THE HONG KONG
POLYTECHNIC UNIVERSITY
香港理工大學
Pao Yue-kong Library
包玉剛圖書館

Copyright Undertaking

This thesis is protected by copyright, with all rights reserved.

By reading and using the thesis, the reader understands and agrees to the following terms:

1. The reader will abide by the rules and legal ordinances governing copyright regarding the use of the thesis.
2. The reader will use the thesis for the purpose of research or private study only and not for distribution or further reproduction or any other purpose.
3. The reader agrees to indemnify and hold the University harmless from and against any loss, damage, cost, liability or expenses arising from copyright infringement or unauthorized usage.

IMPORTANT

If you have reasons to believe that any materials in this thesis are deemed not suitable to be distributed in this form, or a copyright owner having difficulty with the material being included in our database, please contact lbsys@polyu.edu.hk providing details. The Library will look into your claim and consider taking remedial action upon receipt of the written requests.

Pao Yue-kong Library, The Hong Kong Polytechnic University, Hung Hom, Kowloon, Hong Kong

http://www.lib.polyu.edu.hk

FEATURE REPRESENTATION FOR MINING EVOLUTION PATTERNS IN DYNAMIC DATA

YU YANG

PhD

The Hong Kong Polytechnic University

2021

The Hong Kong Polytechnic University
Department of Computing

Feature Representation for Mining Evolution Patterns in Dynamic Data

Yu Yang

A thesis submitted in partial fulfillment of the requirements for
the degree of Doctor of Philosophy
August 2021

CERTIFICATE OF ORIGINALITY

I hereby declare that this thesis is my own work and that, to the best of my knowledge and belief, it reproduces no material previously published or written, nor material that has been accepted for the award of any other degree or diploma, except where due acknowledgment has been made in the text.

Signature: _____

Name of Student: Yu Yang

Abstract

Feature representation is an encoding process that projects the raw data into a discriminative latent space so as to extract the characteristics, properties, attributes, and underlying patterns from the data and embed them as features for supporting effective machine learning. It is the heart of AI, powering it to develop and train intelligent algorithms by supplying the useful information and discriminative features extracted from large quantities of high-quality data.

In this thesis, I study an important yet overlooked problem of feature representation in dynamic data for capturing and embedding the evolution patterns, thus effectively facilitating the before-the-fact applications such as a prediction. Dynamic data refers to data that changes over time. It contains historical evolution patterns revealing how the data changed over time. Discovering and embedding these evolution patterns introduces additional and effective information to overcome data insufficiency issues in the before-the-fact applications, thereby leading to better performance.

Existing studies either is based on differential equations or employ data-driven methods. The former one suffers from the high sensitivity of noise and uncertainty, while the latter merely focus on synchronous and/or period evolution patterns, overlooking the complexity of dynamics. Therefore, both of them fail to achieve satisfactory prediction performance. I aim to capture and embed the evolution from data with complex dynamics. Multivariate, multi-timescale, and asynchronous dynamics arise naturally in the world. Although these dynamics are very difficult to be fully captured

due to the high stochastic and uncertainty, discovering the evolution patterns from them and embedding into the representation can effectively facilitate a prediction.

To tackle the challenge of multi-variables, I devised a time-capturing dynamic graph embedding algorithm to learn the synchronous linkage evolution from the dynamic connection changes of every vertex over time. To deal with the challenge of synchronization, I propose a time-aware dynamic graph embedding algorithm to fully capture and embed the asynchronous structural evolutions in which the connections of vertices evolve at different times with variant evolution speed. Extensive experiments show that both algorithms achieved significant performance improvement over the state-of-the-art baselines in various graph mining applications. Lastly, a multi-timescale bag-of-regularity method is devised to extract students' learning regularity patterns from their multi-timescale dynamic learning behaviors, thereby achieving impressively high accuracy in early predicting academic at-risk students. I believe this thesis can serve as a solid step towards advanced knowledge discovery and representation in dynamic data.

Publications Arising from the Thesis

1. Yu Yang, Jiannong Cao, Wengen Li, Linchuan Xu, Zhongyu Yao, Ka Ho Wong, Mingjin Zhang, and Esther Ahn Chian Ku, "BigEng: A Big Data-Driven Engine for Inbound Baggage Allocation at Airports", manuscript submitted to *IEEE Transactions on Industrial Informatics (TII)*.

2. Yu Yang, Hongzhi Yin, Jiannong Cao, Tong Chen, Quoc Viet Hung Nguyen, Xiaofang Zhou, and Lei Chen, "Time-aware Dynamic Graph Embedding for Asynchronous Structural Evolution", manuscript submitted to *IEEE Transactions on Knowledge and Data Engineering (TKDE)*.

3. Yu Yang, Jiannong Cao, Milos Stojmenovic, Senzhang Wang, Yiran Cheng, Chun Lum, and Zhetao Li, "Time-capturing Dynamic Graph Embedding for Temporal Linkage Evolution", in *IEEE Transactions on Knowledge and Data Engineering (TKDE)* (2021).

4. Yu Yang, Zhiyuan Wen, Jiannong Cao, Jiaxing Shen, Hongzhi Yin, and Xiaofang Zhou, "EPARS: Early prediction of at-risk students with online and offline learning behaviors", in *25th International Conference on Database Systems for Advanced Applications (DASFAA)*, pp. 3-9, 2020.

5. Yu Yang, Jiannong Cao, Jiaxing Shen, Ruosong Yang, and Zhiyuan Wen, "Learn-

ing Analytics Based on Multilayer Behavior Fusion", in *13rd International Conference on Blended Learning (ICBL)*, pp. 15-24, 2020.

6. Yu Yang, Hanqing Wu, and Jiannong Cao, "Smartlearn: Predicting learning performance and discovering smart learning strategies in flipped classroom",in *2016 International Conference on Orange Technologies (ICOT)*, pp. 92-95, 2016.

7. Jiaxing Shen, Jiannong Cao, Yu Yang, Cho-Li Wang, Tingrui Pei, Zhetao Li, and Alex 'Sandy' Pentland, "Accurate Inference of Social Networks in the Physical World using Transaction Data", manuscript submitted to *38th IEEE International Conference on Data Engineering (ICDE 2022)*.

8. Zhuo Li, Jiannong Cao, Zhongyu Yao, Wengen Li, Yu Yang, and Jia Wang, "Recursive Balanced k-Subset Sum Partition for Rule-constrained Resource Allocation",in *29th ACM International Conference on Information & Knowledge Management (CIKM)*, pp. 2121-2124, 2020.

9. Ka Ho Wong, Jiannong Cao, Yu Yang, Wengen Li, Jia Wang, Zhongyu Yao, Suyan Xu, Esther Ahn Chian Ku, Chun On Wong, and David Leung, "BigARM: A Big-Data-Driven Airport Resource Management Engine and Application Tools", in *25th International Conference on Database Systems for Advanced Applications (DASFAA)*, pp. 741-744, 2020.

10. Jiating Zhu, Yu Yang, Jiannong Cao, and Esther Chak Fung Mei, "New product design with popular fashion style discovery using machine learning", in *International Conference on Artificial Intelligence on Textile and Apparel*, pp. 121-128, 2018.

Acknowledgments

A four-and-a-half-year journey of pursuing a Doctor of Philosophy (Ph.D.) is not quite long but it is an important and unforgettable milestone in my life. I overcome challenges, beat frustrations, and finally reach the destination. The journey is tough, but it is so much more than worthy. Instead of publishing a few papers, my understanding of the Ph.D. is more about scientific thinking, high-level abstraction, logical organization, and innovative solutions, thus resulting in seeing the truth beyond the world.

First of all, I would like to thank my family for giving me unlimited supports on my Ph.D. journey. Thank my mother Ms. Hongying Wu for supporting and understanding my every decision without hesitation. Thank my wife Ms. Siyin He for taking care of the whole family especially for our lovely children Tianran Yang and Danran Yang. Due to the pandemic of COVID-19, I cannot return home and get together with them. Although we usually feel lonely and helpless from each other, we always encourage and support each other, solving every problem and challenge we faced. Thank Tangyuan, Rongxin, Rongwen, and Pidan for saving me from frustrations and depressions every time. I will always miss you all.

Next, I would like to give my biggest thanks to my Ph.D. advisor Prof. Jiannong Cao. He is always energetic to do everything with the heart, being my model. He taught me to pursue the truth beyond the world and guard me against being lost. He created opportunities for me in accordance with my aptitude. Thanks for giv-

ing me opportunities to have this unforgettable journey in the Internet and Mobile Computing Lab (IMCL) and for having the invaluable advisory to everything.

Besides, I would like to thank all my collaborators on my Ph.D. journey. Thank Prof. Milos Stojmenovic for working together with me on any problem that I meet at any time. He always has interesting ideas and encourages me to think creatively. Thank Prof. Xiaofang Zhou and Dr. Hongzhi Yin for the unforgettable research cooperation at the University of Queensland. It not only broadens my research horizons but also greatly improves the methodological skills in research. Thank Dr. Frances Fan for the domain knowledge in education, which helps me a lot in conducting research on learning analytics.

Furthermore, I would like to thank all my colleagues in IMCL. Thank Dr. Wengen Li for a great job in our iconic project BigARM. It is too impressive for me on every time we work together to make our job perfectly meet the higher and higher expectation of the Airport Authority of Hong Kong. Thank Dr. Jiaxing Shen, Dr. Shan Jiang, Dr. Yanni Yang, Dr. Senzhang Wang, Dr. Xiulong Liu, Dr. Linchuan Xu, Dr. Yuqi Wang, Dr. Xuefeng Liu, Dr. Zhuo Li, Dr. Jia Wang, Dr. Chun-Tung Li, Dr. Lei Yang, Dr. Yuvraj Sahni, Dr. Divya Saxena, Dr. Tarun Kulshrestha, Dr. Yanwen Wang, Dr. Kongyang Chen, Dr. Fuliang Li, Dr. Peiyuan Zhou, Mr. Hanqing Wu, Mr. Ruosong Yang, Mr. Zhiyuan Wen, Ms. Jiating Zhu, Mr. Mingjin Zhang, Mr. Shuaiqi Liu, Mr. Qianyi Chen, Mr. Zhixuan Liang, Ms. Junchen Zhu, Ms. Yuqing Zhao, Mr. Yinfeng Cao, Mr. Yiran Cheng, Mr. Ka Ho Wong, Mr. Zhongyu Yao, Ms. Suyan Xu, and Ms. Esther Ahn Chian Ku for the great support and enjoyable time on my Ph.D. journey.

Last but not least, I would like to thank Prof. Zhong Ming, Prof. Qiang Huang, and Dr. Yan-Ran Li from Shenzhen University to educate me with a solid research foundation and recommend me to pursue the Ph.D. with Prof. Cao at the IMCL. Otherwise, the dream is over before it starts.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background & Motivation

Artificial Intelligence (AI) was first coined by Prof. John McCarthy as "the science and engineering of making intelligent machines" [62] at a summer research workshop at Dartmouth College in 1956. In the first decades of the 21st century, driven by big data and highly mathematical-statistical machine learning, AI achieved great success in industry and academia, creating a new era. Feature representation is the heart of AI, powering it to develop and train intelligent algorithms by supplying the useful information and discriminative features extracted from large quantities of high-quality data.

Feature representation is an encoding process that maps the raw data into a discriminative latent space so as to extract the characteristics, properties, attributes, and underlying patterns from the data and embed them as features for supporting effective machine learning [9] [52]. On one hand, it reveals underlying patterns and factors hidden in the data for AI to understand the world around us. On the other hand, it transforms the raw data as numeric feature vectors, thus easily facilitating machine learning algorithms. The performance of intelligent algorithms is heavily dependent

on the expressiveness of the feature vectors [9].

The methodologies of feature representation can be categorized into two. One is feature engineering and the other is representation learning. Feature engineering is the process of hand-crafting the appropriate numeric representation of raw data based on human ingenuity and prior knowledge [116]. Statistical analysis is one of the most common feature engineering approaches to represent raw data by using their statistical characteristics such as maximum, minimum, mean, median, variance, percentiles, probability distributions, etc. Another feature engineering approach is to manually extract invariant properties of raw data and embed them into representations. Gobor [79], SIFT [55], SURF [4], LBP [40], Bag-of-words [97], as well as their multilevel and high-dimensional extensions [113] [17], achieved robust results in various AI applications such as object detection, speech recognition, image retrieval, text classification, etc. Unfortunately, hand-crafted features suffered from a lack of distinctiveness and universality [91], which leads to unsatisfactory performance of machine learning algorithms and weakens their generalization ability.

In representation learning, features are directly learned from data. It automatically discovers useful information and underlying patterns from the data and embeds them into distinctive representations for downstream machine learning tasks such as classification, regression, and prediction [9] [54]. In deep learning, the feedforward neural network is exactly performing representation learning [25]. Specifically, the last layer of a feedforward neural network is typically a linear classifier and/or regressor depending on specific tasks. The rest of the network learns representations from the input data so as to provide informative features for the last layer to achieve the best task performance. When performing the back propagation to optimize the network, it actually is employing the task-specific information to supervise the representation learning process, thus making the learned features distinctive and discriminative for the task. In addition to "deep" models like auto-encoder [38], seq2seq [82], word2vec [63], BERT [21], etc., there also exist many methods on "shallow" represen-

tation learning such as principal component analysis (PCA) [96], linear discriminant analysis (LDA) [8], non-negative matrix factorization (NMF) [93], sparse dictionary learning [2], manifold learning [108], etc.

Although feature representation has been studied for many years, most of the existing works are merely applicable to static data that remains the same over time. Even if it uses the representation learning approaches, it essentially learns the task-specific invariant properties of the static data. Hence, they achieve impressive performance improvement in the after-the-fact applications such as detection, tracking, and recognition, but fail in predictions. Prediction is quantitative forecasting of what will happen in the future with currently observed data, which is a before-the-fact task [45]. Since static data remain unchanged after collection, the prediction output will be the same as the one in the corresponding after-the-fact tasks with currently observed data, thus leading to meaningless results.

Mining the evolution in dynamic data for feature representation is the cornerstone of the after-the-fact task like a prediction. Dynamic data refers to data that change over time [7]. Mathematically, given a static data observation at time t_0, denoting as $x(t_0)$, its dynamics comes from an evolution function $\tilde{x} = f\left(x(t_0), t\right)$, determining how $x(t_0)$ will change over time t and being $\tilde{x}$ [13] [34]. Any data associated with time can be counted as dynamic data and it widely exists in the world. For instance, When $x(t_0) \in \mathbb{R}^1$ is a real number and changes over time, its corresponding $\tilde{x}$ is a univariate time series. If the observed data $x(t_0) \in \mathbb{R}^n$, its $\tilde{x}$ becomes a n-variate time series. If $x(t_0)$ is a static graph, $\tilde{x}$ will be a dynamic graph indicating the connection changes amongst vertices in $x(t_0)$ over time. Since the prediction is made before fully observing the entire data, very limited information can be extracted from the partial data, therefore failing to achieve satisfactory prediction performance. However, the dynamic data contain historical observations that carry the evolution patterns revealing how the data observation changed over time. Leveraging the evolution patterns during feature representation introduces additional and reliable information, thereby being able to

improve the prediction performance.

This thesis studies an important yet overlooked problem of feature representation in dynamic data for capturing and embedding the evolution patterns, thus effectively facilitating the before-the-fact applications such as a prediction. Due to the complex nature of dynamics, it is very challenging to effectively capture and embed the evolution patterns in dynamic data. Existing studies can be categorized into two. One is based on differential equations and the other employs data-driven approaches. The former one assumes that the state observation x obeys differential equations involving time derivatives $\frac{\mathrm{d}x}{\mathrm{d}t} = f(x,t)$, where $f(x,t)$ is the evolution function. After solving the differential equations under specific assumptions, e.g., the Markov hypothesis, $f(x,t)$ is differentiable to time t, etc., the evolution function f will be obtained for prediction. These symbolic approaches are popular in the field of physics, biology, civil engineering, mechanics, etc., for the applications such as climate prediction [68], drug resistance analytics[41], structural health monitoring [1], control [20], etc. Although the differential equation approaches are determinate and explainable, they are too sensitive to noise and uncertainty [13], thus being weak in robustness and generalization when being applied to prediction tasks.

On the other hand, the data-driven approaches first extract discriminative features at each time step and then model their time varied difference to discover the evolution patterns. Because of the robustness to noise and well elimination of uncertainty, these approaches are widely employed in AI-related applications such as accident prediction [109], human action prediction [45], disease progression prediction [88], flow prediction [57], etc. However, most existing studies degenerate the time-varying evolutions into sequential changes so that eases machine learning models, especially recurrent neural networks, to discover and embed the evolution patterns. It is worthwhile noting that, after degeneration, the models are merely capable to capture the synchronous and periodic evolution patterns. In the over words, they assume that the dynamics are invariant over time, which overlooks the complexity of dynamics

and results in unsatisfactory prediction performance. Multivariate, multi-timescale, and asynchronous dynamics arise naturally in the world. Although these dynamics are very difficult to be fully captured due to the high stochastic and uncertainty, discovering the evolution patterns from them and embedding into the representation can effectively facilitate a prediction. For example, temperature varies over daily, seasonal, and yearly time scales. Even under the same time scale, the speed of temperature change in different areas in the world varies a lot. Should such complex dynamics of temperature be degenerated as synchronous and/or periodic changes, the accuracy in temperature or climate prediction will definitely be impaired.

1.2 Research Challenges

This thesis is confronted with the following challenges brought by dynamic data to feature representation.

- Multi-variables. When there exist multiple variables dynamically evolving, they will interact and influence each other. Meanwhile, this interrelationship amongst multiple variables will also dynamically change over time. Therefore, in feature representation, it is not only necessary to preserve the individual evolutions, but also to model the dynamic interrelationship amongst them, which is very challenging. Besides, the dynamic interrelationship amongst multiple variables will introduce additional noise, stochastic, and uncertainty, thus making the evolution pattern mining and feature representation become even more difficult.

- Asynchronization. The dynamics in multivariate data are not always invariant. Dynamic variables or objects evolve at different times with variant evolution speed, thereby being asynchronous. Some of them start evolving early, while others are later. Some evolve quickly, while others evolve slowly. This makes

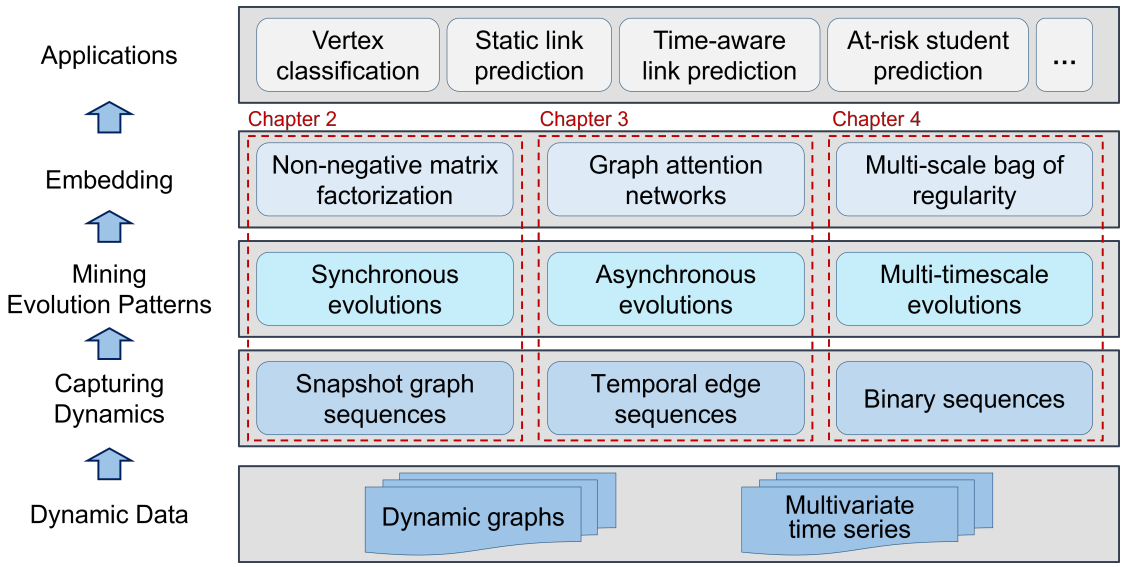

Figure 1.1: Research framework.

the evolution patterns very complicated, thereby being difficult to fully capture and embed in feature representation. Although preserving such evolutionary patterns within asynchronous dynamics is difficult, it can effectively facilitate the feature representation to accurately predict the future.

- Multi-timescales. The coherence of evolution for dynamic variables or objects across multiple scales in time leads them to have multi-timescale dynamics. At each timescale, they have their own changing patterns and eventually constitute the overall evolution. In addition, the dynamic variables or objects can have different evolving periods at each time scale and even asynchronous evolution, therefore making the feature representation challenge to obtain the evolution patterns from the multi-timescale dynamics.

1.3 Research Framework

This thesis studies the problem of feature representation in dynamic data for discovering and embedding the complex evolution patterns, thus eventually achieving highly

accurate results in downstream prediction tasks. As showed in Fig. 1.1, the research framework consists of five layers. The bottom layer is the dynamic data tackled by this thesis. Specifically, this study focus on feature representation in dynamic graphs and multivariate time series. Graphs are one of the most widely used data representations to model pairwise relations between objects. In many real-world applications, e.g., social networks, biological networks, information diffusion networks, and interaction networks, these relations naturally change over time. Vertices could join quickly or slowly, leave at their own pace, and even re-join the graph, thereby making the graph dynamic. Multivariate time series are sequences of multiple variables collected at successive and equal time intervals. Due to its natural temporal ordering properties, it widely appears in statistics, signal processing, physiology, econometrics, computational finance, meteorology, etc. This thesis researches the generic feature representation approaches for mining the evolution patterns from both dynamic graphs and multivariate time series, which can be directly applied to solve the specific problems in the above-mentioned areas.

In the second bottom layer, the raw input dynamic data are transformed into a proper data structure, thereby capturing the particular dynamics for later embedding in feature representation. Specifically, snapshot graph sequences and temporal edge sequences are respectively employed to capture the synchronous and asynchronous structural evolutions in the dynamic graphs. The multivariate time series will be transformed into a set of binary sequences, thereby easing the mining of multi-timescale evolutions. Compared to the raw dynamic data, the transformed data not only preserve the target evolution patterns for mining and feature representation but also reduce the dimensionality and eliminate the noise and uncertainty. This enables feature representation algorithms to more effectively discover evolutionary patterns and embed them as more discriminative features, eventually improving prediction accuracy.

In the embedding layer, this thesis extensively studies both feature engineering and

representation learning approaches to effectively and efficiently embed the evolution patterns into features. In particular, a novel non-negative matrix factorization algorithm with regression co-training is proposed in Chapter 2 to learn the synchronous linkage evolution from the dynamic connection changes of every vertex over time, thereby capturing and embedding the multivariate dynamics in the dynamic graphs. To deal with the challenges of asynchronous dynamic, a time-aware dynamic graph embedding algorithm is present in Chapter 3. It leverages the graph attention networks [86] and the self-attention mechanism [85] to embed the asynchronous structural evolutions in which the connections of vertices evolve at different times with variant evolution speed. Besides, a new feature engineering approach, namely multi-timescale bag-of-regularity, is creatively proposed in Chapter 4 to investigate the multi-timescale dynamics in the binary sequences and embed the multi-timescale and multi-period repetitive patterns.

Lastly, the obtained features carrying the evolution patterns of the dynamic data are applied in downstream prediction tasks including but not limited to vertex classification, static link prediction, time-aware link prediction, at-risk student early prediction. It is worthwhile noting that the time-aware link prediction is a specific application for dynamic graph embedding abstracted from many data mining applications such as forecasting future crowd flow, recommending items at varying time intervals, predicting fraud victims, and the time of victimization. It simultaneously predicts whether a pair of vertices will form an edge and when this edge will appear, which is used to benchmark the effectiveness of dynamic graph embedding algorithms for the first time.

1.4 Thesis Organization

The rest of this thesis is organized as follows:

- In Chapter 2, I studied the problem of mining and embedding the multivariate asynchronous linkage evolution in a dynamic graph. The dynamic graphs are modeled as a sequence of snapshot graphs, appending the respective timespans of edges (ToE), which captures the multivariate synchronous dynamics for vertices in the graphs. A non-negative matrix factorization model is proposed to infer a common latent space for capturing the synchronous structural evolutions while co-training a linear regressor to embed the ToE at the same time. Eventually, it overcomes the challenges of multivariate dynamics and successfully embeds the asynchronous linkage evolutions, thereby achieving significant performance improvement over the state-of-the-art baselines in the applications of vertex classification, static and time-aware link prediction.

- In Chapter 3, I studied an important yet overlooked problem in dynamic graph embedding, namely fully capturing and embedding the asynchronous structural evolutions of a dynamic graph. Such asynchronous nature can be interpreted as the starting time and duration of a new edge that emerges between two vertices and is of great importance to a wide range of time-sensitive applications, e.g., online information propagation modeling and crime prediction. In light of the challenges brought by asynchronous dynamics, I formulate a dynamic graph as a set of temporal edges, coupled with the respective ToE and ToV (joining time of vertices) as two crucial indicators of the asynchronous properties. Lastly, a novel time-aware dynamic graph embedding algorithm is proposed to discover the patterns of asynchronous structural evolutions in the dynamic graph and embed them into vertex representations. Extensive evaluations on large-scale real-world datasets show that this approach outperforms the state-of-the-art in a wide range of graph mining applications, thereby demonstrating its effectiveness and superiority in dealing with asynchronous dynamics.

- In Chapter 4, I studied the problem of early predicting academic at-risk students with dynamic learning behaviors. During the semester, students' learning

patterns vary over daily, weekly, and monthly time scales. With the observation that study routines of good students are periodical and at-risk students usually have more drop-out friends, a novel multi-timescale bag-of-regularity method is proposed to discover students' learning regularity patterns, overcoming the challenges brought by the multiple-timescale dynamics in their learning behaviors. Next, a co-occurrence network is constructed to approximate students' underlying social networks and encode the social homophily as features through graph embedding. With the fused features of learning regularity and social homophily, it can achieve impressive accuracy in very early predicting at-risk students.

- In Chapter 5, I summarized the contributions made by this thesis and drew the conclusions. Importantly, I further discussed the open challenges and future directions in feature representation at the end of this thesis.

Chapter 2

Time-capturing Dynamic Graph Embedding for Temporal Linkage Evolution

Dynamic graph embedding learns representation vectors for vertices and edges in a graph that evolves over time. In this chapter, I study the feature representation to capture and embed the synchronous evolution of vertices' temporal connectivity, dealing with the challenges of multi-variables. Existing work studies the vertices' dynamic connection changes but neglects the time it takes for edges to evolve, thus failing to embed temporal linkage information with the synchronous evolution patterns in the dynamic graph. To capture vertices' temporal linkage evolution, dynamic graphs are modeled as a sequence of snapshot graphs, appending the respective timespans of edges (ToE). The snapshot graph sequence will capture the synchronous evolution of vertices' connections in the dynamic graph. A common latent representation space for all snapshot graphs is learned by a matrix-factorization-based model to embed vertices' dynamic connection changes while co-training a linear regressor to embed ToE. Extensive evaluations on several datasets show that the proposed algorithm

can achieve significant performance improvements, i.e. 22.98% on average across all datasets, over the state-of-the-art baselines in a wide range of graph mining tasks including vertex classification, static link prediction, time-aware link prediction, and ToE prediction.

2.1 Introduction

Dynamic graph embedding captures and encodes the evolution of vertex properties and connections as low dimensional representation vectors in order to benefit downstream machine learning applications. Existing works model the dynamic graph as either a sequence of static snapshot graphs [120] [110] [118] [119] [29] [26] [35] [98] [58] or neighborhood formation sequence sampled from the temporal random walk [65] [121] [70]. These approaches merely capture the sequential changes of static graph structure throughout the snapshot graph sequence as well as the sequential linkage evolution among vertices for embedding. However, the time it takes for vertex connections to evolve is also dynamic and it is neglected by the above approaches. Here, I tackle the problem of embedding the temporal linkage evolution of vertices in a dynamic graph, while simultaneously preserving their dynamic connection changes and timespans of edge formation (ToE).

ToE preserves important duration of edge formation information as well as the temporal dependencies of vertices while the dynamic graphs evolve. For example, in a dynamic transaction network, buyers could appear at any time to trade with sellers and disappear afterward, thereby forming an edge. The ToE in this case represents how long the buyer takes to complete the transaction after the seller posts a sell order, which carries important trading behavior and may be used to form trading strategies. Cautious traders may prefer to spend a significant amount of time looking for the best price of an item. Thus, the edges they construct may have a relatively long ToE. Other traders may complete a transaction as soon as goods appear on the

market, therefore resulting in a significantly shorter ToE. It is possible for buyers to complete the transaction with one of multiple sell orders posted by the same seller at different times, in which the ToE serves as discriminative information. Should ToE be neglected and merely reduced to the dynamic connectivity changes among vertices, the above trading patterns and strategies would be totally lost.

There are two major challenges in jointly embedding the dynamic linkage evolution and ToE for preserving the temporal evolutionary patterns of a dynamic graph. The first challenge is capturing and learning the multivariate linkage evolution patterns of a dynamic graph from their local dynamic instances, which is the snapshot graph, in an interpretable manner. The vertices' connections and ToEs in every snapshot graph are highly dynamic, therefore making it difficult to reconstruct the global evolution process of a dynamic graph from the snapshot graph sequence in an interpretable manner. Another challenge is preserving the temporal dependency among vertices while embedding ToE. If the ToEs are aggregated for each vertex directly and appended with other vertex attributes, which is a common approach for embedding vertex attributes in static graphs [101] [112], the temporal dependency among vertices will gradually be lost due to information loss through aggregation [28]. Therefore, the embedding algorithm should maximally prevent vanishing temporal dependency while embedding ToE.

To address the above challenges, I first model the dynamic graph as a sequence of snapshot graphs with ToE for every edge. I then propose a matrix factorization based **T**ime **C**apturing **D**ynamic **G**raph **E**mbedding algorithm named **TCDGE**, which infers a common latent space for capturing the structural and temporal evolution of the dynamic graph and encodes them into representation vectors. This approach differs from TNE [120], which embeds the snapshot graphs into separate latent spaces, since I learn a common latent space from every snapshot graph for representing the vertices' dynamic connections. When vertex connections evolve, their projected positions in the latent space will change accordingly. Therefore, vertices' moving trajectories

within the common latent space reflects their evolutionary patterns in the dynamic graph.

In order to embed ToE into the representations and preserve the temporal dependencies among vertices, I first concatenate the representation of every two vertices that form an edge as features for representing their temporal dependency. I then regard the ToE as discriminative information and co-train a linear regressor using the above features while learning the common latent space. The optimization algorithm I present in this chapter is generic for any linear regressor such as the LASSO regression, the ridge regression, and the elastic net regression. Finally, vertices' temporal dependency and ToE will be gradually embedded into the representations as well as the latent space during co-training.

To overcome the bottleneck of time efficiency in factorizing large-scale matrices, I optimize the latent space and the representation of the vertices by a projected gradient approach. Meanwhile, I propose a singular value decomposition (SVD) based approach to initialize the embedding algorithm. It not only helps boost the convergence speed for the algorithm but also prevents it from converging to a meaningless local optimal. Inspired by negative sampling, I introduce negative samples to co-train the regression model. Negative samples are constructed as any two vertices without any edges between them and I set their ToE to zero. This indicates that these two vertices have no temporal dependencies in the snapshot graph. Consequently, the TCDGE algorithm is very time efficient and scalable, even though the model is complicated with high-order polynomials.

The contributions are highlighted as follows:

- I propose a matrix factorization based dynamic graph embedding algorithm to embed the temporal linkage evolution by learning a common latent space for capturing the global evolutionary patterns throughout the sequence of snapshot graphs while co-training a linear regressor, i.e., LASSO, to embed ToE for pre-

serving vertices' temporal dependency. This approach differs from end-to-end embedding algorithms, which usually are black boxes, by interpretively capturing vertices' temporal linkage evolution as their moving trajectories within the latent space.

- I initialize the embedding algorithm by an SVD-based method and introduce negative samples to co-train the linear regressor. Thus, the embedding algorithm is very time efficient and scalable.

- I propose a new task, namely time-aware link prediction, to validate the effectiveness of dynamic graph embedding algorithms in preserving the temporal dynamics.

- I conduct experiments on three public datasets over four machine learning applications. The experimental results show that the proposed model achieves performance improvements of 17.00%, 22.91% and 11.88%, respectively, over the state-of-the-art baselines in vertex classification, ToE prediction, static and time-aware link prediction.

2.2 Literature Review

Starting with DeepWalk [71], numerous static graph embedding methods have been proposed to encode the graph structure and attributes such as high-order proximities [32] [14], vertices' centrality [18], vertex and edge attributes [112] [28], text semantics [101] [100], and communities [92] [89]. In addition to embedding a single homogeneous graph, EOE [99] and HWNN [81] infer a common latent space for respectively embedding coupled heterogeneous graphs and hypergraph. In addition to these unsupervised methods, there are several works focusing on task specific graph representation learning [51] [111]. It simultaneously train a discriminator or classifier using the labels of edges or vertices while learning the embeddings. The discrimi-

nator serves as a supervisor to make the final learned representation robust enough for discriminating the labels in specific applications. I borrow the idea of learning discriminative information while embedding the graph structure to co-train a linear regressor for encoding the ToE and temporal dependencies of vertices into the final representations.

In dynamic graph embedding, the main issue becomes handling the dynamic evolving nature of vertices and edges, and encoding their evolutionary patterns. Existing works learn the structural differences of a graph at different timestamps by either matrix factorization or deep learning approaches. For matrix factorization approaches, TNE [120] is a pioneering work that factorizes the consecutive snapshot graphs into different latent spaces with a temporal smoothness regularization. TMF [110] learns the first-order neighborhood information while factorizing the adjacency matrices of snapshot graphs. DHPE [119] employs the generalized SVD to preserve the high-order proximities and Timers [114] explores the timing of restarting SVD to overcome the error accumulation while embedding the dynamic graph. However, they fail to preserve the global structural evolution of the whole dynamic graph over time. In addition, none of them embed temporal information of vertices and edges like ToE with the structural evolution.

There exist deep learning methods that capture the specific evolution process in dynamic graphs. DynamicTriad [118] models the triad closure process when a graph evolves. HTNE [121] models the neighborhood formation sequences as a Hawkes Process with a time-aware weights. EPNE [90] learns the periodic linkage evolution patterns by causal convolutions. However, these specific dynamic processes merely exist in some particular graphs. For example, the triad closure process is not common in other networks except social networks, thus leading to poor performance. I embed temporal linkage evolution without pre-assuming any dynamic processes and give an interpretation about what happens in the latent space when the dynamic graph evolves over time.

There are also methods that approach the graph evolution process by incrementally appending out-of sample vertices or edges into the existing in-sample graph. DepthLGP [58] first encodes the network properties by a high-order Laplacian Gaussian Process using in-sample vertices, and then trains a deep neutral network to transform the latent network properties states into the final representations. GraphSAGE [35] leverages the features from a vertex's local neighborhood for new coming vertex by using graph convolutional networks. MVC-DNE [102] incorporates both the graph structure and the vertex properties for learning embeddings on incomplete graph. DynGEN [29] adopts auto-encoders to incrementally handle the growing graph and its extended version Dyngraph2Vec [26] trains a LSTM to capture the evolution throughout snapshot graphs. DySAT [74] employs the self-attention mechanism to capture the structure difference throughout the snapshot graph sequence instead of using LSTM. DynGraphGAN [98] learns long-term structural evolution via adversarial training. However, none of them model the ToE and temporal dependencies of vertices, thereby failing to preserve the complete evolutionary pattern of the dynamic graph in both structural and temporal domains, which is one of the main contributions of this work.

2.3 Problem Definition

In this section, I give a complimentary definition of dynamic graphs with synchronous evolution and then properly formulate the dynamic graph embedding problem.

Since $V_t \subseteq V$ for any t, the network structure in G_t evolves over time which also leads to G evolving. At time t, the edge $e_{i,j}^{t,\delta}$ links the upcoming vertex v_i^t to an existing one $v_j^{t'}$ which joins the graph at time t'. The temporal dependency among vertices is reflected by the ToE δ. It is possible for v_i^t to form edges with the same vertex appearing at different times $v_j^{t'}$ and $v_j^{t''}$. These two edges link the same vertex pair but have different ToE δ, which gives the dynamic graph the ability to distinguish the

edges between the same pair of vertices but established at two different timestamps.

Definition 1. ***Dynamic Graph with Synchronous Evolution.*** *A dynamic graph $G = \{G_{t_1}, G_{t_2}, \cdots, G_{t_n}\}$ is a sequence of directed or undirected snapshot graphs G_t, where $G_t = (V_t, E_t, W_t)$ is a snapshot graph at time $t \in \{t_1, t_2, \ldots, t_n\}$. V_t is a subset of the vertex set $V = \{v_1, v_2, \ldots, v_m\}$. The edge $e_{i,j}^{t,\delta} = (v_i^t, v_j^{t'}, \delta) \in E_t$ in G_t represents the connection between an upcoming vertex v_i^t joining at time t and an existing vertex $v_j^{t'}$ appearing at time t', where $i, j \in \{1, 2, \ldots, m\}$, $t' \leq t$ and $\delta = t - t'$ is the ToE of $e_{i,j}^{t,\delta}$. Each edge $e_{i,j}^{t,\delta}$ is associated with an edge weight $w_{i,j}^{t,\delta} \in W_t$.*

Definition 2. ***Dynamic Graph Embedding.*** *Given a dynamic graph $G = \{G_{t_1}, G_{t_2}, \cdots, G_{t_n}\}$ and assuming that the maximum number of vertices m is known, the objective is to learn a mapping function $f : v \mapsto r_v \in \mathbb{R}^k$ for $\forall v \in V$ such that r_v preserves the temporal linkage evolution of vertex v in terms of the dynamic connection changes and temporal dependency, where k is a positive integer indicating the dimension of the representation r_v.*

Definition 1 provides a generic description of the dynamic graph. When $t_n = 1$, the dynamic graph G degenerates into a static graph. If I assume $t = t' + 1$ for all edges, G becomes a continuous-time dynamic graph defined in [65]. If I assume $t' = t$, G becomes a structure evolving dynamic snapshot graph sequence which is adopted by most of the approaches in dynamic graph embedding literature [120] [110] [118] [119] [29] [26] [35] [98] [58]. When I assume $t' = t$, $V_t \subseteq V_{t+1}$ and $E_t \subseteq E_{t+1}$, G becomes a growing graph, where the vertices and edges are only appended to the graph but not removed. This definition of a dynamic graph is generic and captures both the structure and temporal dynamics.

2.4 Capturing the Evolution of Dynamic Graphs

In this section, I introduce the intuitions of capturing the evolution of a dynamic graph and interpret what happens in the latent representation space when the dynamic graph evolves.

Since each snapshot graph G_t is an instance of the dynamic graph G at time t, the dynamic change throughout the snapshot graph sequence exactly reflects the evolution of G. From a vertex point of view, this evolution process consists of the sequential changes of vertices' connections with their corresponding ToE. Embedding the dynamic graph G becomes inferring a latent space H with k dimensions that maximizes the retention of vertices' temporal connections and attributes. When projecting the snapshot graph G_t into the latent space H, every vertex in G_t obtains a response vector r_t, which is its embedding, showing its position in H. If vertices have similar connectivity and ToE, they should be close to each other in H, which means the distance between their embeddings is small.

When either vertices' connections evolve or their ToE changes, resulting from the evolution of the dynamic graph, their embeddings will change accordingly, therefore causing their position in H to move. The trajectory of every vertex in H carries its evolution process throughout the snapshot graph sequence. An example of this idea is shown in Fig. 2.1, where vertices c and d have similar connectivity as well as ToE among their connections so that their embeddings in the latent space H should be close to each other, and their moving trajectories are also similar. Since the silent vertices disconnect from any existing vertices, they should be projected to the same position in H no matter which snapshot graph they leave. The connectivities of vertices a and c are different in the three snapshot graphs resulting in different temporal linkage evolution, which leads to their moving trajectories being far away from each other. Finally, the embedding of any vertex v that preserves its temporal linkage evolution is obtained by Eq. (2.1), and represents its moving trajectory in H,

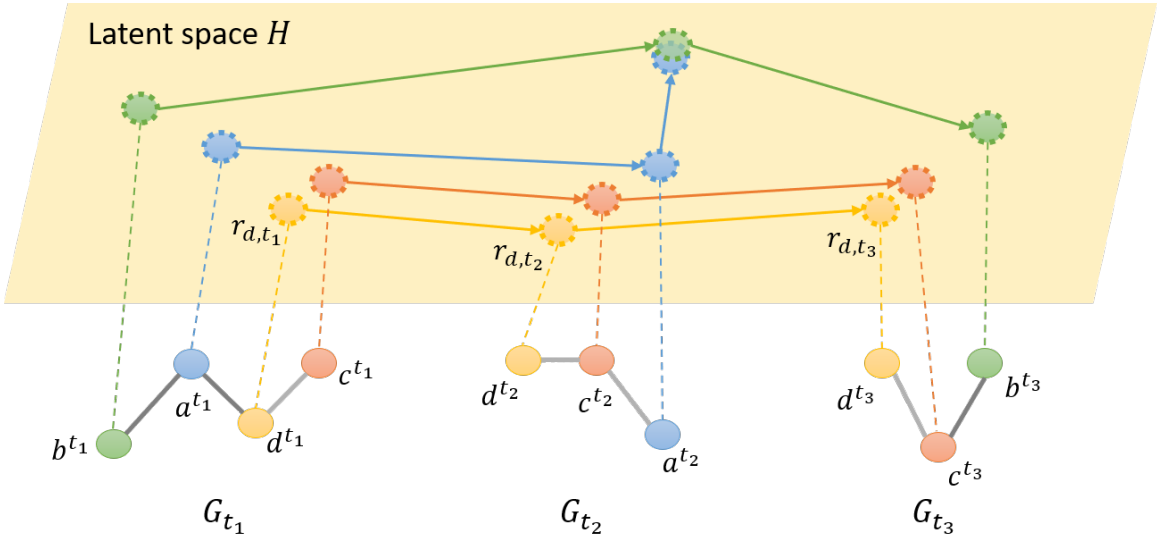

Figure 2.1: An illustration of the evolution of a dynamic graph. a, b, c, and d are vertices in a dynamic graph $G = \{G_{t_1}, G_{t_2}, G_{t_3}\}$, where their edge colors represents different ToE.

where $r_{v,t} \in \mathbb{R}^k$ is its learned representation from the snapshot graph G_t, and T is a transpose operator.

$$r_v = [r_{v,t_1}^T, r_{v,t_2}^T, \cdots, r_{v,t_n}^T]^T \tag{2.1}$$

In the next section, I will propose the dynamic graph embedding model and an optimization algorithm to efficiently infer the latent space H for embedding vertices' temporal linkage evolution.

2.5 Embedding Temporal Linkage Evolution

In this section, I present the details of the proposed time capturing dynamic graph embedding (TCDGE) model for encoding vertices' temporal linkage evolution as representations. Plus, I illustrate the optimization algorithm and training procedure to efficiently train the TCDGE model.

Table 2.1: Notations for time capturing dynamic graph embedding

Symbols	Description
G_t	A snapshot graph at time t, $t = t_1, t_2, \ldots, t_n$
m	The maximum number of vertices in G
$A_t \in \mathbb{R}^{m \times m}$	The adjacency matrix of G_t
$\hat{A}_t \in \mathbb{R}^{m \times m}$	1-step probability transition matrix obtained from A_t
$M_t \in \mathbb{R}^{m \times m}$	The high-order proximity matrix of G_t
$M_t(u) \in \mathbb{R}^{m \times 1}$	The high-order proximity vector of vertex u at t
$H \in \mathbb{R}^{m \times k}$	The inferred latent representation space
$W_t \in \mathbb{R}^{k \times m}$	The learned representation matrix at t
$W_t(u) \in \mathbb{R}^{k \times 1}$	The representation vector of vertex u at t
$y^t_{u,v} \in \mathbb{R}$	The ToE of an edge linked vertices u and v at t
$x \in \mathbb{R}^{(2k+1) \times 1}$	The learned coefficients of a linear regressor

2.5.1 Time Capturing Dynamic Graph Embedding Model

Before introducing the TCDGE model to solve the challenges, I first list the notations that will be used in the remainder of this chapter in Table 2.1.

The representations of vertices in a latent space should reconstruct the original dynamic graph reasonably well with the inverted latent space projector. Thus, I minimize the quadratic reconstruction loss under non-negative constraints for inferring the common latent space H and encode the representations of vertices in each snapshot graph G_t:

$$\arg\min_{H, W_t} \frac{1}{2} \sum_{t=1}^{n} \|G_t - HW_t\|_F^2 \quad s.t. \quad \forall W_t \geq 0, \quad H \geq 0 \tag{2.2}$$

Adjacency matrices are commonly used to capture the linkage information among vertices in a graph. However, the adjacency matrices of real world graphs are usually very sparse such as those for information networks, transaction networks, etc., which

introduces bias into machine learning algorithms and leads to imprecise results [30]. Additionally, the adjacency matrix only captures 1-step direct connections of vertices and is weak at representing the high-order neighborhood structure of the graph. One common approach to overcome this issue is to extract the high-order proximities of a graph from its adjacency matrix [101, 14]. In this chapter, I employ high-order proximity matrix M_t of G_t as input, where

$$M_t = \hat{A}_t + \hat{A}_t^{\,2} + \cdots + \hat{A}_t^{\,m} \tag{2.3}$$

and $\hat{A}_t$ is the 1-step probability transition matrix obtained from the adjacency matrix A_t after a column-wise normalization. If a vertex leaves G_t, meaning that it has no connection with any existing vertices at t, I define it as a silent vertex and all elements in its corresponding A_t column are zero. Consequently, its corresponding vector in M_t is also a zero vector leading the optimally learned representations to also be zero vectors. Finally, the evolving structure of G is preserved in a sequence of high-order proximity matrices M_t. By factorizing them, I infer a common latent space H and encode the structural dynamics of the dynamic graph into the representation W_t.

In solving the second challenge and further capturing the temporal dynamics, which are the temporal dependencies of every pair of vertices carried by the ToE of their linked edges, the objective is to embed the ToE into the representations W_t while factorizing M_t. I regard every single edge as a data sample to encode their ToE individually, which is different from treating all edges in a snapshot graph as a matrix M_t for embedding the graph structure. Inspired by discriminative embedding [51] [111], I treat ToE as "supervised" information to co-train a linear regressor while encoding the representations W_t. In other words, I employ the ToE to guide the embedding process and transfer it into the learned representations. Specifically, I constrain the learned W_t such that it should have the ability to simultaneously reconstruct the graph structure and accurately predict the ToE using the co-trained regressor x. Given the ToE of an edge connecting two vertices u and v, I concatenate their representation vectors $W_t(u)$ and $W_t(v)$ together as the feature of their corresponding

edge to co-train a linear regressor for estimating its ToE as follows:

$$J_{tp} = \sum_{t=1}^{n} \sum_{u,v} \left(y_{u,v}^{t} - \begin{bmatrix} W_t(u)^T & W_t(v)^T & 1 \end{bmatrix} x \right)^2 + \alpha \phi(x) \tag{2.4}$$

where $\phi(x)$ is a regularization of x and $\alpha > 0$ is a regression parameter. 1 is a constant for linear regression. J_{tp} is a LASSO regressor when $\phi(x) = \|x\|_1$, and it becomes a ridge regressor or an elastic net regressor if $\phi(x)$ is $\|x\|_2^2$ or $\|x\|_2^2 + \alpha' \|x\|_1$ respectively.

The temporal dependencies of u and v are embedded into their corresponding representations by the co-trained regressor since the representations of both source and target vertices are involved to regress the ToE of the edge they formed. If there does not exist any edges between u and v at time t, I set the corresponding $y_{u,v}^{t} = 0$, indicating that there is no temporal dependency between these two vertices at time t. When the dynamic graphs are undirected, I let $y_{u,v}^{t} = y_{v,u}^{t}$ so that every pair of vertices corresponds to the same ToE no matter how I concatenate their representation $W_t(u)$ and $W_t(v)$. In addition, if u appears multiple times in the dynamic graph, such as a seller that posts multiple selling announcements at different times in a dynamic transaction network that I mentioned in Section 2.1, ToE is exactly the unique discriminative information for the new coming vertex v, identifying which u it connects to. Such temporal dependencies between u and v are accurately preserved by the co-trained regressor which adopts the concatenation of their representations $W_t(u)$ and $W_t(v)$ as a feature to regress their corresponding ToE.

The co-trained linear regressor x allows this approach to identify the exact source vertex by estimating the ToE when performing link prediction. Although existing approaches can achieve the same goal by training an extra discriminator using well learned representations, their performance is not satisfactory due to the absence of discriminative information, such as ToE, for identifying the source vertex while learning the embeddings (please refer to the experimental results in Section 2.6.5). Therefore, the learned representation W_t has the ability to reconstruct the dynamic graph structure and preserve the temporal dependencies of vertices by approximating the ToE

of every edge.

Lastly, I assume that the graph evolves smoothly instead of being totally reconstructed at every time step. Thus, I penalize vertices's sharp changes of position in the latent space by minimizing the ℓ_2 distance between representations in two consecutive snapshot graphs:

$$\begin{aligned} J_{sm} = &\sum_{t=1}^{n} \sum_{u} \left(1 - W_t(u)^T W_t(u)\right)^2 \\ &+ \sum_{t=2}^{n} \sum_{u} \left(1 - W_t(u)^T W_{t-1}(u)\right)^2 \end{aligned} \tag{2.5}$$

In order to maintain stability when factorizing H and W_t from M_t, I employ quadratic regularizations $J_{reg} = \|H\|_F^2 + \sum_{t=1}^{n} \|W_t\|_F^2$ to prevent H and W_t from becoming sparse rapidly. Therefore, the overall TCDGE model is

$$\underset{H \geq 0, W_t \geq 0, x}{\arg\min} \frac{1}{2} \sum_{t=1}^{n} \|M_t - H W_t\|_F^2 + \frac{\lambda_1}{2} J_{reg} + \frac{\lambda_2}{2} J_{sm} + \frac{\lambda_3}{2} J_{tp} \tag{2.6}$$

where $\lambda_1 > 0$, $\lambda_2 > 0$, and $\lambda_3 > 0$ are model parameters. It co-trains a linear regressor to embed the ToE y, which carries the timespan of edges and temporal dependencies of vertices, into the representation W_t while encoding the high-order proximities by factorizing M_t for simultaneously preserving the structural dynamics. Since the ToE is an attribute of the dynamic graph and naturally exists, the proposed TCDGE is still an unsupervised representation learning approach.

2.5.2 Optimization Algorithm

In this subsection, I will explain how the optimization problem (2.6) was solved in detail. I aim to find the optimal latent space H, the representations of vertices W_t and the regression coefficient x. It is suitable to use an alternating directions method to solve this optimization problem by fixing H and x to solve W_t followed by fixing W_t to update H and x.

Optimizing Vertex Presentation W_t

Since $W_t(u)$ and $W_t(v)$ are a part of W_t, it is difficult to handle the integrated vector $[W_t(u)^T, W_t(v)^T, 1]$ in J_{tp} when solving for W_t. Thus, I let $x = [x_u^T, x_v^T, x_0]^T$, where $x_u \in \mathbb{R}^{k\times 1}$, $x_v \in \mathbb{R}^{k\times 1}$, and $x_0 \in \mathbb{R}$, and rewrite J_{tp} as

$$J_{tp} = \sum_{t=1}^{n} \sum_{u,v} \left(y_{u,v}^t - W_t(u)^T x_u - W_t(v)^T x_v - x_0\right)^2 + \alpha\phi\left(x\right) \tag{2.7}$$

I obtain the objective function of optimizing W_t as Eq. (2.8), which is a fourth-order polynomial and is non-convex.

$$\begin{aligned} \arg\min_{W_t \geq 0} \frac{1}{2} \sum_{t=1}^{n} \|M_t - HW_t\|_F^2 + \frac{\lambda_1}{2} \sum_{t=1}^{n} \|W_t\|_F^2 + \frac{\lambda_2}{2} J_{sm} \\ + \frac{\lambda_3}{2} \sum_{t=1}^{n} \sum_{u,v} \left(y_{u,v}^t - W_t(u)^T x_u - W_t(v)^T x_v - x_0\right)^2 \end{aligned} \tag{2.8}$$

Therefore, I adopt a block coordinate descent approach to solve W_t. When updating $W_t(u)$ for each vertex u at time t, I fix the H, x, and $W_t(v)$ of all the other vertices v at time t as well as all the representations W that are not at time t. Consequently, the W_t problem becomes a convex optimization problem as shown in Eq. (2.9).

$$\begin{aligned} \arg\min_{W_t(u) \geq 0} f(W_t(u)) = \arg\min_{W_t(u) \geq 0} \frac{1}{2} \|M_t(u) - HW_t(u)\|_2^2 + \frac{\lambda_1}{2} \|W_t(u)\|_2^2 \\ + \frac{\lambda_2}{2} \left(\left(1 - W_t(u)^T W_t(u)\right)^2 + \left(1 - W_t(u)^T W_{t-1}(u)\right)^2 \right) \\ + \frac{\lambda_3}{2} \sum_{v} \left(y_{u,v}^t - W_t(u)^T x_u - W_t(v)^T x_v - x_0\right)^2 \\ + \frac{\lambda_3}{2} \left(y_{u,u}^t - W_t(u)^T x_u - W_t(u)^T x_v - x_0\right)^2 \end{aligned} \tag{2.9}$$

If I choose to ignore situations where vertices can link to themselves (self-links), the last term in Eq. (2.9) could be removed. I use the projected gradient methods [53] to solve this convex optimization problem and obtain the updating function of $W_t(u)$:

$$W_t(u) = \max\left\{W_t(u) - \beta \bigtriangledown f(W_t(u)), 0\right\} \tag{2.10}$$

where $\beta > 0$ is the learning rate and the gradient $\bigtriangledown f(W_t(u))$ satisfies

$$\begin{aligned}\bigtriangledown f(W_t(u)) &= H^T H W_t(u) - H^T M_t(u) + \lambda_1 W_t(u) \\ -\lambda_2 \left(W_t(u)\left(1 - W_t(u)^T W_t(u)\right) + W_{t-1}(u)\left(1 - W_t(u)^T W_{t-1}(u)\right)\right) \\ -\lambda_3 \sum_v \left(y^t_{u,v} - W_t(u)^T x_u - W_t(v)^T x_v - x_0\right) x_u \\ -\lambda_3 \left(y^t_{u,u} - W_t(u)^T (x_u + x_v) - x_0\right)(x_u + x_v)\end{aligned} \tag{2.11}$$

To ensure a sufficient decrease of Eq. (2.10) and to speed up convergence, I update the learning rate β with a scaling factor θ to make the new $W_t(u)$ satisfy

$$f(W_t^{i+1}(u)) - f(W_t^i(u)) \leq \sigma_1 \bigtriangledown f(W_t^i(u))^T \left(W_t^{i+1}(u) - W_t^i(u)\right) \tag{2.12}$$

where i is the number of iterations and σ_1 is a tolerance. With the proof by Bertsekas in [12], there always exists a $\beta > 0$ that satisfies the rule (2.12) and every limit point of $\{W_t^i(u)\}_{i=1}^{\infty}$ is a stationary point of the bound-constrained optimization problem (2.9) [53]. After optimizing every $W_t(u)$ for every vertex in G_t, an optimal W_t is obtained. The pseudo code for solving W_t is presented in Algorithm 1.

Optimizing Common Latent Space H

When fixing W_t and x, the H optimization problem can be addressed by solving

$$\arg\min_H h(H) = \arg\min_{H \geq 0} \frac{1}{2} \sum_{t=1}^{n} \|M_t - H W_t\|_F^2 + \frac{\lambda_1}{2} \|H\|_F^2 \tag{2.13}$$

This is also a convex bound-constrained optimization problem that is again solvable using the projected gradient method, which is similar to the approach I employed in solving W_t. The updating function of H is

$$H = \max\{H - \beta \bigtriangledown h(H), 0\} \tag{2.14}$$

where the gradient of $h(H)$ is

$$\bigtriangledown h(H) = \sum_{t=1}^{n} (H W_t - M_t) W_t^T + \lambda_1 H \tag{2.15}$$

Algorithm 1 The projected gradient algorithm of solving W_t.

Input: M_t, H, x, W_t^0, y^t, λ_1, λ_2, λ_3, $0 < \theta < 1$, $0 < \sigma_1 < 1$

Output: W_t

```
repeat
    for u = 1, 2, ..., m do
        β^0 = 0.01
        for i = 1, 2, ... do
            β^i = β^{i-1}
            if β^i satisfies Eq. (2.12) then
                repeat
                    β^i = β^i / θ
                until β^i does not satisfy Eq. (2.12)
            else
                repeat
                    β^i = β^i · θ
                until β^i satisfies Eq. (2.12)
            end if
            Update W_t(u) by using Eq. (2.10)
        end for
    end for
until converge.
return W_t
```

When optimizing H, I adopt the same learning rate updating strategy in solving W_t here to ensure sufficient decent under the condition (2.16). σ_2 is the tolerance and i is the number of iterations.

$$h(H^{i+1}) - h(H^i) \leq \sigma_2 \bigtriangledown h(H^i)^T \left(H^{i+1} - H^i\right) \tag{2.16}$$

Co-training Linear Regressor for Embedding ToE

Fixing H and W_t for all t to optimize x is a standard linear regression problem. When rewriting the J_{tp} in Eq. (2.4) in matrix form, I obtain the objective function of optimizing x by Eq. (2.17), where $Z = [Z_1^T, Z_2^T, \cdots, Z_n^T]^T$ and $y = [y_1^T, y_2^T, \cdots, y_n^T]^T$,

which is a standard linear regression problem.

$$\arg\min_x \frac{\lambda_3}{2} J_{tp} = \arg\min_x \frac{\lambda_3}{2} \|y - Zx\|_2^2 + \frac{\alpha\lambda_3}{2} \alpha\phi(x) \tag{2.17}$$

Z_t for $t = 1, \cdots, n$ contains the concatenated features of any pair of vertices in the snapshot graph G_t as showed in Eq. (2.18) and $y_t \in \mathbb{R}^{m^2}$ is the corresponding ToE. The standard algorithm can be directly applied to solve the linear regression problem with different regularization $\phi(x)$ and finally get x.

$$Z_t = \begin{bmatrix} W_t(1)^T & W_t(1)^T & 1 \\ W_t(1)^T & W_t(2)^T & 1 \\ \vdots & \vdots & \vdots \\ W_t(1)^T & W_t(m)^T & 1 \\ W_t(2)^T & W_t(1)^T & 1 \\ \vdots & \vdots & \vdots \\ W_t(2)^T & W_t(m)^T & 1 \\ \vdots & \vdots & \vdots \\ W_t(m)^T & W_t(m)^T & 1 \end{bmatrix} \in \mathbb{R}^{m^2 \times (2k+1)} \tag{2.18}$$

Since the connections of vertices usually evolve very frequently in a dynamic graph, which leads to substantial changes to the concatenated edge features but only has a slight impact on ToE, LASSO is very robust for embedding the ToE and unlikely to overfit. In the remainder of this chapter, I specifically employ the LASSO regressor, letting $\phi(x) = \|x\|_1$, for illustration. To obtain the optimal LASSO regressor x, I first let $g(x) = \frac{\lambda_3}{2}\|y - Zx\|_2^2$, and then compute its gradient by $\nabla g(x) = \lambda_3(Z^T Zx - Z^T y)$. Lastly, I employ the FISTA algorithm [6] to solve the LASSO problem and obtain an optimal x with

$$x = S_{\frac{\alpha\lambda_3}{2}}(x - \gamma\nabla g(x)) \tag{2.19}$$

where $S(\cdot)$ is a soft-threshold calculator. $\gamma = 1/\lambda_{max}(Z^T Z)$ where $\lambda_{max}(Z^T Z)$ is the maximum eigenvalue of $Z^T Z$ which is the smallest Lipschitz constant of $\nabla g(x)$. The pseudo code of the FISTA algorithms for optimizing x is presented in Algorithm 2.

Algorithm 2 A FISTA algorithm for optimizing x.

Input: Z, y, α, λ_3, x^0, τ^0

Output: x

1: $L = \lambda_{max}(Z^T Z)$

2: $\beta_3 = \frac{1}{L}$

3: **repeat**

4: $\quad x^{i+1} = S_{\frac{\alpha\lambda_3}{2}}(\gamma^i - \beta_3 \nabla g(\gamma^i))$

5: $\quad \tau^{i+1} = \frac{1+\sqrt{1+4\tau^{i^2}}}{2}$

6: $\quad \gamma^{i+1} = x^{i+1} + \left(\frac{\tau_i - 1}{\tau_{i+1}}\right)(x^{i+1} - x^i)$

7: **until** converge.

8: **return** x^{i+1}

2.5.3 Efficient Training Procedure and Convergence

The major limitation of matrix-factorization-based algorithm is computation complexity and scalability. Although the projected gradient algorithm and the FISTA algorithm are one of the most efficient method to compute matrix factorization and LASSO regression respectively, we further identify two bottlenecks in further improving the training efficiency and making TCDGE converge faster. One bottleneck is the initialization of W_t and H to make them close to the optimal point for reducing the training time while preventing them from sticking into meaningless local optima. The other bottleneck is that very large-scale training samples make the FISTA algorithm very time-consuming in computing the gradient $\nabla g(x)$. Training sets containing too many edges with zero ToE impair the training precision of linear regressor as well. Here, I present an initialization approach using singular value decomposition (SVD) and an efficient FISTA training procedure to address the above efficiency bottlenecks.

Initialization of W_t and H by SVD

The TCDGE algorithm cannot be initialized by randomly generated H^0 and W_t^0. Usually, M_t is a sparse matrix but randomly generated H^0 and W_t^0 are all dense

matrices. It will make either or both H and W_t become zero matrices after a few iterations. Thus, the algorithm stops at a local optimum and outputs meaningless results.

To avoid reaching the zero local optimal point, the initialized H^0 and W_t^0 should meet the requirement $\|M_t - H^0 W_t^0\|_F^2 \leq \|M_t\|_F^2$ [53]. Therefore, I adopt SVD to initialize H^0 and W_t^0 as follows. First, I decompose every M_t and obtain its left-singular matrix U_t, singular value matrix I_t, and right-singular matrix S_t. Then, I select the rectangular diagonal sub-matrix from I_t corresponding to the top k singular values, and the first k columns from U_t and S_t denoted as $I_{t,k}$, $U_{t,k}$ and $S_{t,k}$. Finally, I initialize H^0 and W_t^0 by:

$$H^0 = \frac{1}{n}\sum_{t=1}^{n} U_{t,k} \quad and \quad W_t^0 = I_{t,k} S_{t,k}^T \tag{2.20}$$

Using SVD to initialize the embedding algorithm prevents it from being stuck in the zero local optimum and allows it to pursue meaningful results.

Efficient Linear Regressor Training with Negative Sampling

To capture all of the temporal dependencies among the m vertices in a dynamic graph consisting of n snapshot graphs, $m^2 \times n$ training samples in Z are used to co-train the linear regressor in every time step. Because of the high dimensionality of Z, computing the gradient $\nabla g(x)$ is very time-consuming. Meanwhile, many vertices usually do not connect to each other in real cases. Thus, edges with zero ToE are much more common than nonzero ToE edges, which causes imbalance issues and impairs the precision of the co-trained regression model.

Inspired by negative sampling [64], I mark all edges with nonzero ToE as positive samples and randomly choose a set of zero ToE edges, following a uniform distribution, as negative samples to jointly train the regressor. Different from deep learning models that just select a very small number of negative samples based on the label difference for training, I restrict the number of negative samples to half of the number of positive ones because negative samples in this model indicate vertices having no temporal

dependency which is one of the most important pieces of information that should be learned by the regressor.

After negative sampling, the training samples in Z are dramatically reduced and positive samples become majorities, therefore saving the computational cost in calculating $\nabla g(x)$ and preventing the regressor from being dominated by the negative samples, which makes it converge quickly and precisely. I have tried selecting negative samples based on a probability distribution that is proportional or inversely proportional to the vertex degree but the experimental results show that this is rarely much different from following the uniform distribution.

Convergence and Stop Criteria

The overall work flow of the TCDGE algorithm is presented in Algorithm 3, which essentially is a block-wise coordinate descent algorithm. Therefore, its convergence can be guaranteed according to the proof of convergence of block-wise coordinate descent [84]. Both algorithms for optimizing W_t and H stop when they meet the condition in Eq. (2.21) and Eq. (2.22), which ensures the optimization outputs are close to a stationary point [53]. ϵ is a very small positive number. For the jth element a_j in vector a, $p(\cdot)$ equals the gradient at a_j if $a_j > 0$ else $p(\cdot)$ equals the negative gradient at a_j.

$$\|p(\bigtriangledown h(H^i))\|_2 \leq \epsilon \| \bigtriangledown h(H^1)\|_2 \tag{2.21}$$

$$\|p(\bigtriangledown f(W_t^i(u)))\|_2 \leq \epsilon \| \bigtriangledown f(W_t^1(u))\|_2 \tag{2.22}$$

The FISTA algorithm for embedding the ToE stops when the residual of x is less than a small positive number ϵ'.

2.6 Experimental Results and Analysis

In this section, I conduct extensive experiments to showcase the effectiveness and efficiency of the TCDGE algorithms in the data mining tasks of vertex classification,

Algorithm 3 The TCDGE algorithm.

Input: M_t, y, Z, x^0, τ^0, λ_1, λ_2, λ_3, α, $0 < \theta < 1$, $0 < \sigma_1 < 1$, $0 < \sigma_2 < 1$

Output: H, W_t, x

Initialize H^0 and W_t^0 by Equation 2.20
Initialize x by the FISTA algorithm
repeat
    for $t = 1, 2, \ldots, n$ **do**
        Update W_t by Algorithm 1
    end for
    Update H by the projected gradient algorithm
    Update x by Algorithm 2
until converge.
return H, W_t, x

ToE prediction, static link prediction, and time-aware link prediction.

2.6.1 Experimental Setting

Datasets

Three public real-world datasets are considered when validating the performance of TCDGE on data mining applications, whose statistics are presented in Table 2.2.

UCI Messages[1] [66] is an online communication network of students. A vertex represents a student that has sent or received messages. The ToE is the communication time interval between a pair of students. The communication lasts 7 months so that a dynamic graph containing 7 snapshot communication graphs has been built for capturing their dynamic communication behaviors.

[1] `http://konect.uni-koblenz.de/networks/opsahl-ucsocial`

Table 2.2: Statistics of datasets

Dataset	$\|V\|$	$\|E\|$	$\|G_t\|$	Mean ToE	Std ToE	#Classes
UCI Messages	1899	22640	7	0.7387 (days)	2.1762	-
Transaction	5881	35592	11	1.4637 (months)	1.9303	2
Co-authorship	10374	60101	5	1.3834 (years)	1.0414	3

Transaction[2] [48] is a bitcoin transaction network. A vertex is a trader who buys and sells bitcoins and an edge forms while two traders complete a transaction. The ToE is the time interval between buying and selling. Each snapshot graph carries the transactions in a 6 month period. Since bitcoin traders are anonymous, there is a need to maintain a record of their reputation to prevent transactions with fraudulent and risky traders. Traders rate each other's trustworthiness on a scale of -10 (total distrust) to +10 (total trust) with a step of 1 after completing each transaction, so that I label traders whose average score is above 1 as trustworthy while the rest are deemed untrustworthy. Finally, I obtain 1092 untrustworthy traders and 4789 trustworthy ones.

I derive a *Co-authorship*[3] network for publications from 2010 to 2014 in three research areas including networking (NW), data mining (DM) and artificial intelligence (AI) from the DBLP. A vertex is an author and two authors form an edge when they coauthor a chapter. The ToE indicates the time interval between co-authorship. I deem researchers that have coauthored with not less than 6 other authors and at least coauthored with one of them twice in that period. The snapshot graphs represent the co-authorship in every year. I label the vertices by their research areas which they published most in. Finally, I obtains 3405 authors in NW, 2909 authors in DM, and 4060 authors in AI.

Baseline Methods

[2] `https://snap.stanford.edu/data/soc-sign-bitcoin-otc.html`

[3] `http://projects.csail.mit.edu/dnd/DBLP/`

I benchmark the TCDGE algorithm to 7 state-of-the-art methods listed below using their published codes.

- *DeepWalk*[4] [71] is a static graph embedding algorithm that employs skip-gram to encode linkage relationships among vertices searched by the random walk. I tested the combination of hyper parameters given window sizes $ws \in \{5, 8, 10\}$, walk lengths $wl \in \{10, 20, 30, 40\}$, and numbers of walks $nw \in \{20, 40, 60\}$, and report the best results.

- *Temporal Network Embedding (TNE)*[5] [120] is a matrix factorization based dynamic graph embedding method that encodes the structure evolving patterns in different latent spaces. I tested the hyper parameter $\lambda \in \{0.01, 0.1, 1, 10\}$, and report the best results.

- *Timers*[6] [114] is an incremental SVD approach for dynamic graph embedding which overcomes the error accumulation issues by restarting SVD when the error margin exceeds a threshold. I use the default parameter settings $\theta = 0.17$.

- *DynamicTriad*[7] [118] preserves the triad closure process while embedding the structural evolution. I tested all combinations of hyper parameters $\beta_0, \beta_1 \in \{0.01, 0.1, 1, 10\}$, and report the best results.

- *GraphSAGE*[8] [35] is a graph convolutional network approach for embedding the structural evolution of a dynamic graph. I train a two layer model with respective neighborhood sample sizes 25 and 10, as described in the original paper. I test different aggregators including GCN, mean, mean-pooling, and LSTM and report the performance of the best performing aggregator in each dataset.

[4]https://github.com/phanein/deepwalk
[5]https://github.com/linhongseba/Temporal-Network-Embedding
[6]https://github.com/ZW-ZHANG/TIMERS
[7]https://github.com/luckiezhou/DynamicTriad
[8]https://github.com/williamleif/GraphSAGE

- *DynGEN*[9] [29] adopts a deep auto-encoder to embed the structure changes throughout the snapshot graph sequence. I train a two layer model and adopt the default parameter settings that are recommended by the authors.
- *DynG2vecAERNN*[9] [26] is an extension of DynGEN which first adopts a deep neutral network to encode the structure of each snapshot graph, and then employs an LSTM to embed the sequential evolution of every vertex throughout the snapshot graphs. A two layer model is trained with the default parameter setting as described in the original paper.

In order to verify the effectiveness of learning the common latent space H to capture the linkage evolution, I experiment with the TCDGE without embedding ToE by setting $\lambda_3 = 0$, namely TCDGE-noToE. Meanwhile, I test another variant TCDGE, namely TCDGE-wgToE, that adopts the ToE as weights of the adjacency matrix of each snapshot graph but does not co-train any regression model, thus verifying the effectiveness of the co-training approach.

Evaluation Metrics

I employ micro-F1 and macro-F1 scores as evaluation metrics for the task of vertex classification as seen below:

$$\text{Micro-F1} = \frac{2\sum_i \text{TP}_i}{\sum_i (2\text{TP}_i + \text{FP}_i + \text{FN}_i)} \tag{2.23}$$

$$\text{Macro-F1} = \frac{1}{c}\sum_i \frac{2\text{TP}_i}{2\text{TP}_i + \text{FP}_i + \text{FN}} \tag{2.24}$$

where TP_i, FP_i, and FN_i are the true positive, false positive, and false negative results of the ith predicted class, respectively. The macro-F1 score is the mean of the class-wise F1 score that is sensitive to the performance in classifying each individual class. The micro-F1 score measures the overall classification performance regardless of the accuracy in individual classes. Higher micro-F1 and macro-F1 scores indicate better vertex classification performance.

[9]https://github.com/palash1992/DynamicGEM

I evaluate the performance of ToE prediction by measuring the Root Mean Square Error (RMSE) between the predicted ToE and the ground truth as

$$\text{RMSE} = \sqrt{\frac{\sum_{y \in \mathcal{S}_{test}} (y - \hat{y})^2}{|\mathcal{S}_{test}|}} \tag{2.25}$$

where y denotes the real ToE in the test set $\mathcal{S}_{test}$ and $\hat{y}$ is the predicted one. $|\mathcal{S}_{test}|$ is the number of test samples in $\mathcal{S}_{test}$. The smaller the RMSE, the more accurate the ToE prediction.

In link prediction, I employ the average area under the curve (AUC) of the receiver operating characteristic (ROC) curve as the performance metric. The higher the AUC, the better the link prediction performance.

Parameter Setting

The experiments have been conducted with $k = 45$ as the dimension of the representation vector for both the TCDGE and all baselines in all testing datasets. For the parameters of TCDGE, I set the scaling factor $\theta = 0.5$, tolerance $\sigma_1 = \sigma_2 = 0.01$. I co-train a LASSO regressor for the TCDGE with initial regression parameter $\alpha = 1$ Since W_t will be updated at each time step, making Z change dynamically, the regression parameter α cannot be fixed. Otherwise, the LASSO cannot adequately fit the ToE by using the new Z at each time. In addition, the training error of LASSO will gradually accumulate so that the reconstruction error of the overall embedding model will progressively increase, thus leading to poor embedding results. I adopt θ to dynamically update α 10 times using the same updating strategies in the projected gradient algorithm for learning the best LASSO regressor x at each time. Finally, I report the best results by testing the combination of model parameters given $\lambda_1 \in \{0.001, 0.01, 1\}$ and $\lambda_2, \lambda_3 \in \{0.0001, 0.001, 0.01, 0.1, 1\}$ for the data mining tasks presented in the following subsections. All experiments are conducted on a standard workstation with 2 Intel Xeon Gold 6128 CPUs and 64GB RAM, and are implemented in MATLAB.

Table 2.3: Vertex classification results

	Transaction		Co-authorship	
	Micro-F1	Macro-F1	Micro-F1	Macro-F1
DeepWalk	0.8855	0.7229	0.5645	0.5684
TNE	0.8673	0.6777	0.4221	0.4064
Timers	0.7810	0.4896	0.3358	0.2965
GraphSAGE	0.6205	0.6256	0.4793	0.4537
DynamicTriad	0.8652	0.6737	0.5243	0.5189
DynGEN	0.8316	0.6680	0.5150	0.5065
DynG2vecAERNN	0.8140	0.4721	0.4681	0.5398
TCDGE-noToE	0.8621	0.7390	**0.6921**	**0.7220**
TCDGE-wgToE	0.8625	0.7402	0.6701	0.6885
TCDGE	**0.9032**	**0.7725**	0.6728	0.7079

2.6.2 Vertex Classification

Vertex classification aims to identify the unique label of vertices using their learned representations in the dynamic graph G. I first learn the representation of vertices in every snapshot graph G_t. Then, concatenate the representations W_t together by Eq. (2.1) for classification. A support vector machine (SVM) with a Gaussian kernel is trained by using these features to classify their corresponding labels. It tests the embedding algorithms' ability to capture the global graph evolutionary patterns in G for all timestamps. Since the UCI messages dataset does not contain vertex labels, I compare the classification performance in both bitcoin transactions and co-authorship datasets. I repeat the 5-fold cross-validation on both datasets 10 times and compare the average performance in macro-F1 and micro-F1 scores. I did not adopt any extra methods to handle the issues of unbalanced labels in the bitcoin transaction

dataset but straightforwardly train the SVM for testing the actual performance of the TCDGE algorithm in the case of label unbalanced classification. The results are shown in Table 2.3.

In the bitcoin transaction dataset, the TCDGE algorithm achieves the best performance, and outperforms the best baseline by 2.00% in micro-F1 scores and by 4.36% in macro-F1 scores. In the co-authorship dataset, TCDGE and its variants, TCDGE-noToE and TCDGE-wgToE, dramatically outperform all 7 other baseline methods. This indicates that capturing the moving trajectories of vertices in the common latent space, learned throughout the snapshot graph sequence by the proposed approach, embeds the evolution of a dynamic graph better than the baseline methods. In addition, the temporal evolution patterns captured by the approach work much better than the baselines in unbalanced label classification.

In the co-authorship dataset, TCDGE-noToE performs the best. This may be because the standard deviation of its ToE is relatively small meaning that the time intervals of co-authoring papers are not as significant as who they co-author with over time for classifying their research areas. Therefore, purely capturing the linkage evolution may be good enough for classifying authors' research areas from their co-authorship, and the TCDGE and TCDGE-wgToE achieve close performance, yet slightly worse than TCDGE-noToE but still much better than the baselines.

When people trade bitcoins, the time interval between transactions becomes important for measuring traders' trading behavior and strategies, which results in higher standard deviation of ToEs. Since the TCDGE algorithm successfully embeds both structural evolution and the temporal information of edges at the same time, it achieves the highest macro and micro F1 scores and dramatically outperforms the traditional models which merely capture the linkage information. Although TCDGE-wgToE leverages the ToE as weights of the adjacency matrices for embedding, the temporal dependency among vertices gradually diminishes during embedding due to the aggregation throughout the snapshot graph sequence, which is consistent with the

Table 2.4: Average RMSE of ToE prediction

	UCI Messages	Transaction	Co-authorship
DeepWalk	2.5934	2.1084	1.0395
TNE	2.5335	2.0932	1.0377
Timers	2.1536	2.1074	1.0428
GraphSAGE	2.1471	2.1004	1.0392
DynamicTriad	2.3329	2.3485	1.3425
DynGEN	2.4160	2.4053	1.0395
DynG2vecAERNN	2.1640	1.9756	0.9891
TCDGE-noToE	2.1957	2.0920	1.0371
TCDGE-wgToE	2.1595	2.0659	1.0452
TCDGE	**2.1419**	**1.7798**	**0.8967**

conclusion drawn in [28]. However, co-training the LASSO regressor has the ability to better preserve the temporal dependency among vertices and encode it into the final representation. Therefore, embedding the temporal dependency together with the structural evolution among vertices into a common latent space makes the learned representation vectors preserve the global structural and temporal evolutionary patterns from the whole dynamic graph, which is more discriminative and leads to better classification results.

2.6.3 ToE Prediction

The objective of ToE prediction is to estimate the ToE of an edge given the representation of its source and target vertices for testing how effectively the learned representations capture temporal information. The experiment is conducted under the leave-one-snapshot-graph-out cross-validation setting. Since the TCDGE co-trains a LASSO regressor simultaneously with the representation learning, a snapshot graph

is selected for testing at each round and I use the rest of the snapshot graphs to train the model until every snapshot graph serves as the testing graph once. When testing the baseline methods, I first generate all the representations from every snapshot graph, and then employ them to further train a LASSO regressor under the same cross-validation setting. I repeat each experiment 10 times and report the average RMSE.

The ToE prediction results are presented in Table 2.4. The TCDGE achieves a 12.85% lower RMSE on average against all baseline methods and outperforms the best baseline by 6.50% indicating that the temporal dynamics are preserved by the proposed co-training approach, which results in much lower ToE prediction errors than the baseline approaches that ignore it. The representations learned by the TCDGE carry both structural evolution of the dynamic graph and its ToE such that it is more effective when discriminating temporal information than those approaches that purely embed the graph structure, which leads to better performance in ToE prediction.

2.6.4 Static Link Prediction

Static link prediction aims to predict whether a pair of vertices will form an edge at time $t+1$, given their embeddings learned at t. This task ignores the joining time of source vertices, which is widely adopted by the existing work to test the performance of learned embeddings. Here I employ the cosine distance to measure the similarity of two vertices in the latent space and calculate the probability of forming a new edge by the sigmoid function. I predict the links in snapshot graph G_{t+1} by using the representation W_t under the same experimental settings as those of [120]. The performance is measured by the average AUC for predicting G_2 to G_n.

The results are reported in Table 2.5. Overall, the proposed TCDGE algorithm outperforms all baselines by 27.56% on average with respect to the AUC, and achieves

Table 2.5: Average AUC of static link prediction

	UCI Messages	Transaction	Co-authorship
DeepWalk	0.6619	0.9028	0.5977
TNE	0.6524	0.8264	0.5861
Timers	0.4943	0.4938	0.5156
GraphSAGE	0.5091	0.5624	0.5890
DynamicTriad	0.5187	0.4197	0.5950
DynGEN	0.6028	0.5826	0.4874
DynG2vecAERNN	0.4977	0.5218	0.4949
TCDGE-noToE	0.7003	0.9194	0.6044
TCDGE-wgToE	0.6969	0.9187	0.6037
TCDGE	**0.7314**	**0.9248**	**0.6142**

2.22% higher AUC than the best baseline method TCDGE-noToE on average in all three datasets. The baseline approaches only learn from the linkage information. However, the TCDGE algorithm not only learns the evolving patterns of who the vertices link to, but also embeds how they link by capturing their ToEs and temporal dependency such that the edges between the same pair of vertices but established at two different timestamps can be distinguished. Therefore, the TCDGE algorithm achieves better static link prediction performance in terms of higher AUC than all baselines.

2.6.5 Time-aware Link Prediction

Time-aware link prediction is a unique application for dynamic graph embedding, which aims to identify the joining time of existing vertices on top of the static link prediction. It performs two tasks at the same time. One is to predict whether a pair of vertices will form an edge at time $t + 1$ when given their representations

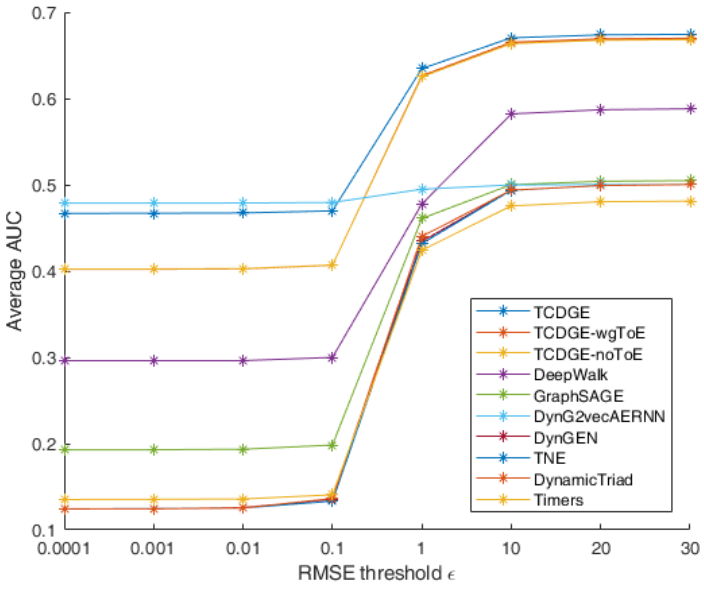

Figure 2.2: Time-aware link prediction results with varying threshold ϵ in the UCI message dataset.

at time t. The other is to predict the joining time of existing vertex to identify the unique one since it can join the dynamic graph several times. Specifically, data mining applications such as predicting which sell order will be completed by a buyer, predicting the future victims of fraud and when the fraud will happen, recommending items at an appropriate time, etc., can all be abstracted as time-aware link prediction applications.

Since the joining time of an existing vertex is equal to the difference between the ToE and the joining time of an upcoming vertex, predicting the joining time of existing vertices at the time when the upcoming one joins the dynamic graph is the same as predicting the ToE of the edge they form. Thus, I predict the ToE instead of the actual joining time of existing vertices in this experiment.

I conduct the experiment under the one-snapshot-graph-ahead cross-validation setting in which a snapshot graph G_t $(t > 1)$ is selected for testing at each round and I use

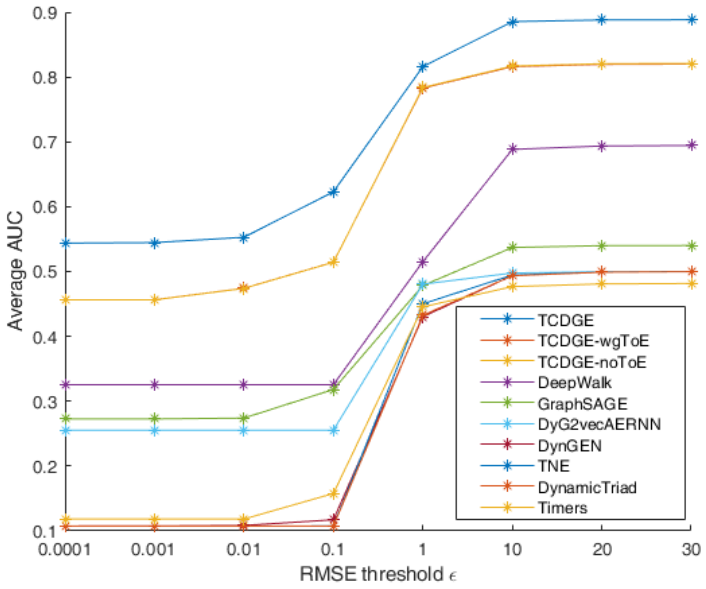

Figure 2.3: Time-aware link prediction results with varying threshold ϵ in the Transaction dataset.

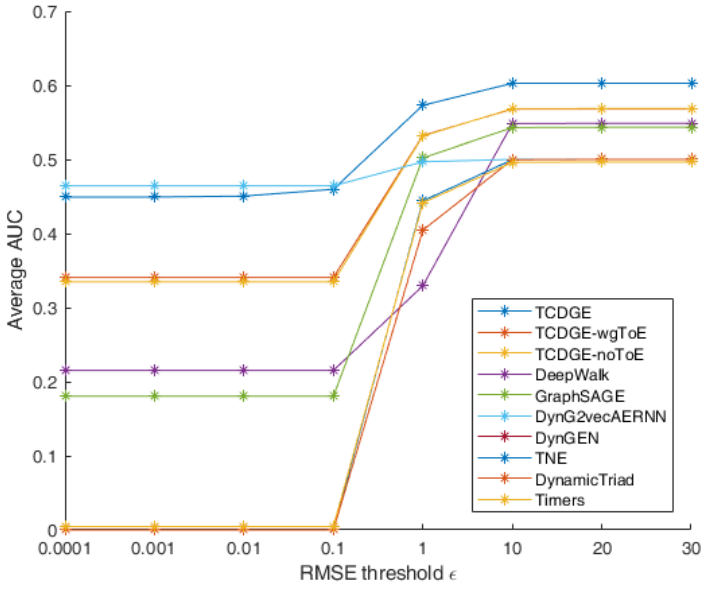

Figure 2.4: Time-aware link prediction results with varying threshold ϵ in the Co-authorship dataset.

the snapshot graph sequence $\{G_1, \cdots, G_{t-1}\}$ to train the model until every snapshot graph except G_1 serves as the testing graph once. Since none of the baselines can achieve the two goals in time-aware link prediction simultaneously, I employ the same cross-validation setting to obtain baselines' representations and then further train a LASSO regressor to predict the ToE. I adopt the same approach used in static link prediction to determine whether there exists an edge connecting a pair of vertices here.

A temporal link has been correctly predicted if and only if the model correctly predicts that a pair of vertices formed an edge and the RMSE of ToE prediction for this edge is less than a threshold ϵ. To test how the prediction accuracy of ToE affects time-aware link prediction, I perform time-aware link prediction in three datasets and test the threshold ϵ from 0.0001 to 30. The experiment repeats 10 times for each threshold and the average AUC are reported in Fig. 2.2 to Fig. 2.4.

The TCDGE performs the best in all three testing datasets when $\epsilon > 0.1$. It also achieves the highest AUC in the bitcoin transaction dataset and dramatically outperforms other baselines except DynG2vecAERNN in the remaining two datasets when $\epsilon \leq 0.1$. DynG2vecAERNN works better in ToE prediction than other baselines (refer to Table 2.4) but is comparatively much worse in link prediction (refer to Table 2.5) such that it achieves relatively high AUC with small ϵ but it cannot correctly predict more temporal links when relaxing the threshold ϵ. Although the TCDGE performs slightly worse than DynG2vecAERNN with small ϵ, it becomes the best of all when $\epsilon = 1$, and AUC increases slowly when $\epsilon > 1$. This indicates that the RMSE of ToE prediction for most temporal edges predicted by the TCDGE is less than 1. Consequently, the LASSO co-training approach preserves the temporal dynamics well while embedding the ToE, therefore resulting in superior performance in time-aware link prediction.

2.6.6 Parameter Sensitivity Analysis

The TCDGE defined by Eq. (2.6) is dependent on regularizer weights λ_1, λ_2, λ_3 and a hyperparameter k which is the dimension of the latent representation space as well as the dimension of the learned embeddings. The selection of λ_1, λ_2 and λ_3 highly depends on the input data and the selection approach has been illustrated in section 2.6.1. Therefore, I conduct sensitivity analysis on the hyperparameter k from 15 to 285 in vertex classification, ToE prediction, and static link prediction. The co-authorship dataset is adopted here because the scale is relatively large compared to the other two datasets and the number of vertices in the three categories are almost balanced, which is more common in daily life. I fix $\lambda_1 = 0.0001$, $\lambda_2 = \lambda_3 = 0.01$ and only vary k at each time. As shown in Fig. 2.5 to Fig. 2.7, when k increases, both F1-scores in vertex classification increase almost linearly and gradually converge. The RMSE of ToE prediction decreases exponentially and converges with increasing k. The average AUC of static link prediction is not sensitive to the dimension of the representations. Since all results eventually converge to the best case when the k is high enough, the TCDGE is not sensitive to the dimension of the common latent space k.

2.6.7 Convergence and Training Efficiency

I demonstrate the convergence of the TCDGE algorithm in the co-authorship dataset which has the highest number of vertices. Fig. 2.8 shows the loss of the objective function in Eq. (2.6), fidelity term $\frac{1}{2}\sum_{t=1}^{n}\|M_t - HW_t\|_F^2$, the LASSO regressor J_{tp} in Eq. (2.4) when $\phi(x) = \|x\|_1$, and the temporal smoothness regularization J_{sm} in Eq. (2.5).

The TCDGE converges in very few iterations because of the initialization set by the SVD and the effectiveness of the projected gradient method for solving W_t and H. The initialization approach not only prevents the TCDGE from being stuck in the

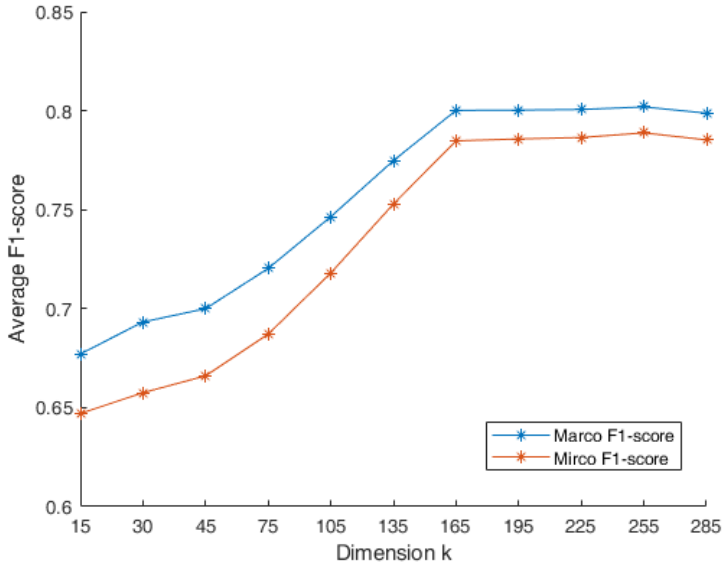

Figure 2.5: Average F1-score of vertex classification with varying the hyperparameter of dimension k in the Co-Authorship dataset.

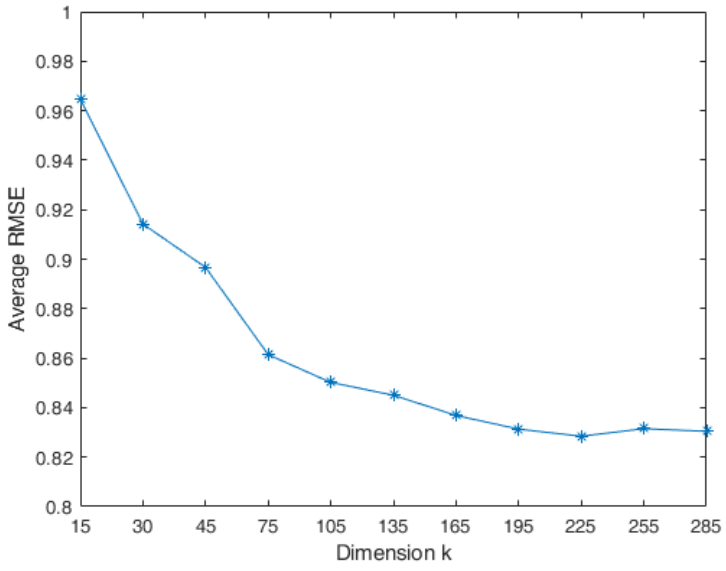

Figure 2.6: Average RMSE of ToE prediction with varying the hyperparameter of dimension k in the Co-Authorship dataset.

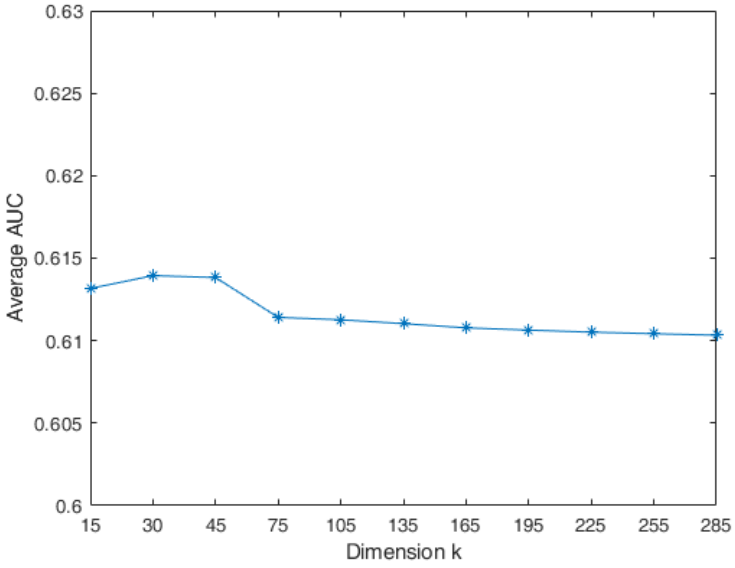

Figure 2.7: Average AUC of static link prediction with varying the hyperparameter of dimension k in the Co-Authorship dataset.

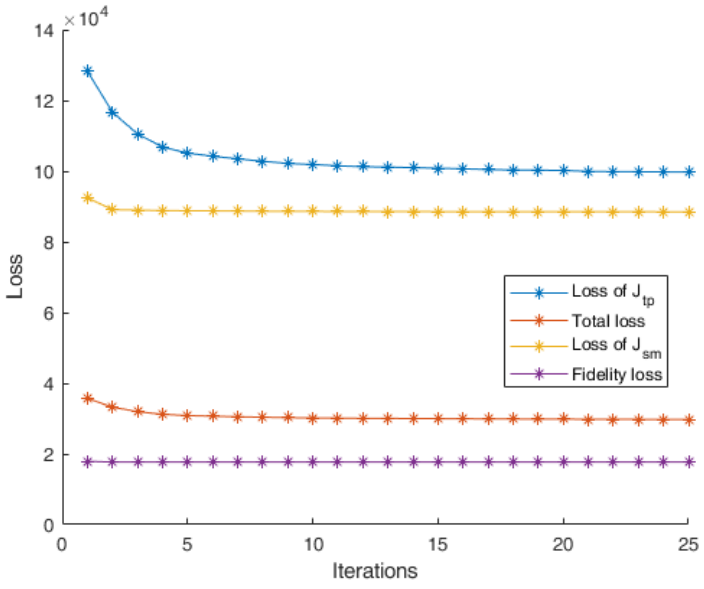

Figure 2.8: Convergence curves of TCDGE in the co-authorship dataset.

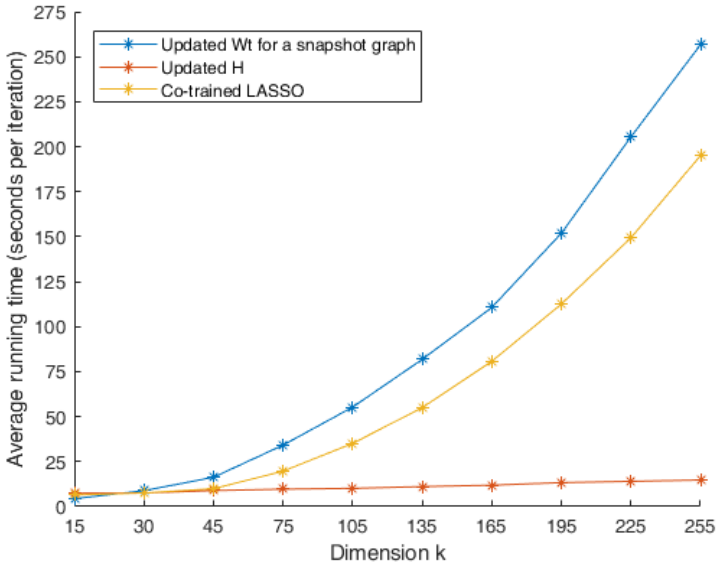

Figure 2.9: Training efficiency of TCDGE with varying k in the co-authorship dataset.

zero local optimum, but also generates an approximation of M_t upon initialization, which already decreases the loss of fidelity. In the projected gradient method, a scaler θ is employed to search for a good learning rate to ensure the sufficient decrease of the gradient, thereby boosting the convergence speed of the overall TCDGE algorithm.

Fig. 2.9 shows the average running time of each iteration while training the TCDGE by varying the hyperparameter k. As k increases, the running time for updating H at each iteration hardly increases. The running time of updating W_t and co-training x almost linearly grows with increasing k. It takes less than 1 minutes to finish updating the representation for over ten thousand vertices when $k <= 105$ and less than 5 minutes when $k = 255$.

Although the TCDGE model looks complex, it converges quickly in terms of a small number of iterations and a very short running time for encoding the representation W_t, learning the common latent space H, and co-training the LASSO regressor x, demonstrating the effectiveness of the projected gradient method and the the proposed efficient training procedure.

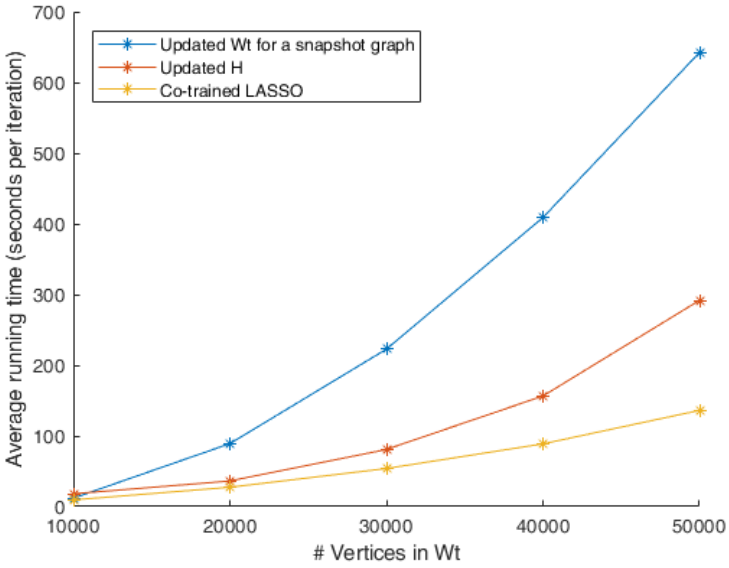

Figure 2.10: Scalability test results of TCDGE with varying number of vertices in the synthesized dataset.

2.6.8 Scalability of TCDGE

I synthesize two datasets on top of the co-authorship dataset to test the scalability of the TCDGE. One is to fix the number of snapshot graphs but augment the number of vertices in every snapshot graph by sampling vertices and edges in the other snapshot graphs as new vertices and edges of the current graph. This tests the scalability of the project gradient approach for updating W_t and the LASSO co-training. The experimental results are shown in Fig. 2.10. As the number of vertices in the snapshot graph increases, the running time for updating W_t grows almost linearly. The running time for co-training LASSO and updating H becomes slightly longer, but still much slower than the growth rate of updating W_t.

The other synthesized dataset fixes the number of vertices in every snapshot graph but augments the number of snapshot graphs to test the scalability of learning the common latent space H. I divide the vertices of each existing snapshot graph into 5-folds based on the degree of vertices. I take a fold from each existing snapshot

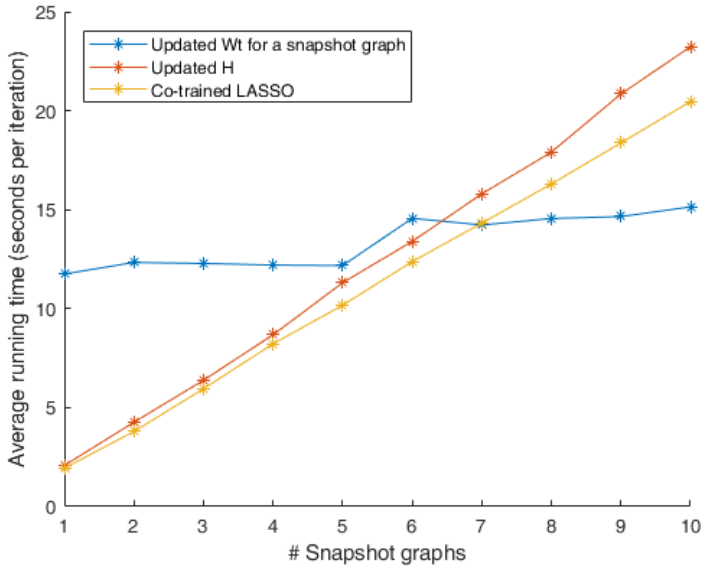

Figure 2.11: Scalability test results of TCDGE with varying number of snapshot graphs in the synthesized dataset.

graph without duplication to synthesize a new snapshot graph. In Fig. 2.11, the experimental results indicates that the running time of learning the common latent space grows linearly. Consequently, the TCDGE algorithm has very good scalability although the embedding model is complicated with high-order polynomials.

2.7 Chapter Summary

In this chapter, I model a dynamic graph as a snapshot graph sequence appended with ToE for every edge, which captures the multivariate dynamics over vertices. A time capturing dynamic graph embedding model is presented to embed the synchronous linkage evolution while accounting for their evolution duration. In particular, it encodes every vertex's temporal connection changes as its moving trajectories within the inferred common latent representation space. The experimental results show that this method can achieve significant performance improvements over existing state-

of-the-art approaches. The paper [104] arising by this study has been accepted to publish in the IEEE Transactions on Knowledge and Data Engineering (TKDE) in 2021.

Chapter 3

Time-aware Dynamic Graph Embedding for Asynchronous Structural Evolution

In this chapter, I study the feature representation to fully capture and embed the asynchronous evolution patterns in a dynamic graph, dealing with the challenges of multivariate dynamics. Despite the benefits of learning vertex representations (i.e., embeddings) for dynamic graphs, existing works merely view a dynamic graph as either sequential or synchronous changes within the vertex connections, neglecting the crucial asynchronous nature of such dynamics where the evolution of each local structure starts at different times and lasts for various durations. To capture the asynchronous structural evolutions within the graph, I innovatively formulate the dynamic graphs as temporal edge sequences associated with the joining time of vertices (ToV) and timespan of edges (ToE). Then, a time-aware Transformer is proposed to embed vertices' dynamic connections and ToEs into the learned vertex representations. Meanwhile, I treat each edge sequence as a whole and embed its ToV of the first vertex to further encode the time-sensitive information. Extensive evaluations

on several datasets show that the proposed approach outperforms the state-of-the-art in a wide range of graph mining tasks. At the same time, it is very efficient and scalable for embedding large-scale dynamic graphs.

3.1 Introduction

Graph are one of the most widely used data structures to represent pairwise relations between entities. In many real-world applications, these relations naturally change over time, making the graph dynamic. For example, in an online forum, users can post messages at any time and form a discussion network. Some of them join the discussion by replying or citing the published posts, thus forming edges among existing vertices. Some may choose to unfollow other users or become inactive in the discussion, and this will make them disconnected from other vertices. Since vertices can join, leave and re-join a network at any time, the dynamics of graphs are beneficial for modeling various types of data like traffic, financial transactions, and social media.

Dynamic graph embedding is an effective means to encode the evolutions of vertices' connections and properties into vector representations to facilitate downstream applications. In general, to capture the dynamic changes of vertex connections over time, existing approaches represent the dynamic graph as either a snapshot graph sequence (SGS) [26] [58] [59] [83] [87] [94] [98] [118] [119] [120] or neighborhood formation sequence (NFS) [11] [15] [24] [65] [70]. On one hand, SGS segments the graph information into several time slots, where a static snapshot graph is built to represent the node connectivity within each time slot. On the other hand, NFS captures dynamic graph information by using temporal random walk to sample the sequential connections amongst vertices.

Unfortunately, both SGS and NFS incur inevitable information loss about the dynamics properties of a graph, thus impeding the expressiveness of the eventually learned

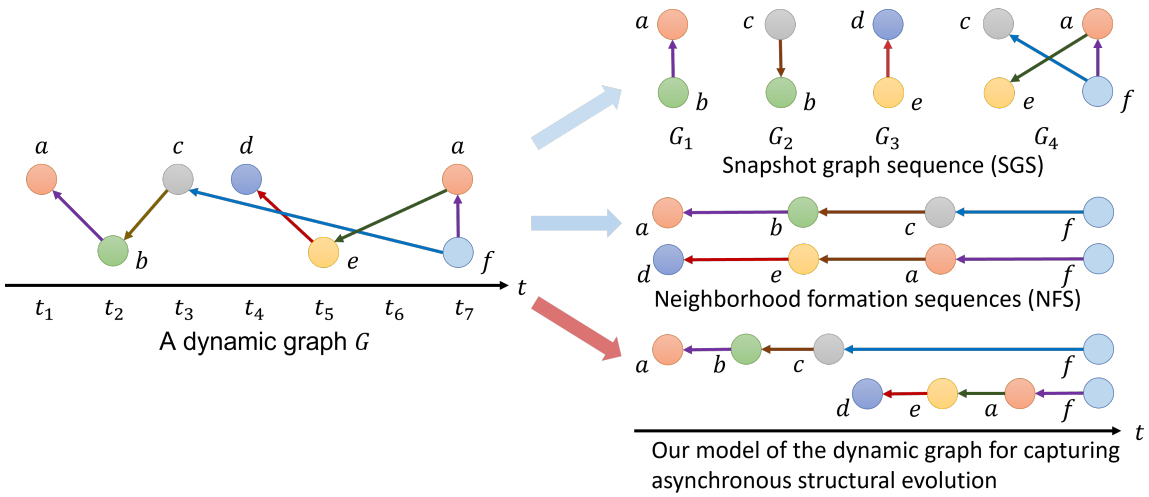

Figure 3.1: Modeling a dynamic graph with SGS, NFS, and the proposed one.

node embeddings. In Fig. 3.1, I provide an example on how SGS and NFS represent the dynamic graph G, respectively. For SGS, a snapshot graph is built for each of the 4 time slots ($[t_3, t_4]$ and $[t_5, t_6]$ are omitted as there are no new connections). Obviously, SGS tends to suffer from sparsity issues with a fine-grained time granularity, e.g., apart from G_4, the vertices and edge in G_1, G_2, and G_3 are too sparse for effectively learning node embeddings. Should these three snapshot graphs be merged to alleviate the sparsity, the subtle structural evolutions at different time steps will be neglected. Furthermore, a largely overlooked fact in dynamic graphs is that, the structural evolutions are asynchronous, i.e., the exact time consumed by each edge update varies in different cases. For example, on the left of Fig. 3.1, the connections among vertices $\{a, b, c, f\}$ and $\{d, e, a, f\}$, evolve asynchronously since the edges have different starting times and durations. As a result, modeling G as a SGS shown in Fig. 3.1 will interrupt the continuous evolutions of vertices $\{a, b, c, f\}$ with an irrelevant snapshot G_3. Therefore, incorrect graph dynamics will be embedded, as such asynchronous temporal information is totally lost when treating a dynamic graph as an array of static snapshots.

Compared with SGS, NFS is more robust to sparsity as a certain amount of sub-

structures of the dynamic graph is considered for different time steps owing to the temporal random walk strategy. However, it still fails to capture the graph's asynchronous structural evolutions. Taking G in Fig. 3.1 as an example, the local structure among $\{a, b, c, f\}$ and $\{d, e, a, f\}$ starts evolving at t_1 and t_4, respectively. The former structure takes 6 time steps to completely link four vertices, which is slower than the latter one that takes just 3 steps. Nevertheless, as shown in the middle-right part of Fig. 3.1, the NFS model neglects this important temporal discrepancy, thus being unable to preserve such asynchronous structural evolutions.

In this chapter, I study the important yet overlooked problem in dynamic graph embedding, namely fully capturing the asynchronous structural evolutions of a graph. Such asynchronous nature can be interpreted as the starting time and duration of a new edge that emerges between two vertices, and is of great importance to a wide range of time-sensitive applications, e.g., online information propagation modeling and crime prediction. However, to achieve the goal, I are confronted with three major challenges. (1) Capturing the asynchronous structural evolutions in a dynamic graph. The representation of a dynamic graph should capture not only the dynamic connections among vertices but also the starting time and duration of such evolutions, so as to provide sufficient knowledge about the asynchronous characteristics of a dynamic graph. (2) Embedding spatial and temporal dynamics of edges. When vertices join, leave, and re-join the graph, the edges they formed will change accordingly, which brings spatial dynamics. Meanwhile, it takes a different amount of time for vertices to establish new links, leading to dramatic temporal dynamics. To preserve the dynamic edge formation, both the spatial and temporal dynamics should be fully encoded in the dynamic node embeddings. (3) Preserving asynchronous evolutions of local structures. Some local structures evolve early, while others evolve later. The embedding algorithm should account for the evolution starting time for every local structure along with its dynamic edges, such that the patterns within asynchronous structural evolutions can be effectively learned to facilitate predictions into the future.

In light of these challenges, I propose to represent a dynamic graph as a set of temporal edges, coupled with the respective joining time of vertices (ToV) and timespan of edges (ToE) as two crucial indicators of the asynchronous properties. A time-centrality-biased temporal random walk is then developed to sample the local structures as temporal edge sequences. The ToV of the first vertex in a temporal edge sequence corresponds to the evolution starting time of the local structure, and the total ToE indicates how long the evolution of the current local structure takes. Hence, the temporal edge sequences successfully capture the asynchronous structural evolutions for learning expressive node embeddings. Moreover, I propose a novel **T**ime-**A**ware **D**ynamic **G**raph **E**mbedding (TADGE) algorithm to embed the asynchronous structural evolutions into vertex representations. In order to thoroughly incorporate both the spatial and temporal dynamics of the edge formation, I design a time-aware Transformer model. Intuitively, a vertex forms a new edge by linking another vertex that has high affinity with it. Therefore, I build a Transformer [85] to embed vertices' dynamic connections as their self-attentive embeddings through an encoder-decoder framework, making the learned embeddings for every vertex carry the structural information and ToE from its connected neighbors. In order to preserve the asynchronous evolutions of local structures, I learn an overall representation of every edge sequence by accounting for its evolution starting time. The sequence-level representation is learned under the constraint that any pair of embeddings at different time steps should help accurately estimate the evolving time interval between the corresponding local structures. Lastly, I fuse the vertex- and sequence-level representations to generate the final embedding for each vertex, encoding the patterns of its asynchronous structural evolutions in the dynamic graph to support downstream tasks.

The contributions of this study are highlighted as follows:

- **A New Problem.** To the best of my knowledge, I are the first to study the problem of embedding the asynchronous structural evolutions of a dynamic graph, in which the evolution starting time and duration of each local structures

vary significantly.

- **A Novel Representation of Dynamic Graphs.** I propose a time-centrality-biased temporal random walk to innovatively formulate the dynamic graph as temporal edge sequences associated with ToV and ToE tags, which preserves the asynchronous structural evolutions for learning vertex embeddings.

- **A New Approach for Dynamic Graph Embedding.** I propose TADGE, which is a novel time-aware graph embedding approach that learns expressive dynamic vertex embeddings by fusing information of both the dynamic edge formation and the asynchronous local structure evolutions.

- **Extensive Experiments.** I conduct experiments on several large-scale public datasets on dynamic graphs. Experimental results demonstrate the superiority of TADGE, which outperforms the state-of-the-arts and also shows significant advances in training efficiency and scalability.

3.2 Literature Review

The main issue in dynamic graph embedding is capturing and encoding the dynamic evolving nature of vertices and edges. Modeling the dynamic graph in a proper manner is the foundation as it captures the dynamics of vertices and edges for later embedding. Existing approaches model the dynamic graph as either a snapshot graph sequence [26] [58] [59] [83] [87] [94] [98] [118] [119] [120] or neighborhood formation sequences sampled from the temporal random walk [11] [15] [24] [65] [70]. These approaches merely capture the synchronous structural evolutions and their limitations have been discussed in Section 3.1. The dynamic graph model present in this chapter captures not only the dynamic connection changes amongst vertices but also the asynchronous evolution starting time and duration for embedding.

Given the above dynamic graph models, the embedding algorithm aims to encode the captured evolution patterns as representations of vertices or edges. TNE [120] is a pioneer work that embeds the structural difference in consecutive snapshot graphs. A series of similar approaches are continuously published, such as DHPE [119], TMF [110], Timers [114], and DynGraph2Vec [27]. Instead of measuring the overall difference between snapshots, DynamicTriad [118] embeds the triad formation process amongst vertices. DyRep [83] models the structural evolutions as a latent mediation process bridging topological evolutions and dynamic activities between vertices. GraphSAGE [35] learns the structure by a graph neural network, leveraging neighborhood information to generate embeddings for the new coming vertices.

Embedding the sequential edge formation is originally proposed in [65] which embedding continuous-time dynamic graph. Following this idea, a series works has proposed to learn the sequential information together with edge formation [11] [24] [70] [15] by using the graph attention [87] [75] [24], generative adversarial networks [98] [117]. Although none of the above-mentioned works embed the asynchronous structural evolutions, they inspire us to design the TADGE for jointly embedding the pair-wise connections and the local structures.

There are a few works focusing on temporal network embedding. M^2DNE [56] embeds the temporal edge formation process and the evolving scale of the graph. It introduces a time decay function while calculating the attention value between connected vertices, which treats the temporal information as an edge attribute for embedding. EPNE [90] embeds the periodic connection changes amongst vertices by causal convolutions. Although references [73] and [49] claim themselves as the temporal network embedding, they degenerate the temporal network as neighborhood formation sequence before embedding, thus merely preserving sequential structural evolutions without temporal information. None of above mentioned works deal with the dynamic ToV and ToE at the same time. Hence, they fail to embed the asynchronous structural evolutions but I fill this research gap in this chapter.

3.3 Problem Definition

In this section, I give the definition of a dynamic graph and formulate the problem of dynamic graph embedding for preserving the asynchronous structural evolutions.

In a dynamic graph G, vertices join the graph at any time as either isolated vertices or forming edges with existing ones. Thus, I denote a vertex v_i joining G at time t as v_i^t where t is the joining time of the vertex (ToV) and $i = 1, 2, \cdots, n$. Since vertices join, leave and rejoin the graph dynamically, which triggers the structural evolution, I denote all ToVs of v_i as its ToV set T_{v_i} in which the number of appearances of v_i in G is $|T_{v_i}|$. $|\cdot|$ is an operator for counting the number of elements in a set. When v_i^t links to an existing vertex $v_j^{t'}$ whose ToV $t' < t$ and $j = 1, 2, \cdots, n$, they form a temporal edge $e_{v_i,v_j}^{t,\delta} = (v_i, v_j, t, \delta, w)$ at time t, where $\delta = t - t'$ is the timespan of the edge (ToE) indicating how long it takes for v_i^t to form an edge with $v_j^{t'}$, and w is the edge weight. If $i = j$, a self-link is formed.

Next, I give a formal definition of a dynamic graph that is composed of the dynamic appearing of vertices at different times with the edges they formed, and then formulate the problem of dynamic graph embedding for preserving asynchronous structural evolutions.

Definition 3. ***Dynamic Graph.*** *A dynamic graph $G = \{V, E, T_V\}$, where $V = \{v_1, v_2, \cdots\}$ denotes a vertex set containing $|V|$ vertices in G, and $T_V = \{T_{v_1}, T_{v_2}, \cdots\}$ is the ToV set of every vertex in V. E is the temporal edge set in which $e_{v_i,v_j}^{t,\delta} = (v_i, v_j, t, \delta, w) \in E$ is a temporal edge linking $v_i, v_j \in V$. $t \in T_{v_i}$ is the ToV of an upcoming vertex v_i and indicates the edge forming time. $t' \in T_{v_j}$ is the ToV of an existing vertex v_j and $t' < t$. $\delta = t - t'$ is the ToE. w is the edge weight.*

Definition 4. ***Dynamic Graph Embedding for Preserving Asynchronous Structural Evolution.*** *Given a dynamic graph $G = \{V, E, T_V\}$, the objective is to learn a mapping function $f : v \mapsto r_v \in \mathbb{R}^k$ for $\forall v \in V$ such that the representation r_v preserves the asynchronous structural evolution of v in terms of asynchronous evolu-*

tion starting time and duration, where k is a positive integer indicating the dimension of r_v.

Since I aim to embed the asynchronous structural evolutions in a dynamic graph, I assume $|V|$ is well-known and mainly focus on the edge updates in this study. Meanwhile, my approach is compatible with any inductive learning schemes to handle newly joined vertices.

3.4 Capturing Asynchronous Structural Evolutions in the Dynamic Graph

Directly learn embeddings from the whole graph G is difficult due to its complexity and dynamics. Thus, it is necessary to transform the original dynamic graph into a proper format that captures its dynamic structural evolutions and is easy for later embedding. However, regardless of modeling the dynamic graph as either SGS or NFS, the asynchronous structural evolutions cannot be captured fairly well. Inspired by [65] and [18], I propose a time-centrality-biased temporal random walk to sample the dynamic graph and model it as a set of temporal edge sequences that properly captures the asynchronous structural evolutions.

Specifically, I first randomly select a set of initial vertices via uniform distribution, which appear at m different times and satisfy $|V| \leq m < \sum_{v \in V} |T_v|$. Since there are $|V|$ vertices totally appearing $\sum_{v \in V} |T_v|$ times in G, each of vertices' occurrence has the same probability $p = \frac{1}{\sum_{v \in V} |T_v|}$ to be selected as the initial one. This makes the sequences sampled by the following temporal random walk cover the entire graph evenly.

Next, I perform the temporal random walk to sample the connected vertices starting from every initial vertex. The walker will only visit vertices whose ToV is greater

than the previous one, making the sampled sequences are in line with the structural evolutions of G. Usually, the upcoming vertices have higher chances to form edges with the high centrality ones [33]. As the events/relationships happen in recent times may have stronger influence to the current vertex comparing to that of happened long time ago, I heavily sampling vertices that are close in time to the current one. In the other words, the walker prefers to visiting a vertex through the edge with a smaller ToE. Therefore, given a vertex v^t, the temporal random walk samples the next vertex $v_{next}^{t'}$ from its neighborhood set Γ_{v^t} based on a transition probability by Eq. (3.1) following a time-centrality-biased distribution, where the ToV of every vertex in Γ_{v^t} is greater than t, $degree(\cdot)$ is the vertex's degree for measuring its centrality, δ_{v,v^t} is the ToE of the edge connecting two vertices v and v^t.

$$p_{rw}(v_{next}^{t'}) = \frac{degree(v_{next}^{t'})}{\sum_{v \in \Gamma_{v^t}} degree(v)}(1 - \frac{\delta_{v_{next}^{t'}, v^t}}{\sum_{v \in \Gamma_{v^t}} \delta_{v,v^t}}) \tag{3.1}$$

Leveraging the centrality of vertices and ToE to sample the dynamic graph makes the vertex distribution and its linkage evolutionary patterns in the generated corpus more consistent with the original graph, thus providing comprehensive information for later embedding.

I record the first ℓ vertices that the walker visits to construct the edge sequences s_t, $t = 1, 2, \cdots, m$. The ToV of the first vertex in s_t exactly is the evolution starting time and the total sum of ToE of all edges in s_t indicates the time the evolution of s_t lasts. Finally, the dynamic graph has been modeled as a set of edge sequence $G = \{s_1, s_2, \cdots, s_m\}$. An example is showed on the button-right in Fig. 3.1 demonstrating that the time-centrality-biased temporal random walk is capable of capturing the asynchronous structural evolutions in the dynamic graph.

In order to easily facilitate the embedding algorithm, I further attach a virtual vertex $\langle EOS \rangle$ with zero ToE to the end of every s_t. If the walker early stops before reaching the maximum sampling number, I supplement $\langle EOS \rangle$ at the end to make the sequence have the same length as others.

3.5 Embedding Asynchronous Structural Evolutions in The Dynamic Graph

In this section, I present the details of the proposed TADGE to embed the asynchronous structural evolutions. TADGE consists of three parts: **(1) Edge Formation Embedding:** a Time-aware Transformer to embed the dynamic connection changes amongst vertices while preserving the ToE; **(2) Structural Evolution Embedding:** a multi-head self-attention model to embed the asynchronous evolution starting time for every local structure in the dynamic graph; and **(3) Representation Fusion:** encoding final vertex representation by fusing the above edge formation and structural evolution embeddings. I introduce them in the following subsections and present the training strategies at the end.

3.5.1 Embedding Dynamic Edge Formation with ToE

When a vertex comes and forms a new edge, it actually is selecting an existing vertex in the graph to connect based on past connection evolution and ToE. From the vertex point of view, the edge sequences carrying vertices' connection evolutions can be regarded as the sequences of vertices linking by these edges. Since the edges have different ToE, the time interval between consecutive vertices in the sequence varies from each other, thus bringing time-varying sequential dependency to vertices in the edge sequence. Therefore, I embed the dynamic edge formation by learning (1) the time-varying sequential dependency amongst vertices, and (2) pairwise connections amongst vertices.

Inspired by the R-Transformer [95], I propose a Time-aware Transformer consisting of a t-LSTM model and a Transformer to respectively learn vertices' sequential dependency with ToE and the pairwise connections in the edge sequence s_t. Fig. 3.2 shows the overall architecture. $\hat{r}_{v_i}$ is the obtained edge formation embedding for v_i.

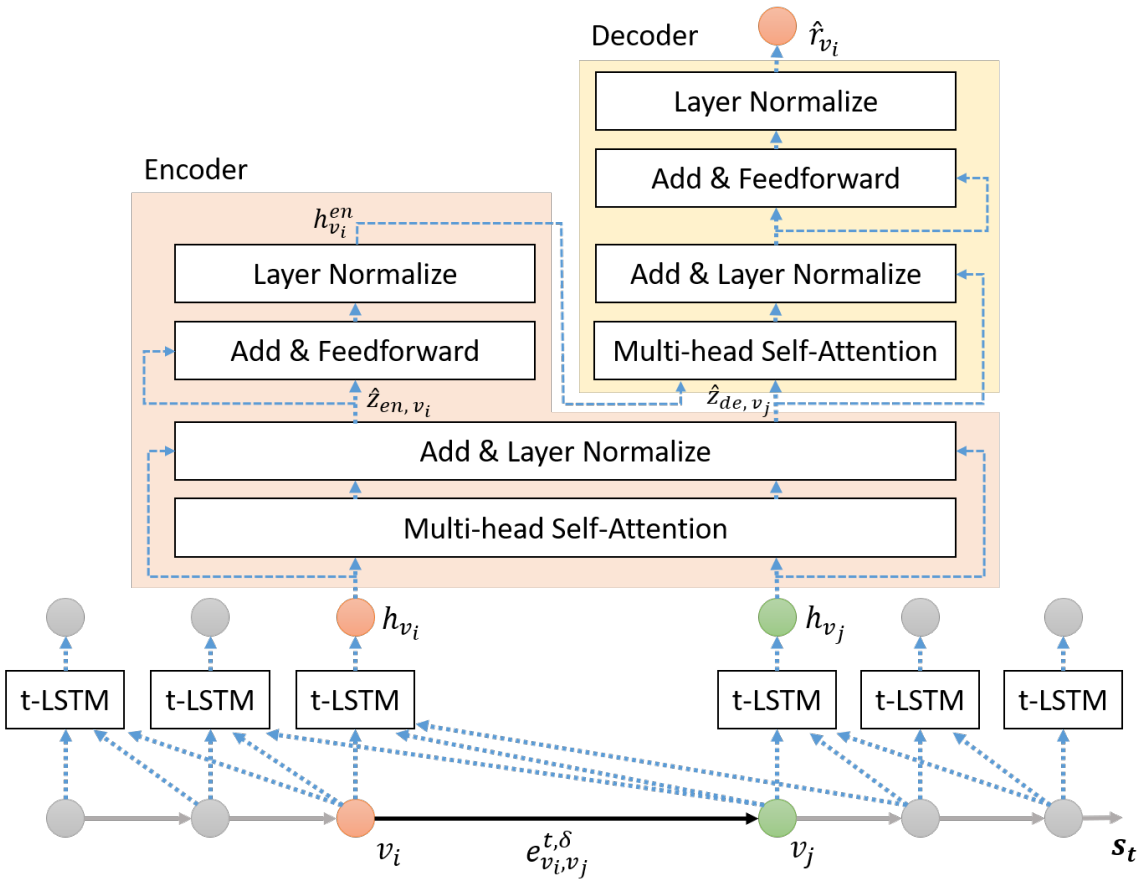

Figure 3.2: An illustration of the Time-aware Transformer for embedding v_i.

Learning Time-varying Sequential Dependency

Let us denote the target vertex that I are going to embed as v_i. To learn the sequential impact of ahead connected vertices and the ToE to v_i in s_t, I build a t-LSTM model that starts from a standard LSTM unit and connects to time-aware LSTM units [5]. It discounts the short-term effects from the long-term memory and supplements the impact of ToE between two consecutive nodes in the sequence, thus learning the time-varying sequential dependency.

Specifically, given the input information $x_{v_i} \in \mathbb{R}^k$ of v_i and its connected vertex v_j's overall memory C_{v_j} and the hidden state $h_{v_j} \in \mathbb{R}^k$, the time-aware LSTM unit first

decomposes C_{v_j} into short-term memory $C^S_{v_j}$ and long-term memory $C^L_{v_j}$ as below:

$$C^S_{v_j} = \tanh\left(W_d C_{v_j} + b_d\right) \tag{3.2}$$

$$C^L_{v_j} = C_{v_j} - C^S_{v_j} \tag{3.3}$$

where $\{W_d, b_d\}$ are the trainable weights and bias for the subspace decomposition. Next, I further decay the decayed short-term memory $C^S_{v_j}$ by a heuristic function in Eq. (3.4) such that the longer ToE the fewer effects to the short-term memory. δ_{v_i,v_j} is the ToE of the edge linking v_i and v_j while e is the Euler's number.

$$\hat{C}^S_{v_j} = \frac{C^S_{v_j}}{\log(e + \delta_{v_i,v_j})} \tag{3.4}$$

Lastly, I compose the adjusted overall memory back by

$$\tilde{C}_{v_j} = C^L_{v_j} + \hat{C}^S_{v_j} \tag{3.5}$$

The rest parts in the time-aware LSTM unit are the same as the standard LSTM as showed below:

$$f_{v_i} = \sigma(W_f x_{v_i} + U_f h_{v_j} + b_f) \tag{3.6}$$

$$g_{v_i} = \sigma(W_g x_{v_i} + U_g h_{v_j} + b_g) \tag{3.7}$$

$$o_{v_i} = \sigma(W_o x_{v_i} + U_o h_{v_j} + b_o) \tag{3.8}$$

$$\bar{C}_{v_i} = \sigma(W_c x_{v_i} + U_c h_{v_j} + b_c) \tag{3.9}$$

$$C_{v_i} = f_{v_i} * \tilde{C}_{v_j} + g_{v_i} * \bar{C}_{v_i} \tag{3.10}$$

$$h_{v_i} = o_{v_i} * \tanh(C_{v_i}) \tag{3.11}$$

C_{v_i} is the current memory v_i has. h_{v_i} is the output hidden state of the vertex v_i carrying the time-varying sequential dependency between itself and vertices in s_t. $\sigma(\cdot)$ is a sigmoid function and $*$ is the element-wise multiplication. $\{W_f, U_f, b_f\}$, $\{W_g, U_g, b_g\}$, $\{W_o, U_o, b_o\}$, and $\{W_c, U_c, b_c\}$ are the trainable weights and bias of the forget gate f, input gate g, output gate o, and candidate memory $\bar{C}$. Consequently, the t-LSTM model is capable to learn the impact of past connections and ToE on

the target vertex. Different from the STGN [115] which over-counts the short-term memory to the long-term one, the t-LSTM eliminates the impact of time-varied short-term memory from the long-term one, thereby making it more effective to learn the temporal dependency without over-weighting the time intervals.

Embedding Pairwise Connection between Vertices

On top of the t-LSTM, I embed the direct connection between v_i and v_j by the self-attention mechanism in a Transformer architecture [85]. Intuitively, the edge formation process is to select an existing vertex from the graph for the upcoming one v_i to link to. In another word, v_i is able to view the candidate vertices in the graph and then determines which one it will form the edge with. I model this process by the self-attention mechanism that contains two steps. First, I build an encoder to compute the self-attention between v_i and vertices in the local structure s_t that v_i belongs to. It measures the impact of connected vertices in the local structure for v_i to form edges. Second, regarding the encoder output as the context information of forming the edges, I further build a decoder to learn which vertices v_i exactly connects to, thus well embedding the dynamic edge formation.

Encoder. Given the t-LSTM embeddings of ℓ vertices in the edge sequence s_t, denoting as $H_v = [h_{v_1}, h_{v_2}, \cdots, h_{v_\ell}]^T \in \mathbb{R}^{\ell \times k}$, I start with the multi-head self-attention module:

$$Z_{en}^o = \text{softmax}\left(\frac{Q_{en}^o {K_{en}^o}^T}{\sqrt{k}} + M\right) V_{en}^o \tag{3.12}$$

where $Z_{en}^o \in \mathbb{R}^{\ell \times k}$ is the computed attentions from the attention-head $o = 1, 2, \cdots, \bar{o}$ while $\sqrt{k}$ is a scaling factor to smooth the softmax calculation for avoiding extremely large values of the inner product. $Q_{en}^o, K_{en}^o, V_{en}^o \in \mathbb{R}^{\ell \times k}$ respectively represent the queries, keys, and values of the self-attention obtained by the linear projection of H_v,

$$Q_{en}^o = H_v W_{en}^{Q,o}, K_{en}^o = H_v W_{en}^{K,o}, V_{en}^o = H_v W_{en}^{V,o} \tag{3.13}$$

where $W_{en}^{Q,o}, W_{en}^{K,o}, W_{en}^{V,o} \in \mathbb{R}^{k \times k}$ are trainable projection weights for the queries, keys, and values at every attention-head. Since the vertex can merely interact with

those that have already existed in the graph, I introduce a constant attention mask $M \in \{0, -\infty\}^{\ell \times \ell}$, setting the corresponding attention value to zero for the not-yet-happened interactions between vertices. Next, I concatenate all attentions obtained from every attention-head as Z_{en} and propagate low-layer queries Q_{en} to higher layers in residual connections [37] with the layer normalization [3] as showed in Eq. (3.16), which improves the expressive capability and prevents from the vanishing gradient during training.

$$Z_{en} = [Z_{en}^1, Z_{en}^2, \cdots, Z_{en}^{\bar{o}}] \in \mathbb{R}^{\ell \times \bar{o}k} \tag{3.14}$$

$$Q_{en} = [Q_{en}^1, Q_{en}^2, \cdots, Q_{en}^{\bar{o}}] \in \mathbb{R}^{\ell \times \bar{o}k} \tag{3.15}$$

$$\hat{Z}_{en} = LN(Z_{en} + Q_{en}) = S_{ln} * \frac{Z_{en} + Q_{en} - \mu}{\epsilon} + B_{ln} \tag{3.16}$$

$LN(\cdot)$ is the layer normalization function where μ and ϵ are the mean and variance of elements in the input tensor, and $\{S_{ln}, B_{ln}\}$ is the scaling weights and bias to be learned for maintaining the representation ability of the network.

Lastly, I fuse the multi-head attentions $\hat{Z}_{en}$ by a two-layer fully connected feed-forward network (FFN) with non-linear activation $\text{ReLU}(\cdot) = \max\{0, \cdot\}$ under the residual connection setting and following by the layer normalization to obtain the attention embedding H_v^{en}:

$$H_v^{en} = LN(\text{ReLU}(\hat{Z}_{en} W_f^1 + B_f^1 + \hat{Z}_{en}) W_f^2 + B_f^2) \tag{3.17}$$

where $W_f^1 \in \mathbb{R}^{\bar{o}k \times \bar{o}k}$, $W_f^2 \in \mathbb{R}^{\bar{o}k \times k}$, $B_f^1 \in \mathbb{R}^{\ell \times \bar{o}k}$ and $B_f^2 \in \mathbb{R}^{\ell \times k}$ are the trainable weights and bias of the FFN.

Decoder. The goal of the decoder is to learn which vertices the encoding one connects to so that an edge forms. Thus, the input of decoder is the t-LSTM embeddings of the second vertex to the last one in s_t, denoting as $H_v' = [h_{v_2}, h_{v_3}, \cdots, h_{v_\ell}, h_{EOS}]^T \in \mathbb{R}^{\ell \times k}$, and their attentions $\hat{Z}_{en}'$ are obtained by re-using the self-attention module in the encoder through Eq. (3.12) to (3.16).

To further learn the pair-wise connection in s_t, I build another self-attention module that employs $\hat{Z}'_{en}$ as the queries Q^o_{de}, and the self-attentive embeddings of source vertices H^{en}_v as the keys K^o_{de} and the values V^o_{de} as showed in Eq. (3.18),

$$Q^o_{de} = \hat{Z}'_{en} W^{Q,o}_{de}, K^o_{de} = H^{en}_v W^{K,o}_{de}, V^o_{de} = H^{en}_v W^{V,o}_{de} \tag{3.18}$$

where $W^{Q,o}_{de} \in \mathbb{R}^{\bar{o}k \times k}$ and $W^{K,o}_{de}, W^{V,o}_{de} \in \mathbb{R}^{k \times k}$ are trainable projection weight matrices at every attention-head $o = 1, 2, \cdots, \bar{o}$. Then, I calculate their self-attention by:

$$Z^o_{de} = \text{softmax}\left(\frac{Q^o_{de} {K^o_{de}}^T}{\sqrt{k}} + M\right) V^o_{de} \tag{3.19}$$

$$Z_{de} = [Z^1_{de}, Z^2_{de}, \cdots, Z^{\bar{o}}_{de}] \in \mathbb{R}^{\ell \times \bar{o}k} \tag{3.20}$$

$$Q_{de} = [Q^1_{de}, Q^2_{de}, \cdots, Q^{\bar{o}}_{de}] \in \mathbb{R}^{\ell \times \bar{o}k} \tag{3.21}$$

$$\hat{Z}_{de} = LN(Z_{de} + Q_{de}) \tag{3.22}$$

Lastly, a two-layer fully connected FFN with ReLU activation is built to obtain the edge formation embedding $\hat{R}_v$:

$$\hat{R}_v = LN(\text{ReLU}(\hat{Z}_{ed} W^3_f + B^3_f + \hat{Z}_{ed}) W^4_f + B^4_f) \tag{3.23}$$

where $W^3_f \in \mathbb{R}^{\bar{o}k \times \bar{o}k}$, $W^4_f \in \mathbb{R}^{\bar{o}k \times k}$, $B^3_f \in \mathbb{R}^{\ell \times \bar{o}k}$ and $B^4_f \in \mathbb{R}^{\ell \times k}$ are the trainable weights and bias of this FFN.

Since the vertices' sequential dependency and ToE have been well preserved by the t-LSTM module and the encoder-decoder merely needs to learn the pairwise connection between vertices in s_t, it is not necessary for the Time-aware Transformer model to incorporate with the position embedding which is used in the original Transformer. In order to further boost the representation power of the Time-aware Transformer model, I respectively stack N blocks of multi-head attention for encoder and decoder In addition, I adopt dropout on every residual FFN as a regularization strategy to prevent the model from over-fitting. Consequently, the Time-aware Transformer model is capable to simultaneously embed the edge connecting pairs of vertices together with their ToE.

3.5.2 Structure Embedding with Evolution Starting Time

When I treat every edge sequence s_t as a whole, the dynamic graph becomes a sequence $G = \{s_1, \cdots, s_t, \cdots, s_m\}$ representing the evolving time series of the local structures. This new sequence carries the global evolution patterns of G that consists of the asynchronous evolving of every local structure. I are going to learn a representation vector r_{s_t} for every local structure s_t where $t = 1, 2, \cdots, m$. The structure embedding r_{s_t} should preserve the sequential dependency of every local structure in G and its evolution starting time which exactly is the ToE of its first vertex.

Let's denote the edge formation embedding of a vertex v_i as $\hat{r}_{v_i}$ that corresponds to a row vector in $\hat{R}_v$. I first aggregate the $\hat{r}_{v_i}$ for every v_i in s_t to get the initial structure embedding as showed in Eq. (3.24). Since the virtual vertex $\langle EOS \rangle$ appears at the end of every s_t and does not contribute anything to the structures, I merely count the first ℓ vertices in s_t to get the initial structure embedding regardless they are $\langle EOS \rangle$ or not. For $\forall s_t \in G$, the initial structure embeddings can be written into a matrix form $H_s = [h_{s_1}, h_{s_2}, \cdots, h_{s_m}]^T \in \mathbb{R}^{m \times k}$.

$$h_{s_t} = \sum_{i=1}^{\ell} \hat{r}_{v_i}, \forall v_i \in s_t, t = 1, \cdots, m. \tag{3.24}$$

Next, I build a self-attention model to learn the sequential relationship amongst the initial structure embedding of every local structures in G as showed below:

$$Q_s^o = H_s W_s^{Q,o}, K_s^o = H_s W_s^{K,o}, V_s^o = H_s W_s^{V,o} \tag{3.25}$$

$$Z_s^o = \text{softmax}\left(\frac{Q_s^o K_s^{oT}}{\sqrt{k}}\right) V_s^o \tag{3.26}$$

$$Z_s = [Z_s^1, Z_s^2, \cdots, Z_s^{\bar{o}}] \in \mathbb{R}^{m \times \bar{o}k} \tag{3.27}$$

$$Q_s = [Q_s^1, Q_s^2, \cdots, Q_s^{\bar{o}}] \in \mathbb{R}^{m \times \bar{o}k} \tag{3.28}$$

$$\hat{Z}_s = LN(Z_s + Q_s) \tag{3.29}$$

where $Q_s^o, K_s^o, V_s^o \in \mathbb{R}^{m \times k}$ respectively represent the queries, keys, and values of the self-attention obtained by the linear projection of H_s. $W_s^{Q,o}, W_s^{K,o}, W_s^{V,o} \in \mathbb{R}^{k \times k}$

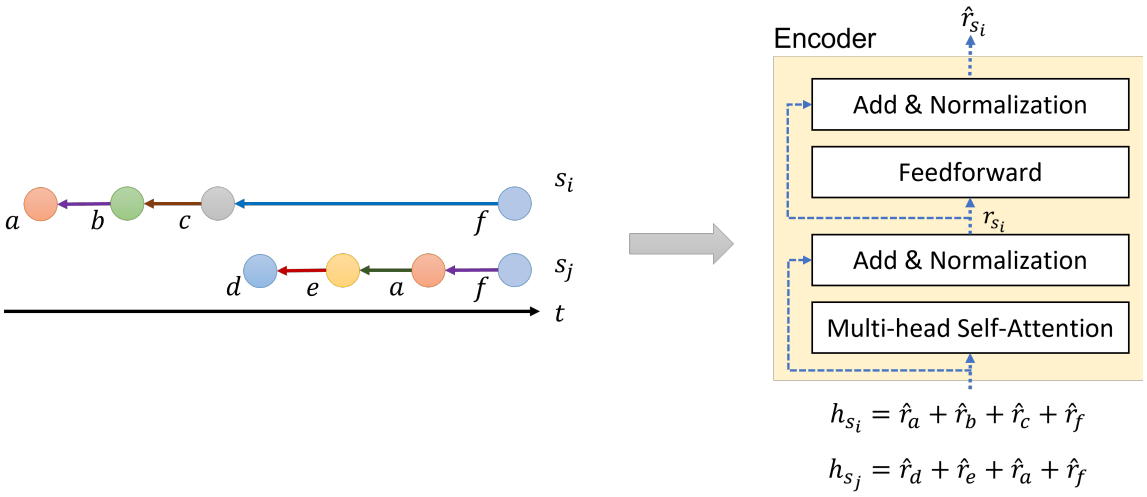

Figure 3.3: An illustration of embedding local structures s_i by the self-attention mechanism.

are trainable projection weight matrices at every attention-head $o = 1, 2, \cdots, \bar{o}$. I again build a two-layer fully connected FFN with the ReLU activation to obtain the structure embedding $\hat{R}_s$:

$$\hat{R}_s = LN(\text{ReLU}(\hat{Z}_s W_f^5 + B_f^5 + \hat{Z}_s)W_f^6 + B_f^6) \tag{3.30}$$

where $W_f^5 \in \mathbb{R}^{\bar{o}k \times \bar{o}k}$, $W_f^6 \in \mathbb{R}^{\bar{o}k \times k}$, $B_f^5 \in \mathbb{R}^{m \times \bar{o}k}$ and $B_f^6 \in \mathbb{R}^{m \times k}$ are the trainable weights and bias of the FFN. Fig. 3.3 shows the model architecture for embedding the local structures in the dynamic graph.

To this end, I obtain the structure embedding $\hat{r}_{s_t}$ for every s_t in G, which is corresponding to the tth row of the $\hat{R}_s$, preserving the sequential relationship amongst every local structure in G and itself. I do not strictly constrain that the upcoming local evolution of s_t must be s_{t+1} by training a decoder, as how I embed the pair-wise connection between vertices, because the evolution of local structures happens spontaneously and they do not have strict pair-wise relationships. Embedding the structural evolution in this loose coupling manner gives the model a better generation capacity.

In order to encode the asynchronous evolution timing of every local structure into the structure embedding $\hat{R}_s$, I train the above attention model by a regression task that estimating the evolution time interval $\delta_{s_t,s_{t'}}$ between any two local structures s_t and $s_{t'}$. The benefits are in two folds. First, it well preserves the sequential dependency amongst local structures while merely embeds their absolute evolution starting time cannot achieve. Second, it augments the scale of training samples so that the model is easy to converge and avoids under-fitting. However, when the scale of the dynamic graph becomes extremely large, it will generate a huge number of local structures to train the model, which dramatically impairs the training efficiency. I adopt a sliding window with length ℓ_s to sample the local structure sequence $G = \{s_1, s_2, \cdots, s_m\}$. The sliding step size $\tilde{o}$ satisfies $1 \leq \tilde{o} < \ell_s$ for ensuring two consecutive subsequence having overlap. Otherwise, the continuous evolution of the whole dynamic graph will be intermittent. I then learn the structure embeddings in each sliding window, thus reducing the size of training set for improving the training efficiency. To keep the article organization consistent, I present the details of the regression task for embedding the evolution starting time and discuss the criteria of improving the training efficiency in Section 3.5.4.

3.5.3 Representation Fusion

For any vertex v_i in an edge sequence s_t, I fuse its edge formation embedding $\hat{r}_{v_i}$ together with its structure embedding $\hat{r}_{s_t}$ by summing them up to obtain $\acute{r}_{v_i} = \hat{r}_{v_i} + \hat{r}_{s_t}$. Then, I input it into a FFN with the $ReLU(\cdot)$ activation to encode the final vertex representation as showed in Eq. (3.31).

$$r_{v_i} = LN(\mathrm{ReLU}(\acute{r}_{v_i}^T W_f^7 + b_{v_i}^7 + \acute{r}_{v_i}^T)W_f^8 + b_{v_i}^8) \tag{3.31}$$

where $W_f^7, W_f^8 \in \mathbb{R}^{k\times k}$ and $b_{v_i}^7, b_{v_i}^8 \in \mathbb{R}^{1\times k}$ are the trainable weights and bias of v_i of the FFN.

3.5.4 Training TADGE

In order to effectively embed the asynchronous structural evolution, I train the TADGE model by three self-supervised tasks simultaneously.

Training Task 1: Self-identification for Edge Formation Embedding

I employ a vertex classification task to train the Time-aware Transformer for learning the edge formation embedding. Intuitively, no matter how v_i's connections change, its embedding $\hat{r}_{v_i}$ should well identify v_i itself since it is representing v_i's edge formation information instead of other vertices'. I employ a softmax with cross-entropy loss to train the Time-aware Transformer well-classifying vertices' own identity as showed in Eq. (3.32).

$$\begin{aligned}\mathcal{L}_v = -\sum_{v_i \in V} \Big(& y_{v_i}^T \log \left(\text{softmax} \left(\hat{R}_v \hat{r}_{v_i} \right) \right) \\ & + \left(1 - y_{v_i}^T\right) \log \left(1 - \text{softmax} \left(\hat{R}_v \hat{r}_{v_i} \right) \right) \Big)\end{aligned} \tag{3.32}$$

$y_{v_i} \in \mathbb{R}^{|V| \times 1}$ is the self-identification of v_i in one-hot encoding that the ith element in y_{v_i} is 1 and others are 0.

Training Task 2: Evolution Time Interval Regression for Structural Evolution Embedding

Embedding the evolution starting time is the key to preserve the asynchronous structural evolutions. I employ a regression task on approximating the evolution time interval $\delta_{s_t, s_{t'}}$ between any pair of local structures s_t and $s_{t'}$ while learning their structure embeddings $\hat{r}_{s_t}$ and $\hat{r}_{s_{t'}}$ in $\hat{R}_s$. Mathematically, this regression task is written in Eq. (3.33), where $w_s \in \mathbb{R}^{k \times 1}$ is the trainable linear projection weights.

$$\mathcal{L}_s = \frac{1}{m(m-1)} \sum_{s_t \in G} \sum_{\substack{s_{t'} \in G \\ t' < t}} \left(w_s^T \left(\hat{r}_{s_t} + \hat{r}_{s_{t'}} \right) - \delta_{s_t, s_{t'}} \right)^2 \tag{3.33}$$

There are $m(m-1)/2$ pairs of local structures being used to calculate the gradient of Eq. (3.33). When m becomes extremely large, the gradient calculation will take a long time, which is an efficiency bottleneck of training the TADGE. Inspired by the negative sampling, I adopt a sliding window with length ℓ_s to sample the whole structure sequence $G = \{s_1, s_2, \cdots, s_m\}$ and construct the structure pairs within the siding window as training samples. The sliding step size $\tilde{o}$ should satisfy $1 \leq \tilde{o} < \ell_s$ for ensuring two consecutive subsequence having overlap. Otherwise, the continuous evolutions of the whole dynamic graph will be intermittent. Thus, the maximum number of sliding windows m_w satisfies $m_w \leq m - \ell_s + 1$. Since there are $\ell_s(\ell_s-1)/2$ training samples in each sliding window, $m_w\ell_s(\ell_s-1)/2$ training samples are obtained in total. According to the Lemma 1 and 2, when the sliding windows satisfy $\ell_s^2 - \ell_s < m$, the scale of training samples will definitely be reduced while ensuring the overlap of sliding windows, therefore improving the training efficiency.

Lemma 1. *$m_w \frac{\ell_s(\ell_s-1)}{2} < \frac{m(m-1)}{2}$ if and only if $m_w < \frac{m^2}{\ell_s^2}$.*

Proof 1. *Since $m > \ell_s \geq 1$, I have $\frac{\ell_s - 1}{\ell_s} < \frac{m-1}{m}$. When $m_w < \frac{m^2}{\ell_s^2}$, $m_w \frac{\ell_s(\ell_s-1)}{2} < \frac{m^2(\ell_s-1)}{2\ell_s} < \frac{m(m-1)}{2}$. When $m_w \geq \frac{m^2}{\ell_s^2}$, $m_w \frac{\ell_s(\ell_s-1)}{2} \geq \frac{m^2(\ell_s-1)}{2\ell_s} = \frac{m}{2}(m - \frac{1}{\ell_s}) \geq \frac{m(m-1)}{2}$.*

Lemma 2. *When $m > \ell_s$, $m_w \leq m - \ell_s + 1$ and $m_w \frac{\ell_s(\ell_s-1)}{2} < \frac{m(m-1)}{2}$ simultaneously establish if $\ell_s^2 - \ell_s < m$.*

Proof 2. *Since $m - \ell_s + 1$ is the upper bound of m_w and $m_w \frac{\ell_s(\ell_s-1)}{2} < \frac{m(m-1)}{2}$ established when $m_w < \frac{m^2}{\ell_s^2}$, I have $m - \ell_s + 1 < \frac{m^2}{\ell_s^2}$, such that $m - \ell_s < \frac{m^2}{\ell_s^2} - 1 = \frac{(m-\ell_s)(m+\ell_s)}{\ell_s^2}$. Therefore, $\ell_s^2 - \ell_s < m$.*

Training Task 3: Time-aware Edge Reconstruction for Final Representations

Edge reconstruction task has been widely adopted to train the static graph embedding algorithms. It assumes that the representation of connected vertices should be close.

However, this assumption oversimplified the edge formation process in the dynamic graph since the temporal information such as ToV and ToE has been neglected. A vertex can join the dynamic graph many times forming edges with the same pair of vertices but having different ToV and ToE. In the dynamic graph, when vertices are connected and have similar ToV and ToE, their representation should be close.

I propose a new task, namely time-aware edge reconstruction, to simultaneously estimate the ToE while reconstructing the edges between pairs of vertices. Given the final representation $R_v = [r_{v_1}, r_{v_2}, \cdots]^T \in \mathbb{R}^{|V| \times k}$, the time-aware edge reconstruction not only classifies whether there is an edge between any pair of vertices, but also regresses the corresponding ToE at the same time. The objective function is showed in Eq. (3.34), where $\mathcal{L}_e$ and $\mathcal{L}_{ToE}$ respectively are for the edge reconstruction and the ToE regression.

$$\mathcal{L}_r = \mathcal{L}_{edg} + \mathcal{L}_{ToE} \tag{3.34}$$

Similar to identifying vertex itself in Section 3.5.4, I again built a softmax with cross-entropy loss and employ the adjacency matrix $Y_{adj} \in \mathbb{R}^{|V| \times |V|}$ as the labels for edge reconstructing, showing in Eq. (3.35).

$$\begin{aligned} \mathcal{L}_{edg} = &-Y_{adj} \log \left(\text{softmax} \left(R_v R_v^T \right) \right) \\ &- (1 - Y_{adj}) \log \left(1 - \text{softmax} \left(R_v R_v^T \right) \right) \end{aligned} \tag{3.35}$$

$$\mathcal{L}_{ToE} = \frac{1}{2\hat{m}} \sum_{v_i \in V} \sum_{v_j \in \Gamma_{v_i}} \left(w_{ToE}^T \left(r_{v_i} + r_{v_j} \right) - \delta_{v_i, v_j} \right)^2 \tag{3.36}$$

Since ToE does not exist if there is no connection between vertex pairs, I ignore them and build the ToE regression $\mathcal{L}_{ToE}$ by Eq. (3.36), where Γ_{v_i} is the neighborhood set of v_i, δ_{v_i, v_j} in the ToE of the edge linking v_i and v_j, $w_{ToE} \in \mathbb{R}^{k \times 1}$ is the trainable linear projection weights, and $\hat{m}$ is the number of training edges.

Optimization Strategy

As TADGE is built upon the deep neural network structure, I initialize the representation R_v by using DeepWalk [71] and then apply Adam [43], a mini-batch stochastic gradient descent optimizer, to learn the model parameters by minimizing the joint loss $\mathcal{L} = \mathcal{L}_v + \mathcal{L}_s + \mathcal{L}_r$. Thanks to the above three training tasks, the asynchronous structural evolutions will be gradually embedded into the vertex representation. Meanwhile, I normalize the ToE and the time interval between structures to $[0, 1)$ by an arc-cotangent function $\delta = 2 \arctan(\delta)/\pi$ to suppress the influence of very large absolute value of δ on the model convergence.

3.6 Experiments

In this section, I validate the effectiveness of the proposed TADGE in three public real-world datasets and benchmark against the state-of-the-art baseline methods on several data mining applications.

3.6.1 Experimental Setting

Datasets

I benchmark the TADGE algorithm in three public real-world datasets, whose properties are introduced below.

- **Transaction.** This is a dynamic bitcoin transaction network[1] [48] on the Bitcoin OTC platform. A vertex is a trader who buys and sells bitcoins. Two traders form an edge when they complete a transaction. The ToE is the time

[1] `https://snap.stanford.edu/data/soc-sign-bitcoin-otc.html`

Table 3.1: Statistics of datasets

Dataset	$\|\mathbf{V}\|$	$\|\mathbf{E}\|$	Mean Degree	Mean ToE	Std. ToE	#Classes
Transaction	5,881	35,592	3.665	30.693 days	72.848	3
Hyperlink	54,075	571,927	7.701	58.350 days	121.937	2
Discussion	194,085	1,443,339	3.987	76.575 days	218.925	-

interval between buying and selling. Each trader is associated with a trustworthy label in low, middle, and high for classification.

- **Hyperlink.** This is a dynamic subreddit-to-subreddit hyperlink network[2] [47] extracted from the posts that create hyperlinks from one subreddit to another on Reddit. The vertex is a subreddit and the edge is a hyperlink connecting two subreddits. Each vertex has a binary semantic label for classification.

- **Discussion.** This is a dynamic discussion network[3] [69] extracted from a stack exchange website, namely Super User. The vertex is a user who posts, replies, comments and answers questions on the website. Once a user interact with others, an edge is formed between them.

The statistics of these datasets are presented in Table 3.1.

Baseline Methods

The baseline methods for comparison are introduced below. I first choose the popular random-walk-based dynamic graph embedding methods and graph neural network models as the common baselines. Both of them can only embed synchronous structural evolutions. Besides, I replace the Time-aware Transformer and the structure embedding in the TADGE with the graph attention network (GAT) [87] to compare the effectiveness of the TADGE against GAT while embedding the asynchronous

[2] `https://snap.stanford.edu/data/soc-RedditHyperlinks.html`

[3] `https://snap.stanford.edu/data/sx-superuser.html`

structural evolutions. Lastly, I further evaluate the vertex representation merely learned by the Time-aware Transformer for exploring how TADGE benefits from the structure embedding.

- **CTDNE** [65]. The Continuous-Time Dynamic Network Embedding consists of a temporal random walk and a skip-gram algorithm to embed the continuous-time dynamic graph. It merely embeds the synchronous sequential edge formation but neglects the evolution timing and duration.

- **GraphSAGE**[4] [35]. This is a graph neural network approach for embedding the dynamic graph by computing the graph convolution from the vertices' connection change over time. I test different aggregators including GCN, mean, mean-pooling, and LSTM and report the best results in each dataset.

- **GAT**[5] [87]. This is one of the most popular attention-based approach for graph embedding, expanding the perception range of vertices from their local neighborhoods to all vertices in the whole graph. It has been showed the effectiveness on dynamic graph embedding and regarded as one of the state-of-the-art method. In order to benchmark the TADGE against GAT, I train the model with a loss function $\mathcal{L} = \mathcal{L}_v + \mathcal{L}_r$ so that the dynamic edge formation and the ToE are preserved by the GAT for comparison.

- **GAT-strc** I supplement the structure embedding presented in Section 3.5.2 into the above GAT, thus making it preserve the asynchronous structural evolutions. I employ the graph attention instead of the self-attention to learn the structure embedding. I train the model by using the same loss function $\mathcal{L}$ as the TADGE for a fair comparison.

- **TADGE-tTran** I learn the vertex representation by the Time-aware Transformer which merely preserves the dynamic edge formation with the ToE. I

[4]https://github.com/williamleif/GraphSAGE

[5]https://github.com/PetarV-/GAT

Table 3.2: Parameter Setting of Temporal Random Walk

Dataset	m	Max. ℓ	Min. ℓ
Transaction	10,000	5	3
Hyperlink	200,000	5	3
Discussion	300,000	10	3

remove the structural attention from the TADGE while setting $\hat{r}_{s_t} = 0$ and $\mathcal{L}_s = 0$ to train the model.

I do not benchmark the TADGE against those methods regarding the SGS as inputs. The proposed graph model contains finer grain edge information than the snapshot graphs. Due to the time granularity issues of the SGS, I fail to construct the same training and test sets as TADGE for a fair comparison.

Experiment Setup

To split training and test sets, I first randomly select m initial vertices and then sample the dynamic graph into an edge sequence set $G = \{s_1, s_2, \cdots, s_m\}$ as described in Section 3.4. Due to different scales of datasets, I configure the temporal random walk with the settings in Table 3.2.

I set the minimum walk length ℓ is 3 so that there are at least 2 edges in every edge sequence indicating the structural evolutions. Besides, I found that vertices have limited multi-hop connection in the dynamic graph, which is very different from that of in a static graph. Over 98% of multi-hop connections in all three datasets does not exceed the set maximum walk length. Lastly, I randomly select 80% of edge sequence for model training and the rest is for testing. Please noted that I skip the graph sampling step in CTDNE and GraphSAGE, training and testing them by using the same input edge sequences as the TADGE for a fair comparison.

All experiments are conducted with $k = 128$ as the dimension of representation

vectors for the TADGE and all baseline methods in three datasets. Following the recommended parameter setting of the Transformer in [85], I set the number of head $\bar{o} = 8$ and stack $N = 6$ blocks for the Time-aware Transformer and GAT to learn the embeddings in the Hyperlink and Discussion datasets. Due to the small scale of the Transaction dataset, I set $\bar{o} = 4$ and $N = 3$. In training, each batch contains 200, 500, and 300 edge sequences for the datasets of Transaction, Hyperlink, and Discussion respectively. The learning rate of the Adam gradient descent optimizer is set to 0.005. For CTDNE and GraphSAGE, I adopt the optimal parameters in their original papers. Both TADGE and baseline methods are trained with enough epochs for ensuring the convergence. All experiments are conducted on a standard workstation with Intel Xeon Gold 5122 CPUs, an RTX2080TI GPU, and 32GB RAM.

Evaluation Metrics

Evaluating Classification Performance. I adopt Micro-F1 and Macro-F1 scores as evaluation metrics for the classification tasks. Mathematically, they are defined as below:

$$\text{MicroF1} = \frac{2\sum_{i=1}^{c} \text{TP}_i}{\sum_{i=1}^{c} (2\text{TP}_i + \text{FP}_i + \text{FN}_i)} \tag{3.37}$$

$$\text{MacroF1} = \frac{1}{c}\sum_{i=1}^{c} \frac{2\text{TP}_i}{2\text{TP}_i + \text{FP}_i + \text{FN}} \tag{3.38}$$

where c is the total number of classes and TP_i, FP_i, and FN_i respectively are the true positive, false positive, and false negative of the predicted results for the ith class. In multi-class classification, the Micro-F1 scores measure the overall classification accuracy regardless of the performance in classifying individual classes. The Macro-F1 scores are the mean of class-wise F1 scores in which it is sensitive to the classification performance of minority classes while the Micro-F1 scores are not.

Evaluating Regression Performance. I evaluate the regression performance by measuring the Root Mean Square Error (RMSE) between the predicted values and

the ground truths. Mathematically, they are defined as below.

$$\text{RMSE} = \sqrt{\frac{\sum_{y \in \mathcal{S}_{test}} (y - \hat{y})^2}{|\mathcal{S}_{test}|}} \tag{3.39}$$

RMSE measures the gap between the truth value y and the predicted one $\hat{y}$ in the test set $\mathcal{S}_{test}$, revealing the regression errors.

3.6.2 Experimental Results and Analysis

Extensive experiments are conducted to validate the effectiveness and efficiency of the proposed TADGE and report the results.

ToE Prediction

Given the embedding of vertex pairs, the ToE prediction is estimating the ToE of edge they formed, thus testing how effectively the learned vertex representations preserve the temporal dynamics. For the CTDNE and GraphSAGE, which do not leverage ToE regression as a training task, I make use of the embeddings obtained in their training phases to train a regression model that employs $\mathcal{L}_{ToE}$ in Eq. (3.36) as the loss function.

The test results of ToE prediction are presented in the Table 3.3. The lower the RMSE value, the more accurate the ToE prediction. The TADGE achieves the lowest RMSE and dramatically outperforms all baseline methods in three datasets, demonstrating that the temporal dynamic has been well preserved by the TADGE. Comparing to those merely embedding the dynamic edge formation, i.e., GAT and TADGE-tTran, the ToE prediction error drops a lot when they further embed the asynchronous evolutions of local structures, thus validating the benefits of preserving asynchronous structural evolutions in ToE prediction.

Static Edge Prediction

The objective of static edge prediction is to determine whether a pair of vertices will

Table 3.3: RMSE of ToE prediction

	Transaction	Hyperlink	Discussion
CTDNE	0.4397	0.4582	0.4718
GraphSAGE	0.4465	0.4464	0.4748
GAT	0.4304	0.4125	0.3394
GAT-strc	0.4025	0.4099	0.3317
TADGE-tTran	0.3959	0.4073	0.3298
TADGE	**0.3795**	**0.4004**	**0.3170**

Table 3.4: Results of Static Edge Prediction

	Transaction		Hyperlink		Discussion	
	Micro-F1	Macro-F1	Micro-F1	Macro-F1	Micro-F1	Macro-F1
CTDNE	0.5177	0.4103	0.7253	0.4527	0.6897	0.3839
GraphSAGE	0.6132	0.5157	0.7114	0.3732	0.6125	0.3788
GAT	0.8300	0.8033	0.9362	0.9071	0.7140	0.6794
GAT-strc	0.8787	0.8627	0.9389	0.9140	0.7377	0.7050
TADGE-tTran	0.9111	0.8864	0.9536	0.9328	0.8060	0.7721
TADGE	**0.9133**	**0.8939**	**0.9732**	**0.9612**	**0.8799**	**0.8567**

form an edge in future timestamps when giving their current representations. This task ignores the ToV, which is widely adopted by existing work to validate the performance of embedding algorithms in preserving the linkage structures of the graph. I employ the cosine distance to measure the similarity of vertices' representation and predict whether they form an edge by a sigmoid function. The Micro-F1 and Macro-F1 scores are adopted to evaluate the prediction performance. The Micro-F1 scores measure the overall prediction accuracy regardless of the respective performance of each vertex while the Marco-F1 scores indicate the vertex-wise average performance regardless of how often they appear in the graph.

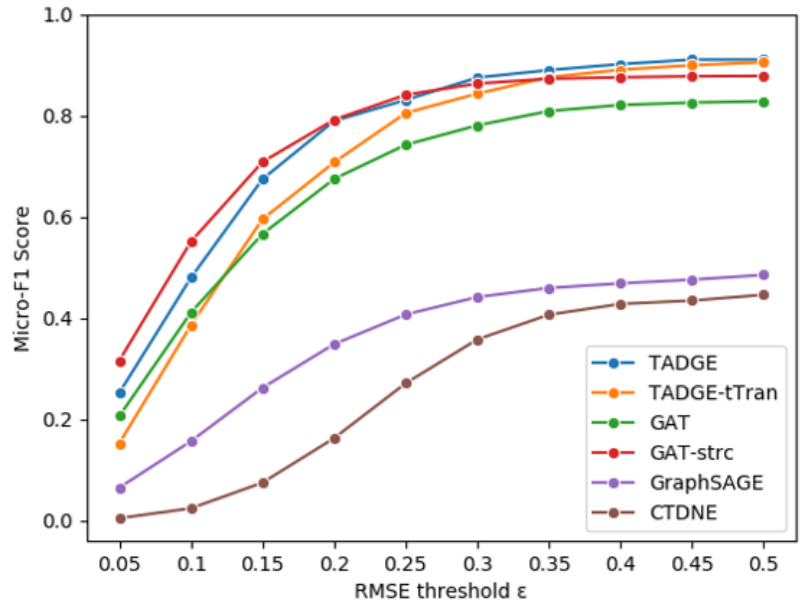

Figure 3.4: Time-aware edge prediction results with varying RMSE threshold in the Transaction dataset.

The results are summarized in Table 3.4. The TADGE achieved the highest Micro-F1 and Macro-F1 scores in all three datasets. In the Time-aware Transformer, it first learns vertices' self-attentions within the local structure by an encoder. Regarding this encoded vertex attentions as the context information, a decoder is then built to further embed the exact connections between the vertex pairs within the local structure, thus better preserving the dynamic edge formation and resulting in higher F1 scores than other baseline methods. Besides, embedding the asynchronous structural evolutions further boosts the effectiveness of preserving the dynamic edge formation. Consequently, the TADGE dramatically outperforms all baseline methods, especially in the Discussion dataset which is with the largest scale and highly dynamic connections.

Time-aware Edge Prediction

Time-aware edge prediction is a specific application for dynamic graph embedding abstracted from many data mining applications such as forecasting future crowd flow,

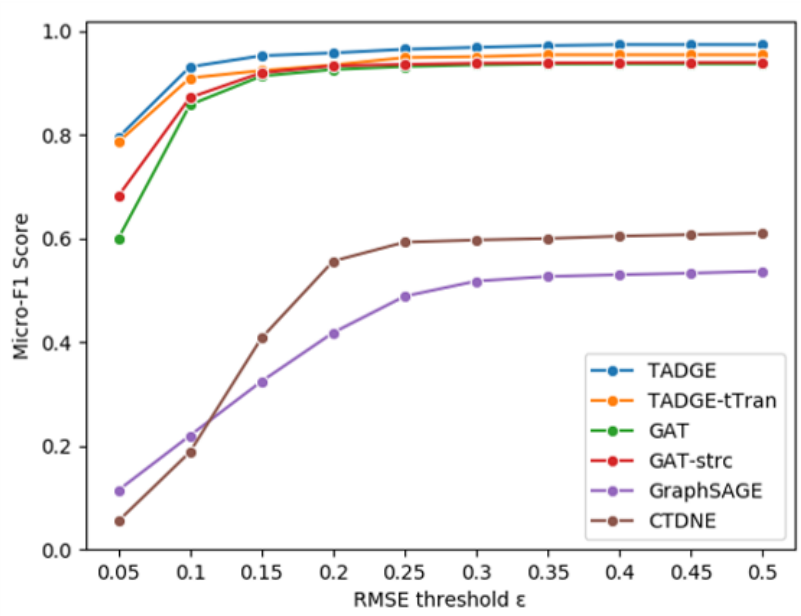

Figure 3.5: Time-aware edge prediction results with varying RMSE threshold in the Hyperlink dataset.

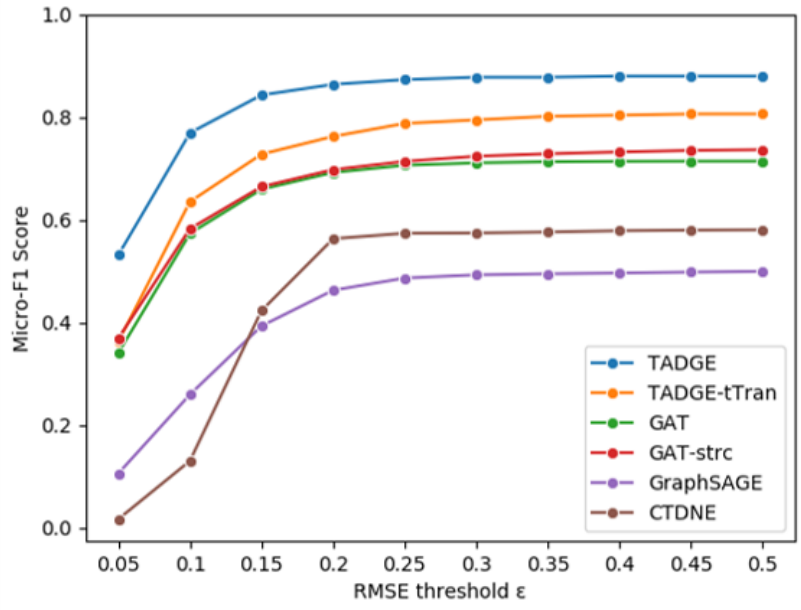

Figure 3.6: Time-aware edge prediction results with varying RMSE threshold in the Discussion dataset.

recommending items at varying time intervals, predicting fraud victims and the time of victimization. It simultaneously performs static edge prediction and ToE prediction. An edge is correctly predicted if and only if true positive results are obtained in static edge prediction and the RMSE of ToE estimation is smaller than a threshold ε. In order to expose the effect of ToE prediction error on the performance of the time-aware edge prediction, I test the threshold ε from 0.05 to 0.5 and report the Micro-F1 scores in Fig. 3.4 to Fig. 3.6.

The TADGE performs the best in the datasets of Hyperlink and Discussion when $\varepsilon \geq 0.05$. This shows that the RMSE of ToE prediction for most edges predicted by the TADGE in these datasets is less than 0.05. Although the TADGE performs slightly worse than the GAT-strc in the Transaction dataset with small ε, it becomes the best of all when $\varepsilon > 0.25$. The performance superiority of TADGE against the CTDNE and the GraphSAGE demonstrates that temporal information carried by the ToV and ToE is a key aspect of preserving the dynamics connection between vertices. Consequently, the TADGE preserves the spatial and temporal dynamics of the edge formation very well, therefore resulting in superior performance in the time-aware edge prediction.

Self-identification of Vertices

Highly accurate self-identification of vertices is very important to the dynamic graph embedding algorithm. The vertex representation will change as the graph evolves. If the updated representation fails to recognize the vertex it is representing, the edge prediction results will be totally wrong even the algorithm is with high edge prediction accuracy. For example, supposed the algorithm can correctly predict the edge linking vertices a and b when their representation r_a and r_b are given. However, if r_a is wrongly recognized as a representation of vertex c, the output of edge prediction will become vertices c and b are connected, which is not true. Therefore, the representation of vertices in the dynamic graph should well identify themselves no matter how their connections change.

Table 3.5: Results of Self-identification of Vertices

	Transaction		Hyperlink		Discussion	
	Micro-F1	Macro-F1	Micro-F1	Macro-F1	Micro-F1	Macro-F1
GAT	0.9119	0.8842	0.9033	0.8537	0.7419	0.6928
GAT-strc	0.9262	0.8989	0.9110	0.8642	0.7557	0.7118
TADGE-tTran	0.9068	0.8630	0.9321	0.8968	0.7981	0.7586
TADGE	**0.9312**	**0.9098**	**0.9453**	**0.9167**	**0.8349**	**0.7975**

To validate the effectiveness of the TADGE in self-identification of vertices, I first generate the representation for every vertex in the test set and then classify its own identity. Since the CTDNE and the GraphSAGE merely output a fixed representation for every vertex and do not update the representations in a generic manner, which is not applicable for the application of self-identification, I benchmark the TADGE against the rest baseline methods in three testing datasets. The self-identify results of baseline methods are obtained following the same procedure of the TADGE. As shown in Table 3.5, the TADGE achieves the highest Micro-F1 and Macro-F1 scores, which dramatically outperforms others. With the increasing scale and dynamics of the graph, the performance improvement of the TADGE becomes more and more significant. Therefore, the embeddings learned by the TADGE can accurately recognize the represented vertex and eventually guarantee the reliability of the TADGE in the practical use of the edge prediction.

Vertex Classification

Vertex classification aims to identify the unique labels of the vertices using their learned embeddings. A support vector machine (SVM) with a Gaussian kernel is trained by using the embeddings obtained from the training set with the corresponding vertex labels. After obtaining vertices' embeddings in the test set, I input them into the well-trained SVM and classify their labels. It tests how well the embedding algorithm is in preserving the evolutionary patterns. Since the Discussion dataset

Table 3.6: Results of Vertex Classification

	Transaction		Hyperlink	
	Micro-F1	Macro-F1	Micro-F1	Macro-F1
CTDNE	0.5969	0.3680	0.7119	0.4653
GraphSAGE	0.6037	0.3711	0.7939	0.7361
GAT	0.5952	0.4256	0.7156	0.6705
GAT-strc	0.6173	0.4424	0.8067	0.7562
TADGE-tTran	0.6042	0.4122	0.7596	0.7247
TADGE	**0.6302**	**0.4644**	**0.8140**	**0.7634**

does not contain vertex labels, I compare the classification performance measured by the Micro-F1 and Macro-F1 scores in both Transaction and Hyperlink datasets.

The results are presented in Table 3.6. At the first glance, it is clear that the TADGE achieves the highest Micro-F1 and Macro-F1 scores in both datasets. Remarkably, when neglecting the asynchronous structural evolving and merely embedding the pair-wise dynamic connection changes, the classification accuracy drops significantly. Since TADGE and GAT-strc are trained by the same loss functions, the performance improvement of TADGE comes from the excellent preservation of dynamic connection changes by the Time-aware Transformer and super expression ability of the self-attention mechanism. Comparing to CTDNE and GraphSAGE, the superior performance of TADGE demonstrates that embedding the temporal information, i.e., ToV and ToE, makes the embeddings more discriminative. In summary, the TADGE is capable to well preserve the asynchronous structural evolution patterns of vertices, thus leading to better vertex classification performance.

Parameter Sensitivity Analysis

I investigate the performance fluctuations of TADGE with varied hyper-parameters.

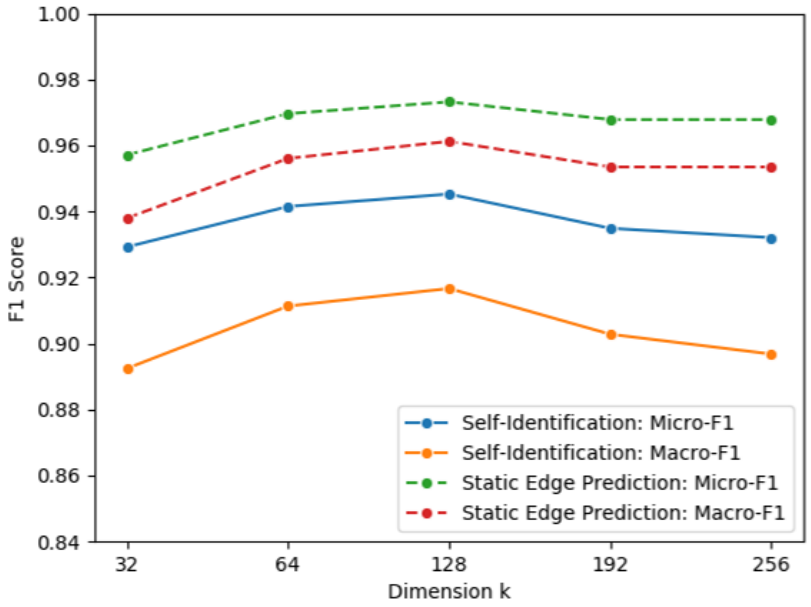

Figure 3.7: Results in the self-identification of vertex and static edge prediction with varying the hyperparameter of dimension k in the Hyperlink dataset.

In particular, I study the sensitivity of TADGE to the embedding dimension k and the number of blocks N and heads $\bar{o}$ in self-identification of vertices, static edge prediction, and ToE prediction using the Hyperlink dataset. I vary the value of one hyper-parameter while fixed the others.

Impact of k

The embedding dimension k is an important hyper-parameter affecting the expressiveness of TADGE. I exam k in $\{32, 64, 128, 192, 256\}$. As the results in Fig. 3.7 and Fig. 3.8, with the increasing k, the performance fluctuations in self-identification of vertex and static edge prediction have similar patterns. Both Micro-F1 and Macro-F1 scores increase at the beginning, indicating that the embeddings' expressiveness becomes more powerful as k increases. After reaching the best F1 scores when $k = 128$, the classification performance slightly drops afterward. In ToE prediction, TADGE gets the smallest RMSE when $k = 64$, and the prediction errors with the other k values are close. Hence, TADGE is not sensitive to k in ToE prediction.

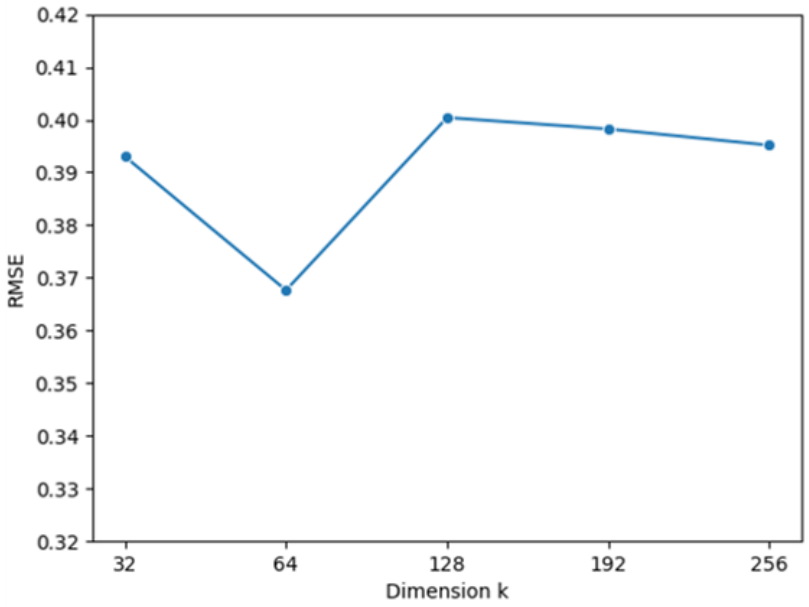

Figure 3.8: Results in the ToE prediction with varying the hyperparameter of dimension k in the Hyperlink dataset.

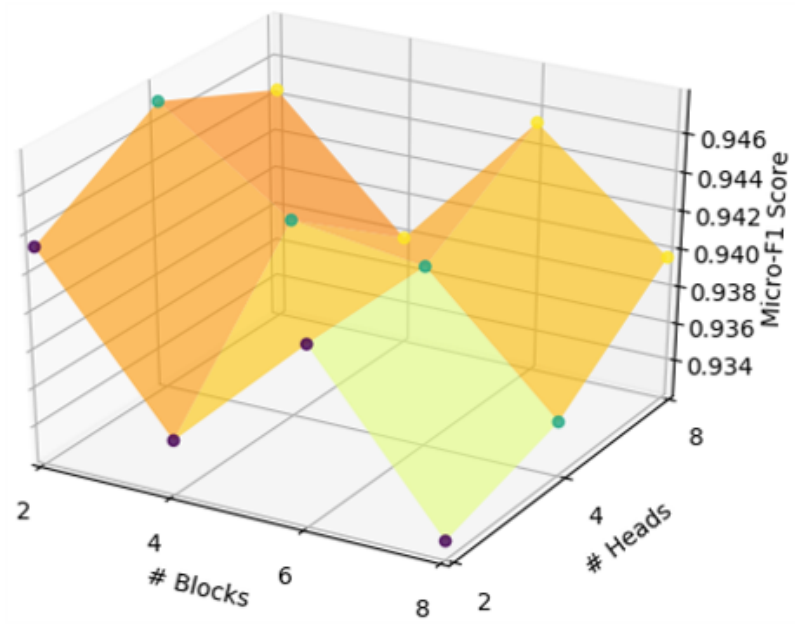

Figure 3.9: Micro-F1 scores of self-identification of vertices with varying the hyperparameter of blocks N and heads $\bar{o}$ in the Hyperlink dataset.

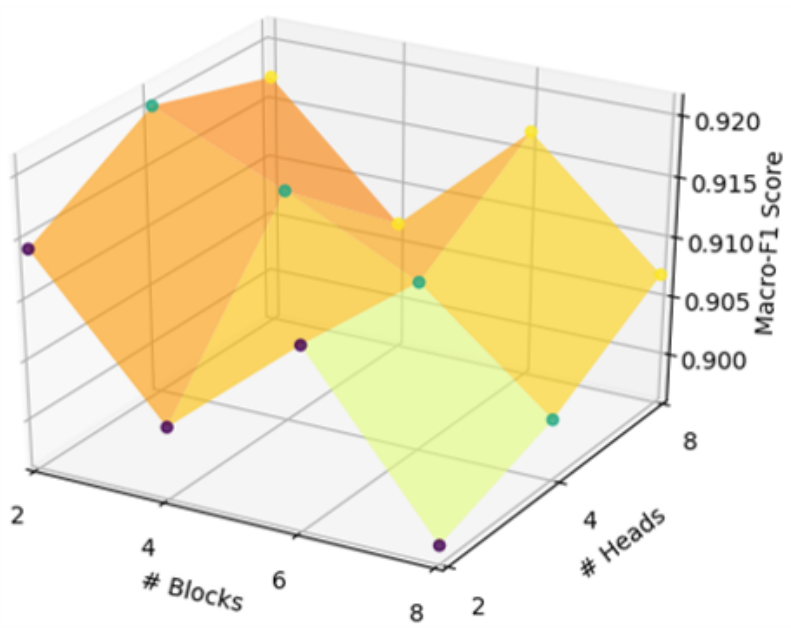

Figure 3.10: Macro-F1 scores of self-identification of vertices with varying the hyperparameter of blocks N and heads $\bar{o}$ in the Hyperlink dataset.

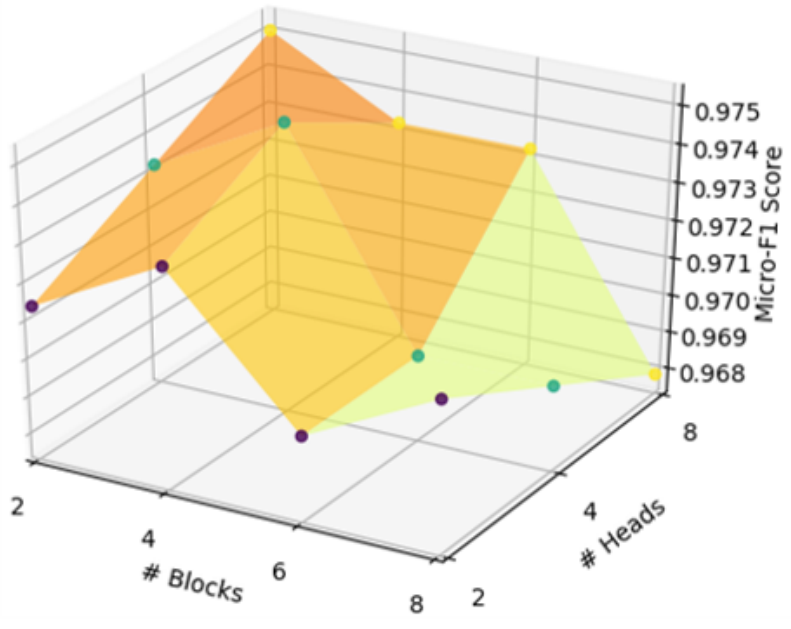

Figure 3.11: Micro-F1 scores of static edge prediction with varying the hyperparameter of blocks N and heads $\bar{o}$ in the Hyperlink dataset.

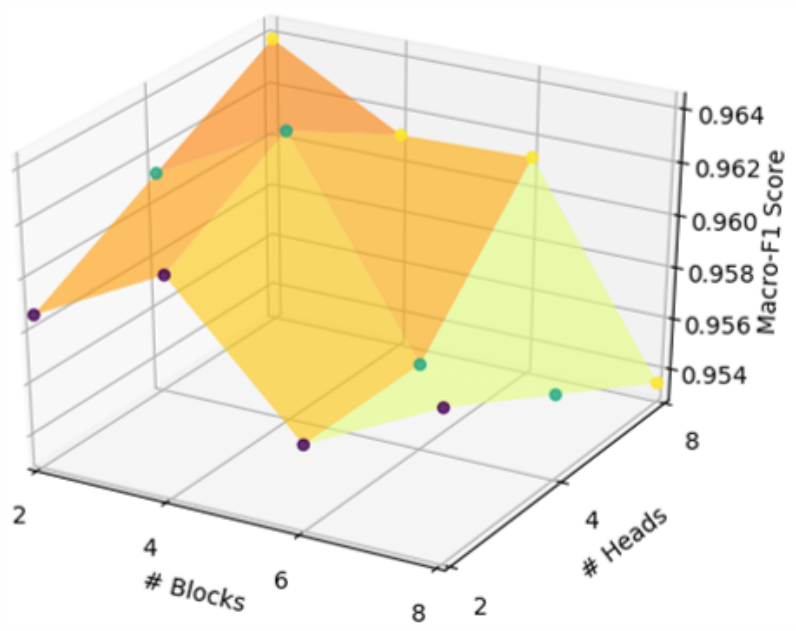

Figure 3.12: Macro-F1 scores of static edge prediction with varying the hyperparameter of blocks N and heads $\bar{o}$ in the Hyperlink dataset.

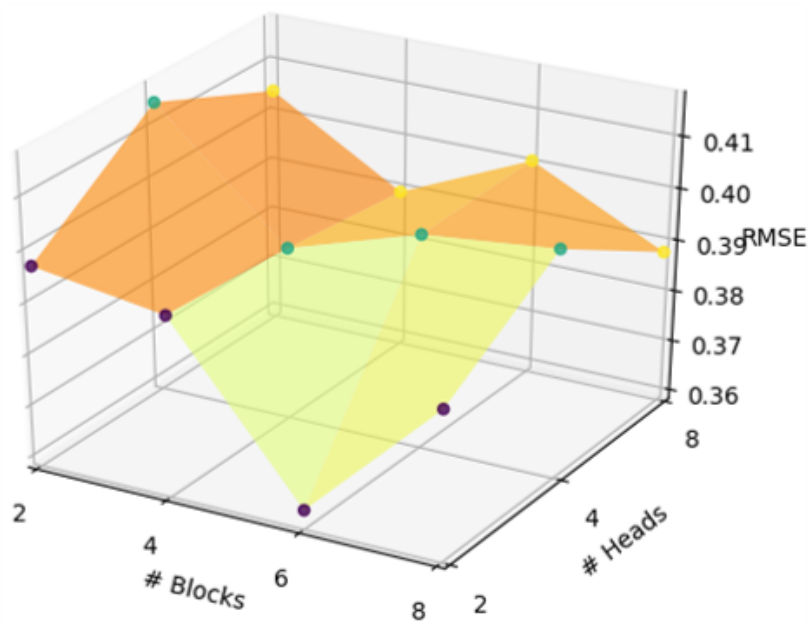

Figure 3.13: RMSE of ToE prediction with varying the hyperparameter of blocks N and heads $\bar{o}$ in the Hyperlink dataset.

Impact of the number of blocks N and heads $\bar{o}$.

The number of blocks and heads are the key parameters affecting the expression ability of the self-attention mechanism in the TADGE to learn the asynchronous structural evolutions. I test the combination of $N \in \{2, 4, 6, 8\}$ and $\bar{o} \in \{2, 4, 8\}$ to evaluate their impacts on the performance fluctuations of TADGE. As the results in Fig. 3.9 to Fig. 3.13, in self-identification of vertices, higher F1 scores are achieved when having 2 or 6 blocks. With the increasing number of blocks, the general trend of classification performance goes downward. When there are fewer blocks, TADGE with 4 heads achieves the best self-identification results. However, more heads are required for getting the higher F1 scores when over 6 blocks are employed. In the static edge prediction, the increasing number of blocks has a negative impact on both Micro-F1 and Macro-F1 scores while, in general, more heads the better performance. Similar trends are observed in ToE prediction. In summary, TADGE prefers more heads but fewer blocks so as to better embed the asynchronous structural evolutions in the dynamic graph.

Convergence and Training Efficiency Analysis

I demonstrate the convergence and training efficiency of TADGE in the Hyperlink datasets. In Fig. 3.14, I observe that training loss drops quickly and converges within a hundred epochs. The overall training accuracy in both edge reconstruction and self-identification of vertices quickly converges to over 0.9 Micro-f1 score (refer to Fig. 3.15). Although the loss changes slightly, the Macro-f1 scores in both tasks gradually increase and eventually reach the convergence, which is consistent with the conclusion drawn in [85]. Because the appearing times of vertices in a dynamic graph follow the long-tailed distribution, there are a number of vertices having very few connections to others. They fail to provide enough linkage information for the embedding algorithms to learn. Consequently, in the self-identification tasks, the TADGE converges to very high Micro-F1 scores but with relatively low Macro-F1 scores. Better dealing with this minority of vertices will be a direction for further

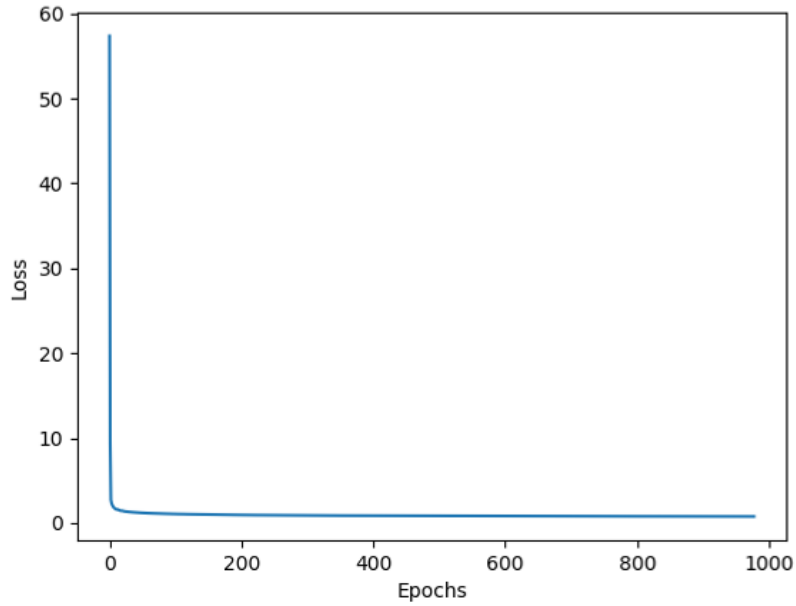

Figure 3.14: Training loss of TADGE in the Hyperlink dataset.

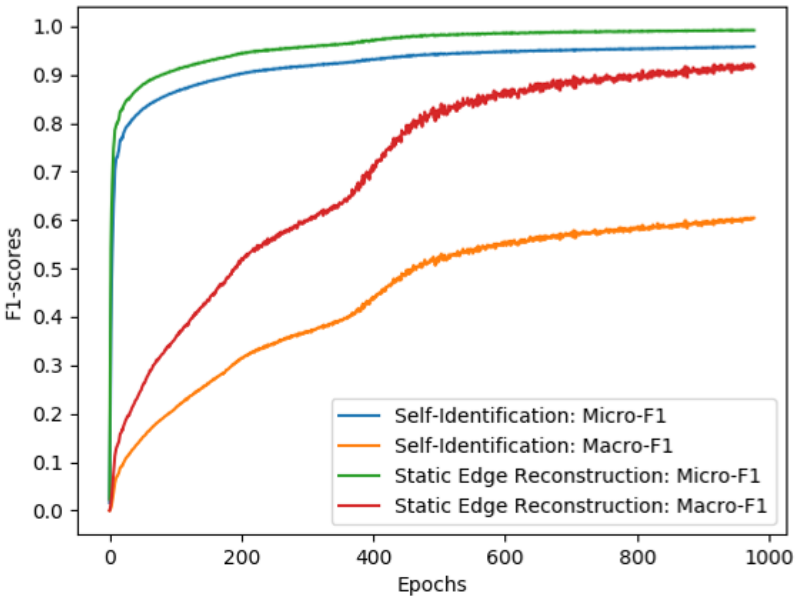

Figure 3.15: Training F1 scores of TADGE in the Hyperlink dataset.

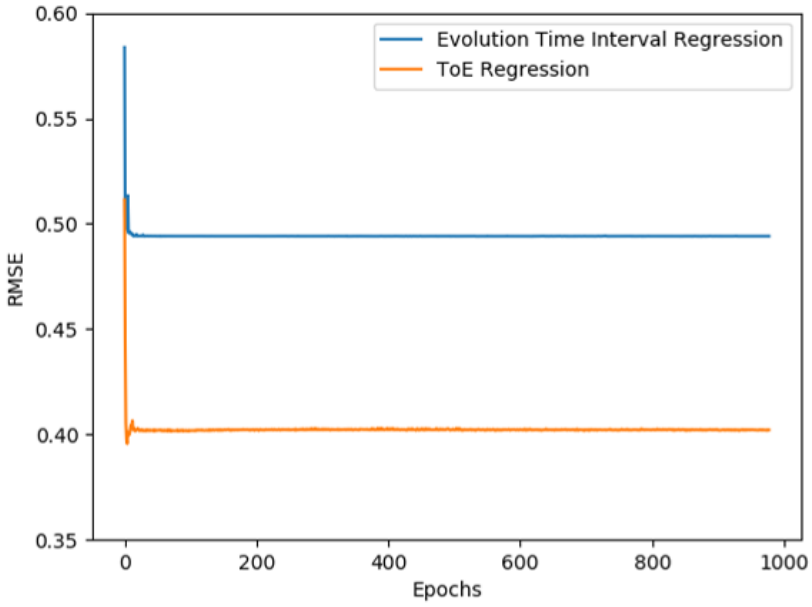

Figure 3.16: Training RMSE of TADGE in the Hyperlink dataset.

extending this study. In both ToE prediction and structural evolution time interval regression, which are two regression tasks, the training RMSE converges within 20 epochs (refer to Fig. 3.16), indicating the advantageous convergence speed of the TADGE in the regression tasks.

Fig. 3.17 shows the average running time of every epoch while training the TADGE with varying dimension k. The shadow shows the variance of training time. As k increases, the running time of updating the weights of TADGE at each epoch fluctuates slightly. This validates that the training time is not sensitive to the dimension of the representations. Hence, I conclude that TADGE has very good convergence and training efficiency.

Scalability Analysis

I conduct a scalability test for the TADGE in the Discussion dataset which contains $194,085$ vertices and $1,443,339$ edges. I vary the proportion of the training data in $\{0.2, 0.4, 0.6, 0.8, 1.0\}$ to train the TADGE and report the average training time of

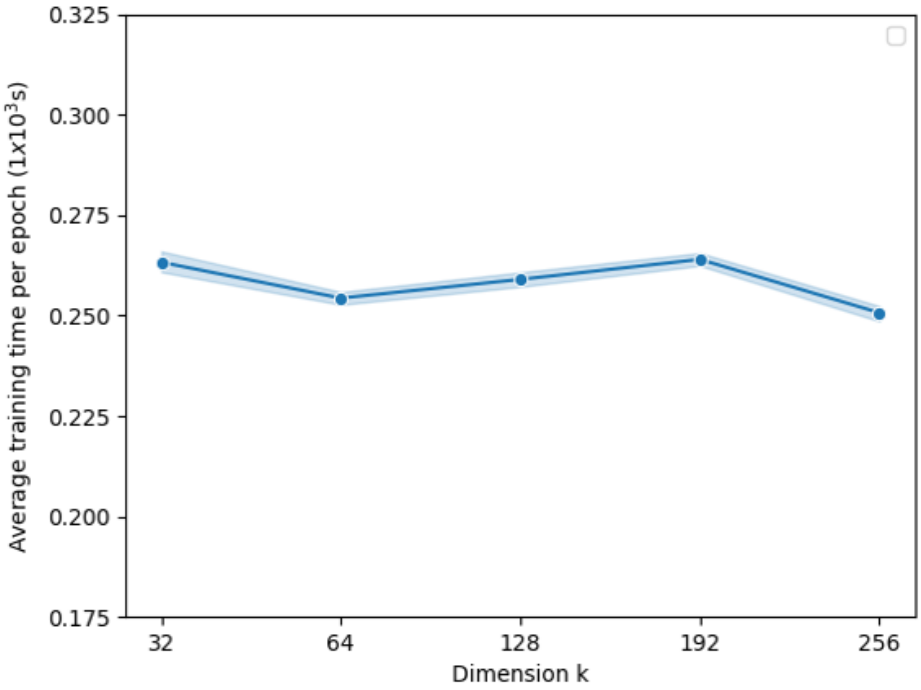

Figure 3.17: Training efficiency of TADGE with varying k in the Hyperlink dataset.

every epoch. Ideally, the training time should increase linearly when I enlarge the scale of training data. The growth of training time is showed in Fig. 3.18. The line indicates the average running time of updating the weights of TADGE at each epoch and the shadow shows the variance. As the scale of training data gradually enlarges, the training time grows linearly from 0.141×10^3 seconds to 0.686×10^3 seconds. Consequently, I conclude that the TADGE has very good scalability for embedding very large-scale dynamic graphs.

3.7 Chapter Summary

In this chapter, I generically formulate a dynamic graph as a set of temporal edges, appending the respective joining time of vertices (ToV) and timespan of edges (ToE), which successfully captures the asynchronous dynamics in the graph. A time-centrality-

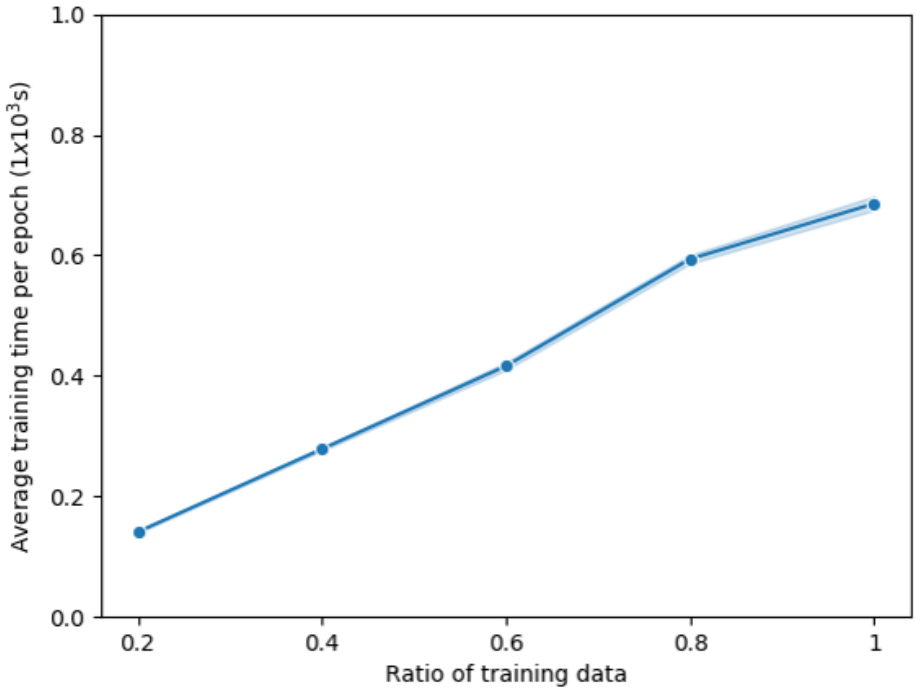

Figure 3.18: Training time of TADGE per epoch with varying data proportions in the Discussion dataset.

biased temporal random walk is proposed to sample the dynamic graph as a set of temporal edge sequences for capturing the asynchronous structural evolutions. A TADGE model containing a Time-aware Transformer and a structural embedding model is then proposed to simultaneously embed the dynamic connection changes with ToE and the asynchronous evolution starting time of every local structure. The experimental results show that the TADGE achieves significant performance improvement over the state-of-the-art approaches in various data mining tasks, thus validating the effectiveness of TADGE to embed the asynchronous structural evolutions. Besides, TADGE is very efficient and scalable when handling large-scale dynamic graphs. This study has been submitted to the IEEE Transactions on Knowledge and Data Engineering (TKDE) in 2021.

Chapter 4

Early Prediction of At-Risk Students with Multi-timescale Dynamic Learning Behaviors

In this chapter, I study the feature representation to capture and embed the dynamic learning behavior for early predicting academic at-risk students, tackling the challenge of multi-timescales. Early prediction of students at risk (STAR) is an effective and significant means to provide timely intervention for dropout and suicide. Existing works mostly rely on either online or offline learning behaviors which are not comprehensive enough to capture the whole learning process. In addition, they merely focus on static or period learning behavior and overlook the multi-timescale dynamics, thereby leading to unsatisfying prediction performance. A novel algorithm (EPARS) is proposed to early predict STAR in a semester by modeling online and offline dynamic learning behaviors. The online behaviors come from the log of activities when students use the online learning management system. The offline behaviors derive from the check-in records of the library. The main observations are two folds. Significantly different from good students, STAR barely have regular and clear study

routines. A multi-timescale bag-of-regularity method is devised to extract the multi-period regularity patterns of students' learning behaviors. Second, friends of STAR are more likely to be at risk. A co-occurrence network is constructed to approximate the underlying social network and encode the social homophily as features through network embedding. With the fused features of learning regularity and social homophily, the EPARS can achieve impressively high accuracy in very early predicting at-risk students.

4.1 Introduction

Predicting students at risk (STAR) plays a crucial and significant role in education as STAR keep raising public concern of dropout and suicide among adolescents [67] [80]. STAR refer to students requiring temporary or ongoing intervention to succeed academically [72]. Students may be at risk for several reasons like family problems and personal issues including poor academic performance. Those students will gradually fail to sustain their studies and then drop out which is also a waste of educational resources [10]. Early prediction of STAR offer educators the opportunity to intervene in a timely manner.

Traditionally, many universities identify STAR by their academic performance which sometimes is too late to intervene. Existing works are largely based on either online behaviors or offline behaviors of students [36] [46] [60]. For example, STAR are predicted in a particular course from in-class feedback such as the grade of homework, quiz, and mid-term examination [60]. In this case, only static learning behavior has been captured for prediction. HoIver, due to the complex nature of STAR [23], either online and offline behaviors only capture part of the learning processes. For example, some students prefer learning with printed documents so they become inactive in online learning platforms after downloading learning materials. This process is difficult to capture through their online learning behaviors. Therefore, existing work

can hardly capture the whole learning processes in a comprehensive way and thus leads to poor performance in the early prediction of STAR.

In this work, I aim to predict STAR before the end of a semester using both online and offline dynamic learning behaviors. STAR are defined as students with an average GPA below 2.0 in a semester. Online behaviors are extracted from click-stream traces on a learning management system (LMS). These traces reveal how students use various functionalities of LMS. While the offline behaviors derive from library check-in records. To achieve the goal, I encounter the following three major challenges: (1) Multi-timescale dynamics of learning behavior. During the semester, the patterns of students' learning behavior varies over daily, weekly, and monthly time scales. Therefore, it is challenging to capture them fairly Ill for classifying STAR. (2) Label imbalance. The number of STAR is significantly smaller than that of normal students, which makes it an extreme label-imbalance classification problem. The classifier will be easily dominated by the majority class (normal students). (3) Data insufficiency. Students, especially STAR, are usually inactive at the early stage of a semester. As a result, the behavior traces are far from enough for accurate early prediction of STAR.

In light of these challenges, I propose a novel algorithm (EPARS) for early prediction of at-risk students. EPARS captures students' regularity patterns of learning processes in a robust manner. Besides, it also models social homophily among students to perform highly accurate early STAR prediction. The intuitions behind EPARS are two-fold. First, good students usually follow their study routines periodically and show clear regularities of learning patterns [107]. HoIver, the study routines of STAR are disorganized leading to irregular learning patterns, which is different from good students. Second, students tend to have social tie with others who are similar to them according to the theory of social homophily [61] and existing studies found that at-risk students had more dropout friends [23].

Based on both intuitions, I first propose a multi-timescale bag-of-regularity method to extract discriminative features from the regularity patterns of students' learning

behaviors. Unlike the traditional approaches using entropy for measuring the regularities, which cannot work Ill on sparse data, I ignore the inactive behavior subsequence and capture the regularity patterns in a multi-timescale manner. The proposed approach can capture the regularity patterns fairly well even though the data are very sparse. Therefore, it overcomes the challenge of multi-timescale dynamics and extracts discriminative features from the regularity patterns for classifying STAR.

In order to model the social homophily, I construct a co-occurrence network from the library check-in records to approximate social relationships among students. Co-occurrence networks have been widely used in modeling social relationship and achieved great success in many application scenarios [77] [78] After that, I embed the co-occurrence networks and learn a representation vector for every student with the assumption that students' representation vectors are close when they have similar social connections. Modeling the social homophily provides extra information to supplement the lack of behavior trace for STAR at the beginning of a semester, which solves the data insufficiency problems and makes EPARS capable of early predicting STAR. Moreover, I oversample the training samples of STAR by random interpolating using SMOTE [16], which overcomes the label imbalance problem while training the classifiers.

The contributions are summarized as follows.

- A multi-timescale bag-of-regularity approach is proposed to extract regularity patterns of learning behaviors, which overcomes the challenges of multi-timescale dynamics and is robust for sparse data. This approach is also generic for extracting repeated patterns from any given sequence.
- The social homophily among students are learned by embedding a co-occurrence network constructed from their library check-in records, which reliefs the data insufficiency issues.
- Extensive experiments on a university-scale dataset show that the proposed

EPARS is effective on STAR early prediction in terms of 14.62% $\sim$ 38.22% accuracy improvement to the baselines.

4.2 Literature Review

There are various reasons for students being at-risk, including school factors, community factors, and family factors. Most of the existing works focus on school factors due to the convenience of data collection. The classification models used include Logistic Regression, Decision Trees, and Support Vector Machines. The main difference of these works relies on the input features, which could be generally classified into offline and online.

The offline learning behaviors contain check-ins of classes or libraries, quiz and homework grades, and records of other activities conduct in the offline environment. These kinds of works are quite straight forward to monitor the student learning activities for identification. Early researchers design the Personal Response system and utilize the order of students' device registration to help identify STAR [31]. Besides, questionnaires and personal interviews are also applied to collect student information for identification [19]. These methods show accurate results in an early stage of a semester. Moreover, Marbouti et al. also proposed to identify STAR at three time-points (week 2, 4, and 9) in a semester using in-term performance consists of homework and quiz grades and mid-term exam scores [60]. These methods rely heavily on domain knowledge, and collecting these offline learning data is very high labor cost and time-consuming, such that they are not practical for large scale STAR prediction.

With the popularization of online learning, researchers have turned their attention to analyzing student behavioral data on online learning platforms such as MOOCs and OpenEdX. The online learning behaviors are collected from the trace that students

left in the online learning system such as click-stream logs in functional modules of the systems, forum posts, assignment submission, etc. Kondo et al. early detect STAR from the system login and assignment submission logs on the LMS [44], but their results may be partial since most students are not actively engaged with LMS. Shelton et al. designed a multi-tasks model to predict outstanding students and STAR [76], which purely uses the frequency of module access as features. [39] proposed a personalized model for predicting STAR enrolling in different courses, but it is hardly generalized to various courses, especially the totally new one. Instead of purely using statistic features, I further extract students' regularity patterns and social homophily for early predicting STAR.

4.3 Problem Formulation

This section gives the formal problem definition of STAR early prediction which is essentially a binary classification problem. I will introduce the exact definition of STAR, the input data, and the meaning of early prediction.

According to the student handbook of the university, when a student has a Grade Point Average (GPA) lower than 2.0, he/she will be put on academic probation in the following semester. If a student is able to pull his/her GPA up to 2.0 or above at the end of the semester, the status of academic probation will be lifted. Otherwise, he/she will be dropped out. Therefore, STAR are defined as students whose average GPA is below 2.0 in a semester.

The input data are two folds. One is the records of students' online activities in the Blackboard, a learning management system. The Blackboard has several modules including course participation, communication and collaboration, assessment and assignments. Students could browse and download course-related materials including lecture keynotes, assignments, quizzes, lab documents etc. They can also take online

quizzes and upload their answers for assessment. Besides, students could communicate over the different posts and collaborate on their group assignments. Students' click operations in the Blackboard will be recorded (online traces). The other is the check-in records of the library. Students have to tap their student cards before entering the library (offline records).

Early prediction means the input data are collected before the end of a semester. Given online traces and offline records accumulated within t ($t < t_{end}$) where t_{end} is the end time of a semester, the objective is to identify STAR as accurate as possible.

4.4 Data Description

Students' online and offline learning traces and their average GPA are collected in an Asian University in 2016 to 2017 academic year. The online learning traces come from how students use the Blackboard, a learning management system, to learn. There are many functions in the Blackboard but some of them are rare to be used by students. Thus, I collect the click-stream data with timestamps from some of the most popular modules in the Blackboard including log-in, log-out, course materials access, assignment, grade center, discussion board, announcement board, group activity, personal information pages, etc. Offline learning traces come from students' library check-in records which indicating when they go to library. Since students do not need to tap their student cards when they leave the library, the check-out records will not be marked down and I exclude it in this study.

All $15,503$ undergraduate students in the whole university involved in this study. Every student has a unique but encrypted ID for linking their LMS click-stream data, library check-in records, and GPA. The overview of collected data are showed in Tab. 4.1. There are 225 and 319 STAR in semester one and two respectively, which are 1.45% and 2.06% of all students. This makes the STAR early prediction as an

Table 4.1: Data overview

	Semester 1		Semester 2	
	STAR	Other Std	STAR	Other Std
Population	391	15,112	225	15,278
# click-stream logs in LMS	2,225,605	95,949,014	1,019,134	70,874,428
Avg. # click-stream logs	5,692.0844	6,349.1936	4,529.4844	4,638.9860
Avg. # click-stream logs in first 2 weeks	301.4041	399.9502	243.0400	284.4368
Avg. # click-stream logs in last 2 weeks	526.6522	545.4346	336.9133	304.7331
# library check-in	14,045	636,353	6,245	517,557
Avg. # library check-in	35.9207	42.1091	27.7556	33.8760
Avg. # library check-in in first 2 weeks	1.7877	2.3303	1.3889	1.8424
Avg. # library check-in in last 2 weeks	2.9834	3.3760	2.3444	2.4547

extremely label imbalance classification problem. In addition, students left over 170 million click-stream logs but only 1.7 million library check-in records in the whole academic year such that the data density between online and offline learning trace are also imbalance. Compared to the last two weeks of the semester, all students are less active in the first two weeks and STAR are even less active than normal students which cause data inefficiency problems for early predict STAR at the beginning of the semester.

4.5 Methodologies

In this section, I will elaborate on the proposed EPARS including multi-timescale bag-of-regularity, social homophily, and data augmentation.

4.5.1 multi-timescale Bag-of-Regularity

In order to extract the regularity patterns from students' learning traces, I propose multi-timescale bag-of-regularity here, which is robust for sparse data.

Based on Hugh Drummond's definition, behavior regularity is repeatedly occurring of a certain behavior in descriptions of patterns [22]. Students usually have their own repeated patterns for using LMS and going to the library. For instance, some students prefer to go to the library every Monday and Thursday. It is possible for us to illustrate their repeated patterns on multiple timescales such as they will not go to library after the day they go there; they go to the library two and three days apart alternately. If I purely extract the regularity patterns on a single timescale, it hardly captures the complete picture and leads to information loss. This motivates us to extract the regularity patterns in multi-timescales. In addition, traditional approaches, such as entropy, measure the regularities in a global perspective. When students' library check-in data are sparse, those approaches will regard their library check-in as outliers and consider their general regularity patterns as never go to the library, which are incorrect. Therefore, I focus on the multi-timescale behavior traces students leave during learning for extracting their learning regularity patterns.

First of all, I construct a binary sequence from students' behavior traces. When they have certain behaviors, such as check-in to the library, I mark it as 1 in the sequence. The time granularity for constructing the binary sequence depends on the application and the time granularity I used in this study is a day. Next, I sample subsequences of length ℓ centered on every nonzero element in the sequence. The length of subsequences $\ell = 2+(s-1)\times z$ where $s \in \{1, 2, \cdots, S\}$ is the timescale and z is the step-size between scales. This sampling approach guarantees that no all-zeros sequence will be sampled for the following regularity measurement which gives the proposed method the ability to overcome data sparsity issues. Every subsequence actually is a behavior pattern that is viewed on different timescales.

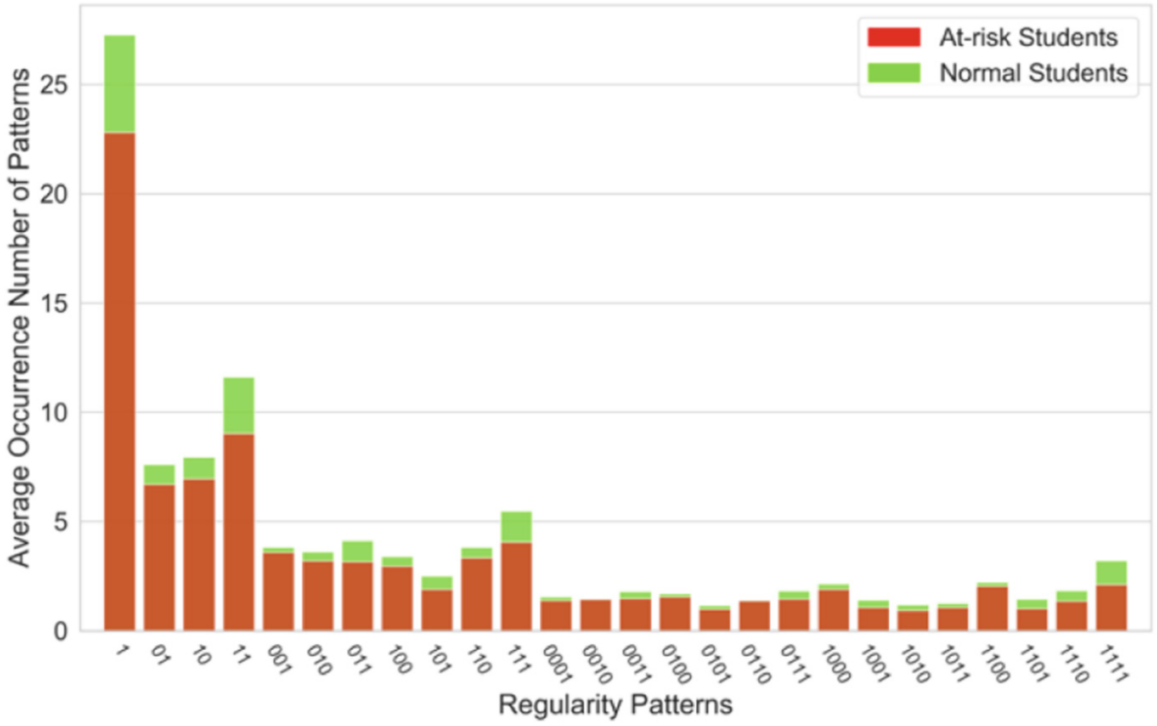

Figure 4.1: Regularity patterns of at-risk students and normal students.

After sampling the multi-timescale dynamic behavior patterns, I explore the repeated patterns from them to obtain the regularities. Since the regularity is repeatedly occurring of behavior patterns, I ignore the subsequences that the times of occurrences are less than a threshold n. For the subsequence of length ℓ in timescale s, it contains $2^\ell - 1$ different behavior pattern excluding all-zeros one. I regard them as a bag and count the number of occurrences of every behavior pattern. Finally, a $(2^\ell - 1) \times 1$ vector r_s is obtained, which carries the behavior regularities on timescale s.

Figure 4.1 shows the average occurrence number of each library check-in pattern between STAR and normal students. The horizontal axis represents the library check-in patterns at scale 1 to 4. For example, pattern 110 represents a three-day pattern of students' library check-in behavior in which they continuously go to the library for first 2 days but not go there on the third day. The patterns at scale 1 exactly

is the total number of library check-ins. This figure indicates that STAR have less continuous library studies than normal students.

Lastly, I concatenate the regularity vectors r_s in every timescale as the representation of regularity on multi-timescales. The proposed bag-of-regularity approach explores the regularity patterns of behaviors in multi-timescales such that it can extract richer information from the sparse input sequence. The regularity features extracted from dense LMS data and sparse library check-in records by the multi-timescale bag-of-regularity are on the same timescale-space so that I can simply concatenate them together as the final regularity features for STAR prediction and the performance is fairly well. In addition, the proposed multi-timescale bag-of-regularity is generic for extracting repeated patterns from any given sequence since it will transform the input sequence into a binary sequence before extracting regularities.

4.5.2 Social Homophily

A co-occurrence network is constructed to model the social relationship among students. If students are friends, they are more likely to learn together because of the social homophily [61]. They have a higher probability to go to the library together comparing to strangers. Thus, I assume that two students are friends if they go to library together. If the time difference of the library check-in between two students is less than a threshold δ, I treat this as the co-occurrence of two students in the library. In other words, they go to the library together. Based on this, I construct a co-occurrence network $G(V, E, W)$ where nodes V are students and there is an edge $e \in E$ linking two nodes if students go to the library together. Each edge is accompanied by a weight value $w \in W$ showing how many times they co-occurrence in the library. I constrain $w \geq \sigma$ which is a threshold to filter out the "familiar strangers".

Figure 4.2 illustrates the constructed co-occurrence network partially. Each red node represents one student, while the edges between nodes indicate the co-occurrences of

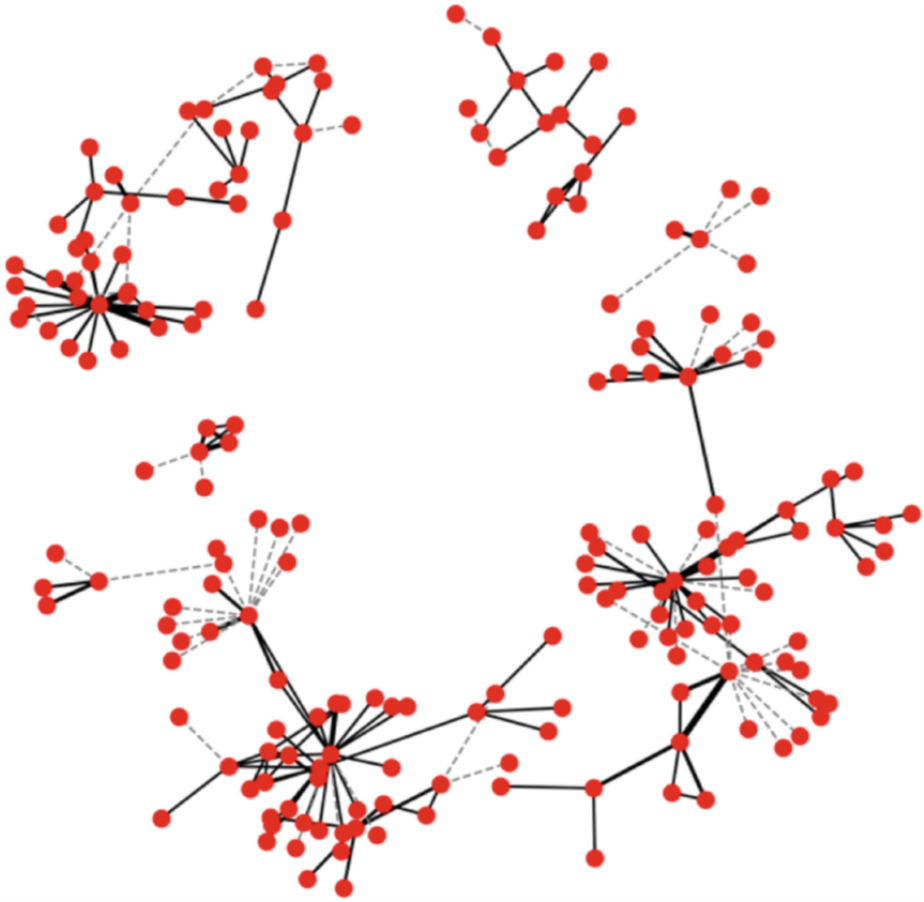

Figure 4.2: A constructed co-occurrence network with $\sigma = 5$.

students when they check-in to the library. The width of the edges shows the number of co-occurrence time between them. I do not construct the co-occurrence network from the LMS log-in traces because the LMS log-in frequency is too high and it will involve too many "familiar strangers" in the network. This will introduce significant biases for learning the social homophily later.

Next step is to learn students' social homophily from the co-occurrence network. Network embedding has been widely applied in encoding the connectivities among nodes as representation and well preserves the graph properties [50] [102]. Here, I embed the co-occurrence network by Node2Vec [32] and learn a representation vector for every node which preserves the connectivities among students. In addition, I

constrain that the learned representation of nodes should be close when they have similar connections. Specifically, I first exploring diverse neighborhoods for every node by a biased random walk. Let us denote c_i as the ith node in the walk. I sample node sequences with transition probability

$$p(c_i = u | c_{i-1} = v) = \begin{cases} \frac{\alpha_{pq} w_{uv}}{Z} & \text{if } (u, v) \in E \\ 0 & \text{Otherwise} \end{cases} \tag{4.1}$$

where Z is a constant for normalization and α_{pq} in Eq. (4.2) is the sampling bias.

$$\alpha_{pq} = \begin{cases} 1/p & \text{if } d_{uv} = 0 \\ 1 & \text{if } d_{uv} = 1 \\ 1/q & \text{if } d_{uv} = 2 \\ 0 & \text{Otherwise} \end{cases} \tag{4.2}$$

d_{uv} denotes the shortest path distance between nodes u and v. Parameters p and q make the trade-offs between depth-first and breadth-first neighborhood sampling.

To learning the final representation of every node, I train a Skip-gram model [71] by maximizing the log-probability of its network neighborhood conditioned on its feature representation as showed in Eq. (4.3) where $f(\cdot)$ is a mapping function from node to feature representations and $N_s(u)$ is u's neighborhood sampling by the above random walk.

$$\max_f \sum_{u \in V} \log \left(\prod_{v_i \in N_s(u)} \frac{\exp\left(f(u) \cdot f(v_i)\right)}{\sum_{v \in V} \exp\left(f(u) \cdot f(v)\right)} \right) \tag{4.3}$$

I adopt the stochastic gradient ascent to optimize the above objective function over the model parameters and obtain the representation of every node which carrying its social homophily. Learning students' social homophily provides extra information for dealing with the data insufficiency issues such that it makes the EPARS have the ability to early predict STAR.

4.5.3 Data Augmentation

To deal with the extremely label imbalance issues, I oversample the STAR by a synthetic minority over-sampling technique (SMOTE) [16] while constructing the training set. For each STAR training sample, denoted as x, I first search its k-nearest neighbors from all STAR samples in training set by the Euclidean distance in the feature space, and the k is set to 10 in the experiment. Next, I randomly select a sample x' from the k nearest neighbors and synthesize a new STAR example by Eq. (4.4) where ω is a random number between 0 and 1.

$$x_{new} = x + (x' - x) \times \omega \tag{4.4}$$

After the data augmentation, STAR have the same amount as the normal students in the training set; this allows the classifier to avoid being dominated by the majority of the normal students during training. SMOTE synthesizes new examples between any of the two existing minority samples by a linear interpolation approach. Compared with a widely used under-sampling technique EasyEnsemble, SMOTE introduces random perturbation into the training set while generating the synthetic examples, which provide the trained classifier better generalization.

4.6 Experiments

Extensive experiments are conducted to showcase the effectiveness of proposed EPARS. In particular, I aim to answer the following research questions (RQ) via experiments:

- RQ1: How effective is the EPARS in predicting STAR?
- RQ2: How early does the EPARS well predict STAR?
- RQ3: How effective is SMOTE for data augmentation in EPARS?
- RQ4: Is the EPARS sensitive to hyper-parameters?

4.6.1 Experiment Protocol

Experiment Setting

In the collected dataset, each student has an independent label of either STAR or the normal student in each semester. Thus, I treat students in different semesters as a whole in the experiments. When predicting STAR at any time t before the end of the semester t_{end}, I extract features from their online and offline learning traces from the beginning of a semester to the current time t. After feature extraction, I synthesize new STAR examples to augment the training set. I conduct experiments under the 5-fold cross-validation setting and repeat 10 times. The average results will be reported in the next subsection. Several classifiers are tested, including the Logistic Regression, Support Vector Machine (SVM), Decision Tree, Random Forest, and the Gradient Boosting Decision Tree (GBDT). GBDT outperforms all other classifiers in the experiments, so I only report the results of GBDT due to the space limit.

Parameter Setting

I set the maximum scale of regularity $S = 4$, the co-occurrence threshold δ to be 30 seconds, the linking threshold $\sigma = 2$, and the dimension of embedding to be 64 for EPARS. I select $k = 10$ neighborhood for SMOTE to augment the training set. The classifier GBDT is trained with parameters that the number of estimators is 100, maximum depth of the decision tree is 10, and the learning rate is 0.1.

Evaluation Metrics

I evaluate the performance of EPARS from two aspects. Since the STAR prediction is a binary classification problem, I adopt Area Under the receiver operating characteristics Curve (AUC) to measure the classification performance. The AUC indicates how capable the model is to distinguish between STAR and the normal students. Moreover, since the focus is to find out the STAR as accurate as possible, I measure the accuracy of the proposed model in predicting STAR by the number of true posi-

tive predictions divided by the total number of STAR in the test set. I denote it as ACC-STAR, which indicates how many percentages of STAR are correctly predicted.

Baseline Approaches.

As mentioned in the introduction, the major contribution is to achieve better STAR early prediction performance, in terms of higher AUC and ACC-STAR, with features extracted from students' learning regularity and social homophily. To verify the effectiveness of EPARS, I set four baseline models, including SF, DA, DA-Reg, and DA-SoH. SF uses only the statistically significant behavior features as input to predict STAR without data augmentation. The process of discovering significant statistical features will be presented in the next paragraph. DA uses the same features as SF and augments the training set using SMOTE. Comparing SF and DA, I can verify whether SMOTE can solve the label imbalance challenge well and results in better classification performance. DA-Reg and DA-SoH integrate the regularity features and the social homophily to the DA, respectively. They are to verify the effectiveness of the proposed multi-timescale bag-of-regularity and the social homophily modeling approach in STAR prediction.

To discover the significant statistical features, I perform an ANOVA (analysis of variance) test to figure out what behaviors are statistically significant for distinguishing between STAR and the normal students. I have 13 kinds of clickstream behaviors on the LMS and 28 kinds of library check-in behaviors at different times of the day and different periods in the semester. Due to the space limited, I report the statistically significant features and some of the insignificant features discovered from the ANOVA in Table 4.2. It is interesting to note that STAR use the LMS less than the normal students, but they will check the announcement and lectures' information more. There is no significant difference in accessing the course materials and checking assignment results. Besides, STAR go to the library less than the normal students at the beginning of a semester. Still, they prefer more to be there after business hours. Lastly, I select the statistically significant features as the SF baseline to benchmark

Table 4.2: Results of the ANOVA test

Features	P-value	F-value	Mean STAR	Mean Others
# LMS Login	0.0020	9.5112	127.4987	144.8043
# LMS Logout	0.0000	34.5301	8.9318	20.1348
# Check announcement	0.0158	5.8311	41.4436	36.8361
# Course access	0.7328	0.1165	4.2677	4.5667
# Grade center access	0.7694	0.0859	10.5486	10.2108
# Discussion board access	0.0020	9.5951	11.7979	19.2444
# Group access	0.0209	5.3385	13.2782	20.1268
# Check personal info	0.0000	16.7953	0.2283	1.6585
# Check lecturer info	0.0000	106.1638	9.7297	5.5440
# Journal page access	0.0199	5.4191	0.2283	1.6585
# Lib check-in	0.0700	3.2829	42.8163	47.3589
# Lib check-in in the morning	0.0001	14.7133	7.0367	9.4206
# Lib check-in in the afternoon	0.0023	9.3196	27.0604	31.9419
# Lib check-in after midnight	0.0000	43.9327	4.0105	1.6927
# Lib check-in before exam months	0.0123	6.2740	33.9265	39.0143
# Lib check-in at the first month	0.0004	12.5447	8.4724	10.6052

the proposed EPARS.

4.6.2 Experimental Results

RQ1: To verify the effectiveness of the proposed EPARS in predicting STAR, I extract features from the whole semester data to train the GBDT and benchmark EPARS with four baselines. This experiment evaluates the performance of EPARS when students' all learning behaviors in a whole semester is known. The results are presented in Tab. 4.3.

Comparing the experimental results between SF and DA, it is confirmed that the

Table 4.3: Results of predicting STAR using the whole semester learning behavior

Metric	SF	DA	DA-Reg	DA-SoH	EPARS
AUC	0.8423	0.8442	0.8611	0.8623	**0.8684**
ACC-STAR	0.5395	0.6079	0.6842	0.6184	**0.7237**

data augmentation approach overcomes the data imbalance challenges to some extent and achieves improvement in both AUC and ACC-STAR. In addition, the regularity features extracted by the multi-timescale bag-of-regularity method can improve the accuracy of predicting STAR a lot, which indicates that the regularity of learning is a distinguished feature between STAR and the normal students, and the multi-timescale bag-of-regularity can well extract their regularity patterns efficiently. Compared with DA-Reg, DA-SoH achieves a higher AUC score and has better overall classification performance. However, its ACC-STAR is much lower than DA-Reg's, suggesting that it cannot identify STAR as accurate as DA-Reg. In other words, social homophily helps identify the normal students a lot rather than recognizing STAR. This shows that the proposed approach is capable of well modeling the social homophily among students. Nevertheless, STAR may have similar linkage patterns with "familiar strangers" in the co-occurrence network since STAR are very handful. Combining the regularity patterns of learning and social homophily, which is the proposed EPARS, achieves the best performance in predicting STA in terms of 19.05%, 5.77% and 17.03% ACC-STAR improvement to DA, DA-Reg and DA-SoH, respectively. This indicates that friends of STAR are more likely to be at-risk if their regularity patterns of learning behaviors are also similar. Therefore, the regularity features can help eliminate the "familiar strangers" and result in better STAR prediction performance.

RQ2: To demonstrate the effectiveness of the proposed methods in early predicting STAR, I conduct experiments in every week's data of the semester. For each week, I extract features of students' learning traces from the beginning of the semester to the

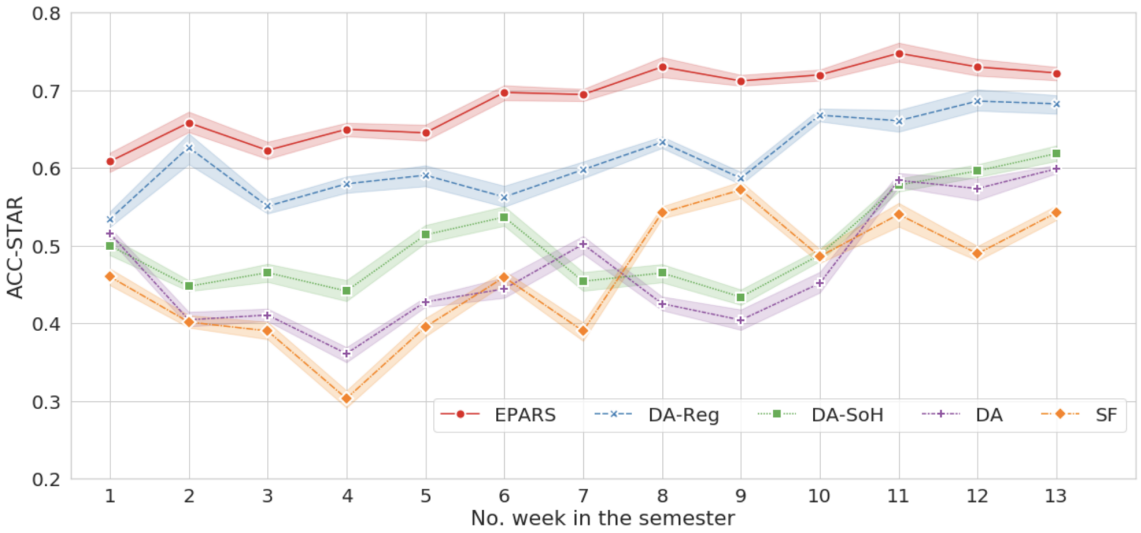

Figure 4.3: Results of STAR early prediction.

end of that week. I repeat the experiment for 10 times, and the average ACC-STAR of early predicting STAR is presented in Fig. (4.3) in which the solid lines are the average ACC-STAR, and the shadows represent the error spans.

The EPARS outperforms all other baselines from the first week to the end of the semester. It is worth mention that the EPARS can correctly predict 61.84% STAR only based on the online and offline learning traces of the students in the first week, which outperforms SF, DA, DA-Reg, and DA-SoH 38.22%, 17.50%, 14.62%, and 22.38%, respectively. In the first four weeks, the prediction performance of SF keeps on decreasing. One possible reason is that some normal students are not active in the beginning of the semester, so that they may have similar behavior patterns with STAR and cause misclassification. Students' social homophily and regularity patterns of learning behaviors are much more discriminable especially in the early stage of a semester. The performance of EPARS is almost converged in the middle of a semester while other baselines are still gradually increasing or concussion. It shows that the EPARS can leverage less information but achieves better performance

Table 4.4: Evaluation of data augmentation

	# STAR after DA	# Normal Std after DA	AUC	ACC-STAR
SF	305	11295	0.8342	0.5526
RU	305	305	0.8211	0.5316
RO	11295	11295	0.8458	0.5645
SMOTE	11295	11295	**0.8684**	**0.7237**

in early predicting STAR.

RQ3: To verify the effectiveness of using SMOTE for dealing with the label imbalance issues, I conduct a comparative experiment among random undersampling (RU), random oversampling (RO) and SMOTE. RU and RO are widely adopted in existing work for STAR prediction [36] [42]. RU randomly deletes examples with the majority labels until the labels of training samples are balanced while RO randomly resamples the minority examples until the numbers of the minority are the same as the majority one. I regard SF as baseline and launch above data augmentation approach for predicting STAR before the end of a semester. I repeat the experiment 10 times and report the average AUC and ACC-STAR in Tab. 4.4.

The first two columns show the number of examples in the training set after data augmentation in each fold of the experiment. Experimental results show that RO slightly outperforms the baselines but the performance of RU is worse than the baselines. In the case of extremely label imbalance, undersampling technique drops most of negative training samples and constructs a very small training set, which cannot provide enough information to well train a classifier. Although RO augments the minority examples by oversampling, most synthesis examples are the same so that the classifier is very easy to overfit and results in poor testing accuracy. SMOTE synthesizes the minority examples by linear interpolation which not only increases the number of minority samples but also enriches the diversity of the training set. Thus, it achieves the best STAR prediction accuracy in such an extremely label imbalance

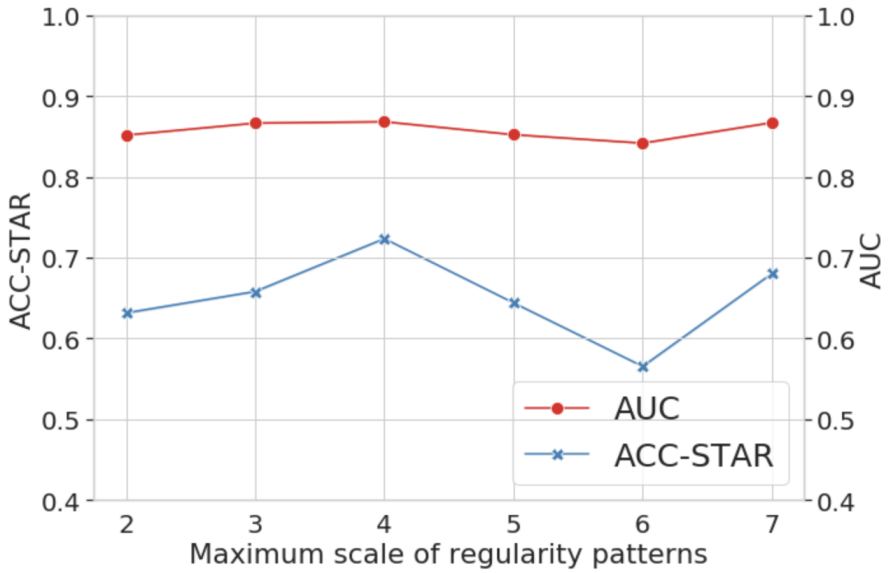

Figure 4.4: Results of testing the maximum timescale S of multi-scale bag-of-regularity.

classification task.

RQ4: I test how sensitive EPARS is to the hyper-parameters and discuss how to select hyper-parameters for EPARS. I focus on three hyper-parameters of EPARS. One is the maximum time-scale S of the multi-scale bag-of-regularity. The other two are co-occurrence threshold δ and linking threshold σ between pairs of students when constructing co-occurrence networks for further modeling the social homophily.

While I are testing the maximum scale S, I fix all other parameters and vary S from 2 to 7 because the minimum time length of the repeated pattern is two days, and the course schedule is a 7-day cycle. The prediction results are shown in Fig. 4.4. I found that the overall classification performance measured by AUC is not sensitive to the maximum scale S, but it affects a lot on the correctness of identifying STAR. EPARS achieves the best performance when $S = 4$. The reason may be in two folds. One reason is that the regularity patterns of the scale 5 to 7 can be synthesized by the scale of 2 to 4. Thus it has already captured almost all regularity when setting

Table 4.5: Results of testing co-occurrence threshold δ

δ	Ave #edge per week	AUC	ACC-STAR
10 seconds	14263	**0.8699**	0.5921
30 seconds	39386	0.8684	**0.7237**
60 seconds	77318	0.8576	0.6316

Table 4.6: Results of testing linking threshold σ

σ	AUC	ACC-STAR
2 times	**0.8684**	**0.7237**
3 times	0.8615	0.6184
4 times	0.8554	0.5658
5 times	0.8122	0.5395

the maximum scale $S = 4$. The other reason is that the output feature vector of multi-scale bag-of-regularity is short and dense when $S = 4$. It will dramatically become sparse when $S \geq 4$ in this cases, which makes the performance worse.

I further test how co-occurrence threshold δ and linking threshold σ affect the modeling of social homophily and present the results in Tab. 4.5 and 4.6. $\delta = 30$ is the best since smaller δ will make the co-occurrence network unable to capture enough social relationship for learning the social homophily and larger δ will introduce a large number of "familiar strangers" which also damages the prediction performance. Similar results are found in the result of testing linking threshold σ. When increase σ, both AUC and ACC-STAR are dropping. The reason is that STAR and some ordinary students go to the library less often than outstanding students so that higher σ may filter out their social interaction and results in worse prediction performance.

4.7 Chapter Summary

In this chapter, I present EPARS, a novel algorithm to extract students' multi-timescale regularity patterns of learning and social homophily from dynamic learning behaviors for early predicting STAR. One of the major contributions is to devise a multi-timescale bag-of-regularity method to extract regularity features from multi-timescale dynamic learning behaviors, which is robust for sparse data. In addition, I model students' social relationships by constructing a co-occurrence network from library check-in records and embed their social homophily as feature vectors. Before training a classifier, I oversample the minority examples to overcome the label imbalance issues. Extensive experiments are conducted on a large scale dataset covering all undergraduate students in the whole university. Experimental results indicate that the EPARS improves the accuracy of baselines by 14.62% $\sim$ 38.22% and 5.77% $\sim$ 34.14% in predicting STAR in the first week and the last week of a semester, respectively. The research papers [105] [103] [106] arising from this study has been respectively published in proceedings of the 25th International Conference on Database Systems for Advanced Applications (DASFAA), the 13th International Conference on Blended Learning (ICBL) and 4th IEEE International Conference on Orange Technologies (ICOT).

Chapter 5

Conclusions and Future Directions

In this thesis, I study an important yet overlooked problem that is feature representation in dynamic data for discovering and embedding complex evolution patterns. I investigated the challenging issues and identified three grand challenges brought by dynamic data to feature representation. The first one is multi-variate dynamics. It causes the interrelationship amongst multiple variables to change over time and introduces additional noise, stochastic, and uncertainty, thus making the embedding of evolution patterns difficult. Second, data asynchronously evolve over time. It evolves at different times with variant evolution speed, which makes the evolution patterns very complicated, thereby being challenging to capture and embed in feature representation. Lastly, the dynamics of data vary in multiple timescales. At each timescale, dynamic variables or objects have their own changing patterns with different evolving periods and eventually constitute the overall evolutions. Hence, it is challenging for feature representation to fully capture the overall picture of the entire evolutions.

In light of these challenges, I proposed three novel feature representation algorithms to respectively discover and embed the patterns of multivariate synchronous evolutions, synchronous evolutions, and multi-timescale evolutions from the dynamic graphs and the multi-variate time series. Specifically, in Chapter 2, I presented a time-capturing

dynamic graph embedding algorithm to learn the asynchronous linkage evolution amongst vertices while accounting for their evolution duration. The dynamic graph was modeled as a snapshot graph sequence appending with the timespan of edges (ToE), thereby well capturing the multivariate synchronous linkage evolution for embedding. A linear regressor was co-trained to embed ToE while inferring a common latent space for capturing the structural difference amongst consecutive snapshots by a matrix-factorization-based model, thereby successfully embedding vertices' synchronous linkage evolution and achieving dramatic performance improvement in graph mining applications.

In Chapter 3, I designed a time-aware dynamic graph embedding algorithm to fully capture and embed the asynchronous structural evolutions in a dynamic graph. The dynamic graphs were innovatively formulated as temporal edge sequences associated with ToE and ToV (joining time of vertices) to fully capture the asynchronous structural dynamics for embedding. The dynamic connection changes with ToE and the asynchronous evolution starting time of every local structure were simultaneously embedded by a Time-aware Transformer and a structural embedding model, thus effectively and time-efficiently preserving the asynchronous structural evolution and eventually outperforming the state-of-the-art in graph mining applications.

It is worth mentioning that both of the dynamic graph embedding algorithms proposed in Chapter 2 and 3 are trained under self-supervision which are general feature representation methods being independent of applications. It applicable for many kinds of graph tasks, such as link prediction, vertex classification, vertex clustering, vertex ranking, graph kernels, and higher-order graph analysis. In addition to graph data, these approaches are also applicable to diverse non-graph data such as time-series, grid, text, etc. for capturing and embedding dynamics.

In Chapter 4, I devised a multi-timescale bag-of-regularity method to extract students' learning regularity patterns from their multi-timescale dynamic learning behaviors. In addition, I model students' social relationships by constructing a co-occurrence net-

work from library check-in records and embed their social homophily as feature vectors. With the fused features of learning regularity and social homophily, it achieved significant accuracy improvement in early predicting academic at-risk students.

Although this thesis presents several novel methods for feature representation in dynamic data, several open challenges are remaining to be addressed in the future. First of all, dynamic data are generated in a streaming manner while most of the existing feature representation methods, especially the representation learning approaches, process them batch-by-batch, thereby failing to embed the real-time dynamic changes in time-sensitive applications. When new data is arriving at the system in a very rapid manner, the information it carries is much less than that accumulated in the historical data. This brings the two issues. One is difficult to accurately discriminate whether it is an outlier or a new change which is significantly important. The other is how to effectively update the long-term and short-term evolutions with this small amount of information. Long-term evolutions usually develop slightly and slowly, while the short-term ones may change dramatically in an instant. These two major issues point out a promising direction for the future development of feature representation.

The other open challenge is causal reasoning. The advantage of feature representation is its super ability to extract comprehensive and discriminative information from data and further transform them as knowledge about the world for solving complex problems. It enables the intelligent agent to understand the world, which is the foundation of causal reasoning. How to leverage data-driven approaches to determine causal relationships instead of correlation is an open research problem. In addition, the reasoning should not be stationary. Similarly, with the continuous development of human understanding of the world and things, reasoning results under the same conditions should change over time, making them dynamic. How to achieve dynamic causal reasoning becomes another open problem. I believe that solving these grand problems will lead the whole area towards the era of artificial general intelligence.

References

[1] Douglas E Adams and Madhura Nataraju. A nonlinear dynamical systems framework for structural diagnosis and prognosis. *International Journal of Engineering Science*, 40(17):1919–1941, 2002.

[2] Michal Aharon, Michael Elad, and Alfred Bruckstein. K-svd: An algorithm for designing overcomplete dictionaries for sparse representation. *IEEE Transactions on signal processing*, 54(11):4311–4322, 2006.

[3] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[4] Herbert Bay, Andreas Ess, Tinne Tuytelaars, and Luc Van Gool. Speeded-up robust features (surf). *Computer vision and image understanding*, 110(3):346–359, 2008.

[5] Inci M Baytas, Cao Xiao, Xi Zhang, Fei Wang, Anil K Jain, and Jiayu Zhou. Patient subtyping via time-aware lstm networks. In *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 65–74, 2017.

[6] Amir Beck and Marc Teboulle. A fast iterative shrinkage-thresholding algorithm for linear inverse problems. *SIAM journal on imaging sciences*, 2(1):183–202, 2009.

[7] Nathaniel Beck and Jonathan N Katz. Modeling dynamics in time-series–cross-section political economy data. *Annual Review of Political Science*, 14:331–352, 2011.

[8] Peter N. Belhumeur, Joao P Hespanha, and David J. Kriegman. Eigenfaces vs. fisherfaces: Recognition using class specific linear projection. *IEEE Transactions on pattern analysis and machine intelligence*, 19(7):711–720, 1997.

[9] Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1798–1828, 2013.

[10] Johannes Berens, Kerstin Schneider, Simon Görtz, Simon Oster, and Julian Burghoff. Early detection of students at risk–predicting student dropouts using administrative student data and machine learning methods. *CESifo Working Paper Series*, 2018.

[11] Ferenc Béres, Domokos M Kelen, Róbert Pálovics, and András A Benczúr. Node embeddings in dynamic graphs. *Applied Network Science*, 4(1):64, 2019.

[12] Dimitri Bertsekas. On the goldstein-levitin-polyak gradient projection method. *IEEE Transactions on automatic control*, 21(2):174–184, 1976.

[13] Steven L Brunton and J Nathan Kutz. *Data-driven science and engineering: Machine learning, dynamical systems, and control*. Cambridge University Press, 2019.

[14] Shaosheng Cao, Wei Lu, and Qiongkai Xu. Grarep: Learning graph representations with global structural information. In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*, pages 891–900, 2015.

[15] Xiaofu Chang, Xuqin Liu, Jianfeng Wen, Shuang Li, Yanming Fang, Le Song, and Yuan Qi. Continuous-time dynamic graph learning via neural interaction

processes. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, pages 145–154, 2020.

[16] Nitesh V Chawla, Kevin W Bowyer, Lawrence O Hall, and W Philip Kegelmeyer. Smote: synthetic minority over-sampling technique. *Journal of artificial intelligence research*, 16:321–357, 2002.

[17] Dong Chen, Xudong Cao, Fang Wen, and Jian Sun. Blessing of dimensionality: High-dimensional feature and its efficient compression for face verification. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3025–3032, 2013.

[18] Hongxu Chen, Hongzhi Yin, Tong Chen, Quoc Viet Hung Nguyen, Wen-Chih Peng, and Xue Li. Exploiting centrality information with graph convolutions for network representation learning. In *Proceedings of the 35th IEEE International Conference on Data Engineering*, pages 590–601, 2019.

[19] Samuel PM Choi, Sze Sing Lam, Kam Cheong Li, and Billy TM Wong. Learning analytics at low cost: at-risk student prediction with clicker data and systematic proactive interventions. *Journal of Educational Technology & Society*, 21(2):273–290, 2018.

[20] Fritz Colonius and Wolfgang Kliemann. *The dynamics of control*. Springer Science & Business Media, 2012.

[21] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[22] Hugh Drummond. The nature and description of behavior patterns. In *Perspectives in ethology*, pages 1–33. Springer, 1981.

[23] Stephen Ellenbogen and Claire Chamberland. The peer relations of dropouts: a comparative study of at-risk and not at-risk youths. *Journal of adolescence*, 20(4):355–367, 1997.

[24] Ahmed Fathy and Kan Li. Temporalgat: Attention-based dynamic graph representation learning. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 413–423, 2020.

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

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

[27] Palash Goyal, Sujit Rokka Chhetri, and Arquimedes Canedo. dyngraph2vec: Capturing network dynamics using dynamic graph representation learning. *Knowledge-Based Systems*, 187:104816, 2020.

[28] Palash Goyal, Homa Hosseinmardi, Emilio Ferrara, and Aram Galstyan. Capturing edge attributes via network embedding. *arXiv preprint arXiv:1805.03280*, 2018.

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

[30] Sander Greenland, Mohammad Ali Mansournia, and Douglas G Altman. Sparse data bias: a problem hiding in plain sight. *BMJ*, 352:i1981, 2016.

[31] Edwin R Griff and Stephen F Matter. Early identification of at-risk students using a personal response system. *British Journal of Educational Technology*, 39(6):1124–1130, 2008.

[32] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 855–864, 2016.

[33] Yupeng Gu, Yizhou Sun, Yanen Li, and Yang Yang. Rare: Social rank regulated large-scale network embedding. In *Proceedings of the 2018 World Wide Web Conference*, pages 359–368, 2018.

[34] Shengnan Guo, Youfang Lin, Huaiyu Wan, Xiucheng Li, and Gao Cong. Learning dynamics and heterogeneity of spatial-temporal graph data for traffic forecasting. *IEEE Transactions on Knowledge and Data Engineering*, 2021.

[35] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*, pages 1024–1034, 2017.

[36] Jiazhen He, James Bailey, Benjamin IP Rubinstein, and Rui Zhang. Identifying at-risk students in massive open online courses. In *Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

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

[38] Geoffrey E Hinton and Richard S Zemel. Autoencoders, minimum description length, and helmholtz free energy. *Advances in neural information processing systems*, 6:3–10, 1994.

[39] Li Chin Ho and Kyong Jin Shim. Data mining approach to the identification of at-risk students. In *2018 IEEE International Conference on Big Data (Big Data)*, pages 5333–5335. IEEE, 2018.

[40] Di Huang, Caifeng Shan, Mohsen Ardabilian, Yunhong Wang, and Liming Chen. Local binary patterns and its application to facial image analysis: a

survey. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 41(6):765–781, 2011.

[41] Trachette Jackson and Ami Radunskaya. *Applications of dynamical systems in biology and medicine*, volume 158. Springer, 2015.

[42] Sandeep M Jayaprakash, Erik W Moody, Eitel JM Lauría, James R Regan, and Joshua D Baron. Early alert of academically at-risk students: An open source analytics initiative. *Journal of Learning Analytics*, 1(1):6–47, 2014.

[43] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[44] Nobuhiko Kondo, Midori Okubo, and Toshiharu Hatanaka. Early detection of at-risk students using machine learning based on lms log data. In *2017 6th IIAI International Congress on Advanced Applied Informatics (IIAI-AAI)*, pages 198–201. IEEE, 2017.

[45] Yu Kong and Yun Fu. Human action recognition and prediction: A survey. *arXiv preprint arXiv:1806.11230*, 2018.

[46] Irena Koprinska, Joshua Stretton, and Kalina Yacef. Students at risk: Detection and remediation. In *Proceedings of the 8th International Conference on Educational Data Mining*, pages 512–515, 2015.

[47] Srijan Kumar, William L Hamilton, Jure Leskovec, and Dan Jurafsky. Community interaction and conflict on the web. In *Proceedings of the 2018 World Wide Web Conference on World Wide Web*, pages 933–943, 2018.

[48] Srijan Kumar, Francesca Spezzano, VS Subrahmanian, and Christos Faloutsos. Edge weight prediction in weighted signed networks. In *Proceedings of the 16th IEEE International Conference on Data Mining*, pages 221–230, 2016.

[49] Srijan Kumar, Xikun Zhang, and Jure Leskovec. Predicting dynamic embedding trajectory in temporal interaction networks. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1269–1278, 2019.

[50] Chaozhuo Li, Senzhang Wang, Dejian Yang, Zhoujun Li, Yang Yang, Xiaoming Zhang, and Jianshe Zhou. Ppne: property preserving network embedding. In *International Conference on Database Systems for Advanced Applications*, pages 163–179. Springer, 2017.

[51] Juzheng Li, Jun Zhu, and Bo Zhang. Discriminative deep random walk for network classification. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, pages 1004–1013, 2016.

[52] Yali Li, Shengjin Wang, Qi Tian, and Xiaoqing Ding. Feature representation for statistical-learning-based object detection: A review. *Pattern Recognition*, 48(11):3542–3559, 2015.

[53] Chih-Jen Lin. Projected gradient methods for nonnegative matrix factorization. *Neural computation*, 19:2756–2779, 2007.

[54] Zhiyuan Liu, Yankai Lin, and Maosong Sun. *Representation learning for natural language processing*. Springer Nature, 2020.

[55] David G Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110, 2004.

[56] Yuanfu Lu, Xiao Wang, Chuan Shi, Philip S Yu, and Yanfang Ye. Temporal network embedding with micro-and macro-dynamics. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, pages 469–478, 2019.

[57] Yisheng Lv, Yanjie Duan, Wenwen Kang, Zhengxi Li, and Fei-Yue Wang. Traffic flow prediction with big data: a deep learning approach. *IEEE Transactions on Intelligent Transportation Systems*, 16(2):865–873, 2014.

[58] Jianxin Ma, Peng Cui, and Wenwu Zhu. Depthlgp: Learning embeddings of out-of-sample nodes in dynamic networks. In *AAAI*, 2018.

[59] Franco Manessi, Alessandro Rozza, and Mario Manzo. Dynamic graph convolutional networks. *Pattern Recognition*, 97:107000, 2020.

[60] Farshid Marbouti, Heidi A. Diefes-Dux, and Krishna Madhavan. Models for early prediction of at-risk students in a course using standards-based grading. *Computers and Education*, 103:1 – 15, 2016.

[61] Peter V Marsden. Homogeneity in confiding relations. *Social networks*, 10(1):57–76, 1988.

[62] John McCarthy, Marvin L Minsky, Nathaniel Rochester, and Claude E Shannon. A proposal for the dartmouth summer research project on artificial intelligence, august 31, 1955. *AI magazine*, 27(4):12–12, 2006.

[63] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[64] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

[65] Giang Hoang Nguyen, John Boaz Lee, Ryan A Rossi, Nesreen K Ahmed, Eunyee Koh, and Sungchul Kim. Continuous-time dynamic network embeddings. In *3rd International Workshop on Learning Representations for Big Networks*, 2018.

[66] Tore Opsahl and Pietro Panzarasa. Clustering in weighted networks. *Social networks*, 31(2):155–163, 2009.

[67] Ricardo Orozco, Corina Benjet, Guilherme Borges, María Fátima Moneta Arce, Diana Fregoso Ito, Clara Fleiz, and Jorge Ameth Villatoro. Association between attempted suicide and academic performance indicators among middle and high school students in mexico: results from a national survey. *Child and adolescent psychiatry and mental health*, 12(1):9, 2018.

[68] Timothy N Palmer. A nonlinear dynamical perspective on climate prediction. *Journal of Climate*, 12(2):575–591, 1999.

[69] Ashwin Paranjape, Austin R Benson, and Jure Leskovec. Motifs in temporal networks. In *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining*, pages 601–610, 2017.

[70] Hao Peng, Jianxin Li, Hao Yan, Qiran Gong, Senzhang Wang, Lin Liu, Lihong Wang, and Xiang Ren. Dynamic network embedding via incremental skip-gram with negative sampling. *Science China Information Sciences*, 63(10):1–19, 2020.

[71] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 701–710, 2014.

[72] V Richardson. At-risk student intervention implementation guide. *The Educationand Economic Development Coordinating Council At-Risk Student Committee*, page 18, 2005.

[73] Emanuele Rossi, Ben Chamberlain, Fabrizio Frasca, Davide Eynard, Federico Monti, and Michael Bronstein. Temporal graph networks for deep learning on dynamic graphs. *arXiv preprint arXiv:2006.10637*, 2020.

[74] Aravind Sankar, Yanhong Wu, Liang Gou, Wei Zhang, and Hao Yang. Dynamic graph representation learning via self-attention networks. *arXiv preprint arXiv:1812.09430*, 2018.

[75] Aravind Sankar, Yanhong Wu, Liang Gou, Wei Zhang, and Hao Yang. Dysat: Deep neural representation learning on dynamic graphs via self-attention networks. In *Proceedings of the 13th International Conference on Web Search and Data Mining*, pages 519–527, 2020.

[76] Brett E Shelton, Juan Yang, Jui-Long Hung, and Xu Du. Two-stage predictive modeling for identifying at-risk students. In *International Conference on Innovative Technologies and Learning*, pages 578–583. Springer, 2018.

[77] Jiaxing Shen, Jiannong Cao, and Xuefeng Liu. Bag: Behavior-aware group detection in crowded urban spaces using wifi probes. In *The World Wide Web Conference*, pages 1669–1678. ACM, 2019.

[78] Jiaxing Shen, Jiannong Cao, Xuefeng Liu, and Shaojie Tang. Snow: Detecting shopping groups using wifi. *IEEE Internet of Things Journal*, 5(5):3908–3917, 2018.

[79] Linlin Shen and Li Bai. A review on gabor wavelets for face recognition. *Pattern analysis and applications*, 9(2):273–292, 2006.

[80] Ralph Stinebrickner and Todd Stinebrickner. Academic performance and college dropout: Using longitudinal expectations data to estimate a learning model. *Journal of Labor Economics*, 32(3):601–644, 2014.

[81] Xiangguo Sun, Hongzhi Yin, Bo Liu, Hongxu Chen, Jiuxin Cao, Yingxia Shao, and Nguyen Quoc Viet Hung. Heterogeneous hypergraph embedding for graph classification. In *Proceedings of the 14th ACM International Conference on Web Search and Data Mining*, pages 725–733, 2021.

[82] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.

[83] Rakshit Trivedi, Mehrdad Farajtabar, Prasenjeet Biswal, and Hongyuan Zha. Dyrep: Learning representations over dynamic graphs. In *7th International Conference on Learning Representations, ICLR 2019*, 2019.

[84] Paul Tseng. Convergence of a block coordinate descent method for non-differentiable minimization. *Journal of optimization theory and applications*, 109(3):475–494, 2001.

[85] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[86] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

[87] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *International Conference on Learning Representations*, 2018.

[88] Hua Wang, Feiping Nie, Heng Huang, Jingwen Yan, Sungeun Kim, Shannon Risacher, Andrew Saykin, and Li Shen. High-order multi-task feature learning to identify longitudinal phenotypic markers for alzheimer's disease progression prediction. *Advances in neural information processing systems*, 25:1277–1285, 2012.

[89] Jia Wang, Jiannong Cao, Wei Li, and Senzhang Wang. Cane: community-aware network embedding via adversarial training. *Knowledge and Information Systems*, 63(2):411–438, 2021.

[90] Junshan Wang, Yilun Jin, Guojie Song, and Xiaojun Ma. Epne: Evolutionary pattern preserving network embedding. In *Proceedings of the 24th European Conference on Artificial Intelligence*, pages 1603–1610, 2020.

[91] Mei Wang and Weihong Deng. Deep face recognition: A survey. *arXiv preprint arXiv:1804.06655*, 2018.

[92] Xiao Wang, Peng Cui, Jing Wang, Jian Pei, Wenwu Zhu, and Shiqiang Yang. Community preserving network embedding. In *AAAI*, pages 203–209, 2017.

[93] Yu-Xiong Wang and Yu-Jin Zhang. Nonnegative matrix factorization: A comprehensive review. *IEEE Transactions on knowledge and data engineering*, 25(6):1336–1353, 2012.

[94] Yuandong Wang, Hongzhi Yin, Tong Chen, Chunyang Liu, Ben Wang, Tianyu Wo, and Jie Xu. Passenger mobility prediction via representation learning for dynamic directed and weighted graph. *arXiv preprint arXiv:2101.00752*, 2021.

[95] Zhiwei Wang, Yao Ma, Zitao Liu, and Jiliang Tang. R-transformer: Recurrent neural network enhanced transformer. *arXiv preprint arXiv:1907.05572*, 2019.

[96] Svante Wold, Kim Esbensen, and Paul Geladi. Principal component analysis. *Chemometrics and intelligent laboratory systems*, 2(1-3):37–52, 1987.

[97] Lei Wu, Steven CH Hoi, and Nenghai Yu. Semantics-preserving bag-of-words models and applications. *IEEE Transactions on Image Processing*, 19(7):1908–1920, 2010.

[98] Yun Xiong, Yao Zhang, Hanjie Fu, Wei Wang, Yangyong Zhu, and S Yu Philip. Dyngraphgan: Dynamic graph embedding via generative adversarial networks. In *Proceedings of the International Conference on Database Systems for Advanced Applications*, pages 536–552, 2019.

[99] Linchuan Xu, Xiaokai Wei, Jiannong Cao, and Philip S Yu. Embedding of embedding: Joint embedding for coupled heterogeneous networks. In *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining*, pages 741–749, 2017.

[100] Linchuan Xu, Xiaokai Wei, Jiannong Cao, and Philip S Yu. On exploring semantic meanings of links for embedding social networks. In *Proceedings of the 2018 World Wide Web Conference*, pages 479–488, 2018.

[101] Cheng Yang, Zhiyuan Liu, Deli Zhao, Maosong Sun, and Edward Y Chang. Network representation learning with rich text information. In *IJCAI*, pages 2111–2117, 2015.

[102] Dejian Yang, Senzhang Wang, Chaozhuo Li, Xiaoming Zhang, and Zhoujun Li. From properties to links: Deep network embedding on incomplete graphs. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, pages 367–376, 2017.

[103] Yu Yang, Jiannong Cao, Jiaxing Shen, Ruosong Yang, and Zhiyuan Wen. Learning analytics based on multilayer behavior fusion. In *13th International Conference on Blended Learning*, pages 15–24. Springer, 2020.

[104] Yu Yang, Jiannong Cao, Milos Stojmenovic, Senzhang Wang, Yiran Cheng, Chun Lum, and Zhetao Li. Time-capturing dynamic graph embedding for temporal linkage evolution. *IEEE Transactions on Knowledge and Data Engineering*, 2021.

[105] Yu Yang, Zhiyuan Wen, Jiannong Cao, Jiaxing Shen, Hongzhi Yin, and Xiaofang Zhou. Epars: Early prediction of at-risk students with online and offline learning behaviors. In *International Conference on Database Systems for Advanced Applications*, pages 3–19. Springer, 2020.

[106] Yu Yang, Hanqing Wu, and Jiannong Cao. Smartlearn: Predicting learning performance and discovering smart learning strategies in flipped classroom. In *2016 IEEE International Conference on Orange Technologies (ICOT)*, pages 92–95. IEEE, 2016.

[107] Huaxiu Yao, Defu Lian, Yi Cao, Yifan Wu, and Tao Zhou. Predicting academic performance for college students: A campus behavior perspective. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 10(3):24, 2019.

[108] Kai Yu, Tong Zhang, Yihong Gong, et al. Nonlinear learning using local coordinate coding. In *NIPS*, volume 22, pages 2223–2231. Citeseer, 2009.

[109] Le Yu, Bowen Du, Xiao Hu, Leilei Sun, Liangzhe Han, and Weifeng Lv. Deep spatio-temporal graph convolutional network for traffic accident prediction. *Neurocomputing*, 423:135–147, 2021.

[110] Wenchao Yu, Charu C Aggarwal, and Wei Wang. Temporally factorized network modeling for evolutionary network analysis. In *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining*, pages 455–464, 2017.

[111] Daokun Zhang, Jie Yin, Xingquan Zhu, and Chengqi Zhang. Collective classification via discriminative matrix factorization on sparsely labeled networks. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*, pages 1563–1572, 2016.

[112] Daokun Zhang, Jie Yin, Xingquan Zhu, and Chengqi Zhang. User profile preserving social network embedding. In *IJCAI*, pages 3378–3384, 2017.

[113] Wenchao Zhang, Shiguang Shan, Wen Gao, Xilin Chen, and Hongming Zhang. Local gabor binary pattern histogram sequence (lgbphs): A novel non-statistical model for face representation and recognition. In *Tenth IEEE International Conference on Computer Vision (ICCV'05) Volume 1*, volume 1, pages 786–791. IEEE, 2005.

[114] Ziwei Zhang, Peng Cui, Jian Pei, Xiao Wang, and Wenwu Zhu. Timers: Error-bounded svd restart on dynamic networks. In *AAAI*, 2018.

[115] Pengpeng Zhao, Anjing Luo, Yanchi Liu, Fuzhen Zhuang, Jiajie Xu, Zhixu Li, Victor S Sheng, and Xiaofang Zhou. Where to go next: A spatio-temporal

gated network for next poi recommendation. *IEEE Transactions on Knowledge and Data Engineering*, 2020.

[116] Alice Zheng and Amanda Casari. *Feature engineering for machine learning: Principles and techniques for data scientists*. O'Reilly Media, Inc., 2018.

[117] Dawei Zhou, Lecheng Zheng, Jiawei Han, and Jingrui He. A data-driven graph generative model for temporal interaction networks. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 401–411, 2020.

[118] Le-kui Zhou, Yang Yang, Xiang Ren, Fei Wu, and Yueting Zhuang. Dynamic network embedding by modeling triadic closure process. In *AAAI*, 2018.

[119] Dingyuan Zhu, Peng Cui, Ziwei Zhang, Jian Pei, and Wenwu Zhu. High-order proximity preserved embedding for dynamic networks. *IEEE Transactions on Knowledge and Data Engineering*, 2018.

[120] Linhong Zhu, Dong Guo, Junming Yin, Greg Ver Steeg, and Aram Galstyan. Scalable temporal latent space inference for link prediction in dynamic social networks. *IEEE Transactions on Knowledge and Data Engineering*, 28(10):2765–2777, 2016.

[121] Yuan Zuo, Guannan Liu, Hao Lin, Jia Guo, Xiaoqian Hu, and Junjie Wu. Embedding temporal network via neighborhood formation. In *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 2857–2866, 2018.