5-2018

Latent Representation and Sampling in Network: Application in Text Mining and Biology.

Tanay Kumar Saha
Purdue University

LATENT REPRESENTATION AND SAMPLING IN NETWORK: APPLICATION IN TEXT MINING AND BIOLOGY

A Dissertation

Submitted to the Faculty

of

Purdue University

by

Tanay Kumar Saha

In Partial Fulfillment of the

Requirements for the Degree

of

Doctor of Philosophy

May 2018

Purdue University

West Lafayette, Indiana

THE PURDUE UNIVERSITY GRADUATE SCHOOL
STATEMENT OF DISSERTATION APPROVAL

Dr. Mohammad Al Hasan, Co-Chair

Department of Computer Science, Purdue University Indianapolis

Dr. Jennifer Neville, Co-Chair

Department of Computer Science, Purdue University West Lafayette

Dr. David Gleich

Department of Computer Science, Purdue University West Lafayette

Dr. Xia Ning

Department of Computer Science, Purdue University Indianapolis

Dr. Alex Pothen

Department of Computer Science, Purdue University West Lafayette

Approved by:

Dr. Voicu S. Popescu by Dr. William J. Gorman

Head of the Graduate Program

To my parents and grandparents for their unconditional love, and patience.

ACKNOWLEDGMENTS

First and foremost I want to thank my PhD advisor Dr. Mohammad Al Hasan. It has been an honor to be his student. He has taught me how to pick important problems, think critically and independently about the problem, and how to design good experiments to validate the ideas. He also gave me freedom to investigate various research domains, and collaborate with many researchers from industry and academia. Under his supervision, I have worked on various important domains and tasks which made my PhD journey challenging, but engaging. I appreciate all his contributions of time, ideas, and funding to make my PhD experience productive and stimulating. But, mostly I am grateful to him for understanding my excitement and frustrations, and guide me accordingly. I also learned a lot by taking his graduate level data mining and algorithm course, and serving as a teaching assistant for his data mining and artificial intelligence course.

Next I like to pay my gratitude to my co-advisor Dr. Jennifer Neville for her guidance, and encouragement. I appreciate all her contributions of time and ideas. I am also indebted to my thesis committee members: Dr. David Gleich, Dr. Alex Pothen, and Dr. Xia Ning for their time, and suggestions to improve my thesis work. I also like to thank Dr. William Gorman for his continuous effort and timely communication to help me on various occasions.

I also would like to thank Dr. Mourad Ouzzani, Dr. Sanjay Chawla, Dr. Fabrizio Sebastiani, and Dr. Ahmed K. Elmagarmid from Qatar Computing Research Institute (QCRI) for the stimulating discussion for designing a light-weight abstract screening platform, Rayyan. Rayyan has now 10000 users from different parts of the world. I also appreciate Dr. Shafiq Joty who is now in Nanyang Technological University (NTU) for contributing part of my thesis. The collaboration started when both of us were working for QCRI. We both were interested in exploring options for incorporating

linguistic information while learning the embedding and improve the representation of textual, and network units. I am grateful to him for all his time, ideas, and motivating discussions in the field of representation learning, natural language processing, and neural networks.

I like to thank Dr. Ataur Katebi for introducting me to the problem of finding functional motif from biological network and also to Dr. Dhifli Wajdi from University of Lille for his contribution to collect a dataset for the work. Both of them contributed generously in discussion for the project. Thanks to Dhifli and Katebi for their time.

I also like to thank Dr. Feng Li from School of Engineering and Technology, IUPUI for introducing me the Android Malware Detection Project. The project was interesting to me cause the project uses one of my previously developed algorithm. Thanks Dr. Li for his time and interest in my work.

Also, I like to thank Jianwu, Hui, and Pranay from NEC labs at Princeton, NJ for giving me the opportunity to work with them during Summer 2017. Under their support and guidance, I was able to provide a semantic analysis framework for root cause analysis in the heterogeneous log analysis domain. I learned a lot from the weekly hour-wide discussion to design the product.

I also would like to thank faculty members of the Department of Computer and Information Science, Indiana University Purdue University Indianapolis (IUPUI). I learned a lot from them in the past six years. I appreciate Dr. Murat Dundar for his Machine Leanring course. I would also like to thank Dr. Xia Ning for serving as a committee member in my oral qualifier exam, preliminary exam, and final dissertation exam. I thank her for involvement in every step of my PhD journey. I also like to thank Dr. Rajeev R. Raje for all the friendly discussion about various topics.

I am also indebted to IUPUI stuff members Nicole, Nancy, and Joan for their continuous and whole hearted effort to make me feel welcome in the department from the start of my PhD journey. Additionally, I like to thank Purdue stuff members Sandra, Kelsey, and Joey for their support in various occasion which made it easier for me to go through the Purdue system quickly.

Finally, I am grateful to my parents, younger brother, friends, and my lab-mates for their support. Thanks to my parents for raising me with the love of learning and being supportive in all my pursuits. Thanks to my close friends Jesun Sahariar Firoz and Kishwar Ahmed for their companionship which have been my greatest motivation to complete my PhD degree.

TABLE OF CONTENTS

Page

Page

Page

Page

Page

LIST OF TABLES

Table Page

LIST OF FIGURES

Figure Page

Figure Page

Figure Page

Figure Page

ABSTRACT

Tanay Kumar Saha Ph.D., Purdue University, May 2018. Latent Representation and Sampling in Network: Application in Text Mining and Biology. Major Professor: Mohammad Al Hasan.

In classical machine learning, hand-designed features are used for learning a mapping from raw data. However, human involvement in feature design makes the process expensive. Representation learning aims to learn abstract features directly from data without direct human involvement. Raw data can be of various forms. Network is one form of data that encodes relational structure in many real-world domains. Therefore, learning abstract features for network units is an important task.

In this dissertation, we propose models for incorporating temporal information given as a collection of networks from subsequent time-stamps. The primary objective of our models is to learn a better abstract feature representation of nodes and edges in an evolving network. We show that the temporal information in the abstract feature improves the performance of link prediction task substantially.

Besides applying to the network data, we also employ our models to incorporate extra-sentential information in the text domain for learning better representation of sentences. We build a context network of sentences to capture extra-sentential information. This information in abstract feature representation of sentences improves various text-mining tasks substantially over a set of baseline methods.

A problem with the abstract features that we learn is that they lack interpretability. In real-life applications on network data, for some tasks, it is crucial to learn interpretable features in the form of graphical structures. For this we need to mine important graphical structures along with their frequency statistics from the input dataset. However, exact algorithms for these tasks are computationally expensive,

so scalable algorithms are of urgent need. To overcome this challenge, we provide efficient sampling algorithms for mining higher-order structures from network(s). We show that our sampling-based algorithms are scalable. They are also superior to a set of baseline algorithms in terms of retrieving important graphical sub-structures, and collecting their frequency statistics.

Finally, we show that we can use these frequent subgraph statistics and structures as features in various real-life applications. We show one application in biology and another in security. In both cases, we show that the structures and their statistics significantly improve the performance of knowledge discovery tasks in these domains.

1. INTRODUCTION

In classical machine learning, hand-designed features are used for learning a mapping from raw data. The raw data can be structured such as, networks or un-structured such as, text. However, human involvement in feature design makes the process expensive and time-consuming. With the availability of large amount of data, there exist a plethora of research works in learning abstract representation. The objective of these works is to learn abstract features (latent features) from the data without direct human supervision. This particular area is known as unsupervised feature learning. The models for unsupervised feature learning are very effective for capturing hidden relationship among the atomic units (such as nodes and edges in a network) of data and improving the performance of various downstream tasks in both the structured and unstructured data representations [1].

Networks are fundamental data structures used for compactly capturing the relational structure in many important real-world domains, such as biology, social, language, and security. Network data consists of nodes and edges. Nodes represent entities and edges represent the relationship among the entities. There are plenty of data around us which can be represented as a network. An important fact of these networks is that they change over time; for example, in social networks, relationship among people changes in different phases of life. Anatomical activities among the regions of human brain are also not static rather dynamic in nature [2]. However, until recently, most of the representation learning works consider a network as static [3]. So, providing models for learning the representation of various network units such as, nodes and edges that can capture both the temporal connectivity structure and the higher order relation among the nodes is an important research problem. A solution to this problem can be used for explaining the dynamics of an evolving network, and also for solving various prediction tasks in a dynamic network.

Understanding the dynamics of temporal evolution of networks can help solve complex tasks involving social and interaction networks. For instance, capturing temporal dynamics of user interactions can explain how communities are formed and dissolved in a network over time. Temporal co-movement of financial asset prices explains how financial assets are clustered pronouncedly during an economic downturn, causing a cascading effect that leads to a financial crisis. Temporal network models can explain the way social network topologies facilitate (or inhibit) grievances to intensify collective organization, leading to imminent crisis and conflict in a community [4]. So, models that can incorporate both the temporal and network proximity information are of enormous importance.

In evolving network, for any given time-stamp, we have relationship structures among the entities in that timestamp. Additionally, we have a collection of networks for the remaining timestamps as external information. Likewise, in text domain, we have words, phrases, sentences, paragraphs, and documents as content and external information about the content are often can be presented, or constructed as a network. For example, synonymy, hypernymy, and hyponymy of words are encoded in semantic lexicons (like WordNet or Framenet) in the form of networks. This semantic lexicons can be used as an external information while learning the representation of words.

Recent studies [5–7] on learning distributed representations for *words* have shown that lexicons like WordNet [8] or Framenet [9] can improve the quality of word vectors that are trained solely on unlabeled text data. Though the external information is readily available for the words, it is not the case for the sentences. But, sentences rarely stand on their own in a well-formed text. On a finer level, sentences are connected with each other by certain logical relations (e.g., *elaboration*, *contrast*) to express the meaning as a whole [10]. On a coarser level, sentences in a text address a common topic, often covering multiple subtopics [11]. Therefore, constructing context network of sentences and models for incorporating external information in the sentence representation to improve various information retrieval tasks is important.

Unfortunately, features learned from latent representation models lack interpretability. For some tasks, such as, finding functional motifs from a set of biological networks or for classifying mobile apps (given as function call graphs) as malignant or benign, it is critical to learn interpretable features. To learn interpretable features we may need to (*i*) collect statistics (frequency, concentration) about higher-order structures (involving more than a single node or an edge) such as, a substructure containing fixed number of nodes, as well as, (*ii*) mine frequent substructures in a single large network or in a set of networks.

In this thesis, we have developed machine learning models for learning representation of dynamic networks incorporating temporal and network proximity. We have also proposed models for learning sentence representation, where external information regarding the sentences is provided as a network. Finally, we have proposed efficient methods for computing interpretable feature representation of a network, which can be used for building interpretable machine learning models. For each of these tasks, my proposed solution is novel in its approach, and methodology, and it substantially improves the existing state-of-the-art methodologies in terms of prediction performance and computation cost.

In the following subsections, we provide a short overview of the technical aspects of my proposed solutions. First, we briefly discuss our models for learning latent representation of nodes and edges in an evolving network. Second, we present an overview of our models for learning latent representation of sentences. Finally, we discuss the challenges of substructure mining from large networks and my approach for overcoming the challenges, along with a discussion of real-life applications where my proposed substructure mining methods exert enormous impact.

1.1 Models for Latent Representation of Nodes and Edges in an Evolving Network

Learning latent representation for nodes and edges in an evolving network involves modeling both the temporal dynamics of network and also the network proximity information encoded in the structure. Most of the existing latent representation learning techniques for network units (such as, nodes and edges) consider only the network topology [12–14], a few consider nodal attributes and topology [15], and almost all ignore the temporal evolution of a network. In [16], we propose latent representation learning models for nodes in a dynamic network which overcome the above limitation by considering two different kinds of temporal smoothness: (*i*) retrofitted, and (*ii*) linear transformation.

The retrofitted model tracks the representation vector of a vertex over time, facilitating vertex-based temporal analysis of a network. On the other hand, linear transformation based model provides a smooth transition operator which maps the representation vectors of all vertices from one temporal snapshot to the next (unobserved) snapshot—this facilitates prediction of the state of a network in a future time-stamp. Even though our models are task agnostic, we show that our models perform substantially better than the latent representation methods which capture both the temporal and network proximity information in the link prediction task.

The task of link prediction is to predict the link state of the network at a future time given a collection of link states at earlier time points. This is a critical task along the understanding of dynamic network. Additionally, solving link prediction task in an evolving network is more difficult than its counterpart in a static network because an effective feature representation of node-pair instances (edges) for the case of a dynamic network is hard to obtain. If we restrict our attention only to link prediction not to learn representation of nodes in each snapshot, then directly learning the representation of edges can help us improve the task performance.

In [17], we propose DyLink2Vec[1] to overcome this problem. Our method for metric embedding of node-pair models the metric embedding task as an optimal coding problem where the objective is to minimize the reconstruction error, and it solves this optimization task using a gradient descent method. We validate the effectiveness of the learned feature representation by utilizing it for link prediction in various real-life dynamic networks. Specifically, we show that our proposed link prediction model, which uses the extracted feature representation for the training instances, outperforms several existing methods that use well-known link prediction features.

1.2 Models for Latent Representation of Sentences

The retrofitted model that we develop earlier for capturing the temporal smoothness can be used for incorporating extra-sentential context for learning better latent representation of sentences. The learned representation can be used in various data mining tasks, such as classification, clustering, summarization, and many others. Recent studies on learning distributed representations for *words* have shown that semantic relations between words (e.g., synonymy, hypernymy, hyponymy) encoded in semantic lexicons (like WordNet or Framenet) can be used as an external information to improve the quality of the word vectors that are trained solely on unlabeled data.

Our sentence representation learning techniques [18,19] are reminiscent of this line of research because we give models to incorporate extra sentential context for learning representation of sentences. However, in terms of learning problem we have a couple of crucial differences. First, we are interested in the representation of sentences as opposed to words, for the former such resources are not readily available. Second, our main goal is to incorporate extra-sentential context in some form of inter-sentence relations as opposed to semantic relations between words. These differences posit many new research challenges: (*i*) how can we obtain extra-sentential context that

[1] DyLink2Vec stands for **Link to Vec**tor in a **Dy**namic network

can capture semantic relations between sentences? (*ii*) how can we effectively exploit the inter-sentence relations in our representation learning model?

To solve the first problem we introduce the concept of context-network for sentences and for solving the second problem we propose retrofitting and regularizing models using the context network [18] as well as through a joint model [19] which predicts and regularize based on the context network of sentences. The joint model is generic in terms of context and different modes of data the model can handle. By employing our models, we show significant improvement over a set of baselines in the topic classification, clustering, and summarization tasks.

1.3 Methods for Frequent Subgraph Mining and Its Applications

Frequent subgraph mining (FSM) which mines frequent substructures from a set of networks is an important research task in Network Mining Domain. It has application in various disciplines, including cheminformatics for solving QSAR (Quantitative Structure Activity Relationship) task, and in bioinformatics for finding structural motifs. Most of the existing methods for this task explicitly or implicitly solve the subgraph isomorphism task which is computationally expensive, so they suffer from the lack of scalability problem when the graphs in the input database are large. We propose FS^3, which is a sampling based method and thus scalable [2]. It mines a small collection of subgraphs that are most frequent in the probabilistic sense. FS^3 performs a Markov Chain Monte Carlo (MCMC) sampling over the space of a fixed-size subgraphs such that the potentially frequent subgraphs are sampled more often. Besides, FS^3 is equipped with an innovative queue manager. It stores the sampled subgraph in a finite queue over the course of mining in such a manner that the top-k positions in the queue contain the most frequent subgraphs. Our experiments on database of large graphs show that FS^3 is efficient, and it obtains subgraphs that are the most frequent amongst the subgraphs of a given size.

[2]The name FS^3 should be read as *F-S-Cube*, which is a compressed representation of the 4-gram composed of the bold letters in **F**ixed **S**ize **S**ubgraph **S**ampler

To show the effectiveness of FS^3, we further study its performance over three biological databases and also analyze the characteristics of the mined graphs as a feature for studying the functionality of protein with the help of a human expert. For this, we model the interface region of a protein complex by graphs [20] and extract interface patterns of the given complex in the form of frequent subgraphs using FS^3. We show that a systematic review of the mined subgraphs provides an effective method for the discovery of functional motifs that exist along the interface region of a given protein complex.

Another important task in graph mining is to find subgraphs which are candidate for motif in a given network. For this, we need to count each topology's frequency in the input network as well as in many randomized networks. Counting a topology's frequency in a single network is a challenging task as it requires solving subgraph isomorphism, a known $\mathcal{NP}$-complete problem. As the size of the motif grows, the number of candidate motifs increases exponentially, and the task becomes more challenging. To cope with the enormous computation cost of exhaustive counting of the frequency of candidate motifs, researchers consider various sampling based methods that obtain an approximation of relative frequency measure (which we call concentration) over all the candidates of a given size. In [21], we propose an improved sampling based method on finding concentration of the prospective motifs.

Small substructures of 3, 4, and 5 nodes (also known as graphlets) have been used as feature representation for solving various tasks, such as, name disambiguation and studying network properties. In [22], we propose a novel topological signature of Android apps based on the function call graphs (FCGs) extracted from their Android App PacKages (APKs). We use an extended version of the sampling method proposed in [21] to capture the invocator-invocatee relationship at the local neighborhoods in the function call graphs (FCG) of Android app.

Android systems are widely used in mobile and wireless distributed systems. However, with the popularity of Android-based smartphones/tablets comes the rampancy of Android-based malware. Using function call graphs of benign and malware apps, we

Table 1.1.: Various aspects of models and application presented in this dissertation.

	Node Embed	Edge Embed	Link Predict.	Sentence Embed	Topic Classi., Clust, Summa.	Finding freq. Substruct	Finding Func. Motif	Classif. Android Apps
Chapter 3	✓		✓					
Chapter 4		✓	✓					
Chapter 5				✓	✓			
Chapter 6				✓	✓			
Chapter 7						✓		
Chapter 8						✓	✓	
Chapter 9						✓		
Chapter 10						✓		✓

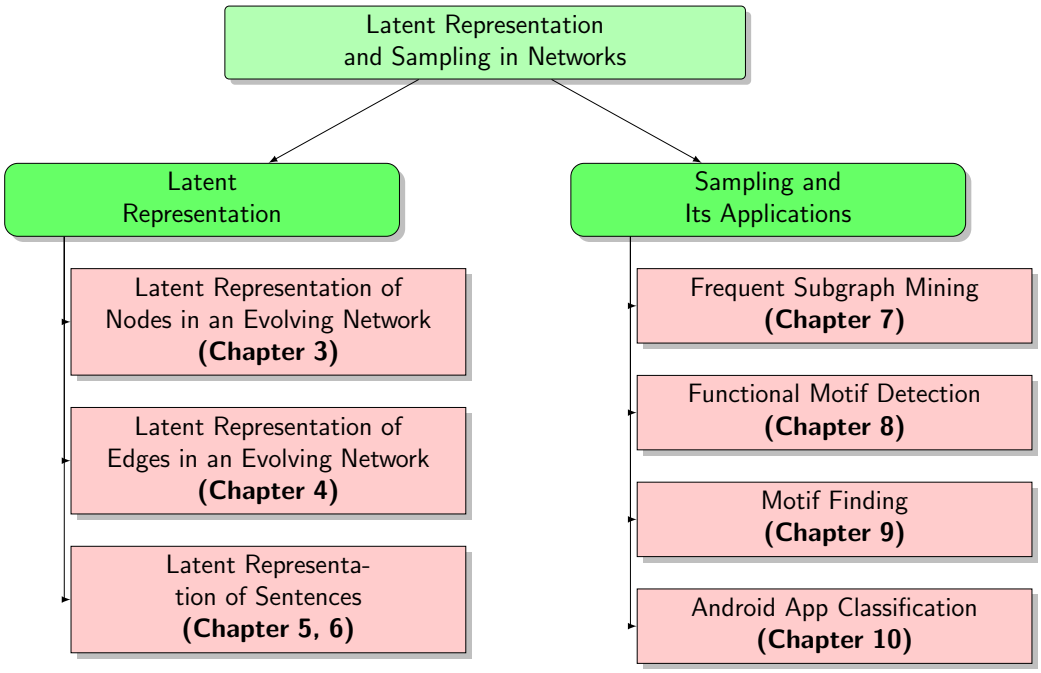


Fig. 1.1.: Visual depiction of the thesis organization.

demonstrate that our method, ACTS (App topologiCal signature through graphleT Sampling), can detect malware and identify malware families robustly and efficiently. More importantly, we show that the statistics and structures retrieved using the sampling algorithm [21] help finding function call structure which discriminates between the benign and malware app, and thus facilitates the interpretation of the results which is very important in the security domain.

1.4 Contribution of This Dissertation

We summarize the major contributions of this dissertation:

1. We propose latent representation learning models for learning representation of nodes in a dynamic networks for each snapshot by modeling the temporal evolution inside the latent representation model. We consider two different kinds of temporal smoothness models: (*i*) retrofitted, and (*ii*) linear transformation. Please see Chapter 3 for details.

2. We propose DyLink2Vec (Chapter 4) to overcome the problem of learning an effective feature representation of node-pair instances (edges) for the case of dynamic network. We show that our proposed link prediction model, which uses an auto-encoder model to learn latent feature representation for the edges outperforms several existing methods that use well-known link prediction features.

3. We propose models to effectively incorporate extra-sentential context in some form of inter-sentence relations in the representation learning models for sentences. Please see Chapter 5 and 6 for details.

4. We propose a method for frequent subgraph mining, called FS^3(Chapter 7), that is based on sampling of subgraphs of a fixed size. We also demonstrate that the algorithm, FS^3 can be applied to find essential structures from the interfacial region of a set of oligomeric proteins (Chapter 8).

5. We also provide sampling based algorithm for finding concentration of prospective motifs in a single large network (see Chapter 9) and show that the proposed method can be used to collect features from Function Call Graphs (FCGs) extracted from their Android App Packages (APKs) to classify apps from google app store as malignant or benign (Chapter 10).

1.5 Organization of This Dissertation

We organize our dissertation into two subparts. From Chapter 3 to Chapter 6, we describe latent representation methods for network and texual units, and from Chapter 7 to Chapter 10, we describe the techniques for extracting substructures and collecting statistics about them and finally present their novel usage in bio-Informatics and security domain. Finally, in Chapter 11, we give future directions and conclude the thesis. In order to better understand the topics presented in different chapters, we summarize various aspects of models, and applications presented in the dissertation in Table 1.1 and Figure 1.1.

2. RELATED WORK

In this chapter, we discuss the related work into the following four categories, namely (*i*) Representation learning of network units such as nodes and edges, (*ii*) Representation learning of textual units particularly for sentences, (*iii*) Substructure Mining, and (*iv*) , (*v*) Applications of Substructure Mining.

2.1 Representation Learning of Network Units

Represention learning of network units especially for nodes in a static network is a very popular research topic. A good number of models for learning latent representation of vertices [12–14, 23–26] in a static network have been proposed. These models vary in the way they exploit information from the static network as they learn the low-dimensional representation of the vertices. Most of the models vary based on the information extracted from the static network and how they are exploited to learn the low-dimensional representation of the nodes. The learned representation node vectors reflect the proximity and similarity among the nodes from different perspectives. For example, DeepWalk [12] and Node2Vec [13] design random walks to find the node pairs that should be considered similar, and then use word2vec's skip-gram model [27] to learn representations. LINE [14] extracts two kinds of proximities among the nodes: (*i*) direct link (first-order), and (*ii*) structural (second-order) proximity. SDNE [23] also captures first and second-order proximity, but it uses an encoder-decoder framework and Laplacian regularization to capture the proximity between a pair of vertices. Benson's model [24] extracts proximity among the nodes using network motifs. Qiu et al. [28] show that DeepWalk, LINE, and Node2Vec models can be unified under the matrix factorization framework. Most of these methods [12–14, 23, 24] are un-

supervised in nature, i.e. they do not use any human supervision to either learn or improve the representation.

However, when labels are available for a small subset of nodes, the problem of representation learning on networks can be framed as graph based semi-supervised learning. The label information is smoothed over the graph via some form of explicit graph-based regularization. Most of the semi-supervised embedding method [29, 30] learns embeddings in a neural network through imposing regularization on the graph structure or use graph structure as feature. A recent method GCN [25] encodes the graph structure directly using a neural network model and trains on a supervised target for all nodes with labels and thus can avoid explicit graph-based regularization in the loss function.

Modeling complex distributions over graphs and then efficiently sampling from these distributions is challenging due to the non-unique, high-dimensional nature of graphs and the complex, non-local dependencies that exist between edges in a given graph. Deep neural network based generative models can capture the complex distribution [31]. Generative modeling over network is another avenue of research which is also becoming popular. However, in this dissertation, we only restrict our attention on unsupervised models for learning representation for nodes and edges.

In early days, the network embedding (Graph Embedding) algorithms mostly aim to reduce dimensionality and learn the manifold that the data lies in. Manifold learning models, such as ISOMAP and Locally Linear Embedding (LLE), also aim to reduce dimensionality. ISOMAP [32] uses the geodesic distances among the nodes to learn a low-dimensional vector representation for each node. The geodesic distance is defined as the sum of the edge weights along the shortest path between two nodes in the network, and can be regarded as a proxy for the proximity among the nodes. LLE [33] eliminates ISOMAP's need to estimate the pairwise distances between widely separated nodes. The model assumes that each node and its neighbors lie on or near a locally linear patch of a manifold and subsequently, learns a neighborhood preserving latent space representation by locally linear reconstruction. Please see [3] and [34]

for a detailed survey on latent space representation techniques of nodes in a static network.

Although we found many embedding methods for static networks, we found only one related work for dynamic networks. Zhu et al. [35] attempt dynamic link-prediction by adding a temporal-smoothing regularization term to a non-negative matrix factorization objective. Their goal is to reconstruct the adjacency matrix of different time-stamps of a graph. They use a Block-Coordinate Gradient Descent (BCGD) algorithm to perform non-negative factorization. Their formulation is almost identical to the algorithm of Chi et al. [36], who perform evolutionary spectral clustering that captures temporal smoothness. Because matrix factorization provides embedding vectors of the nodes for each time-stamp, the factorization by-product from this work can be considered as dynamic network embeddings. In this dissertation, we propose two different models: (*i*) Retrofitting and (*ii*) Linear Transformation to capture temporal smoothness and network proximity simultaneously (please see Chapter 3). For the retrofitted model [6], we were inspired by its performance in modeling external information in representation learning of textual unit and for transformation models [27], we were inspired by its effectiveness in projecting representation of words from one languge to another for improving performance in machine translation. However, none of the existing works use retrofitted models or the linear transformation models to incorporate the temporal smoothness for learning the node representation in a dynamic network.

There are few other works which model dynamic networks or solve link prediction on dynamic networks. But, they do not learn latent vectors of the vertices for each time-stamps. For instance, the method in [37] computes a number of different node similarity scores by summing those similarities with weights learned for different time-stamps of the network. In order to predict the future node similarity scores, a time series forecasting model, ARIMA is used. But this approach fails to capture signals from neighborhood topology, as each time-series model is trained on a separate t-size feature sequence of a node-pair. Tylenda et al. [38] shows that time of interactions

between nodes is a dominant feature for ranking neighboring nodes and apply the time-aware feature representation technique to predict links in bibliographical network. The model assumes external information (such as, last time of collaboration) about the network is available. Rahman et al. [39] use graphlet transitions over two successive snapshots to solve the dynamic link prediction problem. A deep learning solution is proposed in [40], which uses a collection of Restricted Boltzmann Machines with neighbor influence for link prediction in dynamic networks. Matrix and tensor factorization based solutions are presented in [41, 42].

As described in the earlier paragraphs, there exist a growing list of recent works which use unsupervised methodologies for finding metric embedding of nodes in the static graph [12, 14, 43] and to learn representation of node-pair instances they use some type of compositional method such as, hadamard product, weighted-l1, weighted-l2 or simple averaging. There are also methods for learning features for edges in the dynamic network settings as described in the previous paragraph. However, no such work exists for finding feature representation of node-pair instances for the purpose of link prediction in a dynamic network. In this dissertation, we propose one method (please see Chapter 4) for metric embedding of node-pair in dynamic network by modeling the metric embedding task as an optimal coding problem where the objective is to minimize the reconstruction error, and it solves this optimization task using a gradient descent method.

2.2 Representation Learning of Textual Units

Recently, learning distributed representation of textual units such as words, phrases, and sentences has gained a lot of attention due to its applicability and superior performance over bag-of-words (BOW) features in a wide range of text processing tasks [5–7,44–46]. These models can be categorized into two groups: (*i*) task-agnostic or unsupervised models, and (*ii*) task-specific or supervised models. Task-agnostic models learn general purpose representation from naturally occurring unlabeled train-

ing data, and can capture interesting linguistic properties [47–49]. On the other hand, task-specific models are trained to solve a particular task, e.g., sentiment analysis [50], machine translation [51], and parsing [52].

The Word2vec model [53] to learn distributed representation of words is very popular for text processing tasks. The model also scales well in practice due to its simple architecture. Sen2Vec [44] extended Word2vec [53] to learn the representation for sentences and documents. The model maps each sentence to a unique id and learns the representation for the sentence using the contexts of words in the sentence—either by predicting the whole context independently (DBOW), or by predicting a word in the context (DM) given the rest. In this dissertation, we extend the DBOW model to incorporate inter-sentence relations in the form of a discourse context or a similarity context. We do this using a graph-smoothing regularizer in the original objective function, or by retrofitting the initial vectors with different types of context.

Retrofitting and regularization methods [5–7] have been explored to incorporate lexical semantic knowledge into word representation models. Our overall idea of using external information is reminiscent of these models with two key differences: (*i*) the semantic network (WordNet, FrameNet) is given for the case of the existing works, whereas we construct the network using similarities between sentences (nodes); (*ii*) we also explore discourse context that incorporate knowledge from adjacent sentences.

Adjacent sentences have been used previously for modeling task-agnostic representation of sentences. For example, Hill et al. [48] proposed FastSent, which learns word representation of a sentence by predicting words of its adjacent sentences. It derives a sentence vector by summing up the word vectors. The auto-encode version of FastSent also predicts the words of the current sentence. FastSent is fundamentally different from our models as we consider nearby sentences as atomic units, and we encode the sentence vector directly.

Hill et al. [48] also proposed two other models, Sequential Denoising Autoencoder (SDAE) and Sequential Autoencoder (SAE). SDAE employs an encoder-decoder framework, similar to neural machine translation (NMT) [51], to denoise an origi-

nal sentence (target) from its corrupted version (source). SAE uses the same NMT framework to reconstruct (decode) the same source sentence. Both SAE and SDAE compose sentence vectors sequentially, but they disregard context of the sentence.

Another context-sensitive model is Skip-Thought [54], which uses the NMT framework to predict adjacent sentences (target) given a sentence (source). Since the encoder and the decoder use recurrent layers to compose vectors sequentially, SDAE and Skip-Thought are very slow to train. Furthermore, by learning representations to predict content of neighboring sentences, these methods (FastSent and Skip-Thought) may learn linguistic properties that are more specific to the neighbors rather than the sentence under consideration.

In contrast, we encode a sentence directly by treating it as an atomic unit, and we predict the words to model its content. Similarly, our model incorporates contextual information by treating neighboring sentences as atomic units. This makes our model quite efficient to train and effective for many tasks as we have shown.

2.3 Substructure Mining

Frequent subgraph discovery is a well-studied problem with many existing methods, including Subdue [55], AGM [56], FSG [57], gSpan [58], DMTL [59], and Gaston [60]. They work well for problem instances where the graphs in the graph database are small and sparse, but they do not scale well with the size and the density of the input graphs. Note that, the lack of scalability issue of the existing methods for the large input graph is not a limitation of the existing methods, rather it is due to the strict definition of the FSM task itself.

To alleviate the scalability concern, researchers have proposed some alternative solutions, which do not discover all the frequent subgraphs. The first such attempt is to discover only a subset of frequent subgraphs, which are maximal [61, 62], or closed [63]. However for large input graphs, algorithms for finding maximal or closed frequent subgraphs are not scalable, as they prune only a small part of the search

space. Later, Chaoji et al. [64] have proposed ORIGAMI, a graph mining method that returns a set of random maximal frequent subgraphs. Hasan and Zaki proposed MCMC sampling based methods for uniform sampling of a set of frequent [65] and maximal frequent [66] subgraphs. Due to the uniformity guaranty, such methods provide a small set of frequent subgraphs which are ideal as a representative pattern set. However, all the above methods still solve subgraph isomorphism test for ensuring the minimum support threshold, which makes them inefficient when the input graphs become large. In this dissertation, our proposed subgraph mining method complement existing works as we are interested to obtain a solution for mining frequent subgraphs from large input graph, for which existing methods do not scale.

Studying the local topology is an important step for modeling the interaction among the entities in a network. There exist works [67–71] that mine frequent subgraphs from a single input graph. They aim to discover network motifs in a single network. In a seminal work around a decade ago, Shen-orr et al. [72] hypothesized that network motifs play an important role in carrying out the key functionalities that are performed by the entities in a biological network. Since then, researchers have also discovered that network motifs are building block for complex networks from many diverse disciplines including biochemistry, neurobiology, ecology, engineering [73], proteomics [74], social sciences [75] and communication [76].

Finding network motifs is computationally demanding. To identify whether a given subgraph topology is a motif, we need to count the topology's frequency in the input network as well as in many randomized networks. Counting a topology's frequency in a single network is a challenging task as it requires solving subgraph isomorphism, a known $\mathcal{NP}$-complete problem. As the size of the motif grows, the number of candidate motifs increases exponentially, and the task becomes more challenging. To cope with the enormous computation cost of exhaustive counting of the frequency of candidate motifs, researchers consider various sampling based methods that obtain an approximation of relative frequency measure (which we call concentration) over all the candidates of a given size. Most notable among these methods are

MFinder [67], MODA [77], and RAND-ESU [68]. Besides these approximate methods, exact motif counting methods are also available, such as, GTrieScanner [69], ESU [68], Grochow-Kellis algorithm [78], Kavosh [70], and NetMODE [79]; However, their application is limited to small networks only. In this dissertation, we propose methods which focus on finding concentration of prospective motifs on a single large network using a novel sampling based method.

2.4 Modeling Interface Region as Network and Mining Functional Motif

There are several works that represent a protein structure as a network consisting of a set of nodes and the relationship between the nodes. However, the way different works model the network differs. Across these works, the nodes can represent amino acid residues [80–85], functional atoms from the side chains [86,87], secondary structure elements [88–90], proteins [91, 92], protein complexes [93], and interaction pseudoatoms [94]. Edges also has different connotations in different works. For instances, edges connect nodes if they interact with each other [80, 81], or if they are nearer to each other spatially [82, 87], or if they are within the interacting distance of each other [86]. Some works create edges between two nodes if the nodes are part of a functional unit in a pathway or in a biological process [91, 92], or if side-chains interact with each other [95]. Our work [20] differs in the method of construction and analysis of these networks from previous studies. We use C_α carbon (backbone carbon) of a particular residue as a node. So, the C_α carbons from all the residues of a particular protein represent the set of nodes and we connect two nodes if their C_α carbons are spatially nearer to each other. Existing works use a graph to capture the entire protein structure, but we capture dense interfacial region between different subunits of the same structure.

Network representation of proteins has been used for various purposes; for example, to study the evolution of protein-protein interactions [82], to summarize how central network elements are enriched in active centers and ligand binding sites di-

recting the dynamics of entire protein [87], to classify protein 3D-structures [84, 85], to characterize the topological role of residues [83], to offer a comprehensible view of critical residues and to facilitate the inspection of their organization [96], to detect cancer-associated functional residues [91], to uncover distinct cancer-specific functional modules [92], to document functional components and sub-components of proteins [97], and to compare two networks (Oligomeric vs Monomeric) [81] for getting insight into the protein association. Greene *et al.* [98] authored a good review article which surveys several key advances in the expanding area of protein structure and folding research using network approaches. To the best of our knowledge we are the first to extend graph mining methodologies for mining interfacial networks to discover important functional units (such as, lock structure in HIV and hugging point in TIM structure), or to find family specific active sites from enzymes.

2.5 Classifying Android Apps

As the use of Android continues to grow, so does the threat of malware. Malicious behaviors observed in such malware include the theft of private information stored on the device, device fingerprinting, abusing premium service, and rooting the device as a backdoor for further attacks [99]. Detecting such malware is a critical task for the security research community.

It is observed that variants of malware form families through code sharing and their common lineage [99]. Therefore, instead of identifying individual malware and extracting a signature from it, we can identify the commonality within the same malware family and generate signatures that capture such commonality. Recently, various machine learning/data mining (i.e., pattern mining) techniques are applied to detect Android malware [100–105] or closely related tasks such as identifying repackaged apps [106, 107]. Beyond the common pattern mining framework, these works differ significantly in their selection and construction of features, their quantification/metrication of such features, their choice of pattern mining algorithms, and, in

totality of these fine points of design, their applicability, robustness, and efficiency in detecting malware.

A number of different app representations have been studied for malware detection. For example, [101] propose a compact representation of source code, the code property graph, that combines abstract syntax trees, control flow graphs, and program dependence graphs [101]. Other approaches do not require the source, but instead focus on features at different abstract levels: from the low-level platform opcode level [104], through the intermediate function call [100] and Android framework API [103] level, to the high semantic level that includes features such as network addresses and Android specific artifacts such as permission and intents [102]. Yet, other works formulate malware detection as different pattern mining tasks such as frequent subgraph mining [105].

Due to the availability of off-the-shelf obfuscation solutions (such as the free ProGuard [108] and the commercial DexGuard [109]) and the growing number of Android apps, it is critical for any proposed malware detection algorithm to be robust and efficient. In practice, efficiency and robustness are often at odds. At one extreme, as two straightforward examples, cryptographical hashes or package names are highly efficient but fragile app signatures. They are efficient to obtain/compute but can easily be changed without essentially affecting the app [104]. At the other extreme, measuring similarities of some high-level graph-based representation of the app, such as code property graphs [101], are more robust, but, as observed by [100], "is a non-trivial problem whose complexity hinders the use of these features for malware detection."

Martinelli et al. [105] formulates the malware detection problem as a subgraph mining problem. Pržulj et al. [110] first propose and coin the term graphlet. Two recent advances on graph mining, MHRW [21] and GUISE [111], inspire our use of GFD as a robust and efficient topological signature for apps.

A related problem to malware detection is app repackaging, in which an app is transformed for a similar but different app through repackaging [106]. Repackaged apps are often seen on alternative Android app market, and is a major vector for

carrying and propagating malware. Zhou et al. [107] propose a system called AppInk that applies watermarking to prevent app repackaging.

Tainting analysis (e.g., TaintDroid [112] and FlowDroid [113, 114]) and Android app analysis frameworks (e.g., DroidScope [115] and CopperDroid [116]) can be used to further analyze malware families identified by our proposed method, named ACTS [22].

3. MODELS FOR CAPTURING TEMPORAL SMOOTHNESS IN EVOLVING NEWTORKS FOR LERANING LATENT REPRESENTATION OF NODES

3.1 Introduction

Accurate modeling of temporal evolution of networks can help solve complex tasks involving social and interaction networks. For instance, capturing temporal dynamics of user interactions can explain how communities are formed and dissolved in a network over time. Temporal co-movement of financial asset prices explains how financial assets are clustered pronouncedly during an economic downturn, causing a cascading effect that leads to financial crisis. Temporal network models can explain the way social network topologies facilitate (or inhibit) grievances to intensify collective organization, leading to imminent crisis and conflict in a community [4]. Unfortunately, most of the network based analyses consider only the network topology [12–14], a few consider nodal attributes and topology [15], and almost all ignore the temporal evolution of a network. The objective of this work is to capture temporal evolution of networks by learning latent representation of vertices over time.

Over the past few years, there has been a surge in research [12–14, 23, 117] on embedding the vertices of a network into a low-dimensional, dense vector space. These embedding models utilize the topological information of a network to maximize objective functions that capture the notion that nodes with similar topological arrangements should be distributed closely in the learned low-dimensional vector space. The embedded vector representation of the vertices in such a vector space enables effortless invocation of off-the-shelf machine learning algorithms, thereby facilitating several downstream network mining tasks, including node classification [26], link prediction [13], and community detection [117]. However, most of the existing

network embedding methods, including DeepWalk [12], LINE [14], and Node2Vec [13] only consider a static network in which the time-stamp of the edges are ignored. The embedding vectors of the nodes do not have any temporal connotation. These time-agnostic models may produce incorrect analysis—for example, in a static link-prediction task, node vectors might have been learned (inadvertently) by using future edges, but foreseeing future edges is impossible in a real-time setup. In summary, temporal network models should learn latent representation of vertices by considering the edges in their temporal order to make the model interpretable along the time axis, leading to the discovery of temporal evolution patterns of a dynamic network.

If a temporal network is represented as a collection of snapshots at discrete time intervals, one may attempt to use static network models (e.g., LINE, Node2Vec) for learning vector representation of vertices at each time-stamp independently. Through the embedding vectors of each vertices, these models encode useful semantic information, specifically, proximity and homophily relation among the vertices. But, the learning is limited to only one given time-stamp. More importantly, due to the independence in learning process across the time-stamps, the latent vectors of the vertices are embedded in different affine spaces for different temporal snapshots of the network. Therefore, there is no temporal mapping across the affine spaces to connect the embedding vectors of the same vertex across different time-stamps. Another related objective that we may have is to capture the temporal progression of the vertices in a latent space, say, for solving the task of community evolution over time; existing embedding models for static networks also fail to fulfill this objective.

We want to emphasize the differences between the two objectives which we have discussed in the above paragraph. For the first objective, we want an operator to transform the coordinates of the identical vertices (as obtained from the embedding of a dynamic graph in different time-stamps) from one affine space to another affine space. We refer to this objective as *global temporal smoothness* as we achieve this by considering all the vertices of a network holistically. Transformation here acts as a smoothness operator to connect the embedding vectors of the vertices over the time

space. On the other hand, for the second objective, we apply temporal smoothness over the vertices independently to ensure that their vectors have a smooth progression through the time-stamps. We call this *local temporal smoothness*. The existing network models fail to return temporal representation vectors of the vertices fulfilling either of the objectives; overcoming this limitation is the main motivation of this work. Note that, some earlier works have used temporal smoothness for evolutionary clustering [36] and link prediction in a dynamic network [35]. Both works use an identical objective function which minimizes a matrix-factorization coupled with a temporal smoothing regularization. But, these models only use first-order proximity. To the best of our knowledge, no models consider higher-order proximities and temporal smoothness to provide latent representation of vertices for each time-stamp of a dynamic network.

In this work, we propose two embedding models, (*i*) retrofitted, and (*ii*) linear transformation, each fulfilling one of the smoothness objectives. Figure 3.1 gives a conceptual demonstration of these models. The *retrofitted* model satisfies the local temporal smoothness objective by assuming that the evolution of the network is vertex-centric. In each time stamp, a small fraction of the vertices experience changes in their neighborhood. The *retrofitted* model smoothly updates (retrofits) the embedding vectors of vertices, which are attached to the new edges in a given time stamp. As shown in the top example of Figure 3.1 the presence of new edges AF and CD in the graph G^{t+1}, updates the vector representation of the vertices A, C, D and F from their prior position corresponding to G^t. For the first time-stamp though, the model employs an existing latent representation model (neural network, manifold learning, or matrix-factorization based like PCA or SVD) to learn the representation vector of the vertices, but for subsequent time-stamps the position of the vectors are updated by a local update method as discussed above. The retrofitted model enables temporal tracking of the vertices of a network which can be instrumental for solving an evolutionary clustering of the vertices or to discover the evolution of communities over time.

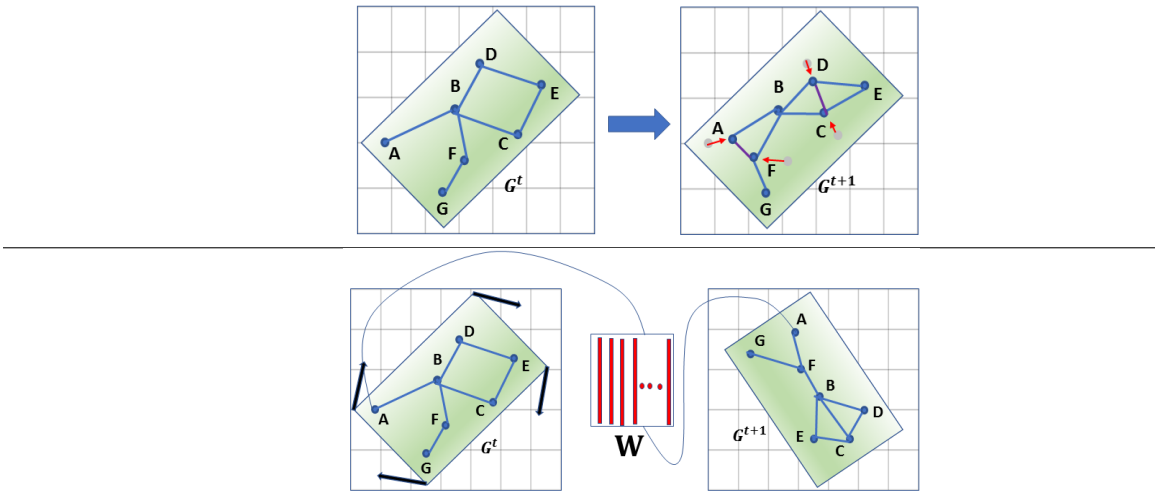


Fig. 3.1.: A conceptual sketch of retrofitting (top) and linear transformation (bottom) based temporal smoothness.

Our second model, the *linear transformation* model assumes that the network evolution over time is a global process, which makes the evolution network-centric, instead of vertex-centric. To accommodate this assumption, this model attempts to fulfill the global temporal smoothness objective by considering the temporal evolution of a network as a linear transformation operator over the vertex embedding vectors of successive time-stamps, as shown in Figure 3.1(bottom). In this figure we show the (independently learned) embedding vectors of the vertices in G_t and G_{t+1}, and our objective is to learn the transformation operator $\mathbf{W}$, which can map the vectors of each vertices from its position in G_t to its position in G_{t+1}. Once learned, the operator $\mathbf{W}$ is able to map the latent representation from a known snapshot to the next (unobserved) snapshot. The model first obtains embedding functions of all temporal snapshots of the graph and then it learns the transformation operation which best explains the evolution of vertex embedding vectors across different time-stamps. We explore two ways to learn the transformation matrix: homogeneous and heterogeneous. In *homogeneous mapping*, we assume that the transformation operation is the same across any two successive time-stamps. So, we learn a single (shared) transformation matrix that maps the representation from a snapshot to the next snapshot. *Heterogeneous mapping* on the other hand refrains from the uniformity assumption, considering that every pair of time-stamps has a different transformation geometry. So, the model learns a projection matrix for every subsequent time-pairs and then combines them while performing smoothing in the time dimension.

Contributions of this paper are summarized as below.

1. We propose two novel models for learning the vertex representation of a dynamic network having many temporal snapshots. In these models we introduce two different kinds of temporal smoothness concepts: global and local, which complement each other.

2. We validate our proposed models by utilizing them for solving the temporal link prediction task on nine different datasets from three different domains: citation,

social, and messaging. Experimental results show that when compared against an existing state-of-the-art temporal smoothness based dynamic link prediction model, for all datasets our proposed methods improve the link prediction performance by values ranging from 0.20 to 0.60 on different metrics, such as, AUC, PRAUC, and NDCG.

3. We made our code, datasets, and experimental setups publicly available at (`https://gitlab.com/tksaha/temporalnode2vec.git`) to support the spirit of reproducible research and to enable further development in this area.

3.2 Related Work

Recently, models for learning latent representation of vertices [12–14, 23–26] of a static networks have become very popular. These models vary in the way they exploit information from the static network as they learn the low-dimensional representation of the vertices. For example, DeepWalk [12] and Node2Vec [13] design random walks to find the node pairs that should be considered similar, and then use word2vec's skip-gram model [47] to learn representations. LINE [14] extracts two kinds of proximities among the nodes: (*i*) direct link (first-order), and (*ii*) structural (second-order) proximity. SDNE [23] also captures first and second-order proximity, but it uses an encoder-decoder framework and Laplacian regularization to capture the proximity between a pair of vertices. Benson's model [24] extracts proximity among the nodes using network motifs. Qiu et al. [28] show that DeepWalk, LINE, and Node2Vec models can be unified under the matrix factorization framework.

Manifold learning models, such as ISOMAP and Locally Linear Embedding (LLE), also aim to reduce dimensionality. ISOMAP [32] uses the geodesic distances among the nodes to learn a low-dimensional vector representation for each node. LLE [33] eliminates ISOMAP's need to estimate the pairwise distances between widely separated nodes. The model assumes that each node and its neighbors lie on or near a

locally linear patch of a manifold and subsequently, learns a neighborhood preserving latent space representation by locally linear reconstruction.

Although we found many embedding methods for static networks, we found only one related work for dynamic networks. Zhu et al. [35] attempt dynamic link-prediction by adding a temporal-smoothing regularization term to a non-negative matrix factorization objective. Their goal is to reconstruct the adjacency matrix of different time-stamps of a graph. They use a Block-Coordinate Gradient Descent (BCGD) algorithm to perform non-negative factorization. Their formulation is almost identical to Chi et al. [36] who perform evolutionary spectral clustering that captures temporal smoothness. Because matrix factorization provides embedding vectors of the nodes for each time-stamp, the factorization by-product from this work can be considered as dynamic network embeddings. In experiment section we compare our proposed methods with this work.

There are few other works which model dynamic networks or solve link prediction on dynamic networks. But, they do not learn latent vectors of the vertices for each time-stamps. For instance, the method in [37] computes a number of different node similarity scores by summing those similarities with weights learned for different time-stamps of the network. Rahman et al. [39] use graphlet transitions over two successive snapshots to solve the dynamic link prediction problem. A deep learning solution is proposed in [40], which uses a collection of Restricted Boltzmann Machines with neighbor influence for link prediction in dynamic networks. Matrix and tensor factorization based solutions are presented in [41, 42].

3.3 Problem Formulation

Let $\mathcal{T} = \{1, 2, \ldots, T\}$ be a finite set of time-stamps for an evolving (undirected) network G, and for $t \in [1, T]$, $G_t = (V_t, E_t)$ denotes the network state at time t with V_t being the set of vertices and E_t being the set of edges of graph G_t at t'th time-stamp. The sequence of network snapshots is thus represented by $\mathcal{G} = (G_1, G_2, \ldots, G_T)$. For

simplicity, we assume that all the networks in $\mathcal{G}$ have the same vertex set, i.e., $G_t = (V, E_t)$ for $t = 1, 2, \ldots, T$.[1] We also assume that apart from the link information in a network, no other attribute data for the nodes or edges are available.

Now, let $\phi_t : V \rightarrow \mathbb{R}^d$ be the mapping function at time-stamp t that returns the distributed representations, i.e., real-valued d-dimensional vectors representing the vertices in G_t. In terms of data structure, ϕ_t is simply a look-up matrix of size $|V| \times d$, where $|V|$ is the total number of vertices in the network. The task of dynamic network embedding is to approximate ϕ_t from the sequence of first t network snapshots, represented as, $\mathcal{G}_t = (G_1, G_2, \ldots, G_t)$. Unlike existing embedding models, for learning embedding function ϕ_t, we want to utilize both the topological information in G_t and the trends in temporal dynamics exhibited by the sequence of network snapshots up to time t.

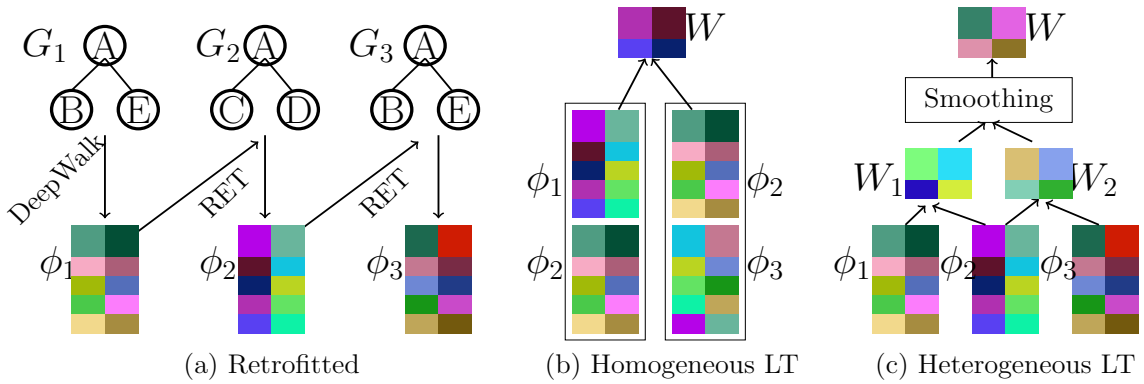
In this work, we propose two different models: retrofitted, and linear projection, each feeding on a specific temporal smoothness assumption.

1. For the retrofitted model, we first learn ϕ_1 from the network information in G_1 using any of the state-of-art static network embedding methods (e.g., Node2Vec, DeepWalk). Then we capture the temporal network dynamics by retrofitting ϕ_1 successively with the network snapshots $G_2, G_3, \ldots, G_t$.

2. For the linear transformation model, we learn a linear transformation matrix $W \in \mathbb{R}^{d \times d}$ to map ϕ_{t-1} to ϕ_t. The matrix W is trained after all the ϕ_i for $1 \leq i \leq t$ are obtained by using one of the existing static embedding methods on network snapshots in $\mathcal{G}_t = (G_1, G_2, \ldots, G_t)$.

To validate our proposed models we use temporal link prediction as an example task, where we predict the links in a future snapshot of a network, namely G_{T+1}. However, we would like to point out that our proposed embedding methods are agnostic to the task at hand.

[1]This assumption is not a limitation, as the proposed models can easily be adapted for the case when this assumption does not hold.



(a) Retrofitted (b) Homogeneous LT (c) Heterogeneous LT

Fig. 3.2.: Toy illustration of our method. ϕ's represent the embedding vectors of the vertices. (a) for retrofitted model, we first learn ϕ_1 by using any of static embedding learning models. We then use retrofitting to learn ϕ_2 and ϕ_3 using (ϕ_1, G_2) and (ϕ_2, G_3), successively. (b) & (c) for linear transformation (LT) models, we first learn ϕ_1, ϕ_2, and ϕ_3 by using any of static embedding models, and use these embeddings to learn a transformation matrix W. Please see Section 3.4 for details about how to learn W for homogeneous and heterogeneous transformation models.

3.4 Method

In the following, we describe our proposed models in detail.

3.4.1 Retrofitted Model

This model is based on the local temporal smoothness assumption, where the smoothness is applied to different vertices independently. This assumption is needed to track the embedding vector of the vertices as the network evolves. As shown in Figure 3.2(a), we do not learn the mapping function, ϕ, from different temporal snapshots of the graphs. Instead, we learn ϕ_1 from G_1 by using any of the static network embedding models discussed in Section 3.2. Then, we transform ϕ_1 to ϕ_t, by iteratively retrofitting information from later network snapshots $G_2, G_3, \ldots, G_t$.

In retrofitting, we revise $\phi_{t-1}(v)$ by using the neighborhood information available from the graph snapshot at time t, so that the resulting vector $\phi_t(v)$ is similar to

the prior vector $\phi_{(t-1)}(v)$ and at the same time close to the vectors of its adjacent nodes in G_t. The similarity between $\phi_t(v)$ and $\phi_{t-1}(v)$ enables the vertex v to move smoothly in the embedded space as time progresses from $t-1$ to t. The closeness of $\phi_t(v)$ with its neighbor at time t satisfies the proximity requirement of any static network embedding model. Thus, we minimize the objective function:

$$J(\phi_t) = \underbrace{\sum_{v \in V} \alpha_v ||\phi_t(v) - \phi_{(t-1)}(v)||^2}_{\text{Temporal Smoothing}} + \underbrace{\sum_{(v,u) \in E_t} \beta_{u,v} ||\phi_t(u) - \phi_t(v)||^2}_{\text{Network Proximity}}, \tag{3.1}$$

where α controls the strength to which the algorithm matches the prior vectors, for supporting *temporal smoothness*, and β controls the emphasis on *network proximity*. The quadratic cost in Equation 3.1 is convex in ϕ_t, and has a closed form solution [118]. The closed form expression requires an inversion operation, which can be expensive for large networks. The Jacobi method, an online algorithm, is more efficient as it solves the problem iteratively. The Jacobi method utilizes the following update rule:

$$\phi_t(v) \leftarrow \frac{\alpha_v \phi_{(t-1)}(v) + \sum_u \beta_{v,u} \phi_t(u)}{\alpha_v + \sum_u \beta_{v,u}}. \tag{3.2}$$

Algorithm 1: Jacobi method for retrofitting.

Input :
- Graph $G_t = (V, E_t)$
- Prior vectors $\phi_{(t-1)}$
- Probabilities $\alpha_{\mathbf{v}}$ and $\beta_{v,u}$

Output: Retrofitted vectors ϕ

$\phi \leftarrow \phi_{(t-1)}$ `// initialization`

repeat
  for *all* $v \in V_t$ **do**
    $\phi_t(v) \leftarrow \frac{\alpha_v \phi_{(t-1)}(v) + \sum_u \beta_{v,u} \phi_t(u)}{\alpha_v + \sum_u \beta_{v,u}}$
  end
until *convergence*;

In our experiments, we set $\beta_{v,u} = \frac{1}{degree(v)}$, and use the same α, which we tune using a held-out validation set, for all nodes $v \in V$. In other words, we vary weights for

temporal smoothness while fixing the weights for *network proximity*. It is clear from Eq. 1 that $\phi_t(v)$ is a convex combination of v's embedding at $t-1$ and the centroid of u's neighbors' embeddings at t. Algorithm 1 formally describes the training procedure of our retrofitted model. We experiment with several static embedding models to learn the embeddings of the first snapshot, ϕ_1 (see Section 3.5.4 for details).

Except for the first time-stamp, the retrofitted model does not "learn" from data about how to transform the embeddings, rather it presumes smoothing criteria. This presumption is effective when the smoothing criteria are met, but may be ineffective otherwise. Therefore, retrofitting is not a generic solution. In addition, retrofitting is limited because it requires a network snapshot G_t to perform inference at time t. To address these limitations, we propose linear transformation models.

3.4.2 Linear Transformation Models

In a dynamic network, we can expect that the network evolves by following a domain dependent pattern. So, the vertex representation vectors of two different time-stamps should have a similar transformation. In our transformation based models, we exploit this similarity by learning a linear mapping from a source (ϕ_{t-1}) to a target (ϕ_t) embedding space.

Our goal is to learn a transformation matrix, $W \in \mathbb{R}^{d \times d}$, that can transform ϕ_{t-1} to ϕ_t, given the network snapshots, $\mathcal{G} = (G_1, G_2, \ldots, G_T)$, and their corresponding statically-learned network embedding matrices, $\Phi = (\phi_1, \phi_2, \ldots, \phi_T)$. We explore two types of models: homogeneous and heterogeneous. In the homogeneous model, we assume that the transformation matrix is the same for the time-stamps, 1 to T, i.e., W is shared across snapshots. On the other hand, for the heterogeneous model, we assume a different W for each pair of time-stamps, resulting in $T-1$ different transformation matrices. We form the final transformation matrix, W, by combining the $T-1$ matrices.

Homogeneous Transformation Model

Our homogeneous transformation model is shown in Figure 3.2(b). First, we construct both a source matrix X by vertically stacking the embedding matrices ϕ_1, ϕ_2, $\ldots$, ϕ_{T-1}, and a target matrix Z by vertically stacking the matrices ϕ_2, ϕ_3, $\ldots$, ϕ_T (as shown in Eq. 3.3), given the sequence of (static) embedding matrices $\Phi = (\phi_1, \phi_2, \ldots, \phi_T)$. Corresponding rows X_u and Z_u represent the embedding vectors for node u at network snapshots G_t and G_{t+1}, respectively, for $t = 1, 2, \ldots, T-1$. To learn the matrix W, we minimize the objective function:

$$J(W) = ||WX - Z||^2, \quad \text{where} \quad X = \begin{bmatrix} \phi_1 \\ \phi_2 \\ \vdots \\ \phi_{T-1} \end{bmatrix}; Z = \begin{bmatrix} \phi_2 \\ \phi_3 \\ \vdots \\ \phi_T \end{bmatrix}. \tag{3.3}$$

We solve Eq. 3.3 with gradient descent. One can use stochastic gradient descent with minibatch to scale to large X and Z matrices.

Heterogeneous Transformation Model

In our heterogeneous model, we minimize an objective function similar to Eq. 3.3, but learn a different projection matrix for each pair of network snapshots. Given T different embedding matrices $\Phi = (\phi_1, \phi_2, \ldots, \phi_T)$, we learn $T-1$ different transformation matrices by minimizing the objective function:

$$J(W_t) = ||W_t\phi_t - \phi_{t+1}||^2, \quad for \quad t = 1, 2, \ldots, (T-1). \tag{3.4}$$

Then, we obtain the final transformation matrix W by combining the projection matrices from times $t = 1, 2, \ldots, (T-1)$. Figure 3.2(c) depicts this process for three snapshots. To smooth the projection matrices from Eq. 3.4, we experiment with different smoothing combinations:

(a) *Uniform smoothing:* We weight all projection matrices equally, and linearly combine them:

$$(avg) \quad W = \frac{1}{T-1}\sum_{t=1}^{T-1} W_t. \tag{3.5}$$

(b) *Linear smoothing:* We increment the weights of the projection matrices linearly with time:

$$(linear) \quad W = \sum_{t=1}^{T-1} \frac{t}{T-1} W_t. \tag{3.6}$$

(c) *Exponential smoothing:* We increase weights exponentially, using an exponential operator (exp) and a weighted-collapsed tensor (wct):

$$(exp) \quad W = \sum_{t=1}^{T-1} \exp^{\frac{t}{T-1}} W_t \tag{3.7}$$

$$(wct) \quad W = \sum_{t=1}^{T-1} (1-\theta)^{T-1-t} W_t. \tag{3.8}$$

3.5 Experimental Settings

To evaluate the performance of our dynamic network embedding models visually, we show a network visualization video over time using the Enron email dataset (see section 3.7). For quantitative evaluation, we solve the **temporal link prediction** task using the vertex embedding vectors from our models. Temporal link prediction is an extension of the well-known missing link prediction problem in a static network. It is defined as follows. Given a sequence of T snapshots of an evolving network, $\mathcal{G} = (G_1, G_2, \ldots, G_T)$, predict the links in G_{T+1}; in other words, construct a function $f(u, v)$ that predicts whether an edge $e(u, v)$ exists between any two nodes $u, v \in V_{T+1}$. In our retrofitted model, we use Hadamard product (element-wise multiplication) of node embeddings of time T (i.e., $\phi_T(u)$ and $\phi_T(v)$) as the input feature representation of the node-pair $\{u, v\}$. For the transformation models, we use the hadamard product of node embeddings of time $T+1$ i.e., $\phi_{T+1}(u)$ and $\phi_{T+1}(v)$ learned using $W \cdot \phi_T$ as the

input feature representation. Then, we use a logistic regression model (a Scikit-learn implementation with default parameter settings) to obtain f.

For each dataset described in Section 3.5.1, we randomly select 50% of the total positive edges from G_T as training and 30% as test, and we leave the remaining as validation. We randomly select equal amounts of negative edges for the train, test and validation sets for a balanced classification. We tune the hyper-parameters on the validation set, and evaluate our models on the test sets. We repeat this process 10 times to get 10 different negative edge sets; and we report our average performance over these sets.

3.5.1 Datasets

We perform temporal link prediction on nine datasets of three classes of networks: four academic collaboration networks, three messaging networks, and two social networks. The datasets vary in size and density. In Table 3.1, we report the number of nodes, the number of distinct edges (across all time stamps), and the number of interactions (counting plurality of an edge across different timestamps) for each dataset. We have published all the processed datasets along with our code release.

(Collaboration Networks) DBLP2, DBLP3, NIPS, HepPH

Both DBLP2 and DBLP3 datasets (obtained from `arnetminer.org`) have 10 time stamps with the paper citation information of about $49,455$ author-pairs and around 1.4 million papers. To create the co-authorship network, we add edges between people who co-authored a paper. We consider publications between 2000-2009, each year as a time stamp. Since DBLP2 and DBLP3 datasets are very sparse, we preprocess the data to retain only the active authors, whose last published papers are on or after the year 2010. For DBLP2, we retain authors who participated in at least two publications in seven or more time stamps. For DBLP3, we retain the authors with at least four publications in seven or more time-stamps. The NIPS [119] dataset con-

Table 3.1.: Temporal link prediction datasets. Nodes and edges denote the distinct number of vertices and edges over all the time-stamps. We also report the number of distinct interactions after removing self-edges. Number of snapshots denote the total number of time spans of the data.

Data	#Nodes	#Edges	Interactions	#Snapshot
DBLP2	315	943	2552	10
DBLP3	653	3379	9080	10
NIPS	2865	4733	5461	17
HepPH	28093	3148447	3718015	9
CollegeMsg	1899	13838	18127	10
SMS-A	44430	52222	144164	30
Email-EU	986	16064	81147	30
Facebook	63731	817035	817035	5
Facebook2	663	5271	11697	9

sists of collaboration information among 2865 NIPS authors. The dataset contains 17 snapshots for volumes 1-17. HepPH is the collaboration graph of authors of scientific papers from arXiv's High Energy Physics - Phenomenology. An edge between two authors represents a coauthored publication, and its time-stamp denotes the publication date. We divide HepPH dataset into nine snapshots.

(Messaging Networks) CollegeMsg, Email-EU, SMS-A [120]

The CollegeMsg dataset is comprised of messages from an online social network at the University of California, Irvine. An edge represents a private message between users. Email-EU dataset is a collection of emails between members of a European research institution, such that an edge represents an email. In the SMS-A dataset, an edge is an SMS text between persons. We divide CollegeMsg into 10 shapshots, and the other two datasets into 30.

(Social Networks) Facebook, Facebook2

In the Facebook dataset, a node represents a user in the Facebook friendship network, and an edge represents a friendship relation between two users. Time-stamps denote the time the friendship was established. We divide this dataset into 5 snapshots. Facebook2 is a network of Facebook wall posts [121]. Each node is a Facebook user account, and each edge represents a user's post on another user's wall. Facebook2 has 9 time-stamps because we preprocess this dataset in the same way as Xu [42], where each time-stamp represents 90 days of wall posts.

3.5.2 Evaluation Metrics

To evaluate link prediction performance, we use three metrics: area under the Receiver Operating Characteristic (ROC) curve (AUC), area under the Precision-Recall curve (AUPRC) [122], and Normalized Discounted Cumulative Gain (NDCG),

an information retrieval metric. AUC is equal to the probability that a classifier will, for the link prediction task, rank a randomly chosen positive instance (a node-pair which has an edge at time T) higher than a randomly chosen negative instance (a node-pairs with no edge at time T). AUC values range from 0.0 to 1.0. The second metric AUPRC considers the ranked sequence of node pairs based on their likelihood to form an edge at time T. We create a precision-recall curve by computing precision and recall at every position in the ranked sequence of node pairs. AUPRC is the average value of precision over the interval of lowest recall (0.0) to highest recall (1.0) AUPRC values range from 0.0 to 1.0. NDCG measures the performance of a link prediction system based on the graded relevance of recommended links. $NDCG_P$ varies from 0.0 to 1.0, where 1.0 represents the ideal ranking of edges. P, a number chosen by the user, is the number of links ranked by the method. We choose $P = 50$ in all our experiments.

3.5.3 Competing Methods

Our objective is to compare the relative quality of latent vertex embeddings in a dynamic network. So, we only compare our proposed methods with existing vertex embedding models which can provide explicit latent vectors for all the vertices at every time-stamp of a dynamic network. The majority of dynamic link prediction methods [37, 39–42] do not satisfy this requirement, except Zhu et al. [35]. They propose a method for performing dynamic link prediction called BCGD (Block Coordinate Gradient Descent) which performs non-negative matrix factorization with temporal smoothness. From the factorization of the adjacency matrix at each time stamp, we can obtain latent vectors of the vertices. BCGD minimizes the objective function:

$$J_{BCGD} = \underbrace{\sum_{t=1}^{T-1} ||G_t - \phi_t(u)\phi_t(v)||^2}_{\text{Network Proximity}} + \underbrace{\lambda \sum_{t=1}^{T-1} \sum_{u} 1 - \phi_t(u)\phi_{t-1}(u)^T}_{\text{Temporal Smoothing}} \; s.t. \; \phi_t \geq 0. \tag{3.9}$$

The temporal smoothing part of Equation 3.9 penalizes sharp change of latent position of a node u, whereas, the first part captures the latent proximity. We use the author-provided implementation of the incremental-BCGD algorithm and tune its parameters identically along with our proposed methods.

3.5.4 Different Configurations of the Proposed Models

We vary our three proposed models: Homogeneous (homogeneous transformation), Heterogeneous (heterogeneous transformation), and Retrofitted models, by choosing seven different base representation-learning methods: three random-walk based (Deepwalk, LINE, Node2Vec), two matrix factorization based (PCA, tsvd), and two manifold based (LLE, ISOMAP). For the transformation models we utilize the base method to learn the vertex representation vectors on the snapshots 1 to (T-1). Our retrofitted models use the base representation methods to learn the vertex representation vectors on the first snapshot of the graph. To vary our heterogeneous transformation models further, we experiment with different smoothing functions: Uniform, Linear, and Exponential (exp and wct). For fair comparison, we set the representation dimensions equal to 64 for all models. However, we have reported results over other representation dimensions (see Section 3.6.4).

DeepWalk, LINE, and Node2Vec methods are trained with stochastic gradient descent. We used negative sampling, with 5 noise samples, to significantly decrease training time. We also used subsampling of frequent words. We tune DeepWalk's

window size parameter from the set {8, 12, 15}. We tune LINE's iteration parameter from the set {5, 10, 20} (millions of iterations). We tune Node2Vec's p and q parameters, which control the amount of exploration vs exploitation in the random walk, from the set {0.1, 0.3, 0.5}. For PCA, tsvd, LLE, and IsoMap, we use the implementation provided by scikit-learn with the default settings.

In our Retrofitting models, we iterate 20 times. We tune α_v, which controls the weight of the prior vector, from the set of values: {0.1, 1, 10}. For our Homogeneous and Heterogeneous models, we use Batch Gradient Descent with 10,000 iterations. We also perform gradient clipping, which clips values of multiple tensors by the ratio of the sum of their norms, with a clipping ratio of 5.0.

3.6 Results and Discussion

In Table 3.2, 3.3 and 3.4, we show the comparison between the competing method (BCGD) and our proposed models: retrofitting (RET), homogeneous transformation (HomoLT), and heterogeneous transformation (HeterLT). Our models' performances vary for each base embedding method they implement. Therefore, we only report the results from the best-performing base embedding model (name in parentheses). In cases where the best-performing base models differ for different metrics, we present the results of the base model that is best overall. For all datasets and all metrics the best performing model's performance is shown in boldface font.

Table 3.2, 3.3 and 3.4. show that one of our three proposed methods outperforms BCGD in all three metrics over all datasets. This suggests that the latent embedding vectors from our proposed models are better for link prediction than BCGD's embedding vectors. We think BCGD may under-perform because it can only exploit edge-based proximity when learning latent embedding vectors by factoring the adjacency matrix. Whereas, our models capture more complex network proximity by implementing, as base embedding models, state-of-the-art static node embedding methods. In addition, our local and global temporal smoothness methods provide a

Table 3.2.: Performance of seven of our homogeneous models, 28 of our heterogeneous models, and 7 of our retrofitted models on citation datasets. The highlighted results are statistically significant over the baseline with $p < 0.001$. For fair comparison, we set the latent dimension size to 64.

(a) DBLP2

Method	AUC ± sd	AUPRC ± sd	$NDCG_P$ ± sd
BCGD	0.7932 ± 0.03	0.8023 ± 0.03	0.8686 ± 0.02
RET (tsvd)	**0.8360 ± 0.02**	**0.8571 ± 0.01**	
HomoLT (DeepWalk)	0.7422 ± 0.03	0.7691 ± 0.04	0.8496 ± 0.05
HeterLT (DeepWalk, avg)	0.7413 ± 0.02	0.7798 ± 0.02	0.8662 ± 0.02

(b) DBLP3

Method	AUC ± sd	AUPRC ± sd	$NDCG_P$ ± sd
BCGD	0.8564 ± 0.01	0.8599 ± 0.02	0.9524 ± 0.02
RET (LLE)	**0.8898 ± 0.01**	**0.8926 ± 0.01**	**0.9776 ± 0.01**
HomoLT (LINE)	0.7557 ± 0.02	0.7818 ± 0.02	0.9671 ± 0.01
HeterLT (LINE, linear)	0.8301 ± 0.01	0.8680 ± 0.007	0.9934 ± 0.004

(c) HepPH

Method	AUC ± sd	AUPRC ± sd	$NDCG_P$ ± sd
BCGD	0.5732 ± 0.003	0.6170 ± 0.003	0.5542 ± 0.02
RET (LINE)	0.5677 ± 0.004	0.5782 ± 0.003	0.8659 ± 0.04
HomoLT (DeepWalk)	0.5889 ± 0.003	0.6269 ± 0.005	0.8376 ± 0.04
HeterLT (DeepWalk, avg)	**0.6058 ± 0.02**	**0.6346 ± 0.02**	**0.9300± 0.03**

(d) NIPS

Method	AUC ± sd	AUPRC ± sd	$NDCG_P$ ± sd
BCGD	0.5157 ± 0.002	0.5457 ± 0.02	0.6118 ± 0.02
RET (LLE)	0.5427 ± 0.04	0.5542 ± 0.02	0.5988 ± 0.08
HomoLT (Node2Vec)	**0.5633 ± 0.01**	**0.6123 ± 0.01**	0.7947 ± 0.02
HeterLT (Node2Vec, wct)	0.5581 ± 0.01	0.6211 ± 0.01	**0.8072 ± 0.03**

better dynamic network model than BCGD. Below, we present the results in detail by grouping them over the three different kinds of networks.

3.6.1 Link Prediction in Citation Network

Results of three collaboration networks are given in Table 3.2. Among the citation networks, DBLP2 and DBLP3 results are similar (see Table 3.2a and 3.2b); for both the datasets, the retrofitted model (RET) performs the best in all three metrics. For DBLP2, RET improves the AUC, PRAUC, and NDCG values of the competing BCGD method by 0.04, 0.06, and 0.06 units, which translates to 5%, 7%, and 7% improvement, respectively. For DBLP3, the improvement of RET on these three metrics are 0.04, 0.03, and 0.02. Interestingly, the base learning models of the best performing RET differs over these datasets, for DBLP2, it is tsvd and for DBLP3, it is LLE. The performance of HomoLT and HeterLT in comparison to BCGD are mixed over different metrics. For instance, they are better in NDCG metric, but marginally worse in the AUC, and PRAUC metrics.

For the NIPS dataset (see Table 3.2d), the transformation models (HomoLT and HeterLT) are the best performing models. Homogeneous and Heterogeneous models with Node2Vec as the base embedding model have the best performance; HomoLT is the winner in AUC and PRAUC, and HeterLT is the winner in NDCG. In fact, the NDCG value of HeterLT is .8072, which is better than the same for BCGD by 0.19 units, more than 30% improvement! For AUC metric, the improvement is around 0.04 unit, and for PRAUC metric the improvement is around 0.06 unit for both the homogeneous and heterogeneous models. In NIPS dataset, RET model has a mixed performance compared to BCGD, the former wins in AUC and PRAUC, but loses in NDCG, both marginally. An explanation of sub-optimal performance by the retrofitted model in this dataset may be because of its large number of time snapshots (17); because of this, the vectors of the last time snapshot, which are obtained by 16 iterations of retrofitting of the base embedding vectors of first snapshot, may

have wandered away from their optimal position. Another explanation is that in this dataset the number of unique edges (4733) is quite close to the number of total interaction (5461); i.e., edges are not repeated so each new snapshot is very different than the previous snapshots and retrofitting may not the ideal approach for capturing the temporal smoothness of this dataset. HepPH dataset also has the same behavior as NIPS (results are available in the same table). In this dataset the best performing model is HeterLT with Deepwalk as the base embedding. In fact, for the HepPH dataset (see Table 3.2c), HeterLT has more than 30% improvement over the BCGD model in NDCG metric, and around 5% improvement in two other metrics. For this dataset also, RET has mixed performance with respect to BCGD. The suboptimal performance of retrofitting models may be due to the very small ratio of the number of distinct edges and the total number of interactions.

3.6.2 Link Prediction in Messaging Network

The results of the messaging datasets are shown in Table 3.3a, 3.3b and 3.3c.

For Email-EU, all of our proposed methods perform better than the BCGD model by a substantial margin; HeterLT (LINE) performs the best in all three metrics combined. For example, HeterLT (LINE) model improves the NDCG value of BCGD by 0.38 units, from 0.6154 to 0.9959! Similar large improvements can also be seen in the other two metrics (please see Table 3.3a for the detailed results). For this dataset, retrofitting models also perform substantially better than BCGD. A possible explanation is the high ratio of distinct edge and the number of interactions, i.e., the earlier edges are repeated in later iterations, so the retrofitting based temporal smoothness of the node vectors are sufficient for capturing the network dynamics in this dataset. CollegeMsg dataset (see Table 3.3b) also has similar behavior with HeterLT (LINE) as the winner among all, again with substantial performance gain (around 20% to 30% improvement of performance value across all three metrics). For this dataset, retrofitting results are poor which could be due to small ratio of

Table 3.3.: Performance of seven of our homogeneous models, 28 of our heterogeneous models, and 7 of our retrofitted models on messaging datasets. The highlighted results are statistically significant over the baseline with $p < 0.001$. For fair comparison, we set the latent dimension size to 64.

(a) Email-EU

Method	AUC ± sd	AUPRC ± sd	$NDCG_P$ ± sd
BCGD	0.6215 ± 0.01	0.5946 ± 0.02	0.6154 ± 0.13
RET (LINE)	0.9049 ± 0.005	0.9009 ± 0.009	0.9725 ± 0.02
HomoLT (LINE)	0.8789 ± 0.009	0.8694 ± 0.01	0.9705 ± 0.01
HeterLT (LINE, wct)	**0.9211 ± 0.008**	**0.9283 ± 0.006**	**0.9923 ± 0.008**

(b) CollegeMsg

Method	AUC ± sd	AUPRC ± sd	$NDCG_P$ ± sd
BCGD	0.6663 ± 0.01	0.6691 ± 0.02	0.7266 ± 0.06
RET (PCA)	0.6291 ± 0.01	0.6435 ± 0.02	0.8381 ± 0.06
HomoLT (LINE)	0.7460 ± 0.01	0.7788 ± 0.01	0.9571 ± 0.01
HeterLT (LINE, wct)	**0.7517 ± 0.01**	**0.7913 ± 0.008**	**0.9685 ± 0.01**

(c) SMS-A

Method	AUC ± sd	AUPRC ± sd	$NDCG_P$ ± sd
BCGD	0.7350 ± 0.003	0.7770 ± 0.003	0.9312 ± 0.01
RET (Node2Vec)	**0.7737 ± 0.007**	**0.8089 ± 0.006**	**1.00 ± 0.00**
HomoLT (DeepWalk)	0.6306 ± 0.005	0.6861 ± 0.006	0.9850 ± 0.01
HeterLT (DeepWalk, avg)	0.6413 ± 0.03	0.6969 ± 0.02	0.99 ± 0.03

Table 3.4.: Performance of seven of our homogeneous models, 28 of our heterogeneous models, and 7 of our retrofitted models, on social network datasets. The highlighted results are statistically significant over the baseline with $p < 0.001$. For fair comparison, we set the latent dimension size to 64.

(a) Facebook

Method	AUC ± sd	AUPRC ± sd	$NDCG_P$ ± sd
BCGD	0.6431 ± 0.002	0.6576 ± 0.003	0.3694 ± 0.02
RET (Node2Vec)	**0.859 ± 0.004**	**0.859 ± 0.005**	0.9817 ± 0.006
HomoLT (DeepWalk)	0.6061 ± 0.004	0.6141 ± 0.004	0.7113 ± 0.02
HeterLT (LINE, avg)	0.6258 ± 0.02	0.6983 ± 0.02	**0.9823 ± 0.03**

(b) Facebook2

Method	AUC ± sd	AUPRC ± sd	$NDCG_P$ ± sd
BCGD	0.7537 ± 0.02	0.7190 ± 0.02	0.7957 ± 0.05
RET(PCA)	**0.8202 ± 0.01**	**0.8144 ± 0.01**	**0.9519 ± 0.02**
HomoLT (DeepWalk)	0.7252 ± 0.02	0.7144 ± 0.02	0.8422 ± 0.03
HeterLT (Node2Vec, linear)	0.7792 ± 0.01	0.7788 ± 0.01	0.9200 ± 0.01

distinct edge vs interaction count, and large number of temporal snapshots. For SMS-A dataset (results is shown on Table 3.3c), retrofitted method with Node2Vec as the initial representation generator performs the best. The model achieves 0.04 unit improvement in AUC, 0.03 unit in AUPRC and 0.07 unit in NDCG over BCGD. For this dataset, the ratio of distinct edge vs interaction count is higher than the CollegeMsg dataset, which could be a reason for the RET model to perform better.

3.6.3 Link Prediction in Social Network

Results on social networks are shown in Table 3.4. For Facebook and Facebook2, the retrofitted method with Node2Vec and PCA performs better than BCGD. RET (Node2Vec) performs the best in Facebook dataset. The model gains 0.21 unit improvement over BCGD in AUC metric, 0.20 unit in AUPRC, and around 0.60 unit in NDCG metric. For Facebook2 dataset, RET (PCA) performs the best. RET (PCA) improves 0.07 unit over BCGD in AUC metric, 0.10 unit in AUPRC met-

ric, and 0.16 unit in the NDCG metric. Heterogeneous method with Node2Vec as the representation generator along with linear or exponential smoothing operator also performs better than BCGD achieving around 0.06 unit improvement in AUPRC, and 0.13 unit improvement in NDCG metric. The Facebook datasets have small number of timestamps, which is a likely reason for RET model to perform better than the transformation based models on these datasets.

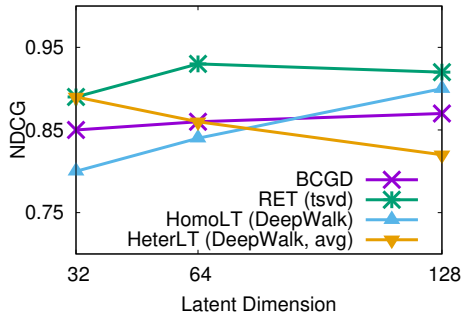
3.6.4 Effect of Latent Dimensions

In Figure 3.3, we compare the performance of the baseline model (BCGD) with our best models for three different dataset (one representative dataset from each network group) and over three different latent dimension: 32, 64, and 128 using the NDCG metric. In all three datasets, the performance of multiple of our models is consistently better than BCGD over all three latent dimensions. BCGD shows marginal improvement as the latent dimension increases. Most of our models stay flat or increase slowly as the dimension size increases because our models already achieve very high NDCG even in the low dimension (at dimension size, 32). The performance of RET (tsvd) and HeterLT (Node2Vec, avg) decease in DBLP2 and Facebook2, respectively as the dimension size increases from 64 to 128. However, the decrement is only around 2 points in both cases. To summarize, our models show robust and consistently better performance than BCGD for a widely varying number of latent dimensions.

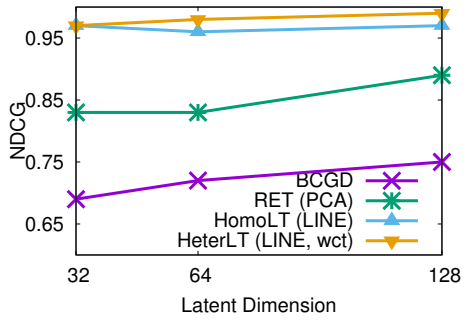
3.7 Dynamic Network Visualization

To demonstrate the smooth transition of the nodes between snapshots using the retrofitted model, we created an animation using the Enron dataset. The full video can be seen at: `https://www.youtube.com/watch?v=FtcaF0cv6iU`. The dataset was divided into 18 equal-length time-stamps and the retrofitted model was applied. We

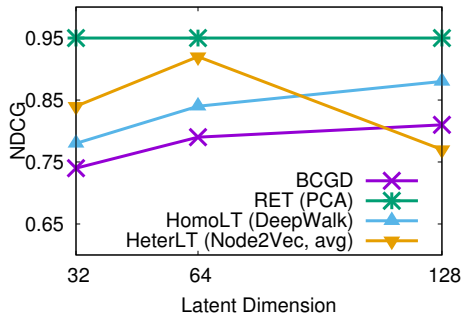
[1]https://www.cs.cmu.edu/ enron/



(a) DBLP2

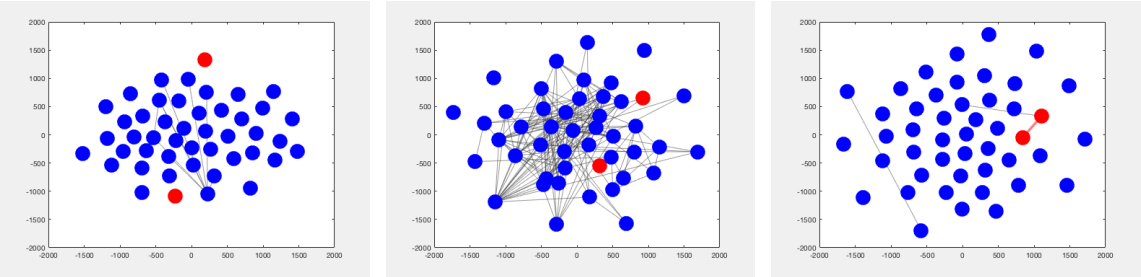


(b) CollegeMsg



(c) Facebook2

Fig. 3.3.: Effect on latent dimension. We evaluate the performance of our best models along with the baseline model to study the effect of different latent dimensions (in our case, 32, 64, and 128).



(a) Snapshot 1 (b) Snapshot 11 (c) Snapshot 18

Fig. 3.4.: 2-D dynamic network vizualization of the Enron network. Nodes represent people in the Enron email network and edges represent an email between two people. We highlight the red nodes to show how our retrofitted model smoothly brings two nodes closer together before they form their first edge.

then used TSNE to project the nodes into a 2-dimensional space so that they can be visualized as frames of an animation. Figure 3.4 shows the 2-dimensional network at time-stamps $t = 1, 11, 18$. The animation demonstrates how the retrofitting model brings two faraway nodes (red colored) in close proximity over time before an edge is created between them in the final snapshot.

3.8 Chapter Summary

In this work we propose models for learning latent embedding vectors of vertices for all different temporal snapshots of a dynamic network. The proposed models exploit temporal smoothing either at the node-level through retrofitting, or at the network level through smooth linear transformation. Extensive experiments over 9 dynamic networks from various domains show that our proposed models generate superior vertex embedding than existing state-of-the-art methods for solving the task of temporal link prediction. Visualization of embedding vectors over time shows the utility of the retrofitted model for tracking the vertices over time to understand the evolution patterns of a dynamic network.

4. DyLink2Vec: EFFECTIVE FEATURE REPRESENTATION FOR LINK PREDICTION IN DYNAMIC NETWORKS

4.1 Introduction

Understanding the dynamics of an evolving network is an important research problem with numerous applications in various fields, including social network analysis [123], information retrieval [124], recommendation systems [125], and bioinformatics [126]. A key task towards this understanding is to predict the likelihood of a future association between a pair of nodes, having the knowledge about the current state of the network. This task is commonly known as the *link prediction* problem. Since, its formal introduction to the data mining community by Liben-Nowell et al. [127] about a decade ago, this problem has been studied extensively by many researchers from a diverse set of disciplines [128–131]. Good surveys [132, 133] on link prediction methods are available for interested readers.

The majority of the existing works on link prediction consider a static snapshot of the given network, which is the state of the networks at a given time [127,128,130,134]. Nevertheless, for many networks, additional temporal information such as the time of link creation and deletion is available over a time interval; for example, in an on-line social or a professional network, we usually know the time when two persons have become friends; for collaboration events, such as, a group performance or a collaborative academic work, we can extract the time of the event from an event calendar. The networks built from such data can be represented by a *dynamic network*, which is a collection of temporal snapshots of the network. The link prediction[1] task on such

[1]Strictly speaking, this task should be called as *link forecasting* since the learning model is not trained on the links at time t; however, we refer it as link prediction due to the popular usage of this term in the data mining literature.

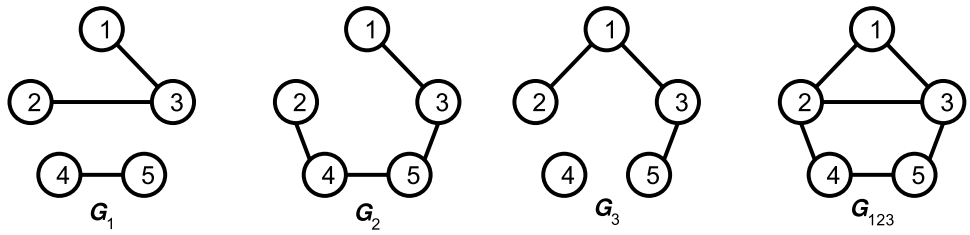


Fig. 4.1.: A toy dynamic network. G_1, G_2 and G_3 are three snapshots of the Network. G_{123} is constructed by superimposing G_1, G_2 and G_3.

a network is defined as follows: *for a given pair of nodes, predict the link probability between the pair at time* $t+1$ *by training the model on the link information at times* $1, 2, \cdots, t$.

Link prediction methods for static networks fail to take advantage of the temporal link formation patterns that are manifested by the sequence of multiple temporal snapshots. For illustration, let us consider a toy dynamic network having three temporal snapshots G_1, G_2 and G_3 (see Figure 4.1). A static link prediction which only considers the latest time stamp G_3 forfeits the temporal signals that are available from prior snapshots G_1 and G_2. Thus, it is oblivious of the fact that the edge $(4, 5)$ once existed. On the other hand, if the static link prediction method runs on a superposition [130] of all the available snapshots (G_{123}), it fails to preserve the temporal variation in the dataset. For example, the superimposed static snapshot fails to distinguish the link strength between the edges $(3, 5)$ and $(4, 5)$—even though both edges appear twice in G_1, G_2 and G_3, the recency of $(3, 5)$ may make it more likely to re-appear than $(4, 5)$.

A key challenge of link prediction in a dynamic setting is to find a suitable feature representation of the node-pair instances which are used for training the prediction model. For the static setting, various topological metrics (common neighbors, Adamic-Adar, Jaccard's coefficient) are used as features, but they cannot be extended easily for the dynamic setting having multiple snapshots of the network. In fact, when multiple (say t) temporal snapshots of a network are provided, each of these scalar

features becomes a t-size sequence. Flattening the sequence into a t-size vector distorts the inherent temporal order of the features. Güneş et al. [37] overcome this issue by modeling a collection of time series, each for one of the topological features. But such a model fails to capture signals from the neighborhood topology of the edges. There exist few other works on dynamic link prediction, which use probabilistic (non-parametric) and matrix factorization based models. These works consider a feature representation of the nodes and assume that having a link from one node to another is determined by the combined effect of all pairwise node feature interactions [41, 135, 136]. While this is a reasonable assumption to make, the accuracy of such models are highly dependent on the quality and availability of the node features, as well as the validity of the above assumption.

There exist a growing list of recent works which use unsupervised methodologies for finding metric embedding of nodes in a graph [12, 14, 43]. The main idea of such methods is to discover latent dependency among the graph vertices and find metric embedding of vertices that captures those relationships. The majority of these works use training methods inspired from neural-network language modeling, such as skip-gram with negative sampling. However, no such work exists for finding feature representation of node-pair instances for the purpose of link prediction in a dynamic network.

In this work, we propose DyLink2Vec (DyLink2Vec stands for **Link to Vec**tor in a **Dy**namic network. The proposed methodologies maps node-pairs (links) in a dynamic network to a vector representation), a novel learning method for obtaining a feature representation of node-pair instances, which is specifically suitable for the task of link prediction in a dynamic network. DyLink2Vec considers the feature learning task as an optimal coding problem, such that the optimal code of a node-pair is the desired feature representation. The learning process can be considered as a two-step compression-reconstruction step, where the first step compresses the input representation of a node-pair into a code by a non-linear transformation, and the second step reconstructs the input representation from the code by a reverse process

and the optimal code is the one which yields the least amount of reconstruction error. The input representation of a node-pair is constructed using the connection history and the neighborhood information of the corresponding nodes (details in Section 4.4). After obtaining an appropriate feature representation of the node-pairs, a standard supervised learning technique can be used (we use AdaBoost) for predicting link states at future times in the given dynamic network.

Below we summarize our contributions in this work:

- We propose DyLink2Vec for finding metric embedding of node-pairs for the task of link prediction over a dynamic network.
- We validate the effectiveness of DyLink2Vec node-pair embedding by utilizing it for link prediction on four real-life dynamic networks.
- We compare the performance of DyLink2Vec embedding based dynamic link prediction model with multiple state-of-the-art methods. Our comparison results show that the proposed method is significantly superior than all the competing methods.

The paper is organized as follows. In Section 4.2 we discuss related work. Section 4.3 defines the problem. In Section 4.4 we discuss the proposed learning method DyLink2Vec. In Section 4.5 we detail the link prediction method using DyLink2Vec. Section 4.6 presents the experimental results to validate the effectiveness of our method. Finally, Section 4.7 concludes the paper.

4.2 Related Work

In recent years, the link prediction problem has been studied using a multitude of methodologies. The earliest link prediction methodologies use topological features in a supervised classification setting [127, 132]. More recent methodologies use matrix factorization based approach [134]. Such methodologies learn latent node representation and predict link strength by the dot product of the latent vectors of

corresponding nodes. The objective function of these methods may contain appropriate penalty terms for regularization, and also terms for explicit node and edge features (if available). Recently, Bayesian nonparametric latent feature models have also been proposed for link prediction [128]. Unfortunately, all the above methods fail to capture the temporal evolution of the network on a dynamic network setting.

A few methods have been developed for link prediction on dynamic networks. The method proposed by Güneş et al. [37] capture temporal patterns in a dynamic network using a collection of time-series on topological features. But this approach fails to capture signals from neighborhood topology, as each time-series model is trained on a separate t-size feature sequence of a node-pair. Matrix and tensor factorization based solutions are presented in [41]. Given a three dimensional tensor representation of a dynamic network, the proposed methods use CANDECOMP/PARAFAC (CP) decomposition to capture structural and temporal patterns in the dynamic network. We observe that these methods work well for smaller network, but their prediction performance becomes worse as the network grow larger.

The nonparametric link prediction method presented in [135] uses features of the node-pairs, as well as the local neighborhood of node-pairs. This method works by choosing a probabilistic model based on features (common neighbor and last time of linkage) of node-pairs. Stochastic block model based approaches [42,136] divide nodes in a network into several groups and generates edges with probabilities dependent on the group membership of participant nodes. While probabilistic model based link prediction performs well on small networks, they become computationally prohibitive for large networks. A deep learning based solution proposed by Li et al. [40] uses a collection of Restricted Boltzmann Machines with neighbor influence for link prediction in dynamic networks. Tylenda et al. [38] proposed time-aware link prediction method for evolving social networks with hyper-edges.

4.3 Problem Definition

Let $G(V, E)$ be an undirected network, where V is the set of nodes and E is the set of edges $e(u, v)$ such that $u, v \in V$. A dynamic network is represented as a sequence of snapshots $\mathbb{G} = \{G_1, G_2, \ldots, G_t\}$, where t is the number of time stamps for which we have network snapshots and $G_i(V_i, E_i)$ is a network snapshot at time stamp $i : 1 \leq i \leq t$. In this work, we assume that the vertex set remains the same across different snapshots, i.e., $V_1 = V_2 = \cdots = V_t = V$. However, the edges appear and disappear over different time stamps. We also assume that, in addition to the link information, no other attribute data for the nodes or edges are available.

Adjacency matrix representation of a network snapshot G_i is represented by a symmetric binary matrix $\mathbf{A}_i(n \times n)$, where n is the number of vertices in G_i. For two vertices u and v, $\mathbf{A}_i(u, v) = \mathbf{A}_i(v, u) = 1$, if an edge exists between them in G_i, and 0 otherwise. The adjacency vector of a node u at snapshot G_i is a $1 \times n$ row vector defined as $\mathbf{a}_i^u = \mathbf{A}_i(u, 1 : n)$.

Problem Statement: Given a sequence of snapshots $\mathbb{G} = \{G_1, G_2, \ldots, G_t\}$ of a network, the task of metric embedding of the node-pairs (u, v) is to obtain a vector $\alpha^{uv} \in \mathbb{R}^l$ (l is the dimensionality of embedding) such that node-pairs having similar local structures across different time snapshots are packed together in the embedding. Once such metric embedding of a node-pair (u, v) is obtained, we use it as the feature representation of this node-pair while predicting the link status between u and v in G_{t+1}. Note that, we assume that no link information regarding the snapshot G_{t+1} is available, except the fact that G_{t+1} contains the identical set of vertices.

4.4 Metric Embedding of Node-Pairs

A key challenge for dynamic link prediction is choosing an effective metric embedding for a given node-pair. Earlier works construct feature vector by adapting various topological similarity metrics for static link prediction or by considering the feature

values of different snapshots as a time-series. DyLink2Vec, on the other hand, learns the feature embedding of the node-pairs by using an optimization framework, considering both network topology and link history. Assume a node-pair (u, v) for which we are computing the metric embedding $\alpha^{uv} \in \mathbb{R}^d$. Since we want α^{uv} to facilitate link prediction on dynamic graphs, the vector α^{uv} must capture two aspects that influence the possibility of link between u and v in G_{t+1}. The first aspect is the similarity between u and v in terms of graph topology across different timestamps, and the second aspect is the history of collaboration between u and v—both in the graph snapshots $G_1, \cdots, G_t$.

Consideration of first aspect requires to impart topological similarity signals between u and v into the desired embedded vector α^{uv} by considering u and v's relation across all the timestamps. To fulfill this objective, we start with a feature vector, $\mathbf{a}^{uv}_{[1,t]}$ of size nt for a node pair (u, v) by taking the element-wise summation of adjacency vectors of u and v over all the timestamps. Thus, for a snapshot G_i, the adjacency summation vector is $\mathbf{a}^{uv}_i = \mathbf{a}^u_i + \mathbf{a}^v_i$, and the entire feature vector is the concatenation of $\mathbf{a}^{uv}_i$'s from a continuous set of network snapshots, i.e., $\mathbf{a}^{uv}_{[1,t]} = \mathbf{a}^{uv}_1 \mid\mid \mathbf{a}^{uv}_2 \mid\mid \ldots \mid\mid \mathbf{a}^{uv}_t$. Here, the symbol $\mid\mid$ represents concatenation of two horizontal vectors (e.g., 0 1 0 $\mid\mid$ 0.5 0 1 = 0 1 0 0.5 0 1).

Example: Consider the toy dynamic network shown in Figure 4.2. The dynamic network $\mathbb{G} = \{G_1, G_2, G_3, G_4\}$ has four snapshots. The task is to predict the edges in snap G_5 (not shown in this figure). The set of nodes does not change over time ($V_1 = V_2 = \cdots = V_5$). In Figure 4.3(a) we show the adjacency matrix of the dynamic network $\mathbb{G}$ at time-stamp 1. In Figure 4.3(b), we show the computation of adjacency vectors of two node-pairs, namely $\mathbf{a}^{23}_1$ and $\mathbf{a}^{45}_1$. ■.

The second aspect, history of collaboration between a node-pair is captured by taking cumulative sum of link history, weighted by a time decay function.

$$\mathbf{wclh}^{uv}_{[1,t]} = CumSum(\mathbf{wlh}^{uv}_{[1,t]})$$

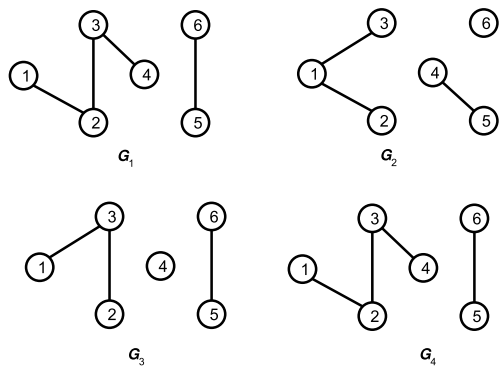


Fig. 4.2.: A toy dynamic network $\mathbb{G}$ with four snapshots G_1, G_2 , G_3 and G_4. Note that the number of nodes remains constant (6) even though the links (edges) may change over time.

	1	2	3	4	5	6
$\mathbf{a}_1^1$	0	1	0	0	0	0
$\mathbf{a}_1^2$	1	0	1	0	0	0
$\mathbf{a}_1^3$	0	1	0	1	0	0
$\mathbf{a}_1^4$	0	0	1	0	0	0
$\mathbf{a}_1^5$	0	0	0	0	0	1
$\mathbf{a}_1^6$	0	0	0	0	1	0

(a)

1	0	1	0	0	0	$\mathbf{a}_1^2$
0	1	0	1	0	0	$+\mathbf{a}_1^3$
1	1	1	1	0	0	$\mathbf{a}_1^{23}$
0	0	1	0	0	0	$\mathbf{a}_1^4$
0	0	0	0	0	1	$+\mathbf{a}_1^5$
0	0	1	0	0	1	$\mathbf{a}_1^{45}$

(b)

Fig. 4.3.: (a) Adjacency matrix $\mathbf{A}_1$. Each row represents adjacency vector of the corresponding node (b) Computation of node-pair adjacency vector $\mathbf{a}_1^{23}$ and $\mathbf{a}_1^{45}$.

Here $\mathbf{wlh}^{uv}_{[1,t]} = w_1 \cdot \mathbf{A}_1(u,v) \,||\, w_2 \cdot \mathbf{A}_2(u,v) \,||\, \ldots \,||\, w_t \cdot \mathbf{A}_t(u,v)$ and $w_i = i/t$ is the time decay function. Time decay function w_i prioritize more recent linkage information, while cumulative sum rewards reappearance of links (between u and v) over different time snapshots.

Finally, the feature vector for a node-pair (u,v), $\mathbf{e}^{uv}$, is the concatenation of $\mathbf{a}^{uv}_{[1,t]}$ and $(\mathbf{wclh}^{uv}_{[1,t]})$; i.e., $\mathbf{e}^{uv} = \mathbf{a}^{uv}_{[1,t]} \,||\mathbf{wclh}^{uv}_{[1,t]}$. DyLink2Vec's optimization framework converts $\mathbf{e}^{uv}$ to the optimal feature representation α^{uv} by using a non-linear transformation function h discussed in Section 4.4. Note that, through h, the proposed method models complex functions of the entries in $\mathbf{e}^{uv}$, which makes the embedded feature vector α^{uv} very effective for link prediction in dynamic network.

Our proposed method is different—both, in methodologies and also in objective—from the existing works [12, 14] which construct metric embedding of the vertices of a network. Existing works find embedding of a vertex from a static network, whereas we find embedding of a node-pair from a dynamic network. The learning method of the existing works follow language model, whereas our method follows a compression-reconstruction framework which preserves higher-order neighborhood and link history patterns of the node-pair in its embedded representation. Below we discuss the compression-reconstruction framework which yields the optimal metric embedding through a principled approach.

Optimization Framework for DyLink2Vec: In this section, we discuss the optimization framework which obtains the optimal metric embedding of a node pair by learning an optimal coding function h. For this learning task, let's assume $\widehat{\mathbf{E}}$ is the training dataset matrix containing a collection of node-pair feature vectors. Each row of this matrix represents a node-pair (say, u and v) and it contains the feature vector $\mathbf{e}^{uv}$ which stores information about neighborhood and link history, as we discussed earlier. The actual link status of the node-pairs in $\widehat{\mathbf{E}}$ in G_{t+1} is not used for the learning of h, so the metric embedding process is unsupervised. In subsequent discussion, we write $\mathbf{e}$ to represent an arbitrary node pair vector in $\widehat{\mathbf{E}}$.

Now, the coding function h compresses $\mathbf{e}$ to a code vector α of dimension l, such that $l < k$. Here l is a user-defined parameter which represents the code length and k is the size of feature vector. Many different coding functions exist in the dimensionality reduction literature, but for DyLink2Vec we choose the coding function which incurs the minimum reconstruction error in the sense that from the code α we can reconstruct $\mathbf{e}$ with the minimum error over all $\mathbf{e} \in \widehat{\mathbf{E}}$. We frame the learning of h as an optimization problem, which we discuss below through two operations: Compression and Reconstruction.

Compression: It obtains α from $\mathbf{e}$. This transformation can be expressed as a nonlinear function of linear weighted sum of the entries in vector $\mathbf{e}$.

$$\alpha = f(\mathbf{W}^{(c)}\mathbf{e} + \mathbf{b}^{(c)}) \tag{4.1}$$

$\mathbf{W}^{(c)}$ is a $(k \times l)$ dimensional matrix. It represents the weight matrix for compression and $\mathbf{b}^{(c)}$ represents biases. $f(\cdot)$ is the Sigmoid function, $f(x) = \frac{1}{1+e^{-x}}$.

Reconstruction: It performs the reverse operation of compression, i.e., it obtains $\mathbf{e}$ from α (which was constructed during the compression operation).

$$\beta = f(\mathbf{W}^{(r)}\alpha + \mathbf{b}^{(r)}) \tag{4.2}$$

$\mathbf{W}^{(r)}$ is a matrix of dimensions $(l \times k)$ representing the weight matrix for reconstruction, and $\mathbf{b}^{(r)}$ represents biases.

The optimal coding function h constituted by the compression and reconstruction operations is defined by the parameters $(\mathbf{W}, \mathbf{b}) = (\mathbf{W}^{(c)}, \mathbf{b}^{(c)}, \mathbf{W}^{(r)}, \mathbf{b}^{(r)})$. The objective is to minimize the reconstruction error. Reconstruction error for a neighborhood based feature vector (e) is defined as, $J(\mathbf{W}, \mathbf{b}, \mathbf{e}) = \frac{1}{2} \parallel \beta - \mathbf{e} \parallel^2$. Over all possi-

ble feature vectors, the average reconstruction error augmented with a regularization term yields the final objective function $J(\mathbf{W}, \mathbf{b})$:

$$J(\mathbf{W}, \mathbf{b}) = \frac{1}{|\widehat{\mathbf{E}}|} \sum_{\mathbf{e} \in \widehat{\mathbf{E}}} (\frac{1}{2} \parallel \beta^{uv} - \mathbf{e}^{uv} \parallel^2) + \frac{\lambda}{2} (\parallel \mathbf{W}^{(c)} \parallel_F^2 + \parallel \mathbf{W}^{(r)} \parallel_F^2) \tag{4.3}$$

Here, λ is a user assigned regularization parameter, responsible for preventing over-fitting. $\parallel \cdot \parallel_F$ represents the Frobenius norm of a matrix. In this work we use $\lambda = 0.1$.

To this end, we discuss the motivation of our proposed optimization framework for learning the coding function h. Note that, the dimensionality of α is much smaller than $\mathbf{e}$, so the optimal compression of the vector $\mathbf{e}$ must extract patterns composing of the entries of $\mathbf{e}$ and use them as high-order latent feature in α. In fact, the entries in $\mathbf{e}$ contain the neighborhood (sum of adjacency vector of the node pair) and link history of a node-pair for all the timestamps; for a real-life network, this vector is sparse and substantial compression is possible incurring small loss. Through this compression the coding function h learns the patterns that are similar across different node-pairs (used in $\widehat{\mathbf{E}}$). Thus the function h learns a metric embedding of the node-pairs that packs node-pairs having similar local structures in close proximity in the embedded feature space. Although function h acts as a black-box, it captures patterns involving neighborhood around a node pair across various time stamps, which obviates the manual construction of a node-pair feature—a cumbersome task for the case of a dynamic network.

Optimization

The training of optimal coding defined by parameters $(\mathbf{W}, \mathbf{b})$ begins with random initialization of the parameters. Since the cost function $J(\mathbf{W}, \mathbf{b})$ defined in Equation (4.3) is non-convex in nature, we obtain a local optimal solution using the gradient

descent approach. Such approach usually provides practically useful results (as shown in the Section 4.6). The parameter updates of the gradient descent are similar to the parameter updates for optimizing Auto-encoder in machine learning. One iteration of gradient descent updates the parameters using following equations:

$$
\begin{aligned}
W_{ij}^{(c)} &= W_{ij}^{(c)} - \sigma \frac{\partial}{\partial W_{ij}^{(c)}} J(W, b) \\
W_{ij}^{(r)} &= W_{ij}^{(r)} - \sigma \frac{\partial}{\partial W_{ij}^{(r)}} J(W, b) \\
b_i^{(c)} &= b_i^{(c)} - \sigma \frac{\partial}{\partial b_i^{(c)}} J(W, b) \\
b_i^{(r)} &= b_i^{(r)} - \sigma \frac{\partial}{\partial b_i^{(r)}} J(W, b)
\end{aligned}
\tag{4.4}
$$

Here, l appropriately identifies the weight and bias parameters $l \in \{1, 2\}$. σ is the learning rate. $W_{ij}^{(1)}$ is the weight of connection between node j of the input layer to node i of the hidden layer.

Now, from Equation (4.3), the partial derivative terms in equations (4.4) can be written as,

$$
\begin{aligned}
\frac{\partial}{\partial W_{ij}^{(c)}} J(W, b) &= \frac{1}{|\widehat{\mathbf{E}}|} \sum_{\mathbf{e} \in \widehat{\mathbf{E}}} \frac{\partial}{\partial W_{ij}^{(c)}} J(\mathbf{W}, \mathbf{b}, \mathbf{e}) + \lambda W_{ij}^{(c)} \\
\frac{\partial}{\partial W_{ij}^{(r)}} J(W, b) &= \frac{1}{|\widehat{\mathbf{E}}|} \sum_{\mathbf{e} \in \widehat{\mathbf{E}}} \frac{\partial}{\partial W_{ij}^{(r)}} J(\mathbf{W}, \mathbf{b}, \mathbf{e}) + \lambda W_{ij}^{(r)} \\
\frac{\partial}{\partial b_i^{(c)}} J(W, b) &= \frac{1}{|\widehat{\mathbf{E}}|} \sum_{\mathbf{e} \in \widehat{\mathbf{E}}} \frac{\partial}{\partial b_i^{(c)}} J(\mathbf{W}, \mathbf{b}, \mathbf{e}) \\
\frac{\partial}{\partial b_i^{(r)}} J(W, b) &= \frac{1}{|\widehat{\mathbf{E}}|} \sum_{\mathbf{e} \in \widehat{\mathbf{E}}} \frac{\partial}{\partial b_i^{(r)}} J(\mathbf{W}, \mathbf{b}, \mathbf{e})
\end{aligned}
\tag{4.5}
$$

The optimization problem is solved by computing partial derivative of cost function $J(\mathbf{W}, \mathbf{b}, \mathbf{e})$ using the back propagation approach [137].

Once the optimization is done, the metric embedding of any node-pair (u, v) can be obtained by taking the outputs of compression stage (Equation (4.1)) of the trained optimal coding $(\mathbf{W}, \mathbf{b})$.

$$\alpha^{uv} = f(\mathbf{W}^{(c)}\mathbf{e}^{uv} + \mathbf{b}^{(c)}) = h(\mathbf{e}^{uv}) \tag{4.6}$$

Complexity Analysis

We use Matlab implementation of optimization algorithm L-BFGS (Limited-memory Broyden-Fletcher-Goldfarb-Shanno) for learning optimal coding. We execute the algorithm for a limited number of iterations to obtain unsupervised features within a reasonable period of time. Each iteration of L-BFGS executes two tasks for each node-pair: back-propagation to compute partial differentiation of cost function, and change the parameters $(\mathbf{W}, \mathbf{b})$. Therefore, the time complexity of one iteration is $O(|NP_t|kl)$. Here, NP_t is the set on node-pairs used to construct the training dataset $\widehat{\mathbf{E}}$. k is the length of $\mathbf{e}$ (dimensionality of initial edge features), and l is length of α (optimal coding).

4.5 Link Prediction Using Proposed Metric Embedding

For link prediction task in a dynamic network, $\mathbb{G} = \{G_1, G_2, \ldots, G_t\}$; we split the snapshots into two overlapping time windows, $[1, t-1]$ and $[2, t]$. Training dataset, $\widehat{\mathbf{E}}$ is feature representation for time snapshots $[1, t-1]$, the ground truth $(\widehat{\mathbf{y}})$ is constructed from G_t. DyLink2Vec learns optimal embedding $h(\cdot)$ using training dataset $\widehat{\mathbf{E}}$. After training a supervised classification model using $\widehat{\alpha}$=$h(\widehat{\mathbf{E}})$ and $\widehat{\mathbf{y}}$, prediction dataset $\overline{\mathbf{E}}$ is used to predict links at G_{t+1}. For this supervised prediction task, we experiment with several classification algorithms. Among them SVM (support vector machine) and AdaBoost perform the best.

Algorithm 2: Link Prediction using DyLink2Vec.

```
procedure LPFS^3(𝔾, t) Input  : 𝔾: Dynamic Network, t: Time steps
                       Output: ȳ: Forecasted links at time step t + 1
    Ê=NeighborhoodFeature(𝔾,1,t − 1)
    ŷ=Connectivity(G_t)
    Ē=NeighborhoodFeature(𝔾,2,t)
    h=LearningOptimalCoding(Ê)
    α̂=h(Ê)
    ᾱ=h(Ē)
    C=TrainClassifier(α̂, ŷ)
    ȳ=LinkForecasting(C, ᾱ)
    return ȳ
end procedure
```

The pseudo-code of DyLink2Vec based link prediction method is given in Algorithm 2. For training link prediction model, we split the available network snapshots into two overlapping time windows, $[1, t-1]$ and $[2, t]$. Neighborhood based features $\widehat{\mathbf{E}}$ and $\overline{\mathbf{E}}$ are constructed in Lines 2 and 4, respectively. Then we learn optimal coding for node-pairs using neighborhood features $\widehat{\mathbf{E}}$ (in Line 5). Embeddings are constructed using learned optimal coding (Lines 6 and 7) using output of compression stage (Equation 4.6). Finally, a classification model C is learned (Line 8), which is used for predicting links in G_{t+1} (Line 9).

4.6 Experiments and Results

We demonstrate the performance of DyLink2Vec using four real world dynamic network datasets: **Enron**, **Collaboration**, **Facebook1** and **Facebook2**. We show performance comparison between DyLink2Vec based link prediction method and existing state-of-the-art dynamic link prediction methodologies. Experimental results also include the discussion of DyLink2Vec's performance for varying length of time stamps in the network, and varying degree of class imbalance in training dataset. Bellow, we discuss the datasets, evaluation metrics, competing methods, implementation details and results.

4.6.1 Dataset Descriptions

Here we discuss the construction and characteristics of the datasets used for experiments.

Enron email corpus [138] consists of email exchanges between Enron employees. The Enron dataset has 11 time stamps and $16,836$ possible node-pairs; the task is to use first 10 snapshots for predicting links in the 11th snapshot. Following [136], we aggregate data into time steps of 1 week. We use the data from weeks 147 to 157 of the data trace for the experiments. The reason for choosing that window is that the snapshot of the graph at week 157 has the highest number of edges.

Collaboration dataset has 10 time stamps with author collaboration information about $49,455$ author-pairs. The Collaboration dataset is constructed from citation data containing 1.4 million papers [139]. We process the data to construct a network of authors with edges between them if they co-author a paper. Considering each year as a time stamp, the data of years 2000-2009 (10 time stamps) is used for this experiment, where the data from the first nine time stamps is used for training and the last for prediction. Since this data is very sparse, we pre-process the data to retain only the active authors, who have last published papers on or after year 2010; moreover, the selected authors participate in at least two edges in seven or more time stamps.

Facebook1 and Facebook2 are network of Facebook wall posts [121]. Each vertex is a Facebook user account and an edge represents the event that one user posts a message on the wall of another user. Both Facebook1 and Facebook2 has 9 time stamps. Facebook1 has $219,453$ node-pairs. Facebook2 is an extended version of Facebook1 dataset with $883,785$ node-pairs. For pre-processing Facebook1 we follow the same setup as is discussed in [42]; wall posts of 90 days are aggregated in one time step.

We filter out all people who are active for less than 6 of the 9 time steps, along with the people who have degree less than 30. Facebook2 is created using a similar method, but a larger sample of Facebook wall posts is used for this dataset.

4.6.2 Evaluation Metrics

For evaluating the proposed method we use two metrics, namely, area under Precision-Recall (PR) curve (PRAUC) [122] and an information retrieval metric, Normalized Discounted Cumulative Gain (NDCG). PRAUC is best suited for evaluating two class classification performance when class membership is skewed towards one of the classes. This is exactly the case for link prediction; the number of edges ($|E|$) is very small compared to the number of possible node-pairs $\binom{|V|}{2}$ In such scenarios,

area under the Precision-Recall curve (PRAUC) gives a more informative assessment of the algorithm's performance than other metrics such as, accuracy. The reason why PRAUC is more suitable for the skewed problem is that it does not factor in the count of true negatives in its calculation. In skewed data where the number of negative examples is huge compared to the number of positive examples, true negatives are not that meaningful.

We also use NDCG, an information retrieval metric (widely used by the recommender systems community) to evaluate the proposed method. NDCG measures the performance of link prediction system based on the graded relevance of the recommended links. $NDCG_k$ varies from 0.0 to 1.0, with 1.0 representing ideal ranking of edges. Here, k is a parameter chosen by user representing the number of links ranked by the method. We use $k = 50$ in all our experiments.

Some of the earlier works on link prediction have used area under the ROC curve (AUC) to evaluate link prediction works [37,140]. But recent works [141] have demonstrated the limitations of AUC and argued in favor of PRAUC over AUC for evaluation of link prediction. So we have not used AUC in this work.

4.6.3 Competing Methods for Comparison

We compare the performance of DyLink2Vec based link prediction method with methods from four categories: (1) topological feature based methods, (2) feature time series based methods [37], (3) a deep learning based method, namely DeepWalk [12], and (4) a tensor factorization based method CANDECOMP/PARAFAC (CP) [41].

Besides these four works, there are two other existing works for link prediction in dynamic network setting; one is based on deep Learning [40] (Conditional Temporal Restricted Boltzmann machine) and the other is based on a signature-based nonparametric method [135]. We did not compare with these models as implementations of their models are not readily available, besides, both of these methods have numerous parameters which will make reproducibility of their results highly improb-

able and thus, conclusion derived from such experiments may not align with true understanding of the usefulness of the methods. Moreover, none of these methods give unsupervised feature representation for node-pairs in which we claim our main contribution.

For **topological feature based methods**, we consider four prominent topological features: Common Neighbors (CN), Adamic-Adar (AA), Jaccard's Coefficient (J) and Katz measure ($Katz$). However, in existing works, these features are defined for static networks only; so we adapt these features for the dynamic network setting by computing the feature values over the collapsed[2] dynamic network.

We also combine the above four features to construct a combined feature vector of length four (**J**accard's Coefficient, **A**damic-Adar, **C**ommon Neighbors and **K**atz), which we call $JACK$ and use it with a classifier to build a supervised link prediction method, and include this model in our comparison.

Second, we compare DyLink2Vec with **time-series based** neighborhood similarity scores proposed in [37]. In this work, the authors consider several neighborhood-based node similarity scores combined with connectivity information (historical edge information). Authors use time-series of similarities to model the change of node similarities over time. Among 16 proposed methods, we consider 4 that are relevant to the link prediction task on unweighted networks and also have the best performance. TS-CN-Adj represents time-series on normalized score of Common Neighbors and connectivity values at time stamps $[1, t]$. Similarly, we get time-series based scores for Adamic-Adar (TS-AA-Adj), Jaccard's Coefficient (TS-J-Adj) and Preferential Attachment (TS-PA-Adj).

Third, we compare DyLink2Vec with **DeepWalk** [12], a latent node representation based method. We use DeepWalk to construct latent representation of nodes from the collapsed dynamic network. Then we construct latent representation of node-pairs by computing cross product of latent representation of the participating nodes. For example, if the node representations in a network are vectors of size l, then

[2]Collapsed network is constructed by superimposing all network snapshots(see Figure 4.1).

the representation of a node-pair (u, v) will be of size l^2, constructed from the cross product of u and v's representation. The DeepWalk based node-pair representation is then used with a classifier to build a supervised link prediction method. We choose node representation size $l = 2, 4, 6, 8, 10$ and report the best performance.

Finally, we compare DyLink2Vec with a tensor factorization based method, called **CANDECOMP/PARAFAC (CP)** [41]. In this method, the dynamic network is represented as a three-dimensional tensor $\mathcal{Z}(n \times n \times t)$. Using CP decomposition $\mathcal{Z}$ is factorized into three factor matrices. The link prediction score is computed by using the factor matrices. We adapted the CP link prediction method for unipartite networks; which has originally been developed for bipartite networks.

4.6.4 Implementation Details

We implemented DyLink2Vec algorithm in Matlab version $R2014b$. The learning method runs for a maximum of 100 iterations or until it converges to a local optimal solution. We use coding size $l = 100$ for all datasets[3]. For supervised link prediction step we use several Matlab provided classification algorithms, namely, AdaBoostM1, RobustBoost, and Support Vector Machine (SVM). We could use neural network classifier. But, as our main goal is to evaluate the quality of unsupervised feature representation, so, we use simple classifiers. Supervised neural network architecture may result in superior performance, but, it is out of scope of the main goal of the paper. We use Matlab for computing the feature values (CN, AA, J, Katz) that we use in other competing methods. Time-series methods are implemented using Python. We use the ARIMA (autoregressive integrated moving average) time series model implemented in Python module **statsmodels**. The DeepWalk implementation is provided by the authors of [12]. We use it to extract node features and extend

[3]We experiment with different coding sizes ranging from 100 to 800. The change in link prediction performance is not sensitive to the coding size. At most 2.9% change in PRAUC was observed for different coding sizes.

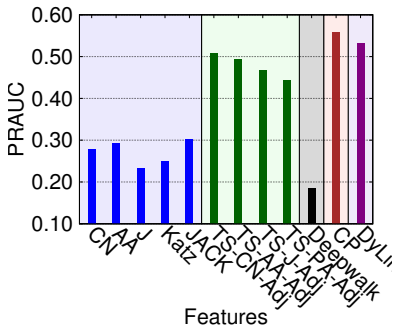
it for link prediction (using Matlab). Tensor factorization based method CP was implemented using Matlab Tensor Toolbox.

4.6.5 Performance Comparison Results with Competing Methods

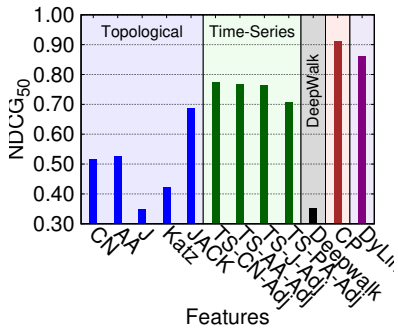
In Figure 4.4 we present the performance comparison results of DyLink2Vec based link prediction method with the four kinds of competing methods that we have discussed earlier. The figure have eight bar charts. The bar charts from the top to the bottom rows display the results for Enron, Collaboration, Facebook1 and Facebook2 datasets, respectively. The bar charts in a row show comparison results using PRAUC (left), and $NDCG_{50}$ (right) metrics. Each chart has twelve bars, each representing a link prediction method, where the height of a bar is indicative of the performance metric value of the corresponding method. In each chart, from left to right, the first five bars (blue) correspond to the topological feature based methods, the next four (green) represent time series based methods, the tenth bar (black) is for DeepWalk, the eleventh bar (brown) represents tensor factorization based method CP, and the final bar (purple) represents the proposed method DyLink2Vec.

DyLink2Vec vs. Topological

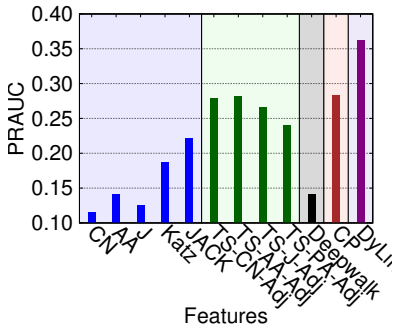
We first analyze the performance comparison between DyLink2Vec based method and topological feature based methods (first five bars). The best of the topological feature based methods have a PRAUC value of 0.30, 0.22, 0.137 and 0.14 in Enron, Collaboration, Facebook1, and Facebook2 dataset (see Figures 4.4(a), 4.4(c), 4.4(e) and 4.4(g)), whereas the corresponding PRAUC values for DyLink2Vec are 0.531, 0.362, 0.308, and 0.27, which translates to 77%, 65%, 125%, and 93% improvement of PRAUC by DyLink2Vec for these datasets. Superiority of DyLink2Vec over all the topological feature based baseline methods can be attributed to the capability of Neighborhood based feature representation to capture temporal characteristics of



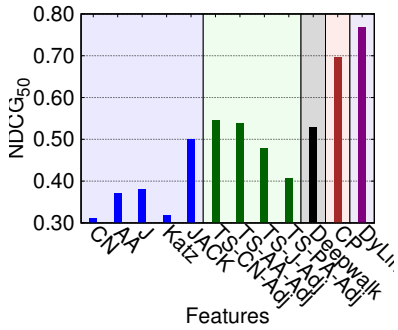
(a) Enron Network



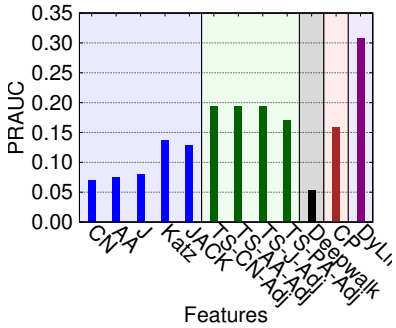
(b) Enron Network



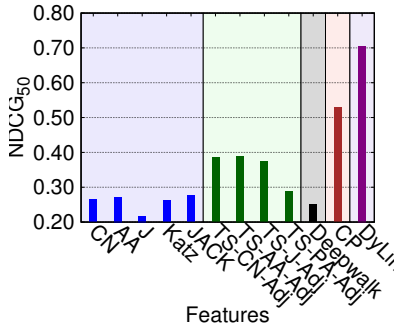
(c) Collaboration Network



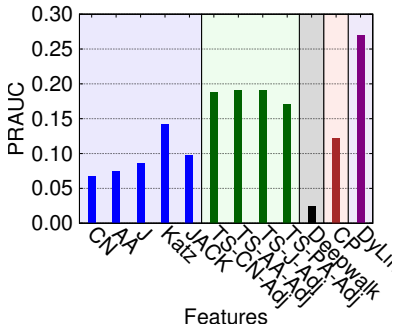
(d) Collaboration Network



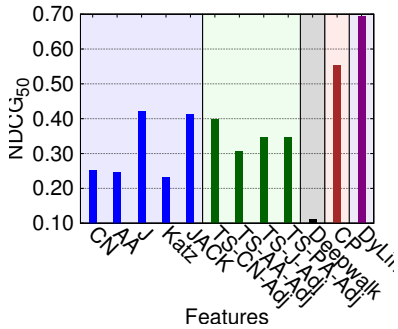
(e) Facebook1 Network



(f) Facebook1 Network



(g) Facebook2 Network



(h) Facebook2 Network

Fig. 4.4.: Comparison with competing link prediction methods. Each bar represents a methods and the height of the bar represents the value of the performance metrics. The group of bars in a chart are distinguished by color, so the figure is best viewed on a computer screen or color print.

local neighborhood. Similar trend is observed using $NDCG_{50}$ metric, see Figures 4.4(b), 4.4(d), 4.4(f) and 4.4(h).

DyLink2Vec vs. Time-Series

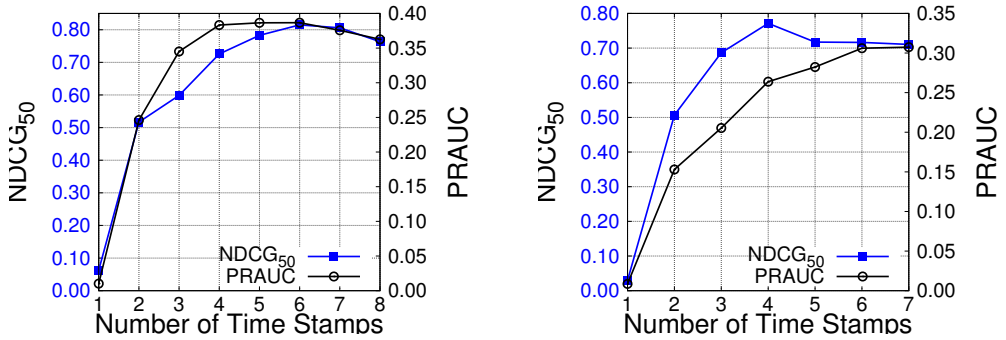
The performance of time-series based method (four green bars) is generally better than the topological feature based methods. The best of the time-series based method has a PRAUC value of 0.503, 0.28, 0.19, and 0.19 on these datasets, and DyLink2Vec's PRAUC values are better than these values by 6%, 29%, 62%, and 42% respectively. Time-series based methods, though model the temporal behavior well, probably fail to capture signals from the neighborhood topology of the node-pairs. Superiority of DyLink2Vec over Time-Series methods is also similarly indicated by information retrieval metric $NDCG_{50}$.

DyLink2Vec vs. DeepWalk

The DeepWalk based method (black bars in Figure 4.4) performs much poorly in terms of both PRAUC and $NDCG_{50}$—even poorer than the topological based method in all four datasets. Possible reason could be the following: the latent encoding of nodes by DeepWalk is good for node classification, but the cross-product of those codes fails to encode the information needed for effective link prediction.

DyLink2Vec vs. CANDECOMP/PARAFAC (CP)

Finally, the tensor factorization based method CP performs marginally better (around 5% in PRAUC, and 6% in $NDCG_{50}$) than DyLink2Vec in small and simple networks, such as Enron (see Figure 4.4(a, b)). But its performance degrades on comparatively large and complex networks, such as Collaboration, Facebook1 and Facebook2. On Facebook networks, the performance of CP is even worse than the time-series based methods (see Figures 4.4(e) and 4.4(g)). DyLink2Vec comfort-



(a) Collaboration Network (b) Facebook1 Network

Fig. 4.5.: Change in link prediction performance with number of time stamps. X-axis represents size of training window used for link prediction. Largest possible window size depends on number of time stamps available for the dataset.

ably outperforms CP on larger graphs, see Figures 4.4(c, d, e, f, g, h). In terms of PRAUC, DyLink2Vec outperforms CP by 28%, 94%, and 120% for Collaborative, Facebook1 and Facebook2 networks respectively. This demonstrates the superiority of DyLink2Vec over one of the best state-of-the-art dynamic link prediction. A reason for CP's bad performance on large graphs can be its inability to capture network structure and dynamics using high-dimensional tensors representation.

Performance across datasets

When we compare the performance of all the methods across different datasets, we observe varying performance. For example, for both the metrics, the performance of dynamic link prediction on Facebook graphs are lower than the performance on Collaboration graph, which, subsequently, is lower than the performance on Enron graph, indicating that link prediction in Facebook data is a harder problem to solve. In these harder networks, DyLink2Vec perform substantially better than all the other competing methods that we consider in this experiment.

4.6.6 Performance with Varying Length of Time Stamps

Besides comparing with competing methods, we also demonstrate the performance of DyLink2Vec with varying number of available time snapshots. For this purpose, we use DyLink2Vec with different counts of past snapshots. For example, Collaboration dataset has 10 time stamps. The task is to predict links at time stamp 10. The largest number of past snapshots we can consider for this data is 8, where $\widehat{\mathbf{E}}$ is constructed using time stamps $[1-8]$, and $\overline{\mathbf{E}}$ is constructed using time stamps $[2-9]$. The smallest number of time stamps we can consider is 1, where $\widehat{\mathbf{E}}$ is constructed using $[8-8]$, and $\overline{\mathbf{E}}$ is constructed using $[9-9]$. In this way, by varying the length of historical time stamps, we can evaluate the effect of time stamp's length on the performance of a link prediction method.

The result is illustrated in Figure 4.5. The x-axis represents the number of time stamps used by DyLink2Vec, the left y-axis represents the $NDGC_{50}$ and the right y-axis represents the PRAUC. Figures 4.5(a), and 4.5(b) corresponds to the results obtained on Collaboration and Facebook1, respectively.

We observe from Figure 4.5 that the performance ($NDGC_{50}$ and PRAUC) of link prediction increases with increased number of time stamps. But beyond a given number of snapshots, the performance increment becomes insignificant. The performance starts to deteriorate after certain number of snapshots (see Figure 4.5(a)). This may be because of the added complexity of the optimization framework with increased number of time stamps. We also observe consistent improvement of performance with the number of snapshots for the Facebook1 data (Figure 4.5(b)), which indicates that for this dataset link information from distant history is useful for dynamic link prediction. We do not show results of Enron and Facebook2 for this experiment, because of space constraint, however, they show similar trends.

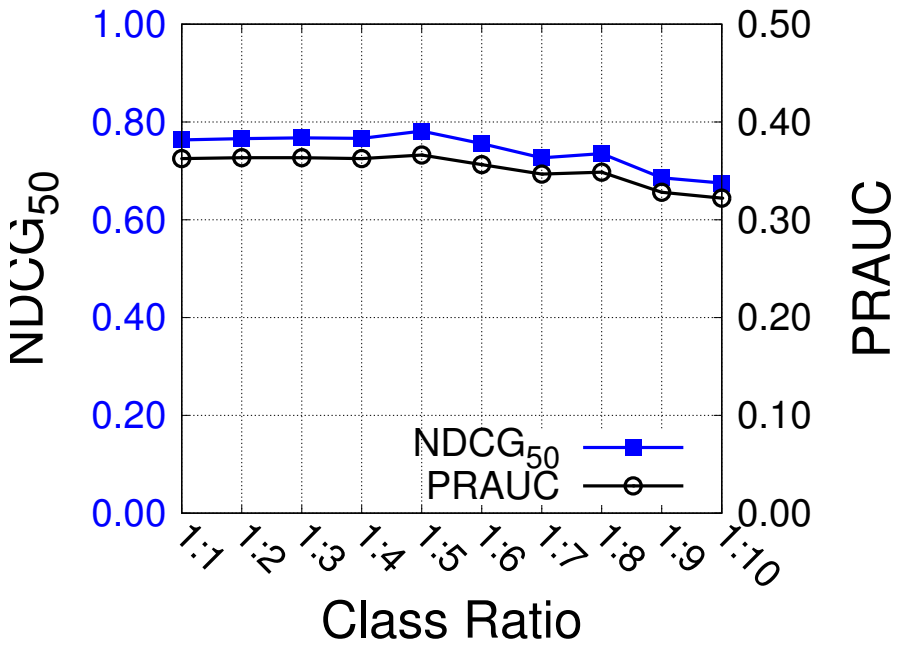


Fig. 4.6.: Effect of class imbalance in link prediction performance on Collaboration network.

4.6.7 Effect of Class Imbalance on Performance

In link prediction problem, class imbalance is a prevalent issue. The class imbalance problem appears in a classification task, when the dataset contains imbalanced number of samples for different classes. In link prediction problem, the number of positive node-pairs (with an edge) is very small compared to the number of negative node-pairs (with no edge), causing class imbalance problem.

To demonstrate the effect of class imbalance in link prediction task, we perform link prediction using DyLink2Vec embeddings with different level of class imbalance in the training dataset. We construct the training dataset by taking all positive node-pairs and sampling from the set of negative node-pairs. For a balanced dataset, the number of negative samples will be equal to the number of all positive node-pairs considered. Thus, the balanced training dataset has positive node-pairs to negative node-pairs ratio $1:1$. At this point, the only way to increase the size of the data is to increase the sample size for negative node-pairs. Consequently, the ratio of classes also increases towards negative node-pair. Figure 4.6 shows gradual decrease in link prediction performance in Collaboration network with the increase of imbalance (see ratios in X-axis) in the dataset (despite the fact that the dataset gets larger by adding negative node-pairs).

This result advocates towards the design choice of under-sampling [130] of negative node-pairs by uniformly sampling from all negative node-pairs, so that the training set has equal numbers of positive and negative node-pairs. Under-sampling, helps to mitigate the problem of class imbalance while also reducing the size of the training dataset.

4.7 Chapter Summary

In this paper, we present DyLink2Vec a learning method for obtaining feature representation of node-pairs in dynamic networks. We also give classification based link prediction method, which uses DyLink2Vec feature representation for future

link prediction in dynamic network setup. The proposed link prediction method outperforms several existing methods that are based on topological features, time series, deep learning and tensor analysis.

5. REGULARIZED AND RETROFITTED MODELS FOR LEARNING SENTENCE REPRESENTATION WITH CONTEXT

5.1 Introduction

Many sentence-level text processing tasks rely on representing the sentences using fixed-length vectors. For example, *classifying* sentences into topics using a statistical classifier like Maximum Entropy requires the sentences to be represented by vectors. Similarly, for the task of *ranking* sentences based on their importance in the text using a ranking model like LexRank [142] or SVMRank [143], one needs to first represent the sentences with fixed-length vectors. The most common approach uses a bag-of-words or a bag-of-ngrams representation, where each dimension of the vector is computed by some form of term frequency statistics (e.g., *tf*idf*).

Recently, *distributed representations*, in the form of dense real-valued vectors, learned by neural network models from unlabeled data, has been shown to outperform traditional bag-of-words representation [44]. Distributed representations encode the semantics of linguistic units and yield better generalization [47, 144]. However, most existing methods to devise distributed representation for sentences consider only the content of a sentence, and disregard relations between sentences in a text by and large [44, 48]. But, sentences rarely stand on their own in a well-formed text. On a finer level, sentences are connected with each other by certain logical relations (e.g., *elaboration*, *contrast*) to express the meaning as a whole [10]. On a coarser level, sentences in a text address a common topic, often covering multiple subtopics; i.e., sentences are also topically related [11]. Our main hypothesis in this paper is that distributed representation methods for sentences should not only consider the content of the sentence but also the contextual information in the text.

Recent studies [5–7] on learning distributed representations for *words* have shown that semantic relations between words (e.g., synonymy, hypernymy, hyponymy) encoded in semantic lexicons like WordNet [8] or Framenet [9] can improve the quality of word vectors that are trained solely on unlabeled data [5–7]. Our work in this paper is reminiscent of this line of research with a couple of crucial differences. Firstly, we are interested in representation of sentences as opposed to words, for the former such resources are not readily available. Secondly, our main goal is to incorporate extra-sentential context in some form of inter-sentence relations as opposed to semantic relations between words. These differences posit a number of new research challenges: (*i*) how can we obtain extra-sentential context that can capture semantic relations between sentences? (*ii*) how can we effectively exploit the inter-sentence relations in our representation learning model? and finally, (*iii*) how can we evaluate the quality of the vectors learned by our model?

To tackle the first issue, we explore two different methods to obtain extra-sentential context. In our first method, we consider the adjoining sentences of a sentence in a text as the context. We call this *discourse context* since it captures the actual order of the sentences. In our second method, we build a similarity network of sentences, and consider adjacent nodes (i.e., one-hop neighbors) of a sentence as its context. We call this *similarity context* since it is based on a direct measure of similarity. Our choice of network to encode context is due to the fact that networks provide flexible ways to represent relations between any pair of sentences [142, 145].

We address the second challenge by proposing two different approaches to exploit the context information. In our first approach, we first learn sentence vectors using an existing content-based model, Sen2Vec [44]. Then, we refine these vectors to encourage the new estimated vectors to be similar to the vectors of its neighbors and similar to their prior Sen2Vec representations. The refinement is performed by using an efficient iterative algorithm [6, 118]. We call this model *retrofitted* model since it retrofits the initially learned Sen2Vec vectors using contextual information. In our second approach, we alter the objective function of Sen2Vec with a regularizer or

prior that encourages neighboring sentences to have similar vector representations. We call this *regularized* model. In this approach, the vectors are learned from scratch by jointly modeling the content of the sentences and the relation between sentences.

Several recent methods also exploit contextual information to learn sentence vectors, e.g., FastSent [48] and Skip-Thought [54]. These methods learn sentence representations by predicting *content* (words or word sequences) of adjoining sentences. By learning representations that can predict contents of adjacent sentences, these methods may learn semantic and syntactic properties that are more specific to the neighbors rather than the sentence under consideration. Furthermore, these methods either make simple BOW (bag of words) assumption or disregard context when extracting a sentence vector. By contrast, our models learn sentence representations directly, and they treat adjacent sentences as atomic linguistic units.

Different approaches to evaluate sentence representation methods have been proposed in the past including sentence-level prediction tasks (e.g., sentiment classification, paraphrase identification) and sentence-pair similarity computation task [44,48]. These approaches evaluate sentences independently out of context. Instead, in this paper, we propose an evaluation setup, where extra-sentential context is available to infer sentence vectors. We evaluate our models on three different types of tasks: *classification, clustering* and *ranking*. In particular, we consider the tasks of classifying and clustering sentences into topics, and of ranking sentences in a document to create an extractive summary of the document (i.e., by selecting the top-ranked sentences). There are standard datasets with document-level topic annotations (e.g., *Reuters-21578*, *20 Newsgroups*). However, to our knowledge, no dataset exists with topic annotations at the sentence level. We generate sentence-level topic annotations from the document-level ones by selecting subsets of sentences that can be considered as representatives of the document and label them with the same document-level topic label. We use the standard DUC 2001 and 2002 datasets to evaluate our models on the summarization task, where we compare the system-generated summaries with the human-authored summaries.

Our evaluation on these tasks across multiple datasets shows impressive results for our model, which outperforms the best existing models by up to 6.29 F_1-score in classification, 12.78 V-score in clustering, 2.90 ROUGE-1 score in summarization. We found that the discourse context perform better on topic classification and clustering tasks, and similarity context performs better on summarization. We have implemented all our proposed models in a flexible software stack, which enables effective evaluation of existing or future sentence representation learning models. We make our code[1] publicly available.

The rest of the paper is organized as follows. In Section 5.3, we present a content-only model followed by two extensions of this model, which incorporate contextual information. In Sections 5.4, 5.5 and 5.6, we discuss experimental settings and results. Section 5.2 gives an account on related work, and finally, we conclude with a discussion of future work in Section 5.7.

5.2 Related Work

Recently, learning distributed representation of words, phrases, and sentences has gained a lot of attention due to its applicability and superior performance over bag-of-words (BOW) features in a wide range of text processing tasks [5–7, 44–46]. These models can be categorized into two groups: (*i*) task-agnostic or unsupervised models, and (*ii*) task-specific or supervised models. Task-agnostic models learn general purpose representation from naturally occurring unlabeled training data, and can capture interesting linguistic properties [47–49]. On the other hand, task-specific models are trained to solve a particular task, e.g., sentiment analysis [50], machine translation [51], and parsing [52]. Our focus in this paper is on learning distributed representation of sentences from unlabeled data.

The Word2vec model [53] to learn distributed representation of words is very popular for text processing tasks. The model also scales well in practice due to its

[1] `https://github.com/tksaha/con-s2v/tree/jointlearning`

simple architecture. Sen2Vec [44] extended Word2vec [53] to learn the representation for sentences and documents. The model maps each sentence to an unique id and learns the representation for the sentence using the contexts of words in the sentence – either by predicting the whole context independently (DBOW), or by predicting a word in the context (DM) given the rest. In our work, we extend the DBOW model to incorporate inter-sentence relations in the form of a discourse context or a similarity context. We do this using a graph-smoothing regularizer in the original objective function, or by retrofitting the initial vectors with different types of context.

In [5–7], retrofitting and regularization methods have been explored to incorporate lexical semantic knowledge into word representation models. Our overall idea of using external information is reminiscent of these models with two key differences: (*i*) the semantic network (WordNet, FrameNet) is given in their case, whereas we construct the network using similarities between sentences (nodes); (*ii*) we also explore discourse context that incorporate knowledge from adjacent sentences.

Adjacent sentences have been used previously for modeling task-agnostic representation of sentences. For example, Hill et al. [48] proposed FastSent, which learns word representation of a sentence by predicting words of its adjacent sentences. It derives a sentence vector by summing up the word vectors. The auto-encode version of FastSent also predicts the words of the current sentence. FastSent is fundamentally different from our models as we consider nearby sentences as atomic units, and we encode the sentence vector directly.

Hill et al. [48] also proposed two other models, *Sequential Denoising Autoencoder* (SDAE) and *Sequential Autoencoder* (SAE). SDAE employs an encoder-decoder framework, similar to neural machine translation (NMT) [51], to denoise an original sentence (target) from its corrupted version (source). SAE uses the same NMT framework to reconstruct (decode) the same source sentence. Both SAE and SDAE compose sentence vectors sequentially, but they disregard context of the sentence.

Another context-sensitive model is Skip-Thought [54], which uses the NMT framework to predict adjacent sentences (target) given a sentence (source). Since the en-

coder and the decoder use recurrent layers to compose vectors sequentially, SDAE and Skip-Thought are very slow to train. Furthermore, by learning representations to predict content of neighboring sentences, these methods (FastSent and Skip-Thought) may learn linguistic properties that are more specific to the neighbors rather than the sentence under consideration.

In contrast, we encode a sentence directly by treating it as an atomic unit, and we predict the words to model its content. Similarly, our model incorporates contextual information by treating neighboring sentences as atomic units. This makes our model quite efficient to train and effective for many tasks as we have shown.

5.3 Methodology

Let $\phi : V \rightarrow \mathbb{R}^d$ be the mapping function from sentences to their distributed representations, i.e., real-valued vectors of d dimensions. Equivalently, ϕ can be thought of as a look-up matrix of size $|V| \times d$, where $|V|$ is the total number of sentences. Our goal is to learn ϕ by exploiting information from two different sources: (i) the content of the sentence, $\mathbf{v} = (v_1, v_2 \cdots v_m)$; and (ii) the context of the sentence, $N(\mathbf{v})$. In the following subsections, we first describe an existing model that considers only the content of a sentence (Subsection 5.3.1). We then formalize types of extra-sentential context (Subsection 5.3.2). Finally, we present our models that extend the content-based model to incorporate contextual information (Subsections 5.3.3 – 5.3.4).

5.3.1 Content-based Model: Sen2Vec

Le and Mikolov [44] proposed two log-linear models for learning vector representation of sentences: (a) a distributed memory (DM) model, and (b) a distributed bag of words (DBOW) model. As shown in Figure 5.1, both models are trained solely based on the content of the sentences. In the DM model, every sentence in V is represented by a d dimensional vector in a shared lookup matrix $\phi \in \mathbb{R}^{|V| \times d}$. Similarly,

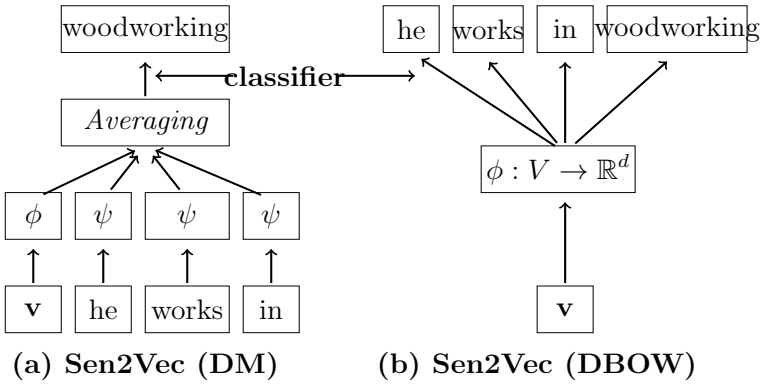


Fig. 5.1.: Distributed Memory (DM) and Distributed Bag of Words (DBOW) versions of Sen2Vec.

every word in the vocabulary $\mathcal{D}$ is represented by a d dimensional vector in another shared lookup matrix $\psi \in \mathbb{R}^{|\mathcal{D}|\times d}$. Given an input sentence $\mathbf{v} = (v_1, v_2 \cdots v_m)$, the corresponding sentence vector from ϕ and the corresponding word vectors from ψ are averaged to predict the next word in a context. More formally, the DM model minimizes the following loss (negative log likelihood):

$$\begin{aligned}\mathcal{L}_c(\mathbf{v}) &= \sum_{t=k}^{m-k} -\log P(v_t|\mathbf{v}; v_{t-k+1}, \cdots, v_{t-1}) \\ &= \sum_{t=k}^{m-k} -\log \frac{\exp(\omega(v_t)^T\mathbf{z})}{\sum_{v_i\in\mathcal{D}} \exp(\omega(v_i)^T\mathbf{z})} \end{aligned} \tag{5.1}$$

where $\mathbf{z}$ is the average of $\phi(\mathbf{v}), \psi(v_{t-k+1}), \cdots, \psi(v_{t-1})$ *input* vectors, and $\omega(v_t)$ is the *output* vector representation of word v_t. The sentence vector $\phi(\mathbf{v})$ is shared across all (sliding window) contexts extracted from the same sentence, thus acts as a distributed memory. Instead of predicting the next word in the context, the DBOW model predicts the words in the context independently given the sentence id as input. More formally, DBOW minimizes the following loss:

$$\begin{aligned}\mathcal{L}_c(\mathbf{v}) &= \sum_{t=k}^{m-k} \sum_{j=t-k+1}^{t} -\log P(v_j|\mathbf{v}) \\ &= \sum_{t=k}^{m-k} \sum_{j=t-k+1}^{t} -\log \frac{\exp(\omega(v_j)^T\phi(\mathbf{v}))}{\sum_{v_i\in\mathcal{D}} \exp(\omega(v_i)^T\phi(\mathbf{v}))} \end{aligned} \tag{5.2}$$

Training of the models is typically performed using gradient-based online methods, such as stochastic gradient descend (SGD). Unfortunately, this could be impractically slow on large corpora due to summation over all vocabulary items $\mathcal{D}$ in the denominator (Equations 5.1 and 5.2), which needs to be performed for every training instance $(\mathbf{v}, v_j)$. To address this, Mikolov et. al [47] use *negative sampling*, which samples negative examples to approximate the summation term. For instance, for each training

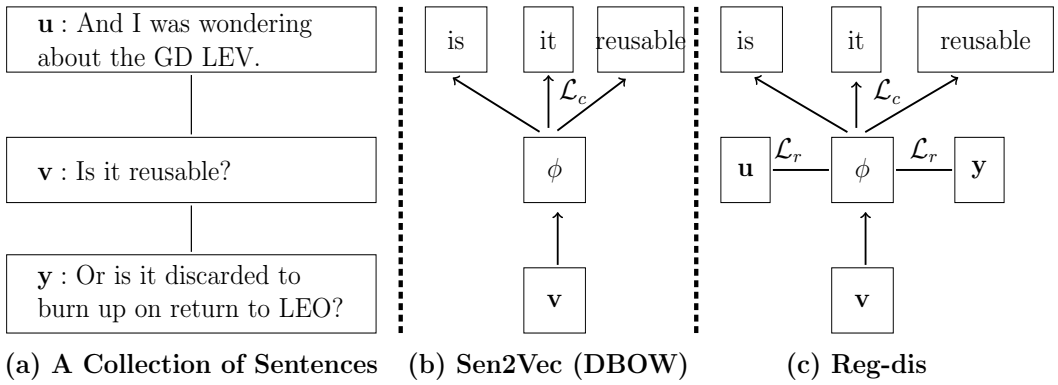


Fig. 5.2.: (c) presents an instance of our regularized model for learning representation of sentence **v** in comparison to (b) Sen2Vec (DBOW) model within a context of two other sentences: **u** and **y** in (a). Directed and undirected edges indicate prediction loss and regularization loss, respectively. (Collected from:newsgroup/20news-bydate-train/sci.space/61019. The central topic is "science.space".).

instance $(\mathbf{v}, v_j)$ in Equation 5.2, we add S negative examples $\{(\mathbf{v}, v_j^s)\}_{s=1}^{S}$ by sampling v_j^s from a known noise distribution μ (e.g., *unigram*, *uniform*). The log probability is then formulated as such to discriminate a *positive* instance v_j from a *negative* one v_j^s.

$$\log P(v_j|\mathbf{v}) = \log \sigma\left(\omega(v_j)^T \phi(\mathbf{v})\right) + \log \sum_{s=1}^{S} \mathbb{E}_{v_j^s \sim \mu}\, \sigma\left(-\omega(v_j^s)^T \phi(\mathbf{v})\right) \tag{5.3}$$

where σ is the sigmoid function defined as $\sigma(x) = 1/(1 + e^{-x})$. The loss in Equation 5.3 can be optimized efficiently as S is a small number (5 – 10) compared to the vocabulary size $|\mathcal{D}|$ (26K – 139K).

Both DM and DBOW models attempt to capture the overall semantics of a sentence by looking at its content words. However, sentences in a well-formed text are rarely independent, rather the meaning of one sentence depends on the meaning of other sentences in its context. For instance, consider the sentences in Figure 5.2(a), which are taken from the *science.space* category of the *Newsgroups* dataset. Here, the paragraph is talking about *shuttle's reusability* for the missions in space. If we con-

sider the sentences independently, it is very hard to understand the topic. Sentence, **u** is talking about shuttle, **v** is raising concern about its reusability and sentence **y** is elaborating on **v** to convey the concern more straightforwardly. When the sentences are considered together, it becomes easier to interpret. This suggests that representation learning models should also consider extra-sentential context to learn better representations for sentences.

5.3.2 Context Types

We distinguish between two types of context: discourse context and similarity context, as we elaborate on them below.

Discourse Context

Sentences in a text segment (e.g., paragraph) are semantically related by certain coherence relations (e.g., *elaboration*, *contrast*), and they address a common topic [11]. This indicates that adjacent sentences of a particular sentence is essential to better understand the meaning of the sentence. The discourse context of a sentence is comprised by its previous and the following sentences in a text.

Similarity Context

While the sequential order of the sentences carries important information, sentences that are far apart in the temporal order can also be related. In an empirical evaluation of data structures for representing discourse coherence, [146] advocates for a graph representation of discourse allowing non-adjacent connections. Moreover, graph-based methods for topic segmentation [145] and summarization [142] rely on complete graphs of sentences, where edge weights represent cosine similarity between sentences. Therefore, we consider a context type that is based on a direct measure of

similarity, and considers relations between all possible sentences in a document and possibly across multiple documents.

Our similarity context allows any other sentence in the corpus to be in the context of a sentence depending on how similar they are. To measure the similarity, we first represent the sentences with vectors learned by Sen2Vec [44], then we measure the cosine distance between the vectors. We restrict the context size of a sentence for computational efficiency, while still ensuring that it is informative enough. We achieve this by imposing two kinds of constraints. First, we set thresholds for intra- and across-document connections: sentences in a document are connected only if their similarity value is above a pre-specified threshold δ, and sentences across documents are connected only if their similarity value is above another pre-specified threshold γ. Second, we allow up to 20 most similar neighbors. We call the resulting network *similarity network*. Equation 5.4 formalizes the similarity network construction strategy explained above.

$$(\mathbf{u}, \mathbf{v}) = \begin{cases} 1, \ if \ \sigma(\mathbf{u}, \mathbf{v}) \leq \delta \mid \mathbf{u} \in D_\ell, \mathbf{v} \in D_m, \ell = m, \mathbf{v} \in top_{20} \\ 1, \ if \ \sigma(\mathbf{u}, \mathbf{v}) \leq \gamma \mid \mathbf{u} \in D_\ell, \mathbf{v} \in D_m, \ell \neq m, \mathbf{v} \in top_{20} \\ 0, \ otherwise \end{cases} \tag{5.4}$$

where D_l and D_m refer to l-th and m-th documents in the corpus, respectively. In the following two subsections, we present two different methods to incorporate context (discourse or similarity) for learning vector representation of sentences.

5.3.3 Retrofitted Models: Ret-dis, Ret-sim

We explore the general idea of *retrofitting* [6] to incorporate information from both the content and context of a node (sentence) in a joint learning framework. Let $\phi'(\mathbf{v})$ denote the vector representation for sentence $\mathbf{v}$ that has already been learned by our content-based model (Sen2Vec) in Section 5.3.1. Our aim is to retrofit this vector

using either discourse context or similarity context such that the revised vector $\phi(\mathbf{v})$: (i) is similar to the prior vector $\phi'(\mathbf{v})$, and (ii) is also similar to the vectors of its adjoining sentences (discourse context) or its adjacent nodes (similarity context). To this end, we define the following objective function to minimize:

Algorithm 3: Jacobi method for retrofitting.

Input :
- Graph $G = (V, E)$
- Prior vectors ϕ'
- Probabilities $\alpha_{\mathbf{v}}$ and $\beta_{\mathbf{v},\mathbf{u}}$

Output: Retrofitted vectors ϕ

$\phi \leftarrow \phi'$ `// initialization`

repeat

  for *all* $\mathbf{v} \in V$ **do**

    $\phi(\mathbf{v}) \leftarrow \frac{\alpha_{\mathbf{v}}\phi'(\mathbf{v}) + \sum_{\mathbf{u}} \beta_{\mathbf{v},\mathbf{u}}\phi(\mathbf{u})}{\alpha_{\mathbf{v}} + \sum_{\mathbf{u}} \beta_{\mathbf{v},\mathbf{u}}}$

  end

until *convergence*;

$$J(\phi) = \sum_{\mathbf{v} \in V} \alpha_v ||\phi(\mathbf{v}) - \phi'(\mathbf{v})||^2 + \sum_{(\mathbf{v},\mathbf{u}) \in E} \beta_{u,v} ||\phi(\mathbf{u}) - \phi(\mathbf{v})||^2 \tag{5.5}$$

where α values control the strength to which the algorithm should match the prior vectors, and β values control the degree of smoothness based on the graph similarity. The quadratic cost in Equation 5.5 is convex in ϕ, and has a closed form solution [118]. The closed form expression requires an inversion operation, which could be expensive for big graphs. A more efficient way is to use the Jacobi method, an online algorithm to solve the Equation iteratively. The Jacobi method leads to following update rule:

$$\phi(\mathbf{v}) \leftarrow \frac{\alpha_{\mathbf{v}}\phi'(\mathbf{v}) + \sum_{\mathbf{u}} \beta_{\mathbf{v},\mathbf{u}}\phi(\mathbf{u})}{\alpha_{\mathbf{v}} + \sum_{\mathbf{u}} \beta_{\mathbf{v},\mathbf{u}}} \tag{5.6}$$

In our case, we set $\alpha_{\mathbf{v}} = 1$, and $\beta_{\mathbf{v},\mathbf{u}} = \frac{1}{degree(\mathbf{v})}$, i.e. we give higher weights to vectors learned from Sen2Vec than to its contextual counterpart. Similar settings have been used in [6]. In Algorithm 3, we formally describe the training procedure of our retrofitted model. We use the DBOW model to learn the prior vectors ϕ'. We

name the model that consider discourse context as RET-dis and the model considering similarity context as RET-sim.

5.3.4 Regularized Models: Reg-dis, Reg-sim

Rather than retrofitting the vectors learned from a content-based model using context as a post-processing step, we can incorporate neighborhood information directly into the objective function of the content-based model as a regularizer, and learn the sentence vectors in a single step. We define the following objective to minimize:

$$J(\phi) = \sum_{\mathbf{v} \in V} \Big[\mathcal{L}_c(\mathbf{v}) + \beta \mathcal{L}_r(\mathbf{v}, N(\mathbf{v})) \Big] \tag{5.7}$$

$$= \sum_{\mathbf{v} \in V} \Big[\mathcal{L}_c(\mathbf{v}) + \beta \sum_{(\mathbf{v},\mathbf{u}) \in E} ||\phi(\mathbf{u}) - \phi(\mathbf{v})||^2 \Big] \tag{5.8}$$

where the first component $\mathcal{L}_c(\mathbf{v})$ refers to the loss of the content-based model described in Section 5.3.1. The second component $\mathcal{L}_r(\mathbf{v}, N(\mathbf{v}))$ is a Laplacian regularizer with β being the regularization strength. The regularizer brings the vector representation of a sentence closer to its context. Depending on the context type, this leads to different objectives. The model that uses discourse context (call this REG-dis) trains the vectors to be closer to the adjacent sentences in a text. Similarly, the model with similarity context (call this REG-sim) trains the vectors to be closer to its neighbors in the similarity network. As in the retrofitted models, we use DBOW as our content-based model.

Since the regularized models learn the vectors from scratch in one shot by considering information from both sources, the two components can be better adjusted to produce better quality vectors. Figure 5.2 (c) shows one instance of our model with discourse context ($\mathbf{u}$ and $\mathbf{y}$ are the adjoining sentences of sentence $\mathbf{v}$). Algorithm 4 formally describes the training procedure for the regularized models. First, we initialize the model parameters: ϕ, ψ and ω, and compute the unigram distribution over

words as our noise distribution μ. In each epoch of SGD (Line 3), we iterate over the sentences, and take one gradient step to learn word embeddings (Step b) and take two gradient steps (Steps d and e) to learn sentence embeddings: one for the content prediction loss, and the other for the regularization loss.

Algorithm 4: Training REG with SGD.

Input : set of sentences V, graph $G = (V, E)$, window size k
Output: learned sentence vectors ϕ
1. Initialize model parameters: ϕ, ψ and ω's;
2. Compute noise distribution: μ
3. **repeat**
    for *each sentence* $\mathbf{v} \in V$ **do**
        for *each content word* $v \in \mathbf{v}$ **do**
            `// for word vectors`
            for *each word* v_i *around* v *for window* k **do**
                (a) Generate a positive pair (v_i, v) and S negative pairs $\{(v_i, v^s)\}_{s=1}^{S}$ using μ;
                (b) Take a gradient step for $\mathcal{L}_c(v_i, v)$
            end
            `// for sentence vectors`
            (c) Generate a positive pair $(\mathbf{v}, v)$ and S negative pairs $\{(\mathbf{v}, v^s)\}_{s=1}^{S}$ using μ;
            (d) Take a gradient step for $\mathcal{L}_c(\mathbf{v}, v)$;
            (e) Take a gradient step for $\mathcal{L}_r(\mathbf{v}, N(\mathbf{v}))$;
        end
    end
until *convergence*;

5.4 Evaluation Method: Tasks, Datasets and Metrics

We evaluate our representation learning models on three different tasks that involve *classification*, *clustering*, and *ranking* sentences. These are the three fundamental information system tasks, and good performance over these tasks will indicate the robustness of our models in a wide range of downstream applications.

For classification (or clustering), we measure how effective the learned vectors are when they are used for classifying (or clustering) sentences based on their *topics*. Text

Table 5.1.: Basic statistics of the DUC datasets.

Dataset	# Doc.	# Sen. (Avg)	# Sum. (Avg)
DUC 2001	486	40	2.17
DUC 2002	471	28	2.04

categorization is now a standard task for evaluating cross-lingual *word* embeddings [147]. For ranking, we evaluate how effective the vectors are when they are used to rank sentences for generating an *extractive summary* [148] of a document.

As our representation learning models exploit inter-sentence relations,[2] which can possibly be constrained by document boundaries (e.g., *similarity context*), therefore, for topic classification and clustering, we require datasets containing documents with sentence-level annotations. However, to the best of our knowledge, no dataset exists with topic annotations at the sentence level. We generate sentence-level topic annotations from the document-level ones by selecting subsets of sentences that can be considered as representatives of the document using an extractive summarization tool, and label the selected sentences with the same document-level topic. In both of our tasks, extractive summarization is a key component, therefore in the following, we first describe the summarization task.

5.4.1 Extractive Summarization (Ranking) Task

The Extractive Summarizer

Extractive summarization is often considered as a ranking problem with the goal to select the most important sentences to form a compressed version of the source document. Unsupervised methods are the predominant paradigm for determining sentence importance [148]. We use popular graph-based algorithm LexRank [142] for this purpose. To get the summary sentences of a document, we first build a weighted

[2]For this reason, we did not evaluate our models on tasks previously used to evaluate sentence representation models.

graph, where nodes represent the sentences of a document and edge weights represent cosine similarity between learned vector space representations (using any vector space representation models of our choice) of the two corresponding sentences. To make the graph sparse, we avoid edges with weight less than 0.10. We then run the PageRank algorithm [149] on the graph to determine the rank of each sentence in a document, and thereby *extract* the key sentences as summary of that document. The dumping factor in PageRank was set to 0.85.

Datasets

We use the benchmark datasets from DUC 2001 and 2002, where the task[3] is to generate a 100-words summary for each document in the datasets. Table 5.1 shows some basic statistics about the datasets. DUC-2001 and DUC-2002 has 486 and 471 documents respectively. The average number of sentences per document is 40 and 28, respectively. For each document, 2-3 short reference (human authored) summaries are available, which we use as gold summaries in our evaluation. The human authored summaries are of approximately 100 words. On average, the datasets have 2.17 and 2.04 human authored summaries per document, respectively. The sentence representations are learned independently a priori from the same source documents.

Metrics

We use the widely used automatic evaluation metric ROUGE [150] to evaluate the system-generated summaries. ROUGE is a recall oriented metric that computes n-gram recall between a candidate summary and a set of reference (human authored) summaries. Among the variants, ROUGE-1 (i.e., $n = 1$) has been shown to correlate well with human judgments for short summaries [150]. Therefore, we only report ROUGE-1 in this paper. The configuration for ROUGE in our case is: *-c* 99 *-2 -1 -r*

[3]http://www-nlpir.nist.gov/projects/duc/guidelines

Table 5.2.: Statistics about Reuters and Newsgroups dataset.

Dataset	#Doc.	Total #sen.	Annot. #sen	Train #sen.	Test #sen.	#Class
Reuters	9,001	42,192	13,305	7,738	3,618	8
Newsgroups	7,781	95,809	22,374	10,594	9,075	8

1000 *-w* 1.2 *-n* 4 *-m -s -a -l 100*. Depending on the task at hand, ROUGE collects the first 100 words from the summary after removing the stop words to compare with the corresponding reference summaries.

5.4.2 Topic Classification and Clustering Tasks

Classification and Clustering Tools

We train a maximum entropy (MaxEnt) classifier using the vectors learned from the models with no additional fine-tuning for evaluation. Following [54], we restrict ourselves to linear classifier. The two main reasons are: (*i*) it makes reproducing results of experiments straight-forward, and (*ii*) it allows us to better analyze the quality of the learned vector representation. For clustering, we use k-means++ [151] algorithm for producing the clusters given the vector representation from the models. One can use non-linear classifiers (e.g., neural networks) or spectral clustering algorithms [152, 153] to achieve additional performance gain, but it is not the goal of our paper.

Datasets

We use *20-Newsgroups* and *Reuters-21578* datasets for the classification and clustering tasks. These datasets are publicly available and widely used for text categorization tasks.

20 Newsgroups

This dataset is a collection of approximately $20,000$ news documents[4]. The documents are organized into 20 different topics. Some of these topics are closely related (e.g., *talk.politics.guns* and *talk.politics.mideast*), while others are diverse in nature (e.g., *misc.forsale* and *soc.religion.christian*). We selected 8 diverse topics in our experiments from the 20 topics. The selected topics are: *talk.politics.mideast*, *comp.graphics*, *soc.religion.christian*, *rec.autos*, *sci.space*, *talk. politics.guns*, *rec.sport. baseball*, and *sci.med*.

Reuters-21578

Reuters Newswire[5] has 21578 documents covering 672 topics. We use "ModApte" train-test split and selected documents only from the most 8 frequent topics. The selected topics are: *acq, crude, earn, grain, interest, money-fx, ship,* and *trade*.

Generating Sentence-level Topic Annotations

As discussed earlier, for our evaluation on topic classification and clustering tasks, we have to create topic annotations at the sentence-level from the document-level topic labels. One option is to assume that all the sentences from a document have the same topic label as the document. However, this naive assumption propagates a lot of noises. Although sentences in a document collectively address a common topic, not all sentences are directly linked to that topic, rather some of them play supporting roles. To minimize this noise, we use the extractive (unsupervised) summarizer described in Section 5.4.1 to select the top $P\%$ (in our case, $P = 20$) sentences as representatives of the document and label them with the same topic label as the document. We used Sen2Vec [44] representation to compute cosine similarity between two sentences in LexRank. Table 5.2 shows statistics of the resulting datasets. Note that the

[4]http://qwone.com/ jason/20Newsgroups/
[5]http://kdd.ics.uci.edu/databases/reuters21578/

sentence vectors are learned independently from an entire dataset (Total #sen.), and the annotated part (Annot. #sen.) is used for topic classification and clustering evaluation.

Metrics

We use accuracy (Acc), Macro-averaged F1 measure (F1), and Cohen's Kappa (κ) as evaluation metrics for comparing the performance of various vector representation methods on topic classification task. For measuring topic clustering performance [154], we use V-measure (V), and adjusted mutual information (AMI) score. V-measure is the harmonic mean of the *homogeneity* and *completeness* score. The idea of homogeneity is that the topic distribution within each cluster should be skewed to a single topic. Completeness score determines whether all members of a given topic are assigned to the same cluster. On the other hand, AMI measures the agreement of two assignments, in our case the clustering and the topic distribution. It is normalized against chance. All these measures are bounded by $[0, 1]$. Higher score means a better clustering.

5.5 Experimental Settings

In this section, we briefly discuss the models that we compare with and the settings (hyperparameters, training) for our models.

5.5.1 Models Compared

We compare our models against a non-distributed baseline and a number of existing distributed representation models.

Non-Distributed Baseline

We implement a **TF-IDF** model as our non-distributed baseline. The model encodes representation of a sentence as the count of a set of word-features weighted by tf-idf. We use all the words in the corpus as features.

Sen2Vec

We described Sen2Vec in details as a content-only model in Section 5.3.1. We use Mikolov's implementation[6] of Sen2Vec as it gave better results than gensim's [7] version when validated on the sentiment treebank [155]. Following the recommendation by [44], we concatenate the vectors learned by DM and DBOW models. The concatenated vectors gave improvements over individual ones on our tasks. The vector dimensions in DM and DBOW were fixed to 300, thus the concatenation yields vectors of 600 dimensions. For this model, we only tune the window size (k) hyper-parameter.

W2V-avg

Sen2Vec model learns word representation along with the sentence representation. To encode a sentence using W2V-avg, we perform an averaging operation on the vector representation (learned from Sen2Vec) of all the words in a particular sentence. For this model, we consider window size (k) as a tuning parameter.

C-PHRASE

C-PHRASE [156] learns vector representation of words. It extended the CBOW model [47] to consider the hierarchical nature of syntactic phrasing. As the implementation of this model is not publicly available, we use pretrained word vectors from

[6]https://code.google.com/archive/p/word2vec/
[7]https://radimrehurek.com/gensim/

Table 5.3.: Similarity network statistics.

Dataset	# Nodes	# Edges	Avg. # Edges
20 Newsgroups	95809	1370149	14.03
Reuters-21578	42192	471163	11.17
DUC 2001	19549	321423	20.15
DUC 2002	13129	216492	16.49

author's webpage.[8] We first perform simple addition of word sequences of a sentence for obtaining vector representation of a sentence, and then normalize the vector. Normalized vectors performed better on our tasks than the ones obtained through simple addition. The latent dimension of the pretrained word vectors is 300.

FastSent

FastSent [48] is an additive model that learns representation of words in a sentence by predicting words of adjacent sentences. We use the autoencode version of the model, which also predicts the words of the current sentence. In FastSent, a sentence vector is obtained by adding the word vectors. We run the model on our corpus to learn sentence representations of 600 dimensions, and tune the window size (k) hyperparameter on the dev. set.

Skip-Thought

Skip-Thought [54] uses an encoder-decoder approach to reconstruct adjacent sentences of an input sentence. Training Skip-Thought is computationally expensive [48], and it requires a lot of data to learn an effective model. We use the pre-trained combine-skip model[9], which was trained on the book corpus [157] along with vocabulary expansion. Skip-thought vectors are of 4800 dimensions.

[8]http://clic.cimec.unitn.it/composes/cphrase-vectors.html
[9]https://github.com/ryankiros/skip-thoughts

Table 5.4.: Optimal values of the hyper-parameters for different models on different tasks.

Dataset	Task	Sen2Vec	FastSent	W2V-avg	REG-sim	REG-dis
		(win. size)			*(win. size, reg. str.)*	
Reuters	clas.	8	10	10	(8, 1.0)	(8, 1.0)
	clus.	12	8	12	(12, 0.3)	(12, 1.0)
Newsgroups	clas.	10	8	10	(10, 1.0)	(10, 1.0)
	clus.	12	12	12	(12, 1.0)	(12, 1.0)
DUC 2001	rank.	10	12	12	(10, 0.8)	(10, 0.5)
DUC 2002	rank.	8	8	10	(8, 0.8)	(8, 0.3)

5.5.2 Hyper-Parameter Tuning and Training Details

All of our models except the retrofitted ones (i.e., RET-sim, RET-dis) are trained with stochastic gradient descent (SGD), where the gradient is obtained via back-propagation. We used subsampling of frequent words in the classification layer as described in [47], which together with negative sampling give significant speed-ups in training. The number of noise samples (S) in negative sampling was 5. In all our models, the embeddings vectors (ϕ, ψ) were of 600 dimensions, which were initialized with random numbers sampled from a small uniform distribution, $\mathcal{U}(-0.5/d, 0.5/d)$. The weight vectors ω's were initialized with zero. Increasing the dimension may increase performance, however, it also increases the complexity of the model. So, we keep it 600, which is a reasonable size [48]. For RET-sim, and RET-dis, the number of iteration was set to 20 following [6]. For the similarity context, the intra- and across-document thresholds δ and γ were set to 0.5 and 0.8, respectively. Table 5.3 shows the basic statistics of the resultant similarity network for all of our datasets.

For each dataset, we randomly selected 20% documents from the whole set to form a held-out validation set on which we tune the hyperparameters of the models. To find the best parameter values, we optimize F1 for classification, AMI for clustering and ROUGE-1 for summarization on the validation set. Window size (k) parameter for our model and the baselines were tuned over {8, 10, 12 } and the regularization strength parameter was tuned over {0.3, 0.6, 0.8, 1.0}. Table 5.4 shows the optimal values of each hyper-parameter for the four datasets. We evaluate our models on the test set with these optimal values, run each test experiment five times and take the average to avoid any random behavior appearing in the results. We observed the standard deviation to be quite low.

5.6 Results and Discussion

We present our results on topic classification and clustering in Table 5.5 and Table 5.6, and results on ranking (summarization) in Table 5.7. The results in each

Table 5.5.: Performance of our models on topic classification task in comparison to Sen2Vec.

	Topic Classification Results					
	Reuters			*Newsgroups*		
	F_1	Acc	κ	F_1	Acc	κ
Sen2Vec	83.25	83.91	79.37	79.38	79.47	76.16
TF-IDF	(−) 3.51	(−) 2.68	(−) 3.85	(−) 9.95	(−) 9.72	(−) 11.55
W2V-avg	(+) 2.06	(+) 1.91	(+) 2.51	(−) 0.42	(−) 0.44	(−) 0.50
C-PHRASE	(−) 2.33	(−) 2.01	(−) 2.78	(−) 2.49	(−) 2.38	(−) 2.86
FastSent	(−) 0.37	(−) 0.29	(−) 0.41	(−) 12.23	(−) 12.17	(−) 14.21
Skip-Thought	(−) 19.13	(−) 15.61	(−) 21.8	(−) 13.79	(−) 13.47	(−) 15.76
Ret-sim	(+) 0.92	(+) 1.28	(+) 1.65	(+) 2.00	(+) 1.97	(+) 2.27
Ret-dis	(+) 1.66	(+) 1.79	(+) 2.30	(+) 5.00	(+) 4.91	(+) 5.71
Reg-sim	(+) **2.53**	(+) **2.53**	(+) **3.28**	(+) 3.31	(+) 3.29	(+) 3.81
Reg-dis	(+) 2.52	(+) 2.43	(+) 3.17	(+) **5.41**	(+) **5.34**	(+) **6.20**

Table 5.6.: Performance of our models on topic clustering tasks in comparison to Sen2Vec.

	Topic Clustering Results			
	Reuters		*Newsgroups*	
	V	AMI	V	AMI
Sen2Vec	42.74	40.00	35.30	34.74
TF-IDF	(−) 21.34	(−) 20.14	(−) 29.20	(−) 30.60
W2V-avg	(−) 11.96	(−) 10.18	(−) 17.90	(−) 18.50
C-PHRASE	(−) 11.94	(−) 10.80	(−) 1.70	(−) 1.44
FastSent	(−) 15.54	(−) 13.06	(−) 34.40	(−) 34.16
Skip-Thought	(−) 29.94	(−) 28.00	(−) 27.50	(−) 27.04
Ret-sim	(+) 3.72	(+) 3.34	(+) 5.22	(+) 5.70
Ret-dis	(+) 4.56	(+) 4.12	(+) 6.28	(+) 6.76
Reg-sim	(+) 4.76	(+) 4.40	(+) **12.78**	(+) 12.18
Reg-dis	(+) **7.40**	(+) **6.82**	(+) 12.54	(+) **12.44**

Table 5.7.: ROUGE-1 scores of the models on DUC datasets in comparison with Sen2Vec.

	DUC'01	DUC'02
Sen2Vec	43.88	54.01
TF-IDF	(+) **4.83**	(+) 1.51
W2V-avg	(−) 0.62	(+) 1.44
C-PHRASE	(+) 2.52	(+) 1.68
FastSent	(−) 4.15	(−) 7.53
Skip-Thought	(+) 0.88	(−) 2.65
Ret-sim	(−) 0.62	(+) 0.42
Ret-dis	(+) 0.45	(−) 0.37
Reg-sim	(+) 2.90	(+) **2.02**
Reg-dis	(−) 1.92	(−) 8.77

table are shown in four groups. Sen2Vec belongs to the first group. The second group contains other existing models described in Section 5.5.1. The third group contains our *retrofitted* models with discourse context (RET-dis) and similarity context (RET-sim). Finally, the fourth group contains our *regularized* models, again considering discourse (REG-dis) and similarity (REG-sim) contexts. We report absolute value of the performance metrics for Sen2Vec, and for other models, we present their scores relative to Sen2Vec. In the following, we highlight the key points of our results.

Skip-Thought and FastSent perform poorly on our tasks

Unexpectedly, FastSent and Skip-Thought perform quite poorly on our tasks. Skip-Thought, in particular, has the worst performance on topic classification and clustering tasks. The model gives small improvement over Sen2Vec on ranking task in one of the datasets (DUC'01). These results contradict the claim made by [54] that skip-thought vectors are generic. To investigate if the poor results are due to shift of domains (*book* vs. *news*), we also trained Skip-Thought on our training corpora with vector size 600 and vocabulary size $30K$. The performance was even worse. We hypothesize, this is due to our training set size, which may not be enough for the heavy model. Another reason could be that Skip-Thought does not perform any inference to extract the vector using a context – although the model was trained to generate neighboring sentences, context was ignored when the encoder was used to extract the sentence vector. Also, by learning representations to predict contents of adjacent sentences, the learned vectors might capture linguistic properties that are more specific to the neighbors than the current sentence. Similar justification holds for FastSent, which performed quite poorly in five out of six settings (Tasks + Datasets combinations). Furthermore, FastSent does not learn sentence representation directly, rather it adds word vectors to get sentence representations.

Existing distributed methods show promising results

Apart from Skip-Thought and FastSent, other existing distributed models show promising results. As Table 5.5 shows, Sen2Vec outperforms TF-IDF representation by a good margin on both classification and clustering tasks – up to 11.6 points on classification, and up to 30.6 points on clustering. W2V-avg shows 2 points improvement over Sen2Vec in topic classification on Reuters. The performance of C-PHRASE and W2V-avg is close to Sen2Vec for classification, however, the models lag substantially behind on clustering. Overall, Sen2Vec appears to be the strongest baseline for these two tasks.

In the ranking task (Table 5.7), Sen2Vec gets ROUGE-1 scores of 43.88 and 54.01 on DUC'01 and DUC'02 datasets, respectively. C-PHRASE outshines other distributed models on this task, and provides 2.52 and 1.68 points improvements over Sen2Vec. W2V-avg shows mixed results in summarization; it performs better than Sen2Vec on one dataset and worse on the other. Surprisingly, TF-IDF becomes the best performer on DUC'01, and gives improvements of 4.15 points over Sen2Vec. Overall, the results indicate that TF-IDF is a strong baseline for the summarization task.

Regularized and Retrofitted models outperform Sen2Vec

The retrofitting and regularized models improve over Sen2Vec on both classification and clustering tasks, showing gains of up to 6.2 points on classification and up to 12.8 points on clustering. We observe similar patterns in ranking given that the model considers the right context (ignoring the mixed results for retrofitted models). The improvements in most cases are significant. This demonstrates that contextual information is beneficial for these tasks.

Regularized models are the best performer

Our regularized models (Reg-sim, Reg-dis) performs best in five out of six settings (Dataset + Task combination). From the results presented in Table 5.5, we observe that regularized models are the top-performer in topic classification and clustering tasks. For topic classification on Newsgroups, our model gives around 6 points improvement over Sen2Vec and 8 points over C-PHRASE in all the metrics (F_1, *Acc* and κ). The improvements are even larger for clustering – about 13 points over Sen2Vec and 15 points over C-PHRASE. Similarly, on Reuters dataset, Reg-dis gives around 3 and 7 points improvements over Sen2Vec in topic classification and clustering tasks, respectively.

Regularized models also perform well on summarization task in Table 5.7 – best in DUC'02 and second best in DUC'01. Given that the existing models fail to beat the TF-IDF baseline on this task, our results are rather encouraging.

Regularization is better than retrofitting given the right context

From the third and fourth groups of results in Table 5.5, it is clear that Reg-dis and Reg-sim are better models than their retrofitted counterparts. Reg-sim also outperforms Ret-sim in ranking (Table 5.7) by 2 to 3 points. The good performance comes from the fact that regularized models consider contextual information during training rather than in the post-processing step. Thus, the model can better adjust contributions from different components (prediction vs. regularization) accordingly.

Discourse context is good for topic classification and clustering

Discourse context perform better than similarity context in most cases on classification and clustering tasks. From Table 5.5, we notice that, Ret-dis outperform Ret-sim by up to 3 points in classification and by about 1 point in clustering. Reg-dis and Reg-sim perform similarly on Reuters dataset for classification and on

Newsgroup dataset for clustering. However, REG-dis outperform REG-sim by a wide margin on Newsgroup dataset for classification and on Reuters dataset for clustering. The primary reason is that sentences appearing together in a discourse tend to address the same (sub)topic [11]. Discourse context is cheaper to obtain as it is readily available (consider only adjoining sentences). For obtaining similarity context, we need to obtain the similarity network as described in Section 5.3.2.

Similarity context is good for summarization

Similarity context is more suitable than discourse context for summarization – REG-sim is the best performer in DUC'02 dataset and the second best in DUC'01 dataset. Similarity context is based on a direct measure of similarity, and consider relations beyond adjacency. From a context of topically similar sentences, our model learns representations that capture linguistic aspects related to information centrality.

Other comments

We also experimented with *Sequential Denoising Autoencoder* (SDAE) and *Sequential Autoencoder* (SAE) models proposed in [48]. However, they performed poorly on our tasks (thus not shown in the table). For example, SAE gave accuracies of around 40% on reuters and 18% on newsgroups. This is similar to what [48] observed. They propose to use pretrained word embeddings to improve the results. We did not achieve significant gains by using pretrained embeddings on our tasks.

5.7 Chapter Summary

In this paper, we have proposed a set of novel models for learning vector representation of sentences that consider not only content of a sentence but also context of a sentence in the text. We have explored two different ways to incorporate contextual information: (i) by retrofitting the initial vectors learned from a content-based model

using context, and (*ii*) by regularizing the content-based model with a graph smoothing factor. We have also introduced two types of context: (*i*) discourse context, and (*ii*) similarity context.

While existing evaluation methods ignore contexts, we created an evaluation setup that allows one to infer sentence vectors using contextual information. We evaluated our models on tasks involving classifying and clustering sentences into topics, and ranking sentences for extractive single-document summarization. Our results across multiple datasets show impressive gains over existing distributed models in all evaluation tasks. The discourse context was found to be beneficial for topic classification and clustering, whereas the similarity context was beneficial for summarization.

In this study, we restrict the evaluation of our models on *topic classification and clustering* using automatically annotated dataset. We would like to explore further how our models perform compared to the existing compositional models [50, 54], where documents with sentence-level sentiment annotation exists. Existing datasets – IMDB [158] or Sentiment Treebank [50], are not suitable for our purpose because in IMDB, there is no sentence-label annotation, and in sentiment treebank, there is no contextual information. We plan to create a dataset through manual annotation in the future.

6. Con-S2V: A GENERIC FRAMEWORK FOR INCORPORATING EXTRA-SENTENTIAL CONTEXT INTO Sen2Vec

6.1 Introduction

For many text processing tasks that involve classification, clustering, or ranking of sentences, vector representation of sentences is a prerequisite. Bag-of-words (BOW) based vector representation has been used traditionally in these tasks, but in recent years, it has been shown that *distributed representation*, in the form of condensed real-valued vectors, learned from *unlabeled* data outperforms BOW based representations [44]. It is now well established that distributed representation captures semantic properties of linguistic units and yields better generalization [144, 159].

However, most of the existing methods to devise distributed representation for sentences consider only the content of a sentence or its grammatical structure [44,155] disregarding its context. But, sentences rarely stand on their own in a text, rather the meaning of one sentence depends on the meaning of others within its context. For example, sentences in a text segment address a common topic [11]. At a finer level, sentences are connected by certain coherence relations (e.g., *elaboration*, *contrast*) and acts together to express a coherent message holistically [10].

Our work is built on the following hypothesis: *since the meaning of a sentence can be best interpreted within its context, its representation should also be inferred from its context.* Several recent works attempt to learn sentence representations which support the above hypothesis by utilizing words or word sequences of neighboring sentences [48, 54]. However, by learning representations to predict content of neighboring sentences, existing methods may learn semantic and syntactic properties that are more specific to the neighbors rather than the sentence under consideration. Fur-

thermore, these methods either make a simple BOW assumption or disregard context when extracting a sentence vector.

In contrast to the existing works, we consider neighboring sentences as *atomic* linguistic units, and propose novel methods to learn the representations of a given sentence by jointly modeling content and context of a sentence. Our work considers two types of context: *discourse* and *similarity*. The discourse context of a given sentence $\mathbf{v}$ comprises with its previous and the following sentence in the text. On the other hand, the similarity context is based on a user defined similarity function; thus it allows any sentences in the text to be in the context of $\mathbf{v}$ depending on how similar that sentence is with $\mathbf{v}$ based on the chosen function.

Our proposed computational model for learning the vector representation of a sentence comprises three components. The first component models the content by asking the sentence vector to predict its constituent words. The second component models the *distributional* hypotheses [160] of a context. The distributional hypothesis conveys that the sentences occurring in similar contexts should have similar representations. Our computation model captures this preference by using a context prediction component. Finally, the third component models the *proximity* hypotheses of a context, which also suggests that sentences that are proximal should have similar representations. Our method achieves this preference by using a Laplacian regularizer. To this end, we consider the sentence representation learning problem as an optimization problem whose objective function is built with expressions from the above three components and we solve this optimization problem by using an efficient online algorithm.

We evaluate our sentence representation for learning models on multiple information retrieval tasks: topic classification and clustering, and single-document summarization. Our evaluation on these tasks across multiple datasets shows impressive results for our model, which outperforms the best existing models by up to 7.7 F_1-score in classification, 15.1 V-score in clustering, 3.2 ROUGE-1 score in summarization. We found that the discourse context performs better on topic classification and clustering

tasks, while similarity context performs better on summarization. We make our code[1] and pre-processed dataset[2] publicly available.

6.2 Related Work

Extensive research has been conducted on learning distributed representation of linguistic units both in supervised (task-specific) and in unsupervised (task-agnostic) settings. In this paper, we focus on learning *sentence* representations from *unlabeled* data.

Two log-linear models are proposed in [53] for learning representations of words: continuous bag-of-words (CBOW) and continuous skip-gram. CBOW learns word representations by predicting a word given its (intra-sentential) context. The skip-gram model on the other hand learns representation of a word by predicting the words in a context. [156] proposed C-PHRASE, an extension of CBOW, where the context is extracted from a syntactic parse of the sentence. Simple averaging or addition of word vectors to construct sentence vectors often works well [161], and serves as baselines in our experiments.

CBOW and skip-gram models are extended in [44] to sentences and documents by proposing distributed memory (DM) and distributed bag-of-words (DBOW) models. In these models, similar to words, a sentence is mapped to an unique id and its representation is learned using contexts of words in the sentence. DM predicts a word given a context and the sentence id, where DBOW predicts all words in a context independently given the sentence id. Since these models are agnostic to sentence structure, they are quite fast to train. However, they disregard extra-sentential context of a sentence.

Sequential denoising autoencoder (SDAE) and FastSent are proposed in [48] for modeling sentences. SDAE employs an encoder-decoder framework, similar to neural machine translation (NMT) [51], to denoise an original sentence (target) from its

[1] `https://github.com/tksaha/con-s2v/tree/jointlearning`
[2] `https://www.dropbox.com/sh/ruhsi3c0unn0nko/AAAgVnZpojvXx9loQ21WP_MYa?dl=0`

corrupted version (source). FastSent is an additive model to learn sentence representation from word vectors. Given a sentence as BOW, it predicts the words of its adjacent sentences. The auto-encode version of FastSent also predicts the words of the current sentence. SDAE composes sentence vectors sequentially, but it disregards context of the sentence. FastSent, on the other hand, is a BOW model that considers neighboring sentences.

Another context-sensitive model is Skip-Thought [54], which uses the NMT framework to predict adjacent sentences (target) given a sentence (source). Since the encoder and the decoder use recurrent layers to compose vectors sequentially, SDAE and Skip-Thought are very slow to train. Furthermore, by learning representations to predict content of neighboring sentences, these methods (FastSent and Skip-Thought) may learn linguistic properties that are more specific to the neighbors rather than the sentence under consideration.

By contrast, we encode a sentence by treating it as an atomic unit like word, and similar to DBOW, we predict the words to model its content. Similarly, context is considered in our model by treating neighboring sentences as atomic units. This abstraction makes our model quite fast to train.

6.3 The Model

We hypothesize that the representation of a sentence depends not only on its content words, but also on other sentences in its context. It will be convenient to present our learning model using graph.

Let $G = (V, E)$ be a graph, where $V = \{\mathbf{v}_1, \mathbf{v}_2, \cdots, \mathbf{v}_{|V|}\}$ represents the set of sentences in our corpus, and edge $(\mathbf{v}_i, \mathbf{v}_j) \in E$ reflects some relation between sentences $\mathbf{v}_i$ and $\mathbf{v}_j$. A sentence $\mathbf{v}_i \in V$ is a sequence of words $(v_i^1, v_i^2, \cdots, v_i^M)$, each coming from a dictionary $\mathcal{D}$. We define $\mathcal{N}(\mathbf{v}_i)$ as the set of neighboring sentences of $\mathbf{v}_i$, which constitutes extra-sentential context for sentence $\mathbf{v}_i$. We formalize relation between sentences and context later in Section 6.3.3.

Let $\phi : V \rightarrow \mathbb{R}^d$ be the mapping function from sentences to their distributed representations, i.e., real-valued vectors of d dimensions. Equivalently, ϕ can be thought of as a look-up matrix of size $|V| \times d$, where $|V|$ is the total number of sentences. Our aim is to learn $\phi(\mathbf{v}_i)$ by incorporating information from two different sources: (i) the content of the sentence, $\mathbf{v}_i = (v_i^1, v_i^2, \cdots, v_i^M)$; and (ii) the context of the sentence in the graph, i.e., $\mathcal{N}(\mathbf{v}_i)$. Let $\langle v_i \rangle_t^l = (v_i^{t-l}, \ldots, v_i^t, \ldots, v_i^{t+l})$ denote a window of $2l+1$ words around the word v_i^t in sentence $\mathbf{v}_i$, and $C_i = |\mathcal{N}(\mathbf{v}_i)|$ denote the context size for sentence $\mathbf{v}_i$. We define our model as a combination of three different loss functions:

$$J(\phi) = \sum_{\mathbf{v}_i \in V} \sum_{\substack{v \in \langle v_i \rangle_t^l \\ j \sim \mathcal{U}(1, C_i)}} \big[\mathcal{L}_c(\mathbf{v}_i, v) + \mathcal{L}_g(\mathbf{v}_i, \mathbf{v}_j) + \mathcal{L}_r(\mathbf{v}_i, \mathcal{N}(\mathbf{v}_i))\big] \tag{6.1}$$

where loss $\mathcal{L}_c(\mathbf{v}_i, v)$ is used to model the content of a sentence $\mathbf{v}_i$, and other two loss functions are for modeling the context of the sentence. We define $\mathcal{L}_c(\mathbf{v}_i, v)$ as the cost for predicting the content word v using the sentence vector $\phi(\mathbf{v}_i)$ as input features. Similarly, $\mathcal{L}_g(\mathbf{v}_i, \mathbf{v}_j)$ is defined as the cost for predicting a neighboring node $\mathbf{v}_j \in \mathcal{N}(\mathbf{v}_i)$, again using the sentence vector $\phi(\mathbf{v}_i)$ as input. The third loss $\mathcal{L}_r(\mathbf{v}_i, \mathcal{N}(\mathbf{v}_i))$ is a graph smoothing regularizer defined over the context of $\mathbf{v}_i$, which encourages two proximal sentences to have similar representations.

To learn the representation of a sentence $\mathbf{v}_i$ using Eq. 6.1, for each content word v in a window $\langle v_i \rangle_t^l$, we sample a neighboring node $\mathbf{v}_j$ from $\mathcal{N}(\mathbf{v}_i)$, uniformly at random, with replacement. We use the sentence vector $\phi(\mathbf{v}_i)$ (under estimation) to predict v and $\mathbf{v}_j$, respectively. A regularization is performed to smooth the estimated vector with respect to the neighboring vectors. Fig. 6.1 shows instances of our model for learning the representation of sentence $\mathbf{v}_2$ within a context of two other sentences: $\mathbf{v}_1$ and $\mathbf{v}_3$.

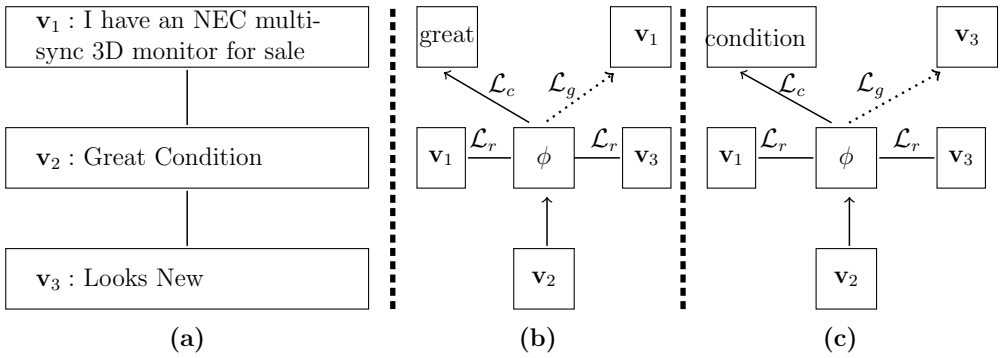


Fig. 6.1.: Two instances (see (b) and (c)) of our model for learning representation of sentence $\mathbf{v}_2$ within a context of two other sentences: $\mathbf{v}_1$ and $\mathbf{v}_3$ (see (a)). Directed and undirected edges indicate prediction loss and regularization loss, respectively, and dashed edges indicate that the node being predicted is randomly sampled. (Collected from: 20news-bydate-train/misc.forsale/74732. The central topic is "forsale".).

We can use the standard *softmax* function for the prediction tasks. Formally, the negative log probability of an item o (can be a content word or a neighboring node) given the sentence vector $\phi(\mathbf{v}_i)$ is

$$-\log p(o|\mathbf{v}_i) = -\mathbf{w}_o^T \phi(\mathbf{v}_i) + \log \sum_{o' \in \mathcal{O}} \exp\left(\mathbf{w}_{o'}^T \phi(\mathbf{v}_i)\right) \tag{6.2}$$

where $\mathcal{O}$ is the set of all possible items (i.e., vocabulary of words or set of all nodes), and $\mathbf{w}$'s are the weight parameters. Optimization is typically performed using gradient-based online methods, such as stochastic gradient descend (SGD), where gradients are obtained via backpropagation.

Unfortunately, training could be impractically slow on large corpora due to summation over all items in $\mathcal{O}$ (Eq. 6.2), which needs to be performed for every training instance $(\mathbf{v}_i, o)$. Several methods have been proposed to address this issue including *hierarchical softmax* [162], *noise contrastive estimation* [163], and *negative sampling* [47]. We use negative sampling, which samples negative examples to approximate the summation term. Specifically, for each training instance $(\mathbf{v}_i, o)$, we add S negative examples $\{(\mathbf{v}_i, o^s)\}_{s=1}^S$ by sampling o^s from a known noise distribution ψ (e.g., *unigram*, *uniform*). The negative log probability in Eq. 6.2 is then formulated as such to discriminate a *positive* instance o from a *negative* one o^s:

$$-\log \sigma\left(\mathbf{w}_o^T \phi(\mathbf{v}_i)\right) - \log \sum_{s=1}^{S} \mathbb{E}_{o^s \sim \psi}\, \sigma\left(-\mathbf{w}_{o^s}^T \phi(\mathbf{v}_i)\right) \tag{6.3}$$

where σ is the sigmoid function defined as $\sigma(x) = 1/(1+e^{-x})$, and $\mathbf{w}$'s and $\phi(\mathbf{v}_i)$ are similarly defined as before. Negative sampling thus reduces the number of computations needed from $|\mathcal{O}|$ to $S+1$, where S is a small number (5 – 10) compared to the vocabulary size $|\mathcal{O}|$ (26K – 139K).

In the following, we elaborate on our methods for modeling content and context of a sentence.

6.3.1 Modeling Content

Our approach for modeling content of a sentence is similar to the distributed bag-of-words (DBOW) model of [44]. Given an input sentence $\mathbf{v}_i$, we first map it to a unique vector $\phi(\mathbf{v}_i)$ by looking up the corresponding vector in the sentence embedding matrix ϕ. We then use $\phi(\mathbf{v}_i)$ to predict each word v sampled from a window of words in $\mathbf{v}_i$. Formally, the loss for modeling content using negative sampling is

$$\begin{aligned}\mathcal{L}_c(\mathbf{v}_i, v) = &- \log\sigma\left(\mathbf{w}_v^T \phi(\mathbf{v}_i)\right) \\ &- \log \sum_{s=1}^{S} \mathbb{E}_{v^s \sim \psi_c}\, \sigma\left(-\mathbf{w}_{v^s}^T \phi(\mathbf{v}_i)\right)\end{aligned} \tag{6.4}$$

where σ is the sigmoid function as defined before, $\mathbf{w}_v$ and $\mathbf{w}_{v^s}$ are the weight vectors associated with words v and v^s, respectively, and ψ_c is the noise distribution from which v^s is sampled. In our experiments, we use unigram distribution of words raised to the 3/4 power as our noise distribution, in accordance to [47].

By asking the same sentence vector (under estimation) to predict its words, the content model captures the overall semantics of the sentence. The model has $O(d \times (|V| + |\mathcal{D}|))$ parameters.

6.3.2 Modeling Context

Our content model above attempts to capture the overall meaning of a sentence by looking at its words. However, sentences in a text are not independent, rather the meaning of a sentence depends on its neighboring sentences. For instance, consider the second and the third sentences in Fig. 6.1(a). When the sentences are considered in isolation, one cannot understand what they are talking about (i.e., *monitor for sale*). This suggests, since meaning of a sentence can be best interpreted within its context, the representation of the sentence should also be inferred from its context. We distinguish between two types of contextual relations between sentences: (*i*) dis-

tributional similarity, and (*ii*) proximity. Each of these corresponds to a loss in our model (Eq. 6.1), as we describe them below.

Modeling Distributional Similarity: Our sentence-level distributional hypothesis [160] is that if two sentences share many neighbors in the graph, their representations should be similar. We formulate this in our model by asking the sentence vector to predict its neighboring nodes. More formally, the loss for predicting a neighboring node $\mathbf{v}_j \in \mathcal{N}(\mathbf{v}_i)$ using the sentence vector $\phi(\mathbf{v}_i)$ is

$$\begin{aligned}\mathcal{L}_g(\mathbf{v}_i, \mathbf{v}_j) = &- \log \sigma\left(\mathbf{w}_j^T \phi(\mathbf{v}_i)\right) \\ &- \log \sum_{s=1}^{S} \mathbb{E}_{j^s \sim \psi_g}\, \sigma\left(-\mathbf{w}_{j^s}^T \phi(\mathbf{v}_i)\right)\end{aligned} \tag{6.5}$$

where $\mathbf{w}_j$ and $\mathbf{w}_j^s$ are the weight vectors associated with nodes $\mathbf{v}_j$ and $\mathbf{v}_j^s$, respectively, and ψ_g is the noise distribution over nodes from which $\mathbf{v}_j^s$ is sampled. Similar to our content model, ψ_g is defined as unigram distribution of nodes raised to the 3/4 power. The unigram distribution is computed based on the occurrences of the nodes in the neighborhood sets, $\{\mathcal{N}(\mathbf{v}_i)\}_{i=1}^{|V|}$. This model has $O(d \times (|V| + |V|))$ parameters.

Modeling Proximity: According to our proximity hypothesis, sentences that are proximal in their contexts, should have similar representations. We use a Laplacian regularizer to model this. Formally, the regularization loss for modeling proximity for a sentence $\mathbf{v}_i$ in its context $\mathcal{N}(\mathbf{v}_i)$ is

$$\mathcal{L}_r(\mathbf{v}_i, \mathcal{N}(\mathbf{v}_i)) = \frac{\lambda}{C_i} \sum_{\mathbf{v}_k \in \mathcal{N}(\mathbf{v}_i)} ||\phi(\mathbf{v}_i) - \phi(\mathbf{v}_k)||^2 \tag{6.6}$$

where $C_i = |\mathcal{N}(\mathbf{v}_i)|$ as defined before, and λ is a hyper-parameter to control regularization strength.

Rather than including the Laplacian as a regularizer in the objective function, another option is to first learn sentence embeddings using other components of the model (e.g., first two loss functions in Equation 6.1), and then *retrofit* them using the Laplacian as a post-processing step. [6] adopted this approach to incorporate lexical semantics (e.g., synonymy, hypernymy) into word representations. We compare our approach with retrofitting in Section 6.5.

6.3.3 Context Types

In this section we characterize context of a sentence. We distinguish between two types of context: discourse context and similarity context.

Discourse Context: The discourse context of a sentence is formed by the previous and the following sentences in the text. As explained before, the order of the sentences carries important information. For example, adjacent sentences in a text are logically connected by certain coherence relations (e.g., *elaboration*, *contrast*) to express the meaning [10]. On a coarser level, sentences in a text segment (e.g., paragraph) address a common (sub)topic [11]. The discourse context thus captures both coherence and topic structures of a text.

Similarity Context: While the discourse context covers important discourse phenomena like *coherence* and *cohesion* [164], some applications might require a context type that is based on more direct measures of similarity, and considers relations between all possible sentences in a document and possibly across multiple documents. For example, graph-based methods for topic segmentation [145] and summarization [142] rely on complete graphs of sentences, where edge weights represent cosine similarity between sentences. In an empirical evaluation of data structures for representing discourse coherence, [146] advocate for a graph representation of discourse allowing non-adjacent connections.

Our similarity context allows any other sentence in the corpus to be in the context of a sentence depending on how similar they are. To measure the similarity, we first represent the sentences with vectors learned by Sen2Vec [44], then we measure the cosine between the vectors. We restrict the context size of a sentence for computational efficiency, while still ensuring that it is informative enough. We achieve this by imposing two kinds of constraints. First, we set thresholds for intra- and across-document connections: sentences in a document are connected only if their similarity value is above 0.5, and sentences across documents are connected only if their similarity is above 0.8. Second, we allow up to 20 most similar neighbors.

Algorithm 5: Training CON-S2V with SGD.

Input : set of sentences V, graph $G = (V, E)$
Output: learned sentence vectors ϕ
1. Initialize model parameters: ϕ and $\mathbf{w}$'s;
2. Compute noise distributions: ψ_c and ψ_g
3. **repeat**
    for *each sentence* $\mathbf{v}_i \in V$ **do**
        for *each content word* $v \in \mathbf{v}_i$ **do**
            (a) Generate a positive pair $(\mathbf{v}_i, v)$ and S negative pairs $\{(\mathbf{v}_i, v^s)\}_{s=1}^{S}$ using ψ_c;
            (b) Take a gradient step for $\mathcal{L}_c(\mathbf{v}_i, v)$;
            (c) Sample a neighboring node $\mathbf{v}_j$ from $\mathcal{N}(\mathbf{v}_i)$;
            (d) Generate a positive pair $(\mathbf{v}_i, \mathbf{v}_j)$ and S negative pairs $\{(\mathbf{v}_i, \mathbf{v}_j^s)\}_{s=1}^{S}$ using ψ_g;
            (e) Take a gradient step for $\mathcal{L}_g(\mathbf{v}_i, \mathbf{v}_j)$;
            (f) Take a gradient step for $\mathcal{L}_r(\mathbf{v}_i, \mathcal{N}(\mathbf{v}_i))$;
        end
    end
until *convergence*;

6.3.4 Training

Algorithm 5 illustrates the SGD-based algorithm to train our model. We first initialize the model parameters; the sentence vectors ϕ are initialized with small random numbers sampled from uniform distribution $\mathcal{U}(-0.5/d, 0.5/d)$, and the weight

parameters $\mathbf{w}$'s are initialized with zero. We then compute the noise distributions ψ_c and ψ_g for $\mathcal{L}_c(\mathbf{v}_i, v)$ and $\mathcal{L}_g(\mathbf{v}_i, \mathbf{v}_j)$ losses, respectively.

We iterate over the sentences in our corpus in each epoch of SGD, as we learn their representations. Specifically, to estimate the representation of a sentence, for each word token in the sentence, we take three gradient steps to account for the three loss functions in Eq. 6.1. By making the same number of gradient updates, the algorithm weights equally the contributions of content and context.

6.4 Evaluation Tasks

Different methods have been proposed to evaluate sentence representation models [48]. However, unlike most existing methods, our model learns representation of a sentence by exploiting contextual information in addition to the content.[3] To be able to evaluate our models, we thus require corpora of annotated sentences with ordering and document boundaries preserved, i.e., documents with sentence-level annotations. To the best of our knowledge, no previous work has used or released such corpora for learning sentence representation. In this work, we automatically create large corpora of documents with sentence-level topic annotations, which are then used to evaluate our models on topic *classification* and *clustering* tasks. Additionally, we evaluate our models on a *ranking* task of generating *extractive* single-document summaries. In the interest of coherence, we present the summarization task, followed by topic classification and clustering.

6.4.1 Extractive Summarization

Extractive summarization is often considered as a ranking problem, where the goal is to select the most important sentences to form an abridged version of the source document(s) [148]. Unsupervised methods are the predominant paradigm

[3]For this reason, we did not evaluate our models on tasks previously used to evaluate sentence representation models.

Table 6.1.: Basic statistics about the DUC datasets.

Dataset	#Doc.	#Avg. Sen.	#Avg. Sum.
DUC 2001	486	40	2.17
DUC 2002	471	28	2.04

for determining sentence importance. We use the popular graph-based algorithm LexRank [142]. The input to LexRank is a graph, where nodes represent sentences and edges represent cosine similarity between *vector representations* (learned by models) of the two corresponding sentences. We run the PageRank [149] on the graph to compute importance of each sentence in the graph.[4] The top-ranked sentences are extracted as the summary sentences.

Data: We use the benchmark datasets from DUC-2001 and DUC-2002, and evaluate our representation models on the official task of generating a 100-words summary for each document in the datasets.[5] The sentence representations are learned independently a priori from the same source documents. Table 6.1 shows some basic statistics about the datasets. For each document, 2-3 short ($\approx$ 100 words) human authored reference summaries are available, which we use as gold summaries for automatic evaluation.

Metric: We use the widely used automatic evaluation metric ROUGE [150] to evaluate the system-generated summaries. ROUGE computes n-gram recall between a system-generated summary and a set of human-authored reference summaries. Among the variants, ROUGE-1 (i.e., $n = 1$) has been shown to correlate well with human judgments for short summaries [150]. Therefore, we only report ROUGE-1 in this paper.

[4]The dumping factor in the PageRank was set to 0.85.
[5]http://www-nlpir.nist.gov/projects/duc/guidelines

Table 6.2.: Statistics about Reuters and Newsgroups.

Dataset	#Doc.	Total #sen.	Annot. #sen	Train #sen.	Test #sen.	#Class
Reuters	9,001	42,192	13,305	7,738	3,618	8
Newsgroups	7,781	95,809	22,374	10,594	9,075	8

6.4.2 Topic Classification and Clustering

We evaluate our models by measuring how effective the learned vectors are when they are used as features for classifying or clustering the sentences into topics. Text categorization has now become a standard in evaluating cross-lingual word embeddings [147]. We use a MaxEnt classifier and a K-means++ [151] clustering algorithm for classification and clustering tasks, respectively.

Data: We use the standard text categorization corpora: *Reuters-21578* and *20-Newsgroups*. Reuters-21578 (henceforth Reuters) is a collection of $21,578$ news documents covering 672 topics.[6] 20-Newsgroups (henceforth Newsgroups) is a collection of about $20,000$ news articles organized into 20 different topics.[7] We used the standard train-test splits (*ModApte* split for Reuters) split, and selected documents only from the 8 most frequent topics in both datasets. The selected topics for Reuters dataset are: *acq, crude, earn, grain, interest, money-fx, ship,* and *trade*. The topics selected for Newsgroups dataset are: *sci.space*, *sci.med*, *talk.politics.guns*, *talk.politics.mideast*, *rec.autos*, *rec.sport.baseball*, *comp.graphics*, and *soc.religion.christian*.

Generating Sentence-level Topic Annotations: As mentioned above, both *Newsgroups* and *Reuters* datasets come with document-level topic annotations. However, we need sentence-level annotations for our evaluation. One option is to assume that all the sentences of a document share the same topic label as the document. However, this naive assumption induces a lot of noise. Although sentences in a document collectively address a common topic, not all sentences are directly linked to that topic, rather they play supporting roles. To minimize this noise, we employ our extractive summarizer introduced in Section 6.4.1 to select the top 20% sentences of each document as representatives of the document, and assign them the same topic label as the topic of the document. We used Sen2Vec [44] representation to compute

[6] http://kdd.ics.uci.edu/databases/reuters21578/
[7] http://qwone.com/ jason/20Newsgroups/

cosine similarity between two sentences in LexRank. Table 6.2 shows statistics of the resulting datasets. Note that the sentence vectors are learned independently from an entire dataset (#Total Sen. column in Table 6.2).

Metrics: We report raw **acc**uracy, macro-averaged $\mathbf{F_1}$-score, and Cohen's $\boldsymbol{\kappa}$ for comparing classification performance. For clustering, we report **V**-measure [154] and adjusted mutual information or **AMI** [165]. We use all the annotated sentences (train+test in Table 6.2) for comparing clustering performance.

6.5 Experiments

In this section, we present our experiments — the models we compare, their settings, and the results.

6.5.1 Models Compared

We compare our representation learning model against several baselines and existing models. We also experiment with a number of variations of our proposed model considering which components of the model are active, types of context, and how we incorporate the context. For clarity, in our tables we show results divided into five evaluation groups:

(I) Existing Distributed Models: This group includes Sen2Vec [44], W2V-avg, C-PHRASE [156], FastSent [48], and Skip-Thought [54].

We used Mikolov's implementation[8] of **Sen2Vec**, which gave better results than gensim's version when validated on the sentiment treebank [166]. Following the recommendation by [44], we concatenate the vectors learned by DM and DBOW models. The concatenated vectors also performed better on our tasks.

[8]https://code.google.com/archive/p/word2vec/

For **W2V-avg**, we obtain a sentence vector by averaging the word vectors learned by training a skip-gram Word2vec [47] on our training set. Since code for **C-PHRASE** is not publicly available, we use pre-trained word vectors (of 300 dimensions) available from author's webpage.[9] We first add the word vectors to obtain a sentence vector, then we normalize the vector with l_2 normalization. Normalized vectors performed better on our tasks than the ones obtained by simple addition.

We use the auto-encode version of **FastSent** (FastSent+AE) since it considers both content and context of a sentence. For **Skip-Thought**, we use the pre-trained combine-skip model that concatenates the vectors encoded by uni- and bi-skip models.[10] The resultant vectors are of 4800 dimensions. The model was originally trained on the BookCorpus[11] with a vocabulary size of $20K$ words, however, it uses publicly available CBOW Word2vec vectors to expand the vocabulary size to $930,911$ words.

(II) Non-distributed Model: We use **TF-IDF** as our non-distributed baseline, where a sentence is represented by tf*idf weighting of its words.

(III) Retrofitted Models: We compare our approach of modeling context with the retrofitting method of [6]. We first learn sentence vectors using the content model only (i.e., by turning off contextual components in Eq. 6.1). Then we retrofit these vectors with the graph Laplacian $\mathcal{L}_r(\mathbf{v}_i, \mathcal{N}(\mathbf{v}_i))$ to encourage the revised vectors to be similar to the vectors of neighboring sentences and also similar to their prior representations. We consider two types of graph contexts: discourse (Ret-dis) and similarity (Ret-sim).

(IV) Regularized Models: We compare with a variant of our model, where the loss to capture distributional similarity $\mathcal{L}_g(\mathbf{v}_i, \mathbf{v}_j)$ is turned off. This model considers the same information as the retrofitting model (i.e., content and proximity), but trains the vectors in a single step. Its comparison with our complete model will tell us how

[9] http://clic.cimec.unitn.it/composes/cphrase-vectors.html
[10] https://github.com/ryankiros/skip-thoughts
[11] http://yknzhu.wixsite.com/mbweb

Table 6.3.: Optimal values of the hyper-parameters for different models on different tasks.

Dataset	Task	Sen2Vec	FastSent	W2V-avg	Reg-sim	Reg-dis	Con-S2V-sim	Con-S2V-dis
		(win. size)			*(win. size, reg. str.)*		*(win. size, reg. str.)*	
Reuters	clas.	8	10	10	(8, 1.0)	(8, 1.0)	(8, 0.8)	(8, 1.0)
	clus.	12	8	12	(12, 0.3)	(12, 1.0)	(12,0.8)	(12, 0.8)
Newsgroups	clas.	10	8	10	(10, 1.0)	(10, 1.0)	(10, 1.0)	(10, 1.0)
	clus.	12	12	12	(12, 1.0)	(12, 1.0)	(12, 0.8)	(10, 1.0)
DUC 2001	sum.	10	12	12	(10, 0.8)	(10, 0.5)	(10, 0.3)	(10, 0.3)
DUC 2002	sum.	8	8	10	(8, 0.8)	(8, 0.3)	(8, 0.3)	(8, 0.3)

much distributional similarity contributes to the overall performance. We define regularizers on two types of contexts: discourse (Reg-dis) and similarity (Reg-sim).

(V) Our Models: We experiment with two variants of our combined model, Con-S2V: one with discourse context (Con-S2V-dis), and the other with similarity context (Con-S2V-sim).

6.5.2 Model Settings

The representation dimensions were set to 300 in DM and DBOW models. The concatenation of the two vectors yields 600 dimensions for Sen2Vec. For a fair comparison, the dimensions in all other models that we train (except pre-trained C-PHRASE and Skip-Thought) were fixed to 600. All the prediction-based models were trained with SGD. Retrofitting was done using iterative method [6] with 20 iterations. The number of noise samples (S) in negative sampling was set to 5. We also used subsampling of frequent words [47], which together with negative sampling give significant speed-ups in training.

For each dataset described in Section 6.4, we randomly selected 20% documents from the training set to form a held-out validation set on which we tune the hyper-parameters. *Window size (k)* is a hyper-parameter that is common to all models. The regularized models have an additional hyper-parameter, *regularization strength*

Table 6.4.: Performance of our models on topic classification task in comparison to Sen2Vec.

	Topic Classification Results					
	Reuters			*Newsgroups*		
	F_1	Acc	κ	F_1	Acc	κ
Sen2Vec	83.25	83.91	79.37	79.38	79.47	76.16
TF-IDF	(−) 3.51	(−) 2.68	(−) 3.85	(−) 9.95	(−) 9.72	(−) 11.55
W2V-avg	(+) 2.06	(+) 1.91	(+) 2.51	(−) 0.42	(−) 0.44	(−) 0.50
C-PHRASE	(−) 2.33	(−) 2.01	(−) 2.78	(−) 2.49	(−) 2.38	(−) 2.86
FastSent	(−) 0.37	(−) 0.29	(−) 0.41	(−) 12.23	(−) 12.17	(−) 14.21
Skip-Thought	(−) 19.13	(−) 15.61	(−) 21.8	(−) 13.79	(−) 13.47	(−) 15.76
Ret-sim	(+) 0.92	(+) 1.28	(+) 1.65	(+) 2.00	(+) 1.97	(+) 2.27
Ret-dis	(+) 1.66	(+) 1.79	(+) 2.30	(+) 5.00	(+) 4.91	(+) 5.71
Reg-sim	(+) **2.53**	(+) **2.53**	(+) **3.28**	(+) 3.31	(+) 3.29	(+) 3.81
Reg-dis	(+) 2.52	(+) 2.43	(+) 3.17	(+) **5.41**	(+) **5.34**	(+) **6.20**
Con-S2V-sim	(+) 3.83	(+) 3.55	(+) 4.62	(+) 4.52	(+) 4.50	(+) 5.21
Con-S2V-dis	**(+) 4.29**	**(+) 4.04**	**(+) 5.22**	**(+) 7.68**	**(+) 7.56**	**(+) 8.80**

(λ). We tuned with $k \in \{8, 10, 12\}$ and $\lambda \in \{0.3, 0.6, 0.8, 1\}$, and we optimized F_1 for classification, AMI for clustering, and ROUGE-1 for summarization. Table 6.3 shows the hyper-parameters and their optimal values for each task. We evaluated our models on the test sets with these optimal values. We ran each experiment five times and take the average of the evaluation measures to avoid any randomness in results.

6.5.3 Classification and Clustering Results

Table 6.4 and 6.5 shows the results of the models on topic classification and clustering tasks, respectively. The scores are shown in comparison to Sen2Vec.

Unsurprisingly, Sen2Vec outperforms TF-IDF representation (row 6) by a good margin on both tasks. It gets improvements of up to 11.6 points on classification, and up to 30.6 points on clustering. This is inline with the finding of [44], and

Table 6.5.: Performance of our models on topic clustering tasks in comparison to Sen2Vec.

	Topic Clustering Results			
	Reuters		*Newsgroups*	
	V	AMI	V	AMI
Sen2Vec	42.74	40.00	35.30	34.74
TF-IDF	(−) 21.34	(−) 20.14	(−) 29.20	(−) 30.60
W2V-avg	(−) 11.96	(−) 10.18	(−) 17.90	(−) 18.50
C-PHRASE	(−) 11.94	(−) 10.80	(−) 1.70	(−) 1.44
FastSent	(−) 15.54	(−) 13.06	(−) 34.40	(−) 34.16
Skip-Thought	(−) 29.94	(−) 28.00	(−) 27.50	(−) 27.04
Ret-sim	(+) 3.72	(+) 3.34	(+) 5.22	(+) 5.70
Ret-dis	(+) 4.56	(+) 4.12	(+) 6.28	(+) 6.76
Reg-sim	(+) 4.76	(+) 4.40	(+) **12.78**	(+) 12.18
Reg-dis	(+) **7.40**	(+) **6.82**	(+) 12.54	(+) **12.44**
Con-S2V-sim	**(+) 14.98**	**(+) 14.38**	(+) 13.68	(+) 13.56
Con-S2V-dis	(+) 9.30	(+) 8.36	**(+) 15.10**	**(+) 15.20**

demonstrates the benefits of using distributed representation over sparse BOW representations.

Simple averaging of Word2vec vectors performs quite well for classification, especially, on Reuters, where it outperforms Sen2Vec by 1.9 to 2.5 points. [48] also reported similar findings on five out of six datasets. However, averaging does not perform well on clustering, where the scores are 10.2 to 18.5 points below than Sen2Vec.

Simple addition-based composition of C-PHRASE word vectors performs poorly on both tasks – lower than Sen2Vec by 2 to 3 points on classification and by 1.4 to 11.9 points on clustering.

Unexpectedly, FastSent and Skip-Thought perform quite poorly on both tasks. Skip-Thought, in particular, has the worst performance on both tasks. These results contradict the claim made by [54] that skip-thought vectors are generic. To investigate if the poor results are due to shift of domains (*book* vs. *news*), we also trained Skip-Thought on our training corpora with vector size 600 and vocabulary size $30K$. The performance was even worse. We hypothesize, this is due to our training set size, which may not be enough for the heavy model. Also, Skip-Thought does not perform any inference to extract the vector using a context – although the model was trained to generate neighboring sentences, context was ignored when the encoder was used to extract the sentence vector.

Regarding FastSent, although its classification performance on Reuters is comparable to Sen2Vec, it performs poorly on Newsgroups, where the measures are 12.2 to 14.3 points lower than Sen2Vec. The differences get bigger in clustering. The reason could be that FastSent does not learn sentence representations directly, rather it simply adds the word vectors. Note that FastSent was outperformed by TF-IDF in all classification tasks in [48]. Since both Skip-Thought and FastSent learn representations by predicting contents of adjacent sentences, the learned vectors might capture linguistic properties that are more specific to the neighbors.

We also experimented with SAE and SDAE auto-encoders proposed in [48]. However, they performed poorly on our tasks (thus not shown in the table). For example,

SAE gave accuracies of around 40% on reuters and 18% on newsgroups. This is similar to what [48] observed. They propose to use pretrained word embeddings to improve the results. We did not achieve significant gains by using pretrained embeddings on our tasks.

Interestingly, the retrofitting and regularized models improve over Sen2Vec on both tasks, showing gains of up to 6.2 points on classification and up to 12.8 points on clustering. The improvements in most cases are significant. This demonstrates that proximity hypothesis is beneficial for these tasks.

When we compare regularized models with retrofitted ones, we observe that regularized models consistently outperform the retrofitted counterparts on both tasks with improvements of up to 1.6 points on classification and up to 7.6 points on clustering. This demonstrates that incorporating contextual information by means of regularization is more effective than retrofitting. This could be due to the fact that regularization approach induces contextual information while learning the vectors from scratch as opposed to revising them in a post-processing step.

Finally, we observe further improvements for our complete models (Con-S2V variants) on both tasks. Compared to the best regularized models, our models deliver improvements of up to 2.6 points on classification and up to 7.6 points on clustering. This demonstrates that by including the neighbor prediction component to model distributional similarity, our model captures complementary contextual information to what is captured by the regularized models. A comparison between the context types reveals that discourse context is more beneficial than similarity context in most cases, especially for classification. For clustering, similarity context gives better results in a few cases (e.g., on Reuters). Overall, our best model outperforms the best existing model by up to 8.8 and 15.20 points on classification and clustering tasks, respectively.

Table 6.6.: ROUGE-1 scores of the models on DUC datasets in comparison with Sen2Vec.

	DUC'01	DUC'02
Sen2Vec	43.88	54.01
W2V-avg	(−) 0.62	(+) 1.44
C-PHRASE	(+) 2.52	(+) 1.68
FastSent	(−) 4.15	(−) 7.53
Skip-Thought	(+) 0.88	(−) 2.65
TF-IDF	**(+) 4.83**	(+) 1.51
Ret-sim	(−) 0.62	(+) 0.42
Ret-dis	(+) 0.45	(−) 0.37
Reg-sim	(+) 2.90	(+) 2.02
Reg-dis	(−) 1.92	(−) 8.77
Con-S2V-sim	(+) 3.16	**(+) 2.71**
Con-S2V-dis	(+) 1.15	(−) 4.46

6.5.4 Summarization Results

Table 6.6 shows ROUGE-1 scores of our models on DUC datasets for the summary length of 100 words. W2V-avg performs well achieving comparable score to Sen2Vec on DUC'01 and 1.4 points improvement on DUC'02. C-PHRASE outperforms Sen2Vec by 2.5 and 1.7 points on DUC'01 and DUC'02, respectively. FastSent and Skip-Thought again perform disappointingly. Sen2Vec outperforms FastSent by 4.15 and 7.53 points on DUC'01 and DUC'02, respectively. Skip-Thought performs comparably to Sen2Vec on DUC'01, but gets worse on DUC'02.

Interestingly, TF-IDF performs quite well on this task. It gives the top score on DUC'01 (i.e., 48.7 ROUGE-1), and an improvement of 1.5 points over Sen2Vec on DUC'02. These results suggest that existing distributed representation methods are inferior to traditional methods in modeling aspects that are necessary for measuring sentence importance.

Retrofitted models give mixed results and fail to get significant improvement over Sen2Vec. On the other hand, with similarity context, regularized model improves over Sen2Vec by 2 to 3 points. This again suggests that regularization is a better method to incorporate context proximity. By including the neighbor prediction component to incorporate distributional similarity, our combined model improves the scores further; it achieves the second best result on DUC'01, and becomes top-performer on DUC'02. It is not surprising that similarity context is more suitable than discourse context for this task. From a context of topically similar sentences, our model learns representations that capture linguistic aspects related to information centrality. Given that the existing models fail to beat the TF-IDF baseline on this task, our results are rather encouraging.

6.6 Chapter Summary

We have presented a novel model to learn distributed representation of sentences by considering content as well as context of a sentence. Our results on tasks involving classifying, clustering and ranking sentences confirm that extra-sentential contextual information is crucial for modeling sentences, and this information is best captured by our model that comprises a neighbor-based prediction component and a regularization component to capture distributional similarity and contextual proximity, respectively.

One important property of our model is that it encodes a sentence directly, and it considers neighboring sentences as atomic units. Apart from the improvements that we achieve in various tasks, this property makes our model quite efficient to train compared to *compositional* methods like encoder-decoder models (e.g., SDAE, Skip-Thought) that compose a sentence vector from the word vectors. Encoder-decoder approaches attempt to capture the structure of a sentence, which could be beneficial to model long distance relations between words (e.g., negation in sentiment classification). It would be interesting to see how our model compares with compositional models on sentiment classification task. However, this would require creating a new

dataset of comments with sentence-level sentiment annotations. We intend to create such datasets and evaluate the models in the future.

7. FS^3: A SAMPLING BASED METHOD FOR TOP-K FREQUENT SUBGRAPH

7.1 Introduction

Frequent subgraph mining (FSM) is an important research task. It has application in various disciplines, including cheminformatics [167] for solving QSAR (Quantitative Structure Activity Relationship) task, and in bioinformatics [168] for finding structural motifs. The main objective of FSM is finding subgraph patterns that are frequent across a collection of graphs. This task is additionally useful in applications related to graph classification [169], and graph indexing [170]. Over the years, a good number of algorithms for FSM task have been proposed, examples include Subdue [55], AGM [56], FSG [57], gSpan [58], DMTL [59], and Gaston [60]. These algorithms are proven to be effective for finding frequent subgraphs from input graphs which are small and sparse. However, for general graphs, FSM task is not scalable due to the inherent complexity of this task. In fact, Horváth et al. have shown that FSM cannot be solved in output polynomial time [171]. Lack of scalability of FSM task has also been shown empirically. For instance, Chaoji et al. have applied FSM on a small dataset (only 3 graphs) of protein-protein interaction (PPI) graphs, each graph having 2154 nodes on average; but the most efficient of the existing FSM algorithms cannot mine all the frequent subgraphs from this dataset in a days of running even with 100% support value [64]. In this era of big data, we are collecting graphs of even larger size, so an efficient algorithm for FSM is of huge demand.

We provide some quantitative evidences so that a reader can comprehend the intractability of the FSM task. For this we mine subgraphs from a protein structure (PS) dataset (see Section 7.6 for details) that contains only 90 graphs, each having 67

Table 7.1.: Highlights of the lack of scalability of existing frequent subgraph mining methods while mining the PS dataset. Time indicates the running time of the fastest version of Gaston [60].

Dataset Statistics: # graphs: 90, avg. # vertices: 67, avg. # edges: 268
 # node labels: 20, # edge labels: 3

Time vs Max. subgraph size (min-sup is fixed at 40%)		**Time vs different minsup (Max-size is fixed at 8)**		**Search Space vs subgraph size**	
Max-size	Time	Support (%)	Time	Size	Induced Subgraph Count
8	6 minutes	28	1.1 hours	6	26 millions
9	2.8 hours	22	3.5 hours	7	157 millions
10	> 1.5 days	17	9 hours	8	947 millions
		11	>16 hours	9	5000 billions

vertices and 268 edges, on average. First we use a 64-bit binary of gSpan software [1]. Using a large 40% support the mining task could last only for few minutes, after that the OS aborted the gSpan process because by that time it had consumed more than 80% of 128 GB memory of a server machine. We then attempted the identical mining task using Gaston software [60] [2], which kept running for more than 2 days. Then we ran the same software with a restriction on the maximum size of the subgraphs to be mined (only Gaston allows such an option), yet the mining task seems to be insurmountable.

Table 7.1 shows more detailed postmortem of Gaston's lack of scalability for the subgraph mining task on the PS dataset. When the maximum subgraph size is 8, with 40% support the mining task finishes in 6 minutes. But, for size 9, it takes 2.8 hours, and for size 10, it does not terminate in 1.5 days. Note that, in real-life application 40% is considered too large for a support threshold, so we also show some results for smaller support value while restricting the maximum size of subgraph to 8. Even for a restriction of max-size 8, for 11% support the process takes more than 16 hours. In the same table, we also show the size of the subgraph space [3] rounded to nearest millions. As we can see, the subgraph space grows exponentially, and for size 9 the number of subgraphs exceeds 5 billions. Graphs with 67 vertices and 268 edges are actually considered small in many domains, still we can't mine such graphs completely using existing subgraph mining methods.

Lack of scalability of the existing subgraph mining methods for relatively larger input graphs is simply due to the inherent intractability of the FSM task as it is defined. Note that, following the existing definition of FSM, these algorithms ensure completeness, i.e., they enumerate all the subgraphs that are frequent under a user-defined minimum support value. So, they must traverse the entire subgraph space,

[1] gSpan is the most polular among the existing graph mining methods. We use the Linux binary available from the inventors: `http://www.cs.ucsb.edu/~xyan/software/gSpan.htm`

[2] Gaston is the fastest graph mining algorithm at present as verified by independent comparison, see [172]

[3] It consists of graphs that are induced subgraph of at least one of the database graphs.

which grows exponentially with the size of the input graph. Although existing methods prune part of the search space by using the support values of the known frequent subgraphs and applying anti-monotone properties of the support, this effort cannot cope up with the exponential expansion of the search space as the input graphs become larger and denser. Besides, any exact method for frequent subgraph mining needs to solve numerous subgraph isomorphism (SI)—a known $\mathcal{N}P$-complete problem, which denies the FSM problem the status of output polynomial class [171]. One may sacrifice the completeness and obtain a subset of frequent patterns as a partial output by using one of the existing algorithms; however, because of artificial order of enumeration imposed by the above mining algorithms, the patterns in the partial output are not representative of the entire set of frequent patterns. Even distributed methods of FSM [173, 174] fail to reverse the scalability downfall, as the gain of distributed methods is polynomial, whereas the search space expansion with respect to the size of the input graphs is exponential.

To cope with the scalability problem, in recent years researchers have proposed alternative paradigms of frequent subgraph mining, which are neither complete, nor enumerative. Some of these works find frequent patterns considering their subsequent application in knowledge discovery tasks. For example, there are methods [175, 176] that directly mine frequent subgraphs for using them as features for graph classification. Another family of works [65, 66] perform MCMC random walk over the space of *frequent* patterns and sample only a subset of all the frequent subgraphs. However, the above sampling based methods also solve numerous SI task for ensuring that the random walk traverses only over the frequent patterns, so they are also not scalable when the input graphs are large.

There also exist some methods that find a subset of frequent subgraphs, such as, frequent induced subgraphs (AcGM [177]), maximal frequent subgraphs (SPIN [61], MARGIN [62]), or close frequent subgraphs (CloseGraph [63]). In each of these cases, since the objective is to mine a specific subset of frequent subgraphs, effective pruning strategies can be exploited, which, sometimes, offer noticeable speed-up over

traditional frequent subgraph mining. Nevertheless pruning typically offers a constant factor speed-up, which is not much beneficial while mining large input graphs. Also, like traditional subgraph mining all these methods perform numerous SI task for ensuring the minimum support threshold, so they also are not scalable. We ran both AcGM, and SPIN on the PS dataset; for a 10% support both the methods run for a while, but the mining task was aborted by the OS after the software consumed more than 100 GB of memory.

Scalable subgraph mining is achievable if the database contains graphs from a restricted class for which the SI task is tractable (polynomial). Some recent works on subgraph mining actually explored this option, examples include mining outerplaner graphs [178], or mining graphs with bounded treewidth [179], or graphs where each of the vertices have a distinct label [180]. However, except chemical graphs, for which the treewidth value is around 3, general graphs from other domains rarely adhere to such restrictions. The good mining performance on treelike graph is probably the reason that the existing methods only use chemical graphs for presenting their experiment results [4]. For general graphs the only viable option is to discard the SI test altogether. A recent work, called GAIA [169] uses this idea; however, the scope of GAIA is limited for mining only discriminatory subgraphs that are good for graph classification, so it is not applicable for mining frequent subgraphs.

For frequent subgraph mining task, discarding SI test is possible, only if we relax the minimum support constraint such that the returned subgraphs are likely to be frequent, but they do not necessarily satisfy a user-defined minimum support requirement. This seems to be an over-simplification which evades the main purpose of frequent pattern mining—after all, in pattern mining, the minimum support constraint is the threshold that decides which of the candidate patterns are frequent and which are not. However, in practice, the minimum support constraint has small significance, because a user seldom knows what is the right value of minimum support

[4]DTP dataset (available from `http://dtp.nci.nih.gov/branches/dscb/repo_open.html`) is the most popular graph mining dataset, which is mostly tree with an average vertex and edge size of 31 and 34 respectively.

parameter to find the best patterns for her anticipated use [181]. Further, it is a hard-constraint which can discard a supposedly good pattern that narrowly misses the support threshold. An alternative to minimum support constraint can be a size constraint, in which a user provides a size for the pattern that she is looking for; in the context of subgraph mining, the size can be the number of vertices (or edges) that a pattern should have. The argument in favor of this choice is that it is easier for an analyst to define a size constraint than defining a minimum support constraint using his domain knowledge—a size constraint can be equal to the size of a meaningful sub-unit in the input graph. For instance, if the input graph is a social network, a size constraint can be equal to the size of a typical community in that network.

In this work, we propose a method for frequent subgraph mining, called FS^3, that is based on sampling of subgraphs of a fixed size [5]. Given a graph database $\mathcal{G}$, and a size value ℓ, FS^3 samples subgraphs of size-ℓ from the database graphs using a 2-stage sampling. In the first stage of a sampling iteration, FS^3 chooses one of the database graphs (say, G_i) uniformly, and in the second stage it chooses a size-ℓ subgraph of g using a Markov Chain Monte Carlo (MCMC) sampling. The sampling distribution of the second stage is biased such that it over-samples the graphs that are likely to be frequent over the entire database $\mathcal{G}$. FS^3 runs the above sampling process for many times, and uses an innovative priority queue to hold a small set of most frequent subgraphs. The unique feature of FS^3 is that unlike earlier works which are based on sampling [65], FS^3 does not perform any subgraph isomorphism (SI) test, so it is scalable to large graphs. By choosing different values of ℓ, user can find a succinct set of frequent subgraphs of different sizes. Also, as the number of samples increases, FS^3's output progressively converges to the top-k most frequent subgraphs of size ℓ. So user can run the sampler as long as he wants to obtain more precise results.

We claim the following contributions in this work:

[5]The name FS^3 should be read as *F-S-Cube*, which is a compressed representation of the 4-gram composed of the bold letters in **F**ixed **S**ize **S**ubgraph **S**ampler.

- We propose FS^3, a sampling based method for mining top-k frequent subgraphs of a given size, ℓ. FS^3 is scalable to large graphs, because it does not perform the costly subgraph isomorphism test.
- We design several innovative queue mechanisms to hold the top-k frequent subgraphs as the sampling proceeds.
- We perform an extensive set of experiments and analyze the effect of every control parameter that we have used to validate the effectiveness and efficiency of FS^3.

7.2 Background

7.2.1 Graph, Induced Subgraph, Frequent Subgraph Mining

Let $G(V, E)$ is a *graph*, where V is the set of vertices and E is the set of edges. Each edge $e \in E$ is denoted by a pair of vertices (v_i, v_j) where, $v_i, v_j \in V$. A graph without self-loop or multi edge is a simple graph. In this work, we consider simple, connected, and undirected graphs. A *labeled graph* $G(V, E, L, \Psi)$ is a graph for which the vertices and the edges have labels that are assigned by a labeling function, $\Psi : V \cup E \rightarrow L$ where L is a set of labels.

A graph $G' = (V', E')$ is a *s*ubgraph of G (denoted as $G' \subseteq G$) if $V' \subseteq V$ and $E' \subseteq E$. A graph $G' = (V', E')$ is a *v*ertex-induced subgraph of G if G' is a subgraph of G, and for any pair of vertices $v_a, v_b \in V'$, $(v_a, v_b) \in E'$ if and only if $(v_a, v_b) \in E$. In other words, a *vertex-induced* subgraph of G is a graph G' consisting of a subset of G's vertices together with all the edges of G whose both endpoints are in this subset. In this paper, we have used the phrase *induced subgraph* for abbreviating the phrase vertex-induced subgraph. If G' is a (induced or non-induced) subgraph of G and $|V'| = \ell$, we call G' a ℓ-subgraph of G.

Let, $\mathcal{G} = \{G_1, G_2, \ldots, G_n\}$ be a graph database, where each $G_i \in \mathcal{G}, \forall i = \{1 \ldots n\}$ represents a labeled, undirected and connected graph. $\mathbf{t}(g) = \{G_i : g \subseteq G_i \in \mathcal{G}\}, \forall i =$

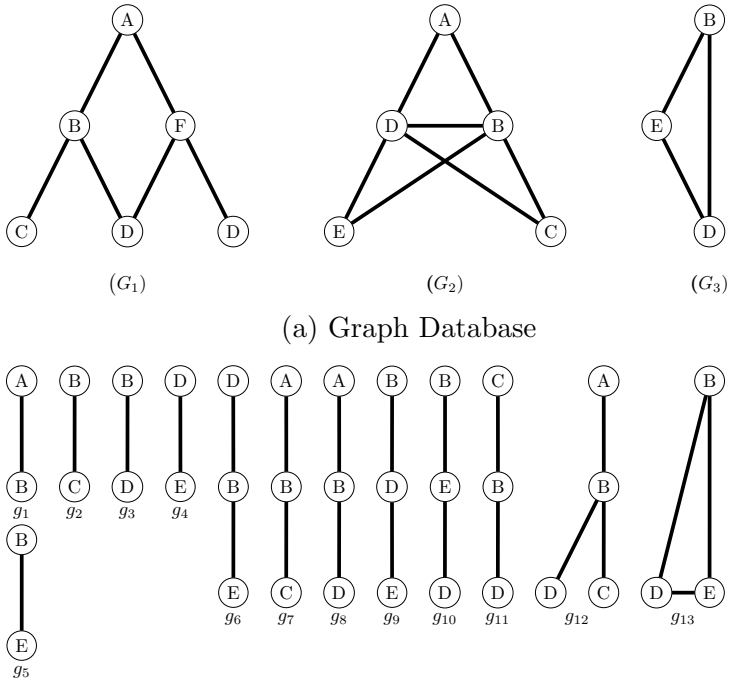


Fig. 7.1.: (a) Graph database with 3 graphs (b) Frequent subgraph of (a) with $minsup = 2$.

$\{1 \ldots n\}$, is the *support-set* of the graph g. This set contains all the graphs in $\mathcal{G}$ that have a subgraph isomorphic to g. The cardinality of the *support-set* is called the *support* of g. g is called frequent if *support* $\geq \pi^{\mathbf{min}}$, where $\pi^{\mathbf{min}}$ is predefined/user-specified *minimum support (minsup)* threshold. Given the graph database $\mathcal{G}$, and minimum support $\pi^{\mathbf{min}}$, the task of a frequent subgraph mining algorithm is to obtain the set of frequent subgraphs (represented by $\mathcal{F}$). While computing support, if an FSM algorithm enforces induced subgraph isomorphism, it obtains the set of frequent induced subgraphs (represented by $\mathcal{F}_I$). It is easy to see that $\mathcal{F} \subseteq \mathcal{F}_I$.

Example: In Figure 7.1, G_3 is a subgraph of G_2; g_{12} is a subgraph of G_1, and G_2, but it is an induced subgraph of G_1 only. Let's consider the graphs in Figure 7.1(a) as a database of 3 graphs, $\mathcal{G} = \{G_1, G_2, G_3\}$; with $\pi^{\mathbf{min}} = 2$, there are thirteen frequent subgraphs, which are shown in Figure 7.1(b). If we want to obtain only the induced frequent subgraphs, $g_6, g_8, g_9, g_{10}, g_{11}$, and g_{12} are not frequent for a minimum support of 2, however the remaining patterns are frequent. ■

7.3 Related Works

Frequent subgraph discovery is a well-studied problem with many existing methods, including Subdue [55], AGM [56], FSG [57], gSpan [58], DMTL [59], and Gaston [60]. All of these methods follow the definition of FSM provided in Section 7.2. They work well for problem instances where the graphs in the graph database are small and sparse, but they do not scale well with the size and the density of the input graphs. Note that, the lack of scalability issue of the existing methods for the large input graph is not a limitation of the existing methods, rather it is due to the strict definition of the FSM task itself.

To alleviate the scalability concern, researchers have proposed some alternative solutions, which do not discover all the frequent subgraphs. The first such attempt is to discover only a subset of frequent subgraphs, which are maximal [61, 62], or

closed [63]. However for large input graphs, algorithms for finding maximal or closed frequent subgraphs are not scalable, as they prune only a small part of the search space. Later, Chaoji et al. [64] have proposed ORIGAMI, a graph mining method that returns a set of random maximal frequent subgraphs. Starting from a null graph, ORIGAMI extends the graph by adding edges randomly as long as the graph remains frequent. Repeating this process returns a random subset of maximal frequent subgraphs. The major advantage of ORIGAMI over the existing methods is that the former generates patterns in a random order, so an incomplete pattern-set from ORIGAMI is more representative than an equal-sized pattern-set obtained from a partial run of other existing graph mining methods. Later, Hasan and Zaki proposed MCMC sampling based methods for uniformly sampling of a set of frequent [65] and maximal frequent [66] subgraphs. Due to the uniformity guaranty, such methods provide a small set of frequent subgraphs which are ideal as a representative pattern set. However, all the above methods still solve subgraph isomorphism test for ensuring the minimum support threshold, which makes them inefficient when the input graphs become large. Our work complement existing works as we are interested to obtain a solution for mining frequent subgraphs from large input graph, for which existing methods does not scale.

There also exist works [67–71, 182] that mine frequent subgraphs from a single input graph. Our work is related to these works as our propose method samples subgraphs from a single graph which is chosen uniformly from the graph database. However the objective of our work is different from these works, as they aim to discover network motifs in a single network, but we are interested to find subgraphs that are frequent over a collection of graphs.

7.4 Problem Formulation and Solution Approach

Our objective is to obtain a small collection of frequent subgraph patterns from a database of large input graphs. For this, we like to design a subgraph mining method

that does not perform the costly subgraph isomorphism (SI) test. Without SI test, the exact support values of a (sub)graph in the database graphs are impossible to obtain. So, we deviate from the traditional definition of *frequent* that is used in the FSM literature, rather we call a graph frequent if its *expected-support* (defined in the next paragraph) is comparably higher than that of other same-sized graphs. When a graph grows larger, its support-set naturally shrinks, so keeping the size as an invariant makes sense, otherwise the output set of our method will be filled with small patterns (one-edge or two-edge) that have the highest support among all the frequent patterns. However, note that the size is only a parameter, not a constraint; i.e., a user can always run different mining sessions with different size values as she desires. A formal description of our research task is as below: given a graph database $\mathcal{G} = \{G_i : 1 \leq i \leq n\}$, a user-defined size value ℓ and an integer k, return a list of top-k frequent ℓ-subgraphs , where the frequency of a subgraph is determined probabilistically.

Our solution to this task is a sampling-based method, call FS^3—a sampling iteration of FS^3 samples a random size-ℓ subgraph (induced or non-induced depending on the user requirement) g from one of the database graphs (say G_i), the later chosen uniformly. We call g frequent, if an identical copy of it is sampled from many of the input graphs in different sampling iterations of FS^3. In a sampling session, the number of distinct input graphs from which g is sampled is called its *expected-support* and is denoted as $support_a(g)$. Clearly actual support of g ($support(g)$) is an upper bound of the expected support of g ($support_a(g)$); generally speaking, these two variables are positively correlated, so we use expected-support as a proxy of real support, and thus FS^3 returns those ℓ-subgraphs that are among the top-k in terms of *expected-support*.

There are several challenges in the above solution approach. First, when the input graphs in $\mathcal{G}$ are large, for a typical ℓ-value, the number of possible ℓ-subgraphs of G_i is in the order of millions (or even billions, see Table 7.1), so if we sample a ℓ-subgraph from G_i uniformly out of all ℓ-subgraphs of G_i, the chance that we will sample a frequent ℓ-subgraph is infinitesimally small. More challenging is the fact

that we do not know how many ℓ-subgraphs exist for each of the input graphs in $\mathcal{G}$, so a direct sampling method is impossible to obtain. To cope with these challenges, FS^3 invents a novel Markov Chain Monte Carlo sampling which performs a random walk over the space of ℓ-subgraphs of the graph G_i; in this sampling, the desired distribution is non-uniform, which biases the walk to choose ℓ-subgraphs that are potentially frequent. Besides the above, another challenge of our solution approach is that we do not have unlimited memory, so during the sampling process, we can store only a limited number of sampled subgraphs in a priority queue; when the queue gets full, we have to identify which of the sampled subgraphs we will continue to maintain in the queue. FS^3 solves this with a novel queue management mechanism.

7.5 Method

FS^3 has two main components. A ℓ-subgraph sampler, and a queue manager. The first component samples a ℓ-subgraph using MCMC sampling from a database graph, G_i, later chosen uniformly. The second component maintains a priority queue of top-k frequent subgraphs of the input database $\mathcal{G}$. We discuss each of the components in the following subsections.

7.5.1 MCMC Sampling of a ℓ-subgraph from a Graph Database

The sample space of MCMC walk of FS^3 is the set of ℓ-subgraphs of a database graph G_i. At any given time, the random walk of FS^3 visits one of the ℓ-subgraphs of G_i. It then populates all of its neighboring ℓ-subgraphs and (probabilistically) chooses one from them as its next state using MH algorithm. Below, We discuss the setup of MCMC sampling, including target distribution, and state transition.

Target Distribution: The target distribution of the MCMC walk of FS^3 is biased so that the ℓ-subgraphs that are likely to be frequent are sampled more often. Formally, this distribution is a scoring function $f : \Omega \rightarrow \mathbb{R}_+$; f maps each graph in Ω (set

of all ℓ-subgraphs) to a positive real number such that the higher the support of a graph, the higher its score. For efficiency sake, we want the scoring function f to be locally computable, and computationally light. It is not easy to find such a distribution up-front, because the support information of a ℓ-subgraph is not available until we discover that graph; even if we have discovered the graph, and its partial support is available to us, we cannot use that partial support information in the target distribution, because if we do so it will bias the walk towards some patterns that are already been discovered, but they may not be amongst the most frequent ones. Also remember, FS^3 excludes the option of finding actual support of an ℓ-subgraph, because its goal is to avoid subgraph isomorphism test altogether.

In FS^3, we have used two kinds of scoring functions: s_1 and s_2. For a subgraph g, $s_1(g)$ is the average of the (actual) *support* of the constituting edges of g. Mathematically, $s_1(g) = \frac{1}{|E(g)|}\sum_{e \in E(g)} support(e)$. Here, $E(g)$ denotes the set of edges of g. $s_2(g)$ is the cardinality of the intersection set generated by intersecting the *support-set* of each of the constituting edges of g, i.e., $s_2(g) = \left|\bigcap_{e \in E(g)} support\text{-}set(e)\right|$. The intuition behind these choices is that if g is frequent, all its edges are frequent, so its score $s_1(g)$ is high, same is true for $s_2(g)$. The reverse is not necessarily true, i.e., there can be a graph, for which the average support or the set intersection count of its edge-set is high, but the graph is infrequent, so the above scoring functions may sample a few false positive (however, no false negative) patterns. Nevertheless, in real-life graphs the actual support of a subgraph is significantly correlated with its s_1 and s_2 score, which we will show in the experiment section. Besides, when the sampling process discovers an ℓ-subgraph, its scores can be computed instantly from the *support-set* of its edges—the latter can be obtained cheaply during the initial read of the database graphs.

Example: Let us consider the graphs in Figure 7.1(a) as a database of 3 graphs, $\mathcal{G} = \{G_1, G_2, G_3\}$, and g_{13} in Figure 7.1(b) as a sampled graph, G from the graph database, $\mathcal{G}$. Now, from Figure 7.1, we find the *support-set* of edges BD, BE and

DE of g_{13} which are $\{G_1, G_2, G_3\}$, $\{G_2, G_3\}$, and $\{G_2, G_3\}$ respectively. So, for g_{13}, $s_1(g_{13}) = \frac{3+2+3}{3} = 2.67$, and $s_2(g_{13}) = 2$. ■

Proposition 1 $s_1(g) \geq support(g)$ *and* $s_2(g) \geq support(g)$

Proof Consider an edge $e \in E(g)$. Since $e \in E(g)$, *support-set*$(e) \supseteq$ *support-set*(g), hence $support(e) \geq support(g)$. Since this hold for all the edges, average-support of the edges is an upper bound of the support of g; hence, $s_1(g) \geq support(g)$.

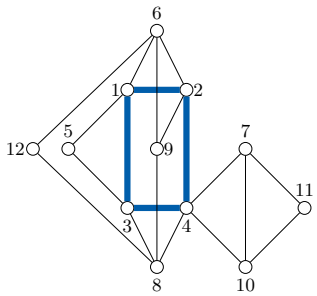
To compute $s_2(g)$ we intersect the *support-set*(e) of all edges $e \in g$. Thus, $s_2(g)$ considers the support of the edge-set of g, without considering the graphical constraint imposed by g, so $s_2(g) \geq support(g)$. ■

7.5.2 Markov Chains, and Metropolis-Hastings (MH) Method

The main goal of the Metropolis-Hastings algorithm is to draw samples from some distribution $s(x)$, called the *target distribution*. Here it is s_1 or s_2 as discussed in the previous paragraph. It can be used together with a random walk to perform Markov Chain Monte Carlo (MCMC) sampling. For this, the MH algorithm draws a sequence of samples from the target distribution as follows: (1) It picks an initial state (say, x) (2) From current state x, it samples a neighboring point y using a distribution $q(x, y)$, referred as *proposal distribution* discussed in the next paragraph; (3) Then, it calculates the *acceptance probability* given in Equation 7.1, and accepts the proposal move to y with probability $\alpha(x, y)$. The process continues until the Markov chain reaches to a stationary distribution. In this work we used MH algorithm for sampling a size-ℓ subgraph from the database graphs.

$$\alpha(x, y) = \min\left(\frac{s(y)q(y, x)}{s(x)q(x, y)}, 1\right) \tag{7.1}$$

State Transition: FS^3's MCMC walk changes state by walking from one ℓ-subgraph (say g) to a neighboring ℓ-subgraph. In our neighborhood definition, for a ℓ-subgraph



(i)

1: $\langle 1,2,3,5\rangle, \langle 1,2,4,5\rangle, \langle 1,3,4,5\rangle, \langle 1,2,3,6\rangle, \langle 1,2,4,6\rangle, \langle 1,3,4,6\rangle$

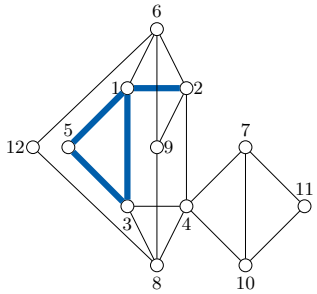
2: $\langle 1,2,3,6\rangle, \langle 1,2,4,6\rangle, \langle 2,3,4,6\rangle, \langle 1,2,3,9\rangle, \langle 1,2,4,9\rangle, \langle 2,3,4,9\rangle$

3: $\langle 1,2,3,5\rangle, \langle 1,3,4,5\rangle, \langle 2,3,4,5\rangle, \langle 1,2,3,8\rangle, \langle 1,3,4,8\rangle, \langle 2,3,4,8\rangle$

4: $\langle 1,2,4,7\rangle, \langle 1,3,4,7\rangle, \langle 2,3,4,7\rangle, \langle 1,2,4,8\rangle, \langle 1,3,4,8\rangle, \langle 2,3,4,8\rangle$
$\langle 1,2,4,10\rangle, \langle 1,3,4,10\rangle, \langle 2,3,4,10\rangle$

(ii)

(a) (i) A database graph G_i with the current state of FS^3's random walk (ii) Neighborhood information of the current state $\langle 1,2,3,4\rangle$.



(i)

1: $\langle 1,2,5,6\rangle, \langle 1,2,3,6\rangle$

2: $\langle 1,2,3,4\rangle, \langle 1,2,4,5\rangle, \langle 2,3,4,5\rangle, \langle 1,2,3,6\rangle, \langle 1,2,4,6\rangle, \langle 2,3,4,6\rangle$
$\langle 1,2,3,9\rangle, \langle 1,2,4,9\rangle, \langle 2,3,4,9\rangle$

3: $\langle 1,2,3,4\rangle, \langle 1,3,4,5\rangle, \langle 2,3,4,5\rangle\langle 1,2,3,8\rangle, \langle 1,3,4,8\rangle, \langle 2,3,4,8\rangle$

5: $\langle\rangle$

(ii)

(b) (i) The state of random walk on G_i (Figure 7.2a) after one transition (ii) Updated Neighborhood information.

Fig. 7.2.: State transition.

all other ℓ-subgraphs that have $\ell - 1$ vertices in common are its neighbor subgraph/state. To obtain a neighbor subgraph of g, FS^3 simply replaces one of the existing vertices of g with another vertex which is not part of g but is adjacent to one of g's vertices. Also, note that in g, if FS^3 includes all the edges of G_i that are induced by the set of the selected vertices, the sampled subgraph of FS^3 is always a connected induced subgraph of the database graphs. On the other hand, if it does not enforce this restriction, the sampled subgraph is a non-induced subgraph. Another important fact is that the neighborhood relation that is defined above is symmetric, which is important in MCMC walk for maintaining the detailed balance equation [183].

Example: Suppose FS^3 is sampling 4-subgraphs from the graph G_i shown in Figure 7.2a(i) using MCMC sampling. Let, at any given time the 4-subgraph, $\langle 1, 2, 3, 4\rangle$ (shown in bold lines) is the current state of this random walk. In Figure 7.2a(ii), we list its neighbor states as four comma-separated lists, one in each row. The neighbor-list in the top row is labeled by '1', which indicates that these neighbors can be obtained from the current 4-subgraph $\langle 1, 2, 3, 4\rangle$ by retaining the vertex 1 and replacing exactly one of the remaining vertices ($\{2, 3, 4\}$) with a new vertex which is adjacent to vertex 1, ensuring connectedness. Similarly, the neighbors in the second list are obtained by retaining the vertex 2 and replacing one of the remaining vertices with a vertex from 2's adjacency list. The information in the third and fourth lists are populated in a similar manner. As shown in the top-list, $\langle 1, 2, 3, 5\rangle$ is a neighbor of $\langle 1, 2, 3, 4\rangle$; if the random walk transitions to this state, the current state becomes $\langle 1, 2, 3, 5\rangle$, which is shown in Figure 7.2b(i). In Figure 7.2b(ii), we show the updated neighbor lists considering the new state. Note that, here also we have 4 set of neighbors corresponding to 4 vertices of $\langle 1, 2, 3, 5\rangle$. The neighbor-list corresponding to vertex 5 is empty, as besides 1 and 3 (which are part of current state), 5 has no other adjacent vertices that can be used as a replacement vertex to build a new state. ■

Algorithm 6: SampleIndSubGraph Pseudocode.

```
Input :
        - Graph G_i
        - Size of subgraph, ℓ
[1] x ← State saved at G_i ;
[2] d_x ← Neighbor-count of x ;
[3] a_sup_x ← score of graph x ;
[4] while a neighbor state y is not found do
[5]     y ← a random neighbor of x;
[6]     d_y ← Possible neighbor of y ;
[7]     a_sup_y ← score of graph y ;
[8]     accp_val ← (d_x * a_sup_y)/(d_y * a_sup_x) ;
[9]     accp_probablility ← min(1, accp_val) ;
[10]    if uniform(0, 1) ≤ accp_probability then
[11]        return y ;
```

Proposal Distribution: As discussed in Section 7.5.2, for applying MH algorithm, we also need to decide on a proposal distribution, q. For FS^3's random walk the proposal distribution is uniform, i.e., in the proposal step FS^3 chooses one of g's neighbors uniformly. If a p-subgraph g has d_g neighbors, and h is one of them, using proposal distribution, the probability of choosing h from g is $q(g, h) = 1/d_g$. ■

In Figure 6 we show the MH subroutine that samples a ℓ-subgraph from a database graph G_i. In Line 1, it obtains the ℓ-subgraph, x (a state of the Markov chain) that was saved during the last sampling from G_i in one of the previous iterations. If the saved state is empty (happens only if it is the first graph sampled from G_i), it simply obtains one of the ℓ-subgraphs by growing from a random edge of G_i and returns it. In Line 2, it populates the neighbors of x and returns the neighbor-count. In Line 3, it computes the score of the graph x based on the chosen scoring function (s_1 or s_2). It then chooses y uniformly from all the neighbors of x, populates the neighbors of y and computes y's score (Line 5-7). Considering the chosen scoring function as the desired target distribution, it computes the acceptance probability of the transition from x to y using Equation 7.1. The while loop (Line 4-11) continues until a valid next state (a neighboring ℓ-subgraph) is found. It then returns the newly sampled subgraph y.

Example: Let's name the graphs (bold lines) in Figure 7.2a and 7.2b as g_1 and g_2 respectively. Neighbor count of g_1 is 27 and g_2 is 17 (total number of states in angular bracket). Say, average-edge-support of g_1 and g_2 in some given graph database are 4 and 10 (taken arbitrarily) respectively. Then the acceptance probability of a transition from g_1 to g_2 is: $\min\{1, \frac{27*10}{17*4}\} = 1$. ■

Proposition 2 FS^3*'s random walk is ergodic.*

Proof A Markov chain is ergodic if it converges to a stationary distribution. To obtain a stationary distribution the random walk needs to be finite, irreducible and

aperiodic. The state space consisting of all ℓ-subgraphs is finite for a given ℓ. We also assume that the input graph G is connected, so in this random walk any state y is reachable from any state x with a positive probability and vice versa, so the random walk is irreducible. Finally, the walk can be made aperiodic by allocating a self-loop probability at every node. Thus the proposition is proved. ■

Proposition 3 *The random walk of* FS^3 *achieves the target probability distribution, which is proportional to the chosen scoring function (s_i)*

Proof An ergodic random walk achieves the target probability distribution if it satisfies the reversibility condition i.e., for two neighboring states x and y, $\pi(x)T(x,y) = \pi(y)T(y,x)$, where π is the target distribution and $T(x,y)$ is the transition probability from x to y. For FS^3 the target distribution for a graph x, $\pi(x) = \frac{s_i(x)}{K}$, where K is a normalizing constant. Now, from Figure 6, it is easy to see that $\pi(x)T(x,y) = \frac{s_i(x)}{K \cdot d_x} \min\left\{1, \frac{d_x * s_i(y)}{d_y * s_i(x)}\right\} = \frac{1}{K} \min\left\{\frac{s_i(x)}{d_x}, \frac{s_i(y)}{d_y}\right\}$. Since the neighborhood relation is symmetric, there can be a transition from the state y to x and using that we have $\pi(y)T(y,x) = \frac{s_i(y)}{K \cdot d_y} \min\left\{1, \frac{d_y * s_i(x)}{d_x * s_i(y)}\right\} = \frac{1}{K} \min\left\{\frac{s_i(y)}{d_y}, \frac{s_i(x)}{d_x}\right\}$. So, $\pi(x)T(x,y) = \pi(y)T(y,x)$, which proves the proposition. ■

7.5.3 Queue Manager

FS^3 runs the ℓ-subgraph sampler for a large number of iterations so that in these iterations, the most frequent patterns have a chance to be sampled a number of times that is proportional to its support. Since, the number of possible ℓ-subgraphs in a database of large graphs can be very large, it may not be feasible to store all of them in the main memory. So FS^3 stores only a finite number of *best* graphs in a priority queue. The queue manager component of FS^3 implements the policy of this priority queue (PQ).

For a graph g stored in the PQ, the queue manager stores four pieces of information regarding this graph: (1) the canonical label [6] of g; (2) the *expected-support* value ($support_a(g)$) at that instance along with the support-list; (3) the score of g, i.e. $s_1(g)$ or $s_2(g)$ depending on which of the target distribution is used; and (4) the time (iteration counter is used as time variable) when the $support_a(g)$ was last incremented. The canonical label is used to uniquely identify a graph in PQ to overcome the fact that different sampling iterations may return different isomorphic forms of the same graph. The other pieces of information are used to implement the policy of the PQ.

Queue Eviction Strategy If the new sample is an existing graph in PQ, no eviction is necessary. We simply insert the id of the corresponding database graph (from where the sample was obtained) into the support-list of the graph and update the time variable. In case the id already presents in the support-list, nothing happens. On the other hand, if the new sample is a graph that does not present in PQ and PQ is full, we may choose to accommodate the new graph by evicting one of the graphs in the PQ, if certain conditions are satisfied.

To expedite the eviction decision, we maintain a total order in the PQ using a composite order criteria and the last graph in that total order is possibly evicted. The order uses three variables in lexicographical order: (1) expected-support (high to low); (2) score value, s_1 or s_2, depending on which one is used as the target distribution of the MCMC sampling (high to low); and (3) time (recent to old). Thus, the graph with the least expected support occupies the last position in PQ. However, if more than one graphs have the same value for the least expected-support, the tie situation is resolved by placing the graph with the smallest score value in the last position. Note that for FS^3's sampling, tie on expected-count is common as the search space is very large. If there is a tie for the score value also, it is resolved by considering the graph

[6]canonical label is a string represent of a graph which is unique over all isomorphisms of that graph; for our work we use min-dfs canonical code which is discussed in [58]

with the oldest update time. The intuition behind the above eviction mechanism is easy to understand; The pattern in the last position has small expected-support (first criterion), or small score, s_1 or s_2 (second criterion), or it is not being sampled from different graphs for a long time (third criterion), which makes it less likely to be frequent.

However, FS^3's queue manager does not simply evict the last element in PQ to insert the newly sampled graph (say, g), rather it first confirms whether g is a better replacement for the graph that would be evicted from the PQ. The decision is made by using the following heuristic. If the average of the scores (s_1 or s_2) of the graphs that are at the tail (lower half) of the PQ is smaller than $s_1(g)$ (or $s_2(g)$), then g is considered as a better replacement, and the last graph in the sorted order is evicted. If the above condition does not satisfy, graph g is simply ignored, and the sampling continues. The biggest advantage of this conditional eviction is that FS^3 does not generate the canonical code of g, if g is an unpromising pattern. Since, canonical code generation is much costlier than sampling, the time saved by avoiding the code generation can be spent for performing many other sampling iterations. For implementing the data structure of queue manager with the queue eviction policies, FS^3 uses multi-index map data structure [7], which sorts the graphs uniquely on the canonical label and non-uniquely on the various criteria that we describe above.

7.5.4 FS^3 Pseudocode

The entire pseudo-code of FS^3 is shown in Figure 7. In each iteration, it samples a ℓ-subgraph, h from a randomly selected database graph G by calling SampleIndSubGraph routine shown in Figure 6. In Line 6-7, it tests whether the score of h is better than the average score of the graphs in the bottom-half part of PQ. If this test fails it ignores h and proceeds with the next sampling iteration; otherwise, it generates the canonical code of h in Line 8 and use it as a key to search h is in the

[7]We used boost multi-index container (`http://www.boost.org/doc/libs/1_53_0/libs/multi_index/doc/index.html`) as our data structure

Algorithm 7: FS^3 Pseudocode.

```
Input :
        - Graph Database, 𝒢
        - Size of subgraph, ℓ
        - Number of samples, mIter
[1] iter ← 0, Q ← ∅;
[2] while iter ≤ mIter do
[3]     iter = iter + 1 ;
[4]     Select a graph G ∈ 𝒢 uniformly ;
[5]     h ← SampleIndSubgraph(G, p) ;
[6]     if Q.full = true and h.score() < Q.lowerHalfAvgScore() then
[7]         continue ;
[8]     h.code ← GenCanCode(h) ;
[9]     if h ∈ Q then
[10]        prevSupport = h.idset.size() ;
[11]        h.idset = h.idset ∪ G.id ;
[12]        if h.idset.size() > prevSupport then
[13]            h.insertTime = iter ;
[14]    else
[15]        if Q.full = true then
[16]            Q.evictLast() ;
[17]        h.idset = {G.id} ;
[18]        h.insertTime = iter ;
[19]        Q = Q ∪ {h} ;
[20]    return Q ;
```

PQ. If h is not in PQ FS^3 saves the graph h in the priority queue along with its support-list which contains only $G.id$. On the other hand, if h exists in the queue, FS^3 updates its support list, and also updates its insert-time variable. For each graph $G \in \mathcal{G}$, the sampling process saves the latest visiting graph (state), so that any later sampling from this graph starts from the saved state. In this way, FS^3 runs $|\mathcal{G}|$ copy of MCMC samplers, one for one of the input graphs in $\mathcal{G}$.

7.5.5 Computation Complexity and the Choice of Parameters

FS^3 has three parameters: (1) iteration count, (2) subgraph size and (3) queue size, which decides the runtime and memory complexity of the algorithm. In each iteration, there are three major steps: sampling, canonical code generation, and queue operation. Sampling populates the neighbor-lists, and its cost is linear with the size of neighbor-list of the current state, which is approximately equal to the subgraph size (ℓ) times the average degree of the graphs in the database. Canonical code generation is the most costly operation as this cost is the same as the cost of graph isomorphism. For general graph, this cost grows exponentially with the size of ℓ. However, for labeled graph this cost is polynomial with respect to ℓ, if the multiplicity of the labels in a graph is bounded by a much smaller constant than ℓ. We use min-dfs canonical code, which can be computed very efficiently [58]. The cost of queue operation grows logarithmically with the size of the queue.

Typically, the user chooses ℓ parameter by using her domain knowledge. Also, for applications where frequent subgraphs of different sizes are required, multiple runs FS^3 with different size value can be used. Iteration count should be chosen based on the size of the search space; the larger the search space, the higher number of iterations should be used so that the expected support values of sampled subgraphs are close approximation of their actual support values. However, in real-life guessing the size of the search space can be difficult, so we propose different methods for automatically finding a suitable iteration count; more details of this is provided in Section 7.7.2.

Finally, the queue size should also be chosen based on the size of the search space; for larger search space large queues are better. Since the queue management is very efficient, user can simply select a large queue considering available memory.

7.5.6 Theoretical Analysis of FS^3

FS^3 ranks the subgraph patterns based on the expected support ($support_a$). In this section, we analyze the expected value of $support_a$ for an ℓ-subgraph pattern g. To simplify the analysis, we will assume that in each sampling iteration (in Line 5 of Figure 7), FS^3 returns one of the ℓ-subgraphs of the chosen database graph uniformly. This assumption actually perform a worst-case analysis, because in general FS^3 performs a biased sampling in which the presumable frequent ℓ-subgraphs are sampled with higher probability.

Let, $\mathcal{G} = \{G_1, G_2, \ldots, G_n\}$ be a graph database with n graphs. Let's use x_j to denote the number of distinct ℓ-subgraphs in the graph G_j. Assume that the (induced) support of a subgraph pattern g in the database $\mathcal{G}$ is s, and the id of the graphs in which g occurs are $z_1, z_2, \cdots, z_s$.

If FS^3 makes t sampling iterations, on average t/n samples are obtained from the graph $G_{z_i:1\leq i\leq s}$. Under the uniform sampling assumption, the probability of sampling g from G_{z_i} in at least one of t/n iterations is equal to $1-(1-1/x_{z_i})^{t/n}$. Since the number of sampling iterations is typically very large, the above term is equal to $1-(1-\frac{t}{n\cdot x_{z_i}}) = \frac{t}{n\cdot x_{z_i}}$. So, the expected support of g, $E\big[support_a(g)\big] = \frac{t}{n} \times (1/x_{z_1} + 1/x_{z_2} + \cdots + 1/x_{z_s})$. If the number of samples are in the same order as the number of ℓ-subgraphs in the database graphs, the expected support converges to the actual support and the estimation is unbiased. Note that, even if the value of x_{z_i} are large (in the order of millions), FS^3 can sample millions of iterations in a few minutes, thus it can bring the $support_a$ value close to the actual support effectively. On the other hand, existing methods are not scalable as performing millions of SI test will take months, if not years.

However, FS^3 performs much better than a uniform sampler, as it actually performs a *support-biased sampling i.e. sampling is biased to sample more subgraphs which have greater* s_1 *or* s_2 *value.* In real-life dataset, the support of ℓ-subgraphs follows a heavy-tail distribution, in which a small number of truly frequent patterns have high support, but the majority of the ℓ-subgraphs have small support. Thus, the acceptance probability of sampling a frequent pattern g from the graph G_{z_i} is much higher than $\frac{t}{n \cdot x_{z_i}}$. In Section 7.6.6, we will compare between FS^3 and a modified version of FS^3 that uses the uniform sampling to show that FS^3's performance is substantially better.

7.6 Experiments

We implement FS^3 as a C++ program, and perform a set of experiments for evaluating its performance for mining frequent subgraphs of a given size. We run all the experiments in a computer with 2.60GHz processor and 4GB RAM running Linux operating system.

7.6.1 Datasets

We use three datasets for our experiments. The first is a protein structure dataset that we call PS. In this dataset, each graph represents the structure of a protein in the TIM (Triose Phosphate Isomerage) family. To construct a graph from a protein structure, we treat each amino acid residue as a vertex (labeled by letter code of the amino acids), and connect two vertices with an edge if the Euclidean distance between the C_α atom of the corresponding residues is at most 8Å. An edge also has a label of 1 or 2 based on whether the distance is below or above 4Å. Frequent subgraphs in such a dataset are common structure of the homologous proteins. The statistics of this dataset is available in Table 7.1; the same table also shows that existing graph mining methods are not able to mine subgraphs from this dataset. Our second dataset is a synthetic dataset (we call it Syn) that we build using the generator used in [184] with

parameter (ngraphs, size, nnodel, nedgel) =(0.1, 250, 20, 5). The subgraph space of this dataset is even larger than the PS dataset, and hence, it is more difficult to mine. Our last dataset is called Mutagenicity II (we will call it Mutagen dataset for abbreviation); it has been used in earlier works on graph mining [185]. Note that, it contains mostly chemical graph (avg. vertex count=14, avg. edge count=14), and existing graph mining methods can mine this dataset easily. We use this dataset only for comparing precision because ground truth of frequent subgraphs for this graph is easy to obtain.

7.6.2 Experiment Setup

FS^3 finds top-k frequent subgraphs with high probability. So, we measure the performance of FS^3 both from the execution time, and the quality of results for k=500 (unless specified otherwise). To obtain the quality, we use two metrics, that are pr@500 (precision at 500), and rank correlation metric, Tau-b. If $\mathcal{H}_a$ is the set of 500 most frequent subgraphs of a given size obtained by FS^3 and $\mathcal{H}$ is the corresponding true set of the same size based on actual support, the metric pr@500 is $\frac{|\mathcal{H} \cap \mathcal{H}_a| \times 100}{500}$; i.e, it finds the percentage of graphs in $\mathcal{H}$ that also present in $\mathcal{H}_a$. The higher the value of pr@500, the better the performance of FS^3. Note that, for a graph dataset that has one billion of subgraphs of a given size, sampling frequent graphs that belong to set $\mathcal{H}$ is not easy. A dumb sampler has a pr@500 value equal to 500 divided by one billion.

The metric, pr@500 only considers the presence or absence of a true positive (actually frequent) graph in $\mathcal{H}_a$, but it does not consider the order of graphs in $\mathcal{H}_a$ and the order of graphs in $\mathcal{H}$; in other words, it does not check whether actual support and expected support (as obtained by FS^3) have positive correlation or not. So, we also use Tau-b metric, which is the rank correlation between actual support and expected support of the objects in $\mathcal{H} \cap \mathcal{H}_a$. Tau-$b$ varies between -1 and 1. A value of 0 means no correlation, and the higher the value above 0, the better the

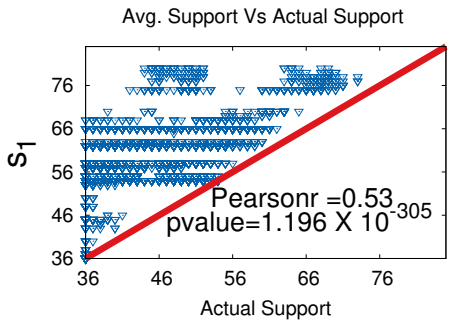
correlation. A strong correlation provides the evidence that FS^3 can indeed rank the patterns in the order of their actual support.

For computing pr@500 and Tau-b, we need to know the true set of top 500 frequent patterns of a given size. This is difficult to obtain for PS and Syn dataset, which we can not mine with the existing methods. To solve this problem, we have used GTrieScanner [69]; for an input graph GTrieScanner dumps all of its ℓ-subgraphs; by running this program for all the input graphs in a graph database, and grouping those by the canonical-code of those ℓ-subgraphs, we compute the actual support value of all the ℓ-subgraphs. Such exhaustive enumeration of actual support was only possible for the Mutagen dataset for all sizes, and for the PS and Syn dataset for size up to 8. For the later two datasets, for size larger than 8, the size of the dump of GTrieScanner exceeds more than 1 TB of physical space of a hard-disk, which is impossible for us to post-process. Also note that, GTrieScanner generates only the induced subgraphs, so for this comparison we run FS^3 for its induced subgraph sampling setup.

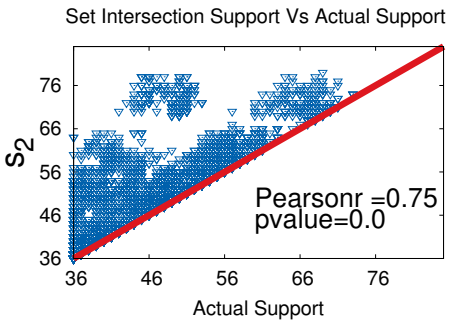
Performance of FS^3 depends on the number of iterations, scoring function used, size of the sampled patterns, and of-course the dataset. Also, choices of these values affect the running time of an iteration. So, when comparing among different sampling scenarios of FS^3 we plot the performance metric along the y-axis and the time along the x axis, and use a smooth curve to show the trend. Since, our method is randomized, all performance metric values are average of 10 distinct runs. We keep the priority queue size at 100K for all our experiments, (memory footprint around 200 MB) unless specified otherwise. Majority of our results are obtained by running experiments on the PS dataset.

7.6.3 Correlation between Actual Support and Scores

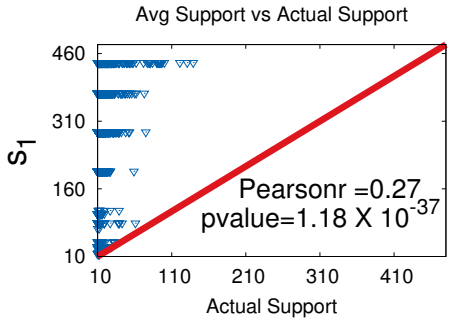
In this experiment, we use PS and Mutagen dataset and mine a collection of frequent patterns for a suitable size value using GTrieScanner. For each of the frequent patterns, we also compute their score value, s_1 and s_2, which we have used for con-



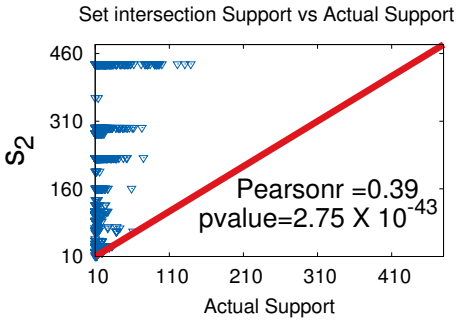
(a) Correlation with s_1 (PS)



(b) Correlation with s_2 (PS)



(c) Correlation with s_1 (Mutagen)



(d) Correlation with s_2 (Mutagen)

Fig. 7.3.: Correlation between support and score of a pattern.

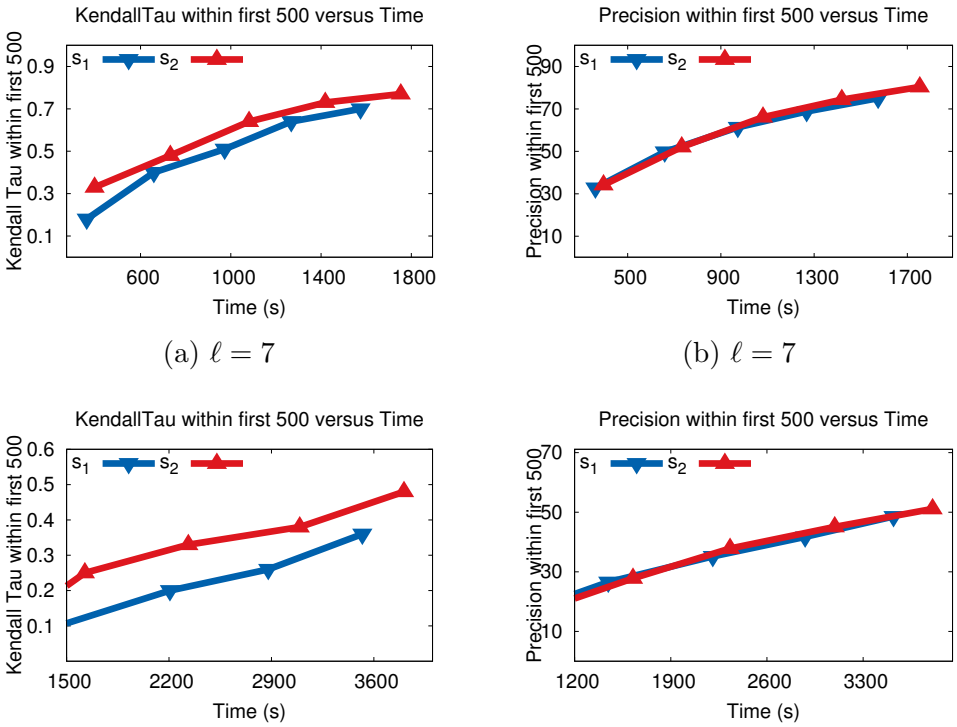
structing the target distribution of MCMC sampling. Our objective is to analyze how good our scoring functions are as a proxy of actual support of a graph.

Figure 7.3a and 7.3b show our finding for the set of frequent size-6 patterns in the PS dataset, and Figure 7.3c and Figure 7.3d show the same for the size-8 patterns in the Mutagen dataset. In these figures we show the scatter plot between actual support vs s_1 value (left plot) and s_2 value (right plot) of these patterns. As we can see the actual support is significantly (p-value is 0) correlated with both of the scoring functions, for both the datasets. For PS dataset, Pearson correlation value between actual support and s_1 and between actual support and s_2 are 0.53, and 0.75, respectively. For the patterns in Mutagen dataset, the values are 0.27 and 0.39, respectively. The correlation values are smaller for Mutagen dataset—significant (p-value is 0) nevertheless. These results are representative for frequent patterns of all different sizes for both the datasets. Such strong correlations enable the FS^3's MCMC walk to sample top-k frequent patterns effectively.

Another observation from this experiment is that correlation value is higher for the set-intersection support (the s_2 score), which makes s_2 a better choice over s_1. Also, both the score values are always an upper bound of the actual support value (no point below the diagonal line) as we have claimed in Lemma 1.

7.6.4 Performance of FS^3 for Different Sampling Setups

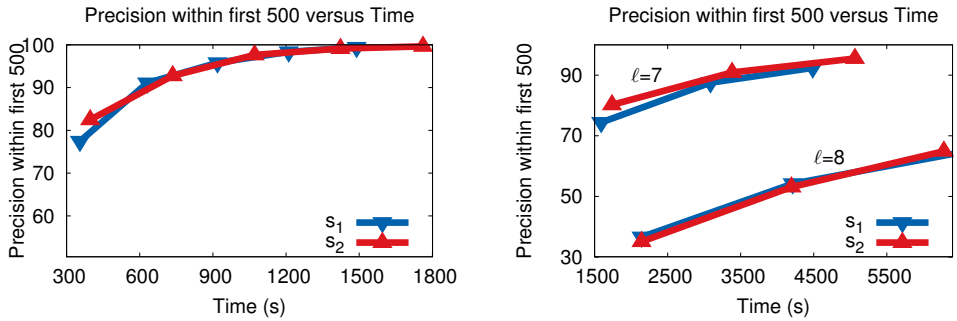
In this experiment, we compare the performance of FS^3 using the scoring function s_1 and s_2 on PS dataset for size 7 and 8 (the true set ($\mathcal{H}$) is known for these sizes). Figure 7.4 shows the results; in the left, we show the results (pr@500, and Tau-b vs time) for size 7, and in the right for the size 8. From the figure, we see that for both the scores, with increasing number of samples both pr@500, and Tau-b metrics increase almost linearly. Another observation from this figure is that the choice of score (s_1 or s_2) has small effect on the performance metric, specifically for pr@500.



(a) $\ell = 7$ (b) $\ell = 7$

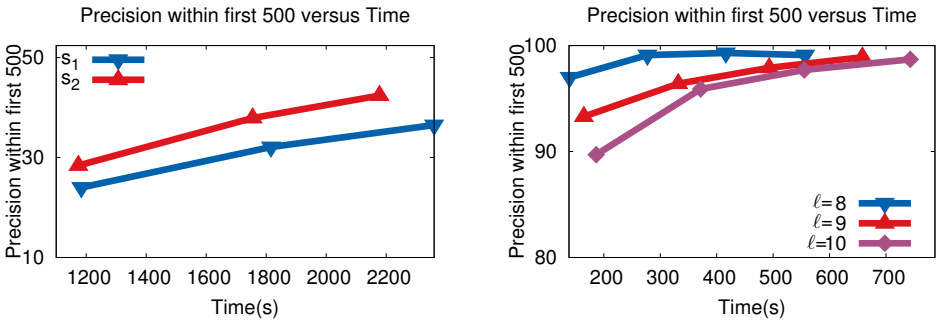
(c) $\ell = 8$ (d) $\ell = 8$

Fig. 7.4.: Kendall Tau, precision within first 500 for PS Dataset.



(a) $\ell = 6$ (b) $\ell = 7, 8$

Fig. 7.5.: Effect of increasing running time for FS^3 versus precision for PS dataset.



(a) Syn, $\ell = 6$

(b) Mutagen, $\ell = 8, 9, 10$

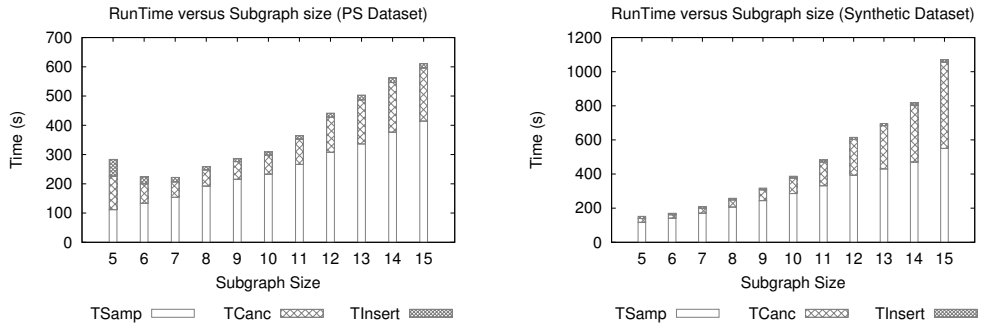
Fig. 7.6.: Precision for Synthetic and Mutagen dataset.

For Tau-b, score s_2 performs slightly better than the score s_1. This trend holds for other two datasets also.

Now, we comment on the values of pr@500 and Tau-b on these figures. From Figure 7.4(d), we see that for size 8, 1500 seconds of running of FS^3 yields pr@500 value of 28%, which increases to 50% for 3700 seconds, i.e., within an hour of sampling time, FS^3 finds 50% of the most frequent graphs from a sampling space of 0.95 billions graphs (See Table 7.1). Also note that the fastest graph mining algorithm, Gaston, could not mine this dataset in 16 hours of time, for 11% support and the max-size of 8. Also, within an hour or running, FS^3's Tau-b value reaches up to 0.42, which is a significant correlation. Now, for size 7, the performance is understandably better than the size 8 (see figure 7.4(a) and (b)), because its search space contains smaller number of subgraphs—157 millions as reported in Table 7.1.

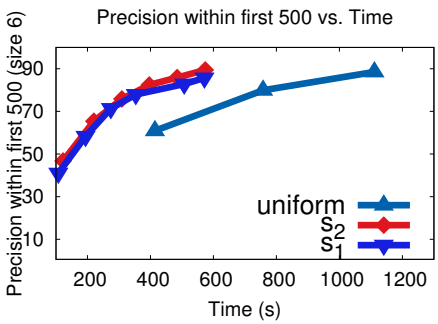
What happens if we run FS^3 for even more iterations? The performance keeps improving as we see in Figure 7.5. By running the sampler for 20 minutes for size 6, 1.4 hour for size 7, and 1.8 hour for size 8, we obtain 99%, 95% and 65% value for the pr@500. The linear trend of the curve for size 8 shows that by running for more time, the pr@500 can be improved even further.

We also run the above set of experiments for the other two datasets. In Figure 7.6a, we show the results for Syn dataset for size 6, for which we obtain pr@500 value of 42% in around 35 minutes. The performance on this dataset is poorer than the PS dataset, because search space in this dataset is much larger than the PS dataset. We cannot show results for higher size for this dataset as we could not generate the ground truth. In Figure 7.6b, we show the results for the Mutagen dataset, which has the smallest subgraph space, so for sizes 8, 9, and 10 this dataset achieves more than 90% pr@500 within 10 minutes.



(a) PS (b) Syn

Fig. 7.7.: Runtime performance of FS^3 for sampling subgraphs of different size.



(a) Effect of Target Distribution

Queue Size	Precision	Kendall Tau	Time (s)
0.5	**52.3**	**35.4**	**3997**
1.0	**54.2**	**42.4**	**4211**
2.0	**53.9**	**55.4**	**4645**

(b) Effect of Queue Size

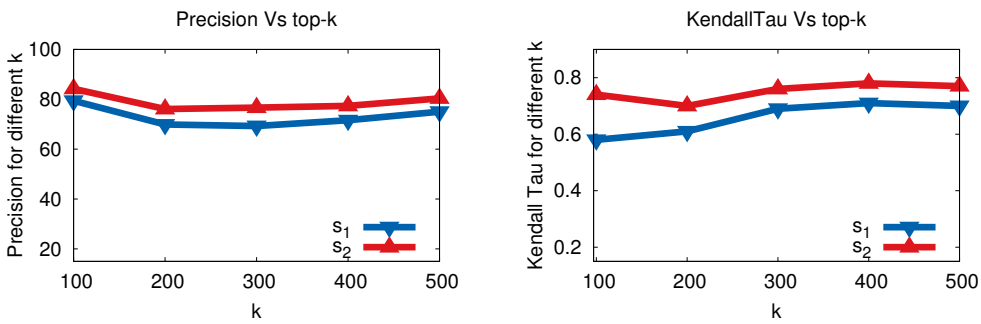
Fig. 7.8.: Effect of queue Size and target distribution.

7.6.5 FS^3's Scalability with the Size, ℓ

The execution time of FS^3 has three components: sampling time, canonical code generation time, and queue insertion time. In this experiment, we check how these times vary as we vary the desired size of the subgraphs to be sampled (ℓ value). For this, we use PS and Syn dataset, and use s_2 scoring function. Figure 7.7 shows the results. As we see in the plots, the execution time increases almost linearly with the value of ℓ for both the datasets. Also, FS^3 spends the majority of its execution time for sampling as it does not generate canonical code in many of its iterations. Queue insertion time is negligible compared to sampling and canonical code generation time.

7.6.6 Impact of Target Distribution and Queue Size

FS^3's MCMC sampling uses s_1 or s_2 score to construct its target distribution. In this experiment, we validate the impact of these choices by comparing their performance with a case, where the target distribution is uniform, i.e., each of the ℓ-subgraphs of a database graph G_i has equal likelihood to be visited, that is the score of any ℓ-subgraph is 1, a constant (let's call it uniform-FS^3). For comparison, we use the pr@500 metric. Figure 7.8a shows the result for PS dataset for size 6. It is clear from this figure that by adopting $s_1(g)$ or $s_2(g)$ as the target distribution, we achieve higher pr@500 at a faster rate. For example, within 7 minutes of sampling,



(a) Precision (b) Kendall Tau

Fig. 7.9.: Performance of FS^3 for different k.

the pr@500 score of uniform-FS^3 is around 55%; on the other hand, for the same time, the pr@500 score is around 85% for both $s_1(G)$ and $s_2(G)$.

For all our experiments we kept the priority queue size fixed to 100K. If we increase the queue size, the memory footprint of the algorithm will increase, but the method will be more accurate, as it will be able to store a large number of potential frequent graphs that may turn out to be frequent at a later time. The improvement is more prominent for the Tau-b metric than the pr@500 metric as shown in Figure 7.8b for PS dataset and subgraph size 8.

7.6.7 Impact of k on FS^3

We also study the performance of FS^3 for different choices of k value, for mining Top-k frequent patterns. For this experiment, we use PS dataset, ℓ=7. Figure 7.9 shows the corresponding result. In Figure 7.9a, we plot the Pr@k values and in Figure 7.9b, we plot the Kendall Tau values for different k's between 100 and 500. We calculate both the statistics by taking the average of 10 independent runs. As we can see, for the entire range of k values, the performance remains almost constant.

7.7 Mixing Rate of Random Walk

One important aspect of any MCMC algorithm (including MH, which is essentially a special kind of MCMC algorithm) is the rate at which the initial distribution converges to the desired distribution. The mixing rate of a random walk has been studied extensively in spectral graph theory [186], since it plays an important role in obtaining efficient MCMC algorithms. A Markov chain is called *rapidly mixing* if it is close to stationary after only a polynomial number of simulation steps, i. e., after $poly(\lg m)$, where m is the number of states in the Markov chain. Note that, m can be exponentially large with respect to the input size of the algorithm. An algorithm that is *rapidly mixing* is considered efficient.

A method to measure the mixing rate is to find the *spectral gap* of the transition probability matrix T. T has m real eigenvalues $1 = \lambda_0 > \lambda_1 \geq \lambda_2 \geq ... \geq \lambda_{m-1} \geq -1$. Then, the *spectral gap* is defined as $\lambda = 1 - max\{\lambda_1, |\lambda_{m-1}|\}$. Since the absolute values of all the eigenvalues are less than one with the largest eigenvalue λ_0 be exactly one, the spectral gap is always between 0 and 1. The higher the spectral gap, the faster the convergence [187]. In [188] it has been shown that the inverse of spectral gap of a reversible Markov chain captures the mixing time of that walk. We compute the spectral gap of the random walk for the case of size-6 subgraphs in the graphs in the Mutagen Dataset; average spectral gap for random walk over different database graphs is 0.08 for the score s_1 and 0.065 for the score s_2, which means that the mixing time is approximately 12 unit and 15 unit, respectively. This mixing time is very good given the size of the search space. We do not provide such research for PS dataset as the state space for those graphs is in the order of several millions, and finding spectral gap of such a large transition matrix is almost impossible unless special hardware is used.

Table 7.2.: Probability of acceptance of FS^3 for Mutagen and PS Dataset.

	Mutagen			PS		
	ℓ= 8	ℓ=9	ℓ=10	ℓ=6	ℓ=7	ℓ=8
Acceptance (%), Strategy $=s_1$	82.70 ± 0.04	83.89 ± 0.03	81.66 ± 0.03	91.08 ± 0.01	92.23 ± 0.02	93.08 ± 0.01
Acceptance (%), Strategy $=s_2$	75.27 ± 0.05	76.74 ± 0.03	75.20 ± 0.03	85.08 ± 0.05	87.46 ± 0.06	89.41 ± 0.07

7.7.1 Percentage of Acceptance

It is well-known that a large number of rejected moves in a Metropolis-Hastings algorithm's execution slows down the mixing of Markov chain; it also indicates a poorly designed proposal distribution [189]. A good proposal distribution should have a high likelihood of acceptance. As we noted in Section 7.5.2, the proposal distribution of FS^{3}' is uniform. In this experiment we will empirically validate whether this is a good choice by observing the acceptance rate of the MCMC random walk over a large number of state transitions. For this, we run FS^{3}' for $1M$ (one million) iterations and record the percentage of accepted transitions and average that over 10 independent runs. We show the result in Table 7.2. As we can see, for Mutagen dataset and for average-support target distribution (s_1), the percentage of acceptance are 82.70, 83.89 and 81.66 for subgraph size-value 8, 9 and 10, respectively; for the intersection set based target distribution (s_2), the values are 75.27, 76.74, and 75.20, respectively. We also show the standard deviation of the acceptance percentage over 10 different runs for each cases, which is very small (less than .05). It indicates that the acceptance probability is consistently high. For the PS dataset, the acceptance probability values are even better—more than 0.90 for strategy s_1 and more than 0.85 for strategy s_2 over different subgraph sizes. The results show that the choice of uniform proposal distribution is a good choice as a proposal distribution. Besides, it is a good-fit for both the target distribution, s_1 and s_2. However, it is a slightly better fit for the distribution s_1 than the distribution s_2.

7.7.2 Choosing Iteration Counts

We have shown in Section 7.6.4 that the performance of FS^{3} improves with the number of sampling iterations (see Figure 7.4). But, how do we know how many iterations would yield a representative set of frequent patterns? A simple heuristics approach for selecting an iteration count is to stop sampling after the PQ becomes stable. For this, we track the number of disruptions in the top-k over a given number

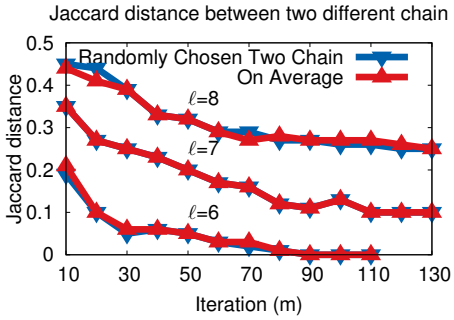


Fig. 7.10.: Jaccard distance vs iteration count.

of sampling iterations; if that value falls below a threshold, we assume that the queue is stable, and return the top-k patterns as the frequent patterns. This approach works for all practical purposes; however, for FS^3, we design a more sophisticated stopping criteria using Gelman-Rubin Diagnostics [190]. We discuss that in the following paragraph.

Instead of running single chain, Gelman and Rubin proposed to run more than one chains simultaneously. If the empirical distribution of the sampling sequence of each of these chains are *similar* to the empirical distribution of the sequence composed of all the chains, they declare convergence. For FS^3, the main focus is the frequent patterns in the top-k positions of the queue. So we claim convergence, if the top-k patterns from multiple chains are similar to each other. To achieve this, we run j (j can be as small as 2) independent copy of FS^3 (each with its own sampler and priority queue) in a multi-threaded implementation, and calculate the average of the $\binom{j}{2}$ pairwise Jaccard distances calculated from the j copies of the top-k patterns obtained from these chains. If the Jaccard distance value converges, we stop sampling and return the best k frequent patterns from the j queues. Given that most of today's processors have many cores, the practical overhead of such as implementation is only the additional memory for the j copies of the PQ.

Figure 7.10 shows the relation between Jaccard distance and iteration count for $j = 10$. In this figure, we show two graphs, one for the average Jaccard distance over all the chains, and the other is the Jaccard distance for a pair of randomly chosen chains. As we can see, for increasingly larger iteration count, the Jaccard distance among the top-k patterns from different chains diminishes and for sufficiently large value, it converges to a small value. For $\ell = 6$ the value reaches almost zero, whereas for $p = 7$ it stabilized around 0.1 and for $\ell = 8$, the value is 0.25 within the iteration count shown in the x-axis. This above result is a testimony of FS^3's effectiveness. Two independent randomized processes obtain an almost identical set of frequent patterns—which is a proof that the frequent patterns that are returned by different runs of FS^3 are truly frequent. Further, even though FS^3 is a randomized method,

the result is reproducible as different runs of the method will return almost identical set of top-k patterns.

7.8 Chapter Summary

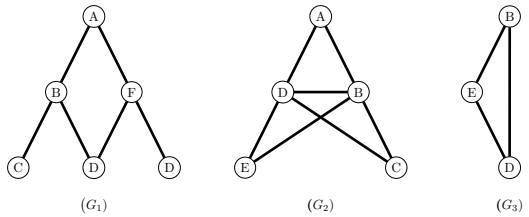
In this paper, we present FS^3, a sampling based method for finding frequent induced subgraph of a given size. For large input graphs, existing algorithms for frequent subgraph mining are completely infeasible; whereas FS^3 can return a small set of probabilistically frequent patterns of desired size in a short amount of time. Our experiments on two real life and one synthetic dataset show that the expected support of the graphs that FS^3 samples has excellent rank correlation with their actual support.

8. DISCOVERY OF FUNCTIONAL MOTIFS FROM THE INTERFACE REGION OF OLIGOMERIC PROTEINS USING FREQUENT SUBGRAPH MINING

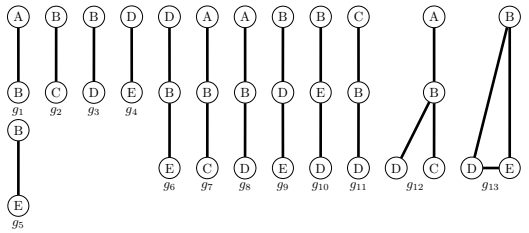
8.1 Introduction

Structural dynamics and functions of many proteins are primarily controlled by the interaction of residues at the interface region. Because of this, studying and analyzing the interface region of a protein is crucial for understanding the underlying protein machinery [82]. In existing literatures, many research works have provided a detailed analysis of the interface region of various proteins. However, in the majority of these works protein interface region is represented through different spatial features; examples include interface area, interface polar residue abundance, hydrogen bonds, solvation free energy gain from interface formation, and binding energy [191]. Such a feature-based representation—although useful for ranking of predicted docked conformation of protein-protein complexes or for building scoring function for docking [192–194]—is not much useful for understanding protein machinery. This is due to the fact that a feature-based representation of interface region works like a black-box without providing much information regarding the functionalities of the protein. So, alternative representations of interface regions are needed for providing a better understanding of functional motifs, which are responsible for carrying out protein's intended functionalities.

Sequence motifs often correspond to the functional regions of a protein, such as, catalytic sites, binding sites, structural motifs, etc. and they are considered to be the building blocks of protein sequences [195–197]. These motifs are conserved across different proteins and possess highly discriminative features for predicting the functions of a protein [198]. However, sequence motifs are limited in their representation



(a) Graph Database



(b) Frequent Subgraphs

Fig. 8.1.: (a) A graph database with 3 graphs (b) All the frequent subgraphs of the graph database in (a) using a minimum support value of 2. If we want to obtain only the induced frequent subgraphs, g_1-g_5, g_7, and g_{13} are frequent for a minimum support of 2.

ability, so in recent years, networks are being used for representing biological data. Besides, network theories are also being used to gain insights into complex biological problems [85, 95, 98, 199]. The concept of *network motif* has also emerged, which has been hypothesized to play an important role in carrying out the key functionalities that are performed by the entities in a biological network [21, 72, 200, 201]. A very recent study [202] showed that the distribution of network motifs influences the organization of metabolic networks. However, the methodologies for network motif discovery [21, 72] yield sub-networks that are frequent in a given network, and hence they are not useful for finding conserved sub-networks at the interface of a set of proteins.

Mining frequent sub-networks (FSM) is an important and well studied task in data mining field; it is defined as finding all subgraphs that appear frequently in a graph dataset given a minimum frequency threshold. There are two variants of this problem—in the first variant [55–60, 62], the dataset has a collection of many graphs, and in the second variant [203–205], the dataset contains a single large graph. For the

latter variant, the frequency of a subgraph is counted as its multiplicity in the large graph. On the other hand, the earlier variant of graph mining counts the frequency of a subgraph over the collection of graphs in the dataset. Thus, for this variant of graph mining, the overall frequency of a subgraph pattern is the number of distinct graphs in which the pattern appears. In this work, we represent the interface region of oligomeric proteins as a set of networks and then use a novel frequent sub-network mining algorithm for finding functional motifs in the interface region. As we discover patterns that span over a set of networks, the algorithms belonging to the first variant are relevant for our task and forthcoming references of frequent graph mining in this paper pertain to the first variant of FSM.

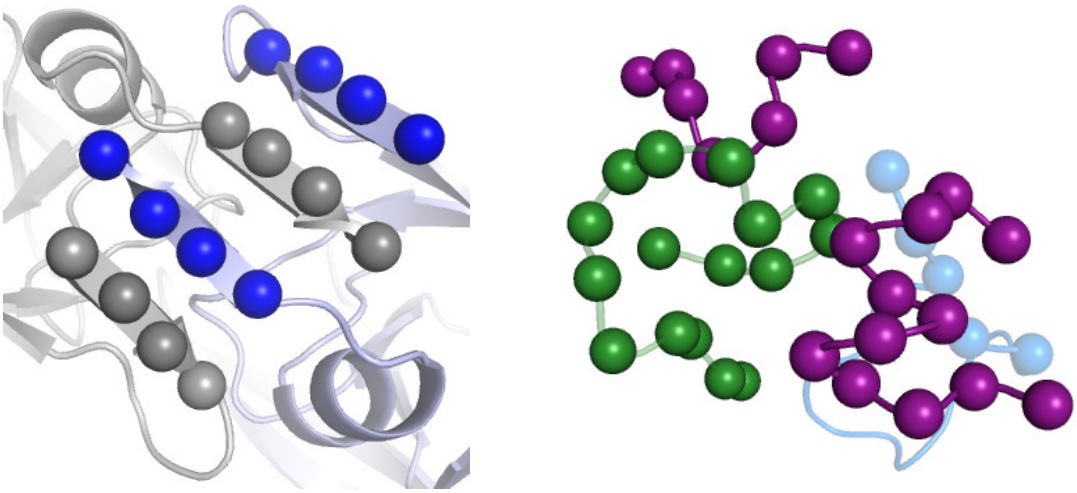
Mining sub-networks from a set of networks is defined as follows: Given a graph dataset $\mathcal{G}$, and a minimum support $\pi^{\mathbf{min}}$, obtain the set of subgraphs whose frequency is higher than $\pi^{\mathbf{min}}$. The set of frequent subgraphs are generally represented by $\mathcal{F}$. In Figure 8.1a, we show a graph dataset with 3 graphs and in Figure 8.1b we show the frequent subgraphs of this dataset considering $\pi^{\mathbf{min}} = 2$. Over the years, a good number of algorithms for frequent sub-network mining (FSM) have been proposed, examples include Subdue [55], AGM [56], FSG [57], gSpan [58], FFSM [206], DMTL [59], and Gaston [60]. Distributed solutions of FSM [205, 207] which runs on map-reduce platform have also been proposed.

Existing FSM algorithms are proven to be effective for finding frequent subgraphs from input graphs which are small and sparse. However, for general graphs, FSM task is not scalable due to the inherent complexity of this task. In fact, Horváth *et al.* have shown that FSM cannot be solved in output polynomial time [171]. The lack of scalability of FSM task has also been shown empirically. For instance, FSM has been applied on a small dataset (only 3 graphs) of protein-protein interaction (PPI) graphs, each graph having 2154 nodes on average; but the most efficient of the existing FSM algorithms cannot mine all the frequent subgraphs from this dataset in days of running even with 100% support value [64]. Distributed solution, such as [207] can successfully overcomes the lack of scalability issues arising from the large number

of graphs in the dataset, but they still remains not scalable when the graphs in the dataset are dense and large. Our investigation finds that any reasonable construction of interface networks on real-life protein data yields large and dense graphs for which existing methods simply fail to find interface patterns in an effective manner.

Existing FSM methods suffer from some other serious limitations when they are used for mining interface patterns. First, existing subgraph mining methods require that the user selects a minimum support threshold value [58–60]. However, when the main objective of subgraph mining is to discover functional motifs from a number of protein conformations, this support value is generally unknown. This is due to the fact that the spatial orientation of the residues in a functional motif across the conformations fluctuates owing to the dynamics of the motif, and a part of the motif may be occluded in some subgraphs, making the motif infrequent. So, choosing a large support threshold may miss a significant part of a functional motifs; on the other hand, choosing a small support threshold may return too many random subgraphs that are frequent simply by chance. The second limitation is that existing algorithms [58–60] enumerate all the frequent subgraphs starting from size-1 and thus they return a large number of unnecessary patterns. But, for functional motifs, the subgraph size of interest is known in many cases; if not known, a reasonable initial guess of the motif size can be made from the knowledge of protein's family and functionalities. So, a novel frequent subgraph mining method is needed which is scalable, not dependent on the minimum support threshold, and able to return frequent subgraphs of a user-specified size.

In this work, we propose a graph mining framework which is particularly suited for the discovery of functional motifs from the interface graphs of a large collection of protein structures. Our proposed approach uses spatial proximity for creating the interfacial network dataset, so, the proteins in a dataset need to have high structural similarity (low structural diversity). For instance, these structures could either be structural conformations of the same protein (see Sections 8.5.1 and 8.5.2) or they could represent multiple proteins from the same functional group (see Sections 8.5.3).



(a) HIV (b) TIM

Fig. 8.2.: (a) Retrieved frequent patterns representing the dimerization lock at the base of HIV- 1 protease structure and (b) along the dimeric interface of triosephosphate isomerase.

The proposed method first creates a dataset of interface graphs, each representing a structure from the database. It then uses a novel sampling based method for mining subgraphs of a given size which are frequent over the graph database with a high probability. In the proposed method, subgraph size is user-defined, which can be chosen from user's domain knowledge of the protein under investigation.

To validate the effectiveness of our method we perform three independent experiments. In the first two experiments, we use two different datasets of protein conformations: (1) HIV-1 protease (329 conformations) and (2) Triosephosphate isomerase (TIM) (86 conformations) and find frequent subgraphs of appropriate size from the given conformations of these proteins. The subgraphs that we mine from the interface networks enable us to discover the functional motifs in the above pair of proteins. The first protein, HIV-1 protease is essential for the life cycle of human immunodeficiency virus (HIV) which causes acquired immunodeficiency syndrome (AIDS) in humans. The second protein, TIM is the fifth enzyme in the glycolysis pathway that produces energy in all living organisms. For both proteins, the large number of structures represent a sample of different conformational states of the proteins that are solved

experimentally and they can explain the functional dynamics and functional motifs of the protein [208]. The 10 most frequent subgraphs mined from the HIV-1 protease using our proposed method collectively capture a 16-residue functional motif, named dimerization lock (shown in Figure: 8.2a) that exists in the interface of the protein. Among these frequent subgraphs, our method retrieves 15 out of 16 residues in 6 subgraphs, 14 residues in 2 subgraphs and 13 residues in the remaining 2 subgraphs. Similarly, frequent subgraphs from TIM retrieve dimerization lock that exists in TIM conformations (shown in Figure: 8.2b).

In the third experiment, we use the Dobson and Doig (D&D) benchmark dataset for enzymes (691 enzymes out of 1178 protein structures) [209]. As enzymes are known to be macromolecular catalysts, discovering functional motifs at the interface region of these proteins is paramount to understanding how they bind and interact with other macromolecules to perform their functions. The subset of enzymes in D&D is composed of groups of proteins from the six top-level classes of enzymes namely: Oxydoreductase, Transferase, Hydrolase, Lyase, Isomerase and Ligase. We use our approach for mining function specific motifs for each of these classes of enzymes. Specifically, for each class, We mine up to 200 most frequent patterns within a size range of 5, 6, 7 and 8 nodes per pattern. By checking the overlap between the set of patterns mined from each class, we show that our approach discovers function specific patterns from each functional class of enzymes. We also show that these patterns include catalytic sites of enzymes that have been identified in the literature.

We claim the following contribution in this paper:

- We propose a method to map the interfacial region of a protein as a network for the discovery of functional motifs by using a sampling based frequent subgraph (FSM) mining method.

- We validate the utility of the proposed FSM method by capturing the locking mechanism at the dimeric interface from different conformations of HIV and TIM protein structures.

- We also observe that our sampling based FSM method enables us to capture function specific patterns at the interface region of 3D structures of proteins belonging to the same functional group.

8.2 Background

Let $G(V, E)$ be an *interfacial network*, where V is the set of vertices and E is the set of edges. For our problem, the vertices are set of residues and the edges are connection among the residues based on their pair-wise physical proximity. Specifically, if the inter- and intra-chain distance between a pair of residue is smaller than a user defined distance threshold, an edge is added between the corresponding pair of vertices.

By construction, the interfacial networks are simple graph which do not have self-loops or multi-edges. Besides, these graphs are undirected, because the Euclidean distance is a symmetric metric. Finally, for all reasonable choices of inter and intra chain distance threshold, these graphs are connected.

A *labeled graph* $G(V, E, L, \Psi)$ is a graph[1] for which the vertices and the edges have labels that are assigned by a labeling function, $\Psi : V \cup E \rightarrow L$ where L is a set of labels. In our case, only vertices have labels, which is a value between 1 to 20, corresponding to the 20 amino acid residues of proteins.

A graph $G' = (V', E')$ is a *s*ubgraph of G (denoted as $G' \subseteq G$) if $V' \subseteq V$ and $E' \subseteq E$. A graph $G' = (V', E')$ is a *v*ertex-induced subgraph of G if G' is a subgraph of G, and for any pair of vertices $v_a, v_b \in V'$, $(v_a, v_b) \in E'$ if and only if $(v_a, v_b) \in E$. In other words, a *vertex-induced* subgraph of G is a graph G' consisting of a subset of G's vertices together with all the edges of G whose both endpoints are in this subset. In this paper, we have used the word *subgraph* for abbreviating vertex-induced subgraph. If G' is a (induced or non-induced) subgraph of G and $|V'| = \ell$, we call G' a ℓ-subgraph of G.

[1]We have used the terms graph and network interchangeably.

Let $\mathcal{G} = \{G_1, G_2, \ldots, G_n\}$ be an interfacial network database, where each $G_i \in \mathcal{G}, \forall i = \{1 \ldots n\}$ represents a labeled, undirected and connected graph. The *support-set* of the graph g is $\mathbf{t}(g)$, and $\mathbf{t}(g) = \{G_i : g \subseteq G_i \in \mathcal{G}\}, \forall i = \{1 \ldots n\}$.

This set contains all the graphs in $\mathcal{G}$ that have a subgraph isomorphic to g. The cardinality of the *support-set* is called the *support* of g. g is called frequent if *support* $\geq \pi^{\mathbf{min}}$, where $\pi^{\mathbf{min}}$ is predefined/user-specified *minimum support (minsup)* threshold. Given the graph database $\mathcal{G}$, and minimum support $\pi^{\mathbf{min}}$, the task of a frequent subgraph mining (FSM) algorithm is to obtain the set of frequent subgraphs (represented by $\mathcal{F}$). While computing support, if an FSM algorithm enforces induced subgraph isomorphism, it obtains the set of frequent induced subgraphs (represented by $\mathcal{F}_I$). It is easy to argue that $\mathcal{F}_I \subseteq \mathcal{F}$.

8.3 Related Work

There are several works that represent a protein structure as a network consisting of a set of nodes and the relationship between the nodes. However, the way different works model the network differs. Across these works, the nodes can represent amino acid residues [80–85], functional atoms from the side chains [86, 87], secondary structure elements [88–90], proteins [91, 92], protein complexes [93], and interaction pseudoatoms [94]. Edges also has different connotations in different works. For instances, edges connect nodes if they interact with each other [80, 81], or if they are nearer to each other spatially [82, 87], or if they are within the interacting distance of each other [86]. Some works create edges between two nodes if the nodes are part of a functional unit in a pathway or in a biological process [91, 92], or if side-chains interact with each other [95]. Our work differs in the method of construction and analysis of these networks from previous studies. In our work, we use C_α carbon (backbone carbon) of a particular residue as a node. So, the C_α carbons from all the residues of a particular protein represent the set of nodes and we connect two nodes if their C_α carbons are spatially nearer to each other. Existing works use a graph

to capture the entire protein structure, but in this work we capture dense interfacial region between different subunits of the same structure.

In existing works, network representation of proteins has been used for various purposes; for example, to study the evolution of protein-protein interactions [82], to summarize how central network elements are enriched in active centers and ligand binding sites directing the dynamics of entire protein [87], to classify protein 3D-structures [84, 85], to characterize the topological role of residues [83], to offer a comprehensible view of critical residues and to facilitate the inspection of their organization [96], to detect cancer-associated functional residues [91], to uncover distinct cancer-specific functional modules [92], to document functional components and sub-components of proteins [97], and to compare two networks (Oligomeric vs Monomeric) [81] for getting insight into the protein association. Greene *et al.* [98] authored a good review article which surveys several key advances in the expanding area of protein structure and folding research using network approaches. To the best of our knowledge we are the first to develop graph mining methodologies for mining interfacial networks to discover important functional units (such as, lock structure in HIV 8.2a and hugging point 8.2b in TIM structure), or to find family specific active sites from enzymes.

8.4 Methods

In Figure 8.3, we provide a pictorial depiction of the proposed method. Given a set of structures of a protein, we first convert each structure into an interfacial network, which is our collection of graphs in the graph dataset. Then we use our designed frequent pattern mining method for mining a set of fixed-size (user defined) subgraphs, which are the most frequent (probabilistically) over the graphs in the graph database. For each of the mined frequent subgraphs, we find their structural embedding in the host graphs, and identify those patterns for which the nodes in a pattern consistently

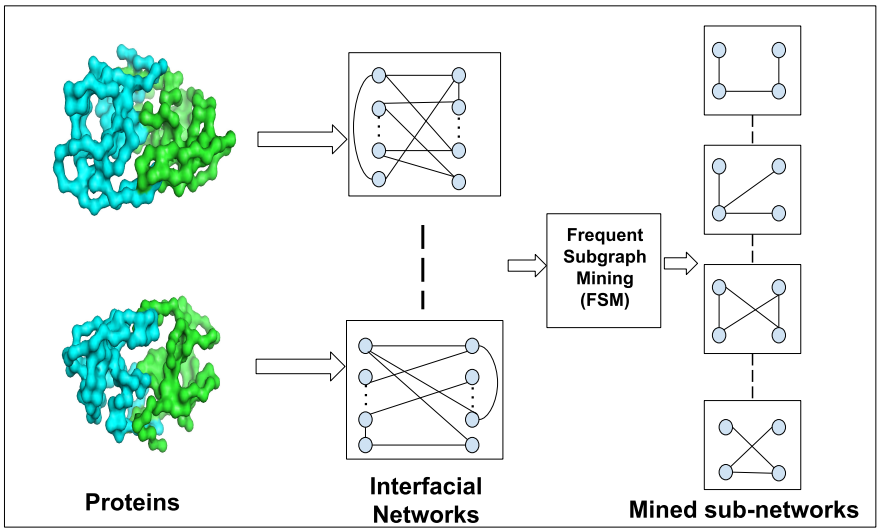


Fig. 8.3.: A pictorial depiction of the proposed method. Given a set of structures of a protein, we first convert each structure into an interfacial network. Then we use a frequent pattern mining method for mining a set of fixed-size (user defined) subgraphs. Finally, for each of the mined frequent subgraphs, we find their structural embedding in the host graphs.

map to a fixed set of residues in all the conformations. We consider these structural patterns as possible candidates of being a functional motif, and study whether these residues correspond to any known oligomerization mechanism. In this work, we use these set of steps to study the dimerization interfaces of HIV-1 and TIM proteins, and also to discover family-specific active sites of various enzyme families. Below, we discuss each of the steps of our method in details.

8.4.1 Modeling Protein as Interfacial Network

For each structure, we first retrieve the C_α carbons along with their 3D co-ordinates from the residues of a pair of chains U_i and U_j. We then construct an interfacial network of the structure by connecting the subset of C_α residues that are in the interface region of either of the chains. We consider a residue (say, v_a) in a chain (U_i) to be at the interface region if it is within a maximum spatial distance (γ) of any C_α residue (say, v_b) in the other chain (U_j), with respect to a distance measure

(Δ) that is the Euclidean distance in our case. The interface C_α carbons are the set of nodes in our interface network. Within each chain, we connect pairs of residues if they are within a spatial proximity of at most δ. We label the nodes from 1 to 20 based on the amino acid types of the corresponding residues. Then, we form edges between nodes (residues) of different chains if they are spatially close to each other. After this step, we obtain an undirected vertex-labeled graphs— corresponding to interfacial network of the input protein structure. Equation (8.1) formally describes the graph modeling process. Note that for interfacial networks, the intra-chain distance threshold (δ) should be made low while the inter-chain distance threshold (γ) should be kept high. This will make the graph model emphasize the interfacial region at the surface between the different chains of the structure (at the 3D level) while making the intra-chain network very sparse to approximately contain at most the connections between amino acids at the primary structure level.

$$e(v_a, v_b) = \begin{cases} 1, \ if \ \Delta(v_a, v_b) \leq \delta \mid v_a \in U_i, v_b \in U_j, i = j \\ 1, \ if \ \Delta(v_a, v_b) \leq \gamma \mid v_a \in U_i, v_b \in U_j, i \neq j \\ 0, \ otherwise \end{cases} \tag{8.1}$$

It is important to note that having a larger distance threshold (values of δ, and γ) makes the interfacial networks denser and thus makes it more likely to find frequent subgraph patterns across different structures. However, the patterns that are discovered using a large threshold are less precise because the edges of these patterns cover a larger range of distances between a pair of residues. On the other hand, if we consider smaller distance threshold we get more precise patterns, but the mining process is less likely to find a frequent pattern. This is similar to precision-recall trade-off in information retrieval. For larger values of δ and γ, the recall increases but precision deteriorates, and for smaller values, the precision improves with a loss of recall.

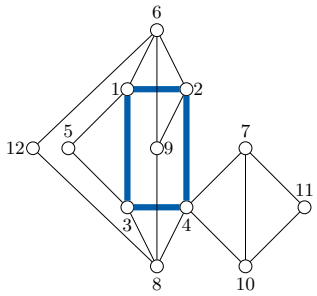
8.4.2 Frequent Subgraph Mining with FS^3

For mining a fixed size frequent subgraph we use a sampling based graph mining algorithm, called FS^3, which we have proposed in one of our recent works [182]. FS^3 is based on sampling of subgraphs of a fixed size [2]. Given a graph dataset $\mathcal{G}$, and a size value ℓ, FS^3 samples subgraphs of size-ℓ from $\mathcal{G}$. The distribution from which the size-ℓ subgraphs is sampled is biased such that the sampling process over-samples the graphs that are likely to be frequent over the graphs in $\mathcal{G}$. FS^3 runs the above sampling process for many times, and uses an innovative priority queue to hold a small set of most frequent subgraphs, which it returns at the end of the sampling process. The unique feature of FS^3 is that unlike earlier works which are based on sampling [65], FS^3 does not perform any subgraph isomorphism (SI) test, so it is scalable to large graphs. By choosing different values of ℓ, user can find a succinct set of frequent subgraphs of different sizes. Also, as the number of samples increases, FS^3's output progressively converges to the top-k most frequent subgraphs of size ℓ. So user can run the sampler as long as he wants to obtain more precise results.

A detail discussion of FS^3 algorithm is out of scope for this paper. However, to make this paper self-sufficient, We describe below some key concepts of FS^3 algorithm. Interested readers are encouraged to read the original FS^3 paper [182] for more details.

Subgraph sampling by FS^3 Algorithm: At each sampling iteration, FS^3 performs a 2-stage sampling process. In the first stage, FS^3 chooses one of the graphs in $\mathcal{G}$ (say, G_i) uniformly, and in the second stage it samples a size-ℓ subgraph of G_i and returns. For the second stage, FS^3 performs a Markov chain Monte Carlo (MCMC) sampling over the ℓ-subgraphs of G_i. The main idea of MCMC sampling is to perform a random walk over the sampling space and subsequently return the sample the walk visits. The transitional probability of the random walk is chosen

[2]The name FS^3 should be read as *F-S-Cube*, which is a compressed representation of the 4-gram composed of the bold letters in **F**ixed **S**ize **S**ubgraph **S**ampler.



(i)

1: $\langle 1,2,3,5\rangle, \langle 1,2,4,5\rangle, \langle 1,3,4,5\rangle, \langle 1,2,3,6\rangle, \langle 1,2,4,6\rangle, \langle 1,3,4,6\rangle$

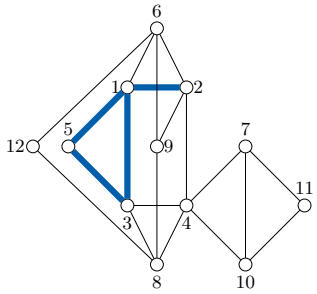
2: $\langle 1,2,3,6\rangle, \langle 1,2,4,6\rangle, \langle 2,3,4,6\rangle, \langle 1,2,3,9\rangle, \langle 1,2,4,9\rangle, \langle 2,3,4,9\rangle$

3: $\langle 1,2,3,5\rangle, \langle 1,3,4,5\rangle, \langle 2,3,4,5\rangle, \langle 1,2,3,8\rangle, \langle 1,3,4,8\rangle, \langle 2,3,4,8\rangle$

4: $\langle 1,2,4,7\rangle, \langle 1,3,4,7\rangle, \langle 2,3,4,7\rangle, \langle 1,2,4,8\rangle, \langle 1,3,4,8\rangle, \langle 2,3,4,8\rangle$
$\langle 1,2,4,10\rangle, \langle 1,3,4,10\rangle, \langle 2,3,4,10\rangle$

(ii)

(a)



(i)

1: $\langle 1,2,3,6\rangle, \langle 1,2,5,6\rangle, \langle 1,3,5,6\rangle$

2: $\langle 1,2,3,4\rangle, \langle 1,2,3,6\rangle, \langle 1,2,3,9\rangle, \langle 1,2,4,5\rangle, \langle 1,2,5,6\rangle, \langle 1,2,5,9\rangle$
$\langle 2,3,4,5\rangle, \langle 2,3,5,6\rangle, \langle 2,3,5,9\rangle$

3: $\langle 1,2,3,4\rangle, \langle 1,2,3,8\rangle, \langle 1,3,4,5\rangle, \langle 1,3,5,8\rangle, \langle 2,3,4,5\rangle, \langle 2,3,5,8\rangle$

5: $\langle\rangle$

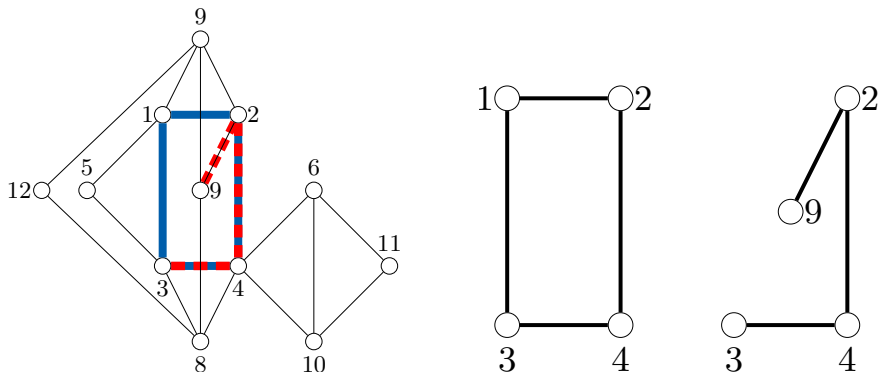
(ii)

(b)

Fig. 8.4.: State transition of the random walk for substructure sampling. (a)(i) A database graph G_i with the current state of FS^3's random walk (a) (ii) Neighborhood information of the current state $\langle 1,2,3,4\rangle$. (b)(i) The state of random walk on G_i (Figure 8.4a) after one transition (b) (ii) Updated Neighborhood information.

so that the stationary distribution of the random walk matches with a user-chosen target distribution. FS^3's target distribution favors ℓ-subgraph so that the sampling process can predominantly sample frequent subgraphs. FS^3's MCMC walk changes state by walking from one ℓ-subgraph (say g) to a neighboring ℓ-subgraph. In our neighborhood definition, for a ℓ-subgraph all other ℓ-subgraphs that have $\ell - 1$ vertices in common are its neighbor subgraph/state. To obtain a neighbor subgraph of g, FS^3 simply replaces one of the existing vertices of g with another vertex which is not part of g but is adjacent to one of g's vertices. Also, note that in g, FS^3 includes all the edges of G_i that are induced by the set of the selected vertices, so the sampled subgraph of FS^3 is always a connected induced subgraph of the graph G_i. For a given graph G_i in $\mathcal{G}$, the currently sampled ℓ-subgraph is saved so that the random walk over G_i can be resumed in a later iteration if the graph G_i is again selected in the first stage of the sampling iteration. Below, we show an example of state transition of FS^3.

Example: Suppose FS^3 is sampling 4-subgraphs from the graph G_i shown in Figure 8.4a(i) using MCMC sampling. Let, at any given time the 4-subgraph, $\langle 1, 2, 3, 4\rangle$ (shown in bold lines) is the current state of this random walk. In Figure 8.4a(ii), we list its neighbor states as four comma-separated lists, one in each row. The neighbor-list in the top row is labeled by '1', which indicates that these neighbors can be obtained from the current 4-subgraph $\langle 1, 2, 3, 4\rangle$ by retaining the vertex 1 and replacing exactly one of the remaining vertices ($\{2, 3, 4\}$) with a new vertex which is adjacent to vertex 1, ensuring connectedness. Similarly, the neighbors in the second list are obtained by retaining the vertex 2 and replacing one of the remaining vertices with a vertex from 2's adjacency list. The information in the third and fourth lists are populated in a similar manner. As shown in the top-list, $\langle 1, 2, 3, 5\rangle$ is a neighbor of $\langle 1, 2, 3, 4\rangle$; if the random walk transitions to this state, the current state becomes $\langle 1, 2, 3, 5\rangle$, which is shown in Figure 8.4b(i). In Figure 8.4b(ii), we show the updated neighbor lists considering the new state. Note that, here also we have 4 set of neighbors corresponding to 4 vertices of $\langle 1, 2, 3, 5\rangle$. The neighbor-list corresponding to



(a) An interface graph (b) Subnetwork patches

Fig. 8.5.: Subnetwork patches embedded in an interface graph.

vertex 5 is empty, as besides 1 and 3 (which are part of current state), 5 has no other adjacent vertices that can be used as a replacement vertex to build a new state.

8.4.3 Finding Sub-Network Embedding in the Interface Graph

Note that FS^3 samples ℓ-node induced subgraphs from the database graphs using a sampling-based method. It makes FS^3 scalable over large networks, but to achieve scalability it also loses completeness, i.e., for a given frequent subgraph, its support-list i.e. relative support-list may miss some of the graphs in $\mathcal{G}$ in which the pattern occurs. Therefore, at the end of the sampling process, for each of the top-k frequent subgraph patterns, we use a subgraph isomorphism algorithm for finding the embedding of the pattern in all the graphs in the database. This step completes the relative support-list of a frequent subgraph pattern and we get the actual support-list. Besides, it provides a mapping between the pattern nodes and a subset of interface graph nodes such that the mapping respects the vertex label. Thus, the embedding process enables us to inspect the subgraph pattern within the native context of residue contact graph.

Additionally, we observe that, in some cases most of the top-frequent subgraphs are almost identical except one or two nodes. After embedding, they map to a patch of

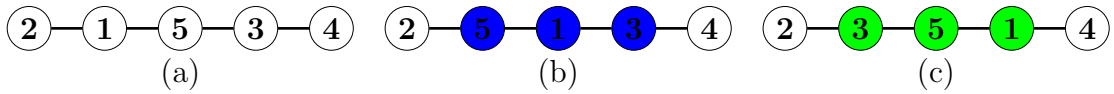


Fig. 8.6.: Random graph generation from a particular graph. Figure (a) is the input graph, Figures (b) and (c) are random graphs using switching algorithm described in Section 8.4.4. Interchanges are shown in blue and green color.

the functional motif, such that super-imposition of the embedded patches of multiple top-frequent patterns cover the entire motif. For visualizing this step, we present Figure 8.5. In Figure 8.5a, we show an example interface graph. The node labels in the figure represent residue ids. In Figure 8.5b, we list two top-frequent patterns. Bold blue and dashed red lines in Figure 8.5a show that super-imposing the embedding of the top two patterns retrieves the entire motif consisting of residues 1, 2, 3, 4 and 9 (shown in color).

For HIV-1 protease, we consider only 10 of the most frequent subgraphs, and the embedding of these subgraphs discovers the entire 16-residue dimeric lock motif in 323 out of 329 patterns. Similar treatment for the TIM protein using 20 most frequent subgraphs finds the dimeric lock in 50 out of 86 structures.

8.4.4 Statistical Significance Test of Discovered Patterns

Statistical significance test of a frequent subgraph g determines the probability (p-value) of observing g as a frequent pattern at equal or a higher support value in a database of random graphs, where the random graphs are constructed from a null model. The subgraph pattern g is statistically significant when it is highly unlikely for g to be frequent under the null model. In existing works [72], statistical significance test has been used to calculate the p-value of network motifs, which are mined from a single large graph. In these works, a set of random graphs are generated from the input graph under a specified random graph model and the subgraphs which appear in the input graph at a much higher frequency than in the random graphs are considered as significant. But, this method does not apply for our task, because in our task the we have a database of input graphs instead of a single graph. So, we generalize the above

method as below. First, we generate a set [3] of clone graph databases each containing the same number of random graphs as our input graph database. The random graphs in the clone databases are generated using a null model, details of which is discussed in the next paragraph. Then, we run our algorithm on the input graph database and on each of the clone graph databases to discover the top-k patterns and their frequencies in these datasets. Finally, we compute the z-score of a mined subgraph pattern. If the support of a subgraph pattern g in an input graph database is $s_{real}(g)$ and the average support and standard deviation in an ensemble of random graph database are $s_{avg}(g)$ and $s_{dev}(g)$, then z-score of g is calculated as shown below:

$$z\text{-score}(g) = \frac{s_{real}(g) - s_{avg}(g)}{s_{dev}(g)} \tag{8.2}$$

Then we obtain the p-value of g by considering that the support of a top-k pattern under the null hypothesis is distributed as a normal distribution. A small p-value confirms that the null hypothesis is discarded and the subgraph pattern g is statistically significant.

Random Graph Generation for Null Model As we have discussed earlier, for significant test we build a set of clone graph databases, each containing the same number of random graphs as the input graph database. Under the null model, the random graphs in the clone databases have the same degree distribution and vertex label distribution. The null hypothesis is that a frequent subgraph g is also frequent in the clone databases.

Generating a random graph (i.e. generating random 0-1 matrices) by keeping the degree distribution the same is a well studied problem. We use switching method proposed by [210]. In this method, for a given adjacency matrix of a particular graph, all the adjacency matrices which can be obtained by switching alternating 1's and 0's along the alternative rectangles or the alternating hexagons are considered to be the neighbor states. A Markov chain can be formed from this state transition and [210] has shown that if we take a particular state after p or less transitions we

[3]size of this set can be anything between 10 and 100, the higher the size the better is the estimates.

sample a random graph uniformly at random where p represents the minimum of the total number of zero's and one's in the random network. This algorithm samples correctly in the limit of long run and in practice is found to give good results compared to other methods [211]. In Figure 8.6, we show an example. Figure 8.6 (a) represents the input network (a line graph) whereas Figure 8.6(b) and Figure 8.6 (c) show two random graphs generated using the switching technique. From the figures, we can see that randomization has rewired the nodes by preserving the degree of all the nodes in the input graph. We do not alter the vertex labels, so the vertex label distribution is identical to the original graph.

For both the TIM and HIV-1 protease structures (discussed in Section 8.5.1 and 8.5.2, respectively), we generate 20 (chosen arbitrarily) clone databases containing random graphs, i.e., for each graph in the host database, we generate 20 random copies of that graph using the method described in the above paragraph. Then we apply FS^3 on both the host (input) database and each of the random graph databases separately with the *same configuration* (size-ℓ) used for the input database. Our experiments show that all our frequent patterns (size 16 for HIV-1, and size 20 for TIM) are highly significant as their frequency in the database of random graphs is zero, but the average support of HIV-1 frequent patterns is 320 (for a database size 329) and the average support of TIM frequent patterns is 50 (for a database size 86). This yields a p-value less than 0.00001 using Laplace correction for the denominator, thus making all the discovered frequent patterns in both datasets highly significant. Interestingly, no frequent patterns exist in the clone databases of random graphs; in fact, the highest support of any subgraph in each of these clone databases is exactly one, that is each subgraph appears in only one random graph.

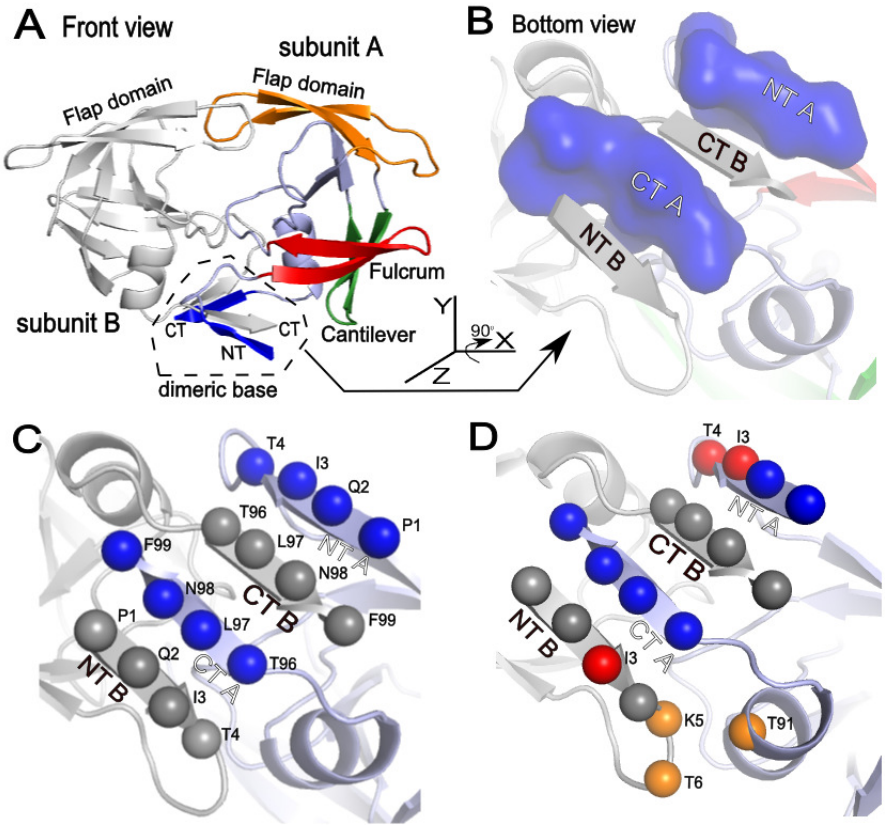


Fig. 8.7.: HIV-1 protease (HIV-1 PR) functional components, interface formation, and computationally retrieved residues from the interface residue network. Panel A shows the macromolecular architecture of the protease (based on PDB: 1a30, a closed conformation), Panel B show the lock formation at the base, Panel C shows the residues in spheres at the dimeric base, and Panel D shows the computationally retrieved residues from the interface networks. (A) Front view of HIV-1 PR dimeric structure (modified from Fig. 2 of [208]). The functionally important components are colored and labeled in subunit A. N-terminal (NT) and C-terminal (CT) strands are colored blue: NT residues 1-4 and CT residues 96-99. NT and CT strands of one subunit form a ridge where CT strand of the other subunit is locked, and vice versa. Fulcrum (red, residues 9-21) - at one end of this component is the C-terminus and on the other end there are the active site region. Flap domain (orange, residues 37-58) has three main regions. Cantilever (green, residues 59-75) is located at the C-terminal end of the Flap domain. (B) Lock formation at the base of the structure - NT and CT strands of chain A form a ridge where CT from B is inserted and vice versa. (C) The residues on NT and CT of each chain forming the lock are identified (PDB 1a30). (D) Blue ones are the correctly recognized interface residues by graph mining. Three residues forming the lock shown in the panels B and C that the mining algorithm failed to identify are colored red. Instead, the mining included the orange residues in the pattern that are not part of the lock pair.

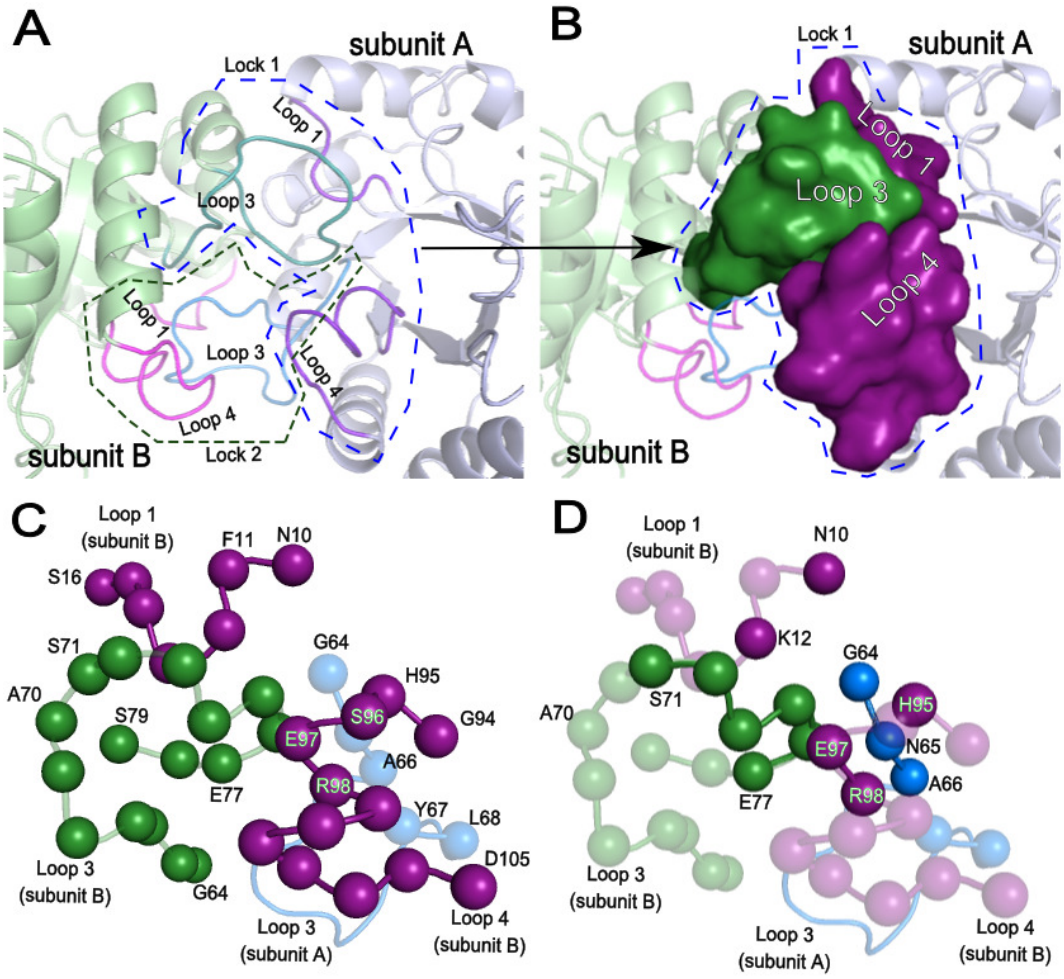


Fig. 8.8.: Type 1 interface of TIM dimeric structure. (A) Loop 3 from subunit A and Loop 1 and Loop 4 from subunit B form a lock at the interface, and vice versa. (B) Surface view of Lock 1. (C) Residues of the loops involved in Lock 1 are shown in spheres. (D) Retrieved residues in Lock 1 are shown in bright color and others are deemed.

8.5 Experimental Results

In this section, we present our experimental findings. Section 8.5.1 and 8.5.2 shows that our graph-mining method retrieves the dimerization locks in each of the protein structure with multiple conformations whereas in Section 8.5.3, we show that our approach captures class specific active sites for the six top-level classes of enzymes each composed of multiple protein structures with a single conformation. In Section 8.5.1 and 8.5.2, we report average pairwise RMSD (Root Mean Square Deviation) distance among conformers[4]. For calculating RMSD distance, we use Kabsch algorithm [212] and Quaternion algorithm [213]. Kabsch algorithm [212] is a simple procedure which determines a best rotation of a given vector set into a second vector set by minimizing the weighted sum of squared deviations. On the other hand, Quaternion algorithm [213] solves for the orientation and the position of an object by minimizing a single cost function associated with the sum of the orientation and position errors.

8.5.1 HIV-1 Protease Structures

HIV-1 PR dimerization occurs at the interface between two homologue structures-each subunit having 99 residues. Each subunit structure can be divided into functionally important components (Fig. 8.7A): 1) Terminal domains (blue, NT strand: residues 1-4 and CT strand: residues 96-99) that form the base of the protease structure. 2) Flap domain (orange, residues 37-58) that opens and closes the structure for substrate recruitment and product release. The coordination of motion between 3) Fulcrum (red, residues 9-21) and 4) Cantilever (green, residues 59-75) controls the opening/closing motion of the Flap domain.

NT (residues 1-4) and CT (residues 96-99) strands from one subunit form a ridge where the CT strand from the partner subunit gets inter-digitated, and vice versa (Fig 8.7B). This interlocked configuration of the terminal strands forms a strongly-

[4] `https://github.com/charnley/rmsd`

bound dimeric base which facilitates the opening-closing motion of the flap tips of the Flap domains.

We selected 329 HIV-1 structures from PDB [214] such that each structure has no missing residues. Subsequently, we have created an interfacial network (connected graph) for each structure considering the interfacial residues that are within 8 Angstrom (Å) distance from any residue from the partner subunit. We also connect two residues within the same subunit if their distance is within 4 angstrom, i.e., we set $\gamma = 8$Å and $\delta = 4$Å. The average number of nodes and edges for these networks are 64.00 and 242.00 respectively. Then, we mined these 329 connected graphs using FS^3, our graph mining method. If the proteins are structurally similar, the frequent subgraphs are more likely to form; so one may opt for more precise results by setting smaller values of δ and γ. For this purpose, structural similarity of a collection of proteins should be obtained by optimally superimposing the proteins one on top of another, and then computing The average RMSD distance. We perform the same by using both the Kabsch and the Quaternion algorithm on our HIV-1 dataset. The median RMSD value was 0.7305 (minimum =0.0, maximum=2.74) when the statistics was calculated over all the 329 conformers of HIV-1.

Figure 8.7C labels the 16 residues of four strands that form the dimeric lock at the base - four residues in each strand. For a pattern of size 16, our method retrieves 13 of these base forming residues. Three residues (I3 on NT B, I3 and T4 on NT A) shown in red were not included, rather K5 and T6 on the coil connecting NT B and Fulcrum and T91 on the helical region at the N-terminal end of CT A got included.

8.5.2 TIM Structures

TIM is the fifth enzyme in the glycolysis pathway that produces energy in all living organisms. The functional oligomeric state of TIM is a homo-dimeric structure in most mesophilic organisms. A TIM subunit has a central barrel formed by

eight strands ($\beta 1-\ \beta 8$) which is surrounded by eight helices ($\alpha 1$-$\alpha 8$). Eight back loops (BL1-BL8) connect from helix to strand and eight front loops (FL1-FL8 or simply Loop 1−Loop 8) connect from strand to helix. Details can be found in [215] (Fig. 8.8A). Two monomeric subunits form the dimeric TIM structure through interaction of a pair of symmetric locks at their interface. We construct interfacial network for each of the 86 triosephosphate isomerase (TIM) PDB structures with γ = 8Å and δ = 4Å. The average number of nodes and edges for these networks are 158.50 and 884.75 respectively. The average RMSD distance using both the Kabsch and the Quaternion algorithm is: 5.76 (min=0.0, max=24.64) and it was calculated over 25 TIM structures for which the number of C_α carbons were the same.

A dimer of two subunits is formed by two symmetric locks at the interface: Loop 1 and Loop 4 of one subunit form a ridge wherein Loop 3 of the partner subunit gets engaged, and vice versa. Figure 8.8A shows such a pair of locks at the dimeric interface of a TIM structure (PDB 1ypi). The space-filled view in Fig. 8.8B illustrates one of these locks more clearly. Figure 8.8C illustrates the residues of the involved loops in spheres (L1 of chain B: $F_{11}K_{12}$ $L_{13}N_{14}G_{15}$ S_{16}, L4 of chain B: G_{94} $H_{95}S_{96}E_{97}$ $R_{98}R_{99}$ $S_{100}Y_{101}$ $F_{102}H_{103}$ $E_{104}D_{105}$, L3 of chain A: $Q_{64}N_{65}$ $A_{66}Y_{67}L_{68}$ $K_{69}A_{70}$**S_{71} $G_{72}A_{73}F_{74}T_{75}G_{76}$ $E_{77}N_{78}S_{79}$**).

Our graph-mining method retrieves the key residues of the locking mechanism. When the pattern-size is 12, the retrieved residues are: L1 (chain A): 10, 12; L4 (chain A): 95, 97, 98; L3 (chain B): 72-77; N-terminal base of L3 (chain A): 64. And, when the pattern-size is 12 are: L1 (chain A): 10, 12; L3 (chain B): 72,..., 77; L4 (chain A): 95, 97, 98; and N-terminal base of L3 (chain A): 63, 64, 65, 66. The residues from the interlocking mechanism that are retrieved by our method are shown in bright spheres (Fig. 8.8D). Interestingly, all the residues of Loop 3 ($S_{71}G_{72}$ $A_{73}F_{74}T_{75}$ $G_{76}E_{77}$ - 7 residues [216]) are successfully retrieved. Moreover, the retrieved patterns reveal that a few residues at the N-terminal region of Loop 3 from chain A (residues G_{64}, N_{65}, and A_{66}) engages in the lock formation.

8.5.3 Enzymes

Enzymes are known to be macromolecular catalysts that speed up biochemical reactions by providing an alternative reaction pathway of lower activation energy. In the absence of enzymatic catalysis, most biochemical reactions are so slow that they would not occur under the mild conditions of temperature and pressure that are compatible with life [217]. Enzymes accelerate the rates of such reactions by well over a million-fold, so reactions that would take years in the absence of catalysis can occur in fractions of seconds if catalyzed by the appropriate enzyme. Enzymes bind their reactants or substrates at a small portion of their structure that is known as the active site. Active sites are substructures on the surface of an enzyme, usually composed of amino acids from different parts of the polypeptide chain that are brought together in the tertiary structure of the folded protein [217]. Hence, mining functional motifs (active sites) from the interface region of enzymes is important for understanding the underlying mechanisms that allow them to interact with other molecules and perform their vital functions that sustain life in the cells. The International Union of Biochemistry and Molecular Biology[5] has developed a classification system for enzymes[6] that, at its top-level, divides them into six groups namely:

- *Oxydoreductase (EC1)*: catalyze oxidation/reduction reactions.
- *Transferase (EC2)*: transfer of a chemical group from substrate to product.
- *Hydrolase (EC3)*: cleavage of bonds by hydrolysis.
- *Lyase (EC4)*: elimination of various bonds by means other than hydrolysis and oxidation.
- *Isomerase (EC5)*: catalyze isomerization changes within a single molecule.
- *Ligase (EC6)*: join two molecules with covalent bonds.

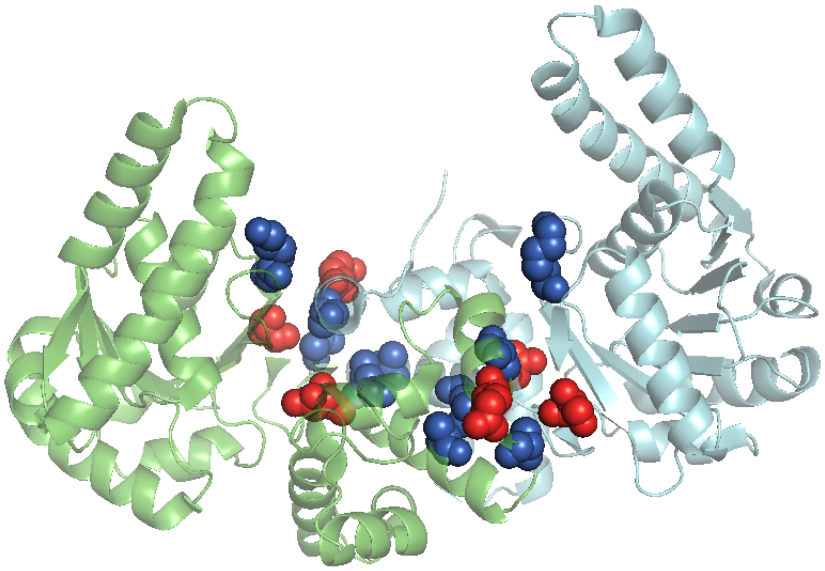
[5]http://iubmb.org/
[6]http://www.enzyme-database.org/

Table 8.1.: Interfacial network statistics for our subset of enzymes from the Dobson and Doig (D&D) protein structure dataset [209].

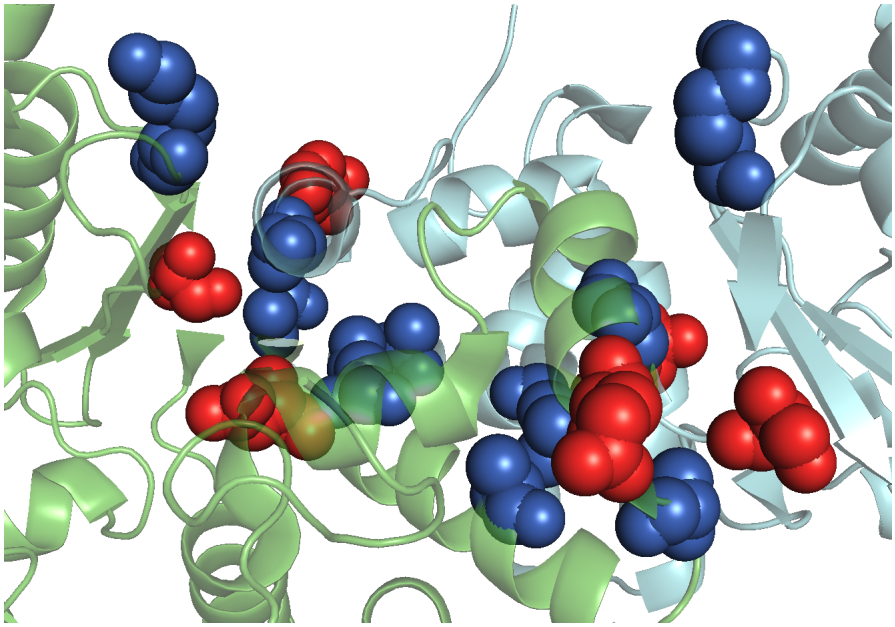
Class	#structures	#Subclasses	#Sub-subclasses	Avg. #Nodes	Avg. #Edges
$EC1$	76	16	35	151	487
$EC2$	84	8	21	102	318
$EC3$	91	8	29	82	262
$EC4$	40	4	9	128	414
$EC5$	21	5	10	118	367
$EC6$	14	4	6	134	403

Unlike the previous two experiments where we mined patterns that are shared across the different conformations of the same protein, in this experiment, we are interested in mining functional motifs that are shared by multiple protein structures within the same group of enzymes but not across the different classes. That is to say, class specific active sites that allow each of the enzyme classes to exert a specific function. Since enzymes need to bind to their substrates at their active sites to perform their biological functions, mining class-wise frequent patterns at the interface region of enzymes could help to unravel class specific active sites. We use enzymes from the Dobson and Doig (D&D) protein structure dataset [209] which originally consists of 1178 proteins divided into a group of 691 enzymes and a second group of 487 non-enzymes. We consider only the subset of oligomeric protein structures from the enzymes, *i.e*, structures with at least two sub-units. The remaining set of enzymes is composed of 326 protein structures. Table 8.1 shows the number of protein structures in each class, the number of EC subclasses and sub-subclasses in each group, as well as, the average number of nodes and edges from the derived graphs.

We have constructed an interfacial network for each PDB structure of the set, based on Equation (8.1) with $\gamma = 10$Å and $\delta = 4$Å. After this step, we obtained a set of undirected, vertex-labeled graphs—each corresponding to one enzyme protein structure. We used FS^3 to discover function specific subgraph motifs across the six



(a) Front view of the L-3-hydroxyacyl-CoA dehydrogenase protein structure.



(b) Zoomed view of the active site of the structure: in red and blue are residues from the catalytic and binding sites respectively

Fig. 8.9.: Retrieved frequent pattern representing active site at the L-3-hydroxyacyl-CoA dehydrogenase protein structure. (a) A front view of the entire structure with the active site and (b) a zoomed view of the active site with the catalytic site (in red) and binding site (in blue).

Table 8.2.: The number of patterns overlaps within different groups for a specific size, ℓ and top-200 patterns.

	# Overlaps for a specific ℓ			
Classes	$\ell = 5$	$\ell = 6$	$\ell = 7$	$\ell = 8$
$EC1 - EC2$	14	8	7	0
$EC1 - EC3$	0	0	0	0
$EC1 - EC4$	20	2	0	0
$EC1 - EC5$	10	0	0	0
$EC1 - EC6$	5	4	1	0
$EC2 - EC3$	17	2	2	0
$EC2 - EC4$	14	0	0	0
$EC2 - EC5$	11	3	0	0
$EC2 - EC6$	8	0	0	0
$EC3 - EC4$	12	0	0	0
$EC3 - EC5$	0	0	0	0
$EC3 - EC6$	3	0	0	0
$EC4 - EC5$	6	0	0	0
$EC4 - EC6$	8	0	0	0
$EC5 - EC6$	3	0	0	0

different classes of enzymes. We mined 200 most frequent patterns for each of the following sizes [7] 5, 6, 7 and 8 from each of the six enzyme classes.

We first validate whether the discovered patterns are abundant across all six enzyme classes or they are frequent only within an enzyme class. Since the enzyme classes are derived from their function, patterns that are frequent only within an enzyme class are functional motif for that class of enzyme. For this validation, we count the number of patterns that occurs over multiple classes of enzymes. For a clean presentation, in Table 8.2 we only show the number of patterns which overlap over a pair of enzyme classes. Along the rows we list $\binom{6}{2} = 15$ pairs of enzyme classes and along the column we list the size of patterns. Each cell entry shows the number of overlapping patterns across the corresponding pair of enzyme classes for the given pattern size. For example, there are 14 patterns (out of 200 most frequent patterns) of size-5 which overlaps across enzyme class 1 and enzyme class 2. We notice that the overlap between the sets of patterns mined from each class is very small for the sizes 5, 6 and 7, and that there is no overlap at all at the size 8. Thus the number of overlaps decreases while increasing the size of patterns. This shows that our modeling and mining method allows to unravel class specific patterns at the interfacial region. Besides, the fact that each of the classes performs a particular function also suggests that the discovered patterns are active sites and they are specific to functions performed by the enzymes in that class.

In fact, the active site of an enzyme is composed of two components. The first component is the catalytic site that is known to be small ($2 - 4$ amino acids [218, 219]), highly conserved, and allows the enzyme to perform its function. The second component is the binding site which allows the recognition and precise positioning of an enzyme's substrate in proximity to the chemically active catalytic residues and lower the energy of the transition state, which aids catalysis [218]. Figure 8.9 shows an example of a protein structure namely the *L-3-hydroxyacyl-CoA dehydrogenase* (PDB IDs: 1F14) from the EC1 class of our dataset and the mapping of a frequent pattern of

[7]Size of a subgraph pattern is the number of vertices in that pattern.

size 8 that we discovered using our subgraph mining method. The pattern contains a catalytic site composed of the residues "Glutamine", "Asparagine", and "Serine" that have been identified at the same structure in the Catalytic Site Atlas[8] [218, 220], a database of both hand-curated and automatically annotated catalytic sites in enzyme structures. Since the catalytic and binding sites co-occur together as part of the same active site, we consider the five remaining residues ("Lysine", "Leucine", and 3 "Alanine") from the pattern as of the binding site. Figure 8.9 shows the catalytic and binding site in red and blue respectively.

8.6 Chapter Summary

In this work, we proposed a method for the discovery of functional motifs from the interface region of dimeric protein structures. Our method uses a graph representation of the interface region of these structures, and mines a fixed-size highly frequent subgraphs over those graphs. We then use a small collection of subgraphs to discover functional motifs at the interface region of the structures. In our experiments, we showed that our method discovers the oligomeric lock motif in the majority of the structures for both HIV-1 protease and TIM protein. We also showed that our method discovers class specific active sites at the interfacial region of the six top-level classes of enzymes.

There are significant scopes for extending this work. First, we plan to make our FS^3 software a stand-alone tool for the functional motif discovery at the interfacial region of proteins. As we have observed highly frequent patterns of a given size although captures the functional motifs, each such patterns sometimes misses a few residues of a functional motifs. At this stage, we manually patch together a collection of patterns to identify the entire functional motifs. One immediate future work is to identify a cluster of similar patterns which overlap the core of a functional motif and

[8] http://www.ebi.ac.uk/thornton-srv/databases/CSA/

then automatically patch them together to discover the functional motifs. Also, we are planning to extend the functionality of our FSM based functional motif discovery tool. Currently, our FSM method counts the frequency of a pattern by its identical occurrences over different graphs. As future work, we are planning to extend our approach with a selection module that accounts for amino acids similarity as in [84, 221] for counting occurrences of a pattern.

9. FINDING NETWORK MOTIFS USING MCMC SAMPLING

9.1 Introduction

Studying the local topology is an important step for modeling the interaction among the entities in a network. In a seminal work around a decade ago, Shenorr et al. [72] hypothesized that network motifs play an important role in carrying out the key functionalities that are performed by the entities in a biological network. Since then, researchers have also discovered that network motifs are building block for complex networks from many diverse disciplines including biochemistry, neurobiology, ecology, engineering [73], proteomics [74], social sciences [75] and communication [76].

Finding network motifs is computationally demanding. To identify whether a given subgraph topology is a motif, we need to count the topology's frequency in the input network as well as in many randomized networks. Counting a topology's frequency in a single network is a challenging task as it requires solving subgraph isomorphism, a known $\mathcal{NP}$-complete problem. As the size of the motif grows, the number of candidate motifs increases exponentially, and the task becomes more challenging. To cope with the enormous computation cost of exhaustive counting of the frequency of candidate motifs, researchers consider various sampling based methods that obtain an approximation of relative frequency measure (which we call concentration) over all the candidates of a given size. Most notable among these methods are MFinder [67], MODA [77], and RAND-ESU [68]. Besides these approximate methods, exact motif counting methods are also available, such as, GTrieScanner [69], ESU [68], Grochow-Kellis algorithm [78], Kavosh [70], and NetMODE [79]; However, their application is limited to small networks only. In this work, our focus is on finding concentration of prospective motifs using a novel sampling based method.

The quality of a sampling based method depends on three critical performance metrics: accuracy, convergence, and execution time. Existing sampling based methods are poor in one or more of the above performance metrics. For instance, MFinder is costly and it scales poorly with the size of the desired motifs. Authors in [68] have shown that the cost of subgraph sampling of MFinder increases exponentially with the size (number of vertex) of the subgraph. It is also poor in terms of accuracy and convergence. A similar method, RAND-ESU [68] is significantly faster than MFinder and yet its scalability is also not that satisfactory. Besides, its sampling accuracy and convergence behavior are also poor.

Another important fact about the existing sampling based methods is that they require random access to any of the vertices or the edges in the networks. This becomes a severe limitation for networks for which such unrestricted access is not available. For an instance, consider the Web network or a hidden network, a user may not have access to any arbitrary vertex/edge in the input network for security reason; rather, the desired node can only be accessed from another node which is one-hop away from it; such scenarios are common in real-life and are considered in the task of snowball sampling [222]. None of the existing methods can be used for finding motifs in a graph that only allows restricted access, such as crawling.

In this work, we propose two random walk based methods, namely MHRW (Metropolis-Hastings random walk) and SRW-RW (Simple Random Walk with Re-weighting) for approximating the concentration of arbitrary-sized pattern graphs in a large network. The underlying mechanism of both the methods is a Monte Carlo Markov Chain (MCMC) sampling over the candidate motif space, which is guaranteed to compute an unbiased estimate of concentration of all the candidate motifs of a given size simultaneously. Since, our methods are based on random walk over the edges of the input graph, they only require a restricted access over the network such that at any given time of the walk the one-hop neighboring nodes of currently visiting candidate are accessible. Besides, the methods are scalable and are significantly faster than the existing methods. They also have better convergence property and

small memory footprint. While preparing for the final manuscript of this work, we have found another work [71] which is similar to our work.

9.2 Background

9.2.1 Graph, Subgraph, Induced Subgraph

Let $G(V, E)$ is a *graph*, where V is the set of vertex and E is the set of edges. Each edge $e \in E$ is denoted by a pair of vertices (v_i, v_j) where, $v_i, v_j \in V$. A graph without a self-loop or multi edge is a simple graph. In this work, we consider simple, connected, and undirected graphs.

A graph $G' = (V', E')$ is a *s*ubgraph of G (denoted as $G' \subseteq G$) if $V' \subseteq V$ and $E' \subseteq E$. A graph $G' = (V', E')$ is a *v*ertex-induced subgraph of G if G' is a subgraph of G, and for any pair of vertices $v_a, v_b \in V'$, $(v_a, v_b) \in E'$ if and only if $(v_a, v_b) \in E$. In other words, a *vertex-induced* subgraph of G is a graph G' consisting of a subset of G's vertices together with all the edges of G whose both endpoints are in this subset. In this paper, we have used the phrase *induced subgraph* for abbreviating the phrase vertex-induced subgraph. If G' is an induced subgraph of G and $|V'| = p$, we call G' a *p-subgraph* of G. An *embedding* of a graph G' in another graph G is a subgraph S of G such that S and G' are isomorphic;

For a given vertex count, the number of distinct graph topologies is fixed. We use the symbol Λ_p to denote the set of all such topologies. To denote one specific topology in Λ_p we use the symbol $\omega_{p,q}$, where q is the order of that topology (considering an arbitrary but fixed ordering) among all the size p topologies. The set of induced embeddings of all graphs in Λ_p in graph G is the collection of p-subgraphs of G. Figure 9.1 shows all the elements of the sets Λ_3, Λ_4 and Λ_5. Using the order of the topologies in this figure, $\omega_{3,1}$ is the 3-node line graph.

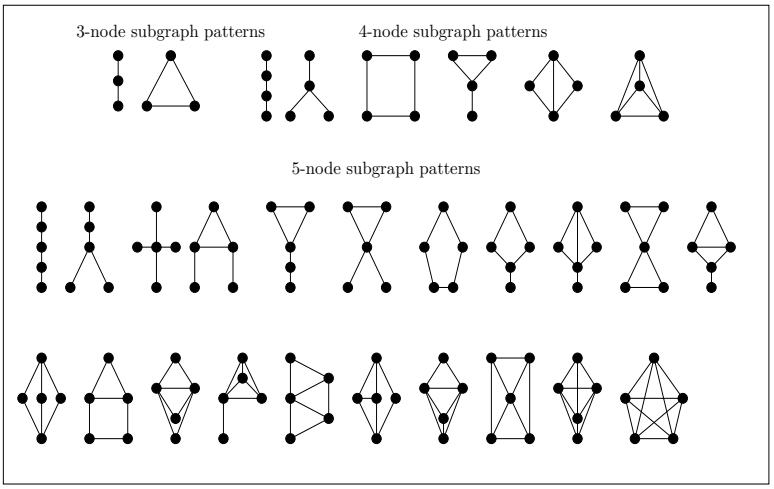


Fig. 9.1.: All 3, 4 and 5 node topologies.

9.2.2 Subgraph Concentration

The frequency of a particular p-subgraph topology g in an input graph G is the number of times it appears in G. We denote it by $f_G(g)$. The concentration of g in G is $C_G(g)$, which is defined as the normalized frequency over the cumulative frequency of all the subgraph topologies in the set Λ_p. Mathematically,

$$C_G(g) = \frac{f_G(g)}{\sum_{h \in \Lambda_p} f_G(h)} \tag{9.1}$$

9.2.3 Motif

A Motif is a subgraph topology which occurs in an input network at a significantly higher frequency than it occurs in a set of random networks with identical characteristics. For this purpose, the random networks are generated from the input network by imposing the constraint that the vertices of a random network has the identical degree distribution as that of the input network. There are several methods for generating random networks with identical degree distribution, but the most popular is the switching algorithm [211], which we use in this work. The significance of frequency deviation between the input network and the set of random networks is typically measured using z-score and p-value. If $\overline{f_{G_r}(g)}$ is the mean frequency of g in a set of randomized graphs G_r (constructed from G), and $\sigma_{G_r}(g)$ is the corresponding standard deviation, then z-score of g for the input network G is defined as:

$$z_G(g) = \frac{f_G(g) - \overline{f_{G_r}(g)}}{\sigma_{G_r}(g)} \tag{9.2}$$

If the z-score of g is greater than some pre-specified threshold then we call g a motif. Since, setting this threshold requires domain expertise, all the existing motif finding methods consider it as a run-time parameter; we also follow the same in our work. For sampling based solution, we use concentration of subgraph instead of their frequency. Hence, z-score is defined as below:

$$\widehat{z}_G(g) = \frac{\widehat{C}_G(g) - \overline{\widehat{C}_{G_r}(g)}}{\widehat{\sigma}_{G_r}(g)} \tag{9.3}$$

In equation 9.3, we use $\widehat{C}_G$, and $\widehat{\sigma}_G$ to denote that they are statistics obtain from random sample of size-p embeddings.

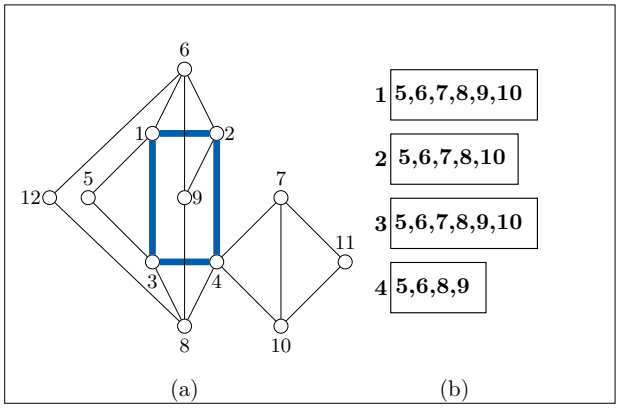
9.2.4 Markov chains, and Metropolis-Hastings (MH) Method

A Markov chain is the sequence of Markov process over the state space S. The state-transition event is guided by a matrix, T, called *transition probability matrix*. The chain is said to reach a stationary distribution π, when the probability of being in any particular state is independent of the initial condition, it is reversible if it satisfies the *reversibility condition* $\pi(i)T(i,j) = \pi(j)T(j,i), \forall i,j \in S$ and it is *ergodic* if it has a stationary distribution. The main goal of the MH is to draw samples from some distribution $\pi(x)$, called the *target distribution*, where, $\pi(x) = f(x)/K$; here K is a normalizing constant which may not be known and difficult to compute. It can be used together with a random walk to perform MCMC sampling. For this, the MH algorithm calculates the *acceptance probability* using the following equation:

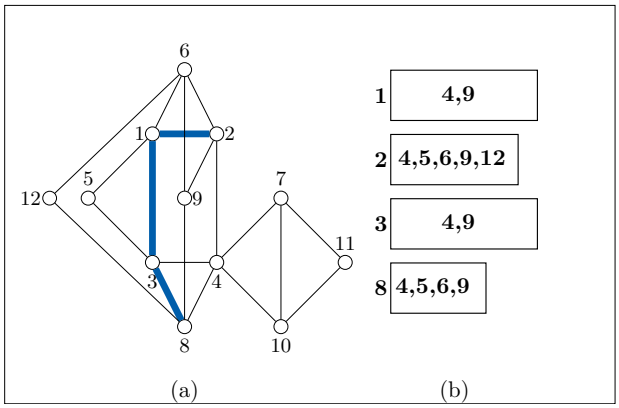
$$\alpha(x,y) = \min\left(\frac{\pi(y)q(y,x)}{\pi(x)q(x,y)}, 1\right) \tag{9.4}$$

9.3 Methods

Given a graph G (which we refer as input graph) and an integer p, a sampling based method samples a small set of p-subgraphs of G. From this set, it approximates the concentration of each topology in Λ_p as shown in section 9.2.3. To measure the exact concentration, one must perform unbiased sampling, where each of the p-subgraphs has an uniform probability to be sampled. This is not an easy task, as the sample space is very large. Besides, a direct sampling method is not applicable because for that we need to enumerate all the p-subgraphs (to obtain the size of the sample



(a) Left: A graph G with the current state of random walk; Right: Neighborhood information of the current state (1,2,3,4)



(b) [Left: The state of random walk on G (Figure 9.2a) after one transition; Right: Updated Neighborhood information

Fig. 9.2.: Neighbor generation mechanism.

space), which we want to avoid. So, an indirect sampling strategy must be followed. Both MFinder [67] and RAND-ESU [68] adopt indirect sampling; however, they differ in the sampling methodologies. MFinder's sampling is biased which requires post-adjustment of concentration for correcting the bias; on the other hand, RAND-ESU guaranty a uniform sampling which requires no correction. For large p, both MFinder and RAND-ESU are costly.

In this paper, we propose MHRW, and SRW-RW for sampling p-subgraphs of a graph using Markov chain Monte Carlo (MCMC) sampling. As a Metropolis-Hasting based method (discussed in sec: 9.2.4), they perform a random walk over the state space so that the stationary distribution of the random walk converges to a desired target distribution. For our task, the state space are the set of p-subgraphs. Since, we want to approximate the concentration of each of the topologies in Λ_p, our target distribution is *uniform*, i.e., we want to sample each of the p-subgraphs with an identical probability. If $\mathcal{P}$ is the set of the p-subgraphs in the input graph G, and π is the target distribution, we want $\pi(g) = 1/|\mathcal{P}|, \forall g \in \mathcal{P}$.

For the random walk of both MHRW and SRW-RW, a neighbor of a p-subgraphs (say, g) is obtained by simply replacing one of its existing vertices of g with another vertex which is not part of g and find the subgraph induced by the new vertex-set. While replacement, the methods ensure that the new set of vertices induce a connected p-subgraph. At every iteration, all possible neighbors are populated using the above strategy. For a state, the number of neighboring states are called its *degree*.

Example: Suppose our sampling method (MHRW or SRW-RW) is sampling a 4-subgraph from the graph G shown in Figure 9.2a(Left). Let, the 4-subgraph $\langle 1, 2, 3, 4\rangle$ (shown in bold lines) is the existing state of this random walk. One of it's neighbor state is $\langle 1, 2, 3, 8\rangle$, which can be obtained by replacing the vertex 4 by the vertex 8. In Figure 9.2a(Right) we show the information of all its neighbors. Box labeled by x contains all the vertices that can be used as a replacement of vertex x to get a neighbor. If the random walk transition chooses to go to the neighbor state $\langle 1, 2, 3, 8\rangle$, it can do so simply by adding the vertex 8 (a vertex in the box labeled by 4) and deleting the vertex 4. The updated state of the random walk along with the updated neighbor-list is shown in Figure 9.2b. The degree of a state is the number of neighbors, which is simply the sum of the entries in each of the boxes; thus the degree of state $\langle 1, 2, 3, 4\rangle$ is 21, and the degree of the state $\langle 1, 2, 3, 8\rangle$ is 13. ■

To apply MH algorithm, we also need to decide on a proposal distribution, q. For MHRW random walk, we choose the proposal distribution to be uniform, i.e., in the

Algorithm 8: MHRW Pseudocode.

```
Input :
        - Graph G
        - Size of subgraph, p
        - Size of the sample set, N: |S|
[1] g ← Starting State ;
[2] M ← φ ;
[3] i ← 0 ;
[4] d_g ← Neighbor count of g ;
[5] while i < N do
[6]     x ← state, S sampled lastly from G ;
[7]     h ← Any neighbor of g chosen
            uniformly at random from (1, |d_g|) ;
[8]     d_h ← Neighbor count of h ;
[9]     accp_val ← d_g/d_h ;
[10]    accp_probablility ← min(1, accp_val) ;
[11]    if uniform(0,1) ≤ accp_probability then
[12]        g ← h ;
[13]        d_g ← d_h ;
[14]        i ← i + 1 ;
[15]        Generate the Canonical code of g ;
[16]        Insert the code into the set M
                and update the count ;
[17] Normalize the frequency using equation 9.5, ∀_i ω_{p,i} ∈ M ;
[18] return M ;
```

proposal step MHRW chooses one of g's neighbors uniformly. If $h \in \mathcal{P}$ and h is a neighbor of g based on our neighborhood definition, using proposal distribution, the probability of choosing h from g, $q(g,h) = 1/d_g$, where d_g is the degree of the state g. Also note, if $m \in \mathcal{P}$, but m is not a neighbor of g, $q(g,m) = 0$, i.e., transitions are allowed among neighboring states only.

Using the proposal (q) and target (π) distributions, MHRW method is simply an implementation of the algorithm that we discussed in Section 9.2.4. A pseudo-code of MHRW is given in Figure 8. At the beginning of the sampling for each topology in Λ_p, we assign a counter which is initialized to 0. As the sampling progress, for each state we identify the specific topology that the state represents, and increment its counter by 1. Thus, if $\mathcal{S}$ is the sample set, the concentration equation defined in 9.1 for g where $g \in \Lambda_p$ becomes:

$$\widehat{C}(g) = \frac{1}{|\mathcal{S}|} \sum_{x \in \mathcal{S}} 1_{(x==g)} \tag{9.5}$$

At any iteration from the current stage g, the method chooses one of its neighbors, (say, h) using the proposal distribution (uniform), and either accept or reject the proposed move using Equation 9.4 i.e. MHRW adjusts the transition probability by accepting or rejecting the proposed transition so that the target distribution is guaranteed to be uniform.

On the other hand, an iteration of SRW-RW (simple random walk with re-weighting) simply chooses one of the neighbors uniformly and make this transition. Thus the difference between MHRW and SRW-RW is that the latter chooses the proposed transition with 100% probability. This does not guarantee uniform sampling of the states (p-subgraphs); rather the states are sampled in proportional to their degree values. In other words, the target distribution of simple random walk is directly proportional to the degree value of the p-subgraphs. So, the concentration of the topologies in Λ_p is also biased in proportional amount. To obtain an unbiased estimate of concentration, the estimated concentration should be re-weighted, which

gives the name simple random walk with re-weighting or in short SRW-RW. After re-weighting the concentration equation (Equation 9.1) of SRW-RW takes the following form:

$$\widehat{C}(g) = \frac{1}{W} \sum_{x \in \mathcal{S}} (1/d_x)_{(x==g)} \tag{9.6}$$

where, W is the sum of the total weights, i.e., $W = \sum_{x \in \mathcal{S}} (1/d_x)$. Such an idea of re-weighting has been used in [223] for approximating degree distribution of a large network by sampling.

Pseudo-code of SRW-RW is similar to the pseudo-code of Figure 8, the only difference is that, there is no acceptance rejection step and in Line 12, instead of incrementing the frequency count by 1, we increment the concentration by $1/d_g$. Finally, we normalize in Line 13 using equation 9.6 instead of equation 9.5.

Claim: For a given p and an input graph G, both MHRW and SRW-RW returns an unbiased estimate of the concentration of a topology in Λ_p.

Proof: Assume $g \in \Lambda_p$ is an arbitrary topology and $\mathcal{S}$ is a set of induced subgraph sampled from G. The expectation of g's concentration in G is $E\left[\widehat{C}(g)\right] = E\left[\frac{1}{|\mathcal{S}|} \sum_{x \in \mathcal{S}} 1_{(x \cong g)}\right] = E\left[P_u(x \cong g)\right]$. Here, $P_u(x \cong g)$ is the probability that a graph x in the sample set $\mathcal{S}$ is isomorphic to the topology g when it is sampled under uniform distribution. But, this value is the exact concentration value of g. So, $E\left[\widehat{C}(g)\right] = E\left[C_g\right] = C_G$. So, MHRW returns an unbiased estimate of the concentration of a topology in Λ_p.

By construction, the stationary distribution π for SRW-RW's random walk is proportional to the degree of a p-subgraph. Thus, for an arbitrary p-subgraph, w, its stationary probability $\pi(w) = d_w/K$ where K is a normalizing constant. For a topology $g \in \Lambda_p$, before re-weighting the expected value of its concentration is equal to $\sum_{w \in \mathcal{P}} \pi(w) \cdot 1_{(w \cong g)} = \sum_{w \in \mathcal{P}} \frac{d_w}{K} \cdot 1_{(w \cong g)}$. However if each sample w of type g contributes only $1/d_w$ instead of 1 in the counter of g, the expected value of concentration becomes $\sum_{w \in \mathcal{P}} \frac{d_w}{K} \cdot (\frac{1}{d_w})_{(w \cong g)} = \frac{1}{K} \sum_{w \in \mathcal{P}} 1_{(w \cong g)} = \frac{1}{K} C(g)$, which is the unbiased con-

Table 9.1.: Dataset statistics.

Graph	Vertex	Edge	Average Degree
Yeast	2,224	6,609	5.94
Jazz	198	2,742	27.49
ca-GrQc	4,158	13,422	6.43
ca-HepTh	8,638	24,806	5.74
ca-AstroPh	17,903	196,972	22.0

centration scaled by a multiplicative constant. Since the concentration of all the topologies in Λ_p sums to 1, the expected value of the concentration returned by equation 9.6 after normalization is an unbiased estimate of the true concentration. ■

9.3.1 Implementation Issues

Starting State. When we start the random walk on G, both MHRW, and SRW-RW starts from an arbitrary p-subgraph. To find it, the methods randomly choose an edge (of G) and initialize the vertex set with the vertices of this edge. Then they populate other vertices adjacent to it and return an induced subgraph of the desired size. As the input graph is connected, this process always returns a p-subgraph of G.
Canonical label of a graph We use *min-dfs-code* [58] for canonical labeling of the graph to unify different isomorphic forms of the same graph.

9.4 Results and Discussion

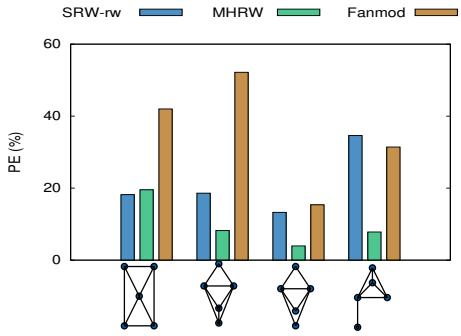
We implement MHRW and SRW-RW in C++ language and perform a set of experiments for evaluating their performance. We run all the experiments in a computer with 2.60 GHz processor and 4 GB RAM running Linux operating system. For experiments, we use graphs of different sizes from different domains. Table 9.1 lists the graphs along with the vertex count, the edge count and the average degree. Since the existing implementation of our methods only consider undirected graphs, all the

input graphs are made undirected if necessary. The graphs are available from the following two web sites [1].

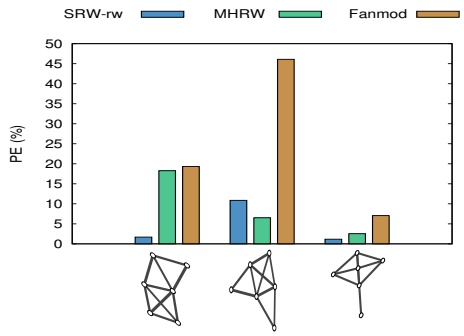
Experimental results in the earlier works show that RAND-ESU is the best among these three methods. In [68], Wernicke have shown that RAND-ESU is significantly faster than MFinder with a better accuracy. Another recent work [77] shows that RAND-ESU is the fastest among a set of methods including MODA. In this paper, we compare the performance of our methods with RAND-ESU to show that our methods are better than RAND-ESU in different performance metrics. We also considered MODA [77] for a comparison, but we found that its available implementation is unstable; the same fact was also reported by the authors of [79]. Note that we do not compare our methods with existing exact algorithm as they do not scale with the size of motif and also with the size of the input graph. For comparison with RAND-ESU, we use the implementation by authors that is available in the FANMOD library. Note that, in this implementation, the algorithm supports subgraph size up to 8. Besides a user need to set some probability values, which we set using the recommendation in FANMOD's documentation. In the result section, we will refer RAND-ESU as FANMOD following the convention in the earlier works.

We use three performance metrics: runtime, error, and convergence to compare our method with others. To compute the error value for a topology g, we first find the exact concentration of g using an exact method, then we find the approximate concentration using the sampling based method; the absolute difference between the above two concentration normalized by the actual concentration is the error for the topology g. However, since the sampling method is a randomized process, instead of using the approximate concentration of a single run, we take the average of the approximate concentration of 10 different runs. We represent the error as percentage and use the symbol $PE(g)$ (percentage error of g) for this metric.

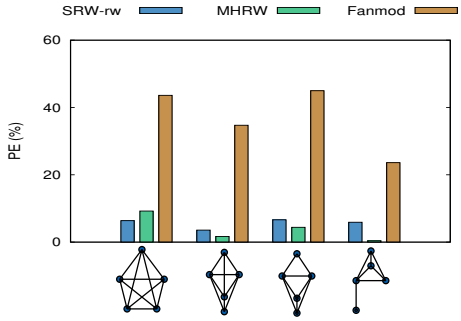
[1] `http://snap.stanford.edu/data/index.html` and `http://www-personal.umich.edu/~mejn/netdata`



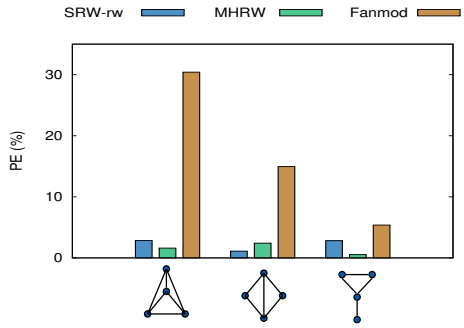
(a) $p = 5(10000)$ (yeast)



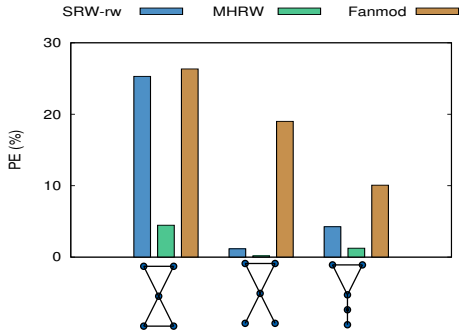
(b) $p = 6(10000)$ (Yeast)



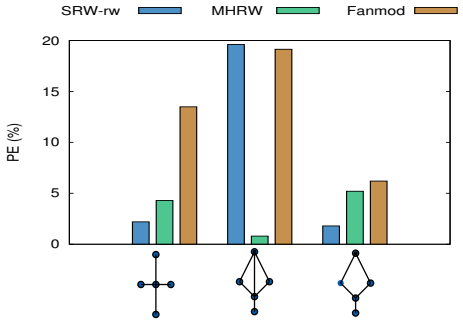
(c) $p = 5(10000)$ (Jazz)



(d) $p = 4(40000)$ (Ca-AstroPh)



(e) $p = 5(10000)$ (Ca-Hepth)



(f) $p = 5(10000)$ (Ca-GrQC)

Fig. 9.3.: Comparison of percentage error value for various methods. The dataset name, motif size, and the number of samples (in parenthesis) are given in figure sub-title.

9.4.1 Error Comparison

We compare the error percentage (PE) of various topologies using SRW-RW, MHRW, and FANMOD algorithms on all the datasets for different size values (p). Instead of showing the PE for all the topologies, we only show it for the topologies that are likely to be motifs, i.e., for these topologies, the $\widehat{z}_G(g)$ value in Equation 9.3 is the highest among all the topologies. For this experiment, we fixed the number of samples to 10000 for all of the experiments except for the experiment of Ca-AstroPh dataset, where we use 40000 samples.

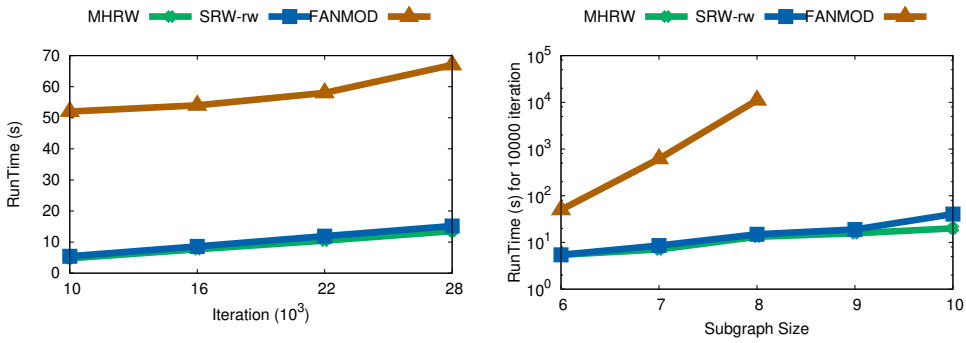
For all the datasets, we see that our methods are significantly better than the FANMOD method based on the PE metric. Specifically, the performance gap between our method and FANMOD is very high for the Ca-AstroPh dataset, which is the largest among all our datasets. The performance of SRW-RW and MHRW are comparable. However, we observe that for topologies for which the concentration is high, MHRW's approximation is better than SRW-RW. On the other hand for graphs for which the concentration is small (see the dense topologies in Figure 9.3b), SRW-RW's approximation is better than MHRW. There are a few occasions where the PE of SRW-RW are as bad as FANMOD; nevertheless, the plots clearly demonstrate the superiority of Markov Chain based techniques over FANMOD in terms of percentage error.

9.4.2 Runtime Comparison

The runtime performance comparison of our methods with FANMOD is shown in Table 9.2. Here, we have fixed the sample count to 10000 for all the methods. To highlight the poor scalability of FANMOD with the size of the motif, we show some of the numbers in bold font. If we carefully observe the table we can see that as the size increases by unity the runtime of FANMOD increases more than 10 times. For the Ca-AstroPh dataset which is the densest, for generating 10000 samples, FANMOD takes 180s, on the other hand both of our methods take about 5 seconds only. For

Table 9.2.: Runtime comparison of our methods with FANMOD.

Dataset	Motif Size	MHRW (s)	SRW-rw (s)	FANMOD (s)
Yeast	5	2.73	3.13	2.73
	6	4.78	5.43	**50**
Jazz	5	5.08	5.71	3.45
	6	9.68	10.92	**52**
Ca-GrQC	3	0.79	1.06	0.026
	4	2.11	2.79	0.275
	5	7.03	10.53	2.79
	6	25.36	32.30	**34**
Ca-Hepth	3	0.60	0.75	0.43
	4	1.43	1.72	0.413
	5	3.03	3.30	5.37
	6	4.98	5.13	**70.41**
Ca-Astroph	3	3.20	4.48	3.35
	4	7.90	9.80	**180.38**



(a) Sample size vs Runtime (b) Subgraph Size vs Runtime

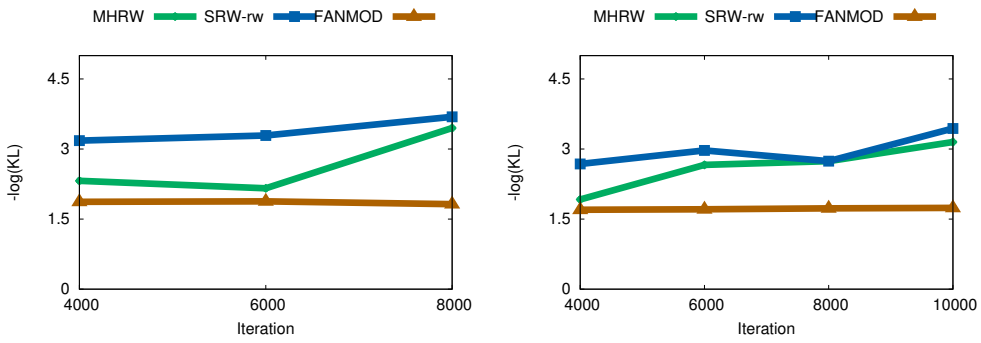
Fig. 9.4.: Runtime performance for different sample sizes and for different subgraph sizes.

this metric also, the performance gap between our methods and FANMOD increases as the dataset or the motif size increases.

We also show the runtime performance of the algorithms with the increasing number of samples in Figure 9.4a for yeast dataset and for subgraph size 5. The time increases mostly linearly for all the datasets; however, both of our methods have much smaller runtime than FANMOD. We also compare the runtime performance of the algorithms for motif sizes from 6 to 10. The result is shown in Figure 9.4b (note that y-axis is in logarithm scale). It is clear from the plot that our methods scale well with the increasing subgraph size. But, for FANMOD the runtime grows exponentially with the subgraph size; for example, to sample 10000 graphs from the yeast dataset, for subgraph size 7 and 8, it takes 616 seconds and 3 hours respectively. On the other hand, for size 8 our methods sample identical number of graphs in only 50 seconds. Also note that, FANMOD runs only for subgraph size up to 8.

9.4.3 Convergence Comparison

In this experiment, we study the convergence using the negative log (KL) metric by varying the number of samples. Figure 9.5a and 9.5b show that as we increase the number of samples both the Markov chain based techniques approximate the



(a) Iteration vs KL (Size 5) (ca-Hepth)

(b) Iteration vs KL (Size 6) (ca-Hepth)

Fig. 9.5.: Comparison of convergence trend of our methods with FANMOD using KL Divergence.

concentration distribution more accurately (increasing value of $-\log(KL)$), on the other hand, for FANMOD the curve is almost flat, i.e. with an increasing number of samples FANMOD does not converge to the true concentration.

9.5 Chapter Summary

In this paper, we propose two methods MHRW, and SRW-RW for approximating the concentration of p-subgraphs in a host network for any given value of p. Our experimental results demonstrates that both of our proposed methods are significantly faster than the best of the existing methods. Moreover, our methods do not require full access over the networks. This makes our method useful for very large network (such as, Web) which can only be crawled.

10. ACTS: EXTRACTING ANDROID APP TOPOLOGICAL SIGNATURE THROUGH GRAPHLET SAMPLING

10.1 Introduction

Rising trends in mobile systems, e.g., the wearable devices, the medical devices and the intelligent vehicle systems, are setup on Android platforms following the big success of it on smartphone market. Since Android applications are specifically designed to have as few implementation dependencies as possible, Android is believed to be adaptive to the new market and dominate the mobile distributed environment soon.

As the use of Android continues to grow, so does the threat of malware. Malicious behaviors observed in such malware include the theft of private information stored on the device, device fingerprinting, abusing premium service, and rooting the device as a backdoor for further attacks [99]. Detecting such malware is a critical task for the security research community.

It is observed that variants of malware form families through code sharing and their common lineage [99]. Therefore, instead of identifying individual malware and extracting a signature from it, we can identify the commonality within the same malware family and generate signatures that capture such commonality. Recently, various machine learning/data mining (i.e., pattern mining) techniques are applied to detect Android malware [100–105] or closely related tasks such as identifying repackaged apps [106, 107]. Beyond the common pattern mining framework, these works differ significantly in their selection and construction of features, their quantification/metrication of such features, their choice of pattern mining algorithms, and, in

totality of these fine points of design, their applicability, robustness, and efficiency in detecting malware.

A number of different app representations have been studied for malware detection. For example, Yamaguchi et al. [101] propose a compact representation of source code, the code property graph, that combines abstract syntax trees, control flow graphs, and program dependence graphs. Other approaches do not require the source, but instead focusing on features at different abstract levels: from the low-level platform opcode level [104], through the intermediate function call [100] and Android framework API [103] level, to the high semantic level that includes features such as network addresses and Android specific artifacts such as permission and Intents [102]. Yet, other works formulate malware detection as different pattern mining tasks such as frequent subgraph mining [105].

Due to the availability of off-the-shelf obfuscation solutions (such as the free ProGuard [108] and the commercial DexGuard [109]) and the growing number of Android apps, it is critical for any proposed malware detection algorithm to be robust and efficient. Our first step towards robustness is to extract from the app under investigation its function call graph (FCG) [100], in which each vertex represents a Java method and each edge represents a method invocation. We concur with [100] that FCG is at a proper abstraction level for detecting malware: In addition to the non-essential transformations mentioned above, it is also immune to, for example, both lower-level opcode/instruction obfuscation or higher-level content encryption.

Based on the extracted FCG, we propose an efficient and robust Android app signature that faithfully captures the invocator-invocatee relationship between several functions, i.e., the topology of local neighborhoods on the FCG. Instead of using vertices and edges (or extension to 1-hop neighborhoods [100]) on the FCG "as is," we leverage recent advances in graph mining to efficiently sample *graphlets* [111, 224] on the FCG. Graphlets are small (e.g., less than 6), connected, vertex-induced embedded subgraphs in an underlying graph, which is the FCG in our case. In the spectrum of purely local (e.g., individual vertices/edges and simple metrics such as degrees) and

fully global (e.g., betweenness centrality [225]) scope of the FCG, our graphlet-based signature takes a unique position: It faithfully captures local topological information at a fine-grained granularity without exponentially inflating the state space.

Given these characteristics, we call our graphlet-based signature a *topological* signature and, accordingly, name our method ACTS (App topologiCal signature through graphleT Sampling). In our experiments, ACTS achieves a cross-validated accuracy as high as 87.9% . In comparison, the same method with a purely local feature (i.e., degree frequency distribution (DFD) [226]) has an average cross-validated accuracy of 75%. Since ACTS only uses structural features, which are orthogonal to semantic features such as bytecode-based vertex typing, it is expected that combining them would give a greater improvement in malware detection accuracy than combining non-orthogonal semantic features.

In summary, our contributions are:

- We propose a novel topological signature for Android apps that fully captures the invocator-invocatee relationship in an app's FCG, which is otherwise lost in a global topological metric such as betweenness centrality [225], without exponentially inflating the state space as in n-hop neighborhoods with $n \geq 3$.
- By leveraging recent advances in graphlet sampling, we make the generation of our proposed topological signature practically efficient without sacrificing its robustness.
- With experiments on real malware/benign app samples, we demonstrate that local topological information captured by our method alone can achieve a high malware detection accuracy, which can be further improved by incorporating (orthogonal) semantic features.

In the rest of the paper, after the preliminaries (Section 10.2), we present our method (Section 10.3) and experiment results on real malware/benign app samples (Section 10.4). We then reflect on our method (Section 10.5) and conclude with a brief review of related works (Section 10.6).

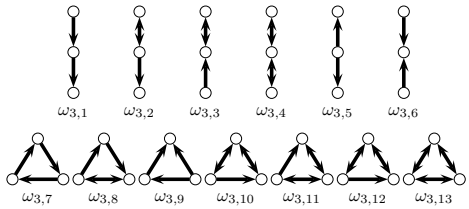


Fig. 10.1.: The 13 unique 3-graphlet types $\omega_{3,i}$ $(i = 1, 2, \ldots, 13)$.

10.2 Preliminaries

10.2.1 Function Call Graph

Function call graph (FCG) is a graph model for functions and their invocation relationship, in which vertices represent functions and a directed edge from vertex v_1 to v_2 represents that v_1 invokes v_2. For an Android app, functions are Java methods, and their invocation relationship can be statically extracted from Java bytecode by searching for the invocation-related opcodes, i.e., invoke-*.

10.2.2 Graphlets

Pržulj et al. [110] first consider a complete set of local graph topologies with 3, 4, and 5 vertices and name them graphlets[1] in their work on characterizing biological networks. Formally, given a graph G, graphlets of G are small, connected, non-isomorphic, and vertex-induced subgraphs of G. Although earlier works [110,111,224] on graphlets focus on undirected graphs, we consider directed graphlets to preserve the inherent directionality of FCGs.

Figure 10.1 enumerates all the 13 unique types of (directed) graphlets $\omega_{3,i}$[2] $(i = 1, 2, \ldots, 13)$ with 3 vertices (the 3-graphlets): They are pair-wise non-isomorphic. These graphlet types do not appear equally likely in an FCG. For instance, although there are many cases in which a function invokes two others ($\omega_{3,5}$) or two different

[1]Graphlet is also used to refer wavelet decomposition of graphs [227], which is an unrelated concept to what we use in this work.

[2]The unique types of n-graphlets are enumerated as $\omega_{n,1}, \omega_{n,2}, \ldots, \omega_{n,N(n)}$, with $N(n)$ being the number of unique types for n-graphlets.

functions invoke the same one ($\omega_{3,6}$), 3 mutually recursive functions ($\omega_{3,13}$) are rare. Later, we will discuss how we use this observation to improve the performance of our method (Section 10.3.3).

For vertices 4, 5, and 6, the number of graphlet types are 199, 9,364, and 1,530,843, respectively [228]. We focus on graphlets with less than 6 vertices in this work because larger graphlet types require extra computations but provide little value in capturing the structure of FCG. Figure 10.2 illustrates our running example: A 4-graphlet g (the grey vertices and their induced edges) embedded in a 6-vertex graph G.

10.2.3 Graphlet Frequency Distribution (GFD)

Graphlet frequency distribution (GFD) of a graph G is the probability distribution of the frequencies of the different graphlet types in G. For instance, since the number of 3-graphlets in a (finite) FCG G is finite, we can, in principle, enumerate all embedded graphlets in G and, for each such embedded graphlet g, identify g with one of the 13 graphlet types in Figure 10.1. At the end of the enumeration, suppose the count (i.e., the frequency) of graphlet type $\omega_{3,i}$ is $f_{3,i}$ ($i \in \{1, 2, \ldots, 13\}$), the frequency distribution density $d_{3,i}$ at $\omega_{3,i}$ is $f_{3,i}/\sum_{i=1}^{13} f_{3,i}$. We call the vector $(d_{3,1}, d_{3,2}, \ldots, d_{3,13})$ the *3-graphlet frequency distribution* (3-GFD) of G. We can compute n-GFD for any n with the same procedure, and concatenate several n-GFDs with different n into a single vector. We can call the concatenated vector a GFD of G if there is no confusion on its constituents.

The above procedure only works in principle. In practice, the fast growing number of apps, the size of real apps' FCGs, and the combined computation complexity of graphlet enumeration and identify graphlet types make the enumeration-and-count procedure impractical to use. Nevertheless, GFD is a step forward towards our goal: It is a metrication from the (combinatorial) graphlet space into the (metric) Euclidean space, where we can apply pattern learning techniques to detect malware. In other

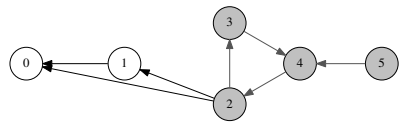


Fig. 10.2.: Our running example: A 4-graphlet g (the grey vertices and their induced edges) embedded in a 6-vertex graph G.

words, GFD preserves the topological information of local neighborhoods in an FCG. Later, after giving a high-level overview of our method (Section 10.3.1), we will focus on how to estimate GFD efficiently (Section 10.3.2).

10.2.4 Metropolis-Hastings (M-H) Algorithm

Markov Chain Monte Carlo (MCMC) [229] is a class of algorithms for sampling from a probability distribution. Given an intended sampling distribution $p(x)$ over a sample space X, the idea behind general MCMC methods (in which the M-H algorithm is a specific method) is to construct a Markov chain over X whose stationary distribution equals to $p(x)$: After the Markov chain mixes (i.e., reaches its stationary distribution and, hence, "forgets" where it begins), the subsequently visited states of the chain can be used as samples from the intended distribution $P(x)$.

Metropolis-Hastings (M-H) algorithm [230] is a specific MCMC method that we use for estimating GFD (Section 10.3.2). In the M-H algorithm, the transition between two consecutive states x and x' in the chain consists of two stages: proposals and acceptance/rejection. Correspondingly, there is a proposal distribution $q(x'|x)$ (the probability of *proposing* x' as the next state given the current state x) and an acceptance distribution $a(x'|x) = \min(1, A(x'|x))$ (the probability of *accepting* x' as the next state given the current state x), in which:

$$A(x'|x) = \frac{p(x')q(x|x')}{p(x)q(x'|x)}. \tag{10.1}$$

Intuitively, for each iteration of the sampling process, we first randomly pick x' with a probability of $q(x'|x)$, and then either accept x' (by sampling x') with a probability of $a(x'|x)$ or reject x' (by sampling x again) with a probability of $1 - a(x'|x)$.

10.3 Method

In this section, after a brief overview of our method (Section 10.3.1), we zoom in on two technical points: Efficient GFD estimation (Section 10.3.2) and FCG-specific GFD dimension reduction heuristics (10.3.3) that distinguish our method.

10.3.1 Overview

Given an Android app's APK (Android PacKage) binary package, we:

- extract an FCG from the APK,
- estimate the GFD of the FCG (Section 10.3.2), and
- project the estimated GFD to a lower dimensional space to reduce the GFD's dimensions (Section 10.3.3).

The projected GFD, which is a vector, is a signature of the app. To stress that this signature preserves detailed topological information on an app's FCG, we call it the *topological signature* (TS) of the app.

Given a pool of both malware and benign app samples, we train a classifier on their TSs to detect malware: If the TS of an app is classified as a malware, the app is flagged as malware.

10.3.2 Efficient GFD Estimation

Suppose we have a *uniform sampler* of the FCG, we can approximate the whole FCG's GFD with our samples' GFD. The more samples we take, the closer the approximation is. Given the large sample space and the (relatively small) number of

bins (i.e., unique graphlet types) for n-graphlets with $n < 6$, we only need to sample a tiny fraction of the sample space to get a close approximation.

This apparently solve the GFD estimation problem. However, the real problem is that we need to *uniformly* sample graphlets from the FCG *without enumerating the sample space*. Fortunately, two recent advances on graph mining, MHRW [21] and GUISE [111], show that GFD can be estimated without enumerating all graphlets. Inspired by these works, we use MCMC to sample the directed FCG.

Sample space and intended distribution

Since our goal is to uniformly sample from all the embedded graphlets in the FCG:

- The sample space X consists of all the embedded graphlets in the FCG.
- The intended distribution $p(x)$ over X is the uniform distributions, i.e., $p(x) = p(x')$ for any $x, x' \in X$.

Suppose we have just sampled graphlet g in the sampling process, the M-H algorithm (Section 10.2.4) says that, if we propose to sample graphlet g' next with a probability of $q(g'|g)$, an acceptance probability of $a(g'|g) = \min(1, A(g'|g))$ (in which $A(g'|g)$ is defined by Equation (10.1)) will eventually lead to a sampling process that have the desired sampling distribution $p(x)$.

FCG-induced Graphlet Graph and Graphlet Neighboring Relationship

To define the proposal distribution $q(x'|x)$, we consider the *FCG-induced graphlet graph* $\mathcal{G}_G$ of the FCG G. The FCG-induced graphlet graph $\mathcal{G}_G$ is an undirected graph with vertices being all the embedded graphlets in the FCG, and edges defined by the *graphlet neighboring relationship* between the vertices. The graphlet neighboring relationship is a symmetric relationship between two graphlet embeddings g_1 and g_2 in the FCG: g_1 and g_2 are graphlet neighbors if and only if they differ by share all but one vertex. In particular, self-neighboring is excluded by this definition because there is no vertex difference, which is required by the definition. Since graphlets on G and

vertices on $\mathcal{G}_G$ have a one-to-one map, we identify a graphlet g on G with the vertex on $\mathcal{G}_G$ that it maps to, and also denote that vertex with g if there is no confusion in the context.

For example, in Figure 10.2, g's neighbors on $\mathcal{G}_G$ are[3] all the 3-graphlets (e.g., $\{v_2, v_3, v_4\}$, $\{v_3, v_4, v_5\}$, etc.), 4-graphlets (e.g., $\{v_1, v_2, v_3, v_4\}$, $\{v_0, v_2, v_4, v_5\}$, etc.), and 5-graphlets ($\{v_1, v_2, v_3, v_4, v_5\}$ and $\{v_0, v_2, v_3, v_4, v_5\}$) that share all but one vertex with it. Conversely, 1. $\{v_1, v_2, v_3\}$ is not a neighbor of g because it does not contain both v_4 and v_5, which are in g; 2. $\{v_0, v_1, v_2, v_3\}$ is not a neighbor of g because it does not contain g's vertices v_4 and v_5 (and g does not contain its vertices v_0 and v_1); 3. $\{v_0, v_1, \ldots, v_5\}$ is not a neighbor of g because g does not contain its vertices v_0 and v_1.

The significance of the graphlet neighboring relationship on $\mathcal{G}_G$ is that it can be efficiently generated by *local information* on the FCG G *without enumerating the whole* G. Specifically, given an embedded graphlet g of G, the neighbors of g on $\mathcal{G}_G$ can be generated by removing, changing, or adding exactly one vertex in g. Hence, we can efficiently compute the degree d_g of g in $\mathcal{G}_G$ by generating and counting g's neighbors.

Proposal and Acceptance Distributions

Let $d(g)$ and $N(g)$ be graphlet g's degree and neighbors in $\mathcal{G}_G$, respectively. Suppose the last graphlet we have sampled is g, our proposal strategy $q(g'|g)$ is to uniformly sample one of its neighbors in $\mathcal{G}_G$, i.e.,

$$q(g'|g) = \begin{cases} \frac{1}{d_g} & \text{if } g' \in N(g), \\ 0 & \text{otherwise.} \end{cases} \tag{10.2}$$

Since d_g can be efficiently computed without enumerating the graph (see above), $q(g'|g)$ can also be efficiently computed since it only requires computing d_g.

[3]Given that graphlets are vertex-induced subgraphs, we use a vertex set to represent the (unique) embedded graphlet having those vertices here.

By Equations (10.1) and (10.2), the resulting acceptance strategy $a(g'|g)$ is:

$$a(g'|g) = \begin{cases} \min(1, \frac{d_g}{d_{g'}}) & \text{if } g' \in N(g), \\ 0 & \text{otherwise.} \end{cases} \tag{10.3}$$

By Equations (10.2) and (10.3), the probability $s(g'|g)$ of sampling g' next given the current sample g is:

$$s(g'|g) = \begin{cases} \min(\frac{1}{d_g}, \frac{1}{d_{g'}}) & g' \in N(g), \\ 1 - \sum_{h \in N(g)} \min(\frac{1}{d_g}, \frac{1}{d_h}) & g' = g, \\ 0 & \text{otherwise.} \end{cases} \tag{10.4}$$

The intuition behind the sampling strategy in Equation (10.4) can be understood in the following two cases.

Case 1. If g is a graphlet that has the highest degree among its neighbors in $\mathcal{G}_G$, i.e., $d_g \geq d_{g'}$ for any $g' \in N(g)$, then $\min(1/d_g, 1/d_{g'}) = 1/d_g$ and, hence, by Equation (10.4), $s(g|g) = 1 - d_g(\frac{1}{d_g}) = 1 - 1 = 0$, i.e., the next sample will *not* be g but one of its neighbors.

Case 2. If g is a graphlet with a relatively low degree among its neighbors in $\mathcal{G}_G$, $s(g'|g)$ in Equation (10.4) will be greater than 0. The greater the degree differences are, the greater $s(g'|g)$ will be. In an extreme case in which g has a single neighbor g' with a degree of 100 (i.e., $d_g = 1$ and $d_{g'} = 100$), $s(g'|g) = 0.01$ and $s(g|g) = 0.99$: If the current sample is g, 99 out of 100 times, the next sample will still be g.

In other words, the sampling process (i.e., the consecutive states of the Markov chain) is more eager to *move away from* the more popular graphlets (i.e., the ones with higher degrees in $\mathcal{G}_G$) and to *stay at* the less popular ones: The former has a better chance than the latter of being revisited later. This results in a fair (i.e., uniform) sampling of all the embedded graphlets in the FCG G.

Algorithm 9: ESTIMATE-GFD.

Input :
- G: the FCG
- t: number of iterations

[1] $g \leftarrow$ a random (initial) graphlet ;
[2] $f_c \leftarrow$ **NEXT-SAMPLE** (G, g, t) ;
[3] **for** $c \in \mathcal{C}$ **do**
[4] $\quad d_c \leftarrow f_c / \sum_{c \in \mathcal{C}} f_c$;
[5] **return** d_c ;

Algorithm 10: NEXT-SAMPLE.

Input :
- G: the FCG
- g: current graphlet sample
- k: remaining iterations

[1] $N(g) \leftarrow g$'s neighbors in $\mathcal{G}_G$;
[2] choose a $g' \in N(g)$ with an equal probability of $1/d_g$;
[3] $a \leftarrow$ a number uniformly sampled from $[0, 1]$;
[4] **if** $a \leq \min(1, d_g/d_{g'})$ **then**
[5] $\quad g \leftarrow g'$;
[6] $\quad c \leftarrow \mathcal{C}(g)$;
[7] $\quad f_c \leftarrow f_c + 1$;
[8] $\quad$ **if** $k > 0$ **then**
[9] $\quad\quad$ **NEXT-SAMPLE** $(G, g, k-1)$;

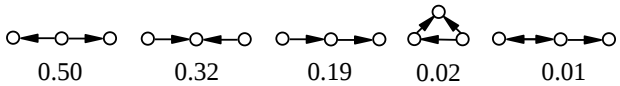


Fig. 10.3.: The 5 3-graphlet types that have a greater-than-2% frequency density in the GFD of at least one app in our experiment, sorted by their average frequency density across all malware/benign app samples in our experiment.

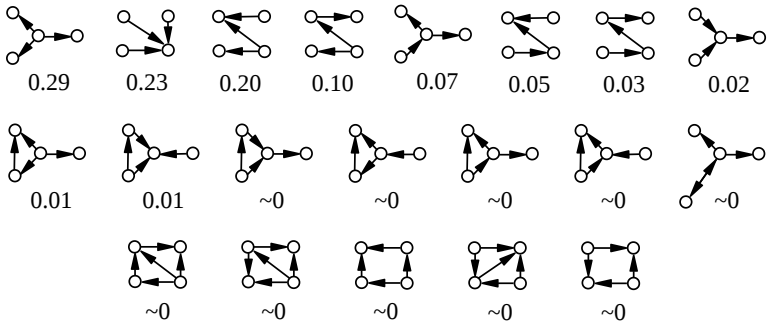


Fig. 10.4.: The 20 4-graphlet types that have a greater-than-2% frequency density in the GFD of at least one app in our experiment, sorted by their average frequency density across all malware/benign app samples in our experiment.

GFD estimation algorithm

Finally, we estimate the GFD for the FCG G from t samples by evaluating ESTIMATE-GFD(G, t) in Algorithm 9. In our experiment, we evaluate multiple t and choose $100,000$ for having both low variance in the sampling result and acceptable efficiency. Note that, given the average size of an FCG G (thousands of vertices) and, hence, the sample space $\mathcal{G}_G$ (for a $1,000$-vertex G, $\mathcal{G}_G$ has a worst-case size of $O(1,000^3)$), $100,000$ iterations are quite small. Indeed, for the largest app in our dataset (the Facebook app, with $47,539$ vertices and $77,900$ edges), ESTIMATE-GFD(G, T) for $T = 100,000$ only takes only about 34 seconds on our desktop workstation with high convergence across multiple runs.

10.3.3 FCG-specific GFD Dimension Reduction Heuristics

The curse of dimensionality [231] plagues many machine learning tasks. Theoretically, by confining the n-graphlets we sampled to $n \in \{3, 4, 5\}$, the GFD vectors we obtain from Algorithm 9 are of $9,576$ ($13 + 199 + 9,364$; Section 10.2.2) dimensions. Reducing the dimensions of these vectors is desirable.

Fortunately, as briefly discussed in Section 10.2.2, not all graphlet types are equally likely to appear in a real FCG. Figures 10.3 and 10.4 show all 3-graphlet and 4-graphlet types that have more a greater-than-2% frequency density in the GFD of *at least one* of the (more than $1,400$) apps (including both malware and benign apps) in our experiment: There are 5 3-graphlet types, 20 4-graphlet types, and 71 5-graphlet types, respectively.

Note that, as we discuss in Section 10.2.2 and is verified here, graphlet types $\omega_{3,5}$ (outgoing invocations) and $\omega_{3,6}$ (incoming invocations) rank among the most frequent 3-graphlet types, while the mutually recursive type ($\omega_{3,13}$) is not. Moreover, except for a few cases of mutual recursion, loops among a few functions of are rare. This suggests that: 1. either inter-function loops have a long chain of invocations, 2. or most functions have a clear invocator-invocatee relationship that is not reciprocal.

These observations suggest that we can significantly cut down the dimensions of GFDs by projecting the GFD vectors onto the *most frequent dimensions*. Indeed, this is what we do in our method after obtaining the full-spectrum (i.e., $9,576$-dimensional) GFD estimation.

10.4 Experiment Results

10.4.1 Datasets

In our experiment, we use the benign app samples from PlayDrone [232] and use the malware samples from the Android Malware Genome Project (AMGP) [99].

For the benign app portion of our datasets, we download the dataset of PlayDrone. There are total 49000 benign samples in 9 different archives. To test the scalability and robust of our algorithm, we randomly and repeatedly choose sets from the PlayDrone and each set has thousands of benign samples. We also check the package name, the version code and the $MD5$ message of each sample to prevent the duplicate in it.

For the malware portion of our datasets, the AMGP lists $1,249$ malware samples of 49 families. The top 9 malware families that have over 40 samples are: DroidKungFu3 (303 samples), AnserverBot (185 samples), BaseBridge (118 samples), DroidKungFu4 (96 samples), Geinimi (69 samples), Pjapps (56 samples), KMin (52 samples), GoldDream (47 samples), and DroidDreamLight (46 samples).

10.4.2 Procedure

We first use Androguard [233] Android app reverse engineering toolkit to extract FCGs from the APK samples. Specifically, we use the androgexf.py script to extract a GEXF[4]-format file that encodes the Java methods and their invocation relations in the APK.

We implement our GFD estimation algorithm (Algorithm 9) to generate a GFD vector for all n-graphlet types for $n \in \{3, 4, 5\}$. The majority of dimensions have a frequency of 0; hence, we use the FCG-specific GFD dimension reduction heuristics (Section 10.3.3) to reduce these $9,576$-dimensional vectors to 96-dimensional ones, which only contain the dimensions that have a frequency density over 2% in at least one of the apps in our datasets. These 96-dimensional vectors are the topological signatures of their corresponding apps.

We then use the LIBSVM [234] support vector machine (SVM) library for classification; the details are mentioned below along with corresponding results.

[4]GEXF (Graph Exchange XML Format); `http://gexf.net/format/`.

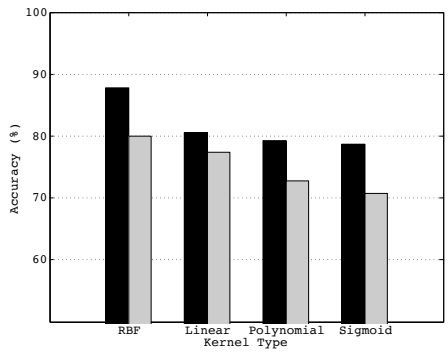


Fig. 10.5.: Malware detection accuracy of SVM-GFD (SVMs with GFD-based signature; dark) and SVM-DFD (SVMs with DFD-based signature; grey) using C-SVC (C-support vector classification) SVMs (support vector machines) with different kernels: RBF (radial basis function), linear, polynomial, and sigmoid.

Table 10.1.: Malware detection false positives (FPs) and false negatives (FNs): SVM-GFD vs. SVM-DFD with different kernels.

	RBF		linear	
	FP	FN	FP	FN
GFD	11.53%	12.78%	19.30%	19.55%
DFD	13.03%	27.07%	17.54%	27.82%
	polynomial		sigmoid	
	FP	FN	FP	FN
GFD	20.80%	20.55%	22.01%	20.55%
DFD	21.30%	33.08%	26.57%	32.08%

10.4.3 Results

Malware detection performance

To understand how the local-topology-preservation property of GFD helps in enhancing malware detection performance, we compare our method with another method in which both the (preceding) FCG extraction phase and (subsequent) learning phase are the same. The only difference is the feature we extract from FCG. Specifically, we use the degree frequency distribution (DFD) for comparison. In DFD,

Table 10.2.: Pair-wise malware family label accuracy (in percentage) of SVM-GFD (GFD) vs. SVM-DFD (DFD) with the linear kernel of the 8 malware families that have over 40 samples in the AMGP dataset: DroidKungFu3 (DKF3; 303 samples) AnserverBot (AB; 185 samples), BaseBridge (BB; 118 samples), DroidKungFu4 (DKF4; 96 samples), Pjapps (P; 56 samples), KMin (KM; 52 samples), GoldDream (GD; 47 samples), and DroidDreamLight (DDL; 46 samples). Since this matrix is symmetric, we only show the upper half of it.

	DKF3		AB		BB		DKF4		P		KM		GD		DDL		Benign	
	GFD	DFD	GFD	DFD	GFD	DFD	GFD	DFD	GFD	DFD	GFD	DFD	GFD	DFD	GFD	DFD	GFD	DFD
DKF3	-	-	92.6	60.09	71.63	67.77	75.94	71.08	84.40	77.38	85.35	78.08	86.82	78.96	86.82	79.14	76.73	71.78
AB	-	-	-	-	76.14	58.29	84.04	62.03	80.58	69.46	94.54	70.30	80.17	71.38	80.17	71.60	84.86	81.62
BB	-	-	-	-	-	-	77.78	53.90	68.18	61.94	83.72	62.88	72.29	64.09	72.29	64.34	81.25	56.25
DFK4	-	-	-	-	-	-	-	-	63.15	58.20	87.16	59.17	67.61	60.43	67.61	60.63	71.88	53.13
P	-	-	-	-	-	-	-	-	-	-	76.85	51.25	54.90	52.38	54.90	52.58	81.58	76.32
KM	-	-	-	-	-	-	-	-	-	-	-	-	89.80	51.31	86.73	51.58	72.12	60.58
GD	-	-	-	-	-	-	-	-	-	-	-	-	-	-	57.61	52.97	73.40	63.83
DDL	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	75.00	67.39
Benign	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

vertices with the same degree frequencies are binned together and counted. DFD is the probability distributions of element counts over these bins. In other words, the only difference between the two methods is whether local topology information of FCG is used in the subsequent learning phase: Our GFD-based method uses this information, while the DFD- based method does not.

For reasons that will be explained shortly, in this experiment, we randomly and repeatedly pick 1200 samples from the benign dataset to compare with the 1200 malware samples. In each comparison, we use the 10-fold cross-verification method, which means that each time 120 benign samples and 120 malware samples are randomly chosen as test set, other samples will be feed as training set and the result shows the overall average accuracy. Then we compare malware detection performance of SVMs with GFD-based signature (SVM-GFD) and SVMs with DFD-based signature (SVM-DFD) using all 4 built-in SVM kernel functions in LIBSVM: RBF (radial basis function: $e^{\gamma|u-v|^2}$), linear ($u' \cdot v$), polynomial ($(\gamma u' \cdot v)^3$), and sigmoid ($\tanh(\gamma u' \cdot v)$), in which u and v are feature vectors, $\gamma = 1/N$, and N is the feature vector dimension. Figure 10.5 shows the accuracy (the samples that are correctly labeled by the SVMs) comparison and Table 10.1 shows the detailed false positives/negatives (the samples that are incorrectly labeled by the SVMs).

The reason we use a 1:1 ratio between malware and benign app dataset is that a skewed dataset may give misleading performance results. Later in this part we will also present the influence of sample bias. In both Figure 10.5 and Table 10.1, the performance of SVM-GFD and SVM-DFD appear to be consistent across learning kernels. The high accuracy of the two algorithms implies that both of them could successfully capture topological features and the information is helpful to Android malware detection.

Comparing these two algorithms, SVM-GFD always give better results (by average 6% margin over the SVM-DFD algorithm, to over 80% accuracy). A recent study [102] on commercial anti-virus scanners' (AntiVir, AVG, BitDefender, ClamAV, ESET, F-Secure, Kaspersky, McAfee, Panda, Sophos) performance on the AMGP dataset

shows that, except for two outliers (23.68% and 1.12%), the commercial AV scanners have accuracy ranging from 84.23% to 98.90%. SVM-GFD attains a comparable accuracy of 87.85% on the full AMGP dataset using only the structural features without any semantic augmentation.

Figure 10.5 suggests that RBF kernel could give a better result than other three kernels both for SVM-GFD and SVM-DFD. SVM-GFD could perform a 78% or higher results on different kernels, while SVM-DFD show 70% accuracy when choosing polynomial or sigmoid kernel. So the SVM-GFD seems more robust than SVM-DFD. Table 10.1 shows that they have different performance among false positives (FP) and false negatives (FN). Because the dataset is 1:1 ratio, FP and FN achieving a nearly 1:1 ratio means the SVM could successfully divide the hyperplane. From Table 10.1 we can see that these two SVM methods tend to give high accuracy under the specified circumstances. And SVM-GFD often have a same FP or FN percentage as SVM-DFD while the other is much better. Also there is a trade-off between FP and FN. Taking the result of SVM-GFD with linear kernel as an example, it has the slightly higher FP than the SVM-DFD while the FP is relative low. In other words, comparing with SVM-DFD, SVM-GFD with linear kernel is an aggressive malware detector that misses less malware at the cost of flagging more benign apps as malicious. The mechanism behind this calls for further research.

Malware family labeling accuracy

To further understand the significance of capturing local topology in FCG for malware detection, we compare our SVM-GFD together with the SVM-DFD in their *malware family labeling accuracy* on the 8 malware families that have over 40 samples in the AMGP dataset (Section 10.4.1). Specifically, we take the family labels on the malware samples in the AMGP dataset as the ground truth, and compare the two methods' accuracy in assigning the correct family labels for the test data sets. We also compare each family with a dynamic benign dataset that has the same number

of samples as the malware family to show the accuracy of malware detection in one certain family. Table 10.2 shows the pair-wise malware family labeling accuracy of SVM-GFD vs. SVM-DFD.

SVM-GFD outperforms SVM-DFD in all pairs of malware families by a margin from 2.32% (P/Pjapps vs. DDL/DroidDreamLight) to 38.49% (KM/KMin vs. GD/GoldDream). The malware and benign software classification result in each family also shows SVM-GFD could achieve 3.24% (AB/AnserverBot vs. Benign) to 25% (BB/BaseBridge vs. Benign) higher performance. Note again, the additional local topological information on FCG captured by GFD, alone, takes the credit for this improvement in accuracy.

Given that we accept the manual labels as the ground truth, malware family labeling accuracy can be interpreted as a measure of how close two malware families are due to code sharing. For instance, in Table 10.2, on the row of DKF3/DroidKungFu3, DKF4/DroiKungFu4 has a low accuracy (75.94%). This lower labeling accuracy may derive from the higher similarity between DKF4 to DKF3 due to their common lineage in the DroidKungFu mega-family.

Performance against sample bias

In Section 10.4.3, we mention the peril of sample bias: If the ratio between positive and negative samples (i.e., benign app and malware samples) is skewed, even a naive strategy can give a misleadingly high accuracy without actually identifying malware from benign apps. In real-world malware detection, positive/negative samples rarely comes in evenly: It is highly likely we have to work with a skewed dataset.

Therefore, we study how SVM-GFD responds to sample bias. In order to avoid the influence of the dataset's size, we first fix the total number of benign and malicious softwares to 1000. Then we perturb the ratio between malware and benign app samples, and study the accuracy response of SVM-GFD/SVM-DFD with the linear kernel. The 10-fold cross-validation method is also employed in this experiment. Fig-

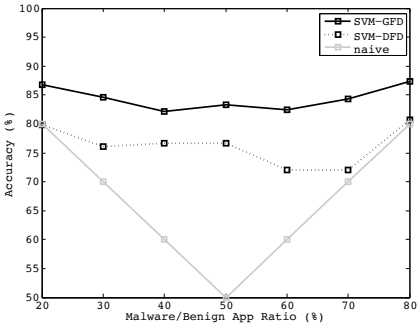


Fig. 10.6.: Accuracy response to different malware/benign-app ratios: SVM-GFD (full line) vs SVM-DFD (dotted line) vs the naive strategy. Percentage on the x axis is the ratio of malware over benign apps in the dataset; y axis is the malware detection accuracy.

ure 10.6 shows the results and indicates that SVM-GFD get higher accuracy among all kinds of malware and benign software combination. SVM-GFD has a variance of 4.1 while SVM-DFD has a variance of 11.4. We conclude that SVM-GFD is more robust than SVM-DFD against sample bias, especially when malware or benign software accounts a small proportion. When the ration between malware and benign software is 2:8, as mentioned above it is a common real-world situation, SVM-GFD outperforms 7% accuracy but SVM-DFD is just the same as the naive strategy.

Most frequent graphlets

To understand why malware detection accuracy improves only by replacing DFD with GFD, we study the most frequent graphlets that appear in benign apps and in malware. Figures 10.7 and 10.8 show the top 5 most frequent graphlet types for all benign app and malware samples in our datasets, respectively. "Most frequent" in this case means that these graphlet types have the highest average GFD densities in that category (benign app or malware).

It is interesting to note that, in addition to different average density values, the types of the most frequent graphlets are different. For example, while $\omega_{3,5}$ (outgoing invocations; Figure 10.1) ranks the first and $w_{3,6}$ (incoming invocations) ranks the third for malware, $\omega_{3,5}$ ranks the third and $\omega_{3,6}$ ranks the first for benign apps. In both cases, these two graphlet types have a graphlet frequency density gap of 0.1 or more between them. And it also happens when a function invokes/is invoked by 3 or more other functions. This suggests that incoming invocations to a same function is more frequent than outgoing invocations from a single function in benign apps, while the reverse is true for malware. The mechanism behind this calls for further research.

GFD estimation efficiency

In our experiment on a desktop workstation (8-core Intel Core i7-3820 CPU at 3.60GHz with 12GB RAM) with $100,000$ sampling iterations (at which point, the

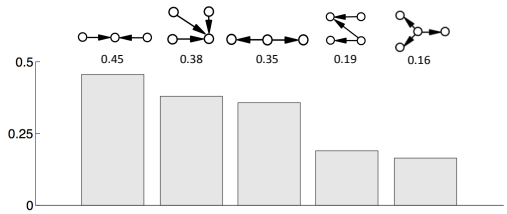


Fig. 10.7.: The top 5 most frequent graphlet types for benign apps, i.e., the ones that have the highest average graphlet frequency densities across all benign apps.

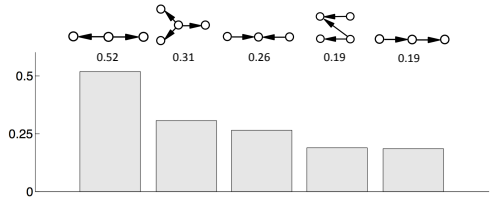


Fig. 10.8.: The top 5 most frequent graphlet types for malware, i.e., the ones that have the highest average graphlet frequency densities across all malware.

GFD estimation has already converged), our GFD estimation algorithm (Algorithm 9) takes less than 3 seconds to complete for many apps whose FCGs have less than $1,000$ vertices. For apps whose FCGs have less than $20,000$ vertices, GFD estimation takes an average of less than 10 seconds. For the most complex app in our data set, Facebook, which has $47,539$ vertices and $77,900$ edges, GFD estimation takes about 34 seconds on average with about 2 seconds variance. While the GFD estimation just takes seconds of work to analyze each single app, the total calculation time mainly depends on the size of the dataset. Because each apps and their FCGs are independent, the topological features extraction work is absolutely convenient for distributed computing system. Analyzing single extraction work, we note that GFD estimation is dominated by the generation of 1-hop neighborhood on $\mathcal{G}_G$ and the graphlet-type identification, which are independent to the size of the graph unless the graph is dense.

By contrast, the DFD calculation needs to traverse every edge and employ a sorting algorithm to the vertices. So it takes more time to do the DFD calculation especially on the complex networks. For instance, DFD calculation takes about 41 seconds for the Facebook application, 7 seconds longer than the GFD estimation. Therefore, GFD estimation, and hence ACTS, is practically efficient and accurate (Section 10.1).

10.5 Further Discussion

In order to verify the effectiveness of the graphlet-based analysis and to better understand why the topological features used in ACTS could result in good performance of benign/malicious software classification, we conducted a few case studies using dynamic analysis that based on semantic features.

In detail, we obtain the critical API calls with the help of online analysis tools, such as Andrubis and SanDroid. These critical calls are represented as edges in the FCG. And if a function invokes one or more times of the critical API calls, we label

the mapping vertex as a critical vertex. Instead of taking the full FCG graph into account, now we can just focus on the graphlets that contain the critical vertices.

Our experiment were taken on four APK files randomly chosen from four different malware families, TapSnake [235], SndApps [236], NickySpy [237] and LoveTrap [238]. The result shows that for each particular malware, its top-2 graphlets with critical vertices are always the same as the top-2 graphlets in GFD generated by ACTS. And obviously, they are different from the top-2 graphlets generated from the benign softwares. It implies that the most frequent graphlets of malware generated by ACTS in Section 10.4.3 always contain the critical API calls. ACTS catches the critical API calls by counting the graphlet distribution, which uses a different route from dynamic analysis but achieves the similar result in malware detection.

We also in-depth analyzed one application *com.typ3studios. airhorn* in the malware family SndApps [236]. There are just four critical graphlets that were obtained through dynamic analysis tools. After embedding the 3-node graphlets in 4&5-node graphlets, we find that there are only 2 kinds of 3-node graphlets that contain the critical API calls, $\omega_{3,5}$ and $\omega_{3,1}$ in Figure 10.1, while the possible 3-node graphlets has 13 types. Also, $\omega_{3,5}$ (outgoing invocations) is included but $w_{3,6}$ (incoming invocations) is not. It supports the result in Figure 10.8 of Section 10.4.3 that outgoing invocations to a same function is more frequent than incoming invocations from a single function in malware.

In the future, we plan to firmly combine ACTS with the dynamic analysis methods. Both the graphlet frequency and the semantic features will be analyzed to reveal the hidden mechanisms of malware.

10.6 Related Works

The present work follows a line of recent works [100–105] that apply advances in machine learning and data mining for Android malware detection. One main focus is on extracting learning features at the different app representation levels: Droid

Analytics [104] focuses on the low-level platform Dalvik opcode level; Gascon et al. [100] study function call graphs; DroidAPIMiner [103] extracts features from Android API calls; Drebin [102] extracts string features from multiple Android-specific sources, e.g., intent/permission requests, API calls, network addresses. Martinelli et al. [105] formulates the malware detection problem as a subgraph mining problem.

Pržulj et al [110] first propose and coin the term graphlet. Two recent advances on graph mining, MHRW [21] and GUISE [111], inspire our use of GFD as a robust and efficient topological signature for apps.

A related problem to malware detection is app repackaging, in which an app is transformed for a similar but different app through repackaging [106]. Repackaged apps are often seen on alternative Android app market, and is a major vector for carrying and propagating malware. Zhou et al. [107] propose a system called AppInk that applies watermarking to prevent app repackaging.

Tainting analysis (e.g., TaintDroid [112] and FlowDroid [113, 114]) and Android app analysis frameworks (e.g., DroidScope [115] and CopperDroid [116]) can be used to further analyze malware families identified by ACTS.

10.7 Chapter Summary

In this chapter, we propose GFD as a feature for Android malware detection and adapt recent advances in graph mining to make GFD estimation robust and efficient. We demonstrate that local topological information (captured by graphlets) is attributed to improvement in malware detection accuracy and efficiency. This provides a new angle to Android malware detection research, and suggests that finding structural features (e.g., graphlets) on a graphical representation of Android apps (e.g., the FCG) that situates between local and global scope as a fertile ground for future research.

11. FUTURE WORK AND CONCLUSION

Understanding the dynamics of temporal evolution of networks can help solve complex tasks involving social and interaction networks. Our latent representation techniques provide the ground-work for understanding the dynamics by providing a framework for learning to position nodes and edges in a temporally smooth way to a lower dimensional space. Thus the opportunity for the future works in the temporal network domain is abundant.

On the other hand, one important property of our sentence embedding learning model is that it encodes a sentence directly, and it considers neighboring sentences as atomic units. Apart from the improvements that we achieve in topic classification, clustering, and summarization tasks, this property makes our model quite efficient to train compared to *compositional* methods like encoder-decoder models (e.g., SDAE, Skip-Thought) that compose a sentence vector from the word vectors. This opens up a research question whether this simple intuition of modeling discourse sentences as atomic units could improve the discourse-informed translation [239]?

Also, methods for mining and calculating substructure statistics is another important avenue for research. Recently, there are a lot of works [240, 241] which provide algorithms for mining and using these structures, and statistics to solve tasks emerging from various application domains. In this thesis, we provide MCMC-based sampling strategy to mine and collect frequency statistics of different size substructures and showed one application in biology and another in the security domain. In the next few paragraphs, we discuss a few possible future directions from both the areas (latent representation and sampling) to explore and conclude the thesis.

Providing expressive models on the tasks of generating random graphs that makes no structural assumptions has become an important research direction in recent years [31, 242]. However, none of the existing generative models take into account

the temporal dimension. Therefore, providing expressive generative models that can handle the temporal dimension will significantly advance the field of network analysis and also provide more ground for analyzing the dynamics of an evolving network.

Recently, there are studies on how to set the time granularity of the temporal network in the neurology domain [243]. The research aims to investigate the effect of the time parameter in an evolving network. So, building new evolving network datasets from various domains and analyzing them would be another important research direction. Moreover, to validate the analysis is statistically significant, we may need to find methods to generate null evolving network which will allow us to distinguish the characteristics which are non-random (specific to the evolving network) and which are random (chance-level).

In our works for providing models to learn latent representation of nodes, we use retrofitted and linear transformation models to capture the temporal smoothness. For edges, we first concatenate the feature representation from different snapshot and use an encoder-decoder models to learn meaningful information data by compressing data in lower dimensional space. The models are shallow in terms of the number of parameters to learn. In both cases, high-capacity sequential deep models like RNN with or without attention [244–246] could be an important direction to explore for modeling more complex structure in the data.

In a recent work [247], substructure information has been proved to be important in capturing the structural information from the network in the latent representation. However, in our latent representation methods, we did not provide ways to collect and combine the information. Our retrofitted models would be particularly suited for the task which we left as a future work.

The models we propose for sentence embedding in Chapter 5 and 6, it would be interesting to see how our model compares with compositional models on sentiment classification task. However, this would require creating a new dataset of comments with sentence-level sentiment annotations. Creating such datasets can be an important research which will help to evaluate latent representation models which want to

encode topical information inside the representation. Moreover, proposing machine translation models which can capture discourse information is a promising direction to pursue.

In our work on sampling, we experimentally show how good our methods are in terms of mixing rate of random walk and suggested experimental methods for deciding on stopping the random-walk. However, we did not provide any theoretical analysis in the direction. It would be an important domain of research to bound the mixing rate [241] based on different higher-order neighborhood generation techniques and also providing the lower bound for number of iterations while performing the random-walk over the higher order substructure space.

To conclude, in this dissertation, we introduce several models for learning latent representation of network and textual units. We also provide sampling based techniques for mining and collecting substructure statistics from a single large network as well as from a set of networks. Finally, we provide two real-life application. One uses these substructures and their statistics to find functional motif from a set of biological network. Another uses the same for classifying android apps as a malignant or a benign app.

REFERENCES

REFERENCES

[1] Y. Bengio, A. Courville, and P. Vincent, "Representation learning: A review and new perspectives," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 8, pp. 1798–1828, 2013.

[2] E. C. Hansen, D. Battaglia, A. Spiegler, G. Deco, and V. K. Jirsa, "Functional connectivity dynamics: Modeling the switching behavior of the resting state," *NeuroImage*, vol. 105, pp. 525 – 535, 2015.

[3] P. Cui, X. Wang, J. Pei, and W. Zhu, "A survey on network embedding," *arXiv preprint arXiv:1711.08752*, 2017.

[4] R. Korolov, D. Lu, J. Wang, G. Zhou, C. Bonial, C. Voss, L. Kaplan, W. Wallace, J. Han, and H. Ji, "On predicting social unrest using social media," in *Proceedings of the IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*, 2016, pp. 89–95.

[5] C. Xu, Y. Bai, J. Bian, B. Gao, G. Wang, X. Liu, and T.-Y. Liu, "Rc-net: A general framework for incorporating knowledge into word representations," in *Proceedings of the ACM International Conference on Information and Knowledge Management (CIKM)*, 2014, pp. 1219–1228.

[6] M. Faruqui, J. Dodge, S. K. Jauhar, C. Dyer, E. Hovy, and N. A. Smith, "Retrofitting word vectors to semantic lexicons," in *Proceedings of the North American Chapter of the Association for Computational Linguistics (NAACL)*, 2015, pp. 1606–1615.

[7] M. Yu and M. Dredze, "Improving lexical embeddings with semantic knowledge," in *Proceedings of the Association for Computational Linguistics (ACL)*, 2014, pp. 545–550.

[8] G. A. Miller, "Wordnet: A lexical database for English," *Communications of the ACM*, vol. 38, no. 11, pp. 39–41, 1995.

[9] C. F. Baker, C. J. Fillmore, and J. B. Lowe, "The Berkeley framenet project," in *Proceedings of the Association for Computational Linguistics (ACL)*, 1998, pp. 86–90.

[10] J. R. Hobbs, "Coherence and Coreference," *Cognitive Science*, vol. 3, no. 1, pp. 67–90, 1979.

[11] M. Stede, *Discourse Processing*. Morgan & Claypool Publishers, 2011.

[12] B. Perozzi, R. Al-Rfou, and S. Skiena, "Deepwalk: Online learning of social representations," in *Proceedings of the ACM International Conference on Knowledge Discovery and Data Mining (KDD)*, 2014, pp. 701–710.

[13] A. Grover and J. Leskovec, "node2vec: Scalable feature learning for networks," in *Proceedings of the ACM International Conference on Knowledge Discovery and Data Mining (KDD)*, 2016, pp. 855–864.

[14] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei, "LINE: Large-scale information network embedding," in *Proceedings of the World Wide Web (WWW)*, 2015, pp. 1067–1077.

[15] X. Huang, J. Li, and X. Hu, "Label informed attributed network embedding," in *Proceedings of the ACM International Conference on Web Search and Data Mining (WSDM)*, 2017, pp. 731–739.

[16] T. K. Saha, T. Williams, M. A. Hasan, S. Joty, and N. K. Varberg, "Models for capturing temporal smoothness in evolving networks for learning latent representation of nodes," *arXiv preprint arXiv:1804.05816*, 2018.

[17] M. Rahman, T. K. Saha, M. A. Hasan, K. S. Xu, and C. K. Reddy, "Dylink2vec: Effective feature representation for link prediction in dynamic networks," *arXiv preprint arXiv:1804.05755*, 2018.

[18] T. K. Saha, S. Joty, N. Hassan, and M. A. Hasan, "Regularized and retrofitted models for learning sentence representation with context," in *Proceedings of the ACM International Conference on Information and Knowledge Management (CIKM)*, 2017, pp. 547–556.

[19] T. K. Saha, S. Joty, and M. A. Hasan, "Con-S2V: A joint learning framework for incorporating extra-sentential context into sen2vec," in *Proceedings of the Joint European Conference on Machine Learning and Knowledge Discovery in Databases (ECML-PKDD)*, 2017, pp. 753–769.

[20] T. K. Saha, A. Katebi, W. Dhifli, and M. A. Hasan, "Discovery of functional motifs from the interface region of oligomeric proteins using frequent subgraph mining," *IEEE/ACM Transactions on Computational Biology and Bioinformatics (TCBB)*, 2017.

[21] T. K. Saha and M. A. Hasan, "Finding network motifs using MCMC sampling," in *Proceedings of the 6th Workshop on Complex Networks (Complex Networks VI)*, 2015, pp. 13–24.

[22] W. Peng, T. Gao, D. Sisodia, T. K. Saha, F. Li, and M. Al Hasan, "ACTS: Extracting android app topological signature through graphlet sampling," in *Proceedings of the IEEE Conference on Communications and Network Security (CNS)*, 2016, pp. 37–45.

[23] D. Wang, P. Cui, and W. Zhu, "Structural deep network embedding," in *Proceedings of the ACM International Conference on Knowledge Discovery and Data Mining (KDD)*, 2016, pp. 1225–1234.

[24] A. R. Benson, D. F. Gleich, and J. Leskovec, "Higher-order organization of complex networks," *Science*, vol. 353, no. 6295, pp. 163–166, 2016.

[25] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2017.

[26] C. Tu, W. Zhang, Z. Liu, and M. Sun, "Max-margin Deepwalk: Discriminative learning of network representation," in *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 2016, pp. 3889–3895.

[27] T. Mikolov, Q. V. Le, and I. Sutskever, "Exploiting similarities among languages for machine translation," *arXiv preprint arXiv:1309.4168*, 2013.

[28] J. Qiu, Y. Dong, H. Ma, J. Li, K. Wang, and J. Tang, "Network embedding as matrix factorization: Unifying DeepWalk, LINE, PTE, and node2vec," in *Proceedings of the ACM International Conference on Web Search and Data Mining (WSDM)*, 2018, pp. 459–467.

[29] Z. Yang, W. W. Cohen, and R. Salakhutdinov, "Revisiting semi-supervised learning with graph embeddings," in *Proceedings of the International Conference on Machine Learning (ICML)*, 2016, pp. 40–48.

[30] J. Weston, F. Ratle, H. Mobahi, and R. Collobert, "Deep learning via semi-supervised embedding," in *Neural Networks: Tricks of the Trade*, 2012, pp. 639–655.

[31] J. You, R. Ying, X. Ren, W. L. Hamilton, and J. Leskovec, "GraphRNN: A deep generative model for graphs," *arXiv preprint arXiv:1802.08773*, 2018.

[32] J. B. Tenenbaum, V. De Silva, and J. C. Langford, "A global geometric framework for nonlinear dimensionality reduction," *Science*, vol. 290, no. 5500, pp. 2319–2323, 2000.

[33] S. T. Roweis and L. K. Saul, "Nonlinear dimensionality reduction by locally linear embedding," *Science*, vol. 290, no. 5500, pp. 2323–2326, 2000.

[34] W. L. Hamilton, R. Ying, and J. Leskovec, "Representation learning on graphs: Methods and Applications," *arXiv preprint arXiv:1709.05584*, 2017.

[35] L. Zhu, D. Guo, J. Yin, G. Ver Steeg, and A. Galstyan, "Scalable temporal latent space inference for link prediction in dynamic social networks," *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, vol. 28, no. 10, pp. 2765–2777, 2016.

[36] Y. Chi, X. Song, D. Zhou, K. Hino, and B. L. Tseng, "Evolutionary spectral clustering by incorporating temporal smoothness," in *Proceedings of the International Conference on Knowledge Discovery and Data Mining (KDD)*, 2007, pp. 153–162.

[37] İ. Güneş, Ş. Gündüz-Öğüdücü, and Z. Çataltepe, "Link prediction using time series of neighborhood-based node similarity scores," *Data Mining and Knowledge Discovery (DMKD)*, vol. 30, no. 1, pp. 147–180, 2015.

[38] T. Tylenda, R. Angelova, and S. Bedathur, "Towards time-aware link prediction in evolving social networks," in *Proceedings of the ACM Workshop on Social Network Mining and Analysis*, 2009, pp. 9:1–9:10.

[39] M. Rahman and M. A. Hasan, "Link prediction in dynamic networks using graphlet," in *Proceedings of the Machine Learning and Knowledge Discovery in Databases (ECML-PKDD)*, 2016, pp. 394–409.

[40] X. Li, N. Du, H. Li, K. Li, J. Gao, and A. Zhang, "A deep learning approach to link prediction in dynamic networks," in *Proceedings of the SIAM International Conference on Data Mining (SDM)*, 2014, pp. 289–297.

[41] D. M. Dunlavy, T. G. Kolda, and E. Acar, "Temporal link prediction using matrix and tensor factorizations," *ACM Transactions on Knowledge Discovery and Database (TKDD)*, vol. 5, no. 2, pp. 10:1–10:27, 2011.

[42] K. S. Xu, "Stochastic block transition models for dynamic networks," in *Proceedings of International Conference on Artificial Intelligence and Statistics (AISTAT)*, 2015, pp. 1079–1087.

[43] S. Cao, W. Lu, and Q. Xu, "Grarep: Learning graph representations with global structural information," in *Proceedings of the ACM International Conference on Information and Knowledge Management (CIKM)*, 2015, pp. 891–900.

[44] Q. V. Le and T. Mikolov, "Distributed representations of sentences and documents," in *Proceedings of the International Conference on Machine Learning (ICML)*, vol. 14, 2014, pp. 1188–1196.

[45] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation." in *Proceedings of the Empirical Methods in Natural Language Processing (EMNLP)*, vol. 14, 2014, pp. 1532–1543.

[46] L. Yang, X. Chen, Z. Liu, and M. Sun, "Improving word representations with document labels," *IEEE/ACM Transactions on Audio, Speech & Language Processing*, vol. 25, no. 4, pp. 863–870, 2017.

[47] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Proceedings of the Advances in Neural Information Processing Systems (NIPS)*, 2013, pp. 3111–3119.

[48] F. Hill, K. Cho, and A. Korhonen, "Learning distributed representations of sentences from unlabelled data," in *Proceedings of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*, 2016, pp. 1367–1377.

[49] T. Kenter, A. Borisov, and M. de Rijke, "Siamese CBOW: Optimizing word embeddings for sentence representations," in *Proceedings of the Association for Computational Linguistics (ACL)*, 2016, pp. 7–12.

[50] R. Socher, A. Perelygin, J. Wu, J. Chuang, C. D. Manning, A. Y. Ng, and C. Potts, "Recursive deep models for semantic compositionality over a sentiment treebank," in *Proceedings of the Empirical Methods in Natural Language Processing (EMNLP)*, 2013, pp. 1631–1642.

[51] K. Cho, B. van Merriënboer, Ç. Gülçehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using RNN encoder–decoder for statistical machine translation," in *Proceedings of the Empirical Methods in Natural Language Processing (EMNLP)*, 2014, pp. 1724–1734.

[52] R. Socher, J. Bauer, C. D. Manning, and N. Andrew Y., "Parsing with compositional vector grammars," in *Proceedings of the Association for Computational Linguistics (ACL)*, 2013, pp. 455–465.

[53] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *arXiv preprint arXiv:1301.3781*, 2013.

[54] R. Kiros, Y. Zhu, R. Salakhutdinov, R. S. Zemel, A. Torralba, R. Urtasun, and S. Fidler, "Skip-thought vectors," in *Proceedings of the International Conference on Neural Information Processing Systems (NIPS)*, 2015, pp. 3294–3302.

[55] D. Cook and L. Holder, "Substructure discovery using minimal description length and background knowledge," *Journal of Artificial Intelligence Research*, vol. 1, pp. 231–255, 1994.

[56] A. Inokuchi, T. Washio, and H. Motoda, "An apriori-based algorithm for mining frequent substructures from graph data," in *Proceedings of the Principles and Practice of Knowledge Discovery in Databases (PKDD)*, 2000, pp. 13–23.

[57] M. Kuramochi and G. Karypis, "An Efficient Algorithm for Discovering Frequent Subgraphs," *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, vol. 16, no. 9, pp. 1038–1051, 2004.

[58] X. Yan and J. Han, "gSpan: Graph-based substructure pattern mining," in *Proceedings of the IEEE International Conference on Data Mining (ICDM)*, 2002.

[59] V. Chaoji, M. Hasan, S. Salem, and M. Zaki, "An Integrated, Generic Approach to Pattern Mining: Data Mining Template Library," *Data Mining and Knowledge Discovery (DMKD)*, vol. 17, no. 3, pp. 457–495, 2008.

[60] S. Nijssen and J. N. Kok, "The gaston tool for frequent subgraph mining," *Electronic Notes in Theoretical Computer Science (ENTCS)*, vol. 127, no. 1, pp. 77–87, 2005.

[61] J. Huan, W. W. 0010, J. Prins, and J. Yang, "SPIN: Mining maximal frequent subgraphs from graph databases," in *Proceedings of the ACM International Conference on Knowledge Discovery and Data Mining (KDD)*, 2004, pp. 581–586.

[62] L. Thomas, S. Valluri, and K. Karlapalem, "Margin: Maximal frequent subgraph mining," in *International Conference on Data Mining*, 2006, pp. 1097–1101.

[63] X. Yan and J. Han, "Closegraph: Mining closed frequent graph patterns," in *Proceedings of the ACM International Conference on Knowledge Discovery and Data Mining (KDD)*, 2003, pp. 286–295.

[64] V. Chaoji, M. Hasan, S. Salem, J. Besson, and M. Zaki, "ORIGAMI: A Novel and Effective Approach for Mining Representative Orthogonal Graph Patterns," *Statistical Analysis and Data Mining (SADM)*, vol. 1, no. 2, pp. 67–84, 2008.

[65] M. Al Hasan and M. J. Zaki, "Output space sampling for graph patterns," *Very Large DataBase Endowment (VLDB)*, vol. 2, no. 1, pp. 730–741, 2009.

[66] M. A. Hasan and M. Zaki, "MUSK: Uniform sampling of k maximal patterns," in *Proceedings of the SIAM Data Mining (SDM)*, 2009, pp. 650–661.

[67] N. Kashtan, S. Itzkovitz, R. Milo, and U. Alon, "Efficient sampling algorithm for estimating subgraph concentrations and detecting network motifs," *Bioinformatics*, vol. 20, no. 11, pp. 1746–1758, 2004.

[68] S. Wernicke, "Efficient detection of network motifs," *IEEE/ACM Transactions on Computational Biology and Bio-informatics (TCBB)*, vol. 3, no. 4, pp. 347–359, 2006.

[69] P. Ribeiro and F. Silva, "G-tries: An efficient data structure for discovering network motifs," in *Proceedings of the ACM Symposium on Applied Computing*, 2010, pp. 1559–1566.

[70] Z. Kashani, H. Ahrabian, E. Elahi, A. Nowzari-Dalini, E. Ansari, S. Asadi, S. Mohammadi, F. Schreiber, and A. Masoudi-Nejad, "Kavosh: A new algorithm for finding network motifs," *BMC Bioinformatics*, vol. 10, no. 1, p. 318, 2009.

[71] P. Wang, J. Lui, B. Ribeiro, D. Towsley, J. Zhao, and X. Guan, "Efficiently estimating motif statistics of large networks," *ACM Transactions on Knowledge Discovery from Data (TKDD)*, vol. 9, no. 2, 2014.

[72] S. S. Shen-Orr, R. Milo, S. Mangan, and U. Alon, "Network motifs in the transcriptional regulation network of escherichia coli," *Nature Genetics*, vol. 31, no. 1, pp. 1061–4036, 2002.

[73] R. Milo, S. Shen-Orr, S. Itzkovitz, N. Kashtan, D. Chklovskii, and U. Alon, "Network motifs: Simple building blocks of complex networks," *Science*, vol. 298, pp. 824–827, 2002.

[74] I. Albert and R. Albert, "Conserved network motifs allow protein–protein interaction prediction," *Bioinformatics*, vol. 20, no. 18, pp. 3346–3352, 2004.

[75] K. Juszczyszyn, P. Kazienko, and K. Musiał, "Local topology of social network based on motif analysis," in *Proceedings of the International Conference on Knowledge-Based Intelligent Information and Engineering Systems (KES)*, 2008, pp. 97–105.

[76] S. Itzkovitz and U. Alon, "Subgraphs and network motifs in geometric networks," *Physical Review E, Statistical, Nonlinear, and Soft Matter Physics*, vol. 71, no. 2, 2005.

[77] S. Omidi, F. Schreiber, and A. Masoudi-Nejad, "MODA: An efficient algorithm for network motif discovery in biological networks." *Genes and Genetic Systems*, vol. 84, no. 5, pp. 385–395, 2009.

[78] J. Grochow and M. Kellis, "Network motif discovery using subgraph enumeration and symmetry-breaking," in *Proceedings of the International Conference on Research in Computational Molecular Biology (RECOMB)*, 2007, pp. 92–106.

[79] X. Li, D. S. Stones, H. Wang, H. Deng, X. Liu, and G. Wang, "Netmode: Network motif detection without nauty," *PloS One*, vol. 7, no. 12, p. e50093, 2012.

[80] G. Amitai, A. Shemesh, E. Sitbon, M. Shklar, D. Netanely, I. Venger, and S. Pietrokovski, "Network analysis of protein structures identifies functional residues," *Journal of Molecular Biology*, vol. 344, no. 4, pp. 1135–1146, 2004.

[81] K. Brinda and S. Vishveshwara, "Oligomeric protein structure networks: Insights into protein-protein interactions," *BMC Bioinformatics*, vol. 6, no. 1, p. 296, 2005.

[82] X. Zhang, T. Perica, and S. A. Teichmann, "Evolution of protein structures and interactions from the perspective of residue contact networks," *Current Opinion in Structural Biology*, vol. 23, no. 6, pp. 954–963, 2013.

[83] A. Giuliani, A. Krishnan, J. P. Zbilut, and M. Tomita, "Proteins as networks: Usefulness of graph theory in protein science," *Current Protein and Peptide Science*, vol. 9, no. 1, pp. 28–38, 2008.

[84] W. Dhifli, R. Saidi, and E. M. Nguifo, "Smoothing 3d protein structure motifs through graph mining and amino acid similarities," *Journal of Computational Biology*, vol. 21, no. 2, pp. 162–172, 2014.

[85] W. Dhifli and A. B. Diallo, "Protnn: Fast and accurate protein 3d-structure classification in structural and topological space," *BioData Mining*, vol. 9, no. 1, p. 30, 2016.

[86] P. P. Wangikar, A. V. Tendulkar, S. Ramya, D. N. Mali, and S. Sarawagi, "Functional sites in protein families uncovered via an objective and automated graph theoretic approach," *Journal of Molecular Biology*, vol. 326, no. 3, pp. 955–978, 2003.

[87] C. Böde, I. A. Kovács, M. S. Szalay, R. Palotai, T. Korcsmáros, and P. Csermely, "Network analysis of protein dynamics," *Febs Letters*, vol. 581, no. 15, pp. 2776–2782, 2007.

[88] K. M. Borgwardt, C. S. Ong, S. Schönauer, S. Vishwanathan, A. J. Smola, and H.-P. Kriegel, "Protein function prediction via graph kernels," *Bioinformatics*, vol. 21, no. suppl 1, pp. 47–56, 2005.

[89] A. Harrison, F. Pearl, R. Mott, J. Thornton, and C. Orengo, "Quantifying the similarities within fold space," *Journal of Molecular Biology*, vol. 323, no. 5, pp. 909–926, 2002.

[90] E. Krissinel and K. Henrick, "Protein structure comparison in 3D based on secondary structure matching (ssm) followed by c-alpha alignment, scored by a new structural similarity function," in *Proceedings of the International Conference on Molecular Structural Biology*, vol. 88, 2003.

[91] R. Shen, N. C. Goonesekere, and C. Guda, "Mining functional subgraphs from cancer protein-protein interaction networks," *BMC Systems Biology*, vol. 6, no. Suppl 3, p. S2, 2012.

[92] R. Shen, X. Wang, C. Guda, and C. B. Guda, "Discovering distinct functional modules of specific cancer types using protein-protein interaction networks," *BioMed Research International*, vol. 2015, 2015.

[93] C. Ding, X. He, R. F. Meraz, and S. R. Holbrook, "A unified representation of multiprotein complex data for modeling interaction networks," *Proteins: Structure, Function, and Bioinformatics*, vol. 57, no. 1, pp. 99–108, 2004.

[94] J. Desaphy, E. Raimbaud, P. Ducrot, and D. Rognan, "Encoding protein–ligand interaction patterns in fingerprints and graphs," *Journal of Chemical Information and Modeling*, vol. 53, no. 3, pp. 623–637, 2013.

[95] M. Bhattacharyya, S. Ghosh, and S. Vishveshwara, "Protein structure and function: Looking through the network of side-chain interactions," *Current Protein and Peptide Science*, vol. 17, no. 1, pp. 4–25, 2016.

[96] N. Tuncbag, F. S. Salman, O. Keskin, and A. Gursoy, "Analysis and network representation of hotspots in protein interfaces using minimum cut trees," *Proteins: Structure, Function, and Bioinformatics*, vol. 78, no. 10, pp. 2283–2294, 2010.

[97] N. W. Lemons, B. Hu, and W. S. Hlavacek, "Hierarchical graphs for rule-based modeling of biochemical systems," *BMC Bioinformatics*, vol. 12, no. 1, p. 45, 2011.

[98] L. H. Greene, "Protein structure networks," *Briefings in Functional Genomics*, vol. 11, no. 6, pp. 469–478, 2012.

[99] Y. Zhou and X. Jiang, "Dissecting Android malware: Characterization and Evolution," in *Proceedings of the IEEE Symposium on Security and Privacy*, 2012, pp. 95–109.

[100] H. Gascon, F. Yamaguchi, D. Arp, and K. Rieck, "Structural detection of Android malware using embedded call graphs," in *Proceedings of the ACM Workshop on Artificial Intelligence and Security (AISec)*, 2013, pp. 45–54.

[101] F. Yamaguchi, N. Golde, D. Arp, and K. Rieck, "Modeling and discovering vulnerabilities with code property graphs," in *Proceedings of the IEEE Symposium on Security and Privacy*, 2014, pp. 590–604.

[102] D. Arp, M. Spreitzenbarth, M. Hübner, H. Gascon, and K. Rieck, "Drebin: Effective and explainable detection of Android malware in your pocket," in *Proceedings of the ISOC Network and Distributed System Security Symposium (NDSS)*, vol. 14, 2014, pp. 23–26.

[103] Y. Aafer, W. Du, and H. Yin, "DroidAPIMiner: Mining API-level features for robust malware detection in android," in *Proceedings of the International Conference on Security and Privacy in Communication Systems*, 2013, pp. 86–103.

[104] M. Zheng, M. Sun, and J. Lui, "Droid Analytics: A signature based analytic system to collect, extract, analyze and associate Android malware," in *Proceedings of the IEEE Trust, Security and Privacy in Computing and Communications (TrustCom)*, 2013, pp. 163–171.

[105] F. Martinelli, A. Saracino, and D. Sgandurra, "Classifying Android malware through subgraph mining," in *Proceedings of the Data Privacy Management and Autonomous Spontaneous Security*, 2014, pp. 268–283.

[106] W. Zhou, Y. Zhou, X. Jiang, and P. Ning, "Detecting repackaged smartphone applications in third-party Android marketplaces," in *Proceedings of the ACM Conference on Data and Application Security and Privacy (CODASPY)*, 2012, pp. 317–326.

[107] W. Zhou, X. Zhang, and X. Jiang, "AppInk: Watermarking Android apps for repackaging deterrence," in *Proceedings of the ACM SIGSAC Symposium on Information, Computer and Communications Security (ASIA CCS)*, 2013, pp. 1–12.

[108] Saikoa, "Proguard," http://proguard.sourceforge.net/, 2014.

[109] ——, "Dexguard," https://www.saikoa.com/dexguard, 2014.

[110] N. Pržulj, D. G. Corneil, and I. Jurisica, "Modeling interactome: Scale-free or geometric?" *Bioinformatics*, vol. 20, no. 18, pp. 3508–3515, 2004.

[111] M. Rahman, M. A. Bhuiyan, M. Rahman, and M. Al Hasan, "GUISE: A uniform sampler for constructing frequency histogram of graphlets," *Knowledge and Information Systems (KAIS)*, vol. 38, no. 3, pp. 511–536, 2014.

[112] W. Enck, P. Gilbert, B.-G. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. N. Sheth, "TaintDroid: An information flow tracking system for real-time privacy monitoring on smartphones," *Communications of the ACM*, vol. 57, no. 3, pp. 99–106, 2014.

[113] S. Arzt, S. Rasthofer, C. Fritz, E. Bodden, A. Bartel, J. Klein, Y. Le Traon, D. Octeau, and P. McDaniel, "Flowdroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps," *ACM Sigplan Notices*, vol. 49, no. 6, pp. 259–269, 2014.

[114] C. Fritz, S. Arzt, S. Rasthofer, E. Bodden, A. Bartel, J. Klein, Y. le Traon, D. Octeau, and P. McDaniel, "Highly precise taint analysis for Android applications," TU Darmstadt, Tech. Rep., 2013.

[115] L.-K. Yan and H. Yin, "DroidScope: Seamlessly reconstructing the OS and Dalvik semantic views for dynamic android malware analysis," in *Proceedings of the USENIX Security*, 2012, pp. 569–584.

[116] A. Reina, A. Fattori, and L. Cavallaro, "A system call-centric analysis and stimulation technique to automatically reconstruct android malware behaviors," in *Proceedings of the European Workshop on System Security (EuroSec)*, 2013.

[117] X. Wang, P. Cui, J. Wang, J. Pei, W. Zhu, and S. Yang, "Community preserving network embedding," in *Proceedings of the Association for the Advancement of Artificial Intelligence (AAAI)*, 2017, pp. 203–209.

[118] P. P. Talukdar and K. Crammer, "New regularized algorithms for transductive learning," in *Proceedings of the European Conference on Machine Learning and Knowledge Discovery in Databases (ECML-PKDD)*, 2009, pp. 442–457.

[119] A. Globerson, G. Chechik, F. Pereira, and N. Tishby, "Euclidean embedding of co-occurrence data," *Journal of Machine Learning Research (JMLR)*, vol. 8, no. Oct, pp. 2265–2295, 2007.

[120] A. Paranjape, A. R. Benson, and J. Leskovec, "Motifs in temporal networks," in *Proceedings of the ACM International Conference on Web Search and Data Mining (WSDM)*, 2017, pp. 601–610.

[121] B. Viswanath, A. Mislove, M. Cha, and K. P. Gummadi, "On the evolution of user interaction in facebook," in *Proceedings of the ACM Workshop on Online Social Networks*, 2009, pp. 37–42.

[122] J. Davis and M. Goadrich, "The relationship between precision-recall and roc curves," in *Proceedings of the International Conference on Machine Learning (ICML)*, 2006, pp. 233–240.

[123] E. Otte and R. Rousseau, "Social network analysis: a powerful strategy, also for the information sciences," *Journal of Information Science*, vol. 28, no. 6, pp. 441–453, 2002.

[124] B. J. Jansen and S. Y. Rieh, "The seventeen theoretical constructs of information searching and information retrieval," *Journal of the American Society for Information Science and Technology*, vol. 61, no. 8, pp. 1517–1534, 2010.

[125] F. Ricci, L. Rokach, B. Shapira, and P. B. Kantor, *Recommender Systems Handbook*, 1st ed. Springer-Verlag New York, Inc., 2010.

[126] D. J. Jacobs, A. J. Rader, L. A. Kuhn, and M. F. Thorpe, "Protein flexibility predictions using graph theory," *Proteins: Structure, Function, and Bioinformatics*, vol. 44, no. 2, pp. 150–165, 2001.

[127] D. Liben-Nowell and J. Kleinberg, "The link-prediction problem for social networks," *Journal of the Association for Information Science and Technology*, vol. 58, no. 7, pp. 1019–1031, 2007.

[128] K. Miller, M. I. Jordan, and T. L. Griffiths, "Nonparametric latent feature models for link prediction," in *Proceedings of the Advances in Neural Information Processing Systems (NIPS)*, 2009, pp. 1276–1284.

[129] B. Taskar, M. fai Wong, P. Abbeel, and D. Koller, "Link prediction in relational data," in *Proceedings of the Advances in Neural Information Processing Systems (NIPS)*, 2004, pp. 659–666.

[130] R. N. Lichtenwalter, J. T. Lussier, and N. V. Chawla, "New perspectives and methods in link prediction," in *Proceedings of the ACM International Conference on Knowledge Discovery and Data Mining (KDD)*, 2010, pp. 243–252.

[131] N. Barbieri, F. Bonchi, and G. Manco, "Who to follow and why: Link prediction with explanations," in *Proceedings of the ACM International Conference on Knowledge Discovery and Data Mining (KDD)*, 2014, pp. 1266–1275.

[132] M. A. Hasan and M. J. Zaki, "A survey of link prediction in social networks," *Social Network Data Analytics*, pp. 243–275, 2011.

[133] P. Wang, B. Xu, Y. Wu, and X. Zhou, "Link prediction in social networks: The state-of-the-art," *Science China Information Sciences*, vol. 58, no. 1, pp. 1–38, 2015.

[134] A. K. Menon and C. Elkan, "Link prediction via matrix factorization," in *Proceedings of European Conference on Machine Learning and Knowledge Discovery in Databases (ECML-PKDD)*, 2011, pp. 437–452.

[135] P. Sarkar, D. Chakrabarti, and M. I. Jordan, "Nonparametric link prediction in dynamic networks," in *Proceedings of the International Conference on Machine Learning (ICML)*, 2012, pp. 1687–1694.

[136] K. S. Xu and A. O. Hero III, "Dynamic stochastic blockmodels for time-evolving social networks," *IEEE Journal of Selected Topics in Signal Processing*, vol. 8, no. 4, pp. 552–562, 2014.

[137] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Parallel distributed processing: Explorations in the microstructure of cognition," pp. 318–362, 1986.

[138] C. E. Priebe, J. M. Conroy, D. J. Marchette, and Y. Park, "Scan statistics on enron graphs," *Computational and Mathematical Organization Theory*, vol. 11, no. 3, pp. 229–247, 2005.

[139] J. Tang, J. Zhang, L. Yao, J. Li, L. Zhang, and Z. Su, "Arnetminer: Extraction and mining of academic social networks," in *Proceedings of the ACM International Conference on Knowledge Discovery and Data Mining (KDD)*, 2008, pp. 990–998.

[140] C. Wang, V. Satuluri, and S. Parthasarathy, "Local probabilistic models for link prediction," in *Proceedings of the IEEE International Conference on Data Mining (ICDM)*, 2007, pp. 322–331.

[141] Y. Yang, R. N. Lichtenwalter, and N. V. Chawla, "Evaluating link prediction methods," *Knowledge and Information Systems (KAIS)*, vol. 45, no. 3, pp. 751–782, 2015.

[142] G. Erkan and D. R. Radev, "LexRank: Graph-based lexical centrality as salience in text summarization," *Journal of Artificial Intelligence Research*, vol. 22, no. 1, pp. 457–479, 2004.

[143] T. Joachims, "Training linear svms in linear time," in *Proceedings of the ACM International Conference on Knowledge Discovery and Data Mining (KDD)*, 2006, pp. 217–226.

[144] Y. Bengio, R. Ducharme, P. Vincent, and C. Janvin, "A neural probabilistic language model," *Journal of Machine Learning Research (JMLR)*, vol. 3, no. Feb, pp. 1137–1155, 2003.

[145] I. Malioutov and R. Barzilay, "Minimum cut model for spoken lecture segmentation," in *Proceedings of the International Conference on Computational Linguistics (ACL)*, 2006, pp. 25–32.

[146] F. Wolf and E. Gibson, "Representing Discourse Coherence: A Corpus-Based Study," *Computational Linguistics*, vol. 31, no. 2, pp. 249–288, 2005.

[147] K. M. Hermann and P. Blunsom, "Multilingual models for compositional distributed semantics," in *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, 2014, pp. 58–68.

[148] A. Nenkova and K. McKeown, "Automatic summarization," *Foundations and Trends in Information Retrieval*, vol. 5, no. 23, pp. 103–233, 2011.

[149] L. Page, S. Brin, R. Motwani, and T. Winograd, "The pagerank citation ranking: Bringing order to the web." Stanford InfoLab, Tech. Rep. 1999-66, 1999.

[150] C.-Y. Lin, "Rouge: A package for automatic evaluation of summaries," in *Proceedings of the Workshop on Text Summarization Branches Out*, 2004.

[151] D. Arthur and S. Vassilvitskii, "k-means++: The advantages of careful seeding," in *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms*, 2007, pp. 1027–1035.

[152] A. Y. Ng, M. I. Jordan, Y. Weiss *et al.*, "On spectral clustering: Analysis and an algorithm," in *Proceedings of the Neural Information Processing Systems (NIPS)*, vol. 14, no. 2, 2001, pp. 849–856.

[153] S. Van Dongen, "A cluster algorithm for graphs," *Report-Information systems*, no. R 0010, pp. 1–40, 2000.

[154] A. Rosenberg and J. Hirschberg, "V-measure: A conditional entropy-based external cluster evaluation measure." in *Proceedings of the Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, vol. 7, 2007, pp. 410–420.

[155] R. Socher, C. C.-Y. Lin, A. Y. Ng, and C. D. Manning, "Parsing natural scenes and natural language with recursive neural networks," in *Proceedings of the International Conference on Machine Learning (ICML)*, 2011, pp. 129–136.

[156] N. T. Pham, G. Kruszewski, A. Lazaridou, and M. Baroni, "Jointly optimizing word representations for lexical and sentential tasks with the C-PHRASE model," in *Proceedings of the Association for Computational Linguistics (ACL)*, 2015, pp. 971–981.

[157] Y. Zhu, R. Kiros, R. Zemel, R. Salakhutdinov, R. Urtasun, A. Torralba, and S. Fidler, "Aligning books and movies: Towards story-like visual explanations by watching movies and reading books," in *arXiv preprint arXiv:1506.06724*, 2015.

[158] A. L. Maas, R. E. Daly, P. T. Pham, D. Huang, A. Y. Ng, and C. Potts, "Learning word vectors for sentiment analysis," in *Proceedings of the Association for Computational Linguistics: Human Language Technologies (ACL HLT)*, 2011, pp. 142–150.

[159] T. Mikolov, W.-t. Yih, and G. Zweig, "Linguistic regularities in continuous space word representations," in *Proceedings of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL HLT)*, 2013, pp. 746–751.

[160] Z. Harris, "Distributional Structure," *Word*, vol. 10, no. 2-3, pp. 146–162, 1954.

[161] J. Mitchell and M. Lapata, "Composition in distributional models of semantics," *Cognitive Science*, vol. 34, no. 8, pp. 1388–1439, 2010.

[162] F. Morin and Y. Bengio, "Hierarchical probabilistic neural network language model," in *Proceedings of the International Workshop on Artificial Intelligence and Statistics*, 2005, pp. 246–252.

[163] M. Gutmann and A. Hyvärinen, "Noise-contrastive estimation: A new estimation principle for unnormalize statistical models," in *Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS)*, vol. 9, 2010, pp. 297–304.

[164] M. Halliday and R. Hasan, *Cohesion in English*. Longman, 1976.

[165] N. X. Vinh, J. Epps, and J. Bailey, "Information theoretic measures for clusterings comparison: Variants, properties, normalization and correction for chance," *Journal of Machine Learning Research (JMLR)*, vol. 11, no. Oct, pp. 2837–2854, 2010.

[166] R. Socher, A. Perelygin, J. Wu, J. Chuang, C. D. Manning, A. Y. Ng, and C. Potts, "Recursive deep models for semantic compositionality over a sentiment treebank," in *Proceedings of the Empirical Methods in Natural Language Processing (EMNLP)*, 2013, pp. 1631–1642.

[167] M. Deshpande, M. Kuramochi, N. Wale, and G. Karypis, "Frequent substructure-based approaches for classifying chemical compounds," *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, vol. 17, no. 8, pp. 1036–1050, 2005.

[168] H. Hu, X. Yan, Y. Huang, J. Han, and X. J. Zhou, "Mining coherent dense subgraphs across massive biological network for functional discovery," *Bioinformatics*, vol. 1, no. 1, pp. 1–9, 2005.

[169] N. Jin, C. Young, and W. Wang, "GAIA: Graph classification using evolutionary computation," in *Proceedings of the ACM Special Interest Group on Management of Data (SIGMOD)*, 2010, pp. 879–890.

[170] X. Yan, P. S. Yu, and J. Han, "Graph indexing: A frequent structure-based approach," in *Proceedings of the ACM Special Interest Group on Management of Data (SIGMOD)*, 2004, pp. 335–346.

[171] T. Horváth, B. Bringmann, and L. De Raedt, "Frequent hypergraph mining," in *Inductive Logic Programming*, 2007, pp. 244–259.

[172] M. Wörlein, T. Meinl, I. Fischer, and M. Philippsen, "A quantitative comparison of the subgraph miners MoFA, gSpan, FFSM, and gaston," in *Proceedings of the Practice of Knowledge Discovery in Databases (PKDD)*, 2005, pp. 392–403.

[173] W. Lin, X. Xiao, and G. Ghinita, "Large-scale frequent subgraph mining in mapreduce," in *Proceedings of the IEEE International Conference on Data Engineering (ICDE)*, 2014, pp. 844–855.

[174] M. A. Bhuiyan and M. Al Hasan, "An iterative mapreduce based frequent subgraph mining algorithm," *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, vol. 27, no. 3, pp. 608–620, 2015.

[175] X. Yan, H. Cheng, J. Han, and P. S. Yu, "Mining significant graph patterns by leap search," in *Proceedings of ACM Special Interest Group on Management of Data (SIGMOD)*, 2008, pp. 433–444.

[176] M.Thoma, H. Cheng, A. Gretton, J. Han, H. Kriegel, A. Smola, L. Song, P. Yu, X. Yan, and K. Borgwardt, "Near-optimal supervised feature selection among frequent subgraphs," in *Proceedings of the SIAM International Conference on Data Mining (SDM)*, 2009, pp. 1076–1987.

[177] A. Inokuchi, T. Washio, K. Nishimura, and H. Motoda, "A fast algorithm for mining frequent connected subgraphs," IBM Research, Tech. Rep., 2002.

[178] T. Horváth, J. Ramon, and S. Wrobel, "Frequent subgraph mining in outerplanar graphs," *Data Mining and Knowledge Discovery (DMKD)*, vol. 21, no. 3, pp. 472–508, 2010.

[179] T. Horváth and J. Ramon, "Efficient frequent connected subgraph mining in graphs of bounded tree-width," *Theoretical Computer Science*, vol. 411, no. 31-33, pp. 2784–2797, 2010.

[180] R. Vijayalakshmi, R. Nadarajan, J. F. Roddick, M. Thilaga, and P. Nirmala, "Fp-graphminer-a fast frequent pattern mining algorithm for network graphs," *Journal of Graph Algorithms and Applications*, vol. 15, no. 6, pp. 753–776, 2011.

[181] E. Keogh, S. Lonardi, and C. A. Ratanamahatana, "Towards parameter-free data mining," in *Proceedings of the ACM Conference on Knowledge Discovery and Data Mining (KDD)*, 2004, pp. 206–215.

[182] T. K. Saha and M. A. Hasan, "FS^3: A sampling based method for top-k frequent subgraph mining," *Statistical Analysis and Data Mining (SADM)*, vol. 8, no. 4, pp. 245–261, 2015.

[183] R. R. Y. and K. D. K., *Simulation and the Monte Carlo Method.* John Wiley and Sons, 2008.

[184] J. Cheng, Y. Ke, W. Ng, and A. Lu, "Fg-index: Towards verification-free query processing on graph databases," in *Proceedings of the ACM Special Interest Group on Management of Data (SIGMOD)*, 2007, pp. 857–872.

[185] B. Bringmann, A. Zimmermann, L. D. Raedt, and S. Nijssen, "Don't be afraid of simpler patterns," in *Proceedings of the Principles and Practice of Knowledge Discovery (PKDD)*, 2006, pp. 55–66.

[186] R. K. Chung, *Spectral Graph Theory.* American Mathematical Society, 1997.

[187] V. Guruswami, "Rapidly mixing markov chains: A comparison of techniques," *A Survey*, 2000.

[188] R. Montenegro and P. Tetali, "Mathematical aspects of mixing times in markov chains," *Foundations and Trends in Theoretical Computer Science*, vol. 1, no. 3, pp. 237–354, 2006.

[189] J. S. Rosenthal, "Optimal proposal distributions and adaptive MCMC," *Handbook of Markov Chain Monte Carlo*, pp. 93–112, 2011.

[190] A. Gelman and D. B. Rubin, "Inference from iterative simulation using multiple sequences," *Statistical Science*, vol. 7, pp. 457–472, 1992.

[191] A. Tomovic and E. J. Oakeley, "Computational structural analysis: Multiple proteins bound to DNA," *PloS One*, vol. 3, no. 9, pp. 32–43, 2008.

[192] R. Chen and Z. Weng, "A novel shape complementarity scoring function for protein-protein docking," *Proteins: Structure, Function, and Bioinformatics*, vol. 51, no. 3, pp. 397–408, 2003.

[193] R. Chen, L. Li, and Z. Weng, "Zdock: An initial-stage protein-docking algorithm," *Proteins: Structure, Function, and Bioinformatics*, vol. 52, no. 1, pp. 80–87, 2003.

[194] I. S. Moreira, J. M. Martins, J. T. Coimbra, M. J. Ramos, and P. A. Fernandes, "A new scoring function for protein–protein docking that identifies native structures with unprecedented accuracy," *Physical Chemistry Chemical Physics*, vol. 17, no. 4, pp. 2378–2387, 2015.

[195] N. Hulo, A. Bairoch, V. Bulliard, L. Cerutti, E. De Castro, P. S. Langendijk-Genevaux, M. Pagni, and C. J. Sigrist, "The prosite database," *Nucleic Acids Research*, vol. 34, no. suppl 1, pp. D227–D230, 2006.

[196] C. G. Nevill-Manning, T. D. Wu, and D. L. Brutlag, "Highly specific protein sequence motifs for genome analysis," *Proceedings of the National Academy of Sciences*, vol. 95, no. 11, pp. 5865–5871, 1998.

[197] J. Y. Huang and D. L. Brutlag, "The emotif database," *Nucleic Acids Research*, vol. 29, no. 1, pp. 202–204, 2001.

[198] R. Saidi, M. Maddouri, and E. M. Nguifo, "Protein sequences classification by means of feature extraction with substitution matrices," *BMC Bioinformatics*, vol. 11, no. 1, p. 1, 2010.

[199] M. Vendruscolo, N. Dokholyan, E. Paci, and M. Karplus, "Small-world view of the amino acids that play a key role in protein folding," *Physical Review E*, vol. 65, no. 6, p. 061910, 2002.

[200] E. Wong, B. Baur, S. Quader, and C.-H. Huang, "Biological network motif detection: Principles and practice," *Briefings in Bioinformatics*, vol. 13, no. 2, pp. 202–215, 2011.

[201] B. K. Kamapantula, M. L. Mayo, E. J. Perkins, and P. Ghosh, "The structural role of feed-forward loop motif in transcriptional regulatory networks," *Mobile Networks and Applications*, vol. 21, no. 1, pp. 191–205, 2016.

[202] S. Gao, A. Chen, A. Rahmani, J. Zeng, M. Tan, R. Alhajj, J. Rokne, D. Demetrick, and X. Wei, "Multi-scale modularity and motif distributional effect in metabolic networks," *Current Protein and Peptide Science*, vol. 17, no. 1, pp. 82–92, 2016.

[203] M. Elseidy, E. Abdelhamid, S. Skiadopoulos, and P. Kalnis, "Grami: Frequent subgraph and pattern mining in a single large graph," *Proceedings of the Very Large Database (VLDB) Endowment*, vol. 7, no. 7, pp. 517–528, 2014.

[204] D. Aparicio, P. Paredes, and P. Ribeiro, "A scalable parallel approach for subgraph census computation," in *Proceedings of the European Conference on Parallel Processing*, 2014, pp. 194–205.

[205] C. H. Teixeira, A. J. Fonseca, M. Serafini, G. Siganos, M. J. Zaki, and A. Aboulnaga, "Arabesque: A system for distributed graph mining," in *Proceedings of the Symposium on Operating Systems Principles (SOSP)*, 2015, pp. 425–440.

[206] J. Huan, W. Wang, and J. Prins, "Efficient mining of frequent subgraphs in the presence of isomorphism," in *Proceedings of the IEEE International Conference on Data Mining (ICDM)*, 2003, pp. 549–552.

[207] M. A. Bhuiyan and M. Al Hasan, "An iterative mapreduce based frequent subgraph mining algorithm," *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, vol. 27, no. 3, pp. 608–620, 2015.

[208] A. R. Katebi, K. Sankar, K. Jia, and R. L. Jernigan, "The use of experimental structures to model protein dynamics," in *Molecular Modeling of Proteins*, 2015, pp. 213–236.

[209] P. D. Dobson and A. J. Doig, "Distinguishing enzyme structures from non-enzymes without alignments," *Journal of Molecular Biology*, vol. 330, no. 4, pp. 771–783, 2003.

[210] A. R. Rao, R. Jana, and S. Bandyopadhyay, "A markov chain monte carlo method for generating random (0, 1)-matrices with given marginals," *Sankhyā: The Indian Journal of Statistics, Series A*, pp. 225–242, 1996.

[211] R. Milo, N. Kashtan, S. Itzkovitz, M. E. Newman, and U. Alon, "On the uniform generation of random graphs with prescribed degree sequences," *arXiv preprint cond-mat/0312028*, 2003.

[212] W. Kabsch, "A solution for the best rotation to relate two sets of vectors," *Acta Crystallographica Section A: Crystal Physics, Diffraction, Theoretical and General Crystallography*, vol. 32, no. 5, pp. 922–923, 1976.

[213] M. W. Walker, L. Shao, and R. A. Volz, "Estimating 3-d location parameters using dual number quaternions," *CVGIP: Image Understanding*, vol. 54, no. 3, pp. 358–367, 1991.

[214] H. M. Berman, J. Westbrook, Z. Feng, G. Gilliland, T. Bhat, H. Weissig, I. N. Shindyalov, and P. E. Bourne, "The protein data bank," *Nucleic Acids Research*, vol. 28, no. 1, pp. 235–242, 2000.

[215] A. R. Katebi and R. L. Jernigan, "The critical role of the loops of triosephosphate isomerase for its oligomerization, dynamics, and functionality," *Protein Science*, vol. 23, no. 2, pp. 213–228, 2014.

[216] E. Lolis, T. Alber, R. C. Davenport, D. Rose, F. C. Hartman, and G. A. Petsko, "Structure of yeast triosephosphate isomerase at 1.9-. ang. resolution," *Biochemistry*, vol. 29, no. 28, pp. 6609–6618, 1990.

[217] G. M. Cooper, *The Cell: A Molecular Approach, 2nd Edition.* Sunderland (MA): Sinauer Associates, 2000.

[218] C. T. Porter, G. J. Bartlett, and J. M. Thornton, "The catalytic site atlas: A resource of catalytic sites and residues identified in enzymes using structural data," *Nucleic Acids Research*, vol. 32, no. suppl 1, pp. D129–D133, 2004.

[219] J. P. Nilmeier, D. A. Kirshner, S. E. Wong, and F. C. Lightstone, "Rapid catalytic template searching as an enzyme function prediction procedure," *PloS One*, vol. 8, no. 5, p. e62535, 2013.

[220] N. Furnham, G. L. Holliday, T. A. de Beer, J. O. Jacobsen, W. R. Pearson, and J. M. Thornton, "The catalytic site atlas 2.0: Cataloging catalytic sites and residues identified in enzymes," *Nucleic Acids Research*, vol. 42, no. D1, pp. D485–D489, 2014.

[221] W. Dhifli, S. Aridhi, and E. M. Nguifo, "Mr-simlab: Scalable subgraph selection with label similarity for big data," *Information Systems*, vol. 69, pp. 155 – 163, 2017.

[222] L. A. Goodman, "Snowball sampling," *The Annals of Mathematical Statistics*, pp. 148–170, 1961.

[223] M. Gjoka, M. Kurant, C. Butts, and A. Markopoulou, "Walking in Facebook: A Case Study of Unbiased Sampling of OSNs," in *Proceedings of the IEEE International Conference on Computer Communications (INFOCOMM)*, 2010, pp. 1–9.

[224] M. Rahman, M. Bhuiyan, and M. A. Hasan, "GRAFT: An approximate graphlet counting algorithm for large graph analysis," in *Proceedings of the ACM International Conference on Information and Knowledge Management (CIKM)*, 2012, pp. 1467–1471.

[225] S. P. Borgatti and M. G. Everett, "A graph-theoretic perspective on centrality," *Social Networks*, vol. 28, no. 4, pp. 466–484, 2006.

[226] S. Dorogovtsev, J. Mendes, and A. Samukhin, "Size-dependent degree distribution of a scale-free growing network," *Physical Review E*, vol. 63, no. 6, p. 062101, 2001.

[227] H. A. Soufiani and E. M. Airoldi, "Graphlet decomposition of a weighted network," *arXiv preprint arXiv:1203.2821*, 2012.

[228] P. Ribeiro, "Efficient and scalable algorithms for network motifs discovery," Ph.D. dissertation, University of Porto, 2011.

[229] W. R. Gilks, S. Richardson, and D. J. Spiegelhalter, "Introducing Markov chain Monte Carlo," in *Markov Chain Monte Carlo in Practice*, 1996, pp. 1–19.

[230] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller, "Equation of state calculations by fast computing machines," *The Journal of Chemical Physics*, vol. 21, no. 6, pp. 1087–1092, 1953.

[231] P. Indyk and R. Motwani, "Approximate nearest neighbors: Towards removing the curse of dimensionality," in *Proceedings of the ACM Symposium on Theory of Computing (STOC)*, 1998, pp. 604–613.

[232] N. Viennot, E. Garcia, and J. Nieh, "A measurement study of google play," in *Proceedings of the ACM International Conference on Measurement and Modeling of Computer Systems*, 2014, pp. 221–233.

[233] A. Labs, "Androguard," https://code.google.com/p/androguard/, 2014.

[234] C.-C. Chang and C.-J. Lin, "LIBSVM: A library for support vector machines," *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 2, no. 3, p. 27, 2011.

[235] S. Narang, "Tapsnake," http://www.symantec.com/connect/blogs/android-tapsnake-mobile-scareware-ads-push-antivirus, 2013.

[236] X. Jiang, "Sndapps," http://www.csc.ncsu.edu/faculty/jiang/SndApps/, 2011.

[237] J. Grunzweig, "Nickyspy," https://www.trustwave.com/Resources/SpiderLabs-Blog/NickiSpy-C---Android-Malware-Analysis--Demo/, 2011.

[238] Y. Li, "Lovetrap," https://www.symantec.com/security_response/writeup.jsp?docid=2011-072806-2905-99&tabid=2, 2011.

[239] R. Bawden, R. Sennrich, A. Birch, and B. Haddow, "Evaluating discourse phenomena in neural machine translation," *arXiv preprint arXiv:1711.00513*, 2017.

[240] G. Han and H. Sethu, "Waddling random walk: Fast and accurate mining of motif statistics in large graphs," in *Proceedings of the IEEE International Conference on Data Mining (ICDM)*, 2016, pp. 181–190.

[241] M. Bressan, F. Chierichetti, R. Kumar, S. Leucci, and A. Panconesi, "Counting graphlets: Space vs time," in *Proceedings of the ACM International Conference on Web Search and Data Mining (WSDM)*, 2017, pp. 557–566.

[242] Y. Li, O. Vinyals, C. Dyer, R. Pascanu, and P. Battaglia, "Learning deep generative models of graphs," *arXiv preprint arXiv:1803.03324*, 2018.

[243] R. K. Darst, C. Granell, A. Arenas, S. Gómez, J. Saramäki, and S. Fortunato, "Detection of timescales in evolving complex systems," *Scientific Reports*, vol. 6, p. 39713, 2016.

[244] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[245] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *arXiv preprint arXiv:1409.0473*, 2014.

[246] S. Sukhbaatar, J. Weston, R. Fergus *et al.*, "End-to-end memory networks," in *Proceedings of the Advances in Neural Information Processing Systems (NIPS)*, 2015, pp. 2440–2448.

[247] T. Lyu, Y. Zhang, and Y. Zhang, "Enhancing the network embedding quality with structural similarity," in *Proceedings of the ACM Conference on Information and Knowledge Management (CIKM)*, 2017, pp. 147–156.

VITA

VITA

Tanay Kumar Saha received his BSc in computer science and engineering from Bangladesh University of Engineering and Technology (BUET) in October, 2009. After graduation, he served as a Lecturer in Jagannath University, Dhaka, Bangladesh. He joined Purdue University in August, 2012 to pursue his Ph.D. degree in computer science under the supervision of Dr. Mohammad Al Hasan. His research focused on information retrieval, deep learning, graph analysis and bio-informatics. During his Ph.D. study, he interned at Qatar Computing Research Institute (QCRI) in 2016 from January to June and NEC labs, America in Summer, 2017.