

SIMILARITIES AND REPRESENTATIONS
OF GRAPH STRUCTURES

DISSERTATION
ZUR
ERLANGUNG DES DOKTORGRADES (DR. RER. NAT.)
DER
MATHEMATISCH-NATURWISSENSCHAFTLICHEN FAKULTÄT
DER
RHEINISCHEN FRIEDRICH-WILHELMS-UNIVERSITÄT BONN

VON
ANTON TSITSULIN
AUS
MOSKAU, RUSSLAND

BONN, 2020

Erster Gutachter:	Prof. Dr. Emmanuel Müller
Zweite Gutachterin:	Prof. Dr. Petra Mutzel
Fachnahes Mitglied:	Prof. Dr. Heiko Röglin
Fachfremdes Mitglied:	Prof. Dr. Ribana Roscher

Tag der mündlichen Prüfung:	29.04.2021
Erscheinungsjahr:	2021

Angefertigt mit Genehmigung
der Mathematisch-Naturwissenschaftlichen Fakultät
der Rheinischen Friedrich-Wilhelms-Universität Bonn

η δε γνώσις αγάπη γίνεται

St. Gregory Nyssen

ABSTRACT

Graphs are a natural representation for diverse systems ranging from social networks to the Web and brain structure. Even when data is not interconnected explicitly, it is often convenient to convert it into a graph for further analysis. Many tasks involving graphs, such as link prediction, community detection, and classification, rely on various definitions of similarities between nodes in graphs or graphs as a whole. However, such similarities are mostly *implicit*, meaning that objects are not represented by a feature vector in some space. In contrast, modern machine learning methods require *explicit* representations of objects in the Euclidean space. To leverage machine learning powers on graph data, we must find suitable explicit representations of graphs.

This thesis develops efficient algorithms for obtaining expressive, explicit representations of graph-structured data. We focus on the *scalability* of the algorithms, as they must have an ability to process Web-sized graphs to be relevant for practice. *Local* graph algorithms have that ability; we introduce scalable local algorithms for representing nodes, edges, and whole graphs as vectors in the Euclidean space. Studying representations through the lens of the underlying similarity allows us to elucidate previous work and introduce extremely desirable properties to our proposed models. Notably, we introduce the first anytime and the first local algorithms for representing graphs' nodes. For the case of whole graphs, we propose the first representation that empowers multi-scale comparison of graphs and a method for its local approximation. We verify experimentally that our methods do not sacrifice the expressivity of representations for algorithms' scalability. We introduce novel applications of graph analysis and use our methods on massive graphs with billions of nodes.

CONTENTS

1	Introduction and Thesis Overview	1
1.1	Overview and Contributions	3
2	Background and Notation	7
2.1	Notation and Common Symbols	7
2.2	Graphs	8
2.3	Matrix Decompositions	11
2.4	Neural Networks	12
I	Similarities and Representations of Nodes	15
3	Introduction and Related Work	17
3.1	Related Work	18
4	Vertex Similarity Embeddings	25
4.1	A Versatile Node Embedding	27
4.2	Experiments	34
4.3	Summary	44
5	Anytime Node Embeddings	45
5.1	Preliminaries	46
5.2	Anytime node embeddings	48
5.3	Experiments	52
5.4	Summary	60
6	Local Node Representations	61
6.1	Problem Statement	62
6.2	Method	64
6.3	Experiments	69
6.4	Summary	76

7	Node Representations for Clustering	83
7.1	Preliminaries	84
7.2	Method	87
7.3	Experiments	89
7.4	Summary	97
II	Similarities and Representations of Graphs	99
8	Introduction and Related Work	101
8.1	Related Work	102
9	Spectral Graph Similarity	105
9.1	Problem Statement	106
9.2	Network Laplacian Spectral Descriptor	108
9.3	Experiments	114
9.4	Summary	122
10	Learning a Spectral Graph Similarity	123
10.1	Learned Spectral Representations	125
10.2	Experiments	128
10.3	Summary	131
11	Efficient Approximation of Spectral Graph Representations	133
11.1	Preliminaries	134
11.2	Stochastic Lanczos Quadrature	136
11.3	Experiments	144
11.4	Summary	150
12	Spectral Graph Similarities for Comparing Distributions	151
12.1	Related Work	152
12.2	Intrinsic Multi-Scale Distance	154
12.3	Experiments	158
12.4	Summary	165

III Summary	167
13 Summary and Future Work	169
13.1 Future Work	171

LIST OF FIGURES

2.1	A simple graph.	8
2.2	A directed graph with a sink node v	8
2.3	A weighted graph; edge thickness represents its weight.	8
2.4	A graph with two clusters. Its cut is shown in gray.	12
2.5	Information flow of a two-layer GCN. Hidden representations are first updated from the green nodes to the red ones; then, red nodes are used to compute the representation of the central node.	13
3.1	A claw graph.	19
3.2	Example of node2vec’s second-order random walks. Here, we walked from node u to v	22
3.3	A star graph.	22
4.1	Three node properties are highlighted on the same graph. Can a single model capture these properties?	26
4.2	An example similarity matrix and its reconstructions by VERSE and SVD. Karate club graph [Zac77]; we set dimensionality $d = 4$ for both methods.	27
4.3	Ranking performance in terms of NDCG for reconstructing PPR similarity, averaged across nodes in a graph.	31
4.4	Classification performance for (a) different dimensionality and (b) PPR damping factor α	42
4.5	Scalability of different methods.	43
4.6	Visualization of a subset of nodes from CoCit graph with selected conferences: ● VLDB, ● ICDE, ● KDD, ● WWW, and ● NeurIPS. Note that the number of nodes per class is the same for all conferences.	44

5.1	Workflow: <code>FREDE</code> iteratively samples transformed <code>PPR</code> rows, periodically compresses the derived sketch and derives singular values by <code>SVD</code> , and returns an embedding with error guarantees at any time.	51
5.2	Covariance error vs. dimensionality d ; <code>FREDE</code> approaches <code>SVD</code> , which yields optimal covariance error.	55
5.3	Classification performance of sketching algorithms on <code>PPI</code> data wrt. number of walks to compute <code>PPR</code>	55
5.4	Classification performance of <code>FREDE</code> with varying percentage of the graph as input on three datasets.	58
6.1	Required (a) running time and (b) memory consumption to generate a node embedding ($d=512$) based on the edge count of each graph ($ E $), with the best line fit. Our method is over 9,000 times faster than the next fastest baseline (<code>FastRP</code>) and uses over 8,000 times less memory than the next most memory-efficient baseline (<code>VERSE</code>), in the largest graph that these baseline methods can process.	73
6.2	The impact of the choice of ϵ on the quality of the resulting embedding (through the Micro-F1 score), average running time and peak memory increase for the YouTube dataset.	74
6.3	CoCit visualization via <code>UMAP</code> ($d=512$). Research areas (● <code>ML</code> , ● <code>DM</code> , ● <code>DB</code> , ● <code>IR</code>).	75
7.1	Optimization progress of <code>MinCut</code> and <code>DMON</code> on Cora dataset. <code>MinCut</code> optimizes the regularizer, while <code>DMON</code> minimizes its main objective.	88
7.2	Illustration of synthetic data. (a) 4-cluster graph adjacency matrix. (b) Covariance matrix of “matched” features: features that are clustered according to the graph clusters. (c) Covariance matrix of “nested” features: features that are clustered by incomplete nesting of the graph clusters. (d) Covariance matrix of “grouped” features: features that are clustered by incomplete grouping of the graph clusters.	92

7.3	Synthetic results on the ADC-SBM model with 6 different scenarios described in Table 7.3. We observe that DMON significantly outperforms other neural graph pooling method baselines. . . .	94
7.4	Synthetic results on the ADC-SBM model with 6 different scenarios described in Table 7.3. We observe that DMON leverages information from both graph structure and node attributes. . . .	94
9.1	Taylor expansion for NETLSD approximation: (a) Relative approximation error of normalized h_t for Erdős–Rényi random graphs, varying time scale t , and (b) the heat trace approximation by two Taylor terms and 100 eigenvalues on a random SBM graph.	110
9.2	The diagonal of \mathbf{H}_t at different scales on the Karate club graph; at a large scale, the field reflects node centrality.	112
9.3	Visualization of the diagonal in the heat kernel matrix \mathbf{H}_t for the pointed vertex at scale $t = 0.3$	112
9.4	Relative error in spectrum computation of $h(G)$, averaged across 2085 graphs.	113
9.5	Two most dissimilar graphs by $h(G)/h(\bar{K})$ across all small-size graphs in datasets used. Communities are colored.	121
9.6	Time to compute 300 eigenvalues on both ends of the spectrum and the approximation of $h(G)$	121
10.1	Different regions of the spectrum have a different impact on the classifier co-trained with the SGR^G . The color map shows the gradient magnitude of the classifier output with respect to the input spectrum visualized in increasing order from left to right, averaged on 600 graphs. Top: SBM; bottom: Erdős–Rényi. . . .	125
10.2	Relative L_1 error in neural representation as a function of a number of computed eigenvalues.	127
11.1	Errors (solid) and error bounds (dotted) for the approximation of matrix exponential action with varying temperature t	141

11.2 Trace estimation errors (solid) and error bounds (dotted) for: (*left*) the number of Lanczos steps m with fixed number of random vectors $n_v = 100$; (*right*) the number of random vectors n_v in Hutchinson estimator with fixed number of Lanczos steps $m = 10$. Lines correspond to varying temperature t 142

11.3 Variance of the trace estimate 142

11.4 SLAQ offers over 200× reduction in average error for VNGE over techniques proposed in [CWLR19] and over 30× improvement over the linear approximation from Chapter 9. 146

11.5 SLAQ offers 22× reduction in average error for NETLSD over linear approximation and 250× over Taylor expansion. 146

11.6 Number of nodes and edges of random Erdős–Rényi graphs does not affect SLAQ’s approximation accuracy. 146

11.7 SLAQ approximation of — NETLSD and - - - - VNGE for Wikipedia graphs across time. Changes that are not explained by local edge differences highlighted in gray. 147

11.8 Parameter sensitivity of SLAQ in terms of approximating NETLSD with (a) different number of starting vectors n_v and (b) different number of Lanczos steps s . Error averaged across 73 graphs from the Network Repository. 148

12.1 Two distributions having the same first 3 moments, meaning FID and KID scores are close to 0. 151

12.2 (a) IMD distances between language pairs for unaligned Wikipedia word embeddings; (b) distances from the simple English Wikipedia visualized for IMD, FID, and KID; (c) pairwise FID distances; (d) pairwise KID distances. We consider 16 languages: Polish, Russian, Greek, Hungarian, Turkish, Arabic, Hebrew, English, Simple English, Swedish, German, Spanish, Dutch, Portugese, Vietnamese, and Waray-Waray. 159

12.3 Comparison of IMD and PIP loss on word embeddings of different dimension. Note how IMD detects subtle changes in the dimensionality. 160

12.4	Training progression in terms of accuracy (dotted) and IMD (solid) on CIFAR-10 and CIFAR-100 datasets for VGG-16 and ResNet-20, with respect to VGG-16.	161
12.5	Values of IMD across convolutional layers of the VGG-16 network on CIFAR-10 and CIFAR-100 datasets.	161
12.6	FID, KID and IMD on the CIFAR-10 dataset with Gaussian blur.	162
12.7	Plotting the normalized heat trace allows interpretation of medium- and global-scale structure of datasets.	164
12.8	Stability and scalability experiment: (<i>left</i>) stability of FID, KID and IMD wrt. sample size on CIFAR-10 and CIFAR-100 dataset; (<i>right</i>) scalability of FID, KID and IMD wrt. sample size on a synthetic dataset.	165

LIST OF TABLES

1.1	An overview of the thesis contributions.	3
2.1	Common symbols and notation.	7
4.1	Comparison of neural embedding methods in terms of the bounded degree (\mathcal{O}_k) and worst-case (\mathcal{O}) time and space complexity, assuming sparse graphs.	32
4.2	Dataset characteristics: number of vertices $ V $, number of edges $ E $; number of node labels $ \mathcal{L} $; average node degree; modularity \mathcal{Q} [Newo6b]; density defined as $ E /\binom{ V }{2}$	35
4.3	Vector operators used for link-prediction task for each $u, v \in V$ and corresponding embeddings $\mathbf{a}, \mathbf{b} \in \mathbb{R}^d$	37
4.4	Link prediction results on the CoAuthor coauthorship graph. Best results per method are underlined.	37
4.5	Link prediction results on the VK coauthorship graph. Best results per method are underlined.	38
4.6	Multi-class classification results in CoCit dataset.	39
4.7	Multi-class classification results in VK dataset.	39
4.8	Multi-label classification results in YouTube dataset.	39
4.9	Multi-class classification results in Orkut dataset.	39
4.10	Node clustering results in terms of NMI.	40
4.11	Node clustering results in terms of modularity.	41
4.12	Graph reconstruction % for all datasets.	41
5.1	Dataset characteristics: number of vertices $ V $, number of edges $ E $; number of node labels $ \mathcal{L} $; average node degree; density defined as $ E /\binom{ V }{2}$	53
5.2	Micro-F1 classification, PPI data.	56
5.3	Micro-F1 classification, POS data.	56
5.4	Micro-F1 classification, BlogCat data.	56
5.5	Micro-F1 classification, CoCit data.	57

5.6	Micro-F1 classification, Flickr data.	57
5.7	Micro-F1 classification, YouTube data.	57
5.8	Link prediction accuracy, CoAuthor data.	58
5.9	Link prediction accuracy, VK data.	58
6.1	Dataset attributes: size of the vertex set $ V $, edge set $ E $, labeled vertices $ S $	71
6.2	Average Micro-F1 classification scores and confidence intervals.	74
6.3	Average ROC AUC scores and confidence intervals for the link prediction task.	75
6.4	Classification micro and macro F1-scores for PPI.	77
6.5	Classification micro and macro F1-scores for BlogCat.	78
6.6	Classification micro and macro F1-scores for CoCit.	79
6.7	Classification micro and macro F1-scores for Flickr.	80
6.8	Classification micro and macro F1-scores for YouTube.	81
6.9	Temporal link-prediction ROC AUC scores for CoAuthor. For each method, we highlight the aggregation function that consistently performs good on all datasets.	82
7.1	Comparison of MinCutPool with using only its orthogonality regularization in terms of graph conductance \mathcal{C} , modularity \mathcal{Q} , and NMI with ground-truth labels.	88
7.2	Dataset statistics.	89
7.3	Synthetic ADC-SBM benchmark scenarios.	93
7.4	Results on four datasets from [SNB ⁺ 08] in terms of graph conductance \mathcal{C} , modularity \mathcal{Q} , NMI with ground-truth labels, and pairwise F1 measure. We group the methods into three categories: baselines using only one aspect of data, neural representation learning, and neural graph pooling methods. We highlight best neural method performance.	95

7.5	Results on four datasets from [SMBG18] in terms of graph conductance \mathcal{C} , modularity \mathcal{Q} , NMI with ground-truth labels, and pairwise F1 measure. We group the methods into three categories: baselines using only one aspect of data, neural representation learning, and neural graph pooling methods. We highlight best neural method performance.	96
9.1	Dataset properties.	114
9.2	Classification ROC AUC in detecting whether a graph is real. . .	116
9.3	Accuracy in 1-NN classification.	116
9.4	Accuracy of 1-NN classification on Reddit-L.	118
9.5	Accuracy in detecting graphs with communities.	119
9.6	Accuracy in detecting graphs with communities, Poisson distribution of graph size.	119
9.7	Accuracy in detecting graphs with communities, Uniform distribution of graph size.	120
10.1	Dataset characteristics: number of graphs $ \mathcal{G} $, number of labels $ L $, avg. number of vertices $\overline{ V }$, avg. number of edges $\overline{ E }$, avg. global clustering coefficient \overline{CC}	128
10.2	Graph classification accuracy on biochemical (top) and social (bottom) graph collections. Best results with representations are highlighted.	129
10.3	Normalized mutual information of clustering on representations. Higher numbers indicate better performance. Best results on each dataset are highlighted in green.	130
10.4	Runtime for the creation of a (dis-)similarity matrix across all graphs from the Reddit-5k dataset evaluated on a single core. Representation computation time for SGR is indicated in two components, spectrum computation + neural network inference. On the datasets, last two rows combined yield total time taken. .	131
11.1	Characteristics of large graphs used in this work: number of vertices $ V $, number of edges $ E $; average node degree; density defined as $ E /\binom{ V }{2}$	145

11.2	Characteristics of dynamic graphs: total number of vertices $ V $, total number of edges $ E $; number of timestamps $ T $; average incoming edges per timestamp $ E / T $	145
11.3	Properties of the graph classification datasets used: number of graphs $ G $; number of labels $ Y $; minimum, average, and maximum number of nodes in graph collection.	145
11.4	1-Nearest neighbor graph classification performance on 4 datasets with <code>VNGE</code> and <code>NETLSD</code> . Exact computation results are in bold. Approximations that are close to or better than the exact metric computation are highlighted in green.	148
11.5	Running time (in seconds) of different approximation techniques and <code>SLAQ</code> for <code>VNGE</code> on large graphs.	148
12.1	<code>IMD</code> agrees with <code>FID</code> and <code>KID</code> across varying datasets for <code>GAN</code> evaluation.	163

EVER-INCREASING capabilities of information processing systems open the possibility to collect and analyze vast amounts of complex data, allowing for better-informed decision making. Traditionally, experts analyze the collected data via the *Knowledge discovery in databases* (KDD) process [HPK01] that proposes to *mine data*¹ for knowledge that can empower human decision making. Alternatively, mining and expert decision making can be merged into a single machine learning model to make accurate decisions faster.

1. In the KDD process, *data mining* is the process of extracting interesting patterns from data.

Both the human-in-the-loop KDD process and end-to-end machine learning require a suitable representation of data to work effectively. In the field of data mining, manual *feature engineering* allows constructing a view of data that allows human experts to analyze it. Conversely, machine learning approaches try to forgo manual feature engineering, which is considered painstakingly difficult, in favor of data-driven *representation learning*, which leverages the hidden symmetries and similarities in data. Learning useful representations is often harder than engineering them—the efforts shift to the engineering of inductive biases [Mit80] of machine learning models.

These two approaches to feature creation inform and advance each other: for example, it is sometimes possible to construct an optimal solution to a learning problem manually. We can then devise approximation algorithms with better algorithmic properties than their learning-based counterparts. Likewise, learning methods benefit from incorporating domain knowledge.

From all the means of representing complex interconnected data, graphs have a unique appeal of the apparent simplicity of their parts—after all, they are merely sets of nodes and edges—and the richness of the information they can model. Graphs provide a powerful abstraction for modeling various types of relations between objects in physical, biological, and social systems. Sometimes, data has a graph structure *naturally*²; other times, it is convenient to convert the data to a graph to explore its intrinsic structure.

2. For example, the social network of Facebook is a natural graph.

The benefits of having the data as a graph are manifold; for example, we can quickly access the local neighborhood of each point. Local graph algorithms provide major efficiency improvements over their global counterparts [Lin87]. The local graph modeling paradigm is one of the few that allows building algorithms that process billions of data points. In contrast, it is notably hard to develop local algorithms for dealing with tabular data, as there is no natural definition of points' local neighborhood.

Graphs also provide us with a natural way of thinking about their *multi-scale* structure ranging from the microscopic ego-network structure of nodes to mesoscopic clusters³ to the overall macroscopic connectivity patterns. Diverse applications require analyzing graphs on different structural scales. While a sociologist classifies friends of a person into social circles—a purely local task, a biologist studies global activity patterns in brain networks. Therefore, *scale-adaptive* methods—ones that can focus the analysis on various structural scales—are oftentimes desirable for real-world applications. It is crucial to distinguish efficient local methods from the myopic single-scale local graph analysis. The very best local algorithms can provide a global view of the graph.

3. Clusters in graphs are also called *communities*.

Low-dimensional features are essential for taming high-dimensional graph-shaped data. However, expressive representations of graphs and their parts are hard to come by, and scalable algorithms are even rarer, meaning it is hard to analyze large real-world graphs. Furthermore, existing solutions dismiss the multi-scale nature of graph data, while diverse applications of graph-based analysis require *versatile* algorithms for creating representations. Another plague of modern representation learning methods is a lack of any theoretical guarantees on the convergence or the solution they produce.

This dissertation focuses on the scalable and principled algorithms for constructing versatile multi-scale representations of graphs. In particular, we focus on low-dimensional node- and graph-level representations suitable for machine learning pipelines and interpretable human decisions. We provide the theoretical foundations, error analysis of our algorithms, and provably scalable methods that efficiently exploit the sparsity and locality of graph data.

1.1 OVERVIEW AND CONTRIBUTIONS

This thesis is organized into two main parts: Part **I** focuses on node-level representations, while Part **II** discusses graph-level ones. Here we present an overview of major contributions and the general structure of the thesis. Table 1.1 presents a bird’s-eye view of our contributions. We group them into three categories for the *challenges* we address: (i) expressivity of our methods for solving tasks from different domains, (ii) theoretical foundations and guarantees that ensure excellent performance of our methods in all conditions, and (iii) scalability that allows applying our methods to any existing graph.

<i>modality</i>	<i>challenge</i>		
	Expressivity	Theory	Scalability
Nodes	Chapters 4, 7	Chapters 5, 6	Chapter 6
Graphs	Chapters 9, 10	Chapters 9, 11, 12	Chapter 11

Table 1.1: An overview of the thesis contributions.

1.1.1 PART I: REPRESENTATIONS AND SIMILARITIES OF NODES

In the first part of this thesis, we study versatile methods for constructing representations of graphs’ nodes in an unsupervised fashion. We begin by introducing the problem and discussing the related work in Chapter 3. As our first contribution, we introduce our neural network-based method, `VERSE`, in Chapter 4. In contrast to existing approaches, `VERSE` allows the analyst to use any similarity measure between nodes to learn node representations tailored to the particular task. For similarity measures amenable to sampling, we introduce an efficient learning procedure linear in the number of nodes in the input graph, superseding state-of-the-art methods in space and time efficiency. Furthermore, we provide additional analysis of heuristic solutions from the previous work and show that the similarity used there implicitly corresponds closely to the well-studied PageRank similarity measure [PBMW99].

While `VERSE` is able to capture diverse similarities empirically, it does not have guarantees on the correctness of the final solution. Moreover, it lacks any guarantees on the representations during the learning process: ultimately, we do not know when the representations are good enough. To address this problem, as our second contribution, in Chapter 5, we introduce `FREDE`, an

4. Anytime algorithm returns a valid solution even when interrupted [Zil96].

*anytime*⁴ algorithm for creating node embeddings with error guarantees. As its core component, covariance sketching, approximates only the relevant part of the solution, `FREDE` is a step forward from the matrix factorization algorithms [QDM⁺18]. In contrast, `FREDE` provides its error guarantees at every execution step. By being frugal about its computations, `FREDE` obtains a valid embedding from only seeing a subset of nodes. Experimentally, `FREDE` only needs to process a small fraction of nodes to produce an embedding comparable with state-of-the-art methods.

In `FREDE`, the final representation of the node algorithmically depends on other nodes' representations, hindering the method's scalability. To overcome this limitation, we introduce `SnapEmbed`, the first local node embedding algorithm, in Chapter 6. We build on the theoretical foundations of `FREDE` to bridge structural and positional node embeddings. In `SnapEmbed`, the representation of a node depends solely on the local neighborhood structure, allowing efficient distributed computation. This property allows us to apply `SnapEmbed` on graphs of unprecedented size. A US patent application was filed based on the approach.

None of the described methods can work on graphs where nodes have attributes attached to them. We address this issue in Chapter 7, where we leverage the machinery of Graph Neural Networks (GNN) to introduce `DMON`, a method that learns node representations by clustering attributed graphs. Unlike reconstruction-based neural network architectures, `DMON` uses a novel clustering objective tailored to real-world graphs. This objective allows `DMON` to scale linearly with the number of edges in a graph. We experimentally confirm its superior performance on a wide range of real and synthetic data.

1.1.2 PART II: REPRESENTATIONS AND SIMILARITIES OF GRAPHS

In the second part of this dissertation, we focus on compact representations of graphs' structure. We provide the necessary background and discuss the related work in Chapter 8. As our first contribution, in Chapter 9, we introduce `NETLSD`, a geometric graph representation that is able to capture the multi-scale structure of graphs. First of its kind, `NETLSD` lower-bounds the spectral Gromov–Wasserstein distance without prior graph alignment. We propose an effective approximation technique to use `NETLSD` for graphs with

millions of nodes. We show that `NETLSD` is competitive with state-of-the-art representations in several tasks, including graph classification.

There is no one-size-fits-all solution for graph representations, just as for representation learning in general [Wol96]—some tasks are inherently more local or more global. To address this issue, in Chapter 10, we introduce `SGR`, the first *self-supervised*⁵ learning approach for creating tailored graph representations. We propose two tasks for self-supervised learning of graph representations: learning to predict a synthetic graph model and learning to predict local clustering coefficients of graphs. By learning to solve synthetic tasks, `SGR` adapts the representations to emphasize features that transfer to real-world tasks. In the experimental evaluations, `SGR` yields consistent improvements over its non-learned counterparts.

While the approximation technique we introduce in Chapter 9 yields consistent performance improvements over the naïve computation, graphs with billions of nodes and edges are still out of reach. To push the scalability limits, Chapter 11 introduces `SLAQ`, a technique for approximating several spectral measures, including `NETLSD`. On a diverse set of real-world graphs, `SLAQ` outperforms existing techniques by two orders of magnitude on average in approximation quality while being comparable time-wise. With `SLAQ`, we scale graph comparison to the most massive graphs, processing graphs with billions of nodes and edges on a single machine in a matter of hours.

In Chapter 12, we study the comparison of unaligned distributions as an application of the proposed representation-based graph similarity. Comparing samples from unknown underlying distributions is a common problem for the field of generative models [Bor19]. We study *intrinsic*⁶ geometry of the data via its graph representation, and propose `IMD`, an intrinsic multi-scale distance between samples that could even be lying in spaces of different dimensionality. We demonstrate several applications where `IMD` is competitive with methods leveraging extrinsic geometry of the data, and several with data lying in different spaces, where `IMD` provides unique insights.

1.1.3 PART III: SUMMARY

We conclude this dissertation by summarizing our core contributions and discussing the open challenges in the area of graph representations.

5. Self-supervised methods learn unsupervised representations using supervised tasks with artificial labels.

6. Intrinsic properties of surfaces are those that are independent of the surface embedding.

BACKGROUND AND NOTATION

In this chapter, we establish the notions and definitions that are used in this dissertation and review the necessary background. There are many different views on graphs, and this section reflects that—we draw inspiration from many different fields, including numerical linear algebra, differential geometry, and machine learning.

2.1 NOTATION AND COMMON SYMBOLS

Throughout the work, we use bold capital letters $\mathbf{A}, \mathbf{B}, \mathbf{C}, \dots$ for matrices, lowercase bold letters for vectors $\mathbf{a}, \mathbf{b}, \mathbf{c}, \dots$, calligraphic letters \mathcal{M}, \mathcal{N} for manifolds, and lowercase a, b, c, \dots for scalars and functions. Some of the notable symbols and exceptions to these rules are presented in Table 2.1 below.

Symbol	Description
V	a set of nodes, $ V = n$
E	a set of edges, $E \subseteq (V \times V)$, $ E = m$
G	a graph $G = (V, E)$
\mathcal{G}	a collection of graphs $\{G_0, \dots, G_n\}$
$\mathcal{N}(v)$	the set of neighbors of the node v , $\mathcal{N}(v) = \{\{u, v\} \in E\}$
$\text{deg}(v)$	the degree of the node v , $\text{deg}(v) = \mathcal{N}(v) $
\mathbf{I}	an identity matrix, $\mathbf{I}_{ii} = 1$
\mathbf{A}	an $n \times n$ adjacency matrix, $\mathbf{A}_{ij} \in \mathbb{R}$
\mathbf{D}	an $n \times n$ degree matrix, $\mathbf{D}_{ii} = \sum_j \mathbf{A}_{ij}$
\mathbf{L}	an $n \times n$ Laplacian matrix, $\mathbf{L} = \mathbf{D} - \mathbf{A}$
\mathcal{L}	an $n \times n$ <i>normalized</i> Laplacian matrix, $\mathcal{L} = \mathbf{D}^{-1/2} \mathbf{L} \mathbf{D}^{-1/2}$
\mathbf{A}^\top	a transpose of a matrix \mathbf{A} , $\mathbf{A}_{ij}^\top = \mathbf{A}_{ji}$
$\text{tr}(\mathbf{A})$	trace of a matrix \mathbf{A} , $\text{tr}(\mathbf{A}) = \sum_i \mathbf{A}_{ii}$
$\mathbf{M} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top$	singular value decomposition of a matrix \mathbf{M}
$\mathbf{M} = \mathbf{Q}\mathbf{\Lambda}\mathbf{Q}^\top$	eigendecomposition of a symmetric matrix \mathbf{M}
λ_i	eigenvalues

Table 2.1: Common symbols and notation.

2.2 GRAPHS

1. An old-fashioned term for simple graphs is a *line graph* [HN53].

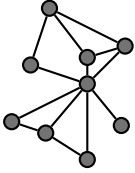


Figure 2.1: A simple graph.

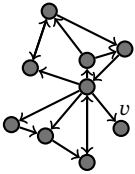


Figure 2.2: A directed graph with a sink node v .

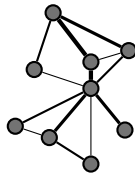


Figure 2.3: A weighted graph; edge thickness represents its weight.

We begin with a definition of a *simple*¹ graph, followed by a short discussion about different types of graphs. Then, we review the properties and results used in the dissertation later on.

Definition 1. A simple graph G is given by its node set V and edge set E of unordered pairs of nodes $E \subseteq V^2 \setminus \Delta_V$,

where $V^2 = V \times V$ is the Cartesian product of V with itself and Δ_V is the diagonal subset of V (the set of pairs $\{v, v\} \in V^2$). We often refer to the size of graphs in terms of their total number of nodes $|V| = n$ and edges $|E| = m$. Nodes in graphs are alternatively called *vertices*; edges are sometimes referred to as *links*. Figure 2.1 presents an example of a simple graph.

A *directed* graph is then one where the edge set consists of ordered pairs of nodes. Many models that work perfectly well on simple graphs break on directed graphs because of the existence of *sink nodes* (illustrated in Figure 2.2). If we model the propagation of information on a directed graph, sink nodes endlessly amass it, causing numerical problems.

When edges represent a non-binary relation, e.g., the number of messages between people in a social network, it is convenient to model them as edge weights $w_{uv} \forall (u, v) \in E$. Graphs with weights associated with edges are called *weighted graphs* (Figure 2.3). It is easier to adapt models designed for simple graphs to the case when weights are non-negative: $w_{uv} \in \mathbb{R}^+$.

2.2.1 MATRICES ASSOCIATED WITH GRAPHS

We can use *adjacency matrices* to model simple, directed, and weighted graphs. Given any sort of graph, we call two vertices u, v adjacent if the edge set E contains the pair $\{u, v\}$. For an ordering of vertices (v_1, \dots, v_n) , an adjacency matrix \mathbf{A} of a graph is a square $n \times n$ matrix whose elements \mathbf{A}_{ij} are 1 if the edge $\{v_i, v_j\}$ exists in E and 0 otherwise. For weighted graphs, the convention is that \mathbf{A}_{ij} equals the weight $w_{v_i v_j}$ of the edge. Adjacency matrices for undirected graphs are symmetric, and their diagonal is always zero.

Importantly for both storage and computation, the adjacency matrix is *sparse*, meaning there are far more zeros than ones. Storage-wise, we may only store the positions and values of the nonzero elements. The benefits of

sparse representations are best seen in large graphs since the number of edges per node typically does not grow with the size of the graph. For example, one could only add 5,000 friends in the Facebook social network, while a median person is acquainted with mere 550 people [DGM⁺11]. For all of the estimated $n = 3$ billion people using Facebook, dense representation of the adjacency matrix would require n^2 bits or approximately 10^6 gigabytes, and only² $64 * 550n + n$ bits (assuming 8-byte integers to index the nodes) or far more manageable 13 terabytes in the sparse format. Even more efficient compression schemes [BV04, CKL⁺09] leverage structural irregularities of graphs to achieve the compression level of a single byte per edge.

2. Assuming Facebook users befriend their every acquaintance.

Computationally, zeros are neutral elements for addition and absorbing elements for multiplication, meaning that adding or multiplying sparse matrices is much faster than their dense counterparts. The most common operations for matrix-based analysis of graphs are matrix-vector and matrix-matrix multiplications. Sparse matrix-vector multiplication (SpMV) requires $\mathcal{O}(m)$ operations in the worst case. Multiplying a sparse $n \times n$ matrix by a dense $n \times k$ matrix takes $\mathcal{O}(mk)$ operations. As for the sparse-to-sparse multiplication (SpGEMM), the complexity for square matrices with m_1 and m_2 nonzero elements is $\mathcal{O}(m_1 m_2)$ average- and $\mathcal{O}(m_1 n)$ worst-case; efficient algorithms are an active research direction.

In the thesis, we extensively use a class of matrices called graph Laplacians. Below, we define two common graph Laplacians for undirected graphs. We refer to [HALo7] for an extensive introduction to the topic.

Definition 2 ([HALo7]). Given two Hilbert spaces $\mathcal{H}(V)$ and $\mathcal{H}(E)$ and the difference operator $d: \mathcal{H}(V) \rightarrow \mathcal{H}(E)$, the graph Laplacian $\Delta: \mathcal{H}(V) \rightarrow \mathcal{H}(V)$ is defined as $\Delta = d^*d$.

The difference operator is essentially the gradient of a scalar function f on a graph; there are two popular choices. By setting $(df)(E_{uv}) = f(v) - f(u)$ we get the combinatorial Laplacian. This formulation does not consider that real-world graphs have an uneven degree distribution [FFF99]. We can normalize the functions by the degrees of nodes $(df)(E_{uv}) = \frac{f(v)}{\sqrt{\deg(v)}} - \frac{f(u)}{\sqrt{\deg(u)}}$, and get the Normalized Laplacian. Both Laplacians can also be represented as positive semidefinite matrices: $\mathbf{L} = \mathbf{D} - \mathbf{A}$ for the combinatorial Laplacian and $\mathcal{L} = \mathbf{D}^{-1/2} \mathbf{L} \mathbf{D}^{-1/2} = \mathbf{I} - \mathbf{D}^{-1/2} \mathbf{A} \mathbf{D}^{-1/2}$ for the normalized one.

Several results tie combinatorial insights about graphs to the algebraic properties of their Laplacians. For example, the number of the connected components in a graph equals the multiplicity of the eigenvalue 0; the first nonzero eigenvalue³ provides insights into the clusterability of graphs [Abb18]. There are many more similarly exciting results; overall, the study of graphs through the lens of matrix analysis is a potent research direction.

3. Also called Fiedler eigenvalue or the algebraic connectivity [Fie73].

2.2.2 PAGERANK

One of the most common tasks in graphs analysis is to determine which nodes are important. This importance of a node is called node *centrality*, first studied in sociology [Fre78]. Besides social sciences, node ranking finds numerous applications in bioinformatics and Web search.

The PageRank system [PBMW99] developed for the Google™ search is the most widely known approach to ranking in graphs. PageRank provides a concise mathematical formulation for a simple idea—important nodes link to other important nodes. If we denote the importance of the node v as s_v , then we can naturally define the importance as

$$s_v = \sum_{\{u,v\} \in E} \frac{s_u}{\deg(v)}. \quad (2.1)$$

In the matrix form, we would try to solve $\mathbf{D}^{-1}\mathbf{A}^T\mathbf{s} = \mathbf{s}$, where $\mathbf{D}^{-1}\mathbf{A}^T$ is a column-stochastic matrix called the transition matrix. Each column of that matrix contains the probabilities of picking outgoing edges of a node randomly. Unfortunately, this formulation may not yield a unique solution.

PageRank fixes the issue by introducing a *preference vector* \mathbf{v} and a teleportation probability α . The preference vector defines a distribution⁴ over the nodes that the random surfer can teleport to with the probability α ⁵. Intuitively, the surfer now has a chance to teleport to a position specified by the preference vector without considering the graph structure. For ranking all nodes in a graph, it makes sense to set $\mathbf{v} = 1/n$ for all nodes to be equiprobable. This subtle change immediately fixes the indeterminism, and leads to the famous PageRank formulation:

$$(1 - \alpha)\mathbf{P}\mathbf{s} - \alpha\mathbf{v} = \mathbf{s}. \quad (2.2)$$

4. Meaning $\mathbf{1}^T\mathbf{v} = 1$.

5. Commonly, $\alpha = 0.15$. Whether there exists a principled way to pick α is an open question.

This equation provides us with an effective iterative way of computing PageRank: iteratively set $\mathbf{s}_i = (1 - \alpha)\mathbf{P}\mathbf{s}_{i-1} - \alpha\mathbf{v}$ starting from $\mathbf{s}_0 = 1/n$. This algorithm corresponds to the power iteration solution of the eigensystem defined in Equation 2.2. It converges linearly with the rate $1 - \alpha$.

An important variation of PageRank is its local version called *personalized PageRank* (PPR). Instead of setting the teleportation vector \mathbf{v} to be equal for all nodes, we can personalize it to some node v by setting \mathbf{v} to 0 in all positions except being 1 at v ⁶. Then, by applying Equation 2.2 to our personalized \mathbf{v} we get the importance of other nodes from the v 's perspective (alternatively, PPR is a measure of similarity between nodes). Notably, PPR is local—it can be well approximated using only its local neighborhood [ACL07].

6. This vector is also called a *one-hot encoding* of v .

2.3 MATRIX DECOMPOSITIONS

As we have seen in the previous sections, matrices provide a solid foundation for analyzing graphs. Decomposition methods seek to break down a matrix into a product of matrices structured in some particular way. In the analysis of graphs, singular value decomposition (SVD) and eigendecomposition are the most prominent ones.

Singular value decomposition exists for any real matrix; it decomposes any matrix $\mathbf{M} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$ into a product of two orthogonal⁷ matrices \mathbf{U} and \mathbf{V} , and a diagonal matrix $\mathbf{\Sigma}$ containing the singular values of \mathbf{M} . This decomposition is special for many reasons; for example, it provides an easy way to obtain the best possible rank- k approximation of any matrix by simply selecting the first k columns of \mathbf{U} and \mathbf{V} , and first k values from $\mathbf{\Sigma}$. Elements of $\mathbf{\Sigma}$, namely $\sigma_0, \dots, \sigma_n$, offer a way to quantify the “approximability” of a matrix:

7. Meaning $\mathbf{U}\mathbf{U}^T = \mathbf{I}$ and $\mathbf{U}^{-1} = \mathbf{U}^T$.

$$\min_{\mathbf{A}_k} \|\mathbf{A} - \mathbf{A}_k\|_2 = \sigma_{k+1}, \quad \min_{\mathbf{A}_k} \|\mathbf{A} - \mathbf{A}_k\|_F = \sqrt{\sigma_{k+1}^2 + \dots + \sigma_n^2}, \quad (2.3)$$

where \mathbf{A}_k is any rank- k real matrix. While it takes a long $\mathcal{O}(n^3)$ time to compute SVD of a square dense matrix, there are several randomized algorithms approximating the first k columns of \mathbf{U} in almost-linear time [HMT11, MM15].

Eigendecomposition is of paramount importance for the graph analysis. Spectral graph theory studies properties of graphs expressed by their eigenvalues and eigenvectors; we refer to [Chu97] for an extensive treatment.

8. This is the reason why undirected graphs are preferred by people who enjoy linear algebra.

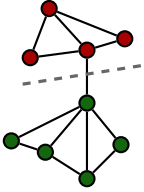


Figure 2.4: A graph with two clusters. Its cut is shown in gray.

9. A cut is number of edges that we cut with the partition.

10. Alternative viewpoint is that the key to success was in engineering structural inductive biases.

All real symmetric matrices are amenable to an eigendecomposition⁸ $\mathbf{M} = \mathbf{Q}\mathbf{\Lambda}\mathbf{Q}^{-1}$ into a product of an orthogonal matrix \mathbf{Q} and a diagonal matrix $\mathbf{\Lambda}$ containing the eigenvalues λ of \mathbf{M} . The eigenvalues alone contain a lot of information about the structure of graphs. In the second part of this thesis, we explore the eigenvalue-based graph descriptors and showcase their usefulness. One of the peculiar properties of the eigendecomposition is that it allows us to effortlessly compute matrix functions: $f(\mathbf{M}) = \mathbf{Q}f(\mathbf{\Lambda})\mathbf{Q}^{-1}$.

Some of the most exciting results in spectral graph theory are connected with the problem of graph clustering. Intuitively, the task is to group nodes such that the nodes in the same group are better connected to ones inside the group than to the ones outside; Figure 2.4 provides an example. The second eigenvalue of Laplacian matrices of graphs is directly related to the visibility of clusters [VLo7]. A simple algorithm to partition a graph into 2 subsets works as follows: first, compute the second eigenvector of the graph; then, assign all vertices with the positive eigenvector value to one class. This algorithm produces an optimal solution for several cut-based⁹ metrics [SMoo].

2.4 NEURAL NETWORKS

Recent advances in neural architecture engineering¹⁰ coupled with the drastic increase in the computational capabilities of massively parallel co-processors opened a possibility to train large models on convenience hardware. Most prominent in the fields of computer vision and natural language processing (NLP), deep neural networks (DNNs) dramatically improved the state-of-the-art performance on several problems in these fields and changed the way we approach problems. This thesis did not elude the overwhelming popularity of DNNs—Chapters 4, 7, and 10 introduce neural network-based models.

For the purposes of this thesis, we can think about DNNs as compositions of linear functions $f(\cdot)$ and element-wise non-linear transformations $\sigma(\cdot)$. These functions (also called *layers*) iteratively modify *hidden representations* \mathbf{h}^l of data to continuously adapt them to the task. In the most simple case of the multi-layer perceptron (MLP) [Ros58], the function $f(\cdot)$ is defined as

$$f(\mathbf{h}^l) = \mathbf{W}^l \mathbf{h}^{l-1} + \mathbf{b}^l, \quad (2.4)$$

where \mathbf{W} is the weight matrix and \mathbf{b} is called the bias vector. A pair of the weight matrix and the bias vector constitute the set of learnable parameters Θ for the layer. In modern MLPs, the non-linearity function of choice is the *rectified linear unit* (RELU) function $\sigma(\mathbf{h}) = \max(0, \mathbf{h})$. To train neural networks, today’s consensus is to use backpropagation [Wer82] and variants of stochastic gradient descent optimization [KB14].

Even simple neural network architectures like two-layer MLPs trained with backpropagation are invaluable for learning representations of words and phrases in NLP [MSC⁺13, BGJM17]. Some architectures are intimately related to matrix factorization—[LG14] proved that SVD provides an optimal solution to the learning problem of such networks. Such neural networks have been successfully adapted to the graph domain [PARS14]. We will review prior and concurrent work on node embeddings in Part I of this thesis.

2.4.1 GRAPH NEURAL NETWORKS

When graphs have additional node-level features attached to them, we can think about the graph structure as the *computational graph*, meaning that hidden representations are propagated across edges in a graph [GMS05]. Graph neural networks (GNNs) formalize this intuition into a powerful class of models that leverage the sparsity of graphs to provide state-of-the-art results on several tasks on attributed graphs. There are two perspectives on GNNs: message-passing [BHB⁺18] and convolutional [BZSL14, BBL⁺17] one.

The most popular GNN model is the *graph convolutional network* (GCN) architecture [KW17] due to its simplicity and speed. Figure 2.5 illustrates the process: representations of nodes are continuously updated to a function of the representations of their neighbors. More formally, the GCN update rule¹¹ is written as follows:

$$\mathbf{h}_v^l = \sigma \left(\mathbf{W}_{self}^l \mathbf{h}_v^{l-1} + \mathbf{W}_{neigh}^l \sum_{u \in \mathcal{N}(v)} \mathbf{h}_u^{l-1} + \mathbf{b}^l \right). \quad (2.5)$$

In contrast to deep convolutional neural networks for images [HZRS16], GCNs are rather shallow—for most applications, just two layers are enough. For additional reading on GNNs and overview of recent architectures, we recommend the following review articles [BBL⁺17, BHB⁺18, CAEHP⁺20].

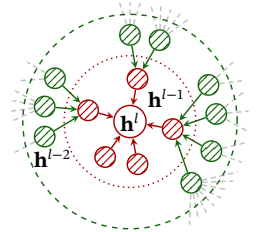


Figure 2.5: Information flow of a two-layer GCN. Hidden representations are first updated from the green nodes to the red ones; then, red nodes are used to compute the representation of the central node.

11. We separately add the representation of the node itself, known as the *skip connection* in the neural network literature.

PART I

SIMILARITIES AND REPRESENTATIONS OF NODES

NODE-CENTRIC tasks, such as node classification, community detection, and link prediction,¹ play a central role in graph analysis. Node representations, or node *embeddings*,² allow us to replace sophisticated algorithms developed for each of these tasks with simple algorithms developed for analyzing tabular data without significant loss in quality. For example, we could employ a basic logistic regression [PARS14] for node classification instead of an advanced label propagation approach [MP07] and a simple dense region search [HAMH16] instead of complex heuristics [AMF10]. This part of the dissertation devotes itself to the design of versatile algorithms that compute expressive representations of nodes in graphs efficiently.

No one-size-fits-all embedding algorithm is possible, as there is a wide range of tasks³. Therefore, users of embedding algorithms (graph analysts) must have an option to control the inner workings of embedding algorithms. Chapter 4 proposes VERSE, our unified framework for neural embedding algorithms that interprets their objectives as optimization for different *node similarity* distributions. We show how complexly parameterized competitors can be interpreted in purely graph mining terms. We propose a scalable optimization procedure that reduces the previous work’s complexity while outperforming competing methods in several tasks.

Learning representations through neural networks lacks clear goalposts in optimization—we do not know when to stop the learning and use the embeddings. Moreover, learning-based approaches are relatively slow—it takes at least a day to learn node embeddings of a 10 million node graph on a single server. Random access patterns prohibit efficient distributed optimization when the graph is large. To alleviate the scalability problems of neural embeddings, we study learning-free algorithms that aim to solve VERSE’s objective. Chapters 5 and 6 present two orthogonal attacks on the problem: FREDE, introduced in Chapter 5, generates provably correct embeddings from the perspective of a subset of nodes; SnapEmbed, in Chapter 6, produces a valid embedding for any node without using representations of other nodes.

1. Link prediction problem is a *node recommendation* problem in disguise.

2. *Graph embeddings* is another term used in the literature; we do not use this term to avoid confusion with representations of whole graphs.

3. This statement is also known as a “no free lunch” theorem [Wol96].

Last, we consider the unsupervised node embedding problem in *attributed* graphs, where nodes have rich information associated with them. In such networks, methods relying solely on graph structure perform poorly. We propose `DMON`, an unsupervised GNN training method that learns representations by clustering graphs. Traditionally, unsupervised deep learning architectures learn by either reconstructing the input [Bal87, HZ94] or maximizing entropy-related objectives [Lin88, MM90, HFLM⁺19]. In contrast, `DMON` optimizes a novel unsupervised objective based on spectral modularity maximization. We show that clusters obtained by `DMON` are comparable with ones obtained by simply clustering the graph structure. However, the clusters obtained by `DMON` are consistent with the graph structure and node attributes and thus are much better at predicting the ground-truth labels.

The following four chapters in this part are based on work as cited below:

- Chapter 4 “Vertex Similarity Embeddings” extends [TMKM18].
- Chapter 5 “Anytime Node Embeddings” extends [TMM⁺20b].
- Chapter 6 “Local Node Embeddings” extends [PTdA⁺20].
- Chapter 7 “Deep Modularity Networks” extends [TPPM20].

We now proceed with an overview of the related work relevant to all chapters.

3.1 RELATED WORK

Since graphs model various real-world systems, the problem of embedding graphs’ nodes in a lower-dimensional space has been historically tackled in many sub-disciplines of computer science, mathematics, and social science. We distinguish five categories of the related work: (i) graph theory and random embeddings with distortion guarantees; (ii) two- or three-dimensional graph drawing approaches; (iii) works on inferring social positions in computational social science; (iv) non-linear dimensionality reduction algorithms, and (v) modern graph mining methods. We give an overview of each field below; the contributions of this dissertation are mostly in the *graph mining* category.

3.1.1 GRAPH THEORY AND RANDOMIZED ALGORITHMS

The first non-constructive embedding of a graph was studied by Erdős [EHT65]. Erdős proposed the definition of a graph dimension to be the smallest d such that every edge has length 1 in \mathbb{R}^d . In such form, the dimensionality is related to the chromatic number of a graph: $\dim G = 2\chi(G)$. However, not all graphs are embeddable under such a strict notion. An example of a non-embeddable graph is a *claw*, shown in Figure 3.1.

Thus, it is more practical to study low-dimensional embeddings with some *distortion*. For a mapping \mathcal{F} between finite metric spaces V_1, V_2 with metrics d_1, d_2 , distortion is defined as

$$\max_{u,v} \frac{d_2(\mathcal{F}(u), \mathcal{F}(v))}{d_1(u, v)} / \min_{u,v} \frac{d_2(\mathcal{F}(u), \mathcal{F}(v))}{d_1(u, v)}.$$

A popular solution is an orthogonal projection into a random d -dimensional subspace [JL84] (we use formulation of [LV99]):

Lemma 1 (Johnson–Lindenstrauss lemma [JL84]). *For every $0 < \epsilon < 1$, every n -point set $S \subset \mathbb{R}^n$ can be mapped into \mathbb{R}^d , $d < 60 \ln n / \epsilon^2$, with distortion $1 + \epsilon$.*

A more balanced trade-off is due to Bourgain’s theorem:

Theorem 1 (Bourgain’s theorem [Bou85, LLR95]). *Every metric space with n elements can be embedded in an $\mathcal{O}(\log n)$ -dimensional Euclidean space with $\mathcal{O}(\log n)$ distortion.*

Linial, London, and Rabinovitch [LLR95] provide an algorithmic construction of such embedding with applications to several graph problems. However, practical applications of these constructions for graph mining have been lacking for 25 years. In the meanwhile, randomized algorithms became an essential tool for numerical linear algebra [Woo14] and data mining [IM98, Aho03].

3.1.2 COMPUTATIONAL SOCIAL SCIENCE

Quantitative social science research on social networks [Mor34] introduced nodes’ *structural positions* independently from the mainstream graph theory research. Sorokin [Sor27] introduced the concept of *social mobility* as a latent space with social hierarchies and roles. Eloquently, he postulates that “to find a position of a man or a social phenomenon in social space means to define his or its relations to other men or other social phenomena chosen as a

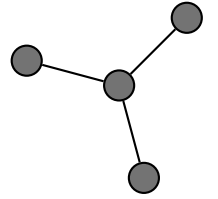


Figure 3.1: A claw graph.

‘points of reference.’” The concept is an extension of Durkheim’s *représentations collectives* [Dur12]—ideas and values irreducible to individual constituents but defined socially.

Computationally, however, explicit definitions of social spaces were not proposed until almost 50 years later. Burt [Bur76] first introduced the notion to computational social science, defining the social position as a cluster to which a node belongs in a hierarchically clustered network. The cluster-based notion of the social spaces prevailed (see a review of different methods [Fau88]) until the introduction of the latent space approaches [HRHo2]. Such approaches used Markov Chain Monte Carlo (MCMC) to infer latent positions of nodes that best explain observed social relations. These methods are limited to tiny networks, as they rely on unstable MCMC estimation.

3.1.3 NON-LINEAR DIMENSIONALITY REDUCTION

Perhaps the most popular node embedding application is in non-linear dimensionality reduction, where they got traction for approximating arbitrary complex surfaces. The three most popular algorithms, Isomap [TDSL00], locally-linear embedding (LLE) [RS00], and Laplacian eigenmaps [BN01], construct a k -nearest neighbor graph and then compute the nodes’ embeddings in that graph.

Unlike Isomap, LLE, and Laplacian eigenmaps solve sparse eigenproblems and thus are efficient: they are the first sub-quadratic node embedding algorithms. They achieve scalability by leveraging fast numerical eigensolvers for sparse matrices [Saa92, GVL12]. To the best of our knowledge, these algorithms were not applied for graph mining.

3.1.4 GRAPH DRAWING

Graph drawing is a special case of embedding into two-dimensional Euclidean space for visualization. Methods for general graph drawing are typically heuristics based on iterative optimization of force-directed layouts of nodes [Ead84]. The force calculation is possible in $\mathcal{O}(n \log n)$ time due to the Barnes-Hut algorithm [BH86]. A review of modern software for graph drawing can be found in [JM12]. We can also visualize a graph by projecting node embeddings in two dimensions—this is an exciting research direction for massive graphs.

3.1.5 GRAPH MINING

Tang and Liu [TLo9a] introduced explicit node representations to the field of graph mining. The proposal is to decompose the *modularity*⁴ matrix [Newo6a] to extract community-like vectors of memberships for nodes. Specifically, embeddings are obtained as the top- k eigenvectors of the modularity matrix. The computations are fast due to the sparse-plus-low-rank structure of the modularity matrix.

4. Modularity is a measure of quality for node clustering [Newo6b].

In follow-up work [TLo9b], the scalability problem is tackled via the modified k -means algorithm [Llo82] applied to the adjacency matrix’s edges. The memory cost of the representation is $\mathcal{O}(m)$ instead of $\mathcal{O}(kn)$ as we only keep the nearest cluster centroid in memory. The representations of nodes are a vector of distances of all edges to their nearest clusters. Embedding vectors are therefore very sparse (average sparsity is less than 1% across datasets). Sparsity is both the curse and the blessing of this method—while the memory footprint of sparse representations is lower than for the dense ones, the downstream models are not well equipped for dealing with sparse data. For example, logistic regression would create a dense projection vector for each embedding dimension, resulting in larger models that are prone to overfitting.

Ahmed et al. [ASN⁺13] study large-scale regularized factorization of the adjacency matrix. The objective in this work is close to the eigendecomposition with a different term to prevent degenerate solutions. The paper investigates the scalability in the distributed setting and applies several heuristics to achieve good performance, scaling linearly to graphs with hundreds of millions of nodes. Nevertheless, embeddings of nodes produced only from the adjacency matrix fail to capture sufficient semantic information [PARS14].

3.1.5.1 NEURAL NODE EMBEDDINGS

DeepWalk [PARS14] is the first neural network-based method for node embedding. It adapts the widely successful word2vec [MSC⁺13] architecture to deal with graph data. Word2vec builds word embeddings by training a single-layer neural network to guess the word’s contextual neighbors in a text. DeepWalk makes a simple yet powerful analogy: it treats nodes in short random walks as words in a natural language. It generates a “corpus” of short random walks; all nodes that co-occur in random walks within a specific window form a

node’s context. DeepWalk’s representations maximize the posterior probability of observing a neighboring vertex in a random walk. To maximize that probability efficiently, DeepWalk uses hierarchical softmax approximation: it constructs a Huffman tree of nodes based on their frequency of appearance. This way, each learning epoch takes $\mathcal{O}(n \log n)$ time. DeepWalk works well in practice; however, it is unclear how its parameters (window size and the length of random walks) relate to the graph structure. We discuss several attempts to introduce graph structure to the neural algorithms in the following.

The first, LINE [TQW⁺15], simplifies DeepWalk’s algorithm by getting rid of random walks altogether. Instead of generating walks, LINE samples edges from a graph, which is more efficient. Importantly for the method’s scalability, LINE chooses to use the negative sampling approximation of softmax, bringing the epoch complexity to $\mathcal{O}(m)$. Embeddings produced by LINE are of slightly worse quality than DeepWalk’s.

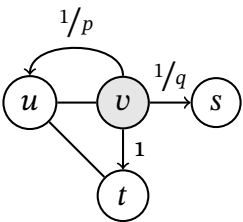


Figure 3.2: Example of node2vec’s second-order random walks. Here, we walked from node u to v .

A method rivaling DeepWalk in popularity⁵, node2vec introduces two additional parameters to the random walk generation of DeepWalk. These two parameters, named p and q , control the second-order Markov process behavior of the random walk. Parameters p and q determine the random walker’s probabilities (weights) to return to the previously visited node as $1/p$; 1 if there is an edge between the previously visited node and the proposed one, and $1/q$ otherwise. We illustrate this confusing behavior in Figure 3.2. In the original paper, breadth-first search and depth-first search are used as a motivation for introducing the additional complexity to DeepWalk. We note that this is impossible to emulate depth-first search using random walks with limited memory.

⁵. Which we can only explain by its catchy name.

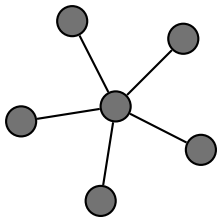


Figure 3.3: A star graph.

While node2vec claims to be a scalable method, the method’s computational complexity regresses to $\mathcal{O}(n^2)$ due to a sophisticated random walk generation process. The original paper incorrectly claims it to be dependent on the average degree $\mathcal{O}(n^m/n)$. It is easy to see that star-like graphs (Figure 3.3) are a counterexample to that statement. Moreover, nodes in graphs with high degrees introduce a complexity factor of $\text{deg}(v)^2$ on average. High computational complexity makes node2vec inapplicable to large scale-free graphs common in real-world scenarios.

3.1.5.2 MATRIX FACTORIZATION EMBEDDINGS

Neural embeddings were the first to attain excellent performance in many classification tasks. However, their objective functions do not provide clear guidance on when the embedding is ready to use. It is thus desirable to have a clearly defined optimum and algorithms for attaining it. Matrix factorization algorithms provide just that.

GraRep [CLX15] repeats Levy and Goldberg’s analysis of word2vec [LG14] for graphs. This paper proved that DeepWalk implicitly factorizes a log-transformed transition probability matrix, summed over the window size. The optimum for a fixed window size is given by sVD ; GraRep finds several such optima and concatenates the embeddings. Since the transition probability matrix is dense, sVD ’s complexity is $\mathcal{O}(n^3)$, unsuitable for large graphs. Nevertheless, this method started an investigation in matrix factorization for the purpose of node embedding.

The first scalable method, HOPE, utilizes generalized sVD to generate an embedding of nodes. This method achieves scalability by factorizing an implicit matrix $\mathbf{M} = \mathbf{M}_a^{-1}\mathbf{M}_b^{-1}$, where \mathbf{M}_a and \mathbf{M}_b are sparse. This formulation supports two standard node similarity definitions: Katz [Kat53] and personalized PageRank [PBMW99]. The decomposition itself is fast; however, the similarity matrix formulation’s lack of nonlinearity means that the embeddings are significantly worse in downstream task performance.

Another approach based on the eigendecomposition is taken by AROPE [ZCW⁺18]. It is based on the fact that any matrix function $f(\mathbf{M})$ can be decomposed as $f(\mathbf{M}) = \mathbf{Q}f(\mathbf{\Lambda})\mathbf{Q}^{-1}$. The suggestion is to take the best rank- k decomposition of the adjacency matrix and use it to approximate Katz node similarity. The method suffers from the same problem as HOPE: the lack of nonlinearity limits the expressiveness of the method.

NetMF [QDM⁺18] extends GraRep’s analysis to node2vec and LINE. The proposed algorithm is very similar to GraRep: we compute the log-transformed transition probability matrix of DeepWalk and apply sVD to it. Overall time complexity is again $\mathcal{O}(n^3)$, unsuitable for large graphs. In follow-up work, NetSMF [QDM⁺19], authors propose to decompose a spectrally sparsified version of the DeepWalk matrix. Satisfactory performance is not possible with sparse enough matrices, leaving the method out of practical considerations.

3.1.5.3 SKETCHING-BASED EMBEDDINGS

Sketching is an invaluable tool for creating approximate algorithms and performing approximate linear algebra calculations [Woo14]. Sketching algorithms relax the optimality constraints of the linear algebra solutions to a probabilistic guarantee that holds with high probability. Random projection methods from Section 3.1.1 are examples of sketching building blocks.

RandNE [ZCL⁺18] proposes to use an orthogonalized random matrix to iteratively project the adjacency matrix of the graph onto the new embedding. Then, the final embedding is a weighted sum of several (4 in the paper) projection steps; the weights are to be optimized via cross-validation on a downstream task. This method scales to graphs with billions of nodes; the dominating factor in the method’s complexity is the orthogonalization of the projection matrix. The requirement to optimize the weights in the sum defeats the purpose of fully unsupervised embedding.

FastRP [CST⁺19] replaces the random projection matrix of RandNE with another variant from [Acho3]. This replacement and the introduction of the degree normalization allow FastRP to forgo orthogonalization. Similarly to RandNE, the weights of the projection steps are tuned to achieve the best performance in some downstream tasks. Again, this supervised parameter tuning defeats the in purely unsupervised applications.

In another vein, NodeSketch produces embeddings by iteratively sketching the Jaccard similarity [Jacoi] of the modified adjacency matrix (with added self-loops). Similarly to the other methods, the final embedding matrix is a weighted sum of the sketching matrices in different steps. Yet again, the parameter tuning is supervised.

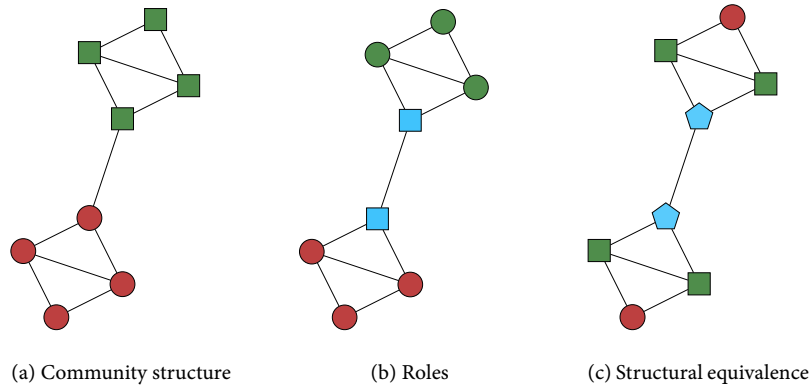
EMBEDDING a web-scale information network into a low-dimensional vector space facilitates tasks such as link prediction, classification, and visualization. Past research has addressed the problem of extracting such embeddings by adopting methods from NLP to graphs, without defining a comprehensible graph-related objective. Nevertheless, as we show, the past works’ objectives implicitly utilize similarity measures among graph nodes.

We carry the similarity velleity of previous works to its logical conclusion. In this chapter, we propose VERtEx Similarity Embeddings (VERSE), a simple, versatile, and memory-efficient method that derives node embeddings explicitly preserving the distributions of a selected vertex-to-vertex similarity measure. At its core, VERSE stands between complex deep learning approaches [CLX16, WCZ16] and the direct decomposition of the similarity matrix [TLo9a, CLX15]. Our method learns such embeddings by training a simple, yet expressive, single-layer neural network. We conduct an extensive experimental study and demonstrate that VERSE, instantiated with diverse similarity measures, outperforms state-of-the-art methods in major data mining tasks and supersedes them in time and space efficiency.

We argue that features extracted by a more *versatile* similarity notion than that of a local neighborhood [PARS14, TQW⁺15] would achieve the flexibility to solve diverse data mining tasks in a large variety of graphs. Figure 4.1 makes a case for such a versatile similarity notion by exposing three distinct kinds of similarity on a graph: *community structure* guides community detection tasks, *roles* are typically used in classification, while *structural equivalence* defines peer correspondences in knowledge graphs. As real-world tasks rely on a mix of such properties, a versatile feature learning algorithm should be capable of capturing all such similarities.

Thanks to its ability to choose any appropriate similarity measure for the task at hand, VERSE adjusts to that task without needing to change its core. Thereby, it fully integrates representation learning with feature engineering:

Figure 4.1: Three node properties are highlighted on the same graph. Can a single model capture these properties?



any similarity measure, including those developed in feature engineering, can be used as input to `VERSE`. For the sake of illustration, we instantiate our generic methodology using three popular similarity measures, namely Personalized PageRank (PPR) [PBMW99], SimRank [JW02], and adjacency similarity. We also show that versatility does not imply a new burden to the user, merely substituting hyperparameter tuning with similarity measure tuning: using PPR as a default choice for the similarity measure leads to good performance in nearly all tasks and networks we examine.

We summarize our contributions as follows:

- We propose a versatile framework for node embeddings that explicitly learns the distribution of any vertex similarity measure for graph nodes.
- We devise an efficient algorithm that minimizes the divergence from real to reconstructed similarity distributions.
- We interpret previous node embeddings through our novel similarity framework’s lens and instantiate `VERSE` with personalized PageRank, SimRank, and Adjacency similarity.
- In a thorough experimental evaluation, we show that `VERSE` outperforms the state-of-the-art approaches in various graph mining tasks in quality while being more efficient in terms of time and space.

4.1 A VERSATILE NODE EMBEDDING

In the following, we denote the representation as a $n \times d$ matrix W ; the embedding of a node v is the row $W_{v,\cdot}$ in the matrix; we denote it as W_v for compactness. Our embeddings reflect vertex similarity *distributions* for some given graph: $\text{sim}_G: V \times V \rightarrow \mathbb{R}$ for every node $v \in V$. As such, we require that the similarities from any vertex v to all other vertices $\text{sim}_G(v, \cdot)$ are amenable to be interpreted as a distribution with $\sum_{u \in V} \text{sim}_G(v, u) = 1$ for all $v \in V$. We aim to devise W by a scalable method that requires neither the $V \times V$ stochastic similarity matrix nor its explicit materialization.

The corresponding node-to-node similarity in the embedded space is $\text{sim}_E: V \times V \rightarrow \mathbb{R}$. As an optimization objective, we aim to minimize the Kullback-Leibler (KL) divergence from the given similarity distribution sim_G to that of sim_E in the embedded space:

$$\sum_{v \in V} \text{KL}(\text{sim}_G(v, \cdot) \parallel \text{sim}_E(v, \cdot)) \quad (4.1)$$

We illustrate the usefulness of this objective using a small similarity matrix. Figure 4.2 shows (a) the Personalized PageRank matrix, (b) the reconstruction of the same matrix by VERSE, and (c) the reconstruction of the same matrix using SVD. It is visible that the nonlinear minimization of KL-divergence between distributions preserves most of the information in the original matrix, while the linear SVD-based reconstruction fails to differentiate some nodes.

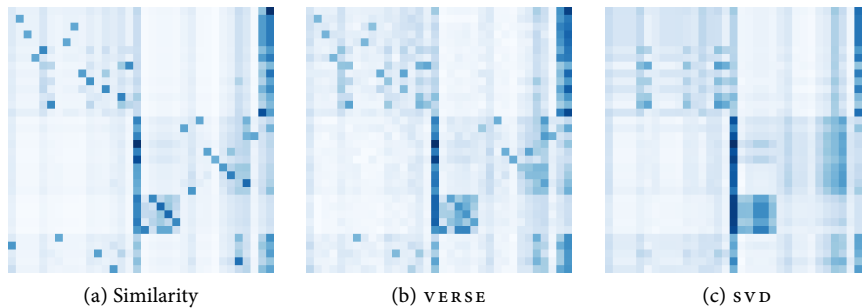


Figure 4.2: An example similarity matrix and its reconstructions by VERSE and SVD. Karate club graph [Zac77]; we set dimensionality $d = 4$ for both methods.

4.1.1 VERSE EMBEDDING MODEL

We define the unnormalized distance between two nodes u, v in the embedding space as the dot product of their embeddings $W_u \cdot W_v^\top$. The similarity distribution in the embedded space is then normalized with softmax:

$$\text{sim}_E(v, \cdot) = \frac{\exp(W_v W^\top)}{\sum_{i=1}^n \exp(W_v \cdot W_i)} \quad (4.2)$$

By Equation 4.1, we should minimize the κ_L -divergence from sim_G to sim_E ; omitting parts dependent on sim_G only, this objective is equivalent to minimizing the cross-entropy loss function [GBC16]:

$$\mathcal{L} = - \sum_{v \in V} \text{sim}_G(v, \cdot) \log(\text{sim}_E(v, \cdot)) \quad (4.3)$$

We can accommodate this objective by stochastic gradient descent, which allows updating the model on each node singularly. However, a naïve version of gradient descent would require the full materialization of sim_E and sim_G . Even in case sim_G is easy to compute on the fly, such as the adjacency matrix, the softmax in Equation 4.2 has to be normalized over all nodes in the graph.

We use Noise Contrastive Estimation (NCE) [GH10, MT12], which allows us to learn a model that provably converges to its objective (see [GH12], Theorem 2). This objective trains a binary classifier to distinguish between node samples coming from the empirical similarity distribution sim_G and those generated by a noise distribution \mathcal{Q} over the nodes. Consider an auxiliary random variable D for node classification, such that $D = 1$ for a node drawn from the empirical distribution and $D = 0$ for a sample drawn from the noise distribution. Given a node u drawn from some distribution \mathcal{P} and a node v drawn from the distribution of $\text{sim}_G(u, \cdot)$, we draw $s \ll n$ nodes \tilde{v} from $\mathcal{Q}(u)$ and use logistic regression to minimize the negative log-likelihood:

$$\mathcal{L}_{NCE} = \sum_{\substack{u \sim \mathcal{P} \\ v \sim \text{sim}_G(u, \cdot)}} \left[\log \Pr_W(D = 1 | \text{sim}_E(u, v)) + \right. \\ \left. s \mathbb{E}_{\tilde{v} \sim \mathcal{Q}(u)} \log \Pr_W(D = 0 | \text{sim}_E(u, \tilde{v})) \right], \quad (4.4)$$

where Pr_W is computed from W as a sigmoid $\sigma(x) = (1 + e^{-x})^{-1}$ of the dot product between vectors W_u and W_v , while we compute $\text{sim}_E(u, \cdot)$ without the normalization of Equation 4.2. As the number of noise samples s increases, the NCE derivative provably converges to the gradient of cross-entropy [MT12]; thus, by virtue of NCE’s asymptotic convergence guarantees, we are in effect minimizing the KL-divergence from sim_G . Theoretical guarantees of NCE depend on s , yet small values work well in practice [MT12]. In our experiments, we use $s = 3$. These convergence guarantees of NCE are not affected by choice of distributions \mathcal{P} and \mathcal{Q} (see [GH12], Corollary 5); however, its performance is empirically dependent on \mathcal{Q} [LA17].

4.1.2 INSTANTIATIONS OF VERSE

While VERSE can be used with any similarity function, we choose to instantiate our model to widely used similarities sim_G , namely personalized PageRank, Adjacency Similarity, and SimRank.

Personalized PageRank [PBMW99] is a common similarity measure among nodes, practically used for many graph mining tasks [GL16, LZ11].

Definition 3. Given a starting node distribution s , damping factor α , and the normalized adjacency matrix \mathbf{A} , the personalized PageRank vector π_s is defined by the recursive equation:

$$\pi_s = \alpha s + (1 - \alpha)\pi_s s \mathbf{A}$$

The stationary distribution of a random walk with restart with probability α converges to PPR [PBMW99]. Thus, a sample from $\text{sim}_G(v, \cdot)$ is the last node in a single random walk from node v . The damping factor α controls the average size of the explored neighborhood. In Section 4.1.5 we show that α is tightly coupled with the window size parameter w of DeepWalk and node2vec.

Adjacency similarity is a straightforward similarity measure; this similarity corresponds to the LINE-1 model and takes into account only the immediate neighbors of each node. More formally, given the out degree $\text{Out}(u)$ of node u

$$\text{sim}_G^{\text{ADJ}}(u, v) = \begin{cases} 1/\text{Out}(u) & \text{if } (u, v) \in E \\ 0 & \text{otherwise} \end{cases} \quad (4.5)$$

We experimentally demonstrate that `VERSE` model is effective even in preserving the adjacency matrix of the graph.

SimRank [JW02] is a measure of structural relatedness between two nodes, based on the assumption that two nodes are similar if they are connected to other similar nodes; SimRank is defined recursively as follows:

$$\text{sim}_G^{\text{SR}}(u, v) = \frac{C}{|I(u)||I(v)|} \sum_{i=1}^{|I(u)|} \sum_{j=1}^{|I(v)|} \text{sim}_G^{\text{SR}}(I_i(u), I_j(v)), \quad (4.6)$$

where $I(v)$ denotes the set of in-neighbors of node v , and C is a number between 0 and 1 that geometrically discounts the importance of farther nodes. SimRank is a recursive procedure that involves computationally expensive operations: the straightforward method has the complexity of $\mathcal{O}(n^4)$.

We can approximate SimRank up to a multiplicative factor dependent on C through random walks [JFW17]. They compute a SimRank approximation through two reversed random walks with restart where the damping factor α is set to $\alpha = \sqrt{C}$. A reversed random walk traverses any edge (u, v) in the opposite direction (v, u) . Since we are only interested in the distribution of each $\text{sim}_G^{\text{SR}}(v, \cdot)$, we ignore the multiplicative factor in the approximation that has little impact on our task.

4.1.3 VERSE ALGORITHM

Algorithm 1 presents the overall flow of `VERSE`. Given a graph, a similarity function sim_G , and the embedding space dimensionality d , we initialize the output embedding matrix W to $\mathcal{N}(0, \frac{1}{d})$. Then, we optimize our objective (Equation 4.4) by gradient descent using the `NCE` objective. To do so, we repeatedly sample a node from the positive distribution \mathcal{P} , sample the sim_G (e.g. pick a neighboring node), and draw s negative examples. The σ in Line 13 represents the sigmoid function $\sigma = (1 + e^{-x})^{-1}$, and λ the learning rate. We choose \mathcal{P} and \mathcal{Q} to be distributed uniformly by $\mathcal{U}(1, n)$.

Algorithm 1 VERSE

```

1: function verse( $G, \text{sim}_G, d$ )
2:    $W \leftarrow \mathcal{N}(0, d^{-1})$  ▷ With  $\mathbf{W} \in \mathbb{R}^{n \times d}$ 
3:   repeat
4:      $u \sim \mathcal{P}$  ▷ Sample a node
5:      $v \sim \text{sim}_G(u)$  ▷ Sample positive example
6:      $W_u, W_v \leftarrow \text{update}(u, v, 1)$ 
7:     for  $i \leftarrow 1 \dots s$  do
8:        $\tilde{v} \sim \mathcal{Q}(u)$  ▷ Sample negative example
9:        $W_u, W_{\tilde{v}} \leftarrow \text{update}(u, \tilde{v}, 0)$ 
10:    until converged
11:   return  $W$ 
12: function update( $u, v, D$ ) ▷ Logistic gradient update
13:    $g \leftarrow (D - \sigma(W_u \cdot W_v)) * \lambda$ 
14:    $W_u \leftarrow g * W_v$ 
15:    $W_v \leftarrow g * W_u$ 

```

As a strong baseline for applications handling smaller graphs, we also consider an exhaustive variant of VERSE, which computes *full similarity distribution* vectors per node instead of performing NCE-based sampling. We name this variant FVERSE and include it in our experimental study.

Figure 4.3 presents our measures on the ability to reconstruct a similarity matrix for (i) VERSE using NCE; (ii) VERSE using Negative Sampling (NS) (also used in node2vec); and (ii) the exhaustive FVERSE variant. We observe that, while NCE approaches the exhaustive method in terms of matching the ground truth top-100 most similar nodes, NS fails to deliver the same quality.

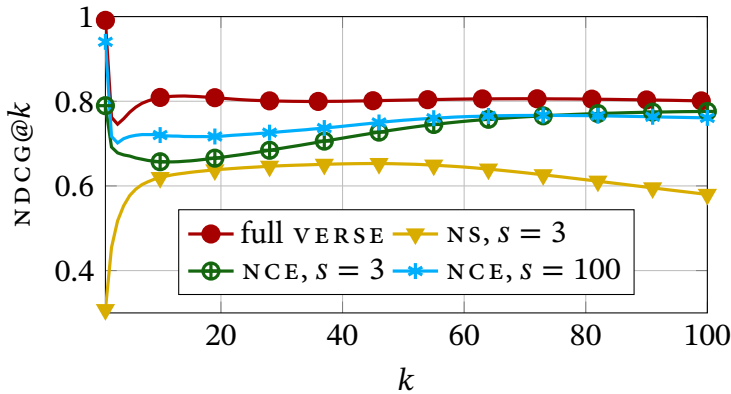


Figure 4.3: Ranking performance in terms of NDCG for reconstructing PPR similarity, averaged across nodes in a graph.

Table 4.1: Comparison of neural embedding methods in terms of the bounded degree ($\mathcal{O}_{\bar{k}}$) and worst-case (\mathcal{O}) time and space complexity, assuming sparse graphs.

<i>method</i>	Time		Space	
	$\mathcal{O}_{\bar{k}}$	\mathcal{O}	$\mathcal{O}_{\bar{k}}$	\mathcal{O}
DeepWalk	$dn \log n$	$dn \log n$	m	m
GraRep	tn^3	tn^3	n^2	n^2
LINE	dsn	dsn	m	m
node2vec	$dsn + k^3$	$dsn + n^3$	$k^2n + k^3$	n^3
HOPE	d^2m	d^2m	m	m
FVERSE	dn^2	dn^2	n^2	n^2
VERSE	dsn	dsn	m	m

4.1.4 COMPLEXITY COMPARISON

Table 4.1 presents the worst-case analysis for bounded degree graphs ($\mathcal{O}_{\bar{k}}$) and worst-case (\mathcal{O}) time and space complexity of VERSE, along with those of methods in previous works; d is the embedding dimensionality, n the number of nodes, m the number of edges, k is the maximum degree (in the bounded-degree case), s the number of negative samples used, and t the number of iterations in GraRep. Methods that rely on fast sampling (VERSE and LINE) require time linear in n and space linear in m in the worst case. DeepWalk requires $\mathcal{O}(n \log n)$ time due to its use of hierarchical softmax. Node2vec stores the neighbors-of-a-neighbor, incurring a cubic cost in terms of the maximum node degree. Thus, VERSE comes at the low end of complexities compared to previous work on node embeddings. Remarkably, even the computationally expensive FVERSE affords complexity comparable to some previous works.

4.1.5 SIMILARITY NOTIONS IN PREVIOUS APPROACHES

We provide additional theoretical considerations of VERSE compared to LINE, DeepWalk, and node2vec, and demonstrate how our general model subsumes and extends previous research in versatility and scalability.

DeepWalk and node2vec inherit the sample generation from word2vec. We derive a relationship between the window size w of that strategy and the damping factor α of Personalized PageRank.

Lemma 2. Let X_r be the random variable representing the length of a random walk r sampled with parameter w by word2vec. Then, for any $0 < j \leq w$,

$$\Pr(X_r = j) = \frac{2}{w(w+1)}(w-j+1). \quad (4.7)$$

Proof. For each node $v \in V$, word2vec strategy samples two random walks of length w starting from $v \in V$. These two random walks represent the *context* of v , where v is the central node of a walk of length $2w + 1$. The model is then trained on increasing context size up to w . Therefore, the number of nodes sampled for each random walk amount to $\sum_{i=1}^w i = \frac{w(w+1)}{2}$. A node at distance $0 < j \leq w$ is sampled $(w - j + 1)$ times; thus, the final probability is $\frac{2}{w(w+1)}(w - j + 1)$. \square

Personalized PageRank provides the maximum likelihood estimation for the distribution in Equation 4.7 for $\alpha = \frac{w-2}{w+1}$. Then, $w = 10$ corresponds to $\alpha = 0.72$, which is close to the standard $\alpha = 0.8$, proved effective in practice [BP98]. On the other hand, $\alpha = 0.95$, which, for example, achieves the best performance on a task in Section 4.2.2, corresponds to $w = 59$. Such large w prohibitively increases the computation time.

LINE introduces the concept of first- and second-order proximities to model complex node relationships. As we discussed, in VERSE, first-order proximity corresponds to the dot-product among the similarity vectors in the embedding space:

$$\text{sim}_E(u, v) = W_u \cdot W_v$$

On the other hand, second-order proximity corresponds to letting VERSE learn one more matrix W' , so as to model *asymmetric* similarities of nodes in the embedding space. We do that by defining sim_E asymmetrically, using both W and W' :

$$\text{sim}_E(u, v) = W_u \cdot W'_v$$

The intuition behind second-order proximity is the same as that of SimRank: similar nodes have similar neighborhoods. Every previous method, except for LINE-1, used second-order proximities, due to the word2vec interpretation of embeddings borrowed by DeepWalk and node2vec. In our model, second-order proximities can be encoded by adding an additional matrix; we empirically evaluate their effectiveness in Section 4.2.

4.2 EXPERIMENTS

We evaluate VERSE against several state-of-the-art node embedding algorithms. For repeatability purposes, we provide all data sets and the C++ source code for VERSE¹, DeepWalk² and node2vec³. We run the experiments on an Amazon AWS c4.8 instance with 60GB RAM. Each method is assessed on the best possible parameters, with early termination of the computation in case no result is returned within one day. We provide the following state-of-the-art node embedding methods for comparison:

1. <https://github.com/xgfs/verse>

2. <https://github.com/xgfs/deepwalk-c>

3. <https://github.com/xgfs/node2vec-c>

4. <https://github.com/tangjianpku/LINE>

- DeepWalk [PARS14]: This approach learns an embedding by sampling random walks from each node, applying word2vec-based learning on those walks. We use the default parameters described in the paper, i.e., walk length $t = 80$, number of walks per node $\gamma = 80$, and window size $w = 10$.
- LINE [TQW⁺15]: This approach learns a d -dimensional embedding in two steps, both using adjacency similarity. First, it learns $d/2$ dimensions using first-order proximity; then, it learns another $d/2$ features using second-order proximity. Last, the two halves are normalized and concatenated. We obtained a copy of the code⁴ and run experiments with total $T = 10^{10}$ samples and $s = 5$ negative samples, as described in the paper.
- GraRep [CLX15]: This method factorizes the full adjacency similarity matrix using SVD, multiplies the matrix by itself, and repeats the process t times. The final embedding is obtained by concatenating the steps. We use $t = 4$ and 32 dimensions for each SVD round; thus, the final embedding has $d = 128$.
- HOPE [OCP⁺16]: This method is a revised Singular Value Decomposition restricted to sparse similarity matrices. We report the results obtained running HOPE with the default parameters, i.e, Katz similarity (an extension of Katz centrality [Kat53]) as the similarity measure and β inversely proportional to the spectral radius. Since Katz similarity does not converge on directed graphs with sink nodes, we used Personalized PageRank with $\alpha = 0.85$ for the CoCit dataset.
- node2vec [GL16]: This is a *hyperparameter-supervised* approach that extends DeepWalk by adding two parameters, p and q , so as to control DeepWalk's

<i>dataset</i>	<i>Size</i>			<i>Statistics</i>		
	$ V $	$ E $	$ \mathcal{L} $	Avg. deg.	\mathcal{Q}	Density
BlogCat	10k	334k	39	64.8	0.24	6.3×10^{-3}
CoCit	44k	195k	15	8.86	0.72	2.0×10^{-4}
CoAuthor	52k	178k	—	6.94	0.84	1.3×10^{-4}
VK	79k	2.7M	2	34.1	0.47	8.7×10^{-4}
YouTube	1.1M	3M	47	5.25	0.71	9.2×10^{-6}
Orkut	3.1M	234M	50	70	0.68	2.4×10^{-5}

Table 4.2: Dataset characteristics: number of vertices $|V|$, number of edges $|E|$; number of node labels $|\mathcal{L}|$; average node degree; modularity \mathcal{Q} [Newo6b]; density defined as $|E|/\binom{|V|}{2}$.

random walk sampling. The special case with parameters $p = 1, q = 1$ corresponds to DeepWalk; yet, sometimes node2vec shows worse performance than DeepWalk in our evaluation, due to the fact it uses negative sampling, while DeepWalk uses hierarchical softmax. We fine-tuned the hyperparameters p and q on each dataset and task. Moreover, we used a large training data to fairly compare to DeepWalk, i.e., walk length $l = 80$, number of walks per node $r = 80$, and window size $w = 10$.

In addition to the node embedding methods described above, we implemented the following baselines:

- Heuristics for link prediction: we train a logistic regression model on a set of common node-specific features, namely node degree, number of common neighbors, Adamic-Adar, Jaccard coefficient, preferential attachment, and resource allocation index [LZ11, LLC10].
- Louvain community detection [BGLLo8]: We employ a standard partition method for community detection as a baseline for graph clustering, reporting the best partition in terms of modularity [Newo6b].

In line with previous research [PARS14, TQW⁺15, GL16] we set the embedding dimensionality d to 128. The learning procedure (Algorithm 1, Line 3) is run 10^5 times for `VERSE` and 250 times for `FVERSE`; the difference in the setting is motivated by the number of model updates which is $\mathcal{O}(n)$ in `VERSE` and $\mathcal{O}(n^2)$ in `FVERSE`.

We use LIBLINEAR [FCH⁺08] to perform logistic regression with default parameter settings. Unlike previous work [PARS14, TQW⁺15, GL16] we employ a stricter assumption for multi-label node classification: the number of correct classes is not known a priori, but found through the Label Powerset multi-label classification approach [TK06]. This makes the results incompatible with the mainstream literature [PARS14, GL16]; in the subsequent chapters, we switch to the evaluation protocols standard to the literature.

For link prediction and multi-label classification, we evaluated each individual embedding 10 times in order to reduce the noise introduced by the classifier. Unless otherwise stated, we run each experiment 10 times, and report the average value among the runs. Throughout our experimental study, we use the above parameters as default, unless indicated otherwise.

We test our methods on six real datasets and report in Table 4.2.

- BlogCat [ZL09] is a network of social interactions among bloggers in the BlogCat website. Node labels represent blog topics provided by authors.
- Microsoft Academic Graph [mag16] is a network of academic papers, citations, authors, and affiliations from Microsoft Academic website released for the KDD-2016 cup. It contains 150 million papers up to February 2016 spanning various disciplines from math to biology. We extracted two separate subgraphs from the original network, using 15 conferences in data mining, databases, and machine learning. The first, CoAuthor, is a co-authorship network among authors. The second, CoCit, is a network of papers citing other papers; labels represent conferences in which papers were published.
- VK is a Russian all-encompassing social network. We extracted two snapshots of the network in November 2016 and May 2017 to obtain information about link appearance. We use the gender of the user for classification and country for clustering.
- YouTube [TLo9b] is a network of social interactions among users of the YouTube video platform. The labels represent video genres.
- Orkut [YL15] is a network of social interactions among users of the Orkut social network platform. The labels represent communities of users. We extracted the 50 biggest communities and use them as labels for classification.

Operator	Result
Average	$(\mathbf{a} + \mathbf{b})/2$
Concat	$[\mathbf{a}_1, \dots, \mathbf{a}_d, \mathbf{b}_1, \dots, \mathbf{b}_d]$
Hadamard	$[\mathbf{a}_1 * \mathbf{b}_1, \dots, \mathbf{a}_d * \mathbf{b}_d]$
Weighted L1	$[\mathbf{a}_1 - \mathbf{b}_1 , \dots, \mathbf{a}_d - \mathbf{b}_d]$
Weighted L2	$[(\mathbf{a}_1 - \mathbf{b}_1)^2, \dots, (\mathbf{a}_d - \mathbf{b}_d)^2]$

Table 4.3: Vector operators used for link prediction task for each $u, v \in V$ and corresponding embeddings $\mathbf{a}, \mathbf{b} \in \mathbb{R}^d$.

The default form of `VERSE` runs Personalized PageRank with $\alpha = 0.85$. For the sake of fairness, we design a *hyperparameter-supervised* variant of `VERSE`, by analogy to the hyperparameter-tuned variant of DeepWalk introduced by node2vec [GL16]. This variant, `HSVERSE`, selects the best similarity with cross-validation across two proximity orders (as discussed in Section 4.1.5) and three similarities (Section 4.1.2) with $\alpha \in \{0.45, 0.55, 0.65, 0.75, 0.85, 0.95\}$ for $\text{sim}_G^{\text{PPR}}$ and $C \in \{0.15, 0.25, 0.35, 0.45, 0.55, 0.65\}$ for sim_G^{SR} .

method	Average	Concat	Hadamard	L_1	L_2
FVERSE	80.06	79.69	<u>86.71</u>	84.49	84.97
VERSE	79.16	78.79	<u>85.69</u>	71.93	72.11
DeepWalk	68.43	68.06	66.54	<u>79.06</u>	78.11
GraRep	74.87	74.91	<u>82.24</u>	80.03	80.05
LINE	77.49	77.39	<u>77.73</u>	70.55	71.83
HOPE	<u>74.90</u>	74.83	74.81	74.34	74.81
HSVERSE	79.52	79.10	<u>86.15</u>	76.45	76.72
node2vec	77.07	76.67	79.42	<u>81.25</u>	80.85
Feature Eng.			77.53		

Table 4.4: Link prediction results on the CoAuthor coauthorship graph. Best results per method are underlined.

4.2.1 LINK PREDICTION

Link prediction is the task of anticipating the appearance of a link between two nodes in a network. Conventional measures for link prediction include Adamic-Adar, Preferential attachment, Katz, and Jaccard coefficient. We train a Logistic regression classifier on edge-wise features obtained with the methods shown in Table 4.3. For instance, for a pair of nodes u, v , the Concat operator returns a vector as the sequential concatenation of the embeddings $f(u)$ and $f(v)$. On the CoAuthor data, we predict new links for 2015 and 2016

co-authorships, using the network until 2014 for training; on VK, we predict whether a new friendship link appears between November 2016 and May 2017, using 50% of the new links for training and 50% for testing. We train the binary classifier by sampling non-existing edges as negative examples. Tables 4.4 and 4.5 report the attained accuracy. As a baseline, we use a logistic regression classifier trained on the respective data sets’ features.

Table 4.5: Link prediction results on the VK coauthorship graph. Best results per method are underlined.

<i>method</i>	Average	Concat	Hadamard	L_1	L_2
FVERSE	74.94	74.81	<u>80.77</u>	78.49	79.13
VERSE	73.78	73.66	<u>79.71</u>	74.11	74.56
DeepWalk	70.05	69.92	69.79	<u>78.38</u>	77.37
LINE	<u>75.17</u>	75.13	72.54	63.77	64.47
HOPE	71.89	<u>71.90</u>	70.22	71.22	70.63
HSVERSE	74.14	74.02	<u>80.26</u>	73.04	73.53
node2vec	71.29	71.22	72.43	78.38	<u>78.66</u>
Feature Eng.			78.84		

VERSE with Hadamard product of vectors is consistently the best edge representation. We attribute this quality to the explicit reconstruction we achieve using noise contrastive estimation. VERSE consistently outperforms the baseline in the tested datasets. Besides, the hyperparameter-supervised HSVERSE variant outruns node2vec on all datasets.

4.2.2 NODE CLASSIFICATION

We now conduct an extensive evaluation on classification and report results for all the methods, where possible, with the CoCit, VK, YouTube, and Orkut graphs. Node classification aims to predict the correct node labels in a graph, as described previously in this section.

We evaluate accuracy by the Micro-F1 and Macro-F1 percentage measures. We report only Macro-F1, since we experience similar behaviors with Micro-F1. For each dataset we conduct multiple experiments, selecting a random sample of nodes for training and leaving the remaining nodes for testing. The results for four datasets, shown in Tables 4.6–4.9, exhibit similar trends: VERSE yields predictions comparable or superior to those of the other contestants, while it scales to large networks such as Orkut. As an exception, LINE

<i>method</i>	<i>labelled nodes, %</i>				
	1%	3%	5%	7%	9%
FVERSE	27.52	29.83	31.01	31.68	32.24
VERSE	27.32	29.42	30.67	31.32	31.83
DeepWalk	26.81	29.27	30.37	31.04	31.43
GraRep	27.68	29.21	30.24	30.23	30.79
LINE	23.68	26.90	27.89	28.49	28.80
HOPE	22.81	26.63	27.59	28.19	28.58
HSVERSE	27.46	29.45	30.67	31.38	31.92
node2vec	27.45	29.66	30.82	31.54	32.04

Table 4.6: Multi-class classification results in CoCit dataset.

<i>method</i>	<i>labelled nodes, %</i>				
	1%	3%	5%	7%	9%
FVERSE	58.32	61.01	61.74	62.26	62.50
VERSE	57.89	60.53	61.43	61.86	62.13
DeepWalk	58.22	60.93	61.79	62.17	62.49
LINE	60.39	62.83	63.58	64.01	64.23
HOPE	54.88	56.65	57.04	57.40	57.68
HSVERSE	58.87	61.67	62.50	62.97	63.16
node2vec	58.85	61.79	62.62	63.04	63.30

Table 4.7: Multi-class classification results in VK dataset.

<i>method</i>	<i>labelled nodes, %</i>				
	1%	3%	5%	7%	9%
VERSE	17.92	22.26	24.07	25.07	25.99
DeepWalk	18.16	21.55	22.89	23.64	24.54
LINE	13.71	17.36	18.69	19.84	20.64
HOPE	9.22	13.80	15.09	16.18	16.78
HSVERSE	18.16	22.84	25.40	27.38	29.09

Table 4.8: Multi-label classification results in YouTube dataset.

<i>method</i>	<i>labelled nodes, %</i>				
	1%	3%	5%	7%	9%
VERSE	25.16	28.22	29.60	31.46	32.63
DeepWalk	24.21	27.99	29.63	30.60	31.27
LINE	26.79	30.89	32.34	32.92	33.65
HSVERSE	27.73	30.70	32.73	34.00	35.20

Table 4.9: Multi-class classification results in Orkut dataset.

outperforms `VERSE` in VK, where the gender of users is better captured using the direct neighborhood. The hyperparameter-supervised variant, `HSVERSE`, is on a par with `node2vec` in terms of quality on CoCit and VK; on the largest datasets YouTube and Orkut, `HSVERSE` keeps outperforming unsupervised alternatives, while `node2vec` depletes the memory.

4.2.3 NODE CLUSTERING

Graph clustering detects groups of nodes with similar characteristics [Newo6b, BGLLo8]. We assess the embedding methods, using the k -means algorithm with k -means++ initialization [AVo7] to cluster the embedded points in a d -dimensional space. Table 4.10 reports the Normalized Mutual Information (NMI) with respect to the original label distribution. On CoAuthor, `VERSE` has comparable performance with DeepWalk; yet on VK, `VERSE` outperforms all other methods.

We also assess node embeddings on their ability to capture the graph community structure. We apply k -means with different k values between 2 and 50 and select the best modularity [Newo6b] score. Table 4.11 presents our results, along with the modularity obtained by the Louvain method, the state-of-the-art modularity maximization algorithm [BGLLo8]. `VERSE` variants produce results almost equal to those of Louvain, outperforming previous methods, while the three methods that could manage the Orkut data perform similarly.

Table 4.10: Node clustering results in terms of NMI.

<i>method</i>	CoCit	VK
<code>FVERSE</code>	33.22	9.24
<code>VERSE</code>	32.93	7.62
DeepWalk	34.33	7.59
<code>LINE</code>	18.79	7.49
GraRep	27.43	—
<code>HOPE</code>	19.05	6.47
<code>HSVERSE</code>	33.24	8.77
<code>node2vec</code>	32.84	8.05
Louvain	30.73	4.54

<i>method</i>	CoCit	CoAuthor	VK	YouTube	Orkut
FVERSE	70.12	80.95	44.59	—	—
VERSE	69.43	79.25	45.78	67.63	42.64
DeepWalk	70.04	73.83	43.30	58.08	44.66
LINE	60.02	71.58	39.65	63.40	42.59
GraRep	67.61	77.40	—	—	—
HOPE	42.45	69.57	21.70	37.94	—
HSVERSE	69.81	79.31	45.84	69.13	—
node2vec	70.06	75.78	44.27	—	—
Louvain	72.05	84.29	46.60	71.06	—

Table 4.11: Node clustering results in terms of modularity.

4.2.4 GRAPH RECONSTRUCTION

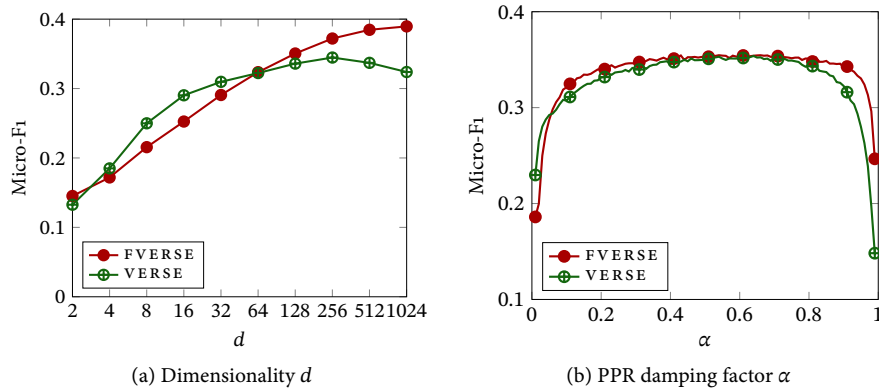
Good node embeddings should preserve the graph structure in the embedding space. We evaluate the performance of our method on reconstructing the graph’s adjacency matrix. Since each adjacent node should be close in the embedding space, we first sort any node other than the one considered by decreasing cosine distance among the vectors. Afterwards, we take a number of nodes equal to the actual degree of the node in the graph and connect to the considered node to create the graph structure.

Table 4.12 reports the relative accuracy measured as the number of correct nodes in the neighborhood of a node in the embedding space. Again, VERSE performs comparably well; its exhaustive variant, FVERSE, which harnesses the full similarity does even better; however, the top performer is HSVERSE, which achieves the obtained result when instantiated to the Adjacency Similarity. This result is unsurprising, given that the adjacency similarity measure tailors HSVERSE for the task of graph reconstruction.

<i>method</i>	CoCit	CoAuthor	VK	YouTube	Orkut
FVERSE	88.96	98.20	66.45	—	—
VERSE	58.73	74.30	50.18	28.64	18.39
DeepWalk	51.54	68.44	43.04	32.21	19.75
LINE	23.32	62.01	42.80	17.76	10.82
GraRep	67.61	77.40	—	—	—
HOPE	25.88	49.70	12.01	33.42	—
HSVERSE	97.53	98.91	78.38	38.34	28.81
node2vec	66.35	72.70	53.70	—	—

Table 4.12: Graph reconstruction % for all datasets.

Figure 4.4: Classification performance for (a) different dimensionality and (b) PPR damping factor α .



4.2.5 PARAMETER SENSITIVITY

We also evaluate the sensitivity of `VERSE` to the parameter choice. Figure 4.4 depicts node classification performance in terms of Micro-F1 on the BlogCat dataset, with 10% of nodes labeled.

The dimensionality d determines the size of the embedding, and hence the possibility to compute more fine-grained representations. The performance grows linearly as the number of dimensions approaches 128, while with larger d we observe no further improvement. Sampled `VERSE` instead, performs comparably better than `FVERSE` in low dimensional spaces, but degrades as d becomes larger than 128; this behavior reflects a characteristic of node sampling that tends to preserve similarities of close neighborhoods in low-dimensional embeddings, while the `VERSE` leverages the entire graph structure for larger dimensionality

The last parameter we study is the damping factor α which amounts to the inverse of the probability of restarting random walks from the initial node. As shown in Figure 4.4, the quality of classification accuracy is quite robust with respect to α for both `VERSE` and `FVERSE`, only compromised by extreme values. An α value close to 0 reduces PPR to an exploration of the immediate neighborhood of the node. On the other hand, a value close to 1 amounts to regular PageRank, deeming all nodes as equally important. This result vindicates our work and distinguishes it from previous methods based on local neighborhood expansion.

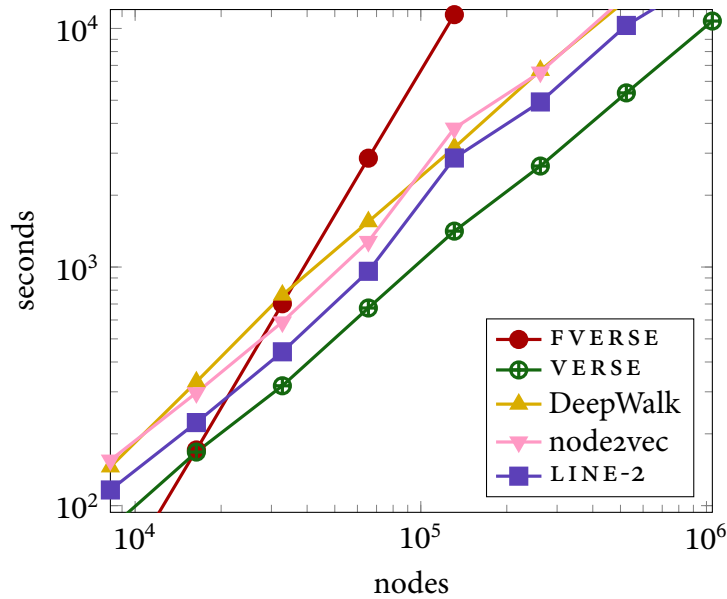


Figure 4.5: Scalability of different methods.

4.2.6 SCALABILITY

We now present runtime results on synthetic graphs of growing size, generated by the Watts Strogatz model [WS98], setting `VERSE` against scalable methods with C++ implementations, namely `DeepWalk`, `LINE`, and `node2vec`. For each method, we report the total wall-clock time, with graph loading and necessary preprocessing steps included. We used `LINE-2` time for fair comparison. As Figure 4.5 shows, `VERSE` is comfortably the most efficient and scalable method, processing 10^6 nodes in about 3 hours, while `DeepWalk` and `LINE` take from 6 to 15 hours.

4.2.7 VISUALIZATION

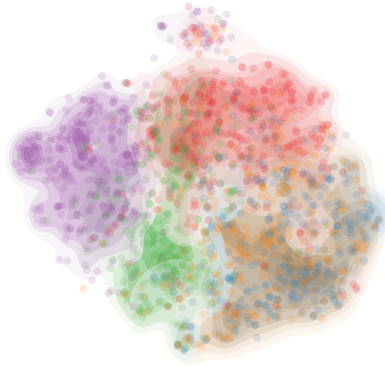
Last, we show how different embeddings are visualized on a plane. We apply `t-SNE` [vHo8] with default parameters to each embedding for a subset of 1500 nodes from the CoCit dataset, equally distributed in 5 classes (i.e., conferences); we set the density areas for each class by Kernel Density Estimation. Figure 4.6 depicts the result. produces well separated clusters with low noise, even finding distinctions among papers of the same community, namely `ICDE` (●) and `VLDB` (●).

Figure 4.6: Visualization of a subset of nodes from CoCit graph with selected conferences:

● VLDB, ● ICDE, ● KDD,

● WWW, and ● NeurIPS.

Note that the number of nodes per class is the same for all conferences.



4.3 SUMMARY

We introduced a new perspective on node embeddings: to be expressive, a node embedding should capture *some* similarity measure among nodes. Armed with this perspective, we developed a scalable embedding algorithm, `VERSE`. In a departure from previous works in the area, `VERSE` aims to reconstruct the distribution of any chosen similarity measure for each graph node. Thereby, `VERSE` brings in its scope a global view of the graph, while substantially reducing the number of parameters required for training. `VERSE` attains linear time complexity, hence it scales to large real graphs, while it only requires space to store the graph. Besides, we have shed light on some previous works on node embeddings, looking at them and interpreting them through the prism of vertex similarity.

Our thorough experimental study shows that, even instantiated with `PPR` as a *default* similarity notion, `VERSE` consistently outperforms state-of-the-art approaches for node embeddings in a plethora of graph tasks, while a hyperparameter-supervised variant does even better. Thus, we have provided strong evidence that embeddings genuinely based on vertex similarity address graph mining challenges better than others.

QUALITY guarantees of neural node embedding algorithms are practically non-existent. Matrix factorization algorithms ameliorate this problem; yet, it is challenging to balance the space complexity and quality of an embedding. To date, no node embedding work combines (i) linear space complexity, (ii) a nonlinear transform as its basis, and (iii) nontrivial quality guarantees. This chapter introduces FREquent Directions Embedding (FREDE), an algorithm based on matrix sketching that combines those three desiderata. Starting out from the observation that embedding methods aim to preserve the covariance among the rows of a similarity matrix, FREDE iteratively improves on quality while individually processing rows of a nonlinearly transformed PPR similarity matrix derived and provides, *at any iteration*, column-covariance approximation guarantees in due course almost indistinguishable from those of the optimal approximation by SVD. Our experimental evaluation on variably sized networks shows that FREDE performs almost as well as SVD and competitively against state-of-the-art embedding methods in diverse graph mining tasks, even when it is based on as little as 10% of node similarities.

We observe that factorization-based embeddings effectively strive to preserve the covariance of a similarity matrix, and that a few nodes acting as oracles approximate the distances among all nodes with guarantees [TZ05]. Given these observations, we adapt a covariance-preserving matrix sketching algorithm, Frequent Directions (FD) [Lib13, GLPW16], to produce a node embedding by factorizing, on a per-row basis, a PPR-like node similarity matrix derived by interpreting VERSE as matrix factorization. Uniquely, FREDE can be distributed, as it inherits the *mergeability* property of FD: two embeddings can be computed independently on different node sets and merged to a single embedding, with quality guarantees that hold *anytime* [Zil96], even after accessing a subset of similarity matrix rows.

We summarize our contributions as follows:

1. We interpret a state-of-the-art node embedding method, `VERSE`, as factorizing a transformed PPR similarity matrix;
2. We propose `FREDE`, an anytime node embedding algorithm that minimizes covariance error on that PPR-like matrix via sketching, with space complexity linear in the number of nodes and time linear in the number of processed rows;
3. In a thorough experimental evaluation, we confirm that `FREDE` is competitive against the state of the art and scales to large networks.

5.1 PRELIMINARIES

Our work builds on the know-how of matrix sketching to derive scalable, anytime node embeddings for practical data science tasks. Here, we review the fundamentals of matrix sketching.

An alternative to `SVD`, matrix sketching, finds a low-dimensional matrix, or a *sketch*, $\mathbf{W} \in \mathbb{R}^{d \times t}$ of a matrix $\mathbf{M} \in \mathbb{R}^{s \times t}$ (s elements, t features) that retains most of the information in \mathbf{M} without striving for matrix reconstruction. Sketching methods operate in streaming fashion, guaranteeing quality when rows arrive one after another. A popular sketch objective [BMD09, BDMI11, Lib13, Woo14] is to preserve the *column covariance* $\mathbf{M}^\top \mathbf{M}$ of \mathbf{M} , i.e., minimize covariance error:

Definition 4 (Covariance error). The column covariance error is the normalized difference between the covariance matrices:

$$ce_k(\mathbf{M}, \mathbf{W}) = \frac{\|\mathbf{M}^\top \mathbf{M} - \mathbf{W}^\top \mathbf{W}\|_2}{\|\mathbf{M} - [\mathbf{M}]_k\|_F^2} \geq \frac{\|\mathbf{M}^\top \mathbf{M} - \mathbf{W}^\top \mathbf{W}\|_2}{\|\mathbf{M}\|_F^2} = ce(\mathbf{M}, \mathbf{W})$$

The covariance error accounts for variance loss in each dimension. The correct k for the best rank k approximation $[\mathbf{M}]_k$ is not known and often requires grid search. Hence, we use the lower bound $ce(\mathbf{M}, \mathbf{W})$ in lieu of $ce_k(\mathbf{M}, \mathbf{W})$. When minimizing reconstruction error by `SVD`, with $[\mathbf{M}]_k = \mathbf{U}_k \Sigma_k \mathbf{V}_k^\top$, we may also optimal covariance error, which depends on the singular value decay of \mathbf{M} , by setting $\mathbf{W} = \Sigma_k \mathbf{V}_k^\top$. Since `SVD` is often computationally heavy, sketching algorithms typically provide error guarantees on ce by row-wise processing of \mathbf{M} .

A desirable sketch property is *mergeability*:

Definition 5 (Mergeability). A sketching algorithm *sketch* is *mergeable* if there exists an algorithm *merge* that, applied on the $d \times t$ sketches, $\mathbf{W}_1 = \text{sketch}(\mathbf{M}_1)$ and $\mathbf{W}_2 = \text{sketch}(\mathbf{M}_2)$, of two $s/2 \times t$ matrices, $\mathbf{M}_1, \mathbf{M}_2$, with $\text{ce}(\mathbf{M}_1, \mathbf{W}_1) \leq \epsilon$ and $\text{ce}(\mathbf{M}_2, \mathbf{W}_2) \leq \epsilon$, produces a $d \times t$ sketch \mathbf{W} of the concatenated matrix $\mathbf{M} = [\mathbf{M}_1; \mathbf{M}_2]$, $\mathbf{W} = \text{merge}(\mathbf{W}_1, \mathbf{W}_2) = \text{sketch}(\mathbf{M})$, that preserves the covariance error bound ϵ , i.e., $\text{ce}(\mathbf{M}, \mathbf{W}) \leq \epsilon$.

We now discuss some representative sketching algorithms.

Hashing. We construct a 2-universal hash function $h : [s] \rightarrow [d]$ and a 4-universal hash function $g : [s] \rightarrow \{-1, +1\}$. Starting with a zero-valued sketch matrix \mathbf{W} , each row \mathbf{M}_i is added to the $h(i)$ -th sketch matrix row with sign $g(i)$: $\mathbf{W}_{h(i)} = g(i) * \mathbf{M}_i$, with complexity linear in matrix size, $\mathcal{O}(st)$. In practice, random assignment of rows is used instead of a hash function. Setting $d = \mathcal{O}(t^2/\epsilon^2)$, hashing achieves $\text{ce} \leq \epsilon$ [Woo14]. This sketch is trivially mergeable: $\text{merge}(\mathbf{W}_1, \mathbf{W}_2) = \mathbf{W}_1 + \mathbf{W}_2$.

Random Projections are a fundamental data analysis tool [Woo14]. Boutsidis et al. [BDMI11] propose a row-streaming matrix sketching algorithm that randomly combines rows of the input matrix. In matrix form, $\tilde{\mathbf{M}} = \mathbf{R}\mathbf{M}$, where the elements R_{ij} of the $d \times s$ matrix \mathbf{R} are uniformly from $\{-1/\sqrt{d}, 1/\sqrt{d}\}$. For each row \mathbf{M}_i , the algorithm samples a random vector $\mathbf{r}_i \in \mathbb{R}^d$ with entries in $\{-1/\sqrt{d}, 1/\sqrt{d}\}$ and updates $\mathbf{W} = \mathbf{W} + \mathbf{r}_i \mathbf{M}_i^T$. This sketch achieves $\text{ce} \leq \epsilon$ with $d = \mathcal{O}(t/\epsilon^2)$, with practical performance exceeding the guarantee [LHC06], and is mergeable with $\text{merge}(\mathbf{W}_1, \mathbf{W}_2) = \mathbf{W}_1 + \mathbf{W}_2$.

Sampling. The Column Subset Selection Problem (CSSP) [BMD09] is to select a small column subset of an entire matrix. In the row-update model, a solution is found by sampling scaled rows $\mathbf{M}_i/\sqrt{d p_i}$ with probability $p_i = \|\mathbf{M}_i\|^2/\|\mathbf{M}\|_F^2$. While the norm $\|\mathbf{M}\|_F^2$ is usually unknown in advance, the method can work with d reservoir samplers, where d is the sketch size. This sketch achieves $\text{ce} \leq \epsilon$ with $d = \mathcal{O}(t/\epsilon^2)$, yet the cost of maintaining reservoir samples is non-negligible. The sketch is mergeable if we use distributed reservoir sampling.

Frequent Directions [Lib13] is the current state of the art in sketching, extends the Misra-Gries algorithm [MG82] from frequent items to matrices and outperforms other methods [CW13, BMD09, BDMI11] in quality. This algorithm sketches a matrix by iteratively filling the sketch with incoming rows, performing svD on the sketch when it cannot hold more rows, and shrinking the accumulated vectors with a low-rank svD approximation. The complexity is $\mathcal{O}(dts)$, due to s/d iterations of computing the $\mathcal{O}(d^2t)$ svD decomposition of a $2d \times t$ matrix \mathbf{W} with $d \ll t$. This sketch achieves $ce \leq \epsilon$ when $d = \mathcal{O}(t/\epsilon)$ and is mergeable with

$$\text{merge}(\mathbf{W}_1, \mathbf{W}_2) = \text{FD}(\text{concatenate}(\mathbf{W}_1, \mathbf{W}_2)). \quad (5.1)$$

The table below lists the embedding dimension d required to attain error bound $ce \leq \epsilon \leq 1$ for different algorithms.

Algorithm	Hashing	RP	Sampling	FD
Dimension d	$\mathcal{O}(t^2/\epsilon^2)$	$\mathcal{O}(t/\epsilon^2)$	$\mathcal{O}(t/\epsilon^2)$	$\mathcal{O}(t/\epsilon)$

We observe that, by putting the node similarity matrix \mathbf{S} in the role of the sketched matrix \mathbf{M} , we can effectively turn a sketching technique to an embedding method. Indeed, recent work [ZCL⁺18] has adapted a sketching algorithm [Vemo5, BDMI11] to node embeddings, yet forfeited¹ its error guarantees. We apply the know-how of state-of-the-art matrix sketching to serve node embedding purposes, leading to *anytime node embeddings* with error guarantees.

1. In our experiments, we use a variant of [ZCL⁺18] with error guarantees as a simple baseline.

5.2 ANYTIME NODE EMBEDDINGS

We observe that svD -based node embeddings, such as HOPE and NetMF, use only one of the two matrices svD produces, \mathbf{U} or \mathbf{V} . For example, NetMF returns $\mathbf{W} = \mathbf{U}_{:d} \sqrt{\mathbf{\Sigma}}_{:,d}$, with $\mathbf{\Sigma}$ truncated to d singular values. Therefore, such methods cannot reconstruct matrix \mathbf{S} ; svD products \mathbf{U} and $\mathbf{\Sigma}$ may only reconstruct the row covariance matrix $\mathbf{S}\mathbf{S}^\top = \mathbf{U}\mathbf{\Sigma}^2\mathbf{U}^\top$, as $\mathbf{W}\mathbf{W}^\top = \mathbf{U}\mathbf{\Sigma}^2\mathbf{U}^\top$, where $\mathbf{W} = \mathbf{U}\mathbf{\Sigma}$; thus, such methods are better understood as ones

implicitly minimizing the covariance error, rather than the reconstruction error [QDM⁺18], in relation to a similarity matrix among graph nodes.

Serendipitously, sketching algorithms aim to reconstruct the column covariance $\mathbf{S}^\top \mathbf{S} = \mathbf{V} \boldsymbol{\Sigma}^2 \mathbf{V}^\top$. Given this relationship, we apply a state-of-the-art matrix sketching algorithm in lieu of SVD to construct a node embedding in anytime fashion, by row updates of any partially materializable similarity matrix \mathbf{S} . Unfortunately, the matrix form of DeepWalk cannot be partially materialized. Next, we propose a partially materializable matrix based on PPR, inspired from the VERSE similarity-based embeddings. As we show in the experiments, this choice attains good quality and time performance. However, our method carries no prejudice with regard to the partially materializable matrix used; other choices are possible, such as, for example, the Node-Reweighted PageRank (NRP) [YSX⁺20]. Our aim is to illustrate the advantageous application of sketching for embedding purposes, while our framework supports any way of deriving the primary input matrix.

5.2.1 A ROW-WISE COMPUTABLE SIMILARITY MATRIX

The first neural similarity-based embedding method that does not require the entire matrix as input is VERSE, as it allows for efficient row-wise computation; in its default version, it uses the PPR similarity measure. To compute PPR_i , we leverage the fact that the probability distribution of a random walk with restart converges to PPR_i vector [PBMW99, BCG10]. Following [LG14, CLX15, QDM⁺18] we show that, under mild assumptions, VERSE with PPR similarity virtually factorizes the $\log(\text{PPR})$ matrix up to an additive constant.

Theorem 2. Let \mathbf{X} be the matrix of VERSE embeddings. If the terms $z_{ij} = \mathbf{x}_i^\top \mathbf{x}_j$ are independent, then VERSE factorizes the matrix $\mathbf{Y} = \log(\text{PPR}) + \log n - \log b = \mathbf{X}\mathbf{X}^\top$.

Proof. Consider the VERSE objective function for the uniform sampling distribution and PPR similarity:

$$\mathcal{L} = \sum_{i=1}^n \sum_{j=1}^n \left[\text{PPR}_{ij} \log \boldsymbol{\Sigma}(\mathbf{x}_i^\top \mathbf{x}_j) + b \mathbb{E}_{j' \sim \mathcal{Q}_i} \log \boldsymbol{\Sigma}(-\mathbf{x}_i^\top \mathbf{x}_{j'}) \right],$$

where $\Sigma(x) = (1 + e^{-x})^{-1}$ is the sigmoid, \mathcal{Q}_i is the noise sample distribution, and b the number of noise samples. Since PPR is right-stochastic and \mathcal{Q}_i is uniform, i.e., $\Pr(\mathcal{Q}_i = j) = \frac{1}{n}$, we can separate the two terms as follows:

$$\mathcal{L} = \sum_{i=1}^n \sum_{j=1}^n \text{PPR}_{ij} \log \Sigma(\mathbf{x}_i^\top \mathbf{x}_j) + \frac{b}{n} \sum_{i=1}^n \sum_{j'=1}^n \log \Sigma(-\mathbf{x}_i^\top \mathbf{x}_{j'}).$$

An individual loss term for vertices i and j is:

$$\mathcal{L}_{ij} = \text{PPR}_{ij} \log \Sigma(\mathbf{x}_i^\top \mathbf{x}_j) + \frac{b}{n} \log \Sigma(-\mathbf{x}_i^\top \mathbf{x}_j).$$

We substitute $z_{ij} = \mathbf{x}_i^\top \mathbf{x}_j$, use our independence assumption, and solve for $\frac{\partial \mathcal{L}_{ij}}{\partial z_{ij}} = \text{PPR}_{ij} \Sigma(-z_{ij}) - \frac{b}{n} \Sigma(z_{ij}) = 0$ to get $z_{ij} = \log \frac{n \cdot \text{PPR}_{ij}}{b}$, hence $\mathbf{X}\mathbf{X}^\top = \log(\text{PPR}) + \log n - \log b = \mathbf{Y}$. \square

Even though this solution is algebraically impossible, as it implies approximating a non-symmetric matrix by a symmetric one, it provides a useful matrix whose covariance we can sketch.

5.2.2 FREDE ALGORITHM

Since the matrix $\mathbf{Y} = \mathbf{X}\mathbf{X}^\top$ has equal row and column ranks, we rewrite the decomposition commutatively, as $\mathbf{Y} = \log(\text{PPR}) + \log n - \log b = \mathbf{X}^\top \mathbf{X}$. We keep the bias parameter b equal to 1, as in NetMF, and apply FD to obtain a $d \times n$ sketch-based embedding \mathbf{W} by processing rows of \mathbf{Y} . Algorithm 2 presents the details of FREDE and Figure 5.1 shows its workflow; it computes rows of the PPR matrix, and hence of the transformed \mathbf{Y} , by sampling, applies the sVD-based FD sketching process periodically with each d rows it processes (Lines 8–12), and returns embeddings with guarantees at any time (Lines 14–15). We keep track of singular values in $\hat{\Sigma}$ alongside the sketch so as to avoid performing sVD upon a request for output; as in [QDM⁺18], we multiply by $\sqrt{\hat{\Sigma}}$ at output time (Line 15), whereas a covariance-oriented sketcher would use $\hat{\Sigma}$. The time to process *all* n nodes with $\mathcal{O}(n/d)$ sVD iterations costing $\mathcal{O}(d^2n)$ is $\mathcal{O}(dn^2)$.

Sketch-based embeddings inherit the covariance error bounds of sketching (Section 5.1), which hold anytime, even after processing only an arbitrary subset of rows. Thus, FREDE embeddings inherit the anytime error guarantees

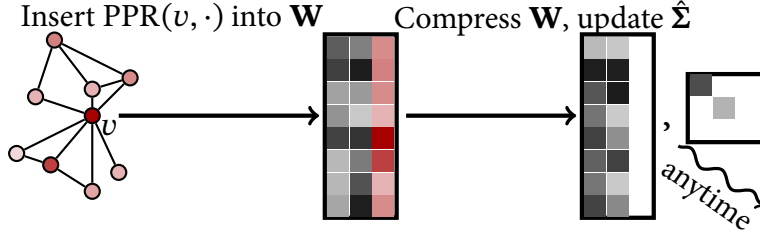


Figure 5.1: Workflow: FREDE iteratively samples transformed PPR rows, periodically compresses the derived sketch and derives singular values by SVD, and returns an embedding with error guarantees at any time.

of Frequent Directions, which are valid after materializing only part of the similarity matrix, and superior to those of other sketch-based embeddings; it achieves $ce \leq \epsilon$ on the submatrix $\mathbf{S}_{[s]}$ built from any size- s subset of processed rows (nodes) when $d = \mathcal{O}(n/\epsilon)$ [GLPW16], independently of s . In Section 5.3.8 we show that FREDE outperforms other sketch-based embeddings in anytime node classification.

Algorithm 2 FREDE

```

1: function FREDE( $G, n, d$ )
2:    $W \leftarrow \text{zeros}(2d, n)$  ▷ all zeros matrix  $W \in \mathbb{R}^{2d \times n}$ 
3:    $\hat{\Sigma} \leftarrow I(2d)$  ▷ diagonal identity matrix  $\hat{\Sigma} \in \mathbb{R}^{2d \times 2d}$ 
4:   for  $v \in V$  do
5:      $x \leftarrow \text{PersonalizedPageRank}(v)$ 
6:      $y \leftarrow \log x + \log n$  ▷ PPR-like similarity row
7:     Insert  $y$  into the last zero valued row of  $W$ 
8:     if  $W$  has no zero valued rows then
9:        $U, \Sigma, V^T \leftarrow \text{SVD}(\hat{\Sigma}W), \sigma \leftarrow \Sigma_{d,d}$ 
10:       $\hat{\Sigma}_{:,d} \leftarrow \sqrt{\max(\Sigma_{:,d}^2 - \sigma^2 I_d, 0)}$  ▷ set  $d^{\text{th}}$  row of  $\hat{\Sigma}$  to 0
11:       $\hat{\Sigma}_d \leftarrow I_d$  ▷ set last  $d$  entries of  $\hat{\Sigma}$  to 1
12:       $W_{:,d} \leftarrow V_{:,d}^T, W_{d,:} \leftarrow \mathbf{0}_{d \times n}$  ▷ zero last  $d$  rows of  $W$ 
13:   return  $\hat{\Sigma}, W_{:,d}$ 
14: function GetEmbedding( $k \leq d$ ) ▷ Anytime
15:   return  $\sqrt{\hat{\Sigma}}W_{:,k}$  ▷ first  $k$  rows
    
```

5.2.3 PARALLELIZATION AND DISTRIBUTION

The steps of Algorithm 2 may be easily parallelized. In particular, Line 5 could employ approximate PPR [YJK18, YSX⁺20], and Line 9 efficient SVD calculations [HILG09]. Such speedups trade quality for scalability. Furthermore,

FREDE can be efficiently distributed across machines for the sake of scalability, with very small communication overhead and preserving its quality guarantees. This appealing characteristic, unique among related works on embeddings, follows from the mergeability property that FREDE inherits from Frequent Directions. In each machine m , we may create a partial embedding matrix W based on the subset of the nodes available to m , and then merge partial embeddings from t servers, i.e., iteratively sketch their concatenations by Equation 5.1 in a hierarchical fashion, incurring a $\log_2 t$ time complexity factor.

5.3 EXPERIMENTS

The primary advantage of FREDE is its anytime character, i.e., its ability to derive embeddings by processing only a fraction of similarity matrix rows. On that front, it may only be compared against other sketch-based embeddings. Here, we also compare FREDE on qualitative performance in data science tasks against other node embeddings to corroborate its practical impact.

5.3.1 COMPARED METHODS

As we have established that nonlinear embeddings outperform linear-transform-based ones in the previous chapter, we evaluate FREDE against three representative state-of-the-art node embeddings based on nonlinear transforms, the classic DeepWalk, neural VERSE, and factorization-based NetMF, three sketching baselines, and exact matrix factorization by SVD:

- DeepWalk [PARS14] learns an embedding by sampling fixed-length random walks from each node and applying word2vec-based learning on those walks; despite intensive research on node embeddings, DeepWalk remains competitive when used with time-tested default parameters [TMKM18]: walk length $t = 80$, number of walks per node $\gamma = 80$, and window size $T = 10$; we use these values.
- VERSE trains a single-layer neural network to learn the PPR similarity measure via sampling, with default parameters $\alpha = 0.85$ and `nsamples` = 10^6 .

<i>dataset</i>	Size			Statistics	
	$ V $	$ E $	$ \mathcal{L} $	Avg. deg.	Density
PPI	4k	77k	50	19.9	5.1×10^{-3}
POS	5k	185k	40	38.7	8.1×10^{-3}
BlogCat	10k	334k	39	64.8	6.3×10^{-3}
CoCit	44k	195k	15	8.9	2.0×10^{-4}
CoAuthor	52k	178k	—	6.9	1.3×10^{-4}
VK	79k	3M	—	34.1	8.7×10^{-4}
Flickr	80k	12M	195	146.5	1.8×10^{-3}
YouTube	1.1M	3M	47	5.2	9.2×10^{-6}

Table 5.1: Dataset characteristics: number of vertices $|V|$, number of edges $|E|$; number of node labels $|\mathcal{L}|$; average node degree; density defined as $|E|/\binom{|V|}{2}$.

- NetMF [QDM⁺18] performs sVD on the closed-form DeepWalk matrix. We use the optimal method, NetMF-small; as it is not scalable, we evaluate it on our three smallest datasets, using the same parameters as in DeepWalk, and bias $b = 1$ as in the original paper; NetMF suffices for a quality comparison with NetSMF [QDM⁺19], as the former represents the full version of the latter.

Sketching baselines. We additionally compare with Hashing, Random Projections and Sampling, three high-performance baseline sketching methods (Section 5.1), computing the sketch and filtering singular value as in FREDE. Our Random Projections baseline is a refined variant of [ZCL⁺18], substituting a crude higher-order matrix approximation with the row-update random projections sketching algorithm applied on the transformed PPR matrix.

On the three smallest datasets, we are able to compare against the exact sVD decomposition of the nonlinearly transformed PPR matrix Y with the same parameters as in FREDE.

5.3.2 PARAMETER SETTINGS

We set embedding dimension $d = 128$ unless indicated otherwise. For sVD, we use the gesdd routine in the Intel MKL library. For classification we use LIBLINEAR [FCH⁺08]. We repeat each experiment 10 times and evaluate each embedding 10 times.

5.3.3 DATASETS

We experiment on 8 publicly available real datasets:

- PPI [SBR⁺06, GL16]: a protein-protein interaction dataset, where labels represent hallmark gene sets of specific biological states.
- POS [Mah11, GL16]: a word co-occurrence network built from Wikipedia data. Labels tag parts of speech induced by Stanford NLP parser.
- BlogCat [ZL09, TLo9b]: a social network of bloggers from the blogcatalog website. Labels represent self-identified topics of blogs.
- CoCit [mag16, TMKM18]: a paper citation graph generated from the Microsoft Academic graph, featuring papers published in 15 major data mining conferences. We use conference identifiers as labels.
- CoAuthor [mag16, TMKM18]: a coauthorsip graph generated from the Microsoft Academic graph. We use snapshots from 2014 and 2016 for link prediction.
- VK [TMKM18]: a Russian all-encompassing social network. Labels represent user genders. We use snapshots from November 2016 and May 2017 for link prediction.
- Flickr [ZL09, TLo9b]: a photo-sharing social network, where labels represent user interests, and edges messages between users.
- YouTube [ZL09, TLo9b]: a video-based social network; labels indicate genre interests.

Table 5.1 summarizes the data characteristics. All algorithms are implemented in Python² and ran on a 2×20-core Intel E5-2698 v4 CPU machine with 384GB RAM and a 64GB memory constraint.

2. Code in the supplementary material; we will open-source the code on GitHub.

5.3.4 SKETCHING QUALITY

As a preliminary test, we assess our choice of sketching backbone against other sketching algorithms and the optimal rank- k covariance approximation obtained by SVD on the full similarity matrix, $\tilde{\mathbf{S}}^T \tilde{\mathbf{S}} = \mathbf{V}_d \Sigma_d^2 \mathbf{V}_d^T$. Figure 5.2 reports the covariance error ce on PPI data, vs. the dimensionality d . We are able to outperform the other sketching algorithms by at least 2 orders of magnitude and, as d grows, it converges to the optimal SVD solution. This

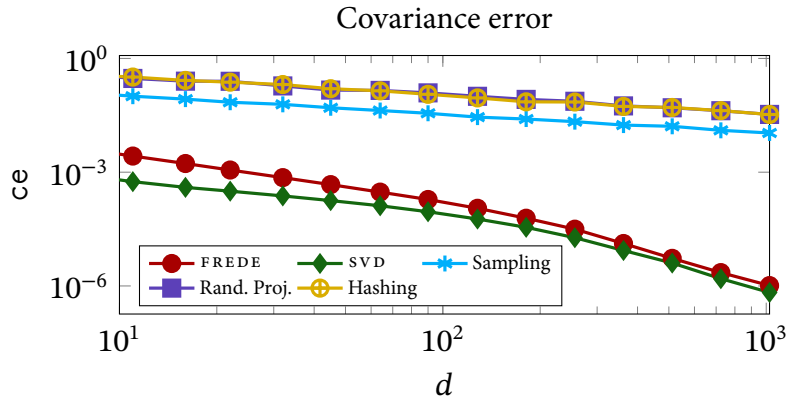


Figure 5.2: Covariance error vs. dimensionality d ; FREDE approaches SVD, which yields optimal covariance error.

result reconfirms that the advantages of FD versus other sketching methods transfer well to the domain of node embeddings. For the sake of completeness, we keep comparing to other sketching methods in the rest of our study, as performance may vary depending on the downstream data science task.

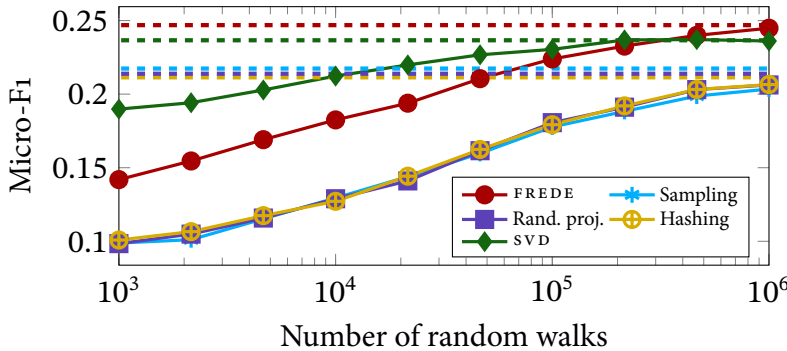


Figure 5.3: Classification performance of sketching algorithms on PPI data wrt. number of walks to compute PPR.

5.3.5 PPR APPROXIMATION

Figure 5.3 shows the performance of sketching algorithms on a node classification task (predicting correct labels) vs. the number of random walks for PPR approximation. As we can see, FREDE consistently outperforms sketching baselines and reaches the exact-PPR solution with 10^6 walks. This result indicates that we can achieve performance obtained using the exact PPR values in downstream tasks even without computing such PPR values with high precision.

Table 5.2: Micro-F1 classification, PPI data.

<i>method</i>	<i>labelled nodes, %</i>				
	10%	30%	50%	70%	90%
DeepWalk	16.33	19.74	21.34	22.39	23.38
NetMF	18.58	22.01	23.87	24.65	25.30
VERSE	16.45	19.89	21.64	23.08	23.84
FREDE	19.56	23.11	24.38	25.11	25.52
SVD	18.31	22.12	23.66	25.03	25.78
Rand. Proj.	16.80	19.99	21.45	22.38	23.14
Sampling	16.25	19.55	20.93	21.85	22.68
Hashing	16.73	19.97	21.51	22.43	23.44

Table 5.3: Micro-F1 classification, POS data.

<i>method</i>	<i>labelled nodes, %</i>				
	10%	30%	50%	70%	90%
DeepWalk	43.42	47.12	48.96	49.86	50.18
NetMF	43.42	46.98	48.52	49.23	49.72
VERSE	40.80	44.70	46.60	47.65	48.24
FREDE	46.59	49.23	50.45	51.02	51.30
SVD	44.69	48.86	50.57	51.53	52.20
Rand. Proj.	40.24	43.87	45.65	46.43	47.18
Sampling	40.35	43.80	45.39	46.30	46.69
Hashing	40.17	43.88	45.44	46.35	46.79

Table 5.4: Micro-F1 classification, BlogCat data.

<i>method</i>	<i>labelled nodes, %</i>				
	10%	30%	50%	70%	90%
DeepWalk	36.22	39.84	41.22	42.06	42.53
NetMF	36.62	39.80	41.05	41.70	42.17
VERSE	35.82	40.06	41.63	42.63	43.14
FREDE	35.69	38.88	39.98	40.54	40.75
SVD	37.60	40.99	42.10	42.66	43.47
Rand. Proj.	30.82	34.43	35.81	36.52	37.16
Sampling	29.44	32.32	33.41	34.04	34.29
Hashing	30.81	34.36	35.82	36.65	37.28

<i>method</i>	<i>labelled nodes, %</i>				
	1%	3%	5%	7%	9%
DeepWalk	37.22	40.34	41.72	42.59	43.16
VERSE	38.95	41.20	42.55	43.41	44.01
FREDE	42.46	44.56	45.39	45.84	46.17
Rand. Proj.	40.89	42.63	43.63	44.32	44.78
Sampling	40.84	42.97	43.93	44.49	44.91
Hashing	40.86	42.66	43.65	44.29	44.83

Table 5.5: Micro-F1 classification, CoCit data.

<i>method</i>	<i>labelled nodes, %</i>				
	1%	3%	5%	7%	9%
DeepWalk	32.39	36.02	37.41	38.15	38.70
VERSE	30.08	34.22	36.06	37.11	37.83
FREDE	30.90	32.98	33.86	34.48	34.88
Rand. Proj.	28.92	32.21	33.82	34.76	35.49
Sampling	28.46	30.97	32.08	32.75	33.24
Hashing	29.07	32.23	33.77	34.75	35.48

Table 5.6: Micro-F1 classification, Flickr data.

<i>method</i>	<i>labelled nodes, %</i>				
	1%	3%	5%	7%	9%
DeepWalk	37.96	40.54	41.75	42.60	43.37
VERSE	38.04	40.50	41.72	42.59	43.33
FREDE	34.51	37.37	38.78	39.40	39.95
Rand. Proj.	33.88	36.10	37.23	37.94	38.38
Sampling	33.97	35.66	36.37	37.19	37.71
Hashing	32.64	35.64	36.92	37.46	38.13

Table 5.7: Micro-F1 classification, YouTube data.

5.3.6 NODE CLASSIFICATION

Tables 5.2–5.7 report classification results in terms of the Micro-F1 measure as it is common in the literature [PARS14, TQW+15]; Macro-F1 results were similar. The experiment features `SVD` where it runs within 64GB. For each dataset, we repeat the experiment 10 times and report the average. Surprisingly, on `PPI` and `POS`, `FREDE` outperforms its exact counterpart, `SVD`, and consistently supersedes its sketching counterparts across all datasets.

Figure 5.4: Classification performance of FREDE with varying percentage of the graph as input on three datasets.

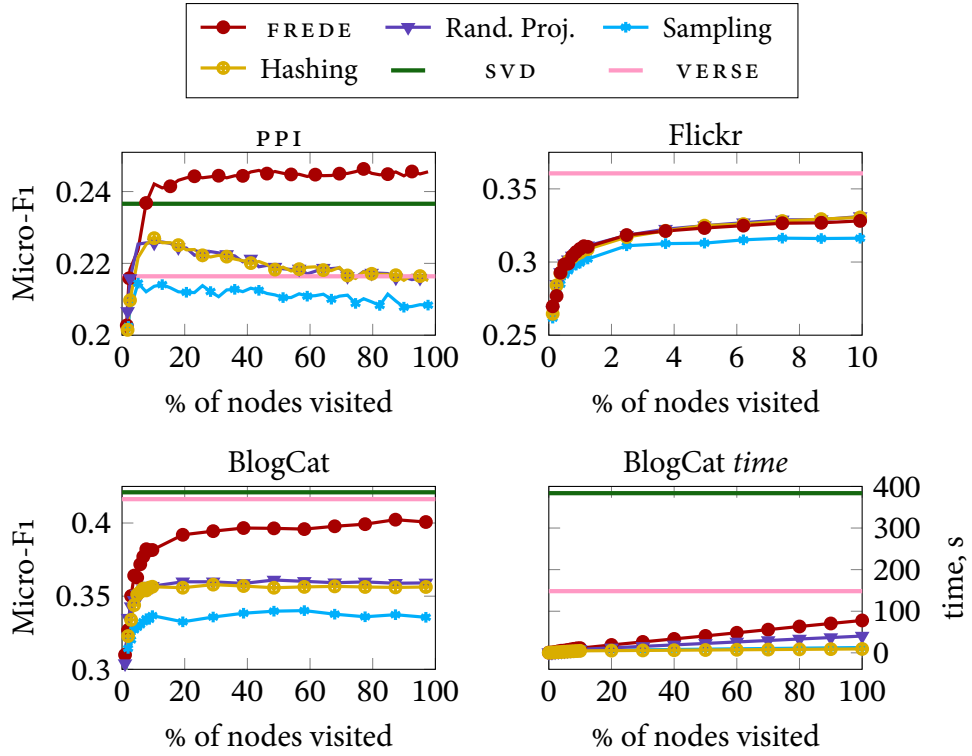


Table 5.8: Link prediction accuracy, CoAuthor data.

<i>method</i>	Average	Concat	Hadamard	L1	L2
DeepWalk	68.97	68.43	66.61	<u>78.80</u>	77.89
VERSE	79.62	79.25	<u>86.27</u>	75.15	75.32
FREDE	81.28	80.95	<u>86.83</u>	81.70	82.37
Rand. Proj.	80.81	80.54	<u>86.73</u>	80.79	81.42
Sampling	80.98	80.74	<u>86.45</u>	79.53	79.51
Hashling	80.84	80.48	<u>86.66</u>	80.59	81.33
Baseline			77.53		

Table 5.9: Link prediction accuracy, VK data.

<i>method</i>	Average	Concat	Hadamard	L1	L2
DeepWalk	69.98	69.83	69.56	<u>78.42</u>	77.42
VERSE	74.56	74.42	<u>80.94</u>	77.16	77.47
FREDE	74.68	74.59	<u>77.63</u>	74.25	73.60
Rand. Proj.	74.41	74.27	<u>77.01</u>	74.33	74.56
Sampling	74.38	74.27	<u>76.82</u>	72.26	71.95
Hashing	74.36	74.27	<u>76.86</u>	74.30	74.56
Baseline			78.84		

5.3.7 LINK PREDICTION

Link prediction is the task of predicting the appearance of a link between pairs of nodes in a graph. Tables 5.8 and 5.9 report link prediction accuracy (predicting the appearance of a link) on CoCit and VK by a logistic regression classifier on edge features. As a baseline, we use common link prediction features (node degree, number of common neighbors, Adamic-Adar index, Jaccard coefficient, and preferential attachment). We represent absent links in the training data by negative sampling, and use 50% of links for training and the remaining 50% for testing. We observe that FREDE outperforms all methods on CoCit, and all sketching baselines on VK. Surprisingly, sketching baselines perform better than state-of-the-art node embeddings on CoCit.

5.3.8 ANYTIME CLASSIFICATION

We study anytime operation (Section 5.2.2) on node classification using 50% of nodes for training and processing PPR rows in random order. Figure 5.4 presents results for PPR-based methods on three datasets. We observe that FREDE outperforms both VERSE and SVD on PPI, as in Table 5.2, after processing only 10% of similarities; its downstream performance grows with the number of nodes visited on all datasets, while that of other sketching baselines drops on PPI data; FREDE also performs competitively on Flickr (we examine up to 10% of nodes, as Random Projections was inefficient; SVD did not run within 64GB) and BlogCat, as in Tables 5.4 and 5.6. The rightmost plot in Figure 5.4 shows runtime on BlogCat; remarkably, while sketchers' runtime grows linearly, those of one-off methods stand apart. These results also illustrate that embedding merging preserves the downstream embedding quality; as Equation 5.1 shows, merging two embeddings amounts to sketching their concatenation; therefore, the sketch operation Algorithm 2 periodically performs with each new d similarity matrix rows it processes can also be viewed as a merge operation.

5.4 SUMMARY

We observed that, since node embeddings aim to preserve similarity matrix covariance, row-wise sketching techniques are naturally suited to that end. We applied a state-of-the-art sketcher, Frequent Directions, on a matrix-factorization interpretation of a state-of-the-art nonlinear-transform embedding, VERSE, to craft FREDE: a linear-space node embedding that allows for scalable data science operations on graph data, as well as for anytime and distributed computation *with error guarantees*. Besides its anytime character, FREDE achieves almost as low covariance error as the exact SVD solution and stands its ground against previous node embeddings even after processing as little as 10% of similarity matrix rows; therefore, it promises significant practical impact.

THIS chapter introduces SnapEmbed, an efficient method for generating single-node representations using local PageRank computations. We theoretically prove that our approach produces representations that are globally consistent in sublinear time. We demonstrate this empirically via extensive experiments on real-world datasets with over a billion edges. Our experiments confirm that SnapEmbed requires drastically less compute time (over 9,000 times faster) and less memory (by over 8,000 times) to produce a single node’s embedding than traditional methods, including DeepWalk, node2vec, VERSE, and FastRP. We also show that our method produces high-quality representations, demonstrating results that meet or exceed the state of the art for unsupervised representation learning on node classification and link prediction tasks.

Node embedding models typically assume that the graph fits in memory [PARS14] and require representations for all nodes to be generated. However, in many real-world applications, it is often the case that graph data is large but also scarcely annotated. For example, the Friendster social graph [YL15] has only 30% nodes assigned to a community, from its total 65M entries. At the same time, many applications of graph embeddings such as classifying a data item only require one current representation for the item itself, and eventually representations of labeled nodes. Therefore, computing a full graph embedding is at worst infeasible and at best inefficient.

These observations motivate the problem which we study in this chapter—the *Local Node Embedding* problem. In this setting, we restrict the embedding for a node to only local structural information and prohibit access to other nodes’ representations in the graph or reliance on a trained global model state. We also require that a local method needs to produce embeddings consistent with all other node’s representations, so final representations can be used in the same downstream tasks that graph embeddings have proved adept at in the past. While this model may seem excessively restrictive, we will show that adhering to these limitations can produce very fruitful results. We show that

this setting is feasible. Our solution can generate embeddings for nodes in massive graphs in an incredibly quick and memory-efficient manner; they are competitive with state-of-the-art methods.

In this work, we introduce SnapEmbed, an efficient method to generate local node embeddings *on the fly* in sublinear time which are globally consistent. Built on previous works linking embedding learning methods to large similarity matrices factorization [TMM⁺20b, QDM⁺18], our method leverages a high-order ranking matrix based on PPR as foundations on which local node embeddings are computed with local hashing [WDL⁺09]. We offer theoretical guarantees on the locality of the computation, as well as the proof of the global consistency of the generated embeddings. We show empirically that our method is able to produce high-quality representations on par with state-of-the-art methods, with efficiency several orders of magnitude better in clock time and memory consumption: running 9,000 times faster and using 8,000 times less memory on the largest graphs that contenders can process.

6.1 PROBLEM STATEMENT

We consider the problem of embedding a single node in a graph quickly. More formally, we consider what we term the LNE problem: given a graph G and any node v , return a globally consistent structural representation for v using only local information around v , in time sublinear to the size of the graph.

A solution to the LNE problem should possess two following properties:

1. *Locality*. The embeddings for a node are computed locally, i.e. the embedding for a node can be produced using only local information and in time independent of the total graph size.
2. *Global Consistency*. A local method must produce embeddings that are globally consistent (i.e. able to relate each embedding to each other, s.t. distances in the space preserve proximity).

While many node embedding approaches have been proposed [CPARS18], to the best of our knowledge we are the first to examine the local embedding problem. Furthermore, no existing methods for positional representations of nodes meet these requirements.

Locality. Node embedding methods that rely on information aggregated from local subgraphs (e.g., sampled by a random walk), such as DeepWalk, node2vec, or VERSE, do not meet our locality requirement. Specifically, they also require the representations of all the nodes around them, resulting in a dependency on information from all nodes in the graph (in addition to space complexity $O(nd)$ where d is the embedding dimension) to compute a single representation. Classical random-projection based methods also require access to the full adjacency matrix in order to compute the higher-order ranking matrix. We briefly remark that even methods capable of local attributed subgraph embedding (such as GCN or DGI) also do not meet this definition of locality, as they require a global training phase to calibrate their graph pooling functions.

Global Consistency. This property allows embeddings produced by local node embedding to be used together, perhaps as features in a model. While existing methods for node embeddings are global ones that implicitly have global consistency, this property is not trivial for a local method to achieve. One exciting implication of a local method that is globally consistent is that it can wait to compute a representation until it is actually required for a task. For example, in a production system, one might only produce representations for immediate classification when they are requested.

We propose our approach satisfying these properties in Section 6.2, and experimentally illustrate the satisfied properties in Section 6.3.

6.2 METHOD

Here we outline our proposed approach for local node embedding. We begin by discussing the connection between a recent embedding approach and matrix factorization. Then, using this analysis, we propose a global embedding method based on a random projection of the PPR matrix. We note that this approach has a tantalizing property—it can be decomposed into entirely local operations per node. With this observation in hand, we present our solution, SnapEmbed. Finally, we analyze the algorithmic complexity of our approach, showing that it is both a local algorithm (which runs in time sublinear to the size of G) and that the local representations are globally consistent.

6.2.1 GLOBAL EMBEDDING USING PPR

Our node embedding algorithm from Chapter 4, VERSE, proposes to learn node embeddings using a neural network that encodes their Personalized PageRank similarity. The VERSE objective function, in the form of Noise Contrastive Estimation [GH10], is:

$$\mathcal{L} = \sum_{i=1}^n \sum_{j=1}^n \left[\text{PPR}_{ij} \log \sigma(\mathbf{x}_i^\top \mathbf{x}_j) + b \mathbb{E}_{j' \sim \mathcal{U}} \log \sigma(-\mathbf{x}_i^\top \mathbf{x}_{j'}) \right], \quad (6.1)$$

where PPR is the PPR matrix, σ is the sigmoid function, b is the number of negative samples, and \mathcal{U} is a uniform noise distribution from which negative samples are drawn. Like many SkipGram [MSC⁺13]-style methods, we can then show that this learning process can be linked to matrix factorization, and get the following lemma (proved in the previous Chapter):

Lemma 3. VERSE implicitly factorizes the matrix $\log(\text{PPR}) + \log n - \log b$ into $\mathbf{X}\mathbf{X}^\top$, where n is the number of nodes and b is the number of samples.

6.2.1.1 HASHING FOR GRAPH EMBEDDING

Lemma 3 provides an incentive to find an efficient alternative to factorize the large similarity matrix $\mathbf{M} = \log(\text{PPR}) + \log n - \log b$. We turn our attention to random-projection based methods, as previous work [ZCL⁺18, CST⁺19] has established this dimensionality-reduction technique as being highly effective for scalable graph embedding. Our choice of projection requires two important properties: (i) providing an unbiased estimator for the inner-product, and (ii)

requiring less than $\mathcal{O}(n)$ memory. The first property is essential to ensure we have a good sketch of \mathbf{M} for the embedding, while the second one keeps our complexity per node sublinear.

In order to meet both requirements, we propose to use the hash kernel [WDL⁺09] to project PPR. In short, given two global hash functions $h_d : \mathbb{N} \rightarrow \{0, \dots, d-1\}$ and $h_{\text{sgn}} : \mathbb{N} \rightarrow \{-1, 1\}$ sampled from universal hash families \mathbb{U}_d and $\mathbb{U}_{-1,1}$ respectively, the hashing kernel $H_{h_d, h_{\text{sgn}}} : \mathbb{R}^n \rightarrow \mathbb{R}^d$ applied to an input vector \mathbf{x} yields $\mathbf{h} = H_{h_d, h_{\text{sgn}}}(\mathbf{x})$ where $\mathbf{h}_i = \sum_k x_k h_{\text{sgn}}(k) \mathbb{1}[h_d(k) = i]$.

We note that although $H_{h_d, h_{\text{sgn}}}$ is proposed for vectors, it can be trivially extended to matrix \mathbf{M} when applied to each row vector of that matrix, e.g., by defining $H_{h_d, h_{\text{sgn}}}(\mathbf{M})_{i,:} \equiv H_{h_d, h_{\text{sgn}}}(\mathbf{M}_{i,:})$. Also, as a random projection, the operator $H_{h_d, h_{\text{sgn}}}(\cdot) \in \{-1, 0, 1\}^{n \times d}$ is sparse with n non-zero entries, exactly one per each row.

Lemma 4 (restated from [WDL⁺09]). *The hash kernel is unbiased:*

$$\mathbb{E}_{h_d \sim \mathbb{U}_d, h_{\text{sgn}} \sim \mathbb{U}_{-1,1}} \left[H_{h_d, h_{\text{sgn}}}(\mathbf{x})^\top H_{h_d, h_{\text{sgn}}}(\mathbf{x}) \right] = \mathbf{x}^\top \mathbf{x}$$

Lemma 5. *The space complexity of $H_{h_d, h_{\text{sgn}}}$ is $\mathcal{O}(1)$ and:*

$$\mathbb{E}_{h_d \sim \mathbb{U}_d, h_{\text{sgn}} \sim \mathbb{U}_{-1,1}} \left[H_{h_d, h_{\text{sgn}}}(\mathbf{M}) H_{h_d, h_{\text{sgn}}}(\mathbf{M})^\top \right] = \mathbf{M} \mathbf{M}^\top$$

Proof. We note that the space complexity required to store a hash function from a universal family is $\mathcal{O}(1)$. Indeed, one can choose a universal hash family such that its elements are uniquely determined by a fixed choice of keys. As an example, the multiplication hash function ([CLRS09]) $h^A(x) = \lfloor n(xA \bmod 1) \rfloor$ requires constant memory to store the key $A \in (0, 1)$.

In order to prove the projection provides unbiased dot-products, considering the expectation per entry, we have:

$$\begin{aligned} & \mathbb{E}_{h_d \sim \mathbb{U}_d, h_{\text{sgn}} \sim \mathbb{U}_{-1,1}} \left[\left(H_{h_d, h_{\text{sgn}}}(\mathbf{M}) H_{h_d, h_{\text{sgn}}}(\mathbf{M})^\top \right)_{i,j} \right] \\ &= \mathbb{E}_{h_d \sim \mathbb{U}_d, h_{\text{sgn}} \sim \mathbb{U}_{-1,1}} \left[\left(H_{h_d, h_{\text{sgn}}}(\mathbf{M}_i) H_{h_d, h_{\text{sgn}}}(\mathbf{M}_j)^\top \right) \right] \\ &= \mathbf{M}_i \mathbf{M}_j^\top \quad \text{From Lemma 4} \\ &= (\mathbf{M} \mathbf{M}^\top)_{i,j}, \end{aligned}$$

which holds for all i, j pairs. \square

Algorithm 3 Global Node Embedding using Personalized PageRank

Input: graph G , dimension d , PPR precision ϵ , hash functions h_d, h_{sgn}
Output: embedding matrix \mathbf{W}

```

1: function GraphEmbed( $G, d, \epsilon, \alpha, h_d, h_{\text{sgn}}$ )
2:   PPR  $\leftarrow$  CreatePPRMatrix( $G, \epsilon, \alpha$ )
3:    $\mathbf{W} = \mathbf{0}_{n \times d}$ 
4:   for  $\pi_i$  in PPR do
5:     for  $r_j$  in  $\pi_i$  do
6:        $\mathbf{W}_{i, h_d(j)} += h_{\text{sgn}}(j) \times \max(\log(r_j * n), 0)$ 
7:   return  $\mathbf{W}$ 

```

Our algorithm for global node embedding is presented in Algorithm 3. It involves first computing the PPR matrix PPR (Line 2) with a generic approach (CreatePPRMatrix), which takes a graph and ϵ , the desired precision of the approximation. We note that any of the many proposed approaches for computing such a matrix (e.g., [JW03, ACL07, LBGS14]) can be used for this calculation. As the PPR matrix could be dense, the same could be said about the implicit matrix \mathbf{M} . We enforce sparsity by applying the max operator, though we note that for high enough ϵ this has no effect. Also, from the implicit target matrix we remove the constant $\log b$. In lines (4-6) we use the provided hash function to accumulate each value inside the corresponding embedding dimension.

We note that interestingly, the projection operation uses only information from each node's individual PPR vector π_i to compute its representation. In the following section, we will show that local calculation of the PPR can be utilized to develop an entirely local algorithm for node embedding.

6.2.2 LOCAL NODE EMBEDDING VIA SNAPEMBED

Having a local projection method, all that we require is a procedure that can calculate the PPR vector for a node in time sublinear to the size of the graph. Specifically, for SnapEmbed we propose that the CreatePPRMatrix operation consists of invoking the SparsePPR routine from Andersen et al. [ACL07] once for each node i . This routine is an entirely local algorithm for efficiently constructing π_i , the PPR vector for node i , which offers strong guarantees. We derive the following lemma from [ACL07]:

Algorithm 4 SparsePPR cf. [ACLo7]**Input:** node v , graph G , precision ϵ , return probability α **Output:** Embedding vector π

```

1: function SparsePPR( $v, G, \epsilon, \alpha$ )
2:    $\mathbf{r} \leftarrow \mathbf{0}_n$  (sparse),  $\pi \leftarrow \mathbf{0}_n$  (sparse)
3:    $\mathbf{r}[v] = 1$ 
4:   while  $\exists w \in G, \mathbf{r}[w] > \epsilon \times \text{deg}(w)$  do
5:      $\hat{r} \leftarrow \mathbf{r}[w]$ 
6:      $\pi[w] \leftarrow \pi[w] + \alpha \hat{r}$ 
7:      $\mathbf{r}[w] \leftarrow \frac{(1-\alpha)\hat{r}}{2}$ 
8:      $\mathbf{r}[u] \leftarrow \mathbf{r}[u] + \frac{(1-\alpha)\hat{r}}{2 \text{deg}(w)}, \forall (w, u) \in G$ 
9:   return  $\pi$ 

```

Theorem 3 (restated from [ACLo7]). *The SparsePPR(v, G, ϵ, α) (4) algorithm has the following properties. For any starting vector v , any constant $\alpha \in (0, 1]$, and any constant $\epsilon \in (0, 1]$, the algorithm computes an ϵ -approximate PPR vector p . Furthermore, the support of p satisfies $\text{vol}(\text{Supp}(p)) \leq \mathcal{O}(1/(1-\alpha)\epsilon)$, and the running time of the algorithm is $\mathcal{O}(1/\alpha\epsilon)$.*

Lemma 6 (ref. Lemma 5). *The SnapEmbed($v, G, d, \epsilon, \alpha$) algorithm computes the local embedding of a node v by exploring at most $\mathcal{O}(1/(1-\alpha)\epsilon)$ nodes in the neighborhood of v .*

Proof. First recall that the only operation that explores the graph in SnapEmbed is SparsePPR, which explores a node w in the graph if and only if a neighbor of w has a positive score and so it is part of the support of π . Furthermore, at the beginning of the algorithm, only v is active and executes a push operation. Then, every node explored by the algorithm is connected to v via a path composed only by the nodes with π score larger than 0. So its distance from v is bounded by the support of the π vector that is $\mathcal{O}(1/(1-\alpha)\epsilon)$, cf. Theorem 3. \square

SnapEmbed, our algorithm for local node embedding, is presented in Algorithm 5. As we will show, it is a self-contained solution for the local node embedding problem that can generate embeddings for individual nodes extremely efficiently. Notably, per Lemma 6, the local area around v explored by SnapEmbed is independent of n . Therefore the algorithm is strictly local.

Algorithm 5 SnapEmbed

Input: node v , graph G , dimension d , PPR precision ϵ , PPR parameter α , hash functions h_d, h_{sgn}

Output: embedding vector \mathbf{w}

```

1: function SnapEmbed( $v, G, d, \epsilon, \alpha, h_d, h_{sgn}$ )
2:    $\pi_v \leftarrow \text{SparsePPR}(v, G, \epsilon, \alpha)$ 
3:    $\mathbf{w} \leftarrow \mathbf{0}_d$ 
4:   for  $r_j$  in  $\pi_v$  do
5:      $\mathbf{w}_{h_d(j)} += h_{sgn}(j) \times \max(\log(r_j * n), 0)$ 
6:   return  $\mathbf{w}$ 

```

6.2.2.1 ANALYSIS

In this subsection, we prove some basic properties of our proposed approach. We first show that the run time of our algorithm is local and independent of n , the number of nodes in the graph. Also, we show that our local computations are globally consistent, i.e. the embedding of a node v is the same independently if we compute it locally or if we recompute the embeddings for all nodes in the graph at the same time. Note that we focus on bounding the running time to compute the embedding for a *single* node in the graph. Nonetheless, the global complexity to compute all the embeddings can be obtained by multiplying our bound by n , although it is not the focused contribution of our approach. Following from [ACLo7], we state the next lemma:

Theorem 4. *The SnapEmbed algorithm has time complexity $\mathcal{O}(d + 1/\alpha(1-\alpha)\epsilon)$.*

Proof. The first step of the SnapEmbed is computing the approximate PPR vector. As noted in Theorem 3, this can be done in time $\mathcal{O}(1/\alpha\epsilon)$. We now analyze the second part of our algorithm, projecting the sparse PPR vector into the embedding space. For each non-zero entry r_j of the PPR vector π , we compute hash functions $h_d(j)$, $h_{sgn}(j)$ and $\max(\log(r_j * n), 0)$ in $\mathcal{O}(1)$ time. The total number of iterations is equal to the support size of π , i.e. $\mathcal{O}(1/(1-\alpha)\epsilon)$.

Finally, we note that our algorithm always generates a dense embedding, handling this variable in $\mathcal{O}(d)$ time complexity. However, in practice this term is negligible as $1/e \gg d$. Summing up the aforementioned bounds we get the total running time of our algorithm: $\mathcal{O}(d + 1/\alpha\epsilon + 1/(1-\alpha)\epsilon) = \mathcal{O}(d + 1/\alpha(1-\alpha)\epsilon)$. \square

Outside of the embedding size d , both the time and space complexity of our algorithm depend only on the approximation factor ϵ and the decay factor α . Both are independent of n , the size of the graph, and m , the size of the edge set. Notably, if $\mathcal{O}(1/\alpha(1-\alpha)\epsilon) \in o(n)$, as commonly happen in real world applications, our algorithm has sublinear time w.r.t. the graph size.

Now we turn our attention to the consistency in our algorithm, by showing that for a node v the embeddings computed by $\text{SnapEmbed}(v, G, d, \epsilon, \alpha)$ and $\text{GraphEmbed}(G, d, \epsilon, \alpha)$ are identical. We denote the graph embedding computed by $\text{GraphEmbed}(G, d, \epsilon, \alpha)$ for node v by $\text{GraphEmbed}(G, d, \epsilon, \alpha)_v$, and we have the following statement about global consistency:

Theorem 5. $\text{SnapEmbed}(v, G, d, \epsilon, \alpha)$ output equals $\text{GraphEmbed}(G, d, \epsilon, \alpha)_v$.

Proof. We begin by noting that for a fixed parameterization, the `SparsePPR` routine will compute a unique vector for a given node. Analyzing now the $\mathbf{W}_{v,j}$ entry of the embedding generated by $\text{GraphEmbed}(G, d, \epsilon, \alpha)$, we have:

$$\mathbf{W}_{v,j} = \sum_{r_k \in \pi_v} h_{sgn}(k) \times \max(\log(r_k * n), 0) \mathbb{1}[h_d(k) = j]$$

The entire computation is deterministic and directly dependent only on the hash functions of choice and the indexing of the graph. By fixing the two hash functions h_d and h_{sgn} , we also have that $\mathbf{W}_{v,j} = \mathbf{w}_j^v$ where $\mathbf{w}^v = \text{SnapEmbed}(v, G, d, \epsilon)$, $\forall v \in [0..n - 1]$, $j \in [0..d - 1]$. \square

6.3 EXPERIMENTS

In the light of the theoretical guarantees about the proposed method, we perform extended experiments in order to verify our two main hypotheses:

1. H1. Computing local node-embedding is more efficient, both memory and time wise, than generating a global embedding.
2. H2. The local representations are consistent and of high-quality, being competitive with or surpassing state-of-the-art methods on several tasks.

We assess H1 in Section 6.3.2, in which we measure the efficiency of generating a single node embedding for each method. Then in Section 6.3.3 we validate H2 by comparing our method against the baselines on multiple datasets using tasks of node classification, link prediction and visualization.

6.3.1 EXPERIMENTAL SETTINGS AND DATASETS

To ensure a relevant and fair evaluation, we compare our method against multiple strong baselines, including DeepWalk [PARS14], node2vec [GL16], VERSE [TMKM18], and FastRP [CST⁺19]. Each method was run on a virtual machine hosted on the Google Cloud Platform, with a 2.3GHz 16-core CPU and 128GB of RAM. We ran all baselines on 128 and 512 embedding dimensions. As we expect our method to perform better as we increase the projection size, we performed an auxiliary test with embedding size 2048 for SnapEmbed. We also make the observation that learning-based methods generally do not scale well with an increase of the embedding space. The following are the description and individual parameterization for each method.

- DeepWalk [PARS14]: Constructs node-contexts from random-walks and learns representations by increasing the nodes co-occurrence likelihood by modeling the posterior distribution with hierarchical softmax. We set the number of walks per node and their length to 80, and context window size to 10.
- node2vec [GL16]: Samples random paths in the graph similar to DeepWalk, while adding two parameters, p and q , controlling the behavior of the walk. Estimates the likelihood through negative sampling. We set again the number of walks per node and their length to 80, window size to 10, and the number of negative samples to 5 per node and $p = q = 1$.
- VERSE [TMKM18]: Minimizes objective through gradient descent, by sampling nodes from PPR random walks and negative samples from a noise distribution. We train it over 10^5 epochs and set the stopping probability to 0.15.
- FastRP [CST⁺19]: Computes a high-order similarity matrix as a linear combination of multiple-steps transitions matrices and projects it into an embedding space through a sparse random matrix. We fix the linear coefficients to $[0, 0, 1, 6]$ and the normalization parameter -0.65 .
- SnapEmbed (this chapter): Approximate per-node PPR vectors with the damping factor α and precision ϵ , which are projected into the

embedding space using two hash functions. In all our experiments, we set $\alpha = 0.15$ and $\epsilon > \frac{1}{n}$, where n is the number of nodes in the graph.

SnapEmbed Instantiation. As presented in Section 6.2, our implementation of the presented method relies on the choice of PPR approximation used. For instant single-node embeddings, we use the highly efficient PushFlow [ACLo7] approximation that enables us to dynamically load into memory at most $2/(1-\alpha)\epsilon$ nodes from the full graph to compute a single PPR vector π . This is achieved by storing graphs in binarized compressed sparse row format that allows selective reads for nodes of interest. In the special case when a full graph embedding is requested, we have the freedom to approximate the PPR in a distributed manner (we omit this from runtime analysis, as we had no distributed implementations for the baselines, but we note our local method is trivially parallelizable).

Dataset	$ V $	$ E $	$ S $	Source
PPI	3.8k	38k	3.8k	[SBR ⁺ 06]
BlogCat	10k	334k	10k	[TL10]
CoCit	44k	195k	44k	[mag16]
CoAuthor	52k	356k	—	[mag16]
Flickr	81k	5.9M	81k	[TL10]
YouTube	1.1M	3.0M	32k	[TL10]
Amazon2M	2.4M	62M	—	[CLS ⁺ 19]
Orkut	3.0M	117M	110k	[YL15]
Friendster	66M	1806M	—	[YL15]

Table 6.1: Dataset attributes: size of the vertex set $|V|$, edge set $|E|$, labeled vertices $|S|$.

Datasets. We perform our evaluations on 10 datasets, as presented in Table 6.1. Note that on YouTube and Orkut the number of labeled nodes is much smaller than the total. We observe this behavior in several real-world application scenarios, where our method shines the most. The graph datasets we used in our experiments are as follows:

- PPI [SBR⁺06]: Subgraph of the protein-protein interaction for Homo Sapiens species and ground-truth labels represented by biological states. Data originally processed by [GL16].

- BlogCat [TL10]: Network of social interactions between bloggers. Authors specify categories for their blog, which we use as labels.
- *Microsoft Academic Graph* [mag16]: Collection of scientific papers, authors, journals and conferences. Two distinct subgraphs were originally processed by [TMKM18], based on co-authorship (CoAuthor) and co-citations (CoCit) relations. For the latter one, labels are represented by the unique conference where the paper was published.
- Flickr [TL10]: Contact network of users within 195 randomly sampled interest groups.
- YouTube [TL10] Social network of users on the video-sharing platform. Labels are represented by groups of interests with at least 500 subscribers.
- Amazon2m [CLS⁺19]: Network of products where edges are represented by co-purchased relations.
- Orkut [YL15]: Social network where users can create and join groups, used at ground-truth labels. We followed the approach of [TMKM18] and selected only the top 50 largest groups.
- Friendster [YL15]: Social network where users can form friendships with each other. It also allows users to form a group which other members can then join.

6.3.2 PERFORMANCE CHARACTERISTICS

Running time. For each method we report the mean total wallclock time required to generate an embedding ($d = 512$) for a node in each dataset (including graph loading, embedding writing, and any pre-processing). For SnapEmbed, given the method’s locality property, the experiments were repeated 1,000 times per dataset (on random nodes); other methods were run five times per dataset. Detailed results can be found in the supplementary material. As Figure 6.1(a) shows, SnapEmbed is the most efficient and scalable method on a per-node basis, drastically outperforming all the other methods. Compared to the next fastest baseline (FastRP) we are over 9,000 times faster in the largest graph both methods can process.

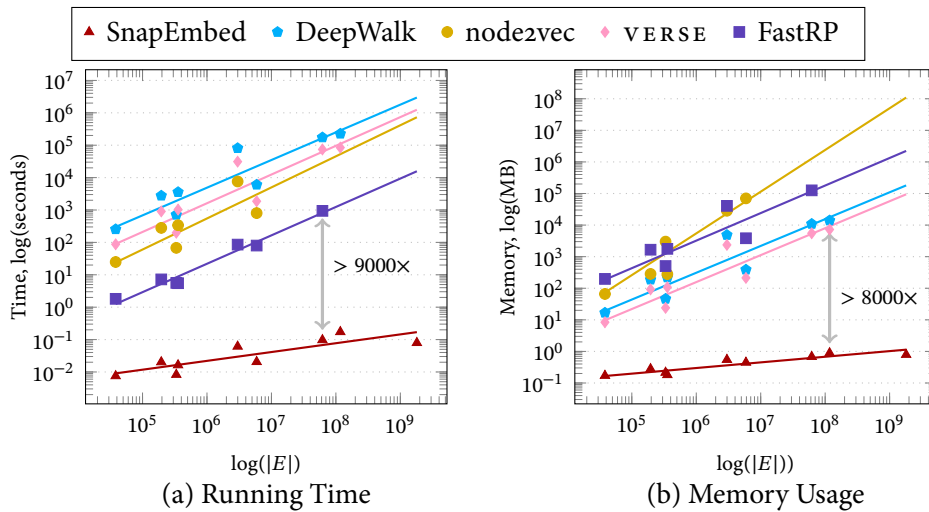


Figure 6.1: Required (a) running time and (b) memory consumption to generate a node embedding ($d=512$) based on the edge count of each graph ($|E|$), with the best line fit. Our method is over 9,000 times faster than the next fastest baseline (FastRP) and uses over 8,000 times less memory than the next most memory-efficient baseline (VERSE), in the largest graph that these baseline methods can process.

Memory Consumption. For each method we also report the total memory consumption [Wol] required to generate an embedding ($d=512$) for a node in the given dataset. For SnapEmbed, given the method’s locality property, the experiments were repeated 1,000 times per dataset with the mean memory consumption reported; other methods were run once. As Figure 6.1(b) shows, SnapEmbed is the most scalable method on a per-node basis having been able to run in all datasets using negligible memory compared to the other methods. Compared to the next most memory-efficient baseline (VERSE) we are over 8,000 times better in the largest graph both methods can process.

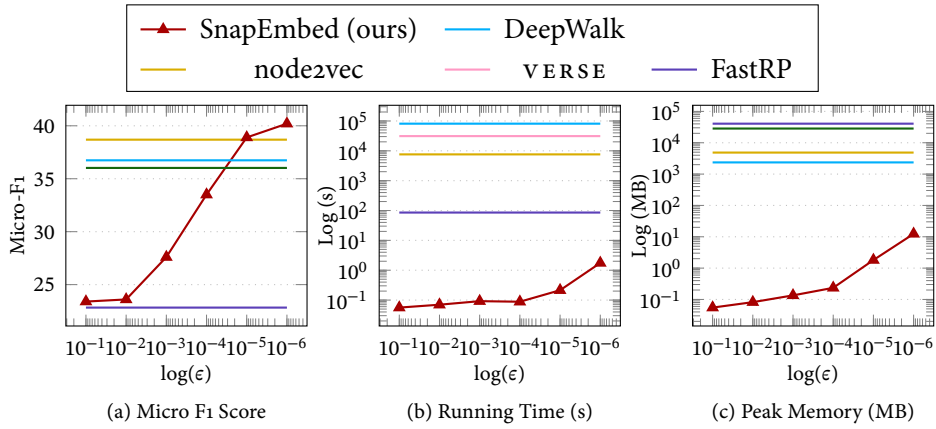
The results of running time and memory analysis confirm hypothesis H1 and show that SnapEmbed has a significant speed and space advantage versus the baselines. The relative speedup continues to grow as the size of the datasets increase. On a dataset with over 1 billion edges (Friendster), we can compute an embedding in 80ms—fast enough for a real-time application!

6.3.3 EMBEDDING QUALITY

Node Classification. This task measures the semantic information in the embeddings by training a classifier on a small fraction of labeled representations. For each method, we perform three different random splits of the data. We use a logistic regression classifier from sklearn [PVG⁺11], except for the case of multi-label classification where we use a one-vs-rest ensemble to rank and

Table 6.2: Average Micro-F1 classification scores and confidence intervals.

<i>Method</i>	PPI	BlogCat	CoCit	Flickr	YouTube	Orkut
DeepWalk	16.08 ± 0.64	32.48 ± 0.35	37.44 ± 0.67	31.22 ± 0.38	38.69 ± 1.17	87.67 ± 0.23
node2vec	15.03 ± 3.18	33.67 ± 0.93	38.35 ± 1.75	29.80 ± 0.67	36.02 ± 2.01	DNC
VERSE	12.59 ± 2.54	24.64 ± 0.85	38.22 ± 1.34	25.22 ± 0.20	36.74 ± 1.05	81.52 ± 1.11
FastRP	15.74 ± 2.19	33.54 ± 0.96	26.03 ± 2.10	29.85 ± 0.26	22.83 ± 0.41	DNC
SnapEmbed	17.67 ± 1.22	33.36 ± 0.67	39.95 ± 0.67	30.43 ± 0.79	40.04 ± 0.97	76.83 ± 1.16

Figure 6.2: The impact of the choice of ϵ on the quality of the resulting embedding (through the Micro-F1 score), average running time and peak memory increase for the YouTube dataset.

select the top- k labels (assuming k is known) following [PARS14]. To simulate the sparsity of labels in the real world, we train on 10% of the available labels for PPI and BlogCat and only 1% for the rest of them.

In Table 6.2 we report the mean Micro-F1 scores with their respective confidence intervals. For each dataset, we perform Welch’s t-test between our method and the best performing contender. We observe that SnapEmbed is remarkably good on the node classification, despite its several approximations and locality restriction. Specifically, on four out of five datasets, no other method is statistically significant above ours, and three of these (PPI, CoCit and YouTube) we achieve the best classification results.

In Figure 6.2, we study how our hyperparameter, the PPR approximation error ϵ , influences both the classification performance, running time, and memory consumption. There is a general sweet spot (around $\epsilon = 10^{-5}$) across datasets where SnapEmbed outperforms competing methods while being orders of magnitude faster.

<i>Method</i>	CoAuthor	BlogCat	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	DNC
VERSE	92.75 \pm 0.73	93.42 \pm 0.35	80.03 \pm 0.99	99.67 \pm 0.18
FastRP	82.19 \pm 2.22	88.68 \pm 0.70	76.30 \pm 1.46	92.12 \pm 0.61
SnapEmbed	90.44 \pm 0.48	92.74 \pm 0.60	82.89 \pm 0.83	99.15 \pm 0.18

Table 6.3: Average ROC-AUC scores and confidence intervals for the link prediction task.

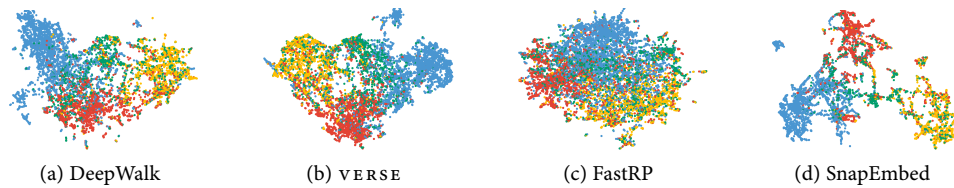


Figure 6.3: CoCit visualization via UMAP ($d=512$). Research areas (● ML, ● DM, ● DB, ● IR).

Link prediction. We conduct link prediction experiments to assess the capability of the produced representations to model hidden connections in the graph. For the dataset which has temporal information (CoAuthor), we select data until 2014 as training data, and split co-authorship links between 2015-2016 in two balanced partitions that we use as validation and test. For the other datasets, we uniformly sample 80% of the available edges as training (to learn embeddings on), and use the rest for validation (10%) and testing (10%). Over repeated runs, we vary the splits. More details about the experimental design are available in the supplementary material. We report results for each method in Table 6.3, which shows average ROC-AUC and confidence intervals for each method. Across the datasets, our proposed method beats all baselines except VERSE, however, we do achieve the best performance on YouTube by a statistically significant margin. We report additional results for node classification (Tables 6.4–6.8) and link prediction (Table 6.9) at the end of this chapter.

Visualization. Figure 6.3 presents multiple UMAP [MHSG18] projections on the CoCit dataset, where we grouped together similar conferences. We note that our sublinear approach is especially well suited to visualizing graph data, as visualization algorithms only require a small subset of points (typically downsampling to only thousands) to generate a visualization for datasets.

The experimental analysis of node classification, link prediction, and visualization show that despite relying on two different approximations (PPR & hashing), SnapEmbed is able to very quickly produce representations which meet or exceed the state of the art in unsupervised representation learning for graph structure, confirming hypothesis H2. We remark that interestingly SnapEmbed seems slightly better at node classifications than link prediction. We suspect that the randomization may effectively act as a regularization which is more useful on classification.

6.4 SUMMARY

This chapter has two main contributions: (i) introducing and formally defining the *Local Node Embedding* problem and (ii) presenting SnapEmbed, a highly efficient method that selectively embeds nodes using only local information, effectively solving the aforementioned problem. As existing graph embedding methods require accessing the global graph structure at least once during the representation generating process, the novelty brought by SnapEmbed is especially impactful in real-world scenarios where graphs outgrow the capabilities of a single machine, and annotated data is scarce or expensive to produce. Embedding selectively only the critical subset of nodes for a task makes more applications feasible in practice and reduces the costs for others.

Furthermore, we show theoretically that our method embeds a single node in space and time sublinear to the size of the graph. We also empirically prove that SnapEmbed is capable of surpassing state-of-the-art methods, while being many orders of magnitude faster than them—our experiments show that we are over 9,000 times faster on large datasets and on a graph over 1 billion edges we can compute a representation in 80ms.

<i>Method</i>	<i>d</i>	<i>Labeled Nodes</i>					
		1.00%		5.00%		9.00%	
		Micro-F1	Macro-F1	Micro-F1	Macro-F1	Micro-F1	Macro-F1
DeepWalk	128	15.72 ± 1.75	12.56 ± 1.84	21.34 ± 1.20	18.59 ± 1.40	24.44 ± 0.32	20.36 ± 2.74
	512	16.08 ± 0.64	12.89 ± 1.66	19.90 ± 1.02	18.08 ± 1.11	21.51 ± 5.75	20.36 ± 5.05
node2vec	128	15.65 ± 1.46	12.07 ± 1.23	20.97 ± 1.26	17.86 ± 0.85	23.99 ± 5.84	19.05 ± 2.25
	512	15.03 ± 3.18	12.19 ± 2.34	21.04 ± 1.90	18.11 ± 2.13	22.02 ± 1.14	18.18 ± 3.47
VERSE	128	14.41 ± 1.40	11.56 ± 1.37	19.63 ± 1.08	16.95 ± 1.61	22.01 ± 2.66	18.71 ± 0.61
	512	12.59 ± 2.54	9.54 ± 2.22	13.62 ± 0.88	11.67 ± 0.85	16.00 ± 0.26	13.66 ± 0.53
FastRP	128	11.73 ± 2.37	7.24 ± 1.49	16.76 ± 0.70	11.03 ± 1.05	19.45 ± 3.10	11.70 ± 2.98
	512	15.74 ± 2.19	11.11 ± 1.20	21.19 ± 2.25	15.72 ± 1.37	21.52 ± 5.31	16.63 ± 1.87
SnapEmbed	128	15.88 ± 1.36	11.67 ± 1.09	20.51 ± 0.70	16.89 ± 0.93	21.82 ± 2.47	17.49 ± 2.36
	512	17.67 ± 1.22	13.04 ± 1.06	23.50 ± 0.97	19.84 ± 1.34	25.36 ± 2.32	21.21 ± 2.92
	2048	18.77 ± 1.22	13.76 ± 1.41	24.30 ± 0.67	20.44 ± 0.85	25.85 ± 2.91	22.03 ± 3.84

Table 6.4: Classification micro and macro F1-scores for PPI.

		<i>Labeled Nodes</i>					
<i>Method</i>	<i>d</i>	10.00%		50.00%		90.00%	
		Micro-F1	Macro-F1	Micro-F1	Macro-F1	Micro-F1	Macro-F1
DeepWalk	128	36.05 ± 0.85	20.91 ± 0.79	41.07 ± 1.05	26.85 ± 0.96	42.69 ± 1.49	28.87 ± 4.61
	512	32.48 ± 0.35	18.69 ± 1.17	37.88 ± 0.61	25.38 ± 0.85	40.14 ± 4.03	26.11 ± 6.42
node2vec	128	33.63 ± 0.96	15.28 ± 0.99	37.18 ± 0.82	20.02 ± 0.44	38.34 ± 3.62	21.26 ± 1.37
	512	33.67 ± 0.93	16.24 ± 1.11	37.42 ± 1.40	21.43 ± 0.73	38.98 ± 4.70	21.94 ± 1.49
VERSE	128	32.57 ± 0.96	18.67 ± 1.46	38.66 ± 0.88	25.0 ± 1.37	39.47 ± 1.34	26.64 ± 1.08
	512	24.64 ± 0.85	12.33 ± 1.58	29.27 ± 0.41	18.48 ± 0.88	33.18 ± 2.51	21.11 ± 2.60
FastRP	128	28.68 ± 0.35	12.74 ± 1.23	31.22 ± 1.34	14.78 ± 0.53	31.61 ± 1.90	15.34 ± 3.27
	512	33.54 ± 0.96	17.83 ± 1.90	36.94 ± 1.08	21.49 ± 0.38	37.62 ± 2.66	22.26 ± 2.98
SnapEmbed	128	27.99 ± 1.20	13.72 ± 1.49	32.40 ± 1.23	18.77 ± 1.40	33.40 ± 2.95	19.94 ± 3.30
	512	33.36 ± 1.11	17.37 ± 1.61	37.76 ± 1.37	23.79 ± 1.61	39.33 ± 3.45	26.14 ± 3.07
	2048	36.05 ± 1.66	19.01 ± 1.93	41.42 ± 1.49	27.16 ± 1.96	42.46 ± 4.35	29.00 ± 3.94

Table 6.5: Classification micro and macro F1-scores for BlogCat.

<i>Method</i>	<i>d</i>	<i>Labeled Nodes</i>					
		1.00%		5.00%		9.00%	
		Micro-F1	Macro-F1	Micro-F1	Macro-F1	Micro-F1	Macro-F1
DeepWalk	128	36.51 ± 0.85	27.54 ± 1.26	41.52 ± 0.03	29.85 ± 1.31	43.21 ± 0.61	30.31 ± 0.50
	512	37.44 ± 0.67	26.57 ± 0.76	39.41 ± 1.11	29.92 ± 0.79	40.95 ± 0.82	31.48 ± 0.91
node2vec	128	37.55 ± 0.99	26.38 ± 0.88	42.92 ± 0.55	31.12 ± 0.41	43.94 ± 0.61	32.03 ± 0.20
	512	38.35 ± 1.75	27.71 ± 1.17	42.53 ± 0.26	31.05 ± 0.50	43.99 ± 0.32	32.14 ± 0.38
VERSE	128	38.52 ± 0.47	28.17 ± 1.20	41.68 ± 0.96	31.14 ± 0.26	43.47 ± 0.26	32.22 ± 0.53
	512	38.22 ± 1.34	27.42 ± 0.91	38.03 ± 0.58	29.50 ± 0.88	38.88 ± 0.61	31.04 ± 0.82
FastRP	128	15.97 ± 0.55	4.18 ± 0.29	16.74 ± 0.64	4.31 ± 0.47	16.62 ± 0.35	4.17 ± 0.29
	512	18.88 ± 1.28	6.63 ± 0.47	26.82 ± 1.23	9.17 ± 0.26	27.91 ± 0.99	8.79 ± 0.38
SnapEmbed	128	38.19 ± 1.07	25.29 ± 1.14	41.23 ± 0.49	27.92 ± 0.63	42.48 ± 0.42	28.44 ± 0.72
	512	39.95 ± 0.67	27.64 ± 1.22	43.01 ± 0.51	30.61 ± 0.51	44.05 ± 0.35	31.50 ± 0.63
	2048	40.49 ± 1.06	28.86 ± 0.81	43.79 ± 0.46	31.69 ± 0.55	44.85 ± 0.46	32.76 ± 0.41

Table 6.6: Classification micro and macro F1-scores for CoCit.

		<i>Labeled Nodes</i>					
		1.00%		5.00%		9.00%	
<i>Method</i>	<i>d</i>	Micro-F1	Macro-F1	Micro-F1	Macro-F1	Micro-F1	Macro-F1
DeepWalk	128	32.55 ± 0.91	13.81 ± 1.72	37.44 ± 0.44	22.58 ± 0.53	38.78 ± 0.23	24.75 ± 0.58
	512	31.22 ± 0.38	13.42 ± 1.23	35.67 ± 0.38	22.72 ± 1.52	37.25 ± 0.09	25.74 ± 0.58
node2vec	128	29.27 ± 0.96	6.40 ± 0.50	34.12 ± 0.47	12.82 ± 0.88	35.15 ± 0.03	14.89 ± 0.47
	512	29.80 ± 0.67	7.14 ± 0.20	34.40 ± 0.26	13.50 ± 0.20	35.39 ± 0.06	15.58 ± 0.58
VERSE	128	28.04 ± 1.84	10.52 ± 2.37	33.52 ± 0.12	19.12 ± 0.41	35.38 ± 0.41	22.31 ± 0.93
	512	25.22 ± 0.20	7.20 ± 1.28	28.25 ± 0.29	14.17 ± 1.02	29.65 ± 0.32	17.09 ± 0.29
FastRP	128	28.20 ± 0.53	9.39 ± 1.61	30.43 ± 0.15	13.82 ± 0.61	30.65 ± 0.29	14.51 ± 0.38
	512	29.85 ± 0.26	12.28 ± 2.72	33.64 ± 0.58	18.94 ± 1.28	34.88 ± 0.58	21.44 ± 1.23
SnapEmbed	128	27.41 ± 0.90	9.14 ± 0.56	31.84 ± 0.25	14.90 ± 0.55	33.14 ± 0.33	17.27 ± 0.65
	512	30.43 ± 0.79	10.78 ± 1.20	34.00 ± 0.25	18.36 ± 0.51	35.37 ± 0.25	21.26 ± 0.48
	2048	31.89 ± 0.62	11.15 ± 1.02	35.94 ± 0.23	19.38 ± 0.85	37.21 ± 0.18	23.02 ± 0.56

Table 6.7: Classification
micro and macro
F1-scores for Flickr.

		<i>Labeled Nodes</i>					
		1.00%		5.00%		9.00%	
<i>Method</i>	<i>d</i>	Micro-F1	Macro-F1	Micro-F1	Macro-F1	Micro-F1	Macro-F1
DeepWalk	128	37.53 ± 1.40	29.04 ± 3.77	41.64 ± 0.15	34.45 ± 0.70	42.97 ± 0.29	35.62 ± 0.93
	512	38.69 ± 1.17	31.11 ± 1.08	40.26 ± 0.38	35.09 ± 0.26	40.74 ± 0.06	36.14 ± 0.23
VERSE	128	37.13 ± 0.41	28.54 ± 2.39	39.74 ± 0.32	33.87 ± 0.67	41.70 ± 0.38	35.04 ± 0.41
	512	36.74 ± 1.05	27.16 ± 0.15	37.47 ± 1.37	32.40 ± 0.91	37.64 ± 0.67	33.00 ± 0.35
node2vec	128	34.64 ± 2.63	25.35 ± 3.83	40.62 ± 1.02	33.26 ± 0.20	42.65 ± 0.70	35.73 ± 0.32
	512	36.02 ± 2.01	25.03 ± 2.89	39.64 ± 0.44	33.78 ± 0.38	40.47 ± 0.85	35.01 ± 1.08
FastRP	128	23.61 ± 1.61	6.24 ± 0.61	24.16 ± 0.96	6.64 ± 1.64	24.50 ± 0.29	7.09 ± 0.35
	512	22.83 ± 0.41	7.21 ± 0.20	23.43 ± 0.55	8.77 ± 0.82	23.76 ± 0.64	9.56 ± 0.91
SnapEmbed	128	37.89 ± 1.02	26.27 ± 1.36	40.90 ± 0.53	31.57 ± 0.86	41.78 ± 0.37	32.73 ± 0.51
	512	40.04 ± 0.97	27.52 ± 1.60	43.31 ± 0.41	33.98 ± 0.81	44.00 ± 0.42	35.56 ± 0.69
	2048	40.91 ± 0.86	28.34 ± 1.43	44.82 ± 0.49	35.16 ± 1.02	45.67 ± 0.32	36.90 ± 0.69

Table 6.8: Classification micro and macro F1-scores for YouTube.

<i>Method</i>	<i>d</i>	<i>Aggregation Function</i>				
		hadamard	cosine	L1	L2	average
DeepWalk	128	75.59 ± 0.88	83.5 ± 0.12	86.99 ± 0.09	87.21 ± 0.73	73.64 ± 1.72
	512	78.42 ± 0.53	82.05 ± 1.20	87.85 ± 0.29	88.43 ± 1.08	79.56 ± 0.70
node2vec	128	80.18 ± 0.67	54.59 ± 0.88	70.14 ± 1.31	70.32 ± 0.58	79.07 ± 0.53
	512	86.09 ± 0.85	42.99 ± 1.66	72.41 ± 1.84	72.70 ± 1.43	84.00 ± 0.38
VERSE	128	93.16 ± 0.44	90.85 ± 0.20	79.24 ± 1.49	80.27 ± 0.41	86.50 ± 0.47
	512	92.75 ± 0.73	90.33 ± 0.20	72.58 ± 1.17	73.82 ± 1.49	86.69 ± 1.02
FastRP	128	60.23 ± 1.78	65.08 ± 0.93	78.51 ± 0.64	77.66 ± 0.23	57.69 ± 1.90
	512	61.16 ± 1.75	70.12 ± 0.38	82.19 ± 2.22	78.51 ± 1.99	63.87 ± 1.49
SnapEmbed	128	89.41 ± 0.67	89.15 ± 0.63	66.19 ± 1.92	66.78 ± 1.90	83.22 ± 0.86
	512	90.44 ± 0.48	90.60 ± 0.55	76.50 ± 1.44	75.76 ± 1.41	85.64 ± 0.67
	2048	89.45 ± 0.62	90.84 ± 0.44	88.42 ± 0.48	84.83 ± 0.67	87.67 ± 1.07

Table 6.9: Temporal link-prediction ROC-AUC scores for CoAuthor. For each method, we highlight the aggregation function that consistently performs good on all datasets.

GRAPH Neural Networks (GNNs) have achieved state-of-the-art results on many graph analysis tasks such as node classification and link prediction. However, important *unsupervised* problems on graphs, such as graph clustering, have proved more resistant to advances in GNNs. Graph clustering has the same overall goal as node pooling in GNNs—does this mean that GNN pooling methods cluster graphs well and extract expressive node representations?

Surprisingly, the answer is no—current GNN pooling methods often fail to recover the cluster structure in cases where simple baselines, such as k-means applied on learned representations, work well. We investigate further by carefully designing experiments to study different signal-to-noise scenarios in the graph structure and attribute data. To address these methods’ poor performance in clustering, we introduce Deep Modularity Networks (DMON), an unsupervised pooling method inspired by the modularity measure of clustering quality, and show how it tackles recovery of the challenging clustering structure of real-world graphs. Similarly, on real-world data, we show that DMON produces high-quality clusters that correlate strongly with ground truth labels, achieving state-of-the-art results with over 40% improvement over other pooling methods across different metrics.

Most existing work on GNNs to leverage higher-order structure does not directly address node partitioning or the estimation of clusters within the computational graph. Furthermore, most works explore these mechanisms only within a semi-supervised or supervised framework, ignoring the fact that *unsupervised* graph clustering is often an extremely useful end-goal in itself—whether for data exploration [PA18], visualization [CDART12, CZQ⁺08], genomic feature discovery [CAT16], anomaly detection [PA16], or for many other use-cases discussed, e.g., in [FH16]. Additionally, many of the existing unsupervised structure-aware methods have undesirable properties, such as relying on a multi-step optimization process that does not allow to optimize the objective via gradient descent end-to-end [PAISM14].

In this work, we take an *ab initio* approach to the clustering problem in the GNN domain, bridging the gap between traditional graph clustering objectives and deep neural networks. We start by drawing a connection between graph pooling, which was typically studied in the literature as a regularizer for supervised GNN architectures, and fully unsupervised clustering.

Specifically, we contribute:

- An unsupervised clustering module for GNNs, `DMON`, that allows optimization of cluster assignments in an end-to-end differentiable way with strong empirical performance.
- An empirical study of the performance of various models on synthetic attributed graphs, illustrating the problems with existing work and how `DMON` allows for improved model performance in those regimes.
- Thorough experimental evaluation on real-world data, showing that many pooling methods poorly reflect hierarchical structures and are not able to make use of either graph structure and node attributes nor leverage joint information.

7.1 PRELIMINARIES

We introduce the necessary background for `DMON`, starting with the problem formulation, reviewing common graph clustering objectives and how they can be made differentiable.

We are interested in measuring the quality of graph partitioning function $\mathcal{F} : V \mapsto \{1, \dots, k\}$ that splits the set of nodes V into k partitions $V_i = \{v_j, \mathcal{F}(v_j) = i\}$. In contrast to standard graph clustering, we are also provided with node attributes $\mathbf{X} \in \mathbb{R}^{n \times s}$.

7.1.1 GRAPH CLUSTERING QUALITY FUNCTIONS

As classical clustering objectives are discrete and therefore unsuitable for gradient-based optimization, `DMON` and the few prior works depend on spectral approximations. To contextualize and motivate our contributions, we review two families of clustering quality functions amenable to spectral optimization, and review some of their shortcomings.

Cut-based metrics. In his seminal work [Fie73], Fiedler suggested that the second eigenvector of a graph Laplacian produces a graph *cut* minimal in terms of the weight of the edges. This plain notion of cut degenerates on real-world graphs, as it does not require partitions to be balanced in terms of size. It is possible to get normalized partitions with the use of ratio cut [WC89], which normalizes the cut by the product of the number of nodes in two partitions, or normalized cut [SM00], which uses the total edge volume of the partition as normalization.

In real networks, however, there is evidence *against* the existence of good cuts [LLDM08] in ground-truth communities. This can be explained by the fact that a single node implicitly participates in many different clusters [ELPL17], e.g., a person in a social network is simultaneously connected with family and work friends, forcing the algorithm to merge these communities together.

Recently, MinCutPool [BGA20] adapted the notion of the normalized cut to use as a regularizer for pooling. While MinCutPool’s objective should, theoretically, be suitable for clustering nodes in graphs, we show that MinCutPool does not optimize its own objective function, using synthetic and real-world experiments.

Modularity. The modularity [Newo6b] objective approaches the same problem from a statistical perspective, incorporating a *null model* to quantify the deviation of the clustering from what would be observed in expectation under a random graph. In a fully random graph with given degrees, nodes u and v with degrees d_u and d_v are connected with probability $d_u d_v / 2m$. Modularity measures the divergence between the intra-cluster edges from the expected:

$$\mathcal{Q} = \frac{1}{2m} \sum_{ij} \left[\mathbf{A}_{ij} - \frac{d_i d_j}{2m} \right] \delta(c_i, c_j), \quad (7.1)$$

where $\delta(c_i, c_j) = 1$ if i and j are in the same cluster and 0 otherwise. Note $\mathcal{Q} \in (-1/2, 1]$ (it is 0 when there is no correlation of clusters with edge density), but it is not necessarily maximized at 1, and is only comparable across graphs with the same degree distribution. While problems with the modularity metric have been identified [GDMC10], it remains one of the most commonly-used and eminently useful graph clustering metrics in the scientific literature [FH16].

7.1.2 SPECTRAL MODULARITY MAXIMIZATION

Maximizing the modularity is proven to be NP-hard [BDG⁺06], however, a spectral relaxation of the problem can be solved efficiently [Newo6a]. Let $\mathbf{C} \in \{0, 1\}^{n \times k}$ be the cluster assignment matrix and \mathbf{d} be the degree vector. Then, using the *modularity matrix* $\mathbf{B} = \mathbf{A} - \frac{\mathbf{d}\mathbf{d}^\top}{2m}$, the modularity \mathcal{Q} is defined as:

$$\mathcal{Q} = \frac{1}{2m} \text{tr}(\mathbf{C}^\top \mathbf{B} \mathbf{C}) \quad (7.2)$$

Relaxing $\mathbf{C} \in \mathbb{R}^{n \times k}$, the optimal \mathbf{C} maximizing \mathcal{Q} is the top- k eigenvectors of the modularity matrix \mathbf{B} . While \mathbf{B} is dense, iterative eigenvalue solvers can take advantage of the fact that \mathbf{B} is a sum of a sparse \mathbf{A} and rank-one matrix $-\frac{\mathbf{d}\mathbf{d}^\top}{2m}$, meaning that the product $\mathbf{B}\mathbf{x}$ can be computed efficiently:

$$\mathbf{B}\mathbf{x} = \mathbf{A}\mathbf{x} - \frac{\mathbf{d}^\top \mathbf{x} \mathbf{d}}{2m}$$

and optimized efficiently with iterative methods such as power iteration or the Lanczos algorithm. One can then obtain clusters by means of spectral bisection [Newo6a] with iterative refinement akin to Kernighan-Lin algorithm [KL70]. However, these formulations operate entirely on the graph structure, and it is non-trivial to adapt them to work with attributed graphs.

7.1.3 GRAPH NEURAL NETWORKS

Graph Neural Networks are a flexible class of models that perform nonlinear feature aggregation with respect to graph structure. For the purposes of this work, we consider transductive GNNs that output a single embedding per node. Graph convolutional networks (GCNs) [KW17] are simple yet effective [SMBG18] message-passing networks that fit our criteria. Let $\mathbf{X}^0 \in \mathbb{R}^{n \times s}$ be the initial node features and $\tilde{\mathbf{A}} = \mathbf{D}^{-\frac{1}{2}} \mathbf{A} \mathbf{D}^{-\frac{1}{2}}$ be the normalized adjacency matrix, the output of t -th layer \mathbf{X}^{t+1} is

$$\mathbf{X}^{t+1} = \text{SeLU}(\tilde{\mathbf{A}}\mathbf{X}^t\mathbf{W} + \mathbf{X}\mathbf{W}_{\text{skip}}) \quad (7.3)$$

We make two changes to the classic GCN architecture: first, we remove the self-loop creation and instead use an $\mathbf{W}_{\text{skip}} \in \mathbb{R}^{s \times s}$ trainable skip connection, and, second, we replace RELU nonlinearity with its self-normalized counterpart [KUMH17] for better convergence.

7.2 METHOD

In this section, we present `DMON`, our method for attributed graph clustering with graph neural networks. Inspired by the modularity quality function and its spectral optimization, we propose a fully differentiable unsupervised clustering objective which optimizes soft cluster assignments using a null model to control for inhomogeneities in the graph. We then discuss the challenge of regularizing cluster assignments, and present collapse regularization that is effective at preventing trivial solutions without compromising optimization of the objective.

7.2.1 `DMON`: DEEP MODULARITY NETWORKS

The challenge of `GNN` clustering with the modularity objective boils down to defining an architecture and the optimization procedure of the cluster assignment matrix \mathbf{C} . In `DMON`, we propose to obtain \mathbf{C} via the output of a softmax function, which allows the (soft) cluster assignment to be differentiable. The input to the cluster assignment can be any differentiable message passing function, but here we specifically consider the case where a graph convolutional network is used to obtain soft clusters for each node as follows:

$$\mathbf{C} = \text{softmax}(\text{GCN}(\tilde{\mathbf{A}}, \mathbf{X})), \quad (7.4)$$

where `GCN` is a (possibly) multi-layer convolutional network operating on an normalized adjacency matrix $\tilde{\mathbf{A}} = \mathbf{D}^{-\frac{1}{2}}(\mathbf{A})\mathbf{D}^{-\frac{1}{2}}$.

We then propose to optimize this assignment with the following objective, which combines insights from spectral modularity maximization (7.2) with a novel regularization to prevent trivial solutions to the optimization problem:

$$\mathcal{L}_{\text{DMON}} = \underbrace{-\frac{1}{2m} \text{tr}(\mathbf{C}^\top \mathbf{B} \mathbf{C})}_{\text{modularity}} + \underbrace{\frac{\sqrt{k}}{n} \left\| \sum_i \mathbf{c}_i^\top \right\|_F}_{\text{collapse regularization}} - 1, \quad (7.5)$$

where $\|\cdot\|_F$ is the Frobenius norm. We decompose the computation of $\text{tr}(\mathbf{C}^\top \mathbf{B} \mathbf{C})$ as a sum of sparse matrix-matrix multiplication and rank-one degree normalization $\text{tr}(\mathbf{C}^\top \mathbf{A} \mathbf{C} - \mathbf{C}^\top \mathbf{d}^\top \mathbf{d} \mathbf{C})$. This allows us to efficiently optimize `DMON` parameters in the sparse regime.

Figure 7.1: Optimization progress of MinCut and DMoN on Cora dataset. MinCut optimizes the regularizer, while DMoN minimizes its main objective.

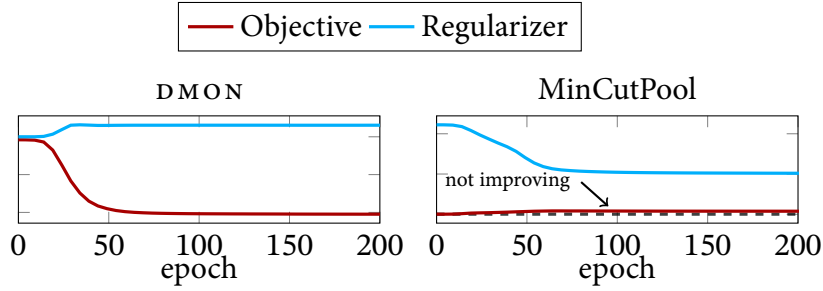


Table 7.1: Comparison of MinCutPool with using only its orthogonality regularization in terms of graph conductance \mathcal{C} , modularity \mathcal{Q} , and NMI with ground-truth labels.

<i>method</i>	Cora			Citeseer		
	<i>graph</i>	<i>labels</i>		<i>graph</i>	<i>labels</i>	
	$\mathcal{C} \downarrow$	$\mathcal{Q} \uparrow$	NMI \uparrow	$\mathcal{C} \downarrow$	$\mathcal{Q} \uparrow$	NMI \uparrow
MinCut	23.3	70.3	35.8	14.1	78.9	25.9
Ortho	28.0	65.6	38.4	18.4	74.5	26.1
<i>method</i>	Coauthor _{CS}			Coauthor _{PHY}		
	<i>graph</i>	<i>labels</i>		<i>graph</i>	<i>labels</i>	
	$\mathcal{C} \downarrow$	$\mathcal{Q} \uparrow$	NMI \uparrow	$\mathcal{C} \downarrow$	$\mathcal{Q} \uparrow$	NMI \uparrow
MinCut	22.7	70.5	64.6	27.8	64.3	48.3
Ortho	27.8	65.7	64.6	33.0	59.5	44.7

7.2.2 COLLAPSE REGULARIZATION

Without additional constraints on the assignment matrix \mathbf{C} , spectral clustering for both min-cut and modularity objectives has spurious local minima: assigning all nodes to the same cluster produces a trivial locally optimal solution that traps gradient-based optimization methods. MinCutPool addresses this problem by adapting spectral orthogonality constraint in the form of soft-orthogonality regularization $\|\mathbf{C}^T \mathbf{C} - \mathbf{I}\|_F$ [BCW18]. We notice that this term is overly restrictive when combined with softmax class assignment – intuitively, when the value range of \mathbf{C} is restricted to $\mathbb{R} \cap [0, 1]$, optimization of the soft-orthogonality regularizer dominates the loss.

We illustrate this problem in Figure 7.1, which depicts the progress of optimization for both methods in terms of their main objective and regularizer term over the course of 200 epochs on Cora dataset. The soft orthogonality regularization term dominates the optimization for MinCutPool, such that the cut objective becomes *worse than random* over the course of training.

<i>dataset</i>	Nodes	Edges	Features	Classes
Cora	2708	5278	1433	7
Citeseer	3327	4614	3703	6
Pubmed	19717	44325	500	3
Amazon PC	13752	143604	767	10
Amazon Photo	7650	71831	745	8
Coauthor CS	18333	81894	6805	15
Coauthor PHY	34493	247962	8415	5

Table 7.2: Dataset statistics.

We experimentally verify that the MinCut clustering objective is a proverbial fifth wheel for MinCutPooling optimization. Table 7.1 presents the difference in performance of the optimization of full MinCutPool objective and simple feature orthogonalization (Ortho in the table). On 3 out of 4 datasets, orthogonality achieves better label correlation than joint optimization. Regularization dominates the method’s performance.

With `DMON`, we fix this problem by proposing a relaxed notion of *collapse regularization* that prevents the trivial partition while not dominating the optimization of the main objective. The regularizer is a Frobenius norm of the (soft) cluster membership counts, normalized to the range $[0, 1]$. It gets the value of 0 when cluster sizes are perfectly balanced, and 1 in the case all clusters collapse to one. We also improve the training by applying dropout [SHK⁺14] to GNN representations before the softmax, preventing the gradient descent from getting stuck in the local optima of the highly non-convex objective function.

7.3 EXPERIMENTS

In this section, we describe empirical experiments—involving both synthetic and real-world data—on `DMON` and baseline methods, to test robustness and performance against both graph clustering and label alignment metrics. We use open-source graph simulation tools, publicly-available datasets, and we release the implementation of `DMON` at this URL¹.

Datasets. We use 7 real-world datasets for assessing model quality. Cora, Citeseer, and Pubmed [SNB⁺08] are citation networks; nodes represent papers

1. https://github.com/google-research/google-research/tree/master/graph_embedding/dmon

connected by citation edges; features are bag-of-word abstracts, and labels represent paper topics. Amazon PC and Amazon Photo [SMBG18] are subsets of the Amazon co-purchase graph for the computers and photo sections of the website, where nodes represent goods with edges between ones frequently purchased together; node features are bag-of-word reviews, and class labels are product category. Coauthor CS and Coauthor PHY [SMBG18] are co-authorship networks based on the Microsoft Academic Graph for the computer science and physics fields respectively; nodes are authors, which are connected by an edge if they co-authored a paper together; node features are a collection of paper keywords for author’s papers; class labels indicate most common fields of study.

Baselines. As we study how to leverage the information from both the graph and attributes, we employ two baselines that employ either strictly graph or attribute information. We give a brief description of the baselines used below:

- k-means(features) is our baseline that only considers the feature data. We use the local Lloyd algorithm [Llo82] with the k-means++ seeding strategy [AV07].
- SBM [Pei14b] is a baseline that only relies on the graph structure. We estimate a constrained stochastic block model with given k , the maximum number of clusters.
- k-means(DeepWalk, features) [PARS14] represents a naïve strategy of concatenating node attributes to learned node embeddings of the graph without attributes.
- k-means(DGI) [VFH⁺19] demonstrates the need for joint learning of clusters and representations. We learn unsupervised node representations of the attributed graph with DGI and run k-means on the resulting representations.
- DiffPool(graph, features) [YYM⁺18] is an early graph pooling method.
- MinCutPool(graph, features) [BGA20] is a deep pooling method that orthogonalizes the cluster representations (cf. Section 7.4 for discussion).
- Ortho(graph, features) [BGA20] is a variant of MinCutPool that only does the cluster orthogonalization without *any* graph-related objective.

Metrics. We measure both the graph-based metrics of clustering and label correlation to study the clustering performance of attributed graphs both in

terms of graph and attribute structure. For experiments on real-world data, we measure both (1) standard graph-based clustering metrics, and (2) correlation to ground-truth node labels. Our graph-based metrics are average cluster conductance (as per definition from [YL15]) and graph modularity [Newo6b]. Our label-based metrics are normalized mutual information (NMI) between the cluster assignments and labels and pairwise F1 score between all node pairs and their associated cluster pairs. For experiments on synthetic data, we report only the NMI against the (simulated) cluster labels. Where possible, we normalize all metrics by multiplying them by 100 for readability and ease of comparison.

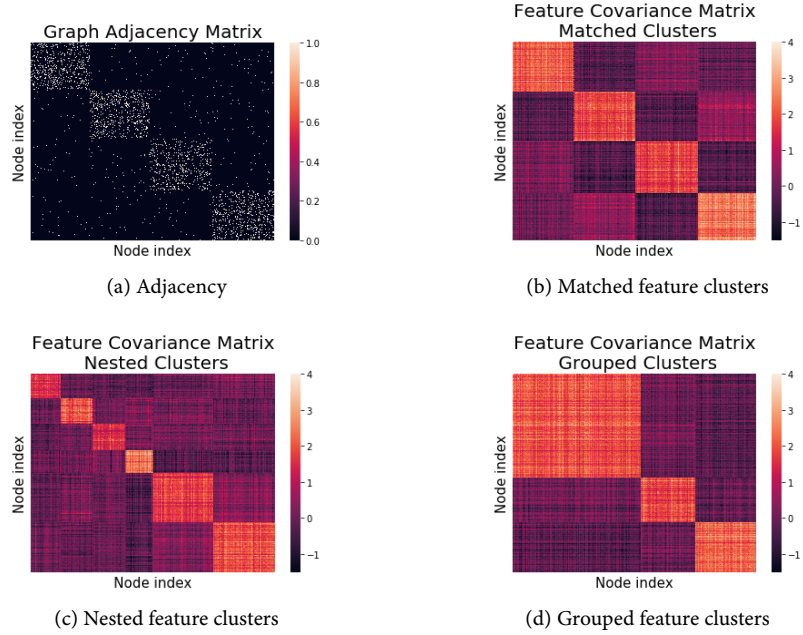
Parameter settings. We run both synthetic and real-world experiments for 10 times and average results across runs. All models were implemented in Tensorflow 2 and trained on CPUs. We fix the architecture for all GNNs (including DMON) have one hidden layer—with 512 neurons for real-world data experiments, and 64 neurons for experiments with smaller synthetic graphs. We set the maximum number of clusters to 16 for all datasets and methods.

7.3.1 SIMULATION EXPERIMENTS ON STOCHASTIC BLOCK MODEL WITH FEATURES

To explore the robustness and sensitivity of DMON and baselines to variance in the graph and node features, we conduct a study on synthetic graphs using an *attributed, degree-corrected* stochastic block model (ADC-SBM). The SBM [SN97] plants a partition of clusters (“blocks”) in a graph, and generates edges via a distribution conditional on that partition. This model has been used extensively to benchmark graph clustering methods [FH16], and has recently been used for experiments on state-of-the-art supervised GNNs [DJL⁺20]. In our version of the model, node features are also generated, using a multivariate mixture model, with the mixture memberships having some association (or de-association) with the cluster memberships. We proceed to describe the graph generation and feature generation components of our model.

Graph generation. We fix a number of nodes n and a number of clusters k , and choose node cluster memberships uniformly-at-random. Define the matrix $\mathbf{D}_{k \times k}$ where \mathbf{D}_{ij} is the expected number of edges between nodes in clusters i

Figure 7.2: Illustration of synthetic data. (a) 4-cluster graph adjacency matrix. (b) Covariance matrix of “matched” features: features that are clustered according to the graph clusters. (c) Covariance matrix of “nested” features: features that are clustered by incomplete nesting of the graph clusters. (d) Covariance matrix of “grouped” features: features that are clustered by incomplete grouping of the graph clusters.



and j . We determine \mathbf{D} by fixing (1) the expected *average* degree of the nodes $d \in \{1, n\}$, and (2) the expected *average* sub-degree $d_{out} \leq d$ of a node to any cluster other than its own. Note that the difference $d_{in} - d_{out}$, where $d_{in} := d - d_{out}$, controls the spectral detectability of the clusters [NN12]. Finally, we generate a power-law vector θ , where θ_i is proportional to i 's expected degree. We use the generated memberships and the generated parameters \mathbf{D} and θ as inputs to the degree-corrected SBM from the `graph-tool` [Pei14b] package.

Feature generation. We generate feature memberships from k_f cluster labels. For graph clustering GNNs that operate both on edges and node features, it is important to examine performance on data where feature clusters diverge from the graph clusters: thus potentially $k_f \neq k$. We examine cases where feature memberships *match*, *group*, or *nest* the graph memberships, as illustrated in Figure 7.2. With feature memberships in-hand, we generate k zero-mean feature cluster centers from a s -multivariate normal with covariance matrix $\sigma_c^2 \cdot \mathbf{I}_{s \times s}$. Then, for feature cluster $i \leq k_f$, we generate its features from a s -multivariate normal with covariance matrix $\sigma^2 \cdot \mathbf{I}_{s \times s}$. The ratio σ_c^2 / σ^2 controls the expected value of the between/within sum-of-squares of the clusters.

Scenario	Parameter	Description
1	$d_{out} \in [2.0, 5.0]$	Increase graph cluster mixing signal. Higher = weaker clusters.
2	$\sigma_c \in [10^{-2}, 10^1]$	Increase feature cluster center variance. Higher = stronger clusters.
3	$\sigma_c \in [10^{-2}, 10^1]$	Increase feature cluster center variance, with nested feature clusters.
4	$\sigma_c \in [10^{-2}, 10^1]$	Increase feature cluster center variance, with grouped feature clusters.
5	$d \in [2^2, 2^7]$	Increase average node degree in a graph. Higher = clearer graph signal.
6	$d_{max} \in [2^2, 2^{10}]$	Increase the degree power law upper-bound. Higher = more extreme power law.

Table 7.3: Synthetic ADC-SBM benchmark scenarios.

The above paragraphs describe a single generation of our synthetic benchmark model, the ADC-SBM. To study model robustness, we define “default” ADC-SBM parameters, and explore model parameters in a range around the defaults. We configure our default model as follows: we generate graphs with $n = 1,000$ nodes grouped in $k = 4$ clusters, and $s = 32$ -dimensional features grouped in $k_f = 4$ matching feature clusters with $\sigma = 1$ intra-cluster center variance and $\sigma_c = 3$ cluster center variance. We mimic real-world graphs’ degree distribution with $d = 20$ average degree and $d_{out} = 2$ average inter-cluster degree with power law parameters $d_{min} = 2$, $d_{max} = 4$, $\alpha = 2$. In total, we consider 6 different scenarios, as described in Table 7.3.

Results. We split the presentation of results for pooling methods (Figure 7.3) and other baselines (Figure 7.4) for ease of understanding. Overall, DMON demonstrates overwhelming superiority over all baselines, with the most significant improvements over other pooling methods. MinCut pooling suffers from the presence of even the weak noise in the graph (Scenario 1) or in the features (Scenario 2). Moreover, it is susceptible to both nested and grouped features (Scenarios 3 and 4), while DMON and DiffPool are less sensitive to these variations. We notice how both DiffPool and MinCutPool are dependent on the sparsity level and degree homogeneity of the graph—DiffPool performs better denser graphs while MinCutPool shows the opposite picture.

Figure 7.3: Synthetic results on the ADC-SBM model with 6 different scenarios described in Table 7.3. We observe that DMoN significantly outperforms other neural graph pooling method baselines.

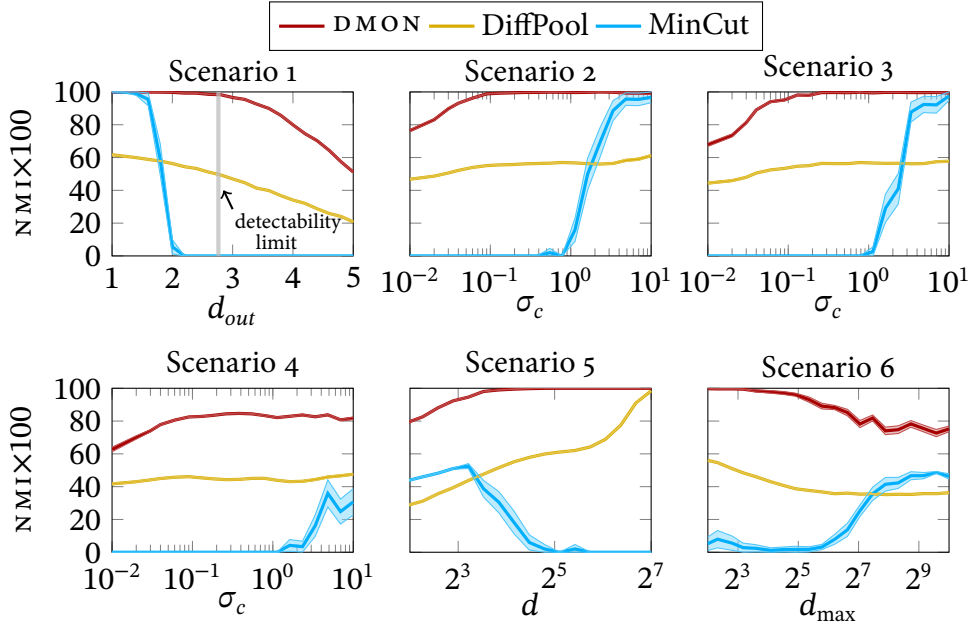
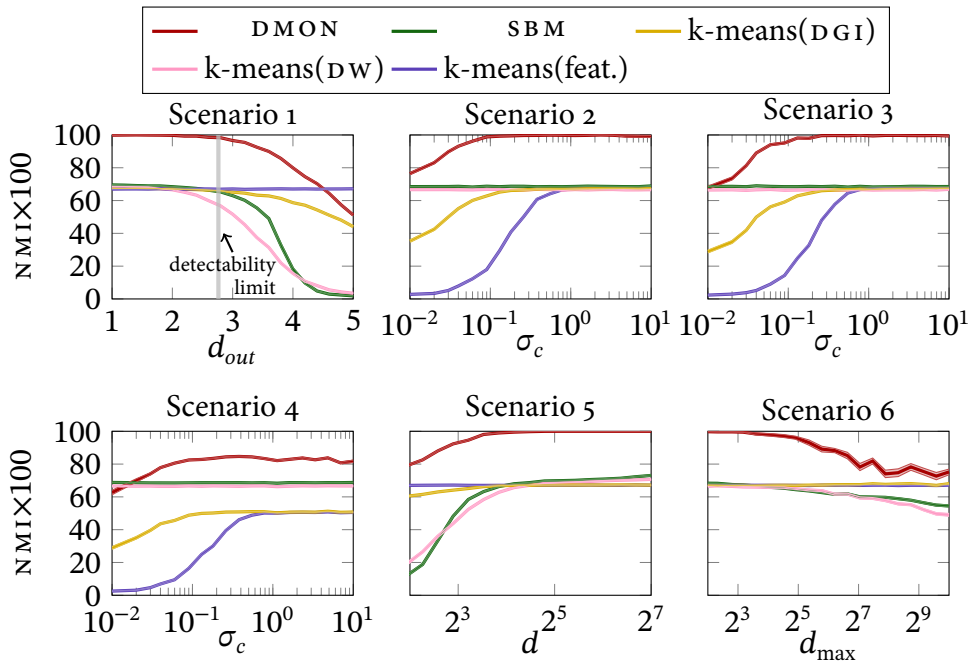


Figure 7.4: Synthetic results on the ADC-SBM model with 6 different scenarios described in Table 7.3. We observe that DMoN leverages information from both graph structure and node attributes.



<i>method</i>	Cora				Citeseer				Pubmed			
	<i>graph</i>		<i>labels</i>		<i>graph</i>		<i>labels</i>		<i>graph</i>		<i>labels</i>	
	$\mathcal{C} \downarrow$	$\mathcal{Q} \uparrow$	NMI \uparrow	F1 \uparrow	$\mathcal{C} \downarrow$	$\mathcal{Q} \uparrow$	NMI \uparrow	F1 \uparrow	$\mathcal{C} \downarrow$	$\mathcal{Q} \uparrow$	NMI \uparrow	F1 \uparrow
k-m(feats)	61.7	19.8	18.5	27.0	60.5	30.3	24.5	29.2	55.8	33.4	19.4	24.4
SBM	15.4	77.3	36.2	30.2	14.2	78.1	15.3	19.1	39.0	53.5	16.4	16.7
k-m(DW)	62.1	30.7	24.3	24.8	68.1	24.3	27.6	24.8	16.6	75.3	22.9	17.2
k-m(DGI)	28.0	64.0	52.7	40.1	17.5	73.7	40.4	39.4	82.9	9.6	22.0	26.4
DiffPool	26.1	66.3	32.9	34.4	26.0	63.4	20.0	23.5	32.9	56.8	20.2	26.3
MinCut	23.3	70.3	35.8	25.0	14.1	78.9	25.9	20.1	29.6	63.1	25.4	15.8
Ortho	28.0	65.6	38.4	26.6	18.4	74.5	26.1	20.5	57.8	32.9	20.3	13.9
DMON	12.2	76.5	48.8	48.8	5.1	79.3	33.7	43.2	17.7	65.4	29.8	33.9

To better understand the limits of the task, we study the performance of our baselines and report the results on Figure 7.4. In particular our interest lies in the performance of the SBM and pure k-means over features, as these two baselines depict the performance possible when utilizing only one aspect of the data. Scenario 1 shows that DMON can effectively leverage the feature signal to obtain outstanding clustering performance even when the graph structure is close to random, far beyond the spectral detectability threshold (pictured in gray). Scenario 2 demonstrates that even in the presence of a weak feature signal DMON outperforms stochastic SBM minimization. We also notice that while the k-means(DGI) baseline offers some improvements over using features or the graph structure alone, it never surpasses the strongest signal provider in the graph, never being better than the best one between k-means(features) and SBM. K-means applied over the extracted DeepWalk representations are also almost never stronger than the community detection using direct SBM likelihood optimization.

7.3.2 REAL-WORLD DATA

We now move on to studies on real-world networks, featuring DMON and 7 baselines on 7 different datasets. DMON achieves better clustering performance than its neural counterparts on every single dataset and metric besides losing twice to DGI+k-means on Cora and Citeseer in terms of NMI. Compared to SBM, a method that exclusively optimizes modularity, we are able to

Table 7.4: Results on four datasets from [SNB⁺08] in terms of graph conductance \mathcal{C} , modularity \mathcal{Q} , NMI with ground-truth labels, and pairwise F1 measure. We group the methods into three categories: baselines using only one aspect of data, neural representation learning, and neural graph pooling methods. We highlight best neural method performance.

method	Amazon PC				Amazon Photo				Coauthor CS				Coauthor PHY			
	graph		labels		graph		labels		graph		labels		graph		labels	
	$\mathcal{C} \downarrow$	$\mathcal{Q} \uparrow$	NMI \uparrow	F1 \uparrow	$\mathcal{C} \downarrow$	$\mathcal{Q} \uparrow$	NMI \uparrow	F1 \uparrow	$\mathcal{C} \downarrow$	$\mathcal{Q} \uparrow$	NMI \uparrow	F1 \uparrow	$\mathcal{C} \downarrow$	$\mathcal{Q} \uparrow$	NMI \uparrow	F1 \uparrow
k-m(feet)	84.5	5.4	21.1	19.2	79.6	10.5	28.8	19.5	49.1	23.1	35.7	39.4	57.0	19.4	30.6	42.9
SBM	31.0	60.8	48.4	34.6	18.6	72.7	59.3	47.4	20.3	72.7	58.0	47.7	25.9	66.9	45.4	30.4
k-m(DW)	67.6	11.8	38.2	22.7	60.6	22.9	49.4	33.8	33.1	59.4	72.7	61.2	44.7	47.0	43.5	24.3
k-m(DGI)	61.9	22.8	22.6	15.0	51.5	35.1	33.4	23.6	35.1	57.8	64.6	51.9	38.6	51.2	51.0	30.6
DiffPool	35.6	30.4	22.1	38.3	26.5	46.8	35.9	41.8	33.6	59.3	41.6	34.4	23.6	53.8	28.7	23.6
MinCut	—	—	—	—	—	—	—	—	22.7	70.5	64.6	47.8	27.8	64.3	48.3	24.9
Ortho	—	—	—	—	—	—	—	—	27.8	65.7	64.6	46.1	33.0	59.5	44.7	23.7
DMON	18.0	59.0	49.3	45.4	12.7	70.1	63.3	61.0	17.5	72.4	69.1	59.8	18.8	65.8	51.9	37.0

Table 7.5: Results on four datasets from [SMBG18] in terms of graph conductance \mathcal{C} , modularity \mathcal{Q} , NMI with ground-truth labels, and pairwise F1 measure. We group the methods into three categories: baselines using only one aspect of data, neural representation learning, and neural graph pooling methods. We highlight best neural method performance.

stay within 3% in terms of modularity, while simultaneously clustering the features. Surprisingly, on Citeseer and Pubmed we achieve better modularity than the method optimizing it directly—we attribute that to very high correlation between the graph structure and the features.

Compared to other pooling methods, DMON improves by over 40% in conductance, modularity and NMI on average. In particular, DiffPool achieves overall poor performance across metrics due to its quadratic reconstruction term that does not scale well with graph sparsity (cf. Scenario 5). MinCut pooling performs better, but only manages to match the performance of non-pooling neural representation learning methods on one dataset in terms of ground-truth label NMI. On Amazon PC and Amazon Photo both MinCut-Pool and its orthogonality-only version failed to converge, even with tuning the parameters. We attribute that to the extremely uneven structure of these graphs, as popular products are co-purchased with a lot of other items, so the effects discussed in [ELPL17, LLDMo8] are prohibiting good cuts. This corresponds to high values of d and d_{\max} in our synthetic scenarios 5 and 6. We also highlight that we beat MinCutPool in terms of conductance (average graph cut) on all datasets, even though it attempts to optimize for this metric.

Overall, DMON demonstrates excellent performance on both graph clustering and label correlation, successfully leveraging both graph and attribute information. Both synthetic and real-world experiments prove that DMON is vastly superior to its counterparts in attributed graph clustering.

7.4 SUMMARY

In this chapter, we study GNN pooling through the lens of attributed graph clustering. We introduce Deep Modularity Networks (DMON), an unsupervised objective and realize it with a GNN which can recover high quality clusters. We compare against challenging baselines that optimize structure (SBM), features (k-means), or both (DGI+k-means), in addition to a recently proposed state-of-the-art pooling method (MinCutPool).

We explore the limits of GNN clustering methods in terms of both graph and feature signals using synthetic data, where we see that DMON better leverages structure and attributes than all existing methods. In extensive experiments on real datasets we show that the clusters found by DMON are more likely to correspond to ground truth labels, and have better properties as illustrated by clustering metrics (e.g. conductance or modularity). We hope that this work will further advancements in unsupervised learning for GNNs as well as attributed graph clustering, allowing further advances in graph learning.

PART II

SIMILARITIES AND REPRESENTATIONS OF GRAPHS

REPRESENTING graphs as low-dimensional vectors in Euclidean space allows us to compare their connectivity structure efficiently. Such comparisons facilitate the analysis of graph collections, for example, graph classification in protein molecule databases or social contact networks. Traditionally, distances between graphs require computationally expensive pairwise comparisons [GXTL10, KJM20]. Moreover, comparing graphs is often focused on local sub-structures and disregards a more global, bird’s-eye view of graphs. This part of the dissertation is devoted to efficient algorithms for computing *multi-scale* representations of graphs.

Chapter 9 introduces our core proposal of this part of the dissertation: NETLSD, a graph representation method that captures graphs’ multi-scale structure. Our method is a considerable advancement over the naïve computation of graph statistics that provide no guarantees on the actual distance between graphs. Instead, NETLSD lower-bounds the Gromov–Wasserstein distance between graphs. We also derive normalization methods to make NETLSD size-invariant, comparing the structure of graphs of entirely different sizes. Efficiency-wise, we propose an approximation algorithm that allows NETLSD to compare graphs with millions of nodes and edges.

While NETLSD captures both local and global information about graphs, it does not tailor the representations to the task at hand. The problem is that many collections of graphs are small; thus, any sufficiently powerful learning algorithm would overfit on that data. Nevertheless, the analyst might know whether they are interested in local or global properties of graphs in that collection. Therefore, Chapter 10 proposes SGR, a *self-supervised*¹ neural network built on top of NETLSD that is trained to solve two tasks: one local and one global to learn tailored representations of graphs.

While NETLSD scales to graphs with millions of nodes, some practical applications necessitate processing even larger graphs. Chapter 11 proposes SLAQ, a method for approximating NETLSD and the Estrada index [Est00], significantly improving the accuracy and speed. We observe several orders

1. Self-supervised training is solving supervised tasks with artificial labels created in an unsupervised way.

of magnitude improvement in approximation accuracy of the method while scaling to graphs with billions of nodes. We also derive theoretical bounds on our methods’ approximation error.

Chapter 12 describes an application of the fast `NETLSD` computation in machine learning. We use `NETLSD` to capture the intrinsic² geometry of data. Focus on the intrinsic geometry of data allows us to study data embedded in spaces of different dimensionality such as representations of images produced by different layers of neural networks. We also apply our measure to the evaluation of generative models and comparison of language structures.

The following four chapters in this part are based on work as cited below:

- Chapter 9 “Spectral Graph Similarity” extends [TMK⁺18b].
- Chapter 10 “Learning a Spectral Graph Similarity” extends [TMK⁺18a].
- Chapter 11 “Efficient Approximation of Spectral Graph Representations” extends [TMP20].
- Chapter 12 “Spectral Graph Similarities for Comparing Distributions” extends [TMM⁺20a].

We now proceed with an overview of the related work relevant to all chapters.

8.1 RELATED WORK

We group the related work into three categories: (i) direct comparison methods; (ii) graph kernels, and (iii) feature-based representations. We give an overview of each category below; this dissertation contributes to the feature-based representation field.

8.1.1 DIRECT COMPARISON METHODS

Graph edit distance (`GED`) [SF83] is the minimal number of edit operations needed to transform one graph into another; unfortunately, its calculation is `NP`-hard [GJ02] and even hard to approximate (`APX`-hard) [Lin94], as it implies determining a *correspondence* among the compared graphs’ nodes, a computationally hard task to begin with. While `GED` admits heuristic approximation [RB09, FSF⁺15] and indexing schemes [ZTW⁺09, ZQYC12, LZ17],

2. Properties of surfaces that are independent of the embedding are called intrinsic.

it is not applicable for generic comparison among large graph collections. Besides, GED treats all edit operations as equal, without discerning the extent to which they may alter the graph topology. Thus, even if the computational obstacles were surmounted, GED would still be unsatisfactory as a measure for multi-scale and different-size graph comparisons.

Some application-specific techniques (e.g., anomaly detection in time-evolving graphs) assume given correspondence between nodes to alleviate the computational burden [KVF13, PDGM10]. Such methods are limited to their specific application scenarios.

8.1.2 GRAPH KERNELS

Graph kernels [KJM20] are similarity functions among graphs, which typically perform an implicit transformation of graph structure to compare two graphs. For example, the Shortest-path kernel [BK05] compares the lengths of all shortest paths between vertices in two graphs. Typically, kernel methods perform *implicit* computations to compute the distance between two graphs without relying on explicit representations. Therefore, their computation time is at least quadratic³ to the number of graphs and requires expensive on-demand computations at comparison time.

Graphlet kernels [SB09] propose to count the number of $\{3, 4\}$ -sized subgraphs (motifs) in a graph and use these counts as features. This kernel's complexity is exponential to the size of mined subgraphs, while the resulting vectors' high dimensionality prohibits their usage as feature vectors.

Weisfeiler–Lehman [SSL⁺11] kernels propose to aggregate discrete information on nodes via the analog of the color refinement heuristic for isomorphism testing. In the absence of node features, node degrees are commonly used as a replacement. This kernel operates in the number of edges in a graph, but it is only available when the features are discrete. An extension to continuous attributes via hashing was proposed in [MKKM16].

The Multi-scale Laplacian Graph kernel (MLG) [KP16] is particularly relevant for this thesis since it is the first to consider multi-scale comparisons of graphs. This kernel models the propagation of information across edges. It iteratively sums information resulting in increasingly global comparisons. However, it has complexity cubic in the number of eigenvalues employed.

3. For explicit representations, approximate nearest neighbor search alleviates the problem of pairwise comparisons.

8.1.3 FEATURE-BASED REPRESENTATIONS

Representation-based methods generate a one-off graph feature vector, based on statistical properties, and use it in subsequent inter-graph comparisons. Preliminary works in this area [BBGO11, BKERF13, BBK⁺16] handcraft features by aggregating local graph properties such as a node's and its neighbors' degrees. Such representations are easy to compute yet focus on local characteristics and are oblivious to global features.

NetSimile is the first method to examine size-independent graph similarity. The proposed way is crude—the method computes and aggregates statistics of nodes and their ego-networks⁴. Despite the simplicity, the approach is reasonably effective and scales linearly to the number of edges in graphs. However, it does not consider any global information about the graph.

Family of Spectral Distances (FGSD) [VZ17] produces a high-dimensional sparse representation as a histogram on the dense biharmonic graph kernel; however, FGSD does not capture multi-scale graph features, and it does not allow comparing graphs of different sizes. Further, it is inapplicable to reasonably large graphs due to its quadratic time complexity.

4. An ego-network of a node is a subgraph with all its neighbors.

GRAPH comparison is ubiquitous in graph mining and analytics. However, it is a hard task in terms of the employed similarity measure's expressiveness and computational efficiency. Arguably, an ideal means for graph comparison should fulfill the following desiderata: first, it should be indifferent to the order of the nodes; we call this property *permutation-invariance*. Second, it would enable graph comparisons both at a local level (expressing, e.g., atomic bond differences among chemical compounds) and at the global level (capturing, e.g., the different topologies of social networks); we call this facility *scale-adaptivity*. Third, it would detect structural similarity regardless of network magnitude (discerning, e.g., the similarity of two criminal networks of different sizes); we call this aptitude *size-invariance*. Unfortunately, no existing means for graph comparison satisfies all three of these requirements.

Apart from these qualitative requirements, a viable means for graph comparison should be *efficiently computable*. Graph analytics tasks often require pairwise graph comparisons within a large collection of graphs, hence should be ideally done in *constant time*, after preprocessing. Unfortunately, existing methods fare even worse in this respect. A popular distance measure among graphs, graph edit distance (GED) [SF83] is NP-hard and APX-hard to compute [Lin94]; intensive research efforts in GED-based graph comparison [ZTW⁺09, ZZL⁺15, YWCW15, GH16, LZ17] have not escaped this reality. Similarly, *graph kernel methods* [KJM20] lack an explicit graph representation and they do not scale well either.

This chapter develops a permutation- and size-invariant, scale-adaptive, and scalable method for graph comparison. We propose an expressive *graph representation*, NETLSD, grounded on spectral graph theory, that allows for *constant-time* similarity computations at several scales; NETLSD extracts compact graph signatures based on the heat or wave kernel of the Laplacian, which inherit the formal invariance properties of the Laplacian spectrum.

9.1 PROBLEM STATEMENT

A *representation* is a function $\sigma : \mathcal{G} \rightarrow \mathbb{R}^d$ from any graph G in a collection of graphs \mathcal{G} to a d -dimensional real-vector; we denote j -th element of the representation as $\sigma_j(G)$. A *representation-based distance* is a function $d^\sigma : \mathbb{R}^d \times \mathbb{R}^d \rightarrow \mathbb{R}_0^+$ on the representations of two graphs $G_1, G_2 \in \mathcal{G}$ that returns a positive real number. We aim to devise a representation-based distance that works independently of graph size.

9.1.1 EXPRESSIVE GRAPH COMPARISON

Our distance should support data mining tasks, such as clustering, classification, and distance-based anomaly detection. Therefore, we require our distance to be a *pseudometric*; namely, it should fulfill the following properties:

- *Symmetry*, for any $G_1, G_2 \in \mathcal{G}$:

$$d^\sigma(\sigma(G_1), \sigma(G_2)) = d^\sigma(\sigma(G_2), \sigma(G_1))$$
- *Triangle inequality*, for any $G_1, G_2, G_3 \in \mathcal{G}$:

$$d^\sigma(\sigma(G_1), \sigma(G_3)) \leq d^\sigma(\sigma(G_1), \sigma(G_2)) + d^\sigma(\sigma(G_2), \sigma(G_3))$$

These properties characterize a large family of distances, yet do not reflect their expressiveness. We require *expressive distances* to be permutation-invariant, scale-adaptive, and size-invariant.

Permutation-invariance implies that if two graphs' structure are the same (i.e., if the two graphs are isomorphic) the distance of their representations is zero. A graph $G_1 = (V_1, E_1)$ is isomorphic to another graph $G_2 = (V_2, E_2)$, or $G_1 \simeq G_2$, if there exists a bijective function $\mu : V_1 \rightarrow V_2$ such that $(\mu(u), \mu(v)) \in E_2$ for every $(u, v) \in E_1$.

Property 1 (Permutation-invariance). A distance d^σ on representation σ is permutation-invariant iff:

$$\forall G_1, G_2, G_1 \simeq G_2 \Rightarrow d^\sigma(\sigma(G_1), \sigma(G_2)) = 0$$

Scale-adaptivity implies that a representation accounts for both local (edge and node) and global (community) graph features. A global feature cannot be captured by *any* combination of features on nodes at distance $r < \mathcal{D}(G) - 1$, where $\mathcal{D}(G)$ is the diameter (longest shortest-path length) of G . Let the set of all subgraphs of G be $\xi(G) = \{g \sqsubset G : \mathcal{D}(g) < \mathcal{D}(G)\}$. We define scale-adaptivity as the property of a representation σ having at least one local feature (i.e., derived only from information encoded in subgraphs $\xi(G)$), and at least one global feature (i.e., derived by strictly more than the information encoded in any $\xi(G)$). Using local features only, a similarity measure would deem two graphs sharing local patterns to have near-zero distance although their global properties (such a page-rank features) may differ, and, in reverse, relying on global features only would miss local structures (such as edge distributions). We aim for a representation adaptive to both local and global structures on demand.

Property 2 (Scale-adaptivity). A representation σ is scale-adaptive iff it contains both local features σ_i and global features σ_j :

- *Local Feature:* $\forall G \exists f(\cdot) : \sigma_i = f(\xi(G))$
- *Global Feature:* $\forall G \nexists f(\cdot) : \sigma_j = f(\xi(G))$

Size-invariance is the capacity to discern that two graphs represent the same phenomenon at different magnitudes (e.g., two criminal circles of similar structures but different sizes should have near-zero distance). We can think of a graph as a representation of a metric space¹ with a small intrinsic dimension. We would then like to abstract away the particular way of sampling that space. Size-invariance postulates that if two graphs originate from the sampling of the same domain \mathcal{M} , they should be deemed similar.

1. It is sometimes convenient to think about the underlying manifold a graph is sampled from.

Property 3 (Size-invariance). A size-invariant distance d^σ fulfills:

$$\forall \mathcal{M} : G_1, G_2 \text{ sampled from } \mathcal{M} \Rightarrow d^\sigma(\sigma(G_1), \sigma(G_2)) = 0$$

An expressive means of graph comparison should employ a representation fulfilling the above properties. In the following, we present NETLSD, a graph representation that allows for easy and expressive graph comparison.

9.2 NETWORK LAPLACIAN SPECTRAL DESCRIPTOR

Defining a representation fulfilling the requirements of permutation-, scale-, size- invariance, and efficiency is challenging; in general, complex structures are hard to compare. Hence, we transfer the problem to the spectral domain. A useful metaphor is heating the graph's nodes and observing heat diffusion as time passes. Another useful metaphor is a system of masses corresponding to the graph's nodes and springs corresponding to its edges. The propagation of mechanical waves through the graph is another way to capture its structural invariants. In both cases, the overall process describes the graph in a permutation-invariant manner and embodies more global information as time elapses. Our representation employs a *trace signature* of heat diffusion or wave propagation process over time. We compare two graphs via the L_2 distance among trace signatures sampled at selected time scales.

9.2.1 SPECTRA AS REPRESENTATIONS

2. For both unnormalized and normalized Laplacians.

The set of eigenvalues $\{\lambda_1, \dots, \lambda_n\}$ of graph's Laplacian matrix² is called the *spectrum* of a graph. The normalized Laplacian $\mathcal{L} = \mathbf{I} - \mathbf{D}^{-1/2} \mathbf{A} \mathbf{D}^{-1/2}$, as opposed to the unnormalized version $\mathbf{L} = \mathbf{D} - \mathbf{A}$, has its spectrum bounded by $0 \leq \lambda_i \leq 2$.

Belkin and Niyogi [BN07] showed that eigenvectors of the normalized Laplacian of a point cloud graph converge to the eigenfunction of the Laplace-Beltrami operator [Ber12] on the underlying Riemannian manifold. In general, the normalized Laplacian has more attractive theoretical properties than its unnormalized counterpart [VLBB08].

The Laplacian spectrum encodes important graph properties, such as the normalized cut size [SMoo] used in spectral clustering. Likewise, the normalized Laplacian spectrum can determine whether a graph is bipartite, but not the number of its edges [Chu97]. Rather than using the Laplacian spectrum per se, we consider an associated heat diffusion process on the graph, to obtain a more expressive representation, in a manner reminiscent of random walk models [Chuo7].

The *heat equation* associated with the Laplacian is

$$\frac{\partial \mathbf{u}_t}{\partial t} = -\mathcal{L} \mathbf{u}_t, \quad (9.1)$$

where \mathbf{u}_t are scalar values on vertices representing the heat of each vertex at time t . The solution to the heat equation provides the heat at each vertex at time t , when the initial heat \mathbf{u}_0 is initialized with a fixed value on one of the vertices. Its closed-form solution is given by the $n \times n$ *heat kernel* matrix:

$$\mathbf{H}_t = e^{-t\mathcal{L}} = \sum_{j=1}^n e^{-t\lambda_j} \phi_j \phi_j^\top, \quad (9.2)$$

where $(\mathbf{H}_t)_{ij}$ represents the amount of heat transferred from vertex v_i to vertex v_j at time t . We can also compute the heat kernel matrix directly by exponentiating the Laplacian eigenspectrum [Chu97]:

$$\mathbf{H}_t = \Phi e^{-t\Lambda} \Phi^\top \quad (9.3)$$

As the heat kernel involves pairs of nodes, it is not directly usable to compare graphs. We rather consider the *heat trace* at time t :

$$h_t = \text{tr}(\mathbf{H}_t) = \sum_j e^{-t\lambda_j}. \quad (9.4)$$

Then our NETLSD representation consists of a *heat trace signature* of graph G , i.e., a collection of heat traces at different *time scales*, $h(G) = \{h_t\}_{t>0}$.

Alternative signatures. The heat kernel can be viewed as a family of *low-pass filters*, $F(\lambda) = e^{-\lambda t}$, parameterized by the scale parameter t , hence it contains low frequency (i.e., large-scale) information at every scale. Other kernels emphasize different frequencies. For example, the *wave equation*,

$$\frac{\partial^2 \mathbf{u}_t}{\partial t^2} = -\mathcal{L} \mathbf{u}_t, \quad (9.5)$$

which describes the amplitude \mathbf{u}_t of a wave propagating in a medium, has, in its turn, a solution given by the *wave kernel*:³

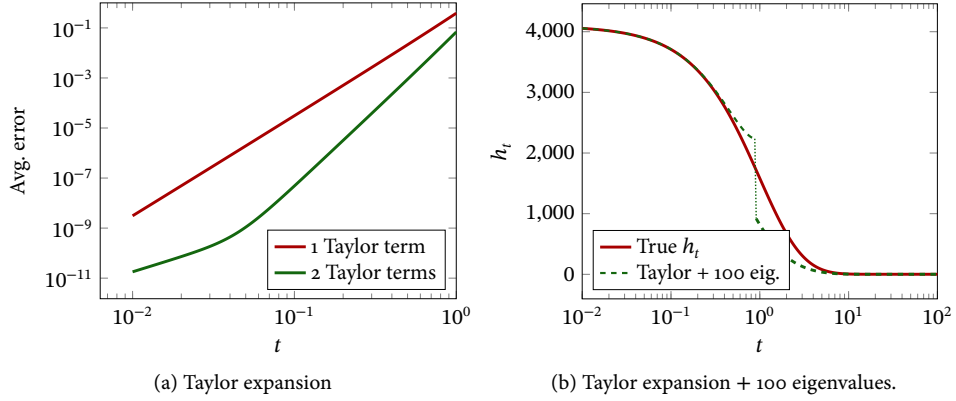
$$\mathbf{W}_t = e^{-it\mathcal{L}} = \sum_{j=1}^n e^{-it\lambda_j} \phi_j \phi_j^\top \quad (9.6)$$

and a corresponding *wave trace signature* with $t \in [0, 2\pi)$:

$$w_t = \text{tr}(\mathbf{W}_t) = \sum_j e^{-it\lambda_j}. \quad (9.7)$$

3. Note that this equation features the complex exponential.

Figure 9.1: Taylor expansion for NetLSD approximation: (a) Relative approximation error of normalized h_t for Erdős–Rényi random graphs, varying time scale t , and (b) the heat trace approximation by two Taylor terms and 100 eigenvalues on a random SBM graph.



9.2.2 SCALING TO LARGE GRAPHS

The full eigendecomposition of the Laplacian $\mathcal{L} = \Phi\Lambda\Phi^\top$ takes $\mathcal{O}(n^3)$ time and $\Theta(n^2)$ memory. This allows to compute signatures of graphs with over a thousand nodes in less than a second on commodity hardware, yet renders direct computation impossible for larger graphs. Thus, we need to approximate heat trace signatures. We propose two different methods to that end.

Our first proposal is to use a Taylor expansion; while this mathematical tool provides a dubious approximation of a matrix exponential, as its convergence rate depends on the largest eigenvalue [MVL03], it is useful on *small time scales* t , and allows for an inexpensive computation of its first two terms,

$$h_t = \sum_{k=0}^{\infty} \frac{\text{tr}((-t\mathcal{L})^k)}{k!} \approx n - t \text{tr}(\mathcal{L}) + \frac{t^2}{2} \text{tr}(\mathcal{L}^2). \quad (9.8)$$

These first two terms are easily computed, even for very large graphs, as $\text{tr}(\mathcal{L}) = n$ and $\text{tr}(\mathcal{L}^2) = \sum_{ij} \mathbf{L}_{ij}^2$ since \mathcal{L} is self-adjoint. This provides a principled way to compare two graphs locally in $\mathcal{O}(m)$.

Figure 9.1a depicts the error in approximating the normalized heat trace by a Taylor expansion for random graphs of varying sizes; this error is independent of graph size, and stays low until time scale 1. At *larger time scales*, the influence of high frequencies (i.e., the higher part of the spectrum) on the heat trace decreases. Thus, one can approximate the heat trace signature using the lower part of the eigenspectrum, as in shape analysis [SOG09, VBCG10, BB11, LB14]. Thus, we may apply the low-order Taylor expansion for *small* t and the trun-

cated spectrum approximation for *large* t . However, this approach misses out on the medium scale, as Figure 9.1b illustrates. Besides, this approximation technique does not allow us to compare graphs of different sizes, since the spectrum discretization ratio is not normalized across the networks.

We conclude that the Taylor expansion is useful on very large graphs, on which eigendecomposition is prohibitive. For manageable graph sizes, we adopt a more accurate strategy [MVL03] based on approximating the *eigenvalue growth rate*, as in [VBCG10]: we compute k eigenvalues on *both* ends of the spectrum, and interpolate a *linear growth* of the interloping eigenvalues. This strategy assumes that on the medium scale the manifold defining the graph is two-dimensional, as Weyl’s law of asymptotic eigenvalue growth suggests [Wey11]. Since we only need to compute extreme eigenvalues, we use the block Krylov-Schur implementation in SLEPc [HRV05]. Graph Laplacians always have a zero eigenvalue with eigenspace dimension equal to the number of connected components; we deflate the search space for the eigensolver thanks to this property. In our experimental study, we employ this *interpolation technique*; In Section 9.3.1, we evaluate its approximation quality.

9.2.3 PROPERTIES OF THE HEAT TRACE

We now discuss how the heat trace signature achieves our target properties.

Permutation-invariance. The permutation invariance of $h(G)$ follows from the properties of the spectrum: isomorphic graphs are isospectral, hence their respective heat trace signatures are equal.

Scale-adaptivity. The heat kernel can be seen as continuous-time random walk propagation, and its diagonal (also referred to as the autodiffusivity function or the heat kernel signature) can be seen as a continuous-time Page-Rank [Chuo7]. Figure 9.2 shows the heat kernel signature with small (a), medium (b), and large (c) t ; with large t , the heat tends to focus on central nodes.

As t approaches zero, the Taylor expansion yields $\mathbf{H}_t \simeq \mathbf{I} - t\mathcal{L}$, meaning the heat kernel depicts *local connectivity*. On the other hand, for large t , $\mathbf{H}_t \simeq \mathbf{I} - e^{-\lambda_2 t} \phi_2 \phi_2^\top$, where ϕ_2 is the Fiedler vector used in spectral graph clustering [SMoo], as it encodes global connectivity. Thus, the heat kernel

Figure 9.2: The diagonal of \mathbf{H}_t at different scales on the Karate club graph; at a large scale, the field reflects node centrality.

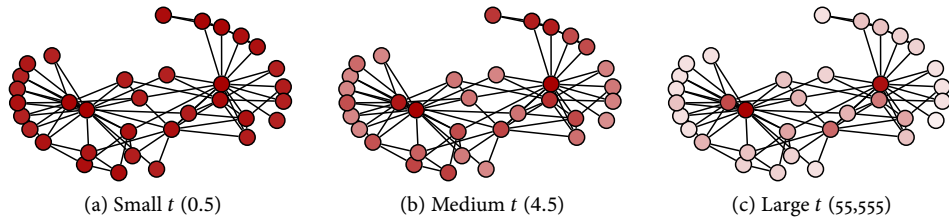
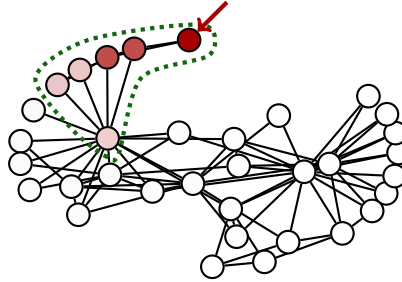


Figure 9.3: Visualization of the diagonal in the heat kernel matrix \mathbf{H}_t for the pointed vertex at scale $t = 0.3$.



localizes around its diagonal, and the degree of localization depends on the scale t ; it can thereby be tuned to produce both local and global features.

Figure 9.3 illustrates heat kernel locality, focusing on a single row of \mathbf{H}_t corresponding to the node marked with a red arrow; the kernel is localized in the region marked with a dotted line.

Size-invariance. While the normalized Laplacian alleviates the problem of different edge densities [Chu97], the Taylor expansion in Equation 9.8 manifests that $h(G)$ contains information about the number of nodes. Fortunately, we can employ *neutral* graphs for normalization, namely the empty and the complete graph with n nodes. Eigenvalues of the normalized Laplacian of an empty graph \bar{K}_n of size n are all zero; for a complete graph K_n , they are given by a vector of one zero and $n - 1$ ones. Thus, the heat traces of these graphs are analytically computed as:

$$h_t(\bar{K}_n) = \frac{1}{n}$$

$$h_t(K_n) = 1 + (n - 1)e^{-t},$$

and their wave traces are:

$$w_t(\bar{K}_n) = \frac{1}{n}$$

$$w_t(K_n) = 1 + (n - 1)\cos(t).$$

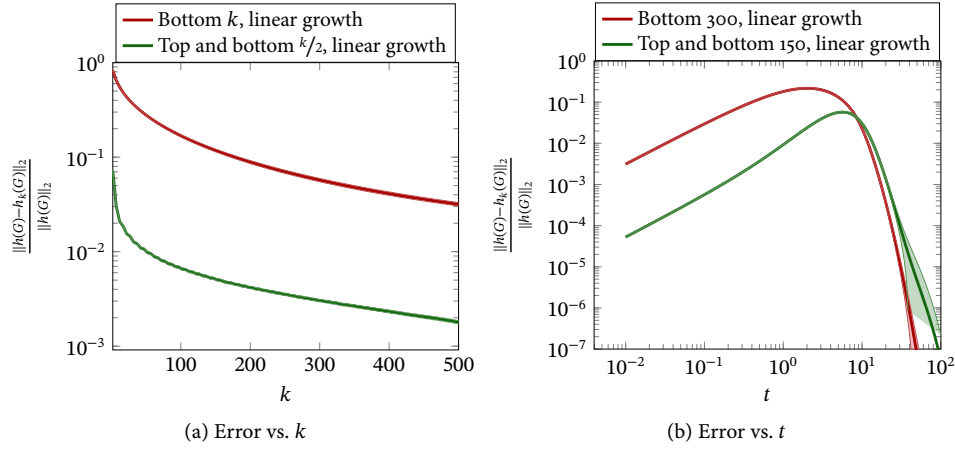


Figure 9.4: Relative error in spectrum computation of $h(G)$, averaged across 2085 graphs.

Either option can be used to normalize the heat or wave trace signatures. Such normalization can be interpreted as a modification of the corresponding diffusivity tensor (for the heat trace) or the elasticity tensor (for the wave trace). We provide more details in Section 9.3.

9.2.4 CONNECTION TO COMPUTATIONAL GEOMETRY

Mémoli [Mém11] suggests a spectral definition of the Gromov–Wasserstein distance between Riemannian manifolds based on matching the heat kernels at all scales. The cost of matching a pair of points (x, x') on manifold \mathcal{M} to a pair of points (y, y') on manifold \mathcal{N} at scale t is given by

$$\Gamma(x, y, x', y', t) = |H_t^{\mathcal{M}}(x, x') - H_t^{\mathcal{N}}(y, y')|.$$

The distance between the manifolds is then defined in terms of the infimal measure coupling:

$$d(\mathcal{M}, \mathcal{N}) = \inf_{\mu} \sup_{t>0} e^{-2(t+t^{-1})} \|\Gamma\|_{L^2(\mu \times \mu)},$$

where the infimum is sought over all measures on $\mathcal{M} \times \mathcal{N}$ marginalizing to the standard measures on \mathcal{M} and \mathcal{N} . Mémoli [Mém11] shows that this distance can be lower bounded by

$$d(\mathcal{M}, \mathcal{N}) \geq \sup_{t>0} e^{-2(t+t^{-1})} |h_t^{\mathcal{M}} - h_t^{\mathcal{N}}|.$$

Table 9.1: Dataset properties.

<i>dataset</i>	$ \mathcal{G} $	$ Y $	vertices $ V $		
			min.	avg.	max.
Mutag	188	2	10	17.93	28
PTC	344	2	2	25.56	109
Proteins	1113	2	4	39.06	620
NCI1	4110	2	3	29.87	111
NCI109	4127	2	4	29.68	111
Enzymes	600	6	2	32.63	126
D&D	1178	2	30	284.3	5748
Collab	5000	3	32	74.49	492
IMDB-B	1000	2	12	19.77	136
IMDB-M	1500	3	7	13.00	89
Reddit-S	9543	2	100	337.8	999
Reddit-M	2000	2	6	429.6	3782
Reddit-X	2085	—	1001	1447.2	5242
Reddit-L	291	2	4445	97491	1632141

In other words, the lower bound is the scaled L_∞ distance between the heat trace signatures $h(\mathcal{M})$ and $h(\mathcal{N})$.

We effectively adopt this result *mutatis mutandis* to graphs, substituting the Laplace–Beltrami operator of the manifold with the normalized graph Laplacian; we then sample the heat trace signature at a finite number of scales, rendering it a versatile vector representation of a graph, as other works produce vector representations of graph vertices [RSF17, TMKM18]. This lower bound implies that, if the distance between heat trace signatures is sufficiently large, the compared graphs cannot be similar. Unlike the spectral Gromov–Wasserstein distance, distances between heat trace signatures are easily *indexable*; thus, the derived bound allows for efficiently pruning of dissimilar graphs while working with the index alone, as with any lower-bounding scheme for high-dimensional search [TYSK09, KK11].

9.3 EXPERIMENTS

4. Source code and data are available at <https://github.com/xgfs/NetLSD>

We evaluate⁴ NETLSD against NetSimile [BKERF13] and FGSD [VZ17]. We run the experiments on a 20-core Intel Xeon CPU E5-2640v4, 3.20GHz ma-

chine with 256GB RAM. Each method is assessed on the *best parameters* through cross-validation. We require each method to complete within one day, or else an early termination is issued. We used graph-tool [Pei14b] for graph manipulation and synthetic graph generation.

We compare to the following comparison methods:

NetSimile [BKERF13] is a representation using handcrafted features obtained by aggregating statistics on nodes and edges (e.g., average degree, standard deviation of the degree of the neighbors). For each graph, the resulting representation has 35 dimensions. As recommended in [BKERF13], we use the Canberra⁵ distance for comparison.

FGSD [VZ17] is a method that computes histograms on the biharmonic kernel of the graph. Such histograms typically bear large dimensionality ($\geq 50,000$) using the recommended bin-width 0.0001.

5. https://en.wikipedia.org/wiki/Canberra_distance

Parameter settings. Unless otherwise stated, we repeat each experiment 100 times and report the average across all trials. We instantiate NETLSD with both heat $h(G)$ and wave $w(G)$ trace signature described in Section 9.2.1. In addition, we evaluate our normalized versions: the normalization with the empty graph $h(G)/h(\bar{K})$, $w(G)/w(\bar{K})$ and the normalization with the complete graph $h(G)/h(K)$, $w(G)/w(K)$. To build NETLSD signature vectors, we need to sample a number of traces, i.e., values of t . After experimentation, we settled for 250 values evenly spaced on the logarithmic scale in the range $[10^{-2}, 10^2]$; we attested this to be a good choice in terms of the quality-size tradeoff, hence use these settings in all experiments, with both heat and wave signatures. With small graphs, we employ the full eigendecomposition to produce the trace for each t value. With the larger Reddit-L graph, we use 300 eigenvalues, 150 from each side of the eigenspectrum, by default, unless indicated otherwise; we validate these choices in Section 9.3.1 (cf. Figure 9.4).

<i>dataset</i>	$h(G)$	$h(G)/h(\bar{K})$	$h(G)/h(K)$	$w(G)$	$w(G)/w(\bar{K})$	$w(G)/w(K)$	FGSD	NetSimile
Mutag	76.03	79.12	78.22	78.18	79.72	79.38	77.79	77.11
PTC	56.41	62.53	63.11	58.55	64.28	60.46	54.75	62.12
Proteins	91.81	94.90	95.31	93.04	89.00	91.27	60.11	85.73
NCI1	69.74	74.55	69.89	70.54	74.14	70.90	64.08	58.58
NCI109	68.60	73.76	69.48	70.75	73.96	70.67	64.28	58.76
Enzymes	92.51	95.20	95.70	94.03	90.77	90.10	53.93	87.38
Collab	59.82	65.85	69.74	69.01	70.35	71.89	55.18	54.43
IMDB-B	67.18	70.58	69.22	75.26	75.54	74.13	56.23	54.44
IMDB-M	74.45	75.51	75.54	77.99	78.68	76.97	56.31	48.06

Table 9.2: Classification ROC AUC in detecting whether a graph is real.

<i>dataset</i>	$h(G)$	$h(G)/h(\bar{K})$	$h(G)/h(K)$	$w(G)$	$w(G)/w(\bar{K})$	$w(G)/w(K)$	FGSD	NetSimile
Mutag	86.47	85.32	84.66	83.35	81.72	82.22	84.90	84.09
PTC	55.30	52.76	51.16	54.97	54.53	53.40	60.28	61.26
Proteins	64.89	65.73	65.36	66.80	65.58	62.27	65.30	62.45
NCI1	66.49	67.44	64.82	70.78	67.67	62.19	75.77	66.56
NCI109	65.89	66.93	64.78	69.32	67.08	63.53	74.59	65.72
Enzymes	31.99	33.31	37.19	40.41	35.78	28.75	41.58	33.23
D&D	69.86	68.38	67.09	68.77	65.39	65.12	70.47	64.89
Collab	68.00	69.42	69.70	75.77	77.24	67.37	73.96	73.10
IMDB-B	68.04	70.17	69.45	68.63	69.33	61.67	69.54	69.20
IMDB-M	40.51	40.34	40.10	42.66	42.00	39.71	41.14	40.97
Reddit-M	43.12	40.62	39.08	41.49	38.65	41.24	41.61	41.32
Reddit-S	83.67	81.77	83.73	84.49	70.47	79.46	88.95	89.65

Table 9.3: Accuracy in 1-NN classification.

Datasets. We use six graph collections from bioinformatics [YV15] and three social networks. Collab, and IMDB-(B/M) graph collections are obtained by sampling neighborhoods of nodes in a collaboration and movie network, respectively. Reddit-S, Reddit-M, and Reddit-L are selected among small (at most 1000 nodes), medium (at most 4000 nodes) and large (> 4000 nodes) subreddits⁶, respectively. We report their main characteristics in Table 9.1: number of graphs $|\mathcal{G}|$, number of labels for classification $|Y|$, minimum, average, and maximum number of vertices.

6. <https://dynamics.cs.washington.edu/data.html>

9.3.1 APPROXIMATION QUALITY

First, we study the quality of the approximation technique of Section 9.2.2. Figure 9.4 reports relative error results in terms of deviation from the exact version, averaged over 2085 Reddit-X graphs. Specifically, Figure 9.4a shows the quality of the approximation varying the number of eigenvalues k , using only the k -smallest (blue line) and the $k/2$ -smallest and $k/2$ -largest eigenvalues (green line). Using eigenvalues from the two ends of the spectrum achieves consistently better performance. Figure 9.4b shows the impact of t in the approximation. The prediction is easier at large t , as the spectrum converges to the constant value 1; at medium t values, the approximation is harder. Still, the use of the lowest and largest eigenvalues delivers almost an order of magnitude higher accuracy than using only one side of the spectrum, vindicating our choice of approximation.

9.3.2 IDENTIFYING REAL-WORLD NETWORKS

We devise a binary classification task of detecting whether a graph is real or synthetic. Such tasks are critical in anomaly detection and detecting bots and trolls in social networks. To render the task challenging enough, rather than generating purely random graphs (which are easy to detect), we produce synthetic graphs by rewiring real ones while preserving their degree distribution, using 10 iterations of shuffling all their edges via Metropolis-Hastings sampling [Pei14a]. We consider a label indicating whether the graph has been rewired. We label 80% of the dataset and test 20%, predicting a graph's label as that of its Nearest Neighbor (1-NN) by each similarity measure, and report the average across 100 trials. Table 9.2 shows our results, in terms of ROC AUC, for all datasets and measures. These results confirm the effectiveness of NETLSD.

9.3.3 GRAPH CLASSIFICATION

We now assess NETLSD on a traditional task of 1-NN graph classification, using labels as provided in the datasets and splitting training and testing as in the previous experiments. Table 9.3 reports the quality in terms of ROC AUC, averaged over 1000 trials. Again, NETLSD is on a par with other methods.

Table 9.4: Accuracy of 1-NN classification on Reddit-L.

Method	k		
	100	200	300
$h(G)$	68.91	68.89	68.01
$h(G)/h(\bar{K})$	62.69	61.74	61.88
$h(G)/h(K)$	70.11	69.40	70.88
$w(G)$	71.27	69.93	68.93
$w(G)/w(\bar{K})$	64.79	65.81	65.90
$w(G)/w(K)$	64.51	69.49	72.64

We additionally report in Table 9.4 the quality results for the largest dataset at our disposal (Reddit-L). NetSimile and FGSD cannot scale to such large graphs and therefore we do not report on them; on the other hand, NETLSD processes graphs with millions of nodes and attains good overall quality.

9.3.4 DISCERNING COMMUNITY STRUCTURES

Communities are sets of vertices sharing common characteristics in terms of connectivity and attributes. Community detection [KG14, SM00] is one of the prototypical, yet only partially solved, tasks. An expressive graph distance should distinguish graphs with community structure from those without.

To evaluate the expressiveness of NETLSD in terms of communities, we devise a graph classification experiment: We generate some graphs with community structure and some without, and the classifier’s task is to predict whether test set graphs have community structure. We employ a simple 1-NN classifier, as our goal is to test the representation’s expressiveness rather than the performance of the classifier. We generate 1000 random graphs with Poisson degree distribution $\mathcal{P}(\lambda)$ with mean degree $\lambda = 10$ and fixed size. Then, we sample another 1000 graphs from the stochastic block model (SBM) [KN11] with 10 communities, following the same degree distribution. The SBM produces graphs with clear community structure as opposed to random ones. We use 80% of the dataset for training, and 20% for testing, repeat the experiment with different training and testing sets, and report the average across trials. Table 9.5 reports the average quality for discerning SBM graphs by 1-NN classification in terms of ROC AUC, as we vary the graph size in (64,128,256,512,1024). We observe that NETLSD significantly outperforms the competitors and improves

<i>Method</i>	Number of nodes n				
	64	128	256	512	1024
$h(G)$	57.40	68.37	77.42	82.83	84.63
$h(G)/h(\bar{K})$	57.42	68.40	77.41	82.84	84.63
$h(G)/h(K)$	56.71	67.96	77.50	83.31	85.12
$w(G)$	57.47	66.97	73.95	78.43	80.26
$w(G)/w(\bar{K})$	57.44	66.98	73.96	78.43	80.25
$w(G)/w(K)$	56.75	66.26	73.06	77.05	78.76
FGSD	58.00	55.73	55.46	53.43	51.57
NetSimile	65.73	61.31	61.51	61.86	61.58

Table 9.5: Accuracy in detecting graphs with communities.

<i>Method</i>	$n \sim \mathcal{P}(\lambda)$				
	64	128	256	512	1024
$h(G)$	54.39	59.01	60.82	57.99	53.80
$h(G)/h(\bar{K})$	54.53	62.27	70.83	76.45	78.40
$h(G)/h(K)$	54.37	60.93	66.86	68.24	65.23
$w(G)$	56.23	63.77	69.57	71.66	70.34
$w(G)/w(\bar{K})$	55.51	63.85	72.12	77.59	79.39
$w(G)/w(K)$	56.69	64.92	71.81	75.91	77.50
FGSD	55.44	54.99	53.86	52.74	50.92
NetSimile	59.55	56.57	59.41	66.23	60.58

Table 9.6: Accuracy in detecting graphs with communities, Poisson distribution of graph size.

in quality on larger graphs. While discriminating very small communities is intuitively harder than distinguishing large ones, the performance of both FGSD and NetSimile drops with increasing size, suggesting that these methods only capture local, small-scale variations. This result verifies the capacity of NETLSD to capture global and local characteristics.

In the second experiment we evaluate the size-invariance of NETLSD: the task is to discriminate fixed-size communities in graphs of increasing size. Therefore, we sample the number of nodes n from a Poisson distribution $\mathcal{P}(\lambda = n)$ with variance λ , and then generate a network with n nodes. Again, we repeat the process 1000 times with purely random generated graphs and 1000 with random graphs with community structure, and perform classifi-

Table 9.7: Accuracy in detecting graphs with communities, Uniform distribution of graph size.

Method	$n \sim \mathcal{U}(10, \lambda)$				
	64	128	256	512	1024
$h(G)$	51.16	51.98	51.62	51.05	50.08
$h(G)/h(\bar{K})$	51.19	53.36	56.63	59.10	59.71
$h(G)/h(K)$	51.69	53.67	55.43	55.29	53.43
$w(G)$	52.38	55.89	59.81	61.18	59.13
$w(G)/w(\bar{K})$	51.61	54.67	57.83	60.06	61.01
$w(G)/w(K)$	52.63	57.48	62.85	67.95	71.19
FGSD	57.92	55.62	54.94	52.74	52.15
NetSimile	63.63	58.31	55.75	54.34	53.11

cation as before. Table 9.6 reports the results, for increasing λ . Once again, NETLSD outperforms the competitors. This result confirms that the normalization proposed in Section 9.2.3 is effective in detecting community structures in graphs of different size. Table 9.6 also indicates that, while both normalizations are similarly effective with the wave kernel, the complete-graph normalization produces worse, yet competitive, results with the heat kernel.

Last, we assess the size-invariance of NETLSD in a tough regime: discriminating communities in graphs with a number of nodes *chosen uniformly at random*. We sample the number of nodes n from the uniform distribution $\mathcal{U}(10, \lambda)$; other experimental settings remain the same. Table 9.7 reports the results with growing λ . We observe that NETLSD outperforms the competitors once again, yet the results suggest that this task is more challenging. Complete-graph normalization for the wave kernel and empty-graph normalization for the heat kernel perform best. We conclude that, for fully size-agnostic comparisons, normalization should be chosen carefully.

9.3.5 CASE STUDY

Here, we visualize the discovery potential of NETLSD as a similarity measure. We run a *furthest pair query* on all graphs in our collection bar those in the Reddit-L dataset, which are harder to visualize readably; thereby, we discover the two graphs of lowest similarity, using the normalized heat kernel. Figure 9.5 shows those two graphs: a *protein interaction network* from the D&D dataset and an *enzyme's tertiary structure* from the Enzymes dataset. These two graphs

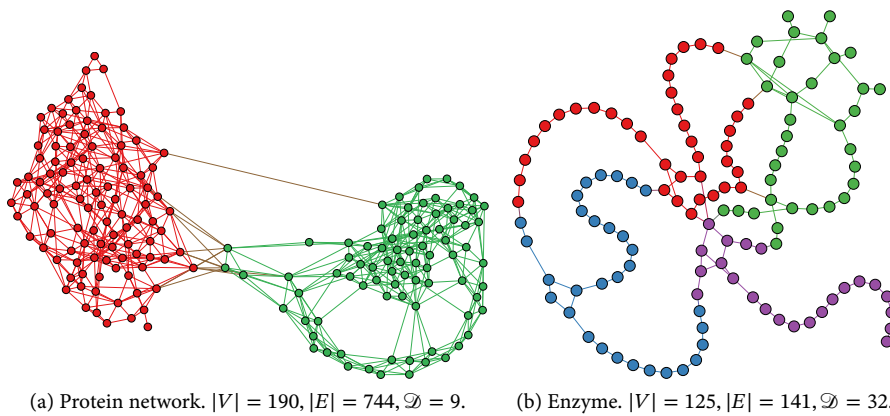


Figure 9.5: Two most dissimilar graphs by $h(G)/h(\bar{K})$ across all small-size graphs in datasets used. Communities are colored.

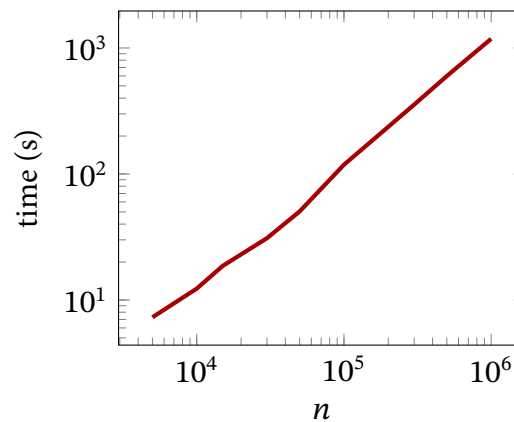


Figure 9.6: Time to compute 300 eigenvalues on both ends of the spectrum and the approximation of $h(G)$.

are conspicuously different across multiple scales, from local patterns to global structure. The interaction network in Figure 9.5a is a small-world graph with a large clustering coefficient (0.425) and small diameter ($\mathcal{D} = 9$), whereas the tertiary structure in Figure 9.5b is a connected collection of long paths with a negligible clustering coefficient (0.006) and large ($\mathcal{D} = 32$) diameter.

9.3.6 SCALABILITY

Last, we corroborate the scalability of NETLSD. Figure 9.6 shows the time to compute 300 eigenvalues for the approximation of $h(G)$ on graphs of increasing size from the Reddit-X dataset. Our method computes the similarity on graphs of one million (10^6) nodes in only 16 minutes, while previous methods could not complete the process within one day. This result illustrates the fitness of NETLSD as a scalable comparison method among real-size graphs.

9.4 SUMMARY

We proposed `NETLSD`, a representation for graph comparison relying neither on graph alignment operations, nor on computationally-demanding kernel computations. `NETLSD` is a multi-scale heat trace signature of a graph Laplacian spectrum, which lower-bounds the Gromov–Wasserstein distance as it incorporates heat traces covering all scales. We derived a novel approximation of heat traces, rendering `NETLSD` efficiently computable, and a normalization scheme, rendering it size-invariant. To our knowledge, this is the first graph representation that achieves these properties and allows for comparisons at multiple scales. Our experiments show that `NETLSD` outperforms `NetSimile` and `FGSD`, two state-of-the-art representation-based methods for graph comparison, on a variety of graph collections.

LEARNING representations allows us to tailor them to the task at hand. Unfortunately, most real-world graph collections are small or do not contain labels, making task-specific learning unattainable. Thus, the question arises: can a graph representation, tailored for a particular purpose, be learned independently of the particular data set? We answer this question positively: we develop `SGR`, the first, to our knowledge, method that learns graph representations via pre-training on *surrogate tasks* using *synthetic data*. Grounded in spectral graph analysis, `SGR` seamlessly learns both global and local graph properties. In extensive experiments, we show how our approach works on unprecedentedly large graph collections, facilitates representation learning across various analytical tasks and application domains, and performs competitively without retraining.

A multitude of application domains from natural sciences to sociology takes different perspectives on graph collections. For example, on a *small scale*, molecules are modeled by atoms and their atomic bonds as nodes and edges in large graph collections. While on a *larger scale*, social network collections are analyzed by their community structures within the networks. Analytical tasks run on such collections to classify, for instance, which drugs can be used for the treatment of a disease or how molecules cluster together in functional groups. To fully discern a graph's properties, representation for such analysis requires a *multi-scale* view of a graph *adapted* to the task at hand. Representations have incorporated properties ranging from *local* (e.g., atomic bonds) to *global* (e.g., community structures). Nevertheless, works to date are agnostic as to which features are most suitable for a particular task.

Supervised neural approaches for graph representation [AT16, NAK16, BBL⁺17] attain excellent performance, and can be seen as the “full” tuning of the representation. However, such neural methods are applicable to particular datasets only, as they require labels to be available; besides, they fail to scale to graphs of a few thousands of nodes requiring significant computational resources to be trained.

This chapter introduces spectral graph representations (SGR), a method for learning graph representations that is at the same time efficient and customizable to multiple scales, analytical tasks, and datasets. We leverage graphs' Laplacian spectrum to *learn* a mapping from a collection of graphs to their vector representation by training a single-layer neural network on global structure recognition and another one on local structures. The global-structure network learns to distinguish synthetic graphs with community structure (i.e., sampled from a stochastic block model [KN11]) from random graphs by the Erdős–Rényi model. Similarly, the local-structure perceptron is trained to predict the *local* clustering coefficient [AdMo6] of a node. Moreover, we combine local and global representations to provide even more expressive representations. SGR representation is self-learning in the sense that it requires no real training data. We conduct an experimental study with several real datasets, using the ensuing graph signature representations on tasks such as graph classification by logistic regression. The results attest to the superiority of our approach on both classification and clustering tasks with real data.

We summarize the contributions of this chapter as follows:

- We propose SGR, the *first* method to *pretrain* representations that works for sparse, variable-size, large graphs.
- We build a model that incorporates arbitrary surrogate tasks, and show its ability to learn *local* and *global* information.
- We run extensive experiments on several real datasets of variable size and obtain performance comparable to or better than computationally demanding state-of-the-art methods.

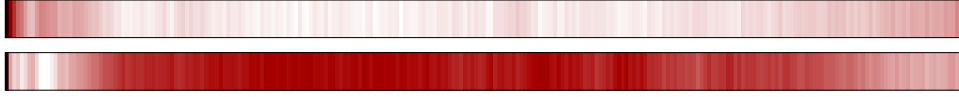


Figure 10.1: Different regions of the spectrum have a different impact on the classifier co-trained with the SGR^G . The color map shows the gradient magnitude of the classifier output with respect to the input spectrum visualized in increasing order from left to right, averaged on 600 graphs. Top: SBM; bottom: Erdős-Rényi.

10.1 LEARNED SPECTRAL REPRESENTATIONS

We conceive of the heat traces as a nonlinear transformation of the graph spectrum of the form $\sum_k f_t(\lambda_k)$ with $f_t(\lambda) = e^{-t\lambda}$. Sampling the time parameter on some grid $\{t_1, \dots, t_d\}$ yields the following d -dimensional representation:

$$\sigma = \left(\sum_k f_{t_1}(\lambda_k), \dots, \sum_k f_{t_d}(\lambda_k) \right).$$

We propose to *extend this view* to a more general parametric family of spectrum transformations. Given a graph G with n vertices, we first compute its spectrum or a part thereof $\{\lambda_k\}$, and interpolate it producing $\lambda(x)$ on the interval $[0, 1]$ such that $\lambda(k/n) = \lambda_k$. The spectrum is then sampled on a fixed grid (x_1, \dots, x_k) with k points, producing an k -dimensional vector $\tilde{\lambda}$ with the entries $\tilde{\lambda}_k = \lambda(x_k)$. Note that $\tilde{\lambda}$ is insensitive to a graph's size and invariant to the ordering of its vertices.

The interpolated and sampled spectrum $\tilde{\lambda}$ undergoes next a parametric non-linear transformation implemented as a single-layered perceptron,

$$\sigma = \psi(\mathbf{W}\tilde{\lambda} + \mathbf{b}),$$

where \mathbf{W} is an $d \times M$ weight matrix, \mathbf{b} is a d -dimensional bias vector, and ψ is an element-wise `SELU` non-linearity [KUMH17]. The resulting d -dimensional spectral graph representation (SGR) is parameterized by $\theta = (\mathbf{W}, \mathbf{b})$.

We propose two distinct regimes to train this representation. To obtain a representation capturing predominantly the *global* structure of the graph (manifested in the lower part of the spectrum), we co-train σ jointly with a binary classifier attempting to distinguish between Erdős-Rényi random graphs and stochastic block model [KN11] graphs of various degrees and sizes, which have very different community structures. The binary classifier is embodied as a single linear layer on top of the output of σ followed by softmax, and is trained using the regular cross-entropy loss.

On the other hand, to train a representation sensitive to *local* features (manifested predominantly in the higher part of the spectrum), we co-train σ

jointly with a regressor attempting to estimate the p -ring clustering coefficient of a graph [AdMo6], sensitive to small-scale edge variations.

We define local p -ring clustering coefficient of a node i in a graph G as

$$C_i^p = \frac{|\{(j, k) : j, k \in \mathcal{N}(i), d_{G'_i}(j, k) = p\}|}{\deg(i)(\deg(i) - 1)},$$

where $\mathcal{N}(i)$ denotes the 1-ring of the vertex i and $\deg(i) = |\mathcal{N}(i)|$ its cardinality, $G'_i \sqsubset G$ is the subgraph of G with the vertex i removed, and $d_{G'_i}(j, k)$ stands for the graph distance between vertices j and k in G' .

The local clustering coefficient is then averaged over all vertices of degree d , and then further averaged, with proper normalization, over all degrees, yielding the global p -ring clustering coefficient

$$C^p(G) = \frac{1}{\Delta(G)} \sum_{d=1}^{\Delta(G)} \frac{1}{|\{i : \deg(i) = d\}|} \sum_{i: \deg(i)=d} C_i^p,$$

where $\Delta(G) = \max_i \deg(i)$ denotes the maximum degree of G . The computation of $C^p(G)$ takes $\mathcal{O}(n * \langle \deg \rangle^{2+p})$ where $\langle \deg \rangle$ is the average degree of the graph.

We implement the regressor as a linear layer on top of σ and train it with the ℓ_2 loss on vectors of the form $(C^1(G), \dots, C^5(G))$, where the G 's are, as in the global case, Erdős-Rényi random graphs and stochastic block model (SBM) graphs of varying degrees and sizes.

In both cases, the classifier and the regressor are tossed away, leaving an appropriately trained graph representation. This approach is inspired by the versatility of image embeddings obtained from deep neural networks trained on visual recognition tasks. We henceforth denote the representation trained on the global task as SGR^G , while SGR^L denotes its local-task counterpart. Last, we also combine the loss of local and global tasks in order to learn SGR^{L+G} , a *joint local-global* representation. We emphasize that the performance in any surrogate task is inconsequential to our method; we use such tasks merely to train the representation.

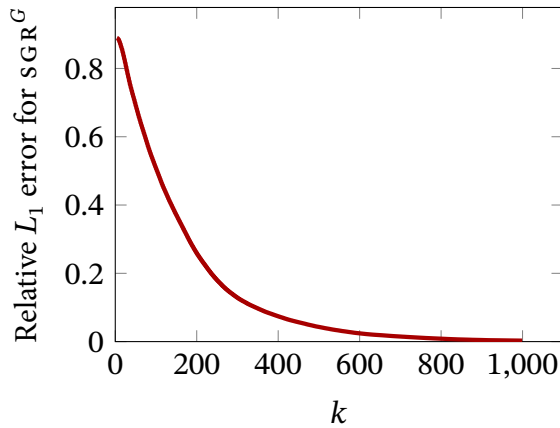


Figure 10.2: Relative L_1 error in neural representation as a function of a number of computed eigenvalues.

10.1.1 GRAPH GENERATION

As discussed, the surrogate tasks used in sGR^G require synthetic graphs that mimic those encountered in the real tasks. We generate graphs sampling from the Poisson degree distribution with 1–100 communities at different signal-to-noise ratios. For SBM with r communities, the signal-to-noise ratio SNR [DKMZ11, Abb18] is

$$\text{SNR} = \frac{(a - b)^2}{r(a + (r - 1)b)},$$

where a and b are intra- and inter-community probabilities, respectively. When $\text{SNR} > 1$ (Kesten–Stigum threshold), community detection (i.e., weak recovery) works effortlessly [Abb18]. Thus, we focus the training on hard, more informative instances with $\text{SNR} \approx 1$.

10.1.2 EFFICIENT COMPUTATION

Full eigendecomposition takes $\mathcal{O}(n^3)$ time and $\mathcal{O}(n^2)$ space. While for graphs with $\Delta(G) \ll n$ the sparse structure of the Laplacian allows to reduce the complexity to $\mathcal{O}(n^2)$, it is still prohibitive for large graphs. Instead, we compute $k \ll n$ top and bottom eigenvalues, and use interpolation in between. This reduces complexity to $\mathcal{O}(n^2k)$ in the general case and to $\mathcal{O}(nk)$ in the case of bounded degree graphs. Figure 10.2 illustrates the stability of sGR^G in terms of the number of eigenvalues.

Table 10.1: Dataset characteristics: number of graphs $|\mathcal{G}|$, number of labels $|L|$, avg. number of vertices $\overline{|V|}$, avg. number of edges $\overline{|E|}$, avg. global clustering coefficient \overline{CC} .

<i>dataset</i>	<i>Dataset statistics</i>		<i>Avg. graph statistics</i>		
	$ \mathcal{G} $	$ L $	$\overline{ V }$	$\overline{ E }$	\overline{CC}
D & D	1178	2	284.32	715.66	0.4574
Enzymes	600	6	32.63	62.14	0.4454
Mutag	188	2	17.93	19.79	0
Mutagenicity	4337	2	30.32	30.77	0
NCI1	4110	2	29.87	32.30	0
NCI109	4127	2	29.68	32.13	0
Proteins	1113	2	39.06	72.82	0.2895
PTC-FR	351	2	14.56	15.00	0
PTC-MR	344	2	14.29	14.69	0
Collab	5000	3	74.49	2457.50	0.9885
IMDB-B	1000	2	19.77	96.53	0.6661
IMDB-M	1500	3	13.00	65.94	0.9091
Reddit-B	2000	2	429.63	497.75	0.0038
Reddit-5k	4999	5	508.52	594.87	0.0010
Reddit-12k	11929	11	391.41	456.89	0.0031

10.2 EXPERIMENTS

We evaluate sgr on classification and clustering tasks on a variety of real graph collections. We compare against state-of-the-art kernels and graph representation methods, in terms of accuracy and running time.

Experimental setup. We ran experiments on a 20-core Intel Xeon CPU E5-2640v4, 3.20GHz machine with 256GB RAM. Unless otherwise stated, we repeat each experiment 100 times and report the average across all trials. We interpolate the spectrum of the normalized Laplacian of each graph in the collection through cubic-spline; then, we return 256 values uniformly sampled in the interpolated spectrum. This interpolated spectrum is the input of the single-layer perceptrons that learn the global and the local representations. As we discussed, the global-structure representation (sgr^G) is the single-layer perceptron trained to distinguish graphs with community structures sampled from stochastic block model from random Erdős–Rényi graphs, while the local-structure perceptron (sgr^L) is trained to recognize small-scale variations on edges by predicting the value of the local clustering coefficient [AdMo6].

dataset	Kernels		Fixed Representations				Self-learned Representations		
	wL	MLG	Λ	NetSimile	FGSD	NETLSD	SGR ^L	SGR ^G	SGR ^{L+G}
D&D	68.27	>1D	64.54	70.02	64.88	75.93	76.22	76.12	76.42
Enzymes	25.11	31.40	25.28	28.06	28.85	25.76	30.02	33.67	35.67
Mutag	81.16	86.54	82.07	83.66	85.23	87.46	86.78	86.97	85.71
Mutagenicity	74.54	76.88	62.51	69.37	74.59	61.39	68.75	69.39	69.21
NCI1	72.26	77.01	66.82	62.07	72.87	69.26	69.44	69.71	69.62
NCI109	72.01	76.07	64.30	61.54	71.90	67.83	67.88	67.82	68.13
Proteins	72.33	73.10	72.18	70.59	63.27	75.11	72.18	73.83	73.20
PTC-FR	65.14	65.56	64.90	66.02	61.42	65.62	62.02	63.36	61.41
PTC-MR	56.79	59.85	57.02	56.85	54.29	59.35	57.67	56.17	56.71
Collab	78.52	>1D	66.15	74.26	70.66	68.83	72.90	71.98	72.58
IMDB-B	72.26	59.18	63.16	70.96	69.20	68.23	68.18	70.38	69.36
IMDB-M	50.75	34.31	41.14	46.80	48.88	45.84	47.02	47.97	47.74
Reddit-B	71.97	>1D	76.25	86.84	87.12	78.84	87.18	87.45	87.76
Reddit-5k	48.57	>1D	48.02	44.96	48.51	53.99	53.38	53.22	53.41
Reddit-12k	36.11	>1D	34.89	55.12	33.23	35.27	40.34	41.24	40.63

We compare sGR against popular graph kernel methods: the Weisfeiler–Lehman kernel (wL) and the state-of-the-art Multiscale Laplacian Graph kernel (MLG) [KP16], using default parameters: height $h = 4$ for wL kernel and $\gamma = 0.01, \eta = 0.1, \text{radius} = 1, \text{levels} = 2$ for MLG kernel. We also compare sGR against NetSimile [BKERF13], FGSD [VZ17], and NETLSD [TMK⁺18b] graph representations. NetSimile constructs features by aggregating statistics on nodes and edges. For FGSD, which computes histograms on the bi-harmonic kernel of the graph, we use the bin width 0.0001, as described in [VZ17]. For NETLSD, we evaluate all proposed kernel variants using cross-validation and pick the best performing one. We additionally report the results of a naïve baseline spectral representation (Λ) that represents the graphs with a 256-dimensional vector sampled uniformly from a cubic spline-interpolated [Die95] spectrum of the normalized Laplacian, which corresponds to sGR without learning.

Datasets. We use 15 graph collections from the standard benchmark for Graph Kernels [KKM⁺16]. Such collections describe either social interactions (e.g., Reddit-B from messages in the Reddit platform) or biological connections (e.g., protein-protein interactions in Proteins). Table 10.1 shows the main data characteristics. The number of graphs in each collection varies from 188

Table 10.2: Graph classification accuracy on biochemical (top) and social (bottom) graph collections. Best results with representations are highlighted.

dataset	Fixed representations				Self-learned representations	
	NetSimile	FGSD	Λ	NETLSD	SGR^L	SGR^G
D&D	4.72	13.56	6.50	15.18	9.14	6.06
Enzymes	8.61	9.43	10.68	13.72	10.16	9.59
Mutag	40.65	26.56	24.34	34.13	28.61	21.05
Mutagenicity	7.82	3.54	2.99	3.57	3.79	2.54
NCI1	3.34	3.99	4.69	6.02	4.69	4.89
NCI109	3.29	4.04	4.52	5.55	5.03	3.82
Proteins	5.51	6.87	11.53	13.74	12.23	8.68
PTC-FR	4.72	4.46	3.73	3.90	4.58	4.14
PTC-MR	4.64	5.50	4.96	5.27	5.56	4.93
Collab	22.71	18.62	19.28	18.46	20.08	18.76
IMDB-B	13.20	10.81	6.69	9.14	7.58	5.95
IMDB-M	7.20	6.92	4.92	5.70	4.94	4.69
Reddit-B	15.15	7.76	9.82	18.10	11.05	8.13
Reddit-5k	16.02	4.29	10.47	17.14	9.51	9.85
Reddit-12k	15.63	5.24	11.40	15.30	11.16	13.28

Table 10.3: Normalized mutual information of clustering on representations. Higher numbers indicate better performance. Best results on each dataset are highlighted in green.

(Mutag) to 12k (Reddit-M-12k), while the average graph size varies from 13 (IMDB-B) to 508 (Reddit-5k). Networks in the standard benchmark datasets have very skewed clustering coefficient distribution (\overline{CC}). Last, note that SGR ignores any edge and node labels.

10.2.1 CLASSIFICATION

In our classification experiment, on each of the datasets we randomly select 80% of the data for training, and 20% for testing. We train an SVM using LibSVM [CL11] with default parameter $C=1$ and graph kernels. For all graph representations, including SGR^L and SGR^G , we train a logistic regression classifier with default $C=1$ and L_2 regularization, using LIBLINEAR solver [FCH⁺08]. Table 10.2 reports the classification accuracy averaged over 100 runs.

Our method attains good quality in almost all datasets except for NCI1, NCI109, and Proteins, for which FGSD stands out. Due to the small average graph size and density of these datasets, the task becomes harder for our approach that relies on local and global graph structures. At the same time, while state-of-the-art kernels (MLG) outperform SGR on some datasets, they fail to deliver results on medium and large collections in less than one day.

<i>dataset</i>	<i>Kernels</i>			<i>Fixed Representations</i>			<i>Self-learned Repr.</i>	
	SP	WL	MLG	NetSimile	FGSD	Λ	SGR^L	SGR^G
Training	—	—	—	—	—	—	2h	8h
Representation	—	—	—	1.2h	7h	4m	4m + 1s	4m + 1s
Similarity	>1D	35m	>1D	0.3s	8m	1.5s	1.5s	1.5s

10.2.2 CLUSTERING

Here, we evaluate the performance of scalable methods, i.e., fixed representations and SGR , on the task of clustering graphs by class. We run k -means clustering on the representations and measure the Normalized Mutual Information (NMI), reported in Table 10.3. Overall, SGR outperforms previous graph representations on biological datasets. Further, when it comes to social graphs, SGR matches the performance of techniques based on hand-crafted features (NetSimile), which are tailored to such datasets.

10.2.3 SCALABILITY AND TIME

Last, we report, in Table 10.4, the time for the neural network training (where applicable), computing representations (where applicable) and computing the square similarity matrices among 5000 graphs on the REDDIT-M-5k dataset, for all competing methods. Notably, SGR is about 20 times faster than the fastest competing method (NetSimile).

10.3 SUMMARY

We introduced SGR , a lightweight and concise graph representation that is self-learned via a single-layer perceptron over a collection of synthetically generated graphs. In particular, SGR learns sets of parameters encoding global and local graph properties as nonlinear transformations of the graphs’ Laplacian spectra; thus, it can adapt to a multitude of analytical tasks and application domains. Through extensive experimentation, we established that SGR achieves accuracy matching (or negligibly below) that of the most computationally demanding kernel methods on graph classification and clustering, while vastly surpassing them in speed.

Table 10.4: Runtime for the creation of a (dis-)similarity matrix across all graphs from the Reddit-5k dataset evaluated on a single core. Representation computation time for SGR is indicated in two components, spectrum computation + neural network inference. On the datasets, last two rows combined yield total time taken.

SPECTRAL analysis provides quintessential tools for studying the graphs’ multi-scale structure and is a well-suited foundation for reasoning about differences between graphs. In practice, however, spectral methods’ applicability is often limited by the scalability of eigendecomposition itself: it takes cubic time to compute graphs’ eigenvalues and eigenvectors. Several graph comparison methods such as von Neumann graph entropy (VNGE) [BGS06, CWLR19] and NETLSD, introduced in Chapter 9, require only information derived from a graph’s eigenvalues—not the full decomposition. Although they depend on less information, such metrics’ naïve computation is just as expensive as a full eigendecomposition. In order to scale to large graphs, these methods resort to low-order (two terms) Taylor expansion having loose bounds and poor empirical approximation performance. There has been no discussion of approximation accuracy or experiments on how it affects performance on downstream tasks.

In this chapter, we propose SLAQ, an efficient and effective approximation technique for computing spectral distances between graphs with billions of nodes and edges. By leveraging recent advances in numerical linear algebra [UCS17], we achieve state-of-the-art approximation accuracy in time linear in the number of graphs’ edges. We derive the corresponding error bounds and demonstrate that accurate computation is possible in time linear in the number of graphs’ edges. In a thorough experimental evaluation, we show that SLAQ outperforms existing methods, oftentimes by several orders of magnitude in approximation accuracy, and maintains comparable performance, allowing us to compare million-scale graphs in a matter of minutes on a single machine.

We summarize the contributions of this chapter as follows:

- We introduce `SLAQ`, an efficient approximation technique for two spectral graph distances, `VNGE` and `NETLSD`.
- We derive corresponding approximation error bounds and experimentally observe an average reduction in the approximation error of $30\times$ – $200\times$ over a diverse set of real-world graphs.
- We demonstrate that faithful approximation is *necessary* for accurate graph comparison and current approximation techniques are unfit for accurate yet fast approximation.
- We show that accurate computation of `VNGE` and `NETLSD` is possible for a graph with billions of nodes and edges on a single machine *in less than an hour*.

11.1 PRELIMINARIES

We review the definition of `VNGE` and two techniques for approximating spectral representations introduced in [CWL19] and Chapter 9. We note that although `NETLSD` and `VNGE` operate on different Laplacian matrices, for convenience’s sake we will refer to the spectrum of both matrices as λ_i .

11.1.1 VON NEUMANN GRAPH ENTROPY

In the Standard Quantum Mechanics model, the state of a quantum mechanical system associated with the n -dimensional Hilbert space is identified with a $n \times n$ positive semidefinite, trace-one, Hermitian *density matrix*. Von Neumann entropy [VN32] is a quantitative measure of mixedness of this density matrix, and is defined as follows:

Definition 6. Von Neumann Entropy \mathcal{H} is defined as $\mathcal{H} = -\sum_i \lambda_i \ln \lambda_i$. It is completely determined by the spectrum.

By convention, $0 \log 0 = 0$. Braunstein et al. [BGSo6] reinterprets the graph Laplacian matrix \mathbf{L} as a quantum mechanical system and introduces von Neumann graph entropy (`VNGE`) by scaling the graph Laplacian \mathbf{L} by its trace to get the density matrix $\mathbf{P} = \frac{1}{\text{tr}(\mathbf{L})}\mathbf{L}$. Scaling the Laplacian does not affect

the shape of its spectrum, as each eigenvalue is simply multiplied by $1/\text{tr}(\mathbf{L})$. This measure is related to the centralization of graphs [SCD18], however, its general structural interpretation is unknown mainly due to the lack of accurate scalable approximation [HEHW12, MRT18].

11.1.2 APPROXIMATION METHODS

Both `VNGE` and `NETLSD` can be represented as a function $\text{tr} f(\mathbf{L})$ of the Laplacian eigenvalues. A naïve approach would be to compute the exact eigenvalues and compute that function as $\sum_i f(\lambda_i)$, however, as we mentioned before, the computational complexity of full eigendecomposition is $\mathcal{O}(n^3)$, which is infeasible for large n .

Below we review approximation techniques that have been proposed in the literature [HEHW12, CWLR19, TMK⁺18b]. We empirically evaluate their approximation performance in Section 11.3.1.

11.1.3 TAYLOR EXPANSION

A natural impulse for dealing with complex matrix functions is to approximate the function with the first few terms of its Taylor expansion. Even though it is known that Taylor expansion provides an unreliable approximation of matrix functions [MVL03], both `NETLSD` and `VNGE` make use of this approximation, as the first two Taylor terms can be computed in $\mathcal{O}(m)$.

The expansion of `NETLSD` depends on the parameter t , and its approximation is reasonable for only small values of t :

$$h_t = \sum_i e^{-t\lambda_i} = \sum_{k=0}^{\infty} \frac{\text{tr}((-t\mathcal{L})^k)}{k!} \approx n - t \text{tr}(\mathcal{L}) + \frac{t^2}{2} \text{tr} \mathcal{L}^2. \quad (11.1)$$

The expansion used for `VNGE` is slightly different [CWLR19, MRT18]:

$$\mathcal{H} = \sum_i \lambda_i \ln \lambda_i \approx 1 - \frac{1}{\text{tr}(\mathbf{L})^2} (\text{tr}(\mathbf{L}) + 2 \text{tr}(\mathbf{L}^2)). \quad (11.2)$$

These first two terms are easily computed, even for very large graphs, as $\text{tr}(\mathcal{L}) = n$ and $\text{tr}(\mathcal{L}^2) = \sum_{ij} \mathcal{L}_{ij}^2$ since \mathcal{L} is self-adjoint, and the error rate of the Taylor expansion of the matrix exponential depends on the largest eigenvalue of the matrix [MVL03].

Chen et al. [CWL_R19] introduce two approximation algorithms for `VNGE` based on a two-term Taylor expansion, `FINGER- $\overline{\mathcal{H}}$` and `FINGER- $\widehat{\mathcal{H}}$` :

$$Q = 1 - \frac{1}{\text{tr}(\mathbf{L})^2}(\text{tr}(\mathbf{D})^2 + 2 \text{tr}(\mathbf{L}^2))$$

$$\text{FINGER-}\overline{\mathcal{H}} = -Q \ln\left(\frac{2 \max \mathbf{D}}{\text{tr}(\mathbf{L})^2}\right), \quad \text{FINGER-}\widehat{\mathcal{H}} = -Q \ln(\lambda_{\max})$$

11.1.4 SPECTRAL INTERPOLATION

We conclude by noting that the Taylor expansion is useful on very large graphs, on which computing any part of the spectrum is prohibitive. For manageable graph sizes, `NETLSD` adopts a more accurate strategy based on approximating the eigenvalue growth rate, adapted from [VBCG10]. It takes $\mathcal{O}(km + k^2n)$ to compute k extremal eigenvalues of a graph [GM09], thus it is possible to compute k eigenvalues on both ends of the spectrum, and interpolate a *linear growth* of the interior eigenvalues.

It is easy to see that the worst-case scenario is the graph with exactly k isolated nodes and a fully connected component having $n - k$ nodes, meaning $\lambda_{:k} = 0$ and $\lambda_{k:} = 2$. Then, absolute error in the approximation of h_t becomes $\left\|n - 2k - \sum_{i=0}^{n-2k} \frac{2^{i-k}}{n-2k}\right\|$. This bound is very loose; we further verify that the approximation accuracy of the linear interpolation strategy is poor in the Section 11.3.1 and that it does not scale to very large graphs in the Section 11.3.5.

11.2 STOCHASTIC LANCZOS QUADRATURE

As noted above, the main approximation techniques that have been proposed for `VNGE` and `NETLSD` have limited guarantees on their approximation quality, and these weak guarantees have not been fully explored in the literature. In this section, we address these deficiencies and propose our method for improved approximation of spectral distances between graphs.

11.2.1 TRACE FUNCTION ESTIMATION

Setting aside computational infeasibility of the naïve eigenvalues calculation, loose Taylor expansion error bounds and linear interpolation heuristics, we attain theoretically guaranteed accuracy and speed by means of stochastic Lanczos quadrature (SLQ) [UCS17].

In trace estimation problems for large and implicit matrices, the standard choice is a Hutchinson estimator [Hut89], which we apply (we denote both Laplacians as \mathbf{L} in this section for brevity) in our setting for the trace of matrix exponential $f(\mathbf{L}) = \exp(-t\mathbf{L})$ or the matrix logarithm $f(\mathbf{L}) = -\mathbf{L} \log \mathbf{L}$:

$$\mathrm{tr}(f(\mathbf{L})) = \mathbb{E}_{p(\mathbf{v})}(\mathbf{v}^\top f(\mathbf{L})\mathbf{v}) \approx \frac{n}{n_v} \sum_{i=1}^{n_v} \mathbf{v}_i^\top f(\mathbf{L})\mathbf{v}_i, \quad (11.3)$$

where \mathbf{v}_i are n_v random vectors drawn from a distribution $p(\mathbf{v})$ with zero mean and unit variance. Practical choices for $p(\mathbf{v})$ include Rademacher or standard normal distributions, with the difference being in the variance or number of random vectors [AT11].

To approximate the bilinear form $\mathbf{v}_i^\top f(\mathbf{L})\mathbf{v}_i$ in Eq. (11.3) with a symmetric real-valued matrix \mathbf{L} , we apply the Lanczos Quadrature [GM09], which uses the Lanczos algorithm to provide orthonormal polynomials for the Gauss quadrature. In other words, we first take an eigendecomposition $\mathbf{L} = \Phi \Lambda \Phi^\top$, then cast the outcome to a Riemann–Stieltjes integral and finally apply the m -point Gauss quadrature rule:

$$\begin{aligned} \mathbf{v}_i^\top f(\mathbf{L})\mathbf{v}_i &= \mathbf{v}_i^\top \Phi f(\Lambda) \Phi^\top \mathbf{v}_i = \sum_{j=1}^n f(\lambda_j) \mu_j^2 \\ &= \int_a^b f(t) d\mu(t) \approx \sum_{k=1}^m \omega_k f(\theta_k), \end{aligned}$$

where $\mu_j = [\Phi^\top \mathbf{v}_i]_j$ and $\mu(t)$ is a piecewise constant measure function:

$$\mu(t) = \begin{cases} 0, & \text{if } t < a = \lambda_n \\ \sum_{j=1}^i \mu_j^2, & \text{if } \lambda_i \leq t < \lambda_{i-1} \\ \sum_{j=1}^n \mu_j^2, & \text{if } b = \lambda_1 \leq t \end{cases}$$

and θ_k, ω_k are the nodes and weights of the quadrature. We obtain the pairs of ω_k, θ_k with the s -step Lanczos algorithm [GM09], which we describe below succinctly.

The s -step Lanczos algorithm computes an orthonormal basis for the Krylov subspace \mathcal{K} spanning vectors $\{\mathbf{q}_0, \mathbf{L}\mathbf{q}_0, \dots, \mathbf{L}^{s-1}\mathbf{q}_0\}$, with the symmetric

Algorithm 6 SLAQ algorithm

```

1: function SLAQ_LSD( $G, s, n_v$ )
2:    $\mathbf{L} \leftarrow \text{Laplacian}(G)$ 
3:   descriptor  $\leftarrow \text{slq}(\mathbf{L}, s, n_v, \exp(x))$ 
4:   return descriptor
5: function SLAQ_VNGE( $G, s, n_v$ )
6:    $\mathbf{P} \leftarrow \text{DensityMatrix}(G)$ 
7:   descriptor  $\leftarrow \text{slq}(\mathbf{P}, s, n_v, x \ln(x))$ 
8:   return descriptor
9: function slq( $\mathbf{L}, s, n_v, \text{fun}$ )
10:   $\mathbf{T} = \text{lanczos}(\mathbf{L}, s, n_v)$   $\triangleright \mathbf{T} \in \mathbb{R}^{n_v \times m \times m}$ 
11:   $\mathbf{\Lambda}, \mathbf{U} \leftarrow \text{eigh}(\mathbf{T})$ 
12:  return  $\frac{1}{n_v} \sum_i^{n_v} \left( \sum_k^s (\text{fun}(\lambda_k^i) [\mathbf{u}_{k,0}^i]^2) \right)$ 

```

matrix \mathbf{L} and an arbitrary *starting unit-vector* \mathbf{q}_0 . The output of the algorithm is an $n \times s$ matrix $\mathbf{Q} = [\mathbf{q}_0, \mathbf{q}_1, \dots, \mathbf{q}_{s-1}]$ with orthonormal columns and an $s \times s$ tridiagonal symmetric matrix \mathbf{T} , such that $\mathbf{Q}^\top \mathbf{L} \mathbf{Q} = \mathbf{T}$, notice that due to this relation each \mathbf{q}_i vector is given as a polynomial in \mathbf{L} applied to the initial vector \mathbf{q}_0 : $\mathbf{q}_i = p_i(\mathbf{L})\mathbf{q}_0$. Since \mathbf{T} is a tridiagonal matrix, the three-term recurrence relation exists between the consequent polynomials p_i . We can now use the Gauss rule with points equal to the eigenvalues of \mathbf{T} , λ_k , and weights set to the squared first components of its normalized eigenvectors, τ_k^2 , respectively (see [GW69, Wil62, GM09]). Now, setting $\mathbf{q}_0 = \mathbf{v}_i$, the estimate for the quadratic form becomes:

$$\mathbf{v}_i^\top f(\mathbf{L})\mathbf{v}_i \approx \sum_{k=0}^{s-1} \tau_k^2 f(\lambda_k), \quad (11.4)$$

$$\tau_k = \mathbf{U}_{0,k} = \mathbf{e}_1^\top \mathbf{u}_k, \quad \lambda_k = \mathbf{\Lambda}_{k,k} \quad \mathbf{T} = \mathbf{U} \mathbf{\Lambda} \mathbf{U}^\top \quad (11.5)$$

Applying (11.4) over n_v random vectors in the Hutchinson trace estimator (11.3) yields the SLQ estimate:

$$\text{tr}(f(\mathbf{L})) \approx \frac{n}{n_v} \sum_{i=0}^{n_v-1} \left(\sum_{k=0}^{s-1} (\tau_k^i)^2 f(\lambda_k^i) \right) = \Gamma. \quad (11.6)$$

We derive error bounds for the estimator based on the Lanczos approximation of the matrix exponential, and show that even a few Lanczos steps, i.e., $s = 10$, are sufficient for an accurate approximation of the quadratic form. However, the trace estimation error is theoretically dominated by the error of the Hutchinson estimator, e.g. for Gaussian $p(\mathbf{v})$ the bound on the number of samples to guarantee that the probability of the relative error exceeding ϵ is at most δ is $8\epsilon^{-2} \ln(2/\delta)$ [RKA15]. Although, in practice, we observe performance much better than the bound suggests. Hutchinson error implies nearing accuracy roughly 10^{-2} with $n_v \geq 10k$ random vectors, however, with as much as $n_v = 100$ the error is already 10^{-3} . Thus, we use default values of $s = 10$ and $n_v = 100$ in all experiments in Section 11.3.

We summarize the overall SLAQ method in Algorithm 6.

11.2.2 TRACE ESTIMATION ERROR BOUNDS

We first rewrite quadratic form under summation in the trace approximation to a convenient form,

$$\begin{aligned} \mathbf{v}^\top f(\mathbf{L})\mathbf{v} &\approx \sum_{k=0}^m \tau_k^2 f(\lambda_k) = \sum_{k=0}^m [\mathbf{e}_1^\top \mathbf{u}_k]^2 f(\lambda_k) = \\ &\mathbf{e}_1^\top \mathbf{U} f(\Lambda) \mathbf{U}^\top \mathbf{e}_1 = \mathbf{e}_1^\top f(\mathbf{T}) \mathbf{e}_1. \end{aligned} \quad (11.7)$$

Since the Krylov subspace $\mathcal{K}_s(\mathbf{L}, \mathbf{v})$ is built on top of vector \mathbf{v} with \mathbf{Q} as an orthogonal basis of $\mathcal{K}_s(\mathbf{L}, \mathbf{v})$, i.e. $\mathbf{q}_0 = \mathbf{v}$ and $\mathbf{v} \perp \mathbf{q}_i$ for $i \in (1, \dots, s-1)$, the following holds:

$$\mathbf{v}^\top f(\mathbf{L})\mathbf{v} \approx \mathbf{v}^\top \mathbf{Q} f(\mathbf{T}) \mathbf{e}_1 = \mathbf{e}_1^\top f(\mathbf{T}) \mathbf{e}_1. \quad (11.8)$$

Thus, the error in quadratic form estimate $\mathbf{v}^\top f(\mathbf{L})\mathbf{v}$ is exactly the error of Lanczos approximation $f(\mathbf{L})\mathbf{v} \approx \mathbf{Q} f(\mathbf{T}) \mathbf{e}_1$. To obtain the error bounds, we adapt Theorem 2 in [HL97], which we recite below.

Theorem 6. Let \mathbf{L} be a real symmetric positive semi-definite matrix with eigenvalues in the interval $[0, 4\rho]$. Then the error in the s -step Lanczos approximation of $\exp^{-t\mathbf{L}} \mathbf{v}$, i.e. $\epsilon_s = \|\exp^{-t\mathbf{L}} \mathbf{v} - \mathbf{Q}_s \exp^{-t\mathbf{T}_s} \mathbf{e}_1\|$, is bounded in the following ways:

$$\epsilon_s \leq \begin{cases} 10e^{-s^2/(5\rho t)}, & \text{if } \sqrt{4\rho t} \leq s \leq 2\rho t \\ 10(\rho t)^{-1}e^{-\rho t}\left(\frac{e\rho t}{s}\right)^s, & \text{if } s \geq 2\rho t \end{cases} \quad (11.9)$$

Since \mathbf{v} is a unit vector, thanks to the Cauchy–Bunyakovsky–Schwarz inequality, we can upper-bound the error of the quadratic form approximation by the error of the $\exp^{-t\mathbf{L}}\mathbf{v}$ approximation, i.e.,

$$|\mathbf{v}^\top f(\mathbf{L})\mathbf{v} - \mathbf{e}_1^\top \mathbf{U} f(\Lambda) \mathbf{U}^\top \mathbf{e}_1| \leq \|\exp^{-t\mathbf{L}}\mathbf{v} - \mathbf{Q}_s \exp^{-t\mathbf{T}_s}\mathbf{e}_1\| = \epsilon_s$$

Following the argumentation in [UCS17], we obtain a condition on the number of Lanczos steps s by setting $\epsilon_s \leq \frac{\epsilon}{2}f_{\min}(\lambda)$, where $f_{\min}(\lambda)$ is the minimum value of f on $[\lambda_{\min}, \lambda_{\max}]$. We now derive the absolute error between the Hutchinson estimate of Equation 11.3 and the sLQ of Equation 11.6:

$$\begin{aligned} \left| \text{tr}_{n_v}(f(\mathbf{L})) - \Gamma \right| &= \frac{n}{n_v} \left| \sum_{i=1}^{n_v} \mathbf{v}_i^\top f(\mathbf{L})\mathbf{v}_i - \sum_{i=1}^{n_v} \mathbf{e}_1^\top f(\mathbf{T}^{(i)})\mathbf{e}_1 \right| \\ &\leq \frac{n}{n_v} \sum_{i=1}^{n_v} \left| \mathbf{v}_i^\top f(\mathbf{L})\mathbf{v}_i - \mathbf{e}_1^\top f(\mathbf{T}^{(i)})\mathbf{e}_1 \right| \\ &\leq \frac{n}{n_v} \sum_{i=1}^{n_v} \epsilon_s = n\epsilon_s, \end{aligned}$$

where $\mathbf{T}^{(i)}$ is the tridiagonal matrix obtained with Lanczos algorithm with starting vector \mathbf{v}_i .

$$\left| \text{tr}_{n_v} f(\mathbf{L}) - \Gamma \right| \leq n\epsilon_s \leq \frac{n\epsilon}{2}f_{\min}(\lambda) \leq \frac{\epsilon}{2} \text{tr}(f(\mathbf{L})) \quad (11.10)$$

Finally, we formulate sLQ as an (ϵ, δ) estimator,

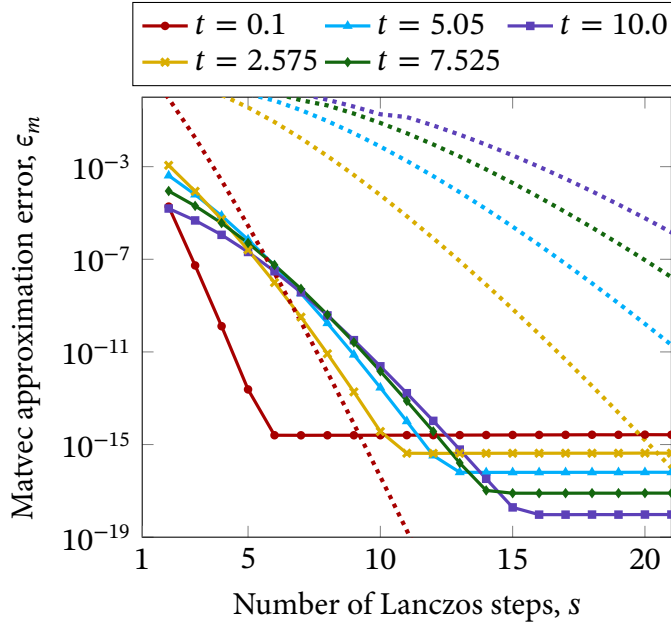


Figure 11.1: Errors (solid) and error bounds (dotted) for the approximation of matrix exponential action with varying temperature t .

$$\begin{aligned}
 1 - \delta &\leq \Pr \left[\left| \text{tr}(f(\mathbf{L})) - \text{tr}_{n_v}(f(\mathbf{L})) \right| \leq \frac{\epsilon}{2} \left| \text{tr}(f(\mathbf{L})) \right| \right] \\
 &\leq \Pr \left[\left| \text{tr}(f(\mathbf{L})) - \text{tr}_{n_v}(f(\mathbf{L})) \right| + \left| \text{tr}_{n_v}(f(\mathbf{L})) - \Gamma \right| \leq \frac{\epsilon}{2} \left| \text{tr}(f(\mathbf{L})) \right| + \frac{\epsilon}{2} \left| \text{tr}(f(\mathbf{L})) \right| \right] \\
 &\leq \Pr \left[\left| \text{tr}(f(\mathbf{L})) - \Gamma \right| \leq \epsilon \left| \text{tr}(f(\mathbf{L})) \right| \right],
 \end{aligned}$$

For the normalized Laplacian \mathbf{L} , the minimum eigenvalue is 0 and $f_{\min}(0) = \exp(0) = 1$, hence $\epsilon_s \leq \frac{\epsilon}{2}$, and the eigenvalue interval has $\rho = 0.5$. We can thus derive the appropriate number of Lanczos steps m to achieve error ϵ ,

$$\epsilon \leq \begin{cases} 20e^{-s^2/(2.5t)}, & \text{if } \sqrt{2t} \leq s \leq t \\ 40t^{-1}e^{-0.5t\left(\frac{0.5et}{s}\right)^s}, & \text{if } s \geq t \end{cases} \quad (11.11)$$

Figure 11.1 shows the tightness of the bound for the approximation of the matrix exponential action on vector \mathbf{v} , $\epsilon_s = \|\exp(-t\mathbf{L}) - \mathbf{Q}_s \exp(-t\mathbf{T}_s)\mathbf{e}_1\|$. We can see that for most of the temperatures t , very few Lanczos steps s

Figure 11.2: Trace estimation errors (solid) and error bounds (dotted) for: (left) the number of Lanczos steps m with fixed number of random vectors $n_v = 100$; (right) the number of random vectors n_v in Hutchinson estimator with fixed number of Lanczos steps $m = 10$. Lines correspond to varying temperature t .

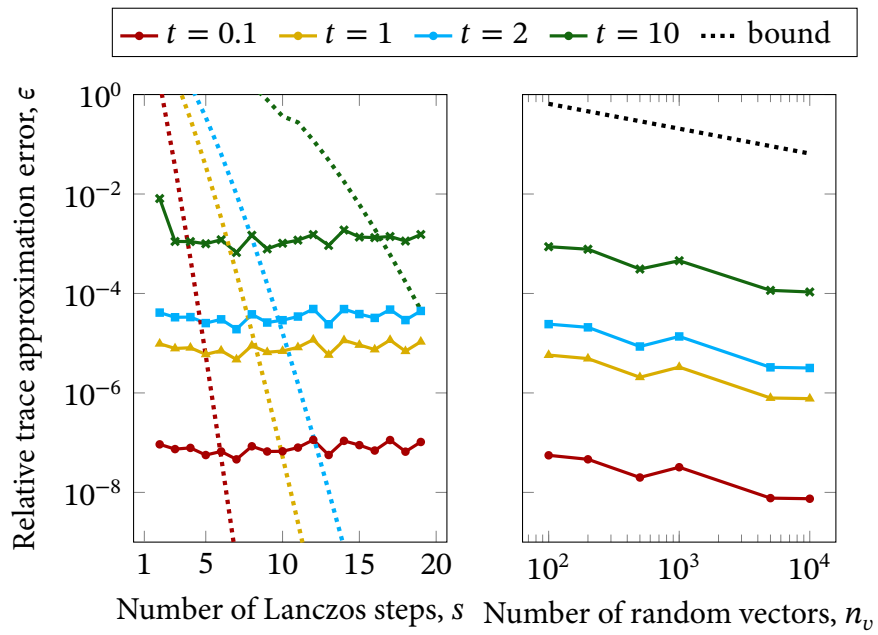
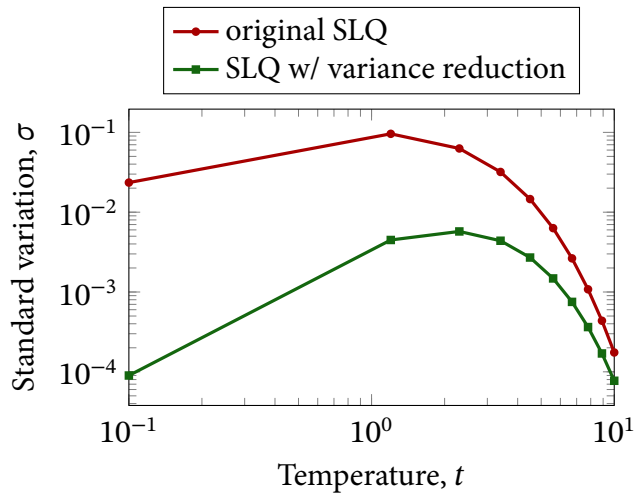


Figure 11.3: Variance of the trace estimate



are sufficient, i.e. we can set $s = 10$. However, the error from the Hutchinson estimator dominates the overall error. Figure 11.2 shows the error of trace estimation does not change with s and for $t = 0.1$ is around 10^{-3} . In case of a Rademacher $p(\mathbf{v})$, the bound on the number of random samples is $n_v \geq \frac{6}{\epsilon^2} \log(2/\delta)$ [RKA15]. Employing 10k vectors results in the error bound of roughly 10^{-2} . In practice, we observe the performance much better than given by the bound, see Figure 11.2. Analogous error estimates for the von Neumann entropy remain an open problem in numerical linear algebra.

One particular benefit of small s value is that we do not have to worry about the orthogonality loss in the Lanczos algorithm which often undermines its convergence [GVL12]. Since we do only a few Lanczos iterations, the rounding errors hardly accumulate causing little burden in terms of orthogonality loss between the basis vectors of the Krylov subspace.

11.2.3 VARIANCE REDUCTION

We reduce the variance of the randomized estimator through control variates. The idea is to use Taylor expansion to substitute a part of the trace estimate with its easily computed precise value,

$$\begin{aligned} \text{tr}(\exp(-t\mathbf{L})) &= \text{s1q}\left[\exp(-t\mathbf{L}) - \left(\mathbf{I} - t\mathbf{L} + \frac{t^2\mathbf{L}^2}{2}\right)\right] + \text{tr}\left(\mathbf{I} - t\mathbf{L} + \frac{t^2\mathbf{L}^2}{2}\right) \\ &= \text{s1q}\left[\exp(-t\mathbf{L}) - \left(\mathbf{I} - t\mathbf{L} + \frac{t^2\mathbf{L}^2}{2}\right)\right] + n + \text{tr}(-t\mathbf{L}) + \frac{t^2\|\mathbf{L}\|_F^2}{2} \\ &= \text{s1q}\left[\exp(-t\mathbf{L})\right] + \text{s1q}\left[t\mathbf{L}\right] - \text{s1q}\left[\frac{t^2\mathbf{L}^2}{2}\right] - tn + \frac{t^2\|\mathbf{L}\|_F^2}{2}, \end{aligned}$$

where we use the fact that $\|\mathbf{L}\|_F = \sqrt{\text{tr}(\mathbf{L}^T\mathbf{L})}$ and that the trace of normalized Laplacian is equal to n . It does reduce the variance of the trace estimate for smaller temperatures $t \leq 1$.

To obtain this advantage over the whole range of t , we utilize the following variance reduction form:

$$\text{tr}(\exp(-t\mathbf{L})) = \text{s1q}\left[\exp(-t\mathbf{L}) - (\mathbf{I} - \alpha t\mathbf{L})\right] + n(1 - \alpha t), \quad (11.12)$$

where there exists an alpha that is optimal for every t , namely setting $\alpha = 1/\exp(t)$. We can see the variance reduction that comes from this procedure in the Figure 11.3.

11.3 EXPERIMENTS

We evaluate SLAQ against all approximation methods proposed in [CWLR19] and Chapter 9, in addition to the exact computation of the spectrum (where allowed by the graph size). We perform our experiments on the Google Cloud’s c2-standard-60 virtual machine with 60 cores and 240GB RAM, averaging 10 times unless stated otherwise. We use LAPACK [ABB⁺99] for linear algebra operations. We open-source the implementation¹.

1. github.com/google-research/google-research/tree/master/graph_embedding/slaq

Parameter settings. Unless otherwise mentioned, we evaluate SLAQ using $n_v = 100$ starting vectors and $s = 10$ Lanczos iterations. We provide an additional experimental investigation into parameter settings of SLAQ in Section 11.3.4. For the linear approximation of [TMK⁺18b], we use the default ($k = 300$) eigenvalues from each end of the spectrum, following the notation of the original paper. Taylor series-based approximation techniques do not depend on any additional parameters.

Datasets. We use four types of graph collections to measure the efficiency and effectiveness of SLAQ. First, we consider the accuracy of the method compared to other approximation techniques on the two subsets of graphs: synthetically generated Erdős–Rényi graphs and 73 graphs from the Network Repository² [RA15] with a number of nodes from 2500 up to 25000. In total, we use 27 biological, 12 interaction, 10 technological networks, 5 small web graphs, and 19 uncategorized networks (mostly, optimization problem graphs).

2. networkrepository.com

We follow up with large graphs to test the ability of SLAQ to efficiently compute descriptors of Web-scale graphs. For that, we use five datasets³:

3. All but ClueWeb09 are from SNAP network collection, available at snap.stanford.edu

- DBLP [YL15] is a co-authorship network from a major online computer science bibliography.
- Orkut [YL15] was an online social network.
- LiveJournal [YL15] is an online blogging community where users can form friendships with each other.
- Friendster [YL15] was an online social network.
- ClueWeb09 [CCS09, RA15] is a web crawl from 2009.

<i>dataset</i>	Size		Statistics	
	$ V $	$ E $	Avg. deg.	Density
DBLP	317k	1.05M	6.62	2.08×10^{-5}
Orkut	3.07M	117.2M	76.28	2.48×10^{-5}
LiveJournal	4M	34.7M	17.35	4.34×10^{-6}
Friendster	65.6M	1.8B	55.06	8.39×10^{-7}
ClueWeb09	4.8B	7.81B	3.27	6.83×10^{-10}

Table 11.1: Characteristics of large graphs used in this work: number of vertices $|V|$, number of edges $|E|$; average node degree; density defined as $|E|/\binom{|V|}{2}$.

<i>dataset</i>	Size		Temporal statistics	
	$ V $	$ E $	$ T $	$ \mathcal{E} / T $
Wiki-nl	1M	20M	95	148337
Wiki-pl	1M	25M	95	182959
Wiki-it	1.2M	35M	95	250633
Wiki-de	2.1M	86M	95	553257

Table 11.2: Characteristics of dynamic graphs: total number of vertices $|V|$, total number of edges $|E|$; number of timestamps $|T|$; average incoming edges per timestamp $|\mathcal{E}|/|T|$.

<i>dataset</i>	$ G $	$ Y $	Vertices $ V $		
			Min.	Avg.	Max.
D & D	1178	2	30	284.32	5748
Collab	5000	3	32	74.49	492
Reddit-5k	4999	5	22	508.52	3648
Reddit-12k	22939	11	2	391.41	3782

Table 11.3: Properties of the graph classification datasets used: number of graphs $|G|$; number of labels $|Y|$; minimum, average, and maximum number of nodes in graph collection.

Next, we investigate benefits of using SLAQ on dynamic Wikipedia link datasets in 4 different languages: Dutch (nl), Polish (pl), Italian (it), German (de). We obtained datasets from [CCS09]⁴ and generated $|T|$ snapshots for every month in the original dataset.

Last, we verify that SLAQ’s improvements in approximation performance enhance downstream task performance. We use three social network datasets and one from the field of bioinformatics⁵:

- D & D [DD03, SSL⁺11, KKM⁺16] is a dataset of protein structures. Each protein is represented by a graph of amino acids that are connected by an edge if they are less than 6 Ångstroms apart. The prediction task is to classify the protein structures into enzymes and non-enzymes.

4. We used preprocessed version from KONECT repository, available at konect.uni-koblenz.de/networks/

5. We obtained them at the graph kernel benchmark collection, available at ls11-www.cs.tu-dortmund.de/staff/morris/graphkerneldatasets

Figure 11.4: SLAQ offers over 200x reduction in average error for VNGE over techniques proposed in [CWLR19] and over 30x improvement over the linear approximation from Chapter 9.

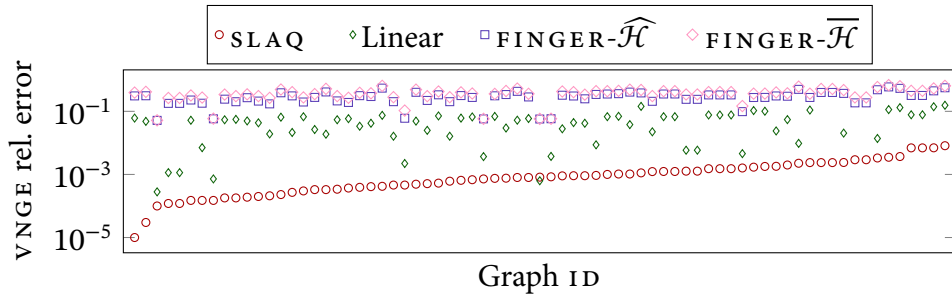


Figure 11.5: SLAQ offers 22x reduction in average error for NetLSD over linear approximation and 250x over Taylor expansion.

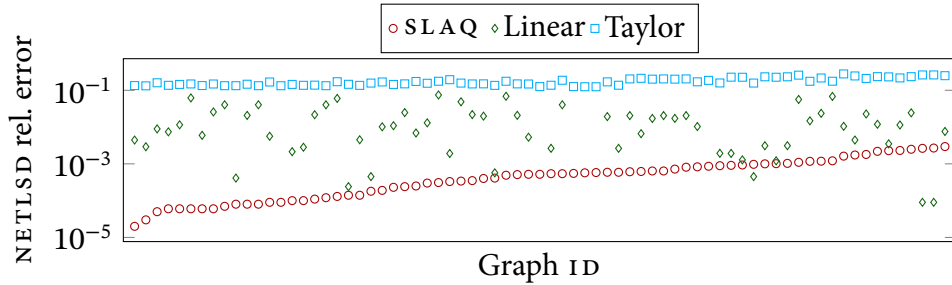
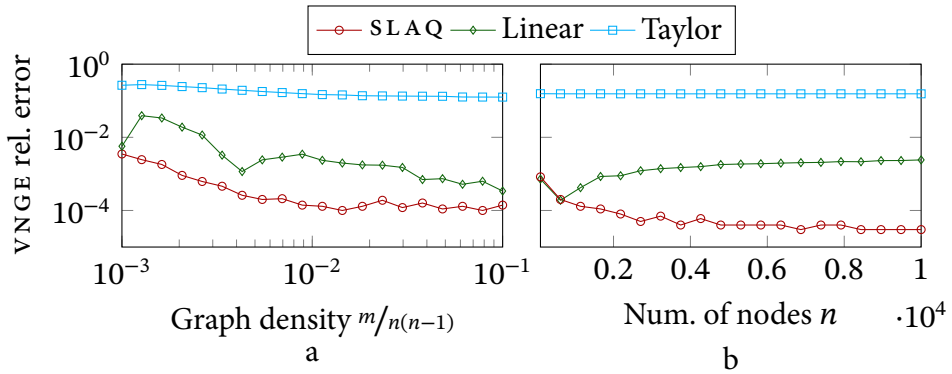


Figure 11.6: Number of nodes and edges of random Erdős-Rényi graphs does not affect SLAQ's approximation accuracy.



- Collab [YV15, LKF05, KKM⁺16] is a collection of collaboration ego-networks of different researchers derived in [YV15] from three datasets introduced in [LKF05]. The task is to determine whether the ego-collaboration graph of a researcher belongs to High Energy, Condensed Matter or Astrophysics field.
- Reddit-5k and Reddit-12k [YV15, KKM⁺16] are two datasets derived from Reddit, an online aggregation and discussion website. Discussions on Reddit are organized into different subcommunities; the task is to determine the community given the structure of the discussion graph.

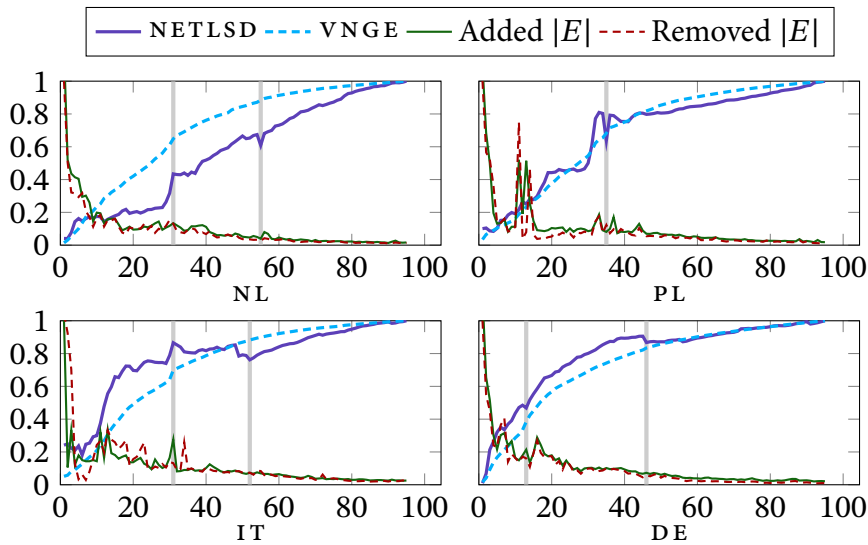


Figure 11.7: SLAQ approximation of NetLSD and VNGE for Wikipedia graphs across time. Changes that are not explained by local edge differences highlighted in gray.

11.3.1 APPROXIMATION ACCURACY

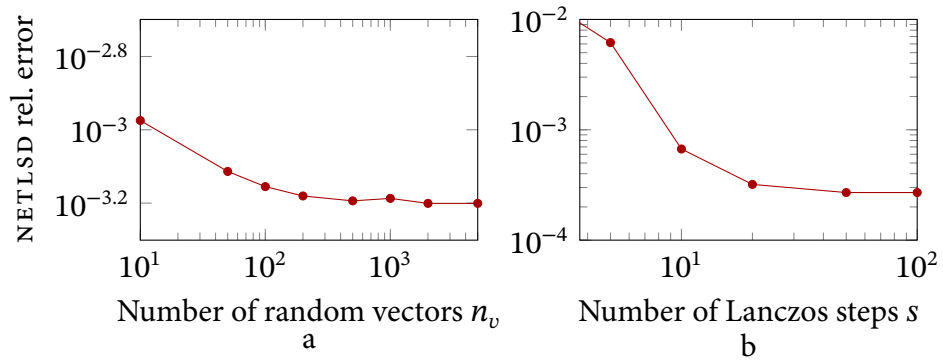
We proceed with evaluation of SLAQ capacity to approximate matrix functions for graph comparison. We compute full spectrum of 73 small graphs and true values of NETLSD and VNGE and report the relative l_2 approximation error with respect to the true graph descriptor. Figure 11.4 demonstrates that SLAQ offers over $200\times$ reduction in the average relative error for VNGE over FINGER techniques [CWL19] and over $30\times$ improvements over the linear interpolation technique from Chapter 9. Figure 11.5 shows that SLAQ offers a $250\times$ reduction over the Taylor expansion and over $22\times$ improvement over the linear interpolation [TMK⁺18b].

We also compare how the approximation accuracy changes for NETLSD on the random Erdős–Rényi graphs. We generate random graphs of size 1000 with varying graph density (number of expected edges) p and random graphs of size 100–10000 with the average degree $m/n = 10$. We report the results in Figure 11.6. We observe that SLAQ’s approximation accuracy is stable across the graph size both in terms of the number of the nodes and graph density.

11.3.2 BENEFITS OF NON-LOCAL APPROXIMATION

We verify that our method has a global view of the graph, i.e. is not dominated by only local information. In order to do that, we compute NETLSD and VNGE for monthly snapshots of dynamic Wikipedia link datasets from January 2003

Figure 11.8: Parameter sensitivity of SLaQ in terms of approximating NetLSD with (a) different number of starting vectors n_v and (b) different number of Lanczos steps s . Error averaged across 73 graphs from the Network Repository.



dataset	VNGE					NETLSD			
	FINGER- $\overline{\mathcal{H}}$	FINGER- $\widehat{\mathcal{H}}$	Linear	SLAQ	Exact	Taylor	Linear	SLAQ	Exact
D&D	63.01	66.38	68.13	65.53	66.40	67.01	67.98	66.77	67.24
Collab	64.95	65.10	55.90	49.04	58.03	61.81	65.17	58.76	63.48
Reddit-5k	30.87	29.85	31.31	31.77	31.43	33.67	32.01	35.48	35.63
Reddit-12k	16.53	16.20	17.18	17.04	16.79	22.67	21.30	25.31	25.52

Table 11.4: 1-Nearest neighbor graph classification performance on 4 datasets with VNGE and NetLSD. Exact computation results are in bold. Approximations that are close to or better than the exact metric computation are highlighted in green.

to December 2010 (a total of $|T| = 96$ snapshots) and report their change as well as the number of edges added/removed each month.

We plot the proportion of cumulative edge additions/deletions and distances between descriptor pairs of snapshots $(0, i)$, where $i \in 1, \dots, |T|$. Figure 11.7 reports the distance values for each language as well as the relative number of incoming and outgoing edges per snapshot. We mark examples of anomalous spikes in NETLSD and VNGE that can not be explained simply by the edge additions and deletions. In these cases, simple approximations like the 2-term Taylor expansion would fail to capture such changes.

Table 11.5: Running time (in seconds) of different approximation techniques and SLaQ for VNGE on large graphs.

dataset	FINGER- $\overline{\mathcal{H}}$	FINGER- $\widehat{\mathcal{H}}$	Linear	SLAQ
DBLP	0.06	0.65	394	28.4
Orkut	1.68	70	8863	899
LiveJournal	0.97	15.6	4727	476
Friendster	1.67	71	OOM	900
ClueWeb09	902	OOM	OOM	3447

11.3.3 GRAPH CLASSIFICATION PERFORMANCE

We test our method in the supervised downstream task, by classifying graphs in binary and multi-class settings. We compute `NETLSD` and `VNGE` descriptors for each of the graphs and use them as feature vectors in classification. Since these graph classification datasets allow direct calculation of the descriptor (maximum number of nodes reported in Table 11.3 is 5748), we can analyze how approximation affects the downstream accuracy.

We use a non-parametric 1-nearest neighbor classification algorithm and repeat the classification using 80/20 training/testing split 1000 times to minimize the biases introduced by the random splitting and the learning algorithm. We report the classification accuracy in Table 11.4.

Surprisingly, on two datasets, `D&D` and `Collab`, the classification accuracy is actually *better* for the low-accuracy approximation. We believe that this relates to the issues with these datasets pointed out by [SW19, CW19]: simple local graph features achieve almost state-of-the-art performance [ARPZ19]. However, for the `Reddit` datasets the improvement given by more accurate approximation is as expected due to the task being more sensitive to global structural information rather than simple node-level statistics.

11.3.4 PARAMETER SENSITIVITY

We investigate the approximation accuracy of `SLAQ` with respect to its hyper-parameters: number of random starting vectors n_v and the number of Lanczos iterations s . Recall that the error bounds in Section 11.2.2 tell us that there are two sources of error in `SLAQ`: one of the Monte Carlo estimation of the quadratic form and one of the Lanczos process. We measure the relative error ratio on the same 73 medium-sized graphs used in Section 11.3.1 with respect to the number of random starting vectors n_v and the number of Lanczos iterations s and report the results in Figure 11.8.

As expected, given enough starting random vectors `SLAQ` only needs a few Lanczos iterations; the default setting of $s = 10$ gives an average error of 6.7×10^{-4} . As for the number of random vectors n_v , we do observe that increasing the number improves performance, but the improvement given by increasing n_v is much slower.

11.3.5 SCALABILITY

We measure the runtime of all approximation techniques on huge graphs with millions of nodes and billions of edges and show that SLAQ is able to process very large graphs on a single machine within reasonable time while offering orders of magnitude better approximation, as measured in Section 11.3.1. We only report the results for VNGE, as the results for NETLSD for Linear interpolation and SLAQ approximation are similar to VNGE counterparts, while Taylor approximation works in the same time as FINGER- $\overline{\mathcal{H}}$.

As FINGER- $\overline{\mathcal{H}}$ only sums the weights of graphs' edges, it serves as a baseline on how much time it takes to scan the edges of a graph. A more useful comparison is FINGER- $\widehat{\mathcal{H}}$, as it reflects the time to compute a *single* eigenvalue of a graph. We approximate the *whole* spectrum at the cost of increased time complexity, however, the largest dataset with almost 5 billion nodes is processed in less than an hour.

11.4 SUMMARY

We propose SLAQ, an approximation technique for fast computation of spectral graph distances, VNGE and NETLSD, leveraging state-of-the-art linear algebra methods. We show that faithful approximation of the graph distance is critical for good downstream task performance and those approximation methods previously introduced in the literature do not offer good approximation quality. SLAQ improves approximation errors of such baseline solutions by at least an order of magnitude averaged across 73 real-world graphs. As SLAQ's computation is linear in the number of edges of graphs, the scalability of our method is on par with approximation techniques introduced for VNGE and NETLSD. To our knowledge, this is the first work that allows accurate comparison of billion-size graphs on a single machine in less than an hour.

ABILITY to represent and compare machine learning models is crucial to quantify subtle model changes, evaluate generative models, and gather insights on neural network architectures. Existing techniques for comparing data distributions focus on global data properties such as mean and covariance; in that sense, they are *extrinsic* and *uni-scale*. We use NETLSD to develop a first-of-its-kind *intrinsic* and *multi-scale* method for characterizing and comparing data manifolds. In a thorough experimental study, we demonstrate that our method effectively discerns the structure of data manifolds even on unaligned data of different dimensionality and showcase its efficacy in evaluating the quality of generative models.

The geometric properties of neural networks provide insights about their internals [MRB18, WHG⁺18] and help researchers in the design of more robust models [ACB17, BSAG18]. Generative models are a natural example of the need for geometric comparison of distributions. As generative models aim to reproduce the true data distribution \mathbb{P}_d using the model distribution $\mathbb{P}_g(\mathbf{z}; \Theta)$, more delicate evaluation procedures are needed. Oftentimes, we wish to compare data lying in entirely different spaces, such as tracking model evolution or comparing models with different representation spaces.

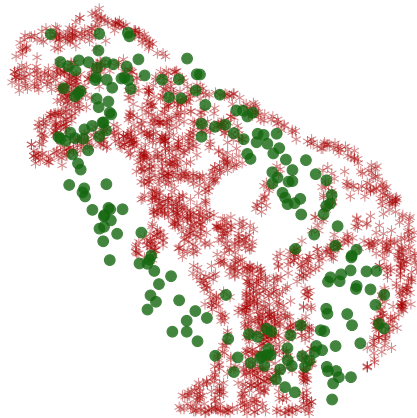


Figure 12.1: Two distributions having the same first 3 moments, meaning FID and KID scores are close to 0.

In order to evaluate the performance of generative models, several *extrinsic* evaluation measures were proposed in previous research, most notably the Fréchet [HRU⁺17] and Kernel [BSAG18] Inception Distances (FID and KID). Such measures only reflect the first two or three moments of distributions, meaning they can be insensitive to global structural problems. We showcase this inadvertence in Figure 12.1: here, FID and KID are insensitive to the data distribution’s global structure. Besides, as FID and KID are based only on *extrinsic* properties they are unable to compare *unaligned* data manifolds.

In this chapter, we start from the observation that models capturing the *multi-scale* nature of the data manifold by utilizing higher distribution moment matching, such as MMD-GAN [LCC⁺17] and sphere-GAN [PK19], perform consistently better than their single-scale counterparts. On the other hand, using *extrinsic* information can be misleading, as it is dependent on factors external to the data, such as representation. To address this drawback, we propose IMD, an Intrinsic Multi-scale Distance (IMD), that is able to compare distributions using only *intrinsic* information about the data, and provide an efficient approximation thereof that renders computational complexity nearly linear. We demonstrate that IMD effectively quantifies the difference in data distributions in three distinct application scenarios: comparing word vectors in languages with *unaligned* vocabularies, tracking dynamics of intermediate neural network representations, and evaluating generative models.

12.1 RELATED WORK

Geometric techniques enhance unsupervised and semi-supervised learning, generative and discriminative models [BN01, ACB17, Mém11]. We outline the applications of the proposed manifold comparison technique and highlight the geometric intuition along the way.

12.1.1 GENERATIVE MODEL EVALUATION

Past research has explored many different directions for the evaluation of generative models. Setting aside models that ignore the true data distribution, such as the Inception Score [SGZ⁺16] and GILBO [AF18], we discuss the most relevant geometric ideas below; we refer the reader to [Bor19] for a comprehensive survey.

Critic model-based metrics. Classifier two-sample tests [LPO17] aim to assess whether two samples came from the same distribution using an auxiliary classifier. The idea is similar to the GAN discriminator network [GPAM⁺14]: if a model distinguishes between samples from the model and the data distributions, these distributions are not entirely similar. The convergence process of some GAN discriminators [ACB17, BSAG18] informs a family of critic-based metrics. Still, training a separate critic model is often computationally prohibitive and requires careful specification. Besides, if the critic model is a neural network, the metric lacks interpretability and training stability.

Advanced GAN models such as Wasserstein, MMD, Sobolev and Spherical GANs impose different constraints on the function class so as to stabilize training [ACB17, MLS⁺18, PK19]. Higher-order moment matching [BSAG18, PK19] enhances performance, enabling GANs to capture multi-scale data properties, while multi-scale noise ameliorates GAN convergence problems [JF19]. Still, no feasible multi-scale GAN evaluation metric has been proposed to date.

Positional distribution comparison. In certain settings, it is acceptable to assign zero probability mass to the real data points [OBO⁺18]. In effect, metrics that estimate a distribution’s location and dispersion provide useful input for generative model evaluations. For instance, the Fréchet Inception Distance (FID) [HRU⁺17] computes the Wasserstein-2 (i.e., Fréchet) distance between distributions approximated with Gaussians, using only the estimated mean and covariance matrices; the Kernel Inception Distance (KID) [BSAG18] computes a polynomial kernel $k(x, y) = (\frac{1}{d}x^\top y + 1)^3$ and measures the associated Kernel Maximum Mean Discrepancy (kernel MMD). Unlike FID, KID has an unbiased estimator [GBR⁺12, BSAG18]. However, even while such methods, may be computationally inexpensive, they only provide a limited view on distributions from a geometric viewpoint.

Intrinsic geometric measures. The Geometry Score [KO18] characterizes distributions in terms of their estimated persistent homology, which roughly corresponds to the number of holes in a manifold. Still, the Geometry Score assesses distributions merely in terms of their *global* geometry. In this work, we aim to provide a *multi-scale* geometric assessment.

12.1.2 SIMILARITIES OF NEURAL NETWORK REPRESENTATIONS

Learning how representations evolve during training or across initializations provides a pathway to the interpretability of neural networks [RGYSD17]. Still, state-of-the-art methods for comparing representations of neural networks [KNLH19, MRB18, WHG⁺18] consider only linear projections. The intrinsic nature of IMD renders it appropriate for the task of comparing neural network representations, which can only rely on intrinsic information.

Pairwise Inner Product (PIP) loss [YS18], an unnormalized covariance error between sets, was introduced as a dissimilarity metric between word2vec embedding spaces with a common vocabulary. We show in Section 12.3.2 how IMD is applicable to this comparison task too.

12.2 INTRINSIC MULTI-SCALE DISTANCE

At the core of deep learning lies the *manifold hypothesis*, which states that high-dimensional data, such as images or text, lie on a low-dimensional manifold [NM10, BN01, BN07]. We aim to provide a theoretically motivated comparison of data manifolds based on rich intrinsic information. Our target measure should have the following properties:

- *intrinsic*—invariant to isometric transformations of the manifold, e.g. translations or rotations.
- *multi-scale*—it captures both local and global information.

We expose our method starting out with heat kernels, which admit a notion of manifold metric and can be used to lower-bound the distance between manifolds.

12.2.1 HEAT KERNELS ON MANIFOLDS AND GRAPHS

Based on the heat equation, the heat kernel captures *all* the information about a manifold’s intrinsic geometry [SOG09]. Given the Laplace–Beltrami operator (LBO) Δ_x on a manifold \mathcal{X} , the *heat equation* is $\frac{\partial u}{\partial t} = \Delta_x u$ for $u : \mathbb{R}^+ \times \mathcal{X} \rightarrow \mathbb{R}^+$. A smooth function u is a *fundamental solution* of the heat equation at point $x \in \mathcal{X}$ if u satisfies both the heat equation and the Dirac condition

$u(t, x') \rightarrow \delta(x' - x)$ as $t \rightarrow 0^+$. We assume the Dirichlet boundary condition $u(t, x) = 0$ for all t and $x \in \partial\mathcal{X}$. The heat kernel $k_{\mathcal{X}} : \mathcal{X} \times \mathcal{X} \times \mathbb{R}^+ \rightarrow \mathbb{R}_0^+$ is the unique solution of the heat equation; while heat kernels can be defined on hyperbolic spaces and other exotic geometries, we restrict our exposition to Euclidean spaces $\mathcal{X} = \mathbb{R}^d$, on which the heat kernel is defined as:

$$k_{\mathbb{R}^d}(x, x', t) = \frac{1}{(4\pi t)^{d/2}} \exp\left(-\frac{\|x - x'\|^2}{4t}\right) \quad (12.1)$$

For a compact \mathcal{X} including submanifolds of \mathbb{R}^d , the heat kernel admits the expansion $k_{\mathcal{X}}(x, x', t) = \sum_{i=0}^{\infty} e^{-\lambda_i t} \phi_i(x) \phi_i(x')$, where λ_i and ϕ_i are the i -th eigenvalue and eigenvector of $\Delta_{\mathcal{X}}$. For $t \simeq 0^+$, according to Varadhan's lemma, the heat kernel approximates geodesic distances. Importantly for our purposes, the Heat kernel is *multi-scale*: for a local domain \mathcal{D} with Dirichlet condition, the localized heat kernel $k_{\mathcal{D}}(x, x', t)$ is a good approximation of $k_{\mathcal{X}}(x, x', t)$ if either (i) \mathcal{D} is arbitrarily small and t is small enough, or (ii) t is for arbitrarily large and \mathcal{D} is big enough. Formally,

Definition 7 (Multi-scale property [Grio6, SOGo9]). (i) For any smooth and relatively compact domain $\mathcal{D} \subseteq \mathcal{X}$, $\lim_{t \rightarrow 0} k_{\mathcal{D}}(x, x', t) = \lim_{t \rightarrow 0} k_{\mathcal{X}}(x, x', t)$ (ii) For any $t \in \mathbb{R}^+$ and any $x, x' \in \mathcal{D}_1$ localized heat kernel $k_{\mathcal{D}_1}(x, x', t) \leq k_{\mathcal{D}_2}(x, x', t)$ if $\mathcal{D}_1 \subseteq \mathcal{D}_2$. Moreover, if $\{\mathcal{D}_n\}$ is an expanding and exhausting sequence $\bigcup_{i=1}^{\infty} \mathcal{D}_i = \mathcal{X}$ and $\mathcal{D}_{i-1} \subseteq \mathcal{D}_i$, then $\lim_{i \rightarrow \infty} k_{\mathcal{D}_i}(x, x', t) = \lim_{t \rightarrow 0} k_{\mathcal{X}}(x, x', t)$ for any t .

A useful invariant of the heat kernel is the *heat kernel trace* $\text{hkt}_{\mathcal{X}} : \mathcal{X} \times \mathbb{R}_0^+ \rightarrow \mathbb{R}_0^+$, defined by a diagonal restriction as $\text{hkt}_{\mathcal{X}}(t) = \int_{\mathcal{X}} k_{\mathcal{X}}(x, x, t) dx = \sum_{i=0}^{\infty} e^{-\lambda_i t}$ or, in the discrete case, $\text{hkt}_{\mathbf{L}}(t) = \text{tr}(\mathbf{H}_t) = \sum_i e^{-t\lambda_i}$. Heat kernels traces have been successfully applied to the analysis of 3D shapes [SOGo9] and graphs (Chapter 9). The heat kernel trace contains *all* the information in the graph's spectrum, both local and global, as the eigenvalues λ_i can be inferred therefrom [Mém11, Remark 4.8]. For example, if there are c connected components in the graph, then $\lim_{t \rightarrow \infty} \text{hkt}_{\mathbf{L}}(t) = c$.

12.2.2 CONVERGENCE TO THE LAPLACE-BELTRAMI OPERATOR

Heat kernels are defined for graphs in terms of their Laplacian matrices. An important property of graph Laplacians is that it is possible to construct a graph

among points sampled from a manifold \mathcal{X} such that the spectral properties of its Laplacian resemble those of the Laplace–Beltrami operator on \mathcal{X} . Belkin and Niyogi [BNo1] proposed such a construction, the point cloud Laplacian, which is used for dimensionality reduction in a technique called Laplacian eigenmaps. Convergence to the LBO has been proven for various definitions of the graph Laplacian, including the one we use [BNo7, HALo7, CLo6, THJ10]. We recite the convergence results for the point cloud Laplacian from [BNo7]:

Theorem 7. Let $\lambda_{n,i}^{t_n}$ and $\phi_{n,i}^{t_n}$ be the i^{th} eigenvalue and eigenvector, respectively, of the point cloud Laplacian \mathbf{L}^{t_n} ; let λ_i and ϕ_i be the i^{th} eigenvalue and eigenvector of the LBO Δ . Then, there exists $t_n \rightarrow 0$ such that

$$\begin{aligned} \lim_{n \rightarrow \infty} \lambda_{n,i}^{t_n} &= \lambda_i \\ \lim_{n \rightarrow \infty} \|\phi_{n,i}^{t_n} - \phi_i\|_2 &= 0 \end{aligned}$$

Still, the point cloud Laplacian involves the creation of an $\mathcal{O}(n^2)$ matrix; for the sake of scalability, we use the k -nearest-neighbors (k NN) graph by OR-construction (i.e., based on bidirectional k NN relationships among points), whose Laplacian converges to the LBO for data with sufficiently high intrinsic dimension [THJ10]. As for the choice of k , a random geometric k NN graph is connected when $k \geq \log n / \log 7 \approx 0.5139 \log n$ [BBSW05]; $k = 5$ yields connected graphs for all sample sizes we tested.

12.2.3 SPECTRAL GROMOV–WASSERSTEIN DISTANCE

Even while it is a multi-scale metric *on* manifolds, the heat kernel can be spectrally approximated by finite graphs constructed from points sampled from these manifolds. In order to construct a metric *between* manifolds, [Mém11] suggests an optimal-transport-theory-based “meta-distance”: a spectral definition of the Gromov–Wasserstein distance between Riemannian manifolds based on matching the heat kernels at all scales. The cost of matching a pair of points (x, x') on manifold \mathcal{M} to a pair of points (y, y') on manifold \mathcal{N} at scale t is given by their heat kernels $k_{\mathcal{M}}, k_{\mathcal{N}}$:

$$\Gamma(x, y, x', y', t) = |k_{\mathcal{M}}(x, x', t) - k_{\mathcal{N}}(y, y', t)|.$$

The distance between the manifolds is then defined in terms of the infimal measure coupling

$$d_{\text{GW}}(\mathcal{M}, \mathcal{N}) = \inf_{\mu} \sup_{t>0} e^{-2(t+t^{-1})} \|\Gamma\|_{L^2(\mu \times \mu)},$$

where the infimum is sought over all measures μ on $\mathcal{M} \times \mathcal{N}$ marginalizing to the standard measures on \mathcal{M} and \mathcal{N} . For finite spaces, μ is a doubly-stochastic matrix. This distance is lower-bounded [Mém11] in terms of the respective heat kernel traces as:

$$d_{\text{GW}}(\mathcal{M}, \mathcal{N}) \geq \sup_{t>0} e^{-2(t+t^{-1})} |\text{hkt}_{\mathcal{M}}(t) - \text{hkt}_{\mathcal{N}}(t)|. \quad (12.2)$$

This lower bound is the scaled L_{∞} distance between the *heat trace signatures* $\text{hkt}_{\mathcal{M}}$ and $\text{hkt}_{\mathcal{N}}$. The scaling factor $e^{-2(t+t^{-1})}$ favors medium-scale differences, meaning that this lower bound is not sensitive to local perturbations. The maximum of the scaling factor occurs at $t = 1$, and more than $1 - 10^{-8}$ of the function mass lies between $t = 0.1$ and $t = 10$.

12.2.4 PUTTING IMD TOGETHER

We employ the heretofore described advances in differential geometry and numerical linear algebra to create `IMD` (*Multi-Scale Intrinsic Distance*), a fast, intrinsic method to lower-bound the spectral Gromov-Wasserstein distance between manifolds. We use `SLAQ` from Chapter 11 to approximate the heat trace.

Algorithm 7 `IMD` algorithm.

```

function imd_desc(X)
    G ← kNN(X)
    L ← Laplacian(G)
    return  $\Gamma = \text{slq}(\mathbf{L}, s, n_v)$ 

function imd_dist(X, Y)
    hktX ← imd_desc(X)
    hktY ← imd_desc(Y)
    return  $\sup e^{-2(t+t^{-1})} |\text{hkt}_X - \text{hkt}_Y|$ 

```

We describe the overall computation of `IMD` in Algorithm 7. Given data samples in \mathbb{R}^d , we build a `kNN` graph G by OR-construction such that its Laplacian spectrum approximates the one of the Laplace-Beltrami operator of the

underlying manifold [THJ10], and then compute $\text{hkt}_G(t) = \sum_i e^{-\lambda_i t} \approx \Gamma$. We compare heat traces in the spirit of Equation (12.2), i.e., $|\text{hkt}_{G_1}(t) - \text{hkt}_{G_2}(t)|$ for $t \in (0.1, 10)$ sampled from a logarithmically spaced grid.

Constructing exact k NN graphs is an $\mathcal{O}(dn^2)$ operation; however, approximation algorithms take near-linear time $\mathcal{O}(dn^{1+\omega})$ [DML11, ABF19]. In practice, with approximate k NN graph construction [DML11], computational time is low while result variance is similar to the exact case. The s -step Lanczos algorithm on a sparse $n \times n$ k NN Laplacian \mathbf{L} with one starting vector has $\mathcal{O}(ksn)$ complexity, where kn is the number of nonzero elements in \mathbf{L} . The symmetric tridiagonal matrix eigendecomposition incurs an additional $\mathcal{O}(s \log s)$ [CR13]. We apply this algorithm over n_v starting vectors, yielding a complexity of $\mathcal{O}(n_v(s \log s + ksn))$, with constant $k = 5$ and $s = 10$ by default. In effect, IMD’s time complexity stands between those of two common GAN evaluation methods: KID, which is $\mathcal{O}(dn^2)$ and FID, which is $\mathcal{O}(d^3 + dn)$. The time complexity of Geometry Score is unspecified in [KO18], yet in Section 12.3.6 we show that its runtime grows exponentially in sample size.

12.3 EXPERIMENTS

We evaluate IMD on the ability to compare intermediate representations of machine learning models. For instance, in a recommender system we could detect whether a problem is related to the representation or the classifier in the end of a pipeline. In this section, we show the effectiveness of our intrinsic measure on multiple tasks and show how our intrinsic distance can provide insights beyond previously proposed extrinsic measures.

Summary of experiments. We examine the ability of IMD¹ to measure several aspects of difference among data manifolds. We first consider a task from unsupervised machine translation with unaligned word embeddings and show that IMD captures correlations among language kinship (affinity or genealogical relationships). Second, we showcase how IMD handles data coming from data sources of unequal dimensionalities. Third, we study how IMD highlights differences among image data representations across initializations and through the training process of neural networks.

1. Our code is available open-source: <https://github.com/xgfs/imd>.

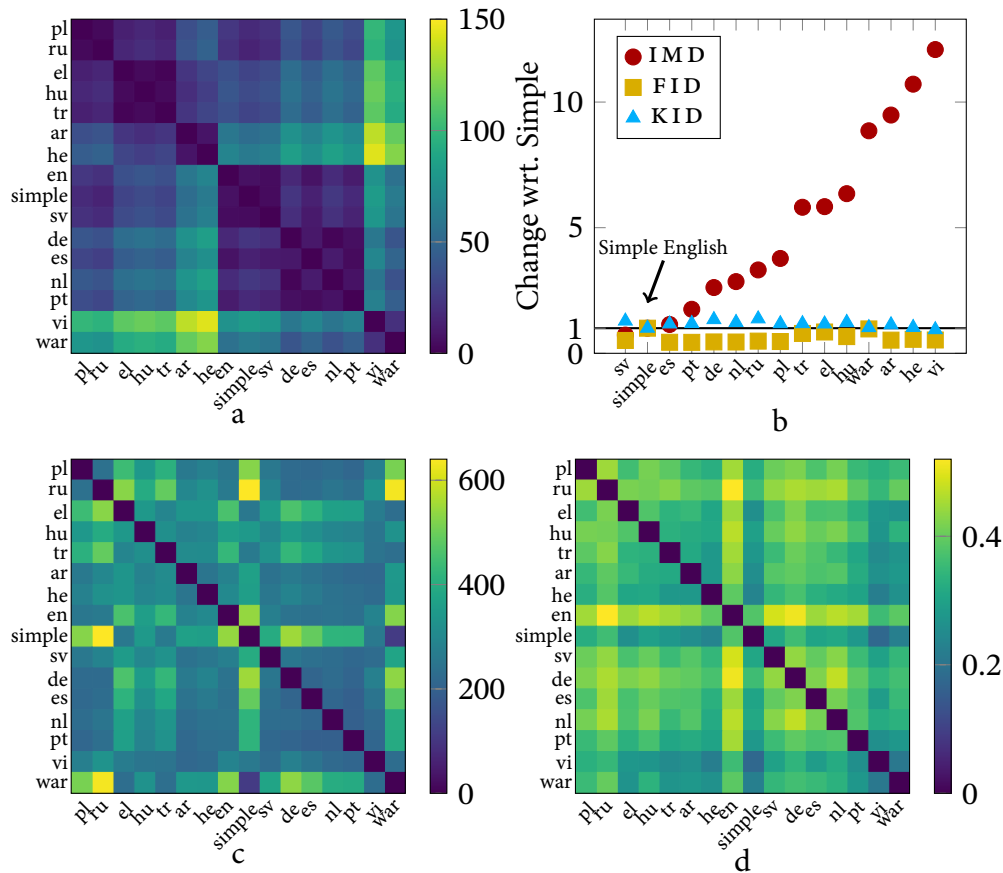


Figure 12.2: (a) IMD distances between language pairs for unaligned Wikipedia word embeddings; (b) distances from the simple English Wikipedia visualized for IMD, FID, and KID; (c) pairwise FID distances; (d) pairwise KID distances. We consider 16 languages: Polish, Russian, Greek, Hungarian, Turkish, Arabic, Hebrew, English, Simple English, Swedish, German, Spanish, Dutch, Portuguese, Vietnamese, and Waray-Waray.

12.3.1 COMPARING UNALIGNED LANGUAGE MANIFOLDS

The problem of unaligned representations is particularly severe in the domain of natural language processing as the vocabulary is rarely comparable across different languages or even different documents. We employ *IMD* to measure the relative closeness of pairs of languages based on the word embeddings with different vocabularies. Figure 12.2 (a) shows a heatmap of pairwise *IMD* scores. *IMD* detects similar languages (Slavic, Semitic, Romanic, etc.) despite the lack of ground truth vocabulary alignment. We use *gensim* [RS10] to learn word vectors on the latest Wikipedia corpus snapshot on 16 languages: Polish, Russian, Greek, Hungarian, Turkish, Arabic, Hebrew, English, Simple English, Swedish, German, Spanish, Dutch, Portuguese, Vietnamese, and Waray-Waray. We then compute *FID*, *KID*, and *IMD* scores on all the pairs, in total we average 100 runs.

Figure 12.3: Comparison of IMD and PIP loss on word embeddings of different dimension. Note how IMD detects subtle changes in the dimensionality.

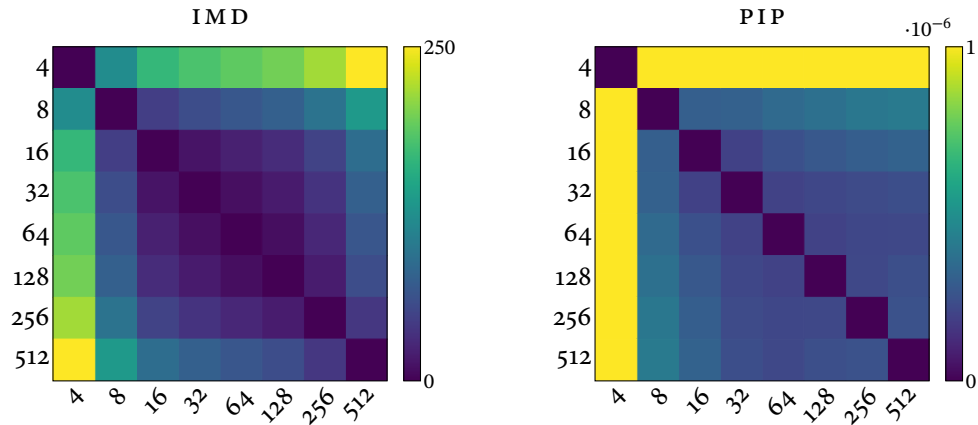


Figure 12.2 (b,c) shows the matrix of distances for FID and KID aligned and colored in the same way as Figure 12.2 (a). We observe FID and KID can not find meaningful structure in the data in the same way as IMD as they rely on extrinsic data properties.

12.3.2 OPTIMIZING DIMENSIONALITY OF WORD EMBEDDINGS

Comparing data having different dimensionality is cumbersome, even when representations *are* aligned. We juxtapose IMD by PIP loss [YS18] which allows the comparison of aligned representations for word embeddings. To this end, we measure IMD distance between English word embeddings of varying dimensions. Figure 12.3 shows the heatmap of the scores between sets of word vectors of different dimensionalities. Closer dimensionalities have lower distance scores for both metrics. However, IMD better highlights gradual change of the size of word vectors, e.g., word vectors of size 4 and 8 are clearly closer to each other than embeddings of size 4 and 16 in terms of IMD, which is not true for PIP.

12.3.3 TRACKING THE EVOLUTION OF IMAGE MANIFOLDS

Next, we employ IMD to inspect the internal dynamics of neural networks. We investigate *the stability of output layer manifolds* across random initializations. We train 10 instances of the VGG-16 [SZ15] network using different weight initializations on the CIFAR-10 and CIFAR-100 datasets. We compare the average IMD scores across representations in each network layer relative to the last layer. As Figure 12.5 (left) shows, for both CIFAR-10 and CIFAR-100,

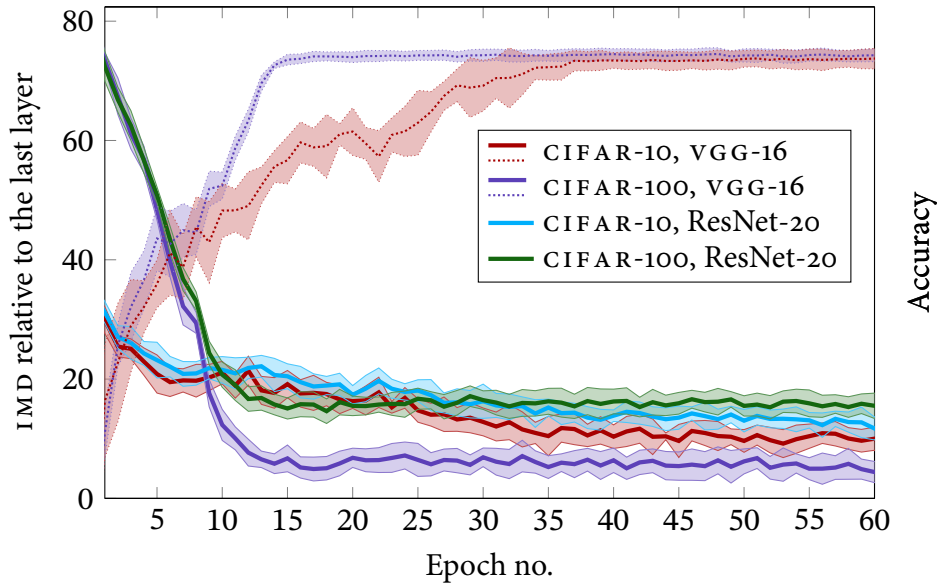


Figure 12.4: Training progression in terms of accuracy (dotted) and IMD (solid) on CIFAR-10 and CIFAR-100 datasets for VGG-16 and ResNet-20, with respect to VGG-16.

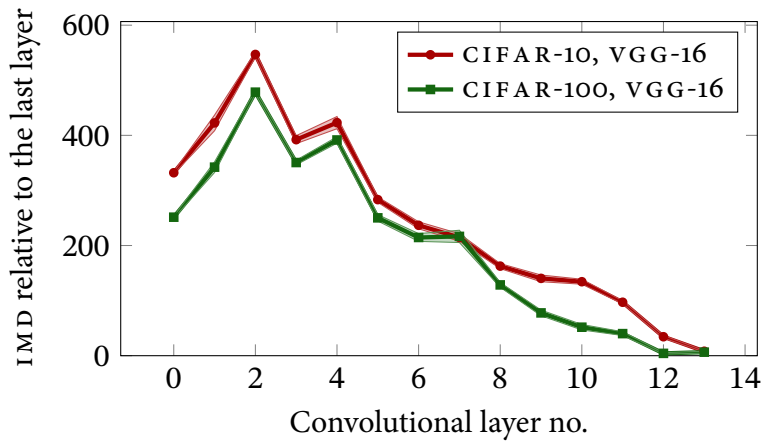
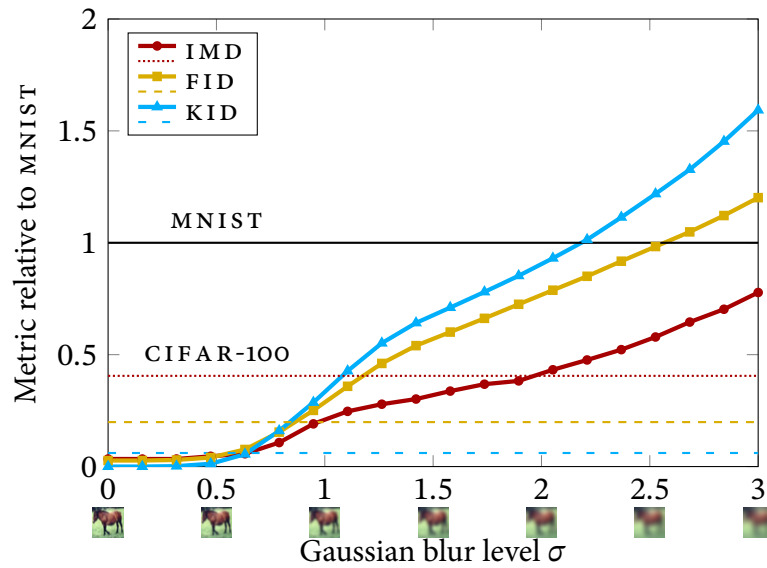


Figure 12.5: Values of IMD across convolutional layers of the VGG-16 network on CIFAR-10 and CIFAR-100 datasets.

the convolutional layers exhibit similar behavior; As IMD shows, consequent layers do not monotonically contribute to the separation of image representations, but start to do so after initial feature extraction stage comprised of 4 convolutional blocks. initializations indicates stability in the network structure.


We now *examine the last network layers* during training with different initializations. Figure 12.4 plots the VGG-16 validation errors and IMD scores relative to the final layer representations of *two* pretrained networks, VGG-16 itself with last layer dimension $d = 512$ and ResNet-20 with $d = 64$ and ~ 50 times less parameters. We observe that even in such unaligned spaces, IMD

Figure 12.6: FID, KID and IMD on the CIFAR-10 dataset with Gaussian blur.



correctly identifies the convergence point of the networks. Surprisingly, we find that, in terms of IMD, VGG-16 representations progress towards not only the VGG-16 final layer, but the ResNet-20 final layer representation as well; this result suggests that these networks of distinct architectures share similar final structures.

12.3.4 EVALUATING GENERATIVE MODELS

We now move on to apply IMD to evaluation of generative models. First, we evaluate the *sensitivity* of IMD, FID, and KID to simple image transformations as a proxy to more intricate artifacts of modern generative models. We progressively blur images from the CIFAR-10 training set, and measure the distance to the original data manifold, averaging outcomes over 100 subsamples of 10k images each. To enable comparison across methods, we normalize each distance measure such that the distance between CIFAR-10 and MNIST is 1. Figure 12.6 reports the results at different levels σ of Gaussian blur. We additionally report the normalized distance to the CIFAR-100 training set (dashed lines .

We observe that FID and KID quickly drift away from the original distribution and match MNIST, a dataset of a completely different nature. Contrariwise, IMD is more robust to noise and follows the datasets structure, as

the relationships between objects remain mostly unaffected on low blur levels. Moreover, with both FID and KID, low noise ($\sigma = 1$) applied to CIFAR-10 suffices to exceed the distance of CIFAR-100, which is similar to CIFAR-10. IMD is much more robust, exceeding that distance only with $\sigma = 2$.

Metric	MNIST		FashionMNIST		CIFAR10		CelebA	
	WGAN	WGAN-GP	WGAN	WGAN-GP	WGAN	WGAN-GP	WGAN	WGAN-GP
IMD	57.74 \pm	10.77 \pm	118.14 \pm	13.45 \pm	18.10 \pm	10.84 \pm	10.11 \pm	2.84 \pm
	0.47	0.42	0.52	0.54	0.36	0.42	0.33	0.31
KID $\times 10^3$	47.26 \pm	5.53 \pm	119.93 \pm	25.49 \pm	93.89 \pm	59.59 \pm	217.28 \pm	92.71 \pm
	0.07	0.03	0.14	0.07	0.09	0.09	0.14	0.08
FID	31.75 \pm	8.95 \pm	152.44 \pm	35.31 \pm	101.43 \pm	80.65 \pm	205.63 \pm	85.55 \pm
	0.07	0.03	0.12	0.07	0.09	0.09	0.09	0.08

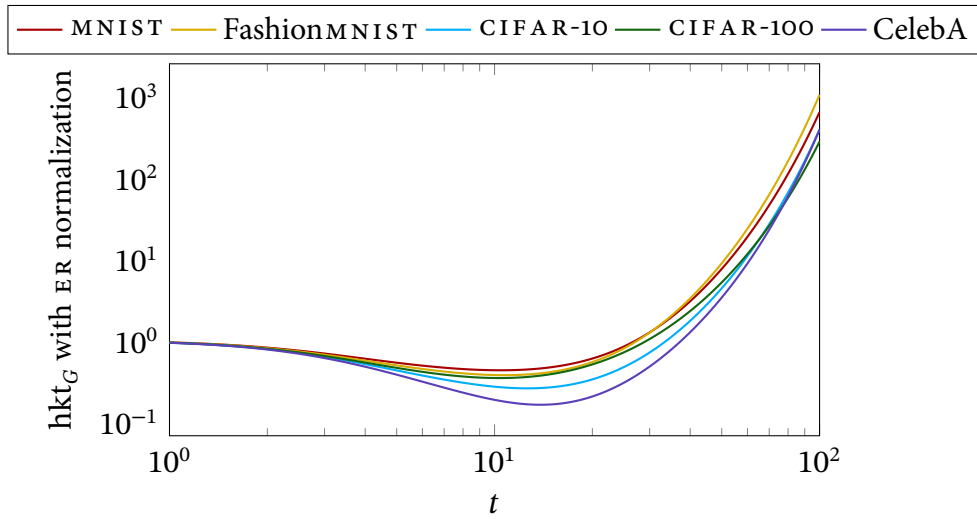
Next, we turn our attention to the *sample-based evaluation* of generative models. We then train the WGAN [ACB17] and WGAN-GP [GAA⁺17] models on four datasets: MNIST, FashionMNIST, CIFAR-10 and CelebA. We sample 10k samples, \mathbf{Y} , from each GAN. We then uniformly subsample 10k images from the corresponding original dataset, \mathbf{X} , and compute the IMD, KID and FID scores between \mathbf{X} and \mathbf{Y} . Table 12.1 reports the average measure and its 99% confidence interval across 100 runs. IMD, as well as both FID and KID, reflect the fact that WGAN-GP is a more expressive model.

Table 12.1: IMD agrees with KID and FID across varying datasets for GAN evaluation.

12.3.5 INTERPRETING IMD

To understand how IMD operates, we investigate the behavior of heat kernel traces of different datasets that are normalized by a null model. [TMK⁺18b] proposed a normalization by the heat kernel trace of an empty graph, which amounts to taking the average, rather than the sum, of the original heat kernel diagonal. However, this normalization is not an appropriate null model as it ignores graph connectivity. We propose a heat kernel normalization by the *expected* heat kernel of an Erdős–Rényi graph. For the purpose of normalizing IMD, we need to approximate that graph’s eigenvalues. It was proved [CO07] that $\lambda_1 \leq 1 - c\bar{d}^{-1/2} \leq \lambda_2 \leq \lambda_n \leq 1 + c\bar{d}^{-1/2}$ for the core of the graph for some constant c . We have empirically found that $c = 2$ provides a tight approximation for random graphs. That coincides with the analysis of [CLV04], who proved that $\lambda_n = (1 + o(1))2\bar{d}^{-1/2}$ if $d_{\min} \gg \sqrt{\bar{d}} \log^3 n$ even though in our

Figure 12.7: Plotting the normalized heat trace allows interpretation of medium- and global-scale structure of datasets.



case $d_{\min} = \bar{d} = k$. We thus estimate the spectrum of a random Erdős–Rényi graph as growing linearly between $\lambda_1 = 1 - 2\bar{d}^{-1/2}$ and $\lambda_n = 1 + 2\bar{d}^{-1/2}$, which corresponds to the underlying manifold being two-dimensional [Wey11].

Figure 12.7 depicts the obtained normalized hkt_g for all datasets we work with. We average results over 100 subsamples of $10k$ images each. For $t = 10$, i.e., at a medium scale, CelebA is most different from the random graph, while for large-scale t values, which capture global community structure, $\frac{d\text{hkt}_g(t)}{dt}$ reflects the approximate number of clusters in the data. Surprisingly, CIFAR-100 comes close to CIFAR-10 for large t values; we have found that this is due to the fact that the pre-trained Inception network does not separate the CIFAR-100 data classes well enough. We conclude that the heat kernel trace is interpretable if we normalize it with an appropriate null model.

12.3.6 VERIFYING STABILITY AND SCALABILITY OF IMD

In addition to the complexity analysis in Section 12.2.4, we assess the *scaling and sample stability* of IMD. Since IMD, like FID, is a lower bound to an optimal transport-based metric, we cannot hope for an unbiased estimator. However, we empirically verify, in Figure 12.8 (left), that IMD does not diverge too much with increased sample size. Most remarkably, we observe that IMD with approximate k_{NN} [DML11] does not induce additional variance, while it diverges slightly from the exact version as the number of samples grows.

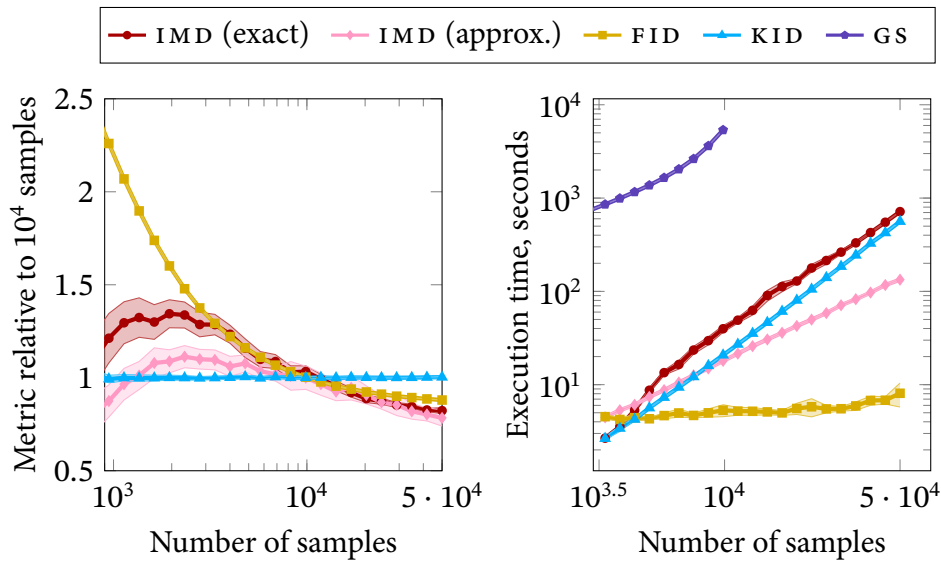


Figure 12.8: Stability and scalability experiment: (left) stability of FID, KID and IMD wrt. sample size on CIFAR-10 and CIFAR-100 dataset; (right) scalability of FID, KID and IMD wrt. sample size on synthetic datasets.

In terms of *scalability*, Figure 12.8 (right) shows that the theoretical complexity is supported in practice. Using approximate k NN, we break the $\mathcal{O}(n^2)$ performance of KID. The time complexity of FID appears constant, as its runtime is dominated by the $\mathcal{O}(d^3)$ matrix square root operation. Geometry Score fails to perform scalably, as its runtime grows exponentially. Due to this prohibitive computational cost, we eschew other comparisons with it. Furthermore, as IMD distance is computed through a low-dimensional heat trace representation of the manifold, we can store HKT for future comparisons, thereby enhancing performance in the case of many-to-many comparisons.

12.4 SUMMARY

We introduced IMD, a geometry-grounded, first-of-its-kind intrinsic multi-scale method for comparing unaligned manifolds, which we approximate efficiently with guarantees, utilizing the Stochastic Lanczos Quadrature. We have shown the expressiveness of IMD in quantifying the change of data representations in NLP and image processing, evaluating generative models, and in the study of neural network representations. Since IMD allows comparing diverse manifolds, its applicability is not limited to the tasks we have evaluated, while it paves the way to the development of even more expressive techniques founded on geometric insights.

PART III

SUMMARY

THIS thesis proposes novel methods for representing graph-structured data in the form suitable for most machine learning algorithms. We analyzed representations of nodes in Part I and graphs in Part II. We introduced several expressive algorithms for each of these data modalities and then proposed local, scalable versions of these algorithms. This section summarizes our main research achievements and presents an outlook on future research directions.

The first part of this thesis introduces four methods for representing nodes as vectors. This part’s primary contribution is proposing expressive families of scalable models. Our neural network-based approach, `VERSE`, decouples the choice of vertex similarity and the optimization model. Besides excellent performance that has been independently verified [CLX19, KAS19, MLDB20], `VERSE` provides a unified framework for learning from vertex similarities, allowing us to analyze existing algorithms. Our framework influenced several investigation lines, particularly in building scalable node embedding systems [YW19, ZXTQ19], improving graph learning [KWG19], and providing better models for other types of graphs [SDT19, PLVT⁺20].

Despite its empirical success, `VERSE` does not have any quality or convergence guarantees—this warranted further investigation. We study `VERSE`’s optimal solution and propose `FREDE`, an anytime algorithm that addresses a major weakness of `VERSE`: the lack of quality guarantees. We are the first to show how to build a valid graph embedding using only a subset of nodes as reference points. We observe comparable or better quality than other methods while processing only 10% of the nodes.

Notwithstanding, the efficiency of `FREDE` is not enough for most extreme¹ graph mining applications. We notice that downstream tasks routinely use only a small part of node embeddings in many real-world application scenarios. Therefore, graph embedding algorithms that can only embed all nodes at once are wasting precious compute and storage capabilities. We take the locality observation in `FREDE` to its logical conclusion and propose a first

1. In terms of the volume and velocity of data.

local node embedding algorithm, SnapEmbed. It can create representations for a single node without peeking at other nodes' embeddings. We prove that such independent actions prognosticatively align to paint an accurate picture of every node in a graph. Since SnapEmbed is a local algorithm, computations can be easily distributed, allowing to embed massive graphs in record time.

None of the methods introduced above deal with attributed graphs. We correct this malentendu with the help of DMON. We depart from the contrastive approaches for training graph neural networks and propose the first clustering-based objective for training GNNs. We continue to focus on scalability, achieving training and inference time linear in the number of graphs' edges. Besides best-in-class scalability, DMON achieves 30–40% improvements in clustering performance over competing approaches.

Part II of this thesis is dedicated to vector representations of whole graphs. This part's core contribution is the proposal of multi-scale representations in Chapter 9. We translate geometric ideas to the graphs' language and propose the first spectral representation of graphs with distance guarantees, NETLSD. We show that no other scalable distance is able to capture the multi-scale structure of graphs as completely as NETLSD and propose an approximation technique to scale the computation to graphs with millions of nodes. Our measure has been applied to hyperparameter optimization of graph-based machine learning algorithms [TMC⁺19] and graph coarsening [JLJ20].

While NETLSD has exciting theoretical guarantees, it is rigid in its choice of the scale it captures—essentially, it is a collection of low-pass filters of the spectrum. We leverage a self-supervised neural network, SGR, to adapt the filters of NETLSD on generated data. We are the first to propose the tasks for the self-supervision of graph representations. We demonstrate that SGR shows consistent improvements over its non-learned counterparts.

Scalability is an overarching topic of this thesis. We adapt state-of-the-art linear algebra techniques and introduce SLAQ, a way of approximating several spectral descriptors, including NETLSD. We derive the error bounds and show that SLAQ outperforms existing techniques by up to two orders of magnitude on average in approximation quality while being comparable time-wise. Thus, SLAQ allows NETLSD to become the first graph comparison processing billion-scale graphs without relying on node correspondence.

Last, we study the comparison of unaligned distributions as an application of NETLSD's fast computation. We propose IMD, a distance between samples from unaligned distributions that leverages the intrinsic geometry of data. Since the underlying geometry of data is well approximated by graphs constructed from it, IMD is competitive with methods leveraging data's extrinsic geometry and provides a unique outlook when the extrinsic information is not reliable. We apply IMD to study various neural networks' inner structures, including ones for learning neural representations of words.

We began the first part by modeling graphs *with* neural networks and finished the second one by analyzing the graph structure *of* neural networks. Our long-term plan is to continue the development of the foundations of graph mining and connect it further with the field of machine learning. We outline some of the research directions below.

13.1 FUTURE WORK

13.1.1 APPLICATION OUTREACH

First and foremost, it is important to expand the use of graph analysis techniques in various application domains. There are fascinating opportunities to apply graph mining systems in chemical and biological modeling, security, financial systems, and many other fields. Problems in these fields possess unique challenges and constraints, and require close collaboration with researchers and practitioners. We firmly believe that collaborations across fields will bring many advances to the field of graph mining.

Some of the most exciting applications of graph mining *within* machine learning lie in the privacy-preserving machine learning. For example, federated learning algorithms operate across multiple decentralized devices holding local data samples, without exchanging them. SnapEmbed is our step towards user-centric graph computations that give us the benefits of sophisticated machine learning models without compromising user privacy. We foresee graph research to bring invaluable insights on organizational patterns of distributed privacy-aware models.

13.1.2 FURTHER RESEARCH DIRECTIONS

There are several research directions that we find particularly exciting. Perhaps, the elephant in the room is that we do not know how to build a graph from data. We do know how to approximate shapes or surfaces with graphs, but we know almost nothing about how to construct graphs that facilitate learning from data. Additionally, the common assumption that the graph is the ground truth is routinely broken in practice. Therefore, methods that question the nature of each edge should have an edge in practical applications.

There are always more graph types coming from various application domains. For example, models for graphs with heterogeneous information on nodes and edges require careful specification to leverage both the graph and attribute structure. Attributes are extremely common in some applications in chemistry and biology. They pose further challenges: what if the data is not aligned to the graph structure at all? what if the graph and the data give completely different signals? An additional challenge is missing data—is there a graph-aware imputation method that can deal with that? Challenging common modeling assumptions is a crucial research direction.

Even in simple graphs, different analysis modalities await their methods. For example, Chapter 4 introduces methods for producing embeddings of edges from representations of nodes. However, is this method optimal? Wouldn't a method specializing in edge representations produce better ones? Representations of subgraphs is an analogue of this problem from Part II. Can we represent subgraphs quicker than treating them as a separate graph? These are questions waiting for a satisfactory answer.

Better models of graphs allow us to intimately study the behaviors of our models in controlled scenarios. For example, we provide additional insights on our techniques from Chapters 7 and 9 via generating graphs. Expressive graphs distance measures inform a new generation of graph generators. Using them, we can study even more realistic properties of real-world graphs and improve the design of learning algorithms.

To summarize, graphs have numerous impactful applications and fascinating research directions. We proposed expressive, scalable models for creating both node and graph representations and evaluated them theoretically and experimentally. We hope that advances in the field will be societally beneficial.

BIBLIOGRAPHY

- [ABB⁺99] Edward Anderson, Zhaojun Bai, Christian Bischof, Susan Blackford, Jack Dongarra, Jeremy Du Croz, Anne Greenbaum, Sven Hammarling, Alan McKenney, and D Sorensen. *LAPACK Users' guide*. SIAM, 1999. Cited on page 144.
- [Abb18] Emmanuel Abbe. Community detection and stochastic block models: Recent developments. *JMLR*, 2018. Cited on pages 10 and 127.
- [ABF19] Martin Aumüller, Erik Bernhardsson, and Alexander Faithfull. ANNBenchmarks: A benchmarking tool for approximate nearest neighbor algorithms. *Information Systems*, 2019. Cited on page 158.
- [ACB17] Martín Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein generative adversarial networks. In *ICML*, 2017. Cited on pages 151, 152, 153, and 163.
- [Acho3] Dimitris Achlioptas. Database-friendly random projections: Johnson-lindenstrauss with binary coins. *Journal of computer and System Sciences*, 66(4):671–687, 2003. Cited on pages 19 and 24.
- [ACLo7] Reid Andersen, Fan Chung, and Kevin Lang. Using pagerank to locally partition a graph. *Internet Mathematics*, 4(1):35–64, 2007. Cited on pages 11, 66, 67, 68, and 71.
- [AdMo6] Alexandre H Abdo and APS de Moura. Clustering as a measure of the local topology of networks. *arXiv preprint physics/0605235*, 2006. Cited on pages 124, 126, and 128.
- [AF18] Alexander A. Alemi and Ian Fischer. GILBO: One metric to measure them all. In *NeurIPS*, 2018. Cited on page 152.
- [AMF10] Leman Akoglu, Mary McGlohon, and Christos Faloutsos. Oddball: Spotting anomalies in weighted graphs. In *PAKDD*, pages 410–421, 2010. Cited on page 17.
- [ARPZ19] Rami Al-Rfou, Bryan Perozzi, and Dustin Zelle. DDGK: Learning graph representations for deep divergence graph kernels. In *WWW*, 2019. Cited on page 149.
- [ASN⁺13] Amr Ahmed, Nino Shervashidze, Shравan Narayanamurthy, Vanja Josifovski, and Alexander J Smola. Distributed large-scale natural graph factorization. In *WWW*. ACM, 2013. Cited on page 21.
- [AT11] Haim Avron and Sivan Toledo. Randomized algorithms for estimating the trace of an implicit symmetric positive semi-definite matrix. *Journal of the ACM (JACM)*, 2011. Cited on page 137.
- [AT16] James Atwood and Don Towsley. Diffusion-convolutional neural networks. In *NIPS*, pages 1993–2001, 2016. Cited on page 123.
- [AV07] David Arthur and Sergei Vassilvitskii. k-means++: The advantages of careful seeding. In *SIAM*, pages 1027–1035, 2007. Cited on pages 40 and 90.
- [Bal87] Dana H Ballard. Modular learning in neural networks. In *AAAI*, 1987. Cited on page 18.
- [BB11] Michael M Bronstein and Alexander M Bronstein. Shape recognition with spectral distances. *TPAMI*, 2011. Cited on page 110.

- [BBGO₁₁] Alexander M Bronstein, Michael M Bronstein, Leonidas J Guibas, and Maks Ovsjanikov. Shape google: Geometric words and expressions for invariant shape retrieval. *TOG*, 30(1):1, 2011. Cited on page 104.
- [BBK⁺₁₆] Stephen Bonner, John Brennan, Ibad Kureshi, G Theodoropoulos, and AS McGough. Efficient comparison of massive graphs through the use of graph fingerprints. In *Twelfth Workshop on Mining and Learning with Graphs (MLG) Workshop at KDD'16*, 2016. Cited on page 104.
- [BBL⁺₁₇] Michael M Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst. Geometric deep learning: going beyond euclidean data. *IEEE Signal Processing Magazine*, 34(4):18–42, 2017. Cited on pages 13 and 123.
- [BBSW₀₅] Paul Balister, Béla Bollobás, Amites Sarkar, and Mark Walters. Connectivity of random k-nearest-neighbour graphs. *Advances in Applied Probability*, 2005. Cited on page 156.
- [BCG₁₀] Bahman Bahmani, Abdur Chowdhury, and Ashish Goel. Fast incremental and personalized PageRank. *PVLDB*, 4(3):173–184, 2010. Cited on page 49.
- [BCW₁₈] Nitin Bansal, Xiaohan Chen, and Zhangyang Wang. Can we gain more from orthogonality regularizations in training deep cnns? In *NIPS*, 2018. Cited on page 88.
- [BDG⁺₀₆] Ulrik Brandes, Daniel Delling, Marco Gaertler, Robert Görke, Martin Hoefer, Zoran Nikoloski, and Dorothea Wagner. Maximizing modularity is hard. *arXiv preprint physics/0608255*, 2006. Cited on page 86.
- [BDMI₁₁] Christos Boutsidis, Petros Drineas, and Malik Magdon-Ismail. Near optimal column-based matrix reconstruction. *FOCS*, pages 305–314, 2011. Cited on pages 46, 47, and 48.
- [Ber₁₂] Marcel Berger. *A panoramic view of Riemannian geometry*. Springer Science & Business Media, 2012. Cited on page 108.
- [BGA₂₀] Filippo Maria Bianchi, Daniele Grattarola, and Cesare Alippi. Spectral clustering with graph neural networks for graph pooling. In *ICML*, 2020. Cited on pages 85 and 90.
- [BGJM₁₇] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information. *ACL*, 2017. Cited on page 13.
- [BGLL₀₈] Vincent D Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. Fast unfolding of communities in large networks. *Journal of statistical mechanics: theory and experiment*, (10), 2008. Cited on pages 35 and 40.
- [BGS₀₆] Samuel L Braunstein, Sibasish Ghosh, and Simone Severini. The laplacian of a graph as a density matrix: a basic combinatorial approach to separability of mixed states. *Annals of Combinatorics*, pages 291–317, 2006. Cited on pages 133 and 134.
- [BH₈₆] Josh Barnes and Piet Hut. A hierarchical O(N log N) force-calculation algorithm. *Nature*, 1986. Cited on page 20.
- [BHB⁺₁₈] Peter W Battaglia, Jessica B Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, et al. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018. Cited on page 13.

- [BKo5] Karsten M Borgwardt and Hans-Peter Kriegel. Shortest-path kernels on graphs. In *Data Mining, Fifth IEEE International Conference on*, pages 8–pp. IEEE, 2005. Cited on page 103.
- [BKERF13] Michele Berlingerio, Danai Koutra, Tina Eliassi-Rad, and Christos Faloutsos. Network similarity via multiple social theories. In *ASONAM*, pages 1439–1440, 2013. Cited on pages 104, 114, 115, and 129.
- [BMD09] Christos Boutsidis, Michael W. Mahoney, and Petros Drineas. An improved approximation algorithm for the column subset selection problem. In *SODA*, 2009. Cited on pages 46, 47, and 48.
- [BN01] Mikhail Belkin and Partha Niyogi. Laplacian eigenmaps and spectral techniques for embedding and clustering. In *NIPS*, pages 585–591, 2001. Cited on pages 20, 152, 154, and 156.
- [BN07] Mikhail Belkin and Partha Niyogi. Convergence of laplacian eigenmaps. In *NIPS*, pages 129–136, 2007. Cited on pages 108, 154, and 156.
- [Bor19] Ali Borji. Pros and cons of gan evaluation measures. *Computer Vision and Image Understanding*, 179:41–65, 2019. Cited on pages 5 and 152.
- [Bou85] Jean Bourgain. On lipschitz embedding of finite metric spaces in hilbert space. *Israel Journal of Mathematics*, 1985. Cited on page 19.
- [BP98] Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual web search engine. *Computer Networks and ISDN Systems*, pages 107 – 117, 1998. Cited on page 33.
- [BSAG18] Miko Thlaj Bińkowski, Dougal J Sutherland, Michael Arbel, and Arthur Gretton. Demystifying MMD gans. In *ICLR*, 2018. Cited on pages 151, 152, and 153.
- [Bur76] Ronald S Burt. Positions in networks. *Social forces*, 1976. Cited on page 20.
- [BV04] Paolo Boldi and Sebastiano Vigna. The webgraph framework i: compression techniques. In *WWW*, 2004. Cited on page 9.
- [BZSL14] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. Spectral networks and locally connected networks on graphs. In *ICLR*, 2014. Cited on page 13.
- [CAEHP⁺20] Ines Chami, Sami Abu-El-Haija, Bryan Perozzi, Christopher Ré, and Kevin Murphy. Machine learning on graphs: A model and comprehensive taxonomy. *arXiv preprint arXiv:2005.03675*, 2020. Cited on page 13.
- [CAT16] Irineo Cabrerós, Emmanuel Abbe, and Aristotelis Tsigirós. Detecting community structures in hi-c genomic data. In *2016 Annual Conference on Information Science and Systems (CISS)*, pages 584–589. IEEE, 2016. Cited on page 83.
- [CCS09] Charles L Clarke, Nick Craswell, and Ian Soboroff. Overview of the TREC 2009 web track. Technical report, DTIC Document, 2009. Cited on pages 144 and 145.
- [CDART12] Stéphan Cléménçon, Hector De Arazoza, Fabrice Rossi, and Viet Chi Tran. Hierarchical clustering for graph visualization. *arXiv preprint arXiv:1210.5693*, 2012. Cited on page 83.
- [Chu97] Fan Chung. *Spectral graph theory*. Number 92. American Mathematical Soc., 1997. Cited on pages 11, 108, 109, and 112.

- [Chuo7] Fan Chung. The heat kernel as the pagerank of a graph. *Proceedings of the National Academy of Sciences*, 104(50):19735–19740, 2007. Cited on pages 108 and 111.
- [CKL⁺09] Flavio Chierichetti, Ravi Kumar, Silvio Lattanzi, Michael Mitzenmacher, Alessandro Panconesi, and Prabhakar Raghavan. On compressing social networks. In *KDD*, 2009. Cited on page 9.
- [CLo6] Ronald R Coifman and Stéphane Lafon. Diffusion maps. *Applied and computational harmonic analysis*, 2006. Cited on page 156.
- [CL11] Chih-Chung Chang and Chih-Jen Lin. LIBSVM: a library for support vector machines. *TIST*, 2(3):27, 2011. Cited on page 130.
- [CLRS09] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to algorithms*. MIT press, 2009. Cited on page 65.
- [CLS⁺19] Wei-Lin Chiang, Xuanqing Liu, Si Si, Yang Li, Samy Bengio, and Cho-Jui Hsieh. Cluster-GCN: An efficient algorithm for training deep and large graph convolutional networks. In *KDD*, 2019. Cited on pages 71 and 72.
- [CLV04] Fan Chung, Linyuan Lu, and Van Vu. The spectra of random graphs with given expected degrees. *Internet Mathematics*, 2004. Cited on page 163.
- [CLX15] Shaosheng Cao, Wei Lu, and Qionгкаi Xu. GraRep: Learning graph representations with global structural information. In *CIKM*, pages 891–900, 2015. Cited on pages 23, 25, 34, and 49.
- [CLX16] Shaosheng Cao, Wei Lu, and Qionгкаi Xu. Deep neural networks for learning graph representations. In *AAAI*, pages 1145–1152, 2016. Cited on page 25.
- [CLX19] Ren-Meng Cao, Si-Yuan Liu, and Xiao-Ke Xu. Network embedding for link prediction: The pitfall and improvement. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 2019. Cited on page 169.
- [COo7] Amin Coja-Oghlan. On the laplacian eigenvalues of $g(n, p)$. *Combinatorics, Probability and Computing*, 2007. Cited on page 163.
- [CPARS18] Haochen Chen, Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. A tutorial on network embeddings. *arXiv preprint arXiv:1808.02590*, 2018. Cited on page 62.
- [CR13] Ed S. Coakley and Vladimir Rokhlin. A fast divide-and-conquer algorithm for computing the spectra of real symmetric tridiagonal matrices. *Applied and Computational Harmonic Analysis*, 34(3):379 – 414, 2013. Cited on page 158.
- [CST⁺19] 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, 2019. Cited on pages 24, 64, and 70.
- [CW13] Kenneth L. Clarkson and David P. Woodruff. Low rank approximation and regression in input sparsity time. In *STOC*, 2013. Cited on page 48.
- [CW19] Chen Cai and Yusu Wang. A simple yet effective baseline for non-attribute graph classification. In *ICLR, RLGM workshop*, 2019. Cited on page 149.
- [CWLR19] Pin-Yu Chen, Lingfei Wu, Sijia Liu, and Indika Rajapakse. Fast incremental von neumann graph entropy computation: Theory, algorithm, and applications. In *ICML*, 2019. Cited on pages xiv, 133, 134, 135, 136, 144, 146, and 147.

- [CZQ⁺08] Weiwei Cui, Hong Zhou, Huamin Qu, Pak Chung Wong, and Xiaoming Li. Geometry-based edge clustering for graph visualization. *IEEE Transactions on Visualization and Computer Graphics*, 14(6):1277–1284, 2008. Cited on page 83.
- [DD03] Paul D Dobson and Andrew J Doig. Distinguishing enzyme structures from non-enzymes without alignments. *Journal of molecular biology*, pages 771–783, 2003. Cited on page 145.
- [DGM⁺11] Thomas A DiPrete, Andrew Gelman, Tyler McCormick, Julien Teitler, and Tian Zheng. Segregation in social networks based on acquaintanceship and trust. *American journal of sociology*, 2011. Cited on page 9.
- [Die95] Paul Dierckx. *Curve and surface fitting with splines*. Oxford University Press, 1995. Cited on page 129.
- [DJL⁺20] Vijay Prakash Dwivedi, Chaitanya K Joshi, Thomas Laurent, Yoshua Bengio, and Xavier Bresson. Benchmarking graph neural networks. *arXiv preprint arXiv:2003.00982*, 2020. Cited on page 91.
- [DKMZ11] Aurelien Decelle, Florent Krzakala, Cristopher Moore, and Lenka Zdeborová. Asymptotic analysis of the stochastic block model for modular networks and its algorithmic applications. *Physical Review E*, 84(6):066106, 2011. Cited on page 127.
- [DML11] Wei Dong, Charikar Moses, and Kai Li. Efficient k-nearest neighbor graph construction for generic similarity measures. In *WWW*, 2011. Cited on pages 158 and 164.
- [Dur12] Émile Durkheim. *Les formes élémentaires de la vie religieuse: le système totémique en Australie*. Alcan, 1912. Cited on page 20.
- [Ead84] Peter Eades. A heuristic for graph drawing. *Congressus numerantium*, 1984. Cited on page 20.
- [EHT65] Paul Erdős, Frank Harary, and William T Tutte. On the dimension of a graph. 1965. Cited on page 19.
- [ELPL17] Alessandro Epasto, Silvio Lattanzi, and Renato Paes Leme. Ego-splitting framework: From non-overlapping to overlapping clusters. In *KDD*, 2017. Cited on pages 85 and 96.
- [Est00] Ernesto Estrada. Characterization of 3d molecular structure. *Chemical Physics Letters*, pages 713–718, 2000. Cited on page 101.
- [Fau88] Katherine Faust. Comparison of methods for positional analysis: Structural and general equivalences. *Social networks*, 1988. Cited on page 20.
- [FCH⁺08] Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. Liblinear: A library for large linear classification. *JMLR*, 9(Aug):1871–1874, 2008. Cited on pages 36, 53, and 130.
- [FFF99] Michalis Faloutsos, Petros Faloutsos, and Christos Faloutsos. On power-law relationships of the internet topology. In *ACM SIGCOMM review*, 1999. Cited on page 9.
- [FH16] Santo Fortunato and Darko Hric. Community detection in networks: A user guide. *Physics reports*, 2016. Cited on pages 83, 85, and 91.
- [Fie73] Miroslav Fiedler. Algebraic connectivity of graphs. *Czechoslovak mathematical journal*, 1973. Cited on pages 10 and 85.

- [Fre78] Linton C Freeman. Centrality in social networks conceptual clarification. *Social networks*, 1978. Cited on page 10.
- [FSF⁺15] Andreas Fischer, Ching Y Suen, Volkmar Frinken, Kaspar Riesen, and Horst Bunke. Approximation of graph edit distance based on hausdorff matching. *Pattern Recognition*, 48(2):331–343, 2015. Cited on page 102.
- [GAA⁺17] Ishaan Gulrajani, Faruk Ahmed, Martín Arjovsky, Vincent Dumoulin, and Aaron C. Courville. Improved training of wasserstein GANs. In *NIPS*, 2017. Cited on page 163.
- [GBC16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT Press, 2016. Cited on page 28.
- [GBR⁺12] Arthur Gretton, Karsten M. Borgwardt, Malte J. Rasch, Bernhard Schölkopf, and Alexander J. Smola. A kernel two-sample test. *Journal of Machine Learning Research*, 2012. Cited on page 153.
- [GDMC10] Benjamin H Good, Yves-Alexandre De Montjoye, and Aaron Clauset. Performance of modularity maximization in practical contexts. *Physical Review E*, 81(4):046106, 2010. Cited on page 85.
- [GH10] Michael Gutmann and Aapo Hyvärinen. Noise-contrastive estimation: A new estimation principle for unnormalized statistical models. In *AISTATS*, pages 297–304, 2010. Cited on pages 28 and 64.
- [GH12] Michael U Gutmann and Aapo Hyvärinen. Noise-contrastive estimation of unnormalized statistical models, with applications to natural image statistics. *JMLR*, 13:307–361, 2012. Cited on pages 28 and 29.
- [GH16] Karam Gouda and Mosab Hassaan. Csi_ged: An efficient approach for graph edit similarity computation. In *ICDE*, pages 265–276, 2016. Cited on page 105.
- [GJ02] Michael R Garey and David S Johnson. *Computers and intractability*, volume 29. wh freeman New York, 2002. Cited on page 102.
- [GL16] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *KDD*, 2016. Cited on pages 29, 34, 35, 36, 37, 54, 70, and 71.
- [GLPW16] Mina Ghashami, Edo Liberty, Jeff M. Phillips, and David P. Woodruff. Frequent directions : Simple and deterministic matrix sketching. *SIAM J. Comput.*, 45:1762–1792, 2016. Cited on pages 45 and 51.
- [GM09] Gene H Golub and Gérard Meurant. *Matrices, moments and quadrature with applications*. Princeton University Press, 2009. Cited on pages 136, 137, and 138.
- [GMS05] Marco Gori, Gabriele Monfardini, and Franco Scarselli. A new model for learning in graph domains. In *IJCNN*, 2005. Cited on page 13.
- [GPAM⁺14] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *NIPS*, pages 2672–2680, 2014. Cited on page 153.
- [Gri06] Alexander Grigor’yan. Heat kernels on weighted manifolds and applications. In *Contemp. Math.*, pages 93–191. AMS, 2006. Cited on page 155.
- [GVL12] Gene H Golub and Charles F Van Loan. *Matrix computations*, volume 3. JHU Press, 2012. Cited on pages 20 and 143.
- [GW69] Gene H Golub and John H Welsch. Calculation of gauss quadrature rules. *Mathematics of computation*, 1969. Cited on page 138.

- [GXTL10] Xinbo Gao, Bing Xiao, Dacheng Tao, and Xuelong Li. A survey of graph edit distance. *Pattern Analysis and applications*, 2010. Cited on page 101.
- [HAL07] Matthias Hein, Jean-Yves Audibert, and Ulrike von Luxburg. Graph laplacians and their convergence on random neighborhood graphs. *JMLR*, 2007. Cited on pages 9 and 156.
- [HAMH16] Renjun Hu, Charu C Aggarwal, Shuai Ma, and Jinpeng Huai. An embedding approach to anomaly detection. In *ICDE*. IEEE, 2016. Cited on page 17.
- [HEHW12] Lin Han, Francisco Escolano, Edwin R Hancock, and Richard C Wilson. Graph characterizations from von Neumann entropy. *Pattern Recognition Letters*, pages 1958–1967, 2012. Cited on page 135.
- [HFLM⁺19] R Devon Hjelm, Alex Fedorov, Samuel Lavoie-Marchildon, Karan Grewal, Phil Bachman, Adam Trischler, and Yoshua Bengio. Learning deep representations by mutual information estimation and maximization. In *ICLR*, 2019. Cited on page 18.
- [HILG09] Michael P Holmes, Jr Isbell, Charles Lee, and Alexander G Gray. Quic-svd: Fast svd using cosine trees. In *NIPS*, pages 673–680, 2009. Cited on page 51.
- [HL97] Marlis Hochbruck and Christian Lubich. On krylov subspace approximations to the matrix exponential operator. *SIAM Journal on Numerical Analysis*, 1997. Cited on page 139.
- [HMT11] Nathan Halko, Per-Gunnar Martinsson, and Joel A Tropp. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM review*, 2011. Cited on page 11.
- [HN53] Frank Harary and Robert Z Norman. *Graph theory as a mathematical model in social science*. University of Michigan, Institute for Social Research, 1953. Cited on page 8.
- [HPK01] Jiawei Han, Jian Pei, and Micheline Kamber. *Data mining: concepts and techniques*. Elsevier, 2001. Cited on page 1.
- [HRH02] Peter D Hoff, Adrian E Raftery, and Mark S Handcock. Latent space approaches to social network analysis. *Journal of the american statistical association*, 2002. Cited on page 20.
- [HRU⁺17] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. GANs trained by a two time-scale update rule converge to a local nash equilibrium. In *NIPS*, 2017. Cited on pages 152 and 153.
- [HRV05] Vicente Hernandez, Jose E. Roman, and Vicente Vidal. SLEPc: A scalable and flexible toolkit for the solution of eigenvalue problems. *ACM Trans. Math. Software*, 31(3):351–362, 2005. Cited on page 111.
- [Hut89] MF Hutchinson. A stochastic estimator of the trace of the influence matrix for laplacian smoothing splines. *Communications in Statistics-Simulation and Computation*, 18(3):1059–1076, 1989. Cited on page 137.
- [HZ94] Geoffrey E Hinton and Richard S Zemel. Autoencoders, minimum description length and Helmholtz free energy. In *NIPS*, 1994. Cited on page 18.
- [HZRS16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016. Cited on page 13.
- [IM98] Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *STOC*, 1998. Cited on page 19.

- [Jaco1] Paul Jaccard. Étude comparative de la distribution florale dans une portion des alpes et des jura. *Bull. Soc. Vaudoise Sci. Nat.*, 37:547–579, 1901. Cited on page 24.
- [JF19] Simon Jenni and Paolo Favaro. On stabilizing generative adversarial training with noise. In *CVPR*, 2019. Cited on page 153.
- [JFW17] Minhao Jiang, Ada Wai-Chee Fu, and Raymond Chi-Wing Wong. Reads: a random walk approach for efficient and accurate dynamic simrank. *VLDB*, 10(9):937–948, 2017. Cited on page 30.
- [JL84] William B Johnson and Joram Lindenstrauss. Extensions of lipschitz mappings into a hilbert space. *Contemporary mathematics*, 26(189-206):1, 1984. Cited on page 19.
- [JLJ20] Yu Jin, Andreas Loukas, and Joseph JaJa. Graph coarsening with preserved spectral properties. In *AISTATS*, 2020. Cited on page 170.
- [JM12] Michael Jünger and Petra Mutzel. *Graph drawing software*. Springer Science & Business Media, 2012. Cited on page 20.
- [JWo2] Glen Jeh and Jennifer Widom. Simrank: a measure of structural-context similarity. In *KDD*, pages 538–543, 2002. Cited on pages 26 and 30.
- [JWo3] Glen Jeh and Jennifer Widom. Scaling personalized web search. In *WWW*, 2003. Cited on page 66.
- [KAS19] Megha Khosla, Avishek Anand, and Vinay Setty. A comprehensive comparison of unsupervised network representation learning methods. *arXiv preprint arXiv:1903.07902*, 2019. Cited on page 169.
- [Kat53] Leo Katz. A new status index derived from sociometric analysis. *Psychometrika*, 18(1):39–43, 1953. Cited on pages 23 and 34.
- [KB14] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. Cited on page 13.
- [KG14] Kyle Kloster and David F Gleich. Heat kernel based community detection. In *KDD*, 2014. Cited on page 118.
- [KJM20] Nils M Kriege, Fredrik D Johansson, and Christopher Morris. A survey on graph kernels. *Applied Network Science*, 2020. Cited on pages 101, 103, and 105.
- [KK11] Shrikant Kashyap and Panagiotis Karras. Scalable k NN search on vertically stored time series. In *KDD*, pages 1334–1342, 2011. Cited on page 114.
- [KKM⁺16] Kristian Kersting, Nils M. Kriege, Christopher Morris, Petra Mutzel, and Marion Neumann. Benchmark data sets for graph kernels, 2016. <http://graphkernels.cs.tu-dortmund.de>. Cited on pages 129, 145, and 146.
- [KL70] Brian W Kernighan and Shen Lin. An efficient heuristic procedure for partitioning graphs. *The Bell system technical journal*, 1970. Cited on page 86.
- [KN11] Brian Karrer and Mark EJ Newman. Stochastic blockmodels and community structure in networks. *Physical review E*, 83, 2011. Cited on pages 118, 124, and 125.
- [KNLH19] Simon Kornblith, Mohammad Norouzi, Honglak Lee, and Geoffrey Hinton. Similarity of neural network representations revisited. In *ICML*, 2019. Cited on page 154.

- [KO18] Valentin Khrulkov and Ivan V. Oseledets. Geometry score: A method for comparing generative adversarial networks. In *ICML*, 2018. Cited on pages 153 and 158.
- [KP16] Risi Kondor and Horace Pan. The multiscale laplacian graph kernel. In *NIPS*, pages 2990–2998, 2016. Cited on pages 103 and 129.
- [KUMH17] Günter Klambauer, Thomas Unterthiner, Andreas Mayr, and Sepp Hochreiter. Self-normalizing neural networks. In *NIPS*, 2017. Cited on pages 86 and 125.
- [KVF13] Danai Koutra, Joshua T Vogelstein, and Christos Faloutsos. Deltacon: A principled massive-graph similarity function. In *SDM*, pages 162–170, 2013. Cited on page 103.
- [KW17] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *ICLR*, 2017. Cited on pages 13 and 86.
- [KWG19] Johannes Klicpera, Stefan Weissenberger, and Stephan Günnemann. Diffusion improves graph learning. In *NeurIPS*, 2019. Cited on page 169.
- [LA17] Matthieu Labeau and Alexandre Allauzen. An experimental analysis of noise-contrastive estimation: the noise distribution matters. *EACL*, 2017. Cited on page 29.
- [LB14] Roei Litman and Alexander M. Bronstein. Learning spectral descriptors for deformable shape correspondence. *IEEE Trans. Pattern Anal. Mach. Intell.*, 36(1):171–180, 2014. Cited on page 110.
- [LBGS14] Peter A Lofgren, Siddhartha Banerjee, Ashish Goel, and C Seshadhri. Fastppr: Scaling personalized pagerank estimation for large graphs. In *KDD*, 2014. Cited on page 66.
- [LCC⁺17] Chun-Liang Li, Wei-Cheng Chang, Yu Cheng, Yiming Yang, and Barnabás Póczos. MMD GAN: Towards deeper understanding of moment matching network. In *NIPS*, 2017. Cited on page 152.
- [LG14] Omer Levy and Yoav Goldberg. Neural word embedding as implicit matrix factorization. In *NIPS*, 2014. Cited on pages 13, 23, and 49.
- [LHC06] Ping Li, Trevor J. Hastie, and Kenneth Ward Church. Very sparse random projections. In *KDD*, 2006. Cited on page 47.
- [Lib13] Edo Liberty. Simple and deterministic matrix sketching. In *KDD*, 2013. Cited on pages 45, 46, and 48.
- [Lin87] Nathan Linial. Distributive graph algorithms global solutions from local data. In *28th Annual Symposium on Foundations of Computer Science*, pages 331–335. IEEE, 1987. Cited on page 2.
- [Lin88] R. Linsker. Self-organization in a perceptual network. *IEEE Computer*, 21:105–117, 1988. Cited on page 18.
- [Lin94] Chih-Long Lin. Hardness of approximating graph transformation problem. In *ISAAC*, pages 74–82, 1994. Cited on pages 102 and 105.
- [LKF05] Jure Leskovec, Jon Kleinberg, and Christos Faloutsos. Graphs over time: densification laws, shrinking diameters and possible explanations. In *KDD*, 2005. Cited on page 146.
- [LLC10] Ryan N Lichtenwalter, Jake T Lussier, and Nitesh V Chawla. New perspectives and methods in link prediction. In *KDD*, pages 243–252, 2010. Cited on page 35.

- [LLDMo8] Jure Leskovec, Kevin J Lang, Anirban Dasgupta, and Michael W Mahoney. Statistical properties of community structure in large social and information networks. In *WWW*, 2008. Cited on pages 85 and 96.
- [Llo82] Stuart Lloyd. Least squares quantization in pcm. *IEEE transactions on information theory*, 1982. Cited on pages 21 and 90.
- [LLR95] Nathan Linial, Eran London, and Yuri Rabinovich. The geometry of graphs and some of its algorithmic applications. *Combinatorica*, 1995. Cited on page 19.
- [LPO17] David Lopez-Paz and Maxime Oquab. Revisiting classifier two-sample tests. In *ICLR*, 2017. Cited on page 153.
- [LV99] László Lovász and Katalin Vesztegombi. Geometric representations of graphs. *Paul Erdős and his Mathematics*, 1999. Cited on page 19.
- [LZ11] Linyuan Lü and Tao Zhou. Link prediction in complex networks: A survey. *Physica A: Statistical Mechanics and its Applications*, 390(6):1150–1170, 2011. Cited on pages 29 and 35.
- [LZ17] Yongjiang Liang and Peixiang Zhao. Similarity search in graph databases: A multi-layered indexing approach. In *ICDE*, pages 783–794, 2017. Cited on pages 102 and 105.
- [mag16] Microsoft academic graph (mag) - kkd cup 2016. <https://www.kdd.org/kdd-cup/view/kdd-cup-2016/Data>, 2016. Cited on pages 36, 54, 71, and 72.
- [Mah11] Matt Mahoney. Large text compression benchmark. <http://www.mattmahoney.net/text/text.html>, 2011. Cited on page 54.
- [Mém11] Facundo Mémoli. A spectral notion of gromov–wasserstein distance and related methods. *Applied and Computational Harmonic Analysis*, 30(3):363–401, 2011. Cited on pages 113, 152, 155, 156, and 157.
- [MG82] Jayaved Misra and David Gries. Finding repeated elements. *Science of Computer Programming*, 1982. Cited on page 48.
- [MHSG18] Leland McInnes, John Healy, Nathaniel Saul, and Lukas Großberger. Umap: Uniform manifold approximation and projection. *Journal of Open Source Software*, 2018. Cited on page 75.
- [Mit80] Tom M Mitchell. The need for biases in learning generalizations. 1980. Cited on page 1.
- [MKKM16] Christopher Morris, Nils M Kriege, Kristian Kersting, and Petra Mutzel. Faster kernels for graphs with continuous attributes via hashing. In *ICDM*, 2016. Cited on page 103.
- [MLDB20] Alexandru Cristian Mara, Jeffrey Lijffijt, and Tijl De Bie. Benchmarking network embedding models for link prediction: are we making progress? In *DSAA*, 2020. Cited on page 169.
- [MLS⁺18] Youssef Mroueh, Chun-Liang Li, Tom Sercu, Anant Raj, and Yu Cheng. Sobolev GAN. In *ICLR*, 2018. Cited on page 153.
- [MM90] David JC MacKay and Kenneth D Miller. Analysis of Linsker’s simulations of Hebbian rules. In *NIPS*, 1990. Cited on page 18.

- [MM15] Cameron Musco and Christopher Musco. Randomized block krylov methods for stronger and faster approximate value decomposition. In *NIPS*, 2015. Cited on page 11.
- [Mor34] Jacob Levy Moreno. Who shall survive?: A new approach to the problem of human interrelations. 1934. Cited on page 19.
- [MPo7] Sofus A Macskassy and Foster Provost. Classification in networked data: A toolkit and a univariate case study. *JMLR*, 2007. Cited on page 17.
- [MRB18] Ari Morcos, Maithra Raghu, and Samy Bengio. Insights on representational similarity in neural networks with canonical correlation. In *NeurIPS*, 2018. Cited on pages 151 and 154.
- [MRT18] Giorgia Minello, Luca Rossi, and Andrea Torsello. On the von Neumann entropy of graphs. *Journal of Complex Networks*, 2018. Cited on page 135.
- [MSC⁺13] Tomas Mikolov, Ilya Sutskever, Kai Chen, Gregory S. Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality. In *NIPS*, 2013. Cited on pages 13, 21, and 64.
- [MT12] Andriy Mnih and Yee Whye Teh. A fast and simple algorithm for training neural probabilistic language models. In *ICML*, pages 1751–1758, 2012. Cited on pages 28 and 29.
- [MVL03] Cleve Moler and Charles Van Loan. Nineteen dubious ways to compute the exponential of a matrix, twenty-five years later. *SIAM review*, 45(1):3–49, 2003. Cited on pages 110, 111, and 135.
- [NAK16] Mathias Niepert, Mohamed Ahmed, and Konstantin Kutzkov. Learning convolutional neural networks for graphs. In *ICML*, 2016. Cited on page 123.
- [Newo6a] Mark EJ Newman. Finding community structure in networks using the eigenvectors of matrices. *Physical review E*, 2006. Cited on pages 21 and 86.
- [Newo6b] Mark EJ Newman. Modularity and community structure in networks. *PNAS*, pages 8577–8582, 2006. Cited on pages xvi, 21, 35, 40, 85, and 91.
- [NM10] Hariharan Narayanan and Sanjoy Mitter. Sample complexity of testing the manifold hypothesis. In *NIPS*, 2010. Cited on page 154.
- [NN12] Raj Rao Nadakuditi and Mark EJ Newman. Graph spectra and the detectability of community structure in networks. *Physical review letters*, 2012. Cited on page 92.
- [OBO⁺18] Augustus Odena, Jacob Buckman, Catherine Olsson, Tom B. Brown, Christopher Olah, Colin A. Raffel, and Ian J. Goodfellow. Is generator conditioning causally related to GAN performance? In *ICML*, 2018. Cited on page 153.
- [OCP⁺16] Mingdong Ou, Peng Cui, Jian Pei, Ziwei Zhang, and Wenwu Zhu. Asymmetric transitivity preserving graph embedding. In *KDD*, 2016. Cited on page 34.
- [PA16] Bryan Perozzi and Leman Akoglu. Scalable anomaly ranking of attributed neighborhoods. In *SDM*, 2016. Cited on page 83.
- [PA18] Bryan Perozzi and Leman Akoglu. Discovering communities and anomalies in attributed graphs: Interactive visual exploration and summarization. *ACM TKDE*, 12(2), 2018. Cited on page 83.
- [PAISM14] Bryan Perozzi, Leman Akoglu, Patricia Iglesias Sánchez, and Emmanuel Müller. Focused clustering and outlier detection in large attributed graphs. In *KDD*, 2014. Cited on page 83.

- [PARS14] Bryan Perozzi, Rami Al-Rfou', and Steven Skiena. Deepwalk: online learning of social representations. In *KDD*, 2014. Cited on pages 13, 17, 21, 25, 34, 35, 36, 52, 57, 61, 70, 74, and 90.
- [PBMW99] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: bringing order to the web. 1999. Cited on pages 3, 10, 23, 26, 29, and 49.
- [PDGM10] Panagiotis Papadimitriou, Ali Dasdan, and Hector Garcia-Molina. Web graph similarity for anomaly detection. *Journal of Internet Services and Applications*, pages 19–30, 2010. Cited on page 103.
- [Pei14a] Tiago P Peixoto. Efficient monte carlo and greedy heuristic for the inference of stochastic block models. *Physical Review E*, 2014. Cited on page 117.
- [Pei14b] Tiago P. Peixoto. The graph-tool python library. *figshare*, 2014. Cited on pages 90, 92, and 115.
- [PK19] Sung Woo Park and Junseok Kwon. Sphere generative adversarial network based on geometric moment matching. In *CVPR*, 2019. Cited on pages 152 and 153.
- [PLVT⁺20] Léo Pio-Lopez, Alberto Valdeolivas, Laurent Tichit, Élisabeth Remy, and Anaïs Baudot. MultiVERSE: a multiplex and multiplex-heterogeneous network embedding approach. *arXiv preprint arXiv:2008.10085*, 2020. Cited on page 169.
- [PTdA⁺20] Ștefan Postăvaru, Anton Tsitsulin, Filipe Miguel Gonçalves de Almeida, Yingtao Tian, Silvio Lattanzi, and Bryan Perozzi. InstantEmbedding: Efficient local node representations. *arXiv preprint arXiv:2010.06992*, 2020. Cited on page 18.
- [PVG⁺11] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *JMLR*, 2011. Cited on page 73.
- [QDM⁺18] Jiezhong Qiu, Yuxiao Dong, Hao Ma, Jian Li, Kuansan Wang, and Jie Tang. Network embedding as matrix factorization: Unifying DeepWalk, LINE, PTE, and node2vec. In *WSDM*, 2018. Cited on pages 4, 23, 49, 50, 53, and 62.
- [QDM⁺19] Jiezhong Qiu, Yuxiao Dong, Hao Ma, Jian Li, Chi Wang, Kuansan Wang, and Jie Tang. Netsmf: Large-scale network embedding as sparse matrix factorization. In *WWW*, 2019. Cited on pages 23 and 53.
- [RA15] Ryan A. Rossi and Nesreen K. Ahmed. The network data repository with interactive graph analytics and visualization. In *AAAI*, 2015. Cited on page 144.
- [RB09] Kaspar Riesen and Horst Bunke. Approximate graph edit distance computation by means of bipartite graph matching. *Image and Vision computing*, 27(7):950–959, 2009. Cited on page 102.
- [RGYSD17] Maithra Raghu, Justin Gilmer, Jason Yosinski, and Jascha Sohl-Dickstein. SVCCA: Singular vector canonical correlation analysis for deep learning dynamics and interpretability. In *NIPS*, 2017. Cited on page 154.
- [RKA15] Farbod Roosta-Khorasani and Uri Ascher. Improved bounds on sample size for implicit matrix trace estimators. *Foundations of Computational Mathematics*, 2015. Cited on pages 139 and 143.

- [Ros58] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 1958. Cited on page 12.
- [RS00] Sam T Roweis and Lawrence K Saul. Nonlinear dimensionality reduction by locally linear embedding. *Science*, 2000. Cited on page 20.
- [ŘS10] Radim Řehůřek and Petr Sojka. Software Framework for Topic Modelling with Large Corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, 2010. Cited on page 159.
- [RSF17] Leonardo FR Ribeiro, Pedro HP Saverese, and Daniel R Figueiredo. struc2vec: Learning node representations from structural identity. In *KDD*, 2017. Cited on page 114.
- [Saa92] Youcef Saad. *Numerical methods for large eigenvalue problems*. Manchester University Press, 1992. Cited on page 20.
- [SB09] Nino Shervashidze and Karsten M Borgwardt. Fast subtree kernels on graphs. In *NIPS*, 2009. Cited on page 103.
- [SBR⁺06] Chris Stark, Bobby-Joe Breitkreutz, Teresa Reguly, Lorrie Boucher, Ashton Breitkreutz, and Mike Tyers. BioGRID: a general repository for interaction datasets. *Nucleic Acids Research*, 2006. Cited on pages 54 and 71.
- [SCD18] David E Simmons, Justin P Coon, and Animesh Datta. The von Neumann Theil index: characterizing graph centralization using the von Neumann index. *Journal of Complex Networks*, 2018. Cited on page 135.
- [SDT19] Hooman Peiro Sajjad, Andrew Docherty, and Yuriy Tyshetskiy. Efficient representation learning using random walks for dynamic graphs. *arXiv preprint arXiv:1901.01346*, 2019. Cited on page 169.
- [SF83] Alberto Sanfeliu and King-Sun Fu. A distance measure between attributed relational graphs for pattern recognition. *IEEE Trans. Systems, Man, and Cybernetics*, 13(3):353–362, 1983. Cited on pages 102 and 105.
- [SGZ⁺16] Tim Salimans, Ian J. Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training GANs. In *NIPS*, 2016. Cited on page 152.
- [SHK⁺14] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *JMLR*, 2014. Cited on page 89.
- [SM00] Jianbo Shi and Jitendra Malik. Normalized cuts and image segmentation. *PAMI*, 2000. Cited on pages 12, 85, 108, 111, and 118.
- [SMBG18] Oleksandr Shchur, Maximilian Mumme, Aleksandar Bojchevski, and Stephan Günnemann. Pitfalls of graph neural network evaluation. *arXiv preprint arXiv:1811.05868*, 2018. Cited on pages xviii, 86, 90, and 96.
- [SN97] Tom AB Snijders and Krzysztof Nowicki. Estimation and prediction for stochastic blockmodels for graphs with latent block structure. *Journal of classification*, 1997. Cited on page 91.
- [SNB⁺08] Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad. Collective classification in network data. *AI magazine*, 2008. Cited on pages xvii, 89, and 95.

- [SOG09] Jian Sun, Maks Ovsjanikov, and Leonidas Guibas. A concise and provably informative multi-scale signature based on heat diffusion. In *Computer graphics forum*, volume 28, pages 1383–1392. Wiley Online Library, 2009. Cited on pages 110, 154, and 155.
- [Sor27] Pitirim Aleksandrovich Sorokin. *Social mobility*. Harper & Row, 1927. Cited on page 19.
- [SSL⁺11] Nino Shervashidze, Pascal Schweitzer, Erik Jan van Leeuwen, Kurt Mehlhorn, and Karsten M Borgwardt. Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research*, 12(Sep):2539–2561, 2011. Cited on pages 103 and 145.
- [SW19] Till Schulz and Pascal Welke. On the necessity of graph kernel baselines. In *ECML-PKDD, GEM workshop*, 2019. Cited on page 149.
- [SZ15] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2015. Cited on page 160.
- [TDSL00] Joshua B Tenenbaum, Vin De Silva, and John C Langford. A global geometric framework for nonlinear dimensionality reduction. *Science*, 2000. Cited on page 20.
- [THJ10] Daniel Ting, Ling Huang, and Michael Jordan. An analysis of the convergence of graph laplacians. In *ICML*, 2010. Cited on pages 156 and 158.
- [TKo6] Grigorios Tsoumakas and Ioannis Katakis. Multi-label classification: An overview. *IJDWM*, 3(3), 2006. Cited on page 36.
- [TL09a] Lei Tang and Huan Liu. Relational learning via latent social dimensions. In *KDD*, pages 817–826, 2009. Cited on pages 21 and 25.
- [TL09b] Lei Tang and Huan Liu. Scalable learning of collective behavior based on sparse social dimensions. In *CIKM*, 2009. Cited on pages 21, 36, and 54.
- [TL10] Lei Tang and Huan Liu. Social dimension approach to classification in large-scale networks. 2010. Cited on pages 71 and 72.
- [TMC⁺19] Ke Tu, Jianxin Ma, Peng Cui, Jian Pei, and Wenwu Zhu. AutoNE: Hyperparameter optimization for massive network embedding. In *KDD*, 2019. Cited on page 170.
- [TMK⁺18a] Anton Tsitsulin, Davide Mottin, Panagiotis Karras, Alex Bronstein, and Emmanuel Müller. SGR: Self-supervised spectral graph representation learning. *arXiv preprint arXiv:1811.06237*, 2018. Cited on page 102.
- [TMK⁺18b] Anton Tsitsulin, Davide Mottin, Panagiotis Karras, Alexander M. Bronstein, and Emmanuel Müller. NetLSD: Hearing the shape of a graph. In *KDD*, 2018. Cited on pages 102, 129, 135, 144, 147, and 163.
- [TMKM18] Anton Tsitsulin, Davide Mottin, Panagiotis Karras, and Emmanuel Müller. VERSE: Versatile graph embeddings from similarity measures. In *WWW*, 2018. Cited on pages 18, 52, 54, 70, 72, and 114.
- [TMM⁺20a] Anton Tsitsulin, Marina Munkhoeva, Davide Mottin, Panagiotis Karras, Alex Bronstein, Ivan Oseledets, and Emmanuel Müller. The shape of data: Intrinsic distance for data distributions. In *ICLR*, 2020. Cited on page 102.
- [TMM⁺20b] Anton Tsitsulin, Marina Munkhoeva, Davide Mottin, Panagiotis Karras, Ivan Oseledets, and Emmanuel Müller. Frede: Linear-space anytime graph embeddings. *arXiv preprint arXiv:2006.04746*, 2020. Cited on pages 18 and 62.

- [TMP20] Anton Tsitsulin, Marina Munkhoeva, and Bryan Perozzi. Just SLaQ when you approximate: Accurate spectral distances for web-scale graphs. *arXiv preprint arXiv:2003.01282*, 2020. Cited on page 102.
- [TPPM20] Anton Tsitsulin, John Palowitch, Bryan Perozzi, and Emmanuel Müller. Graph clustering with graph neural networks. *arXiv preprint arXiv:2006.16904*, 2020. Cited on page 18.
- [TQW⁺15] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. LINE: Large-scale information network embedding. In *WWW*, 2015. Cited on pages 22, 25, 34, 35, 36, and 57.
- [TYSK09] Yufei Tao, Ke Yi, Cheng Sheng, and Panos Kalnis. Quality and efficiency in high dimensional nearest neighbor search. In *SIGMOD*, pages 563–576, 2009. Cited on page 114.
- [TZ05] Mikkel Thorup and Uri Zwick. Approximate distance oracles. *JACM*, 52(1):1–24, 2005. Cited on page 45.
- [UCS17] Shashanka Ubaru, Jie Chen, and Yousef Saad. Fast estimation of $\text{tr}(f(a))$ via stochastic lanczos quadrature. *SIAM Journal on Matrix Analysis and Applications*, 2017. Cited on pages 133, 136, and 140.
- [VBCG10] Amir Vaxman, Mirela Ben-Chen, and Craig Gotsman. A multi-resolution approach to heat kernels on discrete surfaces. In *TOG*, volume 29, page 121, 2010. Cited on pages 110, 111, and 136.
- [Vem05] Santosh S Vempala. *The random projection method*. American Math. Soc., 2005. Cited on page 48.
- [VFH⁺19] Petar Veličković, William Fedus, William L Hamilton, Pietro Liò, Yoshua Bengio, and R Devon Hjelm. Deep graph infomax. In *ICLR*, 2019. Cited on page 90.
- [vHo8] L.J.P. van der Maaten and G.E. Hinton. Visualizing high-dimensional data using t-SNE. *JMLR*, 2008. Cited on page 43.
- [VL07] Ulrike Von Luxburg. A tutorial on spectral clustering. *Statistics and computing*, pages 395–416, 2007. Cited on page 12.
- [VLBB08] Ulrike Von Luxburg, Mikhail Belkin, and Olivier Bousquet. Consistency of spectral clustering. *The Annals of Statistics*, pages 555–586, 2008. Cited on page 108.
- [VN32] John Von Neumann. *Mathematische grundlagen der quantenmechanik*. Springer-Verlag, 1932. Cited on page 134.
- [VZ17] Saurabh Verma and Zhi-Li Zhang. Hunt for the unique, stable, sparse and fast feature learning on graphs. In *NIPS*, pages 87–97, 2017. Cited on pages 104, 114, 115, and 129.
- [WC89] Yen-Chuen Wei and Chung-Kuan Cheng. Towards efficient hierarchical designs by ratio cut partitioning. In *IEEE International Conference on Computer-Aided Design*. IEEE, 1989. Cited on page 85.
- [WCZ16] Daixin Wang, Peng Cui, and Wenwu Zhu. Structural deep network embedding. In *KDD*, pages 1225–1234, 2016. Cited on page 25.
- [WDL⁺09] Kilian Weinberger, Anirban Dasgupta, John Langford, Alex Smola, and Josh Attenberg. Feature hashing for large scale multitask learning. In *ICML*, 2009. Cited on pages 62 and 65.

- [Wer82] Paul J Werbos. Applications of advances in nonlinear sensitivity analysis. In *System modeling and optimization*. Springer, 1982. Cited on page 13.
- [Wey11] Hermann Weyl. Über die asymptotische verteilung der eigenwerte. *Nachrichten von der Gesellschaft der Wissenschaften zu Göttingen*, 1911. Cited on pages 111 and 164.
- [WHG⁺18] Liwei Wang, Lunjia Hu, Jiayuan Gu, Zhiqiang Hu, Yue Wu, Kun He, and John Hopcroft. Towards understanding learning representations: To what extent do different neural networks learn the same representation. In *NeurIPS*, 2018. Cited on pages 151 and 154.
- [Wil62] Herbert S Wilf. Mathematics for the physical sciences. 1962. Cited on page 138.
- [Wol] Milian Wolff. A heap memory profiler for linux, 2018. <https://github.com/KDE/heaptrack>. Cited on page 73.
- [Wol96] David H Wolpert. The lack of a priori distinctions between learning algorithms. *Neural computation*, 1996. Cited on pages 5 and 17.
- [Woo14] David P Woodruff. Sketching as a tool for numerical linear algebra. *Theoretical Computer Science*, 2014. Cited on pages 19, 24, 46, and 47.
- [WS98] Duncan J Watts and Steven H Strogatz. Collective dynamics of ‘small-world’ networks. *Nature*, 393:440–442, 1998. Cited on page 43.
- [YJK18] Minji Yoon, Jinhong Jung, and U Kang. Tpa: Fast, scalable, and accurate method for approximate random walk with restart on billion scale graphs. In *ICDE*, pages 1132–1143. IEEE, 2018. Cited on page 51.
- [YL15] Jaewon Yang and Jure Leskovec. Defining and evaluating network communities based on ground-truth. *Knowledge and Information Systems*, 2015. Cited on pages 36, 61, 71, 72, 91, and 144.
- [YS18] Zi Yin and Yuanyuan Shen. On the dimensionality of word embedding. In *NeurIPS*, 2018. Cited on pages 154 and 160.
- [YSX⁺20] Renchi Yang, Jieming Shi, Xiaokui Xiao, Yin Yang, and Sourav S. Bhowmick. Homogeneous network embedding for massive graphs via reweighted personalized pagerank. *PVLDB*, 13(5):670–683, 2020. Cited on pages 49 and 51.
- [YV15] Pinar Yanardag and SVN Vishwanathan. Deep graph kernels. In *KDD*, pages 1365–1374, 2015. Cited on pages 116 and 146.
- [YW19] Yuan Yin and Zhewei Wei. Scalable graph embeddings via sparse transpose proximities. In *KDD*, 2019. Cited on page 169.
- [YWCW15] Ye Yuan, Guoren Wang, Lei Chen, and Haixun Wang. Graph similarity search on large uncertain graph databases. *The VLDB Journal*, 24(2):271–296, 2015. Cited on page 105.
- [YYM⁺18] Zhitao Ying, Jiaxuan You, Christopher Morris, Xiang Ren, Will Hamilton, and Jure Leskovec. Hierarchical graph representation learning with differentiable pooling. In *NeurIPS*, 2018. Cited on page 90.
- [Zac77] Wayne W Zachary. An information flow model for conflict and fission in small groups. *Journal of anthropological research*, 33(4):452–473, 1977. Cited on pages xi and 27.
- [ZCL⁺18] Ziwei Zhang, Peng Cui, Haoyang Li, Xiao Wang, and Wenwu Zhu. Billion-scale network embedding with iterative random projection. In *ICDM*, pages 787–796, 2018. Cited on pages 24, 48, 53, and 64.

- [ZCW⁺18] Ziwei Zhang, Peng Cui, Xiao Wang, Jian Pei, Xuanrong Yao, and Wenwu Zhu. Arbitrary-order proximity preserved network embedding. In *KDD*, 2018. Cited on page 23.
- [Zil96] Shlomo Zilberstein. Using anytime algorithms in intelligent systems. *AI Magazine*, 17(3):73–83, 1996. Cited on pages 4 and 45.
- [ZLo9] R. Zafarani and H. Liu. Social computing data repository at ASU, 2009. Cited on pages 36 and 54.
- [ZQYC12] Yuanyuan Zhu, Lu Qin, Jeffrey Xu Yu, and Hong Cheng. Finding top-k similar graphs in graph databases. In *EDBT*, pages 456–467, 2012. Cited on page 102.
- [ZTW⁺09] Zhiping Zeng, Anthony K. H. Tung, Jianyong Wang, Jianhua Feng, and Lizhu Zhou. Comparing stars: On approximating graph edit distance. *PVLDB*, 2(1):25–36, 2009. Cited on pages 102 and 105.
- [ZXTQ19] Zhaocheng Zhu, Shizhen Xu, Jian Tang, and Meng Qu. Graphvite: A high-performance CPU–GPU hybrid system for node embedding. In *WWW*, 2019. Cited on page 169.
- [ZZL⁺15] Weiguo Zheng, Lei Zou, Xiang Lian, Dong Wang, and Dongyan Zhao. Efficient graph similarity search over large graph databases. *TKDE*, 27(4):964–978, 2015. Cited on page 105.

PUBLICATIONS

1. Anton Tsitsulin, Davide Mottin, Panagiotis Karras, and Emmanuel Müller. *VERSE: Versatile Graph Embeddings from Similarity Measures*. WWW, 2018.
2. Anton Tsitsulin, Davide Mottin, Panagiotis Karras, Alex Bronstein, and Emmanuel Müller. *NetLSD: Hearing the Shape of a Graph*. KDD, 2018.
3. Anton Tsitsulin, Davide Mottin, Panagiotis Karras, Alex Bronstein, and Emmanuel Müller. *Spectral Graph Complexity*. WWW companion, 2019.
4. Anton Tsitsulin, Davide Mottin, Panagiotis Karras, Alex Bronstein, and Emmanuel Müller. *SGR: Self-Supervised Spectral Graph Representation Learning*. CoRR abs/1811.06237, 2018.
5. Anton Tsitsulin, Marina Munkhoeva, and Bryan Perozzi. *Just SLaQ When You Approximate: Accurate Spectral Distances for Web-Scale Graphs*. WWW, 2020.
6. Anton Tsitsulin, Marina Munkhoeva, Davide Mottin, Panagiotis Karras, Alex Bronstein, Ivan Oseledets, and Emmanuel Müller. *The Shape of Data: Intrinsic Distance for Data Distributions*. ICLR, 2020.
7. Anton Tsitsulin, Marina Munkhoeva, Davide Mottin, Panagiotis Karras, Ivan Oseledets, and Emmanuel Müller. *FREDE: Anytime Graph Embeddings*. VLDB, 2021.
8. Ștefan Postăvaru, Anton Tsitsulin, Filipe Miguel Gonçalves de Almeida, Yingtao Tian, Silvio Lattanzi, and Bryan Perozzi. *InstantEmbedding: Efficient Local Graph Representation*. CoRR abs/2010.06992, 2020.
9. Anton Tsitsulin, John Palowitch, Bryan Perozzi, and Emmanuel Müller. *Graph Clustering with Graph Neural Networks*. CoRR abs/2006.16904, 2020.