Unsupervised Structural Embedding Methods for Efficient Collective Network Mining

by

Mark Heimann

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Computer Science and Engineering)
in The University of Michigan
2020

Doctoral Committee:

Assistant Professor Danai Koutra, Chair
Professor Alfred O. Hero III
Dr. Ramakrishnan Kannan, Oak Ridge National Laboratory
Professor Seth Pettie

Mark Heimann
mheimann@umich.edu
ORCID iD: 0000-0002-3082-863X

ACKNOWLEDGEMENTS

I am deeply indebted to Danai Koutra, who has been absolutely everything I could ask for in an advisor. Danai has shown me what it means to be a researcher: how to formulate and evaluate ideas, write papers and give presentations for conferences and interviews. Throughout my PhD she has provided a perfect mix of giving me the space to explore ideas while always providing guidance as specific as needed (down to wording choices or minutiae of figures). She has shown me by example how to maintain grit when a project is stuck, a big deadline looms large, or a paper is rejected–she is supportive when research is going well and when it isn't. I aspire to one day be the kind of mentor to others that Danai has been has been to me.

I would like to thank the other members of my committee, Seth Pettie, Al Hero, and Ramki Kannan, for their insightful feedback at my proposal and leading up to my defense. I am additionally grateful to Ramki for mentoring me during a great summer internship at Oak Ridge National Laboratory and beyond, which gave me a greater vision of what I could be as a researcher. I would also like to thank Emilio Ferrara at the Information Sciences Institute for mentoring me during another wonderful summer internship that gave me a chance to explore connections of my work to social science, and for fostering a collaborative environment that sparked several new research ideas that I have been able to work on since. Similarly, I am also grateful to other professors who were supportive to me at the beginning of graduate school, in particular Jacob Abernethy and Grant Schoenebeck.

GEMS Lab, it has been such a privilege to be your collaborator and labmate over the years, and I can't even begin to say how awed I am by all your talent and accomplishments. I am indebted to Tara Safavi, Di Jin, Yujun Yan, Fatemeh Vahedian, and Jiong Zhu for the privilege of coauthoring with you—I've learned a great deal from you and my work

would not be what it is without you. Caleb Belth, Marlena Duda, Alican Büyükçakır, and Puja Trivedi, I am so impressed with the broad intellectual command that you each have demonstrated and I can't wait to see the extremely bright future that is in store for the lab. I am also thankful to the undergraduate and masters' students I have been able to work with, particularly Xiyuan Chen, Junchen (Mark) Jin, Haoming Shen, and Wei Lee: you've made my job easy and in fact have taught me so much.

To my friends from GradCru, the Graduate Christian Fellowship, and the wonderful Tuesday night dinners hosted by the truly saintly George and Mary Linquist, thank you for the friendship and support that so many of you have provided over the years. I could list a long list of names of people, but I would especially like to thank Deanna Montgomery, Ryan Hayes, Joe Iafrate, Dayna Appiah, Daniel Whitford, and Ritz Raju. I have learned a lot from each of you and am grateful that we have shared good times and hard times of grad school and life. Alex Peplinski and Harsh Bhavsar, we've been housemates for several years, and I very much respect both of you as people and have enjoyed making all kinds of food (from paczkis to palak paneer) and having all kinds of enriching conversations and adventures with you.

As a graduate student, I have also had other notable activities that have enriched my life. I am grateful to the other members of the University of Michigan chess team, with whom I had the opportunity to represent the school twice at collegiate nationals (before I became too old) and attend several other tournaments (I'd especially like to commend the four troopers who drove with me nine hours one way in the middle of July, with questionable climate control in my 1992 Subaru Legacy, to play in the World Open). The unlikely pursuit of my time in Michigan, working out and competitive powerlifting, has provided me with unforgettable experiences and life lessons that I never saw coming prior to grad school. I am grateful to Alex Crichton and Gina Hensley, who have always gone above and beyond to put on well-organized, friendly, and inclusive USA Powerlifting meets in Michigan. Thank you also to the friends I made in the University of Michigan gyms and at meets.

Finally, thank you to my parents and brother for being, through every high and low of the last five years, the most loving and supportive family I could ask for.

TABLE OF CONTENTS

LIST OF FIGURES

Figure

LIST OF TABLES

Table

ABSTRACT

How can we align accounts of the same user across social networks? Can we identify the professional role of an email user from their patterns of communication? Can we predict the medical effects of chemical compounds from their atomic network structure? Many problems in graph data mining, including all of the above, are defined on multiple networks. The central element to all of these problems is cross-network comparison, whether at the level of individual nodes or entities in the network or at the level of entire networks themselves. To perform this comparison meaningfully, we must describe the entities in each network expressively in terms of patterns that generalize across the networks. Moreover, because the networks in question are often very large, our techniques must be computationally efficient.

In this thesis, we propose scalable unsupervised methods that embed nodes in vector space by mapping nodes with *similar structural roles* in their respective networks, even if they come from different networks, to similar parts of the embedding space. We perform *network alignment* by matching nodes across two or more networks based on the similarity of their embeddings, and refine this process by reinforcing the consistency of each node's alignment with those of its neighbors. By characterizing the distribution of node embeddings in a graph, we develop graph-level feature vectors that are highly effective for *graph classification*. With principled sparsification and randomized approximation techniques, we make all our methods computationally efficient and able to scale to graphs with millions of nodes or edges. We demonstrate the effectiveness of structural node embeddings on industry-scale applications, and propose an extensive set of embedding evaluation techniques that lay the groundwork for further methodological development and application.

CHAPTER I

Introduction

Graphs or networks are a natural structure for modeling complex connections between all kinds of entities, whether users making friendships in social networks, scientists collaborating on academic projects, or atoms forming bonds in chemical compounds. The central goal of graph mining is to uncover meaningful insights about the network entities in terms of the connections they form.

To model entities or nodes in a graph, methods for node embedding or representation learning have risen to prominence in recent times due to their success in many graph mining tasks. These methods embed each node in a graph in vector space, so that the node can be compactly represented by a low-dimensional feature vector. For maximum applicability to any data mining task, these methods may be unsupervised: their objective is not to directly learn embeddings that maximize performance on a particular task, but to preserve similarity in the embedding space for nodes or entities whose connections are similar in the graph. Most commonly, connection similarity is taken to mean that the entities themselves are connected, indirectly or directly. For example, nodes that are connected by an edge, or are in indirect proximity through shared neighboring nodes, would be assumed to be similar and would have similar feature representations after embedding.

Such embedding methods have modeled the interactions within a single graph very precisely. They excel at comparing nodes within individual networks and often achieve the state of the art on learning tasks defined over a single network. But where do such methods leave us in a *collective* graph mining context with *multiple* completely separate networks? Nodes in different networks do not share connections with nodes in other networks, nor do any of

their neighbors: whether direct or indirect, there is no notion of cross-network proximity to which we can turn to guide the embedding process.

Our solution is to reconsider what it means for nodes' connections to be similar. Instead of assuming that the nodes share a connection—with each other or with mutual neighbors–we characterize nodes' patterns of connectivity in their respective graphs. We can think of this as the *structural role* that each node plays in its own network. For instance, two nodes that tend to make large numbers of connections have a similar role, even if those connections are not with each other. As proximity-preserving embedding methods do, we can learn from not just nodes' immediate connections but also their neighbors' connections, and more general higher-order patterns of connectivity.

These *structural* node embeddings form the underpinnings of this thesis work. Collective network mining hinges on the problem of cross-network comparison: while this cannot be done meaningfully modeling the nonexistent proximity between nodes in different networks, it is meaningful to compare nodes based on their structural roles. Our methods consist of two parts: an embedding learning phase where we model the structural roles of nodes using embeddings, and an embedding comparison phase where we perform the cross-network comparison using the features learned in the previous phase. To be effective in the comparison stage, our goal is to model structural roles expressively in the embedding learning stage, so that we capture each node's distinctive structural role in its own network precisely while remaining generalizable to different networks.

A key consideration in all our work is computational efficiency, which is necessary for our methods to scale to large networks. Whether within or across networks, a natural approach to determining node similarity is to compare all pairs of nodes. This automatically leads to a requirement of runtime (and possibly space) that scales quadratically in the number of nodes, which limits the applicability of any method using such a subroutine to large real-world graphs, which may contain millions of nodes. A theme throughout our work, both in the embedding learning and the embedding comparison phase, is to circumvent the computational bottlenecks that would be caused by a naive "all pairwise comparison" strategy. Indeed, it is generally not necessary to compare each node to all other nodes, as the vast majority of nodes are dissimilar and thus need not be compared. We use strategic

sparsification and randomized approximation techniques that allow us to make only a small number of relevant comparisons for each node.

1.1 Overview

Following a presentation of preliminary material in Chapter II, this thesis is organized into three parts: node-level comparison, graph-level comparison, and applications and guidelines for future praxis. We summarize these parts in more detail in Table 1.1.

Table 1.1: Thesis Overview

Part	Scale	Objective	Chapter
Methodology	Node-level	**Align** networks by matching nodes with similar structural roles	III
		Refine a node matching by reinforcing consistency between neighboring nodes' matchings	IV
	Graph-level	**Aggregate** node-level role information into graph-level features	V
Praxis	Node-level	**Infer** professional roles of email users from their structural patterns of email communication	VI
	Node/Graph-level	**Establish** evaluation methodology that reveals what structural embedding methods learn, and identify best practices as well as cautionary insights	VII

1.1.1 Node-Level Comparison

In Chapter III, we introduce a new structural embedding method, Cross-Network Matrix Factorization or xNetMF, in the context of one of the most fundamental cross-network node comparison problems: network alignment. Here, the task is to find counterparts in one network for nodes in another network, for example to align users across social networks. Our solution to this problem, REGAL, formulates this problem as a greedy feature matching: a node's alignment in another network is considered to be the node in that network with the most similar feature representation. The problem then boils down to learning an appropriate feature representation, which we do with xNetMF. Our greedy matching of xNetMF embeddings often outperform baselines that solve more complicated optimization problems, indicating that xNetMF finds node feature representations that are very comparable across networks.

While REGAL obtains high accuracy at aligning topologically similar graphs because it detects subtle structural similarities very precisely, we see that as noise obscures the graphs' topological similarity, its accuracy in network alignment decreases sharply. Our goal in Chapter IV is to develop a more robust approach to network alignment. REGAL's greedy node matching procedure means that each node is matched independently of how its neighbors have been matched: thus, two nodes that are close by in one graph may be matched far apart in the other graph. Instead, we would prefer to preserve *matched neighborhood consistency*, keeping nodes that are neighbors in one graph matched to neighbors in another graph. Our solution is to *refine* the initial network alignment found by REGAL, or for that matter any other network alignment method. Our proposed refinement algorithm RefiNA iteratively increases the alignment scores of nodes whose neighbors align, where matched neigbhorhood consistency is better satisfied. It is conceptually simple yet yields dramatic improvements in accuracy and robustness for REGAL and other network alignment methods.

1.1.2 Graph-Level Comparison

Having established the feasibility of cross-network comparison at the node level in Chapters III and VI, we turn our attention to a larger scale of comparison at the *graph* level in Chapter V. Here, we need to aggregate the node-level information that our node embeddings can capture in order to describe an entire graph. We do so in the form of a *graph-level* feature map that captures the distribution of its node embeddings in vector space. Our proposed feature construction, RGM, forms a *randomized* histogram of node embeddings: the dot product between any two graphs' histograms approximates the mean (kernelized) distance between node embeddings in two graphs. This technique is a theoretically principled way of avoiding embedding-based comparison of all pairs of nodes in two graphs, *and* avoiding computing and manipulating a kernel matrix for all pairs of graphs. Our formulations can handle any node embeddings; however, for good performance, the embeddings again need to be comparable across networks.

We apply our methods to the problem of graph classification. In this setting, we want handle an inductive setting where some of the nodes are not embedded at training time. This is because we assume a training and a test set of graphs, where the graphs in the test set are

not provided at training time. (This is in contrast to the transductive setting in Chapter III, where we assumed both graphs to align were given up front and could be embedded jointly.) To allow for inductive learning, we need to embed graphs in an unseen test set in the same subspace as those in a training set. We extend our method xNetMF to the fully inductive setting, where it becomes iNetMF.

1.1.3 Applying Structural Node Embeddings

Chapters III-V develop methodological solutions to several multi-network data mining methods for structural embeddings. The last two research works in this dissertation are intended to pave the way for further application of structural node embeddings to real-world problems.

In Chapter VI, we detail a large-scale problem we solve in collaboration with industry partners by using structural node embeddings, namely the problem of identifying the professional roles of users in email networks. In these networks, nodes are users and edges represent email exchanges. Such exchanges have a sender and a receiver and may be weighted by the number of messages exchanged between the two, meaning that these graphs are weighted and directed. Our hypothesis is that the professional role of a user is related to their structural role in a network. For example, executives at different companies, even if they never talk directly to each other, likely have similar patterns of communication compared to lower-level employees (who likely have less extensive email contacts). Here, we extend xNetMF from Chapter III (where we considered unweighted and undirected graphs) to incorporate weight and direction of communication in an email-centric manner. Doing so, we retain our scalability (we can mine actionable insights from subnetworks of individual companies consisting of only a few dozen nodes to the full email network consisting of millions of email users) while obtaining structural role information that is further enhanced by the edge weights and directions. This work serves as a motivating example of the potential of structural node embeddings to be used for large-scale data mining problems in real-world scenarios.

Finally, having seen (and developed) many applications of structural node embeddings, in Chapter VII we introduce new methods and insights to help guide further study. Our work draws back to decades-old sociological concepts of role equivalencs in networks, which

the literature on structural node embedding methods often refers to loosely without drawing rigorous connections. We curate an extensive collection of synthetic and real datasets that allow us to study three different role equivalence concepts, and propose *intrinsic* evaluations that measure a node embedding method's ability to preserve these equivalences decoupled from any downstream data mining task. We compare a large number of structural embedding methods, our own and others, using this intrinsic evaluation as well as *extrinsic* evaluation on a large number of downstream individual and collective graph mining tasks. This work gives the research community tools to thoroughly evaluate structural embedding methods, identifies some design choices that are promising for futher methodological development, and outlines best practices and pitfalls of current conventions.

1.2 Contributions

This thesis work makes several contributions to the rapidly growing area of node embedding, in particular structural node embedding:

New embedding methods: We introduce several new *structural node embedding* methods. We introduce xNetMF in Chapter III, which we extend to a fully inductive setting under the name iNetMF (Chapter V) and weighted and directed graphs under the name EMBER (Chapter VI). We show that it yields competitive results on many single-network tasks to which structural embeddings have been applied (Chapter VII).

New methods using embeddings: We formulate solutions for several collective network problems using node embeddings. These formulations motivate the development of structural node embeddings, as embeddings preserving proximity within a single graph are not naturally comparable across graphs. We introduce frameworks for network alignment based on node-level comparison (REGAL and the network alignment refinement method RefiNA) and graph classification based on graph comparison using aggregated node embeddings (RGM).

Evaluation methods and insights: based on our experience developing new structural embedding methods and applying them in various contexts, we lay the groundwork to guide future research into structural node embedding. We present new synthetic and real benchmark datasets as well as new evaluation paradigms and methods. Along the way, we identify

best practices and promising design choices in current praxis of structural embeddings and provide cautions of which future research works should be mindful.

Impact Several works in this thesis have had an impact in academia and/or industry:

- Our network alignment method REGAL (Chapter III) has been taught in graduate classes at multiple universities (e.g. the University of Michigan, Purdue University). It is the first to perform unsupervised network alignment with node embeddings, and the conference paper [HSSK18] is one of the most highly cited works on network alignment in the last five years.

- Our graph classification method RGM (Chapter V) won the best student paper award at ICDM 2019.

- The node embedding method EMBER (Chapter VI) was developed in conjunction with industry collaborators and applied to large-scale complex network data from that company.

CHAPTER II

Preliminaries & Related Work

In this chapter, we introduce notation preliminaries on graphs and node embedding. Table 2.1 summarizes this notation, along with some symbols that are common to the node embedding methodology in multiple chapters of this thesis. We then provide a conceptual overview of the difference between two important kinds of node similarity in networks: proximity and structural. This distinction is crucial to motivating our embedding work in subsequent chapters.

Table 2.1: Major symbols and definitions.

Symbols	Definitions
$G_i(V_i, E_i, \mathbf{A}_i)$	graph i with nodeset V_i, edgeset E_i, adjacency matrix $\mathbf{A}_i$
$\mathbf{F}_i$	optional matrix of attributes for each node in graph G_i
w_{uv}	weight of edge (u, v)
n_i	number of nodes in graph G_i
$\Delta_{\text{avg}}, \Delta_{\max}$	average and maximum node degree, respectively
$\mathcal{U} \subseteq V$	Set of nodes to embed
$\mathcal{N}_u^k$	set of k-hop neighbors of node u
$\mathcal{N}_u^{k+}$, $\mathcal{N}_u^{k-}$	k-step in-/out-neighborhoods of node u, respectively, in a directed graph
$\mathcal{P}_{u\to v}^{k+}$	k-step directed path from u to v (i.e., ordered edge set)
$\mathbf{d}_u^{k\pm}$	vector of node degrees in a single set $\mathcal{N}_u^{k\pm}$
$\mathbf{d}_u^{\pm}$	$= \sum_{k=1}^{K} \delta^{k-1} \mathbf{d}_u^{k\pm}$ combined neighbor degree vector for node u
$\mathbf{b}_u$	$= [\mathbf{d}_u^+ \ \mathbf{d}_u^-]$ Concatenated ingoing and outgoing structural behavior histograms for node u in a directed graph
B	number of buckets for degree binning
$\mathbf{f}_u$	F-dimensional attribute vector for node u
$\mathbf{S}$	combined structural and attribute-based similarity matrix
$\tilde{\mathbf{S}}$	approximation of $\mathbf{S}$
$\mathbf{Y}$	matrix with node embeddings as rows
$\mathbf{Y}_i$	Node embedding matrix for graph G_i in $\mathbb{R}^{n_i \times p}$
$\mathbf{Y}_{i,j}$	Vector embedding in $\mathbb{R}^p$ of node j in graph G_i
p	embedding dimension

2.1 Preliminaries on Graphs

Let $G_i(V_i, E_i, \mathbf{A}_i)$ be a **graph** with a set of n_i **nodes** V_i, pairs of which are connected by a set of **edges** E_i. We use the subscript i for disambiguation of different graphs, embeddings, and so on, as in collective network mining we have several graphs by definition. Edges in a graph G_i may be represented by an **adjacency matrix** $\mathbf{A}_i$, whose u, v-th entry is nonzero if and only if there is an edge between nodes u and v. In an **unweighted** graph, $\mathbf{A}_i$ is a binary matrix whose u, v-th entry is 1 if and only if there exists an edge between u and v; in a **weighted** graph, $\mathbf{A}_i$ is real-valued and its u, v-th entry is equal to $w(u, v)$, the weight of the edge between nodes u and v. In an **undirected** network, $\mathbf{A}_i$ is a symmetric matrix–an edge from u to v entails an edge from v to u, while this may not be the case in a **directed** network.

A node's **degree** is given the total weight of all edges incident to it. (In an unweighted graph, we assume all edge weights are one, and this amounts to the number of neighbors the node has.) The average degree of nodes in a dataset of one or more networks is given by Δ_{avg}, and the maximum node degree is $\Delta_{\max}$. In a *directed* network, we may further distinguish **indegree** and **outdegree**, which are based on the weights of edges pointing to or from a node, respectively. The **total degree** counts all edges incident to a node, whether originating from it or pointing to it. We make no distinction between indegree, outdegree, and total degree in undirected graphs, where there is no distinction between the source and the target of an edge.

In a weighted or unweighted graph, if node u forms an edge with node v, then v is a neighbor of u, and the set of all nodes v with which u shares an edge constitutes u's neighborhood. We can generalize the concept of a neighborhood beyond a node's immediate connections to a k-hop neighborhood $\mathcal{N}_u^k$, which includes all nodes with which node u is connected by a path of length at most k.

Optionally, nodes may have **features** or **attributes** that constitute side information beyond the graph structure alone. If these are available, we can construct a matrix of node attributes $\mathbf{F}_i$, whose j-th row contains the features of node j.

2.2 Node Similarity: Proximity versus Structural

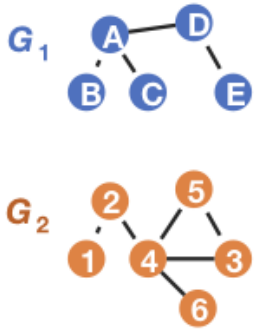


Figure 2.1: Proximity versus Structural Node Similarity. In graph G_1, nodes A and B are in closer proximity than B and C, but nodes B and C have similar structural roles. None of the nodes in G_1 are in proximity with the nodes in G_2, but we can see, for instance that node E in G_1 and node 1 in G_2 share similar structural roles.

In order to compare nodes in a graph or in different graphs, we need to define a measure of node similarity. Often, the similarity of nodes is related to their proximity to each other: nodes that are close to each other in a network (for example, those that share an edge) should have more similar embeddings than those that are not in close proximity in the network. Proximity is not necessarily a shortest-path distance between nodes, and may take into account higher-order connections between nodes beyond just the first-order connections to neighboring nodes. For instance, nodes with many second-order connections, or mutual neighbors of neighbors, are often considered to be more similar than nodes with fewer second-order connections [TQW+15]. Classic graph mining problems that depend on modeling (higher-order) node proximity, to name a few, include belief propagation, semi-supervised learning, and random walk with restart, whose formulations share a unified theoretical framework [KKK+11] and practical matrix methods for fast computation [YHJK18].

Comparing the similarity of their nodes based on their proximity makes sense when the network exhibits *homophily*, where similar nodes (for instance, social network users sharing similar interests or demographic information) do in fact connect to each other in the graph. However, it is necessary to rethink such methods when the graph exhibits *heterophily* [ZYZ+20] and similar nodes may not directly connect. Moreover, proximity-based measures of similarity cannot be used to compare nodes that are not connected by a path of any length: notable cases of this are pairs of nodes in completely separate networks.

Structural similarity, on the other hand, does not compare nodes based on their relative position to each other, but on patterns in their relationships to other nodes. For instance, nodes that form similar numbers of connections may be structurally similar even if they do not form connections with each other. As with proximity, we need not just analyze a node's

immediate connections (counting these gives us the node's degree, which is just one simple statistic with which structural similarity can be compared), but we can perform higher-order analysis that compares nodes not just on the basis of their own connectivity but on their neighbors' (or k-hop neighbors') connectivity. Intuitively, structurally similar nodes play similar "roles" in their respective parts of their networks (e.g. well-connected "hub" nodes, peripheral nodes with few connections, etc.)

In network science, structural roles of nodes have often been captured by hand-engineered statistics such as degree, centrality measures such as PageRank or betweeness centrality, recursive definitions such as SimRank [JW02], and others. Similarly, mathematical sociologists have defined concepts of role equivalence of nodes in networks, from the strict definition of structural equivalence to more relaxed definitions of automorphic and regular equivalence. In Chapter VII, we investigate the extent to which the family of methods we study in this thesis capture several well-studied network-scientific statistics and sociological concepts of equivalence.

Example 2.1. *To illustrate node similarity in a single network, in Figure 2.1, note that in graph G_1, nodes A and B are in close proximity, being connected by an edge. Even nodes B and C, which do not share an edge, both have A as a neighbor, which makes them somewhat in each other's proximity. On the other hand, nodes B and E are located very far apart in the network and would be considered dissimilar by a proximity-based method of node comparison. However, nodes B and E are structurally similar: node B has only one connection (to node A), just like node E has only one connection (to node D). Indeed, nodes B and E are much more comparable based on structural similarity than B and A, the latter being much more highly connected than the former.*

Across networks, we cannot compare nodes by proximity, but we can compare them by structural similarity. For instance, node E in graph 1 is connected to node D (also in graph 1) and node 1 is connected to node 2 (also in graph 2), but the fact that each node has only one connection to a node of degree 2 may indicate that the nodes share a similar structural role.

A proximity-based notion of node similarity is useful for many applications. For instance,

consider the application in Chapter VI of an email communication network consisting of users who work at multiple companies. If such a network were represented (at a small scale) by Figure 2.1, with G_1 and G_2 representing two different companies, node proximity could be used to compare all the users within each company, while users in different companies would be dissimilar. Thus, proximity-preserving node embeddings could be used to predict which company a user belonged to.

On the other hand, if we were trying to learn about the professional role each individual serves at its *respective* company, we would be better served by comparing the structural roles of users in their respective communication networks, not their proximity to each other. In Figure 2.1, node A and node B might be in close proximity to each other, but node A is a "hub" node connected to many other nodes (including nodes such as D that have several independent connections of their own). Rather than assuming A and B are similar because the edge they share makes them in close proximity, we should recognize the difference in their structural role. In Chapter VI we will see that this corresponds to real-world insight; for instance, B is likely a lower level employee reporting mainly to A, a supervisor with more connections. Moreover, we can perform this analysis across companies. For example, even though nodes E in G_1 and 1 in G_2 represent employees at different companies and are not in proximity, the similarity of their structural role indicates that they may share the same professional role (they are likely lower-level employees with fewer direct connections except an immediate supervisor.)

2.3 Preliminaries on Node Embeddings

Node embedding maps each node in one or more graphs into a low-dimensional vector space; that is, it embeds the nodes of each graph into that space. We denote the dimensionality of this space as p, where p is a small number, often a constant or at least asymptotically smaller than the total number of nodes. In Figure 2.2, we visualize the node embedding process in two dimensions. It can be useful to conceptualize the node

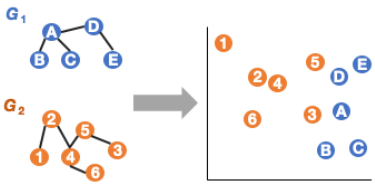


Figure 2.2: Mapping nodes in graphs into vector space (dimension $p = 2$).

embeddings as a set of points in vector space, as we do in Chapter V. However, many of our mathematical operations will represent the embeddings of nodes in graph G_i as a matrix $\mathbf{Y}_i \in \mathbb{R}^{n_i \times p}$, where the embedding of an individual node j in graph G_i is the row vector $\mathbf{Y}_{i,j}$. If we do not need to specify the graph from which a node comes, we may also write its embedding vector simply as $\mathbf{y}_j$.

2.4 Node Embedding: Related Work

The objective of node embedding is to learn similar representations for similar nodes [GF18]. That is, if $\text{sim}(u, v)$ is a graph-based similarity score between nodes u and v and and $s(\mathbf{y}_u, \mathbf{y}_v)$ a vector-based similarity score (such as cosine similarity or dot product) between their respective node embedding vectors, ideally $\text{sim}(u, v) \propto s(\mathbf{y}_u, \mathbf{y}_v)$. Here, graph-based node similarity may be determined based on node proximity or structural roles. For a distinction between the two types of embeddings, see the recent survey [RJK+20].

2.4.1 Proximity-Preserving Embeddings

Proximity-preserving node embeddings may be learned with shallow [GL16] or deep architectures [WCZ16], and the various methods may discern neighborhood structure by sampling node context through random walks [PARS14] or modeling first- and second-order connections [TQW+15]. Extensions may include incorporating textual or other node attributes [HLH17, YLZ+15]. These methods all optimize objectives that encourage nodes in close proximity to have similar features, which is helpful only in the context of a single network. As such, the above methods are transductive, being formulated for a single graph. Recent work inductively learns representations [HYL17, CMX18] using graph convolutional networks. Methods based on graph convolutions can also learn node features in a semi-supervised setting, as opposed to the majority of methods which are unsupervised. However, these methods work by aggregating features for each node based on the features of nodes within their neighborhoods. Thus, nodes in close proximity to each other end up with similar features, and graph neural networks are usually classified as proximity-preserving [RJK+20]. Other neural network models have been used to learn node embeddings, such as LSTMs [TCW+18]; such

methods take inspiration from sociological concepts of role equivalence [BE92] but primarily model node proximity, as characterized by recent work [RJK+20] and empirically confirmed by our study in Chapter VII.

2.4.2 Structural Embeddings

Unlike these methods, the recent work struc2vec [RSF17] preserves *structural* similarity of nodes, regardless of their proximity in the network. Prior to this work, existing methods for structural role discovery mainly focused on hand-engineered features [RA15]. However, for structurally similar nodes, struc2vec embeddings were found to be visually more comparable [RSF17] than those learned by state-of-the-art proximity-based node embedding techniques as well as existing methods for role discovery [HGER+12].

struc2vec uses the same skip-gram neural model used by proximity-preserving embedding methods DeepWalk [PARS14] and node2vec [GL16], but samples node context by performing random walks on an auxiliary graph where nodes are connected according to multiple levels of structural similarity. Thus, context nodes (for which similar embeddings are learned) are structurally similar to each other, not necessarily close in the original graph. Other methods using the skip-gram model to learn structural node embeddings [XQQ+19, ARL+19] perform random walks on the original graph, but define a procedure to *relabel* nodes in the graph according to a structural type. Thus, nodes that are embedded similarly to the same structural types of nodes will have similar embeddings.

Another related structural node embedding method is GraphWave [DZHL18], which relies on heat wavelet diffusion patterns to capture structural representations. Like struc2vec, it is also relatively inefficient on large networks in practice and cannot natively handle attributed graphs. It also assumes a possibly weighted but undirected graph. GraphWave differs from most embedding methods in that it essentially captures a statistical feature descriptor of each node, rather than explicitly modeling relative similarities between some nodes. We show (Chapters VI & VII) that GraphWave can be used for multi-network tasks, although its memory requirements limit its application to large networks.

Our proposed method xNetMF is based on matrix factorization; we implicitly factorize a structural node similarity matrix based on the connectivity patterns within local neighbor-

hoods (see Chapter III for details). Since we first introduced xNetMF, it has been generalized by additional works. Two such methods that we empirically study in Chapter VII are MultiLENS [JRK+19], which can model the distribution of arbitrary node attributes in local neighborhoods, and SEGK [NV19], which uses state-of-the-art kernel methods for comparing entire graphs [SSL+11] to compare local neighborhoods of nodes.

2.4.3 Node Embeddings and Matrix Factorization

Many proximity-preserving node embedding methods based on shallow neural architectures (such as skip-gram) have been shown to implicitly optimize a matrix factorization objective [QDM+18]. Matrix factorization proves an efficient way to analyze a number of node embedding methods [PARS14, TQW+15, GL16, YLZ+15] using nominally diverse techniques. Additionally, using random walks to sample node context before optimizing a skip-gram objective on the context, as the random walk sampling procedure can add some variance to the learned embeddings. Another downside to random walks is that the time taken to sample nodes can grow quite high on large graphs, often far exceeding the cost of learning the embeddings from the sampled context [GL16]. Using matrix factorization to learn embeddings avoids both these downsides, and thus we design structural embedding methods based on matrix factorization.

Part I: Methodology

CHAPTER III

Network Alignment with Structural Node Embedding

Chapter based on work that appeared at CIKM 2018 [HSSK18].

3.1 Introduction

In this chapter, we study **network alignment** or matching, which is the problem of finding corresponding nodes in different networks. Network alignment is crucial for identifying similar users in different social networks, analyzing chemical compounds, studying protein-protein interaction, and various computer vision tasks, among others [BGSW13]. Many existing methods try to relax the computationally hard optimization problem, as *designing features* that can be directly compared for nodes in different networks is not an easy task. However, recent advances [GL16, PARS14, TQW+15, WCZ16] have automated the process of learning node feature representations and have led to state-of-the-art performance in downstream prediction, classification, and clustering tasks. These methods produce node embeddings of a graph in a low-dimensional latent space so that "similar" nodes in a *single* network are embedded close together.

Motivated by the recent advances in node representation learning, which have been shown to far outperform hand-crafted features, we propose network alignment via matching latent, learned node representations. Formally, the problem can be stated as:

Problem 3.1. *Given two graphs G_1 and G_2 with node-sets $\mathcal{V}_1$ and $\mathcal{V}_2$ and possibly node attributes $\mathcal{A}_1$ and $\mathcal{A}_2$ resp., devise an efficient* ***network alignment method*** *that aligns*

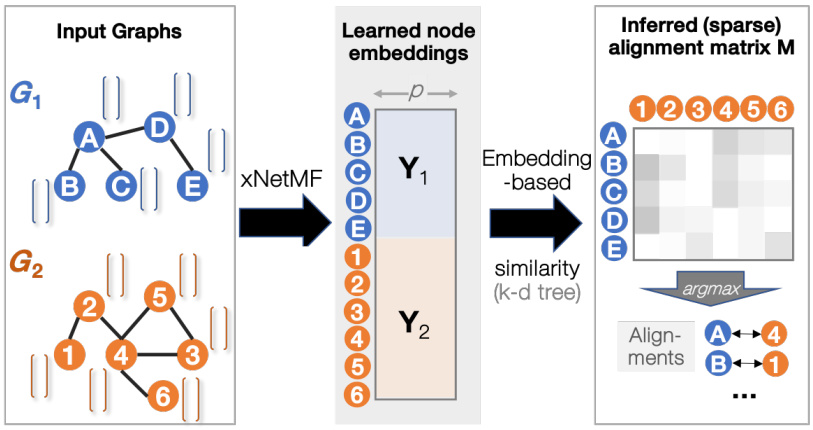


Figure 3.1: Pipeline of proposed graph alignment method, REGAL, based on our xNetMF representation learning method.

nodes by learning ***directly comparable*** *node representations* $\mathbf{Y}_1$ *and* $\mathbf{Y}_2$*, from which a node mapping* $\pi : \mathcal{V}_1 \rightarrow \mathcal{V}_2$ *between the networks can be inferred.*

To this end, we introduce **REGAL**, or REpresentation-based Graph ALignment, a framework that efficiently identifies node matchings by greedily aligning their latent feature representations. REGAL is both highly intuitive and extremely powerful given suitable node feature representations. For use within this framework, we propose Cross-Network Matrix Factorization (**xNetMF**), which we introduce specifically to satisfy the requirements of the task at hand. xNetMF differs from most existing representation learning approaches that (i) rely on proximity of nodes in a *single* graph, yielding embeddings that are not comparable across disjoint networks [HK17], and (ii) often involve some procedural randomness (e.g., random walks), which introduces variance in the embedding learning, even in one network. By contrast, xNetMF preserves *structural* similarities rather than proximity-based similarities, allowing for generalization beyond a single network.

To learn node representations through an efficient, low-variance process, we formulate xNetMF as matrix factorization over a similarity matrix that incorporates structural similarity and attribute agreement (if the latter is available) between nodes in disjoint graphs. To avoid explicitly constructing a full similarity matrix, which requires computing all pairs of similarities between nodes in the multiple input networks, we extend the Nyström low-rank approximation commonly used for large-scale kernel machines [DM05]. xNetMF is thus a principled and efficient *implicit* matrix factorization-based approach, requiring a fraction of

the time and space of the naïve approach while avoiding ad-hoc sparsification heuristics.

Our contributions may be stated as follows:

- **Problem Formulation**: We formulate the important unsupervised graph alignment problem as a problem of learning and matching node representations that *generalize to multiple graphs*. To the best of our knowledge, we are the first to do so.
- **Principled Algorithms**: We introduce a flexible alignment framework, REGAL (Figure 3.1), which learns node alignments by jointly embedding multiple graphs and comparing the most similar embeddings across graphs *without* performing all pairwise comparisons. Within REGAL we devise xNetMF, an elegant and principled representation learning formulation. xNetMF learns embeddings from structural and, if available, attribute identity, which are characteristics most conducive to multi-network analysis.
- **Extensive Experiments**: Our results demonstrate the utility of representation learning-based network alignment in terms of both speed and accuracy. Experiments on real graphs show that xNetMF runs up to 30× faster than several existing network embedding techniques, and REGAL outperforms traditional network alignment methods by 20-30% in accuracy.

For reproducibility, the source code of REGAL and xNetMF is publicly available at `https://github.com/GemsLab/REGAL`.

3.2 Related Work

Here we cover related work for the problem of network alignment or graph matching.

This problem comes up in many application domains: from data mining to security and re-identification [ZT16, KTL13, BGSW13], chemistry, bioinformatics [VM17, SXB08, Kla09], databases, translation [BGSW13], vision, and pattern recognition [ZBV09]. Network alignment is used to align users' accounts across social networks [LCLL16], reveal biological functions shared by different organisms [KMM+10], and integrate multiple data sources to create a holistic worldview network [DZK+18]. We identify two groups of network alignment

methods: those that find a joint assignment via optimization and those that directly compare nodes and match them one at a time.

Optimization. Network alignment is usually formulated as an optimization problem that tries to find a matching that leads to the greatest topological consistency between the networks. One common objective is $\min_{\mathbf{M}} ||\mathbf{M}\mathbf{A}_1\mathbf{M}^T - \mathbf{A}_2||_F^2$ [KTL13], where $\mathbf{A}_1$ and $\mathbf{A}_2$ are the adjacency matrices of the two networks to be aligned, and $\mathbf{M}$ is a permutation matrix or a relaxed version thereof, such as doubly stochastic matrix [VCP+11] or some other concave/convex relaxation [ZBV09]. Umeyama's algorithm [Ume88] directly optimizes this objective using the Hungarian algorithm [PS98], but the cubic time complexity prevents its application to large graphs.

Newer approaches formulate alternative topological consistency objectives using a range of techniques. the intuition of NetAlign [BGSW13] as a message-passing algorithm based on belief propagation, is to "complete squares" by aligning two nodes that share an edge in one graph to two nodes that share an edge in another graph. Similarly, FINAL [ZT16] has an objective of preserving topological consistency between the graphs that may be augmented with node and edge attribute information, if available. MAGNA [SM14] is a genetic algorithm that can evolve network populations to maximize topological consistency criteria such as edge correctness. Recent works frame the network alignment problem in terms of kernel methods [ZXW+19] or optimal transport [XLZD19], but these approaches suffer from high (cubic) computational complexity.

Direct Comparison. Networks can also be aligned by comparing nodes directly. Such approaches may avoid the rigidity of optimization-based methods, which are often difficult to customize when graphs have different sizes, node or edge attributes, and so on, and may be susceptible to poor local minima. GRAAL [KMM+10] computes graphlet degree signatures for each node based on the number of occurrences of various small network motifs in the node's local neighborhood. GHOST [PK12] defines a multiscale spectral signature for each node and, like GRAAL, uses a seed-and-extend heuristic to align nodes across graphs, where very similar nodes are matched first and their neighborhoods are then aligned. UniAlign [KTL13] extracts features for each node from graph statistics (such as degree and various centralities of a node) and uses their similarity to perform network alignment.

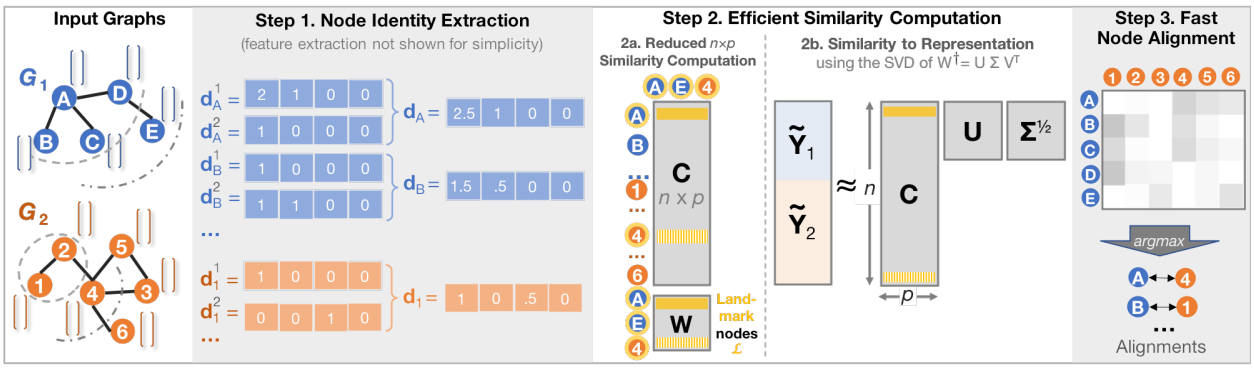


Figure 3.2: Proposed REGAL approach, consisting of 3 main steps. In the example, for the structural identity, up to $K = 2$ hop away neighborhoods are taken into account (the 1-hop and 2-hop neighborhoods for nodes A and 1 are shown with dashed and dash-dotted lines, respectively). The discount factor is set to $\delta = 0.5$. For simplicity, no logarithmic binning is applied on $\mathbf{d}_u^k$.

In contrast to the hand-engineered features in previous work, we use latent features learned by node embedding (for a review of node embedding works, see Chapter II) that better capture subtleties of the graph structure. Prior to this work, using node embeddings designed for social networks to align users [LCLL16] has required the graph to be partially aligned already, so that proximity can be defined between nodes in different networks. Our work in this chapter [HSSK18] was the first to use node embeddings to align graphs in an unsupervised setting, where no node matchings are known in advance. Following the work in this chapter, we proposed another method to use node embeddings for unsupervised network alignment [CHVK20]; here we use node embeddings that preserve proximity within each network separately, but align the embedding spaces so that they are comparable across networks before using the node embeddings to align individual nodes. A similar effect was recently achieved using adversarial training [DKL+19]. Another followup work of ours replaced the implicit matrix factorization with which we learn node embeddings in this chapter with graph convolutional networks using random weights: along with compressing the graph to perform the alignment on a smaller graph, this approach led to comparable accuracy at decreased running time [QSR+20].

3.3 REGAL: REpresentation-based Graph ALignment

In this section we introduce our representation learning-based network alignment framework, REGAL, for Problem 3.1. For simplicity we focus on aligning two graphs (e.g., social or protein networks), though our method can easily be extended to more networks. Let $G_1(V_1, E_1)$ and $G_2(V_2, E_2)$ be two unweighted and undirected graphs with node sets $\mathcal{V}_1$ and $\mathcal{V}_2$; edge sets $\mathcal{E}_1$ and $\mathcal{E}_2$; and possibly node attributes $\mathcal{A}_1$ and $\mathcal{A}_2$, respectively. Note that these graphs do *not* have to be the same size, unlike many other network alignment formulations that have this restriction. Let n be the number of nodes across graphs, i.e., $n = |V_1| + |V_2|$.

The steps of REGAL may be summarized as:

1. **Node Identity Extraction**: The first step extracts structure- and attribute-related information for all n nodes.

2. **Efficient Similarity-based Representation**: The second step obtains the node embeddings, conceptually by factorizing a similarity matrix of the node identities from the previous step. To avoid the expensive computation of pairwise node similarities and explicit factorization, we extend the Nyström method for low-rank matrix approximation to perform an *implicit* similarity matrix factorization by **(a)** comparing the similarity of each node only to a sample of $p \ll n$ "landmark" nodes, and **(b)** using these node-to-landmark similarities to construct our representations from a decomposition of its low-rank approximation.

3. **Fast Node Representation Alignment**: Finally, we align nodes between graphs by greedily matching the embeddings with an efficient data structure that allows for fast identification of the top-α most similar embeddings from the other graph(s).

In the rest of this section we discuss and justify each step of REGAL, the pseudocode of which is given in Algorithm 3.1. Note that the first two steps, which output a set of node embeddings, comprise our xNetMF node embedding method, described in Algorithm 3.2. In later chapters, we will use and extend xNetMF for additional individual and collective network mining tasks. Thus, we present it here as an algorithm that accepts a single input graph. To embed multiple networks, we combine their adjacency matrices as blocks

as a single block-diagonal adjacency matrix. We can thus embed all the nodes in all the input graphs jointly by embedding this combined graph. Embedding matrices for the nodes in an individual input graph can be recovered by selecting rows of the combined graph's embeddings corresponding to nodes in that input graph.

3.3.1 Node Identity Extraction

The goal of REGAL's representation learning module, xNetMF, is to define node "identity" in a way that generalizes across networks. This step is critical because many existing works define identity based on node-to-node proximity, but collective network mining involves comparing nodes that have no direct connections to each other and thus cannot be sampled in each other's contexts by random walks on separate graphs. To overcome this problem, we focus instead on more broadly comparable, generalizable quantities: *structural* identity, which relates to structural roles [HGER$^+$12], and *attribute*-based identity.

Structural Identity. In network alignment, the well-established assumption is that aligned nodes have similar *structural* connectivity or degrees [KTL13, ZT16]. Adhering to this assumption, we propose to learn about a node's structural identity from the degrees of its neighbors. To gain higher-order information, we also consider neighbors up to k hops from the original node.

For a node $u \in V$, we denote $\mathcal{N}_u^k$ as the set of nodes that are exactly $k \geq 0$ steps away from u in its own graph. We want to capture degree information about the nodes in $\mathcal{N}_u^k$. A basic approach would be to store the degrees in a $\Delta_{\max}$-dimensional vector $\mathbf{d}_u^k$, where $\Delta_{\max}$ is the maximum degree in the original graph G, with the i-th entry of $\mathbf{d}_u^k$, or $d_u^k(i)$, the number of nodes in $\mathcal{N}_u^k$ with degree i. For simplicity, an example of this approach is shown for the vectors $\mathbf{d}_A, \mathbf{d}_B$, etc. in Figure 3.2. However, real graphs have skewed degree distributions. To prevent one high-degree node from inflating the length of these vectors, we bin nodes together into $b = \lceil \log_2 \Delta_{\max} \rceil$ logarithmically scaled buckets such that the i-th entry of $\mathbf{d}_u^k$ contains the number of nodes $u \in \mathcal{N}_u^k$ such that $\lfloor \log_2(deg(u)) \rfloor = i$. This has two benefits: (1) it shortens the vectors $\mathbf{d}_u^k$ to a manageable $\lceil \log_2 \Delta_{\max} \rceil$ dimensions, and (2) it makes their entries more robust to small changes in degree introduced by noise, especially for high degrees when more different degree values are combined into one bucket.

Attribute-Based Identity. Node attributes, or features, have been shown to be useful for cross-network tasks [ZT16]. Given F node attributes, we can create for each node u an F-dimensional vector $\mathbf{f}_u$ representing its values (or lack thereof). For example, $f_u(i)$ corresponds to the i^{th} attribute value for node u. Since we focus on node representations, we mainly consider node attributes, although we note that statistics such as the mean or standard deviation of edge attributes on incident edges to a node can easily be turned into node attributes. Note that while REGAL is flexible to incorporate attributes, if available, it can also rely solely on structural information when such side information is not available.

Cross-Network Node Similarity. We now incorporate the above aspects of node identity into a combined similarity function that can be used to compare nodes within *or across* graphs, relying on the comparable notions of structural and attribute identity, rather than direct proximity of any kind:

$$\text{sim}(u, v) = \exp\left[-\gamma_s \cdot ||\mathbf{d}_u - \mathbf{d}_v||_2^2 - \gamma_a \cdot \text{dist}(\mathbf{f}_u, \mathbf{f}_v)\right], \tag{3.1}$$

where γ_s and γ_a are scalar parameters controlling the effect of the structural and attribute-based identity respectively; $\text{dist}(\mathbf{f}_u, \mathbf{f}_v)$ is the attribute-based distance of nodes u and v, discussed below (this term is ignored if there are no attributes); $\mathbf{d}_u = \sum_{k=1}^{K} \delta^{k-1}\mathbf{d}_u^k$ is the neighbor degree vector for node u aggregated over K different hops; $\delta \in (0, 1]$ is a discount factor for greater hop distances; and K is a maximum hop distance to consider (up to the graph diameter). Thus, we compare structural identity at several levels by combining the neighborhood degree distributions at several hop distances, attenuating the influence of distant neighborhoods with a weighting schema that is often encountered in diffusion processes [KVF13].

The distance between attribute vectors depends on the type of node attributes (e.g., categorical, real-valued). A variety of functions can be employed accordingly. For categorical attributes, which have been studied in attributed network alignment [ZT16], we propose using the number of disagreeing features as a attribute-based distance measure of nodes u and v: $\text{dist}(\mathbf{f}_u, \mathbf{f}_v) = \sum_{i=1}^{F} \mathbb{1}_{f_u(i) \neq f_v(i)}$, where $\mathbb{1}$ is the indicator function. Real-valued attributes can be compared by Euclidean or cosine distance, for example.

3.3.2 Efficient Similarity-based Representation

As we have mentioned, many representation learning methods are stochastic [TQW+15, PARS14, WCZ16, GL16, RSF17]. A subset of these rely on random walks on the original graph [PARS14, GL16] or a generated multi-layer similarity graph [RSF17]) to sample context for the SGNS embedding model. For cross-network analysis, we avoid random walks for two reasons: (1) The variance they introduce in the representation learning often makes embeddings across different networks non-comparable [HK17]; (2) They can add to the computational expense. Although the computation of node similarity via random walk with restart can be accelerated in some contexts [YHJK18], actually sampling these random walks requires considerable time and space. Indeed node2vec's total runtime is dominated by its sampling time [GL16].

To overcome the aforementioned issues, we propose a new *implicit* matrix factorization-based approach that leverages a combined structural and attribute-based similarity matrix $\mathbf{S}$, which is induced by our similarity function in Eq. (3.1) and considers affinities at different neighborhoods. Intuitively, the goal is to find $n \times p$ matrices $\mathbf{Y}$ and $\mathbf{Z}$ such that: $\mathbf{S} \approx \mathbf{Y}\mathbf{Z}^\top$, where $\mathbf{Y}$ is the node embedding matrix and $\mathbf{Z}$ is not needed for our purposes. We first discuss the limitations of traditional approaches, then propose an efficient way of obtaining the embeddings *without* ever explicitly computing $\mathbf{S}$.

Limitations of Existing Approaches. A natural but naïve approach is to compute combined structural and attribute-based similarities between *all* pairs of nodes within and across *both* graphs to form the matrix $\mathbf{S}$, such that $\mathbf{S}_{ij} = sim(i, j)\ \forall i, j \in V$. Then $\mathbf{S}$ can be explicitly factorized, for example by minimizing a factorization loss function given $\mathbf{S}$ as input, (e.g., the Frobenius norm $||\mathbf{S} - \mathbf{Y}\mathbf{Z}^\top||_F^2$ [LS01]). However, both the computation and storage of $\mathbf{S}$ have *quadratic* complexity in n. While this would allow us to embed graphs jointly, it lacks the needed scalability for multiple large networks.

Another alternative is to create a *sparse* similarity matrix by calculating only the "most important" similarities, for each node choosing a small number of comparisons using heuristics like similarity of node degree [RSF17]. However, such ad-hoc heuristics may be fragile in the context of noise. We will have no approximation at all for most of the similarities,

and there is no guarantee that the most important ones are computed.

Reduced $n \times p$ Similarity Computation. Instead, we propose a principled way of *approximating* the full similarity matrix $\mathbf{S}$ with a low-rank matrix $\tilde{\mathbf{S}}$, which is *never* explicitly computed. To do so, we randomly select $p \ll n$ "*landmark*" nodes chosen across both graphs G_1 and G_2 and compute their similarities to all n nodes in these graphs using Eq. (3.1). This yields an $n \times p$ similarity matrix $\mathbf{C}$, from which we can extract a $p \times p$ "landmark-to-landmark" submatrix $\mathbf{W}$. As we explain below, these two matrices suffice to approximate the full similarity matrix and allow us to obtain node embeddings *without* actually computing and factorizing $\tilde{\mathbf{S}}$.

To do so, we extend the Nyström method, which has applications in randomized matrix methods for kernel machines [DM05], to node embedding. The low-rank matrix $\tilde{\mathbf{S}}$ is:

$$\tilde{\mathbf{S}} = \mathbf{C}\mathbf{W}^{\dagger}\mathbf{C}^{\top}, \tag{3.2}$$

where $\mathbf{C}$ is an $n \times p$ matrix formed by sampling p landmark nodes from V and computing the similarity of all n nodes of G_1 and G_2 to the p landmarks only, as shown in Figure 3.2. Meanwhile, $\mathbf{W}^{\dagger}$ is the pseudoinverse of $\mathbf{W}$, a $p \times p$ matrix consisting of the pairwise similarities among the landmark nodes (it corresponds to a subset of p rows of $\mathbf{C}$). We choose landmarks randomly; more elaborate (and slower) sampling techniques based on leverage scores [AM15] or node centrality measures offer little, if any, performance improvement.

Because $\tilde{\mathbf{S}}$ contains an estimate for the similarity between any pair of nodes in either graph, it would still take $\Omega(n^2)$ time and space to compute and store. However, as we discuss below, to learn node representations we *never* have to explicitly construct $\tilde{\mathbf{S}}$ either.

From Similarity to Representation. Recall that our ultimate interest is not in the similarity matrix $\mathbf{S}$ or even an approximation such as $\tilde{\mathbf{S}}$, but in the node embeddings that we can obtain from a factorization of the latter. We now show that we can actually obtain these from the decomposition in Eq. (3.2):

Theorem 3.1. *Given graphs $G_1(V_1, E_1)$ and $G_2(V_2, E_2)$ with $n \times n$ joint combined structural and attribute-based similarity matrix $\mathbf{S} \approx \mathbf{Y}\mathbf{Z}^T$, its node embedding matrix $\mathbf{Y}$ can be*

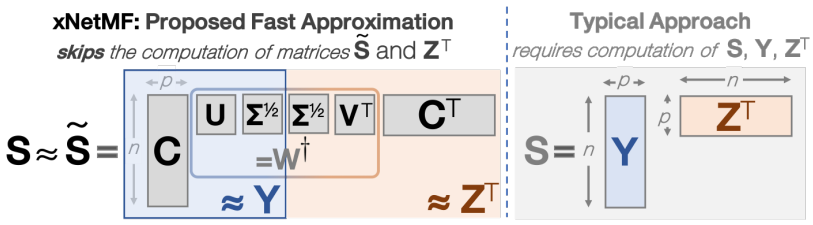


Figure 3.3: Proposed xNetMF (using the SVD of $\mathbf{W}^\dagger$) vs. typical matrix factorization for computing the node embeddings $\mathbf{Y}$. Our xNetMF method leads to significant savings in space and runtime.

approximated as

$$\mathbf{Y} = \mathbf{C}\mathbf{U}\mathbf{\Sigma}^{1/2},$$

where $\mathbf{C}$ *is the* $n \times p$ *matrix of similarities between the* n *nodes and* p *randomly chosen landmark nodes, and* $\mathbf{W}^\dagger = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top$ *is the full rank singular value decomposition of the pseudoinverse of the small* $p \times p$ *landmark-to-landmark similarity matrix* $\mathbf{W}$*.*

Proof. Given the full-rank SVD of the $p \times p$ matrix $\mathbf{W}^\dagger$ as $\mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top$, we can rewrite Eq. (3.2) as $\mathbf{S} \approx \tilde{\mathbf{S}} = \mathbf{C}(\mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top)\mathbf{C}^\top = (\mathbf{C}\mathbf{U}\mathbf{\Sigma}^{1/2}) \cdot (\mathbf{\Sigma}^{1/2}\mathbf{V}^\top\mathbf{C}^\top) = \mathbf{Y}\tilde{\mathbf{Z}}^\top$. □

Now, we *never* have to construct an $n \times n$ matrix and then factorize it (i.e., by optimizing a nonconvex factorization objective). Instead, to derive $\mathbf{Y}$, the only node comparisons we need are for the $n \times p$ "skinny" matrix $\mathbf{C}$, while the expensive SVD is performed only on its small submatrix $\mathbf{W}$. Thus, we can obtain node representations by *implicitly* factorizing $\tilde{\mathbf{S}}$, a low-rank approximation of the full similarity matrix $\mathbf{S}$. The p-dimensional node embeddings of the two input graphs G_1 and G_2 are then subsets of $\mathbf{Y}$: $\mathbf{Y}_1$ and $\mathbf{Y}_2$, respectively. This construction corresponds to the explicit factorization (Figure 3.3), but at significant runtime and storage savings.

As stated earlier, xNetMF, which we summarize in Alg. 3.2, forms the first two steps of REGAL. The postprocessing step, where we normalize the magnitude of the embeddings, makes them more comparable based on Euclidean distance, which we use in REGAL.

3.3.2.1 Connections: xNetMF and SGNS

Here we unpack the key components of the struc2vec framework [RSF17], a random walk-based structural representation learning approach, and we find a matrix factorization

interpretation at the heart of it.

Given a (single-layer) similarity graph $\mathbf{S}$, for each node v, struc2vec samples context nodes $\mathcal{C}$ with m random walks of length ℓ starting from v. The probability of going from node u to node v is proportional to the nodes' (structural) similarity s_{uv}. This yields a co-occurrence matrix $\mathbf{D}$: $d_{uv} = \#(u, v)$ is the number of times node v was visited in context of node u. Afterward, struc2vec optimizes a skip-gram objective function with negative sampling (SGNS):

$$\max_{\mathbf{Y},\mathbf{C}} \sum_{y \in V, c \in \mathcal{C}} \#(y, c) \log \sigma(\mathbf{y}^\top \mathbf{c}) + \ell \cdot \mathbb{E}_{\mathbf{c}' \sim P_D} \log \sigma(-\mathbf{y}^\top \mathbf{c}') \tag{3.3}$$

where $\mathbf{y}$ and $\mathbf{c}$ are the embeddings of a node y, and its context node c, resp.; $P_D(c) = \sum_{y \in V} \#(y, c) / \sum_{y \in V, c \in C} \#(y, c)$ is the empirical probability that a node is sampled as some other node's context; and $\sigma(x) = (1 + e^{-x})^{-1}$ is the sigmoid function. Analysis of SGNS for word embeddings [LXT+15] showed under some assumptions on the upper bound of the co-occurrence count between two words that the objective of SGNS in Eq. (3.3) is equivalent to matrix factorization of the co-occurrence matrix $\mathbf{D}$, or $\text{MF}(\mathbf{D}, \mathbf{Y}^\top \mathbf{C})$. Here MF is the objective of matrix factorization on $\mathbf{D}$ (formally defined in [LXT+15], but in practice other matrix factorization techniques work well).

Now, under these assumptions, we show a connection between optimizing Eq. (3.3) with context sampled from the similarity graph (as in struc2vec), and factorizing the graph (as in xNetMF).

Lemma 3.1. *Equation* (3.3)*, defined over a context sampled by performing m length-1 random walks per node over $\mathbf{S}$, is equivalent to* $\text{MF}(\mathbf{S}, \mathbf{Y}^\top \mathbf{C})$ *in the limit as m goes to ∞, up to scaling of $\mathbf{S}$.*

Proof. This follows from the Law of Large Numbers. As $m \to \infty$, the co-occurrence matrix $\mathbf{D}$ converges to its expectation. This is just $m \cdot \mathbf{S}$, since d_{ij} is the # of times node v_j is sampled in a random walk of length 1 from v_i, which is equal to the # of walks from node v_i times the probability that the walk goes to v_j from v_i, or $m \cdot s_{ij}$. (Since MF is invariant to scaling, we normalize $\mathbf{D}$ w.l.o.g.) □

Algorithm 3.1 REGAL

`Input`: Two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$, dimensionality p, maximum step K, discount factor $\delta \in (0, 1]$, coefficients on structural and attribute information γ_s and γ_a
`Output`: $n_1 \times n_2$ matrix $\mathbf{M}$ specifying alignments between nodes in G_1 and G_2

`S1-2: Structural Node Representation Learning`

$G = \text{CombineGraphs}(G_1, G_2)$ ▷ Make each input graph is a component of one large graph
$\mathbf{Y} = \text{xNetMF}\ (G_1, G_2, p, K, \delta, \gamma_s, \gamma_a)$ ▷ Learn p-dimensional embeddings jointly ▷ for each node in both graphs
$[\mathbf{Y}_1, \mathbf{Y}_2] = \text{SplitEmbeddings}(\mathbf{Y})$ ▷ Split combined embeddings into embeddings for ▷ nodes in each original graph

`S3: Fast Node Representation Alignment`

$\mathbf{M} =$ **empty** ▷ **sparse** $n_1 \times n_2$ matrix $\mathbf{M}$ of possible alignments
$T = \text{KDTree}(\mathbf{Y}_2)$ ▷ Build a k-d tree on the node embeddings of G_2
/* **Match embeddings** to infer alignments */
for $i = 1 \rightarrow n_1$ **do**
    /* For embedding i in G_1, get the α most similar embed. in G_2 and distances*/
    [TOP-α, TOP-dist] = QueryKDTree(T, $\mathbf{Y}_1[i]$, α) ▷ $\mathbf{Y}_1[i]$: i^{th} embedding
    for j in TOP-α **do**
        $m_{ij} = e^{-\text{TOP-dist}[j]}$ ▷ Populating alignment matrix $\mathbf{M}$ with embed. ▷ similarities: $e^{\text{TOP-dist}[j]} = e^{-||\mathbf{Y}_1[i] - \mathbf{Y}_2[j]||_2^2}$
return M ▷ alignments are largest entries in each row or column (Figure 3.1)

Note that in struc2vec, increasing m to sample more context reduces variance in $\mathbf{D}$, but increasing ℓ simply causes the random walks to move further from the original node v and sample context based on similarity to more structurally distant nodes. Lemma 3.1 connects xNetMF to a version of struc2vec with maximal m and minimal ℓ, further justifying its success by comparison.

3.3.3 Fast Node Representation Alignment

The final step of REGAL is to efficiently align nodes using their representations, assuming that two nodes $u \in V_1$ and $v \in V_2$ may match if their xNetMF embeddings are similar. Let $\mathbf{Y}_1$ and $\mathbf{Y}_2$ be matrices of the p-dimensional embeddings for nodes in graphs G_1 and G_2. We take the likeliness of (soft) alignment to be proportional to the similarity between the nodes' embeddings. Thus, we greedily align nodes to their closest match in the other graph based on embedding similarity, as shown in Figure 3.2. This method is simpler and faster than optimization-based approaches, and works thanks to high-quality node feature representations.

Data structures for efficient alignment. A natural way to find the alignments for each

Algorithm 3.2 xNetMF

`Input`: Graph $G = (V, E)$, dimensionality p, maximum step K, discount factor $\delta \in (0, 1]$, coefficients on structural and attribute information γ_s and γ_a
`Output`: $n_1 \times p$ and $n_2 \times p$ matrices of embeddings for nodes in G_1 and G_2 respectively

`S1: Node Identity Extraction`

for node u in V **do**
  for hop k up to K **do** ▷ counts of node degrees of k-hop neighbors of u
    $\mathbf{d}_u^k = \text{CountDegreeDistributions}(\mathcal{N}_u^k)$ ▷ $1 \leq K \leq$ graph diameter
  $\mathbf{d}_u = \sum_{k=1}^{K} \delta^{k-1} \mathbf{d}_u^k$ ▷ discount factor $\delta \in (0, 1]$

`S2: Efficient Similarity-based Representation`

`S2a: Reduced` $n \times p$ `Similarity Computation`

$\mathcal{L} = \text{ChooseLandmarks}(G_1, G_2, p)$ ▷ choose p nodes from G
for node u in V **do**
  for node v in $\mathcal{L}$ **do**
    $c_{uv} = e^{-\gamma_s \cdot ||\mathbf{d}_u - \mathbf{d}_v||_2^2 - \gamma_a \cdot \text{dist}(\mathbf{f}_u, \mathbf{f}_v)}$
    ▷ Used in low-rank approx. of similarity graph (not constructed)

`S2b: Similarity to Representation`

$\mathbf{W} = \mathbf{C}[\mathcal{L}, \mathcal{L}]$ ▷ Rows of $\mathbf{C}$ corresponding to landmark nodes
$[\mathbf{U}, \mathbf{\Sigma}, \mathbf{V}] = \text{SVD}(\mathbf{W}^\dagger)$
$\mathbf{Y} = \mathbf{C}\mathbf{U}\mathbf{\Sigma}^{-\frac{1}{2}}$ ▷ **Embedding**: *implicit* factorization of similarity graph
$\mathbf{Y} = Normalize(\mathbf{Y})$ ▷ **Postprocessing**: make embeddings have magnitude 1
return $\mathbf{Y}$

node is to compute all pairs of similarities between node embeddings (i.e., the rows of $\tilde{\mathbf{Y}}_1$ and $\tilde{\mathbf{Y}}_2$) and choose the top-1 for each node. Of course, this is not desirable due to its inefficiency. Since in practice only the top-α most likely alignments are used, we turn to specialized data structures for quickly finding the closest data points. We store the embeddings $\tilde{\mathbf{Y}}_2$ in a k-d tree, a data structure used to accelerate exact similarity search for nearest neighbor algorithms and many other applications [B+10].

For each node in G_1, we can quickly query this tree with its embedding to find the $\alpha \ll n_2$ closest embeddings from nodes in G_2. This allows us to compute "soft" alignments for each node by returning one or more nodes in the opposite graph with the most similar embeddings, unlike many existing alignment methods that only find "hard" alignments [BGSW13, ZT16, SXB08, Kla09]. Here, we define the similarity between the p-dimensional embeddings of nodes u and v as $sim_{emb}(\mathbf{Y}_1[u], \mathbf{Y}_2[v]) = e^{-||\mathbf{Y}_1[u] - \mathbf{Y}_2[v]||_2^2}$, which converts the Euclidean distance to similarity. Since we only want to align nodes to counterparts in the other graph, we only compare embeddings in $\mathbf{Y}_1$ with ones in $\mathbf{Y}_2$. If multiple top alignments are desired, they may be returned in **sorted** order by their embedding similarity; we use sparse matrix notation in the pseudocode just for simplicity.

Randomized Techniques and Multiple Network Alignment. While k-d trees enable exact nearest-neighbor search, we can also leverage fast techniques for approximate nearest-neighbor search, namely locality-sensitive hashing (LSH). LSH is a randomized technique that maps feature vectors into *hash buckets* using randomized *hash functions*: similar data points will, with high probability, be mapped to the same bucket, while very different data points will with high probability be mapped to different buckets. Thus, when we apply LSH to the node features, the hash buckets group each node along with its nearest neighbors, which represent possible alignments.

Using handcrafted structural network statistics as well as node and edge attributes to form features for each node, we have performed accurate and fast *multiple* network alignment [HLP+18] with locality-sensitive hashing. In multiple network alignment, we must find correspondences between several networks (three or more) simultaneously. We accelerate this process by using LSH to align all networks to a center network, for which we choose the network whose structural and attribute feature distributions are most similar to those of all the other networks. Alignments between all other pairs of peripheral networks (other than the center network) are found by transitivity: nodes in the peripheral networks that align to the same node in the center network are aligned to each other [ZY15].

In this thesis, we present experiments with latent features learned by node embedding, instead of handcrafted features. When applied to node embeddings, exact neighbor search accelerated with k-d trees is highly scalable and allows us to consider million-node graphs (an unprecedented size for network alignment). We refer the reader to [HLP+18] for a greater discussion and evaluation of hashing-based network alignment.

3.3.4 Complexity Analysis

Here we analyze the computational complexity of each step of REGAL. To simplify notation, we assume both graphs have $n_1 = n_2 = n'$ nodes.

1. **Extracting node identity**: It takes approximately $O(n'K\Delta_{\text{avg}}^2)$ time, finding neighborhoods up to hop distance K by joining the neighborhoods of neighbors at the previous hop: formally, we can construct $\mathcal{N}_u^k = \bigcup_{v \in \mathcal{R}_u^{k-1}} \mathcal{R}_v^1 - \bigcup_{i=1}^{k-1} \mathcal{R}_u^i$. We could also use breadth-first

search from each node to compute the k-hop neighborhoods in $O(n'^3)$ worst case time—in practice significantly lower for sparse graphs and/or small K—but we find that this construction is faster in practice.

2. **Computing similarities**: We compute the similarities of the length-b features (weighted counts of node degrees in the k-hop neighborhoods, split into b buckets) between each node and p landmark nodes: this takes $O(n'pb)$ time.

3. **Obtaining representations**: We first compute the pseudoinverse and SVD of the $p \times p$ matrix $\mathbf{W}$ in time $O(p^3)$, and then left multiply it by $\mathbf{C}$ in time $O(n'p^2)$. Since $p \ll n'$, the total time complexity for this step is $O(n'p^2)$.

4. **Aligning embeddings**: We construct a k-d tree and use it to find the top alignment(s) in G_2 for each of the n' nodes in G_1 in average-case time complexity $O(n' \log n')$.

The total complexity is $O(n' \max(pb, p^2, K\Delta_{\text{avg}}^2, \log n'))$. As we show experimentally, it suffices to choose small K as well as p and b logarithmic in n'. With Δ_{avg} often being small in practice, this can yield *sub-quadratic* time complexity. It is straightforward to show that the space requirements are sub-quadratic as well.

3.4 Experiments

We answer three important questions about our methods:

- **Q1** How does REGAL compare to baseline methods for network alignment on noisy real world datasets (Table 3.1), with and without attribute information, in terms of accuracy and runtime?

- **Q2** How scalable is REGAL?

- **Q3** How sensitive are REGAL and xNetMF to hyperparameters?

3.4.1 Experimental Setup.

Data. Following the network alignment literature [KTL13, ZT16], for each real network dataset (Table 3.1) with *adjacency matrix* $\mathbf{A}$, we generate a new network with adjacency matrix $\mathbf{A}^* = \mathbf{PAP}^\top$, where $\mathbf{P}$ is a randomly generated permutation matrix with the nonzero entries representing ground-truth alignments. We add structural noise to $\mathbf{A}'$ by removing edges with probability p_s without disconnecting any nodes.

Table 3.1: Real data used in our experiments.

Name	Nodes	Edges	Description
Facebook [VMCG09]	63 731	817 090	social network
Arxiv [LK14]	18 722	198 110	collaboration network
DBLP [PPRB13]	9 143	16 338	collaboration network
PPI [BSR+08]	3 890	76 584	protein-protein interaction
Arenas Email [Kun13]	1 133	5 451	communication network

For experiments with attributes, we generate synthetic attributes for each node if the graph does not have any. We add noise to these by flipping binary values or choosing categorical attribute values uniformly at random from the remaining possible values with probability p_a. For each dataset and noise level, noise is randomly and independently added.

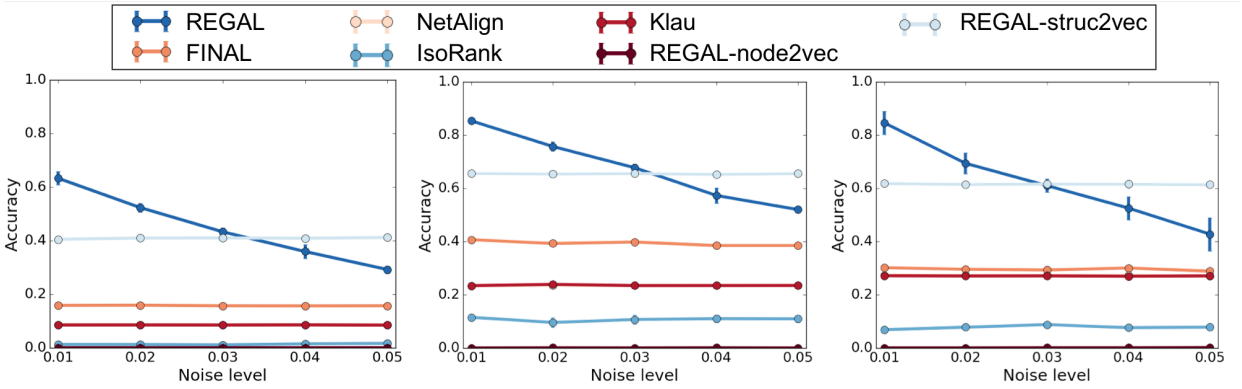
All experiments are performed on an Intel(R) Xeon(R) CPU E5-1650 at 3.50GHz with 256GB RAM, with hyperparameters $\delta = 0.01$, $K = 2$, $\gamma_s = \gamma_a = 1$, and $p = \lfloor 10 \log_2 n \rfloor$ unless otherwise stated. Landmarks for REGAL are chosen arbitrarily from among the nodes in our graphs, in keeping with the effectiveness and popularity of sampling uniformly at random [DM05]. In Sec. 3.4.4, we explore the parameter choices and find that these settings yield stable results at reasonable computational cost.

Baselines. We compare against six baselines. Four are well known existing network alignment methods and two are variants of our proposed framework that match embeddings produced by existing node embedding methods (i.e., not xNetMF). The **four existing network alignment methods** are: **(1) FINAL**, which introduces a family of algorithms optimizing quadratic objective functions [ZT16]; **(2) NetAlign**, which formulates alignment as an integer quadratic programming problem and solves it with message passing algorithms [BGSW13]; **(3) IsoRank**, which solves a version of the integer quadratic pro-

gram with relaxed constraints [SXB08]; and **(4) Klau's** algorithm (Klau), which imposes a linear programming relaxation, decomposes the symmetric constraints and solves it iteratively [Kla09]. These methods all require as input a matrix containing prior alignment information, which we construct from degree similarity, taking the top $\lfloor \log_2 n \rfloor$ entries for each node; REGAL, by contrast, does not require prior alignment information.

For the **two variants** of our framework, which we refer to as **(5) REGAL-node2vec** and (6) **REGAL-struc2vec**, we replace our own xNetMF embedding step (i.e., Steps 1 and 2 in REGAL) with existing node representation learning methods node2vec [GL16] or struc2vec [RSF17]: two recent, state-of-the-art node embedding methods that make a claim about being able to capture some form of structural equivalence. To apply these embedding methods, which were formulated for a single network, we create a single input graph G by combining the graphs with respective adjacency matrices $\mathbf{A}$ and $\mathbf{A}^*$ into one block-diagonal adjacency matrix $[\mathbf{A}\ \mathbf{0}; \mathbf{0}\ \mathbf{A}^*]$. Beyond the input, we use their default parameters: 10 random walks of length 80 for each node to sample context with a window size of 10. For node2vec, we set $p = q = 1$ (other values make little difference). For struc2vec, we use the recommended optimizations [RSF17] to compress the degree sequences and reduce the number of node comparisons, which were found to speed up computation with little effect on performance [RSF17]. As we do for our xNetMF method, we consider a maximum hop distance of $K = 2$.

Metrics. We compare REGAL to baselines with two metrics: **alignment accuracy**, which we take as (# correct alignments) / (total # alignments), and **runtime**. When computing results, we average over 5 independent trials on each dataset at each setting (with different random permutations and noise additions) and report the mean result and the standard deviation (as bars around each point in our plots.) We also show where REGAL's soft alignments contain the "correct" similarities within its top $\alpha \ll n$ choices using the more **general top-α accuracy**: (# correct alignments in top-α choices) / (total # alignments). This metric does not apply to the existing network alignment baselines that do not directly match node embeddings and only find hard alignments.



(a) Arxiv (198 110 **edges)** **(b) PPI** (76 584 **edges)** **(c) Arenas Email** (5 451 **edges)**

Figure 3.4: Accuracy of network alignment methods with varying p_s. REGAL (in dark blue) achieves consistently high accuracy *and* runs faster than its closest competitors (Table 3.2).

Table 3.2: Average (stdev) runtime in sec of alignment methods from 5 trials. The two fastest methods per dataset are in bold. REGAL is faster than its closest competitors in accuracy (Figure 3.4).

Dataset	Arxiv	PPI	Arenas
FINAL	4182 (180)	62.88 (32.20)	3.82 (1.41)
NetAlign	149.62 (282.03)	22.44 (0.61)	**1.89 (0.07)**
IsoRank	**17.04 (6.22)**	**6.14 (1.33)**	**0.73 (0.05)**
Klau	1291.00 (373)	476.54 (8.98)	43.04 (0.80)
REGAL-node2vec	709.04 (20.98)	139.56 (1.54)	15.05 (0.23)
REGAL-struc2vec	1975.37 (223.22)	441.35 (13.21)	74.07 (0.95)
REGAL	**86.80 (11.23)**	**18.27 (2.12)**	2.32 (0.31)

3.4.2 Comparative Alignment Performance

To assess the comparative performance of REGAL versus existing network alignment methods on a variety of challenging datasets, we perform two experiments studying the effects of structural and attribute noise, respectively.

3.4.2.1 Effects of structural noise

In this experiment we study how well REGAL matches nodes based on structural identity alone. This also allows us to compare to the baseline network alignment methods NetAlign, IsoRank, and Klau, as well as the node embedding methods node2vec and struc2vec, none of which was formulated to handle or align attributed graphs (which we study in Sec. 3.4.2.2). As we discuss further below, REGAL is one of the fastest network alignment methods, especially on large datasets, and has comparable or better accuracy than all baselines.

Results. (1) Accuracy. The accuracy results on several datasets are shown in Figure 3.4. The *structural* embedding REGAL variants consistently perform best. Both REGAL (matching our proposed xNetMF embeddings) and REGAL-struc2vec are significantly more accurate than all *non-representation learning baselines* across noise levels and datasets. As expected, REGAL-node2vec does hardly better than random chance because rather than preserving structural similarity, it preserves similarity to nodes based on their proximity to each other, which means there is no way of identifying similarity to corresponding nodes in *other*, disconnected graphs (even when we combine them into one large graph, because they form disconnected components.) This major limitation of embedding methods that use proximity-based node similarity criteria [HK17] justifies the need for *structural* embeddings for cross-network analysis.

Between REGAL and REGAL-struc2vec, the two highest performers, REGAL performs better with lower amounts of noise. This is likely because struc2vec's randomized context sampling introduces some variance into the representations that xNetMF does not have, as nodes that should match will have different embeddings not only because of noise, but also because they had different contexts sampled. With higher amounts of noise (4-5%), REGAL outperforms REGAL-struc2vec in speed, but at the cost of some accuracy. It is also worth noting that their accuracy margin is smaller for larger graphs. On larger datasets, our simple and fast logarithmic binning scheme (Step 1 in Sec. 3.3.1) provides a robust enough way of comparing nodes with high expected degrees. However, on small graphs with a few thousand nodes and edges, it appears that struc2vec's use of dynamic time warping (DTW) better handles misalignment of degree sequences from noise because it is a nonlinear alignment scheme. Still, we will see that REGAL is significantly faster than its struc2vec variant, since DTW is computationally expensive [RSF17], as is context sampling and SGNS training.

Observation 3.1. *Matching structural node embeddings leads to high network alignment accuracy. By avoiding the variance induced by random walks,* REGAL *obtains particularly high accuracy when noise is low.*

(2) Runtime. In Table 3.2, we compare the average runtimes of all different methods

across noise levels. We observe that REGAL scales significantly better using xNetMF than when using other node embedding methods. Notably, REGAL is 6-8× faster than REGAL-node2vec and 22-31× faster than REGAL-struc2vec. This is expected as both dynamic time warping (in struc2vec) and context sampling for SGNS (in struc2vec and node2vec) come with large computational costs. REGAL, at the cost of some robustness to high levels of noise, avoids both the variance and computational expense of random-walk-based sampling. This is a significant benefit that allows REGAL to achieve up to an order of magnitude speedup over the other node embedding methods. Additionally, REGAL is able to leverage the power of node representations and also use attributes, unlike the other representation learning methods.

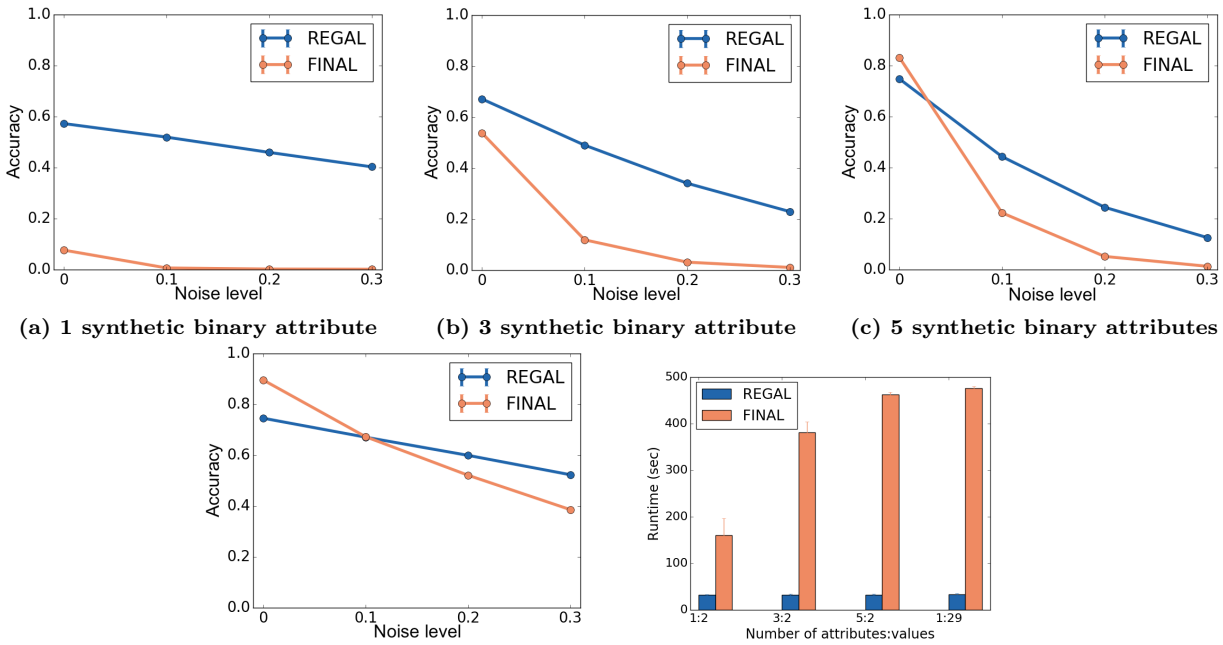
Comparing to baselines that do not use representation learning, we see that REGAL is competitive in terms of runtime as well as significantly more accurate. REGAL is consistently faster than FINAL and Klau, the next two best-performing methods by accuracy (NetAlign is virtually tied for third place with Klau on all datasets). Although NetAlign runs faster than REGAL on small datasets like Arenas, on larger datasets like Arxiv NetAlign's message passing becomes expensive. Finally, while IsoRank is consistently the fastest method, it performs among the worst on all datasets in accuracy. Thus, we can see that our REGAL framework is also one of the fastest network alignment methods as well as the most accurate.

Observation 3.2. REGAL *is faster than the most competitive baseline methods, particularly on large datasets.*

3.4.2.2 Effects of attribute-based noise

In the second experiment, we study REGAL's comparative sensitivity to p_a when we use node attributes. Here we compare REGAL to FINAL because it is the only baseline that handles attributes. We also omit embedding methods othen than xNetMF, since they operate on plain graphs.

We study a subnetwork of a larger DBLP collaboration network extracted in [ZT16] (Table 3.1). This dataset has 1 node attribute with 29 values, corresponding to the top conference in which each author (a node in the network) published. This single attribute



(a) 1 synthetic binary attribute **(b) 3 synthetic binary attribute** **(c) 5 synthetic binary attributes**

(d) Real attribute (29 values) **(e) Runtime with attributes**

Figure 3.5: DBLP Network alignment with varying p_a: REGAL is more robust to attribute noise (plots a-d) and runs faster (plot e) than FINAL for various numbers and types of attributes. In (e) the x-axis consists of <# of attributes: # of values> pairs corresponding to plots (a)-(d).

is quite discriminatory: with so many possible attribute values, a comparatively smaller number of nodes share the same value. We add $p_s = 0.01$ structural noise to randomly generated permutations.

We also increase attribute information by increasing the number of attributes. To do so, we simulate different numbers of binary attributes. We study somewhat higher levels of attribute noise, as they are not strictly required for network alignment.

Results. In Figure 3.5, we see that REGAL mostly outperforms FINAL in the presence of attribute noise (both for real and multiple synthetic attributes), *or* in the case of limited attribute information (e.g., only 1-3 binary attributes in Figure 3.5a-3.5c). This is because FINAL relies heavily on attributes, whereas REGAL uses structural and attribute information in a more balanced fashion.

While FINAL achieves slightly higher accuracy than REGAL with abundant attribute information from many attributes or attribute values and minimal noise (e.g. the real attribute with 29 values in Figure 3.5d, or 5 binary attributes in Figure 3.5c), this is expected due to FINAL's reliance on attributes. Also, in Figure 3.5e where we plot the runtime with

respect to <number of attributes : attribute values>, we see FINAL incurs significant runtime increases as it uses extra attribute information. Even without these added attributes, REGAL is up to two orders of magnitude faster than FINAL.

Observation 3.3. REGAL *is more robust to attribute noise and takes less time to use attribute information effectively compared to the existing attributed network alignment method FINAL.*

3.4.3 Scalability

To analyze the scalability of REGAL, we generate Erdös-Rényi graphs with $n = 100$ to 1,000,000 nodes and constant average degree 10, along with one binary attribute. We generate a randomized, noisy permutation ($p_s = 0.01, p_a = 0.05$) and look for the top $\alpha = 1$ alignments. Thus, we embed *both* graphs–double the number of nodes in a single graph. Figure 3.6 shows the runtimes for the major steps of our methods.

Results. We see that the total runtimes of REGAL's steps are clearly *sub-quadratic*, which is rare for alignment tasks. In practice this means that REGAL can scale to very large networks. The dominant step is computing $O(n \log n)$ similarities to landmarks in $\mathbf{C}$ and using this to form the Nyström-based representation. The alignment time complexity grows the most steeply, as the dimensionality p grows with the network size and increasingly affects lookup times. In practice, though, the alignment adds little overhead time, even for the largest graph, because of the k-d tree. *Without* it, REGAL runs out of memory on 100K or more nodes.

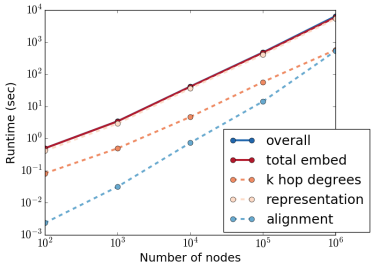
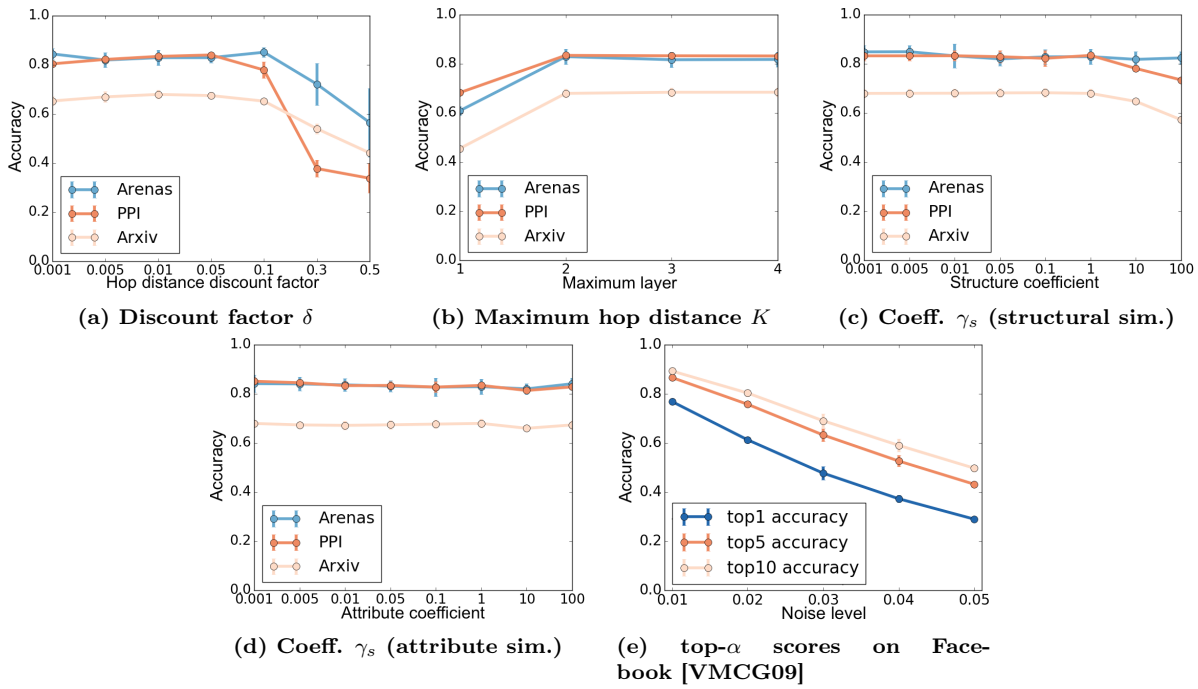


Figure 3.6: REGAL is sub-quadratic.

From a practical perspective, while our current implementation is single-threaded, many steps—including the expensive embedding construction and alignment steps—are easily and trivially parallelizable, offering possibilities for even greater speedups.

Observation 3.4. REGAL *scales sub-quadratically with respect to the input size and thus can handle extremely large graphs.*



(a) Discount factor δ **(b) Maximum hop distance K** **(c) Coeff. γ_s (structural sim.)**

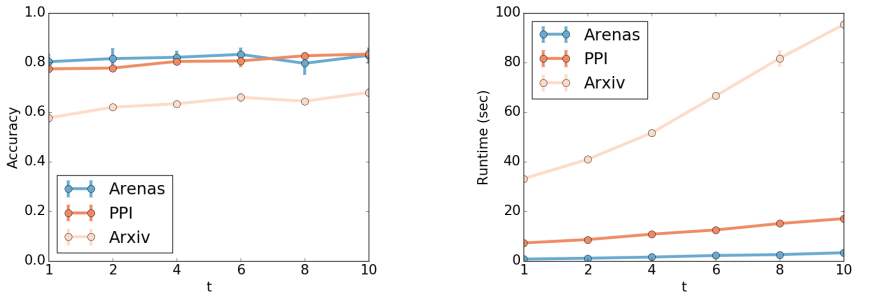
(d) Coeff. γ_s (attribute sim.) **(e) top-α scores on Facebook [VMCG09]**

Figure 3.7: Robustness of REGAL to hyperparameters on different datasets: REGAL is generally robust for a range of values, without fine tuning.

3.4.4 Sensitivity Analysis

To understand how REGAL's hyperparameters affect performance, we analyze accuracy by varying hyperparameters in several experiments. For brevity, we report results at $p_s = 0.01$ and with a single binary noiseless attribute, although further experiments with different settings yielded similar results. Overall we find that REGAL is **robust** to different settings and datasets, indicating that REGAL can be applied readily to different graphs without requiring excessive domain knowledge or fine-tuning.

Results. (1) Discount factor δ and max hop distance K. Figures 3.7a and 3.7b respectively show the performance of REGAL as a function of δ, the discount factor on further hop distances, and K, the maximum hop distance to consider. We find that some higher-order structural information does help (thus $K = 2$ performs slightly better than $K = 1$), but only up to a point. Beyond approximately 2 layers out, the structural similarity is so tenuous that it primarily adds noise to the neighborhood degree distribution (furthermore, computing further hop distances adds computational expense). Choosing δ between 0.01–0.1 tends to yield best performance. Larger discount factors δ tend to do poorly, though



(a) Accuracy w.r.t. # of landmarks **(b) Runtime w.r.t. # of landmarks**

Figure 3.8: Robustness of REGAL to t, which controls the number of landmarks $p = \lfloor t \log_2 n \rfloor$: choosing more landmarks is more computationally expensive but can slightly increase accuracy.

extremely small values may lose higher-order structural information.

(2) Weights of structural γ_s and attributed γ_a similarity. Next, we explore how to set the coefficients on the terms in the similarity function weighting structural and attribute similarity, which also governs a tradeoff between structural and attribute identity. In Figs. 3.7c and 3.7d we respectively vary γ_s and γ_a while setting the other to be 1. In general, setting these parameters to be 1, our recommended default value, does fairly well. Significantly larger values yield less stable performance.

(3) Dimensionality of embeddings p. To study the effects of the rank of the implicit low-rank approximation, which is also the dimensionality of the embeddings, we set the number of landmarks p equal to $\lfloor t \log_2 n \rfloor$ and vary t. Figure 3.8a shows that the accuracy is generally highest for the highest values of t, but Figure 3.8b shows the expected increase in REGAL's runtime as more similarities are computed in $\mathbf{C}$ and higher-dimensional embeddings are compared. To spare no expense in maximizing accuracy we use $t = 10$. However, fewer landmarks still yield almost as high accuracy if computational constraints or high dimensionality are issues.

(4) Top-α accuracy. It is worth studying not just the proportion of correct hard alignments, but also the top-α scores of the soft alignments that REGAL can return. We perform alignment without attributes on a large Facebook subnetwork [VMCG09] and visualize the top-1, top-5, and top-10 scores in Figure 3.7e. Across noise settings, the top-α scores are considerably several percentage points higher than the top-1 scores, indicating that even when REGAL misaligns a node, it often still recognizes the similarity of its true counterpart. REGAL's ability to find soft alignments could be valuable in many applications, like entity

resolution across social networks [KTL13].

Observation 3.5. *It is easy to set parameters for* REGAL *that achieve a balance of good performance and computational efficiency, making* REGAL *a good practical approach for network alignment.*

3.5 Conclusion

Motivated by the numerous applications of network alignment in social, natural, and other sciences, we proposed REGAL, a network alignment framework that leverages the power of node representation learning by aligning nodes via their learned embeddings. To efficiently learn node embeddings that are comparable across multiple networks, we introduced xNetMF within REGAL. To the best of our knowledge, we are the first to propose an unsupervised representation learning-based network alignment method.

Our embedding formulation captures node similarities using structural and attribute identity, making it suitable for cross-network analysis. Unlike other embedding methods that sample node context with computationally expensive and variance-inducing random walks, our extension of the Nyström low-rank approximation allows us to implicitly factorize a similarity matrix *without* having to fully construct it. Furthermore, we showed that our formulation is a matrix factorization perspective on the skip-gram objective optimized over node context sampled from a similarity graph. Experimental results showed that REGAL is up to 30% more accurate than baselines and 30× faster in the representation learning stage.

CHAPTER IV

Refining Network Alignment

4.1 Introduction

In this chapter, we continue to study the problem of unsupervised topological network alignment, in particular focusing on the setting where no node or edge attributes are available and network topology alone must be used to align the networks. With neither anchor links to seed the alignment process nor side information to guide it, the main objective for this task is to preserve some kind of topological consistency in the alignment solution. We theoretically analyze the principle of *matched neighborhood consistency* (MNC), or how well a node's neighborhood maps onto the neighborhood of its counterpart in the other graph (illustrated in Figure 4.1), and show its connection to alignment accuracy. On the other hand, we find that when network alignment methods are inaccurate, the MNC of their solutions breaks down (e.g., Figure 4.1 left).

To address this, we introduce RefiNA, a method for refining network alignment solutions *post hoc* by iteratively updating nodes' correspondences to improve their MNC. By strategically limiting the possible correspondences per node to update in each iteration, we can sparsify the computations to make RefiNA scalable to large graphs. The update rule of our method can be interpreted as a graph filtering procedure that smooths the cross-network alignments until they are consistent within local neighborhoods in each graph—a mechanism similar to the underpinnings of graph neural networks [WSZ+19].

Our method RefiNA can be succinctly expressed as matrix operations in a few lines of code, making it easy-to-adopt by practitioners. In this compact formulation, we incor-

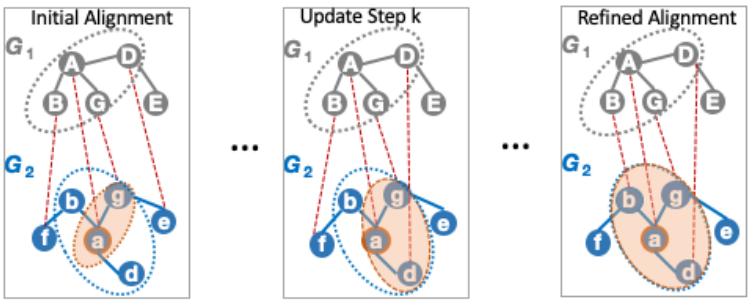


Figure 4.1: RefiNA refines an initial network alignment solution, which maps node A and its neighbors in G_1 far apart in G_2. The refined alignment solution has higher *matched neighborhood consistency*: neighbors of A are aligned to neighbors of a, to which A itself is aligned.

porate several insights that we verify are useful guiding principles for network alignment. Experimentally, we show that RefiNA significantly improves a variety of network alignment methods on many highly challenging datasets, even when starting a network alignment solution with limited accuracy. In particular, when paired with REGAL, it can be viewed as an alternative embedding comparison step for network alignment, one that is more accurate and robust than the greedy matching we performed individually for each node in Chapter III.

Our contributions are as follows:

- **New Algorithm**: We propose RefiNA, a post-processing step that can be applied to the output of any network alignment method. Its compact design incorporates several important insights for network alignment, and it permits a sparse approximation that is scalable to large graphs.

- **Theoretical Connections**: We show a rigorous connection between *matched neighborhood consistency*, the property that RefiNA improves, and alignment accuracy. We also connect RefiNA's technical underpinnings to other graph mining techniques.

- **Experiments**: We conduct thorough experiments on real and simulated network alignment tasks and show that RefiNA improves the accuracy of many methodologically diverse network alignment methods by up to 90%, making them robust enough to recover matchings in 5× noisier datasets than those considered in prior work. We extensively drill down RefiNA to justify the insights that inspire its design.

Code for RefiNA is available at `https://github.com/GemsLab/RefiNA`.

4.2 Theoretical Analysis

We start with the key definitions for network alignment that we use throughout the paper and a list of symbols in Table 4.1. Then, we theoretically justify topological consistency, which is leveraged in some network alignment techniques and is the basis of our refinement approach, RefiNA (§ 4.3).

Table 4.1: Major symbols and definitions.

Symbols	Definitions
$\pi(\cdot)$	An alignment between graphs G_1 and G_2; a function mapping a node in V_1 to a node in V_2
$\mathbf{M}$	$n_1 \times n_2$ matrix specifying correspondences of nodes in V_1 to those in V_2
$\mathcal{N}_G(i)$	Neighbors of node i in graph G
$\tilde{\mathcal{N}}^{\pi}_{G_2}(i)$	"Mapped neighborhood" of node i in G_2; counterparts in G_2 (mapped by π) of nodes in $\mathcal{N}_{G_1}(i)$

4.2.1 Preliminaries

Graphs. Following the network alignment literature [HSSK18, SM14], we consider two unweighted and undirected graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ with their corresponding nodesets V_1, V_2 and edgesets E_1, E_2. We denote their adjacency matrices as $\mathbf{A}_1$ and $\mathbf{A}_2$. Since they are symmetric, $\mathbf{A}_1^\top = \mathbf{A}_1$ and $\mathbf{A}_2^\top = \mathbf{A}_2$, and we simplify our notation below.

Alignment. Alignment is a function $\pi : V_1 \to V_2$ that maps the nodes of G_1 to those of G_2. It is also commonly represented as a $|V_1| \times |V_2|$ alignment matrix $\mathbf{M}$, where $\mathbf{M}_{ij}$ is the (real-valued or binary) similarity between node i in G_1 and node j in G_2. $\mathbf{M}$ can be used to encode a mapping π, e.g., greedy alignment $\pi(i) = \arg\max_j \mathbf{M}_{ij}$. We note that alignment between two graphs should be sought *if* the nodes of the two graphs meaningfully correspond.

Neighborhood & Consistency. Let $\mathcal{N}_{G_1}(i) = \{j \in V_1 : (i, j) \in E_1\}$ be the neighbors of node i in G_1, i.e., the set of all nodes with which i shares an edge. We define node i's "**mapped neighborhood**" in G_2 as the set of nodes onto which π maps i's neighbors: $\tilde{\mathcal{N}}^{\pi}_{G_2}(i) = \{j \in V_2 : \exists k \in \mathcal{N}_{G_1}(i) \text{ s.t. } \pi(k) = j\}$. For example, in Figure 4.1 (first panel), node A's neighbors in G_1 are $B, G,$ and D, which are respectively mapped to nodes $f, g,$ and e, so $\tilde{\mathcal{N}}^{\pi}_{G_2}(A) = \{f, g, e\}$.

Also, we denote the neighbors of i's counterpart as $\mathcal{N}_{G_2}\big(\pi(i)\big)$. In Figure 4.1, node A's counterpart is node a, whose neighbors are $b, g,$ and d. Thus, $\mathcal{N}_{G_2}\big(\pi(A)\big) = \{b, g, d\}$.

Using the terminology of [CHVK20], **matched neighborhood consistency** (MNC) of node i in G_1 and node j in G_2 is the Jaccard similarity of the sets $\tilde{\mathcal{N}}^{\pi}_{G_2}(i)$ and $\mathcal{N}_{G_2}(j)$:

$$\text{MNC}(i, j) = \frac{|\tilde{\mathcal{N}}^{\pi}_{G_2}(i) \cap \mathcal{N}_{G_2}(j)|}{|\tilde{\mathcal{N}}^{\pi}_{G_2}(i) \cup \mathcal{N}_{G_2}(j)|}. \tag{4.1}$$

4.2.2 Justification of Matched Neighborhood Consistency

Several unsupervised network alignment algorithms attempt to enforce some notion of topological consistency in their objective functions. We justify this intuition by showing that a specific form of topological consistency, matched neighborhood consistency or MNC, has a close relationship with alignment accuracy.

Our first result considers alignment of (corrupted) isomorphic graphs. It is common to evaluate unsupervised network alignment in such a paradigm, where the graphs are isomorphic except one graph has noisy or missing edges compared to the other [HSSK18, DKL+19, SM14, KTL13, ZT16, ZTT+17]. When edges are removed independently with probability p, we show that an accurate network alignment entails high MNC.

Theorem 4.1. *For isomorphic graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$, let $\pi(\cdot)$ be the isomorphism. Let $\overline{G}_2 = (V_2, \tilde{E}_2)$ be a noisy version of G_2 created by removing each edge from E_2 independently with probability p. Then for any node i in G_1 and its counterpart $\pi(i)$ in $\overline{G}_2$, $\mathbb{E}\big(MNC(i, \pi(i))\big) = 1 - p$.*

Proof. By Eq. (4.1), $\text{MNC}(i, \pi(i)) = \frac{|\tilde{\mathcal{N}}^{\pi}_{\overline{G}_2}(i) \cap \mathcal{N}_{\overline{G}_2}(\pi(i))|}{|\tilde{\mathcal{N}}^{\pi}_{\overline{G}_2}(i) \cup \mathcal{N}_{\overline{G}_2}(\pi(i))|}$. By definition, $\tilde{\mathcal{N}}^{\pi}_{G_2}(i) = \tilde{\mathcal{N}}^{\pi}_{\overline{G}_2}(i)$; it does not change as neither π nor G_1's adjacency matrix $\mathbf{A}_1$ is affected by the noise. However, $\mathcal{N}_{\overline{G}_2}(\pi(i)) \subseteq \mathcal{N}_{G_2}(\pi(i))$, since under edge removal $\pi(i)$ can only lose neighbors in $\overline{G}_2$ compared to G_2.

Now $\tilde{\mathcal{N}}^{\pi}_{G_2}(i) = \mathcal{N}_{G_2}(\pi(i))$ since by definition an isomorphism is edge preserving, and so $\mathcal{N}_{G_2}(\pi(i)) = \tilde{\mathcal{N}}^{\pi}_{G_2}(i)$, which is the same as $\tilde{\mathcal{N}}^{\pi}_{\overline{G}_2}(i)$. Thus, $\mathcal{N}_{\overline{G}_2}(\pi(i)) \subseteq \tilde{\mathcal{N}}^{\pi}_{\overline{G}_2}(i)$. We can simplify $\text{MNC}(i, \pi(i)) = \frac{|\mathcal{N}_{\overline{G}_2}(\pi(i))|}{|\tilde{\mathcal{N}}^{\pi}_{\overline{G}_2}(i)|} = \frac{|\mathcal{N}_{\overline{G}_2}(\pi(i))|}{|\mathcal{N}_{G_2}(\pi(i))|}$. However, every node $j' \in \mathcal{N}_{G_2}(\pi(i))$ is also in $\mathcal{N}_{\overline{G}_2}(\pi(i))$ as long as the edge $\big(\pi(i), j'\big)\big) \in E_2$ has not been removed from $\tilde{E}_2$, which happens with probability p. So, $\mathbb{E}\Big(\frac{|\mathcal{N}_{\overline{G}_2}(\pi(i))|}{|\mathcal{N}_{G_2}(\pi(i))|}\Big) = \mathbb{E}\big(\text{MNC}(i, \pi(i))\big) = 1 - p$. □

However, this result does not prove that a solution with high MNC will have high accuracy. In fact, we can construct examples where a solution can have perfect MNC and still misaligned nodes. One such (simple) example involves two "star" graphs, each consisting of one central node connected to $n-1$ peripheral nodes (of degree one). Whatever the true correspondence of the peripheral nodes, aligning them to each other in any order would lead to perfect MNC. Prior network alignment work [KTL13] has observed a few such special cases and in fact gives up on trying to distinguish such nodes from a purely topological perspective, simply compressing nodes it cannot hope to distinguish into supernodes.

We formalize these specially challenging cases [KTL13] under the concept of *structural indistinguishability*:

Definition 4.1. *Let $\mathcal{N}_k(u)$ be the subgraph induced by all nodes that are k or fewer hops/steps away from node u. Two nodes u and v are structurally indistinguishable if for all k, $\mathcal{N}_k(u)$ and $\mathcal{N}_k(v)$ are isomorphic.*

Note that nodes in the same graph or different graphs can be structurally indistinguishable, since all we require is an isomorphism between their neighborhoods and not that they have exactly the same neighbors. Our next result proves that for isomorphic graphs, structurally indistinguishable nodes are the only possible failure case for a solution with perfect MNC.

Theorem 4.2. *For isomorphic graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$, suppose there exists $\pi(\cdot)$ that yields MNC = 1 for all nodes. Then, if π misaligns a node v to some node v^* instead of the true counterpart v', it is because v^* is structurally indistinguishable from v'.*

Proof. Since for isomorphic graphs, a node v is structurally indistinguishable from its true counterpart v', and since graph isomorphism is transitive, it suffices to show that v^* is also structurally indistinguishable from v. Suppose for some k, $\mathcal{N}_k(v)$ is not isomorphic to $\mathcal{N}_k(v^*)$. Then by definition there exists neighboring nodes $a, b \in \mathcal{N}_k(v)$ where either $\pi(a)$ or $\pi(b)$ is not in $\mathcal{N}_k(v^*)$, or $\pi(a)$ and $\pi(b)$ do not share an edge.

In case 1, without loss of generality $\pi(b) \notin \mathcal{N}_k(v^*)$. Then no bijective mapping exists between a shortest path between v^* and $\pi(b)$ and a shortest path from v^* to $\pi(b)$. There

will thus be neighbors on one path whose counterparts are not neighbors on the other path, making MNC less than 1: a contradiction.

In case 2, since $\pi(b)$ is the counterpart of a neighbor of a, it must also be a neighbor of the counterpart of a, which is a contradiction of the assumption that $\pi(a)$ and $\pi(b)$ do not share an edge, or else $\text{MNC}(a, \pi(a)) < 1$, another contradiction. Thus, we conclude that the k-hop neighborhoods are isomorphic, that is, v and v^* are structurally indistinguishable. □

Formalizing the intuitions of prior work: MNC was recently used to motivate the *intuition* of using node embeddings to preserve intra-graph proximity even when the goal is to align them across graphs [CHVK20]. Before that, it was also used (not by that name) as a *heuristic* in a recent work on embedding-based network alignment [DYZ19], which multiplied the embedding similarities of nodes in different graphs by the nodes' MNC (using the embedding-based mapping π). Our analysis provides a theoretical justification for both works.

4.3 RefiNA

We consider an *unsupervised* network alignment setting, where an initial alignment solution, $\mathbf{M}_0$, is provided by any network alignment method. Our goal is to improve the accuracy of this solution by leveraging insights from our theoretical analysis in Section 4.2.2. We note that this setup is different from network alignment with ground-truth or seed node correspondences known *a priori* [LCLL16, ZTX$^+$19] since the initial solution does not contain information about which or how many nodes it aligns correctly. Formally, we tackle:

Problem 4.1. *Given a sparse initial alignment matrix* $\mathbf{M}_0$ *between the nodes of two graphs* G_1 *and* G_2*, we seek to* refine *this initial solution into a new, real-valued matrix* $\mathbf{M}$ *of refined similarity scores that encodes a more accurate alignment.*

4.3.1 RefiNA: Improving Matched Neighborhood Consistency

Our theoretical results pave a path to solving Problem 4.1 by increasing the initial solution's MNC. While our results characterize "ideal" cases (perfect accuracy or MNC),

heuristic solutions in prior works have found that increasing MNC tends to increase accuracy [CHVK20, DYZ19].

Given an alignment matrix returned by a network alignment method, how can we improve its MNC in a *principled* way? We first derive a matrix-based form for MNC:

Theorem 4.3. *For binary alignment matrix* $\mathbf{M}$*, its MNC can be written as a matrix* $\mathbf{S}^{MNC}$ *such that* $MNC(i,j) = \mathbf{S}^{MNC}_{ij}$ *as:*

$$\mathbf{S}^{MNC} = \mathbf{A}_1\mathbf{M}\mathbf{A}_2 \oslash (\mathbf{A}_1\mathbf{M}\mathbf{1}^{n_2} \otimes \mathbf{1}^{n_2} + \mathbf{1}^{n_1} \otimes \mathbf{A}_2\mathbf{1}^{n_2} - \mathbf{A}_1\mathbf{M}\mathbf{A}_2) \tag{4.2}$$

where $\oslash$ *denotes Hadamard or elementwise division and* $\otimes$ *is outer product.*

Proof. $\tilde{\mathcal{N}}^{\pi}_{G_2}(i) = \{\ell : \exists k \in V_1 \text{ s.t. } \mathbf{A}_{1_{ik}}\mathbf{M}_{k\ell} \neq 0\}$, and of course $\mathcal{N}_{G_2}(j) = \{\ell : \mathbf{A}_{2_{j\ell}} \neq 0\}$. Since the product of two numbers is nonzero if and only if both numbers are nonzero, $\tilde{\mathcal{N}}^{\pi}_{G_2}(i) \cap \mathcal{N}_{G_2}(j) = \{\ell : \mathbf{A}_{1_{ik}}\mathbf{M}_{k\ell}\mathbf{A}_{2_{j\ell}} \neq 0\}$. For binary $\mathbf{A}_1, \mathbf{A}_2$, and $\mathbf{M}$, the cardinality of this set, which is the numerator of Eq. (4.1), is $\sum_{k\in V_1, \ell \in V_2} \mathbf{A}_{1_{ik}}\mathbf{M}_{k\ell}\mathbf{A}_{2_{j\ell}} = (\mathbf{A}_1\mathbf{M}\mathbf{A}_2)_{ij}$. Meanwhile, the denominator of Eq. (4.1) is $|\tilde{\mathcal{N}}^{\pi}_{G_2}(i) \cup \mathcal{N}_{G_2}(j)| = |\tilde{\mathcal{N}}^{\pi}_{G_2}(i)| + |\mathcal{N}_{G_2}(j)| - |\tilde{\mathcal{N}}^{\pi}_{G_2}(i) \cap \mathcal{N}_{G_2}(j)|$. Plugging in for each individual term, we obtain $\sum_{k\in V_1}\sum_{\ell\in V_2} \mathbf{A}_{1_{ik}}\mathbf{M}_{k\ell} + \sum_{\ell\in V_2}\mathbf{A}_{2_{j\ell}} - \sum_{k\in V_1}\sum_{\ell\in V_2}\mathbf{A}_{1_{ik}}\mathbf{M}_{k\ell}\mathbf{A}_{2_{j\ell}}$. Substituting matrix products, this becomes $\sum_{\ell\in V_2}(\mathbf{A}_1\mathbf{M})_{i\ell} + \sum_{\ell\in V_2}\mathbf{A}_{2_{j\ell}} - (\mathbf{A}_1\mathbf{M}\mathbf{A}_2)_{ij}$. Using all-1 vectors to sum over columns, this is $(\mathbf{A}_1\mathbf{M}\mathbf{1}^{n_2})_i + (\mathbf{A}_2\mathbf{1}^{n_2})_j - (\mathbf{A}_1\mathbf{M}\mathbf{A}_2)_{ij}$. Then, expanding the two left vectors into matrices with outer product: $(\mathbf{A}_1\mathbf{M}\mathbf{1}^{n_2} \otimes \mathbf{1}^{n_2})_{ij} + (\mathbf{1}^{n_1} \otimes \mathbf{A}_2\mathbf{1}^{n_2})_{ij} - (\mathbf{A}_1\mathbf{M}\mathbf{A}_2)_{ij}$. Adding everything together, the denominator is the ij-th entry of the matrix $\mathbf{A}_1\mathbf{M}\mathbf{1}^{n_2} \otimes \mathbf{1}^{n_2} + \mathbf{1}^{n_1} \otimes \mathbf{A}_2\mathbf{1}^{n_2} - \mathbf{A}_1\mathbf{M}\mathbf{A}_2$. □

Given this notation, we can compute refined alignments $\mathbf{M}'$ by performing a multiplicative updating of each node's alignment score (in $\mathbf{M}$) with its matched neighborhood consistency:

$$\mathbf{M}' = \mathbf{M} \circ \mathbf{S}^{\text{MNC}} \tag{4.3}$$

where $\circ$ denotes Hadamard product. Repeating over several iterations can take advantage of an improving alignment solution. The high-level idea of our proposed refinement scheme is to **iteratively increase alignment scores for nodes that have high matched neighborhood consistency**.

The simplest possible refinement algorithm is to repeatedly iterate Eq. (4.3). However, we can introduce some mechanisms that maintain this simplicity while leveraging a set of insights:

• `I1`: **Prioritize high-degree nodes**. Higher degree nodes are easier to align [KTL13], and so it is desirable to give them higher scores particularly in early iterations. To do so, we use only the (elementwise) numerator of Eq. (4.2), which counts the number of matched neighbors shared by a pair of nodes (the denominator normalizes by the neighborhood size, so excluding it increases the score for high degree nodes). Thus, instead of using Eq. (4.3), we simplify our update rule to:

$$\mathbf{M}' = \mathbf{M} \circ \mathbf{A}_1\mathbf{M}\mathbf{A}_2 \tag{4.4}$$

Additionally, this spares us the excess matrix operations required to compute the denominator of Eq. (4.2).

• `I2`: **Do not overly rely on the initial solution**. At every iteration, we add a small ϵ to every element of $\mathbf{M}$. This gives each pair of nodes a token match score whether or not the initial alignment algorithm identified them as matches, which helps us correct the initial solution's false negatives.

• `I3`: **Allow convergence.** Finally, to keep the scale of the values of $\mathbf{M}$ from exploding, we normalize the rows and columns of $\mathbf{M}$. Specifically, we row-normalize $\mathbf{M}$ followed by column-normalizing it at every iteration. While previous methods [GR96] require full normalization per iteration using the Sinkhorn algorithm [Sin64] to produce a doubly stochastic matrix, with RefiNA a single round of normalization suffices and avoids this much greater computational expense (cf. Section 4.4).

Putting it all together, we give the pseudocode of our method RefiNA in Algorithm 4.1. RefiNA is powerful and significantly improves the accuracy of a wide range of alignment methods (cf. experiments in Section 4.4), yet it remains conceptually simple and straightforward to implement. It requires only a few lines of code, with each line implementing an important insight.

Complexity. To simplify notation, we assume that both graphs have n nodes [HSSK18]. For K iterations, our algorithm computes the left and right multiplication of a dense $n \times n$

Algorithm 4.1 RefiNA

`Input`: adjacency matrices $\mathbf{A}_1, \mathbf{A}_2$, initial alignment matrix $\mathbf{M}_0$, number of iterations K, token match score ϵ
`Output`: Refined alignment matrix $\mathbf{M}_K$

1: **for** $k = 1 \to K$ **do** ▷ Number of iterations
2: $\quad \mathbf{M}_k = \mathbf{M}_{k-1} \circ \mathbf{A}_1\mathbf{M}_{k-1}\mathbf{A}_2$ ▷ MNC-based update
3: $\quad \mathbf{M}_k = \mathbf{M}_k + \epsilon$ ▷ Add token match scores
4: $\quad \mathbf{M}_k = \text{RowNormalize}(\mathbf{M}_k)$
5: $\quad \mathbf{M}_k = \text{ColumnNormalize}(\mathbf{M}_k)$
6: **return** $\mathbf{M}_K$

matching matrix with two adjacency matrices of graphs with average degree (number of nonzero entries per row of the adjacency matrix) Δ_{avg_1} and Δ_{avg_2}, respectively. Thus, the time complexity of this update step is $O(n^2(\Delta_{\text{avg}_1} + \Delta_{\text{avg}_2}))$. Normalizing the matrix at each iteration and adding in token match scores requires $O(n^2)$ time. Therefore, the overall time complexity is $O\Big(Kn^2(\Delta_{\text{avg}_1} + \Delta_{\text{avg}_2})\Big)$. While the number of iterations and average node degree are in practice constant or asymptotically smaller than the graph size, the quadratic time and space complexity of RefiNA's dense matrix operations makes it harder to apply to large graphs. We discuss a faster variant of RefiNA next.

4.3.2 Optimizations: Sparse RefiNA

To scale RefiNA to larger graphs, we seek to sparsify it by updating only a small number of alignment scores for each node. Intuitively, we can afford to forego updating alignment scores of pairs of nodes when the updates are small and thus the nodes likely do not align. Concretely, sparse RefiNA replaces Line 3 of Algorithm 4.1 with

$$\mathbf{M}_{k|\mathbf{U}_k} = \mathbf{M}_{k-1|\mathbf{U}_k} \circ \mathbf{U}_k$$

where the update matrix $\mathbf{U}_k = \text{top-}\alpha(\mathbf{A}_1\mathbf{M}\mathbf{A}_2)$ is a sparse version of $\mathbf{A}_1\mathbf{M}\mathbf{A}_2$ containing only the largest α entries per row. $\mathbf{M}_{k|\mathbf{U}_k}$ selects the elements of $\mathbf{M}_k$ (pairs of nodes) corresponding to nonzero elements in $\mathbf{U}_k$. These are the only elements on which we perform an MNC-based update, and the only ones to receive a token match score (Line 4):

$$\mathbf{M}_{k|\mathbf{U}_k} = \mathbf{M}_{k|\mathbf{U}_k} + \epsilon$$

As the elements to update are selected by the size of their update scores, which are

computed using the previous alignment solution $\mathbf{M}$, sparse RefiNA relies somewhat more strongly on the initial solution. However, we still comply with `I2` by updating multiple, i.e. α possible alignments for each node. In practice, we show (§4.4) that a small α gives comparable accuracy to the dense update and is much faster.

Complexity. For K iterations, we compute a sparse update matrix with $O(n\alpha)$ nonzero entries by multiplying matrices with $O(n\Delta_{\text{avg}_1})$, $O(Kn\alpha)$, and $O(n\Delta_{\text{avg}_2})$ nonzero entries, respectively. It takes $O(nK\alpha\Delta_{\text{avg}_1})$ time to compute $\tilde{\mathbf{A}}_1 = \mathbf{A}_1\mathbf{M}_{k-1}$ and $O(nK\alpha\Delta_{\text{avg}_1}\Delta_{\text{avg}_2})$ time to compute $\tilde{\mathbf{A}}_1\mathbf{A}_2$. We then compute $\mathbf{M}_k$ by updating $O(n\alpha)$ entries in $\mathbf{M}_{k-1}$ per iteration. Thus, $\mathbf{M}_k$ may have $O(Kn\alpha)$ nonzero entries and requires $O(Kn\alpha)$ time to update and normalize. Altogether, the runtime is now $O(nK^2\alpha\Delta_{\text{avg}_1}\Delta_{\text{avg}_2})$, i.e. linear in the number of nodes. (This is a worst-case analysis and in practice the runtime scales close to linearly with K.) We can also avoid storing a dense matching matrix, leading to subquadratic space complexity.

4.3.3 Theoretical Connections to Other Graph Methods

We show additional connections between RefiNA and other diverse graph methods: first, *seed-and-extend* as an alignment strategy, and second a graph filtering perspective on RefiNA's update rule similar to the analysis of graph neural networks.

Seed-and-extend alignment heuristic. Many global network alignment methods [KMM+10, PK12] use this heuristic to find node correspondences between two or multiple networks. Given initial pairwise similarities (or alignment costs) between nodes of the compared graphs as $\mathbf{M}$, a pair of nodes i and j with high probability to be aligned (e.g., whose similarity according to $\mathbf{M}$ is above some confidence threshold) are set as the seed regions of the alignment. After the seed (i, j) is selected, the k-hop neighborhoods of i and j (i.e., $\mathcal{N}^k_{i,G_1}$ and $\mathcal{N}^k_{j,G_2}$) in their respective graphs are built. Next, the selected seed (i, j) is extended for the final alignment $\mathbf{M}'$ by greedily matching nodes in $\mathcal{N}^k_{i,G_1}$ and $\mathcal{N}^k_{j,G_2}$, searching for the pairs $(i', j') : i' \in \mathcal{N}^k_{i,G_1}$ and $j' \in \mathcal{N}^k_{j,G_2}$ that are not already aligned and can be aligned with the maximum value of similarity according to $\mathbf{M}$. The process can be written as $\forall v \in \mathcal{N}^k_{i,G_1}, \mathbf{M}'_{v\ell} \neq 0$ if $\ell = \arg\max_{\ell \in \mathcal{N}^k_{j,G_2}} \mathbf{M}_{v\ell}$.

By setting $k = 1$ to consider each seed's direct neighbors, and $\mathbf{M}'_{u^*v^*} \neq 0$ if $u^*, v^* =$

$\arg\max_{u \in V_1, v \in V_2} \mathbf{A}_{1_{iu}} \mathbf{M}_{uv} \mathbf{A}_{2_{jv}}$, we see that the seed-and-extend heuristic analyzes the same set of elements used to compute the update in RefiNA (Eq. (4.4)). However, instead of summing them to update the similarity of seed nodes i and j, it takes the argmax over them to adaptively select the next pair of alignments. Thus, seed-and-extend aligns less well with `I1` by relying heavily on a correct initial solution, as the early alignments are irrevocable and used to restrict the scope of subsequent alignments.

Graph Filtering. The matching matrix $\mathbf{M}$ can also be interpreted as a high-dimensional feature matrix. For example, for each node in G_1, a row of $\mathbf{M}$ may be regarded as an n_2-dimensional feature vector consisting of the node correspondences to each of the nodes in G_2, and similarly the $n_2 \times n_1$ matrix $\mathbf{M}^\top$ contains n_1-dimensional cross-network correspondence features for each node in G_2. For challenging alignment scenarios, these are likely highly noisy features. However, recent works have shown that multiplying a node feature matrix by the graph's adjacency matrix corresponds to a low-pass filtering operation, which is of interest in explaining the mechanisms of graph neural networks [WSZ+19].

We can write our update rule in Eq. (4.4) as a feature matrix left multiplied by adjacency matrices: $\mathbf{A}_1\mathbf{M}\mathbf{A}_2 = \mathbf{A}_1(\mathbf{A}_2\mathbf{M}^\top)^\top$ (for undirected graphs, $\mathbf{A}_2^\top = \mathbf{A}_2$), where $\mathbf{A}_2\mathbf{M}^\top$ produces a filtered set of n_1-dimensional features. By taking the transpose, these may be interpreted as n_2-dimensional features for each node of G_1, which are then filtered again by left multiplication with $\mathbf{A}_1$.[1]

Interpreting RefiNA's updates as graph filtering explains its strong performance, as well as the success of recent supervised graph matching work [ZTX+19, FLM+20] using graph neural networks. Of course RefiNA does not have the learnable parameters and nonlinearities of a graph neural network. However, just as SGC [WSZ+19] recently compares favorably to graph neural networks by replacing their deep nonlinear feature extraction with repeated multiplication by the adjacency matrix, we find that that unsupervised "alignment filtering" is highly effective.

[1] Graph convolutional networks use the augmented normalized adjacency matrix $\tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}$ where $\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}$, which we have not found helpful.

4.4 Experiments

In this section, we first seek to demonstrate the effectiveness of RefiNA: its ability to improve a diverse range of established network alignment methods in a variety of challenging alignment scenarios at reasonable computational cost. We next perform a deep study of RefiNA that verifies the various insights that inspired its design.

4.4.1 Experimental Setup

Data. We choose network datasets from a variety of domains (biological, social, communication networks) and levels of sparsity (Table 4.2).

Table 4.2: Description of the datasets used.

Name	Nodes	Edges	Description
Arenas Email [Kun13]	1 133	5 451	communication network
Hamsterster [Kun13]	2 426	16 613	social network
PPI-H [BSR+08]	3 890	76 584	PPI network (Human)
Facebook [LK14]	4 039	88 234	social network
PPI-Y [SM14]	1 004	8 323	PPI network (Yeast)
LiveMocha [Kun13]	104 103	2 193 083	social network

We consider two scenarios for network alignment: **(1)** ***Simulated noise.*** To create a controlled experiment with networks that exhibit real-world structure yet have a known ground truth alignment, we follow an experimental procedure used in many prior works [HSSK18, CHVK20, DKL+19, ZTT+17]. For each network with adjacency matrix $\mathbf{A}$, we create a random permuted copy $\mathbf{A}^* = \mathbf{PAP}^\top$, where $\mathbf{P}$ is a random permutation matrix. To make the problem more challenging, we add noise by removing each edge with probability $p \in [0.05, 0.10, 0.15, 0.20, 0.25]$. [2] The task is to align each network to its *noisy* permuted copy $\mathbf{A}^{*(p)}$, with the ground truth alignments being given by $\mathbf{P}$. **(2)** ***Real noise.*** Our PPI-Y dataset is a collection of protein-protein interaction (PPI) networks that is commonly used to evaluate biological network alignment [SM14]. The original network is the largest connected component of the yeast (*S.cerevisiae*) PPI network, which has 1,004 proteins (nodes) and

[2] Note that this is 5× more noisy than [HSSK18], which considered noise levels between 1% and 5%. We want to demonstrate that RefiNA can make network alignment possible even in considerably more noisy scenarios.

8,323 PPIs (edges). The standard task is to align this network to copies of itself augmented with 5, 10, 15, 20, and 25 percent additional low-confidence PPIs (added in order of their confidence) [SM14].

Base Network Alignment Methods. As we cannot exhaustively evaluate every network alignment method, we choose a set of base network alignment methods to pair with RefiNA that represent a diverse range of techniques (belief propagation, spectral methods, genetic algorithms, and node embeddings): **(1) NetAlign** [BGSW13], **(2) FINAL** [ZT16], **(3) REGAL** [HSSK18], **(4) CONE-Align** [CHVK20], and **(5) MAGNA** [SM14]. We consider the output of all methods to be a binary matrix $\mathbf{M}$ consisting of the "hard" (one-to-one) alignments they find, to treat methods consistently and to show that RefiNA is capable of refining the most general network alignment solution. It is worth noting that some methods (e.g. REGAL, CONE-Align) can also produce "soft" alignments (real-valued node similarity scores) and our formulation is capable of using those.

Settings of Base Methods. For REGAL's own xNetMF embeddings, we used default embedding dimension $\lfloor 10 \log_2(n_1 + n_2) \rfloor$ [HSSK18], maximum neighborhood distance 2, neighborhood distance discount factor $\delta = 0.1$, and resolution parameter $\gamma_{\text{struc}} = 1$, all recommended parameters. For the NetMF [QDM$^+$18] embeddings used in CONE-Align, we set embedding dimension $d = 128$, context window size $w = 10$, and negative sampling parameter $\alpha = 1$. We used $n_0 = 10$ iterations and regularization parameter $\lambda_0 = 1.0$ for the convex initialization of the subspace alignment, which we performed with $T = 50$ iterations of Wasserstein Procrustes optimization with batch size $b = 10$, learning rate $\eta = 1.0$, and regularization parameter $\lambda = 0.05$ as were suggested by the authors [CHVK20]. For REGAL and CONE-Align, we computed embedding similarity with dot product followed by softmax normalization [FLM$^+$20], using k-d trees to perform fast 10-nearest neighbor search for REGAL on LiveMocha [HSSK18].

NetAlign and FINAL require a matrix of prior alignment information, which we computed from pairwise node degree similarity. Then following [HSSK18, DKL$^+$19], we constructed this matrix by taking the top k ($k = \log_2 n_1$) entries for each node in G_1; that is, the top k most similar nodes in G_2 by degree.

For MAGNA, starting from a random initial population with size 15000, we simulated the

evolution for 2000 steps following [SM14] using edge correctness as the optimizing measure as it is similar to the objectives of our other methods. We used 3 threads to execute the alignment procedure.

• *RefiNA Settings.* By default, we perform $K = 100$ refinement iterations and set the token match score ϵ to be the reciprocal of the smallest power of 10 larger than the number of nodes, which ensures that the token scores added to a node's matches will sum to less than 1 and thus will not drown out the actual update scores. For sparse refinement, we update $\alpha = 10$ entries per node. We justify these choices by sensitivity analysis.

Hardware Configurations. We ran all experiments on an Intel(R) Xeon(R) CPU E5-1650 at 3.50GHz, 256GB RAM.

Metrics. Our primary metric of alignment success is **alignment accuracy**: the number of correctly aligned nodes. We consider datasets where ground truth alignments are known so that we can objectively measure how successfully the nodes have been matched. While the primary goal of network alignment is to uncover the true alignments, we also give **matched neighborhood consistency** (MNC, Eq. (4.1)) as a secondary metric, as our analysis showed its importance (Section 4.2.2).

4.4.2 Alignment Performance: Simulated-noise Scenarios

In Figure 4.2 we report the alignment accuracy of all network alignment methods both with (solid/dashed lines) and without (dotted lines) refinement. For each dataset and each level of noise, we plot the average accuracy and standard deviation across five random alignment scenarios for the simulated noise experiments.

Results. Across all datasets, the accuracy of all methods improves dramatically with refinement. A striking example of this is NetAlign on the PPI-H dataset with 5% noise. Getting just 4% of alignments initially correct, after refinement with RefiNA it achieves 94% accuracy–going from a nearly completely wrong to a nearly completely correct solution. Similarly, CONE-Align achieves well over 90% accuracy on Arenas and PPI-H with refinement *and* sees only a slight drop even at the highest noise levels.

Observation 4.1. *RefiNA greatly improves the accuracy of diverse network alignment meth-*

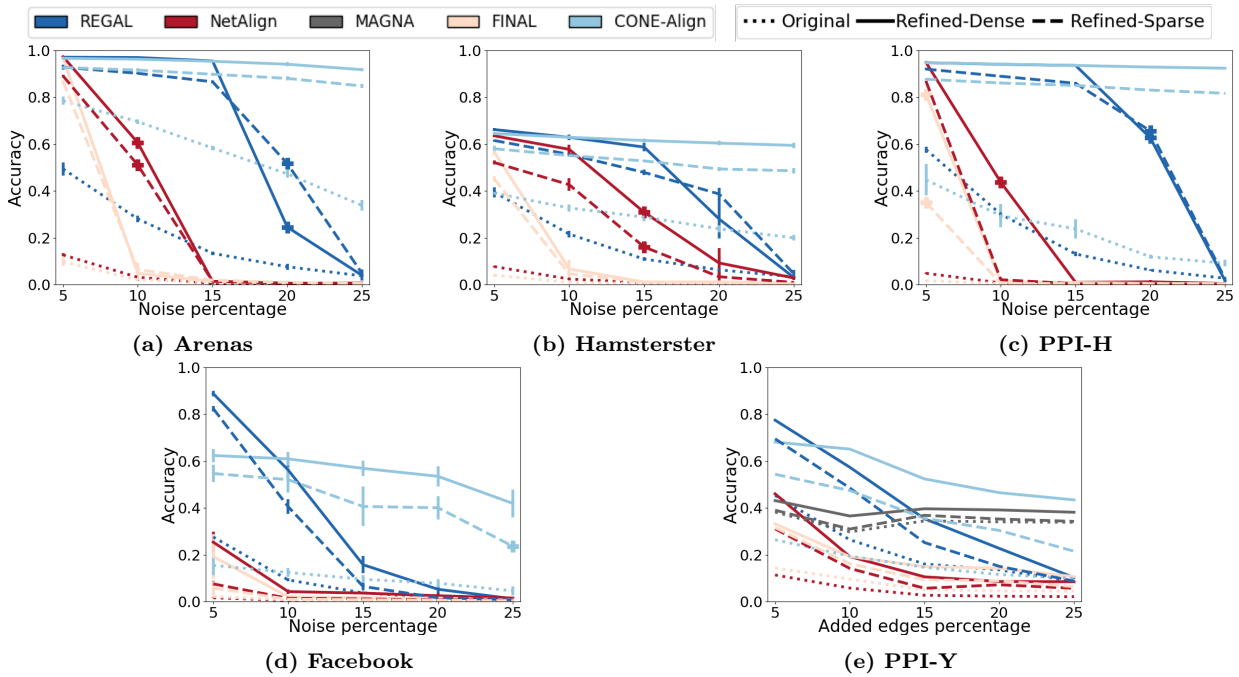


Figure 4.2: Alignment accuracy as a function of topological difference via synthetically generated noise in (a-d) or added low-confidence interactions in (e). With all different base methods and noise levels, refinement with RefiNA improves alignment accuracy, in many cases quite dramatically. While all alignment methods improve in accuracy, embedding-based methods in particular become very robust to noise. Sparse refinement, although slightly less accurate at low noise levels, is in some cases actually *more* accurate at high noise levels. (We run MAGNA only on PPI-Y, the dataset with real noise, due to its high runtime—cf. Figure 4.3b.)

ods across datasets.

We are also able to align networks that are much noisier than those that previous works had considered. Note that our *lowest* level of noise is 5%, which corresponds to the *highest* level of noise in [HSSK18]. With refinement, it is possible to get meaningfully better results at up to 10% noise on most datasets with FINAL, up to 15% noise with NetAlign, up to 20% with REGAL, and up to the full 25% noise that we consider with CONE-Align.

Observation 4.2. *Refinement with RefiNA can make network alignment methods considerably more robust to noise.*

We summarize our results in Table 4.3, where we show the maximum improvement in mean accuracy thanks to RefiNA for each base method on each dataset (across refinement types and noise levels, averaged over five trials). We also show the maximum noise level at which RefiNA is able to yield a noticeable improvement (3% or more). Different methods not only improve by different amounts from RefiNA, but also the maximum noise level at

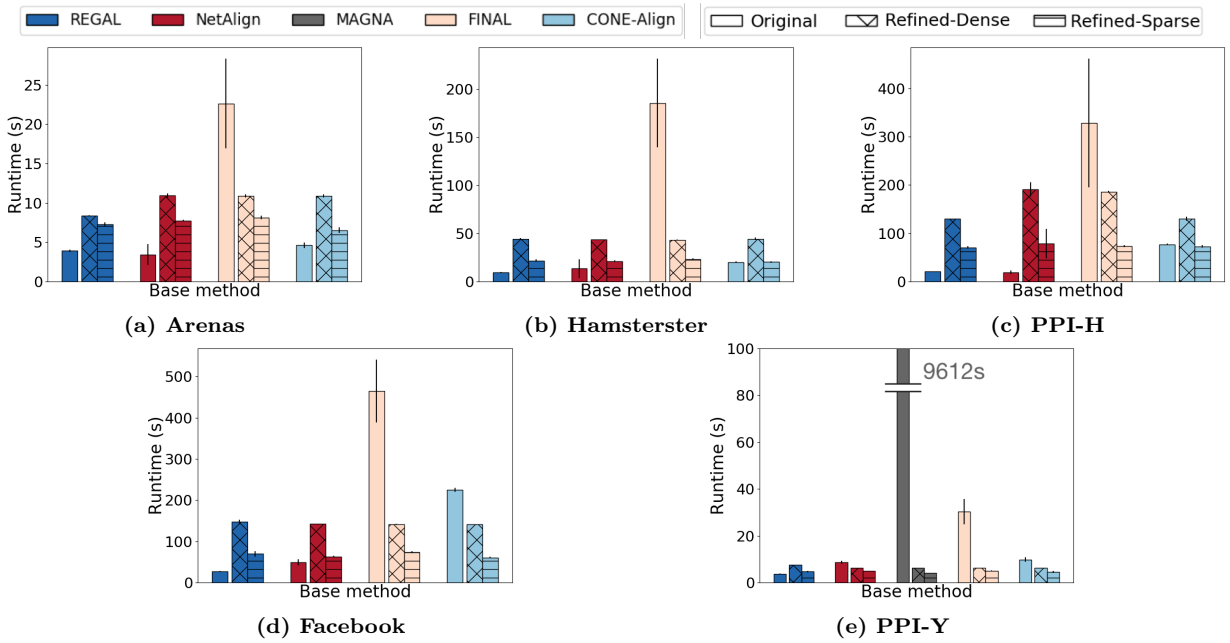


Figure 4.3: Runtime for different refinement variations. Sparse refinement is appreciably faster than dense refinement. Both refinements offer a modest computational overhead, often on par with or faster than the time taken to perform the original network alignment.

which they appreciably improve varies (in general, CONE-Align is the most robust, followed by REGAL, followed by NetAlign and then FINAL.)

The fact that at high noise levels, some methods but not others can be improved implies that RefiNA is indeed improving an initial alignment and not merely performing the alignment from scratch regardless of the initial solution. Precisely characterizing the initial solutions that best lend themselves to refinement is an interesting question for future work. With this said, our goal here is not to rank base methods according to their performance with or without RefiNA (indeed, REGAL and FINAL, which can use node and/or edge attributes, would likely do better if these were available.) Instead, we note that all base methods benefit dramatically from refinement compared to *their own* unrefined solutions.

Observation 4.3. *Different base network alignment methods benefit differently from refinement, although all benefit considerably.*

Compared to dense refinement, sparse refinement is slightly less accurate overall, but the gap between the two variants decreases as the noise level increases. In some cases, for high noise levels, sparse refinement even outperforms dense refinement (e.g., REGAL on Arenas and Hamsterster). One explanation is that the update of all pairwise node similarities may

Table 4.3: Maximum improvement thanks to RefiNA for each base method on each dataset, and maximum noise level at which RefiNA brings noticeable improvement. For all methods and all datasets, RefiNA brings dramatic increases in accuracy and can often bring significant improvement even at very high noise levels.

	REGAL	**NetAlign**	**FINAL**	**CONE-Align**	**MAGNA**
Arenas	82.18% *20% noise*	84.63% *10% noise*	86.93% *5% noise*	58.02% *25% noise*	N/A
PPI-H	80.53% *20% noise*	90.02% *10% noise*	79.57% *5% noise*	83.28% *25% noise*	N/A
Hamsterster	47.86% *20% noise*	55.85% *15% noise*	52.68% *10% noise*	39.50% *25% noise*	N/A
Facebook	61.22% *15% noise*	23.76% *15% noise*	17.03% *5% noise*	48.51% *25% noise*	N/A
PPI-Y	32.17% *20% noise*	34.67% *25% noise*	18.83% *25% noise*	45.72% *25% noise*	6.77% *25% noise*

add small amounts of noise when updating the similarities of clear misalignments. Thus the effect of sparse alignment, which only updates nodes with the highest update scores, could be akin to a kind of regularization, which is worth further study.

Figure 4.3 shows the runtime of sparse and dense refinement as well as the time taken to perform the initial network alignment with each base method. We see that sparse refinement is faster than dense refinement (these graph sizes are manageable for the various base network alignment methods and both refinement variants, but we show more pronounced scalability benefits of sparse refinement on larger graphs in Section 4.4.6). Both also offer a reasonable overhead compared to the time for initial alignment; they are modestly slower than NetAlign and REGAL, two methods that are well-known to be scalable; generally on par with CONE-Align; and generally faster than FINAL and much faster than MAGNA.[3]

Observation 4.4. *Sparse refinement is fast and effective.*

4.4.3 Alignment Performance: Real-World PPI Networks

We now consider the real-world PPI-Y dataset [SM14], where the structural differences between graphs are no longer random noise but instead reflect a real-world phenomenon.

Results. In Figure 4.2e, we once again see that all methods are more accurate with refinement (sparse or dense) compared to their original solutions. In this case, the network alignment method MAGNA, which was developed for this domain (biological network align-

[3]Actual average runtime for MAGNA on PPI-Y is 9612 seconds; we truncate to 100 to avoid distorting the scale of the plot. On account of this high runtime, this is the only dataset on which we run MAGNA.

ment), is more robust (although with refinement, CONE-Align is most accurate). With that said, MAGNA's refined solution is always slightly more accurate than its initial solution, so we see that in real-world problems, RefiNA's refinement benefits methods that were tailored for a specific problem domain.

4.4.4 Convergence: Accuracy and Consistency

One of the parameters of RefiNA is K, the number of iterations for which the initial matching is refined. In Figure 4.4, we plot the performance at each iteration, up to our maximum value of 100, for all methods and datasets (at lowest and highest noise levels, or number of added low-confidence PPIs in the case of PPI-Y). For brevity, we show accuracy and MNC for dense refinement only on the first three datasets (Arenas, Hamsterster, and PPI-H), and the per-iteration accuracy only for sparse and dense refinement on the remaining datasets.

Results. For both accuracy and matched neighborhood consistency, we see similar trends for the same colored curves. As for RefiNA variations, dense refinement of CONE-Align grows in accuracy slightly more steeply than sparse refinement. For MNC, we see that accuracy and MNC tend to have very similar values at 5% noise (thus the lines of the same color are close to overlapping). At 25% noise, MNC is lower than accuracy for highly accurate methods like CONE-Align—this is to be expected because Theorem 4.1 proved that the expected average MNC under even for a perfect alignment solution is bounded by the noise ratio. For the remaining methods which struggle to achieve high accuracy, MNC is higher than accuracy. As Theorem 4.2 showed, it is possible to find a high-MNC alignment that is still not perfectly accurate (due to structural indistinguishability). Indeed, here we see this happening in some especially challenging settings starting from less favorable initial solutions. Nevertheless, in most cases, improving MNC is a successful strategy for accurate network alignment.

We do find differences between different methods as well as different methods on the same dataset. For example, FINAL is generally the slowest method to converge, taking close to 70 iterations to fully converge on the Arenas dataset, with NetAlign taking around 20, REGAL around 10, and CONE-Align around 5. On the PPI-H dataset with 5% noise, FINAL sees slow progress for nearly the full 100 iterations before making rapid progress at the end. While

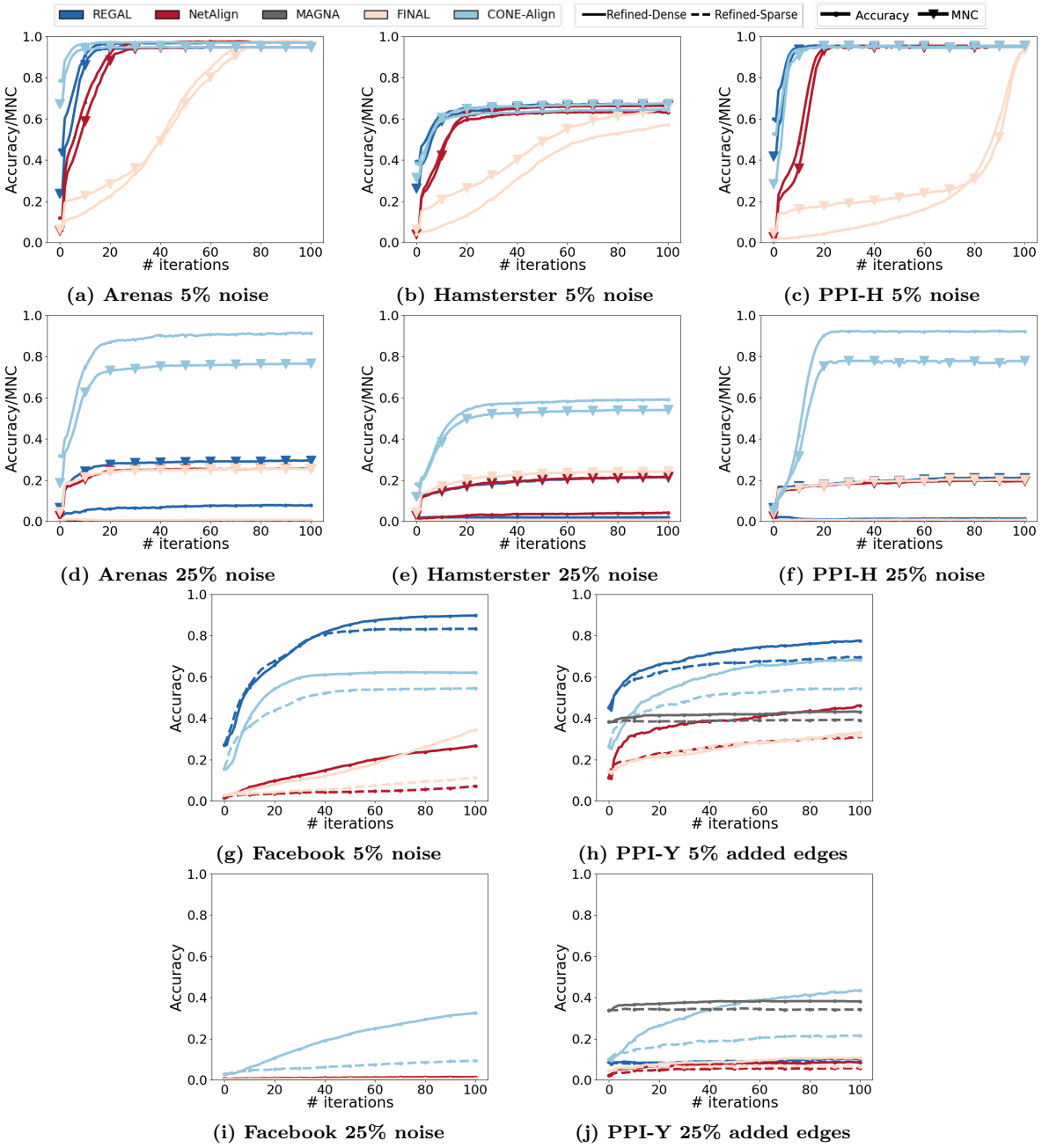
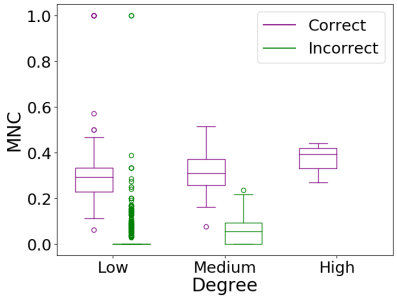
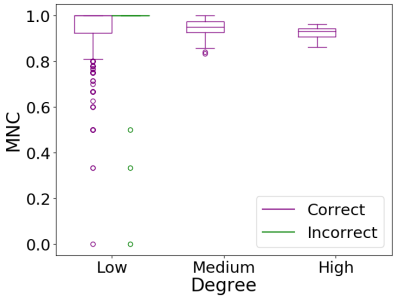


Figure 4.4: Analysis of RefiNA as a function of number of iterations (0 = performance of base method before refinement). Arenas, PPI-H, and Hamsterster datasets show accuracy and MNC, and Facebook and PPI-Y datasets show accuracy of sparse and dense RefiNA versions. Convergence rates are different for different methods, but often happen well before 100 iterations for both sparse and dense RefiNA. Accuracy and MNC consistently increase and follow similar trends, as per their theoretical connection.

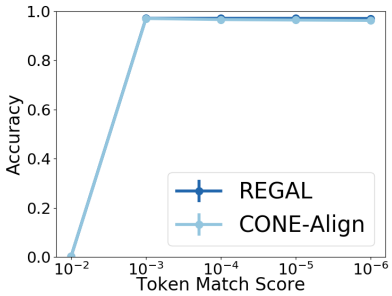
we allow all methods 100 iterations of refinement in our experiments, the elbow-shaped curves indicate that in practice, convergence often happens in far fewer than 100 iterations (with some exceptions for a few methods on the PPI-Y and Facebook datasets). In practice, early



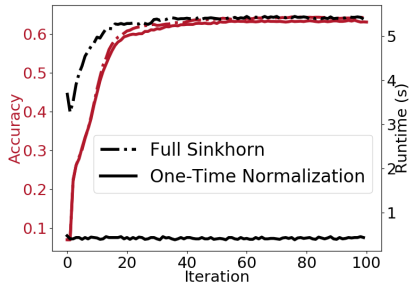
(a) Arenas 5% noise: Distribution of MNC and accuracy by node degree *before* refinement.



(b) Arenas 5% noise: Distribution of MNC and accuracy by node degree *after* refinement.



(c) Accuracy with varying token match score ϵ: Arenas, 5% noise. The methods almost overlap.



(d) Accuracy/runtime per iteration with limited vs full Sinkhorn normalization: NetAlign, Hamsterster 5% noise.

Figure 4.5: Drilldown of RefiNA in terms of our three insights that inspired its design. (a-b) For I1, we see that before and after alignment, high degree nodes are more likely to have high MNC and be correctly aligned. (c) For I2, our token match score admits a wide range of values that yield good refinement performance. (d) For I3, we see that our proposed normalization is just as effective as full Sinkhorn normalization while not incurring the additional computational expense.

convergence can be ascertained by small changes in the discovered alignments.

Observation 4.5. *Accuracy and MNC, sparse and dense refinement tend to trend similarly on each method on each dataset. Although convergence rates differ for different methods and datasets, convergence is quite fast in practice.*

4.4.5 Drilldown: Network Alignment Insights

In this section, we perform a drilldown of RefiNA that gives a deeper understanding into its specific design choices in light of the three insights we laid out in Section 4.3. Unless otherwise specified, we analyze the dense variant.

4.4.5.1 Aligning High Degree Nodes (I1)

One of the network alignment insights inspiring RefiNA's design (Section 4.3) is that high-degree nodes are easier to align. To test this, we analyze the accuracy and MNC on a node-level basis, grouping nodes by degree. In Figs. 4.5a-b, we display our analysis for NetAlign on the Arenas dataset for brevity. We split the nodes into three buckets by degree: with $\Delta_{\max}$ being the maximum node degree in the dataset, the buckets are $[0, \frac{\Delta_{\max}}{3}), [\frac{\Delta_{\max}}{3}, \frac{2\Delta_{\max}}{3}), [\frac{2\Delta_{\max}}{3}, \Delta_{\max}]$. Within each group, we visualize the distribution of MNC for each node, among all those correctly aligned and all those incorrectly aligned.

Results. High-degree nodes are rarely misaligned even before refinement, and the few nodes that are still misaligned after alignment are low-degree nodes. These often still have a high MNC, probably because it is easier for smaller node neighborhoods to be (nearly) isomorphic and thus the nodes to be (nearly) structurally indistinguishable. Recall that Theorem 4.2 showed that such nodes could remain a problem case for alignment accuracy even when MNC is high.

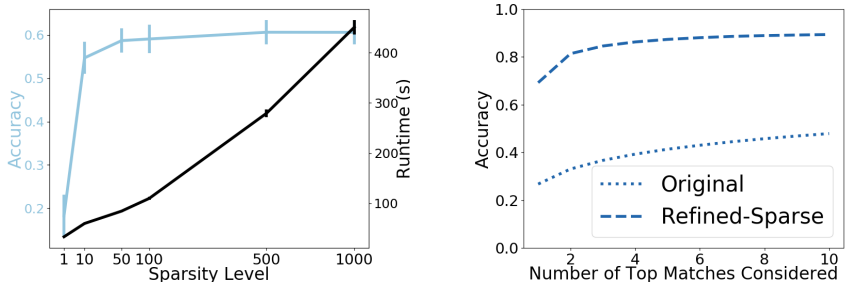
Observation 4.6. *It is easier to align high-degree nodes, verifying `I1` that motivates RefiNA's design choice that especially encourages alignment of high-degree nodes in early iterations.*

4.4.5.2 Token Match Score (I2)

Adding a token match score even to nodes that do not initially align is a simple way to overcome some of the limitations of an erroneous initial solution (our second network alignment insight in §4.3). We study the effects of this parameter ϵ on the Arenas dataset with 5% and 25% noise (we observe similar trends on other datasets), where we vary ϵ from 10^{-2} to 10^{-6} and average RefiNA's performance over five trials.

Results. In Figure 4.5c, we see that the performance can dramatically drop with too large ϵ, where the token match scores overwhelm the alignments that are being discovered. However, it is robust to smaller values of ϵ, and our criterion (Section 4.4.1), which would set $\epsilon = 10^{-4}$ for this dataset, works well.

Observation 4.7. *RefiNA is robust to the token match score, as long as it is not so large that it would drown out the actual node similarity scores.*



(a) Accuracy/total runtime of sparse refinement vs sparsity level α: CONE-Align, Facebook 5% noise.

(b) Top-α accuracy on a time budget: REGAL, LiveMocha 5% noise.

Figure 4.6: Sparse RefiNA. (a) Changing the sparsity of RefiNA's update step allows us to interpolate between an accuracy-runtime tradeoff. (b) Sparse RefiNA can run on large graphs and more than double the accuracy in the same amount of time taken for the initial network alignment. We also see that it can recognize additional meaningful similarities beyond the top 1 alignments.

4.4.5.3 Alignment Matrix Normalization (I3)

RefiNA normalizes the row and columns of the alignment matrix at each iteration, corresponding to our third network alignment insight in §4.3. Sinkhorn's algorithm iteratively repeats this process and converges to a doubly stochastic matrix [Sin64]. Some of the oldest iterative network alignment methods [GR96] need to do this at every iteration, which we consider here. We run RefiNA on NetAlign's initial alignment of the Hamsterster social network with 5% noise, both as proposed, and using the Sinkhorn's algorithm (converging when reaching a tolerance of 10^{-2} or terminating after 1000 iterations of normalization).

Results. We see in Figure 4.5d that our proposed normalization (essentially limited Sinkhorn normalization) yields virtually the same accuracy at every iteration. However, with our proposed normalization, the computation time remains low (well under a second per iteration) and approximately fixed. Meanwhile, Sinkhorn's algorithm takes longer to converge (particularly as the refinement continues) and dominates the running time. It is to RefiNA's advantage compared to previous work [GR96] that we can eschew the full Sinkhorn procedure.

Observation 4.8. *RefiNA can avoid the expensive matrix normalization required by optimization-based alignment approaches.*

4.4.6 Sparse Updates and Scalability

Sparsity Level of Refinement. RefiNA has two versions: a dense version where all nodes' alignment scores are updated, and a sparse version where for each node we only update a fixed number of possible alignments with the largest update scores (Section 4.3.2). By changing α, the number of updates to make per node, we can interpolate between the sparse and dense versions. We study this on Facebook with 25% noise.

Results. Updating just one alignment per node leads to poor performance, which corroborates `I2`: we should not blindly trust the initial solution so much to only consider its top choice. However, the initial solution does provide enough information that using it to make a small number of updates per node greatly improves the accuracy. Further quantities of updates add marginal accuracy returns compared to the extra runtime they require. Thus, sparse refinement offers a favorable balance of accuracy and speed.

"Soft" Alignments, Large Graphs. The favorable computational complexity of sparse RefiNA allows it to scale to large graphs, as we show by using RefiNA to improve the performance of top-α scores on a large dataset: LiveMocha, which has over 100K nodes and a *million* edges[4]. We consider our simulated alignment scenario with 5% noise. As this is a large dataset, we only use REGAL, the most scalable base alignment method we consider, together with sparse refinement (dense refinement runs out of memory). REGAL takes just over 2600 seconds. We consider a *budgeted computation* scenario for RefiNA, running it for the same amount of time (2600s) and evaluating the resulting solution.

We use this opportunity to explore the top-α accuracy, that is, the proportion of correct alignments that fall in the top α choices per node [HSSK18]. While RefiNA starts with a binary alignment matrix, it produces a real-valued cross-network node similarity matrix $\mathbf{M}$ as output. We treat these similarities as soft alignments and rank a node's top alignments by the similarities given by $\mathbf{M}$.

Results. In Figure 4.6b, we see that RefiNA is able to scale to large graphs and yield considerable improvements in accuracy. Indeed, it can more than double the accuracy of

[4]To the best of our knowledge, this is one of the largest graphs on which network alignment has been evaluated; [HSSK18, ZTT$^+$17] used million-node graphs only to benchmark runtime, and [ZT16] considers very large graphs but only performs the alignment on small subnetworks.

REGAL in the same amount of time it took REGAL to obtain its initial solution. Looking at the top-α scores, we see that RefiNA finds more approximate matches as we consider more top matches per node (up to 10, the number of matches we update per node.) This indicates that for applications like cross-platform recommendation, where top-α similarities may be useful [HSSK18], RefiNA produces usable "soft" alignments. Moreover, it indicates that even some of the alignments that RefiNA misses are still near-matches.

Observation 4.9. *Sparse refinement offers a favorable tradeoff of accuracy and runtime, and it allows RefiNA to achieve very good practical performance on extremely large graphs.*

4.5 Conclusion

We have proposed RefiNA, a powerful post hoc method for refining existing network alignment methods–including our methods that greedily match nodes based on embedding similarity (Chapter III)–that yields great improvements in accuracy and robustness. Its compact formulation encodes several insights for network alignment, supported by our extensive theoretical and empirical analysis, and can scale up to large graphs by sparsifying its iterative updates. Moreover, it has connections to other network alignment alignments and graph filtering in graph neural networks. We expect the simplicity of implementation combined with the considerable effectiveness for a wide variety of base network alignment methods to make RefiNA easy-to-adopt. Future work includes exploring the theoretical connection to graph neural networks and incorporating node and edge attributes.

CHAPTER V

Graph-Level Structural Similarity: Network Classification

Chapter based on work that won the Best Student Paper award at ICDM 2019 [HSK19].

5.1 Introduction

Thus far, this thesis has considered data mining problems at the node level: cross-network comparison for network alignment, or intra-network comparison for user stitching. Given that node embeddings are feature vectors that describe individual nodes, it makes sense to use them to compare individual nodes. However, graphs themselves are, after all, comprised of nodes. Thus, the collection of embeddings of the nodes in a graph should contain useful information about the graph itself. In particular, we should be able to compare entire graphs based on the node embeddings that they contain. We now show how to use node embeddings to perform graph-level collective network mining, focusing on the particular task of graph classification.

Graph classification has a wide range of applications from bioinformatics to computer vision to the social sciences. The problem can be solved with supervised machine learning given a suitable means of *graph comparison*. A strong and practical method for graph comparison must (**P1**) expressively and inductively compare graphs; (**P2**) efficiently handle many graphs with many non-aligned nodes; (**P3**) enable the downstream use of fast machine learning models for graph classification.

Figure 5.1: Overview of our framework. Given an input graph, node representations are learned via an appropriate embedding technique (Section 5.3.2). Our proposed feature mapping RGM then aggregates the graph's node embeddings in vector space (Section 5.4).

The first property (**P1**) implies that an ideal method should be flexible in characterizing graphs, and must handle unseen graphs. In terms of flexibility, many methods are constrained to compare topological graph properties via a small number of hand-engineered substructures. For example, most popular graph kernels are instances of the R-convolution framework, which decomposes a graph into substructures such as shortest paths, random walks, or graphlets, and compares graphs on the basis of these substructures [VSKB10]. Similarly, some works simply aggregate statistics (mean, standard deviation) of the distributions of hand-engineered node or edge features [BKERF13]. Moreover, not all methods readily generalize to out-of-sample nodes or graphs, often assuming that the graphs are defined on the same sets of vertices [KF17]. Likewise, optimal assignment kernels [KGW16] are computed by inducing a hierarchy over the training and test data and are thus necessarily transductive.

Furthermore, to perform graph classification effectively on large input graphs, it is necessary to compare graphs not only expressively but also efficiently (**P2**). This means that computing a graph feature representation or evaluating the kernel function between pairs of graphs must be scalable, ideally linear in the number of nodes across graphs. However, many existing feature representations are quadratic in the number of nodes [VZ17], and R-convolution graph kernels can be even slower. For instance, the random walk graph kernel can take $O(n^3)$ time in the number of nodes across graphs [VSKB10].

While the domain-specific challenge to graph classification relies mainly on defining a means of graph comparison, the efficiency of the final graph classifier is also important (**P3**). For instance, graph kernels rely on comparatively slow kernel methods, for which specialized solvers take quadratic time or more in the number of inputs [Joa06]. This limits their

applicability to problems with large *numbers* of graphs. Meanwhile, deep neural networks often take many epochs to train and require specialized hardware. Unsupervised graph feature representations remain a practical, more scalable choice [TMK+18].

In this work, we propose **Randomized Grid Mapping** or RGM, a feature map for graphs that enjoys all of the desiderata mentioned above. RGM **characterizes each graph by the distribution of its node embeddings at multiple levels of resolution in vector space**, where node embeddings may be obtained from any unsupervised approach that generalizes across graphs. We justify RGM with novel theoretical connections to existing implicit kernels. RGM is flexible and capable of handling node labels within the powerful Weisfeiler-Lehman label expansion framework [SSL+11], making it highly expressive (**P1**). Moreover, RGM approximates an implicit kernel in a fast, randomized fashion, leading to graph features that can be constructed in time linear in the number of nodes in that graph (**P2**). Finally, unlike exact kernel methods, RGM yields explicit features that can be used with linear SVMs [Joa06] for scalable classification with many graphs (**P3**).

The contributions of this work include:

- **Feature mappings**: We propose Randomized Grid Mapping (RGM) feature maps, which characterize graphs by the distribution of their node embeddings at multiple levels of resolution. We generalize RGM to the Weisfeiler-Lehman label refinement scheme [SSL+11].
- **Theoretical analysis**: We justify RGM by proving that the dot product of its histograms of node embeddings approximate the Laplacian kernel mean map computed on sets of node embeddings between pairs of graphs. We also prove that we can extend our feature maps to more powerful composite kernels.
- **Extensive experiments**: Our experiments demonstrate that RGM achieves strong classification performance, efficiency, and scalability compared to a wide variety of competitive baselines including graph kernels, unsupervised feature representations, and deep neural networks.

Code for RGM is available at `https://github.com/GemsLab/RGM`.

The rest of this paper is structured as follows. In Section 5.3 we give the preliminaries necessary to introduce RGM. In Section 5.4 we propose and theoretically analyze RGM. In

Section 5.5 we present an extensive range of experimental results. We discuss related work in Section 5.2, and offer concluding takeaways in Section 5.6.

5.2 Related Work

In this section we outline related literature in three directions; see [SERG14] for an overview of network similarity methods from a practitioner's perspective. Table 5.1 qualitatively compares RGM to selected baselines with respect to our three desiderata: (**P1**) expressive and inductive graph comparison; (**P2**) efficient comparison; (**P3**) downstream use of fast machine learning models for graph classification.

Graph kernels. Some graph kernels capture graph similarity from substructures, such as walks [VSKB10], shortest paths [BK05], subtrees [MV09], graphlets [SVP$^+$09], or other subgraphs [KP16]. Others leverage dependencies between these substructures [YV15], study propagation patterns [NGBK16], or characterize a restricted, strictly transductive class of valid optimal assignment kernels [KGW16]. Finally, recent work has considered the trade-offs between using explicit features and the implicit feature mappings of a kernel function [KNKM14], also for a restricted class of graph kernels.

Some works do consider node embeddings for graph classification: [JD15] considers optimal assignment of geometric embeddings, but produces indefinite similarity matrices. RetGK graph kernels [ZWX$^+$18] compute return probabilities of random walks in cubic time. The faster of the two proposed methods, RetGK$_{\text{II}}$, simply averages node feature maps and still applies the kernel trick at the end. More relevant to our work is [NMV17], which apply the PM kernel [GD07] to embeddings formed from the top eigenvectors of a graph's adjacency matrix. We achieve a similar design in a flexible *explicit feature map* that allows for faster training. Finally, RGM compares largely favorably to the concurrently proposed RGE random feature map [WYZ$^+$19] that approximates an EMD-like transportation distance between eigenvector embeddings. RGE samples node embeddings without discerning how they are distributed in vector space, the very information that RGM captures.

Techniques inspired by the Weisfeiler-Lehman test of isomorphism [SSL$^+$11] can improve the performance of methods that use node labels, including ours. Further extensions cap-

ture global and local structure, although they require approximation to be computationally practical [MKM17]. In general, computing graph kernel functions and using them in kernel machines falls short on computational properties laid out in **P2** and **P3**.

Other graph similarity functions (besides kernels) include graph edit distance [BG18], whose computational impracticality for all but small graphs violates **P2**. The scalable graph similarity function DeltaCon [KVF13] is designed for graphs defined over the same set of vertices, limiting its expressivity (**P1**).

Unsupervised feature mappings. An early graph feature map, NetSimile, consists of basic summary statistics from distributions of hand-engineered node and edge features. Such features may be useful for aligning graphs [HLP+18] or exploratory graph analysis with domain knowledge [JK17] but are limited in expressivity. More recently, FGSD [VZ17], uses histograms to characterize a graph based on its biharmonic kernel. However, its practical limitations include quadratic time complexity and inability to use node labels. NetLSD [TMK+18] was shown to be more powerful and scalable, but it too cannot use node label information. These all fall short on **P1** at minimum.

Like our method RGM, all of the above works are unsupervised, which makes training simpler and generally faster. Representations for graphs or subgraphs [AZRP18, IB18] may also be learned by analogy to paragraph or document representation learning in NLP [LM14]. However, these methods require excessive amounts of graph sampling (a computational challenge for **P2**) to achieve competitive results and/or have high variance.

Deep neural networks. Deep neural networks have grown in popularity and have been extended to graph classification tasks. Diffusion-convolution neural networks [AT16] adapt graphs for use with existing convolutional architectures by scanning a diffusion process across each node, which had empirical limits for graph classification. PATCHY-SAN [NAK16] extracts fixed-sizes patches from graphs and then uses graph canonization tools to define a vertex ordering for use with CNNs, which the recent work DGCNN [ZCNC18] does in an end-to-end fashion.

It is also possible to adapt node classification architectures with specialized graph convolutions, such as GraphSAGE [HYL17] and GCN [KW17a], by aggregating the node features. We showed that given the same set of node embeddings, RGM aggregation is often more

Table 5.1: Qualitative comparison of various methods. Existing graph kernels and unsupervised feature representations lack one or more desirable properties that RGM has.

	Expressive	Inductive	Fast Comparison	Fast ML
NetLSD	✗	✓	✓	✓
WLOA	✓	✗	✓	✗
RETGK	✓	✓	✗	✗
WLPM	✓	✓	✓	✗
RGM	✓	✓	✓	✓

effective than the mean- and max-pooling operations that are often used in neural network architectures. The recent hierarchical method DiffPool [YYM+18] performs supervised node pooling at greater computational expense.

It is challenging to make precise statements regarding **P1**, **P2**, and **P3** for deep learning-based methods, as all three depend on how well the training converges. In general, however, neural networks are heavily parametrized and thus more difficult to train, requiring additional computational resources such as GPUs (a practical efficiency issue regarding **P2** and **P3**) and risking overfitting especially on smaller datasets (a concern for expressivity, i.e. **P1**) [ZCNC18]. Only recently have neural network models been designed for limited, noisy data in specific domains such as neuroscience [YZD+19].

5.3 Preliminaries

Table 5.2: Major symbols and definitions.

Symbols	**Definitions**
$\mathbf{Y}_i$	Node embedding matrix for graph G_i in $\mathbb{R}^{n_i \times p}$
$\mathbf{Y}_{i,j}$	Row-vector embedding in $\mathbb{R}^p$ of node j in graph G_i
$\boldsymbol{\delta}, \boldsymbol{\mu}$	Vectors in $\mathbb{R}^p$ of grid cell widths and offsets, resp.
$\mathcal{G}^{[\boldsymbol{\delta},\boldsymbol{\mu}]}$	Random grid parametrized by cell width $\boldsymbol{\delta}$ and offset $\boldsymbol{\mu}$
$\mathbf{h}_i$	Histogram induced by grid $\mathcal{G}$ on G_i's embeddings $\mathbf{Y}_i$

We begin by outlining necessary background on embedding and kernel techniques for graph comparison. For reference, Table 5.2 gives our main symbols.

5.3.1 Problem Definition and Terminology

In graph classification, we are given a collection of training and test graphs of different sizes, with or without node labels. Each graph has a class that must be predicted. The i-th graph (in either the training or test set) is denoted $G_i = (V_i, E_i)$, where V_i and E_i are respectively the nodes and edges of graph G_i. We denote the number of nodes in G_i as $n_i \equiv |V_i|$.

Using node embedding, the graph G_i may be represented as a matrix $\mathbf{Y}_i \in \mathbb{R}^{n_i \times p}$ of p-dimensional vector embeddings. The vector embedding of node j in graph G_i is denoted $\mathbf{Y}_{i,j} \in \mathbb{R}^p$. Without loss of generality, we assume embeddings are normalized to be in $[0, 1]^p$. Our goal is to train a machine learning hypothesis that can successfully predict the classes of the test graphs, given these embeddings.

5.3.2 Node Embedding Techniques

Our proposed feature mapping RGM characterizes the distribution of a graph's node embeddings in latent feature space. While RGM can utilize any existing method for node embedding that inductively generalizes to *multi*-network settings, here we focus on three in particular. The first two have been previously used for graph classification [NMV17, ZWX+18], and the third extends a structural node feature descriptor with subquadratic time complexity previously used for graph alignment [HSSK18]:

Eigenvector Embeddings (EIG). Many graph similarity functions take as embeddings the eigenvectors of the adjacency matrix or the graph Laplacian, with node i represented by the absolute values of the i-th components of the top p eigenvectors [JD15, NMV17]. Eigenvectors capture *global* properties of the graph [NMV17], which may generalize across graphs.

Return Probability Features (RPF). This method describes each node by a vector whose i-th entry represents the probability that an i-step random walk starting at that node returns to itself. This was shown to be an effective structural node feature descriptor [ZWX+18], albeit requiring a full eigendecomposition of the adjacency matrix to compute exactly.

*i*NetMF. We extend the xNetMF [HSSK18] embedding technique, which was originally used

for multi-network alignment. At a high level, xNetMF constructs a histogram per node that captures the degree distribution in that node's (weighted) k-hop neighborhood. xNetMF efficiently computes embeddings using these histograms by comparing each node to a small sample of landmark nodes randomly chosen from all training graphs, then constructing a low-rank implictit factorization of a structural node similarity matrix leveraging the Nyström method [DM05]. We make xNetMF inductive (i.e., **iNetMF**) by reusing the same "landmark nodes" from the training set to embed the test graphs, in effect embedding the test graphs into the same subspace as the training graphs. This can be viewed as a simplified version of latent network summarization [JRK+19] using the fast Nyström decomposition.

Our approach learns node embeddings and derives a graph representation in separate steps. While recent deep learning methods optimize both steps in an end-to-end fashion [XHLJ19], an important advantage of our multi-step framework is that it can readily handle special kinds of graphs using embedding methods tailored for that format of graph: for instance, signed networks [JDE+20], heterogeneous networks [DCS17], dynamic networks [TBL+20], higher-order networks modeling non-Markovian dependencies [BKTK19], and others.

5.3.3 Kernels on Sets of Features

We briefly overview existing kernel methods that operate on sets of features, in this case the node embeddings of pairs of graphs. Such kernels accept node embedding matrices $\mathbf{Y}_1 \in \mathbb{R}^{n_1 \times p}$ and $\mathbf{Y}_2 \in \mathbb{R}^{n_2 \times p}$ for graphs G_1 and G_2, respectively, as input. Embeddings may be compared in two different ways:

Distance-based. One way to compare G_1 and G_2 is to compute the (continuous) distances between pairs of their node embedding vectors. While many such comparison methods exist, such as the Earth Mover's Distance (used in [NMV17]), here we focus on the **Laplacian kernel mean map** [SGSS07], which, for embedding matrices $\mathbf{Y}_1$ and $\mathbf{Y}_2$ and hyperparameter γ controlling the kernel's resolution, is

$$k_{\text{LKM}}(\mathbf{Y}_1, \mathbf{Y}_2; \gamma) = \frac{1}{n_1 n_2} \sum_{i=1}^{n_1} \sum_{j=1}^{n_2} \exp\left(-\gamma ||\mathbf{Y}_{1,i} - \mathbf{Y}_{2,j}||_1\right). \tag{5.1}$$

Equation (5.1) corresponds to the average Laplacian kernel similarity between all pairs of embeddings in graphs G_1 and G_2 and has $O(n_1 n_2 d)$ complexity to compute for p-dimensional node embeddings.

Grid-based. An alternative (discretized) approach is to compute the embeddings' spatial overlap on a grid or histogram. Discrete binning-based approaches have been used to estimate information-theoretic (dis)similarity between datasets [NH18]. Here we focus on the pyramid match or **PM kernel** [GD07], which fits a set of increasingly finer resolution grids to the p-dimensional unit hypercube. As used in graph classification [NMV17], the grid at each level $\ell \in 0, \ldots, L$ has 2^ℓ cells of equal width without offset from the origin. At each level ℓ, these grids induce histograms $\mathbf{h}_1^{(\ell)}$ and $\mathbf{h}_2^{(\ell)}$ capturing the number of node embeddings from $\mathbf{Y}_1$ and $\mathbf{Y}_2$ that map into each grid cell. The intersection of pairs of histograms at level ℓ across cells c is given as $I(\mathbf{h}_1^{(\ell)}, \mathbf{h}_2^{(\ell)}) = \sum_c \min\{h_{1,c}^{(\ell)},\ h_{2,c}^{(\ell)}\}$. The PM kernel, which takes $O((n_1+n_2)pL)$ time to compute, is a weighted sum of new intersections found at each increasingly coarse grid:

$$k_{\text{PM}}(\mathbf{Y}_1, \mathbf{Y}_2; L) = \sum_{\ell=0}^{L-1} \frac{1}{2^{L-\ell}} \left[I\big(\mathbf{h}_1^{(\ell)}, \mathbf{h}_2^{(\ell)}\big) - I\big(\mathbf{h}_1^{(\ell+1)}, \mathbf{h}_2^{(\ell+1)}\big) \right]. \tag{5.2}$$

5.4 RGM: Randomized Grid Mapping

With the necessary background given, we now discuss how we aggregate a collection of node embeddings into a unified *explicit* feature map for a graph. We first propose our histogram-based mapping RGM and prove its connection to the Laplacian kernel mean map. We then generalize RGM to a multiresolution feature map, and further extend it to incorporate node labels within the Weisfeiler-Lehman framework.

5.4.1 Randomized Features of Graphs

In this section we propose our feature mapping RGM, and theoretically justify it by connecting it to the existing kernel techniques discussed in Section 5.3.3.

Histograms of Node Embeddings. RGM builds on the intuition of grid-based binning (Section

5.3.3) for a fast-to-compute *feature mapping* that can be used with linear SVMs for efficient graph classification. Let $\mathcal{G}^{[\boldsymbol{\delta},\boldsymbol{\mu}]}$ be a *randomized grid* specified by p-dimensional random vectors $\boldsymbol{\delta}$ and $\boldsymbol{\mu}$, which respectively specify the cell width and offset of the grid along each dimension. Given graph G_i with node embedding matrix $\mathbf{Y}_i$, with a hash function $\phi(\cdot)$ mapping each node's embedding to a cell in $\mathcal{G}^{[\boldsymbol{\delta},\boldsymbol{\mu}]}$ we induce a histogram $\mathbf{h}_i$. The value of the j-th element of $\mathbf{h}_i$ is

$$h_{i,j} = \sum_{p=1}^{n_i} \mathbb{1}\Big\{\phi\big(\lceil(\mathbf{Y}_{i,p} - \boldsymbol{\mu})/\boldsymbol{\delta}\rceil\big) = j\Big\}. \tag{5.3}$$

We can use $\mathbf{h}_i$ as a *feature vector* for graph G_i. Intuitively, each cell in the histogram represents a region of p-dimensional embedding space, so these features count the number of embeddings that fall into each region of the space. In other words, we describe G_i in terms of the *distribution of its node embeddings* in vector space.

Probabilistic Interpretation. Our randomized grid construction, given a suitable choice of parameters $\boldsymbol{\delta}$ and $\boldsymbol{\mu}$, gives RGM a probabilistic kernel interpretation. Specifically, the dot product of two graphs' RGM histograms approximates the Laplacian kernel mean map between the graphs. We first state a foundational result about random features for general kernel methods:

Lemma 5.1 (Adapted from [RR08])**.** *For vectors $\mathbf{x}_1, \mathbf{x}_2 \in \mathbb{R}^p$, the probability that $\mathbf{x}_1$ and $\mathbf{x}_2$ map to the same cell in random grid $\mathcal{G}^{[\boldsymbol{\delta},\boldsymbol{\mu}]}$ with cell widths $\boldsymbol{\delta}_i$ drawn from a Gamma distribution with shape 2 and scale $\frac{1}{\gamma}$, and offsets $\boldsymbol{\mu}_i \sim$ Uniform$(0, \delta_i)$ sampled independently along each dimension i is equal to the Laplacian kernel $\exp(-\gamma||\mathbf{x}_1 - \mathbf{x}_2||_1)$.*

Using this lemma, we connect the embedding histograms of Equation (5.3) and the Laplacian kernel mean map:

Theorem 5.1. *Let $\mathbf{h}_1$ and $\mathbf{h}_2$ be the normalized histograms induced via* RGM *on graph node embedding matrices $\mathbf{Y}_1$ and $\mathbf{Y}_2$ respectively, by a grid $\mathcal{G}^{[\boldsymbol{\delta},\boldsymbol{\mu}]}$ with random cell widths $\boldsymbol{\delta}_i$ drawn from a gamma distribution with shape 2 and scale $\frac{1}{\gamma}$, and offsets $\boldsymbol{\mu}_i \sim$ Uniform$(0, \delta_i)$ sampled*

independently along each dimension i. Then

$$\mathbb{E}\Big[\langle \mathbf{h}_1, \mathbf{h}_2 \rangle\Big] = \frac{1}{n_1 n_2} \sum_{i=1}^{n_1} \sum_{j=1}^{n_2} \exp(-\gamma ||\mathbf{Y}_{1,i} - \mathbf{Y}_{2,j}||_1),$$

where the right-hand side is equivalent to the Laplacian kernel mean map (5.1) *between embedding matrices* $\mathbf{Y}_1$ *and* $\mathbf{Y}_2$.

Proof. For each node $i \in V_1$, let $\mathbf{f}_i$ be a binary indicator vector with $f_{ic} = \mathbb{1}\{\mathbf{Y}_{1,i} \in \mathcal{G}^{[\boldsymbol{\delta},\boldsymbol{\mu}]}[c]\}$; i.e., 1 if i's embedding falls into grid cell c. We define the indicator vectors for V_2 similarly. Then we have that

$$\langle \mathbf{h}_1, \mathbf{h}_2 \rangle = \frac{1}{n_1 n_2} \sum_{c \in \mathcal{G}} \sum_{i=1}^{n_1} \sum_{j=1}^{n_2} f_{ic}\, f_{jc} = \frac{1}{n_1 n_2} \sum_{i=1}^{n_1} \sum_{j=1}^{n_2} \mathbf{f}_i^\top \mathbf{f}_j,$$

since the product of the numbers of nodes in G_1 and G_2 that fall into the same cell c is the number of the corresponding cross-graph node pairs. This can be determined by multiplying the corresponding indicator vectors for all these pairs, since the product will be 0 when the nodes do not fall into the same cell. Recall that the vectors $\mathbf{f}_i$ and $\mathbf{f}_j$ depend on the parameter γ governing the distribution from which the components of $\boldsymbol{\delta}$ and $\boldsymbol{\mu}$ are sampled. Their dot product is 1 iff the embeddings of node i in G_1 and node j in G_2 map to the same cell in $\mathcal{G}$. Thus, $\mathbb{E}[\mathbf{f}_i^\top \mathbf{f}_j; \gamma] = \exp(-\gamma ||\mathbf{Y}_{1,i} - \mathbf{Y}_{2,j}||_1)$, and the theorem follows from Lemma 5.1. □

This result offers a theoretical connection between our feature maps based on node embedding distributions and graph kernels, namely the Laplacian kernel mean map. Indeed, we see a new connection between distance-based and grid-based embedding comparison techniques (Section 5.3.3): with appropriate grid construction, the latter can be used to approximate the former, in linear time in the number of nodes in each graph. An important advantage that our explicit feature maps have over both kernels is that faster linear machine learning algorithms may be used, which scale better to large numbers of graphs.

5.4.2 Multiresolution Feature Maps

Representing each graph using embedding histograms from Equation (5.3) with grid construction as in Theorem 5.1 allows us to construct a feature map that approximates the

Laplacian kernel mean map for a particular resolution given by a fixed γ. A large value of γ drives the kernel similarity function closer to zero, meaning only extremely similar nodes will contribute meaningfully to the kernel mean map. Meanwhile, a small value drives the kernel similarity function close to one, in which case even rather dissimilar nodes may still measure a relatively high similarity.

Composite Kernels and Composite Feature Maps. Any single kernel or parametrization has strengths and drawbacks, and a feature map that approximates that single kernel shares that kernel's limitations. A powerful and arguably more flexible technique, then, is to create *composite* kernels from linear combinations of single kernels. Defining α_i as the contribution of the i-th kernel k_i, composite kernels have the form

$$K(\mathbf{Y}_1, \mathbf{Y}_2) = \sum_{i=1}^{M} \alpha_i k_i(\mathbf{Y}_1, \mathbf{Y}_2). \tag{5.4}$$

Similarly, we can create composite feature maps with similarly greater expressive power. Specifically, we show that if the individual kernels comprising a composite kernel are approximable by random features, we have a (random) feature map for the corresponding composite kernel:

Lemma 5.2. *Given kernels $k_1(\mathbf{Y}_1, \mathbf{Y}_2), \ldots, k_M(\mathbf{Y}_1, \mathbf{Y}_2)$ with approximate feature maps, the composite kernel $K = \sum_{i=1}^{M} \alpha_i k_i(\mathbf{Y}_1, \mathbf{Y}_2)$ (i.e., Equation (5.4)) has a corresponding approximate feature map.*

Proof. Let $\psi_i(\cdot)$ be a function that, for embeddings $\mathbf{Y}_i$, constructs features approximating the individual kernel k_i: $k_i(\mathbf{Y}_1, \mathbf{Y}_2) \approx \psi_i(\mathbf{Y}_1)^\top \psi_i(\mathbf{Y}_2)$. We define the feature map for embedding $\mathbf{Y}_i$ as $\mathbf{h}_i = [\sqrt{\alpha_1}\psi_1(\mathbf{Y}_i) \mid\mid \ \ldots \ \mid\mid \sqrt{\alpha_M}\psi_M(\mathbf{Y}_i)]$, where $\mid\mid$ denotes vector concatenation. Then

$$\begin{aligned} \mathbf{h}_1^\top \mathbf{h}_2 &= \sum_{i=1}^{M} \alpha_i \psi_i(\mathbf{Y}_1)^\top \psi_i(\mathbf{Y}_2) \\ &\approx \sum_{i=1}^{M} \alpha_i k_i(\mathbf{Y}_1, \mathbf{Y}_2) = K(\mathbf{Y}_1, \mathbf{Y}_2). \end{aligned}$$

□

Multiresolution RGM *Features.* With Lemma 5.2, we now have the tools to develop our

RGM features based on histograms of node embeddings that overcome the limitations of any fixed resolution by combining *multiple* levels of resolution. That is, by *concatenating* node embedding histograms across L levels of resolution, we achieve the effect of a composite Laplacian kernel mean map with different values of γ.

At each level of resolution $\ell \in [0, 1, \ldots, L]$, we construct component histograms $\mathbf{h}_i^{(\ell)}$ from $\mathbf{Y}_i$ using Equation (5.3), with cell widths drawn from a gamma distribution with shape 2 and scale $\frac{1}{2^{\ell+1}}$ along with uniform offsets (recall that the scale corresponds to the inverse of γ in the Laplacian kernel mean map, by Theorem 5.1). The expected cell width along each dimension for $\mathbf{h}_i^{(\ell)}$ is $\frac{1}{2^\ell}$. The earlier histograms will thus have coarse cells that capture many matches, while later histograms will have fine cells that only bin together embeddings very close in vector space, as demonstrated in Figure 5.2.

As in [NMV17], we use a weighing scheme to prioritize matches found at more discriminative finer resolutions: a histogram with expected cell width $\frac{1}{2^\ell}$ has weighing factor $\sqrt{1/2^{L-\ell}}$. The dot product of two component histograms with this weighing factor will then be weighed by $1/2^{L-\ell}$. Putting it all together, for a graph G_i with node embeddings $\mathbf{Y}_i$, our RGM feature map for a set of node embeddings is

$$\mathbf{h}_i = [\sqrt{1/2^L}\mathbf{h}_i^{(0)} \,||\, \sqrt{1/2^{L-1}}\mathbf{h}_i^{(1)} \,||\, \ldots \,||\, \mathbf{h}_i^{(\ell)}] \tag{5.5}$$

This multiresolution design recalls the design of the pyramid match kernel (Section 5.3.3) while retaining the theoretical connections to the Laplacian kernel mean map discussed in the previous section. However, it should be noted that the multiresolution RGM is *not* approximating the PM kernel, but rather has a similar design that compares graphs at multiple levels of resolution in vector space. Two key differences between multiresolution RGM and PM are: (1) PM compares embeddings via histogram intersection versus RGM's dot product, and (2) PM excludes nodes matched at finer levels of granularity before comparing coarser levels of granularity. Concerning the former, RGM's dot product permits the use of faster (linear) machine learning algorithms. Concerning the latter, by including matches in all levels, RGM places further weight on matches found in fine levels of granularity, which are likely to be matched at coarser levels of granularity as well, amplifying the effect that PM attempts to achieve.

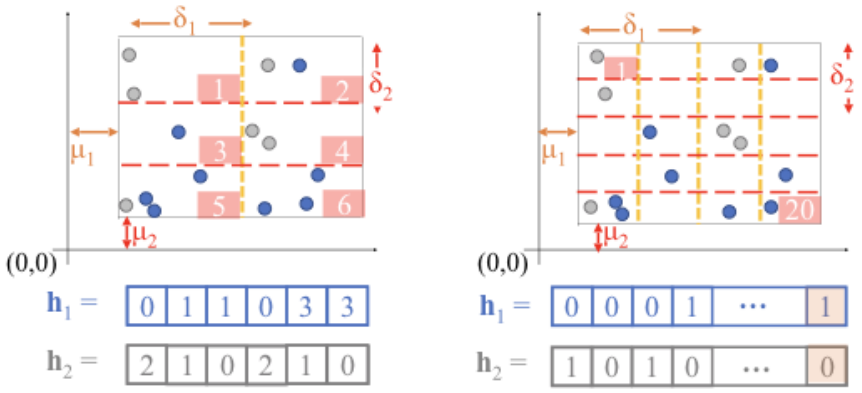


Figure 5.2: Multiresolution feature maps for graphs. We create histograms by binning a graph's node embeddings using grids with randomly chosen cell widths and offsets along each dimension. We use multiple grids parametrized differently in expectation to produce histograms of coarser (left) and finer (right) levels of resolution. The final graph features are a weighted concatenation of these histograms.

Complexity Analysis. Each level of RGM hashes n_i nodes per graph G_i, and each graph is represented by features of p dimensions. Therefore, a single level of RGM is $O(n_i p)$. With L total levels of resolution, RGM's complexity is $O(n_i p L)$.

5.4.3 Handling Node Labels

Node labels may provide an additional source of information beyond the graph topology alone. Without loss of generality, it suffices to consider discrete node labels, as continuous attributes may be hashed into discrete labels [MKKM16]. We review techniques that transform unlabeled graph kernels into labeled ones and make simple labeled kernels more powerful. For each technique, we show that using Lemma 5.2, RGM feature maps can have equivalent capabilities.

Composite Labeled Kernels. Given a kernel between sets of embeddings $k(\mathbf{Y}_1, \mathbf{Y}_2)$, a corresponding composite *labeled* kernel is

$$K_B(\mathbf{Y}_1, \mathbf{Y}_2) = \sum_{b \in B} k(\mathbf{Y}_1^{\{b\}}, \mathbf{Y}_2^{\{b\}}), \tag{5.6}$$

where B is the set of unique node labels, $\mathbf{Y}_i^{\{b\}}$ consists of the embeddings in G_i of nodes with label b, and k is the (unlabeled) base kernel, such as the pyramid match kernel [NMV17], or our multiresolution weighted sum of Laplacian kernel mean maps approximated by RGM.

Intuitively, the idea is to use the base kernel to only compare nodes with the same label.

We follow this intuition to design labeled features by forming multiresolution histograms using Equation (5.5) for embeddings of nodes with each label and concatenating them. From Lemma 5.2, it follows that this labeled version of RGM corresponds to the labeled kernel built on the (multiresolution) Laplacian kernel mean map using Equation (5.6).

Corollary 5.1. *Given graph G_i with embeddings $\mathbf{Y}_i$, the feature map*

$$\mathbf{h}_i = [\mathbf{h}_i^{\{b_1\}} \,||\, \ldots \,||\, \mathbf{h}_i^{\{b_{|B|}\}}] \tag{5.7}$$

approximates the labeled Laplacian kernel mean map.

Each $\mathbf{h}_i^{\{b\}}$ refers to an RGM feature map constructed using Equation (5.5) for nodes with label b only, with embeddings $\mathbf{Y}_i^{\{b\}}$. As each node is still mapped to only one cell in the corresponding grid, the worst-case complexity of RGM is unchanged. Thus, we can maintain linear-time feature construction and training *even with node labels* using RGM.

RGM *with Weisfeiler-Lehman Framework.* We can further generalize the labeled feature maps from Equation (5.7) to the Weisfeiler-Lehman (WL) framework [SSL+11], a state-of-the-art graph kernel framework that over H iterations assigns each node a new label by hashing its neighbors' labels in the previous iterations. Given a labeled graph kernel $K_B(\mathbf{Y}_1, \mathbf{Y}_2)$ as in Equation (5.6), the corresponding WL kernel is

$$K_{\mathrm{WL}}(\mathbf{Y}_1, \mathbf{Y}_2; H) = \sum_{h=0}^{H} K_{B_h}(\mathbf{Y}_1, \mathbf{Y}_2), \tag{5.8}$$

where for H iterations, B_h is the Weisfeiler-Lehman labeling at iteration h, and B_0 is the set of original node labels. In the above, we sum individual kernels that use the WL labelings at each iteration. Thus, applying Lemma 5.2 and Corollary 5.1, we design a version of RGM corresponding to the labeled Laplacian kernel mean map enhanced with the WL framework:

Corollary 5.2. *For a graph G_i, the feature map $\mathbf{h}_i = [\mathbf{h}_i^{\{B_0\}} \,||\, \ldots \,||\, \mathbf{h}_i^{\{B_H\}}]$ is an approximate feature map for the H-iteration Weisfeiler-Lehman Laplacian kernel mean map, where B_h is the WL labeled at iteration h and B_0 is the original set of node labels.*

Table 5.3: Real data [KKM+16] used in our experiments. We give the total number of nodes/edges across all graphs per dataset.

Name	Nodes	Edges	Graphs	Classes	Node labels	Domain
MUTAG	3 371	3 721	188	2	Y	bioinf
PTC-MR	4 916	5 053	344	2	Y	bioinf
NCI1	122 765	132 753	4 110	2	Y	bioinf
IMDB (binary)	19773	96531	1000	2	N	collab
IMDB (multi)	19 502	98 910	1 500	3	N	collab
COLLAB	372 474	12 286 733	5000	3	N	collab

Here, the component histograms $\mathbf{h}_i^{\{B_h\}}$ that we concatenate for each relabeling are constructed using Equation (5.7).

Complexity Analysis. WL RGM takes $O(n_i pLH)$ time for H label expansions. Therefore, by designing linear feature maps to approximate WL graph kernels using node embeddings, we can use the well-documented strengths of WL label expansion [SSL+11] to achieve good performance *faster than exact kernel methods.*

5.5 Experiments

We now study RGM across a range of extensive experiments. We focus on the following research questions:

Q1 How accurately can we classify graphs with RGM feature maps?

Q2 How efficient and scalable is RGM relative to related kernel methods with respect to the number and/or size of the input graphs?

Q3 Can other node embedding or aggregation choices be used in RGM, particularly in an inductive setting?

5.5.1 Experimental Setup

Data. We evaluate our methods on six benchmark graph classification datasets from different domains commonly studied in graph classification–bioinformatics and social collaboration–all publicly available with detailed descriptions at [KKM+16]. Table 5.3 presents aggregate information about each dataset.

Embedding Methods. As discussed in Section 5.3.2, we use three embedding methods with RGM feature maps:

1. **EIG**: Following [NMV17], we take the top 6 eigenvectors of the adjacency matrix to form the embeddings (if a graph has size $n < 6$, we repeat the last features $6 - n$ times).

2. **RPF**: To compute the return probability features, we use the recommended $p = 50$ [ZWX+18].

3. **iNetMF**: We set the maximum hop distance $K = 2$ and discount factor $\delta = 0.1$, following [HSSK18], with embedding dimensionality $p = 100$, as per the literature.

For brevity, in our results we report RGM's performance with the most accurate embedding method for each dataset among EIG, RPF, and iNetMF. In general, they perform comparably across datasets.

Baselines. We compare RGM against several popular baselines from the **graph kernel literature**:

1. **SP** [BK05], or the shortest paths kernel;

2. **GR** [SVP+09], or the graphlets kernel. We follow the literature and using graphlets of size 3 [NMV17];

3. **WL-ST** [SSL+11], or the Weisfeiler-Lehman subtree kernel;

4. **WL-OA** [KGW16], or the Weisfeiler-Lehman optimal assignment kernel;

5. **LWL-3** [MKM17] kernel;

6. **WL-PM** [NMV17], which computes the Weisfeiler-Lehman pyramid match kernel on eigenvector embeddings;

7. **RetGK** [ZWX+18], a graph kernel based on the return probabilities of random walks as captured by RPF. We use RetGK_{II}, which uses approximate random features techniques to avoid a quadratic-time comparison of graphs using RPF and thus conceptually resembles our approach.

From the **unsupervised feature mapping** literature, we compare to:

a) **NetLSD** [TMK+18], which achieved superior performance and scalability over unsupervised feature representations such as NetSimile [BKERF13] and FGSD [VZ17]. We use both the heat and the wave kernel to obtain graph representations, and report the best results for each dataset.

Finally, following existing practice [ZWX+18], we compile reported numbers for **deep neural networks** for further comparison:

i) **DCNN** [AT16], or diffusion-convolutional neural networks;

ii) **PSCN** [NAK16], or the PATCHY-SAN neural network;

iii) **DCGNN** [ZCNC18], a neural architecture that performs end-to-end graph classification.

Note that NetLSD, SP, and GR do not use node labels, while the other baselines do. For datasets without node labels, we give all nodes the same label to start [KGW16]. We fix the number of WL iterations for RGM and WL baselines to $H = 2$ [KNKM14] and the number of levels in PM and RGM to 4 [NMV17]. Other parameters specific to particular baseline methods are set to values recommended by their authors in the papers and/or official implementations.

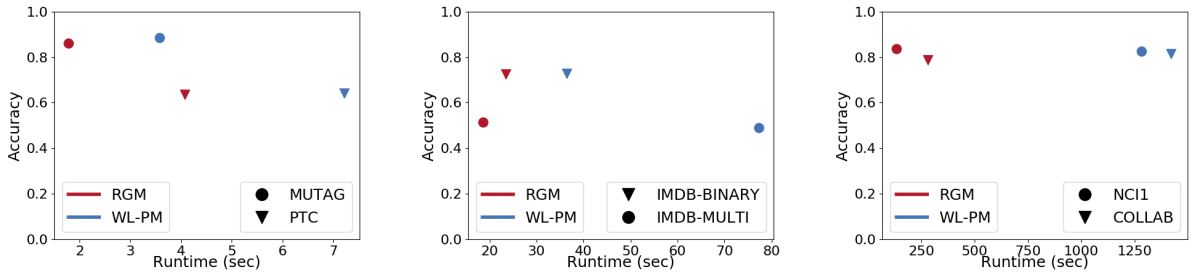
We used MATLAB public implementations of the SP, GR, and WL-ST baselines [She18]. We used the official implementations of NetLSD, LWL-3, and RetGK written in Python, C++, and MATLAB respectively, as well as a MATLAB implementation of WL-OA from the authors of the paper [KGW16]. We implemented the PM kernel following [NMV17] in Python, along with RGM. All experiments ran on an Intel(R) Xeon(R) CPU E5-1650 at 3.50GHz with 256GB RAM.

5.5.2 Accuracy of RGM

Task. We perform 10-fold cross validation averaged over five trials and report the average accuracy and standard deviation.We use a linear SVM to classify feature mappings and a kernel SVM classifier for kernel matrices, all from scikit-learn [PVGea11], limiting the solver to 10^4 iterations and choosing the SVM parameter C by cross-validation from $\{10^{-3}, 10^{-2}, \ldots, 10^3\}$.

Table 5.4: Accuracy of RGM versus graph kernels, feature learning algorithms, and deep neural networks. We see that RGM is one of the most accurate methods on all datasets, compared to baselines from many different fields. (*: Results reported from original papers. For DCNN, we report results, which did not include standard deviations, from the original paper [AT16] on datasets used in that paper. We report the remaining results from [ZCNC18]. >12hr means that computation was not finished within 12 hours.)

Method	MUTAG	PTC-MR	NCI1	IMDB-BINARY	IMDB-MULTI	COLLAB
DCNN*	67.0	55.3	62.6	49.1 ± 1.37	33.5 ± 1.42	52.1 ± 0.71
PSCN*	89.0 ± 4.37	62.3 ± 5.68	76.3 ± 1.68	71.0 ± 2.29	45.2 ± 2.84	72.6 ± 2.15
DGCNN*	85.8 ± 1.66	58.6 ± 2.47	74.4 ± 0.47	70.0 ± 0.86	47.8 ± 0.85	73.8 ± 0.49
NETLSD	82.9 ± 0.58	58.7 ± 1.06	62.6 ± 0.25	64.6 ± 0.39	45.9 ± 1.04	66.7 ± 0.11
GR	83.1 ± 0.77	56.7 ± 0.65	62.8 ± 0.08	55.1 ± 0.83	37.0 ± 1.99	60.4 ± 0.08
SP	88.2 ± 0.24	57.6 ± 0.49	66.2 ± 0.44	51.9 ± 1.31	35.4 ± 1.08	43.0 ± 3.27
WL-ST	86.3 ± 1.13	63.0 ± 1.54	82.2 ± 0.19	72.3 ± 0.35	47.7 ± 0.55	78.4 ± 0.15
LWL3	84.0 ± 1.14	58.8 ± 1.52	77.8 ± 2.12	72.3 ± 0.63	46.0 ± 1.22	>12hr
WL-OA	86.0 ± 0.82	62.2 ± 1.10	82.9 ± 0.23	73.3 ± 0.15	48.2 ± 1.04	80.6 ± 0.29
RETGK	86.3 ± 1.22	61.4 ± 0.87	80.7 ± 0.19	72.6 ± 0.83	45.5 ± 0.79	80.8 ± 0.32
WL-PM	88.4 ± 1.10	82.6 ± 0.21	73.0 ± 0.48	49.1 ± 0.73	81.5 ± 0.35	
RGM	87.8 ± 1.05	63.6 ± 1.53	83.7 ± 0.19	73.0 ± 1.04	51.5 ± 0.40	78.6 ± 0.13



(a) Small datasets: MUTAG & PTC-MR **(b) Mid-size datasets: IMDB-B & IMDB-M** **(c) Large datasets: NCI & COLLAB**

Figure 5.3: Upper left quadrant is best: Accuracy vs runtime for RGM and its closest competitor WL-PM. We denote datasets by marker shape and methods by color. Across all sizes of datasets, RGM has comparable accuracy and considerably faster runtime.

Results. We report graph classification accuracy over all baselines and RGM in Table 5.4. RGM yields highly competitive performance against existing graph kernels. It is the most accurate method on two datasets: the most of any method, tied only with WL-PM. The only dataset where WL-PM outperforms RGM significantly is COLLAB, as it only ekes ahead on PTC-MR. However, RGM outperforms WL-PM significantly on both NCI1 and IMDB-M. Moreover, RGM is never lower than fourth best out of all the baselines on each dataset: a consistent performance (all other baselines besides WL-ST and WL-PM finish in the bottom half at least once).

Compared to the recent feature representation NetLSD, RGM is more accurate on all datasets under consideration. A partial explanation for this may be that NetLSD does not use node labels. However, even on datasets that do not have node labels (the three collaboration datasets), the Weisfeiler-Lehman framework can be used to generate meaningful label expansions that RGM can capitalize on but NetLSD cannot.

Finally, compared to published results from recent and widely used deep neural network methods, RGM performs highly favorably. It is more accurate than all of them on almost all datasets, in many cases (NCI, COLLAB) by a wide margin. One note is that on the smallest datasets MUTAG and PTC-MR, we see extremely high variances especially for PSCN. Many deep learning models for graph classification have been noted [ZCNC18] to overfit on smaller datasets in particular, which is one of the practical difficulties of training them.

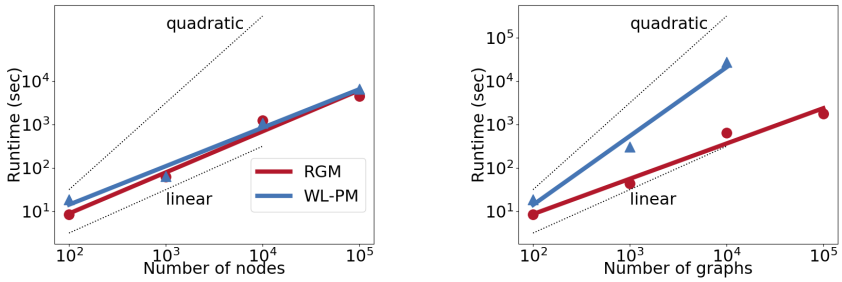
Observation 5.1. RGM *is among the most accurate methods for graph classification, compared to a variety of powerful recent baselines. It is competitive with leading techniques from all three major areas of graph classification literature: unsupervised feature learning, kernels, and deep neural networks.*

5.5.3 Efficiency of RGM

We now focus on the runtime of RGM, as this is a significant practical benefit afforded by explicit feature maps compared to many other methods such as kernels. Here we focus on the pyramid match kernel, which is the most related baseline both conceptually and in terms of results (Table 5.4).

Task. We study the accuracy versus runtime taken to compare embeddings using RGM and WL-PM on our six benchmark datasets. We group the two smallest datasets (the bioinformatics datasets MUTAG and PTC-MR), the two medium-size datasets (the two IMDB datasets), and the two largest datasets (the NCI bioinformatics dataset and the COLLAB dataset) together so that the plots include comparable magnitudes of runtime.

Results. In Figure 5.3, we see that not only does RGM lead to highly *accurate* graph classification, its *runtime* is favorable compared to implicit kernel methods that must compute and manipulate a quadratic kernel matrix. The speedup afforded by RGM is apparent on



(a) Runtime w.r.t. number of nodes **(b) Runtime w.r.t. of graphs**

Figure 5.4: Scalability of RGM. Dotted linear and quadratic slopes plotted for reference. RGM scales linearly with respect to both the *number* and *size* of the input graphs. In contrast, the PM kernel does not scale with the number of graphs.

all sizes of datasets, but is particularly noticeable on large datasets. Meanwhile, accuracy is very comparable, as also seen in Table 5.4.

Observation 5.2. RGM *achieves a favorable balance of accuracy and speed compared to exact kernel methods.*

We further illustrate this point by constructing large datasets and studying the scalability of the two methods as the number or size of the graphs increases in a controlled manner.

Task. To evaluate the scalability of RGM compared to PM, we measure both methods' runtime for graph classification based on comparing embeddings of increasingly large Erdős-Rényi graphs with random binary labels. We use eigenvector embeddings for RGM as well as PM and do not use WL label expansion. For our first experiment, the datasets consist of 100 graphs of 100-100K *nodes* each. In the second experiment, the datasets consist of 100-100K *graphs* of 100 nodes each.

Results. We plot the runtime averaged over five independent trials in Figure 5.4. In Figure 5.4a, we see that both methods scale approximately linearly with the number of nodes in the input graphs, as their asymptotic complexities suggest. However, in Figure 5.4b, the kernel-based classifier used by PM is much slower than the linear SVM that can be used with RGM. Indeed, we cannot even compute the quadratic 100K by 100K kernel matrix for PM within 12 hours. However, RGM finishes well within this timeframe on 100K graphs, and is indeed faster for all numbers of graphs we consider in this experiment. We see that it scales approximately linearly with the number of graphs, in accordance with its asymptotic complexity [Joa06].

Observation 5.3. RGM *is an efficient method for graph comparison and classification, scaling linearly in both the* number *and the* size *of graphs. It can be used on datasets with too many graphs for exact kernel methods such as PM.*

5.5.4 Study of Embedding and Aggregation Methods

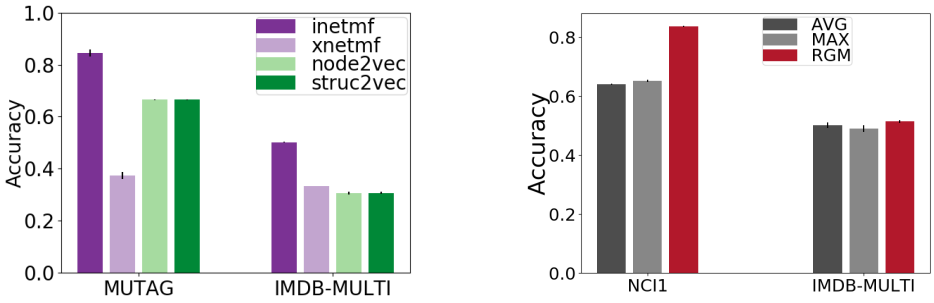
Given that RGM takes node embeddings as input, it can be seen as a two-step process consisting of learning node representations and aggregating them into a feature map for a graph. Here we consider alternative design choices per step.

Task. First, we compare three embedding approaches before constructing RGM feature maps: **node2vec** [GL16], **struc2vec** [RSF17], and **xNetMF** [HSSK18]. These choices reflect different network embedding objectives [RJK+20]: node2vec preserves proximity between nodes, whereas the latter two preserve structural similarity. Moreover, xNetMF is designed for multi-network settings, whereas the other two are designed for single-network settings.

We embed all graphs in training folds together, followed by embedding all test graphs in a separate step (i.e., **inductive learning)**. To embed graphs jointly using the single-network formulation of node2vec ands struc2vec, we combine their adjacency matrices as blocks as a single block-diagonal adjacency matrix. We perform 10 random walks of length 80, use a window size of 10, and set the embedding dimensionality to $p = 100$. For node2vec we set $p = q = 1$.

Results. We see in Figure 5.5a that off-the-shelf node2vec, struc2vec, and xNetMF all perform poorly as base node embedding methods for RGM. node2vec and struc2vec are designed for a single-graph setting, and even xNetMF, although designed for cross-network tasks, assumes a *transductive* setting where all graphs are given up front. In all cases, the feature space learned for the training graphs is *not guaranteed to be comparable to that learned for the test graphs.* However, our modification of xNetMF, iNetMF, performs dramatically better than its transductive counterpart, as well as node2vec and struc2vec. It succeeds in embedding nodes in test data into the subspace spanned by the training landmarks.

Observation 5.4. RGM *can successfully use advances in node embedding to classify graphs. The most important change that existing embedding methods may need, however, is a way to*



(a) Node embedding methods for inductive classification

(b) RGM vs. embedding pooling methods

Figure 5.5: Best choices for node embedding and aggregation. An inductive graph classification setting shows that node embedding methods designed to preserve relative similarities between nodes that are being jointly embedded (e.g. at training or test time) may lead to incomparability between training and test graphs' embeddings. Given suitable node embeddings, RGM works better than simple pooling methods, which less fully capture the distribution of node embeddings.

ensure continuity of the latent feature space across training and test networks.

For iNetMF, there is little difference in performance compared to a transductive setting. This would also be true of RPF and EIG embeddings, where the computation can be done separately for each graph. However, most work in node representation learning optimizes an objective to preserve relative similarities between nodes [GF18]. Without care, such methods may be led astray in an inductive setting.

Next, we consider alternatives for aggregating embeddings.

Task. We compare our RGM feature maps using iNetMF embeddings to two alternative graph representations using feature pooling, which we call **AVG** and **MAX**. These create a p-dimensional feature vector by taking the average or maximum value, respectively, along each feature dimension.

Results. We see that in Figure 5.5b, in terms of constructing feature representations of graphs, the pooling operations MAX and AVG yield inferior performance to our RGM variants. The margin is larger on graphs with node labels, as we illustrate with the largest labeled graph NCI1; it is smaller on the unlabeled collaboration network IMDB-MULTI. These results confirm the benefits of capturing the embedding distribution more comprehensively with RGM.

Observation 5.5. *Capturing the full distribution of embeddings using* RGM *is more ex-*

pressive than pooling the embeddings using simple summary statistics such as mean or max.

5.6 Conclusion

In this chapter we propose RGM, a feature map that captures the distribution of a graph's node embeddings at multiple levels of resolution. We demonstrate theoretical connections between RGM and existing kernel methods, enhancing its performance with node labels using Weisfeiler-Lehman label expansion. We show that RGM is up to 20% more accurate than competitive baselines from graph kernels, feature learning, and deep neural networks. Furthermore, RGM is up to an order of magnitude faster and scales to larger datasets than the most relevant and competitive exact kernel baseline. RGM thus bridges the gap between node-level tools (node embedding) and graph-level tasks (graph classification), efficiently turning feature descriptors of nodes into a principled and powerful feature descriptor for the network.

Part II: Praxis

CHAPTER VI

Node Similarity Application: Professional Role Inference

Chapter based on work that appeared at KDD 2019 [JHS+19].

6.1 Introduction

This thesis has introduced new data mining methodology using node embeddings: we proposed the structural embedding method xNetMF and formulate solutions to collective network mining tasks at the node and graph level. We verified the effectiveness of our methods using well-established benchmark scenarios; here, the focus is less the solution we obtain and more the methods we propose (we care about our success on the data mining task mainly to demonstrate our method's effectiveness). The remainder of this thesis focuses on the question: what does the *praxis* of node embeddings look like? How are node embeddings used to mine actionable new insights on new datasets?

In this chapter, we the problem of inferring employees' roles in an organizational hierarchy, using a unique new dataset comprising *billions of emails across thousands* of organizations collected by the email-based application of Ann Arbor startup Trove[1], with whom we collaborated on this work. This work has the possibility to inform the multitude of existing third-party email clients and applications that leverage emails to help recommend contacts, suggest responses, and organize and filter inboxes. While such applications typically have access to limited metadata about user emails, such as the sender and received time, they often *do not have complete information about the users themselves.* Therefore, inferring

[1] `https://trove.com/`

characteristics about users, such as their professional roles, can inform the personalization of "smart" email applications.

We formulate this problem as **role inference** for nodes in networks. Building on our work in Chapter III, our goal is to learn embeddings that reflect the structural of roles in a network and also preserve additional information captured in edge *weights* and *directionality*. We then show that these embeddings can be used to characterize nodes according to properties that are related to their structural role in the network.

Our approach to professional role inference relies on the inherent network structure of an email corpus, wherein employees are nodes in the email graph and edges capture email exchanges between employees. These edges may be directed (from sender to receiver) and weighted by the number of emails exchanged between the sender and receiver. In contrast to Chapter III, where we found a one-to-one mapping from nodes in one network to nodes in another network, we want to classify nodes according to their professional role (which will of course map many nodes to the same professional role). Note that these roles may be different across companies; for example, a managerial role does not look the same at a tech giant versus a small startup.

Importantly, to ensure a high level of user privacy, our email network is totally anonymized. It does not use incorporate any sensitive data from the email corpus, such as text, sent/received time, or subject line. Thus, we do not use any additional attributes about the nodes in this work, although if such information were available, it could be incorporated into the embedding as in Chapter III. Using this generalized email network, we build on recent advances in network representation learning, which have been shown to be state-of-the-art in difficult supervised learning tasks on networks. Specifically, we propose **EMBER**, short for **Embedding Email-based Roles** and named for our motivating application.

Our approach efficiently learns node representations that preserve structural similarity of the nodes, which allows us to infer the structural roles of the nodes in the network. Intuitively, a node's structural role in an email network corresponds with the professional role of the employee represented by that node. In our email network dataset, we then predict the professional roles of employees by leveraging multi-class classification over their nodes' embeddings, as shown in Figure 6.1.

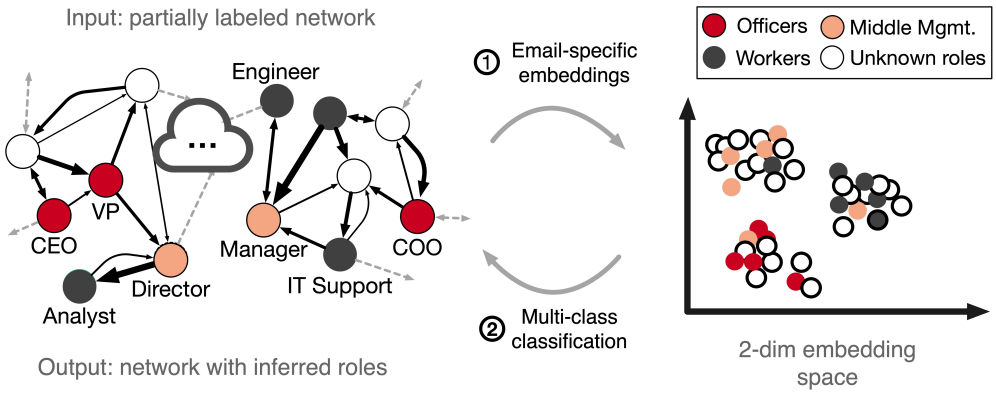


Figure 6.1: EMBER leverages communication volume and reciprocity to (1) compute email-specific structural embeddings, and then (2) infer professional roles via multi-class classification.

The contributions of this work include:

- **Weighted, directed structural embeddings**: We propose EMBER, a powerful and fast approach for embedding some or all of the nodes in a graph in a way that preserves structural similarity. We measure the structural similarity of nodes in terms of both their incoming and outgoing edges as well as the strength (edge weight) of the connections they form.

- **Analysis and insights**: We show that EMBER is effective and efficient in inferring professional roles on several large-scale email corpora. We work with a unique email dataset collected by the Trove email application, comprising several *billion* email exchanges that span *multiple* organizations and sectors, unlikely previously analyzed email corpora. Regarding each organization as a network, we thus show that it is meaningful to compare structural roles of nodes *across* networks.

Code for EMBER is available at `https://github.com/GemsLab/EMBER`.

This chapter is structured as follows. In Section 6.2 we discuss related work.Motivated by our findings, we introduce EMBER in Section 6.4 and then apply it on several large-scale experiments in Section 6.6. Finally, we conclude in Section 6.7 with future directions.

6.2 Related Work

Relevant areas of work include email network analysis, embeddings, and semi-supervised learning over networks. We survey related works on node embeddings in Chapter II. Here we give an overview of alternative approaches to email-network analysis, based on hand-engineered features or supervised learning without features. We qualitatively compare related methods, whether or not they use node embeddings, that can be applied to our problem in Table 6.1, and also compare them experimentally in Section 6.6.

Email network analysis. User behaviors in email networks have been studied for modeling [YDBA17, HL12, ARKG13, Wan14], spam and fraud detection [THC07, KVF13, ATK15], and email ranking [ZWW09] purposes. Most works leverage textual features such as email addresses, body sentiment words, length of subjects [YDBA17], recipients [ZWW09], reply time, and email size [OLJT13] to characterize email behaviors. Recent work learns representations of personal information items (like files, search queries, appointments, etc.) to model user behavior even more comprehensively [SFS+20]. However, to maintain user privacy in the real-world scenarios that interest us, we avoid methods that rely on textual features of email data.

Another direction involves the computation of network centralities. For example, Zhu et al. [ZWW09] propose Inner- and Outer-Pagerank centrality to distinguish nodes that mainly interact within and across communities. Aliabadi et al. [ARKG13] classifies professional roles based on graph centralities including in-/out-/total degree, clustering coefficient, PageRank, HITS, and betweenness.There are other works [SHH+06, RCHS07] combining textual (e.g., mean response time) and network features (e.g., hubs, authorities, cliques). We compare such networked approaches to our own in Section 6.6.

Semi-supervised learning. Professional role inference in email networks (from a technical standpoint, multi-class classification) can also be modeled as semi-supervised learning [Zhu05, BC01] or belief propagation [YFW03, **?**]. The key idea is to leverage not only labeled, but also unlabeled data, during the classification task. One related work from this domain is LinBP [GGKF15], a linearized version of belief propagation that can handle a mix of homophily and heterophily in multi-class settings. It should be noted, though, that such

Table 6.1: Qualitative comparison of EMBER to alternatives. (1-2) Directionality & connection strength: Can the method handle directed and weighted edges? (3) Node specific: Can it embed only a subset of nodes? (4) Proximity independence: Is it independent of node proximity? (5) Scalable: Is it subquadratic in the number of nodes?

	Directionality	Conn. strength	Node specific	Prox. indep.	Scalable
SNA [ARKG13]	✓	✓	✗	✗	✗
Rolx [HGER+12]	✓	✓	✗	✓	✓
LINE [TQW+15]	✓	✓	✗	✓	✓
node2vec [GL16]	✓	✓	✗	✗	✓
struc2vec [RSF17]	✗	✗	✗	✓	✗
GraphWave [DZHL18]	✓	✓	✗	✓	?
DNGR [CLX16]	✓	✓	✗	✗	✗
LinBP [GGKF15]	✗	✓	✗	✗	✓
EMBER	✓	✓	✓	✓	✓

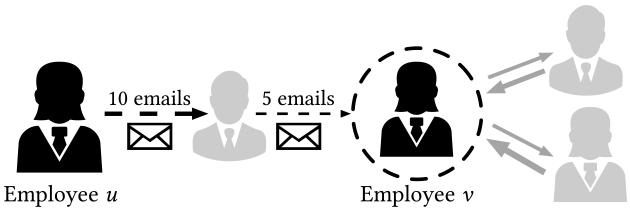


Figure 6.2: Illustrative example of structure in email networks. Employee u's 2-step out-neighborhood $\mathcal{N}_u^{2+}$ consists of employee v, and the weight of the path (Section 6.4.1) from u to v is $10 * 5 = 50$.

methods require explicitly specifying the amount of homophily between connected nodes, which may not be known in advance.

6.3 Preliminaries

We consider a weighted, directed graph $G = (V, E)$. In an email network such as the one we consider, the graph's nodes V represent employees or, more generally, users of the email client in question, and the edges $E \subseteq V \times V$ corresponds to directed communications between employees. An edge has weight w_{uv}, which captures the number of emails employee u has sent employee v, and vice versa for w_{vu}. Let $\mathcal{U} \subseteq V$ be the subset of nodes (employees) in V for whom we want to infer roles (those for whom we do not have ground-truth labels).

Next, we define directed neighborhoods in the email network. Given a node u, let $\mathcal{N}_u^{k+}$ be u's k-step out-neighborhood, or the nodes that can be reached in a directed path of k edges from u. For example, u's out-neighborhood for $k = 1$ are all the nodes toward which u has outgoing edges. Likewise, let $\mathcal{N}_u^{k-}$ be u's k-step in-neighborhood, or the employees

from which u is reachable by a directed path of k edges. We give an illustrative example of directed neighborhoods in the context of email networks in Figure 6.2, where employee (node) u's 2-step out-neighborhood $\mathcal{N}_u^{2+}$ consists of employee v.

Finally, let $\mathcal{P}_{u\to v}^{k+}$ be a directed k-step shortest path from node u to $v \in \mathcal{N}_u^{k+}$. In Figure 6.2, the path $\mathcal{P}_{u\to v}^{2+}$ consists of two edges: one from u to the intermediary gray employee, and one from the intermediary employee to v. Ingoing paths are similarly defined.

6.4 EMBER: Embedding Email-based Roles

Our proposed method, EMBER, is motivated by our observation that outgoing and incoming edges have different semantic meaning (e.g. sending versus receiving an email) and should thus be analyzed separately. Moreover, edge weights should be used when analyzing the effect of a node's connections within its neighborhood: intuitively, a if a node has a (strong) high-weight connection to a neighbor, then that neighbor should influence its structural identity more than a neighbor to which the original node only has a tenuous (low-weight) connection. In the context of email, properties of an employee's regular contacts are probably more informative than properties of the people with whom the employee only exchanges emails a few times.

The steps of EMBER are:

S1 Capturing *weighted* and *directed* local network structure around each node (Section 6.4.1),

S2 Learning embeddings that preserve node similarity based on this local structure (Section 6.4.2),

S3 Role inference via multi-class classification (Section 6.4.3).

In this section, we describe each step in detail, and conclude with the asymptotic complexity of EMBER in Section 6.4.4.

6.4.1 Structural Behavior in Weighted, Directed networks

First, we want to mathematically capture local structure around each node (step **S1**), with the ultimate goal of later obtaining embeddings that preserve the similarity between

nodes with similar local structure, which we will show lends itself very well to professional role inference.

Capturing active communication. Intuitively, an important part of characterizing the neighborhood of each mpde u is identifying important neighbors with which u has strong direct or indirect connections.Given node u's k-step in and out neighborhoods $\mathcal{N}_u^{k+}$ and $\mathcal{N}_u^{k-}$ (Section 6.3), we propose to capture this intuition by *weighting paths* between u and its (in/out) neighbors. These path weights will be used in our final definition of structural behavior, when we formulate a unified version of "what the neighborhood around u looks like" (Section 6.4.1).

We define the weight of an outgoing k-step path $\mathcal{P}_{u\to v}$ as the product of all edge weights in the path, i.e.,

$$\texttt{path\_weight}\left(\mathcal{P}_{u\to v}^{k+}\right) = \prod_{(i,j)\,\in\,\mathcal{P}_{u\to v}^{k+}} w_{ij}, \tag{6.1}$$

There are other ways to define path weights, for example with summations instead of products, but this is not essential to our work and we find empirically that products work well. In our simple example in Figure 6.2, the path weight from employee (node) u to employee v is $10 * 5 = 50$. Note that it is not the *exact* value of the path weight, but rather the relative values of path weights as compared to each other, that will be important.

Structural behavior histograms. As a reminder, our ultimate goal is to define a mathematical notion of "structural behavior" that captures the local structure surrounding each employee in the email network, where local structure includes edge directionality (received/sent emails) and weights (volume of communication). We propose to do this by creating a weighted histogram (i.e., a vector of counts) per node u that captures *what the neighborhood around u looks like*, using the previously defined path weights, as well as the degrees of u's neighbors, which themselves capture how well-connected those neighbors are.

Let $\mathbf{d}_u^{k+}$ ($\mathbf{d}_u^{k-}$) be employee u's outgoing (ingoing) *structural behavior vector* in her k-step neighborhood. Each entry of this vector, or histogram, captures the employees in u's k-step neighborhood of a certain level of connectedness (i.e., of degree Δ), and also incorporates the weight of the path from u to each node of that degree, which can be seen as the importance

of those nodes to u. Here, we use a logarithmic grouping scheme to group larger ranges of high-degree nodes together, to reflect the skewed (power law) distribution of communication commonly observed in real-world social and information networks.

Let D_u^{k+} be the set of nodes in u's k-step out-neighborhood with degree Δ. In other words, $D_u^{k+} = \{v \in \mathcal{N}_u^{k+} \mid \lfloor \log_2(\deg(v)) \rfloor = \Delta\}$. Then, we define the Δ-th entry of u's outgoing structural behavior histogram at k steps as

$$d_{u,\Delta}^{k+} = \sum_{v \in D_u^{k+}} \texttt{path\_weight}\left(\mathcal{P}_{u \to v}^{k+}\right), \tag{6.2}$$

with ingoing structural behavior at k steps defined similarly.

Putting it all together. To capture higher-order information in the network beyond direct connections, we want to capture local structure for each node in the network *across* different distances k. Therefore, we propose a formulation to this end that captures the diminishing importance at higher step distances k (i.e., for nodes not as closely connected to u in the network). As such, given a maximum step distance K (limited by the diameter of the network), we define the overall outgoing structural behavior $\mathbf{d}_u^+$—note the absence of the k superscript here, which distinguishes from the definitions in Section 6.4.1—as a linear combination of k-step structural behaviors $\mathbf{d}_u^{k+}$:

$$\mathbf{d}_u^+ = \textstyle\sum_{k=0}^{K} \delta^k \, \mathbf{d}_u^{k+}, \tag{6.3}$$

where δ^k is a "discount factor" to capture thet diminishing importance of higher step distances. As with all previously described equations, the ingoing behavior histogram $\mathbf{d}_u^{k-}$ is constructed similarly. Finally, to unify the ingoing and outgoing histograms, which will allow us to obtain embeddings as discussed in the next section, we simply concatenate the in- and out-histograms to obtain the final structural behavior vector for node u as $\mathbf{b}_u = [\mathbf{d}_u^+, \mathbf{d}_u^-]$.

6.4.2 From Structural Behavior to Embeddings

So far we have constructed per-node structural behavior histograms $\mathbf{b}_u$ by following ingoing and outgoing paths. Our next goal is to use these histograms to obtain latent features via *embeddings*, which we will show in Section 6.6 are powerful tools for role inference. As

it has been shown that many existing embedding methods implicitly or explicitly factorize a node-to-node similarity matrix $\mathbf{S}$, whose construction varies by method [QDM+18], we take advantage of this connection and turn to fast and theoretically-sound *implicit* matrix factorization for a scalable approach (step **S2**).

To distinguish the conceptual differences between explicit and implicit matrix factorization for node embedding, consider that in the *explicit* matrix factorization approach, we would need to construct and factorize a node-to-node similarity matrix $\mathbf{S}$ that captures the similarity between nodes' structural behavior histograms $\mathbf{b}_u$. But instead of *exactly* constructing the full matrix $\mathbf{S}$, which is quadratic in the number of nodes to embed, and learning an *approximate* factorization of $\mathbf{S}$, we utilize a low-rank *approximation* of $\mathbf{S}$ that *never* has to be computed, because its decomposition has a known, *exact* factorization. Here, we adapt our method from Chapter III for constructing embeddings from the Nyström decomposition:

Theorem 6.1. *Given a network G with a $|\mathcal{V}| \times |\mathcal{V}|$ structural similarity matrix $\mathbf{S} \approx \mathbf{Y}\mathbf{Z}^T$, its node embedding matrix $\mathbf{Y}$ can be approximated as $\mathbf{Y} = \mathbf{C}\mathbf{U}\mathbf{\Sigma}^{1/2}$, where $\mathbf{C}$ is the matrix of similarities between the $|\mathcal{V}|$ nodes and p* landmark *nodes [DM05], and $\mathbf{W}^\dagger = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top$ is the SVD of the pseudoinverse of the $p \times p$ landmark-to-landmark similarity matrix $\mathbf{W}$.*

The key takeaway is that we select a small number p of nodes called *landmarks*, and compare the nodes for whom we want to learn embeddings to infer roles *against the landmarks*. Let us assume that we want to infer the roles for all the nodes V in the network. Therefore, to obtain structural embeddings via the technique above, we only need to perform a *small fraction of node-to-node comparisons* $|\mathcal{V}| \times p$ stored in $\mathbf{C}$ ($p \ll |\mathcal{V}|$), and a few "expensive"' computations on the *small* $p \times p$ (sub)matrix $\mathbf{W}$.

Now, it is only left for us to discuss: (1) how we compute structural similarity between two nodes' structural behavior histograms $\mathbf{b}_u, \mathbf{b}_v$; (2) how we select the landmarks; and (3) how we embed only a set of nodes of interest, which makes EMBER even *more* scalable than the technique described in Theorem 6.1.

Structural user similarity. We define the similarity between two nodes u and v based on their structural email behaviors as $sim(u, v) = e^{-||\mathbf{b}_u - \mathbf{b}_v||}$, where $||\cdot||$ is a vector norm, for example Euclidean distance. Recall that our setting assumes no additional side information

for nodes beyond the structural information in the graph. (This is done for privacy reasons in our email application. However, we could include a term to incorporate the attribute similarity of nodes if this information were available, as in Chapter III.)

Landmark selection. In EMBER the number of landmark nodes p determines the dimensionality of the generated embeddings. The landmark nodes, used for the construction of the "thin" $\mathbf{C}$ similarity matrix, can be sampled uniformly at random [WS01] or according to more sophisticated matrix theoretic methods [KMT12]. Domain-specific heuristics, such as sampling nodes with probability proportional to their degrees, are fast to compute and thus plausible to use. Indeed, for the problem of professional role inference, they lead to more competitive and stable classification accuracy than random selection. Intuitively, since the embeddings preserve similarity with respect to the landmarks, to capture diverse structural behavior in the embeddings it is advantageous to ensure structural diversity in the landmarks.

User subset embedding. In many applications, it is only necessary to embed some of the nodes $\mathcal{U} \subseteq V$, where $|\mathcal{U}| \ll |\mathcal{V}|$. For example, in our application, an email client might only need to infer the organizational roles of a small subset of employees of interest.

As the embedding computations in Theorem. 6.1 involve direct comparison only to landmarks, we can embed *any* subset of nodes, as opposed to the entire network, which makes EMBER unique among representation learning techniques. Specifically, $\mathbf{C}$ can easily be adapted to be a $|\mathcal{U}| \times p$ matrix that holds the user-to-landmark similarities only for *employees of interest*.

6.4.3 Professional Role Classification

Given the embeddings from §6.4.2, we infer organizational roles via multi-class classification (step **S3**). We assume that the roles of some nodes are known, and predict the roles of the remaining nodes using supervised machine learning techniques on their embeddings. We give more details on the task setup in Sec. 6.6.2. The overview of EMBER is given in Algorithm 6.1.

Algorithm 6.1 EMBER: EMBedding Email-based Roles

`Input`: Email network $G = (V, E)$, nodes of interest $\mathcal{U} \subseteq V$, maximum step K, discount factor $\delta \in (0, 1]$
`Output`: Roles for the node of interest $\mathcal{U}$

`S1: Capture structural behavior in email network`

Outgoing / incoming structural behavior histograms $\mathbf{b}_u^{\pm} \leftarrow 0$
for $k = 1 \dots K$ **do**
    Construct k-step outgoing / incoming histograms $\mathbf{b}_u^{k\pm}$ ▷ Eq. 6.2
    Update outgoing / incoming histograms to $\mathbf{b}_u^{\pm} \leftarrow \mathbf{b}_u^{\pm} + \delta^k \mathbf{b}_u^{\pm}$
Concatenate final histograms into $\mathbf{b}_u \leftarrow [\mathbf{b}_u^{+}, \mathbf{b}_u^{-}]$

`S2: Embed nodes in network`

Select set of p landmark nodes ▷ Section 6.4.2
Compute $\mathbf{C}$ as $|\mathcal{V}| \times p$ similarity matrix of behavior histograms $\mathbf{b}_u, \mathbf{b}_v$
Compute the SVD of the pseudoinverse of the small submatrix $\mathbf{W}$ of $\mathbf{C}$
Obtain embeddings $\mathbf{Y} \leftarrow \mathbf{C}\mathbf{U}\mathbf{\Sigma}^{1/2}$ ▷ `Theorem 6.1`

`S3: Role inference`

Learn a classifier with embeddings $\mathbf{Y}$ and the known roles

6.4.4 Computational Complexity

Here, we analyze the complexity of EMBER steps **S1** and **S2**, since **S3** can be implemented with well-studied supervised machine learning methods. Recall that scalability is an important requirement of our approach, since our task is motivated by the prevalence of third-party email applications that handle large amounts of data.

Assuming that we are obtaining embeddings for $|\mathcal{U}|$ employees, step **S1** of EMBER is $O(|\mathcal{U}|K\Delta_{\text{avg}}^2 + |\mathcal{U}|p \log_2 \Delta_{\max})$. Here, Δ_{avg} is the maximum between the average user in-degree and average user out-degree in the email network. In the second term, the factor of $\log_2 \Delta_{\max}$ in the second term comes from logarithmic binning (Section 6.4.1), with $\Delta_{\max}$ is the maximum total degree in the graph and p being the number of landmarks (Section 6.4.2). Step **S2** requires $O(p^3)$ time to compute the pseudoinverse of the $p \times p$ similarity matrix $\mathbf{W}$, and then $O(|\mathcal{U}|p^2)$ time to left multiply it by $\mathbf{C}$. Since $p << |\mathcal{U}|$, the total time complexity for this step is $O(|\mathcal{U}|p^2)$. For large-scale problems, p, Δ_{avg}, and K are all asymptotically much smaller than $|\mathcal{U}|$, meaning that EMBER runs in time subquadratic to $|\mathcal{U}|$.

6.5 Data

In this section, we introduce our datasets and discuss how we standardized and cleaned them.

6.5.1 Email Corpora

New email dataset. Our new dataset, collected by the Trove AI email application, consists of over 3.51 *billion* post-2014 emails from $\sim$ 130 000 users and their contacts. As far as we know, this is the first dataset studied in email network analysis that contains both *intra-* and *inter*-organization emails: exchanges between employees of the same company and exchanges between employees of different companies, respectively. Per record, we retain only a datestamp and the anonymized sender and receiver IDs. We also collected ground-truth organizational roles by gathering email-to-organizational role mappings using an email signature parsing tool and information from a third-party data provider, with the consent of app users. This information is used only for evaluating EMBER.

We construct several weighted, directed email subnetworks from Trove's email corpus. In each network, each node is an employee and directed, weighted edges represent the number of emails from the sender to the receiver. We give some descriptive statistics of the following subnetworks in Table 6.2:

- `Trove`: All email exchanges between employees from several thousand companies during 2017.
- `Trove-19`, ..., `Trove-318`: Each of the five subnetworks captures the internal (intra-organization) emails during 2017 within one company. The number after the dash indicates the number of employees in the respective dataset.
- `Trove-2K`: All email exchanges between the employees of the five companies (`Trove`) and all their contacts (within and across organizations) in 2017.

Established email dataset. We also use the well-studied `Enron` email dataset. This dataset consists of email exchanges in 1999-2002 between the 116 Enron staff [SA06, HHB+03] and their external contacts, for a total of 75 415 email users in the network. This is the only publicly available email corpus containing employee role information. The basic statistics of the `Enron` corpus are given in Table 6.2.

Table 6.2: Overview of our datasets, consisting of sub-networks of `Trove` and `Enron`. We give the number of employees (nodes), connections (unweighted, undirected edges), email exchanges (weighted, directed edges), and the ground-truth distribution of roles (Section 6.5.2: O = Officer; M = middle management; W = worker).

	Employees	Connections	Emails	Max in-degree	Max out-degree	# O | M | W
`Trove-19`	19	47	274	7 (115)	6 (103)	4 | 10 | 5
`Trove-98`	98	101	1769	20 (204)	4 (226)	53 | 32 | 13
`Trove-141`	141	1 242	9565	45 (644)	55 (1659)	23 | 79 | 39
`Trove-183`	183	3136	21 655	56 (827)	75 (1853)	16 | 133 | 34
`Trove-318`	318	1026	12 643	51 (2365)	46 (1306)	30 | 210 | 78
`Trove-2K`	2 414	16 281	183 443	97 (4197)	118 (4392)	495 | 1 300 | 620
`Trove`	9 989 507	40 290 044	568 678 419	51 425(12 066 716)	150 481 (30809076)	495 | 1 300 | 620
`Enron`	75 416	319 935	2 064 442	1 442 (19198)	1 389 (65675)	31 | 44 | 41

6.5.2 Professional Roles

Standardization. While the categorization of professional roles may differ by organization and domain area, we follow established literature [Har90, CB06] in organizational studies to define three hierarchical professional roles. We adopt the terminology of [CB06] in particular, and classify all employees as one of:

- **Officers**: These are "C-Suite" employees, meaning top-level officers such as CEO, COO, and other executives. We also grouped co-founders of organizations into this class.
- **Middle management**: These are middle-level managers responsible for coordinating the vision of officers by directing lower-level employees [CB06]. We included all non-officer employees with titles including "Manager" in this class.
- **Workers**: These are employees who directly contribute to the day-to-day work of the company. As to be expected, the titles in this category are more diverse, and include associates, assistants, engineers, salespeople, etc.

Note that while some organizations may be more or less hierarchical than the categorization we adopt, we use these well-established groupings to delineate between *clearly distinguishable roles* (e.g., salesperson versus CEO) while avoiding *arbitrary distinctions* (e.g., project manager versus senior project manager), which differ between organizations and change over time.

To categorize each employee into a hierarchical role, we match each professional role to a set of manually curated keywords. We manually validate the categorizations due to the

complexity of real-world job descriptions: for example, a "front office executive" is likely a "worker", not "officer". We categorized all employees in both the `Trove` and `Enron` datasets. If an employee's role changed during the period of time that is captured in the email network representation, we use her latest role as ground-truth. We give the distribution of professional roles per dataset in Table 6.2.

6.6 Analysis and Insights

In this section we present analysis and insights by putting EMBER into practice. Our main research questions are:

Q1 How does EMBER compare to the state-of-the-art in professional role inference?

Q2 How efficiently can EMBER infer professional roles?

Q3 Do roles across organizations of different sizes and sectors compare? What insights can we gain from role correspondences across organizations?

We ran all our analyses on a machine with a 6-core 3.50GHz Intel Xeon CPU and 256GB memory.

6.6.1 Experimental Setup

Here we briefly describe how we set up our experiments, including variants of EMBER we studied, baselines to which we compared EMBER, and choices of parameters for all methods compared.

EMBER variants. One of the main hypotheses of this work is that capturing email-specific behavior via sent/received emails and the volume of communication in the network is important in professional role inference. To test this hypothesis, we conduct our role inference experiments with three variants of EMBER beyond the one proposed in Section 6.4: **EMBER-U** operates on unweighted, undirected graphs; **EMBER-D** only uses edge directions; and **EMBER-W** only considers edge weights. We run all variants of EMBER with maximum step distance $K = 2$ and discount parameter $\delta = 0.1$. We select the p landmark nodes with probability proportional to their degrees.

Baselines. Professional role inference can be approached with a variety of techniques. In our evaluation we consider *nine* baselines spanning well-known social network analysis, unsupervised and semi-supervised learning, and network embedding techniques. From the *non-embedding* literature, we compare to:

1. SNA or Social Network Analysis [ARKG13] classifies roles based on graph statistics including degree, clustering coefficient, PageRank, HITS, and betweenness. To make the computation on the two largest networks (`Enron` and `Trove`) feasible, we estimate the betweenness centrality by sampling 1 000 users.

2. RolX [HGER+12] is an *unsupervised* method that automatically infers structural roles via non-negative matrix factorization. We use the default settings provided in the paper.

3. LinBP [GGKF15] is a belief propagation approach that leverages both the input labels and the network structure for classification. As input it requires a matrix of potentials $\mathbf{H}$, which defines the *homophily* between the different professional roles. We set it to [.45 .35 .2; .25 .5 .25; .25 .3 .35] based on the frequency of interactions between officers, middle managers, and workers in `Trove-2K`.

The *embedding* methods that we compare to are:

4. LINE [GL16] We use 2nd-LINE to incorporate 2-order proximity and set other parameters to the provided defaults.

5. DeepWalk [PARS14] is a proximity-based embedding method that obtains node context via random walks.

6. Node2vec [GL16] is a generalization of DeepWalk that strikes a balance between homophily and structural equivalence. We set its random walk hyperparameters $p = 1$ and $q = 100$ to put more emphasis on structural equivalence, as other settings resulted in worse performance.

7. DNGR [CLX16] uses a deep neural network on the positive pointwise mutual information matrix to embed weighted graphs. We use a 3-layer neural network model and set the random surfing probability $\alpha = 0.98$, as recommended in the paper.

8. Struc2vec [RSF17] is an embedding method that preserves structural similarity, unlike the previous approaches. It is the most related to EMBER and RolX. We keep the default settings stated by the authors with all 3 optimizations.

9. GraphWave [DZHL18] computes structural embeddings based on heat wavelet diffusion. To evaluate the characteristic functions we use $\tau = p$ timepoints (equal to the dimensionality), and the default values for all the other parameters.

For all the embedding methods, including ours, we follow the literature by setting the dimension $p = 128$ for the email networks with more than 128 employees. Note that in the case of EMBER, the number of landmarks p corresponds to the embedding dimensionality p. For the smaller networks `Trove-98` and `Trove-19`, we set dimension $p = 64$ and $p = 16$, respectively.

6.6.2 Predicting Professional Roles

In this section, we address question **Q1**, the key application and driver of our work: How accurately can EMBER infer employees' professional roles from email network data?

Methodology. As discussed in Section 6.5.2, we cast the professional role inference problem as a multi-class classification task with three roles: *officers*, *middle management*, and *workers*. We evaluate all methods using the ground truth organizational roles per dataset (Table 6.2). For all the *supervised methods*, we feed the generated node representations (hand-crafted features for SNA, and embeddings learned from the rest) as inputs to the classifier. Our classification model is a one-vs-all SVM with linear kernel (penalty C=1, 10^6 iterations, and 10^{-6} tolerance); other models yielded similar results.

We perform 5-fold cross-validation across methods and datasets, and report the average (across folds) micro-AUC over all classes. For LinBP, which is *semi-supervised*, to imitate the 5-fold CV setting for the supervised methods, we select 80% of employees with ground truth to construct the explicit beliefs matrix $\mathbf{E}$—i.e., the known employee roles. LinBP then directly assigns a class to each user based on her maximum final belief. For RolX, which is an *unsupervised method*, we report the accuracy of the best match between the identified (structural) roles and the ground truth classes. Table 6.3 presents the micro-AUC results.

Table 6.3: Performance (AUC) of role inference across datasets and methods. "—" means that the method failed to finish within our time limit (12 hrs). EMBER and its variants prove strong in the role inference task. Moreover, EMBER outperforms its unweighted/undirected variants, demonstrating the importance of accounting for the volume and reciprocity of email exchanges in role inference. The asterisk, *, denotes statistically significant improvement over the best baseline at $p < 0.05$ in a two-sided t-test.

	SNA	RolX	LinBP	LINE	DW	n2v	s2v	DNGR	GW	E-U	E-D	E-W	**EMBER**
TR-318	.7605	.5670	.6908	.6618	.7602	.7648	.7799	.7131	.7685	.7749	.7563	.7625	**.8045***
TR-183	.7648	.5787	.7718	.5657	.8071	.8223	**.8264**	4925	.6391	.7986	.7838	.8186	.8241
TR-141	.6738	.5591	.7409	.7102	.7191	.7474	.7391	.6235	.7112	.7291	.7309	.6971	**.7568***
TR-98	.6676	.5177	.6323	.6872	.5587	.6198	.6498	.5329	**.7177***	.6040	.5857	.6333	.6911
TR-19	.5429	.6981	.6248	.7184	.5531	.5959	.6102	.6089	.7157	.6837	.7204	.6939	**.7337***
TR-2K	.6305	.5212	.6622	.6771	.6769	.6780	**.6802**	.6527	.6594	.6689	.6345	.6677	.6745
Trove	.6633	.5280	5454	—	.6866	.6951	—	—	—	.6905	.7141	.7122	**.7162***
Enron	.6205	.5197	.5000	.6931	.7201	.7389	—	.5709	—	**.7393**	.7347	.7305	.7305

Findings. We immediately observe from Table 6.3 that while professional role inference is challenging, EMBER is clearly well-suited to the task, justifying our email-centric embedding approach over more generic techniques. Indeed, the email-centric design of EMBER leads to a statistically significant improvement over other methods on most datasets, by an average of 2-20%. In the cases where EMBER is not the highest performer, it is a close second by a statistically insignificant margin. The good performance of EMBER is expected, as it is tailored to email networks and their rich structural information. Note that DNGR and GraphWave failed to finish within our time limit (12hrs) on `Trove` and `Enron` (Table 6.4). LINE and struc2vec failed to finish on `Trove`.

Observation 6.1. *Structural roles of users as nodes in email networks are strongly indicative of their professional roles.* EMBER *captures these well with its email-specific design choices.*

Importantly, we find that for all networks other than `Enron`, EMBER performs best *when using both edge connection strengths and directionality.* This confirms our initial hypotheses that the volume and reciprocity of email activity both characterize behaviors, which in turn distinguish professional roles, and justifies our use of such characteristics in the design of EMBER. That said, the `Enron` dataset is an exception. Here, both edge weights and directionality lead to marginal ($< 1\%$) *decreases* in EMBER's accuracy. We hypothesize that this may be due to diverse, erratic email exchange behavior during the company's fraud crisis, which has been well-documented in the media and literature [WB09].

Table 6.4: Average runtime in seconds, capped at 12h. While RoLX is faster for the smaller datasets, EMBER proves uniquely scalable on the `Trove` and `Enron` networks, which have up to *millions* of edges.

	Trove-318	Trove-2K	Trove	Enron
SNA	6.32	16.45	3193.26	333.33
RolX	**0.14**	**0.16**	2150.53	205.92
LinBP	0.54	2.88	14607.44	1038.09
LINE	171.95	153.12	>12h	267.48
DeepWalk	3.12	21.59	2464.13	255.84
node2vec	2.85	24.55	3484.05	254.60
struc2vec	17.48	188.65	>12h	29286.38
DNGR	21.05	72.83	>12h	>12h
Graphwave	2.73	5.66	>12h	>12h
EMBER	2.50	16.87	**830.80**	**84.98**

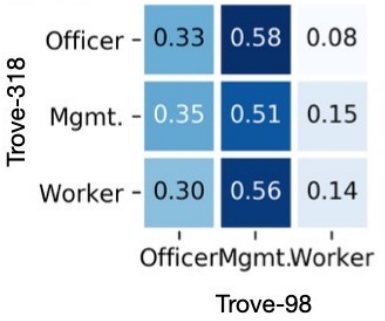
Observation 6.2. *Modeling edge weights and directions generally improves professional role inference.*

6.6.3 Efficiency of Inference

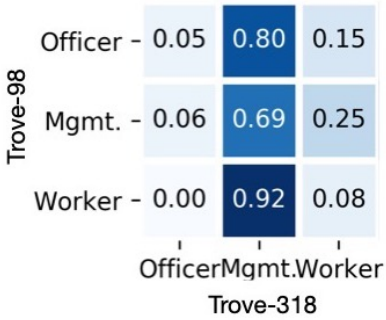
We now turn to question **Q2**: How fast is EMBER? Recall that our initial problem is motivated by the prevalence of third-party email applications that can benefit from role inference over email networks. Therefore, here we investigate whether EMBER is scalable enough to be practical in real-world scenarios.

Methodology. We measured the time required to obtain the roles of employees in email networks of different size in the previously discussed role inference task. In Table 6.4 we report the average runtime in seconds across the 5 folds, and the average across 5 runs of the unsupervised RolX method.

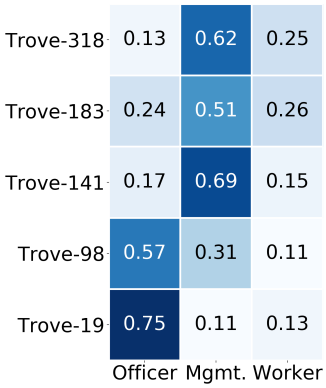
Findings. We find that EMBER proves uniquely scalable for the large-scale `Trove` and `Enron` datasets, being $2.5 - 344\times$ faster than all other methods that complete. This is especially true for the representation learning approaches that are most competitive with EMBER. Indeed, EMBER is over $4\times$ faster than node2vec, and $508\times$ faster than DNGR and GraphWave, based on their (incomplete) runtime of over 12 hours. This is not surprising given that EMBER relies on *implicit* factorization and can embed a given subset of nodes (Section 6.4.2). As a representative example, on the `Trove` network, which has over *40 million edges*, EMBER needs less than *14 minutes* to infer professional roles. EMBER is



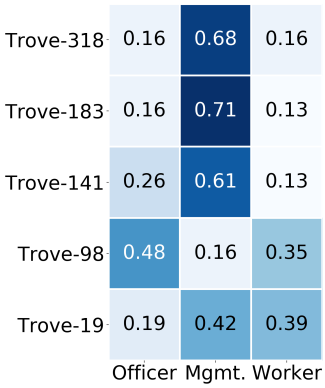
(a) Mapping similar employee behaviors from `Trove-318` to `Trove-98`.



(b) Mapping similar employee behaviors from `Trove-98` to `Trove-318`.



(c) Mapping professors to industry roles.



(d) Mapping graduate students to industry roles.

Figure 6.3: Mapping roles across companies and sectors. (a) and (b) indicate that employees in the bigger company `Trove-318` are similar to positions at and above "management" in the smaller company `Trove-98`, and employees in `Trove-98` are similar to positions at and below "management" in `Trove-318`. (c) and (d) show how similar "Professors" and "Graduate Students" are to job titles in different-sized companies: professors become more "important" in smaller companies (mapping to officers), while students are more similar to the management (or other positions) across companies.

thus highly scalable, making it a practical candidate for real-world analysis of organizational communication, and for third-party email clients that recommend contacts and help prioritize emails.

Observation 6.3. EMBER *is an extremely practical choice for large-scale tasks, much more so than the most competitive baseline methods.*

6.6.4 Comparing Professional Roles

Finally, we address question **Q3** by performing a qualitative study of whether we can use the EMBER embeddings to compare professional roles across organizations. This task

is motivated by the unique nature of our Trove dataset, which comprises emails from many organizations of different sizes and sectors.

Methodology. For the questions we asked in this study specifically, we used both the Trove-2K dataset studied in the previous sections as well as an *academia-specific* dataset collected from a university that collaborates with some of the companies in the Trove dataset. For reference, the academic email network consists of 3 078 users and 231 470 email exchanges.

First, we use EMBER to embed all employees in the Trove-2K network and the academia-specific network. Then, for all pairs of employees, we compute the ℓ_2 norm of the differences between the respective embeddings. We say that employee u at organization A "maps" to employee v at organization B if the ℓ_2 distance between u and v is minimal for all employees compared to u in B: $v = \arg\min_{j \in B} ||\mathbf{y}_u - \mathbf{y}_j||_2$, where $\mathbf{y}_u$ and $\mathbf{y}_j$ correspond to EMBER embeddings of employees u and j, respectively. In Figures 6.3a-6.3b, we show mappings of *officers*, *middle managers*, and *workers* across Trove-318 and Trove-98. The darker the color in the heatmaps, the more frequent is the corresponding employee mapping between the companies.

Findings. Interestingly, most employees at the bigger company (Trove-318) map to *high-ranking* positions at the smaller company (Trove-98), whereas most employees at Trove-98 map to *lower*-ranking positions at Trove-318. One potential explanation is that employees in larger companies may be more well-connected, in and outside of their own companies, and thus appear "higher-ranking" as compared to less well-connected employees at smaller companies. We also observe that middle management roles are similar to *all other* roles across companies, which may be because managers take on many fluid roles in the workplace, from core leadership to more basic day-to-day activities. We see similar patterns across all pairs of companies in the Trove dataset.

Using the academia email network, we also evaluate the similarity between academic roles and industry roles. Here we compare "professors" and "graduate students" to officers, middle management, and workers across the five companies in Trove. We find that professors are indeed similar to CEOs of smaller companies (Trove-98 and Trove-19), and more like managers in bigger companies (Trove-318 through Trove-141). We find this result fairly intuitive, given the day-to-day roles of university professors, who usually manage a (relatively

small) group of students and staff, similar to higher-ranking employees in small companies and middle-ranking employees in large companies. Likewise, we find that graduate students are more like lower-level employees in small companies, suggesting that academic roles have some hierarchical equivalence with industry roles, and especially so in startups.

Observation 6.4. *Professional roles look different at differently sized companies due to the larger overall scale of communication at larger companies. Academic and corporate job hierarchies meaningfully correspond.*

Our analysis has shown that the email-based behaviors of employees are indeed related to the size of the organizations for which they work. Therefore, changes in these role correspondences may inform company dynamics. For example, they may imply ongoing structural shifts which need to be addressed via reorganization [DA79]. We believe that our findings have significant potential to inform business choices in the real world.

6.7 Conclusion

Motivated by the prevalence of email in the workplace and the myriad of third-party email applications that could benefit from inferring characteristics about users, in this paper we study professional role inference in email networks. This chapter serves as a case study for the application of structural node embeddings on a new dataset with both *intra*- and *inter*-organization email exchanges, which enables our unique and extensive experiments and analyses.

We introduce EMBER, which infers roles by leveraging embeddings learned from the *structural behavior* of employees in the network. Our results showed the effectiveness of EMBER, which is $4-25\%$ more accurate and $2.5-344\times$ faster than a wide range of baselines consisting of diverse techniques, from network-scientific feature engineering to node embedding to semi-supervised learning. Our node embeddings also allowed us to uncover interesting new insights about the nature of organizational hierarchy across companies, revealing differences and similarities in roles across companies and employment sectors in a unique case study. We can see that structural node embedding is a powerful tool enabling

a variety of useful qualitative and quantitative insights on large-scale, complex real-world problems.

CHAPTER VII

Evaluating Structural Embeddings

Chapter based on work that appeared at KDD MLG Workshop [JHJK20].

7.1 Introduction

Our final research chapter of this thesis concludes our study of the praxis of node embeddings. In the previous chapter, we saw the effectiveness of structural node embeddings at mining new insights on an industry-scale task, and throughout the thesis we have seen structural embeddings applied in the context of many downstream collective network mining tasks. We hope that the methodology we have introduced in this thesis, and the example use case we have laid out in Chapter VI, will inspire many more effective use cases. To look forward, however, we must first look closely at the current praxis of structural node embeddings. What are current methods learning, and how can we evaluate what a method is learning or should learn? Are the current practices in structural node embedding research optimal for identifying methodological progress?

Though the focus of the previous chapters have largely been on collective graph mining, we focus on the individual graph mining tasks that other research works commonly use to evaluate structural node embedding. (In Section 7.8, we do evaluate a large collection of existing structural embedding methods in the context of our frameworks for collective network mining.) To understand structural node embeddings more deeply, we return to decades-old concepts of network roles and positions in sociology from which many modern methods claim loose inspiration. A *position* or *equivalence class* describes a collection of

individuals with similar *roles*, i.e., similar functions, ties or interactions with individuals in other positions [WF94]. Depending on the type of equivalence (e.g., automorphic, regular—cf. Section 7.3.1), different positions and roles arise that enable both multi-network tasks (e.g., network alignment and classification [HSSK18, RJK$^+$20], transfer learning [HGER$^+$12]) and single-network tasks, including structural role classification, anomaly detection, and identity resolution [JHRK19]. To capture the notion of roles in the network, structural embeddings are typically based on feature-based matrix factorization [HGER$^+$12, HSSK18] or random walks [RSF17], graphlets [ARL$^+$19], or more recently LSTMs [TCW$^+$18].

While proximity-based methods are evaluated rigorously on a set of well-understood tasks using established datasets, the evaluation of structural embeddings is less mature. It relies mostly on limited experiments on a barbell graph, or structural node classification / clustering of *small* real datasets (mainly air-traffic networks) with node labels whose definitions are contrived. It also lacks rigorous connections to the types of equivalence from which role discovery in networks stems.

Our goal in this work is to contribute toward the systematic evaluation of unsupervised feature representations of nodes. In natural language processing, evaluation of unsupervised word representations has long been recognized as an important area of study. Prominent works have as their objective the standardization of evaluation of word embeddings [SLMJ15]. Other works have pointed out additional evaluation methods and challenges to the point where a multi-year workshop has arisen dedicated to the evaluation of word embeddings[1], and the field of word embedding evaluation now warrants a survey [Bak18]. Node embedding, being a comparatively newer area of study, is only now starting to see similar growth, and the recent works that have focused on intrinsic [DG18] or extrinsic evaluation [GHG$^+$19, GVS$^+$19] of node embeddings focus only on proximity-preserving embeddings. Interest in structural embeddings, has been growing, however, and a recent survey distinguishes them from proximity-preserving embeddings [RJK$^+$20]. A standardized analysis of structural embedding methods is essential to ensure that the problem area indeed continues to see forward progress.

Toward this end, we provide **a novel, comprehensive evaluation methodology for**

[1] `https://repeval2019.github.io/`

systematic analysis of structural embedding methods with respect to the sociological theories of equivalence. Our main contributions are:

- **Evaluation Methodology.** This is the first paper to introduce a variety of evaluation methods for *unsupervised structural node embeddings*. These are based on: (i) intrinsic measures related to equivalence definitions (§ 7.3.1), which help us decouple the effectiveness of methods from classifiers in downstream tasks, and (ii) extrinsic measures that characterize their performance in the context of high-value tasks, for which we rethink the ground truth used in prior work.

- **Appropriate Datasets.** We introduce new benchmark datasets, and ways to obtain ground truth roles (§ 7.4). We hope that these datasets will change the way structural embeddings are evaluated.

- **Understanding.** Our empirical analysis of 11 methods (§ 7.3.5) on 35 real and synthetic datasets (§ 7.4) and a variety of tasks shows that different methods win based on different metrics, label definitions, downstream machine learning models, or embedding similarity functions (e.g., cosine vs. Euclidean). This analysis highlights that there is no one optimal structural embedding. Moreover, we evaluate the extent to which sociological equivalences are captured by different structural embedding methods (§ 7.6). Also, besides merely comparing the performance of different methods on downstream tasks, we further analyze their performance at a finer granularity to understand for *which types of nodes* current methods perform best (§ 7.7.4).

- **New Design Insights.** We find that degree distribution in nodes' local neighborhoods is effective as a feature representation in its own right as well as the building block for some of the most successful embedding methods. This can influence the design of future structural embedding methods and/or serve as a standalone baseline for structural embedding tasks.

We have made code that can be used to reproduce our experiments publicly available at `https://github.com/GemsLab/StrucEmbeddingGraphLibrary`.

After reviewing the related work, we present key concepts from social science that have inspired the work on *structural* embeddings.

7.2 Related Work

Understanding Latent Representations. Unsupervised latent feature representations or embeddings have been applied to a wide variety of objects[2], having been shown to yield powerful performance on downstream tasks. However, because these are latent features, it is difficult to interpret them, thus it is unclear why certain methods do well or how to evaluate them. The NLP community has taken this challenge very seriously. Prominent works have as their objective the standardization of evaluation of word embeddings [SLMJ15]. Other works have pointed out additional evaluation methods and challenges to the point where a multi-year workshop has arisen dedicated to the evaluation of word embeddings[3], and the field of word embedding evaluation now warrants a survey [Bak18].

Node embedding, being a comparatively newer area of study, is only now starting to see similar growth. A few recent works [GHG+19, GVS+19] benchmark the performance of popular node embedding algorithms on a variety of tasks and datasets. This can be seen as a form of *extrinsic* evaluation of the embeddings in the context of downstream tasks. Large-scale *intrinsic* evaluation of node embedding is not as common, but another work has sought to understand various proximity-based node embedding methods by seeing what common centralities they are capable of predicting [DG18]. All of the above works, however, focus exclusively on embedding methods that preserve similarity in the latent feature space between nodes that are in close *proximity* in the network. Interest in structural embeddings, has been growing, however, and a recent survey distinguishes them from proximity-preserving embeddings [RJK+20]. A standardized analysis of structural embedding methods is essential to ensure that the problem area indeed continues to see forward progress.

Node Embeddings. For more background information on node embeddings, see Chapter II. In Section 7.3.5, we provide a detailed description of the proximity-preserving and structural node embeddings we empirically evaluate in this chapter.

[2] `https://github.com/MaxwellRebo/awesome-2vec`
[3] `https://repeval2019.github.io/`

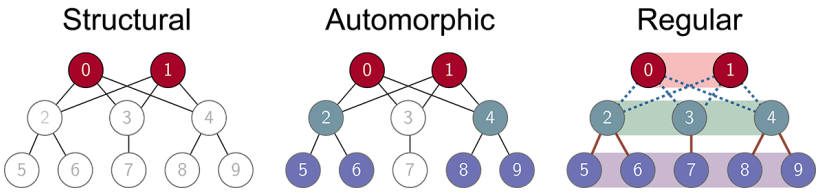


Figure 7.1: Different types of equivalence. Nodes filled with the same color belong to the same equivalent roles.

7.3 Methodology

In this section, we first introduce node embedding and describe in more detail several methods that we will empirically analyze in this work. To understand better what structural node embeddings learn, we turn to concepts introduced in other academic disciplines to analyze the structural roles of nodes: role equivalences in mathematical sociology, as well as statistics developed by network scientists to measure node connectivity and centrality. We then present the tasks for which we evaluate node embeddings, and finally the goals of our research study.

7.3.1 Equivalence in Social Science

Structural embeddings are related to the notions of *social roles* or *positions*, which are central in sociology for understanding how the society or groups are organized. *Role* refers to the patterns of relations between individuals, or the ways in which individuals relate to each other. *Position* or *equivalence class* describes a collection of individuals with similar activity, ties or interactions with individuals in other positions [WF94]. The formal definitions of these terms are based on network methods, which led to their wide adoption in social network analysis. In network analysis, (structural) roles of nodes include centers of stars, peripheral nodes, bridge nodes, members of cliques, and more [HGER+12].

There are different types of equivalence, each of which is based on an equivalence relation that defines a partition of a node-set to mutually exclusive and exhaustive equivalence classes such that the nodes that are equivalent are assigned to the same class. Among the various types of equivalence, we focus on three main types: structural, automorphic, and regular equivalence.

Structural equivalence [LW71] is the simplest and most restrictive notion of equiva-

lence:

Definition 7.1. *Two nodes are* structurally equivalent *iff they have identical connections with identical nodes.*

For example, in Figure 7.1 nodes 0 and 1 are structurally equivalent. Structural equivalence is rarely seen in real-world networks, and it is very strict form of structural similarity that is closely related to proximity: *two structurally equivalent nodes are at most two hops away from each other* [WF94, RA15]. We confirm empirically that proximity-preserving embedding methods best capture this in Section 7.6.

Automorphic equivalence [BE92] was proposed to relax the notion of structural equivalence. Intuitively, two automorphically equivalent nodes are identical with respect to *all* graph theoretic properties (e.g., in-/out-degree, centralities) and may differ only in terms of their labels. Examples include the nodes in each node-set {0, 1}, {2, 4}, and {5, 6, 8, 9} of Figure 7.1. More formally:

Definition 7.2. *Two nodes are* automorphically equivalent *iff there is an automorphism (i.e., an isomorphism in the* same *graph) that maps one node to the other.*

Although automorphic equivalence is less restricted than structural equivalence (and also a superset of structural equivalence), its *exact* format is still expected to be rare in real networks.

Regular equivalence [BE92] is among the most interesting and prevalent types of equivalence in real networks:

Definition 7.3. *Two nodes are* regularly equivalent *if they relate in the same way to* equivalent *nodes.*

This definition is more meaningful in multi-relational networks (e.g., heterogeneous graphs), but it also applies to networks with a single relation. For example, similarly colored nodes in Figure 7.1 correspond to regularly equivalent classes—e.g., nodes {2,3,4} are regularly equivalent because they connect to nodes of the 'red' and 'purple' roles, although they do not have the same degree (and, thus, it is more relaxed notion than automorphic equivalence).

7.3.2 Network Statistics

7.3.3 Tasks

Node embeddings may be used for a variety of downstream tasks. To evaluate the utility of various methods, we compare them based on several families of tasks which we discuss here.

Single-Network Tasks. Structural node embeddings are often used to predict the labels of nodes, when these are thought to correspond to a node's structural role in a network [RSF17]. The problem of **node classification** can be modeled as a supervised machine learning problem that can be solved with any off-the-shelf downstream machine learning classifier [PVGea11] applied to the embeddings of the nodes. A related unsupervised task is **node clustering** [DZHL18], which again can also be solved with standard machine learning methods applied to the features of the nodes obtained via embeddings.

Link prediction seeks to infer whether two unconnnected nodes should share an edge. It is a common task for node embeddings [GL16]; however, the fundamental insight needed for link prediction is the proximity of the nodes (whether or not they should share an edge and be in close proximity). This task is thus more suitable for proximity-preserving node embeddings, and we do not study it further in this work.

Multi-Network Tasks. Structural node embeddings have also been shown to be useful for tasks defined over multiple networks, as structural roles can be compared across networks. A task that exemplifies node comparison across networks is **network alignment**, where the objective is to find correspondences betwen nodes in different networks. REGAL [HSSK18] was shown to yield strong results on this task by computing structural embeddings of nodes in each network and matching nodes simply based on the similarity of their structural roles. The structural embeddings for each node in a network can also be aggregated into a single feature vector for the entire network, which may be used for graph-level tasks like **graph classification**. RGM [HSK19] is a method for constructing graph feature maps from node embeddings that was shown to yield competitive results on graph classification compared to leading graph kernels and graph neural networks, with significantly faster runtime. Thus, given a structural node embedding method, we can use it for graph alignment or classification

by directly substituting it into the REGAL or RGM frameworks, respectively.

7.3.4 Research Goals

In this work, our goal is to help the research community understand and evaluate structural node embeddings. We contribute to the understanding and evaluation of existing structural node embedding methods, but also with our analysis pave the way for better understanding and evaluation of methods that are subsequently developed.

Understanding. Fundamentally, we want to learn what aspects of a "structural role" do node embedding capture. Here, we turn to concepts of role equivalence developed in mathematical sociology (Section 7.3.1), as well as network-scientific statistics (Section 7.3.2), to see how well each embedding method captures these properties.

Evaluation. We propose new methods for intrinsic as well as extrinsic evaluation of structural node embeddings. Intrinsic evaluation directly evaluates the geometry of the node embedding space, independent of any downstream task or method (e.g. a machine learning classifier). The goal is to see how similarities between nodes in the embedding space correlate to similarities defined by a ground-truth task or by the sociological and network scientific concepts we introduce. Extrinsic evaluation, on the other hand, analyzes the performance of a downstream task using the node embeddings. We cast our objectives for understanding as an extrinsic evaluation, by using machine learning to predict role equivalences or network statistics from the node embeddings. We also consider the single- and multi-network tasks discussed in Section 7.3.3.

To study structural embedding methods meaningfully, we need datasets that highlight what they are able to capture. Toward this end, we collect real datasets on which the data mining task of interest (e.g. node labels for classification) relates to the structural roels of each node. We also design an extensive collection of synthetic datasets, going beyond the simpler constructions of previous works [RSF17, DZHL18] specifically to illustrate clear role equivalences (Section 7.3.1).

While we empirically study a large majority of existing structural embedding methods, the purpose of our evaluation is not primarily to choose a "winner" from existing structural embedding methods. We see that various methods have their own strengths and weaknesses,

and indeed our contributions are forward-looking with the goal of positively influencing the development of future structural embedding methods. To be most constructive for future developments in structural embedding methods,

- Identify factors unrelated to the node embeddings that may influence the ranking embeddings. In Section 7.7, we show that on the same graph, different structural embedding methods may appear to be better or worse due to a variety of factors, like the performance metric, distance metric used to compare node embeddings, downstream machine learning model, or definition of node labels. Future works should be mindful of these to avoid reporting apparent gains that are due to some factor other than the quality of the embeddings.

- Highlight successful (and unsuccessful) design choices for different tasks. We design a simple set of baselines, local degree histograms, that are based on design choices that appear to perform well in many of the tasks we consider. Future works can not only compare against these baselines, but also use the ideas they incorporate to develop more effective methodology.

7.3.5 Selection of Structural Embedding Methods

In this work, we propose techniques for evaluating structural node embeddings in particular. We demonstrate these by analyzing a large number of existing node embeddings methods, predominantly structural embeddings but with a few proximity-preserving node embedding methods as well for contrast. In contrast to most of the recent works on graph neural networks [KW17b], all node embedding methods that we consider in this work are unsupervised, as we propose intrinsic evaluation that is not dependent on a downstream task.

• **Proximity methods.** In our analysis, we consider two embedding methods that are primarily proximity-based. **(1) node2vec** [GL16] uses the skip-gram architecture [MSC+13] to learn an embedding for each node that preserves its similarity to other nodes in its context, sampled with biased random walks. **(2) LINE** [TQW+15] optimizes an embedding objective that maximizes the probability of the first and second-order proximities in the network

(direct edges between any two nodes and mutual neighbors that any two nodes share, resp.). Proximity methods are the topic of numerous surveys [GF18, RJK+20], and we refer the interested reader to those.

• **Structural methods.** We also evaluate eight structural embedding approaches: **(3) struc2vec** [RSF17] uses the same skip-gram architecture, but samples context with random walks performed over an auxiliary multilayer graph capturing structural similarity (mainly *degree*) of nodes' neighborhoods at several hop distances. **(4) GraphWave** [DZHL18] computes the heat kernel matrix for a graph and embeds each node by sampling the empirical characteristic function of the distribution of heat it sends to other nodes. **(5) xNetMF** [HSSK18] draws on the connection between the skip-gram architecture matrix factorization [LG14] to find node embeddings that implicitly factorize a structural similarity matrix, defined by comparing the distribution of node degrees in k-hop neighborhoods. Subsequently, **(6) SEGK** [NV19] factorizes a structural similarity matrix using graph kernels to compare the nodes' k-hop neighborhoods. **(7) role2vec** [ARL+19] applies the skip-gram model to a corpus sampled using *attributed* random walks which record the structural type of each node. The method learns the same embedding for nodes of each structural type, which enhances space efficiency. **(8) RiWalk** [XQQ+19] also uses the skip-gram model, but learns an embedding for each node based on the structural types of nodes in its context. **(9) DRNE** [TCW+18] contends that feature propagation is similar to the recursive definition of regular equivalence, and uses an LSTM to learn node embeddings by aggregating the features of their neighbors sorted sequentially by degree. **(10) MultiLENS** [JRK+19], similar to xNetMF, derives embeddings based on matrix factorization that captures the distribution of structural features in nodes' local neighborhoods. While we focus on unsupervised methods in this paper and thus do not consider common graph neural network models [KW17b, HYL17], it has been noted that MultiLENS performs local feature aggregation akin to that of a graph neural network [JRK+19].

In addition to these ten 'hybrid' and structural methods, we also construct variants of **degree distributions** over different neighborhoods, which can be seen as simple, yet strong, baselines for embedding nodes. We represent each node with the degree distribution of its k-hop neighbors—i.e, a histogram of dimension d_{max}, the maximum node degree in each

dataset, in which the i-th entry counts the number of neighbors that are k hops away with degree i. We refer to the 11^{th} family of structural approaches that we consider as `degree` that is simply the node's degree, and `degree1` and `degree2` that are histograms based on 1- and 2-hop neighborhoods.

Embedding Implementations and Hyperparameters. Unless otherwise mentioned, our parameter settings for all methods follow default values suggested in the papers and/or official/available author implementations. For convenience, below we cite the links to exact versions of the code and data we used for our experiments. In order to make the comparison between the embedding methods fair, we transform all the input networks to be *undirected* and *unweighted*. For all methods, we learn 128-dimensional embeddings by default following common practice.

- For node2vec [GL16], we bias the random walks with parameters $p = 1$ and $q = 4$ to tune the walks to capture more structural equivalence using parameter values considered in the original paper [GL16].

- For the skip-gram methods (node2vec, struc2vec [str], RiWalk [riw], and role2vec [rol]), we sample context by performing 10 random walks (80 for struc2vec, which performs these walks on a more complex multi-layer structural similarity network) of length 80 per node. We set the skip-gram window size to 10 and optimize the objective using 10 iterations of gradient descent. Taking care of scalability, we use all three optimizations for struc2vec and degree (or motifs, if applicable) as the feature for role2vec.

- For LINE [LIN], we set the `order` to be 2 and the total number of training samples to be 100 million and negative samples to be 5.

- For GraphWave [gra], we use the automatic selection method of the scale parameter [DZHL18] and `exact` heat kernel matrix calculation.

- For struc2vec, xNetMF [xne], and SEGK [seg], we consider up to 2-hop neighborhoods. In RiWalk, which also has a node neighborhood radius parameter k, we used default setting $k = 4$. We discount the information of distant neighborhoods in xNetMF using a discount factor of 0.1 and set the structural similarity resolution parameter $\gamma = 1$. For SEGK, we

compare neighborhoods using the Weisfeiler-Lehman graph kernel [SSL+11]. We also use the Weisfeiler-Lehman graph kernel in RiWalk to identify structural roles of nodes based on their local neighborhoods (RiWalk-WL in [XQQ+19]).

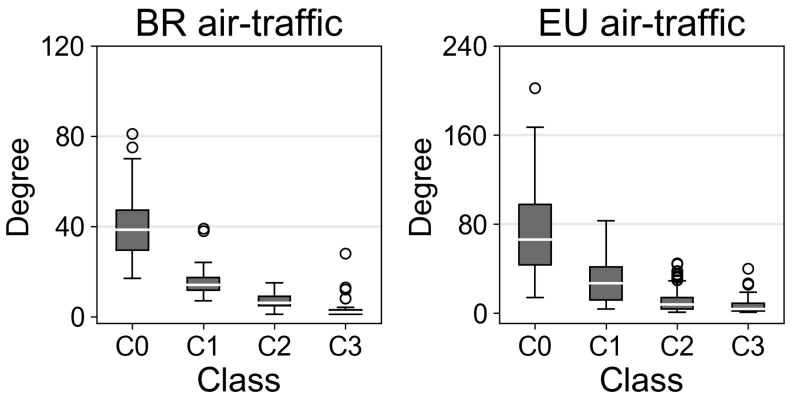
- For DRNE [DRN], we follow the example usage to set the batch size to be 256 and the learning rate to be 0.0025.
- For MultiLENS [mul], we set the `cat` input with all nodes having the same category/type.

7.4 Data and Ground Truth Roles

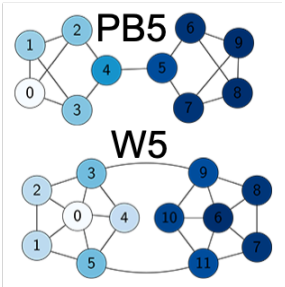
To gain insights into the type of information that is encoded in structural embeddings, we consider several real datasets (Table 7.1), and introduce synthetic data (Figure 7.3, Tab. 7.4), the structure of which we can control and understand better than that of real networks.

Table 7.1: Real Datasets: Single-Network Tasks

Dataset	# Nodes	# Edges	Labels
BlogCatalog [GL16]	10,312	333,983	centralities
Facebook [GL16]	4,039	88,234	equivalences (Section **??**)
ICEWS [BLO+18]	1,255	1,414	military vs media entities
Email-300	318	752	professional roles
Email-2K	2,414	11,995	professional roles
PPI [HYL17]	56,944	818,786	protein cellular functions
BR air-traffic [RSF17]	131	1,038	# landings & take-off, equival. (Section **??**)
EU air-traffic [RSF17]	399	5,995	# landings & take-off, equival. (Section **??**)
US air-traffic [RSF17]	1,190	13,599	# passengers, equivalences (Section **??**)
DD6 [BK05]	4,152	20,640	amino acid properties



(a) Strong connection between the node degree and the class labels in the BR and EU air-traffic data.



(b) Vertex similarity [LHN06] is related to proximity. Lighter color represents higher similarity to node 0 (white node).

Figure 7.2: Limitations of some node labeling methods.

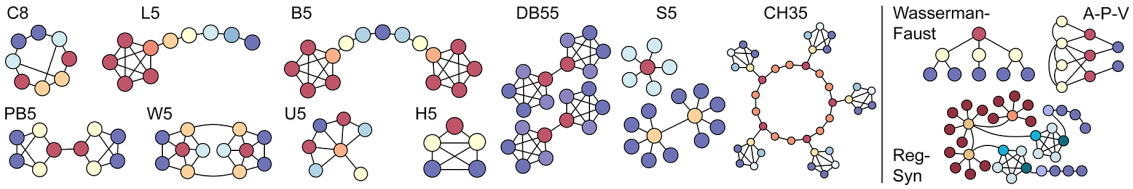


Figure 7.3: Per synthetic base graph, nodes with the same color are automorphically equivalent on the left & regularly equivalent on the right.

7.4.1 Real Network Data: Single-Network Tasks

Limitations of existing datasets. The most commonly used real datasets for evaluating the quality of structural embeddings are **air-traffic networks** from [RSF17], which capture the existence of commercial flights (edges) between airports (nodes) and are thus undirected and unweighted [RSF17]. Their node labels are defined based on either the number of landings and take-offs, or the number of passengers passed by each airport in a given time period: four labels are obtained by splitting the data into quartiles. Although the balanced classes simplify the evaluation, this arbitrary labeling has two drawbacks: (1) it is not clear that splitting the data into four quartiles reflects a real-world phenomenon; and (2) to a large extent, the labels simply capture degree information (Figure 7.2a).

To experiment with the effect of different node labelings to the performance, we also construct an alternative set of node labels constructed by splitting the airport-related statistics (number of landings and take-offs, or passengers) into logarithmic bins (Figure 7.9b). This results in imbalanced classes but produces a distribution of "roles" following the well-known power-law distribution.

More recent work [TCW+18] also used a Jazz collaboration network and BlogCatalog, creating labels using the *vertex similarity* measure [LHN06] as ground truth for regular equivalence. However, as we show in Figure 7.2b, the vertex similarity captures *distances* between nodes rather than similarity in their structural properties, and thus is not an appropriate measure for regular equivalence.

New datasets for structural embeddings. Besides the existing datasets used in prior works on structural embeddings, we also consider large real-world datasets (Tab. 7.1), where we can define the node labels based on the different definitions of equivalence (Section 7.3.1,7.6). We use the **BlogCatalog** and **Facebook** networks from [GL16], which are both social network datasets containing various structural roles.

Real-world data mining tasks are often defined in terms of external node labels, so to this end we propose the use of additional datasets where this information may be better predicted by structural rather than proximity-preserving node features. The first is a knowledge graph of the relationships among socio-political actors from the Integrated Crisis Early Warning System (**ICEWS**) [BLO+18]; it is constructed from events on October 4, 2018 that are automatically extracted from news articles. We group the entity types into broad categories, and our task is to distinguish between "media" entities and "military" entities. We expect that these will have distinct structural roles from each other. Another real dataset we use is the **PPI network** from [HYL17], a multi-network dataset which is claimed to have node labels corresponding to structural roles rather than communities. Finally, we use a network called **DD6**, one of the larger networks from the D&D dataset commonly used to benchmark graph classification [BK05]. This dataset is a protein structure and its nodes, which represent amino acids, have labels representing various properties of the amino acid [BK05]. These labels exhibit very low homophily and are known to be challenging for proximity-based node representation learning methods [LRK+19]. We also use two proprietary email communication networks, **Email-300** and **Email-2K**, for the users in which we have professional roles (e.g., CEO, manager) that are known to be related to regular equivalence [WF94].

Ground-truth Node Equivalences or Roles. For our intrinsic evaluation, instead of arbitrarily defining roles in networks, we leverage existing (exact or approximate) algorithms that aim to identify equivalence classes. All sociological notions of equivalence are computed using the implementations of the CONCOR, MAXSIM, and CATREGE algorithms in the popular UCINET package [BEF02]. The default settings in UCINET are adopted.

Given the adjacency matrix $\mathbf{A}$ of a graph, these approaches produce a pairwise node similarity matrix $\mathbf{S}$ based on their respective equivalence definitions. For *structural equivalence*, CONCOR [BBA75] creates a similarity matrix with entries $s_{ij} = s_{ji}$ corresponding to the Pearson correlation between nodes i and j (i.e., the correlation of their respective rows, $\mathbf{A}_{i,:}$ and $\mathbf{A}_{j,:}$). For *automorphic equivalence*, MAXSIM [EB88] first creates a matrix of geodesic proximities from the adjacency matrix $\mathbf{A}$, and then creates $\mathbf{S}$ by comparing the node distributions of geodesic proximities pairwise. For *regular equivalence*, CATREGE [BE93] searches for matches in successive node neighborhoods, and encodes in $\mathbf{S}$ the iteration in

Table 7.2: Graph classification datasets [KKM+16]. We give the total number of nodes/edges across all graphs per dataset.

Name	Nodes	Edges	Graphs	Classes	Node labels	Domain
PTC-MR	4 916	5 053	344	2	Y	bioinformatics
IMDB-M	19 502	98 910	1 500	3	N	collaboration
NCI1	122 765	132 753	4 110	2	Y	bioinformatics

which two nodes were separated into different groups or classes.

CONCOR also produces a partition that we use as the *ground-truth* equivalence classes (i.e., groups of nodes with similar roles). To obtain the ground truth for MAXSIM and CATREGE, we apply hierarchical clustering on $\mathbf{S}$ (with default settings).

7.4.2 Real Network Data: Multi-Network Tasks

While structural node embeddings are often used for single-network tasks such as node classification and clustering, recent works have used them for multi-network tasks such as network alignment [HSSK18] and classification [HSK19]. In Section7.8, we comprehensively evaluate a large number of structural embedding methods within the embedding-based frameworks proposed to solve these downstream tasks. Here we describe the standard benchmark datasets we use for each tasks.

For graph classification, we use three well-known and publicly available [KKM+16] graph classification benchmark datasets, PTC-MR, IMDB-M, and NCI1 . These correspond to small, medium, and large graph classification datasets as used in recent work [HSK19]. IMDB-M is a social network dataset where the graphs represent actor collaboration networks, and in other two the networks represent small molecules. The molecular datasets also have node labels, which to fairly compare all embedding methods we do not use in the embeddings, but which can be used by a downstream graph classification method. We give detailed descriptions of the datasets in Table 7.2.

For network alignment, we use two datasets from [HSSK18], which again represent social and biological phenomena. We describe the process of constructing a network alignment scenario with known ground-truth correspondences between nodes, which is commonly used in the network alignment literature, in Section7.8.

Table 7.3: Graph alignment datasets.

Name	Nodes	Edges	Description
Arenas Email [Kun13]	1 133	5 451	communication network
PPI [BSR+08]	3 890	76 584	PPI network (Human)

7.4.3 Synthetic Network Data

We also evaluate structural embedding techniques on a variety of synthetically-generated networks—beyond just the commonly-used barbell graph—, as shown in Figure 7.3 (left).

We define two sets of roles per node, based on *structural* and *automorphic*—using the methods CONCOR and MAXSIM (Section 7.4.1), respectively. We also enlarge the small synthetic graphs to enable further extrinsic evaluation (Table 7.4). For *regular* equivalence, since nodes should be assigned to different classes according to their roles, we generate the synthetic graphs accordingly (Figure 7.3, right). Similarly, we enlarge the synthetic graphs by adding more nodes with different roles and connecting them following the rules in the base case (Table 7.4). For all the synthetic graphs generated for the regular equivalence evaluation, the edge type is indicated by the pre-defined roles of the end-points (e.g., hub vs. clique node). The output of CATREGE (Section 7.4.1) generates the *same* role assignment as the pre-defined roles.

7.5 Embeddings and Structural Properties

Many of the existing structural embedding methods (Section 7.3.5) leverage node degree information in various ways. While it is expected that embeddings are *related* to the node degrees, it is not well-understood to *which extent* they capture the degree or other structural information (e.g., centralities). In this section, we seek to gain insights into this via correlation and predictive analysis. While such an analysis will not completely characterize the information captured in structural embeddings, it can help us understand which ones are comparatively *interpretable* in the sense that they encode common network metrics used to characterize a node's structural role.

Methodology. First, to see if similarly embedded nodes have similar structural properties, we perform the following analysis: **(1)** For each node v in graph G, we calculate a property

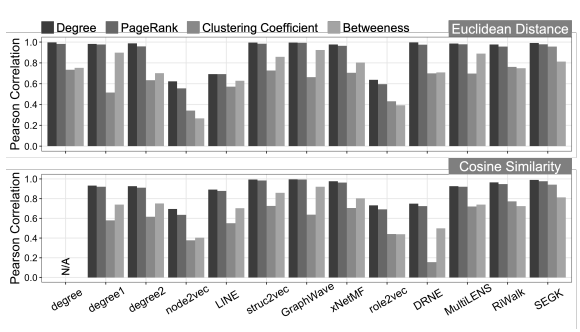
Table 7.4: Enlarged synthetic graphs

Large Graph	Base	Generation
H10_S_L	**H5**	10 H5 on a circle with 2 circular nodes between each connecting circular node with house's side.
H10_T_L	**H5**	10 H5 on a circle with 2 circular nodes between each connecting circular node with house's roof.
Barbell L-A	**B5**	Connecting the out-most nodes on the chain of B5 into a circle.
Barbell L-B	**B5**	Connecting the out-most nodes on the chain of B5 into a circle. Additional 5-clique at each connector.
Ferris Wheel	**C8**	Enlarged version of C8 with similar perturbation.
City of Stars	**S5**	10 normal stars and 5 binary stars as in S5
PB-L	**PB5**	10 half-sided PB5 connected to each node of a 10-node circular graph. All the node degrees are 3.
Conference	**A-P-V**	Mimicking the real-world scenario, we simulate 80 papers with 4∼6 collaborators out of the 120 authors, and assign them to one of the 30 venues.
Reg-Syn-L	**Reg-Syn**	Based on the connection rules in Reg-Syn, we connect 9 stars, 7 cliques and 7 chains of different sizes.
Knitting Wheel	**B5**	10 different sized cliques connected onto a circle with three circular nodes apart each connection.

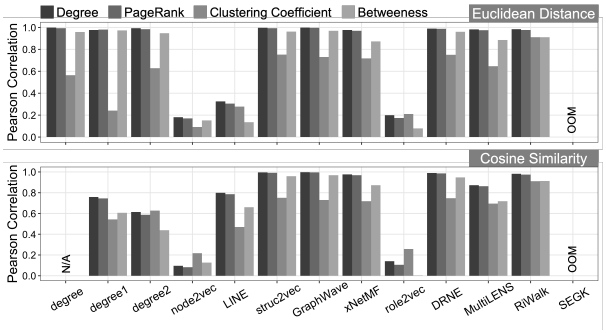
of interest $p_i(v)$. We consider four properties: degree, PageRank (with damping parameter $\alpha = 0.85$ [PBMW99]), clustering coefficient, and betweenness centrality. **(2)** We identify v's k-nearest neighbors (k-NN) in the embedding space $\mathbb{R}^d$ using cosine or Euclidean distance, and compute the average value for each structural property, $\overline{p_{i,kNN}(v)}$. **(3)** Per property p_i, we calculate the Pearson correlation between the structural property of a node and its k-NN across all nodes.

Second, to better understand the extent to which degree is encoded in the structural embeddings, we also perform a predictive task. Given a subset of nodes with their structural embeddings and degrees, we apply k-NN regression and compute the error between the predicted and original degree for the remaining nodes. We report the mean RMSE across 5 folds, using one fold for *training* and four folds for *testing*.

Results. Since this task is based on the intrinsic properties of the embeddings, and not the node labels, theoretically we can use any dataset here. We report results on the prevailing



(a) EU air-traffic: correlation between structural properties of a node and the structural properties of its 5-NN in the embedding space.



(b) BlogCatalog: correlation between structural properties of a node and the structural properties of its 5-NN in the embedding space.

Figure 7.4: Correlation of embeddings with structural properties: Generally, structural methods—except role2vec—do well in preserving the node structural properties in the embedding space $\mathbb{R}^d$. Degree and PageRank are better captured than betweenness and clustering coefficient. As expected, proximity-based embedding methods don't perform well. Differences are observed between Euclidean distance and cosine similarity.

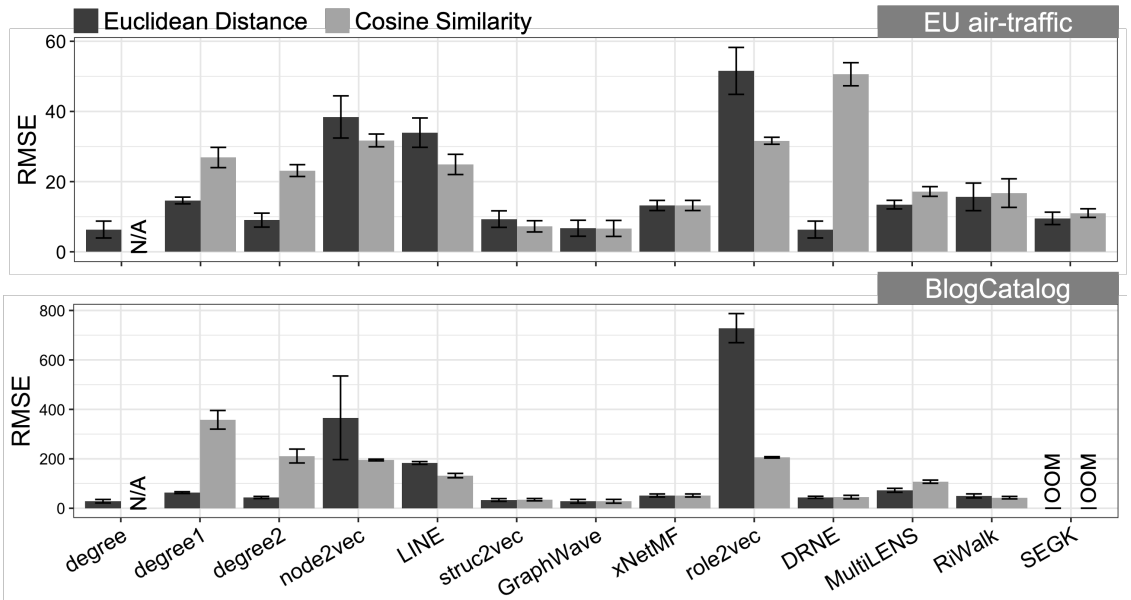


Figure 7.5: RMSE of predicting the node degree from the structural embeddings for two datasets: BlogCatalog (top, max degree=3,992) and EU Air-traffic network (bottom, max degree=202). Error bar shows standard deviation on 5 fold CV with one fold as training and four folds as testing. Performance on the predictive task aligns with the correlation task. Choice of distance metric influences the performance of some methods significantly (e.g., DRNE, role2vec_d).

BlogCatalog and EU air-traffic network datasets. The results are consistent on other data (real and synthetic). The cosine distance is not defined between pure scalars so we leave the result for the degree variant with cosine as N/A (Not Applicable) in all the results.

Based on Figure 7.4, for most structural embedding methods, except role2vec, closely embedded nodes have similar degree/PageRank centralities. These embeddings also contain information about betweenness and clustering coefficient, but less so. On the BlogCatalog

dataset, RiWalk preserves betweenness and clustering coefficient almost as well as degree and pagerank; most other methods have a significant drop in at least one of the two former metrics, as does RiWalk on the EU Air-traffic dataset. Proximity-based embedding methods such as node2vec and LINE do not encode structural properties well.

Observation 7.1. *Current structural node embeddings capture node importance measures such as degree and pagerank well, but discern less clearly the density of connectivity as given by betweenness and clustering coefficient.*

The results for correlation in Figure 7.4a snd 7.4b differ for Euclidean distance and cosine similarity, especially for proximity methods. A further discussion on the usage of similarity measurement is in Section 7.9.

Similar patterns can be observed from the RMSE in the predictive task (Figure 7.5) with 5-NN regression. The maximum node degree for BlogCatalog is 3,992 and EU air-traffic network is 202. With only 20% of the node's degrees as training, struc2vec, GraphWave, xNetMF and MultiLENS can perform well on the predictive task.

7.6 Embeddings and Equivalences

In the literature, there are various claims about the types of equivalence that embedding methods capture, some of which are imprecise. We investigate this by designing experiments for both intrinsic and extrinsic evaluation. Our **intrinsic evaluation** aims to evaluate the quality of embeddings in the context of different types of equivalences *directly*, decoupled from a downstream task. Here, *ground-truth labels* are defined by the equivalence methods (Section 7.3.1, 7.4.1). Our **extrinsic evaluation** relies on classification and clustering, both of which are typically used to evaluate embeddings.

7.6.1 Intrinsic Evaluation

The intrinsic evaluation of structural embeddings seeks to characterize the agreement between the similarities of nodes defined by the different types of equivalence and the node similarities in the embedding space $\mathbb{R}^d$.

Methodology. Given a similarity matrix **S** based on a notion of role equivalence, for each node we calculate the Kendall rank correlation coefficient between its embedding similarity (based on Euclidean distance or cosine similarity[4]) and its structural similarity to all other nodes given by **S**.

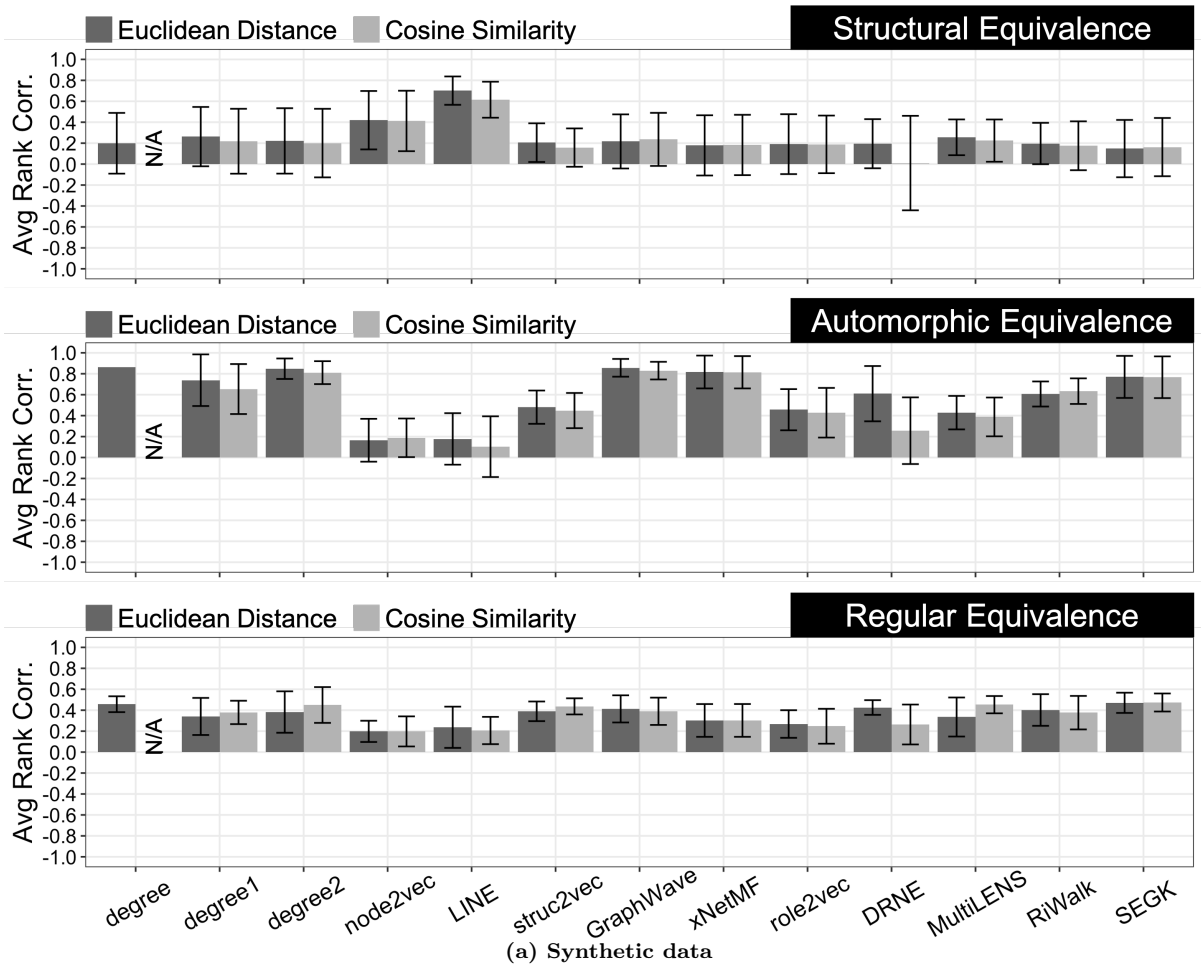
For structural and automorphic equivalence, we perform analysis on a total of 16 synthetic networks (Figure 7.3 left plus the enlarged datasets in the top section of Table 7.4, CH35 excluded as near-duplication of Small Town-S) and 4 real networks (three air-traffic networks + Facebook). One exception is that for structural equivalence, CONCOR encounters an error for City of Stars, for which we skipped evaluation. For regular equivalence, we perform analysis on a total of 5 synthetic datasets (Figure 7.3 right plus the enlarged datsets in the bottom section of Table 7.4, A-P-V excluded as duplication of Conference). None of our real networks can be used with CATREGE to compute regular equivalence for an intrinsic evaluation, as the algorithm requires relationship types and the implementation handles up to 255 nodes. For each type of equivalence, we report the average and the standard deviation of the Kendall rank correlation coefficient across different subsets of our datasets.

Results. Figure 7.6 gives a summarized view of our intrinsic evaluation. It shows, per embedding method, the rank correlation and its standard deviation averaged over all the corresponding datasets. LINE and node2vec rank top in our intrinsic evaluation for **structural equivalence**. This is expected, as despite its name, structural equivalence is actually by definition best captured by proximity-based embedding methods [WF94]. It is defined between two nodes in terms of how many neighbors they share: two nodes are structurally equivalent if they are connected to the exact same nodes. Structural equivalence as defined in mathematical sociology is distinct from the structural similarity that role-based node embeddings try to capture.

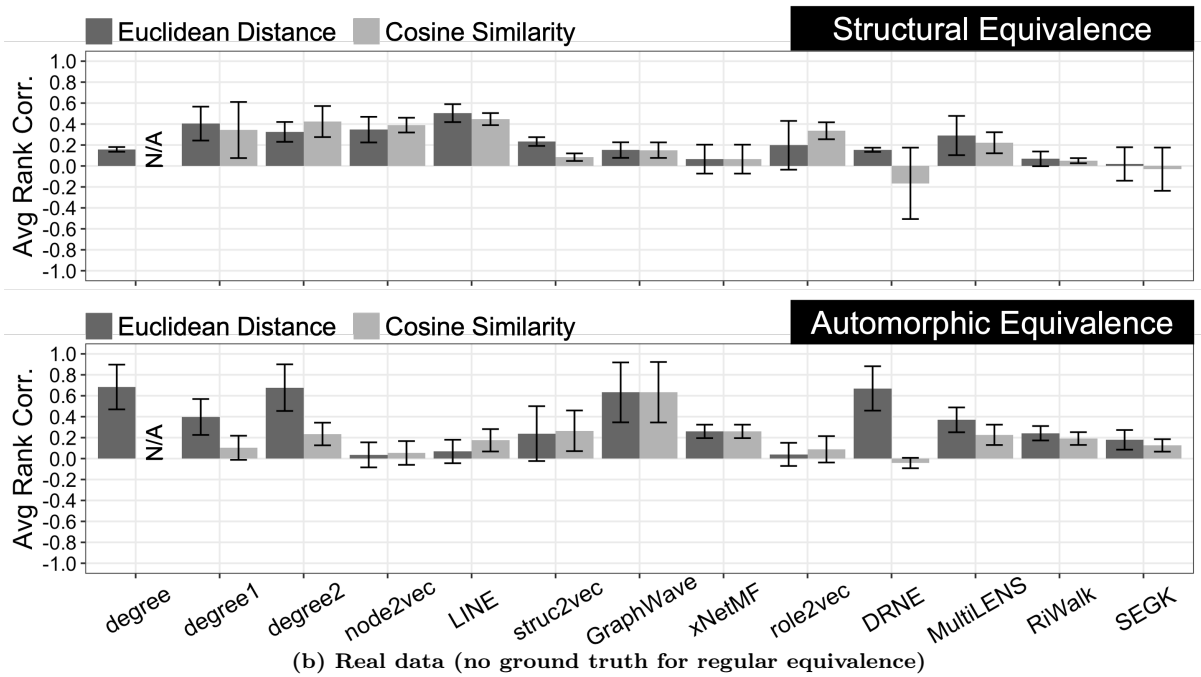
Observation 7.2. *Structural equivalence depends on node proximity and in fact cannot be captured well by structural embeddings, but automorphic equivalence does not depend on node proximity and may be better captured by structural embeddings than by proximity-preserving embeddings.*

On the other hand, structural embedding methods such as GraphWave, xNetMF and

[4]Cosine similarity is not defined for a scalar (e.g., `degree`), in which case we list "N/A" in Figs. 7.6-7.7.

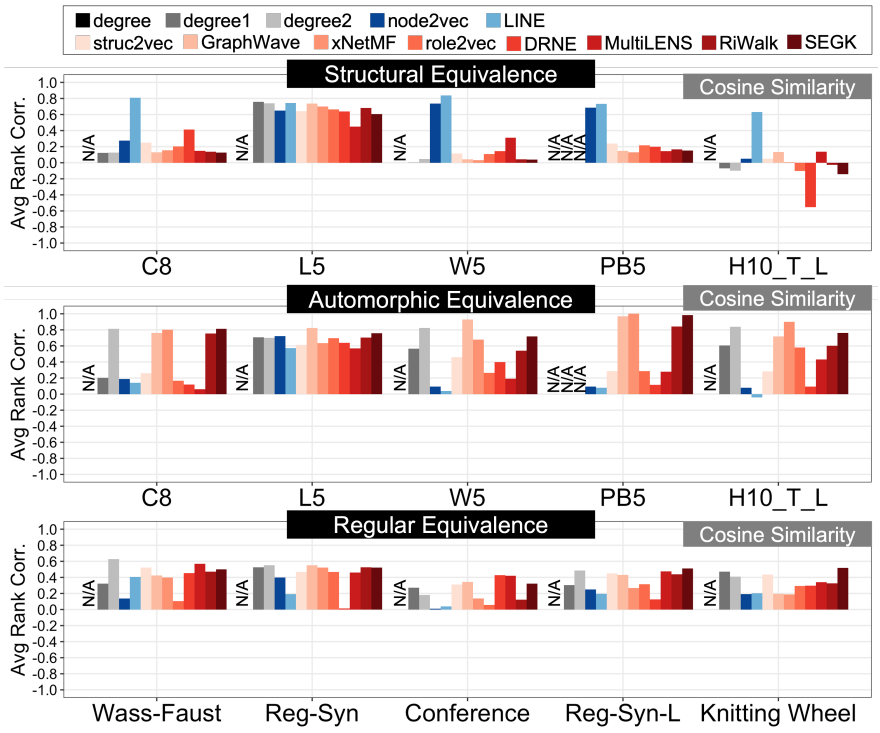


(a) Synthetic data

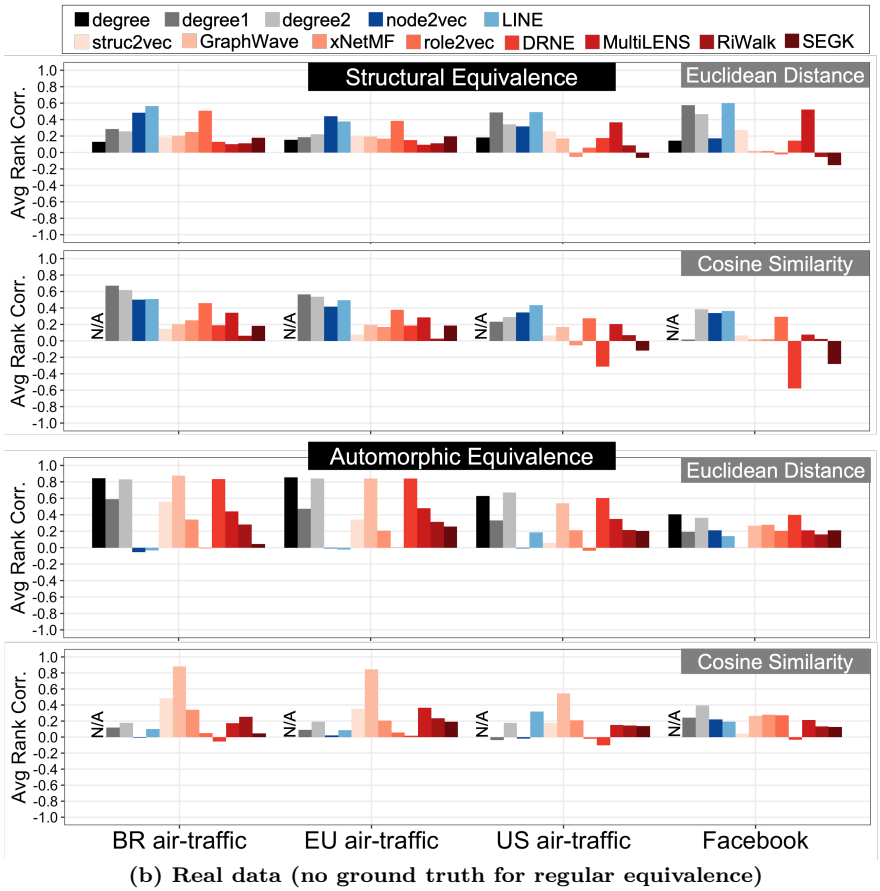


(b) Real data (no ground truth for regular equivalence)

Figure 7.6: Summarized view of intrinsic evaluation: Average correlation (and stdev) between node embeddings and different types of equivalences across all synthetic data (top) and all real data (bottom). Structural embeddings tend to capture automorphic and regular equivalence, while primarily proximity embeddings capture structural equivalence. The choice of distance affects the results.



(a) Synthetic data (only cosine similarity shown for brevity)



(b) Real data (no ground truth for regular equivalence)

Figure 7.7: [Best viewed in color] Detailed view of intrinsic evaluation: correlation with different types of equivalence for specific synthetic (top) and real (bottom) datasets. Performance of embedding methods varies across different datasets and distance choices.

SEGK, as well as `degree2`, work well in terms of **automorphic equivalence**, while the proximity-based methods, like LINE and node2vec do not. This is also expected, as automorphically similar nodes need *not* be in close proximity in the graph. We conjecture that the difference of role2vec on the synthetic datasets and real world datasets might result from the difference in degree distribution and network structure between the synthetic and real datasets.

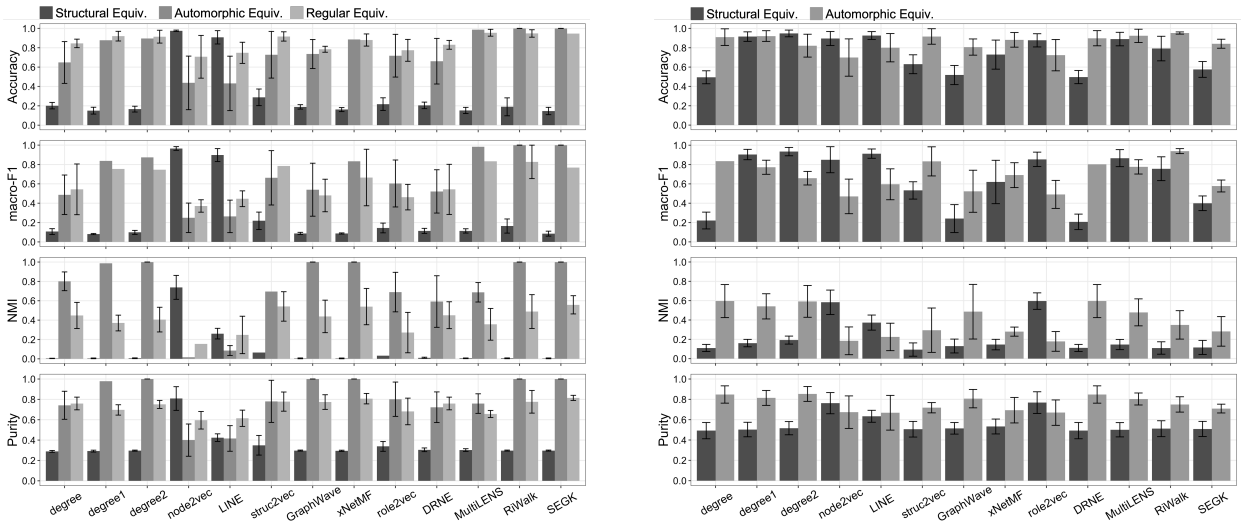
Similarly, the proximity-based node2vec and LINE struggle to capture **regular equivalence**, which among structural embedding methods is generally best captured by `degree`, DRNE, and GraphWave based on Euclidean distance, and `degree2`, MultiLENS, and struc2vec based on cosine similarity. The strong performance of degree distribution features in the intrinsic evaluation using automorphic and regular equivalence is noteworthy.

Observation 7.3. *Node degree, generalized to include the distribution in its k-hop neighborhood, may indeed be a good indicator of the structural position or role of the node in the network.*

In Figure 7.7, we look deeper into these results on a per-dataset basis. While trends are largely similar, some datasets are worth noting individually. For example, we see that the base "L5" has a distinctive "lollipop" shape, where equivalent nodes (in the head) and comparatively near-equivalent nodes (in the stem) are also in close proximity. As a result, proximity-preserving and structural embeddings do comparably well at capturing both structural and automorphic equivalence. We see larger gaps on the remaining synthetic datasets. On real datasets, GraphWave and DRNE capture extremely high automorphic equivalence on the air-traffic datasets, but the difference between them and the other methods disappears on Facebook, a social network dataset.

Observation 7.4. *None of the structural embedding methods are optimized to capture sociological concepts of role equivalence.*

Although we find that structural embedding methods do capture sociological role equivalence to some extent incidentally, it depends on how well the equivalences correspond in any given dataset with the types of similarities each embedding is optimized to preserve (the choice of distance, Euclidean or cosine, has significant impact for some methods, especially in the real data.)



(a) Synthetic data **(b) Real data**

Figure 7.8: Extrinsic evaluation on downstream tasks. Mean and standard deviation is presented for each method on all corresponding synthetic datasets and real datasets for three types of equivalence. Generally, the extrinsic evaluation aligns with the intrinsic evaluation.

7.6.2 Extrinsic Evaluation

We also evaluate the structural embeddings extrinsically by defining *equivalence-specific* node labels.

Methodology. As described in Section 7.6.1, we consider the equivalence-specific similarity matrix $\mathbf{S}$ and the node embeddings $\mathbf{Y}$. To obtain the ground-truth *equivalence classes* (i.e., node labels), we perform hierarchical clustering on $\mathbf{S}$ for MAXSIM and CATREGE, and use the CONCOR partitioning output directly. Again, for the synthetic datasets used for automorphic equivalence evaluation, we manually define the *exact* automorphically equivalent classes (instead of using MAXSIM, which is an approximation). With the classes generated or pre-defined, we perform classification and clustering for extrinsic evaluation.

In Figure 7.8 we show the results for all three types of equivalence on synthetic (left) and real (right) data. For structural and automorphic equivalence evaluation, we use the enlarged synthetic graphs described in the top section of Table 7.4. Again, we exclude City of Stars for structural equivalence evaluation for the same reason explained in Section 7.6.1. For the real data evaluation, we use the three air-traffic networks and Facebook. For regular equivalence, we use the enlarged synthetic graphs described in the bottom section of Table 7.4. No real world dataset is appropriate for regular equivalence evaluation as discussed before.

Results. We generally see similar trends to the intrinsic evaluation. For example, proximity-

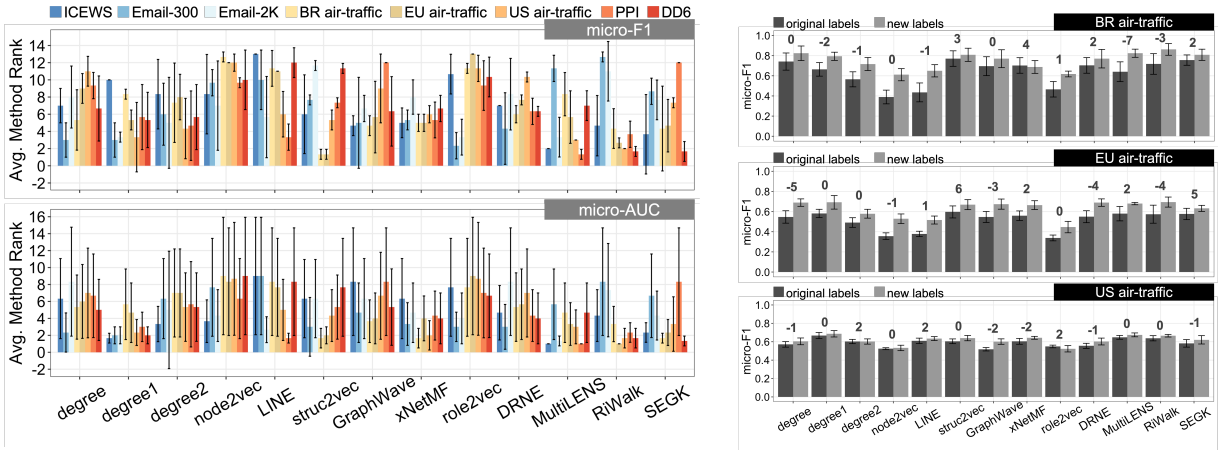
based methods node2vec and LINE are generally best at capturing structural equivalence in both real and synthetic datasets, in supervised and unsupervised downstream tasks. They take a backseat to most other methods, however, at predicting automorphic or regular equivalences. We observe, however, that MultiLENS improves considerably in downstream tasks.

Differences between methods are often more pronounced in synthetic datasets, which are designed to exhibit highly distinctive structural roles. For instance, LINE and node2vec are over 4× more accurate at predicting structural equivalence than structural embeddings GraphWave and xNetMF, a gap that remains but shrinks considerably in the real datasets. Similarly, in synthetic datasets, GraphWave and xNetMF achieve near-perfect clustering scores, as do degree distribution features from 1-hop and 2-hop neighborhoods (which perform competitively with other structural embedding methods at capturing equivalences across our extrinsic evaluations).

Observation 7.5. *The clear structural roles of our synthetic datasets are a good way to expose differences between structural embedding methods.*

In general, we observe similar results between intrinsic and extrinsic evaluation as well as synthetic versus real networks. This suggests that intrinsic evaluation of structural embeddings can *often* be a good proxy of its ability to perform in a downstream task, without adding the additional variable of the downstream machine learning algorithm. Similarly, synthetic networks that can be manufactured to exhibit distinctive structural roles that are known *a priori* are a good controlled experimental environment for structural node embedding. However, researchers should be mindful that there may be exceptions to these trends: MultiLENS is one in both cases, performing far better in extrinsic evaluation and on real data. The word embedding literature has noted that intrinsic evaluations of embeddings may not always accurately predict performance in downstream tasks [CKP16]. Thus, both forms of analysis are worthwhile to perform.

Observation 7.6. *Intrinsic evaluation and/or synthetic datasets are often a good approximation of a method's performance on graph mining tasks, but are not a complete substitute for extrinsic evaluation on real datasets.*



(a) Effect of the classifier across the real data: large standard deviations in the embedding rankings over different classifiers show that they may dramatically affect relative performance.

(b) Different labeling schemes: Numbers represent decrease in ranking under new labeling. Most embeddings' rankings change.

Figure 7.9: The performance of different embedding methods in downstream classification tasks heavily depends on the choice of the classifier and the definition of the ground-truth labels.

7.7 Mining with Structural Embedding

We now compare methods for structural node embedding on real-world networks and task-specific settings on graph mining tasks with *externally given node labels* (unlike Section 7.6.2 that relied on equivalence-defined labels). Specifically, we consider the task of node classification, which can be formulated as a well-studied supervised machine learning problem. Before presenting comparative results, we identify two important real-world observations that can confound the fair evaluation of structural embeddings on real datasets. We thus perform analysis of how methods' performance varies as a function of these factors.

7.7.1 Basic Experimental Configuration

Data. We use all the real datasets in Table 7.1 except for the BlogCatalog and Facebook datasets, which do not have node labels that reflect structural roles of nodes and as the basis for extrinsic evaluation are usually reserved for proximity-preserving node embeddings. The remaining datasets all come with node labels, which we use various machine learning classifiers to predict given the features derived from node embedding.

Classifiers. Our classifiers are all popular machine learning models and have been used to evaluate node embeddings on downstream tasks. Along with their hyperparameter settings, they are:

- Logistic regression and linear SVM: these are two commonly used linear models. We set the parameter $C = 1.0$, and use a one-vs-rest strategy for multiclass classification. The other parameters are set as default from the scikit-learn packages [PVGea11].

- k-nearest neighbors (k-NN): This classifier arguably provides the purest measurement of the geometry of the embedding space, as no additional learning is provided. We use $k = 5$ and Euclidean distance for distance measurement.

We introduce any additional protocols specific to a particular experiment as it becomes relevant.

7.7.2 The Effect of the Classifier

Since the structural embedding methods we consider are unsupervised, they are not optimized for performance on a particular downstream task. Furthermore, we can use any of several different common machine learning metrics to measure task-specific performance. We now study how these downstream variables affect the assessment of the "upstream" embedding methods that are our primary interest.

Methodology. Importantly, we note that the downstream machine learning models used to classify node labels from embeddings can have a significant effect on the results. We illustrate this point through the use of several classifiers: logistic regression, k-nearest neighbors, and a linear SVM. In Figure 7.9a, we report results on all datasets where we average the relative ranks of all of our methods across all classifiers (based on different metrics). We use two different metrics: (micro)-AUC and F1 score.

Results. We see that there is a considerable standard deviation in the rankings, indicating that with simply using a different downstream machine learning model atop the same embeddings can change the evaluation of which embedding method is "better." We also observe a difference between the two metrics, indicating that different embedding methods may be better or worse depending on the evaluation metric used. With many different classifiers and metrics being used in the literature, it is important to keep in mind that these too are variables that may affect the performance.

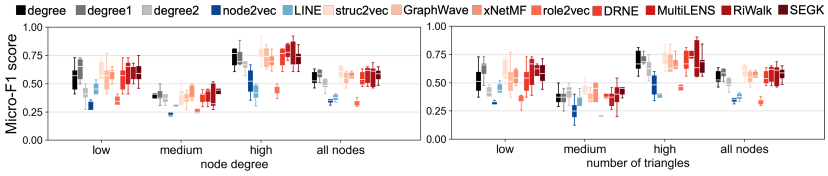


Figure 7.10: [Best viewed in color] Performance by node degree and participating triangles on the original label on EU air-traffic: nodes with more "extreme" degrees are more accurately classified. Box plot based on 5-fold CV results.

7.7.3 The Effect of Label Definitions.

Methodology. In Figure 7.9b, we show the results of different embedding methods on the air-traffic datasets for two different labeling schemes: the original ones resulting in balanced classes, and our relabeling in Section 7.4.1. For brevity, we report Micro-F1 scores obtained using logistic regression, and annotate the decrease in ranking under the new labeling, per method.

Results. We see noticeable differences in performance under the two different labeling methods; In several cases, this can change the comparative ranking of the different methods. For example, MultiLENS and RiWalk are in the middle of the pack under the old labels but the best methods at predicting the new labels.

Recent works have observed that node classification involves a labeling process that may be uncorrelated with the graph itself, which may complicate evaluation [EP19]. In these airport datasets, where the labels were arbitrarily discretized, this issue is even more pronounced. The fact that two (reasonable) ways of generating node labels can yield different results among structural embedding methods suggest that each structural embedding method best captures certain structural roles in the network, and it then becomes an empirical question how well these roles are correlated with the labels. (Note that the airport labels are not connected to any particular roles.) This is the reason why we have performed our previous analysis dissecting the structural role information that each embedding method best captures.

Observation 7.7. *Many factors unconnected to the node embedding process can affect the apparent relative effectiveness of unsupervised structural node embedding methods on downstream graph mining tasks, including:*

- *The downstream machine learning classifier*

- *The metric used to evaluate performance*
- *The way that node labels are defined*

7.7.4 Deeper View Into the Performance Scores

Aggregate performance of a classifier over the whole dataset does not tell the whole story. It is also worth exploring what kinds of nodes (e.g., high degree) can be most easily classified by the various structural embedding methods.

Methodology. For degree-based analysis, per dataset with maximum degree $\Delta_{\max}$, we categorize the nodes into low-degree $[0, \Delta_{\max}^{\frac{1}{3}})$, medium-degree $[\Delta_{\max}^{\frac{1}{3}}, \Delta_{\max}^{\frac{2}{3}})$ and high-degree $[\Delta_{\max}^{\frac{2}{3}}, \Delta_{\max}]$ buckets. We then perform classification evaluation per bucket. We apply the same partitioning methodology for the analysis of participating triangles.We use as a case study the EU air-traffic network (we see similar trends in other data). Its maximum degree and maximum number of participating triangles are 202 and 3450, respectively.

Results. In Figure 7.10, we observe that in general, all methods perform best at classifying nodes with high connectivity, as measured by either degree and/or participating in a large number of triangles. This is not surprising and corroborates the literature, as these nodes' local neighborhoods contain richer information [NM16]. Slightly more surprisingly, the least-connected nodes are the next easiest to classify.

Observation 7.8. *Current structural embedding methods are most effective at distinguishing "extreme" network positions in the latent feature space compared to moderate ones.*

Some network positions are easy to identify. For instance, simply using the node degree as a feature (`degree`) performs best at classifying high degree nodes, but is less effective at classifying low- and medium-degree nodes even compared to `degree1`, where neighbors' degrees are considered as features. In general, however, relative ranks of methods are fairly well-preserved across buckets.

7.7.5 A Comprehensive Embedding Comparison: Single-Network Tasks

Having carefully considered the effects of several external factors, we now offer a more comprehensive comparison of embedding methods in Figure 7.11: we give their general

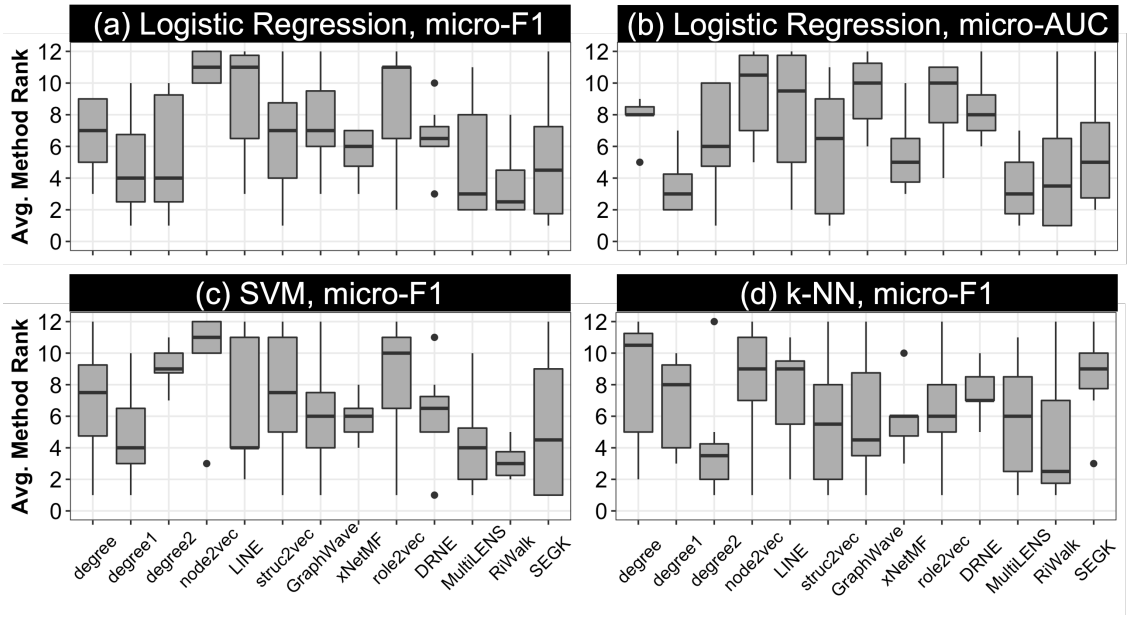


Figure 7.11: Lower is better: performance summarized across all the real datasets. While there is no clear winner, methods based on local degree distribution tend to be consistently top performers.

rankings (lower is better) per classifier and metric across all real datasets. We observe that there is no clear winner of an embedding method, particularly as datasets, labels, classifiers, and metrics may all change. However, we can see that node embedding methods designed to preserve proximity in the network—node2vec and LINE—generally have poorer rankings, as is to be expected for a task where the nodes' structural role carries most of the signal. Other methods are more mixed: e.g., GraphWave achieves a more competitive ranking by both metrics displayed with an SVM and less so with logistic regression.

Some of the best-ranking methods across the board are MultiLENS, SEGK and variations of our degree distribution features. Significantly, they all share common design choices, explicitly modeling a node's position within a local neighborhood using degree-based connectivity (after one iteration, the Weisfeiler-Lehman graph kernel used by SEGK gives nodes the same label if and only if they have same degree, in the absence of other node label information). We believe that the expressive power of local degree distributions has strong implications for future work in structural embedding, as a baseline and an inspiration for methodological design.

Observation 7.9. *Current individual network mining tasks depending on the structural*

roles of nodes can be solved effectively with local aggregation of degree-based connectivity information.

7.8 Multi-Network Tasks

One important benefit of structural embedding methods is that they can be used to compare nodes across graphs [HGER+12, HSSK18, HSK19]. In this section, we apply different structural embeddings to two graph mining tasks involving cross-network comparison: network alignment, which finds node-level matchings between different graphs, and graph classification, which involves comparing entire graphs.

7.8.1 Network Alignment

Methodology. Network alignment can be formulated as nearest-neighbor search given comparable structural node embeddings [HSSK18]. We follow established procedures for simulating a network alignment problem with known ground truth correspondences [HSSK18]: we align a graph with adjacency matrix $\mathbf{A}$ to a randomly permuted version of itself given by $\mathbf{PAP}^\top$ for random permutation matrix $\mathbf{P}$, to which we add noise by removing edges with probability p. We use a k-d tree to quickly match all the nodes in one graph to the nearest neighbor in another graph by embedding similarity, and compute the resulting accuracy. We perform this experiment on the two datasets used in previous works [HSSK18] and described in Section 7.4, using 1% and 5% noise, the lowest and highest noise levels considered in [HSSK18].

Results.

In Figure 7.12, we see that xNetMF, which was originally proposed for graph alignment, captures cross-network node similarities. Proximity-preserving node embedding methods LINE and node2vec are unable to succeed on this task. Neither are role2vec, DRNE or MultiLENS, which may be regarded as hybrids of proximity-preserving and structural embeddings. Node degree alone is too weak a structural descriptor to meaningfully align nodes (many nodes in a network have the same degree), but degree distributions of higher-order local neighborhoods (2nd-order is always better than first-order) are also sufficiently expressive

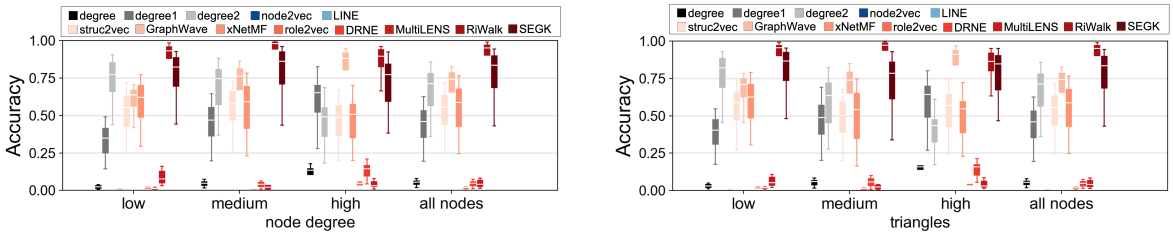


Figure 7.13: Deeper view into performance scores for network alignment.

structural descriptors to perform on par with xNetMF (which also preserves distributional information of neighbor degrees) in many cases.

For this task, some of the most successul methods are successors of xNetMF: SEGK and RiWalk. Both methods generalize the structural connectivity measure between nodes beyond degree alone, which RiWalk notes can be ambiguous [XQQ+19]. In particular, both methods use the Weisfeiler-Lehman neighborhood aggregation method, a well-known heuristic for graph-level similarity which has its roots in a graph isomorphism test [SSL+11]. The neighborhood aggregation process iteratively relabels each node, capturing degree-based statistics in early iterations but gradually building up higher-order information.

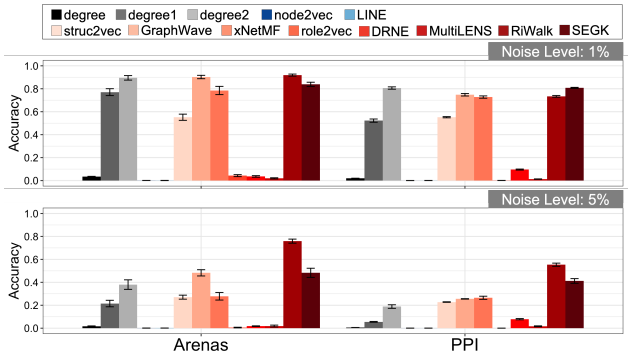


Figure 7.12: Graph alignment results.

Especially in the more challenging alignment settings with 5% noise, RiWalk performs better than all other methods, even SEGK. One reason for this may be because RiWalk is not restricted to local neighborhoods, while methods like SEGK (and xNetMF, struc2vec) model only k-hop neighborhoods of each nodes. Concurrent method GraphWave is also successul (close to SEGK and ahead of xNetMF on the Arenas dataset); GraphWave also considers patterns of local connectivity using heat diffusion processes rather than degree. We note, however, that a partial explanation of the success of structural embedding methods that do not explicitly model node degree may be in part due to the noise model [HSSK18]. The removal of edges may affect the degree distribution more obviously than it affects diffusion processes on graphs.

Observation 7.10. *For alignment of noisy networks, degree-based connectivity alone can be brittle, and the most robust methods generalize the notion of node connectivity beyond degree.*

7.8.2 Graph Classification

Methodology. To classify networks from the structural embeddings of nodes, we use RGM, an unsupervised graph feature map that captures the distribution of the node embeddings in feature space [HSK19]. We then train a linear SVM on the resulting graph features. RGM was shown to work with different choices of node embeddings and yielded comparable or better accuracy to a large variety of baseline graph kernels, neural networks, and unsupervised feature construction methods at faster runtime [HSK19]. We use recommended settings of 4 levels of resolution and 2 iterations of Weisfeiler-Lehman label expansion (when no node labels are available, we begin this process with uniform labels [HSK19]) for RGM. This label expansion helps RGM aggregate the node embeddings more accurately, but we do not use node labels during embedding. For the downstream classification, we consider a linear SVM, as it was shown that RGM with a linear SVM approximates a kernel machine [HSK19].

Results. We plot the results for the different embedding methods in Figure 7.14. For ease of inspection, we also report the numbers in tabular format (Table 7.5). We also give additional context relative to competing methods representing other families of techniques, by including results from the state-of-the-art Weisfeiler-Lehman subtree graph kernel [SSL+11] along with GIN [XHLJ19], a state-of-the-art graph neural network (we use the numbers for the best-performing GIN-0 variant reported in the original paper [XHLJ19]).

We see that particularly on the social networks dataset IMDB-M, skip-gram based methods whose context sampling is not locally restricted (node2vec, role2vec, RiWalk) yield poor performance. node2vec's performance is also explained by the fact that node proximity information does not lead to comparable representations between different graphs, as is confirmed by [HSK19] and by the poor performance of LINE. How-

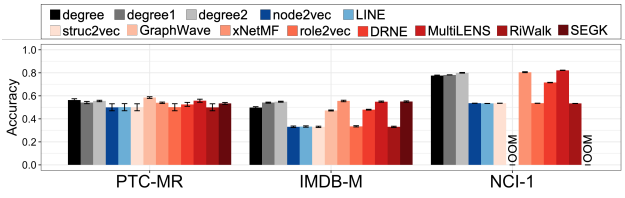


Figure 7.14: Graph classification results. Embedding methods modeling local neighborhoods tend to do best.

Table 7.5: Accuracy of various structural embeddings used in the RGM framework [HSK19] for graph classification, plus strong baselines from other graph classification techniques. (OOM = Out of Memory.) Most accurate embedding method in RGM marked in bold. Average rank computed by accuracy, with ties broken by standard deviation if applicable. Tied methods given rank of highest tie, OOM given a rank below all methods that completed.

Method	PTC-MR	IMDB-M	NCI1	Average Rank
degree	56.3 ± 1.1	49.7 ± 0.9	77.5 ± 0.4	4.33
degree1	54.1 ± 1.0	54.0 ± 0.5	78.2 ± 0.1	4.67
degree2	55.5 ± 0.6	54.9 ± 0.4	80.0 ± 0.3	3.33
node2vec	50.0 ± 3.0	33.1 ± 0.6	53.5 ± 0.1	9.67
LINE	50.1 ± 3.1	33.3 ± 0.6	53.5 ± 0.1	8.67
struc2vec	50.0 ± 3.0	33.0 ± 0.6	53.5 ± 0.1	10
GraphWave	**58.5 ± 0.7**	47.2 ± 0.4	OOM	7
xNetMF	53.9 ± 0.6	**55.5 ± 0.7**	80.5 ± 0.4	3
role2vec	50.1 ± 3.1	33.5 ± 0.5	53.5 ± 0.1	8.33
DRNE	52.6 ± 1.7	47.9 ± 0.4	71.5 ± 0.2	7
MultiLENS	55.7 ± 1.3	54.9 ± 0.5	**82.1 ± 0.1**	2.67
RiWalk	50.0 ± 3.0	33.0 ± 0.6	53.5 ± 0.1	10
SEGK	53.3 ± 0.8	55.0 ± 0.6	OOM	7

ever, RiWalk and role2vec's struggles may be because on these graphs, the random walks oversample the graph and wash out distinguishing structural information.

Observation 7.11. *For graph classification, sampling structural context with random walks risks blurring too much structural information on the small graphs commonly used as benchmarks.*

This generalizes the finding in [HSK19] that methods such as node2vec and struc2vec perform poorly. There, the explanation was that such methods were not inductive; we see that this is true, as LINE, which does not use random walks but does depend on a transductive notion of proximity, also performs equally poorly. However, even structural embedding methods like RiWalk and role2vec, which we saw were useful for cross-network tasks like network alignment, perform poorly here: indicating that the mechanism they all use to sample context may be at fault. Note that the more memory-intensive baselines GraphWave and SEGK are unable to run on the largest NCI1 dataset.

On the other hand, the best performing methods are those that explicitly model local neighborhoods: xNetMF and SEGK. Degree variants also do well, with higher-order hop distances achieving more accuracy on IMDB-M and NCI1. However, in the PTC-MR dataset, which consists of smaller molecular graphs that may not contain the complex structural roles

arising from social behavior, we see less of a gap between all methods, and in fact of the three degree-based methods, Degree does best (by a marginal amount). This indicates that on this dataset, limited local structural information is sufficient.

Observation 7.12. *For graph classification, the best methods locally modeling the connectivity of each node. Considering each node's higher-order connectivity does slightly improve performance on medium to large datasets.*

As an aside, while it is not our goal to set a task-specific state of the art, within RGM the structural embeddings yield competitive performance to other leading methods. Compared to results from the state of the art graph isomorphism networks reported in [XHLJ19] and Weisfeiler-Lehman graph kernels [SSL$^+$11] reported in [HSK19], the best embedding-based methods yield clearly higher numbers on IMDB-M and trail by a fraction of a percentage point on NCI1 (they trail further on PTC-MR). Note that our feature learning method is completely unsupervised (unlike the GIN neural network) and we do not tune the parameters (e.g. number of binning levels) of RGM, which could further improve performance.

7.8.3 A Comprehensive Embedding Comparison: Multi-Network Tasks

For graph classification, the results resemble the results from the single-network tasks in Section 7.7.4. The best methods aggregate local connectivity information for each node, including xNetMF, MultiLENS, SEGK (when it is able to run), and variants of the local degree histograms. On the large datasets, considering second-order neighbors for each node improves over considering only the nodes' features or that of its immediate neighbors, indicating that modeling higher-order connectivity does somewhat help for this task.

For graph alignment, generalizing node connectivity beyond degree is helpful, which is why the most successul methods are RiWalk, SEGK, and GraphWave. This may be in part because the noise model of edge removal [HSSK18] throws off the degree distribution of the graphs, making the degrees in the noisy graph slightly lower. RiWalk and GraphWave are not explicitly confined to modeling any k-hop neighborhood, but SEGK is. This implies that modeling local structural information does not necessarily hurt performance, but using a statistic like degree to assess structural identity that is particularly brittle under the common

noise model is more likely to lower a structural embedding method's performance on network alignment, particularly when the noise is higher.

Observation 7.13. *Multi-network tasks can be solved using node embeddings that capture local structural information. For graph classification, degree is a sufficiently expressive measure of connectivity, but graph alignment requires a more generalized measurement of connectivity.*

.

7.9 Discussion and Conclusions

We conducted a comprehensive empirical study to gain a better understanding of the *equivalence* of the nodes in the networks within the context of embeddings. Our study of the various sociological equivalences confirms that structural equivalence is best captured by proximity-preserving embedding methods like node2vec and LINE, as its definition implies despite its name. On the other hand, methods like struc2vec, xNetMF and GraphWave perform well in automorphic and regular equivalence (though the definition of the latter depends on edge types and is challenging to define in a principled way without this information).

We have split our analysis into two parts (Section 7.6): intrinsic evaluation, which explores the relationship of nodes' embedding similarities and other measures of similarity given by sociological equivalence, and extrinsic evaluation of the embeddings' performance in the context of downstream tasks such as classification or clustering. Our work is one of the first to perform intrinsic *and* extrinsic evaluation of node embeddings (either structural or proximity-based).

While we largely observe similar performance trends in intrinsic and extrinsic evaluation, we also notice some inconsistent trends, a phenomenon which has also been observed in word embedding [CKP16]. For example, MultiLENS is far from a standout in intrinsic evaluation but a top runner in extrinsic evaluation. In both intrinsic and extrinsic clustering evaluation, we have found a complex relationship between the distance metric used (cosine or Euclidean) and the results, which perhaps surprisingly is not always consistent with the metric used in the various embedding objectives.

More generally, we have found that the performance of structural node embedding methods is highly sensitive to many factors that are often chosen seemingly arbitrarily: choice of classifier, performance metric, or node labeling method (Section 7.7). Changing any of these can not only change methods' absolute performance but also their rankings relative to each other, arbitrarily making one embedding method appear better or worse than another. Comparing comprehensively across classifiers, performance metrics, datasets, and labeling schemes, we see that the simple structural property, node degree, can be the building block for some of the most effective methods. Our *local degree histograms* are a simple baseline that proves surprisingly effective across all of our experiments. They may inspire the design of future methods: indeed, they are highly related to xNetMF and MultiLENS, two existing embedding methods that also exhibit generally strong performance.

Overall, our methods provide the structural node embedding research community with new evaluation tools, new insights, and surprising findings about choices in current node embedding praxis that, though they are often made automatically, may significantly impact the apparent success of new methods. We hope that our findings can influence the design of further node embedding methods and also pave the way for future evaluation of existing methods. With new node embedding methods being developed at a breakneck pace, proper evaluation will, as the word embedding community has found, be essential to progress.

CHAPTER VIII

Conclusion

Graphs are a powerful representation of complex interactions between entities. Many real-world problems involve mining data from multiple large graphs collectively, in which case it is desirable to compare entities *across* networks. This thesis has formulated solutions to several collective graph mining problems using node embeddings, which are expressive, efficient techniques for learning features representations for each node in each network. Most embedding objectives preserve proximity between nodes in a network, which is not suitable for comparing nodes in different networks. Instead, we have introduced *structural node embedding* methods that learns similar feature representations for nodes with similar structural roles in their respective network. Such structural roles are comparable across networks, and with embeddings such as xNetMF that preserve structural roles, we have a powerful tool for cross-network comparison at the node and graph level. This thesis has contributed principled, scalable methods both for learning structural node embeddings and for using them to perform **collective network mining** at the **node and graph level**. In addition to methodology, we also advanced the praxis of structural node embedding, by not only showing the success of our methods in solving challenging industry-scale problems, but also providing a suite of benchmark datasets and tasks along with insights that will enable rigorous, standardized development of future works.

8.1 Node-Level Methodology

Network alignment, the task of finding a correspondence between nodes in different networks, is a canonical problem for cross-network node comparison. We proposed a network alignment solution called REGAL, a simple greedy matching of nodes using xNetMF embedding similarity that can achieve high network alignment accuracy. However, because xNetMF models nodes' structural roles and not their relative proximities, nodes that are in close relative proximity in one graph may not remain in close relative proximity when they are mapped to the other graph, violating a desirable principle we defined called *matched neighborhood consistency*. We thus contributed an iterative matrix algorithm called RefiNA that refines an initial network alignment solution to improve matched neighborhood consistency. With RefiNA, REGAL and other network alignment methods achieve up to 90% higher accuracy and allow us to consider 5× noisier graphs than before.

8.2 Graph-Level Methodology

With node embeddings, not only can we compare nodes, but we can also compare entire graphs. Just as a node is well represented by the location of its embedding in vector space, a graph can be well represented by the distribution of its node embeddings in vector space. We designed feature maps for graphs that preserve this information. Our graph feature map, RGM, is a randomized feature construction that elegantly allows us to approximate graph kernel methods. Specifically, the dot product of two graphs' RGM features approximates the Laplacian kernel mean map, or the average pairwise similarity between all the nodes in the two graphs. Again, computing this statistic exactly would involve comparing all nodes between graphs, which as discussed above would incur quadratic time complexity in the number of nodes. With RGM, we can approximate this statistic in linear time. Moreover, we can also avoid the problem of comparing all pairs of *graphs*, another computational bottleneck faced by methods that compute and train a kernel machine on a graph kernel matrix. Thus, our proposed solution scales to large *graphs* and large *numbers* of graphs, while remaining highly accurate compared to state-of-the-art methods for graph classification.

8.3 Praxis

While our thesis has contributed new algorithmic methods, we also envision the applications of structural node embeddings to important real-world problems. We demonstrated their use on a large-scale problem: inferring the professional roles of email users. Although we cannot view the text of their emails for privacy reasons, just the network structure of users' communication provides a powerful signal of their professional role: users with similar professional roles may not interact directly (like executives in different industries) but likely play similar structural roles in their respective parts of the larger who-emails-whom network. We proposed EMBER, which extends xNetMF to handle edge weights and directions to model email communication behavior more precisely. Our proposed solution outperforms a variety of graph mining solutions and allows us to mine email behavior at several scales. We applied EMBER to subnetworks corresponding to individual companies, which allow us to study how professional roles compare and contrast across different size companies. Additionally, EMBER also scales efficiently to infer professional roles in email networks consisting of millions of users.

All of our previous works have demonstrated that structural node embeddings are useful for a wide variety of data mining problems. Thus, we contributed new techniques to *evaluate* structural node embeddings so that the research community can continue to develop effective methods. We evaluated a comprehensive collection of structural node embedding methods, ours and others. We first contributed several new synthetic and real datasets as benchmarks for structural node embedding methods. Our evaluation consisted of two parts: *intrinsic* evaluation of the properties that they capture and *extrinsic* evaluation (evaluating their performance on downstream graph mining tasks). Our insights showed how the structural roles each method models correspond to three different notions of sociological role equivalence of nodes in networks. We also uncovered pitfalls in current evaluation praxis, identified effective methodological design choices, and proposed new best practices.

8.4 Future Directions

This thesis work has shown that structural node embeddings are useful for a variety of graph mining tasks on one or more networks. We now describe two interesting directions for future work.

8.4.1 Evolving Structural Roles in Dynamic Networks

Graphs in the real world are often dynamic, with nodes being added or dropped and new edges forming and disappearing as time goes on. In this thesis, we focused on static graphs with a fixed network structure. Other work of ours [JHRK19] has introduced node embeddings that capture structural behavior in temporal networks with heterogeneous node and edge types. There, we consider a user modeling task in which we seek to learn a structural role for each user (node) that respects its temporal interactions in the network.

An alternative perspective is to study how the structural roles of node *change* over time, a problem which has been studied using hand-engineered node statistics [RGNH12] but not, to the best of our knowledge, with more powerful node embeddings. Understanding how the structural roles of entities in networks change can yield many important real-world insights. For instance, in Chapter VI, where we modeled structural roles of email users in a communication network in order to predict the users' professional roles, the change in a user's structural network role might reflect changes in the user's professional role. As an example, a manager starting to exhibit communication patterns more and more similar to that of an executive might be due for a promotion to an executive position–or may be about to leave the company for a more senior role elsewhere. Widespread changes in structural roles of nodes may also indicate how a graph itself is evolving. In Chapter V, we characterized a graph by the distribution of its (structural) node embeddings in vector space. If many of the structural roles of nodes change, this distribution would also change. Returning to the problem of email users at companies from Chapter VI, we could form a graph-level representation of a company from the distribution of its employee's structural roles. Recall that we saw small startups where even the executives' communication behavior resembled that of managers at larger companies. However, as such a startup grew, we might see its

executives' communication habits change to reflect their increasing professional importance, which would correspondingly reflect on the company's graph-level representation. More generally, changing structural roles at the node and graph level may represent different kinds of exceptional or noteworthy phenomena – whether healthy behavioral evolution, or unhealthy and anomalous activity – in social and information networks.

8.4.2 Complementing Structural Roles with Node Proximities

Structural embeddings are a natural choice for multi-network tasks because the structural roles of nodes do not depend on their proximity to any specific node. However, we showed in Chapter IV that modeling *intra*-network proximity is still helpful for the *cross*-network task of network alignment: our principle of matched neighborhood consistency encourages nodes that are connected (in close proximity) in one graph to remain in close proximity after the cross-graph mapping. The broader question is: can we still benefit from modeling node proximities in individual networks even in collective network mining tasks? Our recent work [CHVK20] provides evidence that proximity-preserving node embeddings can be used for cross-network comparison, if we first align the embedding subspaces.

In Chapter IV, our method RefiNA used the proximity of nodes to enforce consistency in an initial network alignment solution, such as one found using structural embeddings by REGAL. In RefiNA, the node proximities are given by the adjacency matrices. However, the success of proximity-preserving embeddings on single-network tasks shows that such embeddings may better model higher-order proximity beyond the mere existence of direct connections given by the adjacency matrix. Thus, using proximity-preserving embeddings to complement structural node embeddings, whether for network alignment or other collective graph mining tasks, is a promising direction for future work.

Proximity-preserving and structural embeddings may have interesting methodological connections. The structural embedding method GraphWave [DZHL18] derives a structural embedding for each node from the heat kernel, a diffusion process which models proximity between nodes. While a proximity-preserving node embedding method might learn similar embeddings for nodes that send each other a large amount of heat, in GraphWave, a node's embedding models the distribution of heat it sends to all other nodes. Emerging theoretical

work [SR20] suggests that structural and proximity-preserving node embeddings may have more in common than was previously realized, and likens their relationship to that of a distribution and its samples. Fully characterizing the methodological connection of proximity-preserving and structural node embeddings may lead to interesting new proximity-preserving and structural node embeddings methods and applications.

8.4.3 Funding Acknowledgements

The work in this dissertation was supported in part by the National Science Foundation under Grant No. IIS 1743088, an Adobe Digital Experience research faculty award, an Amazon Faculty Award, Trove AI, an Army Young Investigator Award No. W911NF1810397, and the University of Michigan. Any opinions, findings, and conclusions or recommendations expressed in this material do not necessarily reflect the views of the NSF or other funding parties.

8.5 A Confectionery Recap

Following the defense of this dissertation, a celebration ensued, as is commonplace, with cake. The collection of cakes in Figure 8.1 not only celebrates but also illustrates this dissertation's methodological contributions, particularly those in Chapters III and V. The cake decorations display high-level ideas and a few technical details that could be articulated within the limits of the author's baking ability.

Figure 8.1: The "Dessertation"

BIBLIOGRAPHY

BIBLIOGRAPHY

[AM15] Ahmed Alaoui and Michael W Mahoney. Fast randomized kernel ridge regression with statistical guarantees. In *NeurIPS*, 2015.

[ARKG13] Abtin Zohrabi Aliabadi, Fatemeh Razzaghi, Seyed Pooria Madani Kochak, and Ali Akbar Ghorbani. Classifying organizational roles using email social networks. In *Canadian AI*, pages 301–307. Springer, 2013.

[ARL+19] Nesreen K. Ahmed, Ryan A. Rossi, John Boaz Lee, Theodore L. Willke, Rong Zhou, Xiangnan Kong, and Hoda Eldardiry. role2vec: Role-based network embeddings. In *KDD DLG Workshop*, 2019.

[AT16] James Atwood and Don Towsley. Diffusion-convolutional neural networks. In *NeurIPS*, 2016.

[ATK15] Leman Akoglu, Hanghang Tong, and Danai Koutra. Graph based anomaly detection and description: A survey. *DAMI*, 29(3):626–688, 2015.

[AZRP18] Bijaya Adhikari, Yao Zhang, Naren Ramakrishnan, and B Aditya Prakash. Sub2vec: Feature learning for subgraphs. In *PAKDD*. Springer, 2018.

[B+10] Nitin Bhatia et al. Survey of nearest neighbor techniques. *International Journal of Computer Science and Information Security*, 8(2):302–305, 2010.

[Bak18] Amir Bakarov. A survey of word embeddings evaluation methods. *arXiv preprint arXiv:1801.09536*, 2018.

[BBA75] Ronald L Breiger, Scott A. Boorman, and Phipps Arabie. An algorithm for clustering relational data with applications to social network analysis and comparison with multidimensional scaling. *Journal of Mathematical Psychology*, 12(3):328–383, 1975.

[BC01] Avrim Blum and Shuchi Chawla. Learning from Labeled and Unlabeled Data Using Graph Mincuts. In *Proceedings of the 18th International Conference on Machine Learning*, pages 19–26. Morgan Kaufmann, San Francisco, CA, 2001.

[BE92] Stephen Borgatti and Martin Everett. Notions of position in social network analysis. *Sociological Methodology*, 22, 01 1992.

[BE93] Stephen P. Borgatti and Martin G. Everett. Two algorithms for computing regular equivalence. *Social Networks*, 15(4):361 – 376, 1993.

[BEF02] S.P. Borgatti, M. G. Everett, and L. C. Freeman. UCINET 6 for Windows: Software for Social Network Analysis. Harvard, MA, Analytic Technologies, 2002.

[BG18] David B Blumenthal and Johann Gamper. On the exact computation of the graph edit distance. *Pattern Recognition Letters*, 2018.

[BGSW13] Mohsen Bayati, David F Gleich, Amin Saberi, and Ying Wang. Message-passing algorithms for sparse network alignment. *TKDD*, 7(1):3, 2013.

[BK05] Karsten M. Borgwardt and Hans-Peter Kriegel. Shortest-path kernels on graphs. In *ICDM*. IEEE, 2005.

[BKERF13] Michele Berlingerio, Danai Koutra, Tina Eliassi-Rad, and Christos Faloutsos. Network similarity via multiple social theories. In *ASONAM*. IEEE/ACM, 2013.

[BKTK19] Caleb Belth, Fahad Kamran, Donna Tjandra, and Danai Koutra. When to remember where you came from: Node representation learning in higher-order networks. In *ASONAM)*, pages 222–225, 2019.

[BLO+18] Elizabeth Boschee, Jennifer Lautenschlager, Sean O'Brien, Steve Shellman, and James Starz. ICEWS Automated Daily Event Data. `https://doi.org/10.7910/DVN/QI2T9A`, 2018.

[BSR+08] Bobby-Joe Breitkreutz, Chris Stark, Teresa Reguly, Lorrie Boucher, Ashton Breitkreutz, Michael Livstone, Rose Oughtred, Daniel H Lackner, Jürg Bähler, Valerie Wood, et al. The biogrid interaction database: 2008 update. *Nucleic acids research*, 36(suppl 1):D637–D640, 2008.

[CB06] Michael S Cole and Heike Bruch. Organizational identity strength, identification, and commitment and their relationships to turnover intention: Does organizational hierarchy matter? *Journal of Organizational Behavior: The International Journal of Industrial, Occupational and Organizational Psychology and Behavior*, 27(5):585–605, 2006.

[CHVK20] Xiyuan Chen, Mark Heimann, Fatemeh Vahedian, and Danai Koutra. Cone-align: Consistent embedding-based network alignment. In *CIKM*, 2020.

[CKP16] Billy Chiu, Anna Korhonen, and Sampo Pyysalo. Intrinsic evaluation of word vectors fails to predict extrinsic performance. In *Proceedings of the 1st workshop on evaluating vector-space representations for NLP*, pages 1–6, 2016.

[CLX16] Shaosheng Cao, Wei Lu, and Qiongkai Xu. Deep neural networks for learning graph representations. In *AAAI*, pages 1145–1152, 2016.

[CMX18] Jie Chen, Tengfei Ma, and Cao Xiao. Fastgcn: Fast learning with graph convolutional networks via importance sampling. In *ICLR*, 2018.

[DA79] Russell R. Dynes and B.E. Aguirre. Organizational adaptation to crises: Mechanisms of coordination and structural change. *Disasters*, 3(1):71–74, 1979.

[DCS17] Yuxiao Dong, Nitesh V Chawla, and Ananthram Swami. metapath2vec: Scalable representation learning for heterogeneous networks. In *KDD*, pages 135–144, 2017.

[DG18] Ayushi Dalmia and Manish Gupta. Towards interpretation of node embeddings. In *Companion Proceedings of the The Web Conference*, pages 945–952, 2018.

[DKL+19] Tyler Derr, Hamid Karimi, Xiaorui Liu, Jiejun Xu, and Jiliang Tang. Deep adversarial network alignment. *arXiv preprint arXiv:1902.10307*, 2019.

[DM05] Petros Drineas and Michael W Mahoney. On the nyström method for approximating a gram matrix for improved kernel-based learning. *JMLR*, 6:2153–2175, 2005.

[DRN] DRNE codebase. `github.com/tadpole/DRNE`.

[DYZ19] Xingbo Du, Junchi Yan, and Hongyuan Zha. Joint link prediction and network alignment via cross-graph embedding. In *IJCAI*, pages 2251–2257. AAAI Press, 2019.

[DZHL18] Claire Donnat, Marinka Zitnik, David Hallac, and Jure Leskovec. Learning structural node embeddings via diffusion wavelets. In *KDD*, 2018.

[DZK+18] Joel Douglas, Ben Zimmerman, Alexei Kopylov, Jiejun Xu, Daniel Sussman, and Vince Lyzinski. Metrics for evaluating network alignment. *GTA3 at WSDM*, 2018.

[EB88] Martin G. Everett and Steve Borgatti. Calculating role similarities: An algorithm that helps determine the orbits of a graph. *Social Networks*, 10(1):77 – 91, 1988.

[EP19] Alessandro Epasto and Bryan Perozzi. Is a single embedding enough? learning node representations that capture multiple social contexts. In *The Web Conference*, pages 394–404, 2019.

[FLM+20] M. Fey, J. E. Lenssen, C. Morris, J. Masci, and N. M. Kriege. Deep graph matching consensus. In *ICLR*, 2020.

[GD07] Kristen Grauman and Trevor Darrell. The pyramid match kernel: Efficient learning with sets of features. *JMLR*, 8:725–760, 2007.

[GF18] Palash Goyal and Emilio Ferrara. Graph embedding techniques, applications, and performance: A survey. *Knowledge-Based Systems*, 151:78–94, 2018.

[GGKF15] Wolfgang Gatterbauer, Stephan Günnemann, Danai Koutra, and Christos Faloutsos. Linearized and Single-Pass Belief Propagation. *PVLDB*, 8(5), 2015.

[GHG+19] Palash Goyal, Di Huang, Ankita Goswami, Sujit Rokka Chhetri, Arquimedes Canedo, and Emilio Ferrara. Benchmarks for graph embedding evaluation. *arXiv preprint arXiv:1908.06543*, 2019.

[GL16] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *KDD*, 2016.

[GR96] Steven Gold and Anand Rangarajan. A graduated assignment algorithm for graph matching. *IEEE TPAMI*, 18(4):377–388, 1996.

[gra] GraphWave codebase. `github.com/snap-stanford/graphwave`.

[GVS+19] Saket Gurukar, Priyesh Vijayan, Aakash Srinivasan, Goonmeet Bajaj, Chen Cai, Moniba Keymanesh, Saravana Kumar, Pranav Maneriker, Anasua Mitra, Vedang Patel, et al. Network representation learning: Consolidation and renewed bearing. *arXiv preprint arXiv:1905.00987*, 2019.

[Har90] James R Harris. Ethical values of individuals at different levels in the organizational hierarchy of a single firm. *Journal of Business Ethics*, 9(9):741–750, 1990.

[HGER+12] Keith Henderson, Brian Gallagher, Tina Eliassi-Rad, Hanghang Tong, Sugato Basu, Leman Akoglu, Danai Koutra, Christos Faloutsos, and Lei Li. Rolx: structural role extraction & mining in large graphs. In *KDD*, pages 1231–1239. ACM, 2012.

[HHB+03] Mark S. Handcock, David Hunter, Carter T. Butts, Steven M. Goodreau, and Martina Morris. statnet: An r package for the statistical modeling of social networks. `http://www.csde.washington.edu/statnet`, 2003.

[HK17] Mark Heimann and Danai Koutra. On generalizing neural node embedding methods to multi-network problems. In *KDD MLG Workshop*, 2017.

[HL12] Xia Hu and Huan Liu. Social status and role analysis of Palin's email network. In *WWW*, pages 531–532. ACM, 2012.

[HLH17] Xiao Huang, Jundong Li, and Xia Hu. Label informed attributed network embedding. In *WSDM*, 2017.

[HLP+18] Mark Heimann, Wei Lee, Shengjie Pan, Kuan-Yu Chen, and Danai Koutra. Hashalign: Hash-based alignment of multiple graphs. In *PAKDD*, pages 726–739. Springer, 2018.

[HSK19] Mark Heimann, Tara Safavi, and Danai Koutra. Distribution of node embeddings as multiresolution features for graphs. In *ICDM*. IEEE, 2019.

[HSSK18] Mark Heimann, Haoming Shen, Tara Safavi, and Danai Koutra. Regal: Representation learning-based graph alignment. In *CIKM*, 2018.

[HYL17] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *NeurIPS*, pages 1025–1035, 2017.

[IB18] Sergey Ivanov and Evgeny Burnaev. Anonymous walk embeddings. In *ICML*, 2018.

[JD15] Fredrik D Johansson and Devdatt Dubhashi. Learning with similarity functions on graphs using matchings of geometric embeddings. In *KDD*, 2015.

[JDE+20] Amin Javari, Tyler Derr, Pouya Esmailian, Jiliang Tang, and Kevin Chen-Chuan Chang. Rose: Role-based signed network embedding. In *The Web Conference*, pages 2782–2788, 2020.

[JHJK20] Junchen Jin, Mark Heimann, Di Jin, and Danai Koutra. Understanding and evaluating structural node embeddings. In *KDD MLG Workshop*, 2020.

[JHRK19] Di Jin, Mark Heimann, Ryan A. Rossi, and Danai Koutra. Node2bits: Compact time- and attribute-aware node representations for user stitching. In *PKDD*, 2019.

[JHS+19] Di Jin, Mark Heimann, Tara Safavi, Mengdi Wang, Wei Lee, Lindsay Snider, and Danai Koutra. Smart roles: Inferring professional roles in email networks. In *KDD*, 2019.

[JK17] Di Jin and Danai Koutra. Exploratory analysis of graph data by leveraging domain knowledge. In *ICDM*. IEEE, 2017.

[Joa06] Thorsten Joachims. Training linear svms in linear time. In *KDD*, pages 217–226. ACM, 2006.

[JRK+19] Di Jin, Ryan A Rossi, Eunyee Koh, Sungchul Kim, Anup Rao, and Danai Koutra. Latent network summarization: Bridging network embedding and summarization. In *KDD*, 2019.

[JW02] Glen Jeh and Jennifer Widom. Simrank: a measure of structural-context similarity. In *KDD*, 2002.

[KF17] Danai Koutra and Christos Faloutsos. *Individual and Collective Graph Mining: Principles, Algorithms, and Applications*. Synthesis Lectures on Dat Min and Knowl Disc. Morgan & Claypool Publishers, 2017.

[KGW16] Nils M Kriege, Pierre-Louis Giscard, and Richard Wilson. On valid optimal assignment kernels and applications to graph classification. In *Advances in Neural Information Processing Systems*, pages 1623–1631, 2016.

[KKK+11] Danai Koutra, Tai-You Ke, U. Kang, Duen Horng Chau, Hsing-Kuo Kenneth Pao, and Christos Faloutsos. Unifying guilt-by-association approaches: Theorems and fast algorithms. In *PKDD*, 2011.

[KKM+16] Kristian Kersting, Nils M. Kriege, Christopher Morris, Petra Mutzel, and Marion Neumann. Benchmark data sets for graph kernels. `http://graphkernels.cs.tu-dortmund.de`, 2016.

[Kla09] Gunnar W Klau. A new graph-based method for pairwise global network alignment. *BMC bioinformatics*, 10 Suppl 1:S59, 2009.

[KMM+10] Oleksii Kuchaiev, Tijana Milenković, Vesna Memišević, Wayne Hayes, and Nataša Pržulj. Topological network alignment uncovers biological function and phylogeny. *Journal of the Royal Society Interface*, 7(50):1341–1354, 2010.

[KMT12] Sanjiv Kumar, Mehryar Mohri, and Ameet Talwalkar. Sampling methods for the Nyström method. *JMLR*, 13(Apr):981–1006, 2012.

[KNKM14] Nils Kriege, Marion Neumann, Kristian Kersting, and Petra Mutzel. Explicit versus implicit graph feature maps: A computational phase transition for walk kernels. In *ICDM*, pages 881–886. IEEE, 2014.

[KP16] Risi Kondor and Horace Pan. The multiscale laplacian graph kernel. In *NeurIPS*, 2016.

[KTL13] Danai Koutra, Hanghang Tong, and David Lubensky. Big-align: Fast bipartite graph alignment. In *ICDM*, pages 389–398. IEEE, 2013.

[Kun13] Jérôme Kunegis. Konect: the koblenz network collection. In *WWW*, pages 1343–1350. ACM, 2013.

[KVF13] Danai Koutra, Joshua T Vogelstein, and Christos Faloutsos. Deltacon: A principled massive-graph similarity function. In *SDM*, 2013.

[KW17a] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *ICLR*, 2017.

[KW17b] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *ICLR*, 2017.

[LCLL16] Li Liu, William K Cheung, Xin Li, and Lejian Liao. Aligning users across social networks using network embedding. In *IJCAI*, pages 1774–1780, 2016.

[LG14] Omer Levy and Yoav Goldberg. Neural word embedding as implicit matrix factorization. In *NeurIPS*, 2014.

[LHN06] E A Leicht, Petter Holme, and M E J Newman. Vertex similarity in networks. *Physical review. E, Statistical, nonlinear, and soft matter physics*, 73 2 Pt 2:026120, 2006.

[LIN] LINE codebase. `github.com/tangjianpku/LINE`.

[LK14] Jure Leskovec and Andrej Krevl. SNAP Datasets: Stanford large network dataset collection. `http://snap.stanford.edu/data`, June 2014.

[LM14] Quoc Le and Tomas Mikolov. Distributed representations of sentences and documents. In *ICML*, 2014.

[LRK+19] John Boaz Lee, Ryan Rossi, Xiangnan Kong, Sungchul Kim, Eunyee Koh, and Anup Rao. Graph convolutional networks with motif-based attention. In *CIKM*, 2019.

[LS01] Daniel D Lee and H Sebastian Seung. Algorithms for non-negative matrix factorization. In *Advances in neural information processing systems*, pages 556–562, 2001.

[LW71] Francois Lorrain and Harrison C. White. Structural equivalence of individuals in social networks. *The Journal of Mathematical Sociology*, 1(1):49–80, 1971.

[LXT+15] Yitan Li, Linli Xu, Fei Tian, Liang Jiang, Xiaowei Zhong, and Enhong Chen. Word embedding revisited: A new representation learning and explicit matrix factorization perspective. In *IJCAI*, 2015.

[MKKM16] Christopher Morris, Nils M Kriege, Kristian Kersting, and Petra Mutzel. Faster kernels for graphs with continuous attributes via hashing. In *ICDM*, pages 1095–1100. IEEE, 2016.

[MKM17] Christopher Morris, Kristian Kersting, and Petra Mutzel. Glocalized weisfeiler-lehman graph kernels: Global-local feature maps of graphs. In *ICDM*, pages 327–336. IEEE, 2017.

[MSC+13] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *NeurIPS*, 2013.

[mul] MultiLENS codebase. `github.com/GemsLab/MultiLENS`.

[MV09] Pierre Mahé and Jean-Philippe Vert. Graph kernels based on tree patterns for molecules. *Mach. Learn.*, 75(1):3–35, April 2009.

[NAK16] Mathias Niepert, Mohamed Ahmed, and Konstantin Kutzkov. Learning convolutional neural networks for graphs. In *ICML*, 2016.

[NGBK16] Marion Neumann, Roman Garnett, Christian Bauckhage, and Kristian Kersting. Propagation kernels: efficient graph kernels from propagated information. *Mach. Learn.*, 102(2):209–245, 2016.

[NH18] Morteza Noshad and Alfred O Hero. Scalable hash-based estimation of divergence measures. In *AISTATS*, 2018.

[NM16] Sharad Nandanwar and M Narasimha Murty. Structural neighborhood based classification of nodes in a network. In *KDD*, pages 1085–1094, 2016.

[NMV17] Giannis Nikolentzos, Polykarpos Meladianos, and Michalis Vazirgiannis. Matching node embeddings for graph similarity. In *AAAI*, 2017.

[NV19] Giannis Nikolentzos and Michalis Vazirgiannis. Learning structural node representations using graph kernels. *TKDE*, 2019.

[OLJT13] Byung-Won On, Ee-Peng Lim, Jing Jiang, and Loo-Nin Teow. Engagingness and responsiveness behavior models on the enron email network and its application to email reply order prediction. In *The influence of technology on social network analysis and mining*, pages 227–253. Springer, 2013.

[PARS14] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. In *KDD*, 2014.

[PBMW99] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: Bringing order to the web. Technical Report 1999-66, Stanford InfoLab, November 1999. Previous number = SIDL-WP-1999-0120.

[PK12] Rob Patro and Carl Kingsford. Global network alignment using multiscale spectral signatures. *Bioinformatics*, 28(23):3105–3114, 2012.

[PPRB13] Adriana Prado, Marc Plantevit, Céline Robardet, and Jean-Francois Boulicaut. Mining graph topological patterns: Finding covariations among vertex descriptors. *IEEE TKDE*, 25(9):2090–2104, 2013.

[PS98] Christos H Papadimitriou and Kenneth Steiglitz. *Combinatorial optimization: algorithms and complexity*. Courier Corporation, 1998.

[PVGea11] F. Pedregosa, G. Varoquaux, A. Gramfort, and et al. Scikit-learn: Machine learning in Python. *JMLR*, 12:2825–2830, 2011.

[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.

[QSR+20] Kai Qin, Flora D. Salim, Yongli Ren, Wei Shao, Mark Heimann, and Danai Koutra. G-crewe: Graph compression with embedding for network alignment. In *CIKM*, 2020.

[RA15] R. A. Rossi and N. K. Ahmed. Role discovery in networks. *IEEE Transactions on Knowledge and Data Engineering*, 27(4):1112–1131, April 2015.

[RCHS07] Ryan Rowe, German Creamer, Shlomo Hershkop, and Salvatore J Stolfo. Automated social hierarchy detection through email network analysis. In *WebKDD / SNA-KDD*, pages 109–117. ACM, 2007.

[RGNH12] Ryan Rossi, Brian Gallagher, Jennifer Neville, and Keith Henderson. Role-dynamics: Fast mining of large dynamic networks. In *Proceedings of the 21st International Conference Companion on World Wide Web*, pages 997–1006. ACM, 2012.

[riw] Riwalk codebase. `github.com/maxuewei2/RiWalk`.

[RJK+20] Ryan A Rossi, Di Jin, Sungchul Kim, Nesreen K Ahmed, Danai Koutra, and John Boaz Lee. On proximity and structural role-based embeddings in networks: Misconceptions, techniques, and applications. *TKDD*, 2020.

[rol] role2vec codebase. `github.com/benedekrozemberczki/role2vec`.

[RR08] Ali Rahimi and Benjamin Recht. Random features for large-scale kernel machines. In *NIPS*, 2008.

[RSF17] Leonardo FR Ribeiro, Pedro HP Saverese, and Daniel R Figueiredo. struc2vec: Learning node representations from structural identity. In *KDD*, pages 385–394. ACM, 2017.

[SA06] Jitesh Shetty and Jafar Adibi. The enron email dataset database schema and brief statistical report. Technical report, USC Information Sciences Institute, 2006. Online; accessed 3-June-2018.

[seg] Segk codebase. `github.com/giannisnik/segk`.

[SERG14] Sucheta Soundarajan, Tina Eliassi-Rad, and Brian Gallagher. A guide to selecting a network similarity method. In *SDM*. SIAM, 2014.

[SFS+20] Tara Safavi, Adam Fourney, Robert Sim, Marcin Juraszek, Shane Williams, Ned Friend, Danai Koutra, and Paul N Bennett. Toward activity discovery in the personal web. In *WSDM*, pages 492–500, 2020.

[SGSS07] Alex Smola, Arthur Gretton, Le Song, and Bernhard Schölkopf. A hilbert space embedding for distributions. In *ALT*, 2007.

[She18] Nino Shervashidze. Graph kernels in MATLAB. `http://mlcb.is.tuebingen.mpg.de/Mitarbeiter/Nino/Graphkernels/`, 2018.

[SHH+06] Salvatore J Stolfo, Shlomo Hershkop, Chia-Wei Hu, Wei-Jen Li, Olivier Nimeskern, and Ke Wang. Behavior-based modeling and its application to email analysis. *TOIT*, 6(2):187–221, 2006.

[Sin64] Richard Sinkhorn. A relationship between arbitrary positive matrices and doubly stochastic matrices. *The Annals of Mathematical Statistics*, 35(2):876–879, 1964.

[SLMJ15] Tobias Schnabel, Igor Labutov, David Mimno, and Thorsten Joachims. Evaluation methods for unsupervised word embeddings. In *EMNLP*, pages 298–307, 2015.

[SM14] Vikram Saraph and Tijana Milenković. Magna: maximizing accuracy in global network alignment. *Bioinformatics*, 30(20):2931–2940, 2014.

[SR20] Balasubramaniam Srinivasan and Bruno Ribeiro. On the equivalence between node embeddings and structural graph representations. In *ICLR*, 2020.

[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.

[str] struc2vec codebase. `github.com/leoribeiro/struc2vec`.

[SVP+09] Nino Shervashidze, SVN Vishwanathan, Tobias Petri, Kurt Mehlhorn, and Karsten M Borgwardt. Efficient graphlet kernels for large graph comparison. In *AISTATS*, 2009.

[SXB08] Rohit Singh, Jinbo Xu, and Bonnie Berger. Global alignment of multiple protein interaction networks with application to functional orthology detection. *PNAS*, 105(35):12763–8, Sep 2008.

[TBL+20] Puja Trivedi, Alican Büyükçakır, Yin Lin, Yinlong Qian, Di Jin, and Danai Koutra. On structural vs. proximity-based temporal node embeddings. In *KDD MLG Workshop*, 2020.

[TCW+18] Ke Tu, Peng Cui, Xiao Wang, Philip S. Yu, and Wenwu Zhu. Deep recursive network embedding with regular equivalence. In *KDD*, pages 2357–2366, 2018.

[THC07] Chi-Yao Tseng, Jen-Wei Huang, and Ming-Syan Chen. Promail: using progressive email social network for spam detection. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 833–840, 2007.

[TMK+18] Anton Tsitsulin, Davide Mottin, Panagiotis Karras, Alexander Bronstein, and Emmanuel Müller. Netlsd: Hearing the shape of a graph. In *SIGKDD*, pages 2347–2356. ACM, 2018.

[TQW+15] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. Line: Large-scale information network embedding. In *WWW*, pages 1067–1077. ACM, 2015.

[Ume88] Shinji Umeyama. An eigendecomposition approach to weighted graph matching problems. *IEEE TPAMI*, 10(5):695–703, 1988.

[VCP+11] Joshua T. Vogelstein, John M. Conroy, Louis J. Podrazik, Steven G. Kratzer, Donniell E. Fishkind, R. Jacob Vogelstein, and Carey E. Priebe. Fast inexact graph matching with applications in statistical connectomics. *CoRR*, abs/1112.5507, 2011.

[VM17] Vipin Vijayan and Tijana Milenković. Multiple network alignment via multimagna++. *IEEE/ACM TCBB*, 2017.

[VMCG09] Bimal Viswanath, Alan Mislove, Meeyoung Cha, and Krishna P Gummadi. On the evolution of user interaction in facebook. In *WOSN*, pages 37–42. ACM, 2009.

[VSKB10] S. V. N. Vishwanathan, Nicol N. Schraudolph, Risi Imre Kondor, and Karsten M. Borgwardt. Graph kernels. *JMLR*, 11:1201–1242, 2010.

[VZ17] Saurabh Verma and Zhi-Li Zhang. Hunt for the unique, stable, sparse and fast feature learning on graphs. In *NeurIPS*, 2017.

[Wan14] Qinna Wang. Link prediction and threads in email networks. In *DSAA*, pages 470–476, Oct 2014.

[WB09] Garnett Wilson and Wolfgang Banzhaf. Discovery of email communication networks from the enron corpus with a genetic algorithm using social network analysis. In *CEC*, pages 3256–3263. IEEE, 2009.

[WCZ16] Daixin Wang, Peng Cui, and Wenwu Zhu. Structural deep network embedding. In *KDD*, pages 1225–1234. ACM, 2016.

[WF94] Stanley Wasserman and Katherine Faust. *Social Network Analysis: Methods and Applications*. Structural Analysis in the Social Sciences. Cambridge University Press, 1994.

[WS01] Christopher KI Williams and Matthias Seeger. Using the Nyström method to speed up kernel machines. In *NeurIPS*, pages 682–688, 2001.

[WSZ+19] Felix Wu, Amauri Souza, Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Weinberger. Simplifying graph convolutional networks. In *ICML*, pages 6861–6871, 2019.

[WYZ+19] Lingfei Wu, Ian En-Hsu Yen, Zhen Zhang, Kun Xu, Liang Zhao, Xi Peng, Yinglong Xia, and Charu Aggarwal. Scalable global alignment graph kernel using random features: From node embedding to graph embedding. In *KDD*. ACM, 2019.

[XHLJ19] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *ICLR*, 2019.

[XLZD19] Hongteng Xu, Dixin Luo, Hongyuan Zha, and Lawrence Carin Duke. Gromov-wasserstein learning for graph matching and node embedding. pages 6932–6941, 2019.

[xne] xNetMF codebase. `github.com/GemsLab/REGAL`.

[XQQ+19] Ma Xuewei, Geng Qin, Zhiyang Qiu, Mingxin Zheng, and Zhe Wang. Riwalk: Fast structural node embedding via role identification. In *ICDM*. IEEE, 2019.

[YDBA17] Liu Yang, Susan T Dumais, Paul N Bennett, and Ahmed Hassan Awadallah. Characterizing and predicting enterprise email reply behavior. In *SIGIR*, pages 235–244. ACM, 2017.

[YFW03] J.S. Yedidia, W.T. Freeman, and Y. Weiss. Understanding Belief Propagation and its Generalizations. *Exploring Artificial Intelligence in the New Millennium*, 8:236–239, 2003.

[YHJK18] Yujun Yan, Mark Heimann, Di Jin, and Danai Koutra. Fast flow-based random walk with restart in a multi-query setting. In *Proceedings of the 2018 SIAM International Conference on Data Mining*, pages 342–350. SIAM, 2018.

[YLZ+15] Cheng Yang, Zhiyuan Liu, Deli Zhao, Maosong Sun, and Edward Chang. Network representation learning with rich text information. In *IJCAI*, 2015.

[YV15] Pinar Yanardag and SVN Vishwanathan. Deep graph kernels. In *KDD*, 2015.

[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.

[YZD+19] Yujun Yan, Jiong Zhu, Marlena Duda, Eric Solarz, Chandra Sripada, and Danai Koutra. Groupinn: Grouping-based interpretable neural network for classification of limited, noisy brain data. In *KDD*. ACM, 2019.

[ZBV09] Mikhail Zaslavskiy, Francis Bach, and Jean-Philippe Vert. A Path Following Algorithm for the Graph Matching Problem. *TPAMI*, 31(12):2227–2242, December 2009.

[ZCNC18] Muhan Zhang, Zhicheng Cui, Marion Neumann, and Yixin Chen. An end-to-end deep learning architecture for graph classification. In *AAAI*, 2018.

[Zhu05] Xiaojin Jerry Zhu. Semi-supervised learning literature survey. Technical report, University of Wisconsin-Madison Department of Computer Sciences, 2005.

[ZT16] Si Zhang and Hanghang Tong. Final: Fast attributed network alignment. In *KDD*, pages 1345–1354. ACM, 2016.

[ZTT+17] Si Zhang, Hanghang Tong, Jie Tang, Jiejun Xu, and Wei Fan. ineat: Incomplete network alignment. In *ICDM*, pages 1189–1194, 2017.

[ZTX+19] Si Zhang, Hanghang Tong, Jiejun Xu, Yifan Hu, and Ross Maciejewski. Origin: Non-rigid network alignment. In *IEEE International Conference on Big Data (Big Data)*, 2019.

[ZWW09] Tian Zhu, Bin Wu, and Bai Wang. Social influence and role analysis based on community structure in social network. In *ADMA*, pages 788–795. Springer, 2009.

[ZWX+18] Zhen Zhang, Mianzhi Wang, Yijian Xiang, Yan Huang, and Arye Nehorai. Retgk: Graph kernels based on return probabilities of random walks. In *NeurIPS*, 2018.

[ZXW+19] Zhen Zhang, Yijian Xiang, Lingfei Wu, Bing Xue, and Arye Nehorai. KerGM: Kernelized Graph Matching. pages 3330–3341, 2019.

[ZY15] J. Zhang and P. S. Yu. Multiple anonymized social networks alignment. In *ICDM*, pages 599–608, 2015.

[ZYZ+20] Jiong Zhu, Yujun Yan, Lingxiao Zhao, Mark Heimann, Leman Akoglu, and Danai Koutra. Generalizing graph neural networks beyond homophily. *arXiv preprint arXiv:2006.11468*, 2020.