Graph Summarization Meets Representation Learning for Scalable Feature Summarization: Methods and Applications

by

Di Jin

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
2021

Di Jin

dijin@umich.edu

ORCID iD: 0000-0001-8028-0556

ACKNOWLEDGEMENTS

First of all, I want to express my deep thankfulness to Prof. Danai Koutra, who has enlightened my career and taught me step by step to be a researcher. Danai inspired my interests in research when I was a newbie conducting projects under her advisory at CMU, and later on she encouraged me to pursue my PhD. In the past 5 years, Danai has provided me with comprehensive academic guidance that includes the flexibility to formulate new problems, high-level direction when the project is stuck, hands-on suggestion on experimental design and paper expression such as illustration and writing. Beyond research, she assisted to expand my professional career by connecting me with researchers from different groups/industry, which contributes to this thesis work. She was also mentally and financially supportive during the hard time of the pandemic. I hope that I could be the kind of researcher and mentor to others that Danai has been to me someday in the future.

I would like to thank the other members of my committee, H.V. Jagadish, Rada Mihalcea and Lei Ying, for their insightful feedback at my proposal and leading up to my defense. I am additionally grateful to Ryan Rossi for consistently mentoring me during my two great summer internships at Adobe Research, which results in a long-term engagement with Adobe and various joint works, some of which are part of this thesis.

Over the years, it has been such a privilege to work with my collaborators during my internships, from whom I have gained a greater and more diverse vision of research. Specifically, I am indebted to Ryan Rossi and Sungchul Kim at Adobe Research for patiently guiding me in terms of conducting research projects, doing survey, writing papers, and filing patents. Particularly, I have learned the spirit of persistence from our experience in working on a project that went through 4 rejections and finally got accepted. I would like to thank Bunyamin Sisman and Luna Dong at Amazon Research for mentoring me during another wonderful summer internship that gave me the chance to explore my expertise in handling real-world research challenges, and for creating the opportunity to exchange ideas with people from different backgrounds and collaboratively work on relevant topics. I could not have made the next step of my professional career without your generous help. I am thankful to Ying Shan for introducing me to the field of multi-modal fusion and for providing me with all the support to formulate and solve the new research problem. I am also grateful to other researchers and interns for the constructive discussion over the collaborative projects, in particular Eunyee Koh at Adobe Research, John Boaz Lee at facebook, Christos Faloutsos, Hao Wei and Jialong Han at Amazon Research, Yuan Yang at Georgia Tech, Zhongang Qi and Yingmin Luo at Tencent, as well as Wei Lee and Lindsay Snider at Trove.

It has also been such a pleasure to work with my labmates of the GEMS Lab. I am indebted to Mark Heimann, Fatemeh Vahedian, Tara Safavi, Yujun Yan, and Jiong Zhu for the privilege of coauthoring with you—I have learned a lot from your expertise and I could not have written this thesis without your talents and generous contribution. I am thankful to Caleb Belth, Marlena Duda, Puja Trivedi and Alican Büyükçakır for kindly sharing your thoughts on my work and paper: your suggestions have always inspired me to improve my work. I am also thankful to the undergraduate and masters' students I have been able to work with, particularly Junchen (Mark) Jin, Ruowang Zhang, Ruiyu Li, Haoming Shen, Aristotelis Leventidis, Junyue Wu, and Charles Wang: your impressive passion made our work viable and taught me a lot.

Conducting research amid the environment of pandemic and political tension is challenging. As an international student, I am grateful to people from the CSE department, the international office and Rackham for helping me solve all tough administrative issues. In particularly, I am thankful to our department coordinators, Ashley Andreae and Karen Liska for the patient support. I am also greatly thankful to my friends for the mental support to pull me out of depression.

Finally, I want to thank my parents and brother for being there to support me not to give up, and for enlightening me the true value of life. With your selfless love, I could embrace the challenges in the future with endless courage.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

Graphs are ubiquitous as they naturally capture interactions between entities, such as user interaction in online social media, paper citations in bibliographic networks, and user-product preferences in sales networks. Recently, graph representation learning has gained significant popularity in both academia and industry thanks to its state-of-the-art performance in a variety of downstream machine learning (ML) tasks, such as friend recommendations and anomaly detection. Specifically, node representation learning (embedding) aims to find a dense vector of rich latent features per entity that can be used in ML tasks. However, these dense representations with fixed dimensions come with computational and storage challenges for real-world graphs with many millions or billions of nodes, and the "black-box" nature of the latent features impedes interpretability. On the other hand, graph summarization aims to find a concise and interpretable representation of the original graph that describes its key information, but it is often lossy and trades off space and performance in ML tasks.

In this thesis, we bridge the two lines of research, node embedding and graph summarization: we introduce scalable methods for generating summaries of latent or non-latent (original) node features that achieve the state-of-the-art performance on ML tasks while requiring significantly reduced storage and supporting interpretability. Specifically, we introduce a new problem, *latent network summarization*, which summarizes the graph *structural features* in static networks as latent node embeddings for storage and query efficiency, and extend this idea to incorporate *temporal proximity* in temporal summaries of continuous-time dynamic networks. We also perform an extensive systematic study of temporal summaries and show that they capture the graph structure and temporal dependency at least as well as recently-proposed dynamic embedding approaches, while having significantly less complexity (*i.e.*, no transitional or latent variables). Unlike methods that are based on complex models as "black boxes", our temporal summaries are easy-to-understand, which motivates their usage for practitioners in predictive applications. Finally, we summarize the non-latent graph features by modeling *feature importance* as the high-level knowledge through traditional and deep learning models that can be used for graph analysis and transfer learning. Throughout the thesis, we demonstrate the effectiveness, scalability and space efficiency of our methods on industrial applications such as entity linkage, user stitching, professional role inference, and temporal link prediction, and present insights that can inform further methodological development and applications.

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, Army Young Investigator Award No. W911NF1810397, the Advanced Machine Learning Collaborative Grant from Procter & Gamble, 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.

CHAPTER I

Introduction

Graphs, also known as networks, are defined as the collection of entities and their interactions. As the natural representation for data whose underlying structure is non-Euclidean (*i.e.*, data that cannot be sensibly modeled in the n-dimensional Euclidean space $\mathbb{R}^n$), graphs have been widely used to model real-world complex systems. For example, online social media (*e.g.*, LinkedIn, Snapchat) comprise user profiles and their relationships such as professional communication or friendships [JHS+19]; collections of bibliography such as DBLP and ACM Digital Library. consist of authors and papers connected through citation and collaboration relationships [LKF05]; connectomes consist of neural connections that reflect the co-activities of different areas in the brain [SSTZ12]. Studying the dependencies/relationships of entities as captured in graphs (rather than assuming i.i.d. data) is critical to user modeling, personalization [EV03, DDGR07], or machine learning tasks such as clustering, classification, and more [LSK15, AW10, XYFS07, FZX12, LHH+14, ZGY+16, His18]. General graph mining techniques aim to extract and discover the key patterns in large real-world graph datasets, such as finding patterns and correlations between entities to predict outcomes, or finding anomalies to provide data cleaning and human interpretability [RLU14].

Real-world graph data is growing fast. For example, a study in 2019[1] shows that for the Internet, there are 3.7 billion users with 7.5% growth rate. For social networks, there are 0.2 billion monthly active users from Facebook, and there are 1.4 million new user profiles being created per minute. To handle such high volumes of data and understand the key information, graph summarization approaches are normally adopted [LSDK18]. The goal of

[1]Statistics source: https://www.smartinsights.com/ecommerce/social-commerce/

graph summarization is to extract the a summary graph or a set of structures or a compressed data structure to concisely describe the given graph, and can be used for various data mining and machine learning tasks. Most existing works in the field provide summaries by either grouping "similar" nodes/edges or sampling specific nodes/edges to reduce graph complexity, and often information loss is incurred. Therefore, graph summarization techniques usually trade off storage and runtime with performance.

Recently, node representation learning (*node embedding*) has become a popular paradigm in both academia and industry that often achieves the state-of-the-art performance in tasks such as user modeling, personalization and link prediction [TQW+15a, CLX16, GL16, HYL17c, GF18]. The main idea is to learn a mapping that encodes nodes into a low-dimensional latent space $\mathbb{R}^K$, where K is a constant (*e.g.*, often 128) that is significantly smaller than the input graph size. The projected latent features are denoted as Euclidean vectors that measure some notion of similarity between the nodes in the original graph, such as proximity in the graph, or structural similarity (see Chapter II for more details), which facilitate a wide range of machine learning tasks such as node classification and link prediction [RJK+19, JHJK21]. While node embedding has been widely studied and applied, this line of research has several challenges due to the latent nature of the learned features, including:

- **Storage inefficiency**. The dense node-wise latent feature vectors (embeddings) in the same fixed dimension is storage inefficient. Storing the node embeddings often requires at least $10\times$ more storage than the input sparse graph data.

- **Query inefficiency**. Since the node embeddings are vectors of real numbers, the computational complexity of pairwise comparison is $O(n^2)$, which is very expensive compared to binary vector comparisons.

- **Interpretability**. The node embeddings are hard to understand, as they map to latent, automatically learned features, which are unknown functions of (more interpretable) nodes and graph properties (*e.g.*, node proximity, structural roles).

The high-level goal of this thesis is to bridge node embedding and graph summarization in order to maintain the promising performance in machine learning tasks while addressing the above challenges.

1.1 Research Goal

At the first glance, graph summarization and node embedding seem to be entirely different tasks. Indeed, these two tasks study various perspectives of the graph data with different purposes. For example, comparing with graph summarization that aims to provide the graph-level understanding about the data, node embedding focuses on more fine-granularity analysis by modeling the components of the graph, *i.e.*, nodes and edges. Table 1.1 gives a detailed comparison between graph summarization and node embedding with respect to five dimensions.

Table 1.1: Comparison between graph summarization and node embedding.

	Graph summarization	**Node embedding**
Goal	Extract key information depending on applications such as query answering, compression, or data understanding	Preserve proximity or structural information for computational machine learning tasks
Output	Concise graph-level representation or a set of graph structures *e.g.*, supergraph or simplified graph with flexible sizes	Node- or edge-level representation, *i.e.*, node- or edge-wise vectors in fixed dimension
Core techniques	Grouping, partitioning, simplification	Graph-based walks, matrix factorization, low-dimensional approximation
Applications	Efficient query answering, visualization, data compression, community detection, influence analysis	User modeling, entity linkage, recommendation, personalization, link prediction, clustering

However, from the high level, summarization and embedding are implicitly connected. The extracted features of each line of research are mostly related and can be used interchangeably. For example, summarization considers the structures of graph egonets to produce the representative patterns (such as graphlets or motifs), and such information has been widely used for structural-based node embedding. Another example is that community detection techniques in summarization often rely on walk-based approaches such as random walk to detect the graph communities [PL05], and this technique lies in the heart of node embedding based on graph proximity. As each line of research has its own advantages and disadvantages, the research goal of this thesis is to bridge the two lines of research (graph summarization and node representation learning) by deriving a compressed representation that excels in ML tasks while addressing the above three challenges of latent node features through fea-

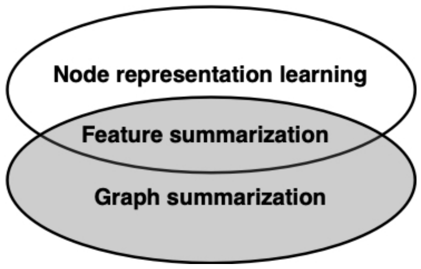


Figure 1.1: The relation between feature summarization, node embedding and graph summarization.

ture summarization. Figure 1.1 depicts the relation between feature summarization, node embedding and graph summarization.

Formally, we give the problem definition as follows.

Problem 1 (Feature summarization)**.** *Given a graph with or without node features, feature summarization aims to derive a compressed representation of the original and/or structural features such that it can perform well in ML tasks, while addressing the incurred challenges,* i.e.*, the storage inefficiency, query inefficiency and interpretability.*

Our solution reconsiders the inner connection between graph summarization and node embedding techniques despite the fact that their main goals or applications vary significantly. The key of such connection is graph features, as they lie at the heart of both summarization and machine learning tasks. Graph summarization mostly looks into the "non-latent" features[2] as the basis to group nodes or edges, for example, PageRank or betweenness in the node surrounding network substructure (k-hop neighborhoods). In contrast, node embedding projects the graph structures (*e.g.*, proximity or structural equivalence) or external information from various sources (*e.g.*, node-wise descriptive texts or other prior knowledge) onto the enriched latent feature space, which results in superior performance in machine learning tasks. To bridge these two tasks, our methods mainly consist of two parts: the feature extraction phase where we characterize graph nodes with features that are used in both fields, and the summarization phase where we derive the high-level knowledge as the summary by compressing the features extracted in the previous phase. In this process, the derived

[2]In this thesis, we use the term "non-latent features" to describe the original features (such as textual description) and graph structural features (such as degree and PageRank), and to distinguish from features that are projected into the latent space.

summary naturally follows the transfer learning paradigm, which is an important application of our methods.

A particular consideration of the work in this thesis is storage efficiency, which is important for scaling to large datasets, query efficiency, and method inductiveness. The purpose of node embedding is to represent nodes in the input graph G through vectors with dimension K. While this representation reduces the input from $N \times N$ adjacency matrix to $N \times K$, the output forms a *dense* matrix comprising continuous real-values that could take more storage than the input *sparse* graph. For example, by following the conventional setting of $K = 128$ for the dimensionality, a graph consisting of 2 million nodes requires roughly 1GB for its embeddings, while the sparse graph file (adjacency matrix) takes only $\sim$ 90MB in storage[3]. For real-world large graphs with billions of nodes, the requirement for storage would increase significantly. Such high volume of storage is a burden to machine I/O operations that could affect the computational performance of downstream tasks. Unfortunately, these issues are rarely tackled by the existing work in representation learning. In summary, node embedding is mostly adopted to generate the latent representation that describes the fine-grained graph components, and thus is advantageous to machine learning tasks but disadvantageous in terms of storage or interpretability. Graph summarization, on the other hand, is used to extract the high-level interpretable key information of the original graph for compression and query efficiency, but is often information-lossy, thus being disadvantageous in terms of machine learning tasks.

Motivated by the needs of storage efficiency, query efficiency, and interpretability, this thesis contributes a range of space- and time-efficient node embedding methods by summarizing the latent and non-latent graph features. These methods aim to derive node embeddings that require significantly reduced storage space with trivial performance trade-off, or with even better performance in machine learning tasks. Additionally, our methods are designed on various types of graph data, which include static homogeneous and heterogeneous graphs, temporal graphs, and knowledge graphs.

[3]https://snap.stanford.edu/data/roadNet-CA.html

1.2 Overview

This thesis is organized as follows: the preliminary material is presented in Chapter II. The representation learning via summarizing the latent features is given from Chapter III to Chapter V, Chapter VI and Chapter VII describe summarizing the non-latent features. Each part addresses node representation learning by incorporating one or more particular properties from graph summarization, such as compression, query efficiency or interpertability, and discusses the applications on real-world data. We summarize these parts in more detail in Table 1.2.

Table 1.2: Thesis Overview.

Feature Type	Graph Type	Objective & chapter	Challenges Addressed	Venue
Latent	Static heterogeneous	**Introduce** latent network summarization that aims to learn a compact, latent representation of the graph structure (Ch. III)	Storage inefficiency	KDD'19 [JRK+19]
	Temporal heterogeneous	**Extend** summarizing graph structural features to the temporal settings (Ch. IV)	Query inefficiency Pairwise comparison	PKDD'19 [JHRK19]
		Establish a framework to systematically study the temporal graph proximity and dependency of temporal graphs (Ch. V)	Model interpretability	WSDM'22 [JKRK22]
Non-latent	Static homogeneous	**Analyze** graph data by selecting diverse, concise and domain-specific features (Ch. VI)	Model and result interpretability	ICDM'17 [JK17]
	Knowledge graph	**Learn** feature importance of relational data from multiple sources as the generic transferable knowledge for entity linkage (Ch. VII)		VLDB'22 [JSW+22]

1.2.1 Latent Feature Summarization

Motivated by the computational and storage challenges that dense embeddings pose, Chapter III introduces the problem of *latent network summarization* that aims to learn a compact, latent representation of the graph structure with dimensionality that is *independent* of the input graph size (*i.e.*, #nodes and #edges), while retaining the ability to derive node representations on the fly. We propose the solution Multi-Lens, an inductive multi-level latent network summarization approach that leverages a set of *relational operators* and *relational functions* (compositions of operators) to capture the structure of egonets and higher-order subgraphs, respectively. The structure is stored in low-rank, size-independent structural feature matrices, which along with the relational functions comprise our latent network summary. Multi-Lens is general and naturally supports both homogeneous *and* heterogeneous graphs with or without directionality, weights, attributes or labels. As application areas, we show the effectiveness of Multi-Lens in detecting anomalies and events in the Enron email communication graph and Twitter co-mention graph. Exploiting graph structural features in node-centric subgraphs produces the high-level knowledge to characterize individual nodes, which can be further used to measure the similarity between nodes that are not connected or far away in the graph. We refer the interested readers to our KDD 2019 paper that learns structural embeddings for professional role inference in weighted and directed graphs [JHS+19].

Having established the feasibility of latent feature summarization that is based on graph structural features, we extend the idea to handle temporal graph data since most real-world graphs naturally evolve over time. In Chapter IV, we introduce a hashing-based technique to summarize the latent graph features and the temporal proximity. We target the real-world task of identity stitching that aims to identify and match various online references (*e.g.*, sessions over different devices and timespans) to the same user in real-world web services. Traditional user stitching approaches, such as grouping or blocking, requires pairwise comparisons between a massive number of user activities, thus posing both computational and storage challenges. To solve the problem in an application-independent way, we describe a solution called Node2bits, an efficient framework that represents multi-dimensional features of node contexts with binary hashcodes. Node2bits takes a heterogeneous network-based approach in which users interact with content (*e.g.*, sessions, websites), and may have attributes (*e.g.*, location). More importantly, Node2bits leverages *feature-based temporal* walks to encapsulate short- and long-term interactions between nodes in heterogeneous web networks, and adopts SimHash to obtain compact binary representations and avoid the quadratic complexity for similarity search. Thus, the output feature summary incorporates the temporal proximity into the summaries, as well as the graph structural information, with significantly reduced storage requirement.

While novel temporal embeddings have been widely studied especially in deep learning, their "black-box" nature impedes people from fully understanding their effectiveness in modeling the temporal patterns. In Chapter V, we aim to fill this gap by introducing a general framework that summarizes the graph temporal proximity and dependency through interpretable temporal network models that are built atop the graph time-series representations. Our framework generalizes static node embeddings derived from the time-series representation of stream data to the dynamic setting by modeling the temporal dependencies with classic models, such as reachability graphs. We conduct a systematic study of *seven* base static embedding methods and *six* temporal network models, and evaluate the model performance via temporal link prediction. Out of the 42 methods that our framework subsumes, we find that our proposed graph time-series representation, which fixes the numbers of edges in snapshots, tends to perform the best. More importantly, we find that our framework can generalize static node embeddings, achieving comparable or better performance than the studied state-of-the-art dynamic embedding approaches, indicating that our interpretable temporal summary framework could capture the graph structures and temporal dependency at least as well as those recent dynamic approaches, but with less complexity.

1.2.2 Non-latent Feature Summarization

Chapter III-V develop new methods to summarize the latent features. However, the non-latent graph features can be summarized as well in order to provide interpretability or derive efficient node embeddings for various machine learning tasks.

Feature selection is a representative approach that simplifies a dataset by choosing features that are relevant to a specific task, such as classification, prediction, and anomaly detection. In Chapter VI, we introduce EAGLE (Exploratory Analysis of Graphs with domain knowLEdge), a novel method that creates interpretable, *feature-based*, and *domain-specific* graph summaries in a fully automatic way. That is, the same graph in different domains—e.g., social science and neuroscience—will be described via different EAGLE summaries, which automatically leverage the respective domain knowledge and expectations. We propose an optimization formulation that seeks to find an interpretable summary with the most representative features for the input graph so that it is: *diverse*, *concise*, *domain-specific*, and *efficient*.

Chapter VII follows the idea of summarizing features by modeling their importance and extending the summaries to handle a more complicated task: transfer learning for multi-source entity linkage. Multi-source entity linkage focuses on integrating knowledge from multiple sources by linking the records that represent the same real world entity. The state-of-the-art entity linkage pipelines mainly depend on supervised learning that requires abundant amounts of training data. However, collecting well-labeled training data becomes expensive when the data from many sources arrives incrementally over time. Moreover, the trained models can easily overfit to specific data sources, and thus fail to generalize to new sources due to significant differences in data and label distributions. To address these challenges, we present AdaMEL, a deep transfer learning framework that learns generic high-level knowledge to perform multi-source entity linkage. AdaMEL models the attribute importance that is used to match entities through an attribute-level self-attention mechanism, and leverages the massive unlabeled data from new data sources through domain adaptation to make it generic and data-source agnostic. In addition, AdaMEL is capable of incorporating an additional set of labeled data to more accurately integrate data sources with different attribute importance.

1.3 Contributions

This thesis contributes to the research fields of graph summarization and node embedding:

New problem and definition We introduce the new problem of scalable feature summarization for graph representation learning. In Chapter III, we give the formal definition of latent network summarization for structural features, which is the first such work in the field. We extend this idea to summarize the structural features with temporal proximity in Chapter IV. In Chapter V, we show that these works yield comparable and even better performance with reduced computational complexity.

New embedding methods for various graph types We propose embedding approaches on graph data with various complexity, including static heterogeneous (Chapter III) and homogeneous graphs (Chapter VI), temporal graphs (Chapter IV- V), and multi-relational knowledge graphs (Chapter VII). We evaluate our work on various machine learning tasks over different graph types, such as temporal link prediction, graph classification, and transfer learning across data sources in order to demonstrate their effectiveness and generality across complex multi-relational graphs.

Practical solutions to real-word problems My thesis works are motivated by real-world tasks that occur in industry, and they are designed to handle various critical challenges brought on by real-world data. For example, Node2bits is motivated by user stitching (Chapter IV), and AdaMEL is motivated by multi-source entity linkage (Chapter VII). Both works address real-world data challenges, such as storage limitation and various data distribution across data sources, which are rarely seen in public benchmark datasets but occur often in real applications. Additionally, we identify best practices for modeling temporal dependencies of snapshots in temporal graphs and provide our findings & ideas for future research works as the reference (Chapter V).

Other Impact Our works on latent feature summarization (Chapter III and Chapter V) were developed in conjunction with industry collaborators and applied by Adobe Research to handle intustrial problems. We also have two pending patents. Moreover, our entity linkage method AdaMEL (Chapter VII) was also accepted and presented in Amazon Machine Learning Conference, an application-oriented conference for scientists and practitioners from industry with acceptance rate $\sim 10\%$.

CHAPTER II

Preliminaries & Related Work

In this chapter, we first present preliminary definitions that are widely used in graph analysis and node embedding. We also provide an overview of two important notions defined on networks — communities and roles — as well as their connections to two categories of node embedding approaches proximity-based and structure-based [RJK+19]. We summarize the general symbols that are used throughout the thesis in Table 2.1, and describe the additional symbols that are specific to each work in the corresponding chapters. Second, we summarize the related work on graph representation learning and graph summarization.

Table 2.1: Summary of general symbols and notations throughout the thesis.

Symbol	Definition
$G = (V, E)$	network with $\|V\| = N$ nodes and $\|E\| = M$ edges
$\mathbf{A}$	adjacency matrix of G with row i $\mathbf{A}_{i,:}$ and column i $\mathbf{A}_{:,i}$
F	size of feature space
λ	regularization parameters
$s(,), d(,)$	similarity and distance between two objects, resp.

2.1 Preliminaries

2.1.1 Graphs

The general definition of a graph is give as follows:

Definition 1 (Graph)**.** *A graph* $G(\mathrm{V}, \mathrm{E})$ *is a collection of* $|V|$ *nodes (vertices) and* $|E|$ *edges (relations). The adjacency matrix associated with the graph* G *is conventionally denoted as* $\mathbf{A}$.

A **heterogeneous network** is defined as $G = (V, E, \theta, \xi)$ with node-set V, edge-set E, a function $\theta : V \rightarrow \mathcal{T}_V$ mapping nodes to their types, and a function $\xi : E \rightarrow \mathcal{T}_E$ mapping edges to their types. Furthermore, we denote a **continuous-time dynamic, heterogeneous network** as $G = (V, E_\tau, \theta, \xi, \tau)$, where E_τ represents the set of temporal edges between vertices V and $\tau : E \rightarrow \mathbb{R}^+$ is a function that maps each edge to a corresponding timestamp. In most cases, when graph homogeneity/heterogeneity is not specified, the continuous-time dynamic graph (temporal graph for short) is denoted as follows:

Definition 2 (Temporal Graph). *Let V be a set of vertices, and $E \subseteq V \times V \times \mathbb{R}^+$ be the set of temporal edges between vertices in V. Each edge (u, v, t) has a timestamp $t \in \mathbb{R}^+$.*

When edges represent contacts—*e.g.*,a phone call or physical proximity—between two entities at a specific point in time, we have an evolving network structure [BF03]. A temporal walk in such a network represents a sequence of contacts that obeys time. That is, if each edge represents a contact between two entities, then a path represents a feasible route for a piece of information.

Definition 3 (Temporal Walks). *A temporal walk from u to w in $G = (V, E)$ is a sequence of edges $e_1, \ldots, e_k$ such that $e_1 = (u_1, u_2, t_1), \ldots, e_k = (u_k, u_{k+1}, t_k)$ where $t_j < t_{j+1}$ for all $j = 1$ to k. We say that u is temporally connected to w if there exists such a temporal walk.*

This definition echoes the standard definition of a path, but adds the additional constraint that paths must respect time, i.e., follow the directionality of time. Temporal walks are inherently asymmetric because of the directionality of time. The notion of temporal walks has been recently used in embedding methods [NLR$^+$18].

2.1.2 Communities and Roles

In a graph, communities are sets of nodes with more connections inside the set than outside based on proximity/closeness or density, whereas roles define sets of structurally similar nodes that are more similar to nodes inside the set than outside. Communities and roles are fundamentally different but important complementary notions. More formally, they are defined as follows:

Definition 4 (Communities)**.** *Communities are* dense, cohesive subsets *of vertices* $\mathcal{C} = \{C_1, \ldots, C_k\}$*. A community* $C_i \subseteq V$ *is "good" if the induced subgraph is dense (* i.e.*, there are many edges between the vertices in* C_i*) and there are relatively few edges from* C_i *to other vertices* $\bar{C}_i = V \setminus C_i$ *[Sch07].*

Definition 5 (Roles)**.** *Roles define sets of nodes that are more structurally similar to nodes inside the set than outside [RA15b]. The terms role and position are used synonymously.*[1]

The term "structurally similar" refers to nodes that have similar structural properties, *e.g.*, the set of nodes might be hubs (star-centers) or bridge-nodes (gatekeepers) that connect different communities.

2.1.3 Node Embeddings

In 2000s, node embedding was proposed for dimensionality reduction of non-relational data so that the higher dimensional features are represented in low-dimensional manifolds [BN02, TDSL00]. Many relevant works are based on the pairwise similarity between node features and graph spectrum (*i.e.*, eigenvalues of graph adjacency matrix or Laplacian matrix), both of which are computationally expensive. On the other hand, state-of-the-art approaches take the graph as input, and learn representations that encode the graph structural information. Figure 2.1 illustrates a toy graph and its node embeddings.

One approach to encode the graph structure is to embed node proximity in the representation [GL16, TQW$^+$15b, PARS14]. Intuitively, a node is more likely to be similar to its 1-hop neighbors than its 2-hop neighbors, or other distant nodes in the graph. As a result, nodes that share common neighbors are embedded more closely. Figure 2.1a shows an example. Node A is incidental to node B so their embeddings — based on proximity — should be close. Similarly, the embedding of node C should also be close to B, but since A is 2-hop away from C, the embedding of A is closer to B than C (as shown in

[1]An equivalent definition of role is in terms of a role assignment function $r : V \rightarrow R$ that maps nodes to a set of roles R. The role assignment function r induces a partition $\mathcal{C} = \{C_1, \ldots, C_k\}$ of V by taking the inverse-images as sets/classes of nodes that play/have the same role. Further, if $\sim$ is an equivalence relation (binary relation on V that is reflexive, symmetric, and transitive), then the set of its equivalence classes is a partition of V (and conversely). Hence, it is equivalent to think of a role as a set of nodes (node partition), function (role assignment), or equivalence relation on V since these are different but equivalent mathematical formulations for the concept of roles.

Figure 2.1b). There is another class of embedding approaches that is based on the local graph structure [RSF17, DZHL18, HSSK18]. These approaches consider the functionality or sturctural role of a node in terms of its connection to the neighbors. For example, the centers of two star-like subgraphs are structurally similar to each other because they play the role of "hubs" that bridges the other nodes. Embedding approaches of this type aim to preserve the structural similarity between nodes. Figure 2.1c depicts exemplary node embeddings based on structural similarity. It can be seen that the periphery nodes (A, C, E) are embedded closer to each other, even though they are more distant in proximity. Proximity-based and structure-based node embedding approaches are fundamentally different but effective in different scenarios depending on specific application.

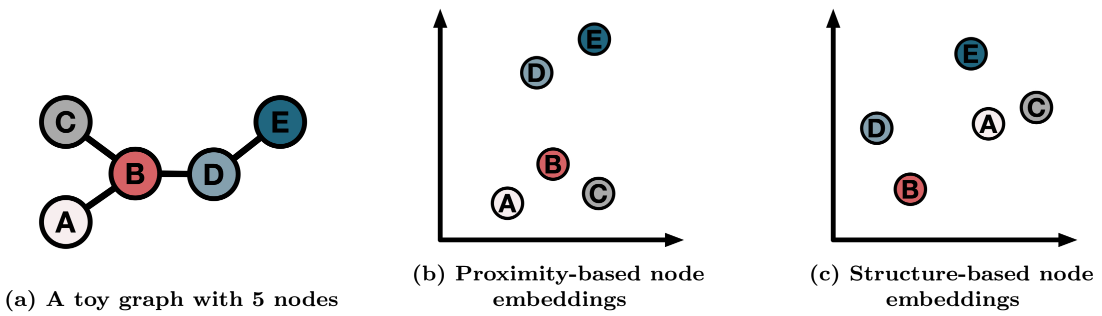

C
B
D
E
A
(a) A toy graph with 5 nodes
(b) Proximity-based node embeddings
(c) Structure-based node embeddings

Figure 2.1: A toy graph and its 2-D embeddings visualized using TSNE.

Many recent works in node embedding derive node features based on the same mechanisms that are used to derive communities (*e.g.*, random walks, feature diffusion), or roles (*e.g.*, graphlets, feature-based matrix factorization). As a result, the latent feature vectors or embeddings given as output from an embedding method can be thought of as either community [HERPF10] or role membership vectors (assuming proper normalization) [ABFX08, RGNH13, HGER$^+$12]. Indeed, many works claim to preserve the notion of communities [CZC$^+$17], roles [RSF17, RAK$^+$18], or even both [GL16]. In this light, recent embedding methods can be seen as approaches for modeling communities or (feature-based) roles [RA15b]. We refer the interested readers to our paper [RJK$^+$19] that was accepted at TKDD 2020 for more details.

2.2 Related Work

In this section, we review the literature from two research areas that are relevant to our work: node embedding and graph summarization. We also qualitatively compare our works to both embedding and summarization methods in Table 2.2. We provide additional related works for the specific topic of node embedding on temporal graphs in Chapter V, and the application for node embedding on entity linkage in Chapter VII.

2.2.1 Node embedding

Node embedding or representation learning has been an active area which aims to preserve a notion of similarity over the graph in node representations [GF18, RZA18]. For instance, [PARS14, TQW$^+$15b, GL16, CLX16, DCS17] define node similarity in terms of proximity (based on the adjacency or positive pointwise mutual information matrix) using random walks (RW) or deep neural networks (DNN). Another class of approaches captures similar node behavioral patterns (roles) or structural similarity [RA15c, ARZ$^+$18, HSSK18, RZA18, JHJK21]. For instance, struc2vec and xNetMF [RSF17, HSSK18] define similarity based on node degrees, while DeepGL [RZA18] learns deep inductive relational functions applied to graph invariants such as degree and triangle counts. [LG14, QDM$^+$18] investigate theoretical connection between matrix factorization and the skip-gram architecture. Rolx [HGER$^+$12] extracts structural information from general network datasets and categorizes nodes through non-negative matrix factorization as 'main-stream' or 'bridge'. To handle heterogeneous graphs, metapath2vec [DCS17] captures semantic and structural information by performing RW on predefined metapaths. role2vec [ARZ$^+$18] proposes a framework for inductive learning by defining attributed RW atop relational operators. There are also works based on specific characteristics in heterogeneous graphs. For example, [GLT$^+$16] proposes to embed heterogeneous networks with hyperedges that represent interactions among involving objects in events. Another work, AspEm represents underlying semantic facets as multiple "aspects" and selects a subset to embed based on datasetwide statistics. There are also works seeking to propagate node features through neural networks. For example, the propagation rule of GCN [KW17] and the mean aggregators used in GraphSAGE [HYL17b].

Table 2.2: Qualitative comparison of the thesis works to existing embedding and summarization methods. Does the method: handle heterogeneous/temporal graphs; yield an output that is size-independent, but node-specific, and representations that are independent of node proximity; support inductive learning and scale well (i.e., subquatratic on the network size)?

	Input		Representations / Output			Method	
	Hetero-geneity	**Temp-oral**	**Size indep.**	**Node specific**	**Proxim. indep.**	**Scalable**	**Induc.**
Aggregation [BGLL08]	✓	✗	✗	✗	✗	✓	✗
Cosum [ZGGS+16]	✗	✗	✗	✗	✓	✗	✗
AspEm [SGZ+18]	✓	✗	✗	✓	✗	✓	✗
metapath2vec [DCS17]	✓	✗	✗	✓	✗	✓	✗
LINE [TQW+15b]	✗	✗	✗	✓	✗	✓	✗
n2vec [GL16]	✗	✗	✗	✓	✗	✓	✗
struc2vec [RSF17]	✗	✗	✗	✓	✓	✗	✗
DNGR [CLX16]	✗	✗	✗	✓	✗	✗	✗
GraphSAGE [HYL17b]	✓	✗	✗	✓	✓	✓	✓
CTDNE [NLR+18]	✗	✓	✗	✓	✓	✓	✗
EAGLE	✗	✗	✓	✗	✓	✓	✗
Multi-Lens	✓	✗	✓	✓	✓	✓	✓
Node2bits	✓	✓	✗	✓	✗	✓	✗

In the field of temporal network embedding, most approaches [ZGY+16, His18] approximate the dynamic network as discrete static snapshots overtime. CTDNE [NLR+18] explores temporal proximity by learning temporally valid embeddings based on a corpus of temporal random walks. Another related field is hashing-based embedding, for example, node2hash [WWGW18] proposes to hash the pairwise node proximity derived from random walk into low-dimensional hashcode as the embeddings. CCTN [LZS+19] embeds and clusters nodes in a network that are not only well-connected but also share similar behavioral patterns (*e.g.*, similar patterns in the degree or other structural properties over time).

2.2.2 Graph Summarization

We give an overview of graph summarization methods, and refer the interested reader to tutorials [LYL13, KBB17] and a comprehensive survey [LSDK18] for more details. Various summarization approaches have been proposed depending on specific applications. Most summarization works fall into 3 categories: (1) aggregation-based which group nodes [NRS08] or edges [MA16]) into super-nodes/edges based on application-oriented criteria or existing clustering algorithms; (2) abstraction-based [SMER06, LL09] which remove less informative nodes or edges; and (3) compression-based [SKZ+15] which aim to minimize the number of bits required to store the input graph.

Aggregation-based methods group nodes/edges into supernodes/superedges based on application-oriented criteria or existing clustering algorithms. [RGM03] proposes the 2-level representation for Web graphs to form supernodes/superedges based on the structural information; [TZHH11] proposes to aggregate weighted graphs by maintaining edge weights up to a certain number of hops; [NRS08] proposes to aggregate graphs through MDL along with corrections to bound errors for recovery. [RGSB17, SWD16] leverage aggregated representations to tackle fast query response while the latter work focuses specifically on knowledge graphs. In addition to grouping nodes with similar structures, [BC08] proposes the concept of virtual node to group edges in Web graphs, which supports scalable pattern mining; [MA16] extends this idea to compresses the neighborhood around high-degree nodes and proposes novel graph query processing operations over the compressed data to improve query performance. SNAP/k-SNAP [THP08] proposes aggregating nodes in the graph by allowing users to specify node attributes and relationships. It also allows users to control (drill-down, roll-up) the sizes of the summaries. Zhang el al. [ZTP10] improves SNAP/k-SNAP to address numerical attributes in graphs and introduces a novel interestingness measure to evaluate the quality of a summary. Note that among existing works, few are designed for heterogeneous graphs. CoSum [ZGGS$^+$16] aggregates nodes with respect to both label and structural similarity to aggregate nodes tackle the problem of entity resolution in heterogeneous graphs. Motif [DS13] is a visualization tool that aggregates common network structures or subnetworks and highlight the node types through unique shapes to address heterogeneity. [CLF$^+$09] proposes Summarize-Mine to create randomized summaries from frequently occurring subgraphs in heterogeneous labeled graphs in an iterative manner. While patterns are removed in this process, true patterns are unlikely to be missing from all rounds.

Abstraction-based methods remove less-informative nodes or edges based on given abstraction criteria. OntoVis [SMER06] supports both semantic abstraction by including only nodes whose types are selected by the users and structural abstraction by removing less-essential graph components, e.g., one-degree nodes and duplicate paths. It also support importance filtering based on graph statistics such as node degree. Li et al. [LL09] propose constructing the egocentric abstraction of social networks through edges. The unique k-step linear combinations of relations in the egonet of a node are identified as features, which are adopted as the summaries based on their frequency or rareness.

Compression-based methods represent the networks in a space-efficient way based on the semantic and structural information of the graph. Kang et al. propose SlashBurn [LKF14], a node reordering method that achieves compression-friendly by identifying the “hub” and “spoke” structures in the graphs; VOG [KKVF15] leverages SlashBurn to extract egonetworks and other disconnected components and constructs a graph vocabulary (i.e., cliques and near-cliques, stars, etc.) to label them through MDL as the model selection criterion. Shah et al. [SKZ$^+$15] extend VOG in temporal graph analysis by identifying the temporal behaviors of local static structures that collectively minimize the global description length of the dynamic graph. Moreover, temporal behaviors (e.g., flickering, periodic, one-shot) are introduced to augment the vocabulary of static graphs.

Part I: Node Embedding via Latent Feature Summarization

CHAPTER III

Latent Structural Feature Summarization for Static Heterogeneous Graphs

This chapter is based on work that appeared at KDD 2019 [JRK$^+$19].

3.1 Introduction

Our first work addresses the storage inefficiency challenge of node embeddings indicated in Chapter I. Recent advances in representation learning for graphs have led to a variety of proximity-based and structural embeddings that achieve superior performance in specific downstream tasks, such as link prediction, node classification, and alignment [GF18, RZA18, HSSK18]. At the same time though, the learned, K-dimensional node embeddings are dense (with real values), and pose computational and storage challenges especially for massive graphs. By following the conventional setting of $K = 128$ for the dimensionality, a graph of 1 billion nodes requires roughly 1TB for its embeddings. Moreover, this dense representation often requires significantly more space to store than the original, sparse adjacency matrix of a graph. For example, for the datasets that we consider in our empirical analysis, the learned embeddings from existing representation learning techniques require $3 - 48\times$ more space than the original edge files.

To address these shortcomings, we introduce the problem of *latent network summarization*. Informally, the goal is to find a low-dimensional representation in a latent space such that it is *independent* of the graph size, *i.e.*, the number of nodes and edges. Among other tasks, the representation should support *on-the-fly* computation of *specific* node embeddings.

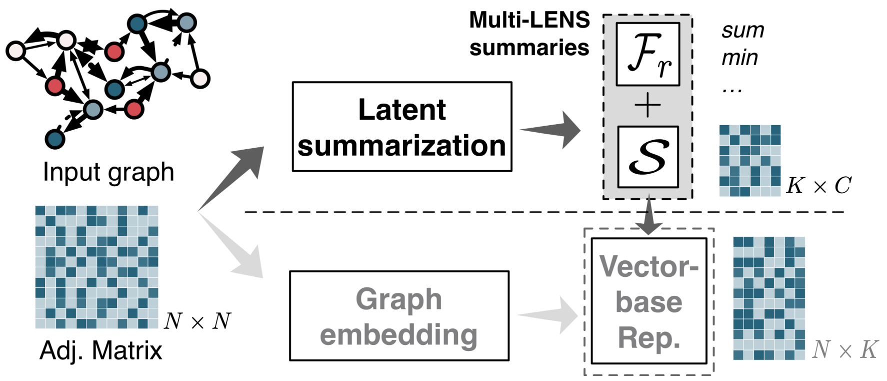


Figure 3.1: Our proposed approach to Latent Network Summarization called Multi-Lens produces a summary consisting of relational functions $\mathcal{F}_r$ and node-independent matrices $\mathcal{S}$ of size $K \times C$. Thus, while embedding methods output N node embeddings of dimensionality K, latent summarization methods produce an output that is independent of N and thus is graph-size independent. Despite not storing the embeddings, Multi-Lens can derive them on the fly.

Latent network summarization and network embedding are *complementary* learning tasks with fundamentally different goals and outputs, as shown in Fig. 3.1. In particular, the goal of network embedding is to derive N node embedding vectors of K dimensions each that capture node proximity or equivalency. Thus, the output is a $N \times K$ matrix that is *dependent* on the size of the graph (number of nodes) [QDM+18, GF18]. This is in contrast to the goal of latent network summarization, which is to learn a *size-independent representation* of the graph. *Latent* network summarization also differs from traditional summarization approaches that typically derive supergraphs (*e.g.*, mapping nodes to supernodes) [LSDK18], which target different applications and are unable to derive node embeddings.

To efficiently solve the latent network summarization problem, we propose Multi-Lens (Multi-level Latent Network Summarization), an inductive framework that is based on graph function compositions. In a nutshell, the method begins with a set of arbitrary graph features (*e.g.*, degree) and iteratively uses generally-defined relational operators over neighborhoods to derive deeper function compositions that capture graph features at multiple *levels* (or distances). Low-rank approximation is then used to derive the best-fit subspace vectors of network features across levels. Thus, the latent summary given by Multi-Lens comprises graph functions and latent vectors, both of which are independent of the graph size. Our main contributions are summarized as follows:

- **Novel Problem Formulation.** We introduce and formulate the problem of *latent* *network summarization*, which is complementary yet fundamentally different from network embedding.

- **Computational Framework.** We propose Multi-Lens, which expresses a class of methods for latent network summarization. Multi-Lens naturally supports inductive learning, on-the-fly embedding computation for all or a *subset* of nodes.

- **Time- and Space-efficiency.** Multi-Lens is *scalable* with time complexity linear on the number of edges, and *space-efficient* with size independent of the graph size. Besides, it is *parallelizable* as the node computations are independent of each other.

- **Empirical analysis on real datasets.** We apply Multi-Lens to event detection and link prediction over real-world heterogeneous graphs and show that it is 3.5%-34.3% more accurate than state-of-the-art embedding methods while requiring 80-2152$\times$ less output storage space for datasets with *millions* of edges.

Next we formally introduce the latent network summarization problem and then describe our proposed framework.

3.2 Latent Network Summarization

Intuitively, the problem of *latent network summarization* aims to learn a compressed representation that captures the main structural information of the network and depends only on the complexity of the network instead of its size. More formally:

Definition 6 (Latent Network Summarization)**.** *Given an arbitrary graph $G = (V, E)$ with $|V| = N$ nodes and $|E| = M$ edges, the goal of latent network summarization is to map the graph G to a low-dimensional $K \times C$ representation $\mathcal{J}$ that summarizes the structure of G, where $K, C \ll N, M$ are independent of the graph size. The output latent representations should be usable in data mining tasks, and sufficient to derive all or a subset of node embeddings on the fly for learning tasks (e.g., link prediction, classification).*

Compared to the network embedding problem, latent network summarization differs in that it aims to derive a *size-independent* representation of the graph. This can be achieved

Table 3.1: **Summary of symbols and notations.**

Symbol	Definition
$\mathbf{A}$	adjacency matrix of G with row i $\mathbf{A}_{i,:}$ and column i $\mathbf{A}_{:,i}$
$\mathcal{T}_V, \mathcal{T}_E$	sets of object types and edge types, respectively
$\mathcal{N}_i, \mathcal{N}_i^t$	non-typed / types (1-hop) neighborhood or egonet of node i
ℓ, L	index for level & total number of levels (*i.e.*, max order of a rel. fns)
$\mathcal{B}$	$=\{\mathbf{b}_i\}$ set of initial feature vectors in length N
$\mathcal{F}_r$	$=\{\mathcal{F}_r^{(1)}, \dots, \mathcal{F}_r^{(L)}\}$, ordered set of relational functions across levels
$\mathcal{F}_b$	$= \{f_{b_i}\}$, set of base graph functions (special relational functions)
Φ	$= \{\phi_i\}$, set of relational operators
$\mathbf{F}^{(0)}$	$N \times \lvert\mathcal{B}\rvert$ base feature matrix derived by the base graph functions $\mathcal{F}_b$
$\mathbf{F}^{(\ell)}$	$N \times \left(\lvert\mathcal{B}\rvert \cdot \lvert\Phi\rvert^{\ell}\right)$ generated feature matrix for level ℓ
$K^{(\ell)}$, K	dimensionality of embeddings at level-ℓ and the final dimensionality
$\mathbf{H}^{(\ell)}$	$N \times \lvert\mathcal{F}_r^{(\ell)}\rvert$ histogram-based representation of feature matrix $\mathbf{F}^{(\ell)}$
$\mathbf{S}^{(\ell)}$	low-rank latent graph summary at level ℓ

in the form of supergraphs [LSDK18] (in the original graph space) or aggregated clusters trivially, but the compressed latent network summary in Definition 6 also needs to be able to derive the node embeddings, which is not the goal of traditional graph summarization methods.

In general, based on our definition, a latent network summarization approach should satisfy the following key properties: **(P1)** generality to handle arbitrary network with multiple node types, relationship types, edge weights, directionality, unipartite or k-partite structure, *etc.* **(P2)** high compression rate, **(P3)** natural support of inductive learning, and **(P4)** ability to on-the-fly derive node embeddings used in follow-up tasks.

3.3 Multi-Lens Framework

To efficiently address the problem of latent network summarization introduced in Section 3.2, we propose Multi-Lens, which expresses a class of latent network summarization methods that satisfies all desired properties **(P1-P4)**. The summary $\mathcal{J}$ given by Multi-Lens contains (i) necessary operators for aggregating node-wise structural features automatically and (ii) subspace vectors on which to derive the embeddings. We give the overview in Figure 3.2. And in addition to symbols listed in Table 2.1, here we list the main symbols and notations used particularly in this work in Table 3.1.

At a high level, Multi-Lens leverages generally-defined relational operators to capture structural information from node neighborhoods in arbitrary types of networks. It recursively applies these operators over node neighborhoods to produce both linear and non-linear functions that characterize each node at different distances (§ 3.3.2). To efficiently derive the contextual space vectors, Multi-Lens first generates histogram-based heterogeneous contexts for nodes (§ 3.3.3), and then obtains the summary via low-dimensional approximation (§ 3.3.4). Before discussing each step and its rationale, we first present some preliminaries that serve as building blocks for Multi-Lens.

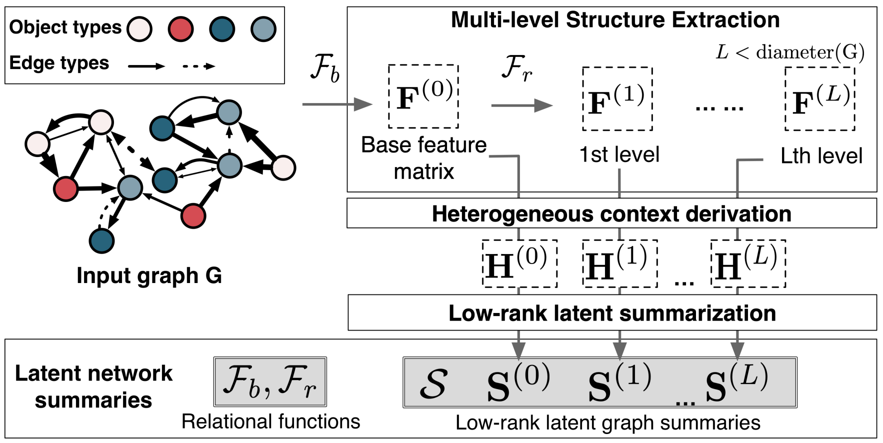

Object types ○ ● ● ●
Edge types → ⇢
Input graph G
$\mathcal{F}_b$
Multi-level Structure Extraction
$L <$ diameter(G)
$\mathbf{F}^{(0)}$
$\mathcal{F}_r$
$\mathbf{F}^{(1)}$
$\mathbf{F}^{(L)}$
Base feature matrix
1st level
Lth level
Heterogeneous context derivation
$\mathbf{H}^{(0)}$ $\mathbf{H}^{(1)}$ ... $\mathbf{H}^{(L)}$
Low-rank latent summarization
Latent network summaries
$\mathcal{F}_b, \mathcal{F}_r$
Relational functions
$\mathcal{S}$ $\mathbf{S}^{(0)}$ $\mathbf{S}^{(1)}$ ... $\mathbf{S}^{(L)}$
Low-rank latent graph summaries

Figure 3.2: Overview of Multi-Lens. Dashed boxes: intermediate results that do not need to store; shaded boxes: outputs that need storing. The size of the latent network summaries, $\mathcal{J} = \{\mathcal{F}, \mathcal{S}\}$, is independent of N, M.

3.3.1 Preliminaries

Recall that our proposed problem definition (§ 3.2) applies to any arbitrary graph **(P1)**. As a general class, we refer to heterogeneous (information) networks or typed networks. We assume that the network is directed and weighted with unweighted and undirected graphs as special cases. For simplicity, we will refer to a graph as $G(V, E)$. Within heterogeneous networks, the typed neighborhood or egonet[1] $\mathcal{N}_t$ of a node is defined as follows:

Definition 7 (Typed neighborhood $\mathcal{N}^t$). *Given an arbitrary node i in graph $G = (V, E)$, the typed t* neighborhood *$\mathcal{N}_i^t$ is the set of nodes with type t that are reachable by following directed edges $e \in E$ originating from i with 1-hop distance and i itself.*

[1]In this work we use neighborhood and egonet interchangeably.

The *neighborhood* of node i, $\mathcal{N}_i$, is a superset of the typed neighborhood $\mathcal{N}_i^t$, and includes nodes in the neighborhood of i regardless of their types. Higher-order neighborhoods are defined similarly, but more computationally expensive to explore. For example, the k-hop neighborhood denotes the set of nodes reachable following directed edges $e \in E$ originating from node i within k-hop distance.

The goal of latent network summarization is to find a size-independent representation that captures the *structure* of the network and its underlying nodes in the latent space. Capturing the structure depends on the semantics of the network (e.g., weighted, directed), and thus different ways are needed for different input networks types. To generalize to *arbitrary* networks, we leverage relational operators and functions [RZA18].

Definition 8 (Relational operator). *A relational operator $\phi(\mathbf{x}, \mathcal{R}) \in \Phi$ is defined as a basic function (*e.g.*, sum) that operates on a feature vector $\mathbf{x}$ associated with a set of related elements $\mathcal{R}$ and returns a single value.*

For example, let $\mathbf{x}$ be an $N \times 1$ vector and $\mathcal{R}$ the neighborhood $\mathcal{N}_i$ of node i. For ϕ being the *sum*, $\phi(\mathbf{x}, \mathcal{R})$ would return the count of neighbors reachable from node i (unweighted out degree).

Definition 9 (Relational function). *A relational function $f \in \mathcal{F}$ is defined as a composition of relational operators $f = \big(\phi_1 \circ \cdots \circ \phi_{h-1} \circ \phi_h\big)(\mathbf{x}, \mathcal{R})$ applied to feature values in $\mathbf{x}$ associated with the set of related nodes $\mathcal{R}$. We say that f is order-h iff the feature vector $\mathbf{x}$ is applied to h relational operators.*

Together, relational operators and relational functions comprise the building blocks of our proposed method, Multi-Lens. Iterative computations over the graph or a subgraph (*e.g.*, node neighborhood) generalize for inductive/across-network transfer learning tasks. Moreover, relational functions are general and can be used to derive commonly-used graph statistics. As an example, the out-degree of a specific node is derived by applying order-1 relational functions on the adjacency matrix over its the egonet, *i.e.*, out-deg(i) $= \sum(\mathbf{A}_{i:}, \mathcal{N})$ regardless of object types.

3.3.2 Multi-level Structure Extraction

We now start describing our proposed method, Multi-Lens. The first step is to extract multi-level strcuture around the nodes. To this end, as we show in Figure 3.2, Multi-Lens first generates a set of simple node-level features to form the base feature matrix $\mathbf{F}^{(0)}$ via the so-called base graph functions $\mathcal{F}_b$. It then composes new functions by iteratively applying a set of relational operators Φ over the neighborhood to generate new features. Operations in both $\mathcal{F}_b$ and Φ are generally defined to satisfy (**P1**).

3.3.2.1 Base Graph Functions

As a special relational function, each base graph function $f_b \in \mathcal{F}_b$ consists of relational operators that perform on an initial $N \times 1$ feature vector $\mathbf{b} \in \mathcal{B}$. The vector $\mathbf{b}$ could be given as the row/column of the adjacency matrix corresponding to node i, or some other derived vector related to the node (*e.g.*, its distance or influence to every node in the graph). Following [RZA18], the simplest case is $f_b = \sum$, which captures simple base features such as in/out/total degrees. We denote applying the same base function to the egonets of all the nodes in graph G as follows:

$$f_b\langle \mathbf{b}, \mathcal{N} \rangle = [f_b(\mathbf{b}, \mathcal{N}_1), f_b(\mathbf{b}, \mathcal{N}_2), \dots, f_b(\mathbf{b}, \mathcal{N}_N)]^T, \mathbf{b} \in \mathcal{B} \tag{3.1}$$

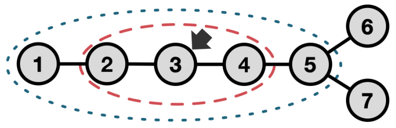
which forms an $N \times 1$ vector. For example, $f_b = \sum \langle \mathbf{A}_{i:}, \mathcal{N} \rangle$ enumerates the out-degree of all nodes in G. By applying f_b on each initial feature $\mathbf{b}$, *e.g.*, $\mathbf{1}^{N \times 1}$ or row/column of adjacency matrix $\mathbf{A}$, we obtain the $N \times B$ base matrix $\mathbf{F}^{(0)}$:

$$\mathbf{F}^{(0)} = [f_b\langle \mathbf{b}_1, \mathcal{N} \rangle, f_b\langle \mathbf{b}_2, \mathcal{N} \rangle, \dots, f_b\langle \mathbf{b}_B, \mathcal{N} \rangle], \mathbf{b}_{1 \cdots B} \in \mathcal{B} \tag{3.2}$$

which aggregates all structural features of the nodes within $\mathcal{N}$. The specific choice of initial vectors $\mathbf{b}$ is not very important as the composed relational functions (§ 3.3.2.2) extensively incorporate both linear and nonlinear structural information automatically. We empirically justify Multi-Lens on the link prediction task over different choices of $\mathcal{B}$ to show its insensitivity.

3.3.2.2 Relational Function Compositions

To derive complex & non-linear node features automatically, Multi-Lens iteratively applies operators $\phi \in \Phi$ (e.g., *mean*, *variance*, *sum*, *max*, *min*, *l2-distance*) to lower-order functions, resulting in function compositions. ℓ such compositions of functions over a node's egonet $\mathcal{N}_i$ captures *higher-order structural features* associated with the $\ell-$hop neighborhoods. For example, assuming $\mathbf{x}$ is the vector consisting of node-wise degrees, the *max* operator captures the maximum degree in the neighborhood $\mathcal{N}$ of a node. The application of the *max* operator to all the nodes forms a new feature vector $\max\langle \mathbf{x}, \mathcal{N} \rangle$ where each entry records the maximum degree in the corresponding neighborhood. Fig. 3.3 shows that the maximum degree of node $\{2, 3, 4\}$ is aggregated for node 3 in $\max\langle \mathbf{x}, \mathcal{N} \rangle$ By iteratively applying *max* to $\max\langle \mathbf{x}, \mathcal{N} \rangle$ in the same way, the maximum value from broader neighborhood $\mathcal{N}$ is aggregated, which is equivalent to finding the maximum degree in the 2-hop neighborhood. Fig. 3.3b depicts this process for node 3.



(a) 1- and 2-hop neighborhood of node 3

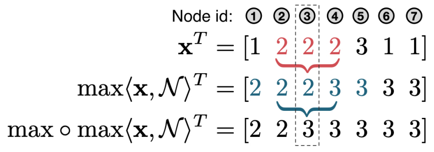

Node id: ① ② ③ ④ ⑤ ⑥ ⑦
$\mathbf{x}^T = [1\ 2\ 2\ 2\ 3\ 1\ 1]$
$\max\langle \mathbf{x}, \mathcal{N} \rangle^T = [2\ 2\ 2\ 3\ 3\ 3\ 3]$
$\max \circ \max\langle \mathbf{x}, \mathcal{N} \rangle^T = [2\ 2\ 3\ 3\ 3\ 3\ 3]$
(b) $\max \circ \max\langle \mathbf{x}, \mathcal{N} \rangle$ captures the max degree in 2-hop neighborhood

Figure 3.3: The composition of relational functions incorporates node degrees (column vector x) in expanded subgraphs.

Formally, at level $\ell \in \{1, \ldots, L\}$, a new function is composed as:

$$f^{(\ell)} = \phi \circ f^{(\ell-1)}, \forall \phi \in \Phi \tag{3.3}$$

where $L < \texttt{diam}(G)$ or the diameter of G, and $f^{(0)} = f_b$(§ 3.3.2.1). We formally define some operators $\phi \in \Phi$ in Table 3.2.

Applying $f^{(\ell)}$ to $\mathbf{F}^{(0)}$ generates order-ℓ structural features of the graph as $\mathbf{F}^{(\ell)}$. In practice, Multi-Lens recursively generates $\mathbf{F}^{(\ell)}$ from $\mathbf{F}^{(\ell-1)}$ by applying a total of $|\Phi|$ operators. The particular order in which relational operators are applied records how a function is generated.

Table 3.2: **Relational operators used in the experiment.**

ϕ	Definition	ϕ	Definition
max/min	$\max / \min_{i \in \mathcal{S}} \mathbf{x}_i$	*variance*	$\frac{1}{\lvert\mathcal{S}\rvert} \sum_{i \in \mathcal{S}} \mathbf{x}_i^2 - (\frac{1}{\lvert\mathcal{S}\rvert} \sum_{i \in \mathcal{S}} \mathbf{x}_i)^2$
sum	$\sum_{i \in \mathcal{S}} \mathbf{x}_i$	*l1-distance*	$\sum_{j \in \mathcal{S}} \lvert x_i - x_j \rvert$
mean	$\frac{1}{\lvert\mathcal{S}\rvert} \sum_{i \in \mathcal{S}} \mathbf{x}_i$	*l2-distance*	$\sum_{j \in \mathcal{S}} (x_i - x_j)^2$

Multi-Lens then collects the composed relational functions per level into $\mathcal{F}_r$ as a part of the latent summary.

In terms of space, Equation (3.3) indicates the dimension of $\mathcal{F}_r$ grows exponentially with $|\Phi|$, *i.e.*, $|\mathcal{F}_r^{(\ell)}| = |\mathcal{B}||\Phi|^{(\ell)}$, which is also the number of columns in $\mathbf{F}^{(\ell)}$. However, the max level L is bounded with the diameter of G, that is $L \leq \mathtt{diam}(G) - 1$ because functions with orders higher than that will capture the same repeated structural information. Therefore, the size of $\mathcal{F}_r$ is also bounded with L. Although the number of relational functions grows exponentially, real-world graphs are extremely dense with small diameters $\mathtt{diam}(G) \propto \log \log N$ [CH03]. In our experiments in § 3.4, $|\mathcal{F}_r| \approx 1000$ for $|\mathcal{B}| = 3$ base functions, $|\Phi| = 7$ operators, and $L = 2$ levels.

3.3.3 Heterogeneous Context

So far we have discussed how to obtain the base structural feature matrix $\mathbf{F}^{(0)}$ and the multi-level structural feature representations $\mathbf{F}^{(\ell)}$ by recursively employing the relational functions. Note that directly deriving the structural embeddings based on these representations would lead to low performance due to skewness in the extracted structural features. Here we discuss an intermediate transformation of the generated matrices that helps capture rich contextual patterns in the neighborhoods of each node, and eventually leads to a powerful summary.

3.3.3.1 Handling skewness

For simplicity, we first discuss the case of a homogeneous network G with a single node and edge type, and undirected edges. To handle the skewness in the higher-order structural features (§ 3.3.2) and more effectively capture the structural identity of each node within its context (i.e., non-typed neighborhood), we opt for an intuitive approach: for each node i and each base/higher-order feature j, we create a histogram $\mathbf{h}_{ij}$ with c bins for the nodes in

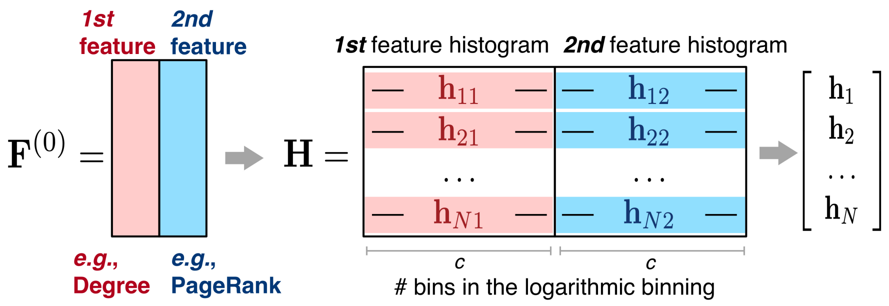


Figure 3.4: Example of creating histogram-based matrix representation $\mathbf{H}^{(0)}$ with $Z = 2$ features in the base feature matrix $\mathbf{F}^{(0)}$. A single object / edge type and no edge directionality is assumed here for simplicity.

its neighborhood $\mathcal{N}_i$. Variants of this approach are used to capture node context in existing representation learning methods, such as struc2vec [RSF17] and xNetMF [HSSK18]. In our setting, the structural identity of node i is given as the concatenation of all its feature-specific histograms.

$$\mathbf{h}_i = \begin{bmatrix} \mathbf{h}_{i1} & \mathbf{h}_{i2} & \cdots & \mathbf{h}_{iZ} \end{bmatrix}, \tag{3.4}$$

where $Z = |\mathcal{B}| + \sum_{\ell=1}^{L} |\mathcal{B}| \cdot |\Phi|^{\ell}$ is the total number of histograms, or the number of base and higher-order features. Each histogram is in logarithmic scale to better describe the power-law-like distribution in graph features and has a total of c bins. By stacking all the nodes' structural identities vertically, we obtain a rich histogram-based context matrix $\mathbf{H} = \begin{bmatrix} \mathbf{h}_1; \, \mathbf{h}_2; \cdots; \mathbf{h}_N \end{bmatrix}$ as shown in Fig. 3.4.

3.3.3.2 Handling object/edge types and directionality

The histogram-based representation that we described above can be readily extended to handle any arbitrary network G with multiple object types, edge types and directed edges **(P1)**. The idea is to capture the structural identity of each node i within its *different* contexts:

- $\mathcal{N}_i^t$ or $\mathcal{N}_i^\tau$: the neighborhood that contains only nodes of type $t \in \mathcal{T}_V$ or edges of type $\tau \in \mathcal{T}_E$, and
- $\mathcal{N}_i^+$ or $\mathcal{N}_i^-$: the neighborhood with only outgoing or incoming edges for node i.

For example, to handle different object types, we create a context matrix $\mathbf{H}_o^t$ by restricting the histograms on neighbors of type t, $\mathcal{N}_i^t$. These per-type matrices can be stacked into a tensor $\underline{\mathcal{H}}$, with each slice corresponding to a node-level histogram, $\mathbf{H}_o^t$ of object type, t. Alternatively, the tensor can be matricized by frontal slices. By further restricting the neighborhoods to contain specific edge types and/or directionality in a similar manner, we can obtain the histogram-based representations $\mathbf{H}_e^t$ and $\mathbf{H}_d^t$, respectively.

By imposing all of the restrictions at once, we can also obtain context matrix $\mathbf{H}$ that accounts for all types of heterogeneity.

3.3.4 Latent Summarization

The previous two subsections can be seen as a general framework for automatically extracting, linear and non-linear, higher-order structural features that constitute the nodes' contexts at multiple levels ℓ. Unlike embedding methods that generate graph-size dependent node representations, we seek to derive a compressed latent representation of G **(P2)** that supports on-the-fly generation of node embeddings and (inductive) downstream tasks **(P3,P4)**. Although graph summarization methods [LSDK18] are relevant as they represent an input graph with a summary or supergraph, it is infeasible to generate latent node representations due to the incurred information loss. Thus, such methods, which have different end goals, do not satisfy **(P4)**.

3.3.4.1 Multi-level Summarization

Multi-Lens explores node similarity based on the assumption that similar nodes should have similar structural context over neighborhoods of different hops. Given the histogram-based context matrix $\mathbf{H}^{(\ell)}$ that captures the heterogeneity of feature values associated with the $\ell-$order egonets in G (§ 3.3.2.2), Multi-Lens obtains the level-ℓ summarized representation $\mathbf{S}^{(\ell)}$ via factorization $\mathbf{H}^{(\ell)} = \mathbf{Y}^{(\ell)}\mathbf{S}^{(\ell)}$, where $\mathbf{Y}^{(\ell)}$ is the dense node embedding matrix that we do *not* store. Then, the latent summary $\mathcal{J}$ consists of the set of relational functions $\mathcal{F}_r$ (§ 3.3.2), and the multi-level summarized representations $\mathcal{S} = \{\mathbf{S}^{(1)}, \ldots, \mathbf{S}^{(\ell)}\}$. Though any technique can be used (e.g., NMF), we give the factors based on SVD for illustration:

$$\texttt{level-}\ell\texttt{ node embeddings (}\underline{\texttt{not stored}}\texttt{):}\ \mathbf{Y}^{(\ell)} = \mathbf{U}^{(\ell)}\sqrt{\Sigma^{(\ell)}} \tag{3.5}$$

$$\texttt{level-}\ell\texttt{ summarized representation:}\quad \mathbf{S}^{(\ell)} = \sqrt{\Sigma^{(\ell)}}\mathbf{V}^{(\ell)T} \tag{3.6}$$

where $\Sigma^{(\ell)}$ are the singular values of $\mathbf{H}^{(\ell)}$, and $\mathbf{U}^{(\ell)T}$, $\mathbf{V}^{(\ell)T}$ are its left and right singular vectors, respectively.

Intuitively, $\mathbf{S}^{(\ell)}$ contains the best-fit $K^{(\ell)}$-dimensional subspace vectors for node context $\mathbf{H}^{(\ell)}$ in the neighborhood at order-ℓ. The summary representations across different orders form the hierarchical summarization of G that contains both local and global structural information, and the derived embedding matrix $\mathbf{Y}^{(\ell)}$ also preserves node similarity at multiple levels. There is no need to store any of the intermediate matrices $\mathbf{F}^{(\ell)}$ and $\mathbf{H}^{(\ell)}$, nor the node embeddings $\mathbf{Y}^{(\ell)}$. The former two matrices can be derived on the fly given the composed relational functions $\mathcal{F}_r$. Then, the latter can be efficiently estimated using the obtained sparse $\mathbf{H}^{(\ell)}$ matrix and the stored summarized matrix $\mathbf{S}^{(\ell)}$ through SVD (§ 3.3.6 gives more details). Moreover, since the elements of the summary $\mathcal{S}$, i.e., the relational functions $\mathcal{F}_r$ and the factorized matrices, are independent of the nodes or edges of the input graph, both require trivial storage and achieve compression efficiency **(P2)**. We provide the pseudo-code of Multi-Lens in Algorithm III.1.

We note that the relational functions $\mathcal{F}_r$ are a key enabling factor of our summarization approach. Without them, other embedding methods cannot benefit from our proposed summarized representations $\mathcal{S}$, nor reconstruct the node context and embeddings.

3.3.4.2 Inductive Summaries (P3)

The higher-order features derived from the set of relational functions $\mathcal{F}_r$ are structural, and thus generalize across graphs [HGER+12, HSSK18, ARZ+18] and are independent of node IDs. As such, the factorized matrices in $\mathcal{S}$ learned on G can be transferred to another graph G' to learn the node embeddings $\mathbf{Y}^{(\ell)}$ of a new, previously unseen graph G' as:

$$\mathbf{Y}'^{(\ell)} = \mathbf{H}'^{(\ell)}(\mathbf{S}^{(\ell)})^{\dagger} \tag{3.7}$$

Algorithm III.1 Multi-Lens

```
    Input: (un)directed heterogeneous graph $G$, a set of relational operators $\Phi$; layer-wise embedding
    dimensionality $K^{(\ell)}$, for $K = \sum_{\ell=1}^{L} K^{(\ell)}$ dimensions in total; number of bins $c$ for the histogram
    representation
    Output: Summary $\mathcal{J} = \{\mathcal{F}, \mathcal{S}\}$
 1: $\mathcal{F}_r \leftarrow f_b$                                          ▷ Base graph functions: Eq. (3.1)
 2: Initialize $\mathbf{F}^{(0)}$                                           ▷ Base feature matrix Eq. (3.2)
 3: for $\ell = 1,\ldots,L$ do                                              ▷ multi-level summarization
 4:     for $i = 1,\ldots,|\Phi|$ do                                        ▷ relational operators
 5:         parallel for $j = 1,\ldots,BR^{\ell-1}$ do                      ▷ Columns in $\mathbf{F}^{(l)}$
 6:             $f = \phi_i \circ f_j^{(\ell-1)}$                           ▷ Compose func. in $\mathcal{F}_r^{(\ell-1)}$
 7:             $\mathbf{F}^{(\ell)} = \mathbf{F}^{(\ell)} \cup \phi_i \langle \mathbf{F}_{:,j}^{(\ell-1)}, \mathcal{N} \rangle$    ▷ Feature concatenation
 8:     Derive heterogeneous context $\mathbf{H}^{(\ell)}$                  ▷ § 3.3.3
 9:     $\mathbf{S}^{(\ell)} = \sqrt{\Sigma^{(\ell)}} \mathbf{V}^{(\ell)T}$    ▷ SVD: $\mathbf{H}^{(\ell)} = \mathbf{U}^{(\ell)} \Sigma^{(\ell)} \mathbf{V}^{(\ell)T}$
10:     $\mathcal{F}_r \leftarrow \mathcal{F}_r \cup f$, $\mathcal{S} \leftarrow \mathcal{S} \cup \mathbf{S}^{(\ell)}$
```

where $\mathbf{S}^{(\ell)} \in \mathcal{S}$ is learned on G, † denotes the pseudo-inverse, and $\mathbf{H'}^{(\ell)}$ is obtained via applying $\mathcal{F}_r$ to G'. The pseudo-inverse, $(\mathbf{S}^{(\ell)})^\dagger$ can be computed efficiently through SVD as long as the rank of $\mathbf{S}^{(\ell)}$ is limited (*e.g.*, empirically setting $K^{(\ell)} \leq 128$) [Bra06].

Equation (3.7) requires the same dimensionality $K^{(\ell)} = K'^{(\ell)}$ and the same number of bins of histogram context matrices $c = c'$ at each level ℓ. The embeddings learned inductively reflect the node-wise structural difference between graphs, G and G', which can be used in applications of graph mining and time-evolving analysis. We present an application of temporal event detection in § 3.4.4.

3.3.4.3 On-the-fly embedding derivation (P4)

Given the summarized matrix $\mathbf{S}^{(\ell)}$ at level ℓ, the embeddings of specific nodes that are previously seen or unseen can be derived efficiently. Multi-Lens first applies $\mathcal{F}_r$ to derive their heterogeneous context $\mathbf{H}_{\text{sub}}^{(l)}$ based on the graph structure, and then obtains the embeddings via Eq. (3.7). We concatenate $\mathbf{Y}^{(\ell)}$ given as output at each level to form the final node embeddings [TQW+15b]. Given that the dimension of embeddings is $K^{(\ell)}$ at level ℓ, the final embedding dimension is $K = \sum_{\ell=1}^{L} K^{(\ell)}$.

3.3.5 Generalization

Here we discuss the generalizations of our proposed approach to labeled and attributed graphs. It is straightforward to see that homogeneous, bipartite, signed, and labeled graphs are all special cases of heterogeneous graphs with $|\mathcal{T}_V| = |\mathcal{T}_E| = 1$ types, $|\mathcal{T}_V| = 2$ and $|\mathcal{T}_E| = 1$ types, $|\mathcal{T}_V| = 1$ and $|\mathcal{T}_E| = 2$ types, and $|\mathcal{T}_V| = \#(\text{node labels})$ and $|\mathcal{T}_E| = \#(\text{edge labels})$ types, respectively. Therefore, our approach naturally generalizes to all of these graphs. Other special cases include k-partite and attributed graphs.

Multi-Lens also supports attributed graphs that have multiple attributes per node or edge (instead of a single label): Given an initial set of attributes organized in an attribute matrix $\mathbf{F}_a$, we can concatenate $\mathbf{F}_a$ with the base attribute matrix and apply our approach as before. Alternatively, we can transform the graph into a labeled one by applying a labeling function $\chi : \mathbf{x} \to y$ that maps every node's attribute vector $\mathbf{x}$ to a label y [ARZ+18]. Besides, our proposed method is easy to parallelize as the relational functions are applied to the subgraphs of each node independently, and the feature values are computed independently.

3.3.6 Complexity Analysis

3.3.6.1 Computational Complexity.

Multi-Lens is linear to the number of nodes N and edges M. Per level, it derives the histogram-based context matrix $\mathbf{H}^{(\ell)}$ and performs a rank-$K^{(\ell)}$ approximation.

Lemma 3.3.1. *The computational complexity of* Multi-Lens *is:*

$$\mathcal{O}((c|\mathcal{F}_r||\mathcal{T}_V||\mathcal{T}_E| + K^2)N + M)$$

Proof. The computational complexity of Multi-Lens includes deriving (a) distribution-based matrix representation $\mathbf{H}$ and (b) its low-rank approximation.

The computational unit of Multi-Lens is the relational operation performed over the egonet of a specific node. Searching the neighbors for all node $i \in V$ has complexity $\mathcal{O}(N+M)$ through BFS. The complexity of step (a) is linear to $|\mathcal{F}_r|$, as indicated in § 3.3.3, this number is $|\mathcal{B}||\Phi|^{\ell} \cdot 2|\mathcal{T}_V||\mathcal{T}_E|c$.

Based on the sparsity of $\mathbf{H}$, Multi-Lens performs SVD efficiently through fast Monte-Carlo Algorithm by extracting the most significant K singular values [FKV04] with computational complexity $\mathcal{O}(K^2N)$. Therefore step (b) can be accomplished in $\mathcal{O}((K^{(\ell)})^2N)$ by extracting the most significant $K^{(\ell)}$ singular values at level ℓ. Furthermore, deriving all K singular values has $\mathcal{O}(K^2N)$ complexity as $\sum_{\ell=1}^{L}(K^{(\ell)})^2 \leq (\sum_{\ell=1}^{L} K^{(\ell)})^2 = K^2$. The overall computational complexity is thus $\mathcal{O}(N|\mathcal{F}_r||\mathcal{T}_E||\mathcal{T}_V|c + K^2N + M)$. Note that both $|\Phi|$ and L are small constants in our proposed method (*e.g.*, $|\Phi| = 7$ and $L \leq 2$). Multi-Lens scales linearly with the number of nodes and edges $(N + M)$ in G. □

As indicated in § 3.3.2, the number of features in $\mathbf{H}^{(\ell)}$ across L layers is equivalent to the number of composed relational functions $|\mathcal{F}_r|$. Since $|\mathcal{F}_r|$ is bounded with L and $L < \mathtt{diam}(G)$, the term $(c|\mathcal{F}_r||\mathcal{T}_V||\mathcal{T}_E| + K^2)$ forms a constant related to graph heterogeneity and structure.

3.3.6.2 Space Complexity.

The runtime and output compression space complexity of Multi-Lens is given in Lemma 3.3.2. In the runtime at level ℓ, Multi-Lens leverages $\mathbf{F}^{(\ell-1)}$ to derive $\mathbf{F}^{(\ell)}$ and $\mathbf{H}^{(\ell)}$, which comprise two terms in the runtime space complexity.

Lemma 3.3.2. *The* M*ulti-Lens space complexity during runtime is* $\mathcal{O}((c|\mathcal{F}_r||\mathcal{T}_V||\mathcal{T}_E| + |\mathcal{F}_r|)N)$. *The space needed for the* output *of* M*ulti-Lens is* $\mathcal{O}(cK|\mathcal{F}_r||\mathcal{T}_V||\mathcal{T}_E| + |\mathcal{F}_r|)$.

Proof. In the runtime at level ℓ, Multi-Lens stores $\mathbf{F}^{(\ell-1)}$ to dervie $\mathbf{F}^{(\ell)}$ and $\mathbf{H}^{(\ell)}$, which take $\mathcal{O}(N|\mathcal{F}_r|)$ and $\mathcal{O}(c|\mathcal{F}_r||\mathcal{T}_V||\mathcal{T}_E|N)$ space, respectively. SVD can be performed with $p \ll N$ sampled rows. For the output, storing the set of ordered compositions of relational functions in the summary requires space complexity $O(|\mathcal{F}_r|)$. For the set of matrices $\mathcal{S}$, we store $\mathbf{S}^{(\ell)}$ across all L levels. As shown in the time complexity analysis, the number of binned features (columns) in $\mathbf{H}$ over all levels is $2|\mathcal{F}_r||\mathcal{T}_V||\mathcal{T}_E|c$, which includes incorporating $|\mathcal{T}_V|$ object types with both in-/out- directionality and all edge types. The size of the output summarization matrices is thus $\mathcal{O}(K|\mathcal{F}_r||\mathcal{T}_V||\mathcal{T}_E|c)$, which is related to the graph heterogeneity and structure and independent of the network size N, M. □

The output of Multi-Lens that needs to be stored (i.e., set of relational functions $\mathcal{F}_r$ and summary matrices in $\mathcal{S}$) is independent of N, M. Compared with output embeddings with complexity $\mathcal{O}(NK)$ given by existing methods, Multi-Lens satisfies the crucial property we desire **(P2)** from latent summarization (Def. 6).

3.4 Experiments

In our evaluation we aim to answer four research questions:

Q1 How much space do the Multi-Lens summaries save (P2)?

Q2 How does Multi-Lens perform in machine learning tasks, such as link prediction in heterogeneous graphs (P1)?

Q3 How well does it perform in inductive tasks (P3)?

Q4 Does Multi-Lens scale well with the network size?

We have discussed on-the-fly embedding derivation (P4) in § 3.3.4.3.

3.4.1 Experimental Setup

3.4.1.1 Data

In accordance with **(P1)**, we use a variety of real-world heterogeneous network data from Network Repository [RA15a]. We present their statistics in Table 3.3.

- **Facebook** [GL16] is a homogeneous network that represents friendship relation between users.

- **Yahoo! Messenger Logs** [RZA18] is a heterogeneous network of Yahoo! messenger communication patterns, where edges indicate message exchanges. The users are associated with the locations from which they have sent messages.

- **DBpedia**[2] is an unweighted, heterogeneous subgraph from DBpedia project consisting of 4 types of entities and 3 types of relations: user-occupation, user-work ID, work ID-genre.

[2] http://networkrepository.com/

- **Digg**^{2} is a heterogeneous network that records the voting behaviors of users to stories they like. Node types include users and stories. Each edge represents one vote or a friendship.

- **Bibsonomy**^{2} is a k-partite network that represents the behaviors of users assigning tags to publications.

Table 3.3: Statistics for the heterogeneous networks that we use in our experiments.

Data	#Nodes	#Edges	#Node Types	Graph Type
facebook	4 039	88 234	1	unweighted
yahoo-msg	100 058	1 057 050	2	weighted
dbpedia	495 936	921 710	4	unweighted
digg	283 183	4 742 055	2	unweighted
bibsonomy	977 914	3 754 828	3	weighted

3.4.1.2 Baselines

We compare Multi-Lens with baselines commonly used in graph summarization, matrix factorization and representation learning over networks, namely, they are: **(1)** Node aggregation or NA for short [ZGGS$^+$16, BGLL08], **(2)** Spectral embedding or SE [TL11], **(3)** LINE [TQW$^+$15b], **(4)** DeepWalk or DW [PARS14], **(5)** Node2vec or n2vec [GL16], **(6)** struc2vec or s2vec [RSF17], **(7)** DNGR [CLX16], **(8)** GraRep or GR [CLX15a], **(9)** Metapath2vec or m2vec [DCS17], and **(10)** AspEm [SGZ$^+$18], **(11)** Graph2Gauss or G2G [BG17]. To run baselines that do not explicitly support heterogeneous graphs, we align nodes of the input graph according to their object types and re-order the IDs to form the homogeneous representation. In node aggregation, CoSum [ZGGS$^+$16] ran out of memory due to the computation of pairwise node similarity. We use Louvain [BGLL08] as an alternative that scales to large graphs and forms the basis of many node aggregation methods.

3.4.1.3 Configuration

We evaluate Multi-Lens with $L = 1$ and $L = 2$ to capture subgraph structural features in 1-hop and 2-hop neighborhoods, respectively, against the optimal performance achieved by the baselines. We derive in-/out- and total degrees to construct the $N \times 3$ base feature matrix $\mathbf{F}^{(0)}$. Totally, we generate ≈ 1000 composed functions, each of which corresponds to a column vector in $\mathbf{F}$. For fairness, we do not employ parallelization and terminate processes exceeding 1 day. The output dimensions of all node representations are set to be $K = 128$. Additionally, we run all experiments on Mac OS platform with 2.5GHz Intel Core i7 and 16GB memory.

We configure the baselines as follows: we use 2nd-LINE to incorporate 2-order proximity in the graph; we run node2vec with grid searching over $p, q \in \{0.25, 0.50, 1, 2, 4\}$ as mentioned in [GL16] and report the best. For GraRep, we set $k = 2$ to incorporate 2-step relational information. For DNGR, we follow the paper to set the random surfing probability $\alpha = 0.98$ and use a 3-layer neural network model where the hidden layer has 1024 nodes. For Metapath2vec, we retain the same settings (number of walks = 1000, walk length = 100) to generate walk paths and adopt a similar the meta-path "Type 1-Type 2-Type 1" as the "A-P-A" schema as suggested in the paper. For Multi-Lens, although arbitrary relational functions can be used, we use order-1 $f_b = \sum$ as the base graph function for simplicity in our experiments. To begin with, we derive in-/out- and total degrees to construct the $N \times 3$ base feature matrix $\mathbf{F}^{(0)}$ denoted as $\big[f_b\langle \mathbf{b}_1, \mathcal{N}\rangle, f_b\langle \mathbf{b}_2, \mathcal{N}\rangle, f_b\langle \mathbf{b}_3, \mathcal{N}\rangle\big]$ where $\mathbf{b}_1 = \mathbf{A}_{i:}$, $\mathbf{b}_2 = \mathbf{A}_{:i}$, and $\mathbf{b}_3 = (\mathbf{A} + \mathbf{A}^T)_{i:}$, for $i \in V$. We set $L = 2$ to construct order-2 relational functions to equivalently incorporate 2-order proximity as LINE does, but we do not limit other methods to incorporate higher order proximity. All other settings are kept default. In table 3.2, we list all relational operators used in the experiment.

3.4.2 Compression rate of Multi-Lens

The most important question for our latent summarization method (Q1) is about how well it compresses large scale heterogeneous data **(P2)**. To show Multi-Lens's benefits over existing embedding methods, we measure the storage space for the generated embeddings by the baselines that ran successfully. In Table 3.4 we report the space required by the Multi-Lens summaries in MB, and the space that the outputs of our baselines require *relative* to the corresponding Multi-Lens summary. We observe that the latent summaries generated by Multi-Lens take up very little space, well under 1MB each. The embeddings of the representation learning baselines take up $80 - 2152\times$ more space than the Multi-Lens summaries on the larger datasets. On Facebook, which is a small dataset with 4K nodes, the summarization benefit is limited; the baseline methods need about $3 - 12\times$ more space. In addition, the node-aggregation approach takes up to $12\times$ storage space compared to our latent summaries, since it generates an $N \times 1$ vector that depends on graph size to map each node to a supernode. This further demonstrates the advantage of our graph-size independent latent summarization.

Table 3.4: Output storage space required for embedding methods relative to the Multi-Lens summaries (given in MB). Multi-Lens requires $3 - 2152\times$ less output storage space than embedding methods.

Data	SE	LINE	n2vec	DW	m2vec	AspEm	G2G	**ML** (MB)
facebook	8.13x	8.48x	12.79x	12.84x	3.82x	8.50x	9.17x	**0.58**
yahoo	187.1x	180.0x	242.2x	231.0x	79.8x	197.4x	195.8x	**0.62**
dbpedia	710.0x	714.2x	996.4x	996.2x	-	749.2x	743.6x	**0.81**
digg	608.2x	612.8x	848.9x	830.3x	259.9x	641.7x	635.2x	**0.54**
bibson.	1512.1x	1523.0x	2152.5x	2152.5x	-	1595.8x	-	**0.75**

3.4.3 Link Prediction in Heterogeneous Graphs

For Q2, we investigate the performance of Multi-Lens in link prediction task over heterogeneous graphs (**P1**). We use logistic regression with regularization strength = 1.0 and stopping criteria= 10^{-4}. An edge e_{ij} is represented by the concatenating the embeddings of its source and destination: $emb(e_{ij}) = [emb(i), emb(j)]$ as used in [RZA18]. For each dataset $G(V, E)$, we create the subgraph $G'(V, E')$ by keeping all the nodes but randomly removing $\sim 40\%$ edges. We run all methods on G' to get node embeddings and randomly select $10\%|E|$ edges as the training data. Out of the removed edges, 25% ($10\%|E|$) are used as missing links for testing. We also randomly create the same amount of "fake edges" for both training and testing. Table 3.5 illustrates the prediction performance measured with AUC, ACC, and F1 scores.

We observe that Multi-Lens outperforms the baselines measured by every evaluation metric. Multi-Lens outperforms embedding baselines by $3.46\% \sim 34.34\%$ in AUC and $3.71\% \sim 31.33\%$

Table 3.5: Link prediction: node embeddings derived by Multi-Lens (ML) outperforms all baselines measured by every evaluation metric. Specifically, Multi-Lens outperforms embedding baselines by $3.46\% \sim 34.34\%$ in AUC and $3.71\% \sim 31.33\%$ in F1 on average. It outperforms even more over the aggregation-based methods. The asterisk * denotes statistically significant improvement over the best baseline at $p < 0.01$ in a two-sided t-test. OOT = Out Of Time (12 hours), OOM = Out Of Memory (16GB).

Data	Metric	NA	SE	LINE	DW	n2vec	GR	s2vec	DNGR	m2vec	AspEm	G2G	**ML**($L1$)	**ML**($L2$)
facebook	**AUC**	0.6213	0.6717	0.7948	0.7396	0.7428	0.8157	0.8155	0.7894	0.7495	0.5886	0.7968	0.8703	**0.8709***
	ACC	0.5545	0.5995	0.7210	0.6460	0.6544	0.7368	0.7388	0.7062	0.7051	0.5628	0.7274	**0.7920***	0.7904
	F1	0.5544	0.5716	0.7210	0.6296	0.6478	0.7367	0.7387	0.7060	0.7041	0.5628	0.7273	**0.7920***	0.7905
yahoo	**AUC**	0.7189	0.5375	0.6745	0.7715	0.7830	0.7535			0.6708	0.5587	0.6988	0.8443	0.8446*
	ACC	0.2811	0.5224	0.6269	0.6927	0.7036	0.6825	OOT	OOM	0.6164	0.5379	0.6564	**0.7587***	**0.7587***
	F1	0.2343	0.5221	0.6265	0.6897	0.7016	0.6821			0.6145	0.5377	0.6562	**0.7577***	**0.7577***
dbpedia	**AUC**	0.6002	0.5211	0.9632	0.8739	0.8774					0.6364	0.7384	**0.9820***	0.9809
	ACC	0.3998	0.5399	0.9111	0.8436	0.8436	OOM	OOT	OOM	OOT	0.5869	0.6625	**0.9186**	0.9151
	F1	0.2968	0.4539	0.9110	0.8402	0.8402					0.5860	0.6613	**0.9186**	0.9150
digg	**AUC**	0.7199	0.6625	0.9405	0.9664	0.9681				0.9552	0.5644	0.8978	**0.9894***	0.9893
	ACC	0.2801	0.6512	0.8709	0.9023	0.9049	OOM	OOT	OOM	0.8891	0.5459	0.8492	**0.9596***	0.9590
	F1	0.2660	0.6223	0.8709	0.9019	0.9046				0.8890	0.5459	0.8492	**0.9595***	0.9590
bibson	**AUC**	0.7836	0.6694	0.9750	0.6172	0.6173					0.6127		**0.9909***	0.9909
	ACC	0.2164	0.6532	0.9350	0.5814	0.5816	OOM	OOT	OOM	OOT	0.5790	OOM	**0.9485***	0.9466
	F1	0.2070	0.6064	0.9349	0.5781	0.5782					0.5772		**0.9485***	0.9466

in F1 score. For runnable baselines designed for node embeddings in homogeneous graphs (baseline **3 - 8**), the experimental result is expected as Multi-Lens incorporates heterogeneous contexts within 2-neighborhood in the node representation. It is worth noting that Multi-Lens outperforms Metapath2vec and AspEm, both of which are designed for heterogeneous graphs. One reason behind is the inappropriate meta-schema specified, as Metapath2vec and AspEm require predefined meta-path / aspect(s) in the embedding. On the contrary, Multi-Lens does not require extra input and captures graph heterogeneity automatically. We also observe the time and runtime space efficiency of Multi-Lens when comparing with neural-network based methods (DNGR, G2G), GraRep and struc2vec on large graphs. Although the use of relational operators is similar to information propagation in neural-networks, Multi-Lens requires less computational resource with promising results. Moreover, the Multi-Lens summaries for both $L = 1$ and $L = 2$ levels achieve promising results, but generally we observe that there is a slight drop in accuracy for higher levels. This indicates that node context at higher levels may incorporate noisy, less-relevant higher-order structural features (§ 3.3.2.2).

3.4.4 Inductive Anomaly Detection

To answer Q3 about inductive learning, we first perform anomalous subgraph detection on both synthetic and real-world graphs. We also showcase the application of Multi-Lens summaries on real-world event detection, in an inductive setting (**P3**).

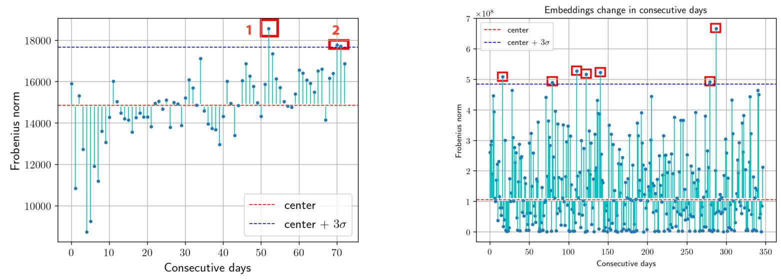
3.4.4.1 Anomalous Subgraph Detection

Following the literature [MBWB15], we first generate two "background" graphs, G_1 and G_2. We then induce an anomalous subgraph into G_2 by randomly selecting n nodes and adding edges to form an anomalous ER subgraph with p and n shown in Table 3.6. We leverage the summary learned from G_1 to learn node embeddings in G_2, we identify the top-n nodes with the highest change in euclidean distance as anomalies, and report the precision in Table 3.6. In the synthetic setting, we generate two Erdős-Rényi (ER) graphs, G_1^{syn} and G_2^{syn}, with 10^4 nodes and average degree 10 ($p^{\text{back}} = 10^{-3}$). In the real-graph setting, we construct G_1^{real} and G_2^{real} using two consecutive daily graphs in the bibsonomy dataset.

	Real Graph					Synthetic Graph		
p \ n	**100**	**200**	**300**	**400**	**500**	**50**	**75**	**100**
0.1	0.200	0.780	0.950	0.973	0.980	0.06	0.3333	0.81
0.3	0.870	0.960	0.990	0.995	0.996	1	1	1
0.5	0.920	0.990	0.993	1	1	1	1	1
0.7	0.940	0.990	1	1	1	1	1	1
0.9	0.980	1	1	1	1	1	1	1

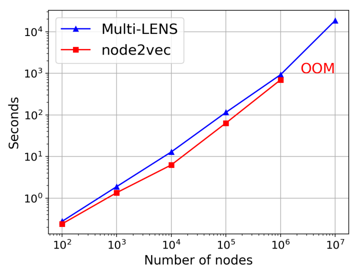
Table 3.6: Anomalous Erdős-Rényi (ER) subgraphs (with n nodes and probability p) detection precision on both synthetic and real-world graphs.

In the synthetic scenario, we observe that Multi-Lens gives promising results by successfully detecting nodes with the most deviating embedding values, except when the size of injection is small. In the case of very sparse ER injections ($p = 0.1$), the anomalies are not detectable over the natural structural deviation between G_1^{syn} and G_2^{syn}. However, denser injections ($p \geq 0.3$) affect more significantly the background graph structure, which in turn leads to notable change in the Multi-Lens embeddings for the affected subset of nodes. For real-world graphs, we also observe that Multi-Lens successfully detects anomalous patterns when the injection is relatively dense, even when the background graphs have complex structural patterns. This demonstrates that Multi-Lens can effectively detect global changes in graph structures.



(a) `Twitter`: Consecutive embeddings change in Twitter during 05/12/2014–07/31/2014.

(b) `Enron`: Consecutive embeddings change in weekdays during 01/01/2001–5/01/2002.



(c) Runtime (in sec) for Multi-Lens vs. node2vec.

Figure 3.5: (a)-(b) Major event detection in real world datasets; (c) Runtime reported on ER graphs with $d_{\text{avg}} = 10$. Multi-Lens scales similarly to node2vec with less memory requirement while node2vec runs out of memory on the graph with 10^7 nodes.

3.4.4.2 Graph-based Event Detection

We further apply Multi-Lens to real-world graphs to detect events that appear unusual or anomalous with respect to the global temporal behavior of the complex network. The datasets we used are the `Twitter`[3] and `Enron`[4] graphs. `Twitter` has totally 308 499 nodes and 2 601 834 edges lasting from 05/12/2014 to 07/31/2014, and `Enron` has totally 80848 nodes and 2 233 042 edges lasting from 01/01/2001 to 05/01/2002. Similar to the synthetic scenario, we split the temporal graph into consecutive daily subgraphs and adopt the summary learned from G_{t-1} to get node embeddings of G_t. Intuitively, large distances between node embeddings of consecutive daily graphs indicate abrupt changes of graph structures, which may signal events.

Fig. 3.5a shows the change of Frobenius norm between keyword / hashtag embeddings in consecutive instances of the daily `Twitter` co-mentioning activity. The two marked days are 3σ (stdev) units away from the median value [KVF13], which correspond to serious events: (1) the Gaza-Israel conflict and (2) Ebola Virus Outbreak. Compared with other events in the same time period, the detected ones are the most impactful in terms of the number of people affected, and the attraction they drew as they are related to terrorism or domestic security. Similarly for `Enron`, we detect several events based on the change of employee embeddings in the Enron corpus from the daily message-exchange behavior. We highlight these events, which correspond to notable ones in the company's history, in Fig. 3.5b. Specifically, the

[3]`http://odds.cs.stonybrook.edu/twittersecurity-dataset/`
[4]`http://odds.cs.stonybrook.edu/enroninc-dataset/`

events detected include: (1) The quarterly conference call where Jeffrey Skilling, Enron's CEO, reports "outstanding" status of the company; (2) The infamous quarterly conference call; (3) FERC institutes price caps across the western United States; (4) The California energy crisis ends; (5) Skilling announces desire to resign to Kenneth Lay, founder of Enron; (6) Baxter, former Enron vice chairman, commits suicide, and (7) Enron executives Andrew Fastow and Michael Kopper invoke the Fifth Amendment before Congress.

3.4.5 Scalability of Multi-Lens

Finally, Q4 concerns the scalability of our approach. To that end, we generate Erdős-Rényi graphs with average degree $d_{\text{avg}} = 10$, while varying the number of nodes from 10^2 to 10^7. For reference, we compare it against one of the fastest and most scalable baselines, node2vec. As shown in Fig. 3.5c, node2vec runs out of memory on the graph with 10^7 nodes, whereas Multi-Lens scales almost as well as node2vec and to bigger graphs, while also using less space.

3.5 Conclusion

This work introduced the problem of *latent network summarization* and described a general computational framework, Multi-Lens to learn such space-efficient latent node summaries of the graph that are completely independent of the size of the network. The output (size) of latent network summarization depends only on the complexity and heterogeneity of the network, and captures its key structural behavior. Compared to embedding methods, the latent summaries generated by our proposed method require 80-2152$\times$ **less** output storage space for graphs with millions of edges, while achieving significant improvement in AUC and F1 score for the link prediction task. Overall, the experiments demonstrated the effectiveness of Multi-Lens for link prediction, anomaly and event detection, as well as its scalability and space efficiency.

CHAPTER IV

Latent Temporal Proximity Summarization for Temporal Graphs

This chapter is based on work that appeared at PKDD 2019 [JHRK19].

4.1 Introduction

In this chapter, we extend the idea of summarizing latent graph structural features to handle the real-world *temporal* graphs. In most real-world graphs, multiple types of entities interact with each other in a sequential manner, and the interactions evolve over time, thus it is important to model both the graph heterogeneity and temporal proximity. To overcome the challenge of expensive pairwise comparisons (Chapter I) incurred by the real numbers in node embedding vectors, we introduce a hashing-based summarization technique that generates binary node embeddings, which guarantee at least $32\times$ less storage requirement than real numbers given by conventional methods. This work is motivated by a practical machine learning problem encountered in industry, *user stitching*, which models the user temporal behaviors in the temporal graphs.

Personalization and recommendations increase user satisfaction by providing relevant experiences and handling the online information overload in news, web search, entertainment, and more. Accurately modeling user behavior and preferences over time are at the core of personalization. However, tracking user activity online is challenging as users interact with tens of internet-enabled devices from different locations daily, leading to fragmented user profiles. Without unified profiles, the observed user data are sparse, non-representative of the population, and insufficient for accurate predictions that drive business success.

The problem of *identity or user stitching* aims to identify and group together logged-in and anonymous sessions that correspond to the *same* user despite taking place across different channels, platforms, devices and browsers [SRSCS15]. This problem is a form of entity or identity resolution [GM13, BG07], also known as entity linking, record linkage, and duplicate detection [Chr12, KTR12, BG07]. Unlike entity resolution where textual information per user (*e.g.*, name, address) is available, identity stitching relies solely on user interactions with online content and web metadata. Although cookies can help stitch several different sessions of the same user, many users have multiple cookies (*e.g.*, a cookie for each device or web browser) [DGZ$^+$12], and most cookies expire after a short time, and therefore cannot help to stitch users over time. Similarly, IP addresses change across locations resulting in fragmentation or even erroneous stitching between users who have the same IP address at different times (e.g., airports). Meanwhile, fingerprinting approaches [Eck10] capture user similarity based on device or browser configurations, not on behavioral patterns that remain consistent across devices or browsers. On the other hand, exhaustive solutions for entity resolution require quadratic number of comparisons between all pairs of entities, which is computationally intractable for large-scale web services. This can be partially handled via the heuristic of blocking [PSGP16], which groups similar entity descriptions into blocks, and only compares entities within the same block.

To overcome these challenges and better tailor to the user stitching setup, our solution is based on the idea that the same user accesses *similar content* across platforms and has *similar behavior* over time. We model the user interactions with different content and platforms over time in a dynamic heterogeneous network, where user stitching maps to the identification of *nodes* that correspond to the same real-world entity. Motivated by the success of node representation learning, we aim to find embeddings of time-evolving 'user profiles' over this rich network of interactions. For large-scale graphs, however, the customary dense node representations for each node can often impose a formidable memory requirement, on par with that of the original (sparse) adjacency matrices [JRK$^+$19]. Thus, to efficiently find *sparse* binary representations and link entities based on similar activity while avoiding the pairwise comparison of all user profiles, we solve the following problem:

Problem 1 (Temporal, Hash-based Node Embeddings). *Given a graph $G(V, E)$, the goal of hash-based network embedding is to learn a function $\chi : V \rightarrow \{0,1\}^d$ such that the derived* binary *d-dimensional embeddings (1) preserve similarities in interactions in G, (2) are space-efficient, and (3) accurately capture temporal information and the heterogeneity of the underlying network.*

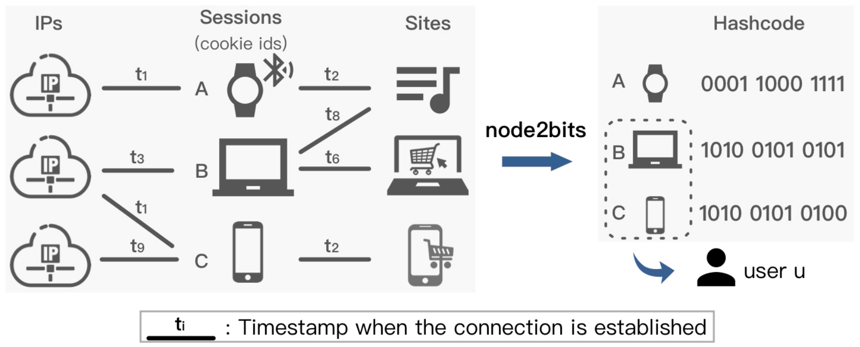


Figure 4.1: Node2bits overview. Node2bits encodes the temporal, heterogeneous information of each node into binary hashcodes for efficient user stitching.

We introduce a *general* framework called Node2bits that captures temporally-valid interactions between nodes in a network, and constructs the contexts based on topological features and (optional) side information of entities involved in the interaction. These feature-based contexts are then turned into histograms that incorporate node type information at different *temporal distances*, and are mapped to binary hashcodes through SimHash [Cha02]. Thanks to locality sensitive hashing [IM98], the hashcodes, which are time-, attribute- and structure-aware, preserve the similarities in temporal interaction patterns in the network, and achieve both space and computational efficiency for similarity search. Given these sparse, hash-based embeddings of all entities, we then cast user stitching as a supervised binary classification task or a hashing-based unsupervised task. As an example, in Fig. 4.1, devices B and C are associated with identical IPs and similar online sales websites visited afterwards, thus they are encoded similarly and could correspond to the same user.

Our contributions are:

- **Embedding-based Formulation:** Going beyond traditional blocking techniques, we formulate the problem of user stitching as the problem of finding temporal, hash-based embeddings in heterogeneous networks such that they maintain *similarities between user interactions over time.*

- **Space-efficient Embeddings:** We propose Node2bits, a practical, intuitive, and fast framework that generates *compact, binary embeddings* suitable for user stitching. Our method combines random walk-based sampling of contexts, their feature-based histogram representations, and locality sensitive hashing to preserve the heterogeneous equivalency of contexts over time.

- **Extensive Empirical Analysis:** Our experiments on real-world networks show that Node2bits outputs a space-efficient binary representation which uses between 63$\times$ and 339$\times$ less space than the baselines while achieving comparable or better performance in user stitching tasks. Moreover, Node2bits is scalable for large real-world temporal and heterogeneous networks.

For reproducibility, the code is at `https://github.com/GemsLab/node2bits`.

4.2 Related Work

Entity Resolution (the general problem under which user stitching falls) has been widely studied and applied in different domains such as databases and information retrieval [DN09, GM13]. Traditional methods that are based on distances can be broadly categorized into (1) pairwise-ER [CR02], which independently decide which pairs are same entity based on a distance threshold, and (2) clustering [DGZ$^+$12], which links nodes in the same cluster. However, these methods are computationally expensive and do not scale to large datasets. Other techniques range from supervised classification [SRSCS15] to probabilistic soft logic [KKP$^+$17] or fingerprinting [Eck10] using side information (*e.g.*, user-agent strings, other web browser features, geo-location). These methods tend to be problem- or even data-specific. On the contrary, our method is general by modeling the data with a heterogeneous, dynamic network that uses both node features (optional) *and* graph structure.

Locality sensitivity hashing (LSH) was first introduced as a randomized hashing framework for efficient approximate nearest neighbor search in high dimensional space [IM98]. It

Table 4.1: Summary of major symbols and their definitions.

Symbol	Definition
$\mathbf{F}$	$N \times \|\mathcal{F}\|$ feature matrix including node attributes and derived features
f_{ij}, $f_{(j)}$	$(i,j)^{th}$ element of $\mathbf{F}$ and index of its j^{th} feature, resp.
$\mathcal{W}$	set of random walks
$(\mathbf{w}_L)_{L\in\mathbb{N}}, \mathbf{w}_L[u]$	sequence of nodes in a random walk of length L, and the index of node u, resp.
L	the maximum length of a random walk
Δt	'temporal distance' in $\mathcal{W}$ based on temporally ordered edge transitions
$\mathcal{C}_u^{\Delta t}$, $\mathcal{C}_u^{\Delta t}\|f$	context of node u at distance Δt, and feature-based context, resp.
$g_i : \mathcal{C} \to \{0,1\}$	i^{th} LSH function that hashes a node context into a binary value
$K^{\Delta t}$, K	embedding dimension at distance Δt, and output dimension $K = \sum_{\Delta t=1}^{\text{MAX}} K^{\Delta t}$
$\mathbf{h}(\mathcal{S}), \mathbf{h}(\mathcal{S}\|\cdot)$	unconditional and conditional b-bin histogram of values in enclosed set $\mathcal{S}$, resp.
$\mathbf{Z}$	$N \times K$ output binary embeddings or hashcodes

specifies a family of hash functions, $\mathcal{H}$, that maps similar items to the same bucket identified through hash codes with higher probability than dissimilar items [RLU14]. LSH families for different distances have been widely studied, such as SimHash for cosine distance [Cha02], min-hash for Jaccard similarity [BGMZ97], and more. In our work, we leverage LSH to construct similarity-preserving and space-efficient node representations for user stitching.

4.3 Preliminaries and Definitions

Before we introduce Node2bits, we discuss two key concepts that our method is based on: our dynamic heterogeneous network model, and temporal random walks. We give the main symbols and their definitions used in this chapter in Table 4.1.

4.3.1 Dynamic Heterogeneous Network Model

As we mentioned above, we model the user interactions with content, websites, devices etc. as a heterogeneous network, which is formally defined as:

Definition 10 (Heterogeneous Nework). *A heterogeneous network $G = (V, E, \psi, \xi)$ is comprised of (i) a nodeset V and edgeset E, (ii) a mapping $\psi : V \to \mathcal{T}_V$ of nodes to their types, and (iii) a mapping $\xi : E \to \mathcal{T}_E$ to edge types.*

Many graph types are special cases of heterogeneous networks: (**1**) homogeneous graphs have $|\mathcal{T}_V| = |\mathcal{T}_E| = 1$ type; (**2**) k-partite graphs consist of $|\mathcal{T}_V| = k$ and $|\mathcal{T}_E| = k - 1$ types;

(**3**) signed networks have $|\mathcal{T}_V| = 1$ and $|\mathcal{T}_E| = 2$ types; and (**4**) labeled graphs have a single label per node/edge.

Most real networks capture evolving processes (e.g., communication, browsing activity) and thus change over time. Instead of approximating a dynamic network as a sequence of lossy discrete static snapshots $G_1, \ldots, G_T$, we model the *temporal interactions* in a *lossless* fashion as a *continuous-time dynamic network* [NLR+18].

Definition 11 (Continuous-Time Dynamic Network). *A continuous-time dynamic, heterogeneous network $G = (V, E_\tau, \psi, \xi, \tau)$ is a heterogeneous network with E_τ temporal edges between vertices V, where $\tau : E \to \mathbb{R}^+$ is a function that maps each edge to a corresponding timestamp.*

4.3.2 Temporal Random Walks

A walk on a graph is a sequence of nodes where each pair of successive nodes are connected by an edge. Popular network embedding methods generate walks using randomized procedures [PARS14, GL16] to construct a corpus of node IDs or node context. In continuous-time dynamic networks, a *temporally valid* walk is defined as a sequence of nodes connected by edges with non-decreasing timestamps (*e.g.*, representing the order that user-content interactions occurred) and were first proposed and used for embeddings in [NLR+18].

Definition 12 (Temporal Walk). *A temporal walk of length L from v_1 to v_L in graph $G = (V, E, \psi, \xi)$ is a sequence of vertices $\langle v_1, v_2, \cdots, v_L \rangle$ such that $\langle v_i, v_{i+1} \rangle \in E_\tau$ for $1 \le i < L$, and the timestamps are in valid temporal order: $\tau(v_i, v_{i+1}) \le \tau(v_{i+1}, v_{i+2})$ for $1 \le i < (L-1)$.*

4.4 Node2bits: Hash-based Emdedding Framework

Motivated by the task of user stitching, we aim to develop Node2bits to compactly describe each node/entity in the context of *realistic interactions* (Problem 1). Accordingly, Node2bits is required to: **(R1)** support heterogeneous networks where the nodes and edges can be of any arbitrary type (*e.g.*, a user, web page, IP, tag, spatial location); **(R2)** preserve the temporal validity of the events and interactions in the data; **(R3)** scale in runtime to large networks

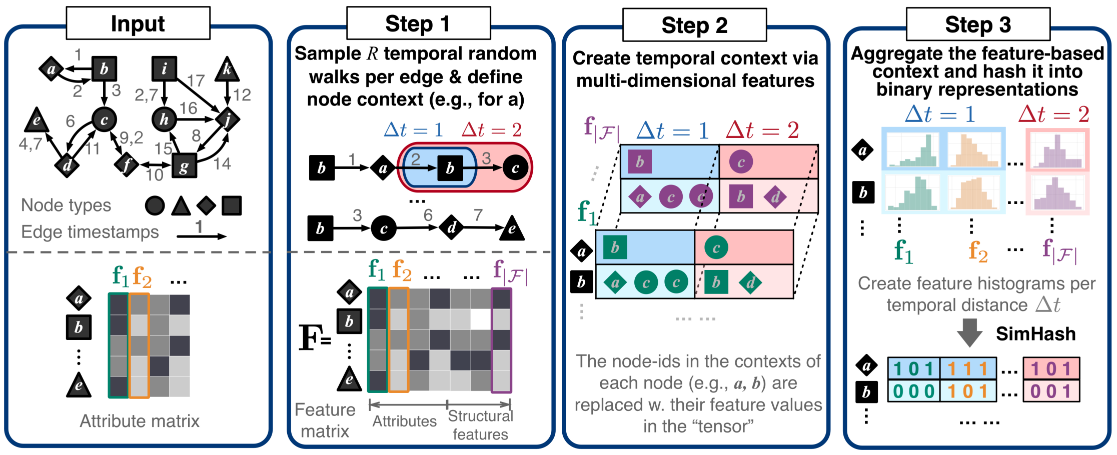

Input
Node types
Edge timestamps
Attribute matrix
Step 1
Sample R temporal random walks per edge & define node context (e.g., for a)
$\Delta t = 1$ $\Delta t = 2$
Feature matrix
Attributes
Structural features
Step 2
Create temporal context via multi-dimensional features
The node-ids in the contexts of each node (e.g., a, b) are replaced w. their feature values in the "tensor"
Step 3
Aggregate the feature-based context and hash it into binary representations
Create feature histograms per temporal distance Δt
SimHash

Figure 4.2: Node2bits workflow. Given a graph and its attribute matrix (optional), Node2bits (1) samples *temporal* random walks to obtain sequences that respect time, derives contexts at different temporal distances (temporal contexts of *a* and *b* are derived from the walk $\{b, a, b, c\}$, as well as the feature matrix F; (2) creates temporal contexts based on multi-dimensional features in F; and (3) aggregates them into feature-based histograms to obtain sparse, binary, similarity-preserving embeddings via SimHash.

with millions of nodes/edges; and **(R4)** scale in memory requirements with space-efficient yet powerful *binary* embeddings that capture ID-independent similarities. Next we detail the three main steps of Node2bits: (§ 4.4.1) Sampling temporal random walks and defining temporal contexts; (§ 4.4.2) Constructing temporal contexts based on multi-dimensional features; (§ 4.4.3) Aggregating and hashing contexts into sparse embeddings. We give the overview of Node2bits in Figure 4.2 and Algorithm IV.1.

4.4.1 Temporal Random Walk Sampling

The first step of Node2bits is to capture interactions in a node's context, which is important for the user stitching task: instead of simple interactions corresponding to pairwise edges, it samples more complex interaction sequences via random walks. But unlike many existing representation learning approaches [PARS14, GL16], our method samples *realistic* interactions in the order that they happen via L-step *temporal* random walks (Definition 12 [NLR+18]), thus satisfying requirement **R2**.

Node2bits defines the *temporal context* $\mathcal{C}_u^{\Delta t}$ of node u at temporal distance Δt as the *collection of entities* that are at Δt-hops away from node u in the sampled random walks. Formally:

$$\mathcal{C}_u^{\Delta t} = \{v \ : \ |\mathbf{w}_L[v] - \mathbf{w}_L[u]| = \Delta t, \ \forall \mathbf{w}_L \in \mathcal{W}\}, \tag{4.1}$$

where $\mathbf{w}_L[\cdot]$ is the index of the corresponding node in the random walk $(\mathbf{w}_L)_{L\in\mathbb{N}}$. For example, in Figure 4.2 (Step 1) the context of node a at temporal distance 2 is $\mathcal{C}_a^{\Delta t=2} = \{c\}$ (highlighted in red). Depending on the temporal context that we want to capture, Δt can vary up to a MAX distance. Intuitively, small values of temporal distance capture more *direct* interactions and similarities between entities. In static graphs, Δt simply corresponds to the distance between nodes in the sampled sequences, without capturing any temporal information.

Temporal locality. The context that is defined above does not explicitly incorporate the *time elapsed* between consecutively sampled interactions. However, when modeling temporal user interactions, it is important to distinguish between short-term and long-term transitions. Inspired by [NLR+18], Node2bits accounts for the *closeness* or *locality* between consecutive contexts (*i.e.*, $\mathcal{C}_u^{\Delta t}$ and $\mathcal{C}_u^{\Delta t+1}$) through different biased temporal walk policies. For example, in the short-term policy, the transition probability from node u to v is given as the softmax function:

$$P[v|u] = \frac{\exp\left(-\tau(u,v)/d\right)}{\sum_{i\in\Gamma_\tau(u)} \exp\left(-\tau(u,i)/d\right)} \tag{4.2}$$

where $\tau()$ maps an edge to its timestamp, $d = \max_{e\in E_\tau} \tau(e) - \min_{e\in E_\tau} \tau(e)$ is the total duration of all timestamps, and $\Gamma_\tau(u)$ is the set of temporal neighbors reached from node u through temporally valid edges. Similarly, in the long-term policy, the transition probability from node u to v is given as in Equation (4.2) but with positive signs in the numerator and denominator.

4.4.2 Temporal Context based on Multi-dimensional Features

The context in Equation (4.1) depends on the node identities (IDs). However, in a multi-platform environment, a single entity may have multiple node IDs, thus contributing to seemingly different contexts. To generate ID-independent contexts that are appropriate for user stitching, we make the temporal contexts attribute- or feature-aware (**R1**), by building on the assumption that corresponding or similar entities have similar features. Formally, we assume that a network may have a set of input node attributes (*e.g.*, IP address, device type), as well as a set of derived topological features (*e.g.*, degree, PageRank), all of which are stored in a $N \times |\mathcal{F}|$ feature matrix $\mathbf{F}$ (Figure 4.2, Step 1). We then generalize our random walks to not only respect time (**R2**) [NLR+18], but also capture this feature information using the notion of attributed/feature-based walks proposed in [ARZ+18]:

Definition 13 (Feature-based Temporal Walk)**.** *A* feature-based temporal walk *of length L from node v_1 to v_L in graph G is defined as a sequence of feature values corresponding to the sequence of vertices in a valid temporal walk (Dfn. 12). For the j^{th} feature $f_{(j)}$, the corresponding feature-based temporal walk is*

$$\langle w_{L,f_{(j)}} \rangle_{L \in \mathbb{N}} = \langle f_{v_1,j}, f_{v_2,j}, \ldots, f_{v_L,j} \rangle, \tag{4.3}$$

where $f_{v_i,j}$ is the value of the j^{th} feature for node v_i, stored in matrix $\mathbf{F}$.

Our definition is general as it allows walks to obey time while each node may have a d-dimensional vector of input attribute values and/or derived structural features, which can be discrete or real-valued [ARZ+18].

4.4.2.1 Temporally-valid, multi-dimensional feature contexts.

Node2bits extends the previously generated temporal contexts to incorporate node features and remove the dependency on node IDs. Following the definition of feature-based temporal walks, given $|\mathcal{F}|$ features, our method generates $|\mathcal{F}|$-dimensional contexts per node u and temporal distance Δt by replacing the node IDs in Equation (4.1) with their corresponding feature values (Figure 4.2, Step 2). Formally, the temporally-valid, multi-dimensional feature contexts are defined as:

$$\mathcal{C}_u^{\Delta t} | f_{(j)} = \{ f_{v,j} \; : \; \forall v \in \mathcal{C}_u^{\Delta t} \} \;\; \forall \text{ feature } f_{(j)} \in \mathcal{F}, \tag{4.4}$$

where $f_{v,j}$ is the value of the j^{th} feature for node v.

4.4.3 Feature-based Context Aggregation and Hashing

The key insight in user stitching is that each user interacts with similarly ‘typed’ entities through similar relations over time: for example, in online-sales logs, a user likely browses similar types of goods in logged-in and anonymous sessions; and in online social networks, accounts sharing near-identical interaction patterns, such as replies or shares, are potentially from the same person. Based on this insight, Node2bits augments the previously generated temporal, multi-dimensional feature contexts with node types (and implicitly the corresponding relations or edge types), which is a key property of heterogeneous networks (**R1**). It subsequently aggregates them and derives similarity-preserving and space-efficient, binary entity representations (**R4**) via locality sensitive hashing.

4.4.3.1 Context Aggregation.

Unlike existing works that aggregate contextual features into a single value such as mean or maximum [HYL17b, RZA18], Node2bits aggregates them into *less lossy* representations: *histograms* tailored to heterogeneous networks by distinguishing between node types (**R1**). Specifically, it models the transitional dependency across node and relation types by further conditioning the derived contexts in Equation (4.4) on the node types $p_i \in \mathcal{T}_V$ (*i.e.*, each temporal context consists of the features of only one node type). We denote the temporal contexts conditioned on both features and node types as $\mathcal{C}_u^{\Delta t}|f, p$. The final histogram representation of node u at temporal distance Δt consists of the concatenation of the histograms over the conditional contexts at Δt (Figure 4.2, Step 3):

$$\mathbf{h}(\mathcal{C}_u^{\Delta t}) = [\mathbf{h}(\mathcal{C}_u^{\Delta t} \mid f_{(1)}, p_1), \mathbf{h}(\mathcal{C}_u^{\Delta t} \mid f_{(2)}, p_1), \dots, \mathbf{h}(\mathcal{C}_u^{\Delta t} \mid f_{(|\mathcal{F}|)}, p_{|\mathcal{T}_V|})]. \tag{4.5}$$

In this representation, the features are binned logarithmically to account for the often skewed distributions of structural features (*e.g.*degree). We note that the histograms can be further extended to edge types as shown in [JRK$^+$19], for example by distinguishing pairs of nodes that are connected by multiple types of edges.

4.4.3.2 Similarity-preserving Representations via Hashing.

Locality sensitive hashing (LSH) has been widely used for searching nearest neighbors in large-scale data mining [RLU14]. In this work, we adopt SimHash [Cha02] to obtain similarity-preserving and space-efficient representations (**R4**) for all the entities in the heterogeneous network based on their aggregated time-, attribute-, and node type-aware contexts given by

Equation (4.5).

Given a node-specific histogram $\mathbf{h}(\mathcal{C}_u^{\Delta t}) \in \mathbb{R}^d$ (with dimensionality $d = |\mathcal{F}||\mathcal{T}_V| \cdot b$, and b being the number of logarithmic bins for the features), SimHash generates a $K^{\Delta t}$-dimensional[1] binary hashcode or sketch $\mathbf{z}_u^{\Delta t}$ by projecting the histogram to $K^{\Delta t}$ random hyperplanes $\mathbf{r}_i \in \mathbb{R}^d$ as follows:

$$g_i(\ \mathbf{h}(\mathcal{C}_u^{\Delta t})\ ) = \text{sign}\left(\mathbf{h}(\mathcal{C}_u^{\Delta t}) \cdot \mathbf{r}_i\right) \tag{4.6}$$

In practice, the hyperplanes do not need to be chosen uniformly at random from a multivariate normal distribution, but it suffices to choose them uniformly from $\{-1, 1\}^d$. The important property of locality sensitive hashing that guarantees that the similarities in the histogram space (which captures the temporal interactions between entities in G) are maintained is the following: for the SimHash family, the probability that a hash function agrees for two different vectors is equal to their cosine similarity. More formally, for two nodes u and v:

$$P(\ g_i(\mathbf{h}(\mathcal{C}_u^{\Delta t})) = g_i(\mathbf{h}(\mathcal{C}_v^{\Delta t}))\ ) = 1 - \frac{\cos^{-1} \frac{\mathbf{h}(\mathcal{C}_u^{\Delta t}) \cdot \mathbf{h}(\mathcal{C}_v^{\Delta t})}{|\mathbf{h}(\mathcal{C}_u^{\Delta t})||\mathbf{h}(\mathcal{C}_v^{\Delta t})|}}{180}. \tag{4.7}$$

In other words, the cosine similarity between nodes u and v in the context-space is projected to the sketch-space and can be measured by the cardinality of matching between $\mathbf{z}_u^{\Delta t}$ and $\mathbf{z}_v^{\Delta t}$, where $\mathbf{z}_\bullet^{\Delta t} = [g_1(\ \mathbf{h}(\mathcal{C}_\bullet^{\Delta t})\ ), g_2(\ \mathbf{h}(\mathcal{C}_\bullet^{\Delta t}), \ldots, g_{K^{\Delta t}}(\ \mathbf{h}(\mathcal{C}_\bullet^{\Delta t})]$.

For each node u in G, the final binary representation is obtained by concatenating the hashcodes for contexts at different temporal distances Δt, resulting in a K-dimensional vector (since $K = \sum_{\Delta t=1}^{\text{MAX}} K^{\Delta t}$):

$$\mathbf{z}_u = [\mathbf{z}_u^{\Delta t=1}\ \mathbf{z}_u^{\Delta t=2}\ \ldots\ \mathbf{z}_u^{\Delta t=\text{MAX}}] \tag{4.8}$$

where we replace the -1 bits with 0s to achieve a more space-efficient representation (**R4**). An example is shown in the second half of Step 3 in Figure 4.2, where the blue shades denote histograms and sketches for contexts in temporal distance $\Delta t = 1$, and red shades correspond to $\Delta t = 2$. Thus, the K-dimensional representation for each node, $\mathbf{z}_u \in \{0, 1\}^K$, captures the

[1]We assume that the length of each sketch at distance Δt is given as $K^{\Delta t} = \frac{K}{\text{MAX}}$.

Algorithm IV.1 Node2bits Framework

```
Require: (un)directed heterogeneous graph $G(V, E, \psi, \xi)$, # random walks $R$ per edge, max walk length $L$, max temporal
    distance MAX, embedding dimensionality $K^{\Delta t}$ at dist. $\Delta t$
1 For each edge $e$, perform $R$ temporal walks based on the short- or long-term policy (§ 4.4.1)
2 Obtain temporal contexts $\mathcal{C}_u^{\Delta t}$ for each node $u$ at temporal distances $\Delta t \leq$ MAX via Eq. (4.1)
3 Construct feature matrix $\mathbf{F}$ with node attributes (if avail.) and topological features (§ 4.4.2)
4 Derive feature-based temporal contexts $\mathcal{C}_u^{\Delta t}|f_{(j)}$ by replacing $v \in \mathcal{C}_u^{\Delta t}$ with the feature value $f_{v,j}$, as shown in Eq. (4.4)
5 for each temporal distance $\Delta t = 1, \ldots$,MAX and node $u \in V$ do
6     Obtain $u$'s final histogram $\mathbf{h}(\mathcal{C}_u^{\Delta t})$ over its contexts using Eq. (4.5)
7     Obtain a $K^{\Delta t}$-dim, sparse, binary hashcode $z_u^{\Delta t}$ based on (modified) SimHash (§ 4.4.3)
8 Obtain the binary N2B embeddings $\mathbf{z}_u$ of all nodes across temporal distances $\Delta t$ via Eq. (4.8)
9 Perform (un)supervised user stitching via binary classification or hashing (§ 4.5.1,4.5.3)
```

similarities between time-, feature- and node type-aware histograms across multiple temporal distances Δt. The similarity between two nodes' histograms can be quickly estimated as the proportion of common bits in their binary representations $\mathbf{z}_\bullet$.

Given these representations, we can perform user stitching by casting the problem as supervised binary classification or an unsupervised task based on the output of hashing (Equation (4.8)), which we discuss in § 4.5.1. Putting everything together, we give the pseudocode of Node2bits in Algorithm IV.1. The runtime computational complexity of Node2bits is $\mathcal{O}(MRL + NK)$, which is linear to the number of edges when $M \gg N$ as K is relatively small (**R3**). The output space complexity is $\mathcal{O}(NK)$-bit. Node2bits requires even less storage if the binary vectors are represented in the sparse format (see § 4.5.4 for empirical results).

4.4.4 Complexity Analysis

Time Complexity. The runtime complexity of Node2bits includes deriving (1) the set of R temporal random walks of length up to L, which is $\mathcal{O}(MRL)$ in the worst case; (2) the feature values of nodes in the walks from step (1); and (3) hashing the feature values of nodes in the context through random projection, which is $\mathcal{O}(NK)$. Thus, the total runtime complexity is $\mathcal{O}(MRL + NK)$, which is linear to the number of edges when $M \gg N$ as K is relatively small (**R3**).

Runtime Space Complexity. The space required in the runtime consists three parts: (1) the set of temporal random walks (represented as vectors) per edge with complexity $\mathcal{O}(MRL)$, (2) the histograms of feature contexts $N|\mathcal{F}||\mathcal{T}_V|$, and (3) the set of randomly-generated hyperplanes NK. Therefore, the total runtime space complexity is $\mathcal{O}(MRL+N(|\mathcal{F}||\mathcal{T}_V|+K))$.

Table 4.2: Network statistics and properties for our six real-world datasets. 'D': directed; 'W': weighted; 'H': heterogeneous; 'T': temporal network.

Data	**Nodes**	**Edges**	$\lvert T_V \rvert$	**D**	**W**	**H**	**T**
citeseer	4460	2892	2		✓	✓	
yahoo	100,058	1,057,050	2	✓	✓	✓	
bitcoin	3,783	24,186	1	✓	✓		✓
digg	283,183	6,473,708	2			✓	✓
wiki	1,140,149	7,833,140	1	✓			✓
comp-X	5,500,802	5,291,270	2	✓	✓	✓	✓

Output Space Complexity. The output space complexity of Node2bits is $\mathcal{O}(NK)$-bit. The space required to store binary vectors is guaranteed to be 32× less than vectors represented with real-value floats (4 bytes) with the same dimension. In practice, Node2bits requires even less storage if the binary vectors are represented in the sparse format (see Section 4.5.4 for empirical results).

4.5 Experiments

We perform extensive experiments on real-world heterogeneous networks to answer the following questions: (**Q1**) Is Node2bits effective in the user stitching task, and how does it compare to traditional stitching and embedding methods? (§ 4.5.2-4.5.3) (**Q2**) Does Node2bits have low space requirements, and is it more space-efficient than the baselines? (§ 4.5.4) (**Q3**) Is Node2bits scalable? (§ 4.5.5)

4.5.1 Experimental Setup

We ran our analysis on Mac OS platform with 2.5GHz Intel Core i7, 16GB RAM.

4.5.1.1 Data

We use five real-world heterogeneous networks from the Network Repository [RA15a], as well as a real, proprietary user stitching dataset, 'Company X' web logs. The latter data form a temporal heterogeneous network consisting of web sessions of user devices and their IP addresses. High degree nodes representing anomalous behavior (*e.g.*, bots or public WiFi hotspots) are filtered out. Our framework is also capable of modeling domain-specific features, such as user-agent strings and geolocation [KKP+17], if this is available. Even without them, however, it achieves strong performance. We give the statistics of all the networks in Table 4.2, and additional details as follows.

- **citeseer**: CiteSeerX is an undirected, heterogeneous network that contains the bipartite relations between authors and papers they contributed.
- **yahoo**: Yahoo! Messenger Logs is a heterogeneous network capturing message exchanges between users at different locations (node attribute).
- **bitcoin**: soc-bitcoinA is a who-trusts-whom network on the Bitcoin Alpha platform. The directed edges indicate user ratings.
- **digg**: This heterogeneous network consists of users voting stories that they like and forming friendships with other users.
- **wiki**: wiki-talk is a temporal homogeneous network capturing Wikipedia users editing each other's Talk page over time.
- **comp-X**: A temporal heterogeneous network is derived from a company's web logs and consists of web sessions of users and their IPs. In the stitching task, we predict the web session IDs that correspond to the same user.

4.5.1.2 Task Setup

With the exception of § 4.5.3, we cast the user stitching task as a binary classification problem, where for each pair of nodes we aim to predict whether they correspond to the same entity (*i.e.*, we should stitch them). We use logistic regression with regularization strength 1.0 and stopping criterion 10^{-4}.

For the real user stitching data, we use the held-out, ground-truth information to evaluate our method. For the five real-world networks without known user pairs, we introduce user replicas following a similar procedure to [BG07]: we sample 5% of the nodes with degrees larger than average, introduce a replica u' for each sampled node u, and distribute the original edges between u and u'. Specifically, each edge remains connected to u with probability p_1, otherwise it connects to the replica node u'. Additionally, each edge that is incident to u has probability p_2 to also connect to u'. Unless otherwise specified, we use $p_1 = 0.6$ and $p_2 = 0.3$.

Given the positive replica pairs, we sample an equal number of negative pairs uniformly at random and include these in the training and testing sets. Comp-X, the dataset with ground-truth replicas, also has pre-defined approximately 50/50 training-testing splits that we use. Afterwards, embeddings are derived for each node pair by concatenation: $[\mathbf{z}(u), \mathbf{z}(u')]$. Using these node pair embeddings, we learn a logistic regression (LR) model and use it to predict the node pairs that should be stitched in the held-out test set. These are the nodes that correspond to the same entity. We measure the predictive performance of all the methods in terms of AUC, accuracy and $F1$ score.

4.5.1.3 Baselines

We compare to various methods that target different graph types:

- **Homogeneous graphs**:

  Static– **(1)** Spectral embedding or SE [vL07], **(2)** LINE [TQW+15b], **(3)** DeepWalk or DW [PARS14], **(4)** node2vec or n2vec [GL16], **(5)** struc2vec or s2vec [RSF17], and **(6)** DNGR [CLX16].

  Temporal– **(7)** CTDNE [NLR+18].

- **Heterogeneous graphs**:

  (8) Common neighbors (CN) [BG07], **(9)** metapath2vec or m2vec [DCS17], and **(10)** AspEm [SGZ+18].

The baselines are configured to achieve the best performance, for $K = 128$-dimensional embeddings, according to the respective papers. We configured all the baselines to achieve the best performance according to the respective papers. For all the baselines that are based on random walks (*i.e.*, node2vec, struc2vec, DeepWalk, metapath2vec, CTDNE), we set the number of walks to 20 and the maximum walk length to $L = 20$. For node2vec, we perform grid search over $p, q \in \{0.25, 0.50, 1, 2, 4\}$ as mentioned in [GL16] and report the best performance. For metapath2vec, we adopt the recommended meta-path "Type 1-Type

2-Type 1" (*e.g.*, type 1 = author; type 2 = publication). In DNGR, we set the random surfing probability $\alpha = 0.98$ and use a 3-layer neural network model where the hidden layer has 1024 nodes. We use 2nd-LINE to incorporate 2nd-order proximity in the graph. For all the embedding methods, we set the embedding dimension to $K = 128$. Unlike those, CN outputs clusters, each of which corresponds to one entity.

4.5.1.4 Node2bits Setup & Variants

Similar to the baselines, Node2bits performs $R = 10$ walks per edge, with length up to $L = 20$, and we set the max temporal distance MAX = 3. On the largest dataset, Comp-X, we use a maximum walk length $L = 5$ and temporal distance MAX = 2. While various node attributes can be given as input to Node2bits, for consistency we derive and use the total, in-/out-degree of each node in $\mathbf{F}$.

We experiment with different variants of Node2bits (or N2B for short): (1) **NODE2BITS-0** applies to static networks; (2) **NODE2BITS-SH** uses the short-term tactic in the random walks (§ 4.4.1); (3) **NODE2BITS-LN** uses the long-term tactic; and (4) **NODE2BITS-U** targets unsupervised user stitching, which most baselines cannot handle (except for CN). To explore our method's performance in unsupervised settings (§ 4.5.3), we directly 'cluster' the LSH-based, binary node representations $\mathbf{z}_u$ generated by NODE2BITS-0. The idea is that nodes that hash to the same 'bucket' likely map to the same entity and should be stitched. To map entities to buckets we use the banding technique [RLU14]: per band—one per representation $\mathbf{z}^{\Delta t}$ at temporal distance Δt—we apply AND-construction on the output of bit sampling, and then OR-construction across the bands.

4.5.2 Accuracy in Supervised User Stitching

We start by evaluating the predictive performance of Node2bits for supervised user stitching on both static and temporal networks.

4.5.2.1 Static Networks

Here we evaluate the effectiveness of multi-dimensional feature contexts. Since static networks lack temporal information, Node2bits performs random walks similarly to existing

Table 4.3: Entity resolution results for *static* networks. Our method outperforms all the baselines. | OOT = Out Of Time (6h); OOM = Out Of Memory (16GB). The asterisk * denotes statistically significant improvement over the best baseline at $p < 0.05$ in a two-sided t-test.

	Metric	CN	SE	LINE	DW	n2vec	s2vec	DNGR	m2vec	AspEm	N2B-0
citeseer	**AUC**	0.9141	0.4846	0.5481	0.5614	0.6188	0.9344	0.5015	0.5546	0.5049	0.9480*
	ACC	0.9141	0.5045	0.5372	0.5579	0.6211	0.8936	0.4688	0.5357	0.5223	0.9196*
	Precision	1.0	0.5051	0.5361	0.5495	0.5983	0.8304	0.4667	0.5392	0.5217	0.8672
	Recall	0.8482	0.4464	0.5532	0.6421	0.7368	0.9894	0.4375	0.4911	0.5357	0.9911
	F1	0.9137	0.5028	0.5371	0.5547	0.6159	0.8926	0.4682	0.5348	0.5222	0.9192*
yahoo	**AUC**	0.6851	0.5378	0.8050	0.7640	0.7636			0.8233	0.4938	0.8088
	ACC	0.6851	0.4760	0.7771	0.7117	0.7233			0.7827	0.5018	0.8010
	Precision	1.0	0.4497	0.7500	0.7063	0.7126	OOT	OOM	0.7126	0.5018	0.7481
	Recall	0.3703	0.2143	0.8313	0.7249	0.7485			0.7485	0.5030	0.9076
	F1	0.6505	0.4375	0.7764	0.7117	0.7231			0.7823	0.5018	0.7987

works to collect nodes in structural contexts. The main difference lies in representing diverse feature histograms. We run Node2bits against both homogeneous and heterogeneous baselines as shown in Table 4.3, and observe that it performs the best in most evaluation metrics on both graphs. Node2bits outperforms existing random-walk based methods as expected: node IDs in the contexts is distorted by the replicas generated, thus feature-based methods should prevail. This is also validated by the results for struc2vec, which captures the equivalency of structural feature sequences in embeddings. metapath2vec and LINE achieve promising result on yahoo but not on citeseer, as the latter is an undirected bipartite graph, node distributions of the 2-order contexts explored by LINE are highly correlated and indistinguishable for stitching. On the contrary, CN (common-neighbors) yields promising result on citeseer but not yahoo. This is likely due to the graph structure, which we explain in more detail in Sec.4.5.3. We encountered out-of-memory errors for DNGR due to the algorithmic complexity and out-of-time-limit for struc2vec.

Conclusion 1. *On static graphs,* Node2bits *achieves comparable performance in AUC, and slightly better $F1$ score with* $0.60\% - 2.10\%$ *improvement over baselines in the stitching task.*

4.5.2.2 Temporal Networks

Table 4.4 depicts the stitching performance of Node2bits using both the short- and long-term tactics against the same set of baselines used in static graphs as well as CTDNE, an embedding framework designed for temporal graphs. We exclude metapath2vec, as metapaths are not meaningful in homogeneous networks, and the method ran out of time for the

Table 4.4: Entity resolution results for *temporal* networks: strong performance for Node2bits variants. | OOT = Out Of Time (6h); OOM = Out Of Memory (16GB); * denotes statistically significant improvement over the best baseline at $p < 0.05$ in a two-sided t-test.

	Metric	CN	SE	LINE	DW	n2vec	s2vec	DNGR	AspEm	CTDNE	N2B-0	N2B-SH	N2B-LN
bitcoin	**AUC**	0.7474	0.5828	0.6071	0.6306	0.6462	0.8025	0.5909	0.5344	0.6987	0.7584	0.7609	0.7380
	ACC	0.7174	0.5842	0.5842	0.6158	0.6158	0.7263	0.5526	0.5316	0.6000	0.7211	0.7268	0.6737
	Precision	1.0	0.6250	0.5755	0.6146	0.6122	0.7263	0.5510	0.5326	0.6234	0.7100	0.7647	0.6667
	Recall	0.4947	0.4211	0.6421	0.6211	0.6316	0.7263	0.5684	0.5158	0.5053	0.7474	0.6842	0.6947
	F1	0.7001	0.5728	0.5828	0.6158	0.6157	0.7263	0.5525	0.5315	0.5964	0.7209	0.7271	0.6735
digg	**AUC**	0.6217	0.5171	0.7878	0.7398	0.7445			0.5105	0.6967	0.8185*	0.7611	0.7587
	ACC	0.6217	0.5152	0.7694	0.6971	0.7013			0.5088	0.5915	0.7982*	0.7418	0.7444
	Precision	1.0	0.5564	0.7371	0.6763	0.6809	OOT	OOM	0.5087	0.6110	0.7453	0.7185	0.7171
	Recall	0.2434	0.0516	0.8376	0.7562	0.7576			0.5138	0.5058	0.9060	0.7952	0.8071
	F1	0.5585	0.3770	0.7683	0.6960	0.7003			0.5088	0.5884	0.7958*	0.7411	0.7433
wiki	**AUC**	0.6997		0.7854					0.5374	0.7707	0.8230	0.8259*	0.8214
	ACC	0.6997		0.7132					0.5141	0.6488	0.7145	0.7510*	0.7103
	Precision	1.0	OOT	0.7274	OOM	OOM	OOT	OOM	0.5011	0.7174	0.7972	0.8268*	-
	Recall	0.3994		0.6819					0.4993	0.4910	0.5753	0.6349	-
	F1	0.6699		0.7129					0.5141	0.6398	0.7088	0.7476*	0.7067
comp-X	**AUC**	0.5970		0.5000					0.5213		0.8095*	0.7496	0.7525
	ACC	0.5970		0.6757					0.5103		0.8414*	0.7959	0.7975
	Precision	-	OOM	-	OOM	OOM	OOT	OOM	-	OOM	-	-	-
	Recall	-		-					-		-	-	-
	F1	0.5189		0.4032					0.5103		0.8154*	0.7581	0.7606

heterogeneous networks. We observe that NODE2BITS-SH outperforms NODE2BITS-LN in most cases, which is reasonable because NODE2BITS-LN derives shorter contexts constrained by temporal-order. We also justify the effectiveness of temporal random walk by comparing it with both NODE2BITS-0 and static baselines where we only make use of the graph structures without specifying edge timestamps. We observe that NODE2BITS-0 is the best-performing method for the digg dataset and Comp-X over the temporal variants of Node2bits. The reason behind this is that there is a tradeoff in constraining temporal walks to respect time: we more accurately model realistic sequences of events at the cost of restricting the possible context. On these particular temporal graphs, walks may already be limited in length by the bipartite structure, so the latter cost becomes more appreciable. Nevertheless, both static and dynamic versions of Node2bits almost always outperform other baselines. In particular, across all datasets, NODE2BITS-SH still outperforms the temporal baseline, CTDNE in all cases, which further demonstrates the effectiveness of multi-feature aggregation.

Node2bits variants outperform the static methods in nearly all cases except the bitcoin dataset where NODE2BITS-SH achieves lower AUC than struc2vec but higher ACC and $F1$-score. This is because Node2bits loses some information when representing the node contexts as binary vectors comparing with real-value representation. However, we consider this loss mild as Node2bits still outperforms all the other static baselines. In addition, struc2vec ran out of time on the larger datasets while Node2bits achieves promising performance efficiently with $3.90\% - 5.16\%$ improvement in AUC and $3.58\% - 4.87\%$ improvement in $F1$ score than the best baseline method. At the same time, our approach uses much less information than the static methods, since the length of the temporal walks are typically shorter than random walks that do not have to respect time.

Conclusion 2. *Dynamic and static variants of* Node2bits *outperform the other baselines by up to* 5.2% *in AUC and* 4.9% *in* $F1$ *score. Between the two dynamic variants, the short-term tactic performs better than the long-term one.*

4.5.3 Accuracy in Unsupervised User Stitching

As mentioned in § 4.5.1, Node2bits can naturally perform unsupervised user stitching by leveraging the generated node representations as hashcodes. Only nodes mapped to the same 'buckets' are candidates for stitching together. This process allows us to stitch entities without involving quadratic comparisons between all pairs of nodes in the graph. Similarly, CN outputs a set of nodes sharing a certain amount of neighbors as the candidates to be stitched together. We evaluate the quality of hashing given by NODE2BITS-U against CN, and make use of the candidates to predict the testing set of node pairs given by following the same setup in § 4.5.2 in an unsupervised scheme.

Based on the results in Table 4.5, we observe that NODE2BITS-U outperforms CN on every dataset other than citeseer. The reason is that in this "author contributes to paper" dataset, author references appearing in the same set of papers have high probability to correspond to the same researcher in reality. Therefore the assumption made by CN suits well this scenario, whereas Node2bits hashes nodes with similar features in the context instead of those with similar neighbor identities (IDs). For datasets with less strict cross-type relationship, Node2bits achieves $2.81\% - 15.12\%$ improvement in accuracy ACC and $4.96\% - 26.66\%$ improvement in $F1$ score (including digg, another bipartite graph with inner connected components of the same node types).

Conclusion 3. *The unsupervised variant of* Node2bits, NODE2BITS-U, *outperforms CN on most graphs.*

Table 4.5: Unsupervised stitching performance between CN and Node2bits.

Metric	citeseer		yahoo		bitcoin		digg		wiki	
	CN	N2B-U	CN	N2B-U	CN	N2B-U	CN	N2B-U	CN	N2B-U
ACC	0.9141	0.8661	0.6851	0.7553	0.7474	0.7684	0.6217	0.7157	0.6997	0.7350
F1	0.9137	0.8660	0.6505	0.7518	0.7301	0.7663	0.5585	0.7074	0.6699	0.7349

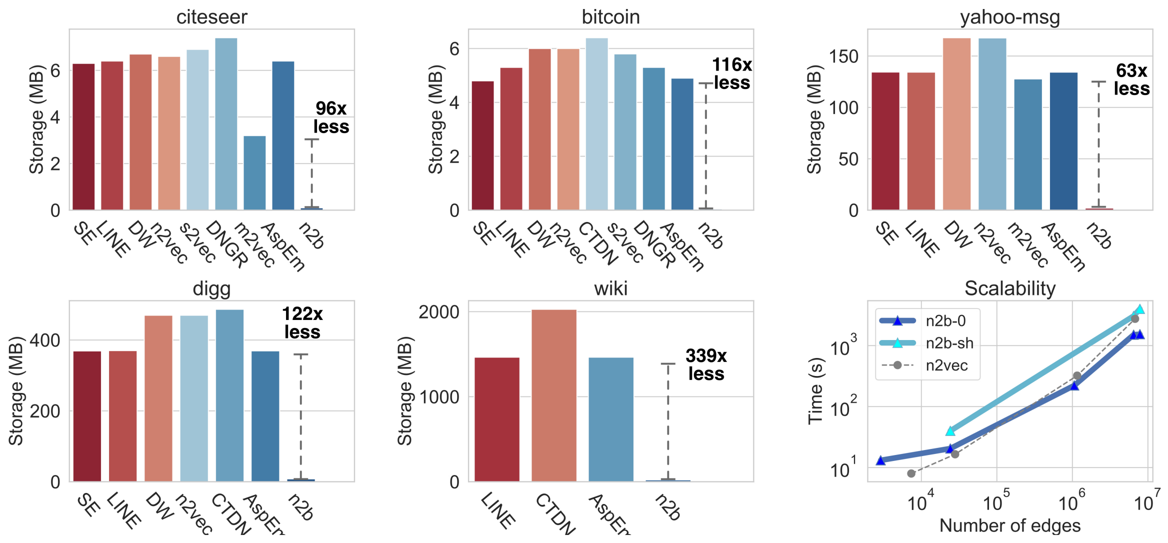


Figure 4.3: First 5 plots: output storage in MB for all the methods that completed successfully in five datasets. Node2bits is also shown to be scalable for large graphs.

4.5.4 Output Storage Efficiency

Next we evaluate space efficiency of our proposed method over baselines that output node embeddings. Instead of real-value matrices, the binary hashcodes generated by Node2bits can be stored in the sparse format so presumably it should take trivial storage. We visualize the storage requirements in Figure 4.3.

Conclusion 4. *Compared to the other methods, Node2bits uses between* 63× *and* 339× *less space (while always achieving comparable or better stitching performance as shown in § 4.5.2).*

4.5.5 Scalability

To evaluate the scalability, we report the runtime of applying Node2bits to obtain node representations for the datasets shown in Table 4.2 versus their numbers of edges. We note that NODE2BITS-SH runs only on temporal networks, i.e., a subset of the datasets. We also visualize the runtime of node2vec as reference, as it is designed for large graphs and is implemented in the same language (Python). Based on the last subplot in Figure 4.3, we observe that Node2bits scales similarly as node2vec with less runtime space as node2vec ran out of memory on the largest dataset (wiki). As shown in the proof, the worst-case time complexity is linear in the edges. We give the exact runtimes in Table 4.6.

Table 4.6: Comparison between Node2bits and baselines in terms of runtime (in seconds). Note the runtime of dynamic Node2bits (short-term) for the *temporal* networks is shown in parentheses.

	citeseer	**yahoo**	**bitcoin**	**digg**	**wiki**
SC	23.72	766.42	4.80	8091.09	1
LINE*	144.94	223.87	134.48	227.28	415.00
DW*	8.90	209.72	16.99	2115.86	-
n2v*	7.99	222.14	15.91	2751.91	-
CTDNE	-	-	13.25	2227.66	4217.19
s2vec*	325.38	-	897.2	-	-
DNGR	128.63	-	97.09	-	-
m2vec	125.98	-	-	-	-
AspEm	0.62	4.70	0.71	15.318	386.24
CN	0.58	19.59	0.70	63.95	109.11
N2B	13.15	221.84	20.52 (39.97)	1507.95 (3062.13)	1537.24 (3997.85)

4.6 Conclusion

We have proposed a hash-based network representation learning framework for identity stitching called Node2bits. It is both time- and attribute-aware, while also deriving space-efficient sparse binary embeddings of nodes in large temporal heterogeneous networks. Node2bits uses the notion of feature-based temporal walks to capture the temporal and feature-based information in the data. Feature-based temporal walks are a generalization of walks that obey time while also incorporating features (as opposed to node IDs). Using these walks, Node2bits generates contexts/sequences of temporally valid feature values. Experiments on real-world networks demonstrate the utility of Node2bits as it outputs space-efficient embeddings that use orders of magnitude less space compared to the baseline methods while achieving better performance in user stitching. An important practical consideration in the application of our work to user stitching is the balance of greater personalization with user privacy.

CHAPTER V

Evaluating Temporal Summaries and Node Embedding

This chapter is based on work that will appear at WSDM 2022 [JKRK22].

5.1 Introduction

In Chapter IV, we proposed Node2bits to summarize the structural features of temporal heterogeneous graphs, and model the temporal proximity through temporal random walks. Recently, temporal node embedding has attracted significant attention in both academia and industry, so many novel approaches have been proposed where the temporal proximity is generally modeled through complex but less-interpretable deep learning techniques, such as RNN, LSTM. While these recent approaches have demonstrated effectiveness on machine learning tasks, it is unclear if complex models are necessary. Motivated by this, in this chapter, we perform a systematic study of temporal embedding models. We propose an efficient framework that includes two different graph time-series representation and generalizes the static node embeddings to the dynamic settings through interpretable network models. We compare the performance with state-of-the-art temporal node embedding approaches and show that our framework performs at least equally well on temporal link prediction tasks. Such finding demonstrates the importance of model interpretability in designing node embedding approaches.

Real-world networks that capture the interaction between entities are growing rapidly, for example, the Internet [CO02], various online social networks (*e.g.*, Facebook, Snapchat), citation networks in academia [LKF05]. Specifically, when nodes and edges continuously change over time with addition, deletion (e.g., a phone call, an email, or physical proximity between two entities), we have a particular type of evolving network structure. Learning an appropriate network representation (embedding) that accurately captures the temporal dynamics and temporal structural properties of these entities is important for many downstream time-series forecasting/prediction tasks such as recommendation and entity resolution. Most recent research efforts devoted in the field follow the common pipeline: given a time-series of graphs, $\mathcal{G} = \{G_1, \cdots, G_k, \cdots, G_T\}$, modeling the individual graph structures (within-snapshot property) along with the temporal dependency (across-snapshot relation), and deriving node embeddings that incorporate both perspectives. While these works show advantage from various perspectives, the promising performance comes at the cost of time and model complexity, such as introducing extra transition variables to reflect the temporal dependency between snapshots [GKHL18], or latent weights on edges between snapshots [SWG+20, PDC+20].

In this work, we propose a general framework that simplifies the above process and can generalize *any* static embedding method to a more powerful and predictive dynamic embedding method without introducing transitional variables. The framework consists of three components: (**C1**) a graph time-series representation, (**C2**) a temporal network model that appropriately models and weights the temporal dependencies in the graph time-series, and (**C3**) a base embedding method to learn a time-series of embeddings along with a fusion mechanism to derive the final temporal node embeddings. The framework is highly expressive as any unique combination of **C1**-**C3** gives rise to a new dynamic embedding method.

While previous works on dynamic modeling and embedding have focused on representing the stream of timestamped edges [NLR+18] using a time-series of graphs based on a specific time-scale τ (*e.g.*, $\tau = 1$ hour, or 1 month) [GKHL18, GCC19, SGR19, LKF05, ZYR+18, SWG+20], we instead propose the notion of an ϵ-graph time-series that uses a fixed number of edges for each graph in the time-series. Theoretically, by fixing the number of edges to be ϵ in each graph, we ensure that every graph in the sequence has an equal probability of giving rise to the same exact distribution of higher-order graphlets and other structural patterns[1],

[1]This is in contrast to graphs with different amounts of edges. E.g., given two arbitrary graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ where $|E_1| \ll |E_2|$, then the counts of all $k \in \{3, 4, \ldots\}$-node network motifs (graphlets) in G_2 are almost surely larger than G_1.

and therefore, the new ϵ-graph time-series forces the models to avoid capturing simple trivial differences due to edge counts, and instead, allow the models to capture actual *structural changes* to the graphs over time.

We also introduce a number of important temporal models that can be leveraged over any graph time-series representation of the edge stream. The first temporal model is based on the notion of a temporal reachability graph (TRG). TRG is derived by transforming a dynamic graph into a static graph where an edge from u to w indicates a temporal walk. The second temporal model is called a weighted temporal summary graph (TSG). Notably, a weighted temporal summary graph captures the temporal recurrence and recency of links by appropriately weighting links with respect to a function f that assigns larger weights to links that are more recent and recurrent whereas links that occur in the more distant past are assigned lower weights. All temporal models can leverage either the new ϵ-graph or τ-graph time-series representation.

This paper aims to provide a systematic exploration of the most useful graph time-series representations and temporal network models (used to incorporate the temporal dependencies into base embedding methods) in downstream temporal prediction tasks. To the best of our knowledge, this is the first work of this kind. Our primary findings are: (1) node embeddings derived from the ϵ-graphs outperforms the τ-graph time-series in the predictive task with higher stableness, and (2) by composing the static node embedding approaches with classic temporal models such as TRG or TSG, our proposed framework performs comparably or even better than recent dynamic embedding approaches with less complexity. Based on these findings, we hope that this work will benefit future research on developing and evaluating better dynamic embedding methods, as well as practitioners who deploy temporal graph embedding in various applications due to its simplicity and effectiveness in performance. Our main contributions are as follows:

- **General Framework.** We describe a general framework for leveraging graph stream data and classic temporal network models for prediction-based applications that can generalize any static graph embedding method.

- **Powerful Graph Time-series Representation.** We introduce the notion of a ϵ-

graph time-series and show its superiority over the conventional way of discretizing the edge stream based on the application time-scale (e.g., hour, day).

- **Systematic Study.** Our framework allows us to systematically study 42 dynamic node embeddings by combining time-series representations, temporal network models, and static methods. Strikingly, our empirical analysis on 8 real-world networks shows that our framework achieves comparable or better predictive performance than existing state-of-the-art, but more complex, *dynamic* node embedding methods.

5.2 Related Work

Snapshot-based approaches.

Most temporal embedding approaches break down the graph into graph-time series based on the application time-scale (1 month, etc.) up to a certain point k, and then derive features from them to make inference on graphs at $k+1$. One direction is to look into the most recent snapshot, for instance, DANE [LDH$^+$17] proposes to embed both nodes and the associated attributes in the graph by minimizing the loss of reconstruction of the snapshot at a given times point k: $\frac{1}{2}\Sigma_{i,j}\mathbf{A}_{ij}^{(k)}||y_i - y_j||^2$, and update the embeddings for snapshot at $k+1$ based on the change of graph structure and node attributes. DynGEM [GKHL18] adopts the deep auto-encoder to generate the nonlinear embeddings from the snapshot at k while addressing stability. TIMERS [ZCP$^+$18] models the relative changes in adjacency matrices between snapshots and leverages incremental SVD to derive embeddings. A more popular direction is to track back a certain number of snapshots from the time point k by deriving node embeddings from each individual tracked snapshot and then merging them through specific operation. Dyngraph2vec [GCC19] leverage totally l snapshots to predict the snapshot at $k+1$. It leverages various deep architectures (*i.e.*, auto-encoder, RNN) to derive latent features by minimizing loss of reconstruction error: $||f(\mathbf{A}_{k-l+1}, \cdots, \mathbf{A}_k) - \mathbf{A}_{k+1}||_F^2$. tNodeEmbed [SGR19] is an end-to-end framework based on node embeddings derived from individual snapshots using static methods. The embeddings are merged by minimizing the loss of specific tasks (*i.e.*, link prediction and node classification) through LSTM. DySAT [SWG$^+$20] leverages

Table 5.1: Qualitative comparison of existing embedding methods on temporal graphs. The graph time-series representation used by the method (application time-scale, or fixed number of edges), the type of temporal model used, and type of embedding fusion used (if any).

	Representation		Temporal Model		
	Time-scale (τ)	#Edges (ϵ)	Snapshot	Weighting	Emb. Fusion
DANE [LDH+17]	✓	✗	✓	✗	✗
DynGem [GKHL18]	✓	✗	✓	✗	✓
TIMERS [ZCP+18]	✓	✗	✓	✗	✗
Dynagraph2vec [GCC19]	✓	✗	✓	✗	✓
tNodeEmbed [SGR19]	✓	✗	✓	✓	✓
EvolveGCN [PDC+20]	✓	✗	✓	✓	✓
DySAT [SWG+20], DyHATR [XYJ+20]	✓	✗	✓	✗	✓
our framework [SWG+20]	✓	✓	✓	✓	✓

the notion of self-attention to compute node representations by jointly employing graph structural property and temporal dynamics. Similarly, DyHATR [XYJ+20] proposes a the hierarchical attention model to capture both the heterogeneity and temporal attention using GRU/LSTM to model the temporal evolution. EvolveGCN [PDC+20] uses GCN to generate node embeddings for the past snapshots, and learns the hidden parameters for the next using GRU/LSTM. Unlike the above methods that jointly explore the graph structural changes with the evolution of the #edges, our proposed ϵ-graph time series does not require the specification of time-scales.

Sequential-interaction-based approaches.

There is another line of works that studies the sequential interaction between nodes in the graph. CTDNE [NLR+18] is the first approach to learn embeddings directly from the stream of timestamped edges at the finest temporal granularity. In that work, they proposed the notion of temporal walks and used it for embeddings [NLR+18]. More recently, node2bits [JHRK19] expanded on this idea by incorporating features in the temporal walks and hashing them. Alternatively, some other work has modeled the node-specific temporal dynamics as the point process where the probability of interaction is represented through different intensity functions. HTNE [ZLL+18] proposes to model the node evolution through the Hawkes process. JODIE[KZL19] models the sequential interaction in bipartite graphs to predict the change of embedding trajectory over time instead of interaction probability. CTDNE, HTNE and JODIE are designed to handle continuously sequential data, which is not the scope of this paper.

5.3 Data

In this study we adopt a variety of real-world temporal networks from SNAP [LK14] and NR [RA15a]. We provide the brief description of the datasets as follows.

Table 5.2: Network statistics and properties.

Data	$\lvert V\rvert$	$\lvert E\rvert$	Type	Timespan
`enron`	151	50,572	Unipartite	38 months
`bitcoin`	3,783	24,186	Unipartite	63 months
`wiki-elec`	7118	107,071	Unipartite	47 months
`stackoverflow`	24,818	506,550	Tripartite	79 months
`fb-forum`	899	33,720	Unipartite	24 weeks
`reallity-call`	6,809	52,050	Unipartite	16 weeks
`wiki-edit`	8,227	157,474	Bipartite	32 days
`contacts-dublin`	10,972	415,912	Unipartite	69 days

The detailed description of the experimental graph datasets is given as follows.

- `enron`[2] records email exchanging between empolyees of Enron from May, 1999 to June, 2002.
- `bitcoin`[3] is a who-trusts-whom network of people who trade using bitcoins from Nov, 2010 to Feb., 2017. We study the user connectivity by dropping the edge signs.
- `wiki-elec`[2] contains the voting history based on the Wikipedia page edit history from Mar., 2004 to Jan., 2008.
- `stackoverflow`[3] is a temporal network consisting of three types of interactions on the stack exchange web site Math Overflow: a user answers questions, a user comments on questions, and a user comments on answers.
- `wiki-edit`[4] is a public bipartite dataset containing one month of edits made by users in the Wikipedia page.
- `fb-forum`[2] is the Facebook-like Forum network that records users' activity in the forum.
- `contacts-dublin`[2] is a human contact network where nodes represent humans and edges between them represent proximity (i.e., contacts in the physical world).

[2] http://networkrepository.com
[3] https://snap.stanford.edu/data/
[4] https://github.com/srijankr/jodie

- `reality-call`[2] is a subgraph of the reality mining study. Nodes are participants and edges are phone calls.

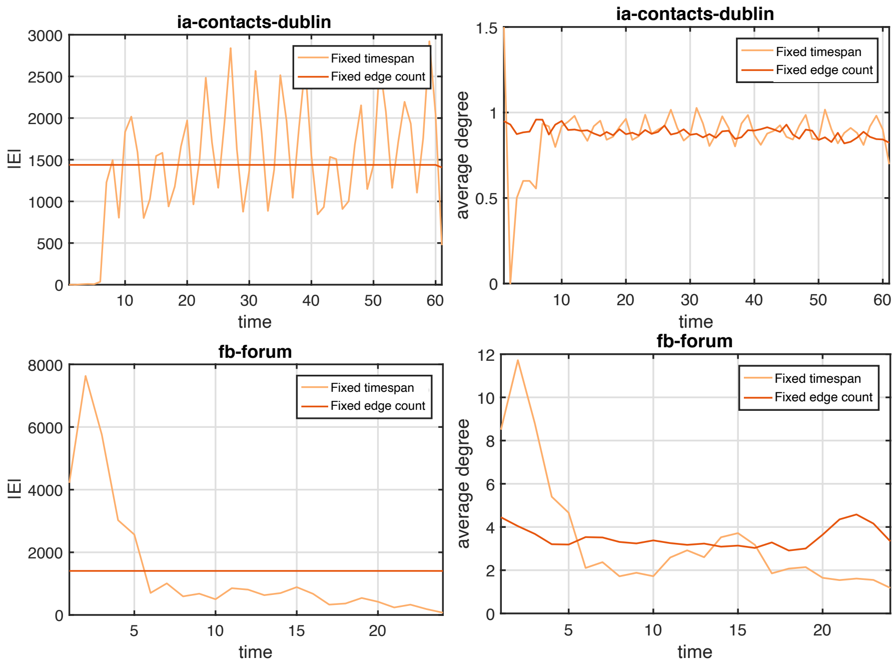


Figure 5.1: Graph properties (#edge and average degree) over two time-series representation (fixed timespans vs. fixed edge count). Fixing the edge number gives more stable temporal patterns while fixing the timespans shows higher fluctuation.

We summarize the graph statistics and temporal timespans in Table 5.2, and analyze the sequential graph statistics of three graphs over time. As the timespans vary from 32 days to 79 months, we adopt the time-scale following Table 5.2 to get the sequential graph time-series. We visualize 2 graph statistics, the number of edges $|V|$ and the average degree on 3 datasets with different time-scales in Figure 5.1, which are `contacts-dublin` (day), `wiki-elec` (month), and `fb-forum` (week). In the figure, we also visualize the same graph statistics using a different time-series representation by fixing the number of edges in each snapshot to $\frac{|E|}{T}$, where T denotes the timespan following the corresponding time-scale. For example, for `wiki-elec`, this number is $\frac{107\,701}{47}$ in each snapshot. From Figure 5.1, we compare the temporal patterns of the two time-series and it can be seen following the fixed edge count in each snapshot gives more stable temporal pattern using both graph statistics. We discuss this new graph time-series in detail in Section 5.5.1. Besides, in this work, we focus on exploring the impacts of graph structures and temporal dependency between snapshots to

Table 5.3: **Summary of notation.**

Symbol	**Definition**
$\mathcal{G} = \{G_k\}$	a graph time-series with snapshots indexed by k.
$G_k = (V_k, E_k)$	a directed and weighted temporal network from $\mathcal{G}$ with $\lvert V_k\rvert$ nodes and $\lvert E_k\rvert$ temporal edges
$\mathbf{A}_k$	adjacency matrix for graph G_k in $\mathcal{G}$.
$G_R = (V, E_R)$	the weighted temporal reachability graph
N_i^R	the set of nodes that are temporally reachable from node i
τ/ϵ	window size representing the timespan / number of edges
α	the decay factor in the temporal summary graph model
θ	the decay factor in the temporal embedding smoothing
f	arbitrary base embedding method
$\mathbf{Z}$	$\lvert V\rvert \times d$ embedding matrix

the predictive tasks, thus we do not leverage node features such as geographic location or content.

5.4 Preliminaries

We summarize symbols and notations specifically used in this chapter in Table 5.3.

5.5 Framework

The framework in this paper provides a fundamental basis for studying different temporal network representations and the utility of these for generalizing existing static embedding methods to temporal network data. The overview is shown in Figure 5.2. Firstly, given the continuous stream of timestamped edges, we derive the time-series of graphs (Section 5.5.1). Then, we use one of the temporal network models to incorporate the temporal dependencies of the graph-based time-series (Section 5.5.2). Lastly, our framework generalizes existing embedding methods and effectively enables the new dynamic variants of these methods to learn more accurate and appropriate time-dependent embeddings. (Section 5.5.3).

5.5.1 Graph Time-Series Representations

We formally introduce two approaches for deriving a time-series of graphs from the stream of timestamped edges. For clarity, we use k to index the snapshots in the time-series in this section to avoid mixing with the timestamp t associated with an edge e.

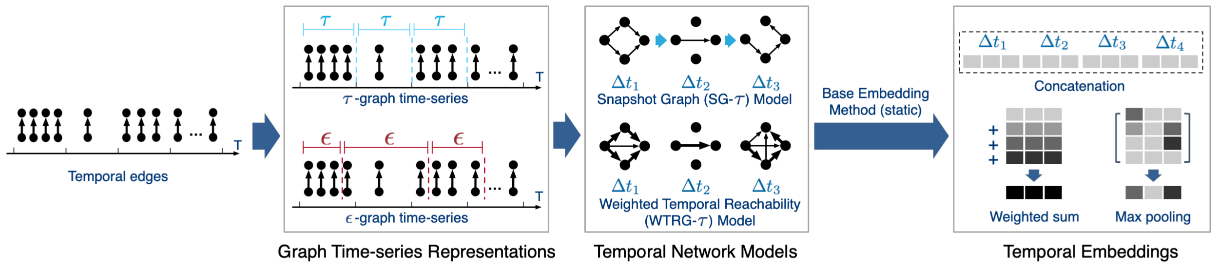

Temporal edges
τ-graph time-series
ϵ-graph time-series
Graph Time-series Representations
Snapshot Graph (SG-τ) Model
Weighted Temporal Reachability (WTRG-τ) Model
Temporal Network Models
Base Embedding Method (static)
Concatenation
Weighted sum
Max pooling
Temporal Embeddings

Figure 5.2: Framework Overview. In the first component of the framework (Sec. 5.5.1), we derive a time-series of graphs from the stream of timestamped edges using either an application-specific time-scale τ (*e.g.*, 1 day) or a fixed number of edges ϵ for each graph in the time-series. Next, given the $\{\tau, \epsilon\}$-graph time-series representation, we incorporate the temporal dependencies and weights with a temporal network model from Sec. 5.5.2. Finally, we use an arbitrary base embedding method to learn a time-series of embeddings and then leverage a temporal fusion mechanism to obtain the final temporal embeddings (Sec. 5.5.3).

5.5.1.1 τ-graph time-series

The τ-graph time-series representation is used by the vast majority of previous work [His16, GCC19].

Definition 14 (τ-graph time-series)**.** *Given a temporal network G=(V, E) representing a continuous edge stream with time-stamped edges E, we define a graph time-series $\mathcal{G}^\tau = \{G_1, \ldots, G_k, \ldots\}$ such that G_1 consists of all edges within the first time scale (period) s, G_2 consists of all edges within the next time period s, and so on. Thus, each graph contains edges within a specific period of time. More formally, let t_0 denote the timestamp of the first edge in the temporal network (stream of timestamped edges) and τ is the application time-scale (*e.g., *1 month), then*

$$E_k = \Big\{(i, j, t) \in E \mid t_0 + k\tau > t \geq t_0 + (k-1)\tau\Big\} \tag{5.1}$$

5.5.1.2 ϵ-graph time-series

While most work uses the previous approach for deriving the graph time-series, we introduce a new alternative based on the idea of using a fixed number of edges. In particular, we propose a new approach that derives a time-series of graphs $\mathcal{G}^\epsilon = \{G_1, \ldots, G_k, \ldots\}$ such that each G_k consists of ϵ edges (Definition 15) and therefore $|E_k| = \epsilon, \forall k$. More formally,

Definition 15 (ϵ-graph time-series)**.** *Given a temporal network $G = (V, E)$ representing a* *continuous edge stream E with timestamped edges and let ϵ denote a fixed number of temporal edges in the stream (ordered by time), we define a graph time-series $\mathcal{G}^{\epsilon} = \{G_1, \ldots, G_k, \ldots\}$ such that $|E_k| = \epsilon$, for all $k = 1, 2, \ldots$. Hence, $G_1 = (E_1, V)$ consists of the first ϵ edges $E_1 = \{e_1, e_2, \ldots, e_{\epsilon}\}$ whereas G_2 consists of the next ϵ edges $E_2 = \{e_{\epsilon+1}, \ldots, e_{2\epsilon}\}$, and so on. More formally, E_k is defined as follows:*

$$E_k = \bigcup_{i=(k-1)\epsilon+1}^{k\epsilon} e_i \;=\; \left\{e_{(k-1)\epsilon+1}, \ldots, e_{k\epsilon}\right\} \tag{5.2}$$

Note in both cases $E_1 \cup \cdots \cup E_k \cup \cdots = E$. Since the proposed ϵ-graph time-series controls for the number of edges over time, embedding methods can more appropriately model and capture the actual change in the structural properties and subgraph patterns over time, as opposed to just the frequency of edges that is captured by the τ-graph time-series representation used in previous work. Another advantage of the ϵ-graph time-series representation is that it preserves the sequential order of timestamped edges *without* suffering from the structural instability of the graph due to the sometimes drastic difference in edge counts from one time to the next. As observed in Fig. 5.1, while the ϵ-graph time-series representation has a fixed number of edges over time, conventionally-used τ representation can significantly deviate with large spikes even between consecutive graphs in the series. If a graph time-series representation is unable to capture the simplest 1st-order subgraph structures (edges), then by definition it cannot capture higher-order subgraph structures that are built on such lower-order ones. Hence, the proposed ϵ-graph time-series representation effectively models the *structural changes* between graphs whereas the τ-graph time-series captures changes in *edge frequencies* for a fixed application-specific time-scale such as 1 day or 1 hour.

5.5.2 Temporal Network Models

Now we introduce temporal network models that incorporate the temporal dependencies into the graph time-series representations to learn more effective time-dependent embeddings.

5.5.2.1 Snapshot Graph (SG) Model

This model simply leverages the $\{\tau, \epsilon\}$-graph time-series representation directly without encoding any additional temporal information into the representation. Hence, the temporal information (edge timestamps) associated with the edges in any graph $G_k \in \mathcal{G}$ is effectively ignored/discarded. For example, e_1 and e_2 are considered to occur simultaneously if they fall into the same snapshot, even though e_2 comes later than e_1 in the actual time-series. Therefore, this model incorporates the temporal dependencies at the level of the graph, *i.e.*, we only know that edges in G_{k-1} occurred before G_k.

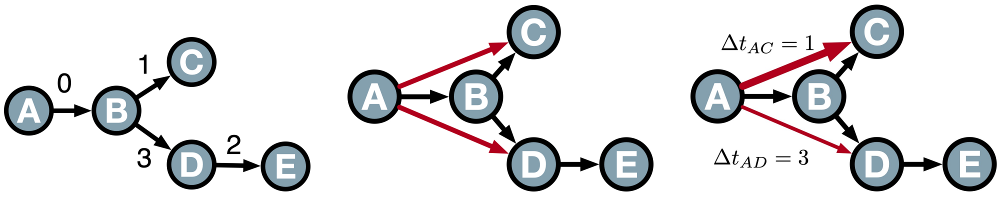
5.5.2.2 Temporal Summary Graph (TSG) Model

The temporal summary graph model incorporates the temporal dependencies by deriving a weighted summary graph from the graph-based time series $\mathcal{G}$ [RN12] where the more recent edges are assigned larger weights than those in the distant past. More formally, let $\mathbf{A}_1, \mathbf{A}_2, ..., \mathbf{A}_k, ..., \mathbf{A}_T$ be a time-series of adjacency matrices of the graph time-series constructed using either Definition 14 or Definition 15. Furthermore, let $\mathbf{A}_k(i,j)$ denote the (i,j) entry of $\mathbf{A}_k$. We define the general *weighted temporal summary graph* (TSG) model as $\mathbf{S} = \sum_{k=1}^{T} f(\mathbf{A}_k, \alpha)$, where f is a decay function for temporally weighting the edges (nonzeros), α is the decay factor ranging in $(0, 1)$, T is the total number of graphs in the time-series, and $\mathbf{S}$ is the weighted temporal summary graph. In this work, we define f as an exponential decay function [RN12], then we obtain

$$\mathbf{S} = \textstyle\sum_{k=1}^{T} (1-\alpha)^{T-k} \mathbf{A}_k \tag{5.3}$$

and the weight for an edge (i,j) is simply $\mathbf{S}(i,j) = \sum_{k=1}^{T} (1-\alpha)^{T-k} \mathbf{A}_k(i,j)$. Alternatively, instead of using all available graphs in the initial time-series, we can use only the L most recent graphs. For example, suppose $\mathcal{G}^\epsilon = \{G_k\}_{k=1}^{T} = \{G_1, \ldots, G_T\}$ is an ϵ-graph time-series with T graphs. Instead of using all T graphs, we can leverage only the most recent L graphs, hence,

$$\mathcal{G}^\epsilon = \{G_k\}_{k=T-L+1}^{T} = \{G_{T-L+1}, \ldots, G_T\} \tag{5.4}$$



(a) A temporal graph (b) TRG (c) Weighed TRG

Figure 5.3: A toy temporal graph (a) and its temporal reachability modeling TRG (b) and WTRG (c). (b) An edge in the vanilla TRG represents a temporally-valid walk. The red edges represents the length-2 walks $\{A, B, C\}$ and $\{A, B, D\}$ in the original graph (c) WTRG extends TRG by assigning weights to indicate the temporal closeness *e.g.*, $\{A, B, C\}$ has higher weights than $\{A, B, D\}$ as C is temporally closer to A than D ($\Delta t_{AC} < \Delta t_{AD}$), which reflects stronger temporal continuity.

The idea of leveraging only the most recent graphs in the time-series was first explored in [RN12] and can be applied to any of the proposed temporal models in this section.

5.5.2.3 Temporal Reachability Graph (TRG) Model

The temporal reachability graph (TRG) is a graph derived from the timestamped edge stream where a link is added between two nodes if they are temporally connected. More formally, an edge (u, v) in the TRG model indicates the existence of a temporal walk from u to v in the original graph. The formal definition is given as follows.

Definition 16 (Temporal Reachability Graph)**.** *Given an interval $\mathcal{I} \in \mathbb{R}^+$, the temporal reachability graph $G_R = (V, E_R)$ is defined as a directed graph where the edge $(u, v) \in E_R$ denotes the existence of a temporal walk leaving u and arriving v within that interval. We denote the number of edges in $\mathcal{I}$ as ω (which could be defined based on $\{\tau, \epsilon\}$-graph time-series).*

A TRG is a static unweighted graph where each edge indicates a temporally-valid walk reaching from the source to the destination. However, it does not capture the strength of reachability. For example in Fig. 5.3a, the walk $\{A, B, C\}$ takes two timestamps while $\{A, B, D\}$ takes four. Intuitively D is harder to reach than C from node A due to less temporal continuity. Vanilla TRG fails to capture such property since all the edges are equally important (shown in 5.3b). This would potentially affect the proximity-based embedding methods as they are based on the closeness of nodes in the graph. To overcome this drawback, we propose an extension of TRG called Weighted TRG (WTRG) that encapsulates the strength of reachability in the graph weights. We define the strength of reachability between

Algorithm V.1 Weighted Temporal Reachability Graph

```
procedure TemporalReach($G = (V, E)$)
    Set $E_R = \emptyset$, sort $E_T$ in reverse time order
    while next edge $(i, j, t) \in E$ do
        for $(k, t_k) \in N_j^R$ do
            $E_R \leftarrow E_R \cup \{(i, k)\}$
            $g_{i,k} = g_{i,k} + e^{-(t_k - t)}$
            $N_i^R \leftarrow N_i^R \cup \{(k, t_k)\}$
        $E_R \leftarrow E_R \cup \{(i, j)\}$
        $g_{i,j} = g_{i,j} + 1$                    ▷ $\Delta t_{i,j} = 0$ as $i, j$ are adjacent
        $N_i^R \leftarrow N_i^R \cup \{(j, t)\}$
    return $G_R = (V, E_R, g)$
```

a pair of nodes (i, j) as a function of both the number of temporally-valid paths and the timestamp difference. The weighting function is given as follows.

$$g_{i,j} = \sum_{w \in \mathcal{W}} e^{-(\Delta t_{i,j} | w)} \tag{5.5}$$

where w is a specific temporally-valid walk from i to j, and $\Delta t_{i,j}$ denotes the temporal delay reaching from i to j along that walk. We depict the process of deriving WTRG in Algorithm V.1. The cornerstone of the algorithm is the temporally-reachable neighborhood N_i^R that records nodes that are reached by i and the latest timestamps associated with temporal paths. We formally define N_i^R as:

Definition 17 (Temporally reachable neighborhood)**.** *Given a node i, its temporally reachable neighborhood N_i^R is defined as the set of tuples $\{(j, t_j)\}$ where j is the node reachable from i following a temporally-valid walk and t_j is the timestamp of the edge reaching j in that walk.*

Given an input temporal edge (i, j, t), Algorithm V.1 loops through reachable neighbors in N_i^R to add edges in E_R and updates the weights based on Eq. (5.5) (line 5-8). It also adds (i, j) to the WTRG as well as the immediate weight (line 9-11). Overall, the computational complexity of the algorithm is $\mathcal{O}(|E| \max d(N^R))$, where $\max d(N^R)$ is the maximum degree of a node in WTRG. While the derived WTRG can be dense with huge amounts of reachable neighbors, we show that this number is bounded by ω, which is the size of the interval associated with the WTRG (Section 5.5.2.4 of the supplementary material). Accordingly, the computational complexity of the algorithm is denoted as $\mathcal{O}(|E|\omega)$. We follow Algorithm V.2

to combine the embeddings over the graph time-series.

5.5.2.4 Computational Complexity of WTRG

In this section we detail the computational complexity in deriving WTRG by showing the following property.

Property 1. *The number of edges in G_R is bounded by the number of temporally-valid walks in G.*

Based on Def. 16, an edge $(u, v) \in E_R$ indicates a temporally-valid walk reaching from u to v in G. However, this edge could correspond to multiple unique temporal walks with different intermediate nodes and associated timestamps, therefore, $|E_R|$ is no more than the number of temporally-valid walks in G.

Let N_i^R denote the temporally reachable nodes of i, $\Delta(G_R) = \max\{d(N_1^R), \dots, d(N_n^R)\}$ is the maximum degree of a node in G_R, and ω is the window size. Then

$$|N_i^R| \leq \Delta(G_R) \leq \omega \tag{5.6}$$

According to Def. 16, a TRG is comprised by edges within the interval with size ω. These edges comprise upto ω different temporal walks originating from a specific node i. Therefore, based on Property 1, the number of edges originating from node i is bounded by the number of temporally-valid walks, which is ω.

5.5.3 Temporal Embeddings

5.5.3.1 Base embedding methods

Given the graph time-series representation and temporal model (Section 5.5.1-5.5.2), the proposed approach can leverage any existing static embedding method to derive time-dependent node embeddings that capture the important temporal dependencies between the nodes as well as the temporal structural (role-based) and proximity-based properties [RJK$^+$19]. We use the proposed framework to generalize a wide variety of static base

Algorithm V.2 General Framework for Temporal Embeddings

```
Ensure: $\epsilon$ or $\tau$ for deriving the graph time-series representation, base embedding method $f$ (e.g., GraphWave, role2vec)
1 Construct a graph time-series $\mathcal{G} = \{G_1, G_2, \ldots, G_T\}$ using a graph time-series representation $\{\tau, \epsilon\}$ from Section 5.5.1.
2 Initialize $\mathbf{Z}_0$ to all zeros
3 for each $G_k \in \mathcal{G}$ do                                   ▷ for $k = 1, 2, \ldots$
4     Use Alg. V.1 to derive the temporal reachability graph for $G_k$
5     Compute node embedding matrix $\mathbf{Z}_k$ using the base embedding method $f$ with the temporal reachability graph from Alg. V.1
6     Concatenate or aggregate (using sum, mean, etc.) the embedding matrix, e.g., $\bar{\mathbf{Z}}_k = (1-\theta)\bar{\mathbf{Z}}_{k-1} + \theta\mathbf{Z}_k$ where $\bar{\mathbf{Z}}_k$ is the temporally weighted embedding using the above exponential weighting kernel $\mathbb{K}(\cdot)$ and $0 \leq \theta \leq 1$ is a hyperparameter controlling the importance of past information relative to more recent (Section 5.5.3.2).
7 return $\bar{\mathbf{Z}}_k$ (temporally weighted embeddings using $\mathbb{K}$ and $\theta$) or $\mathbf{Z} = \begin{bmatrix} \mathbf{Z}_1 & \mathbf{Z}_2 & \cdots & \mathbf{Z}_T \end{bmatrix}$ (concatenated embeddings)
```

embedding methods including both community-based and role-based structural node embedding methods [RJK+19]. Namely, they are: **(1)** LINE [TQW+15b], **(2)** Node2vec [GL16], **(3)** Graph2Gaussian [BG17], **(4)** struc2vec [RSF17], **(5)** Role2vec [ARZ+18], **(6)** Graphwave [DZHL18], and **(7)** multilens [JRK+19]. We provide the detailed configuration of each individual method in Section 5.6.1.3 for reproducibility of the experiments. Among these static methods, approaches (1-3) are community/proximity-based and (4-6) are role-based. Method (7) is a hybrid that is based on structural similarity of node-central subgraphs.

5.5.3.2 Temporal fusion

Given the time-series of node embeddings $\{\mathbf{Z}_k\}_{k=1}^{T}$, we explore two temporal fusion techniques.

Concatenation: Given a time-series of embeddings, one simple approach to obtain a final embedding is to concatenate the embeddings as follows: $\mathbf{Z} = \begin{bmatrix} \mathbf{Z}_1 \cdots \mathbf{Z}_T \end{bmatrix}$. We could further weight the embeddings based on temporal recency, *i.e.*, under-weighting node embeddings that occur in the distant past since they are not as important as the more recent ones for prediction.

Temporally weighting: This technique aggregates (*e.g.*, sum, mean) the embedding matrix, *e.g.*, $\bar{\mathbf{Z}}_k = (1-\theta)\bar{\mathbf{Z}}_{k-1} + \theta\mathbf{Z}_k$ where $\bar{\mathbf{Z}}_k$ is the temporally weighted embedding using the above exponential weighting kernel $\mathbb{K}(\cdot)$. $0 \leq \theta \leq 1$ is a hyperparameter controlling the importance of past information relative to more recent.

5.6 Experiments

In this section, we systematically investigate the effectiveness of each component in the framework, *i.e.*, the different graph time-series representations (Section 5.6.3, temporal network models (Section 5.6.4), and the new dynamic node embedding methods generalized using the proposed framework (Section 5.6.5). More specifically, we aim to explore the following research questions:

- **Q1** How well does the widely-used τ-graph time-series representation perform comparing with the proposed ϵ-graph time-series?
- **Q2** How effective is the proposed WTRG model comparing with the vanilla TRG model? What temporal models are most useful for incorporating temporal dependencies into static embedding methods?
- **Q3** Are the dynamic embedding methods generalized via the framework useful for temporal prediction? How do they compare to the state-of-the-art dynamic methods?

5.6.1 Experimental Setup

5.6.1.1 Data

We learn node embeddings from the graph time-series starting from roughly $\frac{1}{3}$ of the timespans. For example, for the `bitcoin` dataset, we train the classifier based on node embeddings derived from month 20 to month 25 out of 63 months, inclusive. This ensures that there are sufficient edges for training. For all datasets, we perform training on the first 6 graphs and predict links on the 7th graph. Depending on the time-scale shown in Table 5.2, they represent 6 months (`enron, bitcoin, wiki-elec and overflow`), weeks (`fb-forum` and `reality-call`), or days (`wiki-edit` and `contact-dublin`). We create evaluation examples from the links in the 7th graph and an equal number of randomly sampled pairs of unconnected nodes as negative samples [SWG^{+}20].

5.6.1.2 Model configuration and variants

We consider the task of link prediction over time and systematically compare the performance of different temporal network models and representations. Given a set of timestamped edges up to timestamp T, *i.e.*, $\mathcal{G} = \{G_1, \cdots G_T\}$, the temporal link prediction task aims to predict the future links that will form in G_{T+1}. We first follow the conventional setup to construct the τ-graph time-series $\mathcal{G}^\tau = \{G_1, \cdots G_T\}$ for model training and G_{T+1} for testing, where each snapshot $G_k(k \in \{1, 2, \cdots, T\})$ represents edges that occur within a consistent time scale shown in Table 5.2. Then we construct the ϵ-graph time-series representation $\mathcal{G}^\epsilon$. For fair comparison, we set $\epsilon = |E_{T+1}|$ to ensure the trained models based on both ϵ- and τ-based temporal networks are used to predict links in the same hold-out test set G_{T+1}. Thus, graphs in the ϵ-graph time-series $\mathcal{G}^\epsilon = \{G_1, \dots, G_T\}$ and G_{T+1} are also consistent with respect to the ϵ representation, where $|E_1| = |E_2| = \cdots = |E_{T+1}|$.

For each $\{\epsilon, \tau\}$-graph time-series representation, we select a temporal network model from $\{SG, TSG, WTRG\}$ and a base embedding method using the framework. Therefore, we have totally 6 dynamic variants: $\{SG\text{-}\epsilon$, $TSG\text{-}\epsilon$, $WTRG\text{-}\epsilon$, $SG\text{-}\tau$, $TSG\text{-}\tau$, $WTRG\text{-}\tau\}$. To train the classifier, we applying these dynamic variants to derive node embeddings and feed them to the logistic regression model for prediction with regularization strength 1.0 and stopping criteria 10^{-4}. Following [CLX15b], we concatenate the node embeddings $\mathbf{z}_i$ and $\mathbf{z}_j$ to obtain an edge embedding $\mathbf{z}_{ij} = \begin{bmatrix} \mathbf{z}_i \ \mathbf{z}_j \end{bmatrix}$. For temporal fusion, we use the temporally weighting technique from Section 5.5.3.2 with $\theta = 0.8$ for dimensional consistency. The TSG decay parameter α is set to 0.8 for computational fairness. For all experiments, we perform 3 runs and report the average.

5.6.1.3 Base and dynamic embedding method configuration

We configured all the base methods to achieve the best performance according to the respective papers. For all the static methods based on random walks (*i.e.*, node2vec, struc2vec), we perform 20 walks with the maximum walk length $L = 20$. For node2vec, we perform grid search over $p, q \in \{0.25, 0.50, 1, 2, 4\}$ as mentioned in [GL16] and report the best performance. For LINE and Multi-Lens, we incorporate 2nd-order proximity in the graph. For role2vec, we leverage the node degree as the feature for roles. For Graphwave, we perform the method to automatically select the scaling parameter with exact heat kernel matrix calculation. We set the final embedding with dimension $K = 128$ for evaluation, and leverages the weighted summation fusion approach so that the embedding dimensions of individual graphs are fixed to be the same.

For the state-of-the-art dynamic embedding methods, we follow the configuration given by the paper/code repository. Specifically, for CTDNE, we set #walks= 10, the walking length $L = 20$. For node2bits, we perform short-term temporal random-walk with scope to be 3. The the #walks and the walking length are set to be the same as CTDNE. For DANE, we leverage both the offline computation model to derive node embeddings based on the first 6 graphs, and the online model to derive node embeddings for the 6th graph based on the first 5. We set the intermediate embedding dimensions to be 100 for both models and report the best performance. For TIMERS, we set the tolerance threshold value that is used to restart the optimal SVD calculation to be 0.17 as provided in the code repository. For DyAE/DyAERNN, we leverage the 2-layer auto-encoder/decoder with 400 and 200 units, respectively. We set the regularization hyperparameter to be 10^{-6}, bounding ratio for number of units in consecutive layers to be 0.3 as suggested in the paper, and perform grid search in the range of $\pm 10\%$ of the default value. In the learning stage, the sgd learning rate is set to be 10^{-6} with minibatch size to be 100. Lastly, for DySAT, we leverage the base model and perform grid search on the default hyperparameters in the range of $\pm 10\%$ of the default values.

5.6.2 WTRG vs. TRG

We first study the effectiveness of WTRG model over the vanilla TRG model. As WTRG incorporates the strength of reachability in edge weights, we consider embedding methods that handles weighted graphs, namely, they are node2vec, struc2vec and multilens. We run both methods on two datasets using both TRG and WTRG with τ-graph time series as shown in Table 5.4.

The first observation from Table 5.4 is that structure-based embedding methods tend to outperform node2vec, the proximity-based method. In addition, we observe that WTRG

Table 5.4: **Performance of WTRG over TRG on τ-graph time series.**

		`bitcoin`		`wiki-elec`	
Method	Metric	TRG	WTRG	TRG	WTRG
node2vec	AUC	0.9214	**0.9239**	**0.7348**	0.7344
	ACC	0.8294	**0.8412**	**0.6171**	0.6144
	F1	0.8285	**0.8408**	**0.5909**	0.5889
struc2vec	AUC	0.9274	**0.9301**	0.7840	**0.7933**
	ACC	0.7959	**0.8109**	0.6583	**0.6703**
	F1	0.7925	**0.8081**	0.6388	**0.6534**
multilens	AUC	0.9226	**0.9389**	0.8106	**0.8143**
	ACC	0.8656	**0.8793**	0.7438	**0.7539**
	F1	0.8655	**0.8792**	0.7385	**0.7493**

improves most embedding methods in link prediction, except for node2vec on `wiki-elec` dataset. One possible reason is that the random walker in WTRG are more likely to visit nodes that are close in time, and thus limiting the derived embeddings to incorporate distant neighborhood information. We put this deep study of WTRG in the future work. Nevertheless, for embedding methods that are based on structural information, WTRG outperforms TRG by 0.8% in AUC ,1.3% in ACC, and 1.4% in F1 score on average. As we observe that the WTRG model tends to outperform the vanilla TRG model, we use WTRG in the rest of the experiments.

5.6.3 Fixed #edges (ϵ) vs. time-scale (τ)

In this section, we investigate the effectiveness of different graph time-series representations (**Q1**). Due to the massive amount of experimental results, we first define 2 evaluation metrics for this experiment. These newly proposed measurement are for readers to have a clear overview of the comparison results across all components in the proposed framework across all the datasets.

We first evaluate the general performance of each temporal model through the mean ranking (and std) across all datasets and embedding methods in terms of the AUC, ACC and F1 score. We leverage the following metrics to better interpret the results. Let $\mathbf{y}_{jk} \in \mathbb{R}^{|\mathcal{M}|}$ denote the vector of AUC (or ACC, F1) scores of the temporal models $\mathcal{M}$ for an embedding method $f_j \in \mathcal{F}$ and graph dataset k. Further, let $\pi(\mathbf{y}_{jk}, M_i)$ denote the rank of the temporal model $M_i \in \mathcal{M}$ for a given embedding method f_j and graph dataset $d_k \in \mathcal{D}$. The mean rank is computed as

$$\mathrm{MR}_i = \frac{1}{|\mathcal{D}||\mathcal{F}|} \sum_{d_k \in \mathcal{D}} \sum_{f_j \in \mathcal{F}} \pi(\mathbf{y}_{jk}, M_i) \tag{5.7}$$

Therefore, smaller values of MR indicate better model performance. We report the results in Table 5.5. In addition to the general performance, we also provide an intuitive ranking based on the number of times each model performs the best following [RAEZ18]. This metric s_i reflects the occurrence of temporal model M_i to be optimal:

$$s_i = \sum_{d_k \in \mathcal{D}} \sum_{f_j \in \mathcal{F}} \mathbb{I}\big\{\pi(\mathbf{y}_{jk}, M_i) = 1\big\} \tag{5.8}$$

where $\mathbb{I}\{\pi(\mathbf{y}_{jk}, M_i) = 1\}$ returns 1 if $\pi(\mathbf{y}_{jk}, M_i) = 1$ and 0 otherwise. $\mathbb{I}\{\pi(\mathbf{y}_{jk}, M_i) = 1\}$ indicates that the temporal model M_i performs best for the given graph dataset d_k and base embedding method f_j. Thus, s_i denotes the total score of model M_i based on the number of times temporal model M_i appeared first in the ranking across all base embedding methods and graph datasets.

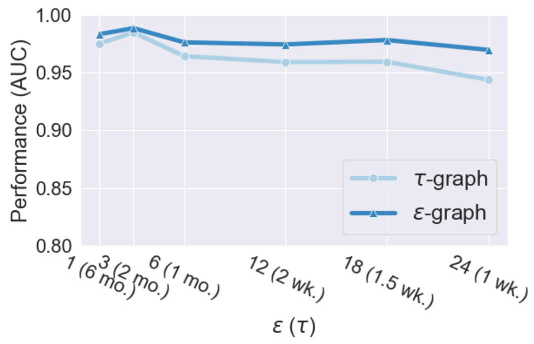
Performance. Based on the results shown in Table 5.5, our first observation is that the top-3 temporal models are those that use the proposed ϵ-graph time-series representation. These models perform comparably well in terms of AUC, ACC and F1 and are in general better than τ-graph time-series representation used in previous work. This finding indicates the general effectiveness of ϵ-graph time-series in representing the temporal network. We also compute an overall score by summing over each s_i for all evaluation criterion (bottom row in Table 5.6). We observe that the top models are ϵ-based, which demonstrate the effectiveness of ϵ-graph time-series in capturing the graph *structural changes* over edge *frequency changes* in prediction tasks.

Sensitivity Analysis. We also conduct a parameter sensitivity analysis on two datasets, `bitcoin` and `fb-forum`, to evaluate the impact of different values of τ and ϵ on the overall performance. For `bitcoin`, we train our model on temporal data spanning 6 months and test on the 7th month. For`fb-forum`, we train on 12 weeks and test on the 13rd week. We create the τ-graph time series using different scales (e.g., months, weeks, days), and generate the same number of ϵ-graphs with equal number of edges. Based on the result shown

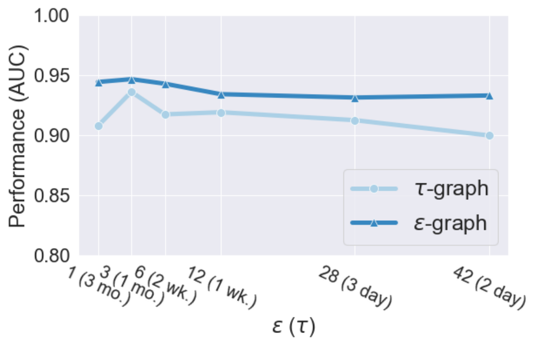
Table 5.5: Mean rank (and std.) of the temporal network models across all base embedding methods and graphs based on AUC, ACC and F1, lower is better. The top-3 temporal network models are based on the new ϵ-graph time-series representation (fixed #edges).

Temporal Model	Mean Rank (MR) AUC	ACC	F1
WTRG-ϵ	$\mathbf{2.30 \pm 2.16}$	2.73 ± 1.95	2.66 ± 1.90
TSG-ϵ	2.43 ± 1.84	$\mathbf{2.61 \pm 2.19}$	2.70 ± 2.24
SG-ϵ	2.57 ± 1.64	2.66 ± 1.86	$\mathbf{2.59 \pm 1.88}$
WTRG-τ	2.80 ± 1.96	2.80 ± 1.98	2.86 ± 1.99
SG-τ	2.95 ± 1.91	2.84 ± 1.87	2.82 ± 1.83
TSG-τ	3.63 ± 1.75	3.46 ± 1.87	3.45 ± 1.80
SG	4.32 ± 1.88	3.89 ± 1.91	3.93 ± 1.94

in Figure 5.4, we observe that our model running on the ϵ-graph time series consistently outperforms the τ-graph time series, while being more robust. This also indicates that in practice, the ϵ-graph time series can be used as an alternative to create temporal graph snapshots for various mining tasks, especially in preliminary graph analysis when the optimal timescale is undetermined.



(a) Prediction performance on `bitcoin`



(b) Prediction performance on `fb-forum`

Figure 5.4: Sensitivity analysis. Link prediction performance on τ- and ϵ-graph. The τ-graphs are created based on different timescales, the ϵ-graphs are created via equal division. mo.: month. wk.: week.

Conclusion 5. Overall, the proposed ϵ-graph time-series representation based on a fixed number of edges outperforms the time-scale τ-graph time-series across different scales, while being more robust.

5.6.4 Temporal Model Comparison

To answer **Q2**, we follow the formulation in Section 5.6.3 to quantitatively evaluate and rank the temporal models according to their effectiveness in prediction. We show the complete performance of temporal network models that perform the best following Equation (5.8) with respect to individual datasets in Table 5.6 to supplement the mean ranking in Table 5.5.

Notably, the TSG-ϵ model has the highest # of first ranks across all datasets, especially on datasets with short timespans (*i.e.*, `wiki-edit` and `contacts`). It also has the highest # of first ranking, in terms of ACC and F1, which indicates that this model is generally promising but at the same time less stable than the other ϵ-based models. We also confirm this finding in Table 5.5 as it shows relatively higher variance of ranking. WTRG-ϵ performs the second best and is a close second to TSG-ϵ on datasets with long timespans. This is potentially due to how they model the temporal recency: TSG models the past information with exponential decay (5.3), while WTRG models it with the absolute temporal difference (5.5). Thus, TSG could still capture the temporal evolution within a relatively short period of time. Nevertheless, both TSG and WTRG perform well on all datasets even though spikes and flucation are observed such as `fb-forum` (Figure 5.1). It can be seen that there is not a single temporal model that prevails across all datasets. On the other hand, the WTRG model tends to performs well regardless of the timescales in graph representation, while TSG model tends to perform well on graphs with short timespans. Besides, the temporal models that are combined with the proposed ϵ-graph time-series representation tend to outperform their other τ-counterparts, which is consistent with our previous findings from Section 5.6.3.

Table 5.6: Temporal model performance across the temporal graphs. Each (i, j) is the # of times temporal model $M_j \in \mathcal{M}$ in graph G_i performed best comparing to the other models across all base embedding methods $f \in \mathcal{F}$ and evaluation criterion. We bold the temporal model that performs best overall for each graph.

	TSG-ϵ	**WTRG-ϵ**	**SG-ϵ**	**SG-τ**	**WTRG-τ**	**SG**	**TSG-τ**
`bitcoin`	**6**	**6**	4	5	0	0	0
`stackoverflow`	1	4	3	3	**9**	0	1
`enron`	4	1	1	3	**8**	4	0
`wiki-elec`	2	6	**7**	6	0	0	0
`fb-forum`	**10**	**10**	0	1	0	0	0
`wiki-edit`	**7**	3	2	3	2	2	2
`reality-call`	1	0	2	4	**6**	4	4
`contacts-dublin`	**9**	2	8	1	1	0	0
overall score	40	32	27	26	26	10	7

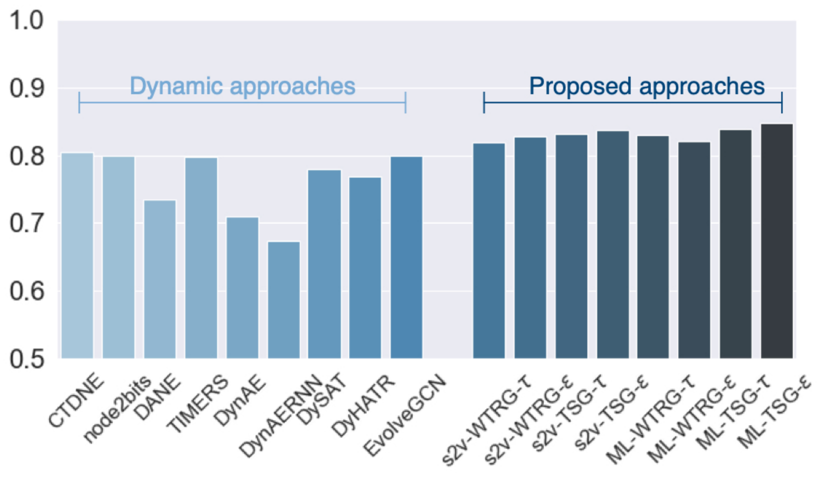


Figure 5.5: Predictive results of the dynamic embedding methods and our framework. Our proposed framework approximates well to approaches specifically designed for temporal graphs with comparable or even better performance (ML = multilens, s2v = struc2vec).

Conclusion 6. Out of all models, WTRG-ϵ and TSG-ϵ tend to perform the best. Empirically, WTRG-ϵ is more stable overall (Table 5.5) and TSG-ϵ performs well on datasets with short timespans (Table 5.6).

5.6.5 Dynamic Embeddings: Variants vs. State-of-the-art

To answer **Q3**, we first use the framework to derive new dynamic embedding methods (by selecting the representation, temporal model, base embedding method, etc.), then we compare their performance to the state-of-the-art dynamic embedding methods on all 8 datasets. One would presumably expect that the state-of-the-art methods for dynamic node embeddings will outperform the dynamic embedding methods generalized by our framework. This is because the state-of-the-art methods are typically more complex and have been designed specifically for learning such dynamic node embeddings. We use 9 recent state-of-the-art dynamic methods during 2017 $\sim$ 2020 as baselines, including CTDNE [NLR+18], node2bits [JHRK19], DANE [LDH+17], DynGem [GKHL18] TIMERS [ZCP+18], DynAE/DynAERNN [GCC19], DySAT [SWG+20], DyHATR [XYJ+20], and EvolveGCN[PDC+20].

Figure 5.5 shows the mean AUC for each method where the average is taken over all graphs investigated. As representative dynamic embedding methods from the proposed framework, we use 4 dynamic embedding variants of struc2vec (s2v-TSG-ϵ/τ, s2v-WTRG-ϵ/τ) and 4 variants of MultiLENS (ML-TSG-ϵ/τ, ML-WTRG-ϵ/τ). Strikingly, we observe that the dynamic embedding methods from the framework perform comparably or even better than the state-of-the-art methods that are designed particularly for temporal graphs and time-series prediction. In particular, ML-TSG-ϵ performs best with a mean gain of 12.34% followed by s2v-TSG-ϵ with a gain in AUC of 10.97%. Also, we note that our proposed framework is computationally efficient. Taking multilens as an base embedding method, our proposed framework has the computational complexity $\mathcal{O}(|E||\mathcal{G}|)$ where $|\mathcal{G}|$ is the number of graphs in the time-series, and the number of parameters to learn is $\mathcal{O}(|V||\mathcal{G}|)$. This is also validated empirically throughout our experiment.

Conclusion 7. The dynamic embeddings derived from our framework leveraging conventional static embedding methods (Section 5.5) perform better than state-of-the-art dynamic embedding methods.

Notably, in this experiment we do not aim to show substantial improvement of our framework over all the dynamic approaches, especially those based on deep learning, since node features are not used. Instead, these results show that our proposed framework could capture the graph structures and temporal dependency at least as good as those recent dynamic approaches with less complexity (*i.e.*, no transitional or latent variables), Unlike methods that are based on complex models as "black boxes", the components of our proposed framework are easy-to-understand, which further motivates its usage for practitioners in predictive applications.

5.7 Complete Experimental Results

In this section, we show the complete experimental results using the both the dynamic approaches generated from our proposed framework, and the state-of-the-art approaches specifically designed to handle dynamic graphs. In total, we perform 3 runs of the experiments on 8 dynamic graphs and report the average AUC, ACC and F1 metrics with the standard deviation. The dynamic methods generated from our proposed framework include 7 base static embedding methods coupled with 6 temporal models as well as the static form, *i.e.*, not using the temporal information. We also include 7 state-of-the-art dynamic embedding

Table 5.7: Complete experimental results on the first 2 datasets. The values are represented using percentage %.

Base Method	Temporal Model	bitcoin			stackoverflow		
		AUC	ACC	F1	AUC	ACC	F1
Node2vec	**Static**	89.05 ± 0.44	76.74 ± 0.70	75.89 ± 0.74	95.97 ± 0.02	84.34 ± 0.16	84.09 ± 0.17
	SG-τ	93.80 ± 0.62	86.63 ± 0.20	86.58 ± 0.20	96.11 ± 0.05	88.00 ± 0.14	87.94 ± 0.15
	WTRG-τ	95.15 ± 0.45	88.71 ± 0.61	88.65 ± 0.63	96.45 ± 0.05	89.67 ± 0.34	89.65 ± 0.35
	SG-ϵ	95.08 ± 0.63	88.76 ± 0.67	88.74 ± 0.67	96.11 ± 0.05	88.13 ± 0.29	88.08 ± 0.29
	WTRG-ϵ	95.89 ± 0.19	89.82 ± 0.47	89.79 ± 0.48	96.51 ± 0.02	89.41 ± 0.09	89.37 ± 0.09
	TSG-τ	89.37 ± 0.40	76.79 ± 1.02	75.97 ± 1.12	96.08 ± 0.04	84.65 ± 0.25	84.42 ± 0.26
	TSG-ϵ	90.46 ± 0.05	77.15 ± 0.09	76.37 ± 0.09	96.18 ± 0.02	84.69 ± 0.17	84.45 ± 0.19
LINE	**Static**	87.58 ± 0.00	75.86 ± 0.00	75.11 ± 0.00	97.08 ± 0.00	91.41 ± 0.00	91.39 ± 0.00
	SG-τ	89.75 ± 0.00	82.48 ± 0.00	82.46 ± 0.00	96.79 ± 0.00	91.87 ± 0.00	91.87 ± 0.00
	WTRG-τ	89.49 ± 0.00	82.24 ± 0.00	82.24 ± 0.00	96.95 ± 0.00	91.92 ± 0.00	91.92 ± 0.00
	SG-ϵ	89.18 ± 0.00	83.49 ± 0.00	83.48 ± 0.00	96.96 ± 0.00	91.85 ± 0.00	91.85 ± 0.00
	WTRG-ϵ	91.68 ± 0.00	84.81 ± 0.00	84.81 ± 0.00	96.79 ± 0.00	91.48 ± 0.00	91.48 ± 0.00
	TSG-τ	87.59 ± 0.00	76.48 ± 0.00	75.86 ± 0.00	97.25 ± 0.00	91.44 ± 0.00	91.42 ± 0.00
	TSG-ϵ	89.28 ± 0.00	76.71 ± 0.00	75.99 ± 0.00	97.23 ± 0.00	91.26 ± 0.00	91.23 ± 0.00
Struc2vec	**Static**	91.56 ± 0.16	81.70 ± 0.28	81.40 ± 0.33	96.99 ± 0.02	84.82 ± 0.17	84.57 ± 0.17
	SG-τ	95.19 ± 0.59	87.83 ± 0.68	87.78 ± 0.68	96.92 ± 0.05	88.38 ± 0.29	88.30 ± 0.30
	WTRG-τ	92.10 ± 0.97	84.22 ± 1.36	84.19 ± 1.35	96.66 ± 0.10	88.12 ± 1.00	88.04 ± 1.03
	SG-ϵ	96.37 ± 0.68	89.85 ± 0.25	89.83 ± 0.25	97.01 ± 0.04	88.39 ± 0.11	88.30 ± 0.11
	WTRG-ϵ	92.42 ± 0.33	83.78 ± 0.65	83.74 ± 0.65	96.67 ± 0.14	87.50 ± 0.59	87.40 ± 0.60
	TSG-τ	91.61 ± 0.16	81.93 ± 0.13	81.65 ± 0.15	96.95 ± 0.03	84.92 ± 0.11	84.67 ± 0.11
	TSG-ϵ	92.17 ± 0.07	81.85 ± 0.75	81.46 ± 0.79	97.16 ± 0.03	84.67 ± 0.15	84.38 ± 0.16
Role2vec	**Static**	85.02 ± 0.04	76.27 ± 0.24	75.49 ± 0.24	92.10 ± 0.15	84.00 ± 0.05	83.77 ± 0.05
	SG-τ	94.90 ± 0.84	87.77 ± 0.63	87.73 ± 0.63	95.45 ± 0.11	85.26 ± 0.14	85.15 ± 0.16
	WTRG-τ	93.07 ± 1.11	85.64 ± 1.12	85.63 ± 1.12	95.51 ± 0.05	88.27 ± 0.71	88.24 ± 0.72
	SG-ϵ	93.33 ± 0.66	85.70 ± 0.96	85.68 ± 0.97	95.34 ± 0.20	83.94 ± 0.06	83.67 ± 0.07
	WTRG-ϵ	93.59 ± 0.33	86.81 ± 1.13	86.78 ± 1.12	95.90 ± 0.08	86.89 ± 0.85	86.79 ± 0.87
	TSG-τ	85.50 ± 0.26	75.96 ± 0.43	75.16 ± 0.44	92.36 ± 0.05	84.00 ± 0.03	83.76 ± 0.03
	TSG-ϵ	86.03 ± 0.20	75.78 ± 0.88	75.01 ± 0.91	93.13 ± 0.10	83.80 ± 0.06	83.52 ± 0.07
Graphwave	**Static**	91.73 ± 0.00	77.96 ± 0.00	77.25 ± 0.00	96.89 ± 0.00	84.49 ± 0.00	84.26 ± 0.00
	SG-τ	99.13 ± 0.00	91.90 ± 0.00	91.85 ± 0.00	96.84 ± 0.00	84.49 ± 0.00	84.26 ± 0.00
	WTRG-τ	99.13 ± 0.00	91.36 ± 0.00	91.29 ± 0.00	96.89 ± 0.00	84.51 ± 0.00	84.28 ± 0.00
	SG-ϵ	99.48 ± 0.00	91.12 ± 0.00	91.05 ± 0.00	97.07 ± 0.00	84.25 ± 0.00	83.97 ± 0.00
	WTRG-ϵ	99.42 ± 0.00	89.88 ± 0.00	89.77 ± 0.00	97.07 ± 0.00	84.25 ± 0.00	83.97 ± 0.00
	TSG-τ	91.98 ± 0.00	77.26 ± 0.00	76.46 ± 0.00	96.96 ± 0.00	84.49 ± 0.00	84.26 ± 0.00
	TSG-ϵ	91.96 ± 0.00	76.56 ± 0.00	75.76 ± 0.00	96.99 ± 0.00	84.25 ± 0.00	83.97 ± 0.00
G2G	**Static**	85.16 ± 0.87	76.37 ± 0.91	75.97 ± 1.02	93.15 ± 0.10	83.80 ± 0.11	83.66 ± 0.12
	SG-τ	72.64 ± 1.56	67.63 ± 1.51	67.61 ± 1.52	92.94 ± 0.18	86.95 ± 0.28	86.94 ± 0.28
	WTRG-τ	80.32 ± 0.92	72.04 ± 1.15	72.04 ± 1.15	95.64 ± 0.30	90.30 ± 0.17	90.29 ± 0.17
	SG-ϵ	71.66 ± 3.08	67.00 ± 2.46	66.96 ± 2.47	92.32 ± 0.51	86.58 ± 0.41	86.58 ± 0.41
	WTRG-ϵ	77.97 ± 1.37	71.08 ± 2.18	71.07 ± 2.18	95.37 ± 0.24	90.67 ± 0.38	90.65 ± 0.39
	TSG-τ	85.59 ± 0.91	76.20 ± 0.66	75.75 ± 0.66	93.27 ± 0.32	83.91 ± 0.54	83.77 ± 0.57
	TSG-ϵ	87.20 ± 0.87	77.98 ± 0.72	77.63 ± 0.76	93.83 ± 0.10	84.03 ± 0.19	83.86 ± 0.19
Multilens	**Static**	91.28 ± 0.00	81.85 ± 0.00	81.63 ± 0.00	97.12 ± 0.00	90.24 ± 0.00	90.19 ± 0.00
	SG-τ	82.43 ± 0.00	75.47 ± 0.00	75.06 ± 0.00	96.95 ± 0.00	92.98 ± 0.00	92.98 ± 0.00
	WTRG-τ	84.48 ± 0.00	77.80 ± 0.00	77.55 ± 0.00	96.97 ± 0.00	92.16 ± 0.00	92.15 ± 0.00
	SG-ϵ	89.56 ± 0.00	80.22 ± 0.00	79.99 ± 0.00	97.24 ± 0.00	92.84 ± 0.00	92.84 ± 0.00
	WTRG-ϵ	87.33 ± 0.00	79.13 ± 0.00	78.94 ± 0.00	96.79 ± 0.00	92.56 ± 0.00	92.55 ± 0.00
	TSG-τ	91.28 ± 0.00	81.85 ± 0.00	81.63 ± 0.00	97.12 ± 0.00	90.24 ± 0.00	90.19 ± 0.00
	TSG-ϵ	92.50 ± 0.00	82.01 ± 0.00	81.67 ± 0.00	97.18 ± 0.00	89.84 ± 0.00	89.78 ± 0.00
CTDNE		92.70 ± 0.12	86.29 ± 0.14	86.24 ± 0.14	97.43 ± 0.01	91.81 ± 0.22	91.79 ± 0.22
Node2bits		88.97 ± 0.16	80.69 ± 0.48	80.43 ± 0.51	97.01 ± 0.03	90.85 ± 0.15	90.82 ± 0.15
DANE		73.84 ± 0.00	67.29 ± 0.00	64.92 ± 0.00	75.31 ± 0.00	74.13 ± 0.00	73.22 ± 0.00
TIMERS	-	63.50 ± 0.00	61.68 ± 0.00	58.83 ± 0.00	96.30 ± 0.00	89.00 ± 0.00	88.93 ± 0.00
DynAE		61.41 ± 0.21	58.57 ± 0.00	55.36 ± 0.00	73.23 ± 0.94	71.57 ± 0.26	70.82 ± 0.31
DynAERNN		57.30 ± 3.93	58.36 ± 0.46	54.83 ± 0.61	72.48 ± 0.92	71.20 ± 1.32	70.34 ± 1.63
DySAT		61.08 ± 0.12	58.10 ± 0.68	58.09 ± 0.68	92.97 ± 1.14	84.06 ± 0.97	84.15 ± 1.51

approaches. We leverage Equation (5.7) and Equation (5.8) to aggregate the extensive results for interpretation. In Table 5.7, we only show the complete results on the first 3 datasets used in this work due to the space limit.

5.8 Conclusion

Despite the recent increasing interest in temporal networks in the field of representation learning, there has been relatively little work that systematically studies the properties of temporal network models and their cornerstones, the graph time-series representations. This works attempts to fill this gap by proposing a general yet powerful framework. Specifically, we introduce the notion of ϵ-graph time-series to address data imbalance that arise with the traditional way of deriving a graph time-series based on a—sometimes arbitrary—time-scale (*e.g.*, 1 day or 1 week). We find that the ϵ-graph time-series is beneficial to most existing embedding methods in temporal link prediction. Furthermore, our proposed framework gives rise to new dynamic embedding methods by combining these graph time-series representations, temporal models, and base static embedding methods. We find that although there is no single temporal model (or embedding method) that could prevail on any dataset, our proposed WTRG model and TSG model along with the ϵ time-series tend to perform the best across all datasets studied. We further show that these dynamic embedding approaches from our framework outperform recent, powerful dynamic embedding methods.

Part II: Node Embedding via Non-latent Feature Importance Summarization

CHAPTER VI

Domain-knowledge-guided Summarization of Graph Collections

This chapter is based on work that appeared at ICDM 2017 [JK17].

6.1 Introduction

So far, we have introduced new embedding methodology using the idea of graph summarization and verified the effectiveness of summarizing latent graph structural features and temporal proximity on a variety of machine learning tasks. Now we turn our focus to original graph features, as these non-latent features are widely used in traditional summarization approaches for applications that require interpretability, such as visualization or query answering. The remainder of this thesis focuses on the following question: how can we leverage these non-latent features to perform ML tasks to achieve state-of-the-art performance, while retaining interpretability? How can we use the non-latent feature summaries as the high-level knowledge to handle more advanced ML scenarios such as transfer learning?

Technological advances have led to a tremendous increase in the collected data at a finer granularity than ever, including scientific data from different domains that has the potential to lead to new knowledge. Graphs are prevalent in scientific and other data, as they naturally encode various phenomena like structural or functional brain connectivity in neuroscience [DH14], compounds in chemistry, protein interactions in biology, symptom relations in healthcare [SSTZ12], behavioral patterns in social sciences, mobility patterns

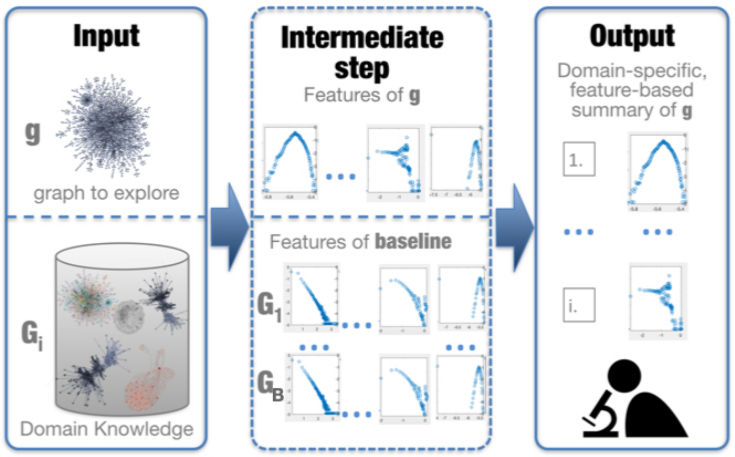


Figure 6.1: Overview of EAGLE: Given an input graph g and a set of B baseline graphs G_i that encode the domain knowledge, we seek to find a domain-specific, feature-based summary of g that is diverse, concise, and interpretable. The summary consists of univariate feature distributions (e.g., degree, PageRank).

in transportation engineering, and more. However, the size and complexity of these graphs call for statistical and programmatic tools that can harness them. Motivated by this need, we focus on the problem of summarizing graph data in a scalable and domain-aware way, enabling the extraction of intelligible information.

The typical first step of exploring a new graph dataset (e.g., brain connectome; social, technological, or communication network) often involves plotting, fitting, seeking for outliers in, and summarizing the distributions of various graph invariants (or features) such as degree, PageRank, radius, local clustering coefficient, eigenvectors, node attributes, and many more. Univariate and bivariate distributions are often used in graph mining to discover anomalous patterns at the node or graph level ([ACK+12, KJNF15, JCB+14]). However, the features to be explored are usually determined in a feature engineering approach, which heavily depends on the analyst's knowledge, intuition, and prior studies. For example, in connectomics, typical features for comparing healthy and non-healthy populations include the average degree, clustering coefficient, path length [BS09, DH14].

Moreover, the features selected in existing techniques are determined by the choice of evaluation metrics and are task-dependent. For example, highly correlated features are more likely to be chosen in clustering; independent features are more likely to be chosen for classification. Recent developments in representation learning study latent feature representations via optimization frameworks. Although they are promising and remove the ad-hoc property of feature engineering, they return latent representations which are hard to interpret and are mostly suited for specific tasks such as link prediction and multi-label classification. Therefore, there is need for a general summarization or feature selection technique for exploring graph properties independent of specific tasks.

Proposed Approach: Motivated by these observations, our proposed method, EAGLE, aims to model the exploratory analysis of graph data as a mathematically rigorous feature selection problem which is automatically *guided by* and, thus, *conditioned on the domain* of the data. Throughout the paper, *features* is used to refer to a combination of graph invariants, or structural node attributes (discrete or continuous—e.g., degree, PageRank, clustering coefficient), and categorical or numerical node attributes. Each feature is represented by its (univariate) distribution over the nodes in the graph. Specifically, EAGLE seeks to summarize an input graph g with the aid of a small set of features by leveraging the information encoded in a set of "baseline" graphs G_i for $i \in \{1, 2, \ldots, k\}$, which, in combination with their invariant distributions, represent the *domain knowledge*.

For instance in Fig. 6.1, let the input graph be a new social network (g) and the domain contain well-established social networks (G_i). A 'surprising' summary of g would consist of a small set of features including the degree distribution (the leftmost distribution in the central box) which follows the Gaussian distribution, while in the domain a power-law distribution is expected. Our approach can be seen either as feature-based graph summarization, or domain-specific feature selection that seeks to choose some features for an input graph *conditioned on* the features of the baseline graphs. This conditional property sets our work apart from traditional feature selection methods that jointly operate on a set of observations (e.g., select features from multiple graphs).

We formalize the problem as an optimization model that outputs an interpretable, feature-based summary satisfying four important properties: diversity, conciseness, domain specificity, and efficiency. Application-wise, we consider the cases where the number of features in the summary (i) can be defined via prior knowledge or domain expertise, or (ii) need to be defined automatically. Our main contributions are:

• **Novel Formulation:** We propose a new mathematical formulation of graph exploration as a *conditional* feature selection problem over structural or other node attributes. The goal of our proposed constrained optimization framework is to find a diverse, succinct, domain-specific summary for the input graph, which is also interpretable.

• **Scalable Algorithms:** We propose Eagle-Fix and Eagle-Flex, two efficient methods for obtaining the desired summaries. To speed up our methods, we carefully handle the correlations between graph features by systematically investigating their affinities in a data-driven way.

• **Experiments:** We compare EAGLE with baseline approaches on a variety of real-world datasets (including social networks, citation networks, and human connectomes) and show that it satisfies all the desired properties and it is scalable. Although our approach is task-independent, we show that it can be applied to traditional graph mining tasks, such as classification.

6.2 Related Work

Our work is related to several research directions that are additional to work described in Chapter II:

Feature selection. The process of feature selection consists of two parts: a search technique for proposing new feature subsets, and a measure for evaluating these different feature subsets. Search techniques vary from exhaustive [GE03] to improved ones, such as greedy hill climbing. Evaluation metrics are divided into three categories: wrappers (which use predictive models to score feature subsets, e.g., [PLD05]), filters (which use measures, such as pointwise mutual information [YP97]), and embedded methods (which perform selection as part of the model construction process [Bac08]). Our proposed method, EAGLE, is the first approach searching for features greedily based on the domain knowledge and expectations and specifically targeting the graph setting. Moreover, while the above methods select features by jointly learning from all the available observations, our method performs a 'customized' feature selection for a given graph *conditioned* on observations from a set of baseline graphs. Though EAGLE is used for summarizing a dataset with desired properties and there is no particular task guiding its evaluation, we showcase how to adapt it for task-dependent evaluation too.

Pattern mining and Summaries. Mining static graphs often involves analyzing the distributions of specific graph invariants (e.g., skewed degree distribution [FFF99] in numerous settings, small-worldness in connectomics [BS09, DH14]), and speeding up their computations (e.g., betweenness centrality [BKMM07]). Moreover, systems [ACK$^+$12, KJNF15] have been proposed for anomaly detection via analyzing specific distributions of graph invariants, and spam detection on bivariate distributions. These methods focus on modeling manually-chosen distributions of invariants and potentially finding outliers in them, while our work aims to *automatically detect the features* that summarize a given graph *depending on its domain*. Moreover, we assume that fast methods are used prior to applying EAGLE in order to obtain the distributions of various node invariants. Although EAGLE finds feature-based summaries for an input graph, our work differs significantly from graph summarization [LDSK16, KKVF14], which typically seeks to find a compact representation of a network with fewer nodes/links.

Similarity/Distance and Interestingness measures. An excellent review of existing distance/similarity measures for distributions is given in [Cha07]. Attempts to define the interestingness of a plot or distribution by studying its geometric properties [GH06] include: SCAGNOSTICS [WAG05], which ranks and guides the interactive exploration of bivariate distributions, and motif-based interestingness measures for local patterns in scatterplots [SSB$^+$15]. However, unlike our work, these methods are unaware of the domain and introduce generic measures that define the 'interestingness' of each plot independently.

6.3 Methodology: EAGLE

Motivated by the large amounts of graph data and the prevalent need for exploratory analysis in various areas (e.g., neuroscience, social science), we focus on generating interpretable graph summaries by leveraging the domain knowledge:

Definition 18 (Domain Knowledge). *We refer to the expected patterns (or laws) for the distributions of node invariants or other attributes in a specific area as the domain knowledge.*

Examples of graph invariants include global structural statistics such as the degree and

PageRank; local structural statistics such as the egonet size, interactions to neighbors, and properties revealed by different algorithms such as community detection. In social science, examples of categorical and numerical attributes are the gender and age of a user, respectively. Our assumption is that the domain expectations are implicitly encoded in a set of *baseline* graphs which belong to that domain. For example, in social networks many distributions of structural attributes (e.g., degree variants, PageRank) are *expected* to follow a power law [FFF99], while in functional connectomes that are produced via neuroimaging techniques more uniform distributions are expected. Based on this definition, we state the problem that we tackle as follows:

Problem 2 (Exploratory Analysis of Graph Data using Domain Knowledge)**.** *Given the node features of a plain or attributed input graph g and a set $\mathcal{G}$ of baseline graphs G_i, $i = 1, \ldots, B$, we seek to find a domain-specific summary consisting of a small set of representative and interpretable features in an efficient way.*

If g is attributed, the features consist of invariants and node attributes. Otherwise, the features include only node invariants. Our main idea is to formulate the exploratory analysis of graphs as an optimization model that will produce as an output a feature-based summary with four desired properties:

• **P1. High Diversity / Coverage.** The summary is required to 'cover' the information or patterns or laws encoded in the baseline graphs: the features in the summary should provide *diverse* aspects of the domain knowledge. We measure *diversity* between the features through the concept of "similarity", so the features in the summary should have trivial dependence.

• **P2. Conciseness.** Although diversity is crucial for good summaries, it connives the "greed" to select features: the most diverse summary should contain many features. To avoid duplication and verbosity, *conciseness* indicates that the number of features in the summary should be small.

• **P3. Domain-specificity.** Based on the information of the baseline graphs $\mathcal{G}$, the summary of g should be related or contrasted to the features of the baseline graphs. For example, if a 'contrasted' summary is required and all the baselines follow a power law degree distribution (e.g., social networks) while g does not, the degree distribution should be included in the summary. However, a 'contrasted' summary in a different domain (e.g., neuroscience) would include different features.

• **P4. Efficiency.** Given the soaring amount of data being generated daily, the computation of the summary must be efficient and scale to large amounts of data.

Moreover, an informal desired property is that the selected features are interpretable and easy-to-understand. To that end, unlike network embedding or factorization-based methods, we seek summaries that do *not* rely on latent features. Next we introduce our proposed optimization framework. For reference, we list the major symbols that are additional to Table 2.1 from Chapter II in Table 6.1.

Table 6.1: Table of symbols.

Symbol	**Definition**
$\mathcal{G}$	a collection of baseline graphs, $\mathcal{G} = \{G_1, G_2, \dots, G_K\}$
g	input graph
K	total number of baseline graphs
F	size of feature space
B	number of buckets in a distribution
$\lambda_{1,2,3}$	regularization parameters
$\mathbf{f}$	$F \times 1$ indicator vector for selected features, $\mathbf{f} \in \{0,1\}^F$
$\mathbf{S_F}$	$F \times F$ pairwise feature relevance matrix for the baseline graphs $\mathcal{G}$
$\mathbf{S_F}_i$	$F \times F$ pairwise feature relevance matrix for baseline graph G_i
$\mathbf{w}$	$K \times 1$ weight vector for the baseline graphs in $\mathcal{G}$, $\sum_i^K w_i = 1$
$\mathbf{h}$	$F \times 1$ vector denoting similarity / distance between equivalent marginal distributions (e.g., degree) of g and $\mathcal{G}$
$\phi(\cdot)$	coupling function of the input graph g and the baseline graphs $\mathcal{G}$

6.3.1 Proposed Formulation

We propose to model the Exploratory Analysis of Graph Data problem as an optimization problem that encodes the above-mentioned desired properties and selects the features to add in the summary such that:

$$\arg\min_{\mathbf{f}} \lambda_1 \underbrace{\mathbf{f}^T \mathbf{S_F} \mathbf{f}}_{\text{1st term}} + \lambda_2 \underbrace{||\mathbf{f}||_0}_{\text{2nd term}} + \lambda_3 \cdot \underbrace{\phi(g, G_1, G_2, \dots, G_B)}_{\text{3rd term}} \tag{6.1}$$

where $\mathbf{f} \in \{0,1\}^F$ is the vector indicating the selected features; $\mathbf{S_F}$ is the aggregated matrix that represents the pairwise feature relevance in the domain of interest, as encoded in the baseline graphs $\mathcal{G}$; $||\mathbf{f}||_0$ is the $l0$-norm of the indicator feature vector; $\phi()$ is a function that couples the input graph g and the baseline graphs, thus grounding the summary to domain; and $\lambda_1, \lambda_2, \lambda_3$ are regularization parameters which are set so that the three terms are comparable (cf. § 6.3.5).

Intuitively, the **first** quadratic **term**, $\mathbf{f}^T\mathbf{S_F}\mathbf{f}$, forces the selected features to be diverse. It uses the baseline graphs to establish the 'norms' in the domain of interest and uses them to capture the relevance between all pairs of graph invariants. Specifically, $\mathbf{S_F}$ represents the aggregate of the 'correlation' or relevance between all F features over the baseline graphs $\mathcal{G}$, while the quadratic term evaluates the sum of relevance scores of selected features. The regularization parameter λ_1 is set to a positive number (discussed later). Unlike existing work, this term quantifies the relevance between *different* graph invariants (e.g., PageRank and local clustering coefficient) in the domain by harnessing the information in the baseline graphs.

The **second term**, $\|\mathbf{f}\|_0$, which is multiplied by a positive regularization parameter λ_2, requires that the summary is concise, i.e., it consists of a few features. Although, ideally, the $l0$-norm encodes this requirement, we will later relax this constraint to the $l2$-norm which is mathematically tractable.

The **last term**, $\phi(g, G_1, \dots, G_k)$, is crucial because it couples the input graph g and the domain knowledge. It can be interpreted as the term that forces the features that will be selected for the summary to come as close (or far) as possible to those of the baseline graphs. That way, it can be tuned to provide an 'ordinary/expected' summary or a 'surprising' summary. This is useful when an analyst who knows the information that is being captured in the baseline graphs (e.g., connectomes of subjects with depression) wants to see a holistic overview of the feature-based similarities and possible differences of a newly obtained graph (e.g., connectome of a new subject). When $\phi()$ is a positive, increasing function of $\mathbf{f}$, we have the so-called "0 pit" problem of Equation (6.1):

Definition 19 (The 0-pit problem)**.** *When the three terms of Equation* (6.1) *are positive, the solution is* $\mathbf{0}^{F\times 1}$ *irrespectively of the input and baseline node invariants, i.e., the objective function falls into a "pit" with optimal value* 0.

To handle this problem, we add constraints to our optimization problem. We elaborate more on the design choices of this term and the additional constraints in Section 6.3.3.

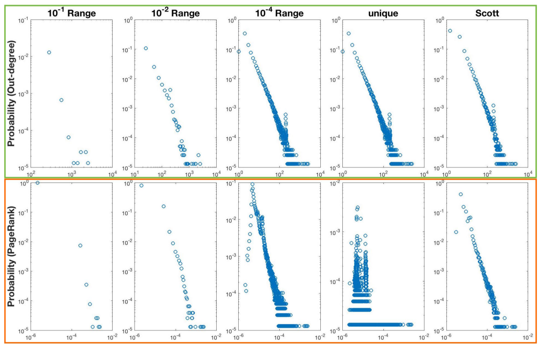
The efficiency of computing the summary comes from our proposed framework, which we discuss in Section 6.3.4. The additional (informal) requirement for interpretability follows from our feature representation in $\mathbf{f}$. As opposed to latent representations that are hard to interpret, in our work the selected features correspond to node invariants (e.g., degree, PageRank) or node attributes, which depend on the domain. Throughout our formulation, we assume that the graph features are represented by their PDFs (Probability Density Function) and adapt appropriate measures to quantify their relevance/dissimilarities.

6.3.2 Proposed Model for Feature Diversity

As we mentioned above, the first term in our proposed optimization function enforces diversity in the selected features so that they are not correlated. In this subsection we discuss how we design $\mathbf{S_F}$ in order to capture the 'correlation' between the node invariants per baseline graph. Assuming that only the PDFs of the node invariants are provided, computing the correlation between the corresponding invariants is not feasible (more information per node would be needed). Thus, we use feature relevance or similarity between different invariants as a surrogate correlation model.

In general, the features (node invariants) that are considered can be: discrete (e.g., degree distribution) or continuous (e.g., PageRank distribution). If we view each PDF i as a vector of length l_i, it can be seen that different invariants are represented by distribution vectors of different lengths, which leads to two main challenges: (i) What is the right length for each distribution vector, or, put differently, what is the proper size of buckets to be used in different node invariant distributions? and (ii) How can we compute the relevance between two PDFs of different lengths? We address these two questions next.

(i) A general feature representation model. In order to compute the relevance between the features in the baseline graphs, we first need to define the feature model. As we mentioned, we view each feature i as the PDF of the corresponding invariant, which can be represented as a vector of length l_i or, equivalently, l_i 'buckets'. If the PDF is organized in a large number of buckets, the histogram "looks" uniform, while a small number of buckets results in information loss by aggregating many original values into one bucket.


(a) Social science: Slashdot [SNA]

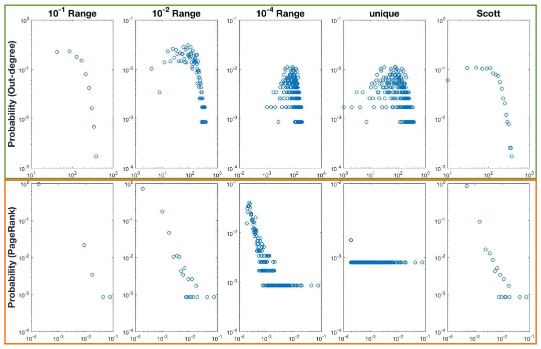

(b) Neuroscience: Functional connectome

Figure 6.2: The discrete and continuous PDFs with different bucket sizing, from left to right, the bucket sizing is: $\frac{1}{10}$, $\frac{1}{100}$, $\frac{1}{10000}$ times the range of values; "unique" means the unique values in the PDF; "Scott" refers to the bucket sizing computed by Scott's rule.

Visualizing the feature distributions involves selecting the number of buckets l_i. For example, for a degree distribution, the number of buckets is equal to the number of unique node degrees, while for a PageRank distribution the number of buckets depends on the analyst and the data at hand. As we see in Fig. 6.2, the number of buckets is critical when computing the relevance between two features via their PDFs, as they can lead to different 'shapes' of distributions, and help with or prevent the detection of patterns (e.g., spikes). Fig. 6.2 indicates that a large number of buckets helps show the pattern of discrete PDFs such as the power-law of the out-degree distribution with 10^{-4} range in Fig. 6.2a, yet a small number of buckets fails to reflect the actual pattern and may miss the spikes that often indicate anomalies. On the contrary, for continuous PDFs, many buckets blur the patterns as the values in the distribution may differ slightly, while fewer buckets may address this problem. This is illustrated through the "uniform" distribution with unique bucketing in Fig. 6.2b.

We propose to find proper bucket sizing for *any* (discrete or continuous) PDF by adapting Scott's reference rule [Sco79]:

$$\text{Bucket size} = 3.5 \cdot \hat{\delta}/n^{1/3} \tag{6.2}$$

where $\hat{\delta}$ is the sample standard deviation and n is the number of elements in the distribution. The distribution plots labeled "Scott" in Fig. 6.2 illustrate the effectiveness of Scott's rule by capturing not only the pattern, but also existing spikes. Scott's rule generates a flexible number of buckets for different PDFs, and it applies to both big and small graphs. There are several variants such as Sturges' formula [Stu26] and Freedman–Diaconis' rule [FD81], all apply to different settings. For generality, we integrate all these rules including the fixed sizing in the proposed framework and use Scott's rule to conduct computation and experiments.

(ii) A surrogate feature correlation model. Assuming that only the PDFs of the node invariants are provided, computing the correlation between the corresponding invariants is not feasible (more information per node would be needed). Thus, we use feature relevance or similarity between different invariants as a surrogate correlation model. Other traditional distance-based measures [Cha07] can be applied when two distribution vectors are of the same length, but, as we saw above, this is usually not the case when dealing with distributions of different invariants, e.g., degree vs. PageRank. For PDFs of different lengths, such as the ones generated by Scott's rule, those measures are not suitable unless they are normalized to have the same length. We discussed the challenges of such normalization above (a general feature representation model).

To emphasize the importance of 'shape' match between distributions of different invariants, and not point-wise match, we propose to leverage the dynamic time warping (DTW) algorithm. DTW is designed to calculate an optimal match between two given sequences by "warping" them non-linearly, so that the distance calculated is independent of variations in the warped dimension. For PDFs that denote the graph statistics distributions, DTW calculates the feature-by-feature distance independent of variations in the number of buckets, which can be converted to similarity in many ways, including $s = (1 + d)^{-1}$. DTW-based similarity works for both cases whether two PDFs are of the same or different lengths.

For generality, we integrate DTW and traditional distance-based methods in the proposed framework and primarily use DTW similarity in our experiments. Per baseline graph G_i, we compute the pairwise feature relevance matrix $\mathbf{S_F}_i$:

$$\mathbf{S_F}_i(f_j, f_l) = s(PDF_{G_i,f_j}, PDF_{G_i,f_l}) \tag{6.3}$$

where PDF_{G_i,f_j} is the PDF for the j^{th} feature of graph G_i, and $s()$ is the desired similarity between two distributions. By definition, the diagonal elements of each relevance matrix are

1. We can obtain the aggregate pairwise feature relevance matrix as their weighted sum:

$$\mathbf{S_F}(f_j, f_l) = \sum_{i=1}^{B} w_i \cdot \mathbf{S_F}_i(f_j, f_l) \tag{6.4}$$

where w_i is the weight or 'importance' of graph G_i in the computation, and $\sum_{i=1}^{B} w_i = 1$.

6.3.3 Proposed Model for Domain-Specificity

The **last term**, $\phi(g, G_1, \ldots, G_k)$, in Eq. (6.1) couples the input graph g and the domain knowledge. Unlike prior work in the literature which focuses on one graph only and assigns interestingness or anomaly scores to a distribution independently of the domain knowledge (e.g., Scagnostics [WAG05]), the third term aims to find the distributions that bear the most or fewest number of similarities with other graphs in the domain.

We propose to model the domain specificity with a simple and intuitive linear formulation, $\phi(g, G_1, \ldots, G_k) = \mathbf{f}^T\mathbf{h}$, where h_j in $\mathbf{h} = [h_1, h_2, \ldots, h_F]$ is the aggregate relation score between the j^{th} marginal distributions (e.g., degree) of g and the baseline graphs G_i. The relation can be set to be a similarity or a distance measure resulting in an 'ordinary' or 'surprising' summary (§ 6.3.5). This choice is directly related to the "0 pit" problem: (i) If $\mathbf{h}$ is modeled as similarity, we need to force the solution of the optimization problem to make selections by adding constraints on $\mathbf{f}$; and (ii) If $\mathbf{h}$ is modeled as distance, the last term becomes negative (i.e., minimizing the 'negative' distance) by setting $\lambda_3 < 0$.

Unlike $\mathbf{S_F}$ which computes the relevance between different invariant distributions of a single graph G_i, $\mathbf{h}$ focuses on the relation between *equivalent* distributions of the input graph g and the baseline graphs G_i. The aggregate relation between the input g and the domain is computed as the weighted average of the relations between all the combinations of g and the baseline graphs G_i. We use h_{sj} to represent the j^{th} entry of the relation vector based on similarity:

$$h_{sj} = \sum_{i=1}^{B} w_i \cdot s(PDF_{g,f_j}, PDF_{G_i,f_j}) \tag{6.5}$$

Similarly, $\mathbf{h_d}$ represents the relation vector based on a distance measure, and is defined equivalently (by replacing $s()$ with a distance measure $d()$.

6.3.4 Algorithm

Our proposed formulation in Optimization Problem 6.1 corresponds to a mixed-integer quadratic programming (MIQP) problem. The problem of 0–1 integer programming is NP-complete and the integral constraints bring challenges such as intractability and poorly-behaved derivatives, which make algorithms such as gradient descent unwarranted. To solve these challenges, we first explain how we approximate MIQP with a sequence of mixed-integer linear programming (MILP), and then propose two solutions to the "0 pit" problem by adding application-driven constraints in Section 6.3.5. We give the theoretical analysis on complexity in Section 6.3.6.

Although the $l0$-norm in Eq. (6.1) encodes the conciseness requirement, we relax it by using the $l2$-norm, which is mathematically tractable. By rewriting $\|\mathbf{f}\|_2^2 = \mathbf{f}^T\mathbf{f}$ and using the $F \times F$ identity matrix $\mathbf{I_F}$, the equation takes the form:

$$\arg \min_{\mathbf{f} \in \{0,1\}^{F \times 1}} \mathbf{f}^T \underbrace{(\lambda_1 \mathbf{S_F} + \lambda_2 \mathbf{I_F})}_{\mathbf{Q}} \mathbf{f} + \mathbf{f}^T \underbrace{\lambda_3 \mathbf{h}}_{\mathbf{r}} . \tag{6.6}$$

The integer vector $\mathbf{f}$ can be expressed as the linear constraint to Eq. (6.6) thus obtaining the form of MIQP:

$$\begin{aligned} \underset{\mathbf{f}}{\text{minimize}} \quad & \mathbf{f}^T \mathbf{Q} \mathbf{f} + \mathbf{r}^T \mathbf{f} \\ \text{subject to} \quad & 0 \le \textstyle\sum_i^F \mathbf{f}(i) \le F \\ & 0 \le \mathbf{f}(i) \le 1, \ i = 1, \dots, F. \end{aligned} \tag{6.7}$$

We apply the cutting plane method [Kel60] to convert Problem 6.7 to a series MILP by introducing a slack variable z:

$$\begin{aligned} \underset{\mathbf{f}, z}{\text{minimize}} \quad & z + \mathbf{r}^T \mathbf{f} \\ \text{subject to} \quad & 0 \le \textstyle\sum_i^F \mathbf{f}(i) \le F \\ & 0 \le \mathbf{f}(i) \le 1, \ i = 1, \dots, F. \\ & \mathbf{f}^T \mathbf{Q} \mathbf{f} - z \le 0, \ z \ge 0 \end{aligned} \tag{6.8}$$

Problem 6.8 gives the local MILP approximation to Problem 6.7 at one step. To further approximate the MIQP, we need to iteratively solve a series of MILP by updating the linear constraints until convergence. To update the linear constraints, we denote $\mathbf{f}$ at the t^{th} iteration as $\mathbf{f}_t$ such that $\mathbf{f}_t = \mathbf{f}_{t-1} + \boldsymbol{\delta}$, where $\mathbf{f}_{t-1}$ is the vector obtained in the previous iteration and $\boldsymbol{\delta}$ is a variable vector. By using first-order Taylor approximation for the last constraint in Problem (6.8), we obtain:

$$\begin{aligned}\mathbf{f}_t^T\mathbf{Q}\mathbf{f}_t - z &= \mathbf{f}_{t-1}^T\mathbf{Q}\mathbf{f}_{t-1} + 2\mathbf{f}_{t-1}^T\mathbf{Q}\boldsymbol{\delta} - z + O(|\boldsymbol{\delta}|^2)\\ &= -\mathbf{f}_{t-1}^T\mathbf{Q}\mathbf{f}_{t-1} + 2\mathbf{f}_{t-1}^T\mathbf{Q}\mathbf{f}_t - z + O(|\mathbf{f}_t - \mathbf{f}_{t-1}|^2)\\ &\approx -\mathbf{f}_{t-1}^T\mathbf{Q}\mathbf{f}_{t-1} + 2\mathbf{f}_{t-1}^T\mathbf{Q}\mathbf{f} - z \leq 0,\end{aligned}$$

where we omitted the second-order terms. Thus, to solve the MIQP of Problem (6.7), we need to solve a series of MILPs in Problem (6.8) combined with this updated linear constraint.

6.3.5 Application-driven Constraints

As we mentioned in Section 6.3.1, the last term of Eq. (6.6) can be tuned to provide an 'ordinary/expected' summary or a 'surprising' summary by identifying features that are similar or dissimilar to the ones in the baseline graphs, respectively. During exploratory analysis, this allows for some flexibility about the type of relevance that is sought between the summary of g and the baseline graphs $\mathcal{G}$. Next, without loss of generality, we focus on surprising summaries, and introduce two application-driven constraints: (i) fixed, and (ii) flexible number of features in the summary. Our analysis can be easily extended to the case of ordinary summaries as well.

A1. EAGLE-FIX: Fixed number of features. In the case of creating a surprising summary for the input graph, the last term in Eq. (6.6) can be set such that $\mathbf{h}$ captures similarities between the features of g and G_i, i.e., it is computed based on Eq. (6.5) and denoted by $\mathbf{h_s}$. To solve the 0-pit problem, we introduce a capacity constraint for the summary, in addition to the constraints that are given in Problem (6.7), and set $\mathbf{r} = \lambda_3\mathbf{h_s}$:

$$\textstyle\sum_i^F \mathbf{f}(i) = c \qquad [\textit{new} \text{ capacity constraint}] \tag{6.9}$$

To prevent the objective function from reaching the optimum with some desired properties

Algorithm VI.1 EAGLE-FIX

Input: Graph g with F invariant distributions; Graph database with G_i ($i = 1 \dots B$) graphs with their F invariant distributions
Output: Binary vector $\mathbf{f}$ of selected features in the summary of g

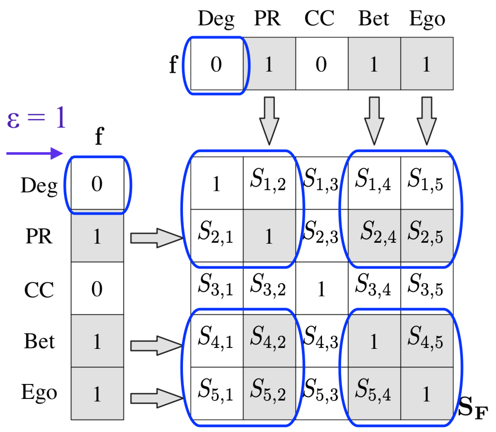
```
1  I. Preprocessing Phase: Computations over the Domain
2  for $i = 1 \dots B$
3      // Step 1: Feature Representation Model
4      for $j = 1 \dots F$
5          $PDF^{new}_{G_i,f_j} = \text{Scott}(PDF_{G_i,f_j})$                ▷ Scott's rule, Eq. (6.2)
6      // Step 2: Feature Diversity Model
7      for $j = 1 \dots F$, and $l = j+1 \dots F$
8              $\mathbf{S_F}_i(f_j, f_l) = s(PDF^{new}_{G_i,f_j}, PDF^{new}_{G_i,f_l})$    ▷ Eq. (6.3)
9  $\mathbf{S_F}(f_j, f_l) = \sum_{i=1}^{B} w_i \cdot \mathbf{S_F}_i(f_j, f_l)$              ▷ Eq. (6.4)
10 II. Query Phase: Summary Creation
11 Step 1: Domain-specificity Model
12 for $l = 1 \dots F$
13     $h_{sl} = \sum_{i=1}^{B} w_i \cdot s(PDF^{new}_{g,f_l}, PDF^{new}_{G_i,f_l})$       ▷ Eq. (6.5)
14 Step 2: Feature Selection
15 $\mathbf{Q} = \lambda_1 \mathbf{S_F} + \lambda_2 \mathbf{I_F}$        ▷ Regularization parameters $\lambda_1, \lambda_2, \lambda_3$
16 $\mathbf{r} = \lambda_3 \mathbf{h}_s$
17 $\mathbf{f} = \text{MIQP}(\mathbf{Q}, \mathbf{r})$                    ▷ Solve Problem (6.7)
```

overwhelming the others, $\lambda_{\{1,2,3\}}$ should be set such that the three terms in Optimization Problem 6.1 are comparable (i.e., of the same scale). The values of these normalization terms are primarily determined by the maximums of (i) $\mathbf{f}^T\mathbf{S_F}\mathbf{f}$, (i) $\|\mathbf{f}\|_2^2$, or $\mathbf{f}^T\mathbf{f}$ and (iii) $\mathbf{f}^T\mathbf{h}$. We discuss the parameter setting in the experiments (Section 6.4).

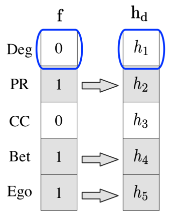
Putting everything together, in the case of finding surprising summaries for a given input, we propose the EAGLE-FIX algorithm, for which we give the pseudocode in Algorithm VI.1.

A2. EAGLE-FLEX: Flexible number of features. In the case of creating a surprising summary for the input graph, we can search for a flexible number of features by setting the last term in Eq. (6.6) such that it captures the distances between the features of g and G_i (i.e., $\mathbf{h} = \mathbf{h_d}$) and $\lambda_3 < 0$. Note that a very small value λ_3 may render the third term smaller than other terms, which would lead the objective function to fall into the "0 pit". Therefore, to determine the regularization parameters in this case, we propose a different technique that obtains the range of λ_3 based on λ_1 and λ_2 values.

• *Upper bound for* λ_3. Suppose there are $c \geq 0$ selections in the solution $\mathbf{f}$. Then, the value of the relaxed objective function (6.6) can be calculated as:



(a) First term: $\mathbf{f}^T\mathbf{S_F}\mathbf{f}$



(b) Third term: $\mathbf{f}^T\mathbf{h_d}$

Figure 6.3: Example: $\mathcal{S} = \{2, 4, 5\}$, $\mathbf{f} = \{0, 1, 0, 1, 1\}$, and degree as the newly added feature (i.e., $\epsilon = 1$). (a) The sum of the shaded areas in $\mathbf{S_F}$ corresponds to the first term. After adding the degree, i.e., $\mathcal{S}' = \mathcal{S} \cup \{1\}$, the sum of the blue rectangles correspond to the first term. (b) Blue rounded rectangles in h_d indicate $h_d(\epsilon)$; The sum of its shaded cells gives the third term.

$$\lambda_1 \textstyle\sum_{i,j\in\mathcal{S}} S_F(i,j) + \lambda_2 c + \lambda_3 \sum_{i\in\mathcal{S}} h_d(i) \tag{6.10}$$

where $\mathcal{S}$ denotes the collection of the indices of selected features $\mathbf{f}$, which is explained in Fig. 6.3. When $c = 0$, $\mathcal{S} = \emptyset$ Similarly, when there are $c + 1$ selections, the value of the objective function is:

$$\lambda_1 \textstyle\sum_{i,j\in\mathcal{S}'} S_F(i,j) + \lambda_2 (c+1) + \lambda_3 \sum_{i\in\mathcal{S}'} h_d(i) \tag{6.11}$$

where $\mathcal{S}' = \mathcal{S} \cup \{\epsilon\}$, and $\{\epsilon\}$ denotes the index of the newly selected feature. Our proposed Optimization Problem 6.1 will only select $c + 1$ features if that further reduces the objective function, which implies that Eq. (6.10) > (6.11), or:

$$\begin{aligned} \lambda_3 &< -\frac{\lambda_1(\sum_{i,j\in\mathcal{S}'} S_F(i,j) - \sum_{i,j\in\mathcal{S}} S_F(i,j)) + \lambda_2}{\sum_{i\in\mathcal{S}'} h_d(i) - \sum_{i\in\mathcal{S}} h_d(i)} \Rightarrow \\ \lambda_3 &< -\frac{\lambda_1(\sum_{i\in\mathcal{S}} S_F(i,\epsilon) + \sum_{i\in\mathcal{S}} S_F(\epsilon,i) + 1) + \lambda_2}{h_d(\epsilon)} \end{aligned} \tag{6.12}$$

By assuming that ϵ corresponds to the maximum entry in $\mathbf{h_d}$, we obtain the upper bound of λ_3:

$$\lambda_3 < -\frac{\lambda_1 + \lambda_2}{\max(\mathbf{h_d})} \tag{6.13}$$

• *Lower bound for λ_3.* By requiring the three terms in the optimization problem to be comparable, we can obtain a lower bound for λ_3. Assuming that $\lambda_3 < 0$ and $c = |\mathcal{S}|$, the third term must be smaller or equal to the maximum of the others:

$$\lambda_3 > -\frac{\max\{\lambda_1 \sum_{i,j \in \mathcal{S}} S_F(i,j), \lambda_2|\mathcal{S}|\}}{\sum_{i \in \mathcal{S}} h_d(i)} \tag{6.14}$$

Inequality (6.14) indicates that the lower bound of λ_3 is determined by $\mathcal{S}$ (it is involved in all the terms of (6.14)). In order to find the exact lower bound, we need to consider all possible sets of $\mathcal{S}$ (or equivalently, all possible binary vectors $\mathbf{f}$), which are $O(2^F)$. Thus, to reduce the complexity of its computation we provide an empirical lower bound, which works well in practice:

$$\lambda_3 \geq -\lceil \frac{\lambda_1 + \lambda_2}{\max(\mathbf{h_d})} \rceil - 1 \tag{6.15}$$

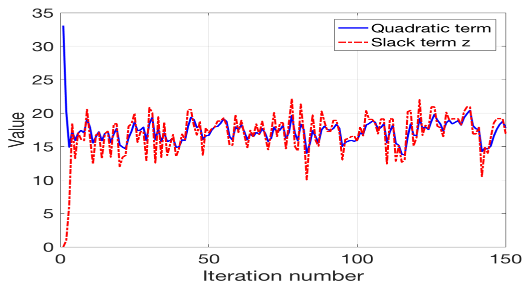
We discuss the choices of λ_1, λ_2 and λ_3 more in Section 6.4.

6.3.6 Complexity

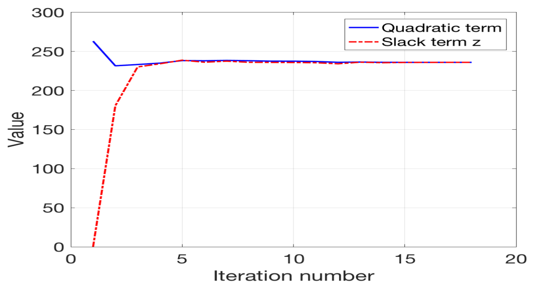
The runtime of EAGLE consists of three parts: (1) computing $\mathbf{S_F}$, (2) computing $\mathbf{h}$, and (3) runtime of MIQP. In the first two parts, the runtime τ of computing similarity / distance between two PDFs is determined by the distance measure. Although τ can be affected by the lengths of PDFs, it is generally trivial.

(1) Since $\mathbf{S_{F_i}}$ is symmetric with diagonal elements equal to 1, the number of similarity computations for one baseline graph is $O(F^2)$. $\mathbf{S_F}$ aggregates B of them, so the complexity for $\mathbf{S_F}$ is $O(\frac{KF(F-1)\tau}{2})$.

(2) The feature-by-feature relation between g and G_i constructs the $\mathbf{h_i}$ vector with $O(F)$ similarity computations. Then, $\mathbf{h}$ aggregates B of them, resulting in $O(KF\tau)$ complexity.



(a) `HepPh` **citation graph: 21 features**



(b) `Slashdot0922` **social graph: 300 perturbed features**

Figure 6.4: Convergence of two runs with MIQP.

(3) The runtime complexity of MIQP depends on the speed of convergence between the quadratic term and its linear approximation. If the convergence criterion is not reached, EAGLE would run with every possible value of $\mathbf{f}$ to reach the minimum, which is $O(2^F)$. However, empirical experiments show that in general EAGLE takes about 20∼30 iterations to reach satisfying approximation, if not converging. This is illustrated in Fig. 6.4, where we set the maximum number of iterations to be 150. Interestingly, we observe that the MIQP runtime does not only depend on the length of vector $\mathbf{f}$, but also on the values of entries in $\mathbf{f}$: If the values are small and close to each other, MIQP would require more comparisons to find the path towards the optimum (Fig. 6.4a); On the contrary, if the values differ tremendously, this procedure becomes much faster (Fig. 6.4b).

6.4 Experiments

In this section we provide thorough experimental analysis to evaluate our proposed approach. Specifically, we consider the evaluation metrics: **(1)** The satisfaction of the desired properties for exploratory analysis (P1-P3); **(2)** The scalability of EAGLE algorithm (P4); and **(3)** Its robustness to the required parameters. Moreover, we present an application of EAGLE to a graph mining task, namely the classification of patients (Schizophrenic) and healthy subjects based on fMRI data.

6.4.1 Baselines

No systematic empirical research exists that addresses the problem of finding graph summaries by automatically leveraging domain knowledge. Moreover, as we discussed in Section 6.2, unlike traditional feature selection methods that choose features by jointly operating on a set of observations, our method is 'conditional': It selects features for an *input* graph conditioned on observations from *other* graphs (domain knowledge). Despite these limitations in the literature, we evaluate the effectiveness of EAGLE against:

• **Random:** This approach randomly selects a subset of features as the summary of the input graph. It is often used as the preliminary analysis given little or no prior knowledge.

• **Scagnostics [WAG05]:** This method was proposed to summarize high-dimensional datasets by detecting anomalies in density, shape, and trend. Since it applies on bivariate distributions, we modified it to compute 9 measures (area of convex hull, skinniness, stringiness, straightness, monotonic score, skewness, clumpy score, striation, and binning score) on each one of F univariate distributions. Features with the top score in at least one measure are included in the summary.

• **Surprising:** This method is a special case of EAGLE with $\lambda_1 = \lambda_2 = 0$ and detects patterns that are different (or surprising) from the ones that appear in the baseline graphs.

6.4.2 Datasets

The real datasets that we used in our experiments are from three different domains: connectomics, citation networks, and social science. We give short descriptions of these datasets in Table 6.2. The first two connectomes, `Brain-Voxel1` and `Brain-Voxel2`, were generated using the traditional network discovery [BS09] process: (i) computation of the pairwise correlations between the 3789 time series obtained during fMRI and (ii) application of threshold ($\theta = 0.9$) to keep the most significant associations and get sparse networks.

6.4.3 Experimental setup

EAGLE is implemented in MATLAB, and all the experiments were performed on a laptop equipped with an Intel Core i7-4870HQ Processor and 16GB memory.

Table 6.2: **Domains and graphs used in our experiments.**

Domain	Name	Nodes	Edges	Description
Connectomics [con]	`Brain-Voxel1`	3 789	399 069	undirected unweighted
	`Brain-Voxel2`	3 789	148 648	undirected unweighted
	`COBRE` [cob12]	1 166	~679 000	undirected unweighted
Citation networks [SNA]	`HepTh`	27 770	352 807	directed unweighted
	`HepPh`	34 546	421 578	directed unweighted
Social science [SNA]	`Epinions`	75 879	508 837	directed unweighted
	`Slashdot0811`	77 360	905 468	directed unweighted
	`Slashdot0922`	82 168	948 464	directed unweighted

EAGLE takes an arbitrary number of graph features as input and outputs a small set of representative features as a summary based on the domain knowledge. The features used in the experiments include 28 common node- and structure-specific invariant distributions and other graph properties: The *node-specific features* that we used are in-degree, out-degree, PageRank, in-closeness, out-closeness, hubs, authorities, clustering coefficient, betweenness, top eigenvectors, network constraint, and roles [HGER$^+$12]. The *structure-specific statistics* comprise egonet features, such as out-degree, out-neighbors, in-degree, in-neighbors, and size of egonets in edges and nodes. Moreover, we considered other features, such as the distribution of communities, weak / strong connected components, in / out-going community affiliations, motifs, community profiles, random left and right singular values, and hops [SNA].

Equation (6.6) defines the relationships between the regularization terms λ_1, λ_2, and λ_3: By observing that the maximum values of the three terms are F^2, F and F, respectively, we define the relationship between the regularization terms $F\lambda_1 = \lambda_2$ and $\lambda_2 = \lambda_3$. For EAGLE-FIX, we set $\lambda_1 = \frac{1}{F}$, $\lambda_2 = 1$, and $\lambda_3 = 1$; for EAGLE-FLEX, we set $\lambda_1 = \frac{1}{F}$, $\lambda_2 = 1$, and compute λ_3 according to Equation (6.15). We use the default value $\mathbf{w} = \{\frac{1}{B}\}^{F\times 1}$ to weigh the contribution of baseline graphs. However, if prior information is available, the weights can be set differently as long as $\sum_i^B w_i = 1$.

6.4.4 Satisfaction of Desired Properties

In this experiment, we quantitatively evaluate the satisfaction of the desired properties by EAGLE and the baselines. We obtain EAGLE summaries for two input graphs, `HepPh` and `Slashdot0922`, considering three domains with different sets of baseline graphs: (i) connectomics using `Brain-Voxel1` and `Brain-Voxel2`; (ii) citation networks using `HepTh`;

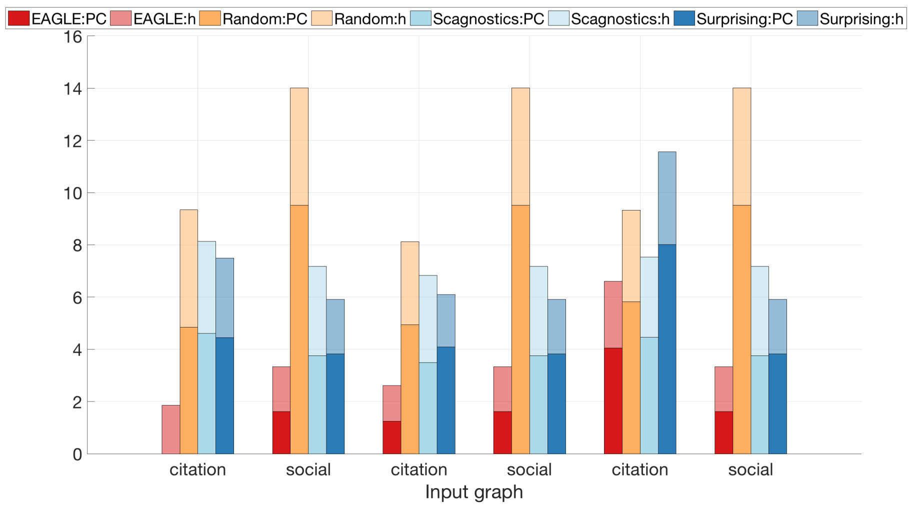


Figure 6.5: Effectiveness in terms of diversity and domain-specificity evaluated using Pearson's correlation coefficient (low values are better). EAGLE achieves the best performance in every case.

and (iii) social networks using `Epinions` and `Slashdot0811`. For fairness, we set all the methods to give the same number of features as Scagnostics, and thus run Eagle-Fix. To evaluate the conciseness of our method, we present experiments in Section. 6.4.6. For all the methods, we evaluate the diversity and domain-specificity ('surprising') of the selected features $\mathbf{f}$ via correlation: We compute the pairwise feature correlation matrix between the F univariate distributions (with the same binning) of the baseline graphs, $\mathbf{C}_{\mathcal{G}}$, and quantify diversity as $\mathbf{f}^T\mathbf{C}_{\mathcal{G}}\mathbf{f}$. Similarly, based on the correlation matrix $\mathbf{C}'_{\mathbf{g}}$ between the input graph and the baseline features, we quantify domain-specificity as $\mathbf{f}^T\mathbf{C}'_{\mathbf{g}}$. For completeness, we apply three different correlation coefficients: Pearson's, Kendall's Tau, and Spearman's Rank. Figure 6.5 illustrates the results for Pearson's correlation (dark shades for diversity, light for domain specificity). Similar patterns are detected by using the other two metrics, which are omitted for brevity (they can be found in our code repository).

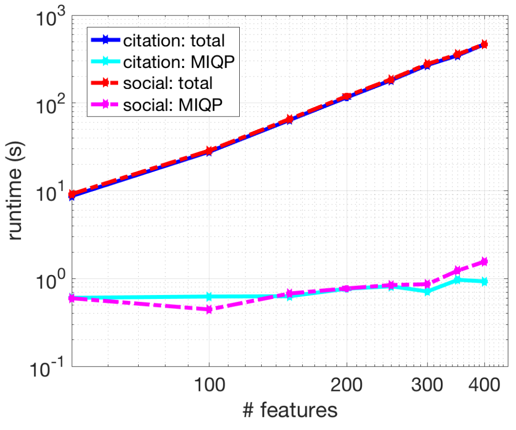
Diversity. Diversity is measured using pairwise feature correlation in $\mathbf{C}_{\mathcal{G}}$, so lower values indicate higher diversity. The results show that EAGLE outperforms all the baselines in every case. We observe an extreme case: the summary of `HepPh` conditioned on citation networks yields almost 0 Pearson correlation value. This demonstrates the effectiveness of EAGLE in selecting features that are diverse especially when the baseline graphs and the input are very similar.

Domain-Specificity. Similar to diversity, we explore 'surprising' patterns of the input graph with respect to the baselines via the feature-wise correlation (low values correspond to high domain-specificity). Figure 6.5 shows that EAGLE outperforms all the baselines by up to $\sim 51.74\%$. Qualitatively, the clustering coefficient distribution and community size distribution are always selected when the input graph and the baseline graphs are from different domains. Intuitively, this is reasonable because the community structure differs in graphs from different domains and both properties are related to it.

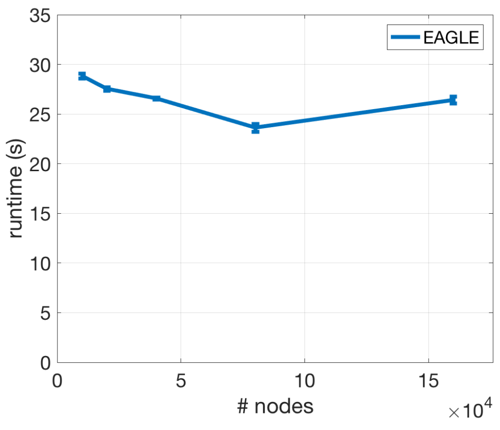
6.4.5 Scalability

We evaluate the scalability of the proposed methods with regard to (a) number of features, and (b) size of the baseline graphs. Here we extend the feature space beyond the original 28 by creating 'perturbed' features with up to 30% random noise.

Number of features. We create a mixed domain containing the citation graph (`HepTh`) and two social graphs (`Epinions` and `Slashdot0922`) as baselines, and run EAGLE-FLEX to summarize two input graphs: `HepPh` and `Slashdot0811`, with the number of features (original and perturbed) varying from 50 to 400. In Fig. 6.6a, we observe that, for both input graphs, EAGLE-FLEX scales linearly and almost identically with the number of features in the semi-logarithmic plot, which indicates its quadratic complexity. Moreover, given identical number of features, the runtime of MIQP on different input graphs is almost the same.

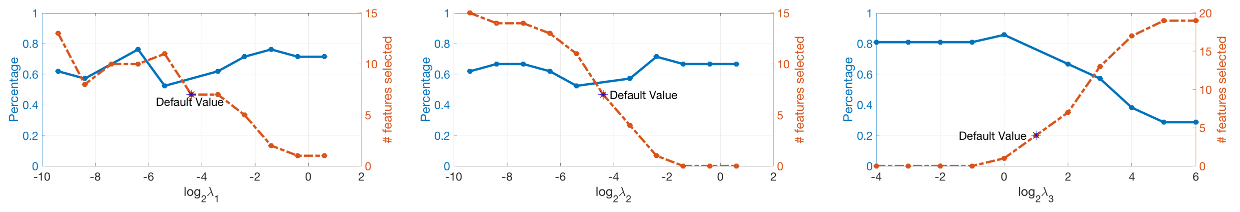


(a) Number of features



(b) Size of baseline graphs

Figure 6.6: Scalability of EAGLE-FLEX on two input graphs (citation and social). (a) In both cases, EAGLE-FLEX scales quadratically in terms of the number of features with similar behavior of MIQP (b) The runtime is independent of the size of baseline graphs.



(a) Sensitivity to λ_1. **(b) Sensitivity to λ_2.** **(c) Sensitivity to λ_3.**

Figure 6.7: Robustness of EAGLE to the regularization parameters. Left y axis: percentage of identical selected features between λ and its default value. Right y axis: total number of invariant distributions included in the summary.

Size of baseline graphs. In this experiment we test the scalability in terms of the size of baseline graphs for a fixed number of selected features. We create a series of synthetic datasets with feature space including 7 global invariant distributions and 13 perturbed invariants. The sizes of the synthetic graphs constructed are 10K, 20K, 40K, 80K, and 160K. The runtime of Eagle-Flex on these datasets is shown in Fig. 6.6b. We observe a relatively "flat" pattern in running time, which indicates that the optimization solver in EAGLE is independent of the size of baseline graphs. Note that there is some fluctuation in the curve: the running time of Eagle-Flex on 40K graphs is the shortest, while that on 10K is the longest. Despite the presence of randomness, this phenomenon points to one direction of our future work, which is to explore the behavior of MIQP in EAGLE on large-scale graphs.

6.4.6 Robustness to parameters

We run Eagle-Flex to evaluate the sensitivity of regularization parameters $\lambda_{\{1,2,3\}}$ and the corresponding conciseness of the summary. The baseline graphs are `HepTh`, `Slashdot0811` and `Brain-Voxel1`, and a total of 28 features are generated (no perturbation). Per regularizer, we perform a grid search over $\{\frac{1}{32}, \frac{1}{16}, \ldots, 16, 32\}$ times its default value (§ 6.4.3), while keeping the other regularizers at their default values. We plot the number of selected features in the summary and the percentage of common selected features between that summary and the 'default' summary (based solely on the default values). These quantities are illustrated as a function of the regularizer values in Fig. 6.7. We note that we do not depict the percentage for the default value (marked with '*') in the blue curve, since it is 100% by definition.

The blue curves in Figures (6.7a)-(b) show that values of regularization around the default give relatively stable results, with 50%∼80% identical features to the default setting.

Figure (6.7c) shows that the selection of features is stable up to the default value, but sensitive for larger λ_3 for which the last term dominates (and thus puts more emphasis on 'surprising' patterns). From the red curves, we observe that the default values lead to few selected features, indicating the conciseness (property P2) of the EAGLE summaries.

6.4.7 Case study: classification on brain graphs

How can EAGLE be applied to graph classification, a traditional data mining task? We focus on the domain of neuroscience, and use `COBRE` [cob12], a dataset from the NIH Center for Biomedical Research Excellence with resting-state fMRI data from 72 patients with schizophrenia and 76 healthy controls. From the 1166 fMRI time series (avg. length 100 timesteps), we created undirected, weighted graphs with $\theta = 0.6$ following the traditional method [BS09] (cf. § 6.4.2).

The task is to use the EAGLE summaries to classify the healthy controls and patients. We create the feature space by calculating the distributions of 11 features: weighted and unweighted degree, PageRank, closeness, eigenvector, clustering coefficient, betweenness, neighbors of the egonets, degree of the egonets, and sizes of egonets in edges and nodes. To obtain feature representations that can be used for classification, we used a random set of 36 healthy subjects as the baseline graphs, and ran both EAGLE-FIX (with $F = \{6, 8, 10\}$) and EAGLE-FLEX on the remaining graphs (40 controls and 40 patients) and obtained both 'surprising' and 'oridinary' summaries for them. We consider two vector representations for the summaries: (i) `Unweighted`: a binary vector $\mathbf{b}$ with 1s for the selected features by EAGLE; and (ii) `Weighted`: a real vector with the importance of each selected feature, i.e., $\mathbf{b} \odot \mathbf{h}$ where $\mathbf{h}$ is given in Eq. (6.5) (or its distance-based counterpart), and $\odot$ denotes component-wise multiplication. We also consider 'Full', which uses vector $\mathbf{h}$ as the representation of each connectome (without feature selection).

As baselines we considered two traditional methods in neuroscience: (a) per connectome, a vector representation with the mean of each feature distribution [DH14] and (b) a 'flat', vectorized ($1 \times N^2$) representation of the $N \times N$ adjacency matrix of the connectome [SKW+13]. For the classification task, we trained an SVM classifier that uses the RBF (radial basis function) kernel on the vector representations of our methods and the baselines, by conducting

10-fold cross validation. Table 6.3 gives the accuracy (AUC) of each method.

Table 6.3: Classification on `COBRE`: AUC scores per method.

Method Category	`Unweighted`		`Weighted`	
	Ordinary	Surprising	Ordinary	Surprising
Eagle-Flex	0.6893	0.5499	0.7096	**0.7296**
Eagle-Fix: 6	0.5114	0.5445	0.6961	**0.7371**
Eagle-Fix: 8	0.6795	0.5904	**0.7216**	0.7079
Eagle-Fix: 10	0.5003	0.4989	**0.7032**	0.6807
Full	-	-	0.6681	0.7147
Baselines	Baseline 1:	0.7028	Baseline 2:	0.1099

According to Table 6.3, we have two observations: (1) Without knowing anything about the dataset, Eagle-Flex provides promising performance on the task of classification, although Eagle-Fix outperforms Eagle-Flex with some explicit settings on F. The Eagle-Flex and Eagle-Fix summaries lead to better performance than the baseline methods, indicating the fact that although not designed explicitly for this, features selected by EAGLE can be applied to specific tasks such as classification; (2) Compared with the use of all weighted features (Full) and selection (Eagle-Flex), we observe that the latter improves the performance over the former by eliminating the noise contained in the dataset, which demonstrates the effectiveness of selected features. Qualitatively, among the 11 features, PageRank is the most frequently picked feature by Eagle-Flex. Weighted and unweighted degree are the most distinguishable features when running Eagle-Fix that are never picked in summaries for controls, but are selected for patients.

6.5 Conclusion

We proposed a novel way to summarize a graph using a set of interpretable features, resulting in a diverse, concise, domain-specific, and efficient-to-compute summary. Our novel formulation targets early data exploration and provides an alternative to the feature engineering process that is often a part of graph mining tasks. We framed the problem as constrained optimization, based on ‘conditional feature selection, which is tailored to the domain expectations and knowledge, in contrast to existing work which views each graph as a unit independent of its domain or many graph observations as a whole. We also introduced two efficient algorithms, Eagle-Fix and Eagle-Flex, which handle the correlations between graph features and find summaries that are fixed or flexible in size. Our experiments showed that the EAGLE variants are effective, their summaries satisfy all the desired properties, outperform alternative approaches that can be cast to solve this problem, and they are effective in data mining tasks such as classification despite not being tailored to it. Future work may explore extensions to more complex design choices or bivariate distributions of features (often used in spam detection), as well as scaling the method up more.

CHAPTER VII

Transfer Learning with Attention-based Summarization of Relational Data in Knowledge Integration

This chapter is based on work that will appear at PVLDB 2022 [JSW$^+$22].

7.1 Introduction

In Chapter VI, we introduced EAGLE to summarize the non-latent features for graph analysis, and show the effectiveness of the selected important features through graph exploration and multi-graph classification tasks. However, using a subset of features as the summary could incur information loss. Additionally, the features from which EAGLE selected are determined based on domain knowledge. Ideally, feature importance should be measured in a "soft" way as a scaling value, and it should be learned automatically. These two requirements motivate us to summarize the non-latent features with more advanced techniques, such as an attention mechanism. In this chapter, address the multi-source entity linkage problem which is important to knowledge integration in industrial settings.

Entity linkage (EL), also known as entity resolution, record linkage, entity matching, is a fundamental task in data mining, database, and knowledge integration with numerous applications, including deduplication, data cleaning, user stitching, and more. The key idea is to identify records across different data sources (*e.g.*, databases, websites, knowledge base, etc.) that represent the *same* real-world entity. For example, some music websites record the song "Hey Jude" by Paul McCartney with the name abbreviation (i.e., "P.M.") while others with the band name (i.e., "The Beatles"). As newly-generated data surge over time, accurately consolidating the same entities across semi-structured web sources becomes increasingly important, especially in areas such as knowledge base establishment [DN09, GM12] and personalization [JHRK19].

Methods for solving the entity linkage problem across data sources include rule reasoning [FJLM09, SME$^+$17], computation of similarity between attributes or schemas [BM03], and active learning [QPS17]. In particular, recent deep learning approaches that are based on heterogeneous schema matching or word matching [MLR$^+$18, NHH$^+$19, LLS$^+$20] have been widely studied. Their promising performance mainly comes from the sophisticated word-level operations such as RNN and Attention [MLR$^+$18, FHHS20] to represent token sequences under attributes as the summarization, or the usage of pretrained language models [KT19] to better learn the word semantics. However, all the above learning approaches implicitly assume that the "matching/non-matching" info for training records is available (*e.g.*, the music records in source 1 and source 2 shown in the two blue tables of Fig. 7.1) and can be queried through the learning process, which does not always hold in practice. In real-world knowledge integration scenarios, new data come incrementally, and it can be either well-labeled (*e.g.*, through manual confirmation) or unlabeled. While the existing frameworks can handle high-quality labeled data, they cannot deal with the massive volume of unlabeled and previously unseen data or missing values. As the example shown in Fig. 7.1, a model trained on the high-quality labeled data (blue tables) would fail to generalize to the new data sources (red tables) with missing and different attribute values (*i.e.*, "Artist"), as well as new attributes or attributes that are rarely seen (*i.e.*, "Gender").

Motivated by real-world knowledge integration settings, we consider three key challenges in the multi-data source scenario: (**C1**) missing attribute values from unseen data sources; (**C2**) new attributes from unseen data sources; and (**C3**) different value distribution in unseen data sources. Based on these challenges, we seek to tackle the following **MEL (multi-source entity linkage) problem**: Given labeled data from a limited set of sources, *what* knowledge can be learned and *how* can it be transferred to automatically handle multiple unseen data sources with different value distribution, missing values and new attributes?

To solve this task, human experts typically rely on prior domain knowledge to learn the

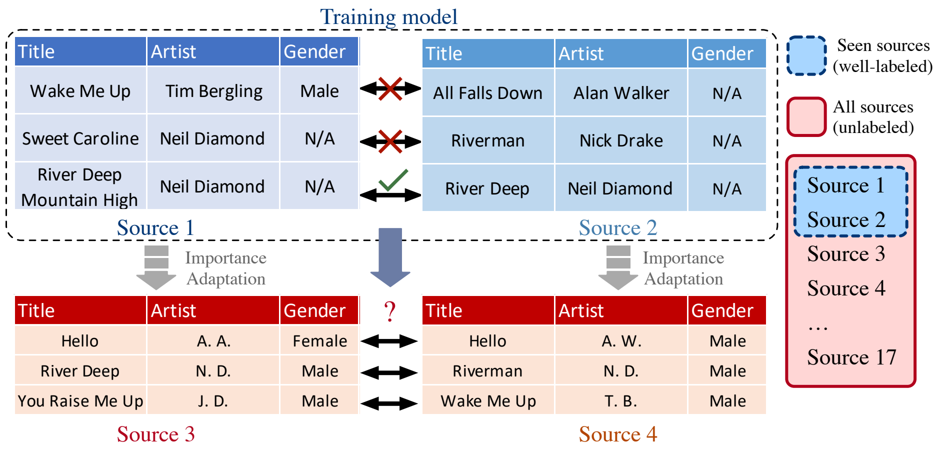


Figure 7.1: Well-labeled data sources (*e.g.*, blue tables) are generally outnumbered by massive unlabeled data in real-world knowledge integration scenarios. Entity linkage models trained only on well-labeled samples fail to handle new sources with different contexts or formats (*e.g.*, red tables). Our proposed framework, AdaMEL automatically learns the attribute importance that adapts to the massive unlabeled data from different sources during training, and then uses it as the transferable knowledge to perform matching.

attribute importance in the seen data sources, and then *transfer* it to match the unseen records based on the similarity of attribute values. As challenges (**C1-C3**) lead to different attribute importance, human experts would update their knowledge learned from the seen data sources and adapt to the unseen data sources. For example in Figure 7.1, when trying to link entities in the red table, the importance of "Artist" learned in the seen data sources (blue table) is down-weighted due to the fact that name abbreviation is less informative. On the other hand, even though it is a rarely-seen attribute in the seen sources, "Gender" would be more important because the gender difference between artists naturally leads to non-matching of music records regardless of the fact that entity pairs could share the same "Title" ("Hello") and very similar "Artist" values ("A. A." and "A. W."). This process, however, is tedious and does not scale to massive unlabeled data involved in real-world entity linkage problems, where large volumes of new data sources continuously arrive.

Following this intuition, we propose AdaMEL, a transfer learning framework that leverages both the labeled and massive unlabeled data to train the model for multi-source entity linkage while addressing the aforementioned challenges (**C1-C3**). We define the attribute importance in entity linkage as the high-level transferable knowledge and automatically learn it through a proposed attribute-level attention mechanism (*what* to transfer). In general, as transfer learning aims to transfers knowledge learned from the domain with abundant training data to a related target domain with limited data, the existing works either rely on increasing the labeling volume by introducing the external data (*e.g.*, public knowledge bases) [ZH19] or reusing the seen training data [TPO+18]. On the contrary, AdaMEL adopts domain adaptation (DA) to jointly update the attention scores for attributes in both the seen and unseen data as the basis for entity linkage (*how* to handle multiple sources), so that the knowledge is adaptive to the continuously incoming data sources. In addition, the insightful feature importance as transferrable knowledge is explicitly defined by AdaMEL to benefit both human interpretation and the performance learning tasks, which is also different from methods that incorporates the knowledge into pretrained models like "black-boxes", such as the contextual word/character embeddings. While the widely-adopted NLP-based attribute summarization in existing works [LLS+20, NHH+19, FHHS20] could accurately capture the word-level semantics using some pretrained language models or domain knowledge for all attributes, they are too computationally expensive for the practical scenario. On the contrary, the feature-level attention is much faster to obtain and we claim that the impact of word-level similarity under some attributes is limited and even harmful for model performance if those attributes are not important.

AdaMEL follows the real-world scenario and assumes that new data sources come from the same or neighboring domains in batches (*e.g.*, music from different websites). Transferring knowledge between irrelevant domains (*e.g.*, celebrities and products) does not produce meaningful outputs and is out of the scope of this paper. We also propose a series of AdaMEL variants for different learning scenarios in practice, namely, they are: (i) AdaMEL-zero that is based on unsupervised DA to handle the general case with massive unlabeled new data, (ii) AdaMEL-few that is based on semi-supervised DA to make use of an additional set of labeled samples from the new data when it is available, and (iii) AdaMEL-hyb that leverages both the sampled labeled and massive unlabeled data. Our contributions are summarized as follows.

- We formulate the problem of MEL in real-world knowledge integration where the incoming data of unseen data sources are associated with missing values, unseen attributes and different value distributions.

- We propose a deep transfer-learning framework that learns the attribute-level importance as the high-level knowledge, and incorporates massive unlabeled data across multiple unseen sources through domain adaptation to make it agnostic and transferable.
- We apply AdaMEL to multi-source entity linkage over both industrial and public datasets, and show that it achieves at least 5.92% improvement in terms of mean average precision compared to the state-of-the-art deep learning EL methods.

7.2 Related Work

Entity Linkage (EL). Entity linkage has been and continues being a fundamental problem in the field of database, data mining and knowledge integration [GM12, KR10, DH05]. Early works are based on the similarity between entity attributes [FJLM09, DH05] through resolving the data conflicts [DN09], linking relevant attributes through semantic matching or rule reasoning [SME^{+}17]. Techniques such as blocking or hashing are normally applied to merge the candidate entities [CR02]. The major drawback of these methods is the dependence on prior knowledge as the useful attributes are normally selected through human efforts. Recently, EL models based on deep neural networks [MLR^{+}18, JT18] have been widely studied due to their capability in automatically deriving latent features and promising results in fields such as CV and NLP [Ben12, LPM15, FAL17]. For example, DeepER [JT18] proposes to leverage RNN to compose the pre-trained word embeddings of tokens within all attribute values, and use them as features to conduct EL as the binary classification task. DeepMatcher [MLR^{+}18] also takes the embeddings of attribute words as the input and uses RNN to obtain the attribute similarity representation.CorDel [WSW^{+}20] proposes to compare and contrast the pairwise input records before getting the embeddings so that small but critical differences between attributes can be modeled effectively. There are also recent works that formulate entity linkage across different data sources as heterogeneous entity matching [LLS^{+}20, FHHS20, NHH^{+}19], for example, EntityMatcher [FHHS20] proposes a hierarchical matching network that jointly match entities in the token, attribute, and entity level. Ditto [LLS^{+}20] proposes to leverage the pretrained language model [JT18, KT19, LLS^{+}20] such as BERT or DistilBERT, as well as domain knowledge and data augmentation to improve the matching quality. The attention mechanisms [LPM15, VSP$^+$17, VCC$^+$18] are generally adopted by these deep models, where the goal is to improve the linkage performance by highlighting valuable embeddings, *e.g.*, word embeddings within the attributes. The basis of these above deep models for heterogeneous schema matching is to accurately summarize the attribute words through advanced NLP techniques such as word token-level RNN (with attention) or pretrained language models. On the contrary, our proposed method does not require sophisticated computation to summarize words in each attribute. AdaMEL focuses on the impact of important attributes in matching and explicitly models their importance using the soft attention mechanism as the transferable knowledge. Such attribute-level importance is agnostic to specific data sources and generalizes better than individual words in the transfer learning paradigm.

Transfer Learning. In the transfer learning scenario, models are trained on a source domain and applied to a related target domain to handle the same or a different task [PY09, GBC16]. The specific transferable knowledge that bridges the source and target domain has significant impact to model performance [YZHY18]. A popular approach is to adapt the pre-trained model for the new task through fine-tuning [KT19], or by adding new functions to specific tasks such as object detection [HGDG17]. In terms of EL, TLER [TPO$^+$18] is a non-deep method that reuses and adopts seen data from the source domain to train models for the new domain. Auto-EM [ZH19] proposes to pre-train models for both attribute-type (*i.e.*, schema) and attribute value matching based on word- and character-level similarity. However, Auto-EM assumes the typed entities are from a single data source and the attributes are seen during training, and thus cannot handle the multi-source scenario with unseen attributes. A specific type of transductive transfer learning that is most relevant to our work is known as *Domain Adaptation*, where the source and target domain share the same feature space with different distributions [SSW15], and models are trained on the same task [WC20]. Many well-designed algorithms propose to map the original feature spaces to a shared latent feature space between domains [DXT12, CSF$^+$12]. DeepMatcher+ [KQG$^+$19] extends DeepMatcher with the combination of transfer learning and active learning to achieve comparable performance with fewer samples. However, this work aims at dataset adaptation rather than the attribute matching, and the focus is not improving the matching performance. Another line of works proposes to pre-train models on the source and target domain (if labeling available) and then

Table 7.1: Summary of notation.

Symbol	Definition
$\mathcal{A} = \{A_j\}$	a set of pre-defined textual attributes (data source schema)
$r, r[A]$	an entity record and the value (word tokens) of attribute A
$\mathcal{D}_S, \mathcal{D}_T$	source and target domain, respectively
$(r, r')_{S/T}$	an entity pair in the source and target domain, respectively
S, S'	set of data sources in general
r^*	the data source that record r is sampled from
$\mathcal{D}^*$	set of data sources in a domain, *e.g.*, $\mathcal{D}^*_S = \{r^*\}_{r \in \mathcal{D}_S}$
F	the number of relational features, $F = 2\|\mathcal{A}\|$
$\mathbf{x}, y$	H-dim latent feature vector of an entity pair and its label
$\mathbf{h}_j$	D-dim token embedding of feature j
f	attention embedding function $\mathbb{R}^{D \times F} \to \mathbb{R}^F$

combine them through specific weighting schemes [SRWS08]. The process of applying the trained model to handle previously unseen data is also known as zero-shot learning [PPHM09]. Unlike the above approaches, AdaMEL explicitly learns feature importance by adapting to the massive unlabeled data from unseen sources as the transferable knowledge for the multi-source EL task.

7.3 Preliminaries

In this section, we first formally define the problem, and then provide several key notions relevant to our proposed solution. Symbols and notations used specifically in this chapter are listed in Table 7.1.

7.3.1 Problem Definition

An entity record is collected from a specific data source such as a website or a database, and is identified by its attributes. For example, a song record $r =$ ("Sweet Caroline", "Neil Diamond", "USA") is specified by the attributes $\mathcal{A} = \{\texttt{title, artist, country}\}$. We start with the formal definition of entity linkage.

Problem 3 (EL: Entity Linkage)**.** *Given two entity records r and r' associated with the same set of attributes $\mathcal{A}$ (schema), entity linkage aims to predict if r and r' refers to the same real-world entity.*

In this chapter, we conduct analysis based on entity pairs (r, r') instead of individual entity records. We now define the MEL problem, which is related to heterogeneous entity matching[1] [NHH$^+$19, FHHS20].

Problem 4 (MEL: Multi-source Entity Linkage)**.** *Given the labeled entity pairs* $\{(r, r')\}_{seen}$ *from a limited set of data sources* $\mathcal{S}$ *where each entity record* r *is associated with attributes* $\mathcal{A}$*, and previously unseen pairs* $\{(r, r')\}_{unseen}$ *from the new data sources* $\mathcal{S}'$ *with attributes* $\mathcal{A}'$*, multi-source entity linkage aims to predict if each pair in* $\{(r, r')\}_{unseen}$ *represents the same real-world entity, where* $(r, r')_{unseen} \in (\mathcal{S} \times \mathcal{S}') \cup (\mathcal{S}' \times \mathcal{S}')$*,* $|\mathcal{S}'| > |\mathcal{S}|$*. Since* $\mathcal{S} \neq \mathcal{S}'$*, certain attributes in* $\mathcal{A}'$ *could be missing (**C1**), new (**C2**), or associated with values from different distributions (**C3**), and thus* $\mathcal{A} \neq \mathcal{A}'$*.*

The key notion in Problem 4 that is different from Problem 3 is that the linkage task is conducted on entity pairs sampled from a wider range of data sources than the labeled data used to train the model (ten or hundred orders of magnitude more in reality). Back to the example shown in Figure 7.1, while the trained model could make perfect prediction based on "Artist" only, it would fail to handle new records because the attribute "Artist" has missing or abbreviated values that contain less info. Moreover, the new data sources contain a rarely seen or unseen attribute ("Gender"). This issue can be addressed by aligning the union of ontology $\mathcal{A} \cup \mathcal{A}'$ with blank "dummy" attributes. Based on our definition, a solution to MEL should be able to (**G1**) make use of the massive unlabeled data from the new sources, and (**G2**) further improve the linkage performance by leveraging a few labeled record pairs from the new sources, if available (*i.e.*, an additional support set).

7.3.2 Terminology

Here we discuss the necessary terminology of our framework,.

Definition 20 (Source & target domain)**.** *The source domain* $\mathcal{D}_S$ *refers to a set of labeled entity pairs* $\{(r, r')\}$ *sampled from limited data sources that the model is trained on. The target domain* $\mathcal{D}_T$ *refers to the set of unlabeled pairs where each pair has at least one entity sampled from the data sources unseen in* $\mathcal{D}_S$*.*

[1]In MEL, the entities come from different data sources, and thus there may be new or missing attributes. On the other hand, in heterogeneous entity matching, the schemas are heterogeneous (i.e., they have different attributes, which may not be aligned) and the entities do not necessarily come from different data sources.

For clarity, we use the superscript * to indicate the data source(s) of a record/domain. Following Definition 20, the seen and unseen set of data sources in Problem 4 are formulated as $\mathcal{S} = \mathcal{D}_S^*$ and $\mathcal{S}' = \mathcal{D}_T^*$. Besides, given a pair in the target domain, it could either contain one entity sampled from the seen data sources in $\mathcal{D}_S^*$ and the other one from the unseen, *i.e.*, $(r, r')_T \in \mathcal{D}_S^* \bigtimes \mathcal{D}_T^*$, or it has both entities sampled from the completely unseen data sources, *i.e.*, $(r, r')_T \in \mathcal{D}_T^* \bigtimes \mathcal{D}_T^*$. In both cases, achieving **G1** requires data in $\mathcal{D}_T$. To achieve **G2**, we introduce the support set.

Definition 21 (Support set)**.** *The support set $\mathcal{S}_U$ refers to a small set of labeled entity pairs sampled from the same set of data sources as the target domain $\mathcal{D}_T^*$. It has at least one data source that is not contained in $\mathcal{D}_S^*$.*

The support set corresponds to the real-world scenario that a few newly incoming entity pairs are well-labeled (*e.g.*, on-the-fly human annotation). Thus, entity pairs in $\mathcal{D}_S$, $\mathcal{D}_T$, as well as $\mathcal{S}_U$ are all required to achieve **G2** for MEL.

7.4 Proposed framework

We propose AdaMEL to address Problem 4, a deep framework that learns attribute importance as the transferable knowledge $\mathcal{K}$ (Section 7.4.1), and adapt it to multiple data-sources via domain adaptation . AdaMEL first extracts the contrastive relational features of entity pairs to derive the embeddings (Section 7.4.2). Then, by using the proposed attention embedding function f, AdaMEL projects features from $\mathcal{D}_S$ and $\mathcal{D}_T$ into the same attention space (Section 7.4.3), and jointly learns the feature importance for data sources in both $\mathcal{D}_S^*$ and $\mathcal{D}_T^*$. This process is conducted in an unsupervised or supervised domain adaptation manner (Section7.4.4), depending on the real-world scenario. The overview is depicted in Figure 7.2.

7.4.1 Formulation

In transfer learning, the generic transferable knowledge $\mathcal{K}$ is key to adapt the model trained on the source domain to the target domain. We denote our domain adaptation

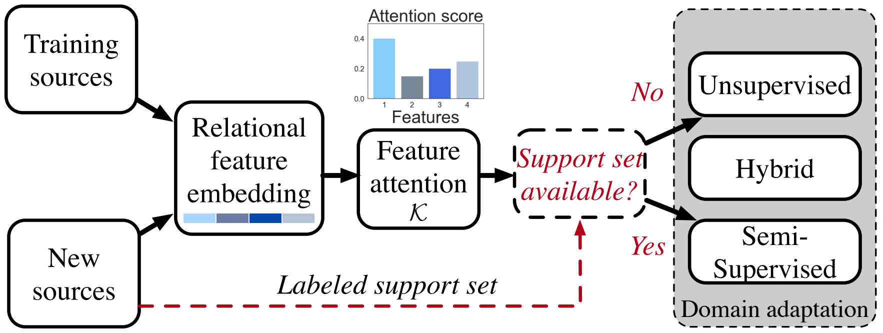


Figure 7.2: Overview. AdaMEL first embeds attributes for records from both the source and target domain to derive the feature representations, and uses the feature attention function to get the attention scores (importance) as the transferable knowledge $\mathcal{K}$. Then, depending on the availability of the labeled support set, AdaMEL uses $\mathcal{K}$ and performs either the unsupervised or semi-supervised manner of domain adaptation for MEL.

solution to MEL as the following binary classification task.

$$y = M(\mathcal{K}, (r, r')) \in \{0, 1\} \tag{7.1}$$

where M represents the deep model that generates the binary prediction y for the entity pair $(r, r') \in \mathcal{D}_T$, where 1 and 0 indicate the matching and non-matching, respectively. As mentioned in Section 7.3.1, the key difference between $\mathcal{D}_S$ and $\mathcal{D}_T$ lies in the difference in data sources, therefore $\mathcal{K}$ should be data-source agnostic to address **(C1)-(C3)**. To ensure $\mathcal{D}_T$ shares the same feature space as $\mathcal{D}_S$ (the prerequisite for domain adaptation), AdaMEL first aligns the ontology so that data sources $\mathcal{D}_S^*$ and $\mathcal{D}_T^*$ share the same attribute schema, but the attribute values (word tokens) can vary significantly. By doing so, entity records reveal the following properties that correspond to the aforementioned challenges: (**C1**) entity records in the source/target domain contain missing values, *i.e.*, $r[A] =$"" (empty string) for $r \in \mathcal{D}_S \cup \mathcal{D}_T$, (**C2**) certain attribute values are completely missing for records in $\mathcal{D}_S$, *i.e.*, $r[A_j] =$"" for $r \in \mathcal{D}_S$, but not in $\mathcal{D}_T$, and (**C3**) rich texts under some attributes in $\mathcal{D}_S$ but sparse in $\mathcal{D}_T$ or vice versa.

7.4.2 Feature Representation

Given entity pairs (r, r') with the aligned attributes $\mathcal{A}$, AdaMEL leverages the attention mechanism to learn the importance of each textual attribute $A \in \mathcal{A}$ as the generic knowledge for transfer learning. However, instead of computing the attribute importance directly, AdaMEL parses each attribute A into 2 contrastive relational features, which are word tokens shared by r and r', and word tokens that only appear in one record but not the other. This is because the similarity or uniqueness of attribute between r and r' gives independent and complementary evidence for linkage [WSW+20]. Taking the attribute $A =$"music version" as an example, a pair of music recordings sharing the same word (*i.e.*, "original" or "remix") is not as strong an identifier for matching as it would be for non-matching if one recording is "original" while the other is "remix". In addition, looking into both the similarity and uniqueness in attribute A between entities would enrich the feature space and facilitate training the deep model. We describe the 2 contrastive relational features of an attribute A as follows.

$$\begin{cases} sim(A) & = \{w\} \text{ for } w \in \{r[A] \cap r'[A]\} \\ uni(A) & = \{w\} \text{ for } w \in \{r[A] \cup r'[A] - r[A] \cap r'[A]\} \end{cases} \tag{7.2}$$

where w is the word token in attribute $r[A]$. For clarity, we uniformly denote shared/unique tokens $sim(A)/uni(A)$ as "features" that contribute independently to entity linkage. Clearly, there are $F = 2|\mathcal{A}|$ features for a pair of entities. To summarize the feature representation, AdaMEL sums up the embeddings of the cropped word tokens [JGBM17, VSP+17, MLR+18] without using more sophisticated operations. The embeddings can be obtained using any pretraining language model, such as BERT [KT19] or Fasttext [JGBM17]. For clarity, we use i as the index of entity pairs and j as the index of features. Thus, the token embedding vector of an entity pair (r, r') is denoted as:

$$\begin{aligned} \mathbf{h} &= [\mathbf{h}_1, \mathbf{h}_2, \cdots, \mathbf{h}_F] \\ &= [\text{emb}(sim(A_j)), \text{emb}(uni(A_j))] \text{ for } j = 1, \cdots, |\mathcal{A}| \end{aligned} \tag{7.3}$$

By doing so, we denote the entity pairs (r, r') as F textual embedding features ($F = 2|\mathcal{A}|$) for matching. The complete process is depicted in Figure 7.3. Besides, AdaMEL leverages per-feature non-linear affine transformation to project the word embeddings to get the latent feature $\mathbf{x}$:

$$\mathbf{x} = [\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_F] = [\sigma(\mathbf{V}_j \mathbf{h}_j + \mathbf{b}_j)] \text{ for } j = 1, \cdots, F \tag{7.4}$$

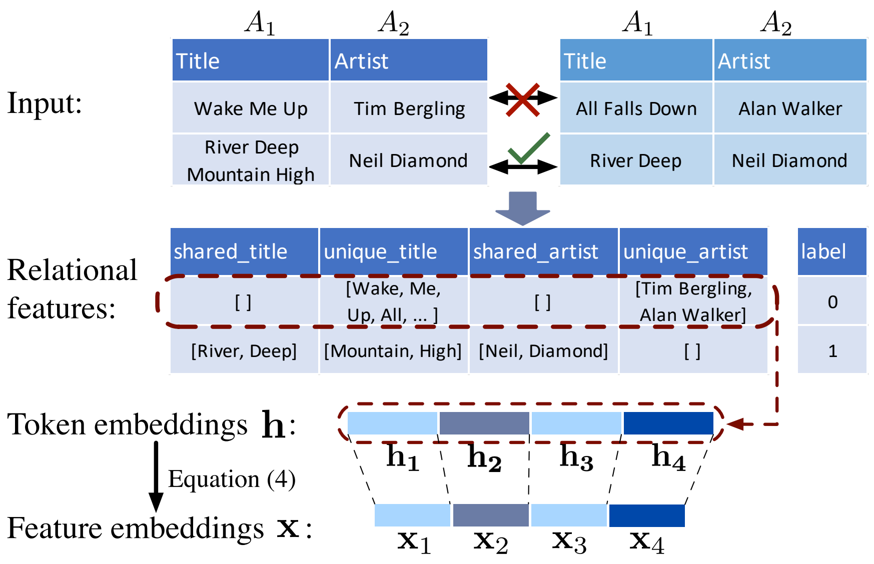


Figure 7.3: AdaMEL processes 1 attribute A as 2 relational features (*i.e.*, $sim(A)$ and $uni(A)$). In this example, $F = 4$ features are generated from $|\mathcal{A}| = 2$ attributes (*i.e.*, "Title" and "Artist"). The empty word tokens are embedded as the fixed normalized non-zero vector to form h (red dashed box). The feature embedding x is obtained through non-linear affine transformation of the token embedding h (Equation (7.4)). Each feature assumes to contribute independently to predict the linkage.

where $\mathbf{V}_j^{H\times D}$ is the learnable weight matrix, $\mathbf{b}_j^H$ is the learnable bias vector, and σ denotes the non-linear activation function (*e.g.*, Relu). With this representation, Equation (7.1) can be rewritten as: $y = M(\mathcal{K}, \mathbf{x}) \in \{0, 1\}$. Next we discuss how AdaMEL learns feature importance [2] as the transferable knowledge $\mathcal{K}$.

7.4.3 Feature Attention Embedding

Given a pair of entities denoted through F features, AdaMEL defines the energy score of feature j as $e_j = a(\mathbf{W}\mathbf{x}_j)$, where $\mathbf{x}_j$ is the H-dimensional representation of latent feature j, $\mathbf{W}^{H'\times H}$ is a shared linear transformation, and a represents the attention mechanism $\mathbb{R}^{H'} \to \mathbb{R}$, as a single-layer neural network (parameterized with $\mathbf{a}$). AdaMEL allows each feature to attend to the label y independently and computes coefficients using the softmax function such that the normalized scores are comparable across all features. Formula in Equation (7.5) computes the attention score of feature j:

$$g(\mathbf{x}_j) = \text{softmax}_j(e_j) = \frac{\exp\left(\mathbf{a}^T \tanh\left(\mathbf{W}\mathbf{x}_j\right)\right)}{\sum_{k=1}^{F} \exp\left(\mathbf{a}^T \tanh\left(\mathbf{W}\mathbf{x}_k\right)\right)} \tag{7.5}$$

[2]In this work, we compute the feature attention as the transferable knowledge, feature importance.

Note that Equation (7.5) only generates the scalar attention score of feature j for an input vector $\mathbf{x}$. To compute the scores of all features, we introduce the attention embedding function f that learns attention scores of all F features as follows.

$$f(\mathbf{x}) = f([\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_F]) = [g(\mathbf{x}_1), g(\mathbf{x}_2), \cdots, g(\mathbf{x}_F)] \tag{7.6}$$

In Equation (7.6), all features share the same $\mathbf{W}$ and $\mathbf{a}$ to compute the attention scores. We denote $f(\mathbf{x})_j = g(\mathbf{x}_j)$, and $\sum_{j=1}^{F} f(\mathbf{x})_j = 1$. AdaMEL takes the generated feature importance vector $f(\mathbf{x})$ as the transferable knowledge $\mathcal{K}$ for the entity pair (r, r'), *i.e.*, $\mathcal{K} = f(\mathbf{x})$.

In the learning process, AdaMEL feeds the feature representation coupled with its attention score to a 2-layer feed-forward neural network Θ to perform the binary classification task:

$$\hat{y} = \Theta(\sigma(f(\mathbf{x}) \odot \mathbf{x})) = \Theta([\sigma(g(\mathbf{x}_1) \cdot \mathbf{x}_1), \cdots, \sigma(g(\mathbf{x}_F) \cdot \mathbf{x}_F)]) \tag{7.7}$$

where $\odot$ denotes the element-wise multiplication, σ denotes the non-linear activation (*e.g.*, Relu) and $\hat{y}$ denotes the inference score for matching. AdaMEL uses the same attention mechanism to handle all records in the training and leverages the cross-entropy loss to update the shared parameters $\mathbf{W}$, $\mathbf{a}$, as well as the learnable $\mathbf{V}$, $\mathbf{b}$ through back-propagation:

$$L_{base} = -\frac{1}{N} \sum_{i=1}^{N} y_i \log \hat{y}_i + (1 - y_i) \log(1 - \hat{y}_i) \tag{7.8}$$

where y_i denotes the label $\{0, 1\}$. To ensure that all learnable parameters can be updated correctly, AdaMEL initializes the missing attribute values (incurred by challenge **C1, C2**) with a fixed normalized non-zero vector.

We name this solution **AdaMEL-base** as it learns f through the labeled data in $\mathcal{D}_S$ and illustrate the architecture in Figure 7.4. The attribute importance learned under the supervision of labeled data in $\mathcal{D}_S$ will be carried over to the unseen data sources and may not generalize well as there is always new data from seen or unseen sources with different distributions (**C3**) in MEL. Next we discuss how AdaMEL adopts $\mathcal{D}_T$ sampled from multiple data sources to alleviate this issue and make $\mathcal{K}$ data-source agnostic.

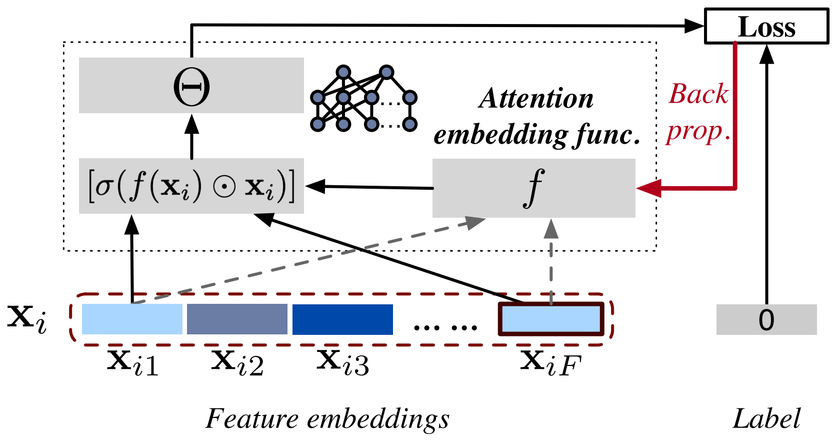


Figure 7.4: AdaMEL-base architecture that updates f via labeled data in $\mathcal{D}_S$. AdaMEL-base first computes the attention vector $f(\mathbf{x}_i)$ for the $i-$th entity pair (dashed line), and then compose it with the feature embeddings (solid line) as the input to the neural network Θ.

7.4.4 Domain Adaptation-based Variants

Based on AdaMEL-base, we propose three variants that leverage domain adaptation to handle different learning scenarios.

7.4.4.1 Unsupervised Domain Adaptation

Our first idea is to adjust the learned attribute importance according to new distribution of unlabeled data. In Equation (7.6), the attention embedding function f contains the shared attention mechanism a parameterized by weight vector $\mathbf{a}$ and the shared transformation matrix $\mathbf{W}$. It only takes the feature embeddings $\mathbf{x}$ as input to compute the attention scores. Since $\mathbf{W}$ and $\mathbf{a}$ are shared across the input data, the attention score vector $f(\mathbf{x})$ can be seen as projecting the input feature embeddings $\mathbf{x}$ into a hyper-plane that is parameterized by $\mathbf{W}$ and $\mathbf{a}$. Without introducing extra information such as entity pair labeling, we can project data from $\mathcal{D}_T$ into the same space as $\mathcal{D}_S$, and it holds as long as the ontology of the unlabeled data aligns with the labeled data, *i.e.*, identical attribute schema between $\mathcal{D}_S$ and $\mathcal{D}_T$.

Therefore, AdaMEL uses the KL divergence to measure the attention score distribution difference between the source and target domain as the regularization term to train the model. The loss is defined in Equation (7.9). At a specific iteration in the training, AdaMEL uses the up-to-date f to project data from both $\mathcal{D}_S$ and $\mathcal{D}_T$ into the same feature attention space. Then, AdaMEL updates $\mathbf{W}$ and $\mathbf{a}$ so that not only the cross-entropy loss introduced in Equation (7.8) is minimized, but also the KL divergence between feature attention distributions for $\mathcal{D}_S$ and $\mathcal{D}_T$. In this way, feature importance for entity records in $\mathcal{D}_S$ is jointly updated with records sampled from a wider range of data sources in $\mathcal{D}_T$, and thus being agnostic to previously unseen data sources that have significantly different value distribution (**C3**).

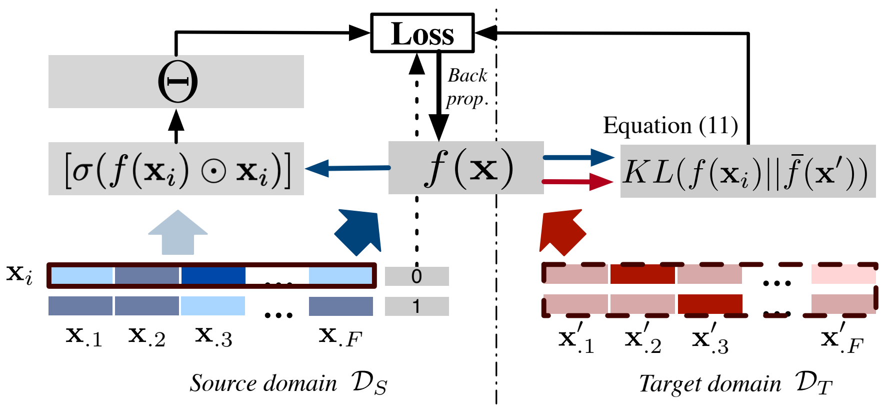
$$L_{\text{un}} = (1-\lambda)L_{\text{base}} + \lambda L_{\text{target}} \tag{7.9}$$

where λ is the hyperparameter that balances between L_{base} and L_{target}. λ also measures the amount of adaptation to the target domain $\mathcal{D}_T$. L_{target} is given as follows.

$$L_{\text{target}} = \text{KL}(f(\mathbf{x}), \bar{f}(\mathbf{x}')) = \sum_{i=1}^{|\mathcal{D}_S|}\sum_{j=1}^{F} \bar{f}(\mathbf{x}')_j \log(\frac{\bar{f}(\mathbf{x}')_j}{f(\mathbf{x}_i)_j}) \tag{7.10}$$

where $\bar{f}(\mathbf{x}')_j = \frac{1}{|\mathcal{D}_T|}\sum_{\mathbf{x}'_i \in \mathcal{D}_T} f(\mathbf{x}'_i)_j$, which represents the attention score for feature j averaged over the unlabeled data. $\mathbf{x}$ and $\mathbf{x}'$ denote the feature vector in the source and target domain, respectively, and $f(\mathbf{x}_i)_j$ denotes the importance of the j-th feature in the i-th entity pair. In practice, AdaMEL adopts batch learning to improve the training efficiency, *i.e.*, minimizing the loss per batch instead of iterating through all records in the data. The unlabeled data could also come in batches, which makes $\bar{f}(\mathbf{x}')$ be the attention vector averaged over the batched unlabeled data instead of all in the target domain. By default, the batches are sampled randomly.

We name this solution **AdaMEL-zero** as it is based on unsupervised domain adaption without using any labeled data in $\mathcal{D}_T$ and performs linkage in the zero-shot manner. This model also follows the design pattern in [GL15]. Figure 7.5 depicts the architecture and the algorithm is given in Algorithm VII.1. Line 3-4 project the affine transformation of entity pairs from both $\mathcal{D}_S$ and $\mathcal{D}_T$. Line 5 computes $\bar{f}(\mathbf{x}')$, the attention vector averaged over entity pairs in the target domain. Line 8-10 computes each attention vector in the sampled batch $f(\mathbf{x}_i)$ and adapt it to $\bar{f}(\mathbf{x}')$ to compute the loss L_{target}. AdaMEL minimizes both the inference loss L_{base} and L_{target} to train the parameters in f and form the transferable knowledge $\mathcal{K} = f(\mathbf{x}_i)$ for $\mathbf{x}_i \in \mathcal{D}_T$ (Line 12). Line 14- 15 denote the inference.



Loss
Back prop.
Θ
$[\sigma(f(\mathbf{x}_i) \odot \mathbf{x}_i)]$
$f(\mathbf{x})$
Equation (11)
$KL(f(\mathbf{x}_i)||\bar{f}(\mathbf{x}'))$
$\mathbf{x}_i$
0
1
$\mathbf{x}_{.1}$ $\mathbf{x}_{.2}$ $\mathbf{x}_{.3}$ $\mathbf{x}_{.F}$
$\mathbf{x}'_{.1}$ $\mathbf{x}'_{.2}$ $\mathbf{x}'_{.3}$ $\mathbf{x}'_{.F}$
Source domain $\mathcal{D}_S$
Target domain $\mathcal{D}_T$

Figure 7.5: AdaMEL-zero architecture that attempts to align the i-th entity pair $f(\mathbf{x}_i)$ in $\mathcal{D}_S$ (solid box) with the averaged $f(\mathbf{x}')$ (dashed box) in $\mathcal{D}_T$. $\mathbf{x}_{.j}$ and $\mathbf{x}'_{.j}$ ($j = 1, \cdots, F$) denote the j-th feature in general from $\mathcal{D}_S$ and $\mathcal{D}_T$, respectively.

7.4.4.2 Semi-supervised Domain Adaptation

In practice, a small number of labels may be available for the entity pairs coming from the target domain (*e.g.*, through on-the-fly human annotation). Entity pairs in this support set $\mathcal{S}_U$ are sampled from the wide range of data sources and provide clues about the data characteristics of the target domain. To leverage this set of labeled data (**G2**), AdaMEL updates the attention embedding function f under the supervision of $\mathcal{S}_U$ so that the projected feature attention vectors of entity pairs in $\mathcal{D}_S$ could match to those in $\mathcal{S}_U$. For this purpose, AdaMEL computes the centroid of the positive entity pairs in $\mathcal{D}_S$ as follows:

$$\mathbf{c}^+_{\mathcal{D}_S} = \frac{1}{|\mathcal{D}_S|} \sum_{(\mathbf{x}_i^+, y_i^+) \in \mathcal{D}_S} f(\mathbf{x}_i) \tag{7.11}$$

The centroid of the negative pairs can be computed in a similar way with negative samples. Intuitively, entity pairs from the data sources unseen in $\mathcal{D}^*_S$ are more important in adaptation than those from the seen sources, and should be highlighted. AdaMEL measures such difference through the Euclidean-distance between $f(\mathbf{x})$ and $\mathbf{c}_{\mathcal{D}_S}$, as the deviating attention vectors are more likely to correspond to unseen data sources in the projected space. In the loss function shown in Equation (7.12), we compare the distance $d(f(\mathbf{x}), \mathbf{c}_{\mathcal{D}_S})$ with the "mean distance to cluster centroids" to give higher weights to entity pairs in $\mathcal{S}_U$ that are deviating

Algorithm VII.1 ADAMEL-ZERO

```
Ensure: $\mathcal{D}_S = \{(\mathbf{h}_i, y_i)\}$, $\mathcal{D}_T = \{\mathbf{h}_i\}$, $\lambda$, batch size $B$
Require: Predicted $\hat{y_i}$ for $\mathbf{h}_i \in \mathcal{D}_T$, updated $\mathbf{a}, \mathbf{W}$
1  Initialize $\mathbf{a}, \mathbf{W}$ and $\mathbf{V}, \mathbf{b}$
2  loop training epochs
3      for $\mathbf{h} \in \mathcal{D}_S \cup \mathcal{D}_T$ do
4          Form $\mathbf{x}$ with $\mathbf{V}, \mathbf{b}$                              ▷ Eq. (7.4)
5      $\bar{f}(\mathbf{x}') \leftarrow \frac{1}{|\mathcal{D}_T|} \sum_{\mathbf{x_i} \in \mathcal{D}_T} f(\mathbf{x_i})$
6      $J \leftarrow 0$                                              ▷ Initialize loss
7      $\mathcal{S}_{\text{batch}} \leftarrow$ RANDOMSAMPLE$(\mathcal{D}_S, B)$
8      for $(\mathbf{x}, y) \in \mathcal{S}_{\text{batch}}$ do
9          $L_{\text{un}} \leftarrow (1-\lambda)L_{\text{base}} + \lambda L_{\text{target}}$          ▷ Eq. (7.9)
10         $J \leftarrow J + \nabla L_{\text{un}}$                       ▷ Update $\mathbf{a}, \mathbf{W}, \mathbf{V}, \mathbf{b}$
11 end loop
12 Form $\mathbf{x}, f$ with updated $\mathbf{a}, \mathbf{W}$, $\mathbf{V}, \mathbf{b}$                  ▷ Eq. (7.6)
13 $\hat{\mathbf{y}} \leftarrow \emptyset$
14 for $\mathbf{x}_i \in \mathcal{D}_T$ do
15     $\hat{\mathbf{y}}_i \leftarrow \Theta(\sigma(f(\mathbf{x}_i) \odot \mathbf{x}_i))$
16 return $\hat{\mathbf{y}}$, $\mathbf{a}, \mathbf{W}$
```

from seen data sources.

$$L_{\text{support}} = \sum_{y_i=1} \frac{d(f(\mathbf{x}_i^+), \mathbf{c}_{\mathcal{D}_S}^+)}{\bar{d}_{\mathcal{D}_S}^+} \log \hat{y}_i + \sum_{y_i=0} \frac{d(f(\mathbf{x}_i^-), \mathbf{c}_{\mathcal{D}_S}^-)}{\bar{d}_{\mathcal{D}_S}^-} \log(1-\hat{y}_i) \tag{7.12}$$

where d denotes the Euclidean distance, $\bar{d}^{+/-}$ represents the mean distance for all positive/negative pairs in $\mathcal{D}_S$ to the corresponding centroid. Thus, by integrating $L_{support}$ with L_{base}, the updated loss of ADAMEL in the supervised setting is denoted as follows:

$$L_{\text{ssl}} = L_{\text{base}} + \phi L_{\text{support}} \tag{7.13}$$

where $\phi \in (0, 1]$ is a hyperparameter that controls the impact of the labeled support set. The training process updates not only parameters in the neural network Θ for better classification performance, but also the attention embedding function f so that the projected positive and negative feature attentions are matched closer. In this process, feature importance from the new data sources unseen in $\mathcal{D}_S^*$ can be incorporated to update the centroids $\mathbf{c}^{+/-}$ in the supervised manner. We name this solution **ADAMEL-FEW** as it uses a few labeled data in $\mathcal{D}_T$, and depicts the process in Algorithm VII.2. Particularly, line 7- 8 denote the training process of f to minimize the loss L_{base}, and line 10- 11 denote the process of further training under the supervision of labeled samples in $\mathcal{S}_U$.

Algorithm VII.2 AdaMEL-few

```
Ensure: $\mathcal{D}_S = \{(\mathbf{h}_i, y_i)\}$, $\mathcal{S}_U = \{(\mathbf{h}_i, y_i)\}$, $\mathcal{D}_T = \{\mathbf{h}_i\}$, $\phi$, $B$
Require: Predicted $\hat{y_i}$ for $\mathbf{h}_i \in \mathcal{D}_T$, updated $\mathbf{a}, \mathbf{W}$
 1 Initialize $\mathbf{a}, \mathbf{W}$ and $\mathbf{V}, \mathbf{b}$
 2 loop training epochs
 3     for $\mathbf{h} \in \mathcal{D}_S \cup \mathcal{D}_T$ do
 4         Form $\mathbf{x}$ with $\mathbf{V}, \mathbf{b}$                              ▷ Eq. (7.4)
 5     $J \leftarrow 0$                                                      ▷ Initialize loss
 6     $\mathcal{S}_{\text{batch}} \leftarrow$ RANDOMSAMPLE($\mathcal{D}_S, B$)
 7     for $(\mathbf{x}, y) \in \mathcal{S}_{\text{batch}}$ do
 8         $J \leftarrow J + \nabla L_{\text{base}}$                                    ▷ Eq. (7.8)
 9     Form $f$ with updated $\mathbf{a}, \mathbf{W}$                                   ▷ Eq. (7.6)
10     Compute $\mathcal{D}_S^+$, $\mathcal{D}_S^-$, $\bar{d}_{\mathcal{D}_S}^+$, $\bar{d}_{\mathcal{D}_S}^-$                         ▷ Eq. (7.11)
11     $L_{\text{ssl}} \leftarrow L_{\text{base}} + \phi L_{\text{support}}$                          ▷ Eq. (7.13)
12     $J \leftarrow J + \nabla L_{\text{ssl}}$                              ▷ Update $\mathbf{a}, \mathbf{W}, \mathbf{V}, \mathbf{b}$
13 end loop
14 Infer $\hat{\mathbf{y}}$                                  ▷ Same as Line 13- 15 of Algorithm VII.1
15 return $\hat{\mathbf{y}}$, $\mathbf{a}, \mathbf{W}$
```

7.4.4.3 Hybrid Model

We further propose a hybrid model that incorporates both the labeled support set as well as the unlabeled data in the target domain in the training process. It can be seen as the composition of AdaMEL-zero and AdaMEL-few. The loss function is as follows.

$$L_{\text{hybrid}} = (1 - \lambda)L_{\text{base}} + \lambda L_{\text{target}} + \phi L_{\text{support}} \tag{7.14}$$

This variant uses the loss L_{target} defined in Equation (7.10) and L_{support} defined in Equation (7.12). We name this hybrid solution as **AdaMEL-hyb**. The algorithm is similar to Algo. VII.2, the main difference is to incorporate L_{un} *i.e.*, Equation (7.9) into the training process (line 7- 8) to learn the parameters simultaneously. The detailed algorithm of AdaMEL-hyb is shown in Algorithm VII.3.

Algorithm VII.3 AdaMEL-hyb

```
Ensure: $\mathcal{D}_S = \{(\mathbf{h}_i, y_i)\}$, $\mathcal{S}_U = \{(\mathbf{h}_i, y_i)\}$, $\mathcal{D}_T = \{\mathbf{h}_i\}$, $\phi$, $B$
Require: Predicted $\hat{y_i}$ for $\mathbf{h}_i \in \mathcal{D}_T$, updated $\mathbf{a}, \mathbf{W}$
Initialize $\mathbf{a}, \mathbf{W}$ and $\mathbf{V}, \mathbf{b}$
loop training epochs
    for $\mathbf{h} \in \mathcal{D}_S \cup \mathcal{D}_T \cup \mathcal{S}_U$ do
        Form $\mathbf{x}$ with $\mathbf{V}, \mathbf{b}$                      ▷ Eq. (7.4)
    $\bar{f}(\mathbf{x}') \leftarrow \frac{1}{|\mathcal{D}_T|} \sum_{\mathbf{x_i} \in \mathcal{D}_T} f(\mathbf{x_i})$
    $J \leftarrow 0$                                              ▷ Initialize loss
    $\mathcal{S}_{\text{batch}} \leftarrow$ RANDOMSAMPLE($\mathcal{D}_S, B$)
    for $(\mathbf{x}, y) \in \mathcal{S}_{\text{batch}}$ do
        $L_{\text{un}} \leftarrow (1-\lambda) L_{\text{base}} + \lambda L_{\text{target}}$     ▷ Eq. (7.9)
        $J \leftarrow J + \nabla L_{\text{un}}$
    Form $f$ with updated $\mathbf{a}, \mathbf{W}$                          ▷ Eq. (7.6)
    Compute $\mathcal{D}_S^+$, $\mathcal{D}_S^-$, $\bar{d}_{\mathcal{D}_S}^+$, $\bar{d}_{\mathcal{D}_S}^-$                ▷ Eq. (7.11)
    $L_{\text{hybrid}} = L_{\text{un}} + \phi L_{\text{support}}$                     ▷ Eq. (7.14)
    $J \leftarrow J + \nabla L_{\text{hybrid}}$                      ▷ Update $\mathbf{a}, \mathbf{W}, \mathbf{V}, \mathbf{b}$
end loop
Infer $\hat{\mathbf{y}}$                        ▷ Same as Line 13- 15 of Algorithm VII.1
return $\hat{\mathbf{y}}$, $\mathbf{a}, \mathbf{W}$
```

7.4.5 Parameter Complexity

We measure the parameter complexity of AdaMEL in terms of the numbers of learnable parameters that comes from three parts: (i) per-feature non-linear affine operations that transform the word token embeddings to the latent feature vectors, (ii) the shared feature attention embedding function f, which includes learning $\mathbf{W}$ and $\mathbf{a}$, and (iii) the multilayer perceptron (MLP) Θ with 1 hidden layer for classification. For (i), there are totally F features, each feature is associated with $\mathbf{V}^{H \times D}$ and $\mathbf{b}$, thus leading to $\mathcal{O}(FDH)$ learnable parameters. For (ii), as $\mathbf{W}^{H' \times H}$ and $\mathbf{a}^{H'}$ are shared across all features, there are totally $\mathcal{O}(HH')$ parameters. The neural network Θ in (iii) takes the concatenated FH'-dim features as input with one H_{hidden}-dim hidden layer. Therefore, AdaMEL has totally $\mathcal{O}(FDH + HH' + FH'H_{\text{hidden}})$ parameters to learn. We discuss the setup values of H, H' and H_{hidden} in the configuration of Section 7.5, and empirically estimate the parameter number in Section 7.5.6.

7.5 Experiments

In this section we describe the experiments to evaluate properties and the performance of AdaMEL. Specifically, we aim to answer the following research questions: **Q1.** Does AdaMEL effectively handle MEL with the data challenges (**C1-C3**) under the transfer learning paradigm? **Q2.** How well does AdaMEL adapt feature importance in the target domain and how does it affect the linkage results? **Q3.** Are generated feature attention values meaningful? **Q4.** How stable is AdaMEL in handling different data sources? **Q5.** How does the size of support set $\mathcal{S}_U$ impact the performance of AdaMEL? We conclude with the model justification (ablation study, limitation).

Table 7.2: Data statistics and properties.

Data	# Records	Entity_types	$\lvert\mathcal{D}_S^*\rvert$	$\lvert\mathcal{D}_T^*\rvert$	$\lvert\mathcal{A}\rvert$
`Monitor`	66,795	Monitor	5	24	13
`Music-3K`	3,070	Artist, Album, Track	3	7	9
`Music-1M`	1,723,426	Artist, Album	3	7	9

7.5.1 Experimental Setup

Data In accordance with (**Q1-Q5**), we use both the public benchmark dataset from the Data Integration to Knowledge Graphs (DI2KG) challenge [di220] and two real-world datasets in different scales from an online-sales company. Both datasets are in the tabular form and the entities are associated with descriptive textual features. The data statistics and source info is given in Table 7.2.

- **Music-1M** is a weakly labeled corpus crawled from 7 public music websites. We name them *website 1-7* for confidentiality. There are 2 entity types: artists and albums. Entity pairs are labeled following the hyperlinks in pages, so there might be mixed-type errors such as matching an artist to her album.

- **Music-3K** is a manually labeled corpus containing the same data sources as **Music-1M**. It has three types: artist, album and tracks. The manual annotation is based on 9 attributes such as the artist name and album title. Errors such as mixed-type matching are carefully corrected.

- **Monitor** contains monitor data from 24 sales websites such as *ebay.com* and *shopmania.com.* We filter out attributes with > 60% empty records, and get totally 13 attributes such as product description, manufacturer info, condition status, etc.

Comparing with the public benchmark datasets [MLR+18], the above datasets are collected from larger ranges of real-world data sources with heterogeneous schemas. The attribute values in the above datasets are generally longer with diverse characters, which makes it harder to summarize the attribute representation. For example, for `Music-3K`, artist type, the averaged attribute length is 25.75 word tokens, and for `Monitor`, the averaged attribute length is 11.73 word tokens. On the contrary, this number is 6.26 and 5.21 word tokens for the benchmark "dirty" and "heterogeneous" `Walmart-Amazon` dataset [FHHS20], respectively. In terms of the `Music` datasets, as the music works come from different countries, many entities are recorded with non-English characters & phrases for attributes such as the title, album and artist names. Unlike `Music-1M` that labels entity pairs through website hyperlinks, `Music-3K` further inspects whether the pair of music works indicate the same physical copy (*i.e.*, "Album"), or the same digital copy in formats such as remix or cover (*i.e.*, "Track"). The `Monitor` dataset is highly imbalanced with more than 99% entity pairs being unmatched. Additionally, less than 50% entity pairs in this dataset have non-missing values for most attributes. Non-missing pairs of 5 attributes only exist in the target domain. These data challenges do not exist in the benchmark datasets [MLR+18], and we detail the analysis in the following part. All the above issues make the datasets more challenging and closer to the real-world knowledge integration scenario.

Table 7.3: Train, support and test statistics in the experiments.

Data	**Entity_type**	**Train** $|\mathcal{D}_S|$	**Support** $|\mathcal{S}_U|$	**Test** $|\mathcal{D}_T|$
`Music-3K`	Artist	374	100	541
	Album	490	100	509
	Track	314	100	542
`Music-1M`	Artist	298 566	100	541
	Album	697 739	100	509
`Monitor`	Monitor	17 766	100	1 432

Public Data Processing Here we detail the processing of the public dataset, `Monitor` that is used in the experiment. We follow the *'monitor_label.csv'* file (1 073 positive pairs and 110 082 negative pairs) from the DI2KG to create the labeled entity pairs that fall into the

24 data sources we are interested in. The resultant dataset, `Monitor` has 734 positive and 66 061 negative pairs. The source domain $\mathcal{D}_S^*$ contains 5 data sources (*i.e.*, $\mathcal{D}_S^* = \{$*ebay.com*, *catalog.com*, *best-deal-items.com*, *cleverboxes.com*, *ca.pcpartpicker.com*$\}$), which contains 302 positive pairs. Thus, the test data includes all the remaining positive 432 pairs and randomly-selected $1,000$ negative pairs.

We provide the code and the splitted public `Monitor` data in the following in the repo `https://github.com/DerekDiJin/AdaMEL-supplementary`.

Public Data Challenges To illustrate the data challenges (**C1-C3**), we provide detailed analysis of the `Monitor` data. We first study the difference of the source and target domain in terms of the attributes, *i.e.*, missing attribute values (**C1**) and new attributes (**C2**). As the attributes associated with entity pairs, we plot the percentage of pairs without missing values per attribute, *i.e.*, $(r[A], r'[A])$ where $r[A] \neq \emptyset, r'[A] \neq \emptyset$ for $A \in \mathcal{A}$ in both the source and target domain. This metric also indicates the difference of data source attributes because for unseen incoming attributes, at least one entities in a pair should have the missing value. The result is depicted in Figure 7.6. Ideally, the percentage bars included in this plot should all be close to 1, and this holds for both data in the source and target domain. In fact, we observe this pattern in the benchmark datasets [MLR+18] (such as Beer, DBLP-ACM, etc.), which indicates few missing values and no significantly different attributes. For the `Monitor` dataset, however, the pattern is different. We first observe that only 2 attributes (*i.e.*, "page_title" and "source") are close-to-1, while for all the remaining 11 attributes, less than 50% entity pairs have complete attribute values. Such data sparsity reflects the challenge (**C1**). In addition, we observe that the percentages of pairs without missing values are significantly different for the source and target domain. Particularly, we find that there are 5 out of 13 attributes only have non-missing entity pairs only in the target domain, which can be seen as new attributes (**C2**).

To illustrate the different attribute value distribution (**C3**), we showcase the attribute value distribution of one attribute, "prod_type" as the representative. We plot the frequency distribution of 10 most frequently appearing word tokens. As shown in Figure 7.7, the distributions of attribute "prod_type" are quite different between the source and target domain, which indicates the challenge we attempt to address.

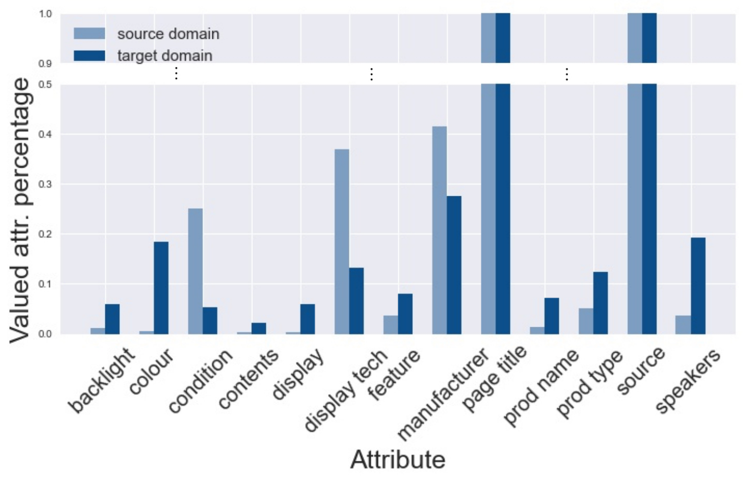


Figure 7.6: `Monitor`: **the challenges of missing attribute values (C1) and new attributes (C2) between $\mathcal{D}_S$ and $\mathcal{D}_T$ is shown with the percentages of entity pairs without missing values per attribute (*i.e.*, nonempty for both entities). For most attributes, the majority of entity pairs have at least 1 entity with missing values. 5 out of 13 attributes have non-missing entity pairs only in the target domain (2 non-missing attributes are "page title" and "source"). For the remaining 6 attributes, the percentage is also significantly different between the source and target domain.**

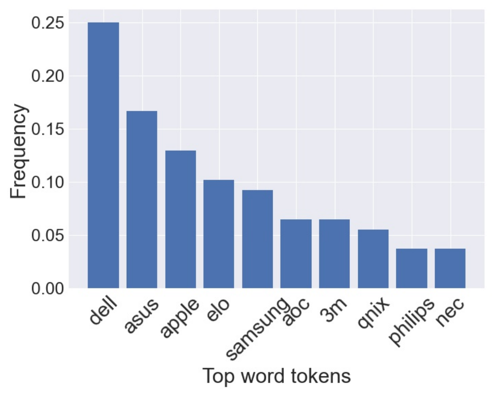
Baselines. The following baselines are used in this work.

- **TLER** [TPO$^+$18] is a non-deep transfer learning framework that defines a standard feature space and reuses the seen data to train models for the new domain.

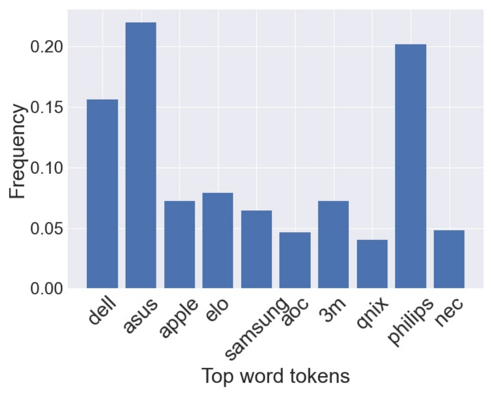
- **DeepMatcher** [MLR$^+$18] is a deep learning framework that consists of 3 modules: attribute embedding, attribute similarity representation, and classification. The public implementation uses Fasttext to embed attribute words and uses attentative RNN to summarize attributes. We report results using the best-performing variant, DeepMatcher-hybrid.

- **EntityMatcher** [FHHS20] is a hierarchical deep framework for heterogeneous schema matching. It jointly matches entities at the level of token, attribute, and entity. The token-level matching strategy allows EntityMatcher to perform cross-attribute alignment. Fasttext is used to embed word tokens.

- **Ditto** [LLS$^+$20] is an EL system that leverages fine-tuned, pre-trained Transformer-based language models (*i.e.*, BERT, DistilBERT, or RoBERTa) with optimization including domain knowledge injection, text summarization, and data augmentation



(a) Frequency of top 10 word tokens in the source domain



(b) Frequency of top 10 word tokens in the target domain

Figure 7.7: `Monitor`: the challenges of different attribute value distribution (C3) shown with the representative attribute "prod_type". The frequency distribution of top 10 word tokens under this attribute is significantly different between the source and target domain.

with difficult samples.

- **CorDel** [WSW+20] adopts an alternative deep architecture to the widely-used "twin architecture". It compares and contrasts word tokens to filter out minor deviations between attribute values before embedding. CorDel also uses Fasttext, and it shows higher performance with reduced runtime compared to DeepMatcher. Out of the variants, CorDel-Attention is reported to perform the best on dirty EL datasets.

We consider these baselines since they are reported to achieve state-of-the-art EL performance and outperform methods such as Seq2SeqMatcher [NHH+19] and DeepMatcher+ [KQG+19]. Most of them are particularly proposed to handle heterogeneous entity linkage.

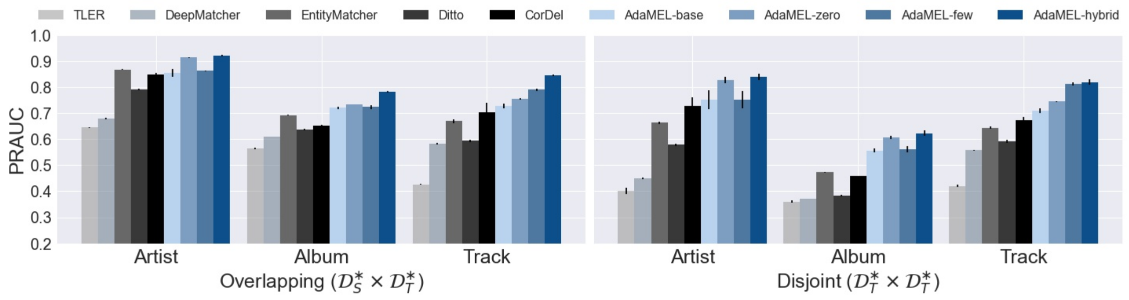
Configuration. In our experiments, we follow the original paper and fine-tune the baselines for optimal performance. The statistics of training, support and testing data is given in Table 7.3. Specifically, `Musci-1M` shares the same testing set as `Musci-3K`. `Monitor` adopts all positive and randomly selected 1000 negative pairs to form the testing set. For DeepMatcher, we use its hybrid variant (bi-directional RNN with attention) to summarize attributes with 2-layer highway neural network ((hidden dim= 300)). The training epoches is set to 40 with batch size = 16. For EntityMatcher, we use the full matching model that uses bi-GRU (hidden size= 150) to embed attribute word sequences with cross-attribute token-level alignment. The training epoch is set to 20 with batch size = 16. For CorDel, we use the attention-based variant that learns the word importance within the same attribute to validate the effectiveness of our attribute-level attention module. Moreover, CorDel-Attention was shown to perform best on long textual attribute values, which matches the property of our input data. All these 3 baselines use the pretrained FastText [JGBM17] to derive the 300-dimensional embeddings for word tokens in each attribute. We set the cropping size = 20 and sum the embeddings of word tokens as the feature embeddings for CorDel. The training epoch is set to 20 with learning rate = 10^{-4} and batch size = 16. For Ditto, we tested its optimization strategies and adopted the "token span deletion" for data augmentation, "general" domain knowledge and retaining high TF-IDF tokens to summarize the input sequences. We also tested all pretrained language models, *i.e.*, bert, distilbert, and albert, and ended up using bert. The training epoch is set to 40 with batch size= 64 and learning rate= 3×10^{-5}.

To evaluate the effectiveness of our proposed framework, we configure AdaMEL with consistent setup as the baselines. Specifically, we use the 300-dim Fasttext to embed word tokens for fairness because 3 of the 4 baselines also use it, even though AdaMEL supports any word embedding techniques such as Bert embedding [KT19]. We set the cropping size= 20 as CorDel. The hyparameters of AdaMEL are given as follows: the dimension of the projected embeddings per feature is $H = 64$, the dimension of the hidden layer in f is $H' = 256$, and the dimension of the hidden layer in Θ is $H_{\text{hidden}} = 256$. The activation σ is set to be Relu. We set $\lambda = 0.98$ and $\phi = 1.0$ in Equation (7.9), (7.13) and (7.14) for AdaMEL variants unless otherwise addressed. To train the AdaMEL, we use Adam optimizer [KB14] for 100 epoches with learning rate = 10^{-4} and batch size = 16. We conduct all experiments 3 times and report the mean and std. We run these experiments on the Linux platform with 2.5GHz Intel Core i7, 256GB memory and 8 NVIDIA K80 GPUs.

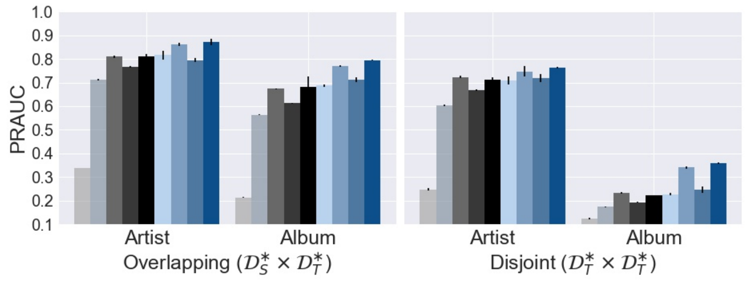
Evaluation Metric. We evaluate the model performance using PRAUC as it measures the precision-recall relation globally and handles data imbalance. We use the python Sklearn library to compute PRAUC based on the open-source implementation of all baselines.

7.5.2 Transfer Learning for MEL

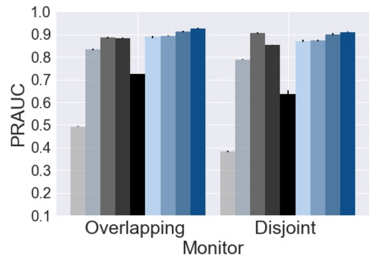
Our first experiment is to verify the effectiveness of AdaMEL variants on the task of MEL (**Q1**). We simulate two real-world scenarios: **(S1)** data in the target domain ($\mathcal{D}_T$) shares common data sources with the source domain ($\mathcal{D}_S$) (*i.e.*, $(r, r')_T \in \mathcal{D}_S^* \bigtimes \mathcal{D}_T^*$), and **(S2)** data



(a) MEL performance on `Music-3K`



(b) MEL performance on `Music-1M`



(c) MEL performance on `Monitor`

Figure 7.8: MEL performance (PRAUC) comparison between AdaMEL variants and baselines. AdaMEL variants outperform baseline heterogeneous entity matching methods in almost all cases. Particularly, AdaMEL-hyb performs the best on all entity types and datasets.

sources in the target domain are disjointed from the source domain (*i.e.*, $(r, r')_T \in \mathcal{D}_T^* \bigtimes \mathcal{D}_T^*$).

Setup. For the `Music` data, we use three data sources (*i.e.*, $\mathcal{D}_S^* = \{$*website 1*, *website 2*, *website 3*$\}$) to train our model and test on all 7 sources (overlapping scenario **S1**) or only the 4 remaining sources (disjoint scenario **S2**) as the target domain $\mathcal{D}_T$. In either scenario, we collect 100 samples (50 positive and 50 negative) from the corresponding $\mathcal{D}_T$ as support set $\mathcal{S}_U$. For the public `Monitor` data, we use entity pairs from 5 sources (*i.e.*, $\mathcal{D}_S^* = \{$*ebay.com*, *catalog.com*, *best-deal-items.com*, *cleverboxes.com*, *ca.pcpartpicker.com*$\}$) to train the models. We use data in all 24 sources as $\mathcal{D}_T$ for **S1**, and the rest 19 data sources for **S2**, respectively. 100 samples are collected as $\mathcal{S}_U$ in the same way as `Music`. We also randomly picked different sources to form $\mathcal{D}_S$ and $\mathcal{D}_T$ to eliminate the randomness, and found similar patterns in the results.

Results. We report the results in Figure 7.8 with complete numerical results in Table 7.5 and 7.4 in the end of this section. Our first observation is that all AdaMEL variants tend to outperform the baseline methods and our base model without adaptation, AdaMEL-base. The heterogeneous entity matching baselines do not perform well on these datasets under the supervision of labeled data only. This is likely because of the long and noisy word sequences in the data and the difference in attribute value distribution across data sources that is unseen during model training. AdaMEL highlights the impact of important features, and only represent the sequences by summing the token embeddings. This confirms our conjecture that learning the attribute-level attention as the transferable knowledge is more effective in handling the MEL task than refining the word-level sequence representation. Also, we observe that out of all variants, AdaMEL-hyb achieves the best performance in all cases with 0.64% $\sim$ 5.50% improvement in PRAUC than the second-best (AdaMEL-zero in most cases), which demonstrates its effectiveness in integrating both the labeled support set $\mathcal{S}_U$ and unlabeled info from the target domain $\mathcal{D}_T$. AdaMEL-zero performs better than AdaMEL-few on the "Artist" and "Album" type, while AdaMEL-few performs better on the "Track" type. This is likely due to the fact that the track records are more diverse than the other types as the digital-format music tracks can be remixed or covered by other artists. Thus, the high-quality labeled samples from $\mathcal{S}_U$ is of higher value. On the contrary, since the records are more consistent for the "Artist" and "Album" type, incorporating more records in $\mathcal{D}_T$ leads to higher improvement in MEL performance. Note Figure 7.8b shows that AdaMEL-few performs slightly worse than AdaMEL-base because the labeled samples from $\mathcal{S}_U$ only overfits to the trained model on the source domain, that deviates the actual feature importance for the massive unlabeled samples in $\mathcal{D}_T$. To summarize, the improvement of AdaMEL-zero, -few and -hyb over the baselines indicates the effectiveness of domain adaptation in incorporating data in $\mathcal{D}_T$.

Overall, AdaMEL variants achieve better performance on the overlapping scenario (**S1**) than the disjoint scenario (**S2**). This is as expected as the disjoint scenario represents an extreme case where data sources in $\mathcal{D}_T$ are less likely or even entirely not used in training the model if the support set is unavailable. Besides, the performance of all approaches running on `Music-1M` is lower than `Music-3K`. The main reason is that the data is weakly labeled as it simply follows the hyperlinks from the websites, and does not distinguish the actual media of the music work (*i.e.*, the physical or digital copy). As the models are tested on the same well-labeled set, training on `Music-1M` could be vulnerable to cases such as mixed-type matching. Nevertheless, we observe that AdaMEL still achieves promising results in both hard cases of transfer learning for MEL, *i.e.*, unseen data sources in the target domain and training on weakly labeled data, which further demonstrates the advantage of AdaMEL.

Table 7.4 gives the result on `Monitor`. Similarly, AdaMEL variants tend to outperform the baselines and AdaMEL-hyb performs the best with at least 0.51% improvement in PRAUC over the second best, EntityMatcher. On average, AdaMEL-hyb outperforms the baseline by 9.39% improvement in the overlapping scenario and 11.55% improvement in the disjoint scenario. These results also validates our findings above.

Table 7.4 records the complete numerical results on the `Monitor` dataset. The results on the `Music` dataset are shown in Table 7.5.

Table 7.4: AdaMEL performance (PRAUC) on `Monitor`. All variants outperform the baseline, AdaMEL-hyb performs the best (marked in bold) with at least 0.51% improvement over the second-best (∗).

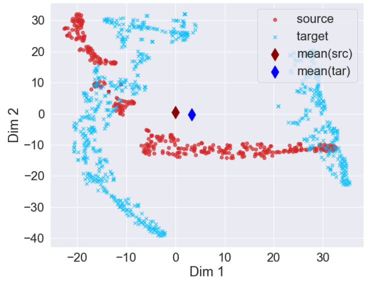
	Method	Overlapping	Disjoint
`Monitor`	TLER	0.4932 ± 0.0028	0.3837 ± 0.0033
	DeepMatcher	0.8336 ± 0.0032	0.7884 ± 0.0011
	EntityMatcher	0.8858 ± 0.0034	0.9051 ± 0.0042*
	Ditto	0.8841 ± 0.0010	0.8518 ± 0.0023
	CorDel-Attention	0.7240 ± 0.0026	0.6353 ± 0.0165
	AdaMEL-base	0.8884 ± 0.0057	0.8711 ± 0.0050
	AdaMEL-zero	0.8930 ± 0.0013	0.8719 ± 0.0030
	AdaMEL-few	0.9127 ± 0.0035*	0.9005 ± 0.0059
	AdaMEL-hyb	**0.9258 ± 0.0025**	**0.9106 ± 0.0029**

Table 7.5: AdaMEL performance (PRAUC) of multi-source entity linkage on the `Music` data. The best score of each entity type is marked in bold. Out of AdaMEL variants, AdaMEL-hyb performs the best with 0.64% ∼ 5.50% improvement over the second-best variant (marked with ∗) in PRAUC. AdaMEL-hyb outperforms the best-performing baselines (including AdaMEL-base) with 8.21% improvement on average.

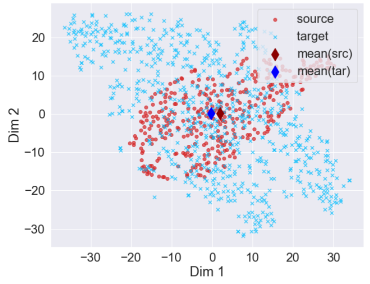
	Method	Overlapping ($\mathcal{D}_S^* \times \mathcal{D}_T^*$)			Disjoint ($\mathcal{D}_T^* \times \mathcal{D}_T^*$)		
		Artist	Album	Track	Artist	Album	Track
`Music-3K`	TLER	0.6454 ± 0.0021	0.5655 ± 0.0032	0.4263 ± 0.0011	0.4014 ± 0.0121	0.3605 ± 0.0033	0.4203 ± 0.0042
	DeepMatcher	0.6794 ± 0.0022	0.6093 ± 0.0009	0.5826 ± 0.0017	0.4492 ± 0.0021	0.3710 ± 0.0012	0.5572 ± 0.0014
	EntityMatcher	0.8682 ± 0.0017	0.6922 ± 0.0021	0.6694 ± 0.0084	0.6629 ± 0.0032	0.4733 ± 0.0014	0.6446 ± 0.0032
	Ditto	0.7920 ± 0.0032	0.6373 ± 0.0042	0.5938 ± 0.0051	0.5786 ± 0.0039	0.3832 ± 0.0027	0.5914 ± 0.0055
	CorDel-Attention	0.8489 ± 0.0047	0.6531 ± 0.0019	0.7032 ± 0.0364	0.7280 ± 0.0315	0.4586 ± 0.0002	0.6738 ± 0.0121
	AdaMEL-base	0.8545 ± 0.0143	0.7204 ± 0.0033	0.7277 ± 0.0077	0.7516 ± 0.0367	0.5569 ± 0.0072	0.7107 ± 0.0093
	AdaMEL-zero	0.9142 ± 0.0018*	0.7338 ± 0.0001*	0.7547 ± 0.0027	0.8263 ± 0.0121*	0.6071 ± 0.0072*	0.7453 ± 0.0012
	AdaMEL-few	0.8633 ± 0.0011	0.7241 ± 0.0080	0.7904 ± 0.0048*	0.7510 ± 0.0331	0.5619 ± 0.0119	0.8129 ± 0.0057*
	AdaMEL-hyb	**0.9211 ± 0.0040**	**0.7833 ± 0.0031**	**0.8454 ± 0.0040**	**0.8390 ± 0.0125**	**0.6229 ± 0.0115**	**0.8193 ± 0.0097**
`Music-1M`	TLER	0.3384 ± 0.0013	0.2128 ± 0.0019	-	0.2465 ± 0.0052	0.1237 ± 0.0031	-
	DeepMatcher	0.7132 ± 0.0033	0.5629 ± 0.0021		0.6033 ± 0.0045	0.1742 ± 0.0013	
	EntityMatcher	0.8098 ± 0.0043	0.6731 ± 0.0024		0.7239 ± 0.0038	0.2331 ± 0.0031	
	Ditto	0.7663 ± 0.0025	0.6123 ± 0.0022		0.6678 ± 0.0019	0.1933 ± 0.0027	
	CorDel-Attention	0.8118 ± 0.0087	0.6811 ± 0.0432		0.7129 ± 0.0096	0.2224 ± 0.0010	
	AdaMEL-base	0.8165 ± 0.0184	0.6872 ± 0.0053		0.7086 ± 0.0180	0.2269 ± 0.0050	
	AdaMEL-zero	0.8607 ± 0.0066*	0.7693 ± 0.0038*		0.7469 ± 0.0228*	0.3407 ± 0.0056*	
	AdaMEL-few	0.7942 ± 0.0090	0.7126 ± 0.0102		0.7177 ± 0.0171	0.2473 ± 0.0131	
	AdaMEL-hyb	**0.8710 ± 0.0130**	**0.7942 ± 0.0015**		**0.7632 ± 0.0034**	**0.3582 ± 0.0043**	

7.5.3 Effectiveness of Adaptation

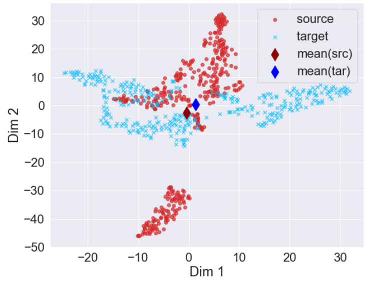
Adaptation is the key component to our proposed method. To evaluate how well AdaMEL learns feature importance adapted to the target domain (**Q2**), we study the effectiveness of



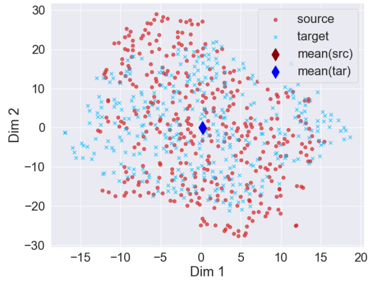
(a) AdaMEL-zero with no adaptation ($\lambda = 0$)



(b) AdaMEL-zero with adaptation ($\lambda = 0.98$)



(c) AdaMEL-hyb with no adaptation ($\lambda = 0$)



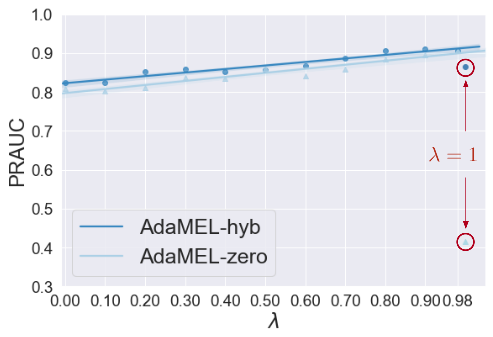
(d) AdaMEL-hyb with adaptation ($\lambda = 0.98$)

Figure 7.9: Source and target domain feature attention vectors are better aligned with high value of λ for both AdaMEL-few and AdaMEL-hyb (visualized with TSNE, dim=2).

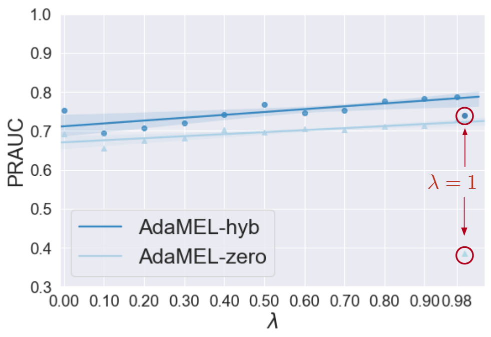
λ adopted in AdaMEL-zero and AdaMEL-hyb as it controls the weight of adapting to unlabeled data in training (larger λ leads to more adaptation).

Setup. We run both variants of AdaMEL on the `Music-3K` dataset and report the performance on MEL. As discussed in Section 7.4.4.1, records from both the source and target domains are projected into the same space using the shared attention embedding function, and AdaMEL attempts to adapt the model to match these feature importance distribution. Intuitively, with sufficient adaptation, feature importance vectors from both domains should align well, and further benefit the linkage task. To validate this conjecture, we visualize the learned feature attention vectors using AdaMEL-zero and AdaMEL-hyb with different values of λ by projecting them into 2-dimensional space using TSNE [MH08]. We also study the linkage performance of AdaMEL-zero and AdaMEL-hyb with different λ values on the "artist" and "album" type of the `Music-3K` dataset.

Results. In Figure 7.9, we observe that for both variants, feature attention vectors from $\mathcal{D}_S$ and $\mathcal{D}_T$ align better when $\lambda = 0.98$ than $\lambda = 0$, which confirms the effectiveness of adaptation.



(a) AdaMEL performance change on `Music-3K`**, artist type**



(b) AdaMEL performance change on `Music-3K`**, album type**

Figure 7.10: AdaMEL-zero and AdaMEL-hyb performance improve with increasing λ from 0 to 0.98 (fitted with linear regression). The performance drops when $\lambda = 1$ as no labeled data in $\mathcal{D}_S$ is used.

In addition, we observe that comparing with AdaMEL-zero (Figure 7.9b), AdaMEL-hyb (Figure 7.9d) generates better adapted results as the projected records from $\mathcal{D}_S$ and $\mathcal{D}_T$ are almost indistinguishable, which is as expected as the labeled support set is leveraged.

To evaluate the impact of adaptation to the linkage results, in Figure 7.10 we show the performance of our variants with different λ values. We observe that as λ approaches (but not equals) to 1, the general performance in terms of PRAUC improves for both AdaMEL-zero (0.8014 - 0.9091) and AdaMEL-hyb (0.8242 - 0.9201), which again demonstrates the effectiveness of adaptation. It is worth noting that when $\lambda = 1$, both AdaMEL-zero and AdaMEL-hyb perform worse without giving meaningful results. This is because at this point, AdaMEL-zero is trained without supervision of the labeling in $\mathcal{D}_S$, and the only term in the loss function is the regularization. AdaMEL-hyb is better as labeling in $\mathcal{S}_U$ is still used, but the overall performance deteriorates due to the lack of labeling from $\mathcal{D}_S$. As a result, the parameters trained ($i.e.$, $\mathbf{a}, \mathbf{W}$) would tend to only "match" the feature distribution between $\mathcal{D}_S$ and $\mathcal{D}_T$ that are not tailored to classification.

7.5.4 Attention Analysis

Setup. To testify whether AdaMEL learns meaningful feature attention values (**Q3**), we showcase the learned feature importance through the attention scores produced by AdaMEL on two datasets: `Music-3K` and `Monitor`. We only report the artist type and omit the other two types for brevity. AdaMEL-hyb is configured with the best performance ($\lambda = 0.98, \phi = 1.0$)

in the previous experiments.

Result. For the Monitor dataset in Table 7.6, we observe the long "tail distribution" of feature importance, *i.e.*, the most important feature is "Page_title_shared" with significantly high scores, while the other features are with roughly the same low scores. On the other hand, we observe the more uniform distribution for the artist type in Music-3K dataset, which makes sense as all top features are related to the artist names. The learned attention scores on both datasets imply that the task of MEL could be addressed with some of the most remarkable features (importance inequality).

Table 7.6: AdaMEL learned importance of top-5 features for Monitor and Music-3K, artist type.

Monitor		Music-3K, artist	
Feature	Score	Feature	Score
Page_title_shared	0.1635	Main_performer_shared	0.0739
Page_title_unique	0.0595	Name_unique	0.0697
Source_shared	0.0535	Name_shared	0.0628
Manufacturer_unique	0.0473	Source_unique	0.0597
Manufacturer_shared	0.0416	Name_Native_Language_shared	0.0583

We further run AdaMEL-hyb on these selected important features only and compare the performance with the result using the other features, as well as all the features. For Monitor, we use 3 attributes (*i.e.*, "Page_title", "Source" and "Manufacturer"). For the artist type of Music-3K, we use the 3 name-related attributes (*i.e.*, "Main_performer", "Name", Name_Native_Language), and "Source". Similarly, for the other two types, we use their corresponding top important attributes, and report the results in Table 7.7. We observe that by using the selected important features only, AdaMEL is capable of achieving comparable and even slightly better performance than using all features with 2.21%, 0.87% and 2.92% improvement in PRAUC on Monitor, Music-3K (artist) and Music-3K (album), respectively. For Music-3K (track), using the top attributes only performs slightly worse than using all attributes, which is likely due to the diversity of track records. Nevertheless, these experimental results show that model training can further benefit from feature importance as using all the possible attributes could introduce irrelevant or noisy input to the model (*e.g.*, using album-related features when inferring the artist type). Also, they shows the effectiveness of the feature attention module of AdaMEL in learning reasonable feature importance.

Table 7.7: Performance (PRAUC) comparison using the selected important features vs. the other features and all features. Numbers in the parenthesis denote the counts of features.

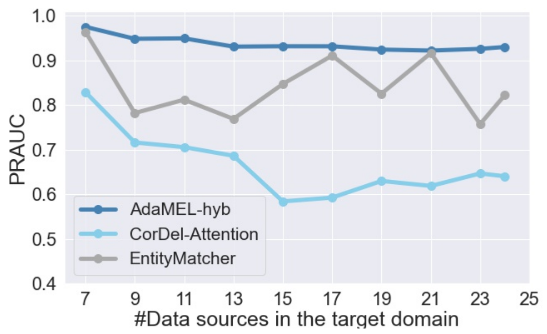
Dataset	Top Attributes (#)	Other Attributes (#)	All Attributes (#)
`Monitor`	**0.9479 ± 0.0007** (3)	0.4276 ± 0.0015 (10)	0.9258 ± 0.0025 (13)
`Music-3K`, artist	**0.9298 ± 0.0036** (4)	0.7966 ± 0.0005 (5)	0.9211 ± 0.0040 (9)
`Music-3K`, album	**0.8125 ± 0.0011** (4)	0.4692 ± 0.0009 (5)	0.7833 ± 0.0031 (9)
`Music-3K`, track	0.8398 ± 0.0004 (3)	0.7026 ± 0.0006 (6)	**0.8454 ± 0.0040** (9)

7.5.5 Data Sources Analysis

In this experiment we simulate the real-world knowledge integration, where new data sources often arrive one by one incrementally (such as in batches from neighboring data sources), and testify the stability of AdaMEL in handling the various data sources under this scenario (**Q4**) .

Setup. We use the public `Monitor` dataset and compare AdaMEL-hyb with the optimal configuration ($\lambda = 0.98, \phi = 1.0$ as shown in Section 7.5.2 and Section 7.5.3) with the best-performing baseline approach, EntityMatcher, and the fastest baseline approach, CorDel-Attention. In this experiment, we use 1500 entity pairs from the same 5 data sources as mentioned in Section 7.5.2 to train the models (*i.e.*, $\mathcal{D}_S^* = \{$*ebay.com*, *catalog.com*, *best-deal-items.com*, *cleverboxes.com*, *pcpartpicker.com*$\}$). To test the performance on MEL, we first randomly select 200 entity pairs from each of 7 data sources (the same 5 data sources as $\mathcal{D}_S^*$ and 2 unseen ones, *i.e.*, $\mathcal{D}_T^* = \mathcal{D}_S^* \cup \{$*yikus.com*, *getprice.com*$\}$) and form totally 1400 pairs to create the target domain. Then, we incrementally add up to 200 entity pairs from 2 new sources ($\Delta\mathcal{D}_T^*$) to $\mathcal{D}_T^*$, such that $\mathcal{D}_T^* = \mathcal{D}_T^* \cup \Delta\mathcal{D}_T^*$. Each of the newly added pairs $\{(r, r')\}$ contains at least one record from $\Delta\mathcal{D}_T$ to ensure new data sources are introduced to the target domain. As AdaMEL-hyb requires a small set of labeled entity pairs from $\mathcal{D}_T^*$, we randomly select 100 labeled samples from all data sources ($\mathcal{D}_S^* \cup \mathcal{D}_T^*$). This small set simulates the on-the-fly manual labeling in the real-world, and we fix it throughout each run of the experiment to ensure the impact of $\mathcal{S}_U$ is consistent. We also record the average runtime over all runs as an empirical study of the model efficiency.

Results. We report the performance of AdaMEL-hyb and the two baselines on MEL in Figure 7.11, as well as their empirical runtime. As shown in the figure, AdaMEL-hyb is more



Method	Runtime (s)
Hybrid	319.20 ± 7.20
CorDel	906.19 ± 46.35
E-Matcher	2500.43 ± 17.56

Figure 7.11: AdaMEL-hyb performs more stably ($0.9750 \sim 0.9219$ in PRAUC) as #data sources increases in $\mathcal{D}_T$ with less runtime.

stable than both EntityMatcher and CorDel-Attention with significantly higher performance in handling the incrementally incoming data sources. This is due to the fact that AdaMEL-hyb continuously updates parameters in the attention embedding function f to adapt to new data sources in $\mathcal{D}_T$. Comparing with CorDel-Attention, EntityMatcher performs better and could occasionally compete with AdaMEL-hyb under some scenarios ($|\mathcal{D}_T^*| = 17, 21$), but it is not stable as the performance fluctuates. Moreover, based on the table in Figure 7.11, AdaMEL-hyb takes much less time to train than CorDel-Attention and EntityMatcher. The empirical runtime comparison corresponds to our analysis in Section 7.4.5 as AdaMEL-hyb does not require sophisticated operations on word-level embeddings and thus having relatively less parameters to train. In practice, the number of parameters to train for AdaMEL-hyb is $\sim 2\,219\,520$, which is much less than the number given by EntityMatcher: $\sim 123\,119\,104$. These findings demonstrate the capability of AdaMEL in consistently handing MEL with a variety of incoming data sources, while being more robust. In addition, they strengthen our claim that finding important features as the transferable knowledge in MEL could benefit the model performance with reduced computational complexity.

7.5.6 Effectiveness of Support Set

Setup. To better understand the effectiveness of the labeled support set (**Q5**), we perform the sensitivity analysis with incrementally increasing numbers of labeled samples in the support set $\mathcal{S}_U$. Following Section 7.5.2, we randomly select 200 additional samples from $\mathcal{D}_T$ of the public `Monitor` dataset and create the support set with totally 300 labeled samples. We run two AdaMEL variants that leverage the support set, AdaMEL-few

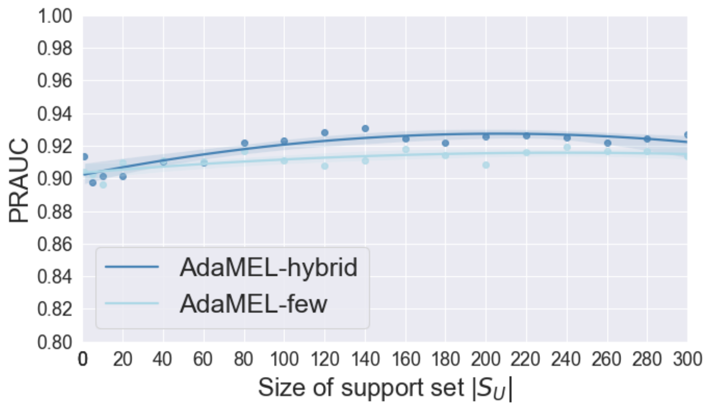


Figure 7.12: Sensitivity analysis of the size of support set $|\mathcal{S}_U|$ fitted with order-2 polynomial regression on AdaMEL-few and AdaMEL-hyb. As more labeled samples are included in $\mathcal{S}_U$, the model performance (PRAUC) increases initially and then flattens out.

($\phi = 1.0$) and AdaMEL-hyb ($\lambda = 0.98, \phi = 1.0$) in this experiment with $|\mathcal{S}_U|$ ranging from 1 to 300 with step size = 20 (specifically, we "zoom in" the smaller values and have $|\mathcal{S}_U| = \{1, 5, 10, 20, 40, 60, \cdots, 300\}$). In each run, the samples in $\mathcal{S}_U$ are randomly selected.

Result. The experimental result is shown in Figure 7.12. Our first observation is that at the initial stage of the experiment, the performance of both AdaMEL-few and AdaMEL-hyb improves as the number of used labeled samples from $\mathcal{S}_U$ increases. Particularly, we observe $\sim 1\%$ performance improvement from $|\mathcal{S}_U| = 1$ to $|\mathcal{S}_U| = 140$ for AdaMEL-few and $2\% \sim 3\%$ improvement for AdaMEL-hyb. This overall performance improvement is as expected since an increasing amounts of labeled samples from $\mathcal{D}_T$ are used to supervise the learning process. In the late stage ($|\mathcal{S}_U| > 140$), we observe that the performance fluctuates in each run and the overall performance saturates. This indicates that the feature importance learned by AdaMEL has sufficiently adapted and does not significantly change as more labeled data are collected in $\mathcal{S}_U$. Moreover, comparing with AdaMEL-few, AdaMEL-hyb performs similarly when the size of support set is small ($|\mathcal{S}_U| \leq 60$), and it consistently outperforms when $|\mathcal{S}_U| > 60$. This is likely due to the bias of feature importance brought by particular labeled samples selected when $|\mathcal{S}_U|$ is small. When $\mathcal{S}_U$ contains more samples, the learned feature importance becomes stable and sufficiently adapted to $\mathcal{S}_U$, and the outperformance given by AdaMEL-hyb over AdaMEL-few comes from the unlabeled samples from $\mathcal{D}_T$. As a rule of thumb, Figure 7.12 indicates that a small support set with $|\mathcal{S}_U| = 100 \sim 200$ labeled samples from $\mathcal{D}_T$ is beneficial to learn feature importance and to improve the MEL performance of AdaMEL. Too few samples would incur bias to the trained model, while too many samples would be expensive to obtain in practice, and does not necessarily help improve the model.

7.5.7 Model Justification

In this section we run experiments to justify the design choices of AdaMEL and its limitation.

7.5.7.1 Ablation Study

We perform the ablation study of AdaMEL that uses the shared and unique contrastive features, as well as using both of them as the default setting. Table 7.8 shows that including the shared and unique attribute values capture different perspectives of the data and thus enriches the feature space. Including both achieves the highest performance with 0.41% – 6.72% improvement over using one feature.

Table 7.8: Ablation study: AdaMEL contrastive features on `Music-3K`, artist and album type. AdaMEL-zero and -few perform similarly.

Dataset	Method	Shared	Unique	Shared & Unique
`Music-3K`, artist	AdaMEL-base	0.7868 ± 0.0045	0.7170 ± 0.0132	**0.8545** ± **0.0143**
	AdaMEL-hyb	0.8539 ± 0.0026	0.8069 ± 0.0112	**0.9211** ± **0.0040**
`Music-3K`, album	AdaMEL-base	0.7163 ± 0.0048	0.5520 ± 0.0044	**0.7204** ± **0.0033**
	AdaMEL-hyb	0.7504 ± 0.0059	0.5879 ± 0.0028	**0.7833** ± **0.0031**

7.5.7.2 Performance on Single Domain

Here we compare AdaMEL-zero and -hyb with DeepMatcher on the benchmark datasets to justify their performance on well-labeled data from the same seen domain without the 3 challenges (**C1 - C3**). From Table 7.9, we observe that AdaMEL-zero does not perform as well as DeepMatcher on these benchmark datasets of one single domain. This shows the limitation of AdaMEL in handling data with no missing values or schema difference. The reason is likely due to the simplicity of AdaMEL architecture, as it aims to learn the data-source-level feature importance instead of improving the token-level embeddings as DeepMatcher or its variants. In the real-world knowledge integration process where data distributions are highly heterogeneous, transferring these token-level contextualized embeddings brings extra computation and does not always generalize well, as shown in Section 7.5.2. Nevertheless, even though AdaMEL is designed to handle data challenges in practice (**C1-C3**), we observe that AdaMEL-hyb performs comparably as DeepMatcher with reduced model complexity, which shows its effectiveness of adaptation.

Table 7.9: Entity linkage performance (F1) of DeepMatcher, AdaMEL-zero and -hyb on the benchmark datasets, single domain scenario. AdaMEL-hyb performs comparably as DeepMatcher.

Type	Datasets	Domain	DeepMatcher	AdaMEL-zero	AdaMEL-hyb
Structured	Amazon-Google	Software	69.3	60.2	65.1
	Beer	Product	78.8	78.6	82.8
	DBLP-ACM	Citation	98.4	98.7	98.9
	DBLP-Google	Citation	94.7	93.1	93.5
	Fodors-Zagats	Restaurant	100	90.0	99.8
	iTunes-Amazon	Music	91.2	91.2	98.7
	Walmart-Amazon	Electronics	71.9	57.8	66.7
Dirty	DBLP-ACM	Citation	98.1	95.7	97.7
	DBLP-Google	Citation	93.8	89.7	91.5
	iTunes-Amazon	Music	79.4	79.3	80.7
	Walmart-Amazon	Electronics	53.8	48.2	52.2

7.6 Conclusion

In this work, we have tackled the problem of multi-source entity linkage (MEL) and have described a deep learning solution based on domain adaptation, AdaMEL. AdaMEL highlights the impact of important attributes in MEL and automatically learns the importance that adapts to both seen and unseen data sources as the generic transferable knowledge. We have proposed a series of variants to handle different real-world learning scenarios, depending on the availability of labeled entity pairs from the target domain. Comparing to heterogeneous schema matching baselines that are mostly based on supervised learning, AdaMEL is able to handle hard transfer learning cases such as unseen data sources in the target domain and achieve on average 8.21% improvement in PRAUC score for MEL. Extensive experiments have demonstrated the effectiveness of AdaMEL in adaptation. Additionally, we have provided an analysis of the learned feature attention, and studied the impact of different data sources and the size of the support set. Future directions include combining our work with advanced NLP techniques for sequence representation in attribute summarization to further improve the model performance in MEL, and extending our framework to handle data sources in different languages.

CHAPTER VIII

Conclusion

8.1 Summary

As a general type of data structure that models the relations between entities, graphs have been used to represent the multi-relational data from a wide variety of domains. Summarizing the key information from the continuously-increasing data volume has become key to reduce the complexity, which facilitates data compression, query efficiency, and interpretability. Node representation learning has been widely used to accurately characterize nodes in the graph for machine learning tasks based on rich latent features, but it suffers the drawbacks such as storage inefficiency, query inefficiency and the lack of interpretability.

This thesis bridged the two lines of research, node embedding and graph summarization via feature summarization. By summarizing features such as structural roles or temporal proximity into the latent space, we have proposed a series of node embedding approaches that achieve the state-of-the-art performance in machine learning tasks, while requiring significantly reduced storage. We have also proposed explicitly summarizing the non-latent features via modeling their importance in specific machine learning tasks, so that the summaries are easy-to-understand and being useful for machine learning tasks. As feature summaries can be projected into spaces that are not specific to individual datasets, they can be naturally applied in inductive learning or transfer learning, and thesis work has described methods for both the latent and non-latent space. Throughout the thesis, scalability has been an important consideration when handling challenges from real-world problems. We demonstrated the effectiveness of our feature summarization approaches through industrial applications, such as entity linkage, user stitching, professional role inference and temporal link prediction.

8.1.1 Latent Feature Summarization

Latent features form the basis for graph representation learning. Specifically for node representation learning, the goal is to project the node proximity or structural similarity into the fixed K-dimensional latent space. While the learned node-wise Euclidean vectors are highly effective in machine learning tasks such as link prediction and node classification, they incur significant challenges to storage. In Chapter III, we formally defined the problem of latent network summarization that aims to derive a compressed graph representation independent to the input graph sizes, and provided our solution, Multi-Lens. Multi-Lens uses a set of relational operations to collect the multi-level node structural contextual features, and derives the latent summary through low-rank matrix factorization. By storing the relational operations and the factorized matrix, Multi-Lens achieves space efficiency and supports on-the-fly node embedding derivation. Compared to embedding methods, the latent summaries generated by Multi-Lens require 80-2152$\times$ **less** output storage space for graphs with millions of edges, while achieving significant improvement in AUC and F1 score on average for the link prediction task in heterogeneous graphs.

Most real-world graph are dynamic where new nodes and edges are added or existing ones are removed continuously over time. One of the limitations of Multi-Lens is that it is designed to handle *static* heterogeneous graphs. Therefore, in Chapter IV, we explored an alternative approach to derive the graph summary, Node2bits, that incorporates the graph structural information and temporal proximity. Node2bits uses the feature histograms to efficiently represent the structural contexts and uses feature-based temporal walks to capture the temporal proximity. Using these walks, Node2bits generates contexts (sequences) of temporally-valid feature values that are further summarized through locality-sensitive hashing. Experiments on real-world networks demonstrated the utility of Node2bits as it outputs space-efficient binary hashcodes as node embeddings that use orders of magnitude less space compared to the baselines while achieving better performance in user stitching.

To better understand how temporal proximity is modeled in recent temporal node embedding approaches, we conducted a systematic study on the properties of temporal network models and their cornerstones, the graph time-series representations in Chapter V. We introduced a general framework that extends the static node embeddings derived from the time-series representation of stream data to dynamic settings using interpretable temporal network models, such as temporal summary graphs (TSG) and temporal reachability graphs (TRG). The temporal summaries in our study consist of graph time-series representations, temporal network models and the embedding composition, all of which are more interpretable than complex methods that model temporal proximity through latent transition or latent variables . Based on the temporal link prediction results using seven base static embedding methods and six temporal network models, we found that many state-of-the-art dynamic embedding approaches do not outperform our framework, and our interpretable temporal summaries are able to capture the graph structures and temporal dependency at least as well as recent dynamic approaches, but less complex.

8.1.2 Non-latent Feature Summarization

In addition to summarizing latent features, this dissertation explored summarizing non-latent features, which are usually more interpretable. Our goal was to generate summaries that are easy to understand (*i.e.*, consisting of known graph features or properties with explicit meanings), while achieving state-of-the art performance in machine learning tasks.

In Chapter VI, we introduced EAGLE, a novel way to summarize a graph using a set of interpretable features, resulting in a diverse, concise, domain-specific, and efficient-to-compute summary. We framed the problem as constrained optimization that is based on feature selection to choose the non-latent graph features to be included in the summary. Our experiments showed that the EAGLE summaries satisfy all the desired properties, outperform alternative approaches for this problem, and are effective in data mining tasks, such as classification, despite not being tailored to it.

The output given by EAGLE can be seen as a "hard" indication of which features are important, *i.e.*, the important features are selected and used, while the unimportant ones are ignored and excluded from the summary. However in reality, the effectiveness of features cannot be simply characterized as "useful" or "not useful", and such hard indication does not model that some features may contribute *more* to the downstream tasks than others.

Therefore, in Chapter VII, we proposed a “soft” way of modeling feature importance via a learned attention mechanism. We tackled the problem of multi-source entity linkage (MEL) and described a deep learning solution based on domain adaptation, AdaMEL. AdaMEL highlights the impact of important attributes in MEL and automatically learns feature importance that adapts to the both seen and unseen data sources as the generic transferable knowledge. We also proposed a series of AdaMEL variants to handle different real-world learning scenarios such as zero-shot learning and few-shot learning. Comparing to heterogeneous schema matching baselines, AdaMEL is able to handle difficult transfer learning cases such as unseen data sources in the target domain and training on weakly-labeled data, while achieving significant average improvement compared to baselines based on supervised learning in PRAUC score for the multi-source entity linkage task. More importantly, we showed that the high-level transferable knowledge is straightforward for humans to interpret.

8.1.3 Histograms in Embedding

Histograms or the distributions of feature values have been an important component of the methods proposed throughout this thesis. Compared to simple aggregators that are commonly used in the literature, such as mean or sum, histogram representation is a more informative and accurate way to capture the structural graph features, we have seen in various settings that node embeddings based on histograms are quite powerful and effective. Throughout this thesis, we have mainly leveraged histograms for three purposes:

To aggregate structural graph feature values in node-centric subgraphs. In Chapter IV, we represented the non-latent structural graph features via histograms, and then conducted latent feature summarization via low-rank approximation. We showed that histograms could enrich the structural feature space by expanding a single value to a sequence (depending on the number of bins in the histogram), and result in node embeddings with improved performance in machine learning tasks, such as link prediction and anomaly detection.

To concisely describe graphs. In Chapter VI, we showed that EAGLE is capable of exploring and summarizing graph data through histogram-represented features such as degree distribution and PageRank distribution. The selected histograms provide accurate graph descriptions that facilitate both human interpretation and machine learning tasks (*e.g.*, graph classification).

To bridge multiple domains. In Chapter VII, we leveraged the difference between feature value distributions from different data sources as the regularization term to learn feature importance as the high-level knowledge. The feature value distributions can be seen as a shared space that bridges both the seen and unseen data sources in order to learn general and source-agnostic high-level knowledge.

All these works have demonstrated the power of feature distributions in node representation learning and graph summarization, and we expect to see more future directions based on histograms.

8.2 Future Directions

This thesis has shown that feature summarization is useful for node embedding in various machine learning and graph mining tasks. We now describe some potential future directions.

Theoretical connections to existing deep learning frameworks. There are several interesting connections between the thesis works and existing deep learning models. In Chapter III, we have shown the power of relational operators/functions in capturing graph structural features. From a high level, these operators can be seen as a generalization of the mean aggregation operation in graph neural networks (GNN) or graph convolutional neural networks (GCNN) [SGT$^+$09, HYL17a, KW17]. Thus an interesting direction is to study the theoretical connection between the relational operations and aggregation operations in GNNs, as well as the connection between relational operator compositions and the feature smoothing process in GNNs. In Chapter VII, we showed that AdaMEL aims to learn the high-level feature importance to address the multi-source entity linkage problem without relying on advanced word token representations. But this does not mean that contextual word representation learning should be excluded from this study. Future directions include combining our work with advanced natural language processing techniques for sequence representation in attribute summarization to further improve the model performance in multi-source entity linkage, and extending our framework to handle faraway data sources, or data sources in different languages.

Extensions to new applications. It would also be interesting to extend our methods to new application domains. For example, in Chapter IV, we showed that Node2bits is efficient in terms of computation and storage. It is worth exploring its performance in other tasks, such as recommendation system or graph alignment. It is also worth exploring how to leverage the findings in Chapter V to design dynamic node embedding approaches in an end-to-end manner by modeling the temporal transition between snapshots with interpretable models.

Modeling deep neural networks as graphs. The success of deep neural networks (NNs) in computer vision and natural language processing has led to increasing interest in understanding the underlying learning dynamics of these models. The representation of many complex NN (*e.g.*, LeNet [LBBH98] or ResNet [HZRS16]) can be seen as graphs with predefined architectures where the nodes and edges represent the neurons and the connections, respectively. Accordingly, the training process of these networks — which aims to minimize the loss on the training data and to achieve the best performance on the holdout set — can be modeled as a dynamic graph with continuously updating edge weights, which is similar to the temporal representations that we investigated in Chapter IV and V. Most of the existing works follow a manually-defined stopping criterion, such as a fixed number of epochs or when the change in loss becomes lower than a threshold. This leads to the research question: can we summarize the dynamics of the training process of NN architectures to get a better understanding of the connection to model performance?

Transfer learning for multi-domain graph classification. As feature summaries encode high-level knowledge that is not specific to nodes or edges in one graph, they can naturally be used for inductive learning or transfer learning. The underlying assumption is that the features from different datasets follow similar patterns. In Chapter III, our method

Multi-Lens derived a latent summary of graph structural features that is independent of the size of the input graph, and we showcased its effectiveness in inductive anomaly detection. In Chapter VII, we showed that the importance of non-latent features can be jointly learned from multiple data sources and improve the performance of entity linkage under the transfer learning scenario. The works in this dissertation follow the above assumption and focus on handling data from the same or a similar domain, such as training the model on a bibliography network and inductively learning the anomalies on another citation network, or transferring the learned feature importance of music recordings from different websites. However, as we showed in Chapter VI, collectively analyzing graph data across different domains may lead to new knowledge discoveries since the domain-associated non-latent features could have different patterns. Therefore, an interesting direction for future work is to transfer the graph-level latent feature summaries of the data from one domain to another for data mining analysis and graph classification. In this process, the graph-level summaries should be capable of aggregating the node embeddings of an individual graph, and general enough to describe graphs from a domain. Deriving feature summaries under such scenario is more challenging and may lead to interesting new representation learning methods and applications.

BIBLIOGRAPHY

BIBLIOGRAPHY

[ABFX08] Edoardo M. Airoldi, David M. Blei, Stephen E. Fienberg, and Eric P. Xing. Mixed membership stochastic blockmodels. *Journal of Machine Learning Research*, 9:1981–2014, 2008.

[ACK$^+$12] Leman Akoglu*, Duen Horng Chau*, U Kang*, Danai Koutra*, and Christos Faloutsos. OPAvion: Mining and Visualization in Large Graphs. In *SIGMOD*, pages 717–720, 2012.

[ARZ$^+$18] Nesreen K. Ahmed, Ryan A. Rossi, Rong Zhou, John Boaz Lee, Xiangnan Kong, Theodore L. Willke, and Hoda Eldardiry. Learning role-based graph embeddings. In *IJCAI StarAI*, 2018.

[AW10] Charu C Aggarwal and Haixun Wang. A survey of clustering algorithms for graph data. In *Managing and mining graph data*, pages 275–301. Springer, 2010.

[Bac08] Francis R Bach. Bolasso: model consistent lasso estimation through the bootstrap. In *ICML*, pages 33–40. ACM, 2008.

[BC08] Gregory Buehrer and Kumar Chellapilla. A scalable pattern mining approach to web graph compression with communities. In *WSDM*, pages 95–106. ACM, 2008.

[Ben12] Yoshua Bengio. Deep learning of representations for unsupervised and transfer learning. In *Proceedings of ICML workshop on unsupervised and transfer learning*, pages 17–36, 2012.

[BF03] Sandeep Bhadra and Afonso Ferreira. Complexity of connected components in evolving graphs and the computation of multicast trees in dynamic networks. In *International Conference on Ad-Hoc Networks and Wireless*, pages 259–270. Springer, 2003.

[BG07] I. Bhattacharya and L. Getoor. Collective entity resolution in relational data. *Transactions on Knowledge Discovery from Data*, 1(1):1–36, 2007.

[BG17] Aleksandar Bojchevski and Stephan Günnemann. Deep gaussian embedding of attributed graphs: Unsupervised inductive learning via ranking. *arXiv:1707.03815*, 2017.

[BGLL08] Vincent D Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. Fast Unfolding of Communities in Large Networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2008(10):P10008, 2008.

[BGMZ97] Andrei Z Broder, Steven C Glassman, Mark S Manasse, and Geoffrey Zweig. Syntactic clustering of the web. *Computer Networks and ISDN Systems*, 29(8):1157–1166, 1997.

[BKMM07] David A. Bader, Shiva Kintali, Kamesh Madduri, and Milena Mihail. Approximating betweenness centrality. In *WAW*, pages 124–137, 2007.

[BM03] Mikhail Bilenko and Raymond J Mooney. Adaptive duplicate detection using learnable string similarity measures. In *KDD*, pages 39–48, 2003.

[BN02] Mikhail Belkin and Partha Niyogi. Laplacian eigenmaps and spectral techniques for embedding and clustering. In *Advances in neural information processing systems*, pages 585–591, 2002.

[Bra06] Matthew Brand. Fast low-rank modifications of the thin singular value decomposition. *Linear algebra and its applications*, 415(1):20–30, 2006.

[BS09] Ed Bullmore and Olaf Sporns. Complex Brain Networks: Graph Theoretical Analysis of Structural and Functional Systems. *Nature Reviews Neuroscience*, 10(3):186–198, 2009.

[CH03] Reuven Cohen and Shlomo Havlin. Scale-free networks are ultrasmall. *Physical review letters*, 90(5):058701, 2003.

[Cha02] Moses S Charikar. Similarity estimation techniques from rounding algorithms. In *STOC*, pages 380–388, 2002.

[Cha07] Sung-Hyuk Cha. Comprehensive survey on distance/similarity measures between probability density functions. *J MMMAS*, 1(2):1, 2007.

[Chr12] Peter Christen. *Concepts and Techniques for Record Linkage, Entity Resolution, and Duplicate Detection.* Springer, 2012.

[CLF$^+$09] Chen Chen, Cindy X Lin, Matt Fredrikson, Mihai Christodorescu, Xifeng Yan, and Jiawei Han. Mining graph patterns efficiently via randomized summaries. *VLDB*, 2(1):742–753, 2009.

[CLX15a] Shaosheng Cao, Wei Lu, and Qiongkai Xu. Grarep: Learning graph representations with global structural information. In *Proceedings of the 24th ACM international on conference on information and knowledge management*, pages 891–900, 2015.

[CLX15b] Shaosheng Cao, Wei Lu, and Qiongkai Xu. Grarep: Learning graph representations with global structural information. In *CIKM*, pages 891–900. ACM, 2015.

[CLX16] Shaosheng Cao, Wei Lu, and Qiongkai Xu. Deep neural networks for learning graph representations. In *AAAI*, pages 1145–1152, 2016.

[CO02] Kerry G Coffman and Andrew M Odlyzko. Growth of the internet. In *Optical fiber telecommunications IV-B*, pages 17–56. Elsevier, 2002.

[cob12] Center for Biomedical Research Excellence. `http://fcon_1000.projects.nitrc.org/indi/retro/cobre.html`, 2012.

[con] Penn dataset. http://www.humanconnectome.org/ccf/.

[CR02] William W Cohen and Jacob Richman. Learning to match and cluster large high-dimensional data sets for data integration. In *KDD*, pages 475–480, 2002.

[CSF$^+$12] Rita Chattopadhyay, Qian Sun, Wei Fan, Ian Davidson, Sethuraman Panchanathan, and Jieping Ye. Multisource domain adaptation and its application to early detection of fatigue. *ACM TKDD*, 6(4):1–26, 2012.

[CZC$^+$17] Sandro Cavallari, Vincent W Zheng, Hongyun Cai, Kevin Chen-Chuan Chang, and Erik Cambria. Learning community embedding with community detection and node embedding on graphs. In *CIKM*, pages 377–386, 2017.

[DCS17] Yuxiao Dong, Nitesh V Chawla, and Ananthram Swami. metapath2vec: Scalable representation learning for heterogeneous networks. In *KDD*, pages 135–144, 2017.

[DDGR07] Abhinandan S. Das, Mayur Datar, Ashutosh Garg, and Shyam Rajaram. Google news personalization: Scalable online collaborative filtering. In *WWW*, pages 271–280, 2007.

[DGZ$^+$12] Anirban Dasgupta, Maxim Gurevich, Liang Zhang, Belle Tseng, and Achint O Thomas. Overcoming browser cookie churn with clustering. In *WSDM*, pages 83–92, 2012.

[DH05] AnHai Doan and Alon Y Halevy. Semantic integration research in the database community: A brief survey. *AI magazine*, 26(1):83–83, 2005.

[DH14] Zhengjia Dai and Yong He. Disrupted structural and functional brain connectomes in mild cognitive impairment and alzheimer's disease. *Neuroscience Bulletin*, 30(2):217–232, 2014.

[di220] di2kg. 2nd international workshop on challenges and experiences from data integration to knowledge graphs. `http://di2kg.inf.uniroma3.it/2020/`, 2020.

[DN09] Xin Luna Dong and Felix Naumann. Data fusion: resolving data conflicts for integration. *VLDB*, 2(2):1654–1655, 2009.

[DS13] Cody Dunne and Ben Shneiderman. Motif simplification: improving network visualization readability with fan, connector, and clique glyphs. In *CHI*, 2013.

[DXT12] Lixin Duan, Dong Xu, and Ivor Wai-Hung Tsang. Domain adaptation from multiple sources: A domain-dependent regularization approach. *IEEE Transactions on neural networks and learning systems*, 23(3):504–518, 2012.

[DZHL18] Claire Donnat, Marinka Zitnik, David Hallac, and Jure Leskovec. Learning structural node embeddings via diffusion wavelets. In *KDD*, volume 24, 2018.

[Eck10] Peter Eckersley. How unique is your web browser? In *International Symposium on Privacy Enhancing Technologies Symposium*, pages 1–18. Springer, 2010.

[EV03] Magdalini Eirinaki and Michalis Vazirgiannis. Web mining for web personalization. *ACM Trans. Internet Technol.*, 3(1):1–27, February 2003.

[FAL17] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *International Conference on Machine Learning*, pages 1126–1135. PMLR, 2017.

[FD81] David Freedman and Persi Diaconis. On the histogram as a density estimator: L 2 theory. *Probability theory and related fields*, 57(4):453–476, 1981.

[FFF99] Michalis Faloutsos, Petros Faloutsos, and Christos Faloutsos. On Power-law Relationships of the Internet Topology. *SIGCOMM*, pages 251–262, 1999.

[FHHS20] Cheng Fu, Xianpei Han, Jiaming He, and Le Sun. Hierarchical matching network for heterogeneous entity resolution. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence*, pages 3665–3671, 2020.

[FJLM09] Wenfei Fan, Xibei Jia, Jianzhong Li, and Shuai Ma. Reasoning about record matching rules. *Proceedings of the VLDB Endowment*, 2(1):407–418, 2009.

[FKV04] Alan Frieze, Ravi Kannan, and Santosh Vempala. Fast monte-carlo algorithms for finding low-rank approximations. *JACM*, 51(6):1025–1041, 2004.

[FZX12] Xu Feng, JC Zhao, and Ke Xu. Link prediction in complex networks: a clustering perspective. *The European Physical Journal B*, 85(1):1–9, 2012.

[GBC16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning.* MIT Press, 2016.

[GCC19] Palash Goyal, Sujit Rokka Chhetri, and Arquimedes Canedo. dyngraph2vec: Capturing network dynamics using dynamic graph representation learning. *Knowledge-Based Systems*, page 104816, 2019.

[GE03] I. Guyon and A. Elisseeff. An introduction to variable and feature selection. *JMLR*, 3:1157–1182, 2003.

[GF18] Palash Goyal and Emilio Ferrara. Graph embedding techniques, applications, and performance: A survey. *Knowledge-Based Systems*, 151:78–94, 2018.

[GH06] Liqiang Geng and Howard J. Hamilton. Interestingness measures for data mining: A survey. *ACM Comput. Surv.*, 38(3), September 2006.

[GKHL18] Palash Goyal, Nitin Kamra, Xinran He, and Yan Liu. Dyngem: Deep embedding method for dynamic graphs. *arXiv preprint arXiv:1805.11273*, 2018.

[GL15] Yaroslav Ganin and Victor Lempitsky. Unsupervised domain adaptation by backpropagation. In *International conference on machine learning*, pages 1180–1189. PMLR, 2015.

[GL16] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *KDD*, pages 855–864, 2016.

[GLT$^+$16] Huan Gui, Jialu Liu, Fangbo Tao, Meng Jiang, Brandon Norick, and Jiawei Han. Large-scale embedding learning in heterogeneous event data. In *ICDM*, pages 907–912. IEEE, 2016.

[GM12] Lise Getoor and Ashwin Machanavajjhala. Entity resolution: theory, practice & open challenges. *Proceedings of the VLDB Endowment*, 5(12):2018–2019, 2012.

[GM13] Lise Getoor and Ashwin Machanavajjhala. Entity resolution for big data. In *KDD*, pages 1527–1527, 2013.

[HERPF10] Keith Henderson, Tina Eliassi-Rad, Spiros Papadimitriou, and Christos Faloutsos. Hcdf: A hybrid community discovery framework. In *SDM*, pages 754–765. SIAM, 2010.

[HGDG17] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *Proceedings of the IEEE ICCV*, pages 2961–2969, 2017.

[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, 2012.

[His16] Ryohei Hisano. Semi-supervised graph embedding approach to dynamic link prediction, 2016.

[His18] Ryohei Hisano. Semi-supervised graph embedding approach to dynamic link prediction. In *International Workshop on Complex Networks*, pages 109–121. Springer, 2018.

[HSSK18] Mark Heimann, Haoming Shen, Tara Safavi, and Danai Koutra. Regal: Representation learning-based graph alignment. In *CIKM*, 2018.

[HYL17a] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *NIPS*, pages 1024–1034, 2017.

[HYL17b] William L Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pages 1025–1035, 2017.

[HYL17c] William L Hamilton, Rex Ying, and Jure Leskovec. Representation learning on graphs: Methods and applications. *arXiv preprint arXiv:1709.05584*, 2017.

[HZRS16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[IM98] Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *STOC*, pages 604–613, 1998.

[JCB$^+$14] Meng Jiang, Peng Cui, Alex Beutel, Christos Faloutsos, and Shiqiang Yang. Catchsync: Catching synchronized behavior in large directed graphs. In *KDD*, pages 941–950, 2014.

[JGBM17] Armand Joulin, Édouard Grave, Piotr Bojanowski, and Tomáš Mikolov. Bag of tricks for efficient text classification. In *Proceedings of the 15th Conference of the European Chapter of ACL: Volume 2, Short Papers*, pages 427–431, 2017.

[JHJK21] Junchen Jin, Mark Heimann, Di Jin, and Danai Koutra. Towards understanding and evaluating structural node embeddings. *arXiv preprint arXiv:2101.05730*, 2021.

[JHRK19] Di Jin, Mark Heimann, Ryan Rossi, and Danai Koutra. node2bits: Compact time-and attribute-aware node representations for user stitching. *arXiv preprint arXiv:1904.08572*, 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 *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2923–2933, 2019.

[JK17] Di Jin and Danai Koutra. Exploratory analysis of graph data by leveraging domain knowledge. In *ICDM*, pages 187–196, 2017.

[JKRK22] Di Jin, Sungchul Kim, Ryan A Rossi, and Danai Koutra. From static to dynamic node embeddings. *arXiv preprint arXiv:2009.10017*, 2022.

[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 *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 987–997, 2019.

[JSW$^+$22] Di Jin, Bunyamin Sisman, Hao Wei, Xin Luna Dong, and Danai Koutra. Deep transfer learning for multi-source entity linkage via domain adaptation. *arXiv preprint arXiv:2110.14509*, 2022.

[JT18] Muhammad Ebraheem Saravanan Thirumuruganathan Shafiq Joty and Mourad Ouzzani Nan Tang. Distributed representations of tuples for entity resolution. *Proceedings of the VLDB Endowment*, 11(11), 2018.

[KB14] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[KBB17] Arijit Khan, Sourav S Bhowmick, and Francesco Bonchi. Summarizing static and dynamic big graphs. *VLDB*, 10(12):1981–1984, 2017.

[Kel60] James E Kelley, Jr. The cutting-plane method for solving convex programs. *J Appl Math*, 8(4):703–712, 1960.

[KJNF15] Danai Koutra, Di Jin, Yuanchi Ning, and Christos Faloutsos. Perseus: an interactive large-scale graph mining and visualization tool. *VLDB Endowment*, 8(12):1924–1927, 2015.

[KKP$^+$17] Sungchul Kim, Nikhil Kini, Jay Pujara, Eunyee Koh, and Lise Getoor. Probabilistic visitor stitching on cross-device web logs. In *WWW*, pages 1581–1589, 2017.

[KKVF14] Danai Koutra, U Kang, Jilles Vreeken, and Christos Faloutsos. VoG: Summarizing and Understanding Large Graphs. In *SDM*. SIAM, 2014.

[KKVF15] Danai Koutra, U Kang, Jilles Vreeken, and Christos Faloutsos. Summarizing and understanding large graphs. *Statistical Analysis and Data Mining: The ASA Data Science Journal*, 8(3):183–202, 2015.

[KQG$^+$19] Jungo Kasai, Kun Qian, Sairam Gurajada, Yunyao Li, and Lucian Popa. Low-resource deep entity resolution with transfer and active learning. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 5851–5861, 2019.

[KR10] Hanna Köpcke and Erhard Rahm. Frameworks for entity matching: A comparison. *Data & Knowledge Engineering*, 69(2):197–210, 2010.

[KT19] Jacob Devlin Ming-Wei Chang Kenton and Lee Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of NAACL-HLT*, pages 4171–4186, 2019.

[KTR12] Lars Kolb, Andreas Thor, and Erhard Rahm. Dedoop: Efficient deduplication with hadoop. *VLDB*, 5(12):1878–1881, August 2012.

[KVF13] Danai Koutra, Joshua T Vogelstein, and Christos Faloutsos. Deltacon: A principled massive-graph similarity function. In *SDM*, pages 162–170. SIAM, 2013.

[KW17] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *ICLR*, 2017.

[KZL19] Srijan Kumar, Xikun Zhang, and Jure Leskovec. Predicting dynamic embedding trajectory in temporal interaction networks. In *KDD*, pages 1269–1278. ACM, 2019.

[LBBH98] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[LDH$^+$17] Jundong Li, Harsh Dani, Xia Hu, Jiliang Tang, Yi Chang, and Huan Liu. Attributed network embedding for learning in a dynamic environment. In *CIKM*, pages 387–396. ACM, 2017.

[LDSK16] Yike Liu, Abhilash Dighe, Tara Safavi, and Danai Koutra. Graph Summarization: A Survey. *CoRR*, abs/1612.04883, 2016.

[LG14] Omer Levy and Yoav Goldberg. Neural word embedding as implicit matrix factorization. In *NIPS*, pages 2177–2185, 2014.

[LHH$^+$14] Fenhua Li, Jing He, Guangyan Huang, Yanchun Zhang, and Yong Shi. A clustering-based link prediction method in social networks. *Procedia Computer Science*, 29:432–442, 2014.

[LK14] Jure Leskovec and Andrej Krevl. SNAP Datasets: Stanford large network dataset collection. http://snap.stanford.edu/data, June 2014.

[LKF05] J. Leskovec, J. Kleinberg, and C. Faloutsos. Graphs over time: densification laws, shrinking diameters and possible explanations. In *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, pages 177–187. ACM, 2005.

[LKF14] Yongsub Lim, U Kang, and Christos Faloutsos. Slashburn: Graph compression and mining beyond caveman communities. *TKDE*, 26(12):3077–3089, 2014.

[LL09] Cheng-Te Li and Shou-De Lin. Egocentric information abstraction for heterogeneous social networks. In *ASONAM*, pages 255–260. IEEE, 2009.

[LLS$^+$20] Yuliang Li, Jinfeng Li, Yoshihiko Suhara, AnHai Doan, and Wang-Chiew Tan. Deep entity matching with pre-trained language models. *Proceedings of the VLDB Endowment*, 14(1):50–60, 2020.

[LPM15] Minh-Thang Luong, Hieu Pham, and Christopher D Manning. Effective approaches to attention-based neural machine translation. In *Proceedings of the 2015 Conference on EMNLP*, pages 1412–1421, 2015.

[LSDK18] Yike Liu, Tara Safavi, Abhilash Dighe, and Danai Koutra. Graph summarization methods and applications: A survey. *CSUR*, 51(3):62, 2018.

[LSK15] Yike Liu, Neil Shah, and Danai Koutra. An empirical comparison of the summarization power of graph clustering methods. *arXiv preprint arXiv:1511.06820*, 2015.

[LYL13] Shou-De Lin, Mi-Yen Yeh, and Cheng-Te Li. Sampling and summarization for social networks. In *SDM*, 2013.

[LZS$^+$19] Yike Liu, Linhong Zhu, Pedro Szekely, Aram Galstyan, and Danai Koutra. Coupled clustering of time-series and networks. In *SDM*, 2019.

[MA16] Antonio Maccioni and Daniel J Abadi. Scalable pattern matching over compressed graphs via dedensification. In *SIGKDD*, pages 1755–1764. ACM, 2016.

[MBWB15] Benjamin A Miller, Michelle S Beard, Patrick J Wolfe, and Nadya T Bliss. A spectral framework for anomalous subgraph detection. *IEEE TSP*, 63(16):4191–4206, 2015.

[MH08] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(Nov):2579–2605, 2008.

[MLR$^+$18] Sidharth Mudgal, Han Li, Theodoros Rekatsinas, AnHai Doan, Youngchoon Park, Ganesh Krishnan, Rohit Deep, Esteban Arcaute, and Vijay Raghavendra. Deep learning for entity matching: A design space exploration. In *Proceedings of the 2018 International Conference on Management of Data*, pages 19–34, 2018.

[NHH$^+$19] Hao Nie, Xianpei Han, Ben He, Le Sun, Bo Chen, Wei Zhang, Suhui Wu, and Hao Kong. Deep sequence-to-sequence entity matching for heterogeneous entity resolution. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, pages 629–638, 2019.

[NLR$^+$18] Giang Hoang Nguyen, John Boaz Lee, Ryan A. Rossi, Nesreen K. Ahmed, Eunyee Koh, and Sungchul Kim. Continuous-time dynamic network embeddings. In *WWW BigNet*, 2018.

[NRS08] Saket Navlakha, Rajeev Rastogi, and Nisheeth Shrivastava. Graph summarization with bounded error. In *SIGMOD*, pages 419–432, 2008.

[PARS14] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. In *KDD*, 2014.

[PDC$^+$20] Aldo Pareja, Giacomo Domeniconi, Jie Chen, Tengfei Ma, Toyotaro Suzumura, Hiroki Kanezashi, Tim Kaler, Tao Schardl, and Charles Leiserson. Evolvegcn: Evolving graph convolutional networks for dynamic graphs. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 5363–5370, 2020.

[PL05] Pascal Pons and Matthieu Latapy. Computing communities in large networks using random walks. In *International symposium on computer and information sciences*, pages 284–293. Springer, 2005.

[PLD05] Hanchuan Peng, Fuhui Long, and Chris Ding. Feature selection based on mutual information criteria of max-dependency, max-relevance, and min-redundancy. *IEEE TPAMI*, 27(8):1226–1238, 2005.

[PPHM09] Mark Palatucci, Dean Pomerleau, Geoffrey E Hinton, and Tom M Mitchell. Zero-shot learning with semantic output codes. *NIPS*, pages 1410–1418, 2009.

[PSGP16] George Papadakis, Jonathan Svirsky, Avigdor Gal, and Themis Palpanas. Comparative analysis of approximate blocking techniques for entity resolution. *VLDB*, 9(9):684–695, 2016.

[PY09] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359, 2009.

[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*, pages 459–467, 2018.

[QPS17] Kun Qian, Lucian Popa, and Prithviraj Sen. Active learning for large-scale entity resolution. In *Proceedings of the 2017 ACM CIKM*, pages 1379–1388, 2017.

[RA15a] Ryan A. Rossi and Nesreen K. Ahmed. The network data repository with interactive graph analytics and visualization. In *AAAI*, 2015.

[RA15b] Ryan A. Rossi and Nesreen K. Ahmed. Role discovery in networks. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 27(4):1112–1131, April 2015.

[RA15c] Ryan A. Rossi and Nesreen K. Ahmed. Role discovery in networks. *TKDE*, 27(4):1112–1131, 2015.

[RAEZ18] Ryan A. Rossi, Nesreen K. Ahmed, Hoda Eldardiry, and Rong Zhou. Similarity-based multi-label learning. In *IJCNN*, pages 1–8, 2018.

[RAK$^+$18] Ryan A. Rossi, Nesreen K. Ahmed, Eunyee Koh, Sungchul Kim, Anup Rao, and Yasin Abbasi-Yadkori. HONE: Higher-Order Network Embeddings. *WWW*, 2018.

[RGM03] Sriram Raghavan and Hector Garcia-Molina. Representing web graphs. In *ICDE*, pages 405–416, 2003.

[RGNH13] Ryan A. Rossi, Brian Gallagher, Jennifer Neville, and Keith Henderson. Modeling dynamic behavior in large evolving graphs. In *Proceedings of the Sixth ACM International Conference on Web Search and Data Mining*, pages 667–676, 2013.

[RGSB17] Matteo Riondato, David García-Soriano, and Francesco Bonchi. Graph summarization with quality guarantees. *DMKD*, 31(2):314–349, 2017.

[RJK$^+$19] Ryan A Rossi, Di Jin, Sungchul Kim, Nesreen K Ahmed, Danai Koutra, and John Boaz Lee. From community to role-based graph embeddings. *arXiv e-prints*, pages arXiv–1908, 2019.

[RLU14] Anand Rajaraman, Jure Leskovec, and Jeffrey D. Ullman. *Mining Massive Datasets*. 2014.

[RN12] Ryan A. Rossi and Jennifer Neville. Time-evolving relational classification and ensemble methods. In *Advances in Knowledge Discovery and Data Mining*, volume 7301, pages 1–13. Springer, 2012.

[RSF17] Leonardo F.R. Ribeiro, Pedro H.P. Saverese, and Daniel R. Figueiredo. Struc2vec: Learning node representations from structural identity. In *SIGKDD*, 2017.

[RZA18] Ryan A. Rossi, Rong Zhou, and Nesreen K. Ahmed. Deep inductive network representation learning. In *WWW BigNet*, 2018.

[Sch07] Satu Elisa Schaeffer. Graph clustering. *Computer science review*, 1(1):27–64, 2007.

[Sco79] David W Scott. On optimal and data-based histograms. *Biometrika*, pages 605–610, 1979.

[SGR19] Uriel Singer, Ido Guy, and Kira Radinsky. Node embedding over temporal graphs. In *IJCAI*, pages 4605–4612, 7 2019.

[SGT$^+$09] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2009.

[SGZ$^+$18] Yu Shi, Huan Gui, Qi Zhu, Lance Kaplan, and Jiawei Han. Aspem: Embedding learning by aspects in heterogeneous information networks. In *SDM*, pages 144–152. SIAM, 2018.

[SKW$^+$13] Chandra Sekhar Sripada, Daniel Kessler, Robert Welsh, Michael Angstadt, Israel Liberzon, K Luan Phan, and Clayton Scott. Distributed effects of methylphenidate on the network structure of the resting brain: a connectomic pattern classification analysis. *Neuroimage*, 81:213–221, 2013.

[SKZ$^+$15] Neil Shah, Danai Koutra, Tianmin Zou, Brian Gallagher, and Christos Faloutsos. Timecrunch: Interpretable dynamic graph summarization. In *KDD*, pages 1055–1064, 2015.

[SME$^+$17] Rohit Singh, Venkata Vamsikrishna Meduri, Ahmed Elmagarmid, Samuel Madden, Paolo Papotti, Jorge-Arnulfo Quiané-Ruiz, Armando Solar-Lezama, and Nan Tang. Synthesizing entity matching rules by examples. *VLDB*, 11(2):189–202, 2017.

[SMER06] Zeqian Shen, Kwan-Liu Ma, and Tina Eliassi-Rad. Visual analysis of large heterogeneous social networks by semantic and structural abstraction. *TVCG*, 12(6):1427–1439, 2006.

[SNA] SNAP. http://snap.stanford.edu/data/index.html#web.

[SRSCS15] Rishiraj Saha Roy, Ritwik Sinha, Niyati Chhaya, and Shiv Saini. Probabilistic deduplication of anonymous web traffic. In *WWW*, pages 103–104, 2015.

[SRWS08] Gabriele Schweikert, Gunnar Rätsch, Christian Widmer, and Bernhard Schölkopf. An empirical analysis of domain adaptation algorithms for genomic sequence analysis. *Advances in neural information processing systems*, 21:1433–1440, 2008.

[SSB$^+$15] Lin Shao, Timo Schleicher, Michael Behrisch, Tobias Schreck, Ivan Sipiran, and Daniel A. Keim. Guiding the exploration of scatter plot data using motif-based interest measures. In *BDVA*, pages 1–8, 2015.

[SSTZ12] Parikshit Sondhi, Jimeng Sun, Hanghang Tong, and ChengXiang Zhai. SympGraph: a framework for mining clinical notes through symptom relation graphs. In *KDD*, pages 1167–1175, 2012.

[SSW15] Shiliang Sun, Honglei Shi, and Yuanbin Wu. A survey of multi-source domain adaptation. *Information Fusion*, 24:84–92, 2015.

[Stu26] Herbert A Sturges. The choice of a class interval. *Journal of the American Statistical Association*, 21(153):65–66, 1926.

[SWD16] Qi Song, Yinghui Wu, and Xin Luna Dong. Mining summaries for knowledge graph search. In *ICDM*, pages 1215–1220, 2016.

[SWG$^+$20] Aravind Sankar, Yanhong Wu, Liang Gou, Wei Zhang, and Hao Yang. Dysat: Deep neural representation learning on dynamic graphs via self-attention networks. In *WSDM*, pages 519–527, 2020.

[TDSL00] Joshua B Tenenbaum, Vin De Silva, and John C Langford. A global geometric framework for nonlinear dimensionality reduction. *science*, 290(5500):2319–2323, 2000.

[THP08] Yuanyuan Tian, Richard A Hankins, and Jignesh M Patel. Efficient aggregation for graph summarization. In *SIGMOD*, pages 567–580. ACM, 2008.

[TL11] Lei Tang and Huan Liu. Leveraging social media networks for classification. *DMKD*, 23(3):447–478, 2011.

[TPO$^+$18] Saravanan Thirumuruganathan, Shameem A Puthiya Parambath, Mourad Ouzzani, Nan Tang, and Shafiq Joty. Reuse and adaptation for entity resolution through transfer learning. *arXiv preprint arXiv:1809.11084*, 2018.

[TQW+15a] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. Line: Large-scale information network embedding. In *WWW*, pages 1067–1077, 2015.

[TQW+15b] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. Line: Large-scale information network embedding. In *WWW*, 2015.

[TZHH11] Hannu Toivonen, Fang Zhou, Aleksi Hartikainen, and Atte Hinkka. Compression of weighted graphs. In *SIGKDD*, pages 965–973. ACM, 2011.

[VCC+18] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *ICLR*, 2018.

[vL07] U. von Luxburg. A tutorial on spectral clustering. *Statistics and Computing*, 17(4):395–416, 2007.

[VSP+17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in NIPS*, pages 5998–6008, 2017.

[WAG05] Leland Wilkinson, Anushka Anand, and Robert L Grossman. Graph-theoretic scagnostics. In *INFOVIS*, volume 5, page 21, 2005.

[WC20] Garrett Wilson and Diane J Cook. A survey of unsupervised deep domain adaptation. *ACM TIST*, 11(5):1–46, 2020.

[WSW+20] Zhengyang Wang, Bunyamin Sisman, Hao Wei, Xin Luna Dong, and Shuiwang Ji. Cordel: A contrastive deep learning approach for entity linkage. *arXiv preprint arXiv:2009.07203*, 2020.

[WWGW18] Qixiang Wang, Shanfeng Wang, Maoguo Gong, and Yue Wu. Feature hashing for network representation learning. In *IJCAI*, pages 2812–2818, 2018.

[XYFS07] Xiaowei Xu, Nurcan Yuruk, Zhidan Feng, and Thomas AJ Schweiger. Scan: a structural clustering algorithm for networks. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 824–833. ACM, 2007.

[XYJ+20] Hansheng Xue, Luwei Yang, Wen Jiang, Yi Wei, Yi Hu, and Yu Lin. Modeling dynamic heterogeneous network forlink prediction using hierarchical attention-with temporal rnn. In *Proceedings of the 2020 European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML-PKDD)*, 2020.

[YP97] Yiming Yang and Jan O Pedersen. A comparative study on feature selection in text categorization. In *ICML*, volume 97, pages 412–420, 1997.

[YZHY18] Wei Ying, Yu Zhang, Junzhou Huang, and Qiang Yang. Transfer learning via learning to transfer. In *ICML*, pages 5085–5094, 2018.

[ZCP$^+$18] Ziwei Zhang, Peng Cui, Jian Pei, Xiao Wang, and Wenwu Zhu. Timers: Error-bounded svd restart on dynamic networks. In *AAAI*, 2018.

[ZGGS$^+$16] Linhong Zhu, Majid Ghasemi-Gol, Pedro Szekely, Aram Galstyan, and Craig A Knoblock. Unsupervised entity resolution on multi-type graphs. In *ISWC*, pages 649–667. Springer, 2016.

[ZGY$^+$16] Linhong Zhu, Dong Guo, Junming Yin, Greg Ver Steeg, and Aram Galstyan. Scalable temporal latent space inference for link prediction in dynamic social networks. *IEEE TKDE*, 28(10):2765–2777, 2016.

[ZH19] Chen Zhao and Yeye He. Auto-em: End-to-end fuzzy entity-matching using pre-trained deep models and transfer learning. In *WWW*, pages 2413–2424, 2019.

[ZLL$^+$18] Yuan Zuo, Guannan Liu, Hao Lin, Jia Guo, Xiaoqian Hu, and Junjie Wu. Embedding temporal network via neighborhood formation. In *KDD*, pages 2857–2866, 2018.

[ZTP10] Ning Zhang, Yuanyuan Tian, and Jignesh M Patel. Discovery-driven graph summarization. In *ICDE*, pages 880–891. IEEE, 2010.

[ZYR$^+$18] Lekui Zhou, Yang Yang, Xiang Ren, Fei Wu, and Yueting Zhuang. Dynamic network embedding by modeling triadic closure process. In *AAAI*, 2018.