

Utrecht University

Master Thesis

Enhancing NNP-NET by Refined Graph Coarsening Strategies

Author:

Jeroen VAN HOUTEN

Supervisors:

Alexandru C. TELEA

Ilan HARTSKEERL

Tamara MTSENTLINTZE

*A thesis submitted in fulfilment of the requirements
for the degree of Master of Science in Game and Media Technology
in the*

Department of Information and Computing Sciences

September 13, 2026

Abstract

Generating high-quality graph layouts for very large graphs poses a trade-off between scalability and accuracy. Recently, NNP-NET introduced a neural-network based architecture to accelerate the high quality tsNET algorithm, achieving a linear time complexity for very large graphs. However, NNP-NET relies on a clustering-based subgraph extraction method that experiences high performance degradation on complex graphs; specifically ones with a high degree heterogeneity (empirically observed as $\kappa \geq 50$) and negative degree assortativity. This degradation behaviour triggers a pivot-points based fallback method that introduces a severe runtime bottleneck, often consuming over 60% of the total pipeline execution time.

To address this vulnerability, this thesis proposes and evaluates alternative subgraph extraction techniques, introducing adaptive solar system collapsing (ASSC) alongside an optimized pivot-points algorithm and an edge-contraction based algorithm. When evaluated on a diverse dataset of 25 real-world graphs, ASSC shows exceptional robustness and efficiency. By hybridizing solar system collapsing with a fine-grained clustering fallback and stall-handling, ASSC completely eliminates the subgraph extraction bottleneck present in NNP-NET.

Empirical results show that on stalling graphs (i.e., graphs that would trigger NNP-NET’s fallback method), ASSC achieves an average total pipeline speed-up of 9.2x compared to the baseline NNP-NET, while matching its high efficiency on other graphs. Moreover, ASSC maintains the high quality of the generated layouts, preserving local neighbourhoods and, in the case of stalling graphs, reducing global stress. Ultimately, the integration of ASSC in NNP-NET’s pipeline solidifies NNP-NET as a highly scalable, high-quality graph drawing pipeline.

CONTENTS

I	Introduction	3
II	Background and Related Work	5
II-A	Quality Metrics	5
II-A1	Stress	5
II-A2	Neighbourhood Preservation	5
II-B	Graph Coarsening and Graph Sparsification	6
II-C	NNP-NET's Clustering Scheme & Pivot-Points Fallback	6
II-C1	Clustering Scheme	6
II-C2	Clustering Scheme Stalling	7
II-C3	Pivot-Points Fallback	8
II-D	Edge Contraction	10
II-E	Solar System Collapsing (SSC)	11
II-F	Additional Methods	12
III	Method	14
III-A	Adaptive Solar System Collapsing (ASSC)	14
III-A1	Refilling Strategy	14
III-A2	Clustering Fallback	14
III-B	Pivot-Points Optimization	16
III-C	Setting Up an Experimental Performance Measurement	17
III-D	Experimental Setup	18
IV	Results	20
IV-A	Execution Time	20
IV-A1	Subgraph Extraction on Standard Topologies	20
IV-A2	Subgraph Extraction on Stalling Topologies	21
IV-A3	End-to-End Pipeline Performance	21
IV-B	Layout Quality	24
IV-B1	Quantitative Metrics	24
IV-B2	Qualitative Visual Analysis	26
V	Discussion	35
V-A	Handling Stalling Behaviour in ASSC	35
V-A1	Over-Coarsened Subgraphs versus Continued Clustering	35
V-A2	Explaining Continued Clustering's Instability	35
V-A3	Preserving Theoretical Time Complexity	36
V-B	Limitations	36
V-B1	Variance	36
V-B2	Hardware Constraints	36
V-B3	Memory Overhead	36
V-C	Future Work	37
V-C1	Understanding Clustering's Stalling Behaviour	37
V-C2	Optimizing NNP-NET's Other Components	37
VI	Conclusion	38
	References	39
	Appendix	40
A	Weighted Graph Example	40
B	Unweighted Graph Example	41

I. INTRODUCTION

Generating good layouts for graphs is a fundamental challenge in the field of graph drawing. In practice, graph data is typically stored as abstract tables of nodes and edges; translating this data into an accurate and visually pleasing 2D layout requires complex algorithms. The quality of these layouts is commonly evaluated using two primary metrics: stress, which measures how accurately the geometric distances in the drawing reflect the true graph-theoretic distances, and neighbourhood preservation, which evaluates whether a node’s graph theoretic neighbours are also the nodes that lie closest to it in the drawing. As modern graphs can contain millions of nodes, computing these layouts at a high quality becomes a massive computational challenge.

Existing graph drawing methods therefore traditionally need to compromise between scalability and layout quality. Methods designed to optimize for stress (such as SFDP by Hu [1] and FM³ by Gronemann [2]) or neighbourhood preservation (such as tsNET* by Krüger et al. [3]) produce excellent results, but their runtime and memory requirements make them unfit for very large graphs. In comparison, highly efficient methods like DRGraph by Zhu et al. [4] and PMDS by Brandes & Pich [5] achieve massive speed-ups, but at a noticeable cost to quality.

To bridge this gap, Hartskeerl et al. [6] introduced NNP-NET, a novel graph drawing method based on tsNET by Krüger et al. [3]. While tsNET utilizes t-distributed Stochastic Neighbour Embedding (t-SNE) by Van der Maaten & Hinton [7] to generate high-quality layouts, NNP-NET accelerates this process to avoid high runtimes. It does this by:

- 1) embedding the graph in 50-dimensional space using Pivot MultiDimensional Scaling (PMDS), as the subsequent neural network (step 3) requires high-dimensional input data rather than standard node and edge tables;
- 2) extracting a small but representative subgraph of the entire graph;
- 3) computing a high-quality ground truth layout for this subgraph using tsNET* (a deterministic refinement of tsNET that initializes node positions using PMDS, as opposed to a random initialization) and training a neural network to predict these 2D positions from the 50D embeddings;
- 4) applying the trained network to predict positions for the full graph, followed by three Laplacian smoothing iterations to remove noise.

While NNP-NET produces graph drawings in an efficient manner in most cases, its primary subgraph extraction method (step 2) fails to perform efficiently in all cases. Specifically, its linear-time clustering method stalls on graphs that contain few ‘hub’ nodes that have many neighbours, and many ‘outskirt’ nodes that have very few neighbours. This will be expanded upon in Chapter II-C. In any case, this stalling behaviour forces the NNP-NET pipeline to fall back on a pivot-points based method. In isolation, this fallback method is on average *slower by three orders of magnitude* than the fast clustering method. As illustrated in Figure 1, this creates a severe bottleneck: a subgraph extraction step that commonly requires less than 10% of the total runtime of NNP-NET suddenly generally takes up more than 80%.

NNP-NET Runtime Breakdown for 'k49_norm_10NN'

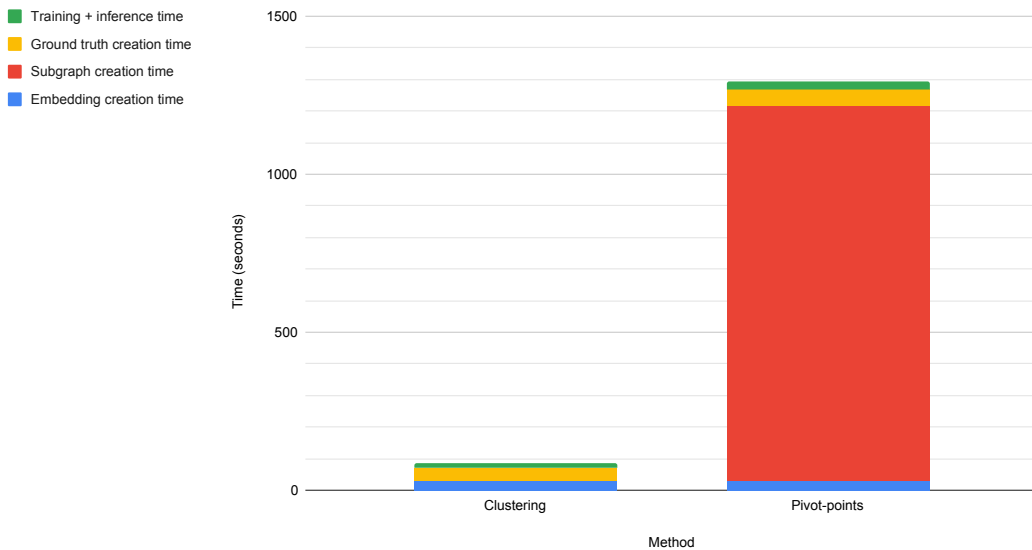


Fig. 1. Runtime breakdown for the graph *k49_norm_10NN*. The stacked segments correspond directly to the four execution steps outlined above.

In fact, the original authors of NNP-NET explicitly identified the need for a more robust subgraph extraction method as the most important direction for future work to address these outlier cases. So to answer this call and address the bottleneck, this thesis addresses the following research question:

Research Question

Which alternative subgraph extraction techniques can replace NNP-NET’s ‘quick’ clustering method by being robust enough to eliminate the need for a fallback method while rivalling or surpassing it in both speed and output quality?

To be considered a viable replacement, candidate methods also need to satisfy the following criteria:

- **Ease of implementation:** The method must have code available online, preferably in C++ (the language NNP-NET is written in) or Python. Getting the implementation working must take no more than a few days.
- **Genericity:** The method has to be able to handle all fully connected undirected graphs. NNP-NET was tested on twenty different graphs from the SparseSuite collection by Davis & Hu [8]. For three of these, the clustering method failed and the slower pivot-points method was used instead. The method must have no issue with any of them.
- **Speed:** The method must resolve the current runtime bottleneck by strictly limiting the time spent on subgraph extraction. We evaluate this performance both practically and theoretically:
 - **Practical performance:** Subgraph extraction using the current ‘quick’ clustering method takes on average less than 9% of NNP-NET’s total runtime, whereas the pivot-points fallback method takes 62% on average. The method must strictly require no more than 10% of the total runtime, ensuring that the total runtime of the NNP-NET pipeline remains consistently low for all graphs.
 - **Theoretical performance:** While ideally the method achieves a linear time complexity of $O(N)$ (with N the number of nodes) or a strictly better time complexity than the pivot-points baseline of $O(NS)$ (with S the number of nodes in the target subgraph), this is not a hard requirement. A method with the same $O(NS)$ complexity remains acceptable provided it possesses a smaller constant factor that guarantees the practical speed-up outlined above.
- **Quality:** To assess quality, we measure the level of *stress* and *neighbourhood preservation* in the final layout produced by NNP-NET. Because the primary objective is accelerating the pipeline, it is anticipated that these quality metrics might worsen when employing faster graph coarsening methods. Therefore, achieving an optimal trade-off is critical: for example, a 2x improvement in speed in exchange for a 10% decrease in neighbourhood preservation represents a worthwhile improvement, whereas a 2x worsening of the stress value for the same speed-up does not. Rather than defining rigid theoretical thresholds beforehand, we will empirically measure these speed-versus-quality trade-offs for each tested method, and the boundaries of acceptable performance will be evaluated in detail in Chapter IV.
- **Ease of use:** The method must be easy to use for an end user. Ideally, there should be no parameters to manually tune; if there are parameters that can be tuned, there should be a ‘default’ value that can be used without significant loss in quality or speed.
- **Weighted graph support:** In case of an edge-weighted graph, the method must extract an edge-weighted subgraph. Preferably, the method takes into account these edge weights when generating the subgraph.

The remainder of this thesis is structured as follows. Chapter II provides the theoretical background on graph coarsening and reviews candidate subgraph extraction schemes from existing literature. Chapter III details the proposed methodology, including the design of the adaptive solar system collapsing algorithm (ASSC), an optimization for the baseline pivot-points method, and the experimental setup. Chapter IV presents the empirical evaluation results, comparing the execution times and layout quality of the candidate methods against the NNP-NET baseline. Chapter V discusses the ‘adaptive’ part of ASSC and the architectural necessity of its stall-handling mechanism. Then it addresses the limitations of the experimental setup and outlines the directions for future research. The thesis closes with concluding remarks in Chapter VI.

II. BACKGROUND AND RELATED WORK

In this chapter, we outline the theoretical concepts and prior work relevant to optimizing NNP-NET’s subgraph extraction. We begin in Section II-A by formally defining the metrics used to evaluate layout quality. Section II-B distinguishes between graph coarsening and graph sparsification, establishing why coarsening is the necessary approach for our purposes. In Section II-C, we detail the baseline subgraph extraction methods and define the exact conditions under which the primary clustering scheme stalls. The remainder of the chapter explores candidate replacement schemes found in the literature. Sections II-D and II-E discuss specific edge contraction and solar system collapsing algorithms, while Section II-F reviews additional methods that were evaluated but ultimately deemed unfit for our criteria. An overview of our candidate methods is provided in Table II, and the standard notation used throughout this thesis is summarized in Table I.

Symbol	Definition
G	The input graph
V	The list of all nodes in G
N	The number of nodes in G
E	The list of all edges in G
n	The target number of nodes for the subgraph of G
Δ	The maximum degree of G
d_{ij}	The graph-theoretic distance between node i and node j .

TABLE I
NOTATION

A. Quality Metrics

Evaluating the quality of a graph layout is inherently subjective, so quantitative metrics are required to objectively compare the impact of different subgraph extraction methods. To maintain consistency with the baseline NNP-NET architecture, we evaluate the generated layouts using two primary metrics: stress and neighbourhood preservation.

1) *Stress*: Stress measures how accurately the distances in the final 2D graph drawing reflect the true graph-theoretic distances of the original graph. To make the metric scale-invariant and ensure the result is bounded between 0 and 1, a normalized formulation with scaling factor a is used. This formulation is equivalent with the one used in Zhu et al. [4] and Grötschla et al. [9]. The stress value σ is calculated using the following formula:

$$\sigma = \min_a \frac{1}{|V|^2} \sum_{i \neq j} \frac{(d_{ij} - a||p_i - p_j||)^2}{d_{ij}^2} \in [0, 1], \quad (1)$$

Here, d_{ij} represents the graph-theoretic distance between node i and node j in the original graph G , and $||p_i - p_j||$ is the distance between their corresponding coordinates in the 2D layout. For brevity, the term ‘distance’ will refer to the graph theoretic distance d_{ij} throughout the rest of this thesis unless otherwise specified. The factor a is the aforementioned scaling factor. It is computed by

$$a = \frac{\sum_{i \neq j} ||p_i - p_j|| / d_{ij}}{\sum_{i \neq j} ||p_i - p_j||^2 / d_{ij}^2}. \quad (2)$$

A lower value for σ indicates a more accurate global layout, with 0 being the theoretical minimum and 1 being the theoretical maximum.

2) *Neighbourhood Preservation*: As the name suggests, neighbourhood preservation evaluates how well neighbourhoods are preserved in a graph drawing. The metric, as used by Krueger et al. [3] and based on Gansner et al. [10], first defines the local neighbourhood $N_G(\mathbf{x}_i, r_G)$ of a node i as:

$$N_G(\mathbf{x}_i, r_G) = \{\mathbf{x}_j \in V \mid d_{ij} \leq r_G\}, \quad (3)$$

where r_G is the maximum distance threshold for two nodes to be considered neighbours, and d_{ij} is their graph-theoretic distance. In practice, r_G is defined as a 2-hop distance for unweighted graphs, and for weighted graphs, it is dynamically scaled to twice the average edge weight of the graph to account for varying distance magnitudes. Using this definition for both the original graph space ($\mathbf{x}$) and the 2D layout space ($\mathbf{y}$), the neighbourhood preservation metric v is computed as follows:

$$v = \frac{1}{|V|} \sum_i \frac{|N_G(\mathbf{x}_i, r_G) \cap N_G(\mathbf{y}_i, r_G)|}{|N_G(\mathbf{x}_i, r_G) \cup N_G(\mathbf{y}_i, r_G)|} \in [0, 1], \quad (4)$$

where a higher value indicates better preservation of local structures.

B. Graph Coarsening and Graph Sparsification

Given a graph $G = (V, E)$, graph sparsification implies leaving V as is, but decreasing the number of edges. This is often done by starting with $G' = (V, \emptyset)$, and then adding edges back in from the original edge set E based on structural criteria or sampling probabilities. Coarsening, on the other hand, as distinguished by authors such as Bravo-Hermsdorff & Gunderson [11] and Loukas [12], implies decreasing both the number of edges *and* the number of nodes. This is often done by acting on the graph itself by clustering nodes or contracting edges (merging neighbouring nodes into a single new node).

In the context of NNP-NET, first recall (as discussed in Chapter I) the purpose of the subgraph extraction method in its pipeline: it must extract a subgraph that is small enough for tsNET* to quickly compute a high-quality ground truth layout, yet representative enough of the overall graph structure so that a neural network can be trained on it to compute a similarly high-quality layout for the entire graph. It was found by Hartskeerl et al. [6, Table 12] that the best results were obtained when this subgraph counted 10,000 nodes. Note that tsNET* has a time complexity of $O(N \log(N))$, so solely decreasing the number of edges like graph sparsification methods do seems not useful from a speed-perspective. An argument could be made for its usefulness in preventing triggering NNP-NET’s slow fallback subgraph extraction method. Namely, this fallback is triggered when the quick clustering method stalls, which tends to happen on dense graphs (more on that in Section II-C). However, as discussed in the remainder of this chapter, there exist in related literature methods specifically designed to combat this stalling behaviour, making it redundant to check for this side effect in graph sparsification. Because graph sparsification fails to reduce the node count required to accelerate tsNET*, and its secondary benefits are better achieved through other means, this thesis focuses exclusively on graph coarsening.

C. NNP-NET’s Clustering Scheme & Pivot-Points Fallback

1) *Clustering Scheme:* As outlined in detail in Algorithm 1, the method by which NNP-NET generates a coarsened graph, given a graph $G = (V, E)$, is to first sort V increasingly by degree. It then iterates through V , where each node is clustered, or ‘merged’, with all its direct neighbours that have not yet been clustered. We obtain a coarsened graph G' when all nodes have been clustered. This scheme is repeated with G' as input graph, until the target of 10,000 or fewer nodes is reached. This method of graph coarsening is similar to the one described by Zhu et al. in DRGraph [4], with the notable difference being that DRGraph iterates through the nodes randomly to minimize computational overhead. In contrast, NNP-NET deliberately sorts and selects nodes with the lowest degree first. While this adds a sorting step, it preserves the graph’s topological skeleton more effectively by ensuring that peripheral nodes are absorbed into local hubs before the hubs themselves are merged.

Algorithm 1 NNP-NET's Clustering Method

Input: Graph $G = (V, E)$, Target size S

Output: Coarser Graph G'

Let $G_{curr} \leftarrow G$

while $|V_{curr}| > S$ **do**

Sort nodes V_{curr} by increasing degree {Select centers with lowest neighbor count first}

Let $Clusters \leftarrow \emptyset$

Let $Marked \leftarrow \emptyset$ {Set of clustered nodes}

for each node $c \in V_{curr}$ **do**

if $c \notin Marked$ **then**

 Create new cluster $C_c \leftarrow \{c\}$

$Marked \leftarrow Marked \cup \{c\}$

for each neighbor v of c **do**

if $v \notin Marked$ **then**

$C_c \leftarrow C_c \cup \{v\}$

$Marked \leftarrow Marked \cup \{v\}$

end if

end for

$Clusters \leftarrow Clusters \cup \{C_c\}$

end if

end for

Construct G_{next} from $Clusters$:

$V_{next} \leftarrow \{c \mid C_c \in Clusters\}$ {Cluster centers become nodes}

$E_{next} \leftarrow \{(u, v) \mid \exists x \in C_u, y \in C_v \text{ s.t. } (x, y) \in E_{curr}\}$ {Connect adjacent clusters}

if $|V_{next}| > 0.95|V_{curr}|$ **then**

break {Stop if reduction is insufficient ($< 5\%$)}

else

$G_{curr} \leftarrow G_{next}$

end if

end while

if $|V_{curr}| \leq S$ **then**

return G_{curr}

else

return **PivotBasedSelection**(G_{curr}, S) {Trigger Algorithm 2}

end if

Time complexity: $O(N)$ (typically bounded iterations)

2) *Clustering Scheme Stalling*: However, this peripheral-first merging strategy can become a vulnerability on specific graph structures, which can cause the clustering scheme to stall. As reflected in the dataset used for this thesis (see Table III in Chapter III-C), this stalling behaviour consistently happens for graphs that have two specific properties that will be defined next: a negative degree assortativity and a high degree heterogeneity.

Degree assortativity, in essence, denotes how probable it is that two random nodes have a similar degree. The closer this value is to 1, the more probable it is that they have a similar degree, and the closer this value is to -1, the more improbable it is that they have a similar degree. It is often denoted by r , and is defined by Newman [13] as

$$r = \frac{\sum_{i=1}^{|E|} (k_{s_i} - \langle k \rangle_e)(k_{t_i} - \langle k \rangle_e)}{\sqrt{\sum_{i=1}^{|E|} (k_{s_i} - \langle k \rangle_e)^2} \sqrt{\sum_{i=1}^{|E|} (k_{t_i} - \langle k \rangle_e)^2}} \in [-1, 1], \quad (5)$$

where k_{s_i} is the degree of the source node for the i 'th edge, k_{t_i} the degree of the target node for the i 'th edge, and $\langle k \rangle_e$ the mean degree of the endpoints of the edges. (Note: this can be different from the mean degree $\langle k \rangle$.) It is the Pearson correlation coefficient of the degrees of the nodes at both ends of an edge.

Degree heterogeneity measures the variance in degrees. A high degree heterogeneity implies that the graph contains a small number of hub nodes, and a large number of outskirts nodes, whereas a low degree heterogeneity implies that the majority of

nodes have a roughly similar number of neighbours. It is denoted as κ , and is defined by Newman [14] as

$$\kappa = \frac{\langle k^2 \rangle}{\langle k \rangle^2}, \quad (6)$$

where $\langle k \rangle$ is the mean of the degrees over the entire graph and $\langle k^2 \rangle$ is the mean of the squared degrees over the entire graph. In the context of this thesis, we empirically define a ‘high’ degree heterogeneity as $\kappa \geq 50$, as this threshold (when combined with a negative degree assortativity) consistently triggers the clustering scheme’s stalling behaviour across our dataset.

Note that a negative degree assortativity does not necessarily imply a high degree heterogeneity. Geographic graphs such as *ok2010* and *tx2010* (describing census blocks in Oklahoma and Texas respectively, see Table VII for visual representations of these graphs), for instance, have a negative degree assortativity but a low degree heterogeneity: since they represent geographic areas and the adjacent areas they border, physical space prevents any single area from having thousands of neighbours. There are no outliers pushing the numerator up, so the degree heterogeneity remains low. However, these networks *do* have a tendency for smaller areas with fewer neighbours (like edge or corner blocks) to connect to central areas with more neighbours, hence the negative degree assortativity.

But a graph possessing *both* a negative degree assortativity and high degree heterogeneity implies the presence of hub-and-outskirt-nodes structures. This specific structure then traps the clustering algorithm. Recall (as outlined in Chapter II-C1) that the algorithm processes nodes in increasing order of degree. Consequently, it begins by processing the low-degree outskirts nodes. When an outskirts node seeks a neighbour to cluster with, the disassortative nature of the graph makes it so that its neighbour is the central hub. Thus, the few available hubs are quickly absorbed into clusters early in the process. Once the hubs are clustered, the remaining outskirts nodes (which only connect to the hub and have no edges between themselves) are left with no unclustered neighbours to merge with. With the vast majority of the network’s nodes stranded as unmergeable singletons, the reduction rate plummets and the algorithm stalls.

To illustrate this stalling behaviour in its purest form, let $K_{2,k}$ be the complete bipartite graph with two nodes on the left and $k > 2$ nodes on the right side, as shown in Figure 2. In this case, the clustering scheme would first create one cluster consisting of the two nodes of the left and one node on the right. After this, no more clusters can be made: the two nodes on the left are already clustered, and the nodes on the right have no other neighbours to cluster with. Hence, the number of nodes of G' is simply given as $|V| - 2 = k$, or the ‘rate’ of coarsening as $\frac{k}{k+2} \rightarrow 1$ as $k \rightarrow \infty$.

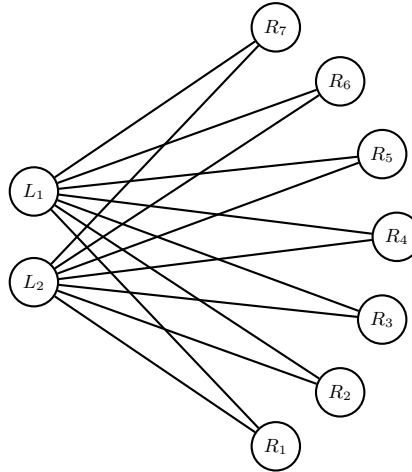


Fig. 2. Bipartite graph with 2 central nodes and 7 outskirts nodes.

3) *Pivot-Points Fallback*: In order to deal with this stalling behaviour, a fallback mechanism is triggered in NNP-NET. The algorithm abandons clustering and utilizes a pivot-points based method, which is detailed in Algorithm 2. This method begins by randomly selecting an initial node, say v , and calculating the distances from v to all other nodes. These values are stored in an array of size $|V|$, which tracks the distance from each node to the set of selected pivot-points.

The distance calculation depends on the graph type: for unweighted graphs, a breadth-first search (BFS) as described by Moore [15] is used, where the distance simply increments by 1 for each edge traversed; for weighted graphs, Dijkstra’s algorithm [16] is applied, which calculates distance by accumulating the non-negative edge weights along the shortest path.

Next, the max-min strategy is applied: the node that has the largest distance to the current set of pivots (i.e., the node farthest from its closest pivot) is selected as the next pivot-point, say v' . Then, a new BFS or Dijkstra’s search is initiated from v' to calculate the distance from v' to all other nodes. For every node, if the newly calculated distance to v' is smaller than its current entry in the array, the array is updated to reflect this new minimum distance. This iterative process of selecting the farthest node and updating the shortest paths continues until n pivot-points have been selected.

Once all n pivot-points are selected, the final subgraph is constructed. Unlike methods that merge nodes into clusters, this approach simply discards all unselected nodes. The resulting subgraph consists exactly of the n selected pivot-points. To preserve the structural topology, these n nodes are fully connected to each other, forming a complete graph. The weight of each edge connecting any two pivot-points is assigned to be the distance between them as calculated on the original graph.

While this pivot-points fallback successfully generates a subgraph for stalling topologies (i.e., topologies that cause the clustering method to stall), it comes with a *massive computational cost*, as is shown in Figure 3.

Algorithm 2 Pivot-Points Based Subgraph Extraction (NNP-NET Fallback)

Input: Graph $G = (V, E)$, Target size S
Output: Subgraph G'
Let $P \leftarrow \emptyset$ {Set of selected pivot nodes}
Select a random node $v_{start} \in V$
 $P \leftarrow P \cup \{v_{start}\}$
while $|P| < S$ **do**
 Let $v_{best} \leftarrow \text{null}$
 Let $d_{max} \leftarrow -1$
 for each node $u \in V \setminus P$ **do**
 $d_{min} \leftarrow \min_{p \in P} \text{dist}(u, p)$ {Shortest path distance to any chosen pivot}
 if $d_{min} > d_{max}$ **then**
 $d_{max} \leftarrow d_{min}$
 $v_{best} \leftarrow u$
 end if
 end for
 $P \leftarrow P \cup \{v_{best}\}$
end while
Construct G' as the subgraph induced by P
Time complexity: $O(N \cdot S)$

Subgraph Extraction Performance on Standard Topologies

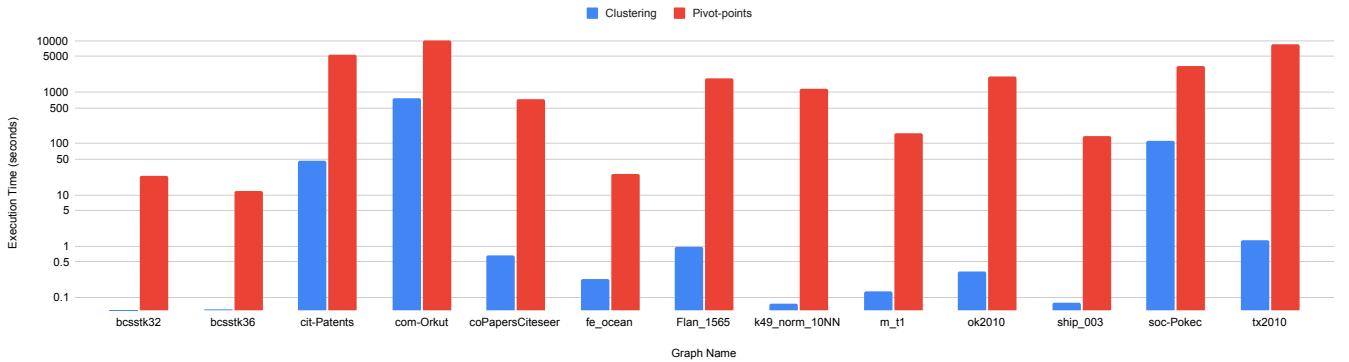


Fig. 3. Comparison of subgraph extraction execution times on standard (i.e., non-stalling) graphs, comparing NNP-NET’s fast clustering method to its fallback pivot-points method.

D. Edge Contraction

The first category of graph coarsening evaluated in this research relies on *edge contraction* driven by *maximal matchings*. In this approach, the algorithm identifies a set of disjoint node pairs (a matching) and merges each pair into a single ‘supernode’ in the coarser graph. To preserve the graph’s structural information, the weights of edges connecting these new supernodes are typically calculated as the sum of the weights of the original edges.

Algorithm 3 outlines the randomized contraction scheme as originally proposed by Hendrickson & Leland [17], and further detailed by Walshaw [18]. The method iterates through the set of unmatched nodes in a random order, pairing each node with a neighbour of minimal weight, this way generating a maximal independent set. For our purposes, the weight of each node will be initialized as 1. After this, node weights will be updated to simply reflect the number of nodes that have been merged into it. Any nodes that remain unmatched are simply carried over to the next level. This method achieves a linear time complexity of $O(N)$.

Algorithm 3 Edge Contraction Coarsening Scheme (Hendrickson & Leland, 1995; Walshaw, 2003)

Input: Graph $G = (V, E)$
Output: Coarser Graph G'
Let $P \leftarrow \emptyset$ {Set of node pairs to merge}
Let $U_V \leftarrow V$ {Set of unmatched nodes}
while $U_V \neq \emptyset$ **do**
 Select a random node $u \in U_V$
 Select a neighbour $v \in \{n \mid (u, n) \in E\} \cap U_V$ with minimal weight
 if no such neighbour exists **then**
 $v \leftarrow u$ {Walshaw’s modification for isolated/unmatchable nodes}
 end if
 $P \leftarrow P \cup \{(u, v)\}$
 $U_V \leftarrow U_V \setminus \{u, v\}$
end while
Construct G' from P :
for each node pair $(u, v) \in P$ **do**
 Merge u and v to form a new node w
 $weight(w) \leftarrow weight(u) + weight(v)$
 if u and v share common neighbours **then**
 Merge parallel edges and sum their weights
 end if
end for
Time complexity: $O(N)$, or $O(N + E)$ if starting from Algorithm 4

An extension to this scheme, proposed by Davis & Hu [8], and outlined in Algorithm 4, introduces a pre-processing step that targets nodes that have the same neighbour set. This extension is specifically designed to overcome the bottleneck found when dealing with structures as shown in Figure 2. As explained in Section II-C, applying standard pair-matching methods to these specific graph topologies introduces severe bottlenecks, drastically reducing execution speed.

To prevent this, before the standard matching phase begins, the algorithm identifies groups of nodes that have identical sets of neighbours. Pairs within these groups are then contracted. To identify those groups quickly, the algorithm uses a sum-based hash and flagging verification. For each node, a hash is computed simply by summing the integer IDs of its neighbours. Nodes are grouped into buckets based on this hash.

To resolve hash collisions within a bucket, the algorithm iterates through its nodes. The first unprocessed node in the bucket is arbitrarily chosen as a baseline ‘representative’, and its neighbours are flagged in a boolean array. Other nodes in the bucket are then verified by checking whether they have the same number of neighbours, and by checking in $O(1)$ time per neighbour if their neighbourhood is marked. Any nodes that do not perfectly match are simply skipped and become candidates to be the next representative in subsequent iterations, so that all distinct structures within a colliding bucket are correctly identified and grouped. Ultimately, this extension incurs a higher computation cost of $O(N + E)$ versus $O(N)$ of the original scheme.

Algorithm 4 Extension to Algorithm 3 (Davis & Hu, 2011)

Input: Graph $G = (V, E)$
Output: Coarser Graph G'
Let $P \leftarrow \emptyset$ {Set of node pairs to merge}
Let $U_V \leftarrow V$ {Set of unmatched nodes}
Find the collection of groups $\mathcal{S}$ such that for each $S \in \mathcal{S}$, all nodes in S share the same neighbour set
for all group $S \in \mathcal{S}$ **do**
 while $|S \cap U_V| > 1$ **do**
 Select two distinct random nodes $u, v \in S \cap U_V$
 $U_V \leftarrow U_V \setminus \{u, v\}$
 $P \leftarrow P \cup \{(u, v)\}$
 end while
end for
Continue with line 5 of Algorithm 3

E. Solar System Collapsing (SSC)

While the extension to Algorithm 3 by Davis & Hu, outlined in Algorithm 4, attempts to address the bottleneck found in structures as shown in Figure 2, it does so by introducing a dedicated pre-processing step. A graph coarsening algorithm that handles these topologies intrinsically within the main coarsening pass might give better results. Specifically, by avoiding the computational overhead of pre-processing to improve execution speed, and by using a bigger capture radius to preserve the graph’s topology more naturally. To this end, we examine the *solar system collapsing* (SSC) scheme proposed by Hachul [19].

This method bears a structural resemblance to the clustering scheme currently employed by NNP-NET (Section II-C). Both methods iterate through the graph and merge neighbourhoods rather than strictly disjoint pairs.

However, a critical distinction lies in the *radius* of capture. NNP-NET’s clustering scheme effectively operates on a radius of 1: a node merges with its immediate available neighbours. As shown in the bipartite failure case (Figure 2), this approach can stall because once the central nodes are saturated, the remaining “outsirt” nodes (distance 2 from each other) have no common neighbours left to merge with.

In contrast, the SSC method operates on a radius of 2. It structurally decomposes the graph into disjoint systems consisting of three roles:

- **Suns:** The centers of the systems.
- **Planets:** The direct neighbours of Suns.
- **Moons:** The neighbours of Planets (that are not themselves Suns or Planets).

Once all nodes in the graph have been identified as a sun, planet, or moon, each ‘solar system’ is collapsed into a single node. Importantly, Hachul proves that this method guarantees that the number of nodes after each iteration is no more than half that of the input graph, and achieves a time complexity of $O(N + E)$.

Algorithm 5 outlines the implementation. Note that while the specific calculation of edge weights in the coarsened graph is omitted for simplicity, it follows the general idea of the previous algorithms of summing weights of merged edges. Lastly, note that there are no parameters to set, making it very easy to use for the end user. One notable downside however is that it lacks fine-grained control as to the number of nodes the final graph should have. Since each iteration at least halves the number of nodes, making sure the final output has (almost) exactly 10,000 nodes becomes infeasible. This issue, and our proposed solution to it, will be discussed in Chapter III-A.

Nonetheless, this method of graph coarsening is used by the ‘Fast Multipole Multilevel Method’ (FM³) of graph drawing by Hachul [19]. In an experimental comparison of various graph drawing techniques, Hachul & Jünger [20] noted that “Only FM³ generates pleasing drawings for the majority of the challenging graphs”.

Algorithm 5 Solar System Collapsing (SSC) Scheme (Hachul, 2005)

Input: Graph $G = (V, E)$
Output: Coarser Graph G'
Let $Suns \leftarrow \emptyset$
Let $Status \leftarrow$ array of size $|V|$ initialized to **Undefined**
Let $Candidates \leftarrow V$ {Copy of V to track available sun candidates}
{**Phase 1: Select Suns**}
while $Candidates \neq \emptyset$ **do**
 Select a node $s \in Candidates$ {Randomly or by Star-Mass metric}
 $Suns \leftarrow Suns \cup \{s\}$
 $Status[s] \leftarrow \mathbf{Sun}$
 {Remove s and its distance-1 and distance-2 neighbours from candidates}
 $Region \leftarrow \{s\} \cup \{n \mid dist(s, n) \leq 2\}$
 $Candidates \leftarrow Candidates \setminus Region$
end while
{**Phase 2: Identify Planets**}
for each $s \in Suns$ **do**
 for each neighbour $p \in \{n \mid (s, n) \in E\}$ **do**
 if $Status[p] = \mathbf{Undefined}$ **then**
 $Status[p] \leftarrow \mathbf{Planet}$
 $Orbit(s) \leftarrow Orbit(s) \cup \{p\}$ { p orbits sun s }
 end if
 end for
end for
{**Phase 3: Identify Moons**}
for each node $m \in V$ where $Status[m] = \mathbf{Undefined}$ **do**
 Select a neighbour p such that $Status[p] = \mathbf{Planet}$
 $Status[m] \leftarrow \mathbf{Moon}$
 $Status[p] \leftarrow \mathbf{PlanetWithMoon}$
 $Orbit(p) \leftarrow Orbit(p) \cup \{m\}$ { m orbits planet p }
end for
{**Phase 4: Contraction**}
 $P \leftarrow \emptyset$
for each $s \in Suns$ **do**
 $System_s \leftarrow \{s\}$
 for each $p \in Orbit(s)$ **do**
 $System_s \leftarrow System_s \cup \{p\}$
 if $Status[p] = \mathbf{PlanetWithMoon}$ **then**
 $System_s \leftarrow System_s \cup Orbit(p)$ {Include p 's moons}
 end if
 end for
 $P \leftarrow P \cup \{System_s\}$
end for
Construct G' by merging each set in P into a single node. If the number of nodes of G' is higher than our target of 10,000, continue with the next iteration by inputting G' .
{Note: Guarantees $|V'| \leq |V|/2$ for connected graphs ($|V| \geq 2$)}

F. Additional Methods

Several additional graph coarsening and related algorithms exist in the broader literature that were ultimately excluded from this research. The exact reasons for rejecting each of these methods are detailed below.

Loukas [12] presents an edge contraction scheme with a more theoretically grounded approach based on *local variation*. Unlike heuristic-based edge contraction, this method uses a cost function to evaluate candidate contraction sets based on a 'local variation cost'. While this approach often yields a coarser graph that better preserves the spectral properties of the

original (which should produce high quality coarsened graphs), it has a higher computational overhead than the simpler edge contraction schemes described in Chapter II-D. Because of this, it was excluded from empirical evaluation.

Harel & Koren [21] describe a scheme that is nearly identical to NNP-NET’s fallback pivot-points method. The difference being that here it is required that the distances between *all* pairs of nodes be computed beforehand, making it computationally a lot more expensive and therefore unfit for our purposes.

Hu [1] proposes a scheme named ‘HYBRID’ that has a similar structure to the one NNP-NET currently employs: it initially uses an edge collapsing scheme, but if the rate of coarsening falls below 0.75, a fallback method is used that is similar to SSC. However, because details on the implementation of HYBRID are lacking, and because there is no code publicly available, it does not pass the ‘ease of implementation’ requirement.

Another edge contraction method is randomized edge contraction, proposed by Loukas & Vnderghyest [22]. However, since only disjoint edges are contracted at each iteration in this method, it retains the stalling behaviour in graph structures as shown in Figure 2. Hence, it fails the ‘genericity’ requirement.

Yet another edge contraction method is the one described by Bravo-Hermsdorff & Gunderson [11]. It employs edge contraction as its primary mechanism for node reduction, but simultaneously performs edge deletion (sparsification) to preserve the graph’s spectral properties. The problem with this method, then, is a lack of publicly available source code. Even more, the graph coarsening scheme as described is too slow to pass the ‘speed’ requirement.

Jin et al. [23] describe two graph coarsening schemes: Multilevel Graph Coarsening (MGC), and Spectral Graph Coarsening (SGC). However, the time complexities for these schemes are respectively (roughly) $O(|E|(|V|^2 - n^2))$ and $O(n|E| + nN^2)$, making them slower than even NNP-NET’s current fallback method on graphs where $N \gg n$.

The graph coarsening scheme described by Gajer & Kobourov [24] only produces *point clouds*. That is, given a graph $G = (V, E)$, it produces a strict subgraph $G' = (V', \emptyset)$ with $V' \subset V$. Potentially some method could be implemented of reinserting edges, but then the question of edge weights becomes non-trivial. In any case, it does not meet the ‘ease of implementation’ requirement.

A rather complex graph coarsening scheme is proposed by Archambault et al. [25]. The scheme consists of multiple passes: detecting trees, detecting biconnected components, detecting high dimensional embedders, detecting complete graphs. However, for near-complete graphs, the performance degrades to $O(N^3)$. Hence, this scheme does not pass the ‘speed’ requirement.

Finally, we will mention a graph coarsening *post-processing* step proposed by Cai et al. [26]. More specifically, a neural network (a Graph cOarsening RefinemEnt Network or GOREN) is trained on some collection of graphs. Then, once a graph coarsening scheme has been applied to a graph, GOREN is applied to the coarsened graph in order to improve the edge weights. In the context of NNP-NET, this could potentially help generate better graph drawings. However, the problem with this approach is the question of *genericity*. In [26], there is not one GOREN applied on all kinds of graphs; instead, for different kinds of graphs (synthetic graphs, large graphs), a different GOREN is trained. This is incompatible with our ‘genericity’ requirement. Potentially, a *foundational* GOREN could be trained with the right (large) dataset of graphs, but this falls grossly outside the ‘ease of implementation’ requirement.

Method	Ease of impl.	Genericity	Speed	Ease of use	Weighted graph support
Clustering	Already impl.	×	$O(N)$	++	Produces weighted subgraph (approx.). Ignores weights during selection.
Pivot-Points	Already impl.	✓	$O(nN)$	++	Produces weighted subgraph (exact). Uses weights during pivot selection.
Pivot-points Optimization	Medium	✓	$O(nN)$	++	Produces weighted subgraph (exact). Uses weights during pivot selection.
Edge Contraction	Easy	×	$O(N)$	++	Produces weighted subgraph (exact). Uses weights during edge contraction.
Edge Contraction Extension	Easy	✓	$O(N + E)$	++	Produces weighted subgraph (exact). Uses weights during edge contraction.
Solar System Collapsing	Medium	✓	$O(N + E)$	++	Produces weighted subgraph (exact). Uses weights during solar collapsing.

TABLE II

AN OVERVIEW OF ALL SUBGRAPH EXTRACTION SCHEMES UNDER CONSIDERATION. ‘GENERICITY’ INDICATES WHETHER THE METHOD CAN ROBUSTLY PROCESS ALL TESTED GRAPHS WITHOUT STALLING, N DENOTES THE NUMBER OF NODES, E DENOTES THE NUMBER OF EDGES, AND n DENOTES THE TARGET NUMBER OF NODES FOR THE OUTPUT GRAPH.

III. METHOD

This chapter details the methodology used to answer our research question outlined in Chapter I. Firstly, in Section III-A, we introduce a refined version of the SSC method mentioned in Chapter II-E. Next, in Chapter III-B, we outline an attempt at optimizing NNP-NET’s pivot-points based coarsening method. Section III-C elaborates on the dataset of 25 graphs used to stress-test the different subgraph extraction methods. Finally, Section III-D outlines the experimental setup, hardware constraints, and benchmark configurations used to evaluate performance.

A. Adaptive Solar System Collapsing (ASSC)

The original SSC method (see Chapter II-E) was designed to rapidly reduce the size of a graph by categorizing nodes into suns, planets, and moons, and subsequently contracting planets and moons into their respective suns. While the aggressive contraction is computationally efficient –and hence viable for our purposes– it suffers from a significant drawback: *a lack of control over the reduction factor per iteration*.

Because of this, the algorithm would ‘overshoot’ its target greatly on every test: when applying SSC to our dataset of 25 graphs (as listed in Table III), the returned subgraphs all overshoot the target of 10,000 nodes; node counts ranged from 258 to 9751, with an average of 2913. Since the quality of the final graph drawing is influenced by the size of the subgraph (as shown by Hartskeerl et al. [6, Appendix C]), resolving this overshoot was necessary.

1) *Refilling Strategy*: To mitigate this, an initial alternative distance-based ‘refilling’ (or partial contraction) strategy was designed and tested. In this experimental approach, if a full contraction iteration would overshoot the target, the algorithm halted standard contraction and selectively promoted nodes to achieve the exact target size. This was executed through the following sequence:

- **Distance Calculation**: The cumulative hop distance from all non-sun nodes (planets and moons) to their assigned suns was computed.
- **Farthest-Node Promotion**: All non-sun nodes were sorted in descending order based on their distance. The farthest nodes were sequentially promoted back to sun status until the projected graph size perfectly matched the 10,000-node target.
- **Orbital Reassignment**: The hierarchy was updated by mapping all remaining non-sun nodes to their respective suns by walking the orbital chain. This seamlessly handled promoted nodes; for example, if a planet was promoted to a sun, its dependent moons would walk up the chain and correctly register the newly promoted node as their new sun.
- **Edge Weight Preservation**: To calculate accurate edge weights in the coarsened graph, a separate list was maintained to preserve the original hop distances, ensuring topological accuracy even after the promotion logic was applied.

Ultimately, this distance-based promotion strategy was discarded because it introduced structural inconsistencies into the final layouts. Visual inspections revealed that the resulting graph drawings were sometimes lopsided, with regions exhibiting disproportionate detail or distinct structural variations compared to the rest of the graph (see Figure 4).

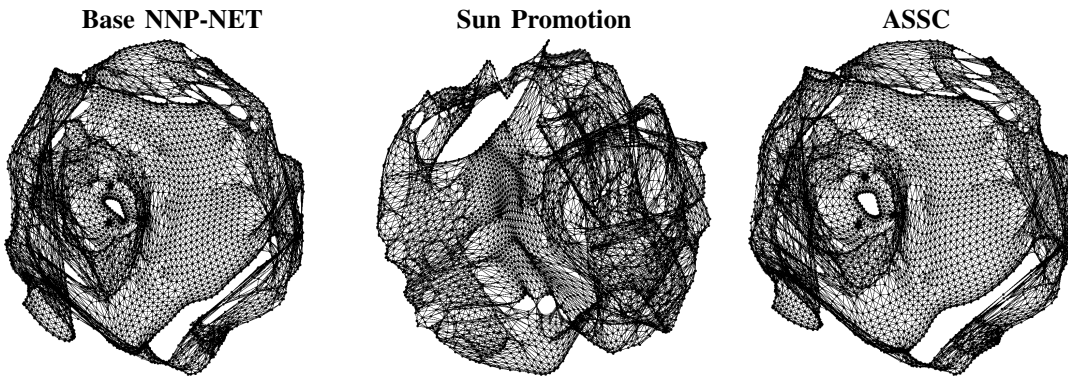


Fig. 4. Visual comparison of the *bcsstk36* graph to illustrate the structural inconsistencies introduced by the distance-based sun promotion strategy. The sun promotion layout exhibits lopsided density compared to the baseline, a distortion that is avoided by the fine-grained clustering fallback used in ASSC.

2) *Clustering Fallback*: Instead, the algorithm was redesigned to bridge the remaining gap to the 10,000-node target by falling back to NNP-NET’s default clustering method. This approach proved highly effective and forms the basis of the ASSC method. It retains the rapid reduction capability of the original approach while aiming to closely target the node count. Crucially, the preceding SSC iterations prove sufficient to collapse the structural bottlenecks that typically cause the standard clustering method to stall.

The method operates through the following four phases:

- 1) **Solar System Collapsing:** The graph is iteratively coarsened using the solar system method as outlined in Algorithm 5. Crucially, the state of the graph prior to each iteration is preserved in memory, alongside the over-collapsed graph if an overshoot occurs.
- 2) **Overshoot Detection:** If a solar system iteration causes the node count to drop below the target size, the algorithm detects the overshoot, temporarily discards the over-collapsed graph, and reverts to the preserved graph from the previous step.
- 3) **Fine-Grained Fallback:** To bridge the remaining gap to the target size, the algorithm switches to NNP-NET’s fast clustering method. Because this method reduces graphs much more gradually, it gives a much more fine-grained control. In our testing, this gradual reduction consistently brought the node count to within a 0.5% margin of the target, provided the clustering wouldn’t stall.
- 4) **Stall Handling:** In one case (the *web-Stanford* graph, visualized later in Table IX), this more fine-grained clustering method stalled (i.e., it failed to reduce the graph size by at least 5% per iteration). If this stalling behaviour is detected, the algorithm abandons the clustering approach to avoid excessive computation and quality degradation (discussed in more detail in Chapter V-A). Instead, it accepts the over-collapsed solar system subgraph generated during Phase 1. This guarantees that the algorithm will always produce a valid, smaller subgraph, even on difficult topologies.

Algorithm 6 outlines the full implementation of ASSC. Despite incorporating the more complex original NNP-NET clustering as a fallback, the ASSC scheme preserves the highly efficient $O(|V| + |E|)$ time complexity of SSC. As is shown by Hachul [19, Lemma 4.6], SSC guarantees that the node count decreases by at least half in each iteration. Because of this, the computational cost of phase 1 and 2 across successive iterations scales proportionally to $(|V| + |E|) + \frac{1}{2}(|V| + |E|) + \frac{1}{4}(|V| + |E|) + \dots$, forming a converging geometric series bounded by $O(|V| + |E|)$.

Phase 3 is only triggered on the penultimate graph, which has already been significantly reduced by Phase 1. Furthermore, the explicit stall condition (enforcing a minimum 5% reduction per iteration) ensures that the fallback phase also forms a geometric series bounded by the same $O(|V| + |E|)$. Thus, while the constant factor is marginally higher due to the fallback and state-preservation overhead, the overall time complexity remains strictly linear with respect to the original graph size.

Algorithm 6 Extension to Algorithm 5: Adaptive Solar System Collapsing (ASSC)

Input: Graph $G = (V, E)$, Target size n
Output: Coarser Graph G'
Let $G_{curr} \leftarrow G$
Let $G_{overshot} \leftarrow \emptyset$
{Phase 1 & 2: Solar System Collapsing & Overshoot Detection}
loop
 Execute Phases 1–4 of Algorithm 5 on G_{curr} to produce a coarser graph G_{temp}
 if $|V_{temp}| \leq n$ **then**
 $G_{overshot} \leftarrow G_{temp}$ {Overshoot detected; preserve over-collapsed graph}
 break
 end if
 $G_{curr} \leftarrow G_{temp}$
end loop
{Phase 3: Fine-Grained Fallback}
Let $reduction \leftarrow 0$
while $|V_{curr}| > n$ **and** $reduction < 0.95$ **do**
 $G_{prev} \leftarrow G_{curr}$
 $G_{curr} \leftarrow \text{NNP-NET Clustering Iteration}(G_{curr}, n)$ {Original NNP-NET method}
 $reduction \leftarrow \frac{|V_{curr}|}{|V_{prev}|}$
end while
{Phase 4: Stall Handling}
if $|V_{curr}| > n$ **then**
 {Fine-grained method stalled (reduced by less than 5%)}
 $G' \leftarrow G_{overshot}$ {Fall back to the overshoot Solar System subgraph}
else
 $G' \leftarrow G_{curr}$
end if
Time complexity: $O(|V| + |E|)$, maintaining the efficiency of SSC.

B. Pivot-Points Optimization

As established in Chapter II-C, unlike methods that merge nodes into clusters, the baseline pivot-points method extracts a subset of n nodes directly from the original graph. It then generates a complete graph on these selected nodes, where the edge weights correspond to the distances in the original graph.

The opportunity for optimization lies in the fact that we do not necessarily need to explore the entire graph during each subsequent BFS or Dijkstra search. During the graph traversal from a newly selected pivot v' , we continually compute the tentative distance to newly discovered nodes. If we find that the calculated distance from v' to a node is strictly worse (i.e., greater than or equal to) the node's current value in the minimum distance array, we can safely prune the search and stop exploring its neighbours. Because edge weights are strictly non-negative, any path continuing through this node will only accumulate additional distance. Therefore, its neighbours cannot possibly achieve a shorter minimum distance via v' than they already have.

Algorithm 7 outlines the full implementation of this optimized coarsening scheme. It is important to note that despite this aggressive pruning, the time complexity of the algorithm remains $O(nN)$. The optimization fundamentally just truncates the search space; it never performs more operations than the unoptimized baseline. But in the worst case scenarios, which are outlined in Appendix A, the algorithm must still traverse the entire graph without pruning any paths every time, taking $O(N)$ operations per pivot. Since we require n pivot-points, that gives us $O(nN)$. However, in practice, this early-stopping mechanism does drastically reduce the average number of nodes and edges explored during subsequent iterations, which results in significantly faster empirical runtimes, as explained in Chapter IV-A2.

Algorithm 7 Pivot-Points Optimization

Input: Graph $G = (V, E)$, Target size n

Output: Coarser Graph G'

Let $P \leftarrow \emptyset$ {Set of pivot nodes}

Let $D \leftarrow$ array of size $|V|$ initialized to ∞ {Minimum distance to any pivot}

{Phase 1: Initial Pivot}

Select the first node in the graph array: $v \leftarrow V[0]$

$P \leftarrow \{v\}$

if G is weighted **then**

 Run Dijkstra’s algorithm from v

else

 Run Breadth-First Search (BFS) from v

end if

For all $u \in V$: set $D[u] \leftarrow \text{distance}(v, u)$

{Phase 2: Max-Min Pivot Selection with Pruning}

while $|P| < n$ **do**

$v' \leftarrow \arg \max_{u \in V} D[u]$ {Select node farthest from current pivots}

$P \leftarrow P \cup \{v'\}$

if G is weighted **then**

 Initialize Dijkstra’s algorithm from v'

else

 Initialize BFS from v'

end if

while search is active and visiting node u with tentative distance $d(u)$ **do**

if $d(u) \geq D[u]$ **then**

 Prune search: do not evaluate neighbours of u along this path

else

$D[u] \leftarrow d(u)$

 Continue exploring neighbours of u

end if

end while

end while

{Phase 3: Subgraph Construction}

Construct G' using only the nodes in P

for each pair of distinct pivots $u, w \in P$ **do**

 Add an edge between u and w in G' with a weight equal to their distance in G

end for

{Note: This discards all non-pivot nodes and creates a complete graph among the pivots}

C. Setting Up an Experimental Performance Measurement

To rigorously evaluate the robustness of the subgraph extraction methods, the evaluation dataset was expanded beyond the original NNP-NET test suite. First, graphs with fewer than 15,000 nodes were discarded, as they are insufficiently large to trigger NNP-NET’s subgraph extraction phase. To populate the remainder of the dataset, 11 additional graphs were empirically selected from the SparseSuite collection by Davis & Hu [8].

Selection prioritized social network and internet graphs, as these domains frequently exhibit the high degree heterogeneity and negative degree assortativity required to stress-test the clustering scheme. The final selection was ultimately bounded by hardware constraints, excluding graphs that exceeded a 16 GB RAM processing limit. This resulted in a final dataset, shown in Table III, of 25 graphs featuring 11 graphs that triggered NNP-NET’s fallback pivot-points based subgraph extraction (denoted as ‘Stalling’). In this table, and everywhere else in this paper, the graphs are ordered in ascending order of node count, and ‘weighted’ refers to the edges having associated weights.

In instances where the SparseSuite collection provided multiple variants of a single graph (for example, auxiliary files containing labels, coordinates, or sub-sampled regions), the primary base file –which consistently corresponded to the largest file size– was selected.

TABLE III

DATASET OF 25 GRAPHS FROM THE SPARSESUITE COLLECTION USED FOR EVALUATING SUBGRAPH EXTRACTION METHODS, SORTED BY NODE COUNT. THE TERMS ‘DEGREE HETEROGENEITY’ AND ‘DEGREE ASSORTATIVITY’ ARE EXPLAINED IN CHAPTER II-C2. A ‘STALLING’ GRAPH IS ONE THAT TRIGGERS NNP-NET’S FALLBACK SUBGRAPH EXTRACTION METHOD, AND ‘WEIGHTED’ IMPLIES THAT THE GRAPH’S EDGES ARE WEIGHTED EDGES.

Graph Name	Node Count	Edge Count	Degree Het.	Degree Assort.	Stalling	Weighted
bcsstk36	23052	560044	52.602	0.070		
k49_norm_10NN	38547	309079	20.358	0.099		✓
bcsstk32	44609	985046	50.473	0.275		
m_t1	97578	4827996	108.116	0.171		
ship_003	121728	3982153	37.686	0.368		
fe_ocean	143437	409593	5.789	0.567		
ok2010	269118	637074	6.484	-0.107		✓
web-Stanford	281903	2312497	3380.830	-0.012	✓	
web-NotreDame	325729	1469679	336.354	-0.023	✓	
coPapersCiteseer	434102	16036720	212.713	0.819		
gsm_106857	589446	10584739	43.534	0.086		
web-BerkStan	685230	7600595	5628.435	-0.011	✓	
eu-2005	862664	18733713	2921.040	-0.024	✓	
tx2010	914231	2228136	6.782	-0.091		✓
web-Google	916428	5105039	269.813	-0.014	✓	
com-Youtube	1134890	2987624	494.530	-0.037	✓	
in-2004	1382908	16539643	1708.786	-0.046	✓	
Flan_1565	1564794	57920625	74.712	0.427		
soc-Pokec	1632803	30622564	73.836	0.001		
as-Skitter	1696415	11095298	1445.005	-0.081	✓	
wiki-Talk	2934385	5021410	73.035	-0.057	✓	
com-Orkut	3072441	117185083	390.345	0.016		
cit-Patents	3774768	16518947	14.496	0.142		
com-LiveJournal	3997962	34681189	123.711	0.045	✓	
uk-2002	18520486	292243663	1091.268	-0.005	✓	

D. Experimental Setup

All testing was done on a Windows 11 PC with an AMD Ryzen 5600X CPU and 16 GB of RAM. Because the original NNP-NET pipeline is written in C++, all candidate subgraph extraction methods were implemented in C++ and integrated directly into the NNP-NET codebase for unbiased comparisons. The complete codebase for these implementations is publicly available at <https://github.com/Jerran144/NNP-NET-Graph-Coarsening-Strategies>. The code provenance is as follows:

- **Pivot-points optimization:** Implemented natively based on the theoretical optimizations described in Section III-B.
- **Edge contraction:** Adapted from the C-based implementation utilized in the SuiteSparse CHOLMOD library. [27]
- **Solar system collapsing (standard and adaptive):** Adapted from the C++ implementation in the Open Graph Drawing Framework (OGDF) [28]. It should be noted that the SSC implementation used in testing contains minor artifacts regarding distance accumulation for promoted planets and suboptimal randomization, which were subsequently identified and rectified in the codebase for the ASSC method.

During testing, hardware limitations dictated performance boundaries. Specifically, the 16 GB RAM ceiling resulted in Out-Of-Memory (OOM) failures thrice during testing: the edge contraction method failed to process *uk-2002*, while the pivot-points optimization failed on both *wiki-Talk* and *uk-2002*, a limitation not present in the base NNP-NET fallback method.

For exact time measuring, the execution time of the algorithm was isolated using high-resolution clocks, explicitly excluding I/O operations (such as reading the source files from disk).

To efficiently evaluate the candidate subgraph extraction methods, the testing was structured into two distinct phases: an initial vetting phase to filter out non-viable algorithms, followed by a comprehensive benchmark on the full dataset:

- 1) **Preliminary Vetting:** A ‘go/no-go’ validation phase was conducted to quickly identify and discard ineffective approaches. To account for the variance naturally introduced by the random weight initialization in NNP-NET’s neural network, three

runs were executed per method per graph. Testing was restricted to a representative subset of six graphs chosen for specific structural challenges: three stalling topologies (*com-Youtube*, *web-NotreDame*, *com-LiveJournal*), one symmetric graph (*Flan_1565*), and two weighted graphs (*tx2010*, *ok2010*) (see Table VII for layouts of these graphs). This phase successfully identified and eliminated several alternate iterations of the Pivot-points optimization that proved slower than the base NNP-NET fallback method.

- 2) **Full Dataset Benchmark:** The methods that passed the initial vetting were later evaluated against the full 25-graph dataset. Due to the large number of graphs and the severe computational cost of calculating the neighbourhood preservation metric, these final benchmark results (presented in Chapter IV) were collected from a single run per graph.

In all tests, the target subgraph size was $n = 10,000$. This target was found by Hartskeerl et al. [6, Appendix C] to give the best trade-off between quality and speed, and keeping the target consistent across tests ensures fair comparisons. Finally, execution times for the entire pipeline were monitored throughout testing. While a soft upper limit of two hours was targeted to maintain the speed requirement outlined in Chapter I, overnight batch processing allowed one highly intensive run (base NNP-NET on *uk-2002*) to reach 3.6 hours.

In addition to recording execution times and quality metrics, the extracted subgraphs were visually rendered, as is shown in Table VIII. These renders allow for a direct visual comparison between the different coarsening methods, making it easier to identify how structural differences in the extracted subgraphs translate into variations in the final layout quality.

IV. RESULTS

This chapter presents the empirical evaluation of the proposed subgraph extraction methods (edge contraction (Chapter II-D), SSC (Chapter II-E), ASSC (Chapter III-A), and the pivot-points optimization (Chapter III-B)), comparing their performance against the baseline NNP-NET pipeline. The primary objective of these experiments is to determine whether the severe runtime bottleneck caused by NNP-NET’s fallback method for subgraph extraction can be eliminated without sacrificing the quality of the final graph drawing.

The subsequent data demonstrates that *this core research problem has been successfully solved*. The introduced methods, most notably ASSC, effectively bypass the stalling behaviour inherent to complex network topologies. Furthermore, the results show that even for standard graphs that did not originally trigger the NNP-NET’s slow fallback method, the proposed solutions generally improve or preserve both overall pipeline speed and layout quality.

This chapter is structured as follows: First, Chapter IV-A presents a comprehensive analysis of execution times, examining both the isolated subgraph extraction phase and the end-to-end pipeline performance. Then, Chapter IV-B assesses the quality of the generated layouts using quantitative metrics (stress and neighbourhood preservation) alongside qualitative visual comparisons.

A. Execution Time

To evaluate the computational efficiency of the proposed subgraph extraction methods, we first analyse their performance in isolation, followed by their impact on the end-to-end NNP-NET pipeline. Average, median, and minimum speed-ups are across all tested methods are shown in Table IV.

TABLE IV
RELATIVE EXECUTION TIME SPEED-UP OF BOTH THE SUBGRAPH EXTRACTION PHASE AND THE END-TO-END PIPELINE COMPARED TO THE BASE NNP-NET BASELINE. VALUES GREATER THAN 1.0 INDICATE A SPEED-UP.

Topology	Method	Subgraph Extraction			End-to-End Pipeline		
		Avg.	Med.	Min.	Avg.	Med.	Min.
Standard	Base NNP-NET	1.00	1.00	1.00	1.00	1.00	1.00
	ASSC	2.37	1.01	0.54	1.19	1.06	0.83
	SSC	3.21	2.64	0.80	2.89	2.12	1.19
	Edge Contraction	0.11	0.09	0.01	1.13	0.93	0.18
Stalling	Base NNP-NET	1.00	1.00	1.00	1.00	1.00	1.00
	ASSC	437.24	387.80	4.03	9.23	5.78	1.47
	SSC	449.55	389.87	4.18	13.83	8.44	1.64
	Pivot-Points Opt.	30.39	18.51	1.54	4.71	2.39	1.14
	Edge Contraction	13.85	9.27	0.30	5.30	3.58	0.59

1) *Subgraph Extraction on Standard Topologies:* As is shown in Figure 5, for standard topologies (graphs that do not trigger NNP-NET’s fallback method) the base clustering algorithm is already highly efficient, typically executing in under two seconds. For these graphs, the differences in execution time between base NNP-NET (blue bars), SSC (yellow bars), ASSC (red bars), and the pivot-points optimization (green bars) are negligible. Notably, the orange bars demonstrate that edge contraction underperforms across this category, executing between 4x and 73x slower than the baseline (with an average slowdown of 9x and a median of 11x).

On the three largest standard topologies (ranging from 1.6 million to 3.8 million nodes), the base NNP-NET extraction time increases considerably, taking between 45 seconds and 13 minutes. Here, the efficiency of the solar system approaches becomes apparent. ASSC achieved speed-ups of 3.2x, 12x, and 2.9x across these three graphs respectively (when ordered by increasing node count), while SSC achieved similar speed-ups of 4.8x, 12x, and 3.1x. The pivot-points optimization offers no benefit here as the fallback is not triggered, and edge contraction remains consistently slower than the baseline (between 4.5x and 8x slower).

Subgraph Extraction Performance on Standard Topologies

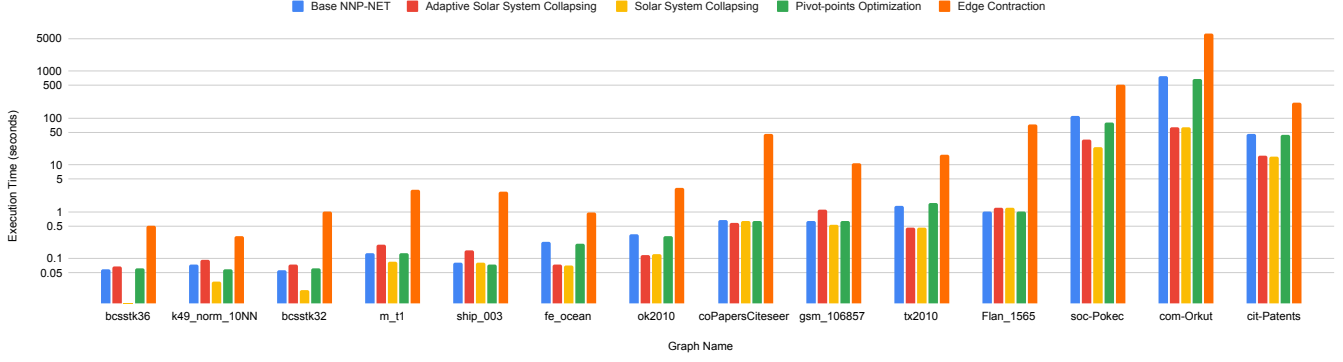


Fig. 5. Comparison of subgraph extraction execution times across evaluated methods on standard topologies (graphs that do not trigger NNP-NET’s fallback method).

2) *Subgraph Extraction on Stalling Topologies*: The primary limitation of the base NNP-NET pipeline is its severe performance degradation on stalling topologies. When evaluating just the subgraph extraction phase on these graphs (see Figure 6), the proposed methods demonstrated significant improvements.

When quantified in aggregate (as shown in Table IV), we see that SSC proved to be the fastest method overall, achieving an average speed-up of 450x (median 390x), ranging from a minimum 4.1x improvement to a maximum of 1161x. ASSC followed very closely, showing an average speed-up of 437x (median 388x), a minimum improvement of 4x, and a maximum of 1054x. In absolute terms, this best-case scenario reduced the subgraph extraction time from 57 minutes to just 3.3 seconds.

The pivot-points optimization yielded more moderate improvements: an average speed-up of 30x (median 19x), peaking at a 94x improvement. Edge contraction showed the weakest performance among the alternatives. While it averaged a 14x speed-up on most stalling topologies, it actually performed 3.4x *slower* than the baseline on the *com-LiveJournal* graph. Interestingly, *com-LiveJournal* is the only stalling graph in the dataset with a positive degree assortativity (see Table III). This topology caused the worst-case relative performance across all tested methods, but while ASSC, SSC, and the pivot-points optimization still improved upon the baseline by at least a factor of 1.5, edge contraction failed to do so.

Subgraph Extraction Performance on Stalling Topologies

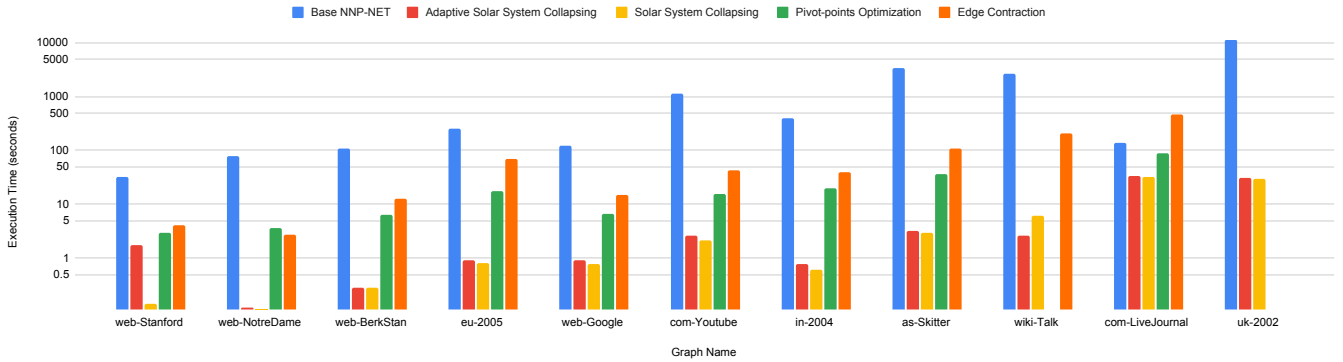


Fig. 6. Comparison of subgraph extraction execution times across evaluated methods on stalling topologies (graphs that trigger NNP-NET’s slower pivot-points fallback method).

3) *End-to-End Pipeline Performance*: While the isolated speed-ups for subgraph extraction are massive, the impact on the total end-to-end pipeline execution time is naturally compressed. This is an expected consequence of Amdahl’s Law; as the subgraph extraction time approaches zero, the total runtime becomes bound by the remaining steps (such as embedding creation, which takes approximately 67 seconds on *as-Skitter* regardless of the subgraph extraction method used).

For standard topologies, where base NNP-NET is already highly efficient, the end-to-end improvements are marginal (see Figure 7). In aggregate (Table IV), ASSC averages a 1.2x speed-up (median 1.06x), and edge contraction averages a 1.13x speed-up. SSC performs noticeably better here, with an average speed-up of 2.9x. This is due to the fact that SSC heavily overshoots the 10,000 node target, producing much smaller subgraphs that subsequently require less time for ground truth generation (see Figures 8 and 9) and neural network training.

End-to-End Pipeline Performance on Standard Topologies

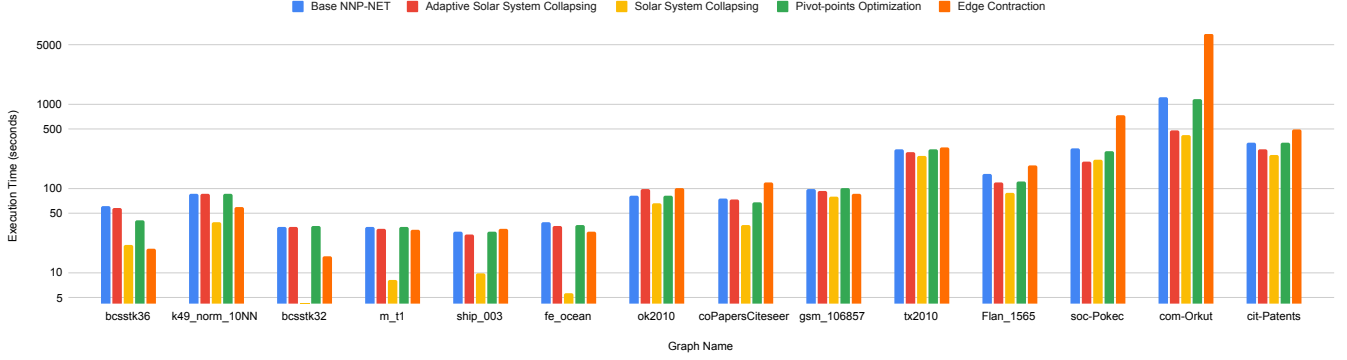


Fig. 7. Total end-to-end pipeline execution times on standard topologies. For these non-stalling graphs, the baseline NNP-NET pipeline is already highly efficient, resulting in marginal comparative improvements.

Ground Truth Generation Time on Standard Topologies

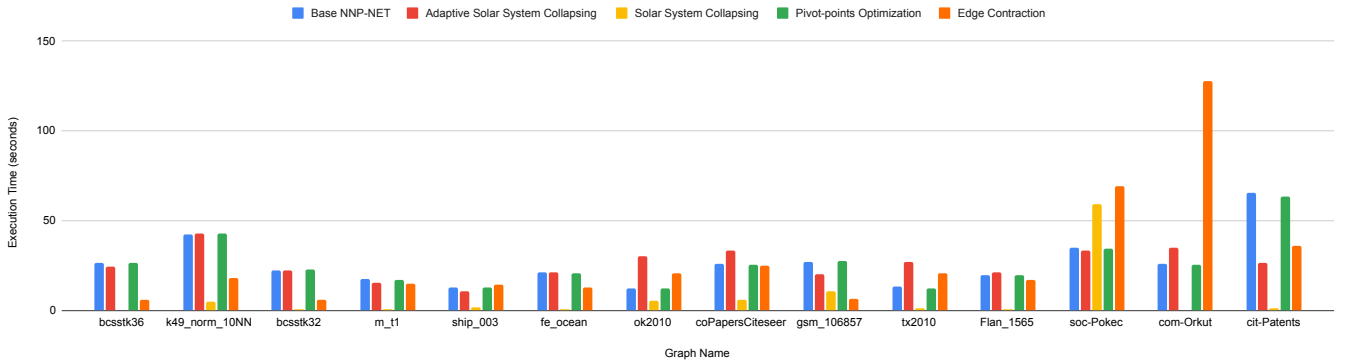


Fig. 8. Execution times for the ground truth generation phase on standard topologies. The notable speed-up for SSC is an artifact of the method overshooting the target node count, resulting in smaller subgraphs that process faster.

Ground Truth Generation Time on Stalling Topologies

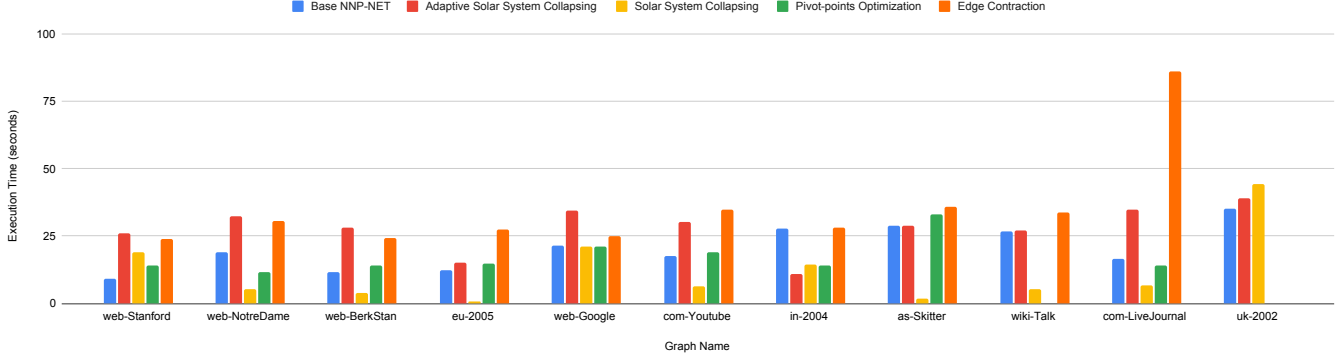


Fig. 9. Execution times for the ground truth generation phase on stalling topologies.

On *stalling topologies*, the pipeline improvements are far more pronounced, as is shown in Figure 10. As highlighted by the red bars, ASSC achieves an average end-to-end speed-up of 9.2x (median 5.8x), peaking at a 26x improvement for *as-Skitter*. Edge contraction (orange bars) and the pivot-points optimization (green bars) yield lower average speed-ups of 5.3x and 4.7x, respectively. SSC is again the fastest, as shown by the yellow bars, averaging a 13.8x speed-up and peaking at 45x. Across all four alternative methods on stalling topologies, the best-case relative performance consistently occurred on *as-Skitter*, while the worst-case consistently occurred on *com-LiveJournal*. Despite the compression of relative speed-ups across the full pipeline, reducing the total processing time of a stalling graph like *as-Skitter* by a factor of 26 confirms that ASSC successfully eliminates NNP-NET’s runtime bottleneck.

End-to-End Pipeline Performance on Stalling Topologies

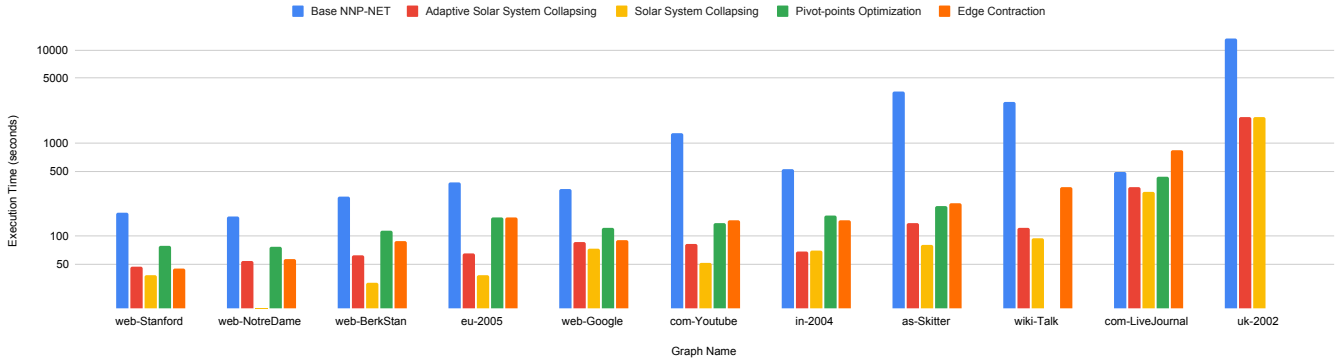


Fig. 10. Total end-to-end pipeline execution times on stalling topologies. The proposed methods, particularly ASSC, successfully eliminate the performance bottleneck caused by the baseline’s fallback method.

To see whether the proposed methods meet the strict performance requirement of taking up no more than 10% of the total pipeline runtime (as established in Chapter I), proportional execution times were analysed as well (see Table V). For standard topologies, base NNP-NET’s clustering method used an average of 8.5% of the total runtime. ASSC improved upon this, reducing this footprint to 2.9%. Edge contraction, on the other hand, failed to meet the threshold: it used up on average 23.9% of the pipeline’s total runtime.

On stalling topologies, then, we see that base NNP-NET’s fallback method dominates the pipeline: it consumes an average of 62% of the total execution time. ASSC removes this bottleneck, using *only 2.5% of the total runtime*, which is remarkably consistent with its performance on standard topologies and sitting well below the 10% target. By comparison, the pivot-points optimization sits on the boundary at exactly 10%, while edge contraction again fails to meet the threshold by taking up 31% of the runtime.

TABLE V
PERCENTAGE OF EXECUTION TIME SPENT ON THE SUBGRAPH EXTRACTION PHASE RELATIVE TO THE TOTAL NNP-NET PIPELINE RUNTIME.

Topology	Method	Average	Median	Maximum
Standard	Base NNP-NET	8.5	0.5	64.3
	ASSC	2.9	0.6	16.5
	SSC	2.8	0.9	14.9
	Edge Contraction	23.9	8.6	92.1
Stalling	Base NNP-NET	62.0	66.6	94.5
	ASSC	2.5	1.6	10.1
	SSC	3.0	1.5	10.9
	Pivot-Points Opt.	10.0	11.1	20.6
	Edge Contraction	31.0	28.1	61.0

B. Layout Quality

To assess the quality of the generated graph drawings, we evaluate the methods quantitatively using stress and neighbourhood preservation metrics (as formally defined in Chapter II-A), and qualitatively through visual inspection of the final renders.

1) *Quantitative Metrics*: For standard topologies, ASSC and base NNP-NET produce comparable quality metrics, as is shown in Figures 11 and 12. In aggregate (Table VI) Base NNP-NET exhibits marginally worse stress (1.055x higher on average), and a marginally worse neighbourhood preservation (1.061x *lower*). The pivot-points optimization behaves identically to the base NNP-NET here. SSC and edge contraction show more variance, with SSC frequently yielding lower stress values as a direct artifact of heavily over-coarsening the subgraphs.

Layout Quality (Stress) on Standard Topologies

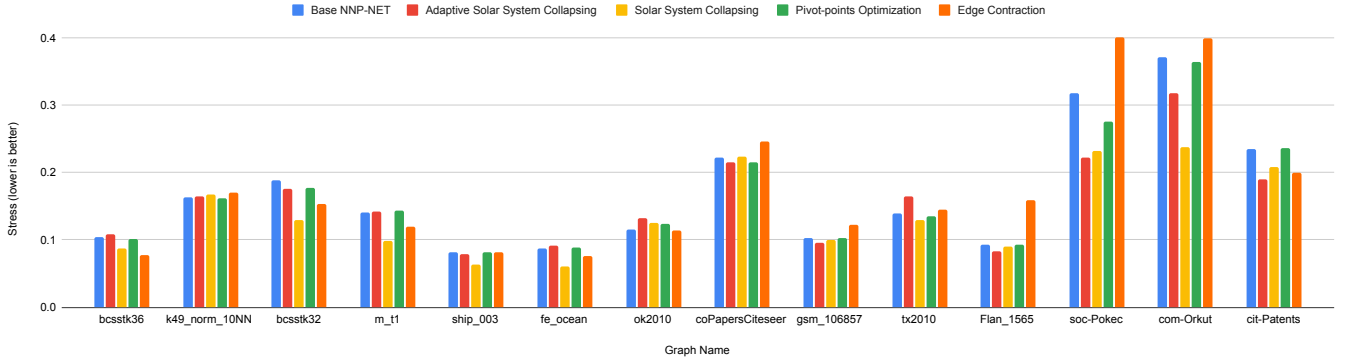


Fig. 11. Quantitative layout quality measured by stress on standard topologies. Lower values indicate better performance.

Layout Quality (Neighbourhood Preservation) on Standard Topologies

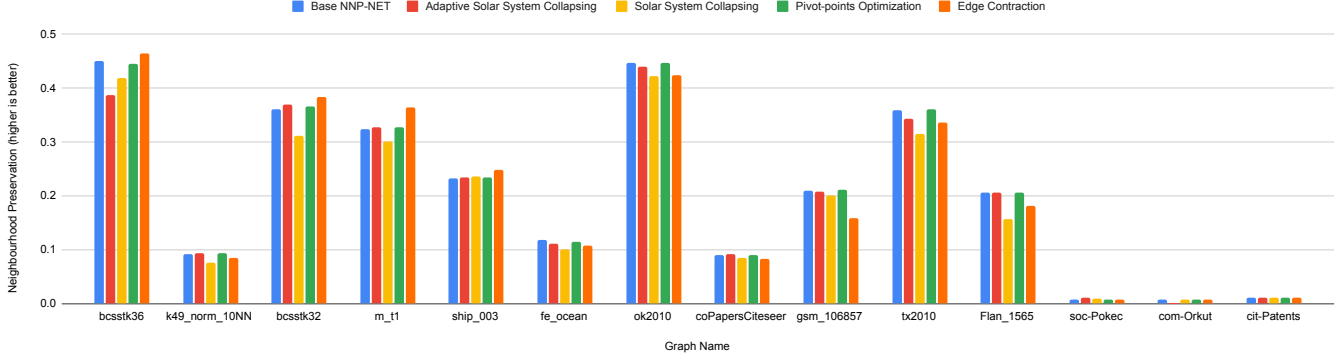


Fig. 12. Quantitative layout quality measured by neighbourhood preservation on standard topologies. Higher values indicate better performance.

TABLE VI

OVERVIEW OF LAYOUT QUALITY METRICS IMPROVEMENT ACROSS ALL TESTED METHODS. FOR BOTH METRICS, VALUES GREATER THAN 1 INDICATE AN IMPROVEMENT OVER BASE NNP-NET. NOTE THAT THE PIVOT-POINTS OPTIMIZATION'S VALUES ARE NOT LISTED, AS ITS OUTPUT IS OF THE SAME QUALITY AS BASE NNP-NET.

Topology	Method	Stress		Neighbourhood Pres.	
		Average	Median	Average	Median
Standard	Base NNP-NET	1.00	1.00	1.00	1.00
	ASSC	1.05	1.03	1.05	1.01
	SSC	1.20	1.15	0.94	0.94
	Edge Contraction	1.00	0.98	0.99	0.97
Stalling	Base NNP-NET	1.00	1.00	1.00	1.00
	ASSC	1.54	1.52	1.12	1.05
	SSC	1.81	1.54	0.88	0.90
	Edge Contraction	1.43	1.22	1.16	1.07

On stalling topologies, the differences in output quality become pronounced (see Figures 13 and 14). SSC (yellow bars) achieves the best stress scores (improving upon the baseline by 1.81x on average) but records the worst neighbourhood preservation. Conversely, ASSC (red bars) secures the best neighbourhood preservation (1.121x higher than the baseline on average) while still maintaining a strong 1.54x average improvement in stress. Edge contraction (orange bars) provides moderate improvements in both metrics, while the green bars of the pivot-points optimization mirrors the blue bars of base NNP-NET.

This inverse relationship between stress and neighbourhood preservation in the standard solar system method is an architectural consequence of subgraph size. SSC consistently overshoots the 10,000-node target (as detailed in Chapter III-A). While these much smaller subgraphs are easier for the subsequent pipeline to embed with low stress due to them having fewer constraints, they lack the topological detail required to preserve local neighbourhoods. ASSC solves this by strictly enforcing the target size, yielding an optimal balance: it retains enough structural detail to accurately preserve local neighbourhoods, while still producing a subgraph that tsNET* can embed with *less stress than base NNP-NET's fallback method*.

Layout Quality (Stress) on Stalling Topologies

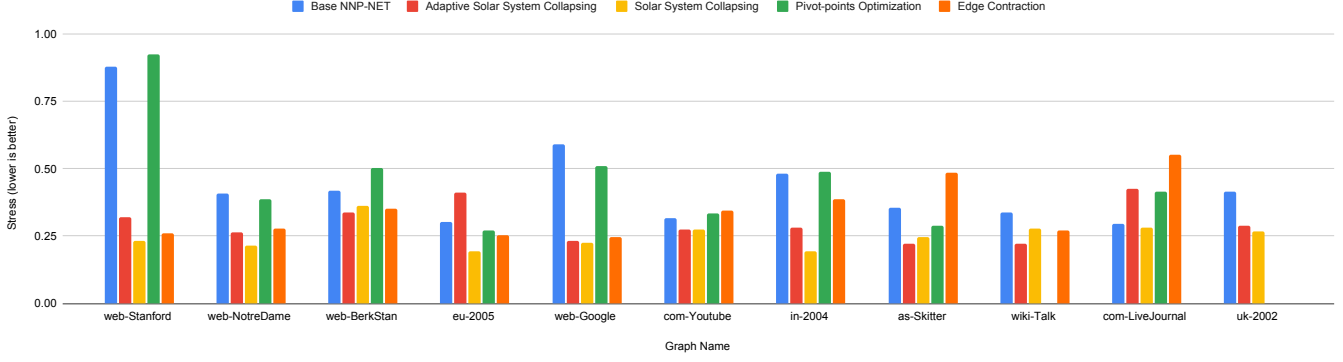


Fig. 13. Quantitative layout quality measured by stress on stalling topologies. Lower values indicate better performance.

Layout Quality (Neighbourhood Preservation) on Stalling Topologies

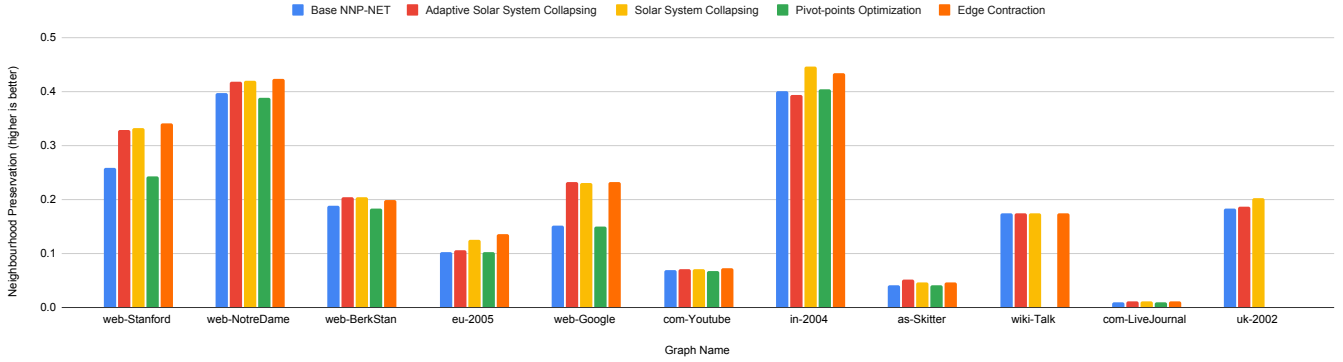


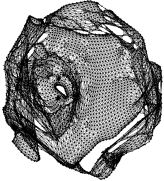
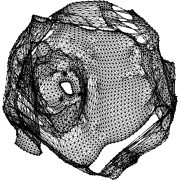
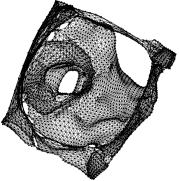
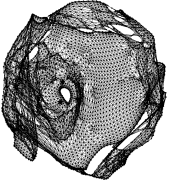
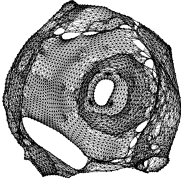
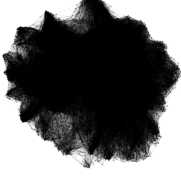
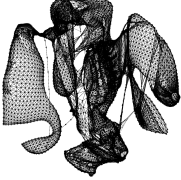
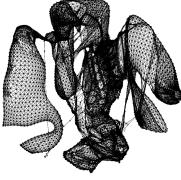
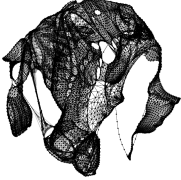
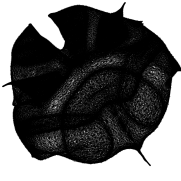
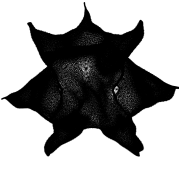
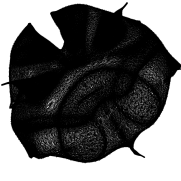
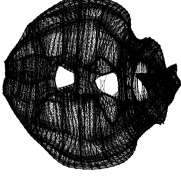
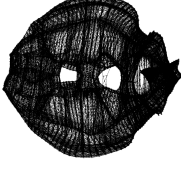
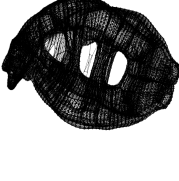
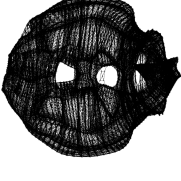
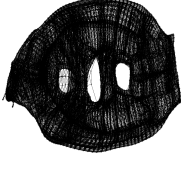
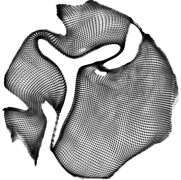
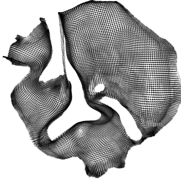
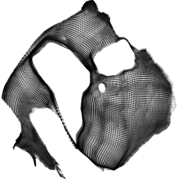
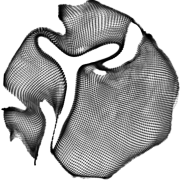
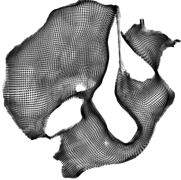
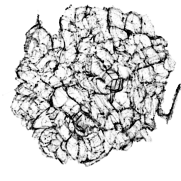
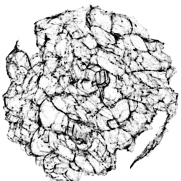
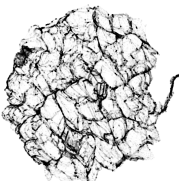
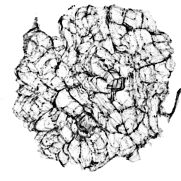
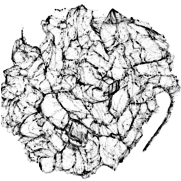
Fig. 14. Quantitative layout quality measured by neighbourhood preservation on stalling topologies. Higher values indicate better performance.

2) *Qualitative Visual Analysis:* A visual inspection of the generated layouts (see Table VII) provides necessary context to the quantitative metrics with regard to the layouts' global structures. For standard non-stalling graphs (for example, *bcsstk36*, *bcsstk32*, *m_t1*, *ship_003*), ASSC consistently mimics the high-quality outputs of the base NNP-NET. By contrast, edge contraction visibly distorts several graphs (such as *fe_ocean* and *gsm_106857*). SSC similarly introduces macroscopic structural garbling in graphs like *bcsstk36*, *bcsstk32*, *m_t1*, *fe_ocean*, and to a lesser extent *Flan_1565*. Importantly, while SSC achieved mathematically favourable stress scores by heavily over-coarsening the subgraphs, this visual garbling reveals that too much information is lost in the process.

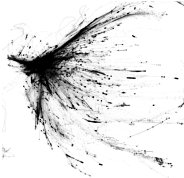
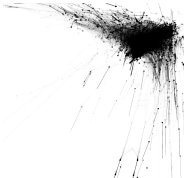
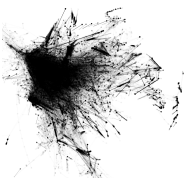
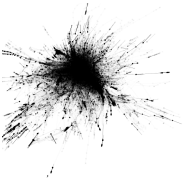
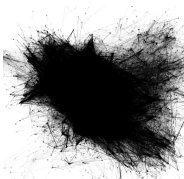
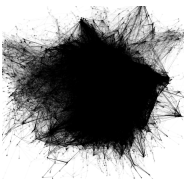
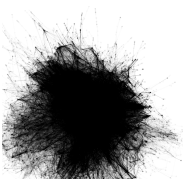
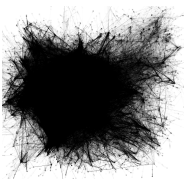
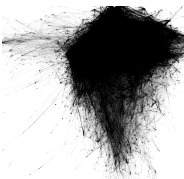
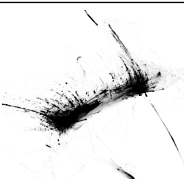
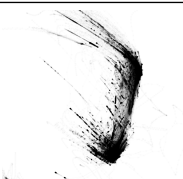
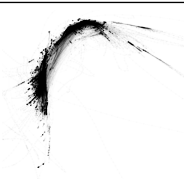
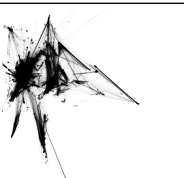
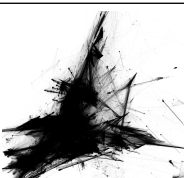
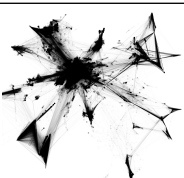
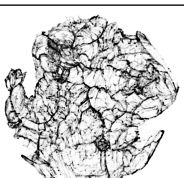
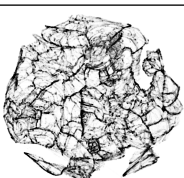
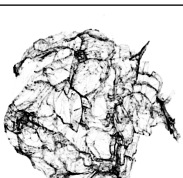
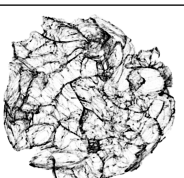
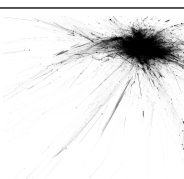
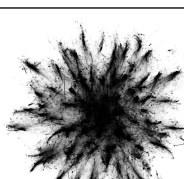
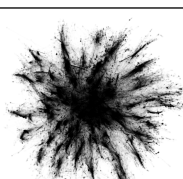
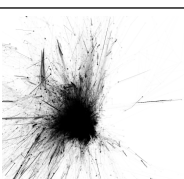
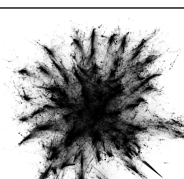
An interesting visual improvement to note is that of the *web-Google* graph, which represents web pages and hyperlinks between them. This graph renders as an unstructured, abstract cloud under both base NNP-NET and the pivot-points optimization. However, when processed with ASSC, SSC, or edge contraction, the layout successfully resolves into a structured collection of distinct clusters.

Finally note that, as shown in Table VIII, the structural differences in the final layout are more often than not already apparent in the extracted subgraphs themselves, demonstrating that the visual fidelity of the final drawing is inherently tied to the quality of the intermediate subgraph.

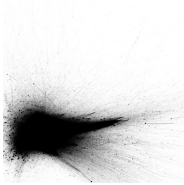
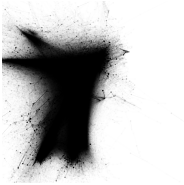
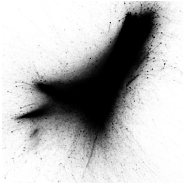
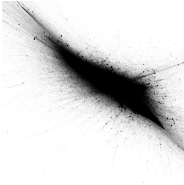
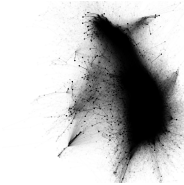
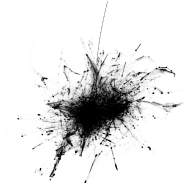
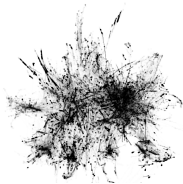
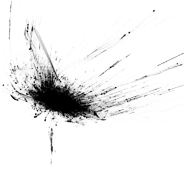
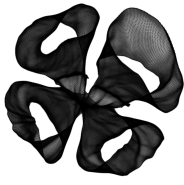
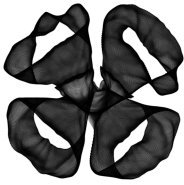
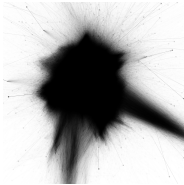
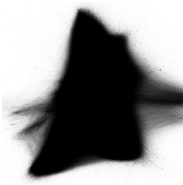
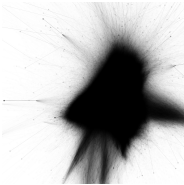
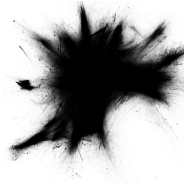
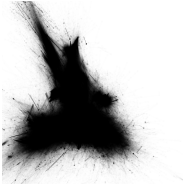
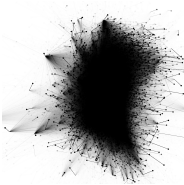
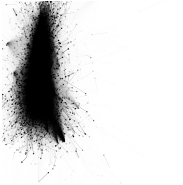
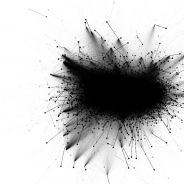
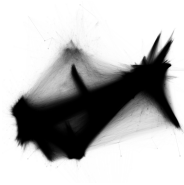
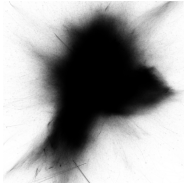
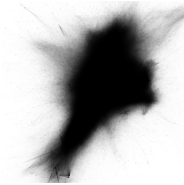
TABLE VII: Visual comparison of generated layouts across tested subgraph extraction methods. **Bold** indicates a stalling graph.

Graph	Base NNP-NET	ASSC	SSC	Pivot-points Optimization	Edge Contraction
bcsstk36					
k49_norm_10NN					
bcsstk32					
m_t1					
ship_003					
fe_ocean					
ok2010					

Continued on next page

Graph	Base NNP-NET	ASSC	SSC	Pivot-points Optimization	Edge Contraction
web-Stanford					
web-NotreDame					
coPapersCiteseer					
gsm_106857					
web-BerkStan					
eu-2005					
tx2010					
web-Google					

Continued on next page

Graph	Base NNP-NET	ASSC	SSC	Pivot-points Optimization	Edge Contraction
com-Youtube					
in-2004					
Flan_1565					
soc-Pokec					
as-Skitter					
wiki-Talk					
com-Orkut					
cit-Patents					

Continued on next page

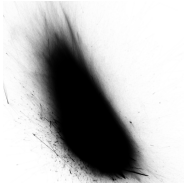
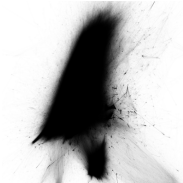
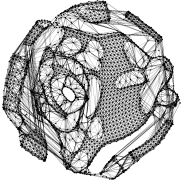
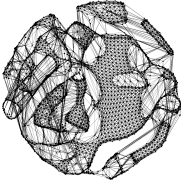
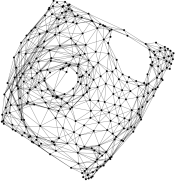
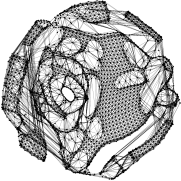
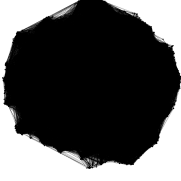
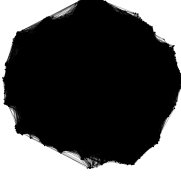
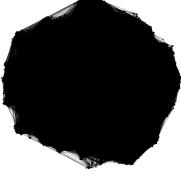
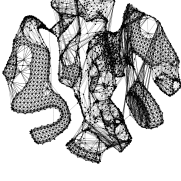
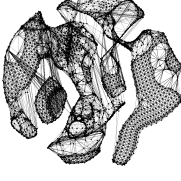
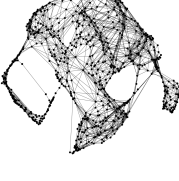
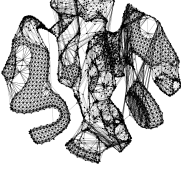
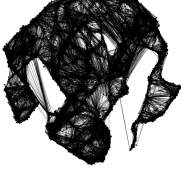
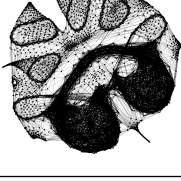
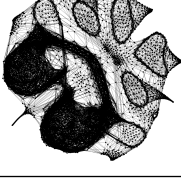
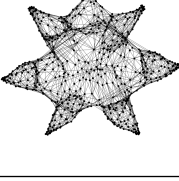
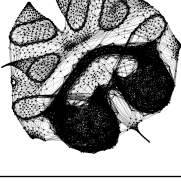
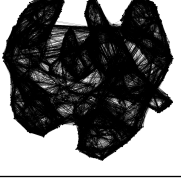
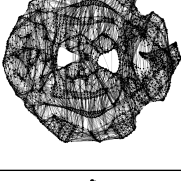
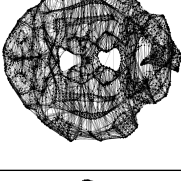
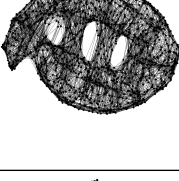
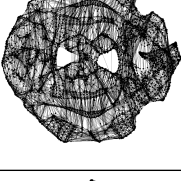
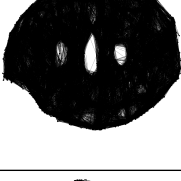
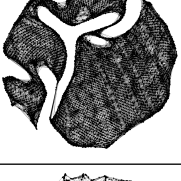
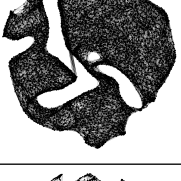
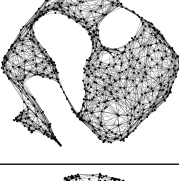
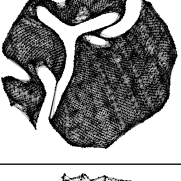
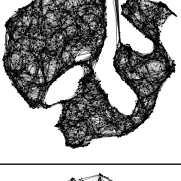
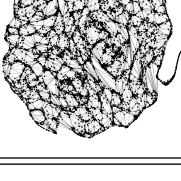
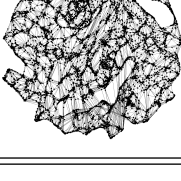
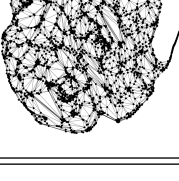
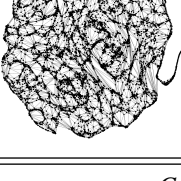
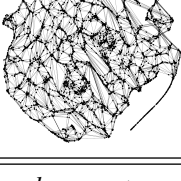
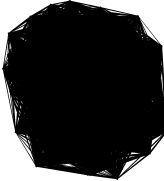
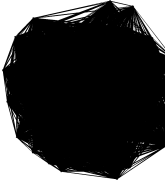
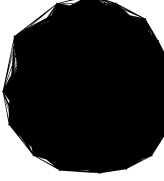
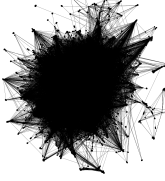
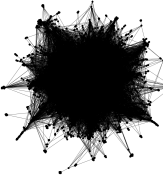
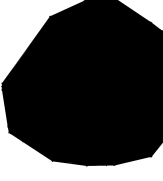
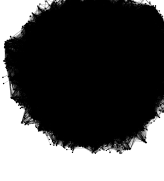
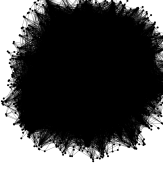
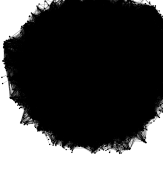
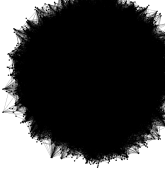
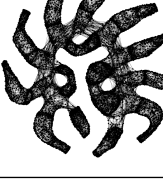
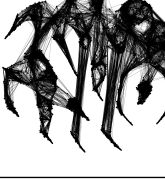
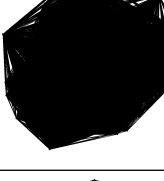
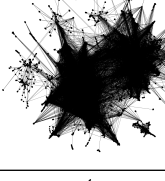
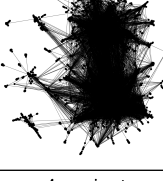
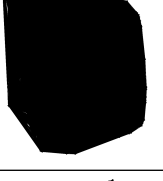
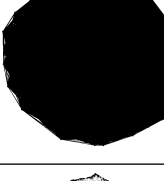
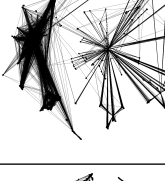
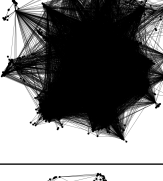
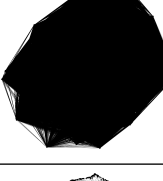
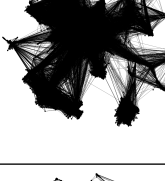
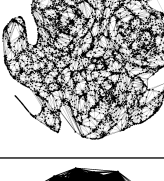
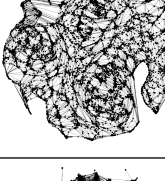
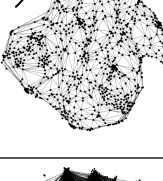
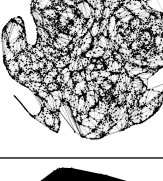
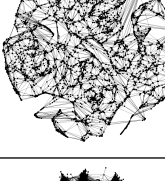
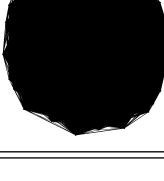
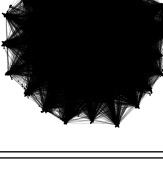
Graph	Base NNP-NET	ASSC	SSC	Pivot-points Optimization	Edge Contraction
com- LiveJournal					

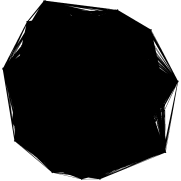
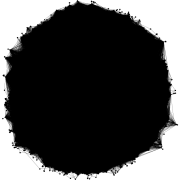
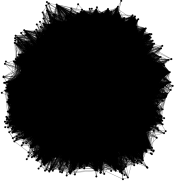
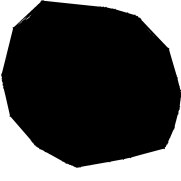
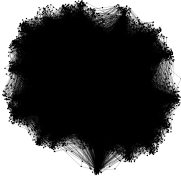
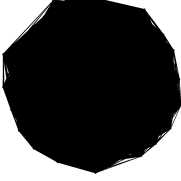
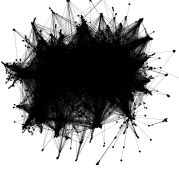
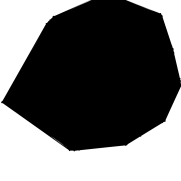
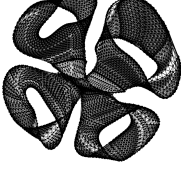
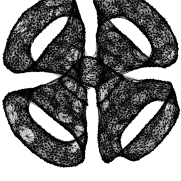
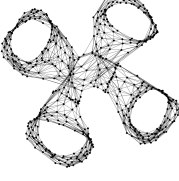
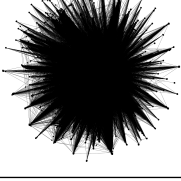
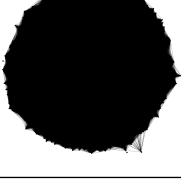
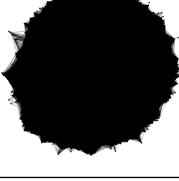
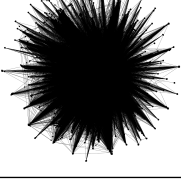
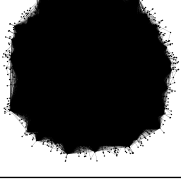
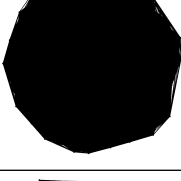
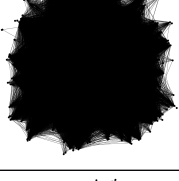
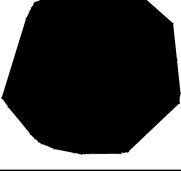
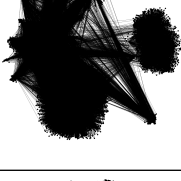
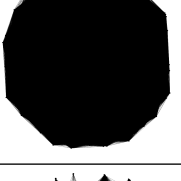
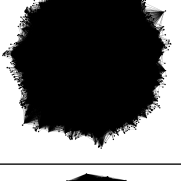
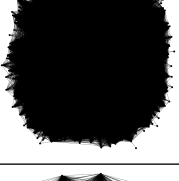
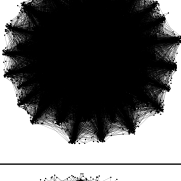
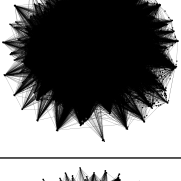
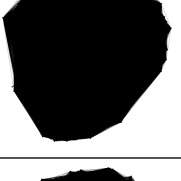
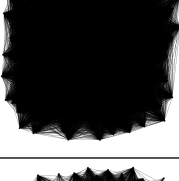
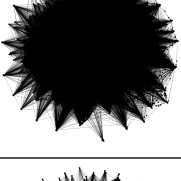
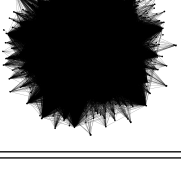
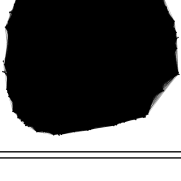
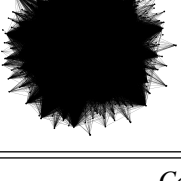
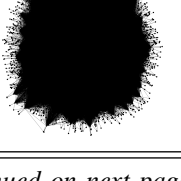
TABLE VIII: Visual comparison of extracted subgraphs across tested methods. **Bold** indicates a stalling graph.

Graph	Base NNP-NET	ASSC	SSC	Pivot-points Optimization	Edge Contraction
bcsstk36					
k49_norm_10NN					
bcsstk32					
m_t1					
ship_003					
fe_ocean					
ok2010					

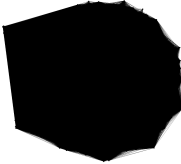
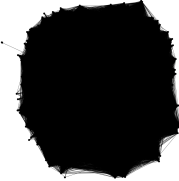
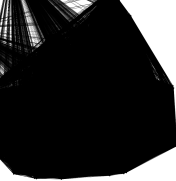
Continued on next page

Graph	Base NNP-NET	ASSC	SSC	Pivot-points Optimization	Edge Contraction
web-Stanford					
web-NotreDame					
coPapersCiteseer					
gsm_106857					
web-BerkStan					
eu-2005					
tx2010					
web-Google					

Continued on next page

Graph	Base NNP-NET	ASSC	SSC	Pivot-points Optimization	Edge Contraction
com-Youtube					
in-2004					
Flan_1565					
soc-Pokec					
as-Skitter					
wiki-Talk					
com-Orkut					
cit-Patents					

Continued on next page

Graph	Base NNP-NET	ASSC	SSC	Pivot-points Optimization	Edge Contraction
com- LiveJournal					

V. DISCUSSION

A. Handling Stalling Behaviour in ASSC

The primary objective of the ASSC method is to eliminate the severe runtime bottlenecks caused by NNP-NET’s fallback subgraph extraction method (see Section III-A), while strictly enforcing a target subgraph size of 10,000 nodes to preserve layout quality. However, to guarantee completion on all graphs, ASSC includes a stall-handling mechanism specifically for its fallback phase: if the reduction rate of the fine-grained clustering drops below 5% per iteration, the algorithm abandons this step and defaults to the over-collapsed subgraph generated during the initial solar system phase.

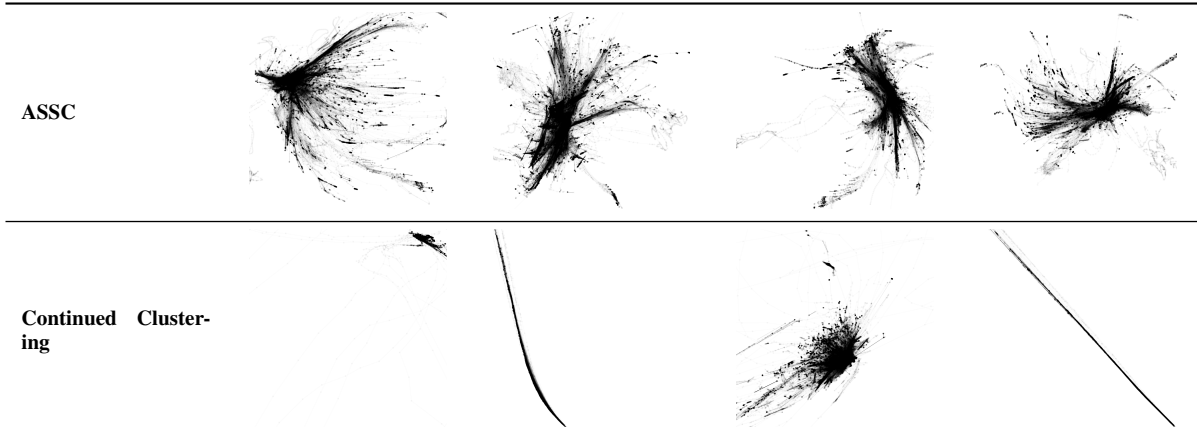
In the evaluation dataset of 25 graphs (see Table III), this stall-handling mechanism was triggered only once, by the *web-Stanford* graph. During this execution, the algorithm detected the stall and fell back to an overshoot subgraph containing 7,591 nodes.

1) *Over-Coarsened Subgraphs versus Continued Clustering*: To accurately quantify the impact of accepting this over-collapsed subgraph, a test was conducted on *web-Stanford* where the stall-handler was disabled, forcing the fine-grained clustering to continue processing the graph until the 10,000-node target was reached.

This resulted in a 42% relative increase in the subgraph extraction phase (taking 2.5 seconds versus 1.76 seconds), a 6% lower stress value, and identical neighbourhood preservation values. However, the resulting end-to-end pipeline execution times differed by less than 2%.

While the quantitative metrics suggested continued clustering yielded a marginally better stress score and identical neighbourhood preservation, visual inspection of the generated layouts show that continued clustering yields inconsistent results. To show this, four runs were conducted for both ASSC and the continued clustering, the results of which can be seen in Table IX. The four ASSC runs with over-coarsened subgraphs consistently resolved into similar abstract, scattered structures. In contrast, the continued clustering showed severe variance: while one run approached the standard output, the other three collapsed into degenerate structures; rendering as either a single dense cluster in the corner of the canvas or a linear string.

TABLE IX
VISUAL COMPARISON OF FOUR INDEPENDENT RUNS ON THE *web-Stanford* GRAPH. STANDARD ASSC CONSISTENTLY PRODUCES A SCATTERED STRUCTURE, WHEREAS CONTINUED CLUSTERING EXHIBITS SEVERE STRUCTURAL VARIANCE AND DEGENERATE GEOMETRIES.



2) *Explaining Continued Clustering’s Instability*: This visual instability highlights that the 5% reduction threshold is an important topological one, besides it being a performance safeguard. As established in Section II-C, graphs exhibiting high degree heterogeneity and negative degree assortativity (such as $K_{2,k}$ -like structures) cause the clustering scheme to rapidly absorb local hubs. This leaves a vast number of ‘outskirt’ nodes stranded without adjacent neighbours to merge with, causing the reduction rate to stall.

If the algorithm is forced to continue clustering past this stall point, it is limited to absorbing only a single outskirts node into a central hub per iteration. This issue is compounded by the algorithm’s processing order, which sorts and evaluates nodes in ascending order of degree. Consequently, it is always the outskirts leaf nodes that initiate these forced merges, repeatedly ‘swallowing’ the central super-nodes from the outside-in.

This outside-in merging mechanism creates a topological distortion due to how the algorithm calculates graph distances. When two nodes merge, the algorithm estimates the physical size of the new cluster by adding the internal merge distance to all

outward-facing edges. Because the central super-node is continuously swallowed by outskirts nodes over dozens of consecutive iterations, this internal distance is repeatedly compounded, causing the artificial edge weights to grow exponentially. By the time the graph is reduced to the 10,000-node target, these very large edge weights completely overshadow the true topological distances.

When the stochastic tsNET* algorithm attempts to generate a ground-truth layout for this distorted subgraph, it typically fails to resolve this extreme variance in edge weights. In most executions, the corrupted distances cause it to crash into a degenerate local minimum, resulting in the single dense cluster or linear string shown in the continued clustering runs. Only occasionally, seemingly by random chance, it manages to balance the corrupted distances and untangle the subgraph to produce a usable layout. Therefore, the stall-handling mechanism (and the acceptance of the over-coarsened subgraph) is a critical safeguard against this topological collapse.

3) *Preserving Theoretical Time Complexity:* Beyond preventing this topological collapse, the 5% reduction threshold is mathematically required to maintain the theoretical linear time complexity of NNP-NET’s clustering phase. As established by Hartskeerl et al. [6], ensuring that the graph is reduced by at least 5% at each coarsening step bounds the maximum number of iterations. Specifically, the cumulative cost of these iterations forms a geometric series bounded by $\sum_{k=0}^{\infty} (0.95)^k = 20$, which strictly confines the overall clustering cost to $O(N)$. Without this safeguard, a stalling clustering phase on a sufficiently large and hostile (i.e., negative degree assortativity and high degree heterogeneity, see Chapter II-C2) graph could require a very large number of iterations, leading to non-linear execution times. Thus, the stall-handler guarantees that ASSC maintains both topological integrity and theoretical efficiency under worst-case scenarios.

B. Limitations

While the proposed subgraph extraction methods (specifically ASSC) demonstrate significant improvements in efficiency and quality in the NNP-NET pipeline, the evaluation methodology is subject to several limitations.

1) *Variance:* The NNP-NET pipeline is inherently stochastic, as randomness is introduced at multiple stages of execution. In the base pipeline and the pivot-points optimization, randomness is introduced during the neural network training and within the tsNET* layout method. Except for the coarsening algorithms present in base NNP-NET and the pivot-points optimization, the different coarsening algorithms introduce further randomness: (A)SSC uses random number generation to select cluster centres (suns) and to resolve ties when assigning moons to planets. Edge contraction, as performed in Algorithm 3, randomly shuffles the node processing order.

Consequently, execution times and quality metrics can fluctuate between identical runs. Data collected during the preliminary vetting phase (where three runs were executed per method) suggests that this variance is generally stable. For instance, across three runs of ASSC on the *tx2010* graph, the stress metric only fluctuated between 0.148 and 0.154, and the total execution time varied by less than one second (spanning 286.58 to 287.48 seconds).

Despite this general stability, multiple trials per graph would have been preferred considering the randomness involved. However, for the final 25-graph benchmark, performing multiple runs was computationally unattainable. The primary bottleneck was the evaluation phase itself: calculating the neighbourhood preservation metric frequently dominated the total runtime, sometimes taking an order of magnitude longer to compute than generating the graph layout. Furthermore, as base NNP-NET scales poorly on stalling topologies, a single run on the largest graphs can take up to 3.6 hours to complete. Taking the fact into account that a single run across all graphs and methods gives 125 runs (25 graphs and 5 competing methods), it was concluded that multiple runs was unfeasible within the time constraints. Therefore, the final benchmark relies on single-run data. While the resulting data as a whole heavily validates ASSC, minor differences between methods on specific graphs may be artifacts of this variance rather than definitive algorithmic superiority.

2) *Hardware Constraints:* The experiments were conducted under strict hardware limitations, bounded by a 16 GB RAM ceiling. This restriction directly impacted the evaluation of certain coarsening methods on the largest datasets. Specifically, the edge contraction method suffered an Out-Of-Memory failure on one graph, while the pivot-points optimization failed on two of the largest graphs in the dataset. While ASSC successfully processed all 25 graphs within the hardware constraints, the inability to test these competing methods on the largest graphs limits the completeness of the comparative benchmark.

3) *Memory Overhead:* The implementation of ASSC incurs a marginally higher memory overhead compared to the baseline. The overshoot detection mechanism requires keeping both the current and the temporarily over-coarsened graphs in memory simultaneously. While this overhead remains bounded by a linear space complexity of $O(|V| + |E|)$ and did not trigger an Out-Of-Memory failure on the evaluated dataset, it is inherently less memory efficient than strictly in-place coarsening methods.

C. Future Work

While the integration of ASSC solves the subgraph extraction bottleneck of NNP-NET, there are still several areas open for future research.

1) *Understanding Clustering’s Stalling Behaviour*: As noted in Chapter V-A, ASSC has an architectural vulnerability when processing specific graphs, forcing it to fall back on an over-coarsened subgraph. In our dataset of 25 graphs, this occurred only on the *web-Stanford* graph, which has a very high degree heterogeneity of about 3,500 relative to its size of about 280,000 nodes. This caused the SSC algorithm to overshoot the 10,000-node target on its very first iteration. Interestingly, the *web-BerkStan* graph has an even higher degree heterogeneity of about 5,600, but survived the first iteration, likely due to its larger size of about 690,000 nodes. Future research could systematically investigate the first-pass coarsening rate of SSC across different graphs to map out exactly where this ‘danger zone’ lies. Preliminary analysis of this dataset shows no direct correlation between degree heterogeneity and the first-pass coarsening rate. So future research would likely need to look at more statistics like degree assortativity and graph diameter, and the interplay between them. This way, it might be possible to define a mathematical threshold that accurately predicts whether a graph requires settling for an over-coarsened subgraph. In that case, a modified initial coarsening pass could be designed to deal with these very specific cases.

2) *Optimizing NNP-NET’s Other Components*: The authors of NNP-NET explicitly identified the subgraph extraction phase as the most important bottleneck to address [6]. With this issue resolved, future work can pivot to the other components of NNP-NET. Swapping out the PMDS embedding method for something else could give higher quality or faster results, and changing the ground truth method from tsNET* to an alternative could allow for other drawing styles.

VI. CONCLUSION

This thesis aimed to find an alternative subgraph extraction method for the NNP-NET pipeline that eliminates the severe runtime bottleneck caused by its fallback method, without compromising layout quality. Based on the empirical evaluation across a 25-graph dataset, ASSC definitively answers this requirement.

ASSC proved robust across all tested topologies without triggering major processing failures. In terms of computational speed, it effectively eliminates the stalling behaviour seen in complex networks, delivering an *average end-to-end pipeline speed-up of 9.2x on stalling topologies*. On standard topologies, it rivals the speed of NNP-NET’s baseline clustering method. Furthermore, ASSC maintains the structural integrity of the resulting layouts. For standard topologies, stress and neighbourhood preservation levels remain practically unchanged compared to the baseline, and a qualitative visual analysis of the final rendered graphs also back up the claim that output quality did not meaningfully improve or degrade. On stalling topologies, ASSC consistently matches or improves upon baseline neighbourhood preservation, and in the majority of cases, successfully reduces stress. Notably, in one case of a highly abstract topology (*web-Google*, shown in Table VII), ASSC resolves the graph into a distinctly structured layout where the baseline fails to do so.

The other three evaluated methods proved insufficient to fully replace the baseline pipeline. SSC achieves significant speed improvements but consistently over-coarsens the subgraphs. This gives the neural network less information to learn from, which in turn results in markedly worse graph drawings. In effect, this method fulfils the speed requirement at the unacceptable cost of output quality.

The pivot-points optimization inherently matches the high-quality outputs of the baseline fallback and offers moderate speed improvements on stalling topologies. However, ASSC is preferred over the pivot-points optimization because it yields superior speed-ups, actively improves layout quality on stalling topologies rather than just matching it, and unifies the subgraph extraction pipeline without introducing new hardware limitations. Crucially, the pivot-points optimization suffered Out-Of-Memory failures on the largest tested graphs, which is a regression from the baseline method.

Finally, edge contraction fails to be viewed favourably over the baseline primarily due to its negative impact on layout quality. While it provides moderate end-to-end speed improvements on stalling topologies, its performance on just the subgraph extraction phase is very low, averaging a 25x slowdown versus the baseline on standard topologies. More importantly, visual analysis on the final graph drawings show that edge contraction is prone to garbling, making it indeed an unsuitable replacement despite its acceptable overall end-to-end execution times.

REFERENCES

- [1] Y. Hu, “Efficient and high quality force-directed graph drawing,” vol. 10, pp. 37–71.
- [2] M. Gronemann, “Engineering the fast-multipole-multilevel method for multicore and SIMD architectures.”
- [3] J. F. Krüger, P. E. Rauber, R. M. Martins, A. Kerren, S. Kobourov, and A. C. Telea, “Graph layouts by t-SNE,” vol. 36, no. 3, pp. 283–294, _eprint: <https://onlinelibrary.wiley.com/doi/pdf/10.1111/cgf.13187>. [Online]. Available: <https://onlinelibrary.wiley.com/doi/abs/10.1111/cgf.13187>
- [4] 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,” vol. 27, no. 2, pp. 1666–1676.
- [5] U. Brandes and C. Pich, “Eigensolver methods for progressive multidimensional scaling of large data,” in *Graph Drawing*, M. Kaufmann and D. Wagner, Eds. Springer, pp. 42–53.
- [6] I. Hartskeerl, T. Mchedlidze, S. van Wagoningen, P. Vangorp, and A. Telea, “NNP-NET: Accelerating t-SNE graph drawing for very large graphs by neural networks: 33rd international symposium on graph drawing and network visualization,” pp. 22:1–22.
- [7] L. v. d. Maaten and G. Hinton, “Visualizing data using t-SNE,” vol. 9, no. 86, pp. 2579–2605. [Online]. Available: <http://jmlr.org/papers/v9/vandemaaten08a.html>
- [8] T. A. Davis and Y. Hu, “The university of florida sparse matrix collection,” vol. 38, no. 1, pp. 1:1–1:25. [Online]. Available: <https://doi.org/10.1145/2049662.2049663>
- [9] F. Grötschla, J. Mathys, R. Veres, and R. Wattenhofer, “CoRe-GD: A hierarchical framework for scalable graph visualization with GNNs.” [Online]. Available: <http://arxiv.org/abs/2402.06706>
- [10] E. R. Gansner, Y. Hu, and S. North, “A maxent-stress model for graph layout,” vol. 19, no. 6, pp. 927–940. [Online]. Available: <https://ieeexplore.ieee.org/document/6329372>
- [11] G. Bravo-Hermesdorff and L. M. Gunderson, “A unifying framework for spectrum-preserving graph sparsification and coarsening.” [Online]. Available: <http://arxiv.org/abs/1902.09702>
- [12] A. Loukas, “Graph reduction with spectral and cut guarantees.” [Online]. Available: <http://arxiv.org/abs/1808.10650>
- [13] M. E. J. Newman, “Assortative mixing in networks,” vol. 89, no. 20, p. 208701. [Online]. Available: <http://arxiv.org/abs/cond-mat/0205405>
- [14] M. Newman, *Networks: An Introduction*. Oxford University Press.
- [15] M. E. F., “The shortest path through a maze,” pp. 285–292, publisher: Harvard University Press. [Online]. Available: <https://cir.nii.ac.jp/crid/1570572700914359680>
- [16] E. Dijkstra, “A note on two problems in connexion with graphs : (numerische mathematik, 1(1959), p 269-271).” [Online]. Available: <https://ir.cwi.nl/pub/9256>
- [17] B. Hendrickson and R. Leland, “A multilevel algorithm for partitioning graphs,” in *Proceedings of the 1995 ACM/IEEE conference on Supercomputing (CDROM) - Supercomputing '95*. ACM Press, pp. 28–es. [Online]. Available: <http://portal.acm.org/citation.cfm?doid=224170.224228>
- [18] C. Walshaw, “A multilevel algorithm for force-directed graph drawing.”
- [19] S. Hachul, “A potential-field-based multilevel algorithm for drawing large graphs.” [Online]. Available: <http://kups.ub.uni-koeln.de/1409/>
- [20] S. Hachul and M. Jünger, “An experimental comparison of fast algorithms for drawing general large graphs,” in *Graph Drawing*, P. Healy and N. S. Nikolov, Eds. Springer, pp. 235–250.
- [21] D. Harel and Y. Koren, “A fast multi-scale method for drawing large graphs,” in *Graph Drawing*, J. Marks, Ed. Springer Berlin Heidelberg, pp. 183–196.
- [22] A. Loukas and P. Vandergheynst, “Spectrally approximating large graphs with smaller graphs,” vol. abs/1802.07510. [Online]. Available: <http://arxiv.org/abs/1802.07510>
- [23] Y. Jin, A. Loukas, and J. F. JaJa, “Graph coarsening with preserved spectral properties,” _eprint: 1802.04447. [Online]. Available: <https://arxiv.org/abs/1802.04447>
- [24] P. Gajer and S. G. Kobourov, “GRIP: Graph dRawing with intelligent placement,” in *Graph Drawing*, J. Marks, Ed. Springer Berlin Heidelberg, pp. 222–228.
- [25] D. Archambault, T. Munzner, and D. Auber, “TopoLayout: Multilevel graph layout by topological features,” vol. 13, no. 2, pp. 305–317.
- [26] C. Cai, D. Wang, and Y. Wang, “Graph coarsening with neural networks,” vol. abs/2102.01350. [Online]. Available: <https://arxiv.org/abs/2102.01350>
- [27] DrTimothyAldenDavis. SuiteSparse/CHOLMOD/partition/cholmod_nesdis.c at dev · DrTimothyAldenDavis/SuiteSparse. [Online]. Available: https://github.com/DrTimothyAldenDavis/SuiteSparse/blob/dev/CHOLMOD/Partition/cholmod_nesdis.c
- [28] ogdf. ogdf/src/ogdf/energybased/multilevel_mixer/SolarMerger.cpp at master · ogdf/ogdf. [Online]. Available: https://github.com/ogdf/ogdf/blob/master/src/ogdf/energybased/multilevel_mixer/SolarMerger.cpp

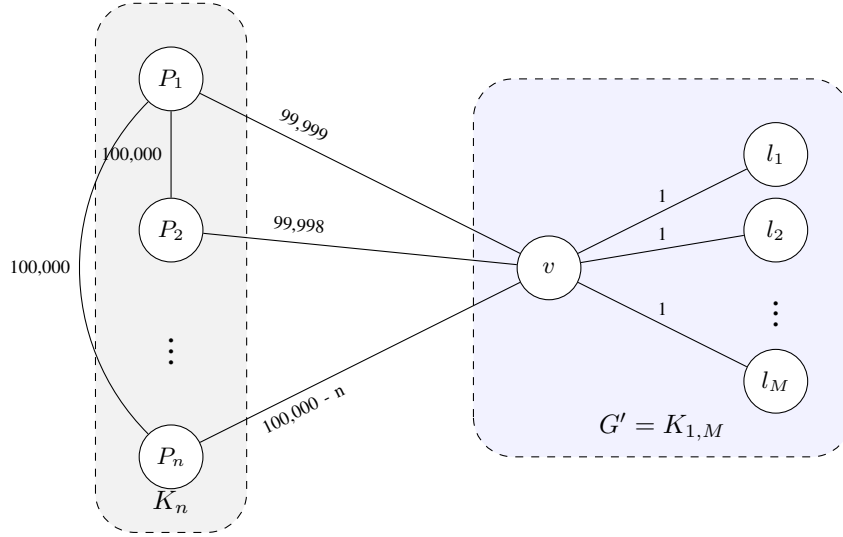


Fig. 15. An example of a sparse graph with weighted edges that forces the optimized pivot-points algorithm to update the distances of all nodes after every iteration, if the array of nodes starts as $[P_1, P_2, P_3, \dots]$.

APPENDIX

The time complexity of the original implementation of the pivot-points based graph coarsening method in NNP-NET is $O(nN)$ [6, Section 3.4], and below we will show, through two examples (also shown in Figures 15 and 16), that the optimization proposed in Chapter III-B does not improve on this theoretical time complexity. (But, as discussed in Chapter IV, does improve performance in practice.)

A. Weighted Graph Example

Let G be a graph such that it contains $K_{10,000}$, the complete graph on 10,000 nodes. We shall name these nodes $P_1, P_2, \dots, P_{10,000}$. Importantly, these will be the first nodes in the array of all nodes of G . Let the edge weight between P_i and P_j be 100,000 for all $1 \leq i < j \leq 10,000$. Now let there be, for some $M \in \mathbb{N}$, a group of $M \gg 10,000$ more nodes (say $G' \subset G$) that amongst themselves form the complete bipartite graph $K_{1,M}$ on M nodes. Let v be the central node of G' that is connected to all other nodes of G' . Let the edge weights of *all* edges in G' be 1. These nodes shall be connected to the P_i as follows: for any i such that $1 \leq i \leq 10,000$, a direct edge exists from P_i to v with edge weight $100,000 - i$. No other edges between P_i and G' exist. Note then that the distance from P_1 to v is 99,999, and the distance from $P_{10,000}$ to v is 90,000. See also Figure 15.

If we let the optimized version of the pivot-points based graph coarsening algorithm (Algorithm 7) run on G , the following happens:

- 1) P_1 is assigned as our first pivot-point. The distances from P_1 to all other nodes in G are calculated using Dijkstra's algorithm.
- 2) Note that the distance from P_1 to any node in G' is either 99,999 or 100,000, and that the distance from P_1 to any other P_i is 100,000. Thus, P_2 is selected as the second pivot-point.
- 3) As the distance from P_2 to G' is exactly 1 less than from P_1 to G' , the distances for all nodes in G' must be updated. However, the furthest distance from the set of pivot-points $\{P_1, P_2\}$ to any other node is again 100,000, so P_3 is selected as the third pivot-point.
- 4) As the distance from P_3 to G' is less again than from either P_1 or P_2 to G' , the distances for all nodes in G' must be updated again. But again the furthest distance from the set of pivot-points to any other node is 100,000, so P_4 is selected as the fourth pivot-point.
- 5) This cycle repeats until all P_i are selected as pivot-points. Hence, the computational complexity of this algorithm on this graph is $O(nN)$, the same as for the original pivot-points implementation of NNP-NET.

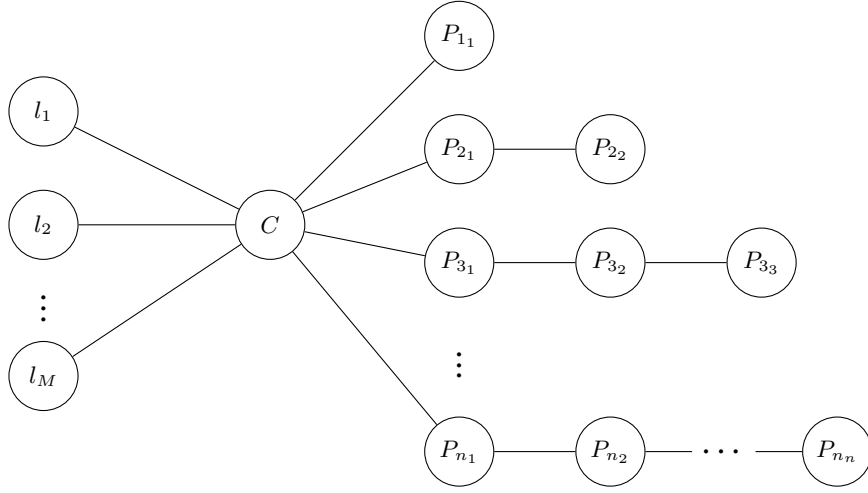


Fig. 16. An example of a sparse unweighted graph that forces the optimized pivot-points algorithm to update the distances of almost all nodes after every iteration, if P_{n_1} is the first node in the array of nodes.

B. Unweighted Graph Example

Let G be an unweighted graph with a central node C . For $M \gg 10,000$ and all i such that $1 \leq i \leq M$, let l_i be a node that is *only* connected to C (the ‘leaves’ of C). Next, we will construct 10,000 paths extending from C . Let $1 \leq i \leq 10,000$, then G contains a path, starting at C , of length i . So (CP_{1_1}) is our first path, $(CP_{2_1}P_{2_2})$ our second, $(CP_{3_1}P_{3_2}P_{3_3})$ our third, up to $(CP_{10,000_1}P_{10,000_2}\dots P_{10,000_{10,000}})$. Importantly, the node $P_{10,000_{10,000}}$ is the first node in the array of all nodes of G . See also Figure 16.

If we let the optimized version of the pivot-points based graph coarsening algorithm (Algorithm 7) run on G , the following happens:

- 1) $P_{10,000_{10,000}}$ is assigned as our first pivot-point. The distances from this node to all other nodes in G are calculated using breadth-first search.
- 2) The furthest node from $P_{10,000_{10,000}}$ is $P_{9,999_{9,999}}$, so it is selected as our second pivot-point. Since the distance from this node to the l_i is one less than from $P_{10,000_{10,000}}$, all these distances must be updated.
- 3) The furthest node from our two pivot-points is $P_{9,998_{9,998}}$, so it is selected as our third pivot-point. Again the distance from our set of pivot-points to all l_i has decreased by one, so all these distances must be updated again.
- 4) This cycle repeats until all P_{i_i} have been selected as pivot-points. Hence, the computational complexity of this algorithm on this graph is $O(nN)$, the same as for the original pivot-points implementation of NNP-NET.

It is worth noting that this example requires at least $\sum_{i=1}^{10,000} i = 50,005,000$ nodes. It is reasonable to assume that with fewer nodes, the time complexity drops, which is supported by the fact that in practice (as discussed in Chapter IV-A2) the subgraph extraction of the optimized version is on average 30x quicker than the unoptimized version.