NNP-NET: Accelerating t-SNE Graph Drawing for Very Large Graphs by Neural Networks

Ilan Hartskeerl
Department of Information and Computing Sciences, Utrecht University, The Netherlands

Tamara Mchedlidze
Department of Information and Computing Sciences, Utrecht University, The Netherlands

Simon van Wageningen
Department of Information and Computing Sciences, Utrecht University, The Netherlands

Peter Vangorp
Department of Information and Computing Sciences, Utrecht University, The Netherlands

Alexandru Telea
Department of Information and Computing Sciences, Utrecht University, The Netherlands

Abstract

tsNET is a recent graph drawing (GD) method that creates high quality layouts but suffers from a very high runtime. We present a new GD method, NNP-NET, which reduces tsNET's time complexity to generate layouts for very large graphs in seconds. Additionally, we extend tsNET to support drawing graphs with edge weights. We accomplish this by replacing tsNET's t-SNE projection with Neural Network Projection (NNP), a fast dimensionality reduction (DR) method that can imitate any given DR method. Our experiments show that NNP-NET gets good quality results when compared to other state-of-the art GD methods while yielding a better computational scalability.

2012 ACM Subject Classification Human-centered computing → Graph drawings; Computing methodologies → Neural networks

Keywords and phrases supervised graph drawing, dimensionality reduction, t-SNE

Digital Object Identifier 10.4230/LIPIcs.GD.2025.22

Supplementary Material *Software (Source Code)*: `https://github.com/IlanHartskeerl/NNP-NET` [26], archived at `swh:1:dir:539da3e3adda73abc8bb1af0c2a5c59fb7abef49`

1 Introduction

Multiple techniques have been proposed to create straight-line drawings of graphs, including classical approaches such as force-directed methods [11, 16], spectral methods [4, 24] and recently machine learning approaches [23, 20, 39]. Dimensionality reduction (DR) methods take a particular approach to GD: They encode the graph structure in a high-dimensional space and map, or project, this data to the 2D space so as to preserve data structure [18, 31, 25]. DR methods offer high flexibility on the information to preserve during projection, can encode node or edge attributes, and are battle-tested in fields such as machine learning and data visualization.

tsNET [30], a relatively recent method in the DR class, uses the well-known t-SNE projection technique [42] to create high-quality graph layouts. Yet, tsNET has quadratic time and space complexity in the number of graph nodes, making it unsuitable for graphs with over a few thousand nodes. While the time complexity can be partially alleviated by GPU-based t-SNE implementations [35, 7], and multilevel schemes and sparse matrices [47] can reduce space complexity, a more radical approach is to entirely *replace* t-SNE. A good candidate here is NNP [14] – a deep-learning projection method that can imitate any DR algorithm,

has linear time complexity in input size, is very simple to use, is robust to small input perturbations, and has out-of-sample ability. Yet, NNP training requires a high-dimensional dataset rather than a distance matrix.

We address this challenge and propose NNP-NET, which replaces tsNET's costly t-SNE step with NNP in a simple and generic fashion, resulting in the following contributions:

Scalability: NNP-NET has a 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 can scale to such sizes;

Quality: NNP-NET produces layouts that visually look similar, and have similar quality metrics, to those created by state-of-the art GD methods;

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

Stability: NNP-NET inherits NNP's robustness to noise;

Ease of use: NNP-NET can be used out-of-the-box without tuning any parameters.

The structure of this paper is as follows. Section 2 presents relevant related work. Section 3 details our method. Section 4 compares NNP-NET with state-of-the-art GD methods on a wide range of graphs and using various quality metrics. Section 5 discusses NNP-NET's strengths and limitations. Finally, Sec. 6 concludes the paper.

2 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.

2.1 Graph Layout Methods

Hundreds of GD methods have been proposed in the past decades [19, 38]. 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 [16, 18, 28, 33]. 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 [27] and the fast multipole multilevel method (FM^3 [22]) 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 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 [34]. DR based methods encode the graph adjacency information E into a distance matrix and then use classical DR techniques (discussed next in Sec. 2.2) 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 [40]). However, this

approach has a time and space complexity of $O(N^2)$. PMDS [4] 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 [30] follows a similar idea but replaces MDS by t-SNE [42], one of the highest-quality existing DR methods [15]. 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[47] improves upon tsNET to achieve a 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 [32] 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 [3] improves upon the earlier method GD^2 [2] by using stochastic gradient descent like its predecessor [46]. SGD^2 [3] 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 [10] 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 [20, 21] and DeepGD [43], two recent methods in this area, were only trained to lay out graphs of about 100 nodes due to long training times. DeepDrawing [45] 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 [44] optimizes for different (combinations of) quality metrics just like SGD^2 [3], but is untested on larger graphs. Graph Neural Drawers (GND) [39] uses graph neural networks and can handle graphs of about 10K nodes. GND has a time complexity of $O(|E|^{\frac{3}{2}})$. CoReGD [23] 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.

2.2 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 $\mathbf{Y} = \{\mathbf{y}_i\} \subset R^m$, $m \ll n$ which preserves the so-called *data structure* of $\mathbf{X}$, such that points close (resp. far) in R^n are mapped to points close (resp. far) in 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 [15].

NNP (Neural Network Projection) [14] 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 [6]; 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.

3 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. Psuedo code for the pipeline can be found in Appendix E.

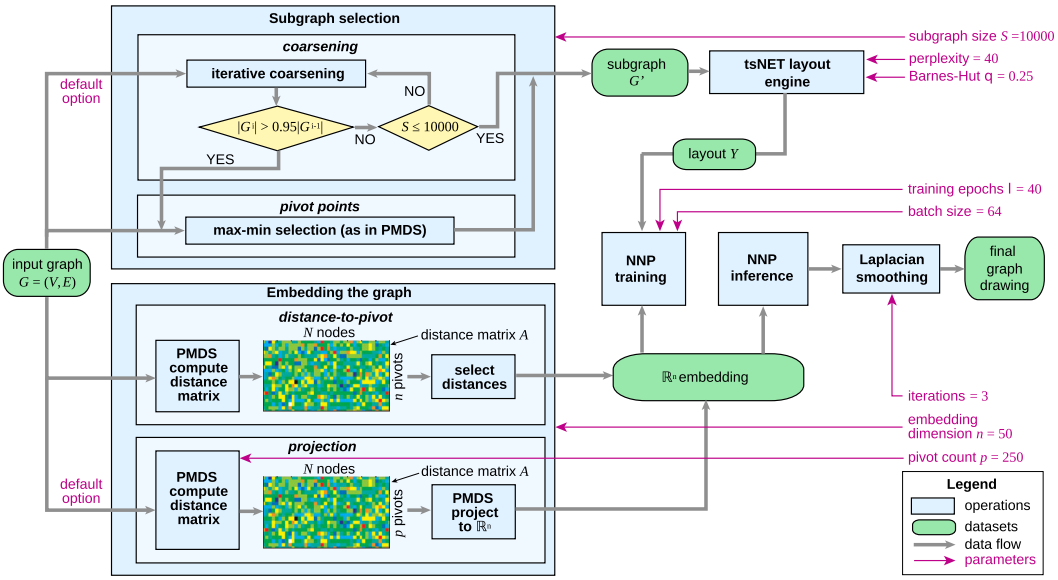


Figure 1 NNP-NET pipeline. For details, see Sec. 3.

3.1 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 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*, uses a smaller matrix A as $\mathbf{E}$, computed the same way as PMDS [4] computes its pivot matrix; and (2) *projection*, project v to $\mathbb{R}^n$ dimensions via PMDS using $p = 250$ pivots. Method (2) was chosen for its better layout quality while sacrificing 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).

3.2 Extracting Suitable Training Data

To train NNP, we need to extract a small but representative subgraph $G' \subset G$. Following various tests (Appendix C), we set the number of nodes in our subgraph of our training set to be equal to $S = 10000$. Then $G' = (V', E')$ and $S = |V'|$. Many multilevel GD methods extract such a subgraph G', *e.g.*, SFDP [27] and FM^3 [22]. 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 [47], *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 $|G^i| > 0.95|G^{i-1}|$. 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 [4]– 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.

3.3 Smoothing

Training NNP on the subgraph G' gives good results in terms of training error (see next Sec. 4). Yet, the produced layouts 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). Weights were used as follows:

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

where V_i are all direct neighbors of v_i. This is done for all nodes in the graph.

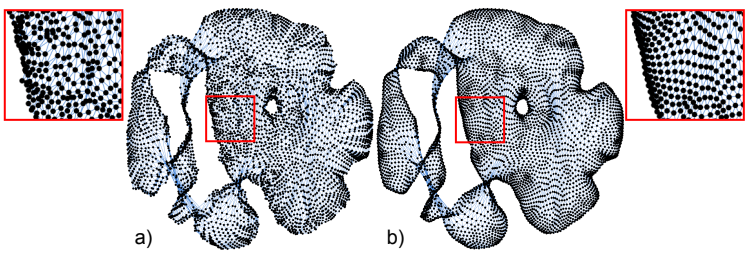


Figure 2 (a) NNP-NET drawing of the 3elt graph. (b) Effect of 2 Laplacian smoothing passes.

3.4 Complexity

Table 1 shows the time complexity of all steps of our method. For creating the $\mathbb{R}^n$ embedding via PMDS (Sec. 3.1), both the dimension count n and pivot count p are constants, yielding linear complexity in the number of nodes N. For creating the subgraph G' (Sec. 3.2), if our coarsening successfully reaches the target size S, the cost reduces to $O(iN)$; the factor $|G'|N$ 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 ts-NET* 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 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.

Table 1 Time complexity of all steps of NNP-NET.

Algorithm step	Time complexity	Parameters
Create the $\mathbb{R}^n$ embedding of G(Sec. 3.1)	$O(npN + p^2N)$	n dimensions, p pivots
Create subgraph G' of size S (Sec. 3.2)	$O(iN + SN)$	i iterations, S nodes
Create ground truth by running ts-NET* 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. 3.3)	$O(N)$	none

4 Results

4.1 Experimental Setup

Implementation. We implement NNP-NET in C++ (code publicly available [26]). We use PMDS from OGDF [8], modified to allow for $n > 3$ output dimensions. We implement tsNET(*) [30] to use both exact t-SNE and the tree-based Barnes-Hut t-SNE approximation [41] 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 [1] using the CPU as the GPU proved slower in our tests due to the small size of this neural network.

Datasets and techniques. Table 2 (left) shows the graphs used in our testing, all coming from the SparseSuite collection [9]. Most graphs used are on the larger side since a key goal we have is to speed up tsNET (Sec. 1). We compare NNP-NET with SFDP [27], FM^3 [22], PMDS [4], DRGraph [47], all state-of-the-art GD methods that aim to handle large graphs; and with tsNET [30] 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 [13]. For DRGraph [47], we use the authors' implementation with their suggested parameter values. Tab. 2 (right) 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 and 16 GB of RAM.

Table 2 Left: Graph datasets used in the evaluation. All graphs come from SparseSuite [9]. Right: Parameters used for NNP-NET.

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	

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

4.2 Quality Metrics

We assess the quality of graph drawings using neighborhood preservation, stress, and Shepard diagrams, in line with [30, 47]. Metrics like edge length deviation [29], crossing number [37], and node-edge occlusion [12], 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 $\mathbf{Y}$ [36]. 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 [30]. 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

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

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

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

with lower values indicating better distance preservation. The factor a scales the drawing in order to minimize stress for a fair comparison. Following [23], we compute it using

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

Table 3 shows neighborhood preservation ν for all our tested drawings. 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. Table 4 shows stress σ for all generated drawings. NNP-NET yields very similar values to tsNET* (both versions). Similarly to ν, we do not see a clear winner between NNP-NET and DRGraph. 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.

Table 3 Neighborhood preservation ν per graph and layout method (higher is better).

	Neighborhood preservation per graph (higher is better)						
Graph	SFDP	FM^3	PMDS	DRGraph	tsNET* exact	tsNET* approx	NNP-NET
dwt_1005	0.5	0.53	0.47	0.5	0.62	0.59	0.53
sierpinski3d	0.51	0.51	0.2	0.52	0.55	0.53	0.52
MISKnowledgeMap	0.17	0.16	0.13	0.33	0.4	0.41	0.24
3elt	0.62	0.65	0.36	0.63	0.66	0.63	0.63
optdigits_10NN	0.52	0.5	0.35	0.62	0.61	0.63	0.61
fe_4elt2	0.47	0.52	0.25	0.56	0.6	0.59	0.59
bcsstk36	0.45	0.41	0.3	0.49	-	0.51	0.44
k49_norm_10NN	0.048	0.045	0.034	0.12	-	0.18	0.093
fe_bcsstk32	0.32	0.38	0.21	0.41	-	0.24	0.37
m_t1	0.3	0.34	0.21	0.37	-	0.35	0.32
ship_003	0.21	0.21	0.17	0.24	-	0.2	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 [5, 17, 47], with graph distance on the x-axis. Figure 3 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* (ts-NET* approx). Overall, we see that SFDP, FM^3, and PMDS preserve distances better than the other methods, which is expected given their design. The remaining methods – which all are based on ts-NET, so favor neighborhood preservation, have quite similar patterns of distance preservation on the smaller graphs (Fig. 3 left column). 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 4 Stress σ per graph and method. Lower is better.

	Stress per graph (lower is better)						
Graph	SFDP	FM^3	PMDS	DRGraph	tsNET* *exact*	tsNET* *approx*	NNP-NET
dwt_1005	0.029	0.026	0.029	0.03	0.038	0.036	0.029
sierpinski3d	0.079	0.078	0.104	0.08	0.077	0.086	0.125
MISKnowledgeMap	0.149	0.171	0.159	0.191	0.188	0.186	0.199
3elt	0.057	0.062	0.056	0.059	0.08	0.079	0.073
optdigits_10NN	0.159	0.144	0.125	0.172	0.142	0.14	0.139
fe_4elt2	0.05	0.069	0.064	0.068	0.095	0.098	0.097
bcsstk36	0.069	0.071	0.068	0.074	-	0.11	0.088
k49_norm_10NN	0.162	0.153	0.155	0.173	-	0.168	0.165
fe_bcsstk32	0.138	0.093	0.098	0.175	-	0.187	0.179
m_t1	0.095	0.079	0.074	0.103	-	0.078	0.147
ship_003	0.069	0.086	0.036	0.042	-	0.059	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

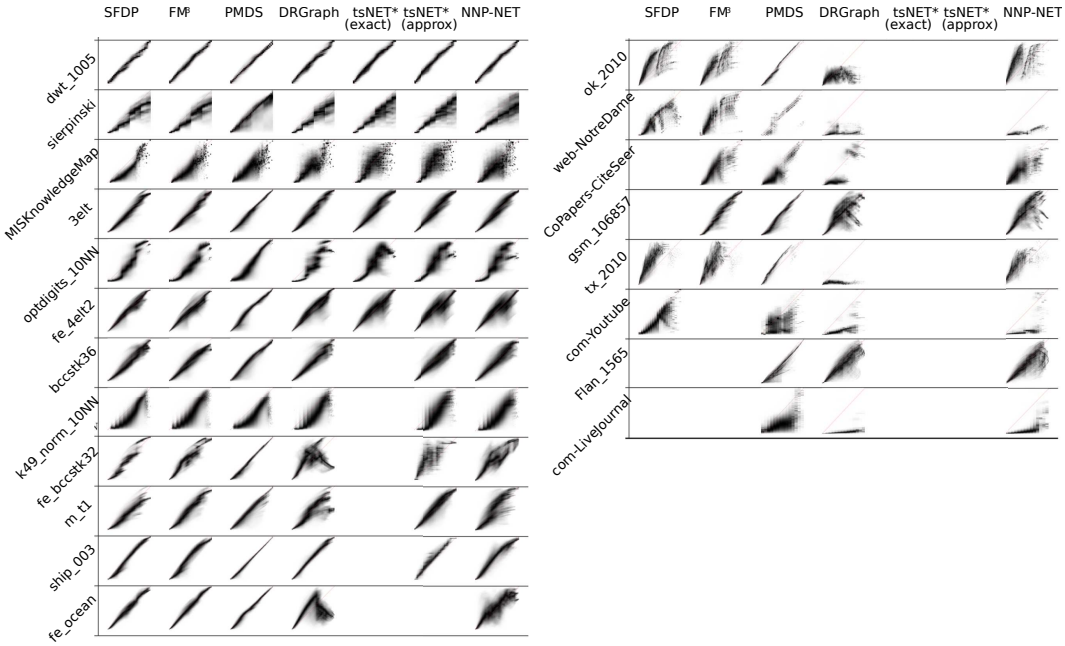


Figure 3 Shepard distance-preservation diagrams for all tested methods on all graphs.

4.3 Visual Comparison

Table 5 shows the drawings created by all tested methods on all graphs, with edges drawn half-transparent to limit visual clutter. As in Tables 3 and 4, 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 ts-NET* 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, 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.

4.4 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. 4). 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 6 shows that both stress and neighborhood preservation improved when including weights.

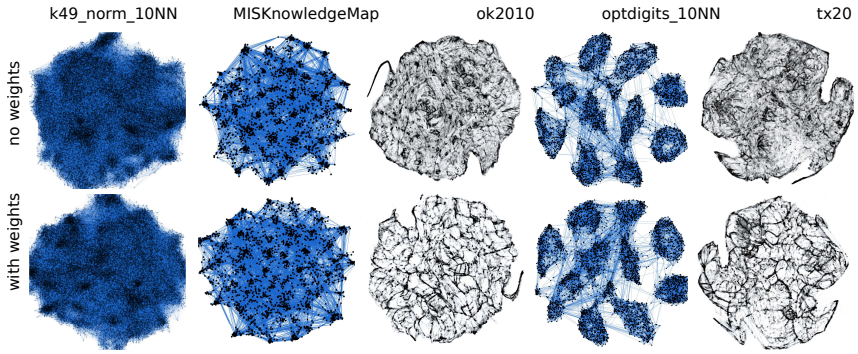


Figure 4 Comparison of NNP-NET drawings, with and without considering weights (Sec. 4.4).

4.5 Execution Time

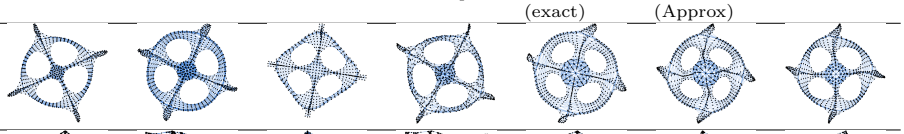
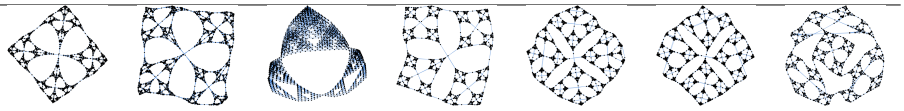
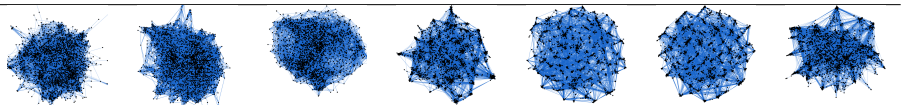
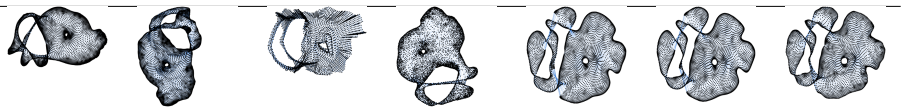
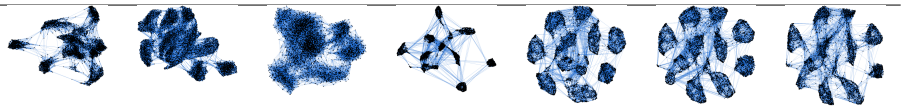
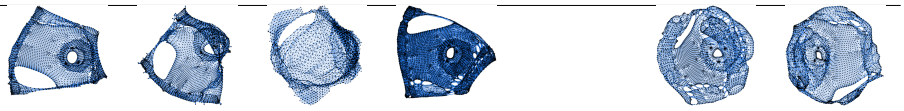
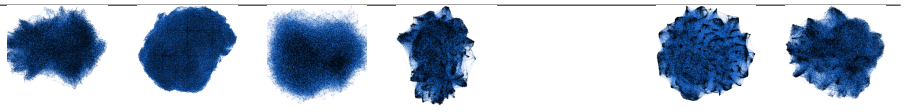
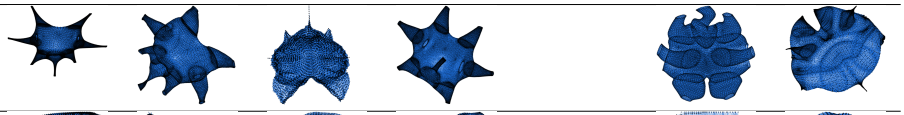
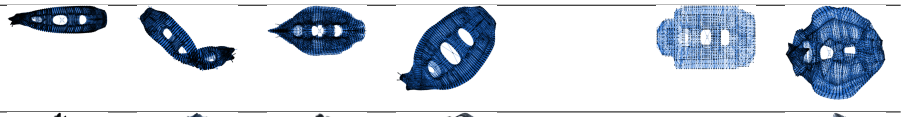
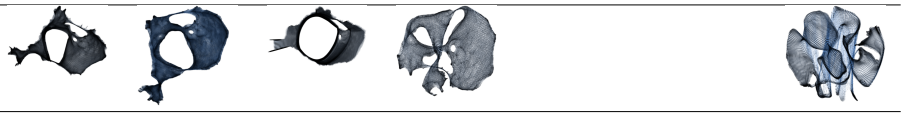
Table 7 shows the execution times for all methods tested with missing values as already explained. NNP-NET takes significantly longer on smaller graphs due to the dominating high constant cost of computing the ground truth. For larger graphs, NNP-NET is slower than PMDS, but yields better quality (see Tabs. 4 and 3). Compared to DRGraph, we see a pattern reversal, with NNP-NET becoming faster for *Flan_1565* and *com-LiveJournal*. To better examine scalability, Fig. 5 shows the time each step of NNP-NET takes *vs* the node count N (both axes use a logarithmic 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 instead of the much faster Breadth-First Search (BFS) 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. 3.2).

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

Table 5 Drawings created by various methods on various graphs (continued on next page).

Graph	SFDP	FM^3	PMDS	DRGraph	tsNET* (exact)	tsNET* (Approx)	NNP-NET
dwt_1005							
sierpinski3d							
MIS-Knowledge-Map							
3elt							
optdigits_10NN							
fe_4elt2							
bcsstk36							
k49_norm_10NN							
fe_bcsstk32							
m_t1							
ship_003							
fe_ocean							

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.

Table 5 Drawings created by various methods on various graphs (continued from previous page).

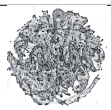
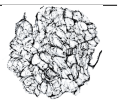
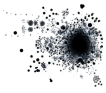
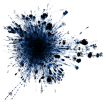
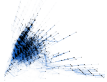
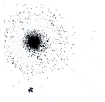
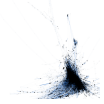
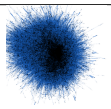
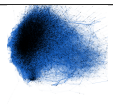
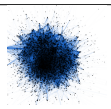
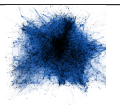
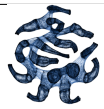
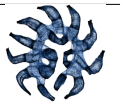
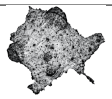
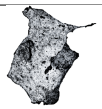
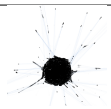
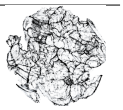
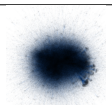
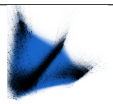
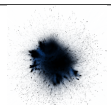
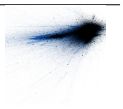
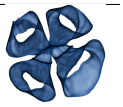
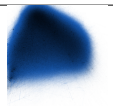
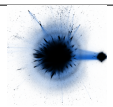
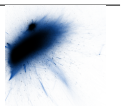
Graph	SFDP	FM^3	PMDS	DRGraph	tsNET* (exact)	tsNET* (Approx)	NNP-NET
ok2010							
web-NotreDame							
CoPapers-Citeseer							
gsm_106857							
tx2010							
com-Youtube							
Flan_1565							
com-LiveJournal							

Table 6 Quality metrics for NNP-NET drawing when using *vs* not using edge weights (Sec. 4.4).

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

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. This is to simulate a situation where the fallback is not needed.

Figure 6 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

Table 7 Execution time in seconds for all tested methods, all graphs (Sec. 4.5).

	Execution time per method						
Graph	SFDP	FM^3	PMDS	DRGraph	tsNET* *exact*	tsNET* *approx*	NNP-NET
dwt_1005	0.34	0.12	**0.02**	0.06	1.44	2.61	10.13
sierpinski3d	0.75	0.12	1.37	**0.11**	6.52	5.42	11.38
MISKnowledgeMap	1.23	0.76	1.20	**0.13**	40.42	13.71	24.27
3elt	1.71	0.20	**0.07**	0.27	110.70	18.92	21.14
optdigits_10NN	2.59	0.58	2.38	**0.30**	155.46	59.68	36.88
fe_4elt2	6.06	0.22	**0.15**	0.59	931.18	107.97	58.96
bcsstk36	10.89	2.28	**0.53**	1.49	-	288.91	56.96
k49_norm_10NN	37.28	**2.14**	37.13	2.66	-	1718.79	156.79
fe_bcsstk32	28.25	3.07	**2.47**	3.64	-	572.96	86.68
m_t1	77.61	14.22	**4.04**	8.21	-	1133.24	87.81
ship_003	94.44	14.24	**5.85**	9.61	-	1456.06	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

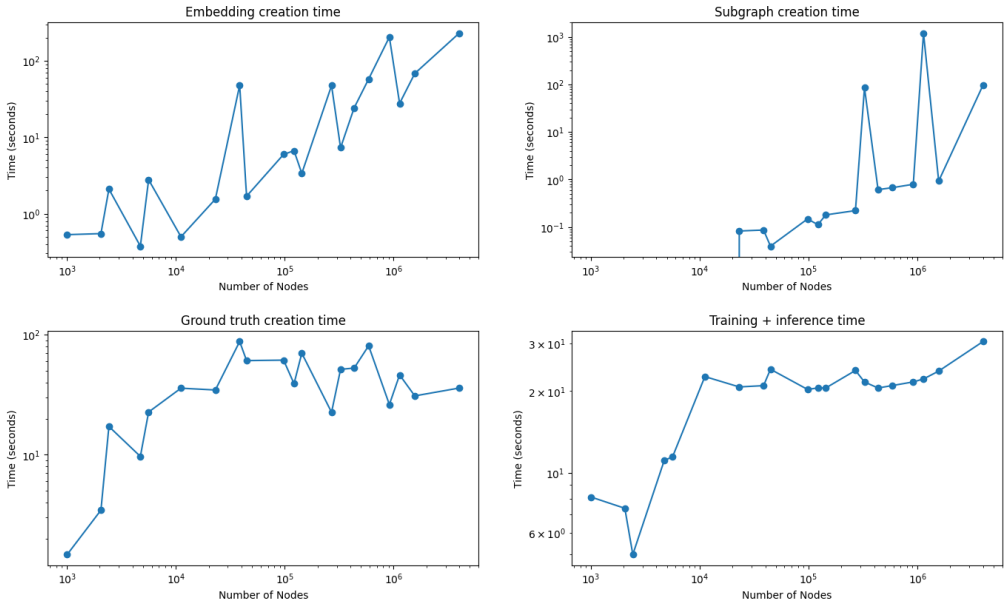


Figure 5 Execution time of each individual component of NNP-NET (Sec. 4.5).

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 effects due to (1) and (2). Actual timings show the same trend, albeit with a bit more noise (orange curve). As Tab. 7 also showed, we see how NNP-NET overtakes DRGraph at about $N = 10^6$ nodes. Given NNP-NET's and DRGraph's appearent trend, NNP-NET will likely stay faster compared to DRGraph for larger graphs.

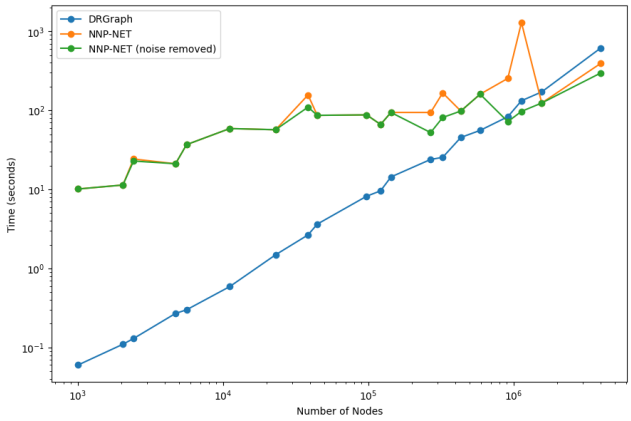


Figure 6 Execution time comparison between DRGraph, NNP-NET, and a theoretical execution time of NNP-NET with reduced noise as explained in sec. 4.5.

5 Discussion

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

Scalability: NNP-NET has 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.

Layout quality: The 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 when comparing with competing methods.

Robustness: NNP-NET could handle all the tested graphs with good results. No component in NNP-NET's pipeline uses complex (parameter-dependent) heuristics. 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 parameters that could be adjusted, all of these were set to the same fixed values for all tests (Tab. 2 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. 4.4).

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.

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 topic we will look into in future work.

6 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 higher running times, and comparable quality, on graphs over 1 million nodes, than its closest competitor, DRGraph.

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 important aspect to look into further, as can boost speed significantly for the graphs that our current fast coarsening heuristic cannot handle.

References

1 Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org. doi:10.48550/arXiv.1603.04467.

2 Reyan Ahmed, Felice De Luca, Sabin Devkota, Stephen Kobourov, and Mingwei Li. Graph drawing via gradient descent, $(GD)^2$. In *Graph Drawing*, pages 3–17. Springer, 2020. doi:10.1007/978-3-030-68766-3_1.

3 Reyan Ahmed, Felice De Luca, Sabin Devkota, Stephen Kobourov, and Mingwei Li. Multicriteria scalable graph drawing via stochastic gradient descent, $(SGD)^2$. *IEEE TVCG*, 28(6):2388–2399, 2022. doi:10.1109/TVCG.2022.3155564.

4 Ulrik Brandes and Christian Pich. Eigensolver methods for progressive multidimensional scaling of large data. In *Graph Drawing*, pages 42–53. Springer, 2007. doi:10.1007/978-3-540-70904-6_6.

5 Ulrik Brandes and Christian Pich. An experimental study on distance-based graph drawing. In *Graph Drawing*, pages 218–229. Springer, 2009. doi:10.1007/978-3-642-00219-9_21.

6 Carlo Bredius, Zonglin Tian, and Alexandru C. Telea. Visual Exploration of Neural Network Projection Stability. In *MLVis*, 2022. doi:10.2312/mlvis.20221068.

7 David M. Chan, Roshan Rao, Forrest Huang, and John F. Canny. t-SNE-CUDA: GPU-accelerated t-SNE and its applications to modern data. In *Computer Architecture and High Performance Computing*, 2018. doi:10.1109/CAHPC.2018.8645912.

8 Markus Chimani, Carsten Gutwenger, Michael Jünger, Gunnar W. Klau, Karsten Klein, and Petra Mutzel. The Open Graph Drawing Framework (OGDF). In R. Tamassia, editor, *Handbook of Graph Drawing and Visualization*, pages 543–569. CRC Press, 2014.

9 Timothy A. Davis and Yifan Hu. The University of Florida sparse matrix collection. *ACM Transactions on Mathematical Software*, 38(1):1–25, 2011. doi:10.1145/2049662.2049663.

10 Sabin Devkota, Reyan Ahmed, Felice De Luca, Katherine E. Isaacs, and Stephen Kobourov. Stress-plus-x (spx) graph layout. In *Graph Drawing*, pages 291–304. Springer, 2019. doi:10.1007/978-3-030-35802-0_23.

11 Peter Eades. A heuristic for graph drawing. *Congressus numerantium*, 42:146–160, 1984.

12 Peter Eades, Michael E. Houle, and Richard Webber. Finding the best viewpoints for three-dimensional graph drawings. In *Graph Drawing*. Springer, 1997. doi:10.1007/3-540-63938-1_53.

13 John Ellson, Emden R. Gansner, Lefteris Koutsofios, Stephen C. North, and Gordon Woodhull. Graphviz— open source graph drawing tools. In *Graph Drawing*, pages 483–484. Springer, 2002. doi:10.1007/3-540-45848-4_57.

14 Mateus Espadoto, Nina Hirata, Tomita Sumiko, and Alexandru C. Telea. Deep learning multidimensional projections. *Information Visualization*, 19(3):247–269, 2020. doi:10.1177/1473871620909485.

15 Mateus Espadoto, Rafael M. Martins, Andreas Kerren, Nina S. T. Hirata, and Alexandru C. Telea. Toward a quantitative survey of dimension reduction techniques. *IEEE TVCG*, 27(3):2153–2173, 2019. doi:10.1109/TVCG.2019.2944182.

16 Thomas M. J. Fruchterman and Edward M. Reingold. Graph drawing by force-directed placement. *Software – Practice and Experience*, 21(11):1129–1164, 1991. doi:10.1002/spe.4380211102.

17 Emden R. Gansner, Yifan Hu, and Stephen C. North. A maxent-stress model for graph layout. *IEEE TVCG*, 19(6):927–940, 2013. doi:10.1109/TVCG.2012.299.

18 Emden R. Gansner, Yehuda Koren, and Stephen C. North. Graph drawing by stress majorization. In *Graph Drawing*, pages 239–250. Springer, 2005. doi:10.1007/978-3-540-31843-9_25.

19 Helen Gibson, Joe Faith, and Paul Vickers. A survey of two-dimensional graph layout techniques for information visualisation. *Information Visualization*, 12(3-4):324–357, 2013. doi:10.1177/1473871612455749.

20 Loann Giovannangeli, Frederic Lalanne, David Auber, Romain Giot, and Romain Bourqui. Deep neural network for drawing networks, $(DNN)^2$. In *Graph Drawing*, pages 375–390. Springer, 2021. doi:10.1007/978-3-030-92931-2_27.

21 Loann Giovannangeli, Frederic Lalanne, David Auber, Romain Giot, and Romain Bourqui. Toward efficient deep learning for graph drawing (*dl4gd*). *IEEE TVCG*, 30(2):1516–1532, 2024. doi:10.1109/TVCG.2022.3222186.

22 Martin Gronemann. Engineering the fast-multipole-multilevel method for multicore and SIMD architectures. *Master's thesis. Technische Univ. Dortmund*, 2009.

23 Florian Grötschla, Joël Mathys, Robert Veres, and Roger Wattenhofer. CoRe-GD: A Hierarchical Framework for Scalable Graph Visualization with GNNs. In *ICLR, Vienna, Austria*, May 2024. URL: https://openreview.net/forum?id=vtyasLn4RM.

24 Kenneth M. Hall. An r-dimensional quadratic placement algorithm. *Management science*, 17(3):219–229, 1970. doi:10.1287/mnsc.17.3.219.

25 David Harel and Yehuda Koren. Graph drawing by high-dimensional embedding. *JGAA*, 8(2):195–214, 2004. doi:10.7155/jgaa.00089.

26 Ilan Hartskeerl, Tamara Mchedlidze, Simon van Wageningen, Peter Vangorp, and Alexandru Telea. NNP-NET. Software, swhId: swh:1:dir:539da3e3adda73abc8bb1af0c2a5c59fb7abef49 (visited on 2025-11-07). URL: https://github.com/IlanHartskeerl/NNP-NET, doi:10.4230/artifacts.25058.

27 Yifan Hu. Efficient, high-quality force-directed graph drawing. *Mathematica journal*, 10(1):37–71, 2005.

28 Mathieu Jacomy, Tommaso Venturini, Sebastien Heymann, and Mathieu Bastian. ForceAtlas2, a Continuous Graph Layout Algorithm for Handy Network Visualization Designed for the Gephi Software. *PLOS ONE*, 9(6):e98679, 2014. doi:10.1371/journal.pone.0098679.

29 Tomihisa Kamada and Satoru Kawai. An algorithm for drawing general undirected graphs. *Inf Process Lett*, 31(1):7–15, 1989. doi:10.1016/0020-0190(89)90102-6.

30 Johannes F. Kruiger, Paulo E. Rauber, Rafael M. Martins, Andreas Kerren, Stephen Kobourov, and Alexandru C. Telea. Graph layouts by t-SNE. *Computer Graphics Forum*, 36(3):283–294, 2017. doi:10.1111/cgf.13187.

31 Joseph B. Kruskal and Judith B. Seery. Designing network diagrams. In *General Conference on Social Graphics*, pages 22–50, 1980.

32 Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S. Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems*, volume 26, 2013. URL: https://proceedings.neurips.cc/paper/2013/hash/9aa42b31882ec039965f3c4923ce901b-Abstract.html.

33 Stephen C. North. *Drawing graphs with NEATO*, 2004. NEATO User's Manual.

34 Fernando V. Paulovich, Alessio Arleo, and Stef van den Elzen. When dimensionality reduction meets graph (drawing) theory: Introducing a common framework, challenges and opportunities. *Computer Graphics Forum*, 44(3):e70105, 2025. doi:10.1111/cgf.70105.

35 Nicola Pezzotti, Julian Thijssen, Alexander Mordvintsev, Thomas Hollt, Baldur van Lew, Boudewijn Lelieveldt, Elmar Eisemann, and Anna Vilanova. GPGPU linear complexity t-SNE optimization. *IEEE TVCG*, 26(1):1172–1181, 2020. doi:10.1109/TVCG.2019.2934307.

36 Helen C. Purchase. Metrics for graph drawing aesthetics. *Journal of Visual Languages & Computing*, 13(5):501–516, 2002. doi:10.1006/jvlc.2002.0232.

37 Helen C. Purchase, Robert F. Cohen, and Murray James. Validating graph drawing aesthetics. In *Graph Drawing*, pages 435–446. Springer, 1996. doi:10.1007/BFb0021827.

38 Roberto Tamassia. *Handbook of graph drawing and visualization*. CRC press, 2013.

39 Matteo Tiezzi, Gabriele Ciravegna, and Marco Gori. Graph neural networks for graph drawing. *IEEE Transactions on Neural Networks and Learning Systems*, 35(4):4668–4681, 2022. doi:10.1109/TNNLS.2022.3184967.

40 Warren S. Torgerson. Multidimensional scaling: I. Theory and method. *Psychometrika*, 17(4):401–419, 1952. doi:10.1007/BF02288916.

41 Laurens van der Maaten. Accelerating t-SNE using tree-based algorithms. *The Journal of Machine Learning Research*, 15(1):3221–3245, 2014. doi:10.5555/2627435.2697068.

42 Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9(11), 2008.

43 Xiaoqi Wang, Kevin Yen, Yifan Hu, and Han-Wei Shen. DeepGD: A deep learning framework for graph drawing using GNN. *IEEE CGA*, 41(5):32–44, 2021. doi:10.1109/MCG.2021.3093908.

44 Xiaoqi Wang, Kevin Yen, Yifan Hu, and Han-Wei Shen. SmartGD: A gan-based graph drawing framework for diverse aesthetic goals. *IEEE TVCG*, 30(8):5666–5678, 2023. doi:10.1109/TVCG.2023.3306356.

45 Yong Wang, Zhihua Jin, Qianwen Wang, Weiwei Cui, Tengfei Ma, and Huamin Qu. DeepDrawing: A deep learning approach to graph drawing. *IEEE TVCG*, 26(1):676–686, 2020. doi:10.1109/TVCG.2019.2934798.

46 Jonathan X. Zheng, Samraat Pawar, and Dan F. M. Goodman. Graph drawing by stochastic gradient descent. *IEEE TVCG*, 25(9):2738–2748, 2019. doi:10.1109/TVCG.2018.2859997.

47 Minfeng Zhu, Wei Chen, Yuanzhe Hu, Yuxuan Hou, Liangjun Liu, and Kaiyuan Zhang. DRGraph: An efficient graph layout algorithm for large-scale graphs by dimensionality reduction. *IEEE TVCG*, 27(2):1666–1676, 2020. doi:10.1109/TVCG.2020.3030447.

A Comparison of Embedding Methods

We further compare the two embedding methods described in Sec. 3.1, namely (1) using the *distances-to-pivot* method and (2) the *projecting* method. Table 8 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 9 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 8 Layouts created using the *distance-to-pivot* and *projection* embedding methods.

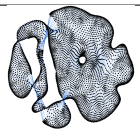
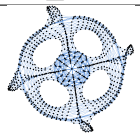
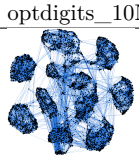
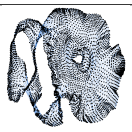
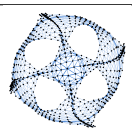
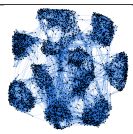
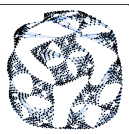
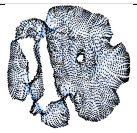
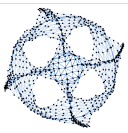
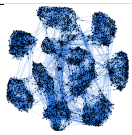
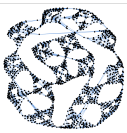
Graph	3elt	dwt_1005	optdigits_10NN	sierpinski3d
Ground truth				
Distance-to-pivot				
Projection				

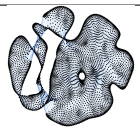
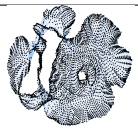
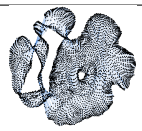
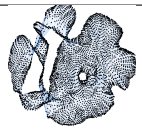
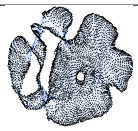
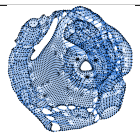
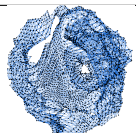
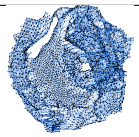
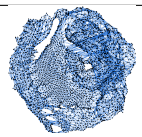
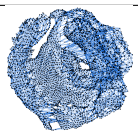
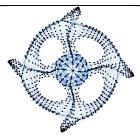
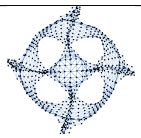
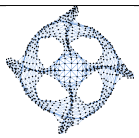
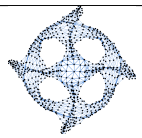
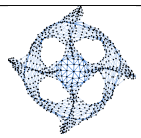
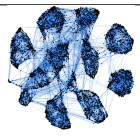
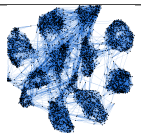
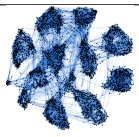
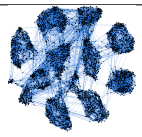
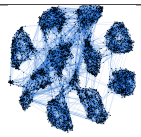
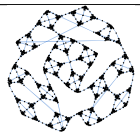
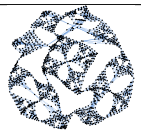
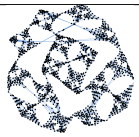
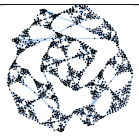
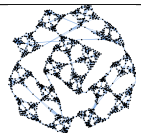
Table 9 Quality metrics obtained using the *distance-to-pivot* and *projection* embedding methods.

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**

B Number of Embedding Dimensions n

We tested different values $n \in \{10, 25, 50, 100\}$ for the number of dimensions n to embed in (Sec. 3.1). Table 10 compares the obtained results and shows that these appear visually very similar for different values of n. Table 11 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. In the same time, execution time increases quite strongly with n. Our setting $n = 50$ balances well quality and time.

Table 10 Results for different embedding sizes n.

Graph	Ground Truth	$n = 10$	$n = 25$	$n = 50$	$n = 100$
3elt					
bcsstk36					
dwt_1005					
optdigits_ 10NN					
sierpinski3d					

C Setting the Extracted Subgraph Size S

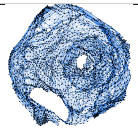
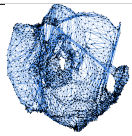
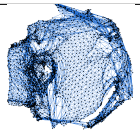
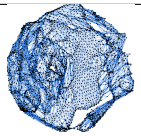
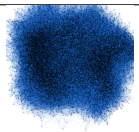
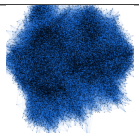
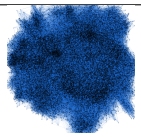
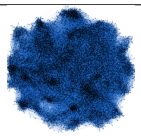
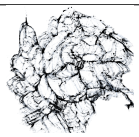
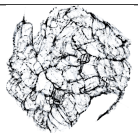
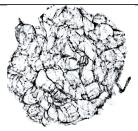
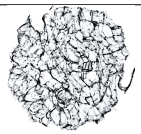
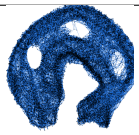
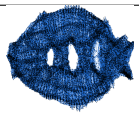
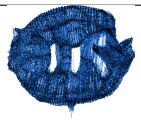
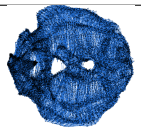
We have compared results obtained for different values $S \in \{1000, 2500, 5000, 10000\}$ for the size S of the coarsened subgraph $\mathbf{G}'$ (Sec. 3.2). Table 12 shows these results. We see that S can significantly affect the obtained layouts: for *k48_norm_10NN*, the results for $S = 10000$ show 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 13 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 11 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**

Table 12 Results using different sizes S for the subgraph $\mathbf{G}'$.

Graph	$\lvert\mathbf{G}'\rvert = 1000$	$\lvert\mathbf{G}'\rvert = 2500$	$\lvert\mathbf{G}'\rvert = 5000$	$\lvert\mathbf{G}'\rvert = 10000$
bcsstk36				
k49_norm_10NN				
ok2010				
ship_003				

D Comparison of Subgraph Extraction Methods

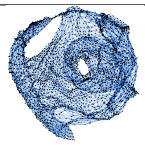
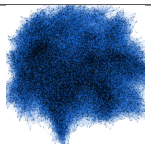
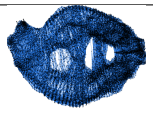
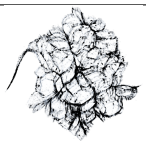
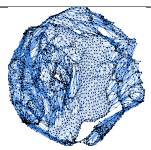
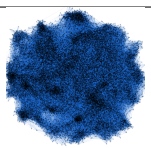
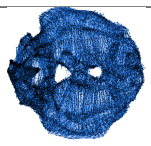
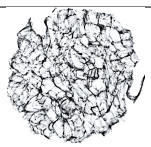
We next compare the *pivot-points* and *coarsening* methods for computing the subgraph $\mathbf{G}'$ from the input graph $\mathbf{G}$ (Sec. 3.2). Both methods receive the parameter S that determines the size of the subgraph $\mathbf{G}'$ to be extracted. Comparing results for the *same* value of S is, however, not directly useful. Indeed, the pivot-points method becomes too slow to be useful in practice for values $S > 1000$. In contrast, the coarsening method scales computationally well to higher values. As such, we chose to compare 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 13 Performance metrics for the different sizes S for the 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

Table 14 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 produce for $S = 1000$ (see Fig. 12). Table 15 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 14 Results obtained by using *pivot-points* and *coarsening* to extract the subgraph $\mathbf{G}'$.

Graph	bcsstk36	k49_norm_10NN	ship_003	ok2010
Pivot-points, $S = 1000$				
Coarsening, $S = 10000$				

E Pseudo Code

The following pseudo code is for the entire 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 15 Performance metrics using *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**