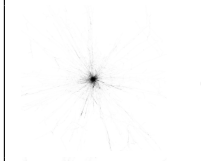
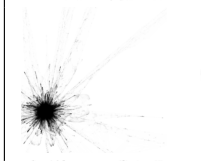
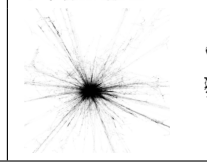
Graph dataset	$\|V\|$	$\|E\|$	**Weights**
delaunay_n23	8388608	50331568	no
asia_osm	11950757	25423206	no
hugetrace_00020	16002413	47997626	no
europe_osm	50912018	108109320	no

	Stress (lower is better)	
Graph	DRGraph	NNP-NET
delaunay_n23	0.86555	0.635423
asia_osm	0.873344(*)	0.265117
hugetrace_00020	0.907514	0.038513
europe_osm	0.833412(*)	0.122452

	Neighborhood preservation (higher is better)	
Graph	DRGraph	NNP-NET
delaunay_n23	0.052861	0.403406
asia_osm	0.491982(*)	0.276969
hugetrace_00020	0.051325	0.576327
europe_osm	0.167938(*)	0.254496

	Execution time (seconds)		
Graph	DRGraph	NNP-NET	Speed-up
delaunay_n23	2940.6	504.1	5.87
asia_osm	4171.0	456.5	9.14
hugetrace_00020	4322.7	603.0	7.16
europe_osm	15491.1	1668.1	9.28

TABLE XI
LAYOUTS CREATED FOR VERY LARGE GRAPHS USING DRGRAPH AND NNP-NET.

Graph	DRGraph	NNP-NET
delaunay_n23		
asia_osm		
hugetrace_00020		
europe_osm		

V. RESULTS ON DYNAMIC GRAPHS

A. Visual Comparison

We tested the ability of NNP-NET to handle dynamic graphs by a set of experiments of increasing complexity as follows.

Stability test: A good dynamic graph drawing method should not yield large/spurious changes in the layout when the input graph changes only slightly. To test this, we took the graph *fe_ocean* (143437 nodes) and created 9 time steps by removing 50 random nodes at each timestep t. Table XII shows NNP-NET's layout results with nodes colored using a fixed gradient to allow spotting the same nodes in different timesteps. As expected, the layouts for all timesteps look almost identical, showing that NNP-NET is *stable* upon small input changes. This proves empirically our stability claim listed in Sec. 1.

Increasing changes: We next tested NNP-NET's behavior on a dynamic graph exhibiting larger changes. For this, we used the *3elt* graph, from which we picked and removed 5 random nodes. At each subsequent timestep, we expanded the 'holes' created in the graph by this removal by deleting the nodes connected to each hole. At the same time, we add 4 nodes at a time to the graph in order to fill an already existing hole that is present in the original graph. These new nodes form a grid structure to patch up the hole. Added nodes are visualized used a yellow color. Table XIII shows the drawing of this dynamic graph. As visible, the layout visibly changes between

TABLE XII
LAYOUT STABILITY TEST UPON SMALL INPUT CHANGES. THE DYNAMIC GRAPH IS OBTAINED BY REMOVING 50 RANDOM NODES AT EACH TIMESTEP FROM THE *fe_ocean* GRAPH.

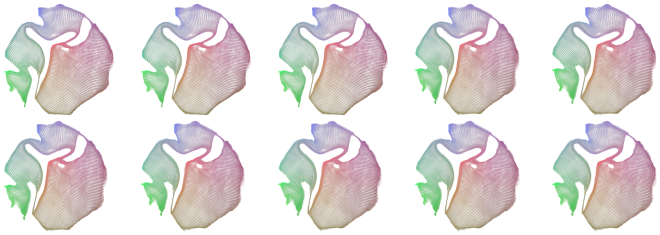
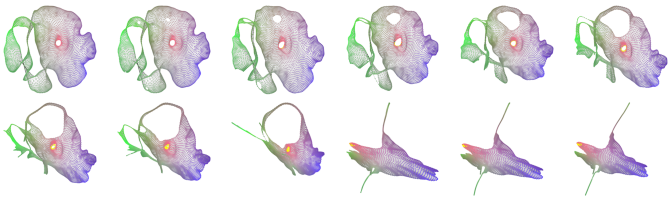
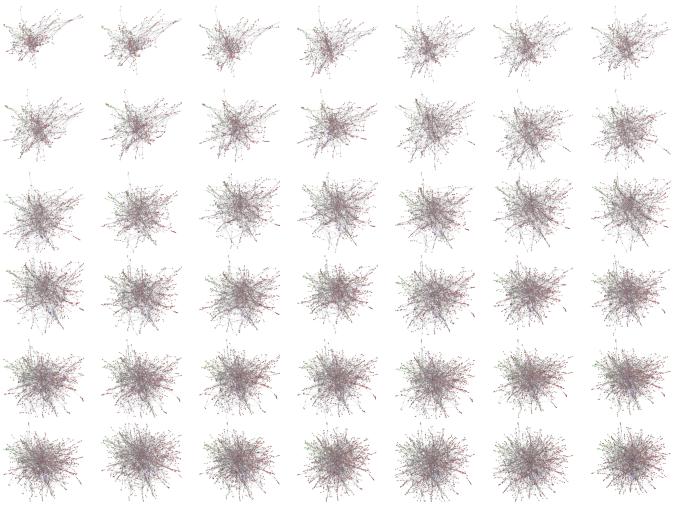


TABLE XIII
DYNAMIC DRAWING OF A GRAPH WITH INCREASING, ACCUMULATED, DELETIONS. NOTE HOW HOLES IN THE GRAPH INCREASE OVER TIME. IN THE LATTER PART, THE GRAPH STRUCTURE CHANGES SIGNIFICANTLY DUE TO DISCONNECTIONS BETWEEN ITS DIFFERENT PARTS.



timesteps as the underlying graph also changes significantly. In particular, we see how the holes grow over time. Towards the end of the sequence, the layout changes more drastically as the 'bridges' formed by nodes between different parts of the graph disconnect due to our repeated deletions. Importantly, we see that, even though the graph structure changes significantly as compared to what NNP-NET has learned, (1) the resulting layouts still look plausible and (2) the nodes that are persistent across time-frames get positioned in the same parts of the layout.

TABLE XIV
DYNAMIC DRAWING OF A RANDOMLY GENERATED GRAPH. EACH TIME STEP ADDS 50 EXTRA NODES AND BETWEEN 50 AND 150 EXTRA EDGES STARTING FROM 500 NODES AND 1118 EDGES.



Random sequence: Our experiments so far consider relatively smooth and localized changes on graphs that have a mesh-like structure. To test NNP-NET against less regular changes on less structured graphs, we created a sequence based on random changes. We first used the dual-Barabási-Albert model [58] to generate an initial graph of 500 nodes and 1118 edges. We next add one node per time-step and randomly connect it to the existing ones by one edge (with 87.5% probability) or by three edges (with 12.5% probability). After 2000 such steps, we obtain a graph of 2500 nodes and 3672 edges. To amplify the change dynamics, we next consider as frames G_t of our dynamic graph to be drawn the timesteps placed 50 edits apart in the above sequence, yielding a total of $T = 2000/50 = 400$ frames. Table XIV shows the results of NNP-NET on this sequence. As for the previous experiments, we see how our method yields a stable set of drawings.

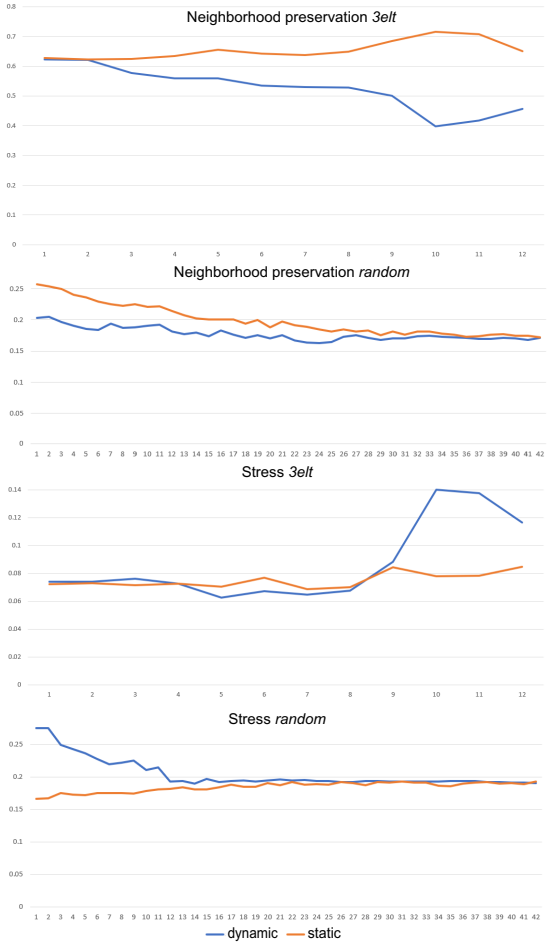


Fig. 6. Neighborhood preservation and stress metrics for the *3elt* and *random* dynamic graphs (Sec. V-B)

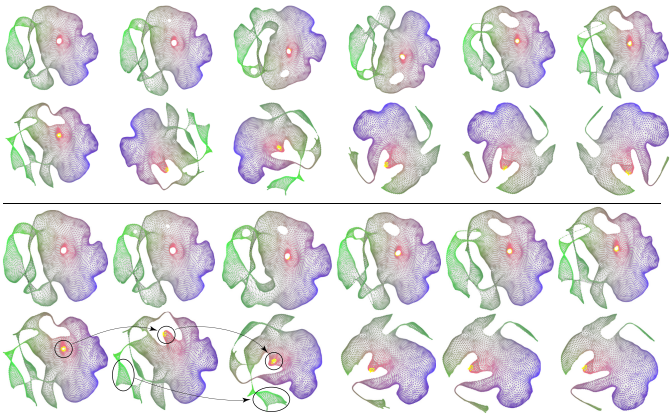
B. Quality metrics analysis for dynamic graph drawing

As outlined earlier, a good dynamic graph drawing should meet two main properties: (1) produce layout changes ΔP_t which are commensurate with the changes ΔG_t in the underlying graph sequence; and (2) produce layouts P_t of the individual frames G_t that have a good quality according to established GD metrics. Our experiments in Sec. V-A empirically showed that

NNP-NET complies with property (1). To test property (2), we next proceed as follows. For each frame G_t of a given dynamic graph $G(t)$, we consider its *dynamic* layout P_t as computed by the NNP-NET method proposed for drawing dynamic graphs as described in Sec. III-E. Additionally, we lay out each G_t by directly running the *static* NNP-NET method on the respective frame, that is, without considering the entire sequence $G(t)$. For both the dynamic and static layouts, we compute and compare the neighborhood preservation and stress metrics used earlier. Our hypothesis is that NNP-NET (when used for dynamic drawing) is a good method if it exhibits quality metric values which are close to its baseline, namely, NNP-NET ran on the individual frames.

Figure 6 shows these metric comparisons for the *3elt* and *random* dynamic graphs discussed earlier in Sec. V-A. As expected, the dynamic metrics tend to be slightly lower than the static (baseline) values. For *e3lt*, we see how the differences increase more significantly in the last part of the sequence. As explained earlier, this is the part where the graph structure changes significantly due to disconnections. Laying out each frame separately allows NNP-NET to keep quality high – however, the *sequence* obtained by depicting such layouts will not be stable and thus suited for visualizing the graph's dynamics.

TABLE XV
NNP-NET RESULTS WHEN RUNNING IT *independently* ON THE SAME FRAMES OF THE GRAPH IN FIG. XIII WITHOUT (TOP) AND WITH ALIGNMENT (BOTTOM).



We illustrate this by showing these frames in Fig. XV (top two rows) for the *3elt* graph. As visible, the drawings exhibit various flips and rotations between frames. In contrast, the dynamic NNP-NET method (see Fig. XIII) keeps the desired stability at the expense of lower quality values for frames undergoing significant changes. Even more importantly, the layouts obtained with dynamic NNP-NET show a *sequence* in which we can trace the accumulated changes in the dynamic graph – see the discussion concerning the growing holes and deleted 'bridges' related to Fig. XIII. Even if we use alignment of the separately-computed frames to eliminate flips and rotations – see Fig. XV (bottom) – such a progression of changes is not easily traceable. For example, the graph area around the small hole showing red nodes, and the light-green component, start jumping around between consecutive frames (see black markers in the figure). Global alignment of such independently computed frames cannot fix such problems.

Note that identical trade-offs between stability and visual quality were observed when projecting time-dependent high-dimensional datasets using various DR methods [59] and, independently, when drawing time-dependent hierarchies using various treemapping algorithms [60]. As in our case, the solution for stability involved considering a *global* layout that takes into account all time-steps of the evolving process rather than a per-timestep layout.

VI. DISCUSSION

We next discuss how NNP-NET fulfills the contributions outlined earlier in Sec. 1.

Scalability: NNP-NET has theoretical linear time complexity in the graph node count N. In practice, it takes significantly more time than other linear time complexity GD methods for smaller graphs, becoming more competitive for larger graphs, eventually outperforming DRGraph for graphs around 1M nodes, and being 5 to 10 times faster than DRGraph for graphs of 10M up to 50M nodes respectively.

Layout quality: Ideally, layouts generated by our method should be of similar quality to the original tsNET* results. On smaller graphs, where we use the entire graph as ground truth, our results are very close to the approximate tsNET* results. Layouts generated for larger graphs using NNP-NET visually look good, and have similar quality metric values, when comparing with competing methods. For very large graphs, we consistently obtain better layout quality in terms of metrics (Tab. X) than DRGraph, our closest competitor for such sizes.

Robustness: NNP-NET could handle all the tested graphs with good results. No component in NNP-NET's pipeline uses complex (parameter-dependent) heuristics. Moreover, we used the same set of fixed parameters for all layouts. As such, we can reasonably claim that our method should be able to handle any graph dataset.

Ease of use: While our method has a number of parameters that could be adjusted, all of these were set to the same fixed values for all tests (Tab. II right), yielding consistent good results. As such, we claim our method is practically parameter-free.

Edge weights: NNP-NET directly handles weighted graphs – something that other methods such as DRGraph cannot. Including weights in the layout computation – apart from the fact that some use-cases require this – has a positive effect on the layout quality (Sec. IV-D).

We next discuss the three key steps of NNP-NET and outline strengths, limitations, and potential ways to overcome the latter.

Embedding method: NNP-NET uses PMDS to create a high-dimensional embedding vector per node. We also tested using the *distance-to-pivot* method for the embedding. This method is faster (as only a subset of the work would have to be done), but gave worse quality results in our testing (Appendix A). Since creating the embedding becomes the cost bottleneck for large graphs, it is worth further studying this approach to see if it can be used to create similar quality layouts in less time. Separately, other layout techniques than PMDS could be used

to generate this embedding. Exploring such alternatives is a low-hanging fruit for further decreasing execution time.

Dynamic graphs: NNP-NET can also lay out dynamic (time dependent) graphs by a simple modification that trains on a subset of such a sequence. Given the out-of-sample property of NNP, our drawings have the desirable properties of stability on small input changes as well as smooth evolution for larger changes. Like for static graphs, the layout cost for dynamic graphs is also linear in the input size. As expected for dynamic graphs, we see a decrease in quality metrics (stress and neighborhood hit) which is a known price to pay to gain stability. Separately, for our method to work, all frames of the dynamic graph to draw should have an overlap of at least $p = 250$ common nodes (see Sec. III-E). We argue this is not a problem for methods, such as ours, which focus specifically on drawing very large graphs of tens of thousands up to millions of nodes. If the time frames of such graphs would not have 250 common nodes, the frame-to-frame changes would be huge (by definition) – so one should question whether drawing the dynamic graph to explore such changes is suitable in itself.

Ground truth: As one of our key goals was to accelerate tsNET*, we logically used tsNET* to create the ground truth that NNP learns from. Using layouts created by other GD methods could give better results. Reducing the size of subgraph G', used for training, clearly decreases quality (Appendix C), which can negate the benefit of using higher-quality, but slower, methods to lay out G'. Conversely, using faster, but lower quality methods to lay out G' can significantly speed up our method for smaller graphs, but can adversely affect quality. Finding the right balance here is a topic for future work.

Subgraph computation: We tested two methods to create the subgraph G' from the full graph G: using pivot points; and iteratively coarsening G. The coarsening method was chosen as it had a significantly lower execution time (for details, see Appendix D). Yet, this method likely distorts distances between nodes. The magnitude of these distortions and their impact is not tested directly. As already outlined, the coarsening method does not always reach the target node count S. While falling back to the pivot method allows NNP-NET to handle its input, this has a high time cost. Experimenting with different alternatives for the subgraph method is a topic we will look into in future work.

Parameter settings: Our method depends on multiple parameters like the number of embedding dimensions n, pivot count p, and subgraph size S. As for other machine learning approaches, we optimize for these hyperparameters on a given set of graphs – in our case, up to size $N = 10000$ nodes (see Appendices). Similar to other works in GD literature, we do not claim that these are *optimal* presets in any sense of the word. Yet, we show that these presets yield satisfactory quality results (and final drawings) for all graphs tested in the paper, up to 50M nodes. Analyzing how these presets can be fine-tuned to yield even higher quality and/or lower computational costs is an interesting direction for future work.

VII. Conclusion

We presented NNP-NET, a new graph drawing method that creates layouts for very large graphs in the style of tsNET. Our results are not only very similar to the original tsNET layouts, but also have a quality comparable to DRGraph, a competing method that also uses tsNET as its base. NNP-NET leverages the speed and simplicity of Neural Network Projections (NNP) to create graph layouts to achieve significantly lower running times, and comparable quality, on graphs over 1 million nodes, than its closest competitor, DRGraph – and actually consistently surpass DRGraph's quality metrics on the largest tested graphs.

NNP-NET also inherits the out-of-sample ability of NNP which allows it to draw dynamic (time-dependent) graphs with a good balance between stability upon input changes and drawing quality metrics. Like for static graphs, the cost of drawing dynamic graphs is theoretically linear in the input size.

Since all steps around NNP can be freely replaced, future work can focus on improvements that these steps can provide: A different embedding method could give better resulting graphs or reduce execution time. Changing the ground truth method would allow for different drawing styles. Changing the subgraph extraction method is the most technical important aspect to look into further, as can boost speed significantly for the graphs that our current fast coarsening heuristic cannot handle. Separately, as we can now (pre)compute the layout of such very large graphs, interactive exploration methods of datasets of this size can be readily tried out to support many practical applications [11], [12].

References

[1] P. Eades, "A heuristic for graph drawing," *Congressus numerantium*, vol. 42, pp. 146–160, 1984.

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

[3] U. Brandes and C. Pich, "Eigensolver methods for progressive multidimensional scaling of large data," in *Proc. Graph Drawing*. Springer, 2007, pp. 42–53.

[4] K. M. Hall, "An r-dimensional quadratic placement algorithm," *Management science*, vol. 17, no. 3, pp. 219–229, 1970.

[5] F. Grötschla, J. Mathys, R. Veres, and R. Wattenhofer, "CoRe-GD: A Hierarchical Framework for Scalable Graph Visualization with GNNs," in *Proc. ICLR*, 2024.

[6] L. Giovannangeli, F. Lalanne, D. Auber, R. Giot, and R. Bourqui, "Deep neural network for drawing networks, $(DNN)^2$," in *Proc. Graph Drawing*. Springer, 2021, pp. 375–390.

[7] M. Tiezzi, G. Ciravegna, and M. Gori, "Graph neural networks for graph drawing," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 35, no. 4, pp. 4668–4681, 2022.

[8] E. R. Gansner, Y. Koren, and S. C. North, "Graph drawing by stress majorization," in *Proc. Graph Drawing*. Springer, 2005, pp. 239–250.

[9] J. B. Kruskal and J. B. Seery, "Designing network diagrams," in *Proc. General Conference on Social Graphics*, 1980, pp. 22–50.

[10] D. Harel and Y. Koren, "Graph drawing by high-dimensional embedding," *J. of Graph Algorithms and Applications*, vol. 8, no. 2, pp. 195–214, 2004.

[11] T. von Landesberger, A. Kuijper, T. Schreck, J. Kohlhammer, J. van Wijk, J.-D. Fekete, and D. Fellner, "Visual analysis of large graphs: State-of-the-art and future research challenges," *Computer Graphics Forum*, vol. 30, no. 6, pp. 1719–1749, 2011.

[12] D. Auber, "Tulip : A huge graph visualisation framework," in *Mathematics and Visualization*. Springer, 2004, pp. 105–126. [Online]. Available: https://github.com/Tulip-Dev

[13] O.-H. Kwon, T. Crnovrsanin, and K.-L. Ma, "What would a graph look like in this layout? a machine learning approach to large graph visualization," *IEEE TVCG*, vol. 24, no. 1, pp. 478–488, 2018.

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

[15] L. van der Maaten and G. Hinton, "Visualizing data using t-SNE," *Journal of Machine Learning Research*, vol. 9, no. 11, 2008.

[16] N. Pezzotti, J. Thijssen, A. Mordvintsev, T. Hollt, B. van Lew, B. Lelieveldt, E. Eisemann, and A. Vilanova, "GPGPU linear complexity t-SNE optimization," *IEEE TVCG*, vol. 26, no. 1, pp. 1172–1181, 2020.

[17] D. M. Chan, R. Rao, F. Huang, and J. F. Canny, "t-SNE-CUDA: GPU-accelerated t-SNE and its applications to modern data," in *Proc. Computer Architecture and High Performance Computing*, 2018.

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

[19] M. Espadoto, N. Hirata, T. Sumiko, and A. C. Telea, "Deep learning multidimensional projections," *Information Visualization*, vol. 19, no. 3, pp. 247–269, 2020.

[20] I. Hartskeerl, T. Mchedlidze, S. van Wageningen, P. Vangorp, and A. Telea, "NNP-NET: Accelerating t-sne graph drawing for very large graphs by neural networks," in *Proc. Graph Drawing*, 2025.

[21] H. Gibson, J. Faith, and P. Vickers, "A survey of two-dimensional graph layout techniques for information visualisation," *Information Visualization*, vol. 12, no. 3-4, pp. 324–357, 2013.

[22] R. Tamassia, *Handbook of graph drawing and visualization*. CRC Press, 2013.

[23] M. Jacomy, T. Venturini, S. Heymann, and M. Bastian, "ForceAtlas2, a Continuous Graph Layout Algorithm for Handy Network Visualization Designed for the Gephi Software," *PLOS ONE*, vol. 9, no. 6, p. e98679, 2014.

[24] S. C. North, *Drawing graphs with NEATO*, 2004, NEATO User's Manual. [Online]. Available: https://www.graphviz.org/pdf/neatoguide.pdf

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

[26] M. Gronemann, "Engineering the fast-multipole-multilevel method for multicore and SIMD architectures," *Master's thesis. Technische Univ. Dortmund*, 2009.

[27] F. V. Paulovich, A. Arleo, and S. van den Elzen, "When dimensionality reduction meets graph (drawing) theory: Introducing a common framework, challenges and opportunities," *Computer Graphics Forum*, vol. 44, no. 3, p. e70105, 2025.

[28] W. S. Torgerson, "Multidimensional scaling: I. Theory and method," *Psychometrika*, vol. 17, no. 4, pp. 401–419, 1952.

[29] M. Espadoto, R. M. Martins, A. Kerren, N. S. T. Hirata, and A. C. Telea, "Toward a quantitative survey of dimension reduction techniques," *IEEE TVCG*, vol. 27, no. 3, pp. 2153–2173, 2019.

[30] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Proc. NIPS*, vol. 26, 2013.

[31] R. Ahmed, F. De Luca, S. Devkota, S. Kobourov, and M. Li, "Multicriteria scalable graph drawing via stochastic gradient descent, $(SGD)^2$," *IEEE TVCG*, vol. 28, no. 6, pp. 2388–2399, 2022.

[32] R. Ahmed, F. DeLuca, S. Devkota, S. Kobourov, and M. Li, "Graph drawing via gradient descent, $(GD)^2$," in *Proc. Graph Drawing*. Springer, 2020, pp. 3–17.

[33] J. X. Zheng, S. Pawar, and D. F. M. Goodman, "Graph drawing by stochastic gradient descent," *IEEE TVCG*, vol. 25, no. 9, pp. 2738–2748, 2019.

[34] S. Devkota, R. Ahmed, F. D. Luca, K. E. Isaacs, and S. Kobourov, "Stress-plus-x (spx) graph layout," in *Proc. Graph Drawing*. Springer, 2019, pp. 291–304.

[35] L. Giovannangeli, F. Lalanne, D. Auber, R. Giot, and R. Bourqui, "Toward efficient deep learning for graph drawing (DL4GD)," *IEEE TVCG*, vol. 30, no. 2, pp. 1516–1532, 2024.

[36] X. Wang, K. Yen, Y. Hu, and H.-W. Shen, "DeepGD: A deep learning framework for graph drawing using GNN," *IEEE CGA*, vol. 41, no. 5, pp. 32–44, 2021.

[37] Y. Wang, Z. Jin, Q. Wang, W. Cui, T. Ma, and H. Qu, "DeepDrawing: A deep learning approach to graph drawing," *IEEE TVCG*, vol. 26, no. 1, pp. 676–686, 2020.

[38] X. Wang, K. Yen, Y. Hu, and H.-W. Shen, "SmartGD: A gan-based graph drawing framework for diverse aesthetic goals," *IEEE TVCG*, vol. 30, no. 8, pp. 5666–5678, 2023.

[39] C. Bredius, Z. Tian, and A. C. Telea, "Visual Exploration of Neural Network Projection Stability," in *Proc. MLVis*, 2022.

[40] F. Beck, M. Burch, S. Diehl, and D. Weiskopf, "A taxonomy and survey of dynamic graph visualization," *Computer Graphics Forum*, vol. 36, no. 1, pp. 133–159, 2017.

[41] Y. Frishman and A. Tal, "Dynamic drawing of clustered graphs," in *Proc. IEEE Symp. on Information Visualization*, 2004, pp. 191–198.

[42] E. F. Vernier, R. Garcia, I. P. d. Silva, J. L. D. Comba, and A. C. Telea, "Quantitative evaluation of time-dependent multidimensional projection techniques," *Computer Graphics Forum*, vol. 39, no. 3, pp. 241–252, 2020.

[43] P. E. Rauber, A. X. Falcão, and A. C. Telea, "Visualizing time-dependent data using dynamic t-SNE," in *Proc. EuroVis – Short Papers*, 2016, p. 73–77.

[44] L. van der Maaten, "Learning a parametric embedding by preserving local structure," in *Proc. AISTATS*, 2009.

[45] J. C. Gower, "Generalized procrustes analysis," *Psychometrika*, vol. 1, no. 40, pp. 33–51, 1975.

[46] I. Hartskeerl, T. Mchedlidze, S. van Wageningen, P. Vangorp, and A. Telea, "NNP-NET source code," 2025. [Online]. Available: {https://github.com/IlanHartskeerl/NNP-NET}

[47] M. Chimani, C. Gutwenger, M. Jünger, G. W. Klau, K. Klein, and P. Mutzel, "The Open Graph Drawing Framework (OGDF)," in *Handbook of Graph Drawing and Visualization*, R. Tamassia, Ed. CRC Press, 2014, pp. 543–569.

[48] L. van der Maaten, "Accelerating t-SNE using tree-based algorithms," *Journal of Machine Learning Research*, vol. 15, no. 1, pp. 3221–3245, 2014.

[49] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/

[50] T. A. Davis and Y. Hu, "The University of Florida sparse matrix collection," *ACM Transactions on Mathematical Software*, vol. 38, no. 1, pp. 1–25, 2011.

[51] J. Ellson, E. R. Gansner, L. Koutsofios, S. C. North, and G. Woodhull, "Graphviz — open source graph drawing tools," in *Proc. Graph Drawing*. Springer, 2002, pp. 483–484.

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

[53] H. C. Purchase, R. F. Cohen, and M. James, "Validating graph drawing aesthetics," in *Proc. Graph Drawing*. Springer, 1996, pp. 435–446.

[54] P. Eades, M. E. Houle, and R. Webber, "Finding the best viewpoints for three-dimensional graph drawings," in *Proc. Graph Drawing*. Springer, 1997.

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

[56] U. Brandes and C. Pich, "An experimental study on distance-based graph drawing," in *Proc. Graph Drawing*. Springer, 2009, pp. 218–229.

[57] E. R. Gansner, Y. Hu, and S. C. North, "A maxent-stress model for graph layout," *IEEE TVCG*, vol. 19, no. 6, pp. 927–940, 2013.

[58] N. Moshiri, "The dual-Barabási-Albert model," *arXiv preprint arXiv:1810.10538*, 2018.

[59] E. Vernier, R. Garcia, I. da Silva, J. Comba, and A. Telea, "Quantitative evaluation of time-dependent multidimensional projection techniques," *Computer Graphics Forum*, vol. 39, no. 3, pp. 241–252, 2020.

[60] E. Vernier, M. Sondag, J. Comba, B. Speckmann, A. Telea, and K. Verbeek, "Quantitative comparison of time-dependent treemaps," *Computer Graphics Forum*, vol. 39, no. 3, pp. 393–404, 2020.

[61] P. Delicado and C. Pachón-García, "Multidimensional scaling for big data," *Advances in Data Analysis and Classification*, vol. 19, p. 649–670, 2025.

APPENDIX A
COMPARISON OF EMBEDDING METHODS

We further compare the two embedding methods described in Sec. III-A, namely (1) using the *distances-to-pivot* method and (2) the *projecting* method. Table XVI shows that the results obtained by the two embedding methods are visually very similar, except for *sierpinski3d*, where the distances-to-pivot method is farther from the ground-truth than the projection method.

Table XVII shows the stress, neighborhood preservation, and error *vs* the ground truth (measured by Euclidean distances between corresponding nodes in the two layouts). We see that the projection method scores a consistently lower error to ground truth compared to the distance-to-pivot method. It also has slightly better quality metric values – though these are less important since the aim of the embedding step is to recreate the ground truth as accurately as possible rather than creating the highest-quality layout. However, the projection method is significantly slower than the distance-to-pivot one.

TABLE XVI
LAYOUTS CREATED BY THE *distance-to-pivot* AND *projection* EMBEDDINGS.

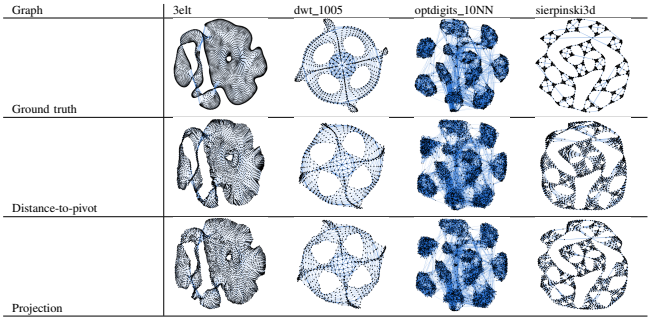


TABLE XVII
QUALITY METRICS OBTAINED USING THE *distance-to-pivot* AND *projection* EMBEDDING METHODS.

Graph	Embedding method	Time	Neighborhood preservation	Stress	Error *vs* ground truth
3elt	distance-to-pivot	**0.015**	0.499	0.078	0.024
3elt	projection	1.228	**0.547**	**0.077**	**0.014**
dwt_1005	distance-to-pivot	**0.007**	0.484	0.041	0.027
dwt_1005	projection	2.253	**0.521**	**0.039**	**0.019**
optdigits_10NN	distance-to-pivot	**0.799**	0.586	**0.134**	0.025
optdigits_10NN	projection	6.649	**0.598**	0.138	**0.017**
sierpinski3d	distance-to-pivot	**0.005**	0.462	**0.132**	0.028
sierpinski3d	projection	3.487	**0.498**	0.137	**0.017**

APPENDIX B
NUMBER OF EMBEDDING DIMENSIONS n AND PIVOTS p

We tested different values $n \in \{10, 25, 50, 100\}$ for the number of dimensions n to embed in (Sec. III-A). Table XVIII shows that the results are visually very similar for different values of n. Table XIX compares the stress, neighborhood preservation, and error to ground truth (the latter defined as in Appendix A). Neighborhood preservation and stress slightly improve with increasing n. More importantly, the error to ground truth decreases more visibly as n increases. At the same time, execution time increases quite strongly with n. Our setting $n = 50$ balances well quality and time.

TABLE XVIII
RESULTS FOR DIFFERENT EMBEDDING SIZES n.

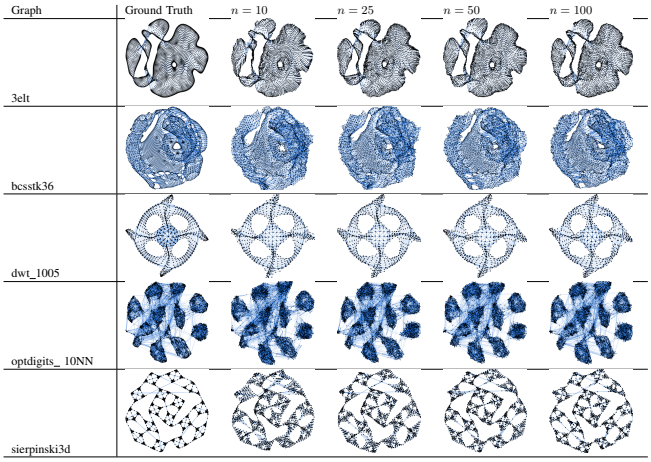


TABLE XIX
QUALITY METRICS FOR THE DIFFERENT EMBEDDING SIZES n.

Graph	n	Embedding Time	Neighborhood preservation	Stress	Error to GT
3elt	10	**15.5**	0.532	0.078	0.013
3elt	25	20.9	**0.580**	**0.077**	0.019
3elt	50	38.2	0.579	**0.077**	**0.011**
3elt	100	105.7	0.574	**0.077**	0.012
bcsstk36	10	**30.2**	0.411	0.102	0.027
bcsstk36	25	36.3	0.435	0.107	0.019
bcsstk36	50	54.4	0.457	**0.105**	**0.014**
bcsstk36	100	118.9	**0.463**	0.107	0.016
dwt_1005	10	**5.5**	0.504	**0.033**	0.018
dwt_1005	25	11.3	0.542	0.035	**0.012**
dwt_1005	50	28.8	0.543	0.035	**0.012**
dwt_1005	100	94.0	**0.544**	0.034	**0.012**
optdigits_10NN	10	57.7	0.607	**0.139**	0.029
optdigits_10NN	25	56.6	0.621	0.140	0.019
optdigits_10NN	50	**45.6**	0.623	0.140	0.015
optdigits_10NN	100	106.9	**0.624**	0.140	**0.014**
sierpinski3d	10	**10.7**	0.495	**0.126**	0.023
sierpinski3d	25	17.8	**0.509**	0.127	0.019
sierpinski3d	50	35.6	0.507	0.131	0.021
sierpinski3d	100	99.2	0.502	0.132	**0.014**

For the number of pivots p in PMDS, we used the fixed setting $p = 250$ which is recommended as default in OGDF [47] and Tulip [12], among other drawing frameworks for large graphs. A recent analysis of MDS-class algorithms examined multiple p settings and also recommended a default of $p = 250$ as a good balance between accuracy and speed [61].

APPENDIX C
SETTING THE EXTRACTED SUBGRAPH SIZE S

We have compared results obtained for different values $S \in \{1000, 2500, 5000, 10000\}$ for the size S of the coarsened subgraph $\mathbf{G}'$ (Sec. III-B). Table XX shows these results. We see that S can significantly affect the obtained layouts: for *k48_norm_10NN*, the results for $S = 10000$ show visibly more clustering than those for $S = 1000$. For *ok2010*, the results for $S = 10000$ show more individual pathways than for $S = 1000$. For *ship_003*, a bend appears for $S = 1000$, which is not present in any drawing created with higher S.

Table XXI examines the neighborhood preservation, stress, and execution time for different S values. Neighborhood preservation clearly improves with increasing S. In contrast, stress does not show a clear correlation with S. Execution time, as expected, significantly increases with S. Since, however, visual results seem to be strongly affected by S, and since we

see improvement in neighborhood preservation with S, we can claim that higher S values will lead to better layouts. As such, we choose conservatively for a high default value $S = 10000$.

TABLE XX
RESULTS USING DIFFERENT SIZES S FOR THE SUBGRAPH $\mathbf{G}'$.

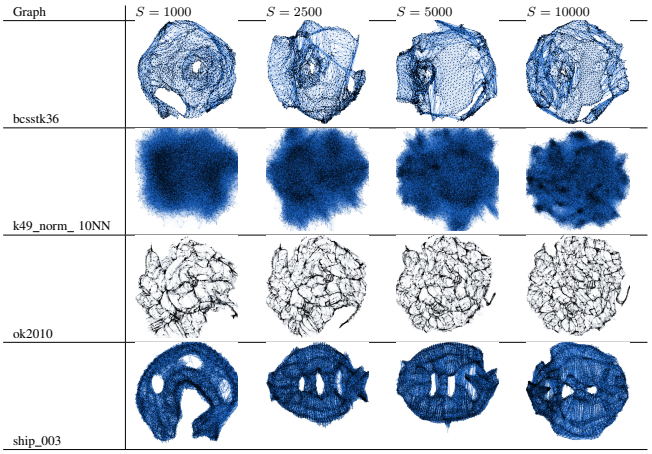


TABLE XXI
PERFORMANCE METRICS FOR DIFFERENT SIZES S OF SUBGRAPH $\mathbf{G}'$.

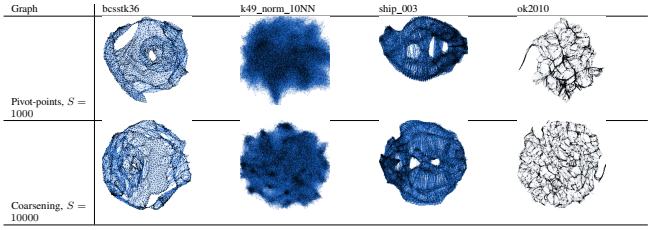
Graph	$S = \lvert\mathbf{G}'\rvert$	Neighborhood preservation	Stress	Time (seconds)
bcsstk36	1000	**0.411**	**0.085**	**32.4**
bcsstk36	2500	0.373	0.133	41.4
bcsstk36	5000	0.340	0.121	58.2
bcsstk36	10000	0.408	0.106	103.6
k49_norm_10NN	1000	0.030	0.170	**74.8**
k49_norm_10NN	2500	0.047	0.169	102.6
k49_norm_10NN	5000	0.060	**0.167**	127.1
k49_norm_10NN	10000	**0.083**	0.169	219.0
ok2010	1000	0.392	0.111	**90.2**
ok2010	2500	0.408	**0.109**	103.2
ok2010	5000	0.424	0.120	107.8
ok2010	10000	**0.436**	0.120	160.2
ship_003	1000	0.155	0.124	**45.1**
ship_003	2500	0.208	**0.073**	56.2
ship_003	5000	0.214	0.092	85.0
ship_003	10000	**0.229**	0.089	131.3

APPENDIX D
COMPARISON OF SUBGRAPH EXTRACTION METHODS

We next compare the *pivot-points* and *coarsening* methods for computing the subgraph $\mathbf{G}'$ from the input $\mathbf{G}$ (Sec. III-B). Both methods have the parameter S that gives the size of the subgraph $\mathbf{G}'$ to extract. Comparing results for the *same* S value is, however, not directly useful. Indeed, the pivot-points method becomes too slow to be practically useful for values $S > 1000$. In contrast, the coarsening method scales computationally well to higher values. As such, we compared the results that both methods produce when S is set to the maximal values they accept, namely $S = 1000$ (pivot-point) and $S = 10000$ (coarsening).

Table XXII shows the obtained results. The pivot-point method clearly yields suboptimal layouts due to its low $S = 1000$ setting – these are similar to what the coarsening method produces for $S = 1000$ (see Fig. XX). Table XXIII compares the values of neighborhood preservation, stress and, execution time for the two methods. The coarsening method yields overall higher neighborhood preservation values than pivot-points (except for *bcsstk36* where it is slightly lower). Stress values are however better for pivot-points, which is not too surprising, given that those are chosen based on PMDS. Overall, the significantly higher execution time of pivot-points (for larger graphs), even when using a S value ten times smaller than for the coarsening method, determined us to choose coarsening as the default technique for subgraph extraction.

TABLE XXII
RESULTS OBTAINED BY USING *pivot-points* AND *coarsening* TO EXTRACT THE SUBGRAPH $\mathbf{G}'$.



APPENDIX E
PSEUDO CODE OF THE FULL NNP-NET PIPELINE

```
// Generate n-dimensional Embedding using PMDS
E ← PMDS(G, n)

// Generate G'
G' ← G
G^next ← ({}, {})
while size(G'.V) > S:
    V^s ← sort(G'.V) // Sorted on degree
    C ← {} // Start with no clusters
    // Choose center nodes and create their clusters
    for v ∈ V^s WHERE v ∉ C:
        C.add({v, {v_i : (v, v_i) ∈ E, v_i ∉ C}})
    for c ∈ C:
        // Add center nodes to the next iteration
        G^next.V.add(c[0])
        // Add all edges for this center node
        for c' ∈ C WHERE (v, v') ∈ G'.E : v ∈ c, v' ∈ c':
            G^next.E.add({c[0], c'[0]})
    // Check if the graph still gets reduced
    // enough to continue
    if size(G^next.V) > size(G'.V)*0.95:
        G' ← pivotfallback(G^next, S)
    else :
        G' ← G^next
        G^next ← ({}, {})

// Generate ground truth
P^G ← tsNET*(G')
// Generating final positions
NNP ← train(E, P^G)
P ← inference(NNP, E)
P ← smoothing(P)
```

TABLE XXIII
PERFORMANCE METRICS OF THE *pivot-points* AND *coarsening* TO EXTRACT THE SUBGRAPH $\mathbf{G}'$.

Graph	Method + points	Neighborhood preservation	Stress	Time
bcsstk36	Pivot, 1000 points	**0.422**	**0.077**	**37.9**
bcsstk36	Coarsening, 10000 points	0.408	0.084	103.6
k49_norm_10NN	Pivot, 1000 points	0.045	**0.166**	**209.4**
k49_norm_10NN	Coarsening, 10000 points	**0.083**	0.171	219.0
ship_003	Pivot, 1000 points	0.194	**0.083**	**78.4**
ship_003	Coarsening, 10000 points	**0.229**	0.124	131.3
ok2010	Pivot, 1000 points	0.382	**0.087**	255.4
ok2010	Coarsening, 10000 points	**0.436**	0.112	**160.2**

Ilan Hartskeerl received both his BSc and MSc degrees in Computer Science from Utrecht University, and is since 2024 employed as a PhD student in the Visualization and Graphics group at Utrecht University. His research interests include graph drawing and virtual reality.

Tamara Mchedlidze is an Assistant Professor at the Department of Information and Computing Sciences at Utrecht University. Her research focuses on algorithms for network visualization and their applications to the visualization of sociological survey data, with the goal of bringing these data to the general public. She earned her PhD in Applied Mathematics from the National Technical University of Athens and was a postdoctoral researcher at Karlsruhe Institute of Technology before joining Utrecht University in 2020. Tamara serves as Co-Editor-in-Chief of Computational Geometry: Theory and Applications and regularly contributes to leading conferences in algorithms, graph drawing, and information visualization. She has received multiple Best Paper and Best Poster awards and is an active member of Women in Computer Science at Utrecht University, promoting equality and well-being in academia.

Simon van Wageningen earned his BSc degree in Computer Science and Neuroscience from University College Roosevelt, and his MSc degree in Data Science from Leiden University. Since 2021, he has been employed as a PhD student at the Department of Information and Computing Sciences at Utrecht University. His research focuses on graph drawing algorithms, the possible advantages of 3D graph drawing over 2D graph drawing, and the use of Machine Learning for graph drawing metric optimization.

Peter Vangorp is an assistant professor in the Visualization and Graphics (VIG) group at Utrecht University, Netherlands since 2022. He received his Ph.D. in Computer Science from KU Leuven, Belgium in 2009. His research interests include computer graphics, virtual reality, human visual perception, and data visualization.

Alexandru Telea received his PhD (2000) in Computer Science from the Eindhoven University of Technology. He was assistant professor in visualization and computer graphics at the same university (until 2007) and then full professor of visualization at the University of Groningen. Since 2019 he is full professor of visual data analytics at Utrecht University. His interests include high-dimensional visualization, visual analytics, and image-based information visualization.