Karlsruher Institut für Technologie

Bachelor thesis

Evaluating Embedding-Based Coarsening for Multilevel Graph Partitioning

Matthias Hilgers

Date: 1. Dezember 2024

Supervisors: Prof. Dr. Peter Sanders
Dr. Lars Gottesbüren
M.Sc. Nikolai Maas
M.Sc. Daniel Seemaier

Institute of Theoretical Informatics, Algorithm Engineering
Department of Informatics
Karlsruhe Institute of Technology

Abstract

Multilevel-Graphpartitionierer sind ein Standardwerkzeug für die Graphpartitionierung. Gleichzeitig finden Graph-Embeddings Anwendung in der Graphenanalyse, um niedrigdimensionale Repräsentationen von Graphen zu erhalten, die die Struktur der Graphen erhalten. Die Lösungsqualität von Multilevel-Graphpartitionierern ist direkt abhängig von der Fähigkeit der verwendeten Coarsening-Methode die Struktur des Eingabe-Graphen zu erhalten. Diese Arbeit untersucht die Anwendung von Embeddings um Multilevel-Graphpartitionierern um in der Coarsening-Phase mehr strukturelle Informationen über den Eingabe-Graphen zur Verfügung zu stellen. Dazu werden verschiedene Coarsening-Strategien mit zwei verschiedenen Embedding-Verfahren in KaMinPar, einem Multilevel-Graphpartitionierer, implementiert und verglichen.

Multilevel graph partitioners are a standard tool in graph partitioning. Simultaneously, graph embeddings find use in graph analysis for their ability to create low-dimensional graph representations that preserve the structure of the input graph. The solution quality of multilevel graph partitioners is greatly dependent on the ability of the used coarsening method to maintain the structure of the input graph throughout all layers of the graph hierarchy. This work explores the use of graph embeddings in multilevel graph partitioners to make global structural information on the input graph available in the coarsening phase. To this end, this work implements different coarsening strategies in KaMinPar, a multilevel graph partitioner and compares them with two embedding methods, Node2Vec and InstantEmbedding.

Acknowledgements

The author acknowledges support by the state of Baden-Württemberg through bwHPC.

Hiermit versichere ich, dass ich diese Arbeit selbständig verfasst und keine anderen, als die angegebenen Quellen und Hilfsmittel benutzt, die wörtlich oder inhaltlich übernommenen Stellen als solche kenntlich gemacht und die Satzung des Karlsruher Instituts für Technologie zur Sicherung guter wissenschaftlicher Praxis in der jeweils gültigen Fassung beachtet habe.

Karlsruhe, den 1. Dezember 2024

Contents

1 Introduction

1.1 Motivation

Graph partitioning is a problem that is used in different domains, such as circuit design[20] and power grid analysis[27] and is an essential preprocessing step for parallel graph analysis techniques, such as parallel pagerank[41]. However, graph partitioning is NP-hard to solve and NP-hard to approximate[4], so that graph partitioners are usually based on heuristics.

A commonly used category of graph partitioners are multi-level graph partitioners. Multi-level graph partitioners work in three phases, the first being the coarsening phase, where the input graph is iteratively shrunken down in a way that – one hopes – preserves the structure of the input graph. In the remaining two phases, a multi-level graph partitioner first finds an initial partition for the coarsest graph acquired from the coarsening phase and then projects that partition to the finer graphs of the coarsening hierarchy, while further refining the partition.

In this process, graph coarsening plays a critical role. Initial partition quality is highly dependent on the coarsener's ability to preserve the graph's structure. Additionally, while low quality initial partitions may be improved in the refinement phase, this process can not replace the need for high quality graph coarsening.

Graph embeddings represent each node in a graph as a low-dimensional vector which summarises each node's role in the graph's global structure. This allows to maintain a view of a graph's global structure that is both space-efficient and easy to interface with. Graph coarseners can be extended to make use of this structural information. Rather than deciding if two nodes should be contracted in the coarse graph based only on their immediate neighborhoods, considering the similarity of the embedding vectors of these two nodes introduces a notion of the global structure of the graph into the decision.

1.2 Contribution

This thesis extends the work of Srybrandt et al.[43] by verifying their results with an implementation of embedding-based graph coarsening in a different graph partitioner, KaMinPar[15, 2] and using a different coarsening scheme based on size constrained Label Propagation, rather than Heavy Edge Matchings. Different strategies are evaluated, including different edge rating functions and cluster overlaying (ref. Section 4.3).

Additionally, statistical tests are done on the effectiveness of embedding-based graph coarsening by testing the correlation of embeddings and the partitions that were found based on these embeddings.

All coarsening methods are evaluated using two embedding methods, InstantEmbedding[35] and Node2Vec[16]. In the past, Safro et al.[40] and Chen et al.[9] have used algebraic distances as another means of making graph structural information available in the coarsening phase on a wider scale. That makes algebraic distances similar to embeddings for the scope of this work and as such, additional experiments are done which compare the use of embeddings with the use of algebraic distances.

The achieved solution quality is found to vary among the compared embedding methods. While Node2Vec-based coarsening is found to improve solution quality over the baseline and algebraic distance-based coarsening, InstantEmbedding-based coarsening has worse solution quality than the baseline and algebraic distance-based coarsening. This effect is found to be alleviated by cluster overlaying, which boosts the solution quality of both embedding-based methods to be higher than the baseline or using cluster overlaying without embeddings.

1.3 Structure

Chapter 2, "Fundamentals", gives an overview of the graph partitioning problem and graph embeddings, while also presenting general definitions used throughout this work. Chapter 3 presents an overview of multilevel graph partitioning and assorted graph coarsening strategies and refinement strategies, as well as an introduction to and comparison of different graph embedding methods. In chapter 4, a run-down of the evaluated algorithms prepares the reader for chapter 5, which details the implementation of the previously introduced algorithms and describes the experiments done to compare them. Finally, chapter 6 contains a conclusion with options for future work.

2 Fundamentals

2.1 General Definitions

For the rest of this work, let $G = (V, E, c, \omega)$ be an undirected graph with node weights $c : V \to \mathbb{N}$ and edge weights $\omega : E \to \mathbb{N}$. To extend c and w to sets of nodes $\tilde{V}$, respective edges $\tilde{E}$, let $c(\tilde{V}) = \sum_{v \in \tilde{V}} c(v)$ and $w(\tilde{E}) = \sum_{e \in \tilde{E}} w(e)$.

2.2 Graph Partitioning

The Graph Partitioning problem is the problem of splitting a graph $G = (V, E)$ into k disjoint blocks V_1, V_2, V_k, such that $\bigcup_{i=1}^{k} V_i = V$. The weight of all blocks have to fulfil a *balance constraint*, i.e. $\forall i \in \{1..k\} : c(V_i) \leq L_{\max} := (1 + \varepsilon)\frac{c(V)}{k} + \max_c c(v)$, where ε is an imbalance parameter.

An optimal partition has a minimal cut $\omega(E_c)$ where $E_c = \{\{u, v\} | u \in V_i, v \in V_j, i < j\} \subseteq E$.

2.3 Graph Embedding

The Graph Embedding problem is the problem of embedding a graph $G = (V, E)$ into a d-dimensional vector space with $d < n$, such that every node v can be represented by an embedding vector $\epsilon(v) \in \mathbb{R}^d$. A good node embedding of a node $u \in V$ captures the role of u in the global structure of the graph. Through this property, graph embeddings can capture the global structure of a graph and make it available to downstream tasks, such as node classification[25] and graph visualisation[48].

A d-dimensional representation of the whole graph G is given by the embedding matrix $W \in \mathbb{R}^{n \times d}$, such that its rows $W[v, 1 : d]$ are the embedding vectors $\epsilon(v)$ for all nodes v.

3 Related Work

3.1 Multi-Level Graph Partitioning

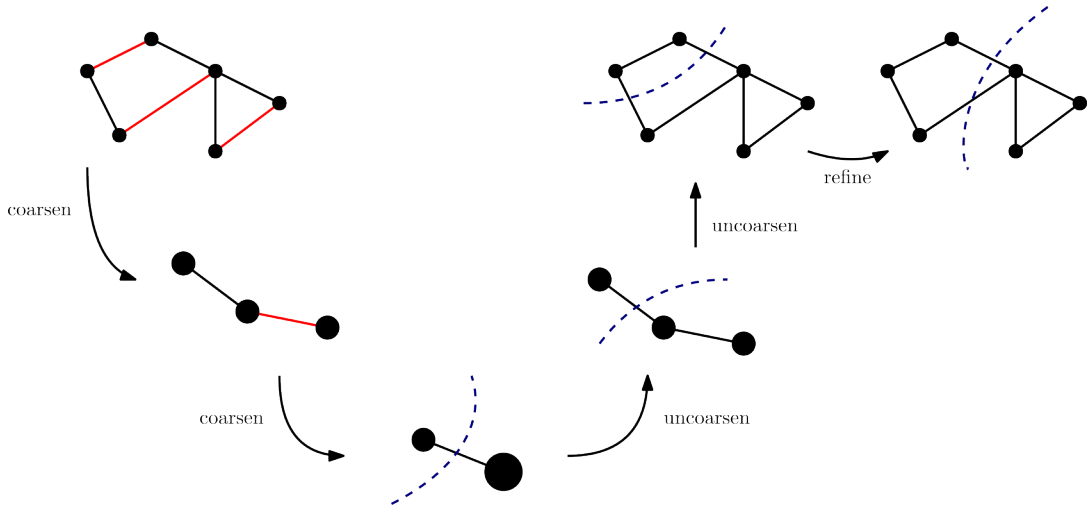


Figure 3.1: Multi-level graph partitioning on an example graph.

Multi-level graph partitioners are a group of graph partitioning methods that work by iteratively shrinking a graph before finding a partition and then transforming the partition to a partition of the original input graph. Multi-level graph partitioners generally work in three phases, which are illustrated in Figure 3.1.

In the first phase, the input graph is repeatedly coarsened, until only kC nodes remain, where C is a tuning parameter. Coarsening is also stopped if it converges, i.e. if successive iterations of the coarsening method fail to reduce the size of the graph by an implementation-dependent threshold.

In the second phase, an initial partition for the coarsest graph is computed. The quality of this partition greatly depends on how well the structure of the input graph can be preserved by the graph coarsening method.

Finally, in the last phase, the partition is projected to the finer graphs. At the same time, the partition gets refined on each level to improve the cut and balance of the partition. As the initial partition is found on the coarsest graph, refinement is necessary to optimise partitions in regard to the additional information available on finer graphs.

Deep Multi-Level Graph Partitioning In deep multi-level graph partitioning, the coarsest graph is further coarsened until it has only $2C$ nodes left. At the same time, each new coarse graph is recursively duplicated and coarsened.

These coarsest graphs are then bipartitioned. During refinement, the best partition is chosen and recursively bipartitioned until a $k-$partition has been found. At this point refinement is continued without further bipartitioning until all coarsening has been undone.

Deep multi-level graph partitioning leads to faster runtimes and better partitioning quality when k is large[15].

3.2 Overview of Graph Coarsening Methods

Graph coarsening is the process of creating a graph G^C from a graph G, such that G^C has less nodes than G, while trying to maintain the structure of G. This section describes two coarsening methods, size constrained label propagation and matching-based methods, in greater detail.

3.2.1 Size Constrained Label Propagation

Size constrained Label Propagation is an extension by Meyerhenke et al.[30] to the *Label Propagation Clustering Algorithm* (LPA) by Raghavan et al.[36].

LPA iteratively improves an initial clustering. In the context of graph coarsening, for the initial clustering each node is assigned to its own cluster. In each iteration, nodes u are then moved to the cluster V_i they have the strongest connection to, i.e. to the cluster V_i that maximises $\omega(\{\{u, v\} \in E \mid v \in V_i\})$. After finishing a certain number of rounds or if no more moves can be made, nodes $u_1, ..., u_i$ that are part of the same cluster get contracted into a single node.

Size constrained LPA introduces a size constraint U that prevents moving a node u to a cluster V_i if $c(V_i) + c(u) > U$ (cf. Line 6 in Algorithm 1). When used in coarsening, U is set to $U := \max(\max_v c(v), \frac{L_{\max}}{f})$, where f is a tuning parameter.

Meyerhenke et al. further note that size constrained LPA is especially suited for irregular graphs and runs in near linear time, as each iteration requires $O(n + m)$ time[30].

3.2.2 Matching-Based Coarsening

Matching-based coarsening methods coarsen a graph by constructing a maximal matching $M \subseteq E$ and then contract that matching by contracting pairs of nodes u, v that are connected by an edge $e \subseteq M$ with a coarse node u^C.

Algorithm 1: Size Constrained Label Propagation

```
Input: Graph G = (V, E), Size Constraint U, Initial Clustering C, Rounds r
Output: Clustering C
1 for i ← 1 to r do
2     foreach u ∈ V do
3         B ← adjacentClustersToNode(u)
4         B̃ ⊆ B all clusters b, where c(b) + c(u) ≤ U
5         b ← arg max_{b∈B̃} ω({{u, v} ∈ E | v ∈ b})
6         if c(b) + c(u) ≤ U then
7             C[u] ← b
8 return C
```

A variety of matching algorithms have been used for graph coarsening. These include but aren't limited to Heavy Edge Matching[22], the Global Path Algorithm[29] and Sorted Heavy Edge Matching[22].

Srybrandt et al.[43] propose an embedding-based coarsening algorithm based on Sorted Heavy Edge Matching. They initially score all nodes u based on the dot products of their embedding with any adjacent v node's embedding (cf. Lines 2-5 of Algorithm 2). Afterwards, they iterate over all nodes in the order of their previously achieved scores. Lines 9-11 of Algorithm 2 show how a node u that is not yet part of the matching will be included through the edge $\{u, v\}$ that maximises a combination of its edge weight $\omega(\{u, v\})$ and the dot product $\epsilon(u)^T \epsilon(v)$ of the embeddings of nodes u, v.

3.3 Partition Refinement

The goal of partition refinement is to improve an existing partition in regards to balance and cut. This section describes three refinement algorithms which are also used in the experiments done for this thesis.

3.3.1 Size Constrained Label Propagation

Size constrained Label Propagation (see Section 3.2.1) can also be used for partition refinement, by using the size constraint $U := \max(\max_v c(v), L_{\max})$. All nodes of one block are then initially assigned to the same cluster.

The algorithm then iterates over all nodes, moving a node u to the cluster that most of its neighbours are in, as long as this does not violate the size constraint. Additionally, if a node u is part of an overloaded cluster V_l, it is moved to the block V_k that has the strongest connection to u and where moving u to V_k does not violate the balance constraint[30].

Algorithm 2: Embedding-based Heavy Edge Matching

```
Input: Graph G = (V, E)
Output: Matching M
M = (V_M, E_M) an empty Matching
V_score an empty node scoring
foreach u ∈ V do
    N ← {v | {u, v} ∈ E}
    V_score[u] ← max_{v∈N} ε^T(u)ε(v) / c(u)c(v)
foreach u ∈ V sorted by decreasing score in V_score do
    if u ∈ V_M then
        continue
    N ← {v | {u, v} ∈ E}
    v ← max_{v∈N\V_M} ε^T(u)ε(v) / c(u)c(v) ω({u, v})
    add {u, v} to M
return M
```

3.3.2 Jet

Jet[13] is a refinement algorithm primarily developed to run on GPUs, although CPU implementations are available (e.g. in KaMinPar[2]).

Jet consists of two parts, which its authors call Jetlp and Jetr, respectively. Jetlp is a synchronous implementation of unconstrained label propagation, while Jetr rebalances a partition. Jetlp additionally locks moved nodes, so that a node u may not be moved in the next iteration, which prevents nodes from oscillating between blocks[13].

Jet works by interleaving runs of Jetlp and Jetr, such that Jetr is run whenever the partition becomes unbalanced.

3.3.3 Fiduccia-Mattheyses Local Search

Fiduccia and Mattheyses propose a local search algorithm[11] for partition refinement which is commonly called the *FM* algorithm. In its original version, the FM algorithm is a sequential refinement algorithm for bipartitions that works in rounds. More modern variants of the FM algorithm can improve k-way partitions and can be parallelised.

A round consists of repeatedly moving the node u that has the highest gain, i.e. u is moved to the other block if out of all nodes that can be moved, moving u would decrease the cut the most. Nodes may only be moved once per round and only if moving would not violate the balance constraint.

Previous implementations of the FM algorithm would consider all boundary nodes for mov-

ing, i.e. all nodes adjacent to another node from a different block, at once. Sanders and Schulz only consider a single boundary node at a time, repeatedly initialising their FM search with a single boundary node[42]. Moving a node u will insert all neighbours of u into a queue of nodes that will be considered for further moves.

Gottesbüren et al. parallelise this approach by starting multiple FM searches at the same time and then joining all obtained move sequences into a single global move sequence. They finally apply the prefix of the move sequence that leads to the highest gain without violating the balance constraint[14]. [14].

3.4 Overview Of Different Graph Embedding Methods

This section will provide an overview of different graph embedding methods, as well as a comparison of the presented graph embedding methods.

3.4.1 Graph Embedding Based On Matrix Dimension Reduction

Graph Embedding methods based on matrix dimension reduction work in two steps. First a similarity matrix $S \in \mathbb{R}^{n \times n}$ of G is computed. S can for example be the squared adjacency matrix A^2[32] or the Personalised Page Rank (PPR) matrix of a graph[35]. The second step is to compute the embedding matrix $W \in \mathbb{R}^{n \times d}$ of G by using dimensionality reduction methods on S.

InstantEmbedding [35] Postăvaru et al. propose InstantEmbedding for computing embeddings of single nodes that can be computed in time independent of the total size of G but are nonetheless globally consistent, i.e., single node embeddings can be combined to produce a consistent graph embedding.

To find a node embedding $\epsilon(v)$ for a node v, InstantEmbedding first computes the Personalised Pagerank (PPR) vector $\mathrm{ppr}(v)$ of v and then uses *feature hashing* to reduce its dimension. Personalised Pagerank is a variant of the original PageRank algorithm proposed by Page and Lawrence[33]. Instead of measuring the importance of every node at once, the PPR vector shows the importance of all nodes in a graph, if a set of nodes is already known to be important.

Feature hashing is a dimensionality reduction technique commonly attributed to Weinberger et al.[49]. Two hash functions $h_{\mathrm{sgn}} : \mathbb{N} \to \{\pm 1\}$ and $h_{\mathrm{d}} : \mathbb{N} \to \{1, ..., d\}$ are sampled from two universal hash families $\mathbb{U}_{\{\pm 1\}}$, $\mathbb{U}_{\{1,...,d\}}$, respectively, are applied to a vector or matrix to reduce its dimensionality. In the case of InstantEmbedding, h_{sgn} and h_{d}

are applied to the logarithm of all PPR vectors $\log \mathrm{ppr}(v)$ by computing

$$\epsilon_i(v) = \sum_{k \in \{k | h_{\mathrm{d}}(k) = i\}} h_{\mathrm{sgn}}(k) * \max\{\log(\mathrm{ppr}_k(v) * n), 0\}.$$

By executing InstantEmbedding for every node of G, an embedding for the full graph can be obtained.

HOPE [32] Ou et al. identify the asymmetric transitivity as a critical property of directed graphs[32] and note a lack of graph embedding methods that preserve asymmetric transitivity. Asymmetric transitivity is the property of a graph that if there is a directed path from a node u to a node v then there is likely a directed edge from u to v.

Their solution is to compute two embedding vectors for each node v, the source vector $\epsilon_{\mathrm{source}}(v)$ and the target vector $\epsilon_{\mathrm{target}}(v)$. If there is an edge $e = (u, v)$ but no edge $\tilde{e} = (v, u)$ then $\epsilon_{\mathrm{source}}(u)$ and $\epsilon_{\mathrm{target}}(v)$ shall have similar values while the values of $\epsilon_{\mathrm{source}}(v)$ and $\epsilon_{\mathrm{target}}(u)$ shall be dissimilar.

Ou et al. find decompositions of different similarity matrices $S = M_g^{-1} M_l$[32]. Their algorithm then applies generalised SVD to M_g^{-1} and M_l to compute the d largest singular values σ_i of S, as well as their corresponding singular vectors $s_{i,\mathrm{source}}$ and $s_{i,\mathrm{target}}$, for $i \in \{1, ..., d\}$. The source and target embedding matrices are then given by $W_{\mathrm{source}} = [\sqrt{\sigma_1} s_{1,\mathrm{source}}, ..., \sqrt{\sigma_d} s_{d,\mathrm{source}}]$ and $W_{\mathrm{target}} = [\sqrt{\sigma_1} s_{1,\mathrm{target}}, ..., \sqrt{\sigma_d} s_{d,\mathrm{target}}]$, respectively.

FastRP [8] FastRP works by reducing the dimensionality of a similarity matrix using very sparse random projection. The idea of random projection is to reduce the dimensionality of a similarity matrix S by multiplying it with a random projection matrix R with dimension $n \times d$. Li et al.[28] recommend the following definition of R:

$$R_{ij} = \begin{cases} \sqrt[4]{m} & \text{with probability } \frac{1}{2\sqrt{m}} \\ 0 & \text{with probability } 1 - \frac{1}{\sqrt{m}} \\ -\sqrt[4]{m} & \text{with probability} \frac{1}{2\sqrt{m}} \end{cases},$$

which Chen et al.[8] call *very sparse random projection*.

As a similarity matrix, FastRP uses a normalised variant of T^k where T is the transition matrix $T = D^{-1}A$ with A the adjacency matrix of G and D the degree matrix of G with

$$D_{ij} = \begin{cases} \deg(v) & \text{if } i = j \wedge v \text{ is the } i\text{th node of } G \\ 0 \end{cases}.$$

3.4.2 Neural Graph Embedding

Neural graph embedding methods apply methods for word representation learning from natural language processing to learn graph embeddings[47, 46].

DeepWalk [34] DeepWalk interprets nodes as words and truncated random walks as sentences. It then applies SkipGram[31] to these random walks to learn a graph embedding. In language modelling, SkipGram learns which words appear in the same context as a given word. In the context of DeepWalk, SkipGram learns W by observing context windows of size w around all nodes v of the generated random walks.

Since DeepWalk learns the similarity of two nodes v, w by comparing the similarity of random walks that v, w are included in, DeepWalk only learns the second-order proximity of a node [44].

Node2Vec [16] Node2Vec uses SkipGram to learn an embedding from a set of biased random walks.

The sampling patterns of these random walks are controlled by two parameters p and q, which allows different notions of node similarity [16]. The return parameter p controls the likelihood of immediately returning to the previous node, whereas the in-out parameter q influences the random walk's bias towards visiting nodes that are not the previous node.

A random walk that just traversed an edge (v, w) and is now at node w will traverse to the adjacent node x which has the highest transition probability π_{wx}. The unnormalised transition probability is given by $\pi_{wx} = \alpha_{pq}(v, x) \cdot w_{wx}$, where w_{wx} is the weight of the edge (w, x),

$$\alpha_{pq}(v,x) = \begin{cases} \frac{1}{p} & \text{if } d_{vx} = 0 \\ 1 & \text{if } d_{vx} = 1 \\ \frac{1}{q} & \text{if } d_{vx} = 2 \end{cases}$$

and d_{vx} is the length of the shortest path from v to x.

LINE [44] Tang et al.[44] propose LINE, a graph embedding method for directed, as well as undirected and weighted, as well as unweighted networks. LINE is built around two optimisation functions that are designed to preserve first-order and second-order proximity, respectively. These optimisation functions are optimised using Stochastic Gradient Descent (SGD).

Unlike DeepWalk, LINE samples nodes using Breadth-First Sampling.

Force2vec [38] Force2vec aims to use force-directed graph layout models (e.g. [12]) from the field of graph drawing to find a graph embedding.

Graph drawing is the problem of embedding a graph in two or three dimensions in such a way that it is "aesthetically pleasing"[38].

Force-directed graph layout models generally work by simulating a system of attractive and repulsive forces. Two nodes v, w will be attracted to each other when there is an edge between v and w, otherwise, they will repulse each other.

Force2vec then learns W by minimising a loss function that is based on these attracting and repelling forces.

VERSE [46] VERSE introduces an optimisation objective that minimises the Kullback-Leibler divergence between the node similarity sim_G in G and the node similarity sim_ϵ in the embedding space using SGD. VERSE allows different choices for sim_G but uses PPR as its default choice[46].

3.4.3 Others

Algebraic Distance [39] The algebraic distance is a measure for the connectivity of two vertices[9]. While not a graph embedding method in a strict sense, algebraic distance-based coarsening can be viewed as implicitly computing a graph embedding W with dimension $d = n$ and using the information from W to adjust the weight of all edges (cf. [9]).

3.4.4 Comparison of Static Graph Embedding Methods

Rahman and Azad[37] compare HOPE, DeepWalk, LINE, VERSE and force2vec. Postăvaru et al.[35] compare DeepWalk, Node2Vec, VERSE, FastRP and InstantEmbedding for generating embeddings of single nodes.

This comparison will refer to several datasets that were used in experiments by Postăvaru et al.[35] and Rahman and Azad[37]. An overview of these datasets is given in Table 3.1.

Table 3.1: Dataset characteristics: number of nodes $|V|$; number of edges $|E|$; number of node labels $|L|$; average node degree; density $\frac{|E|}{\binom{|V|}{2}}$

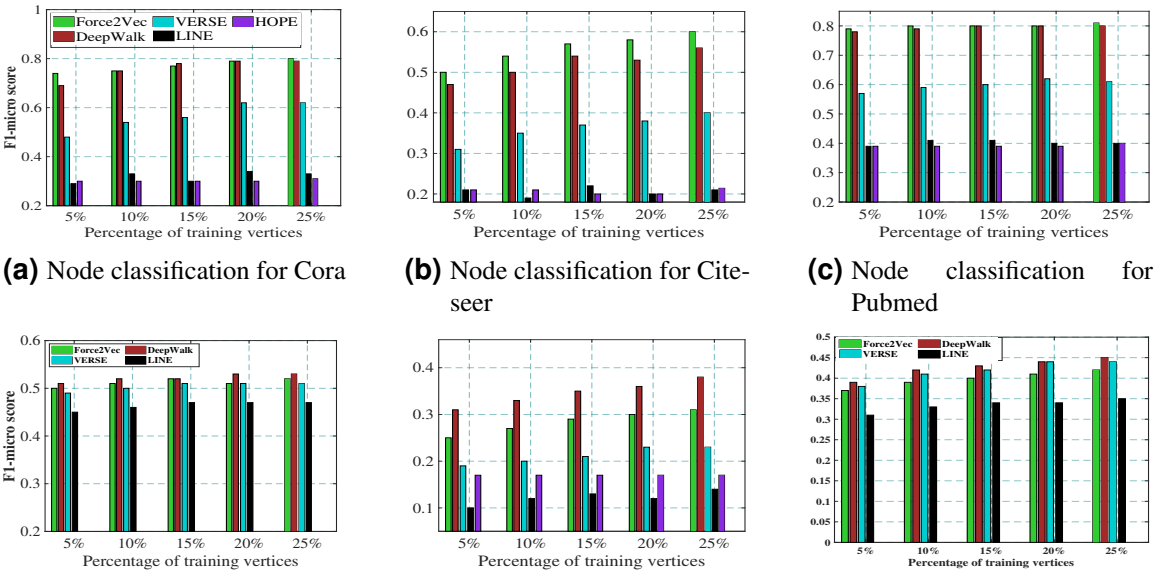
Dataset	$\|V\|$	$\|E\|$	$\|L\|$	Avg. deg.	Density
Cora	2k	5k	7	5	2.5×10^{-3}
Citeseer	3k	4k	6	2.67	8.89×10^{-4}
BlogCatalog	10k	334k	39	64.8	6.3×10^{-3}
Pubmed	20k	44k	3	4.4	2.2×10^{-4}
CoAuthor	52k	178k	–	6.94	1.3×10^{-4}
Flickr	80k	12M	195	146.55	1.8×10^{-3}
YouTube	1.1M	3M	47	5.25	9.2×10^{-6}
Orkut	3M	117M	110k	7.8	2.6×10^{-5}

Embedding Quality Based On Node Classification Postăvaru et al.[35] and Rahman and Azad[37] evaluate the embedding quality of InstantEmbedding, DeepWalk, HOPE, VERSE, LINE, Node2Vec, force2vec and FastRP by using these methods for node classification.

For their node classification experiments, Postăvaru et al. train a simple classifier on 10% of the available labels of BlogCatalog and on 1% of the available labels of CoCit, Flickr, YouTube and Orkut. The results that Postăvaru et al. report are available in Table 3.2. Postăvaru et al. note that InstantEmbedding achieves a higher Micro-F1 score than DeepWalk, Node2Vec, VERSE and FastRP on all their datasets, except for Flickr.

Table 3.2: Average Micro-F1 scores and confidence intervals as reported by Postăvaru et al. for their node classification experiments[35].

	BlogCatalog	CoCit	Flickr	YouTube	Orkut
DeepWalk	32.48 ±*0.35*	37.44 ±*0.67*	31.22 ±*0.38*	38.69 ±*1.17*	87.67 ±*0.23*
Node2Vec	33.67 ±*0.93*	38.35 ±*1.75*	29.8 ±*0.67*	36.02 ±*2.01*	-
VERSE	24.64 ±*0.85*	38.22 ±*1.34*	25.22 ±*0.2*	36.74 ±*1.05*	81.52 ±*1.11*
FastRP	33.54 ±*0.96*	26.03 ±*2.1*	29.85 ±*0.26*	22.83 ±*0.41*	-
InstantEmbedding	33.36 ±*0.67*	39.95 ±*0.67*	30.43 ±*0.79*	40.04 ±*0.97*	76.83 ±*1.16*



(a) Node classification for Cora

(b) Node classification for Citeseer

(c) Node classification for Pubmed

(d) Node classification for Flickr

(e) Node classification for BlogCatalog

(f) Node classification for YouTube

Figure 3.2: Micro-F1 scores for node classification as reported by Rahman and Azad[37].

Rahman and Azad train a logistic regression model for their node classification experiments

on different percentages of their datasets[37]. Their results are reported in Figure 3.2 where the percentage of vertices used for training is given on the x-axis. Rahman and Azad state they were unable to run HOPE on larger graphs, such as Flickr and YouTube, due to HOPE's high memory requirements[37]. While no single method achieves a higher F1-micro score for all datasets and test-training splits than all other methods, Figure 3.2 shows that the three highest F1-micro scores are always achieved by DeepWalk, force2vec and VERSE.

Embedding Quality Based On Link Prediction Postăvaru et al. and Rahman and Azad use DeepWalk, Node2Vec, LINE, HOPE, force2vec, VERSE and InstantEmbedding for link prediction to further quantify the quality of the generated embeddings[35, 47, 37].

Link prediction is the task of predicting edges that might be present in future evolutions of a network. The applications for link prediction include the recommendation of user accounts on platforms such as Facebook[19] and the prediction of drug-disease association in biomedical networks[50].

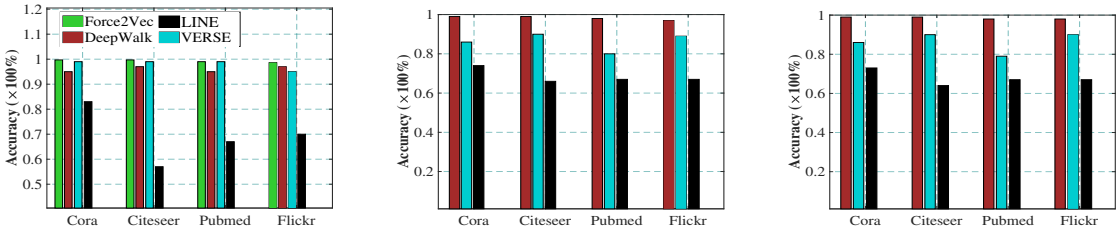
For their link prediction experiments on the CoAuthor dataset, Postăvaru et al. use links until 2014 for training. For testing and validation, they split links from 2015-2016 in two balanced partitions[35]. On all other datasets, Postăvaru et al. uniformly sample 80% of all links for training and 10% for testing and use the remaining 10% for validation[35].

Postăvaru et al. observe that VERSE achieves the highest ROC-AU scores on all datasets, except for YouTube, where the highest ROC-AU score is achieved by InstantEmbedding[35]. Their results are summarised in Table 3.3.

Table 3.3: Average ROC-AUC scores and confidence intervals for link prediction as reported by Postăvaru et al.[35].

	CoAuthor	BlogCatalog	YouTube	Amazon2M
DeepWalk	88.43 $\pm$*1.08*	91.41 $\pm$*0.67*	82.17 $\pm$*1.02*	98.79 $\pm$*0.41*
Node2Vec	86.09 $\pm$*0.85*	92.18 $\pm$*0.12*	81.27 $\pm$*1.58*	–
VERSE	92.75 $\pm$*0.85*	93.42 $\pm$*0.35*	80.03 $\pm$*0.99*	99.67 $\pm$*0.18*
FastRP	82.19 $\pm$*2.22*	88.68 $\pm$*0.7*	76.3 $\pm$*1.46*	92.12 $\pm$*0.61*
InstantEmbedding	90.44 $\pm$*0.48*	92.74 $\pm$*0.6*	82.89 $\pm$*0.83*	99.15 $\pm$*0.18*

Rahman and Azad use 50% of all edges for training and the remaining 50% for testing[37]. They find that when using the Hadamard operator to generate edge embeddings, force2vec and VERSE achieve almost 99% accuracy in all datasets, while DeepWalk achieves a higher accuracy than all other methods when using the weighted L1 operator or the weighted L2 operator[37]. The results of their experiments are summarised in Figure 3.3. Rahman and Azad also observe, that on the Cora and Pubmed datasets, HOPE achieves 79.8% and 80% accuracy, respectively[37].



(a) Link prediction based on hadamard operator

(b) Link prediction based on L1 operator

(c) Link prediction based on L2 operator

Figure 3.3: Link prediction results by Rahman and Azad. Chart adapted from Rahman and Azad[37].

4 Embedding-based Coarsening

This work uses Node2Vec and InstantEmbedding to generate its embeddings. Embeddings are then used in a coarsening algorithm based on Size-Constrained Label Propagation to acquire a coarse graph.

4.1 Size Constrained Label Propagation

Size constrained label propagation is a clustering algorithm first proposed by Meyerhenke et al.[30]. Nodes are assigned to clusters and then join the clusters of those nodes adjacent to them, to which they have the strongest connection without violating a constraint to the maximum node weight of each cluster (see Section 3.2.1). Since the algorithms presented in this thesis are implemented in KaMinPar, a parallel multilevel graph partitioner which uses size constrained label propagation for its coarsening algorithm, this section presents the size constrained label propagation as used in KaMinPar[15], which is shown in Algorithm 3.

KaMinPar specifically uses the size constraint $U = \varepsilon \frac{c(V)}{k'}$, where $k' = \min\{k, \frac{|V|}{t}\}$, where t is a tuning parameter. In Line 4 of Algorithm 3, node traversal is not done in an arbitrary order. Instead, nodes are sorted in exponentially spaced *degree buckets*, such that bucket i contains all nodes u with degree $2^i \leq \deg(u) < 2^{i+1}$. Buckets are then broken down further into chunks, inside which the node order is randomised. Iterating over nodes is done chunk by chunk, with chunks of lower degree buckets being processed before chunks of higher degree buckets. This strategy seeks to apply the findings of Meyerhenke et al., who observe that the solution quality of size constrained label propagation increases if node are visited in increasing degree order[30], while maintaining some diversification through randomisation and improving cache efficiency.

Size constrained label propagation as used in KaMinPar also uses *two-hop clustering*. For every node u where the size constraint prevents u from joining any clusters, the cluster which receives the highest rating by u is stored (see Lines 10-11 of Algorithm 3). If the graph later shrinks by less than 50%, all nodes that could not join any clusters and share the same highest rated cluster are moved to the same new cluster (see Line 12-13 of Algorithm 3).

Algorithm 3: Size Constrained Label Propagation in KaMinPar

```
Input: Graph G = (V, E), Rounds r, Tuning Parameter t
Output: Clustering C
1  U ← ε c(V) / min{k, |V|/t}                          // set the size constraint
2  C ← [v | v ∈ V]                                     // initialise clustering
3  for i ← 1 to r do
4      foreach u ∈ V do
5          B ← adjacentClustersToNode(u)
6          B̃ ⊆ B all clusters b, where c(b) + c(u) ≤ U
7          b ← arg max_{b∈B̃} Σ_{v∈b | {u,v}∈E} rating(u, v)   // select highest rated cluster
8          if c(b) + c(u) ≤ U then
9              C[u] ← b
10         else
11             favouredCluster_u ← arg max_{b∈B} Σ_{v∈b | {u,v}∈E} rating(u, v)
12 if C would shrink G by less than 50% then
13     cluster all nodes in singleton clusters with same favoured clusters
14 return C
```

4.1.1 Using Embeddings

Line 7 of Algorithm 3 shows the use of an edge rating function $\mathrm{rating}(u, v)$ for edges $\{u, v\}$ to find the cluster that a node u should join. In standard size constrained label propagation, $\mathrm{rating}(u, v) = \omega(\{u, v\})$. Part of this work is to examine alternative edge rating functions that use embeddings to influence the score of edges such that edges between similar nodes achieve higher scores than edges between dissimilar nodes.

Srybrandt et al. use Equation 4.1.1 as the rating function in coarsening algorithm based on heavy edge matching[43] (cf. Section 3.2.2), where they use a heavy node penalty to prevent too many heavy nodes being contracted into the same coarse node.

$$\mathrm{rating}(\{u, v\}) = \frac{\epsilon(u)^T \epsilon(v)}{c(u)c(v)} \cdot w(\{u, v\}) \tag{4.1.1}$$

The size constraint of size constrained LPA makes this heavy node penalty somewhat redundant and preliminary experiments done for this thesis show that removing the heavy node penalty improves solution quality. As such, this thesis uses Equation 4.1.2 as its edge rating function in size constrained LPA.

$$\mathrm{rating}(\{u, v\}) = \epsilon(u)^T \epsilon(v) \cdot w(\{u, v\}) \tag{4.1.2}$$

Further experiments are done with the rating functions depicted in Equations 4.1.3 and 4.1.4. These scale the embedding dot product of the incident nodes super-linearly. This gives highly similar nodes a higher chance of getting inserted into the same cluster.

$$\text{rating}_{\text{poly}^x}(\{u,v\}) = (\epsilon(u)^T\epsilon(v))^x \cdot w(\{u,v\}) \tag{4.1.3}$$

$$\text{rating}_{\text{expo}}(\{u,v\}) = 2^{\epsilon(u)^T\epsilon(v)} \cdot w(\{u,v\}) \tag{4.1.4}$$

All of these rating functions have the use of the embedding dot product in common, the computation of which adds $O(d)$ steps to the total runtime of evaluating a single edge during label propagation.

4.2 Embedding Contraction

Computing embeddings is expensive in terms of runtime. Additionally, preliminary experiments done for this thesis and by Srybrandt et al. suggest that computing new embeddings on each coarse graph does not improve solution quality[43], so rather than computing a new embedding at each level, this work projects embeddings to coarse levels. This is done by setting the embedding of a coarse node to the centroid of all contributing nodes, as shown by Algorithm 4. More formally, given a coarse node u^C which consists of nodes $u_1, u_2, ..., u_j$, the embedding $\epsilon(u^C)$ of u^C is defined by $\epsilon(u^C) = \frac{1}{j}\sum_{i=1}^{j}\epsilon(u_i)$.

Algorithm 4: Embedding Contraction

`Input`: Embedding W of size $n \times d$, Clustering C
`Output`: Embedding W'

```
1 n_C the number of clusters in C
2 W' an embedding matrix of size n_C × d
3 foreach Cluster V do
4 |   n_V the number of nodes in cluster V
5 |   u^C the node to which V is contracted in the coarse graph
6 |   W'[u^C, 1 : d] ← (1/n_V) Σ_{v∈V} ε(v)
7 return W'
```

Algorithm 4 illustrates that the impact of embedding contraction on the runtime needed to compute a new coarse graph is not negligible. It requires that each embedding vector is used in vector addition and also that each sum of these vector additions is divided by a scalar, which requires $O(nd)$ steps on each step on each level of the graph hierarchy.

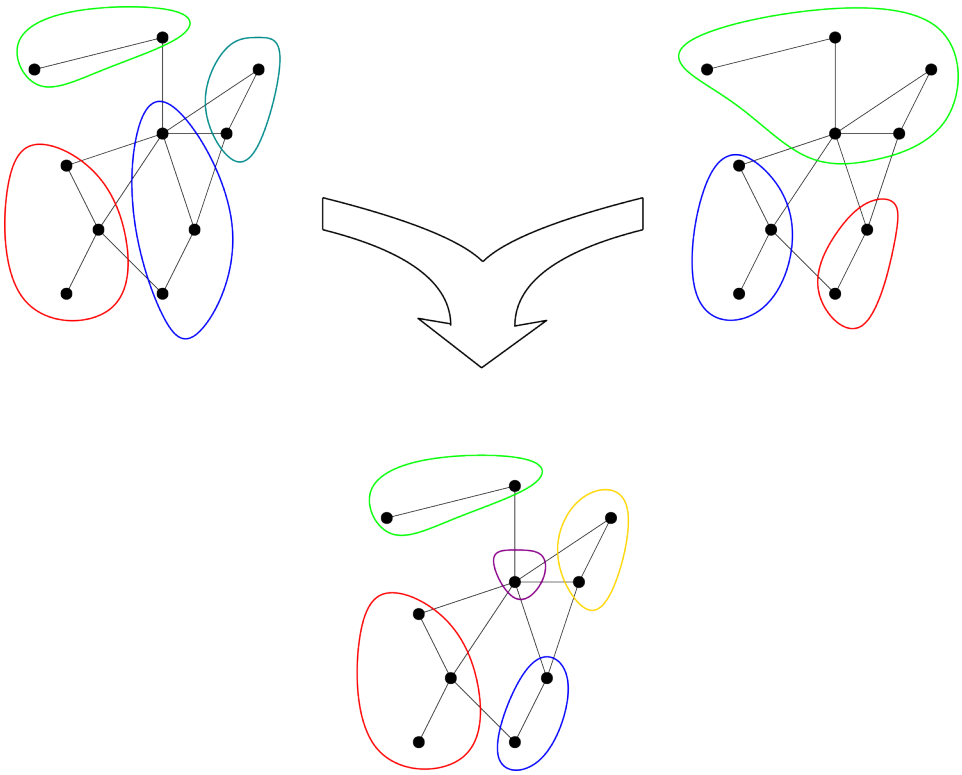


Figure 4.1: Overlaying two clusterings means combining two clusterings so that clusters in the resulting clustering only contain nodes which are part of the same initial clusters.

4.3 Cluster Overlaying

Meyerhenke et al. use the term *overlay clustering* to describe the process of combining two or more clusterings to acquire a new clustering[30]. This technique is also known in the field of machine learning as "ensemble learning", where multiple weak classifiers are combined to form a single stronger classifier.

Combining clusterings is done by placing two nodes u, v in the same cluster C if and only if they are part of the same cluster in both clusterings. More formally, given two clusterings V_a, V_b and two clusters C_a, C_b with nodes $u \in C_a$, $v \in C_b$, u and v are part of the same cluster C if and only if $u \in C_b \wedge v \in C_a$. For an illustration, refer to Figure 4.1.

This work assesses two kinds of overlay clusterings: the first kind combines two clusterings obtained through size constrained LPA, while the second kind combines two size constrained LPA clusterings where one clustering uses embeddings. This approach decreases the rate at which embedding-based LPA clusters similar nodes and prevents “bad” decisions in the coarsening process, while still making the more global structural information stored in graph embeddings available in the coarsening process.

Overlaying increases the total runtime needed to compute a coarse graph: it requires the

LPA algorithm to be executed twice instead of once and requires the subsequent combination of the two obtained clusterings. Additionally, overlaying generally increases the number of clusters present in the final clustering. This means that the coarse graphs resulting from clusterings obtained through overlaying will be bigger, potentially increasing the number of coarsening steps that need to be done before initial partitioning can begin.

4.4 Algebraic Distance

Algebraic distance is a measure of the connectivity between two vertices[9]. The use of it is evaluated in this thesis because it is an approach which captures structural information about the nodes of a graph that has been used in graph coarsening before[9, 40].

Algorithm 5 describes how to compute the algebraic distance. The parameter α in Line 11 of Algorithm 5 is set to $\alpha = 0.5$, as Chen et al. use the same for their own graph partitioning experiments. Chen et al. also note that the number of iterations (cf. Line 6 of Algorithm 5) does not significantly influence the solution quality, so that iterating may be stopped before convergence. Line 14 of Algorithm 5 also performs degree normalisation, which is an optional addition to the method of computing the algebraic distance described by Chen et al.

Algorithm 5: Algebraic Distance

`Input`: Graph $G = (V, E)$, Parameter R
`Output`: Algebraic Distance Matrix W

1 x array of size $|V|$
2 W matrix of size $|V| \times |V|$ *// assume initialisation with* 0
3 `for` $r = 1$ `to` R `do`
4 $x^{(0)}$ array of size $|V|$
5 initialise $x^{(0)}$ with random values
6 `for` $k = 1, 2...$ `do`
7 $\bar{x}$ array of size $|V|$
8 `foreach` $u \in V$ `do`
9 $N \leftarrow \{v \mid \{u, v\} \in E\}$
10 $\bar{x}[u] \leftarrow \sum_{v \in N} \omega(\{u, v\}) x^{(k-1)}[v] \;/\; \sum_{v \in N} \omega(\{u, v\})$
11 $x^{(k)} \leftarrow (1 - \alpha) x^{(k-1)} + \alpha \bar{x}$
12 $x \leftarrow x^{(k)}$
13 `foreach` $\{u, v\} \in E$ `do`
14 $W[u, v] \leftarrow \frac{\max\{|x[u] - x[v]|, W[u,v]\}}{\sqrt{\text{degree}(u) \cdot \text{degree}(v)}}$
15 `return` W

Chen et al. use the inverse of the algebraic distance as an edge rating function during graph coarsening[9]. This thesis employs this edge rating function as well, whenever algebraic distance is used for a configuration. This makes for an edge rating function whose runtime is not much different to using the edge weight $\omega(e)$ to rate an edge e. Instead, it requires an increase of space complexity by $\Omega(m)$, unless edge weights are replaced with the respective algebraic distance on the top level of the graph hierarchy.

5 Experimental Evaluation

5.1 Implementation

All algorithms were implemented as part of the KaMinPar parallel graph partitioner and compiled using g++-14.1. Intel's TBB library[1] was used for parallelisation and Eigen[17] for the implementation of embeddings and operations on embeddings.

5.1.1 Embedding Implementation

Embeddings are implemented as wrappers around dense or sparse matrices, where the exact type has to be chosen in code. For each level of the graph hierarchy, the respective embedding is saved separately, rather than replacing the edge weights of the graph. By default, embeddings are computed once at the top level and then contracted on each level of the hierarchy in accordance with the algorithm outlined in Section 4.1.1, although passing the flag `--c-embedding-recompute` will cause embeddings to be recomputed on each level of the graph hierarchy.

For this work, InstantEmbedding (ref. Section 3.4.1) and Node2Vec (ref. Section 3.4.2) were chosen as embedding methods. Both methods achieve good results in the node classification and link prediction experiments done by Postăvaru et al.[35] and Rahman and Azad[37]. Additional benefits include that Node2Vec's is widely known and that a basic implementation of InstantEmbedding can be done fairly easily.

A parallel implementation of InstantEmbedding was added to KaMinPar, which makes use of InstantEmbedding's ability to compute globally consistent node embeddings independently of all other nodes (cf. Section 3.4.1). For Node2Vec embeddings, Tsitsulin and Bischoff's C++ implementation of Node2Vec[46, 45] was used.

5.1.2 Coarsening by Size Constrained Label Propagation and Heavy Edge Matching

KaMinPar already contains an implementation of a coarsener based on size constrained LPA, which is extended for the purpose of this thesis. To allow overlaying clusters during coarsening, the existing coarsener is extended to optionally overlay two clusterings. Clusterings are then overlayed following the algorithm outlined in 6. All nodes are inserted into

a hash table, where the key for a node u contained in clusters C_a, C_b is the tuple (C_a, C_b). The new clustering is derived by placing all nodes with the same key in the same cluster.

Algorithm 6: Cluster Overlaying

```
Input: Clustering C_a, C_b, Clustering Size n
Output: Clustering C
1 hash table H
2 Clustering C of size n
3 c ← 0
4 for u ∈ {0..n − 1} do
5     k ← (C_a[u], C_b[u])
6     if H does not have key k then
7         H[k] ← c
8         c ← c + 1
9     C[u] ← H[k]
```

KaMinPar's implementation of size constrained LPA[30] is extended to enable the use of Equation 4.1.2 for rating edges. By passing a template parameter, the type of the the LPA implementation can be switched between using normal edge weights and embedding-based edge weights. When using embedding-based edge weights, Equation 4.1.2 is used for rating edges. The LPA implementation iterates over all nodes in parallel and updates cluster weights with atomic fetch-and-add operations.

Additionally, a coarsener based on Heavy Edge Matching was added to KaMinPar, which is a reimplementation of Procedure 1 in [43].

5.2 Experimental Setup

5.2.1 Environment and Parameters

All experiments were conducted on the single partition of bwUniCluster 2, with each instance having 32 cores and 18 gB of memory allocated to it. To increase the observable coarsening depth, all configurations of KaMinPar are run with `--c-contraction-limit 160`.

Configuration Naming Scheme The following experiments use different configurations of KaMinPar, whose names contain consist of affixes which describe the configuration. Some of these affixes appear in configurations used throughout most experiments. These affixes are "Base", which denotes that a configuration uses KaMinPar's

default coarsener based on standard size constrained Label Propagation, "n2v", which denotes the use of Node2Vec-based Label Propagation and "ie", which denotes the use of InstantEmbedding-based Label Propagation.

To denote the strategy refinement, configurations may have the "FM" suffix for FM-based refinement using KaMinPar's `fm` preset or "4xJet" suffix for Jet-based refinement using KaMinPar's `4xjet` preset. Configurations that have neither of these suffixes use KaMinPar's default refinement strategy, which is based on size constrained Label Propagation.

5.2.2 Instances

Graph instances consist of 37 graphs from the 10th DIMACS implementation challenge[5], 5 road networks from the 9th DIMACS implementation challenge[3], 12 VLSI graphs from Testset ALUE[24, 23], 10 graphs from the SNAP Large Network Dataset Collection[26], as well as the social network `wordassociation-2011`[6, 7] and the circuit simulation problem `scircuit_spmv`[18]. To keep embedding time and embedding size down, only graphs with less than 1.5 million nodes and less than 2 million edges were chosen. See Table A.1 for a full list of all graph instances.

Experiments used ten seeds and four values of k ($k \in \{2, 8, 32, 128\}$) and were conducted with a maximum allowed imbalance of 3%. Embeddings for all graphs were precomputed once for every graph and every seed and then reused in all subsequent experiments.

5.2.3 Methodology

Each combination of a graph and a value of k is run ten times with different seeds. The produced edge cuts are then aggregated with the arithmetic mean over all seeds.

Comparison is done using *performance profiles*[10]. Given the set of all algorithms to compare $\mathcal{A}$ and the set of all instances $\mathcal{I}$ and the quality $q_A(I)$ of an algorithm $A \in \mathcal{A}$ on an instance $I \in \mathcal{I}$, $\mathcal{I}_A = \{I \in \mathcal{I} \,|\, q_A(I) \leq \min_{A' \in \mathcal{A}} q_{A'}(I)\}$ is the set of all instances where A achieves a solution within tau of the best solutions for I_A. The fraction of all instances $\frac{|\mathcal{I}_A|}{|\mathcal{I}|}$ is plotted against the ratio τ.

5.3 Evaluation of Embedding-based Coarsening

Figures 5.1 and 5.2 show performance profiles for the average cuts of baseline KaMinPar with LPA coarsening based on embeddings computed with Node2Vec and InstantEmbedding, respectively. While Node2Vec embeddings lead to measurably improved solution quality, the solution quality gets noticeably worse when using InstantEmbedding as the embedding method.

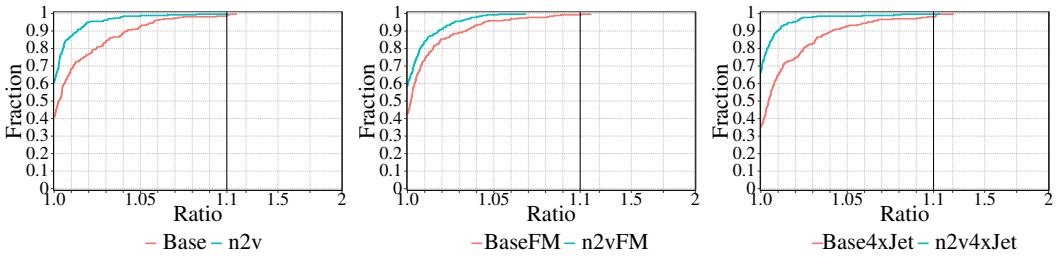


Figure 5.1: Node2Vec-based LPA coarsening achieves a higher solution quality than the baseline.

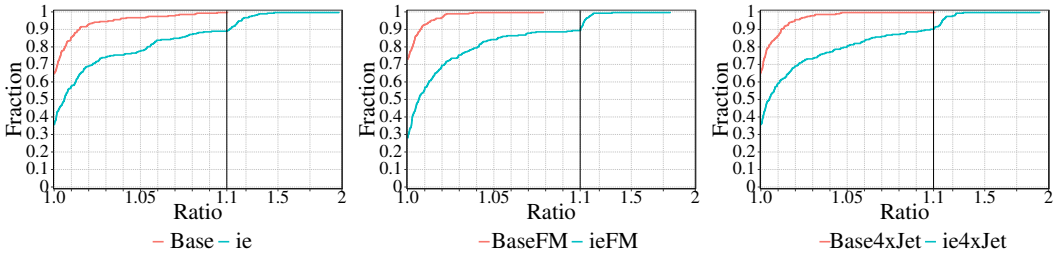


Figure 5.2: Coarsening by LPA based on InstantEmbedding is achieves lower solution quality than the baseline.

Figures 5.1 and 5.2 also show that the impact that embeddings have on the solution quality depends on the used refinement strategy. Figure 5.1 shows that Node2Vec-based LPA coarsening leads to more significant improvements of the solution quality when combined with the default refinement or the 4xJet preset, compared to the FM preset. Likewise, Figure 5.2 shows that the gap in solution quality is InstantEmbedding-based LPA coarsening is significantly higher when using the FM preset, than the gap in solution quality when the default refinement or the 4xJet preset are used.

Table 5.1 shows the standard deviation of the cuts produced by the Base, n2v and ie configurations on each instance. From the Table 5.1 it is evident that while the standard deviation of the cuts produced by all three configurations is similar on most instances, both n2v and

Configuration	Base	n2v	ie
Minimum	0.00	0.00	0.00
1st Quartile	12.69	10.52	12.77
Median	71.05	60.24	65.42
Mean	369.19	434.38	528.44
3rd Quartile	527.68	422.05	462.45
Maximum	4116.00	16995.78	13600.95

Table 5.1: Comparison of the standard deviation of solutions found by the Base, n2v and ie configurations for each instance.

especially ie suffer from some outliers.

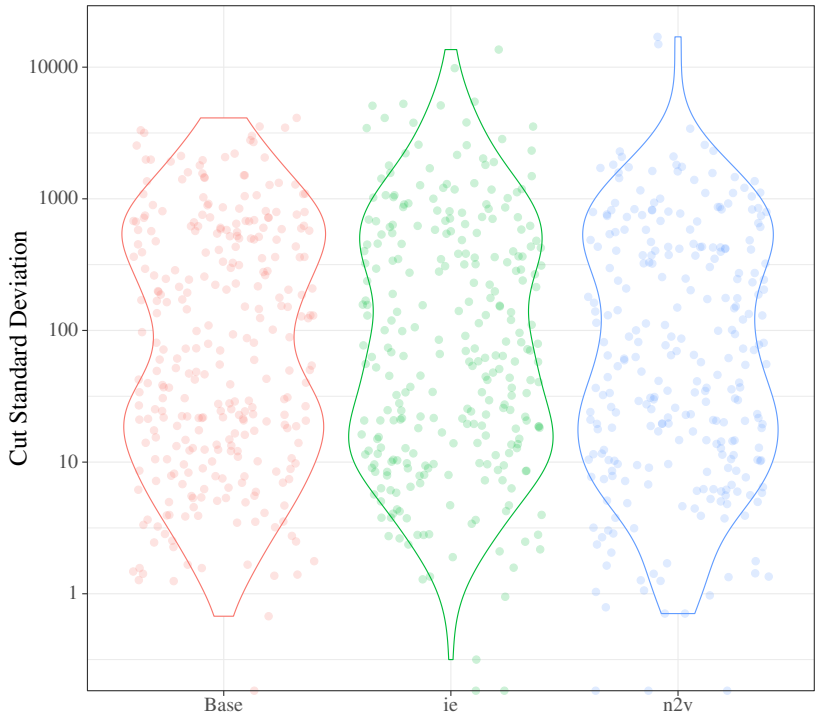


Figure 5.3: Distribution of the standard deviation of the cuts of Base, n2v and ie.

Figure 5.3 shows that for n2v, the standard deviation of the cut is similar to the standard deviation of the cuts produced by Base, except for the graphs `soc-Slashdot0811` and `soc-Slashdot0902` with $k = 2$ which are prominent outliers.

5.3.1 Correlation between Embeddings and Final Partition

The influence of embeddings on the final partition is assessed using Welch's ANOVA. The test compares dot products of the embedding vectors of 4 different node pair categories: intra-block edges and inter-block edges, as well as intra-block non-edges and inter-block non-edges.

The number of runs where the difference between groups is significant at a 5% level varies between configurations ie and n2v, with the difference being significant on 89.9% of all runs of the ie configuration and 92.8% of all runs of the n2v configuration. Let $\mathcal{R}_{\text{ie}}$, $\mathcal{R}_{\text{n2v}}$ be the sets of all runs on which a significant difference between node pair categories is found for the ie, respective n2v, configuration. Further analysis using a Games-Howell post-hoc test shows that the difference between intra-block edges and inter-block edges is significant at a 5% level on 33.3% of $\mathcal{R}_{\text{ie}}$ and on 41.5% of $\mathcal{R}_{\text{n2v}}$.

These results are in line with experiments in Section 5.3, where n2v beats the baseline's solution quality but ie does not beat the baseline's solution quality. The lower percentage of runs with a significant difference between categories of the ie configuration further underlines how the latent structural information of the embedding remains unused.

5.3.2 Alternative Edge Rating Functions

For this experiment, label propagation is done using the edge rating functions $\mathrm{rating}_{\mathrm{poly}^x}(\{u,v\}) = (\epsilon(u)^T\epsilon(v))^x \cdot w(\{u,v\})$ and $\mathrm{rating}_{\mathrm{expo}}(\{u,v\}) = 2^{\epsilon(u)^T\epsilon(v)} \cdot w(\{u,v\})$. Configurations using the edge rating function expo carry the affix "expo", while configurations using an edge rating function poly^x carry the affix "poly$x$".

Figure 5.4 shows that the edge rating functions $\mathrm{rating}_{\mathrm{poly}^2}$ and $\mathrm{rating}_{\mathrm{poly}^3}$ do not significantly change the solution quality when combined with Node2Vec. Figure 5.5 shows that while this holds true for the combination of $\mathrm{rating}_{\mathrm{poly}^2}$ with InstantEmbedding, a combination of $\mathrm{rating}_{\mathrm{poly}^3}$ with InstantEmbedding leads to a lower solution quality.

For $\mathrm{rating}_{\mathrm{expo}}$, results are more different. Figure 5.6 shows that while for both Node2Vec and InstantEmbedding $\mathrm{rating}_{\mathrm{expo}}$ leads to a loss in solution quality, Figure 5.6 also shows that InstantEmbedding is affected more than Node2Vec.

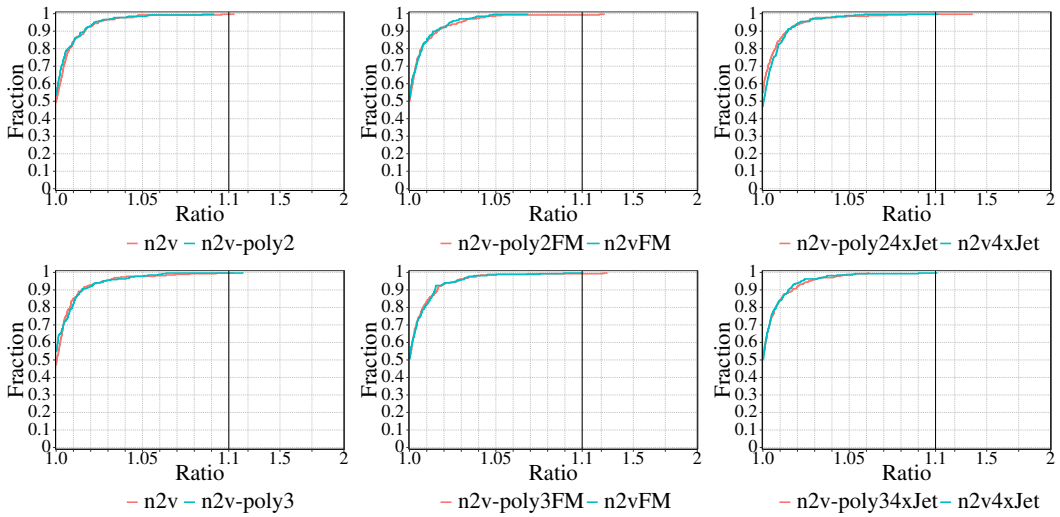


Figure 5.4: Using $\mathrm{rating}_{\mathrm{poly}^2}$ or $\mathrm{rating}_{\mathrm{poly}^3}$ with Node2Vec does not impact solution quality.

While both InstantEmbedding-based configurations and Node2Vec-based configurations do not benefit from any of the alternative edge rating functions, Figures 5.5 and 5.6 show that InstantEmbedding-based configurations are more significantly influenced by the choice of the edge rating function than their Node2Vec-based counterparts.

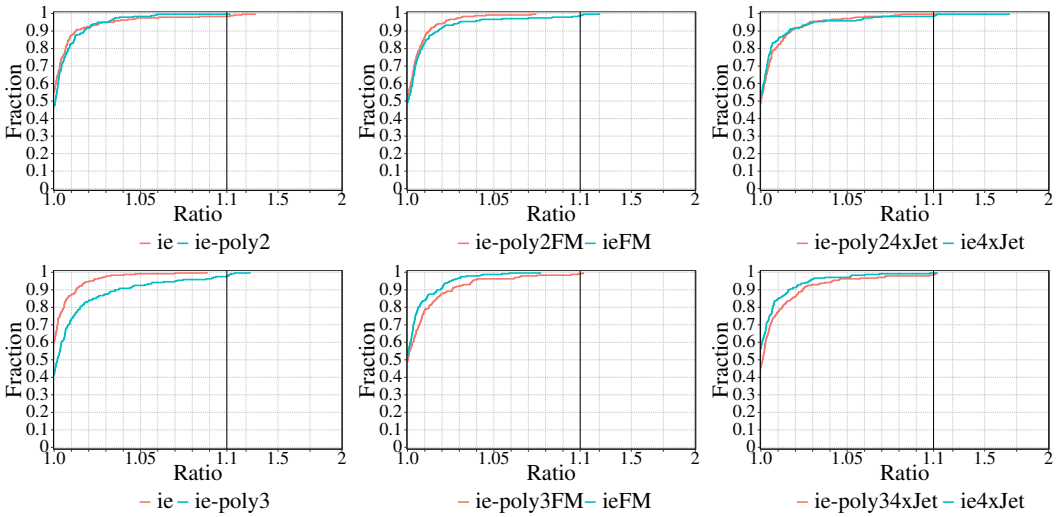


Figure 5.5: While using $\mathrm{rating}_{\mathrm{poly}^2}$ with InstantEmbedding does not impact solution quality, using $\mathrm{rating}_{\mathrm{poly}^3}$ with InstantEmbedding leads to an worse overall worse solution quality.

5.4 Overlaying Clusterings

For this experiment, clusterings obtained through standard size constrained LPA were overlayed with standard clusterings and embedding-based clusterings, respectively. By combining a clustering obtained through standard LPA with an embedding-based clustering, the effect of overeager merging can be reduced which can make the use of embeddings in clustering-based coarsening strategies more worthwhile. The configurations studied for this experiment use the affix “overlay” to show that they use overlaying.

Figures 5.7 and 5.8 compare the solution quality between baseline KaMinPar, KaMinPar with two overlayed LPA clusterings and KaMinPar with an overlaying of a normal LPA clustering with a Node2Vec-based LPA clustering, respective InstantEmbedding-based LPA clustering.

Like in Section 5.3, Node2Vec-based overlay clusterings achieve a measurable increase in solution quality, compared to when overlaying clusterings without embeddings. Importantly, InstantEmbedding-based cluster overlaying achieves higher solution quality than overlaying clusterings without embeddings, which is unlike the results of the previous experiment (see Section 5.3). However, the increase in solution quality is not as high as with Node2Vec-based cluster overlaying.

The differences in solution quality depending on the used refinement strategy observed in Section 5.3 can be observed here as well: Figure 5.7 shows a significantly larger difference in the solution quality of Node2Vec-based overlaying and standard overlaying. Likewise, Figure 5.8 shows a measurable improvement in solution quality when using InstantEmbedding-based overlaying over standard overlaying with the 4xJet preset and also

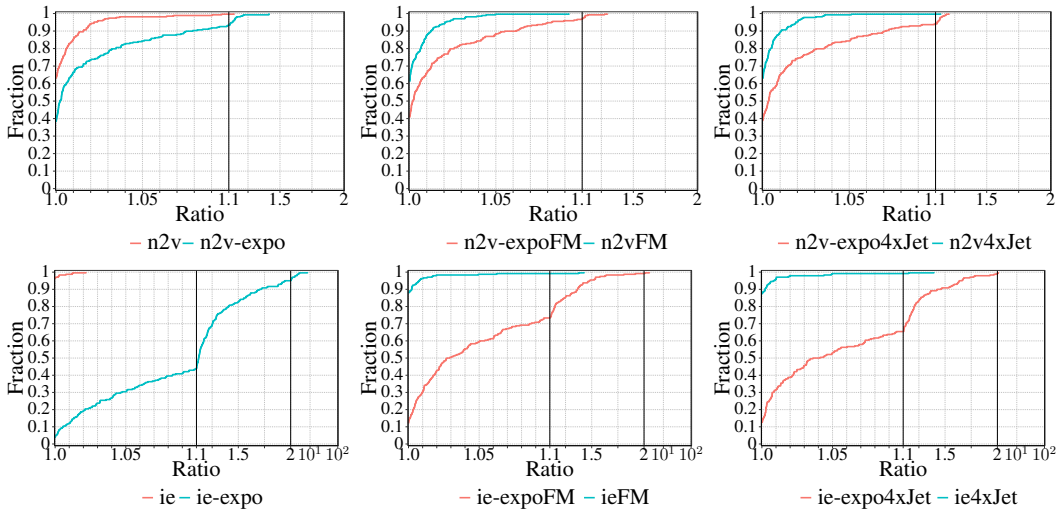


Figure 5.6: Unlike $\mathrm{rating}_{\mathrm{poly}}$, using $\mathrm{rating}_{\mathrm{expo}}$ leads to a significant loss of solution quality.

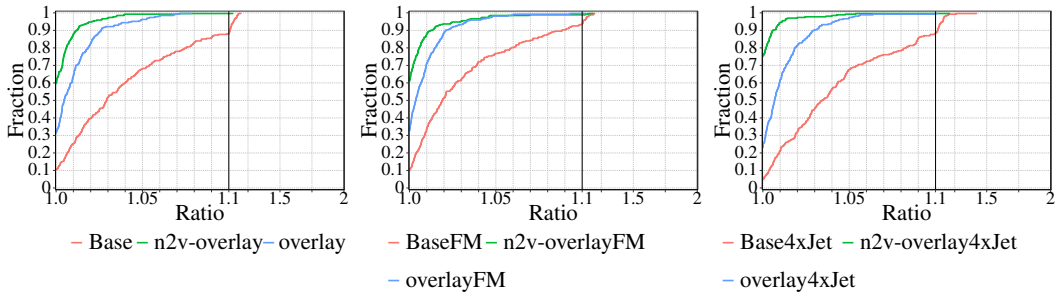


Figure 5.7: Overlaying Node2Vec-based LPA with standard LPA achieves higher solution quality than the baseline and overlaying two clusterings obtained by standard LPA.

the default refinement but only a very small difference in solution quality when comparing InstantEmbedding-based overlaying to standard overlaying with the FM preset.

Figures 5.7 and 5.8 also show that overlaying leads to significant improvement in solution quality over the baseline, regardless of the specific configuration used. Appendix A.2 illustrates that overlaying benefits embedding-based coarsening methods especially on irregular graphs. Both the n2v configuration and the ie configuration generally perform worse on irregular graphs than on regular graphs. Using them in conjunction with cluster overlaying significantly increases the solution quality of both configurations on irregular graphs, with n2v-overlay outperforming both overlay and Base on irregular graphs.

Overlaying comes at the cost of increased coarsening time, with the median coarsening depth of the overlay configuration being twice as high as the median coarsening depth of baseline KaMinPar and maximum coarsening depth being 1.5 times as high.

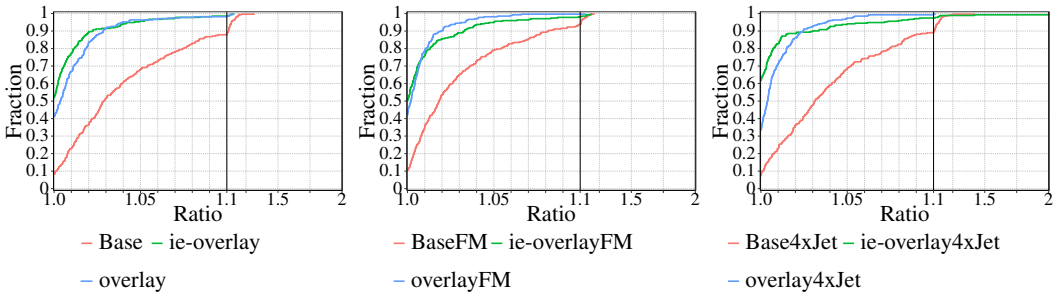


Figure 5.8: Solution quality generally benefits from using InstantEmbedding when combining it with overlaying.

5.5 Algebraic Distance-based Coarsening

Chen et al.[9] first apply algebraic distances in graph coarsening as an exemplary use case of algebraic distances for graph applications. They implement a matching-based graph coarsener in HMetis2[21] and report measurable improvements to the cut of the partitioned graphs. Safro et al.[40] later build a more complex graph coarsener in KaFFPa[42] and report that it achieves lower cuts than alternative matching-based coarseners that do not make use of algebraic distances.

Since algebraic distances are a form of distance metric on graphs as well, algebraic distance-based coarsening provides a natural extension to the scope of this work and as such is implemented in KaMinPar as an embedding with dimension $d = n$. Configurations that use algebraic distances carry the affix “ad”. Additionally, configurations that use heavy edge matching-based coarsening carry the affix “hem”.

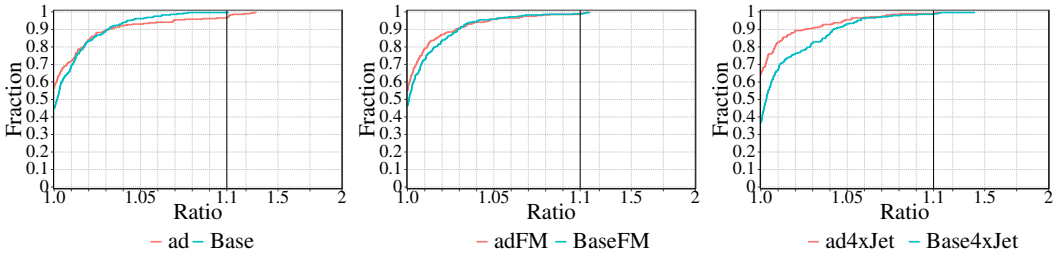


Figure 5.9: Algebraic distances improve the solution quality of LPA-based coarsening, although effectiveness with LPA-based refinement and FM refinement is limited.

Figures 5.9 and 5.10 show performance profiles for the cuts of algebraic distance-informed LPA-based coarsening and algebraic distance-informed matching-based coarsening, respectively. While algebraic distances lead to improvements with all evaluated configurations, the observed improvements are noticeably smaller when comparing configurations based on LPA, with ad and adFM achieving only minor improvements over base KaMinPar and KaMinPar-fm, respectively.

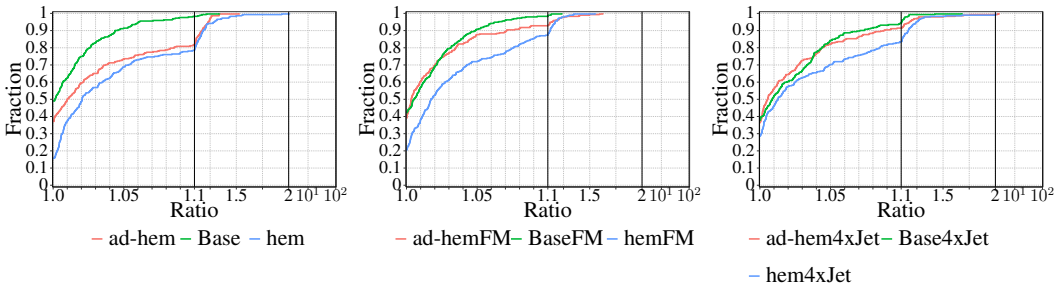


Figure 5.10: Algebraic distances improve the solution quality of matching-based coarsening but are outperformed by the baseline or improve solution quality only marginally.

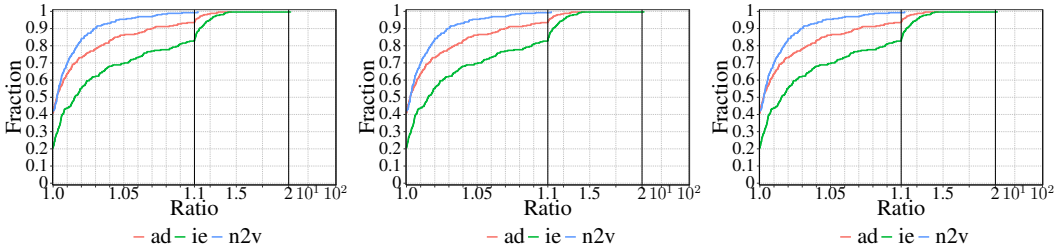


Figure 5.11: Comparison of the solution quality of embedding-based and algebraic distance-based configurations.

Figure 5.11 shows that Node2Vec-based LPA and algebraic distance-based LPA lead to similar solution quality when combined when the FM or 4xJet presets are used during refinement. Figure 5.11 further shows that both algebraic distance-based LPA and Node2Vec-based LPA further lead to a higher solution quality than InstantEmbedding-based LPA.

6 Conclusion and Future Work

This thesis presented two approaches to using embeddings in the coarsening phase of multi-level graph partitioning. These approaches were evaluated by implementing them in KaMinPar and comparing their effectiveness with two embedding methods, Node2Vec and InstantEmbedding.

By extending the edge rating function used in label propagation-based coarsening to consider the node embeddings of participating nodes, embeddings can be used to improve the solution quality of existing graph partitioners. This approach is not agnostic to the used embedding method: while the Node2Vec-based configuration achieved a measurable increase in solution quality, the InstantEmbedding-based configuration experienced a decrease in solution quality. The effectiveness also depends on the used refinement strategy with FM-based refinement generally leading embedding-based coarsening to have worse performance than if the refinement strategy is based on size constrained label propagation or jet. Additionally, experiments with alternative edge rating functions lead to the observation that Node2Vec-based configurations are less susceptible to the choice of edge rating function than InstantEmbedding-based configurations.

By combining the clusterings produced by embedding-based label propagation and standard label propagation, both InstantEmbedding and Node2Vec can be used to increase the solution quality. This effect can be observed regardless of whether the combined clustering is compared to standard label propagation-based coarsening or to a coarsening process that combines two standard clusterings. Additionally, Appendix A.2 shows that overlaying has an especially positive effect on the solution quality on regular graphs.

Algebraic distances, which have already been investigated for their use in graph coarsening[9, 40], were used in label propagation-based coarsening to label propagation-based coarsening based on Node2Vec and InstantEmbedding, respectively. Algebraic distances are observed to improve the solution quality of standard label propagation-based coarsening and configurations using algebraic distance-based label propagation during coarsening find significantly better partitions than configurations that use InstantEmbedding-based label propagation during coarsening. Compared to Node2Vec-based label propagation, label propagation-based coarsening based on algebraic distances leads to a similar solution quality, with some influence from the respective refinement strategy.

6.1 Future Work

While this thesis presents a way of successfully using InstantEmbedding for graph partitioning, its usefulness in this context remains limited. It does, however, show the importance of employing embeddings in a way that works well with the embedding method in question. As such, it would benefit from further research in applying embeddings in graph coarsening and label propagation-based graph coarsening in particular. Based on the observation that there is a significant difference in the solution quality between Multilevel Graph Partitioning with InstantEmbedding-based coarsening and with Node2Vec-based coarsening, an investigation of more different embedding methods appears worthwhile.

Additionally, the scope of this thesis has been limited by the size of the graphs used in experiments. While using smaller graphs allows for experiments to be cheaper concerning runtime and space, it is also limiting in the sense that it is difficult to observe the impact of graph size on the final solution quality. Likewise, only two embedding methods were evaluated, although many other potentially promising embedding methods exist

Meyerhenke et al. describe that the order in which nodes are evaluated during label propagation can have positive effects on the solution quality[30]. Specifically, Meyerhenke et al. mention the positive effects of choosing to iterate nodes based on their degree, which is the basis of the default iteration strategy followed in KaMinPar. Future investigation could thus entail the use of different, embedding-informed iteration strategies during label propagation.

A Experiments

A.1 Instances

Graph Name	Dataset	Nodes $\|V\|$	Edges $\|E\|$
144	DIMACS 10[5]	144649	1074393
3elt	DIMACS 10[5]	4720	13722
4elt	DIMACS 10[5]	15606	45878
598a	DIMACS 10[5]	110971	741934
add32	DIMACS 10[5]	4960	9462
alue3146	Testset ALUE[24]	3626	5869
alue5067	Testset ALUE[24]	3524	5560
alue5345	Testset ALUE[24]	5179	8165
alue5623	Testset ALUE[24]	4472	6938
alue5901	Testset ALUE[24]	11543	18429
alue6179	Testset ALUE[24]	3372	5213
alue6457	Testset ALUE[24]	3932	6137
alue6735	Testset ALUE[24]	4119	6696
alue6951	Testset ALUE[24]	2818	4419
alue7065	Testset ALUE[24]	34046	54841
alue7066	Testset ALUE[24]	6405	10454
alue7080	Testset ALUE[24]	34479	55494
amazon0302	SNAP[26]	262111	899792
bay	DIMACS 9[3]	321270	397415
bcsstk29	DIMACS 10[5]	13992	302748
bcsstk30	DIMACS 10[5]	28924	1007284
bcsstk31	DIMACS 10[5]	35588	572914
bcsstk32	DIMACS 10[5]	44609	985046
bcsstk33	DIMACS 10[5]	8738	291583
brack2	DIMACS 10[5]	62631	366559
ca-GrQc	SNAP[26]	5241	14484
citationCiteseer	DIMACS 10[5]	268495	1156647
coAuthorsCiteseer	DIMACS 10[5]	227320	814134
coAuthorsDBLP	DIMACS 10[5]	299067	977676

col	DIMACS 9[3]	435666	521200
com-amazon	SNAP[26]	548552	925872
com-dblp	SNAP[26]	425957	1049866
crack	DIMACS 10[5]	10240	30380
cs4	DIMACS 10[5]	22499	43858
cti	DIMACS 10[5]	16840	48232
data	DIMACS 10[5]	2851	15093
ecology1	DIMACS 10[5]	1000000	1998000
ecology2	DIMACS 10[5]	999999	1997996
email-Enron	SNAP[26]	36692	183831
email-EuAll	SNAP[26]	265009	364481
fe_4elt2	DIMACS 10[5]	11143	32818
fe_body	DIMACS 10[5]	44775	163734
fe_ocean	DMACS 10[5]	143437	409593
fe_pwt	DIMACS 10[5]	36463	144794
fe_rotor	DIMACS 10[5]	99617	662431
fe_sphere	DIMACS 10[5]	16386	49152
fe_tooth	DIMACS 10[5]	78136	452591
finan512	DIMACS 10[5]	74752	261120
fla	DIMACS 9[3]	1070376	1343951
loc-brightkite	SNAP[26]	58228	214078
loc-gowalla	SNAP[26]	196591	950327
m14b	DIMACS 10[5]	214765	1679018
memplus	DIMACS 10[5]	17758	54196
nw	DIMACS 9[3]	1207945	1410387
ny	DIMACS 9[3]	264346	365050
roadNet-PA	SNAP[26]	1090920	1541898
roadNet-TX	SNAP[26]	1393383	1921660
scircuit_spmv	Hamm[18]	341996	958936
soc-Epinions1	SNAP[26]	75879	405740
soc-Slashdot0811	SNAP[26]	77360	469180
soc-Slashdot0902	SNAP[26]	82168	504230
t60k	DIMACS 10[5]	60005	89440
uk	DIMACS 10[5]	4824	6837
vibrobox	DIMACS 10[5]	12328	165250
wave	DIMACS 10[5]	156317	1059331
web-NotreDame	SNAP[26]	325729	1090108
web-Stanford	SNAP[26]	281903	1992636
whitaker3	DIMACS 10[5]	9800	28989

wiki-Vote	SNAP[26]	7115	100762
wing	DIMACS 10[5]	62032	121544
wing_nodal	DIMACS 10[5]	10937	75488
wordassociation-2011	LAW[7, 6]	10617	63788

Table A.1: Full list of all instances used in experiments (See Section 5.2).

A.2 Graph Types

During the experiments shown in Chapter 5, differences between the solution quality on regular and irregular graphs could be observed.

Figure A.1 shows that for the n2v configuration, solution quality is mostly improved on regular graphs, with the Base configuration beating n2v on irregular graphs with LPA-based refinement as well as FM refinement.

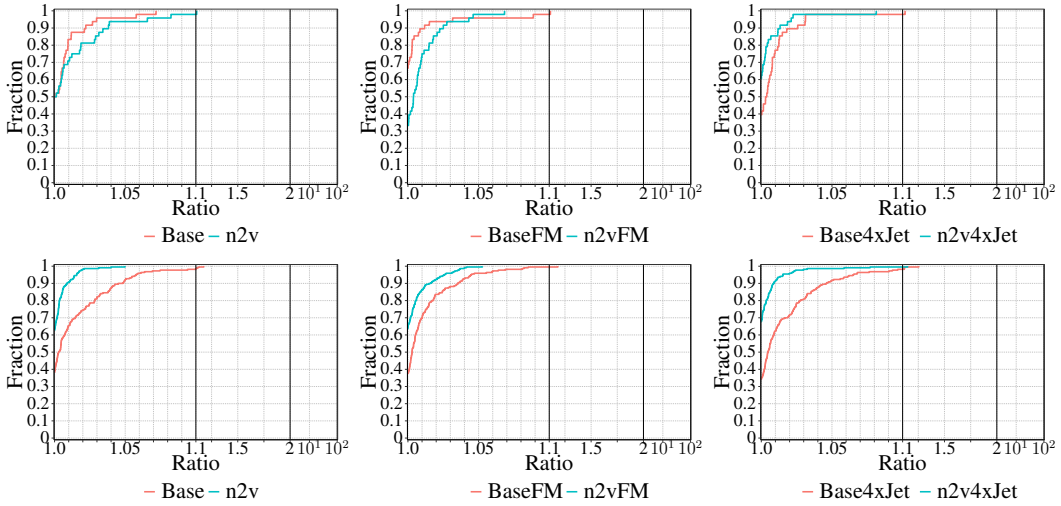


Figure A.1: Irregular graphs on top, regular graphs at the bottom. Node2Vec-based LPA outperforms the baseline on regular graphs but not on regular graph.

Figure A.2 shows that when combined with the LPA-based refinement and Jet refinement, the ie configuration performs significantly better on regular graphs than on irregular graphs.

ie-overlay, shown in Figure A.3, generally has a higher solution quality on irregular graphs, except when used with KaMinPar's default refinement strategy based on LPA,where it has a higher solution quality on irregular graphs.

For n2v-overlay, Figure A.4 shows that the rate at which the solution quality improves upon the solutions of the Base and overlay configurations is different depending on the

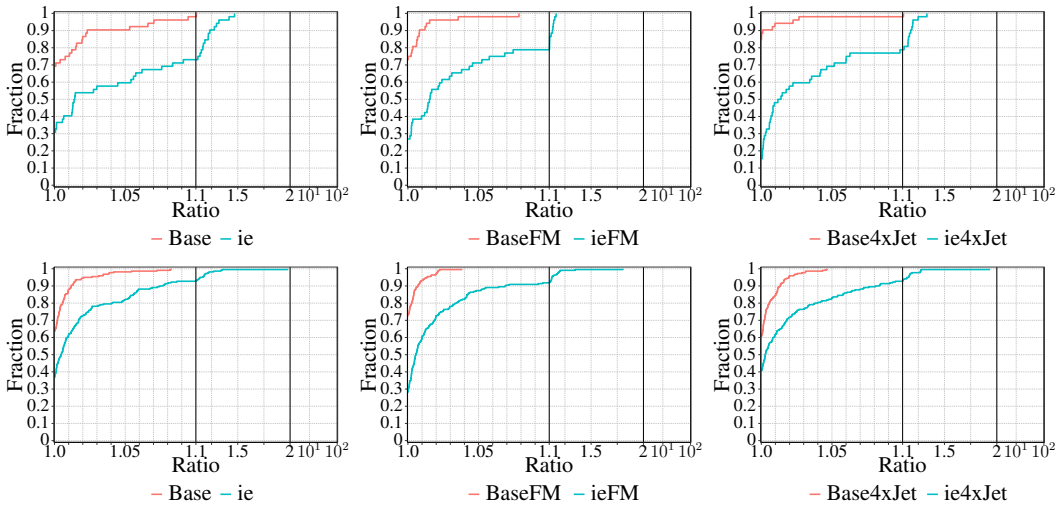


Figure A.2: Irregular graphs on top, regular graphs at the bottom. The performance gap between InstantEmbedding-based LPA and the baseline is smaller on regular graphs than on irregular graphs.

refinement strategy. While n2v-overlay and n2v-overlayFM perform better on irregular graphs, n2v-overlayFM achieves a higher performance on regular graphs.

Figures A.4 and A.3 both show that overlaying improves the solution quality of embedding-based coarsening, with Node2Vec-based coarsening benefiting especially on irregular graphs.

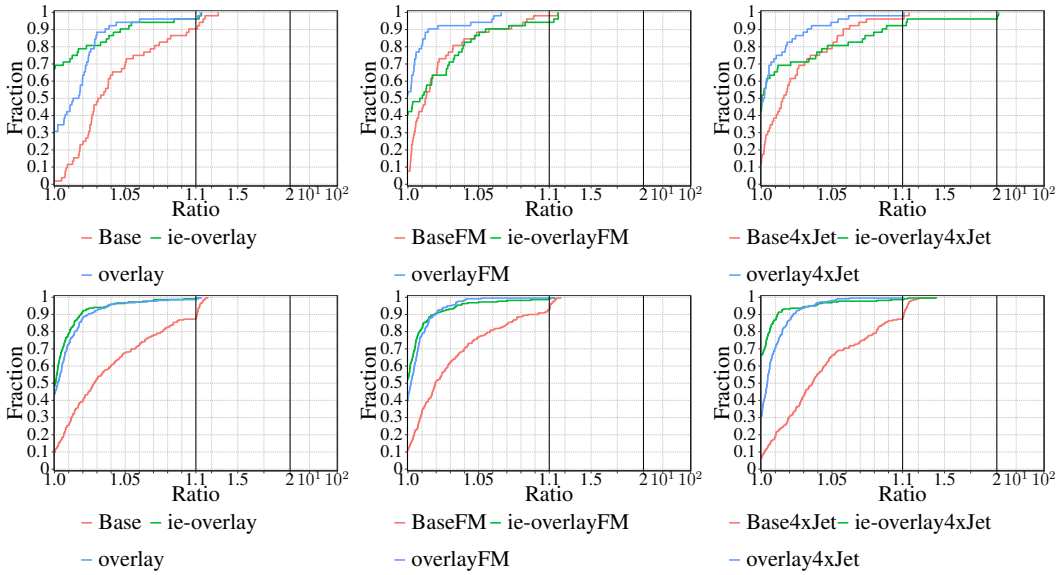


Figure A.3: Irregular graphs on top, regular graphs at the bottom. With overlaying the solution quality of InstantEmbedding-based LPA is better than the baseline and than overlaying without embeddings.

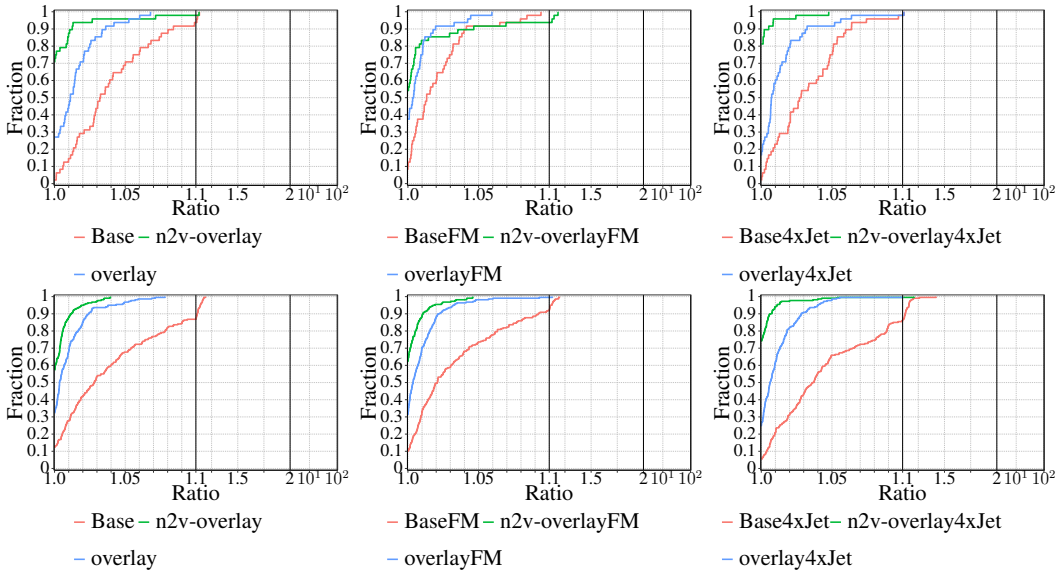


Figure A.4: Irregular graphs on top, regular graphs at the bottom. With overlaying Node2Vec-based LPA achieves the best solution quality on irregular graphs.

Bibliography

[1] Intel Thread Building Blocks. https://www.threadingbuildingblocks.org/.

[2] KaMinPar. https://github.com/KaHIP/KaMinPar.

[3] 9th DIMACS Implementation Challenge: Shortest Paths. https://diag.uniroma1.it/challenge9/download.shtml#benchmark, June 2010.

[4] Konstantin Andreev and Harald Räcke. Balanced graph partitioning. In *Proceedings of the Sixteenth Annual ACM Symposium on Parallelism in Algorithms and Architectures*, pages 120–124, Barcelona Spain, June 2004. ACM. https://dl.acm.org/doi/10.1145/1007912.1007931.

[5] David A. Bader, Henning Meyerhenke, Peter Sanders, Christian Schulz, Dorothea Wagner, and editors. Graph Partitioning and Graph Clustering. In *10th DIMACS Implementation Challenge Workshop*, Georgia Institute of Technology, Atlanta, GA. Contemporary Mathematics 588, 2012-02-13/2012-02-14. American Mathematical Society and Center for Discrete Mathematics and Theoretical Computer Science.

[6] Paolo Boldi, Marco Rosa, Massimo Santini, and Sebastiano Vigna. Layered label propagation: A MultiResolution coordinate-free ordering for compressing social networks. In Sadagopan Srinivasan, Krithi Ramamritham, Arun Kumar, M. P. Ravindra, Elisa Bertino, and Ravi Kumar, editors, *Proceedings of the 20th International Conference on World Wide Web*, pages 587–596. ACM Press, 2011.

[7] Paolo Boldi and Sebastiano Vigna. The WebGraph Framework I: Compression Techniques. In *Proc. of the Thirteenth International World Wide Web Conference (WWW 2004)*, pages 595–601, Manhattan, USA, 2004. ACM Press.

[8] Haochen Chen, Syed Fahad Sultan, Yingtao Tian, Muhao Chen, and Steven Skiena. Fast and Accurate Network Embeddings via Very Sparse Random Projection. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, pages 399–408, Beijing China, November 2019. ACM. https://dl.acm.org/doi/10.1145/3357384.3357879.

[9] Jie Chen and Ilya Safro. Algebraic Distance on Graphs. *SIAM Journal on Scientific Computing*, 33(6):3468–3490, January 2011. http://epubs.siam.org/doi/10.1137/090775087.

[10] Elizabeth D. Dolan and Jorge J. Moré. Benchmarking optimization software with performance profiles. *Mathematical Programming*, 91(2):201–213, January 2002. http://link.springer.com/10.1007/s101070100263.

[11] Charles M. Fiduccia and Robert M. Mattheyses. A Linear-Time Heuristic for Improving Network Partitions. In *19th Conference on Design Automation (DAC)*, pages 175–181, Las Vegas, NV, USA, 1982-06-14/1982-06-16. IEEE.

[12] Thomas M. J. Fruchterman and Edward M. Reingold. Graph drawing by force-directed placement. *Software: Practice and Experience*, 21(11):1129–1164, November 1991. https://onlinelibrary.wiley.com/doi/10.1002/spe.4380211102.

[13] Michael S. Gilbert, Kamesh Madduri, Erik G. Boman, and Siva Rajamanickam. Jet: Multilevel Graph Partitioning on Graphics Processing Units. *SIAM Journal on Scientific Computing*, 46(5):B700–B724, October 2024. https://epubs.siam.org/doi/10.1137/23M1559129.

[14] Lars Gottesbüren, Tobias Heuer, Nikolai Maas, Peter Sanders, and Sebastian Schlag. Scalable High-Quality Hypergraph Partitioning. http://arxiv.org/abs/2303.17679, March 2023.

[15] Lars Gottesbüren, Tobias Heuer, Peter Sanders, Christian Schulz, and Daniel Seemaier. Deep Multilevel Graph Partitioning. https://arxiv.org/abs/2105.02022, 2021.

[16] Aditya Grover and Jure Leskovec. Node2vec: Scalable Feature Learning for Networks. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 855–864, San Francisco California USA, August 2016. ACM. https://dl.acm.org/doi/10.1145/2939672.2939754.

[17] Gaël Guennebaud, Jacob Benoît, et al. Eigen v3. http://eigen.tuxfamily.org/, 2010.

[18] Steve Hamm. Circuit, many parasitics. https://sparse.tamu.edu/Hamm, 2001.

[19] Mohammad Al Hasan and Mohammed J. Zaki. A Survey of Link Prediction in Social Networks. In Charu C. Aggarwal, editor, *Social Network Data Analytics*, pages 243–275. Springer US, Boston, MA, 2011. https://link.springer.com/10.1007/978-1-4419-8462-3_9.

[20] Andrew B. Kahng, Jens Lienig, Igor L. Markov, and Jin Hu. *VLSI Physical Design: From Graph Partitioning to Timing Closure*. Springer International Publishing, Cham, 2022. https://link.springer.com/10.1007/978-3-030-96415-3.

[21] George Karypis and Vipin Kumar. A Software Package for Partitioning Unstructured Graphs, Partitioning Meshes, and Computing Fill-Reducing Orderings of Sparse Matrices.

[22] George Karypis and Vipin Kumar. Parallel multilevel k-way partitioning scheme for irregular graphs. In *Proceedings of the 1996 ACM/IEEE Conference on Super-*

computing, page 35, Pittsburgh Pennsylvania USA, November 1996. IEEE Computer Society. https://dl.acm.org/doi/10.1145/369028.369103.

[23] T. Koch, A. Martin, and S. Voß. SteinLib: An updated library on steiner tree problems in graphs. Technical Report ZIB-Report 00-37, Konrad-Zuse-Zentrum für Informationstechnik Berlin, Takustr. 7, Berlin, 2000. http://elib.zib.de/steinlib.

[24] Thorsten Koch, Alexander Martin, and Stefan Voß. Testset ALUE. https://steinlib.zib.de/showset.php?ALUE.

[25] Lin Lan, Pinghui Wang, Xuefeng Du, Kaikai Song, Jing Tao, and Xiaohong Guan. Node classification on graphs with few-shot novel labels via meta transformed network embedding. *Advances in Neural Information Processing Systems*, 33:16520–16531, 2020.

[26] Jure Leskovec and Andrej Krevl. SNAP Datasets: (Stanford) Large Network Dataset Collection, June 2014.

[27] Hao Li, G.W. Rosenwald, J. Jung, and Chen-ching Liu. Strategic Power Infrastructure Defense. *Proceedings of the IEEE*, 93(5):918–933, May 2005. https://ieeexplore.ieee.org/document/1428007/?arnumber=1428007.

[28] Ping Li, Trevor J. Hastie, and Kenneth W. Church. Very sparse random projections. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 287–296, Philadelphia PA USA, August 2006. ACM. https://dl.acm.org/doi/10.1145/1150402.1150436.

[29] Jens Maue and Peter Sanders. Engineering Algorithms for Approximate Weighted Matching. In Camil Demetrescu, editor, *Experimental Algorithms*, volume 4525, pages 242–255. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007. http://link.springer.com/10.1007/978-3-540-72845-0_19.

[30] Henning Meyerhenke, Peter Sanders, and Christian Schulz. Partitioning Complex Networks via Size-constrained Clustering. http://arxiv.org/abs/1402.3281, March 2014.

[31] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient Estimation of Word Representations in Vector Space. 2013. https://arxiv.org/abs/1301.3781.

[32] Mingdong Ou, Peng Cui, Jian Pei, Ziwei Zhang, and Wenwu Zhu. Asymmetric Transitivity Preserving Graph Embedding. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1105–1114, San Francisco California USA, August 2016. ACM. https://dl.acm.org/doi/10.1145/2939672.2939751.

[33] Lawrence Page. The PageRank citation ranking: Bringing order to the web. Technical report, Technical Report, 1999.

[34] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. DeepWalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD International*

Conference on Knowledge Discovery and Data Mining, pages 701–710, New York New York USA, August 2014. ACM. https://dl.acm.org/doi/10.1145/2623330.2623732.

[35] Ştefan Postăvaru, Anton Tsitsulin, Filipe Miguel Gonçalves de Almeida, Yingtao Tian, Silvio Lattanzi, and Bryan Perozzi. InstantEmbedding: Efficient Local Node Representations. http://arxiv.org/abs/2010.06992, October 2020.

[36] Usha Nandini Raghavan, Réka Albert, and Soundar Kumara. Near linear time algorithm to detect community structures in large-scale networks. *Physical Review E*, 76(3):036106, September 2007. https://link.aps.org/doi/10.1103/PhysRevE.76.036106.

[37] Md. Khaledur Rahman and Ariful Azad. A Comprehensive Analytical Survey on Unsupervised and Semi-Supervised Graph Representation Learning Methods. 2021. https://arxiv.org/abs/2112.10372.

[38] Md. Khaledur Rahman, Majedul Haque Sujon, and Ariful Azad. Force2Vec: Parallel Force-Directed Graph Embedding. In *2020 IEEE International Conference on Data Mining (ICDM)*, pages 442–451, Sorrento, Italy, November 2020. IEEE. https://ieeexplore.ieee.org/document/9338414/.

[39] Dorit Ron, Ilya Safro, and Achi Brandt. Relaxation-Based Coarsening and Multiscale Graph Organization. *Multiscale Modeling & Simulation*, 9(1):407–423, January 2011. http://epubs.siam.org/doi/10.1137/100791142.

[40] Ilya Safro, Peter Sanders, and Christian Schulz. Advanced Coarsening Schemes for Graph Partitioning. *ACM Journal of Experimental Algorithmics*, 19:1–24, February 2015. https://dl.acm.org/doi/10.1145/2670338.

[41] Semih Salihoglu and Jennifer Widom. GPS: A graph processing system. In *Proceedings of the 25th International Conference on Scientific and Statistical Database Management*, pages 1–12, Baltimore Maryland USA, July 2013. ACM. https://dl.acm.org/doi/10.1145/2484838.2484843.

[42] Peter Sanders and Christian Schulz. Engineering Multilevel Graph Partitioning Algorithms. In Camil Demetrescu and Magnús M. Halldórsson, editors, *Algorithms – ESA 2011*, volume 6942, pages 469–480. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011. http://link.springer.com/10.1007/978-3-642-23719-5_40.

[43] Justin Sybrandt, Ruslan Shaydulin, and Ilya Safro. Hypergraph Partitioning With Embeddings. *IEEE Transactions on Knowledge and Data Engineering*, 34(6):2771–2782, June 2022. https://ieeexplore.ieee.org/document/9169850/.

[44] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. LINE: Large-scale Information Network Embedding. In *Proceedings of the 24th International Conference on World Wide Web*, pages 1067–1077, Florence Italy, May

2015. International World Wide Web Conferences Steering Committee. https://dl.acm.org/doi/10.1145/2736277.2741093.

[45] Anton Tsitsulin. Xgfs/node2vec-c. https://github.com/xgfs/node2vec-c, 2020.

[46] Anton Tsitsulin, Davide Mottin, Panagiotis Karras, and Emmanuel Müller. VERSE: Versatile Graph Embeddings from Similarity Measures. In *Proceedings of the 2018 World Wide Web Conference on World Wide Web - WWW '18*, pages 539–548, Lyon, France, 2018. ACM Press. http://dl.acm.org/citation.cfm?doid=3178876.3186120.

[47] Anton Tsitsulin, Marina Munkhoeva, Davide Mottin, Panagiotis Karras, Ivan Oseledets, and Emmanuel Müller. FREDE: Anytime graph embeddings. *Proceedings of the VLDB Endowment*, 14(6):1102–1110, February 2021. https://dl.acm.org/doi/10.14778/3447689.3447713.

[48] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9:2579–2605, 2008.

[49] Kilian Weinberger, Anirban Dasgupta, John Langford, Alex Smola, and Josh Attenberg. Feature hashing for large scale multitask learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 1113–1120, Montreal Quebec Canada, June 2009. ACM. https://dl.acm.org/doi/10.1145/1553374.1553516.

[50] Xiang Yue, Zhen Wang, Jingong Huang, Srinivasan Parthasarathy, Soheil Moosavinasab, Yungui Huang, Simon M Lin, Wen Zhang, Ping Zhang, and Huan Sun. Graph embedding on biomedical networks: Methods, applications and evaluations. *Bioinformatics*, 36(4):1241–1251, February 2020. https://academic.oup.com/bioinformatics/article/36/4/1241/5581350.