© 2024 Dongqi Fu

EMPOWERING GRAPH INTELLIGENCE VIA NATURAL AND ARTIFICIAL DYNAMICS

BY

DONGQI FU

DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Computer Science
in the Graduate College of the
University of Illinois Urbana-Champaign, 2024

Urbana, Illinois

Doctoral Committee:

Professor Tarek Abdelzaher
Professor Jiawei Han
Professor Jingrui He, Chair
Professor Ross Maciejewski, Arizona State University

ABSTRACT

In the era of big data, the relationship between entities has become much more complex than ever before. As a kind of relational data structure, graph attracts much research attention for dealing with this unprecedented phenomenon. The real-world scenarios usually bring two fundamental and pragmatic challenges to graph research. First, the graph structure and features may be complex over time (i.e., time-evolving topological structures, time-evolving node/graph features/labels, etc.). Without proper time information leverage, the resulting problems include but are not limited to ignoring entity temporal correlation, overlooking causality discovery, computation inefficiency, non-generalization, etc. Second, the initial topological structure and node or graph features may be imperfect (e.g., having construction errors, sampling noises, missing features, scarce labels, hard-to-interpret, redundant, privacy-leaking, robustness-lacking, etc.). The corresponding problems include but are not limited to non-robustness, indiscriminative representations, and non-generalization.

Inspired by the above two kinds of problems, my research focuses on **natural dynamics** and **artificial dynamics** for graphs. Natural dynamics can be illustrated as disentangling the spatial-temporal complexity of input graphs with evolving components, e.g., the topology structures and (sub)graph-level features are dependent on time. As for artificial dynamics in graphs, this concept describes how researchers change the existing or construct the non-existing graph-related elements, e.g., graph topology, node/graph attributes, graph neural network (GNN) layer connections, and GNN gradients. In general, studying natural dynamics and artificial dynamics is investigating how to leverage spatial-temporal properties of graphs and augment and prune graph components to upgrade graph-based AI performance in terms of effectiveness, efficiency, trustworthiness, etc.

In this thesis, we propose to build the algorithmic foundation for the next-generation graph AI development with three main pillars, i.e., **natural dynamics of graphs**, **artificial dynamics of graphs**, and **natural + artificial dynamics of graphs**. For example, to name a few, (1) we first finished a literature review for the natural and artificial dynamics of graphs in terms of concept, progress, and future; (2) by studying natural dynamics, we developed more accurate graph classification and more efficient graph alignment algorithms; (3) by studying artificial dynamics, we have developed the explainable node and graph classification algorithms and a powerful graph neural computational framework; (4) by studying natural + artificial dynamics, we obtained efficient algorithms for high-order graph clustering and partitioning algorithms.

ACKNOWLEDGMENTS

In my Ph.D. journey, I am very grateful for receiving valuable support from my advisor, my collaborators, my department, my friends, and my family. Without them, I cannot arrive where I am now.

First and foremost, I would like to express my sincere thanks to my advisor, Prof. Jingrui He. She not only gave me professional academic and research guidance but also taught me how to be a mature researcher and cared so much about my feelings and emotions. She established an excellent role model for me and shaped my life. It is my great honor and fortune to have her be my advisor.

Second, I would like to thank my great professors from the thesis committee, Prof. Ross Maciejewski, Prof. Jiawei Han, and Prof. Tarek Abdelzaher. They gave me really helpful suggestions from different backgrounds and helped me learn the world. To be more specific, I would like to wholeheartedly thank Prof. Hanghang Tong, who always paid attention to my growth and gave me in-time suggestions for my research and career path.

Third, I also want to thank my amazing collaborators from the UIUC iSAIL Lab, UIUC IDEA Lab, Meta, and IBM Research. Their assistance was not only limited to our publications but also helped me obtain domain-specific knowledge, time management ability, and communication skills. Here, I also want to thank the platform of our esteemed Department of Computer Science at the University of Illinois Urbana-Champaign. This remarkable platform expanded my horizons and sent me to the next stage.

Last but not least, I hope my family can accept my lated thanks. My wife is my treasure; she cares about my life, teaches me lessons, reminds me of everything, bears me everything, and brightens my world. To be honest, I can not find a word to express my ultimate thanks to my parents because every word or sentence is tiny compared to what they have done for me. As I grew up, I could feel how hard they raised me up; I could even touch their fear, their happiness, their laugh, and their tears at that time when they were young.

TABLE OF CONTENTS

CHAPTER 1 INTRODUCTION 1
1.1 Background and Motivation 1
1.2 Contribution and Impact 2
1.3 General Notation 3

CHAPTER 2 LITERATURE REVIEW AND ANALYSIS 4
2.1 Overview of Graph AI Research 4
2.2 Why We Need Different Dynamics of Graphs 5
2.3 Organization of Thesis 11

CHAPTER 3 EMPOWERING NODE-LEVEL GRAPH INTELLIGENCE VIA NATURAL AND ARTIFICIAL DYNAMICS 13
3.1 Increasing Node Classification Efficiency and Accuracy via Artificial Dynamics 13
3.2 Obtaining Explainable Node Representation via Artificial Dynamics 28
3.3 Achieving Robust Node Representation via Natural Dynamics 35

CHAPTER 4 EMPOWERING SUBGRAPH-LEVEL GRAPH INTELLIGENCE VIA NATURAL AND ARTIFICIAL DYNAMICS 44
4.1 Increasing Link Prediction Efficiency and Accuracy via Natural Dynamics . . 44
4.2 Local High-Order Graph Clustering via Natural + Artificial Dynamics . . . 57
4.3 Fair High-Order Graph Partitioning via Natural + Artificial Dynamics . . . 75

CHAPTER 5 EMPOWERING GRAPH-LEVEL GRAPH INTELLIGENCE VIA NATURAL AND ARTIFICIAL DYNAMICS 95
5.1 Increasing Graph Classification Accuracy via Natural Dynamics 95
5.2 Increasing Graph Classification Efficiency via Natural Dynamics 108
5.3 Enhancing Graph Matching and Alignment Efficiency and Accuracy via Natural Dynamics 116

CHAPTER 6 EMPOWERING COMBINED-LEVEL GRAPH INTELLIGENCE VIA NATURAL AND ARTIFICIAL DYNAMICS 126
6.1 Fine-Grained Causal Graph Structure Discovery via Natural Dynamics . . . 126
6.2 Privacy-Guaranteed and Utility-Preserved Graph Generation via Natural + Artificial Dynamics 139

CHAPTER 7 CONCLUSION AND FUTURE WORK 157

REFERENCES 160

CHAPTER 1: INTRODUCTION

1.1 BACKGROUND AND MOTIVATION

In the age of big data, the relationships between entities have grown vastly more intricate than ever before. Graphs, a type of relational data structure, have garnered substantial research interest as they offer effective means to grapple with this unprecedented complexity. To be specific, for this ubiquitous data structure, many AI algorithms and tools are proposed and get applied for various application tasks, such as computer vision [1], natural language processing [2], recommender systems [3], drug discovery [4], to name a few. Different from images and texts, graph data is typically non-grid, non-Euclidean, non-IID, and relational, which requires more research attention in adapting/developing deep learning and AI algorithms to/on graph data [5, 6].

Over the long term, graph-based AI methods encounter two overarching challenges as they strive to adapt to the intricacies of structured and attribute-rich data [7, 8, 9]. To be specific, firstly, the graph structure and associated features can be quite intricate and subject to change over time. This includes evolving topological structures and dynamic node or graph features and labels. Failing to incorporate temporal information properly can lead to a range of issues, such as neglecting the temporal relationships between entities, overlooking the discovery of causality, computational inefficiency, and limited generalization, among others. Secondly, the initial topological structure and the features of nodes or graphs may not be perfect; they may contain construction errors, sampling noise and redundancies, missing data, limited label information, and difficult-to-interpret and sensitive attributes. As a consequence, this can result in problems like a lack of robustness, indistinctive representations, limited generalization, and privacy leakage. As a result, a subsequent question arises: how can researchers or practitioners intentionally (1) leverage the latent spatial-temporal information of graphs and (2) polish the provided graph components to address and bridge these gaps effectively?

To answer these questions, in this thesis, we propose to study two kinds of dynamics of graphs, i.e., **natural dynamics** and **artificial dynamics** towards **various aspects of graph AI research concerns** like task effectiveness, efficiency, trustworthiness, etc.

- Natural dynamics highlight that the input graphs themselves undergo evolution; this encompasses changes in the topological structures, node-level attributes, edge-level characteristics, (sub)graph-level features, and labels, parts or all of which are time-dependent. For instance, we try to answer the graph research questions like how

can we take structure evolution patterns to improve the temporal graph classification accuracy [10].

- Artificial dynamics within graphs pertain to researchers actively modifying existing or creating new graph-related elements, such as graph topology, node and graph attributes, connections within Graph Neural Network (GNN) layers, GNN gradients, etc., in order to fix initial imperfections and enhance performance. For instance, we try to answer how to mask node connections to explain the model decision in node and graph classification process [11, 12].

- Moreover, we extend the combination of natural dynamics and artificial dynamics, named **natural + artificial dynamics**, which describes that evolving graphs can also contain imperfect components, and we wish to study how to utilize natural dynamics and artificial dynamics together for the performance upgrade. For illustration, we try to explore how to identify redundant updates in time-evolving graphs and filter them for efficient node clustering [13, 14].

1.2 CONTRIBUTION AND IMPACT

My Ph.D. research mainly covers graph deep learning and network science. I also maintain a strong interest in graph theory and graph data management. In a broader context, my research seeks to explore both the natural and artificial dynamics of graphs with two primary objectives: (1) the discovery of new research topics motivated by real-world applications, and (2) the development of more accurate, efficient, and trustworthy graph AI systems. This pursuit is driven by the overarching goal of contributing to impactful applications. To date, my graph research work endeavors with natural and artificial dynamics have spanned diverse areas, including protein molecule classification and infectious pattern categorization [10, 15], demographic fair community discovery [14], misinformation detection and interpretation [11, 12], financial fraud detection [13], web learning to rank [16], green AI [17], climate forecasting and extreme weather detection [18], privacy-preserving [19, 20, 21, 22, 23, 24], etc.

My prior work has not only highlighted the theoretical depth of research questions arising from the natural and artificial dynamics of graphs, but has also underscored their practical significance through valuable contributions to diverse applications. These research efforts have been funded by multiple federal agencies and companies, including the National Science Foundation (NSF), the Department of Homeland Security (DHS), the Defense Advanced Research Projects Agency (DARPA), International Business Machines Corporation (IBM), and Meta Platforms, Inc.

My research has been recognized by top-tier AI/ML/DS conferences (e.g., ICLR, KDD, WWW, SIGIR, CIKM, etc.) and highly selective research awards (e.g., 2023 Rising Star in Data Science, 2022 C.W. Gear Outstanding Graduate Student, 2022 NeurIPS Top Reviewer, 2022 Rising Star in Data Mining and Management, etc.).

1.3 GENERAL NOTATION

If not specified in each individual work, we generally use lowercase letters (e.g., a) for scalars, bold lowercase letters for column vectors (e.g., $\mathbf{a}$), bold capital letters for matrices (e.g., $\mathbf{A}$), and we denote $\mathbf{A}(i,:)$ as the i-th row of $\mathbf{A}$. We use parenthesized superscript to denote the index (e.g., $\mathbf{A}^{(k)}$) over matrices, also we use unparenthesized superscript as the power operation (e.g., $\mathbf{A}^k$), $\top$ superscript as the transpose operation (e.g., $\mathbf{A}^\top$), and -1 superscript as the inverse operation (e.g., $\mathbf{A}^{-1}$).

CHAPTER 2: LITERATURE REVIEW AND ANALYSIS

2.1 OVERVIEW OF GRAPH AI RESEARCH

To pave the way for investigating the natural and artificial dynamics in graphs, we first introduce graph AI research topics (e.g., graph representations, graph neural networks, and graph AI applications) and their relationships. Then, in the next subsection, we can target each topic and see how natural dynamics and artificial dynamics contribute to them.

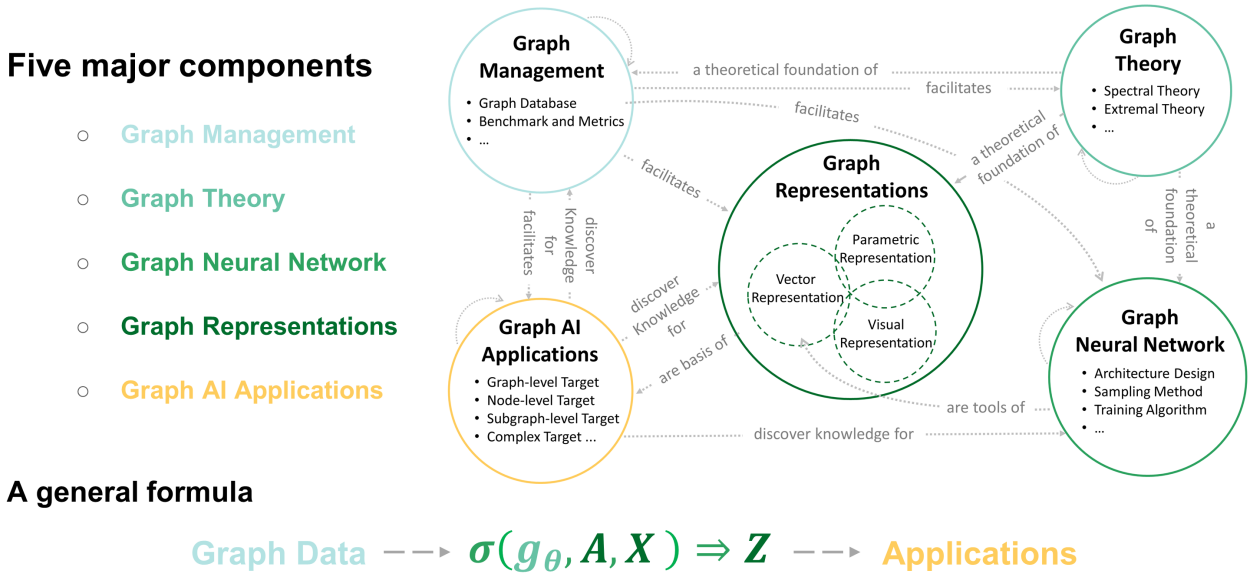


Figure 2.1: Graph AI Research Scope.

The relationships between graph data, graph theory, graph representations, graph neural networks, and graph AI applications can be summarized to some extent, as shown in Figure 2.1.

Although their relationships can be quite complex, we can generally figure out a common path for aligning these five components from the graph data to applications. Mathematically, as the general formula in Figure 2.1 suggested, given the graph structure data, we can first have the initial representation of graphs as adjacency matrix $\mathbf{A}$ and feature matrix $\mathbf{X}$. Then, graph theory will help to select appropriate parameters g_θ. Finally, the graph neural computation framework (like graph neural networks and graph transformers) will usually do non-linear complex neural computations $\sigma(\cdot)$ to obtain the informative hidden representation matrix $\mathbf{Z}$ for supporting many interesting applications.

First and foremost, graph representations are the basis of graph AI research, which projects graphs into a proper space such that graph mining can do various task-specific computa-

tions. To the best of our knowledge, graph representations consist of three components. First, graph embedding represents graphs with affinity matrices like Laplacian matrix and hidden feature representation matrix, on which many mining tasks rely, such as node classification [25]; Second, graph law represents graphs with several parameters which describe the statistical property of graphs such as node degree distribution and edge connection probability, which could help mining tasks like graph generation [26] and link prediction [27]; Third, graph visualization provides the visual representations and can serve for the domain-specific knowledge interpretation [28, 29].

Moreover, graph neural networks (including the recent state-of-the-art neural engines, e.g., graph transformers) are effective tools for extracting meaningful graph embedding vectors (or matrices) by combining deep learning theory and graph theory [30, 31]. Graph neural networks are composed of a family of many specific models with different research concerns like neural architecture [32], sampling strategy and message passing aggregation design [33].

Last but not least, in return, graph AI applications can use the extracted interesting (e.g., non-trivial, implicit, previously unknown, and potentially useful) knowledge from graph data to feedback the graph AI research. For example, we can start from classic application tasks by graph representation and graph neural network, such as node classification [25] aims to classify the node category based on its features, and node clustering [34, 35] is aiming to partition the entire graph into disjoint or overlapped clusters (i.e., subgraphs) based on researchers' objectives (e.g., conductance, betweenness, etc.). On the contrary, clustering can discover knowledge to help GNN implementations, and Cluster-GCN [36] is proposed to sample nodes in a topology-preserved clustering, which could entitle vanilla GCN [25] the fast computation to deal with large-scale graph datasets.

2.2 WHY WE NEED DIFFERENT DYNAMICS OF GRAPHS

In graph AI research development, researchers can encounter practical challenges. For example, during the graph representation learning process, some practical concerns are shown in Figure 2.2. For example, if the input graph structure changes, do we need to compute the Graph Signal from scratch even though the computational cost can be quite heavy? Can we learn some lessons from history to reuse some solutions and track and calibrate them? Also, for non-IID and non-grid graph data, can we use the simple neural computational framework to fit topological constraints? Moreover, if graph data has initial rough (e.g., missing features, limited labels) or even sensitive information (e.g., user identity) representations in the adjacency and attribute matrices, how can we efficiently obtain accurate, secure, and informative hidden representations?

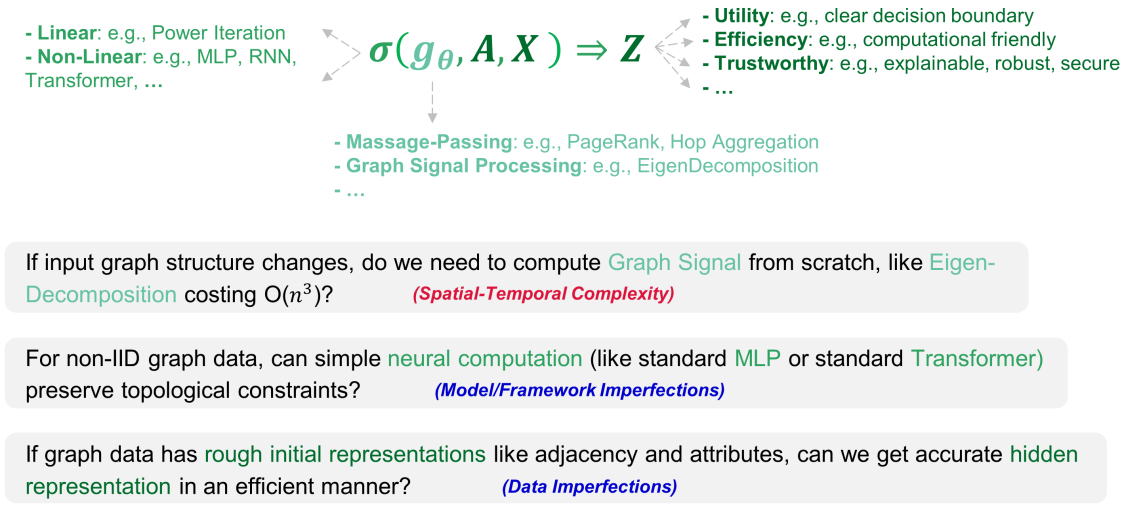


Figure 2.2: Some Practical Concern in Graph Representation Learning.

Those challenges during the graph AI development can be generalized into two categories, as shown in Figure 2.3, i.e., *graph spatial-temporal complexity* and *graph data and model imperfections*.

First, the inherent dynamics of graph structures and associated attributes necessitate a temporal perspective. Time-evolving topological structures and evolving node or graph attributes introduce intricate complications. Failing to appropriately incorporate temporal information results in a range of issues, including the disregard of temporal correlations between entities, oversight of causal effects, computational inefficiency, limitations in generalization performance, etc.

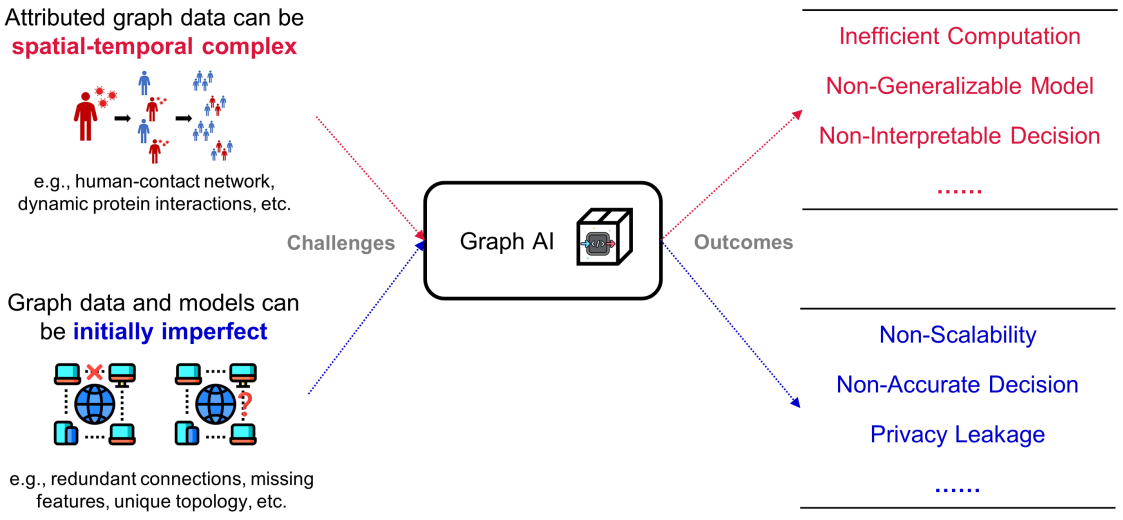


Figure 2.3: Two General Challenges in Graph AI Development.

Second, the initial states of topological structures and node-level/edge-level/(sub)graph-level attributes often exhibit imperfections. These imperfections may encompass connection errors, sampling noises, missing attributes, limited labeling, interpretation challenges, redundancy, and potential privacy leakage. The implications of these imperfections manifest in non-robustness, indiscriminate representations, restrictions on generalization performance, etc. A critical question revolves around whether researchers or practitioners can strategically address these imperfections by enhancing graph research components for better performance.

Motivated by the above two general challenges, in my Ph.D. studies, I proposed three directions of solutions as shown in Figure 2.4.

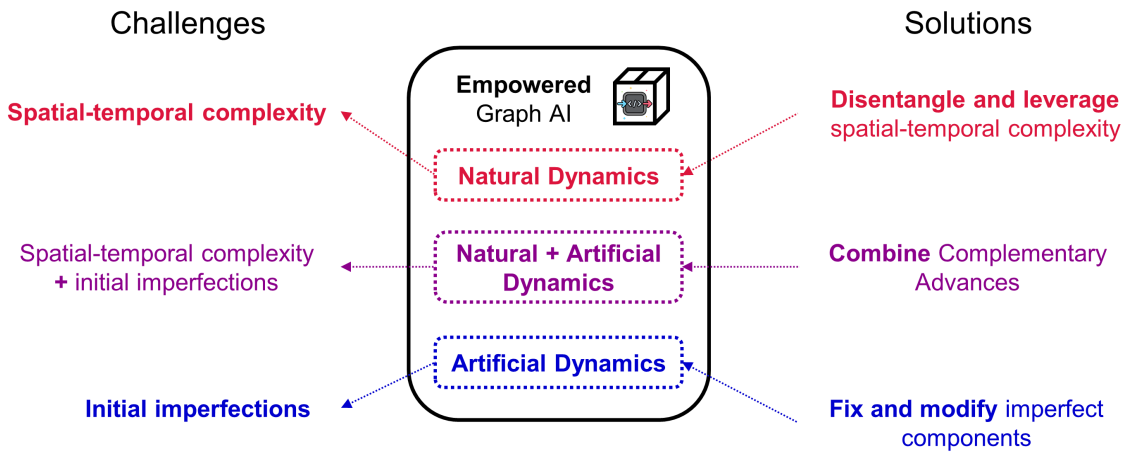


Figure 2.4: Three General Solutions in Graph AI Development.

The first direction is called natural dynamics, which aims to disentangle and leverage spatial-temporal complexity to make the graph data more applicable and empower graph AI development. The second direction is called artificial dynamics, which says that we, as researchers, actively make some changes to the initial graph data and models to fix and modify the imperfections to make them more applicable. And the middle one is called natural + artificial dynamics, which combines the complementary advances from natural dynamics and artificial dynamics, to deal with the scenario that evolving graphs and models have imperfect components.

2.2.1 Natural Dynamics of Graphs

Natural dynamics of graphs means that the input graph (to graph representation learning and graph mining) has the naturally evolving part(s), such as the evolving World Wide Web. Formally speaking, the naturally evolving part means that the topological structures or node

(edge, subgraph, or graph) features and labels depend on time. To be specific, the evolving graph structures can be represented either in

- Continuous time [37] (or Streaming [38]): an evolving graph can be modeled by an initial state G with a set of timestamped events O, and each event can be node/edge addition/deletion; or
- Discrete time [37] (or Snapshots [38]): an evolving graph can be modeled as a sequence of time-respecting snapshots $G^{(1)}, G^{(2)}, \ldots, G^{(T)}$, and each $G^{(t)}$ has its own node set $V^{(t)}$ and edge set $E^{(t)}$.

For these two modelings, the corresponding time-dependent features and labels can be represented in a time series or a sequence of matrices such as $\mathbf{X}^{(1)}, \mathbf{X}^{(2)}, \ldots, \mathbf{X}^{(T)}$. These two modeling methods have non-trivial complements. For example, continuous-time models rapid node/edge-level evolution, i.e., microscopic evolution [39], such as protein molecule interactions in a cell [40]; However, it could not represent the episodic and slowly-changing evolution patterns, which can be captured by discrete-time, i.e., macroscopic evolution [41], such like the periodical metabolic cycles in a cell [40]. Recently, different evolution patterns in a single graph are currently not jointly modeled for improving graph representation comprehensiveness, but some real-world evolving graphs naturally have both evolution patterns. For example, in [40], each dynamic protein-protein interaction network has 36 continuous observations (i.e., 36 edge timestamps), every 12 observations compose a metabolic cycle (i.e., 3 snapshots), and each cycle reflects 25 mins in the real world. Inspired by this observation, a nascent work [10] is recently proposed to jointly model different evolution patterns into the graph representation.

In brief, natural dynamics in graph AI research is a kind of study that models the time-dependent structure and feature information of graphs first and then leverages the modeling to meet real-world graph AI application needs in terms of effectiveness, efficiency, and trustworthiness.

2.2.2 Artificial Dynamics of Graphs

Compared with the natural dynamics concept, artificial dynamics is not fully discussed in the research community. To the best of our knowledge, the first relevant act of conceiving artificial dynamics in graphs appeared in [42], where “artificial jump" is proposed to adjust the graph topology for PageRank realizing the personal ranking function on structured data, i.e., a random surfer would follow an originally non-existing but newly-added highway to jump to a personally-selected node with a predefined teleportation probability.

In today's graph AI research, we extend the artificial dynamics concept such that researchers actively change the existing or construct the non-existing graph research components for performance upgrades in terms of effectiveness, efficiency, and trustworthiness. Also, the scope of artificial dynamics is also enlarged. Artificial dynamics can be added to the graph topology structure, graph representation matrix, graph neural network parameters, graph neural layer connections, and many more [14, 43, 44, 45, 46, 47, 48, 49, 50, 51].

More formally, artificial dynamics of graphs means that the graph research components (e.g., graph topology, graph stream, node/graph attributes/labels, GNNs gradients, and neural architectures, etc.) are deliberately re-designed by researchers to boost the task performance in certain metrics. For the re-designing, researchers can *change (e.g., filter, mask, drop, or augment) the existing elements* or *construct (i.e., from scratch) non-existing elements* to improve the performance (e.g., decision accuracy, model robustness and interpretation, etc.) than the original. To name a few, one example of artificial dynamics can be graph augmentation: DropEdge [52] is proposed to deal with the over-fitting of GNNs by randomly removing a certain amount of edges from the input graphs for each training epoch; DummyNode [53] is proposed to add a dummy node to the directed input graph, which connects all existing n nodes with $2n$ directed edges. The dummy node serves as a highway to extend the information aggregation in GNNs and contribute to capturing the global graph information, such that the graph classification accuracy by GNNs can be enhanced. In addition to the graph augmentation, other specific examples of artificial dynamics can be filtering unimportant coming sub-structures to save computations [13], adding residual connections among GNNs layers to address vanishing gradients [43], and perturbing the GNNs gradients for privacy protection [54].

2.2.3 Natural + Artificial Dynamics of Graphs

As mentioned above, on the one hand, considering the natural dynamics could leverage temporal dependency to contribute to graph research in terms of but not limited to, fast computation (e.g., tracking from the past instead of computing from scratch), causality reasoning (e.g., previous states cause the current state), comprehensive decision (e.g., prediction based on historical behaviors); On the other hand, studying artificial dynamics could help a wide range of targets, such as machine learning effectiveness (e.g., robustness, de-overfitting, de-oversmoothing). Investigating natural dynamics and investigating artificial dynamics not only have shared merits but also have exclusive advantages. Thus, a spontaneous research question is to ask whether natural dynamics can be integrated with artificial dynamics, which aims to keep the shared merits and bring exclusive advantages to synergy complementation.

Definitely, some pioneering works have been proposed to touch this area. To introduce them, throughout the paper, we use **natural + artificial dynamics** to denote the integrated investigation of natural dynamics and artificial dynamics in graph-related research and then present related works in this category.

In each component of Figure 2.1, I made my own contribution to empowering the graph AI development better with different kinds of dynamics studies as shown in Figure 2.5. For example, in graph management, I studied how to increase and decrease the graph data with utility, privacy, and efficiency preserved. In graph theory, I studied two important theories in graph research, PageRank and Eigendecomposition, to investigate high-order, dynamic, and fair graph clustering and partitioning. In the graph neural computational framework, I studied how to develop variant graph neural networks and graph transformers with different research application concentrations.

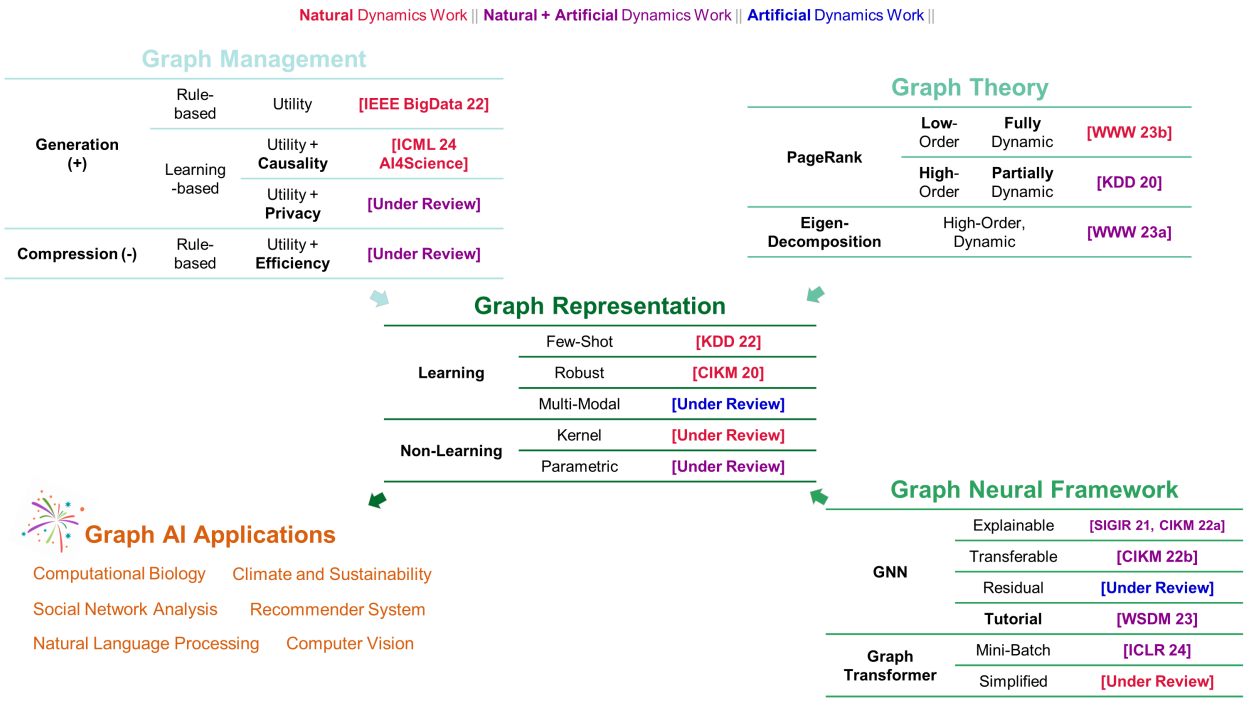


Figure 2.5: Thesis Work Overview.

After the literature review and analysis, we are ready to introduce completed works about natural, artificial, and natural + artificial dynamics research in different graph research topics.

2.3 ORGANIZATION OF THESIS

With the graph AI developed, the downstream applications can be at least categorized into three levels as shown in Figure 2.6. From the bottom to the top, there are node-level property identification, subgraph-level property identification, and graph-level property identification. Beyond that, after comprehending different levels of knowledge, we can step into the combined-level knowledge and intelligence discovery. For example, the graph generation task [55] needs comprehensive knowledge of different aspects, like the node behavior, the cluster closeness, the graph's overall property, etc.

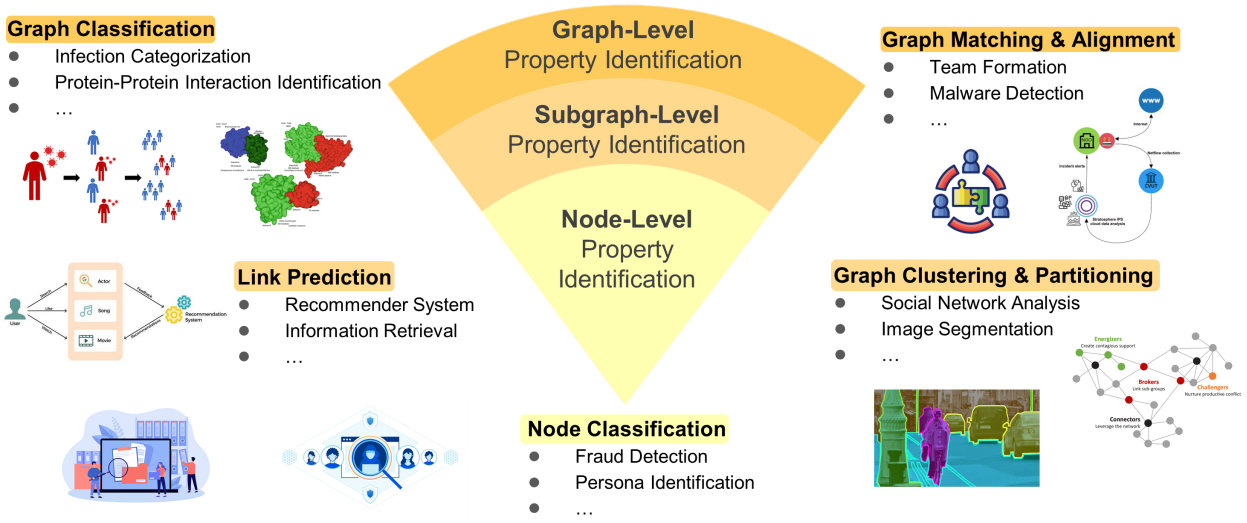


Figure 2.6: Graph AI Application Hierarchy.

Therefore, based on the node level, subgraph level, graph level, and combined level, the rest of the thesis is organized below, to introduce how the proposed natural and artificial dynamics empower the graph AI research and contribute to different requirements of application tasks.

In Chapter 2, we first review the related literature for graph AI research and various formats of graph dynamics, to pave the way for introducing our contribution to graph AI research through dynamics studies. To be specific, we disentangle graph AI applications into four different aspects, i.e., node-level, subgraph-level, graph-level, and combined-level; then, we can introduce how the study of the dynamics empowers graph AI research development and advance different applications in terms of effectiveness, efficiency, and trustworthy (e.g., explanation, robustness, privacy, security).

Hence, in Chapter 3, we start from node-level graph intelligence and introduce how artificial dynamics help node classification tasks enjoy state-of-the-art neural architecture in an effective and efficient manner, how artificial dynamics can make the graph learning model

more transparent and explainable, and how natural dynamics can make the graph learning model more comprehensive and robust.

In Chapter 4, we focus on subgraph-level graph intelligence and start to introduce how natural dynamics can be leveraged to achieve a simple graph learning model to outperform the powerful but heavy neural engines in link prediction tasks. Also, we then introduce how the natural and artificial dynamics can be combined and contribute to each other in addressing the graph local clustering and global partitioning with high-order effectiveness, computational and storage efficiency, and demographic fairness constraints.

Starting from Chapter 5, we step into graph-level graph intelligence and then introduce how natural dynamics can use temporal information to help increase graph classification accuracy and maintain efficiency. Moreover, we introduce how natural dynamics can be fully explored to determine similarity and match across different temporal heterogeneous graphs.

Given the investigation effort in the individual different levels of graph intelligence, in Chapter 6, we consider the more complex scenarios like graph discovery and graph generation because the corresponding tasks need to consider the node, edge, and subgraph behavior together to reach the destination of a complete graph. For example, we study how to discover meaningful graph structures from not-graph-structured data (e.g., time series) through natural dynamics and leverage the discovery to benefit corresponding applications (e.g., time series forecasting and time series anomaly detection). Also, we study how to combine natural and artificial dynamics to generate temporal heterogeneous graphs with the privacy guaranteed and the utility preserved in the generation.

Finally, in Chapter 7, we conclude the thesis by summarizing the completed work and envisioning future research directions.

CHAPTER 3: EMPOWERING NODE-LEVEL GRAPH INTELLIGENCE VIA NATURAL AND ARTIFICIAL DYNAMICS

In this chapter, based on node-level graph intelligence and knowledge discovery, we have three sections introducing how artificial dynamics help increase node classification efficiency and accuracy (Section 3.1), how artificial dynamics help obtain explainable node representation (Section 3.2), and how natural dynamics help achieve robust node representation (Section 3.3).

3.1 INCREASING NODE CLASSIFICATION EFFICIENCY AND ACCURACY VIA ARTIFICIAL DYNAMICS

3.1.1 Overview and Motivation

Transformer architectures have achieved outstanding performance in various computer vision and natural language processing tasks [56, 57]. Then, for non-grid graph data, developing effective graph transformers attracts much research attention, and some nascent works obtain remarkable performance in graph tasks for encoding complex topological and feature information into expressive node representations via attention mechanisms, like GT [58], Gophormer [59], Coarformer [60], Graphormer [61], SAN [62], ANS-GT [63], GraphGPS [64], NAGphormer [65], Exphormer [66] and etc.

Viewing each node as a token, most graph transformers need token-wise dense attention (or global attention) to extract expressive node embedding vectors [67], i.e., most graph transformers typically need to attend every pair of nodes in the input graph [58, 61, 62, 64, 68, 69, 70, 71, 72, 73]. As a consequence, dense attention induces unaffordable computational resource consumption for large-scale graph data due to this quadratic complexity, which hinders graph transformers from scaling up their effectiveness. In the meanwhile, mini-batch training provides a promising direction for graph transformers. Nonetheless, a few node (or token) samples in each batch could not fully enable the dense attention mechanism to capture enough information for each node's representation vector, given the complex topological and feature information of the whole large graph data. NAGphormer [65] is a very recent work targeting mini-batch training, which utilizes self-attention only on multiple hop aggregations for a node's embedding vector and gets good experimental results. On the other side, the hop aggregation may not deal with global, long-range interactions, and heterophilous information well; also it relies on the time-consuming eigendecomposition for the positional encoding.

To deal with this bottleneck, first, we assign each target node a token list that is sampled by personalized PageRank (PPR). The token list consists of PPR-ranked neighbors with weights and will be self-attended by the standard transformer [74] to output the target node representation vector. This graph tokenization operation, based on PPR, separates the complex topological data from the model training process, allowing independent offline graph feature engineering for each node, such that the mini-batch training manner for graph transformers becomes possible by loading a few node token lists in each batch and leveraging self-attention separately. We demonstrate the feasibility of utilizing this graph tokenization-based graph transformer as an effective graph convolutional neural network, leveraging a fixed polynomial filter as described in [25], and incorporating the concept of 'jumping knowledge' from [75]. However, only using personalized PageRank to sample nodes can copy limited topological information into each token list, which could not support different graph inductive biases for model training. To this end, we propose to rewire the input graph by establishing multiple virtual connections for personalized PageRank to transfer local, global, long-range, and heterophilous information of the entire graph into each token list. Then, self-attention on this comprehensive token list enables a good approximation of the whole graph data during each batch of training. Wrapping personalized PageRank based graph tokenization and virtual connection based graph rewiring, we obtain an effective mini-batch graph transformer, called **V**irtual **C**onnection **R**anking based **Graph** Trans**former** (VCR-Graphormer). Comparing the $O(n^3)$ complexity of NAGphormer [65] due to the eigendecomposition, the complexity of VCR-Graphormer is $O(m + klogk)$, where m is the number of edges in the graph and k can be roughly interpreted as the selected neighbors for each node that is much smaller than m or n.

3.1.2 Related Work

Graph data and representation learning techniques have a wide scope of applications like recommendation [76, 77, 78], query answering [16, 79, 80], misinformation detection [12], time series analysis [81], protein classification and generation [10, 82], etc. Recently, graph transformers emerged as one kind of effective graph representation learning model [67], which adapts the self-attention mechanism from transformers [74] to graph data representation learning. For example, GT [58] views each node in the input graph as a token and uses the self-attention mechanism to attend every pair of nodes to get the representation vectors. Then, various graph transformers are proposed following the dense attention mechanism with different positional encodings for node tokens [61, 62, 64, 68, 69, 70, 71, 72, 73], such that the standard transformer can learn topological information during the model training.

For example, in TokenGT [71], each node token is assigned augmented features from orthogonal random features and Laplacian eigenvectors, such that the edge existence can be identified by the dot product of two token vectors. Also, GraphTrans [69] connects a GNN module with the standard transformer and utilizes the GNN to provide topology-preserving features to the self-attention mechanism. To reduce the quadratic attention complexity of dense attention (or global attention), some recent works propose approximation methods [64, 66, 83]. In GraphGPS [64], the linear attention layer of Performer [84] is proposed to replace the standard transformer layer in graph transformers but the performance gain is marginal, where the authors hypothesized that the long-range interaction is generally important and cannot be easily captured in the linear attention mechanism. Exphormer [66] proposes using sparse attention in graph transformers, which intentionally selects several edges (e.g., by establishing expander graphs) to execute the node-wise attention. To address the quadratic complexity, GOAT [83] follows the dense attention but targets to reduce the feature dimension by finding a proper projection matrix, in order to save the efficiency. Among the above, if not all, most graph transformers are designed for graph-level tasks like graph classification and graph regression [85]. Targeting node-level tasks like node classification, some works are deliberately proposed [65, 86]. For example, NAGphormer [65] is proposed to assign each node a token list, and then the self-attention mechanism is only needed for this list to get the node embedding instead of handling all existing node pairs. This paradigm promises scalability for mini-batch training of graph transformers. Based on that, we first show why this paradigm can work well, propose the criteria of a good token list for carrying various graph inductive biases, and propose the efficient and effective graph transformer, VCR-Graphormer.

3.1.3 Preliminary

We first summarize some common notations here, and detailed notions will be explained within context. Given an undirected and attributed graph $G = (V, E)$ with $|V| = n$ nodes and $|E| = m$ edges, we denote the adjacency matrix as $\mathbf{A} \in \mathbb{R}^{n\times n}$, the degree matrix as $\mathbf{D} \in \mathbb{R}^{n\times n}$, the node feature matrix as $\mathbf{X} \in \mathbb{R}^{n\times d}$, and the node label matrix as $\mathbf{Y} \in \mathbb{R}^{n\times c}$. For indexing, we denote $\mathbf{X}(v,:)$ as the row vector and $\mathbf{X}(:,v)$ as the column vector, and we use parenthesized superscript for index $\mathbf{X}^{(i)}$ and unparenthesized superscript for power operation $\mathbf{X}^i$.

Self-Attention Mechanism According to [74], the standard self-attention mechanism is a deep learning module that can be expressed as follows.

$$\mathbf{H}' = \text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d'}})\mathbf{V} \in \mathbb{R}^{d'} \tag{3.1}$$

where queries $\mathbf{Q} = \mathbf{H}\mathbf{W}_Q$, keys $\mathbf{K} = \mathbf{H}\mathbf{W}_K$ and values $\mathbf{V} = \mathbf{H}\mathbf{W}_V$. $\mathbf{H} \in \mathbb{R}^{n\times d}$ is the input feature matrix. $\mathbf{W}_Q, \mathbf{W}_K, \mathbf{W}_V \in \mathbb{R}^{d\times d'}$ are three learnable weight matrices. $\mathbf{H}' \in \mathbb{R}^{n\times d'}$ is the output feature matrix.

Personalized PageRank Targeting a node of interest, the personalized PageRank vector stores the relative importance of the other existing nodes by exploring the graph structure through sufficient random walk steps. The typical personalized PageRank formula can be expressed as below.

$$\mathbf{r} = \alpha\mathbf{P}\mathbf{r} + (1-\alpha)\mathbf{q} \tag{3.2}$$

where $\mathbf{P} \in \mathbb{R}^{n\times n}$ is the transition matrix, which can be computed as $\mathbf{A}\mathbf{D}^{-1}$ (column normalized [87]) or $\mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}$ (symmetrically normalized [33]). $\mathbf{q}$ is a one-hot stochastic vector (or personalized vector), and the element corresponding to the target node equals 1 while others are 0. $\mathbf{r}$ is the stationary distribution of random walks, i.e., personalized PageRank vector, and α is the damping constant factor, which is usually set as 0.85 for personalized PageRank and 1 for PageRank. One typical way of solving a personalized PageRank vector is power iteration, i.e., multiplying the PageRank vector with the transition matrix by sufficient times $\mathbf{r}^{(t+1)} \longleftarrow \alpha\mathbf{P}\mathbf{r}^{(t)} + (1-\alpha)\mathbf{q}$ until convergence or reaching the error tolerance.

3.1.4 Effective Mini-Batch Training and VCR-Graphormer

In this section, we first propose a promising mini-batch way for graph transformer training and give theoretical analysis of its effectiveness. Then, we discuss its limitation and formalize an advanced version, **V**irtual **C**onnection **R**anking based **Graph** Trans**former** (VCR-Graphormer). Detailed proof of the proposed theory can be found in the appendix.

PPR Tokenization for Mini-Batch Training To project non-Euclidean topological information into expressive embedding vectors, most graph transformers need to use dense attention on every pair of nodes in the input graph. Set the $O(n^2)$ attention complexity aside, these graph transformers need to pay attention to every existing node that jeopardizes the mini-batch training for large-scale input graphs, because only a few samples in each batch

are hard to allow dense attention to capture enough topological and feature information.

To achieve mini-batch training, graph topological and feature information should be effectively wrapped into each individual batch, and then the graph learning model can start to learn on each batch. Thus, an inevitable way is to first disentangle the message passing along the topology before the model training and place the disentanglement properly in each batch.

Motivated by this goal, we propose a mini-batch training paradigm based on personalized PageRank (PPR) tokenization. The logic of this paradigm is easy to follow, which tokenizes the input graph by assigning each target node u with a token list $\mathcal{T}_u$. This list $\mathcal{T}_u$ consists of tokens (nodes) that are related to the target node u. Here, we use personalized PageRank (PPR) to generate the token list. Set node u as the target, based on Eq. 3.2, we can get the personalized PageRank vector, denoted as $\mathbf{r}_u$. Then, we sample the top k nodes in $\mathbf{r}_u$, denoted as $\mathcal{R}_u^k$. Basically, nodes in $\mathcal{R}_u^k$ together form the token list $\mathcal{T}_u$. Yet, the list $\mathcal{T}_u$ will feed to the transformer [74], then the positional encoding is necessary. Otherwise, two target nodes that share the same PPR-based neighbors but differ in ranking orders will result in the identical representation vector by self-attention. Actually, PPR ranking scores (i.e., the probability mass stored in the PPR vector $\mathbf{r}_u$) can provide good positional encoding information. Moreover, the PPR ranking scores are personalized, which means two token lists that share exactly the same entries can be discriminated by different PPR ranking scores from different target nodes. Mathematically, our general $\mathcal{T}_u$ is more expressed as aggregated form $\mathcal{T}_u^{Agg}$ and discrete form $\mathcal{T}_u^{Dis}$ as follows.

$$\mathcal{T}_u^{Agg} = \{(\mathbf{P}^l\mathbf{X})(u,:)\}, \text{ s.t. for } l \in \{1,\ldots,L\} \tag{3.3}$$

where $\mathbf{X} \in \mathbb{R}^{n\times d}$ is the feature matrix, and $\mathbf{P}^l\mathbf{X}$ is the l-th step random walk. At the l-th step, if we decompose the summarization, the discrete form is expressed as follows.

$$\mathcal{T}_u^{Dis} = \{\mathbf{X}(i,:)\cdot\mathbf{r}_u(i)\}, \text{ s.t. for } i \in \mathcal{R}_u^k \tag{3.4}$$

where $\mathbf{r}_u$ is the (personalized) PageRank vector at the l-th step starting from node u, and $\mathcal{R}_u^k$ is the set containing the top k entries from $\mathbf{r}_u$.

Based on Eq. 3.3 (or Eq. 3.4), we can reorganize the set $\mathcal{T}_u$ into a matrix $\mathbf{T}_u \in \mathbb{R}^{l\times d}$ in the aggregated form (or $\mathbf{T}_u \in \mathbb{R}^{lk\times d}$ in the discrete form) by stacking entries. Then, we can input $\mathbf{T}_u$ into the standard self-attention mechanism stated in Eq. 3.1 and a pooling function (e.g., mean or sum) to get the representation vector $\mathbf{h}_u$ for node u. This kind of graph tokenization with the standard self-attention mechanism is viable for approximating

a graph convolution neural network as stated in Theorem 3.1.

Theorem 3.1. Sampling token lists through (personalized) PageRank and using token-wise self-attention on each token list, this operation is equivalent to a fixed polynomial filter graph convolution operation [25] with jumping knowledge [75].

Proof. According to [25], given an undirected graph G with n nodes, the corresponding graph convolution operation on a signal matrix $\mathbf{X} \in \mathbb{R}^{n \times d}$ is defined as

$$\mathbf{U} g_\theta(\mathbf{\Lambda}) \mathbf{U}^\top \mathbf{X} \approx \mathbf{U} (\sum_{k=0}^{K} \theta_k \mathbf{\Lambda}^k) \mathbf{U}^\top \mathbf{X} \tag{3.5}$$

where $\mathbf{L} = \mathbf{I} - \mathbf{D}^{-\frac{1}{2}} \mathbf{A} \mathbf{D}^{-\frac{1}{2}} \in \mathbb{R}^{n \times n}$ is the normalized Laplacian matrix, $\mathbf{A} \in \mathbb{R}^{n \times n}$ is the adjacency matrix, and $\mathbf{D} \in \mathbb{R}^{n \times n}$ is the diagonal degree matrix. Then, $\mathbf{U} \in \mathbb{R}^{n \times n}$ is the matrix containing eigenvectors of Laplacian matrix $\mathbf{L}$ and $\mathbf{\Lambda} \in \mathbb{R}^{n \times n}$ is the diagonal matrix storing the corresponding eigenvalues.

Based on $(\mathbf{U} \mathbf{\Lambda} \mathbf{U}^\top)^k = \mathbf{U} \mathbf{\Lambda}^k \mathbf{U}^\top$, extending the above equation can get

$$\mathbf{U} g_\theta(\mathbf{\Lambda}) \mathbf{U}^\top \mathbf{X} \approx (\sum_{k=0}^{K} \theta_k \mathbf{L}^k) \mathbf{X} \tag{3.6}$$

Then, [25] approximately take $K = 1$, $\theta_0 = 2\theta$, and $\theta_1 = -\theta$, obtain

$$\mathbf{U} g_\theta(\mathbf{\Lambda}) \mathbf{U}^\top \mathbf{X} \approx \theta (\mathbf{I} + \mathbf{D}^{-\frac{1}{2}} \mathbf{A} \mathbf{D}^{-\frac{1}{2}}) \mathbf{X} \tag{3.7}$$

Finally, replacing $\mathbf{I} + \mathbf{D}^{-\frac{1}{2}} \mathbf{A} \mathbf{D}^{-\frac{1}{2}}$ above by $\mathbf{P} = (\mathbf{D} + \mathbf{I})^{-\frac{1}{2}} (\mathbf{A} + \mathbf{I}) (\mathbf{D} + \mathbf{I})^{-\frac{1}{2}}$, the final format of graph convolution operation is

$$\mathbf{H}^{(t+1)} = \sigma(\mathbf{P} \mathbf{H}^{(t)} \mathbf{\Theta}^{(t)}) \tag{3.8}$$

where t is the index of the graph convolution layer, $\mathbf{H}^0 = \mathbf{X}$, and σ is a non-linear activation function.

Removing the non-linear activation function at each intermediate layer [88], the final representation output $\mathbf{Z}$ of a T-layer graph convolution neural network can be rewritten as

$$\mathbf{Z} = \sigma(\mathbf{P}^T \mathbf{X} \mathbf{\Theta}^{(1)}, \ldots, \mathbf{\Theta}^{(T)}) = \sigma(\mathbf{P}^T \mathbf{X} \mathbf{\Theta}) \tag{3.9}$$

which shows the above-approximated equation is a GCN with a fixed polynomial filter, which is also the second component of our Eq. 3.11. The only difference is that, the second

component of Eq. 3.11 keeps multiple $t \in \{1, \ldots, T\}$, i.e., $[\mathbf{P}^1\mathbf{X}, \mathbf{P}^2\mathbf{X}, \ldots, \mathbf{P}^T\mathbf{X}]$, which are corresponding to Jumping Knowledge [75] to learn representations of different orders for different graph substructures through the attention pooling.

Moreover, for personalized PageRank, $\mathbf{P}^t\mathbf{X}$ can be replaced as $\alpha(\mathbf{I} - (1-\alpha)(\mathbf{A}+\mathbf{I}))^{-1}\mathbf{X}$, if t is sufficiently large for convergence; otherwise, it can be obtained iteratively by $\mathbf{H}^{(t+1)} = (1-\alpha)\mathbf{P}\mathbf{H}^{(t)} + \alpha\mathbf{X}$, where $\mathbf{H}^{(0)} = \mathbf{X}$, which is

$$(1-\alpha)^t\mathbf{P}^t\mathbf{X} + (1-\alpha)^{t-1}\mathbf{P}^{t-1}\mathbf{X} + \ldots + (1-\alpha)^1\mathbf{P}^1\mathbf{X} + \alpha\mathbf{X} \tag{3.10}$$

QED.

As stated in Theorem 3.1, the graph tokenization operation in Eq. 3.1 is viable for approximating GCN [25], but it is prone to have the latent incapabilities of GCN, such like shallow network or low-pass filter can only preserve local and homophilous information [89] but the deeper network can suffer from over-squashing problem [90]. Hence, the next direction is how to warp good input graph information into the token list to alleviate the aforementioned problems.

Principles of a "Good" Tokenization To solve the problems mentioned in the above subsection, in general, a good token list should contain enough information that can be generalizable to different graph inductive biases, whereas graph inductive bias refers to the assumptions or preconceptions that a machine learning model makes about the underlying distribution of graph data. Because we have compressed the entire graph topology and features (that are easily accessible for each node in the dense or global attention graph transformers [58, 61, 62, 64, 68, 69, 70, 71, 72, 73]) into the token list for the mini-batch training, we need to enable a good approximation of the whole graph data in each sampled token list. By attending to this good approximation, the self-attention mechanism can produce representative node embeddings. To define the goodness of the sampled token list, we propose at least four principles that need to be satisfied.

First, the token list of each node should reflect the local and global topological information of the input graph. For instance, starting from a target node, personalized PageRank can sample the top neighbor nodes, whose ranking can be regarded as local topology [35]. Also, if we eigendecompose the graph Laplacian matrix, the eigenvalues and eigenvectors can preserve the global topological information [91]. **Second**, the information in the token list should preserve long-range interactions of the input graph. The long-range interaction problem refers to the expressive node representation sometimes relying on a relatively long-distance message passing path in GNNs [92, 93, 94], and just stacking more layers in GNNs

Table 3.1: Four Metrics from Inductive Bias and Efficiency for Graph Tokenization Methods

A Good Graph Tokenization Could	Hop2Token [65]	Personalized PageRank	VCR-Graphormer
1. Reflect the input graph topology (local and global)	🙂	😐	🙂
2. Support long-range interaction	😐	☹️	🙂
3. Handle heterophily property	☹️	☹️	🙂
4. Be efficiently realized	😐	🙂	🙂

is found not solving the long-range interaction need but inducing over-squashing problem due to the compressed receptive field for each node [90, 92, 95]. **Third**, the information in the token list should have the ability to contain possible heterophilous information of the graph. The heterophily property of graphs can be interpreted as the close neighbors (e.g., connected nodes) not sharing similar content (e.g., node labels) [89, 96, 97, 97, 98], such that structure-based message passing in GNNs alone may not fully meet downstream tasks when the topological distribution and feature distribution do not conform. **Fourth**, the token list should be efficiently obtained. For example, although eigendecomposition preserves graph topology, it needs $O(n^3)$ computational complexity [99]. In general, the design of graph tokenization for graph transformers should avoid the problems discovered previously in GNNs. To be more specific, the list should be efficiently obtained but contain comprehensive information to provide sufficient graph inductive biases for model training.

The four metrics are listed in Table 3.1, where we then analyze if the related works satisfy the requirements. The nascent graph transformer NAGphormer [65] proposes the Hop2Token method, which first performs eigendecomposition of the Laplacian matrix of the input graph and attaches the eigenvector to each node input feature vector. Then, for a target node, the summarization of the 1-hop neighbors' feature vectors serves as the first entry in the target node's token list. Finally, NAGphormer uses self-attention on the k-length token list to output the representation vector for the target node. Due to the k-hop aggregation and eigendecomposition, Hop2Token could preserve the local and global topological information of the input graph. But for the long-range interaction awareness, hop-based aggregation may diminish node-level uniqueness, and computing the polynomial of the adjacency matrix is costly when k needs to be considerably large. Moreover, Hop2Token pays attention to structurally close neighbors and ignores possible heterophily properties of graphs. Also, additional eigendecomposition costs cubic complexity to obtain higher performance. Personalized PageRank (as discussed in Sec.3.1) samples local neighbors for a target node and overlooks global contexts, also it can not support long-range interaction when necessary. Although it ignores the heterophily, the personalized ranking list can be

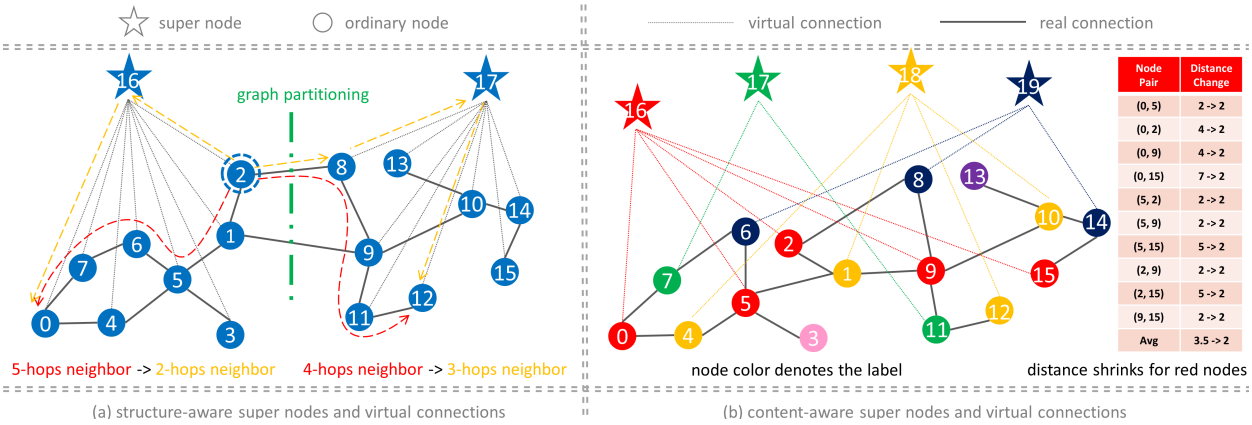


Figure 3.1: (a) Structure-Aware and (b) Content-Aware Virtual Connections on the Same Input Graph.

obtained in a constant time complexity [35]. Our proposed VCR-Graphormer can satisfy these four metrics in its sampled token list. Next, we introduce it with theoretical analysis.

VCR-Graphormer Here, we first introduce how the token list of each node in VCR-Graphormer gets formed, and then we give the theoretical analysis. In general, the token list of VCR-Graphormer consists of four components. For a target node u, the token list $\mathcal{T}_u$ is expressed as follows.

$$\mathcal{T}_u = \{\mathbf{X}(u,:)||1,\ (\mathbf{P}^l\mathbf{X})(u,:)||\frac{L-l+1}{\sum_{l=1}^{L} l},\ \mathbf{X}(i,:)||\bar{\mathbf{r}}_u(i),\ \mathbf{X}(j,:)||\hat{\mathbf{r}}_u(j)\}, \\ \text{s.t.}\quad \text{for}\quad l \in \{1,\ldots,L\}, i \in \bar{\mathcal{R}}_u^{\bar{k}}, j \in \hat{\mathcal{R}}_u^{\hat{k}} \tag{3.11}$$

where these four components have different views of the input graph, and each component has an initial scalar weight, i.e., 1. Also, $|\mathcal{T}_u| = 1 + L + \bar{k} + \hat{k}$. We then introduce details of the four components. $\{\}$ operator means a list. The concatenation operator applies to both scalars and vectors, i.e., the scalar will be converted to a single-element vector before concatenation.

The **first component**, $\mathbf{X}(u,:)||1 \in \mathbb{R}^{d+1}$ is straightforward, which is the input feature of the target node u associated with the weight 1. The **second component** is designed to capture the local topological information around the target node u, based on the aggregated form of PPR sampling as expressed in Eq 3.3. Matrix $\mathbf{P}^l\mathbf{X}$ is the l-th step of random walks, since one more random walk step accesses 1-hop away neighbors [33], those neighbors' features are aggregated based on random walk probability distribution to encode the local topological information for node u. Moreover, the given weight 1 for $\mathbf{P}^l\mathbf{X}$ is distributed

among walk steps by $\frac{L-l+1}{\sum_{l=1}^{L} l}$, which means the closer neighbors' aggregation occupies more weights.

So far, the first two components are proposed to meet the half part of the first metric in Table 3.1, i.e., the local graph topology gets preserved in the token list. After that, the third component is responsible for recording long-range interactions and partial global information (i.e., from the node connections perspective), and the fourth component is designed to record the heterophily information and partial global information (i.e., from the node content perspective).

The **third component** is motivated by graph rewiring, but our goal is to let the token list preserve structure-based global information and extend the narrow receptive field of over-squashing [92] in an efficient manner. To be specific, we break the original structure by inserting several virtual **super nodes** into the graph adjacency matrix. To differentiate from the inserted super nodes, the originally existing nodes are named **ordinary nodes**. To do this, the input graph G is first partitioned into $\bar{s}$ clusters (e.g., using METIS partitioning algorithm [100]). For each cluster, we assign a super node that connects every member in this cluster, where the new edges are called structure-aware virtual connections, as shown in Figure 3.1(a). After that, the original adjacency matrix is transferred from $\mathbf{A} \in \mathbb{R}^{n \times n}$ to $\bar{\mathbf{A}} \in \mathbb{R}^{(n+\bar{s}) \times (n+\bar{s})}$. Note that, in $\bar{\mathbf{A}}$, the inter-cluster edges are retained. Correspondingly, the transition matrix for PPR is denoted as $\bar{\mathbf{P}}$. Then, based on $\bar{\mathbf{P}}$, we can get the personalized PageRank vector $\bar{\mathbf{r}}_u$ through Eq. 3.2. Again, we sample top $\bar{k}$ items in $\bar{\mathbf{r}}_u$ to form the set $\bar{\mathcal{R}}_u^{\bar{k}}$. Finally, we can obtain the complete vector of the third component, i.e., $\mathbf{X}(i,:)||\bar{\mathbf{r}}_u(i) \in \mathbb{R}^{d+1}$, for each node $i \in \bar{\mathcal{R}}_u^{\bar{k}}$. Compared with the second component, we adapt the discrete form as stated in Eq. 3.4 for the third component for the following two reasons. (1) We assume the long-range interaction between nodes is represented in the form of paths other than broadcasts. Then, still concatenating all nodes from far-away hops can induce noise, but personalized PageRank is more suitable for sampling paths with corresponding weights. (2) Multiplying matrices $\mathbf{PX}$ as the second component will at least cost $O(m)$ time complexity, which is linear to the number of edges. On the contrary, personalized PageRank can be completed in a constant time, which is independent of the input graph size [35]. The above two reasons also apply to the design of the following fourth component.

The **fourth component** is designed to capture the content-based global information and encode the heterophily information into the token list. The content here refers to the labels of nodes or pseudo-labels like feature clustering and prototypes [101] when labels are not available. The realization of the fourth component is quite similar to the third component. The difference is that a super node here is assigned to one kind of label in the input graphs, and then it connects every node that shares this label. Those new edges are called content-

aware virtual connections, as shown in Figure 3.1(b). Note that, when the input training set has labels, the number of content-aware super nodes $\hat{s}$ equals the number of node labels c, otherwise it is a hyperparameter, e.g., feature-based clusters in K-means. Also, adding content-aware super nodes is based on the original adjacency matrix, i.e., $\mathbf{A} \in \mathbb{R}^{n \times n}$ to $\hat{\mathbf{A}} \in \mathbb{R}^{(n+\hat{s}) \times (n+\hat{s})}$. Then, given $\hat{\mathbf{P}}$, the procedure of getting $\hat{\mathbf{r}}_u$, $\hat{\mathcal{R}}_u^{\bar{k}}$, and $\mathbf{X}(j,:)||\hat{\mathbf{r}}_u(j) \in \mathbb{R}^{d+1}$ for $j \in \hat{\mathcal{R}}_u^{\hat{k}}$, is identical to the third component.

After stacking elements in the set $\mathcal{T}_u$, we get the matrix $\mathbf{T}_u \in \mathbb{R}^{(1+L+\bar{k}+\hat{k}) \times (d+1)}$ as the input of the standard transformer, i.e., for node u, the initial input $\mathbf{Z}_u^{(0)} = \mathbf{T}_u$. Then, for the t-th layer,

$$\mathbf{Z}_u^{(t)} = \mathrm{FFN}(\mathrm{LN}(\tilde{\mathbf{Z}}_u^{(t)})) + \tilde{\mathbf{Z}}_u^{(t)}, \quad \tilde{\mathbf{Z}}_u^{(t)} = \mathrm{MHA}(\mathrm{LN}(\mathbf{Z}_u^{(t-1)})) + \mathbf{Z}_u^{(t-1)} \tag{3.12}$$

where LN stands for layer norm, FFN is feedforward neural network, and MHA denotes multi-head self-attention. To get the representation vector for node u, a readout function (e.g., mean, sum, attention) will be applied on $\mathbf{Z}_u^{(t)}$. We give the analysis of our proposed techniques as follows.

Remark 3.1. After inserting (structure- or content-aware) super nodes, towards a target ordinary node, the remaining ordinary nodes can be classified into two categories. The first category consists of the nodes whose shortest path to the target node gets decreased. The second category consists of the nodes whose shortest path to the target node remains. We name the first category as **brightened ordinary nodes** and the second category as **faded ordinary nodes**. Faded ordinary nodes denote the local neighbors of the target node, because they are originally two-hop accessible nodes. Brightened ordinary nodes consist of the remaining ordinary nodes in the graph.

Theorem 3.2. After inserting (structure- or content-aware) super nodes and changing the original transition matrix $\mathbf{P}$ into new $\tilde{\mathbf{P}}$, the probability mass of personalized (e.g., starting from node u) PageRank transforming from faded ordinary nodes to brightened ordinary nodes is the summation of all negative entries in the vector $\mathbf{c} = \alpha \sum_i^{\infty} \alpha^i \tilde{\mathbf{P}}^i (\tilde{\mathbf{P}} - \mathbf{P})\mathbf{r}$, where $\mathbf{r}$ is the PageRank vector on the original graph $\mathbf{P}$. Correspondingly, all the positive entries are received mass for brightened ordinary nodes. Vector $\mathbf{c}$ will converge to a constant vector and the sum of vector $\mathbf{c}$ is 0.

The general insights from Theorem 3.2 show the probability mass transfer from faded ordinary nodes to brightened ordinary nodes, which further illustrates that previous hardly-reachable nodes are obtaining more weight to extend the narrow receptive field.

Proof. We first introduce a solution for exactly solving personalized PageRank vector, called cumulative power iteration [102], which is expressed as follows.

$$\mathbf{r} = \sum_{i=0}^{\infty} \mathbf{c}^{(i)},\ \mathbf{c}^{(i)} = (\alpha \mathbf{P})^i (1-\alpha)\mathbf{q} \tag{3.13}$$

where $\mathbf{P} = \mathbf{A}\mathbf{D}^{-1}$ is the column normalized transition matrix, $\mathbf{q}$ is the one-hot stochastic vector, and $\mathbf{r}$ is the personalized PageRank vector. The exactness proof of the above solution can be referred to [102].

Now, suppose we have an original graph whose transition matrix is denoted as $\mathbf{P}$. After inserting some super nodes, we have the new graph whose transition matrix is denoted as $\tilde{\mathbf{P}}$. For dimension consistency, we view the added super nodes as dangling nodes in the original graph, which has no connecting edges.

For the original transition matrix $\mathbf{P}$, we can have the personalized PageRank vector denoted as $\mathbf{r}$. Also, for the new transition matrix $\tilde{\mathbf{P}}$, we can have the personalized PageRank vector denoted as $\tilde{\mathbf{r}}$. Next, given our vector $\mathbf{c} = \alpha \sum_i^{\infty} \alpha^i \tilde{\mathbf{P}}^i (\tilde{\mathbf{P}} - \mathbf{P})\mathbf{r}$, we need to prove that $\tilde{\mathbf{r}} = \mathbf{c} + \mathbf{r}$. Since vectors $\tilde{\mathbf{r}}$ and $\mathbf{r}$ are non-negative personalized PageRank vector, i.e., $\|\tilde{\mathbf{r}}\|_1 = \|\mathbf{r}\|_1 = 1$, then the sum of vector $\mathbf{c}$ is 0.

Motivated by Theorem 3.2 of [87], we next expand vector $\mathbf{c} = \alpha \sum_i^{\infty} \alpha^i \tilde{\mathbf{P}}^i (\tilde{\mathbf{P}} - \mathbf{P})\mathbf{r}$:

$$\begin{aligned}
\mathbf{c} =& \sum_{i}^{\infty} \alpha^i \tilde{\mathbf{P}}^i \alpha (\tilde{\mathbf{P}} - \mathbf{P})\mathbf{r} \\
=& \sum_{i=0}^{\infty} \left(\left((\alpha\tilde{\mathbf{P}})^{i+1} \sum_{j=0}^{\infty} (\alpha\mathbf{P})^j (1-\alpha)\mathbf{q} \right) - \left((\alpha\tilde{\mathbf{P}})^i \sum_{j=0}^{\infty} (\alpha\mathbf{P})^{j+1} (1-\alpha)\mathbf{q} \right) \right) \\
=& \sum_{i=0}^{\infty} \left(\left((\alpha\tilde{\mathbf{P}})^{i+1} \sum_{j=0}^{\infty} (\alpha\mathbf{P})^j (1-\alpha)\mathbf{q} \right) - \left((\alpha\tilde{\mathbf{P}})^i \sum_{j=0}^{\infty} (\alpha\mathbf{P})^j (1-\alpha)\mathbf{q} \right) + \left((\alpha\tilde{\mathbf{P}})^i (1-\alpha)\mathbf{q} \right) \right) \\
=& \sum_{i=0}^{\infty} (\alpha\tilde{\mathbf{P}})^{i+1} \sum_{j=0}^{\infty} (\alpha\mathbf{P})^j (1-\alpha)\mathbf{q} - \sum_{i=0}^{\infty} (\alpha\tilde{\mathbf{P}})^i \sum_{j=0}^{\infty} (\alpha\mathbf{P})^j (1-\alpha)\mathbf{q} + \sum_{i=0}^{\infty} (\alpha\tilde{\mathbf{P}})^i (1-\alpha)\mathbf{q} \\
=& \sum_{i=0}^{\infty} (\alpha\tilde{\mathbf{P}})^i \sum_{j=0}^{\infty} (\alpha\mathbf{P})^j (1-\alpha)\mathbf{q} - \sum_{j=0}^{\infty} (\alpha\mathbf{P})^j (1-\alpha)\mathbf{q} - \sum_{i=0}^{\infty} (\alpha\tilde{\mathbf{P}})^i \sum_{j=0}^{\infty} (\alpha\mathbf{P})^j (1-\alpha)\mathbf{q} + \sum_{i=0}^{\infty} (\alpha\tilde{\mathbf{P}})^i (1-\alpha)\mathbf{q} \\
=& -\sum_{j=0}^{\infty} (\alpha\mathbf{P})^j (1-\alpha)\mathbf{q} + \sum_{i=0}^{\infty} (\alpha\tilde{\mathbf{P}})^i (1-\alpha)\mathbf{q}
\end{aligned} \tag{3.14}$$

Then, based on Eq. 3.13, it can be observed that $\mathbf{c} + \mathbf{r} = \tilde{\mathbf{r}}$. Moreover, since $\|\mathbf{P}\| = \|\tilde{\mathbf{P}}\| = 1$, then $\mathbf{c}$ converge to a constant. QED.

Theorem 3.3. The efficiency of getting the token list in VCR-Graphormer is $O(m + klogk)$, where m is the number of edges in the input graph G, and $k = max(\text{NNZ}(\bar{\mathbf{r}}_u), \text{NNZ}(\hat{\mathbf{r}}_u))$, where $\text{NNZ}(\cdot)$ denotes the number of non-zero entries in a vector. The complexity of self-

attention in VCR-Graphormer is $O((1+L+\bar{k}+\hat{k})^2)$ based on Eq. 3.11.

The general insights from Theorem 3.3 state the computational complexity of the proposed graph transformer, which is less than the quadratic attention complexity of most graph transformers.

Proof. For getting the token list as stated in Eq. 3.11, computing the second component i.e., **PX** costs $O(m)$, where m is the number of edges in the input graph. Then, for computing the third and the fourth components, the analysis is similar. Taking the third component computation as an example, we have below analysis. First, using Metis [100] to partition the input graph into arbitrary clusters costs $O(m)$, suppose the number of nodes n is smaller than the number of edges m, and the number of partitions s is way smaller than the number of edges m [1].

Second, approximately solving the personalized PageRank with Push Operations costs constant time complexity (Lemma 2 in [35] and Proposition 2 in [103]), which constant is only related to error tolerance ϵ and dumping factor α but not the size of the input graph. We denote the constant as c in the following. Since Push Operation [35] is a local operation, most entries in the PageRank vector $\mathbf{r}_u$ are zero (or near zero and can be omitted). Denote the number of non-zero entries in $\mathbf{r}_u$ as k, then sorting them costs $O(klogk)$. For each node $u \in \{1,\ldots,n\}$, we totally need $O(n \cdot c \cdot klogk)$. Since each node's personalized PageRank computation is independent, we can parallelize the computation [11]. Denote we have w parallel workers, the final complexity is $O(\frac{n}{w} \cdot c \cdot klogk)$. When number of workers is sufficient, we can have $O(klogk)$. QED.

3.1.5 Experimental Evaluation

Datasets and Setup First, as shown in Table 3.2, 9 graph datasets are common benchmarks for node classification accuracy performance, 6 of them are relatively small, and 3 of them are large-scale datasets. Small datasets like PubMed, CoraFull, Computer, Photo, CS, and Physics can be accessed from the DGL library [2]. Reddit, Aminer, and Amazon2M are from [104] can be accessed by this link [3]. Moreover, we also include the large-scale heterophious graph dataset, arXiv-Year, from the benchmark [105].

Baseline Methods In homophilous experiments, our baseline algorithms consist of 4 categories. For GNNs, we have GCN [25], GAT [106], APPNP [33], and GPR-GNN [107].

[1] `http://glaros.dtc.umn.edu/gkhome/node/419`

[2] `https://docs.dgl.ai/api/python/dgl.data.html`

[3] `https://github.com/THUDM/GRAND-plus`

Table 3.2: Graph Datasets and Statistics

	PubMed	CoraFull	Computer	Photo	CS	Physics	Reddit	Aminer	Amazon2M
Nodes	19,717	19,793	13,752	7,650	18,333	34,493	232,965	593,486	2,449,029
Edges	88,651	126,842	491,722	238,163	163,788	495,924	11,606,919	6,217,004	61,859,140
Features	500	8,710	767	745	6,805	8,415	602	100	100
Labels	3	70	10	8	15	15	41	18	47

For scalable GNNs, we include GraphSAINT [108], PPRGo [109], and GRAND+ [104]. For graph transformers, we have GT [58], Graphormer [61], SAN [62], and GraphGPS [64]. As for scalable graph transformers, we have NAGPhormer [65] discussed above, and Exphormer [66] is a new scalable graph transformer, which also needs relative dense attention.

In heterophilous experiments, we first follow the corresponding heterogeneous baselines from the benchmark [105], and the performance is based on the given split of the benchmark. In addition to the baselines provided by the benchmark like MixHop [110], GCNII [32], GPR-GNN [107], we include other heterophilous graph deep learning methods WRGAT [111], GGCN [112], ACM-GCN [113], GloGNN [114] and GloGNN++ [114].

Node Classification on Homophilous and Heterophilous Graph Data The node classification accuracy on small-scale graph datasets and large-scale datasets among various baseline algorithms is reported in Table 3.3 and Table 3.4 and 3.5 respectively. Containing more structure-aware neighbors (i.e., larger $\bar{k}$) and more content-aware neighbors (i.e., larger $\hat{k}$) in the token list has the potential to increase the node classification performance. For attention efficiency, we constrain $\bar{k}$ and $\hat{k}$ to be less than 20 in VCR-Graphormer, even for large-scale datasets.

Table 3.3: Node Classification Accuracy on Small-Scale Datasets

		PubMed	CoraFull	Computer	Photo	CS	Physics
GNN	GCN	86.54 ± 0.12	61.76 ± 0.14	89.65 ± 0.52	92.70 ± 0.20	92.92 ± 0.12	96.18 ± 0.07
	GAT	86.32 ± 0.16	64.47 ± 0.18	90.78 ± 0.13	93.87 ± 0.11	93.61 ± 0.14	96.17 ± 0.08
	APPNP	88.43 ± 0.15	65.16 ± 0.28	90.18 ± 0.17	94.32 ± 0.14	94.49 ± 0.07	96.54 ± 0.07
	GPR-GNN	89.34 ± 0.25	67.12 ± 0.31	89.32 ± 0.29	94.49 ± 0.14	95.13 ± 0.09	96.85 ± 0.08
Scalable GNN	GraphSAINT	88.96 ± 0.16	67.85 ± 0.21	90.22 ± 0.15	91.72 ± 0.13	94.41 ± 0.09	96.43 ± 0.05
	PPRGo	87.38 ± 0.11	63.54 ± 0.25	88.69 ± 0.21	93.61 ± 0.12	92.52 ± 0.15	95.51 ± 0.08
	GRAND+	88.64 ± 0.09	71.37 ± 0.11	88.74 ± 0.11	94.75 ± 0.12	93.92 ± 0.08	96.47 ± 0.04
Graph Transformer	GT	88.79 ± 0.12	61.05 ± 0.38	91.18 ± 0.17	94.74 ± 0.13	94.64 ± 0.13	97.05 ± 0.05
	Graphormer	OOM	OOM	OOM	92.74 ± 0.14	94.64 ± 0.13	OOM
	SAN	88.22 ± 0.15	59.01 ± 0.34	89.93 ± 0.16	94.86 ± 0.10	94.51 ± 0.15	OOM
	GraphGPS	88.94 ± 0.16	55.76 ± 0.23	OOM	95.06 ± 0.13	93.93 ± 0.15	OOM
Scalable Graph Transformer	NAGphormer	89.70 ± 0.19	71.51 ± 0.13	91.22 ± 0.14	95.49 ± 0.11	95.75 ± 0.09	**97.34 ± 0.03**
	Exphormer	89.52 ± 0.54	69.09 ± 0.72	91.59 ± 0.31	95.27 ± 0.42	**95.77 ± 0.15**	97.16 ± 0.13
	VCR-Graphormer	**89.77 ± 0.15**	**71.67 ± 0.10**	**91.75 ± 0.15**	**95.53 ± 0.14**	95.37 ± 0.04	**97.34 ± 0.04**

From Table 3.3, our VCR-Graphormer shows the competitive performance among all baseline methods. Moreover, the VCR-Graphormer does not need the time-consuming eigende-

composition, and the corresponding structure-aware neighbors and content-aware neighbors via virtual connections can help preserve the heterophilous and long-range information.

For Table 3.4 and 3.5, VCR-Graphormer also achieves very competitive performance with limited token length, which also demonstrates that several sampled global neighbors may not bridge the effectiveness of eigendecomposition.

Table 3.4: Node Classification Accuracy on Large-Scale Heterophilous Graph Dataset (Without Minibatching)

	arXiv-year
Edge Homophily	0.22
#Nodes	169,343
#Edges	1,166,243
#Features	128
#Classes	5
MLP	36.70 ± 0.21
L Prop 1-hop	43.42 ± 0.17
L Prop 2-hop	46.07 ± 0.15
LINK	53.97 ± 0.18
SGC 1-hop	32.83 ± 0.13
SGC 2-hop	32.27 ± 0.06
C&S 1-hop	44.51 ± 0.16
C&S 2-hop	49.78 ± 0.26
GCN	46.02 ± 0.26
GAT	46.05 ± 0.51
GCNJK	46.28 ± 0.29
GATJK	45.80 ± 0.72
APPNP	38.15 ± 0.26
H_2GCN	49.09 ± 0.10
MixHop	51.81 ± 0.17
GPR-GNN	45.07 ± 0.21
GCNII	47.21 ± 0.28
WRGAT	OOM
GGCN	OOM
ACM-GCN	47.37 ± 0.59
LINKX (Minibatch)	53.74 ± 0.27
GloGNN	54.68 ± 0.34
GloGNN++	54.79 ± 0.25
VCR-Graphormer	54.15 ± 0.09

Table 3.5: Node Classification Accuracy on Large-Scale Heterophilous Graph Dataset (With Minibatching)

	arXiv-year
Edge Homophily	0.22
#Nodes	169,343
#Edges	1,166,243
#Features	128
#Classes	5
MLP Minibatch	36.89 ± 0.11
LINK Minibatch	53.76 ± 0.28
GCNJK-Cluster	44.05 ± 0.11
GCNJK-SAINT-Node	44.30 ± 0.22
GCNJK-SAINT-RW	47.40 ± 0.17
MixHop-Cluster	48.41 ± 0.31
MixHop-SAINT-Node	44.84 ± 0.18
MixHop-SAINT-RW	50.55 ± 0.20
LINKX Minibatch	53.74 ± 0.27
VCR-Graphormer	54.15 ± 0.09

3.1.6 Summary

Graph transformer has been proven as an effective graph learning method for its adoption of attention mechanism that is capable of capturing expressive representations from complex topological and feature information of graphs. Graph transformer conventionally performs dense attention (or global attention) for every pair of nodes to learn node representation vectors, resulting in quadratic computational costs that are unaffordable for large-scale graph data. Therefore, mini-batch training for graph transformers is a promising direction, but limited samples in each mini-batch can not support effective dense attention to encode informative representations. Facing this bottleneck, (1) we start by assigning each node a token list that is sampled by personalized PageRank (PPR) and then apply standard multi-head self-attention only on this list to compute its node representations. This PPR tokenization method decouples model training from complex graph topological information and makes heavy feature engineering offline and independent, such that mini-batch training of graph transformers is possible by loading each node's token list in batches. We further prove this PPR tokenization is viable as a graph convolution network with a fixed polynomial filter and jumping knowledge. However, only using personalized PageRank may limit information carried by a token list, which could not support different graph inductive biases for model training. To this end, (2) we rewire graphs by introducing multiple types of virtual connections through structure- and content-based super nodes that enable PPR tokenization to encode local and global contexts, long-range interaction, and heterophilous information into each node's token list, and then formalize our **V**irtual **C**onnection **R**anking based **Graph** Trans**former** (VCR-Graphormer). Overall, VCR-Graphormer needs $O(m+klogk)$ complexity for graph tokenization as compared to $O(n^3)$ of previous works.

3.2 OBTAINING EXPLAINABLE NODE REPRESENTATION VIA ARTIFICIAL DYNAMICS

3.2.1 Overview and Motivation

Following the success of graph neural networks (GNNs), many efforts concentrate on scaling GNNs in the information propagation process, which aims to reduce the computational complexity of the training stage (i.e., to reduce the training time, the number of parameters, etc.) but maintain the performance in the mean time [33, 88, 109, 115, 116]. For example, in [115], the authors integrate the importance sampling scheme into the message-passing mechanism to sample subgraphs in each layer for scaling the structures of GNNs. Inspired

by [75], the authors in [33] directly replace the message-passing mechanism with the random walk stationary distribution among nodes to avoid stacking layers in GNNs. However, when applying GNNs to the real-world complex settings, in addition to scaling their structures, another critical aspect has been largely overlooked, i.e., evolving the structures of GNNs. Compared with static neural network models with fixed structures and parameters, dynamic models have the potential of simultaneously enjoying compatibility and interpretability due to the adaptation of their structures or parameters to different inputs [117]. Therefore, we make the first attempt to simplify and dynamize the structures of GNNs for scalability and interpretability, which is different from the graph representation learning models learning the evolution pattern [118] or persistent pattern [119] of dynamic graphs.

To this end, we propose a simplified and dynamic graph neural network model, called SDG. In the proposed SDG, we design the dynamic propagation scheme based on the personalized PageRank tracking process without the explicit message-passing to transfer multi-hop neighborhood information. The key idea behind this modification is that the influence of other nodes on a selected node in the graph through a k-layer GNN model is proportional to the k-step random walk distribution starting from that selected seed node [33, 75]. The advantage of the dynamic propagation scheme is that when the structure of the input graph changes, the stationary distribution of random walks can be tracked in a fast and accurate manner instead of resolving from scratch. Hence, we can leverage this property to (1) not only achieve efficient fine-tuning for node-level and graph-level classification on changed graphs, (2) but also investigate the influence of a certain node on the node-level and graph-level label prediction results by masking that node in the proposed dynamic propagation scheme.

3.2.2 Problem Definition

Here, we describe the problem we want to solve in dynamics graph neural network models.

Problem 3.1. Explainable Classification based on Dynamic Graph Neural Network Models

Input: an input graph with node-level (subgraph-level, or graph-level) labels

Output: a dynamic graph neural network model $\mathbb{M}$, which could (i) get fast and accurate classification results when the graph changes and (ii) explain the influence that the change made on the classification decision

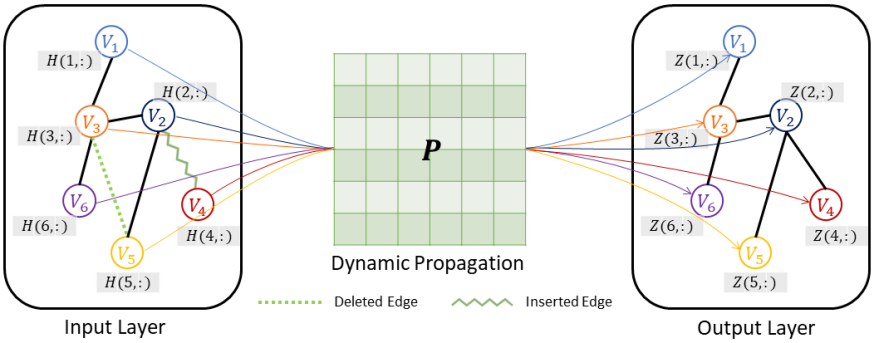


Figure 3.2: Framework of SDG Graph Neural Network.

3.2.3 Proposed Method

The proposed SDG graph neural network model has an input layer and an output layer as shown in Figure 3.2. To dynamically use the stationary distribution to propagate the neighborhood information among nodes instead of stacking layers, we propose the dynamic propagation scheme in our SDG graph neural network, which is expressed as follows.

$$\mathbf{Z} = \text{softmax}(\mathbf{PH}) \tag{3.15}$$

where $\mathbf{P} \in \mathbb{R}^{n \times n}$ is the dynamic propagation matrix, $\mathbf{H} \in \mathbb{R}^{n \times c}$ is the hidden node feature matrix extracted from the input node features $\mathbf{X} \in \mathbb{R}^{n \times d}$ through the model-agnostic neural network f_θ, i.e., $\mathbf{H} = f_\theta(\mathbf{X})$, and $\mathbf{Z} \in \mathbb{R}^{n \times c}$ denotes the predicted label matrix. Finally, the loss function of SDG is defined as follows.

$$\mathcal{L}_{SDG} = -\sum_{i}^{n}\sum_{j}^{c} Y(i,j) \ln Z(i,j) \tag{3.16}$$

where $\mathbf{Y} \in \mathbb{R}^{n \times c}$ and $\mathbf{Z} \in \mathbb{R}^{n \times c}$ are the ground truth and model output respectively.

Next, we mainly introduce how to form the dynamic propagation matrix $\mathbf{P}$ for the changed graph structure.

Dynamic Propagation Scheme The proposed dynamic propagation scheme is realized by tracking the dynamic propagation matrix $\mathbf{P}$, where each row $P(i,:)$ is occupied by the stationary distribution of random walks starting from that node V_i. To be specific, $P(i,:)$ can be fast tracked instead of solving from scratch when the graph topology changes from $\mathbf{A}$ to $\mathbf{A}'$. And the computation of each row is independent, which implies that the tracking process from $\mathbf{P}$ to $\mathbf{P}'$ can be parallelized. We first introduce the tracking process of $\mathbf{P}'$, and then analyze its time complexity and error bound.

As each row $P(i,:)$ encodes the stationary distribution of random walks starting from node V_i, it can be expressed as follows.

$$P(i,:)^\top = \alpha \mathbf{M} P(i,:)^\top + (1-\alpha)\mathbf{r} \tag{3.17}$$

where $\mathbf{M} = \mathbf{A}\mathbf{D}^{-1} \in \mathbb{R}^{n\times n}$ denotes the column-stochastic transition matrix, $\alpha \in [0,1]$ denotes the teleportation probability, and $\mathbf{r} \in \mathbb{R}^n$ denotes the personalized vector with $r(i) = 1$ and other entries equal to 0. For the clear notation, we omit the transpose notation $\top$ for $P(i,:)$ in the following part.

When the graph topology changes from $\mathbf{M}$ to $\mathbf{M}'$ with the updated edge set ΔE, the stationary distribution needs to be updated. To obtain $\mathbf{P}'$, we need to update each row of $\mathbf{P}$. The core idea is to push out the previous probability distribution score from the changed part to the residual part of the graph [13, 87], and then add the pushed out distribution back to the previous distribution in order to finally obtain the new distribution. The tracking process can be described as follows.

$$P(i,:)_{pushout} = \alpha(\mathbf{M}' - \mathbf{M})P(i,:) \tag{3.18}$$

and

$$P(i,:)' = P(i,:) + \sum_{k=0}^{\infty} (\alpha \mathbf{M}')^k \; P(i,:)_{pushout} \tag{3.19}$$

where $P(i,:)_{pushout}$ denotes the distribution score that needs to be pushed out on the residual graph due to the updated edges ΔE, and $P(i,:)'$ denotes the tracked new distribution.

This push out process can be proved to converge to the exact stationary distribution of the new graph through sufficient cumulative power iterations [87, 102], i.e., $k \to \infty$. With this push out process, we can update each row of dynamic propagation matrix $\mathbf{P}$ to further track $\mathbf{P}'$. Next, we analyze the time complexity and accuracy of approximately tracking the whole dynamic propagation matrix $\mathbf{P}'$ with a threshold ϵ as follows.

Theorem 3.4. Given the updated matrix $\mathbf{M}'$, the threshold ϵ, and q distributed machines, each machine at most costs $O(\frac{mn}{q}\log_\alpha(\frac{\epsilon}{2}))$ time to obtain the tracked dynamic propagation matrix $\mathbf{P}'$ with each row bounded by $\frac{\epsilon}{1-\alpha}$ error compared with the stationary distribution in terms of ℓ_1-norm, where m is the number of non-zero entries of $\mathbf{M}'$, and n is the number of nodes in the graph.

Proof. With the threshold ϵ, if $\|(\alpha\mathbf{M}')^k \; P(i,:)_{pushout}\|_1 \le \epsilon$ then we stop the iteration of Eq. 3.19. Since $\mathbf{M}'$ is a column-stochastic matrix, then $\|\mathbf{M}'\|_1 = 1$. Thus, the number of iterations k satisfies $k \le \log_\alpha(\frac{\epsilon}{\|P(i,:)_{pushout}\|_1})$; and $P(i,:)_{pushout} = \alpha(\mathbf{M}'-\mathbf{M})P(i,:)$, then $\|P(i,:$

$)_{pushout}\|_1 \leq 2\alpha$. The computation of each row is independent, such that each distributed machine at most costs $O(\frac{n}{q} m \log_\alpha(\frac{\epsilon}{2}))$ complexity. Also, denoting the early stop at the s-th iteration, then the remaining tracking error is denoted as $\| \sum_{k=s}^{\infty} (\alpha \mathbf{M}')^k \ P(i,:)_{pushout} \|_1$, which can be proved less than $\frac{\epsilon}{1-\alpha}$ just by replacing $P(i,:)_{pushout}$ with $\alpha(\mathbf{M}' - \mathbf{M})P(i,:)$ and setting $k = log_\alpha(\frac{\epsilon}{2\alpha})$. QED.

3.2.4 Case Study Evaluation

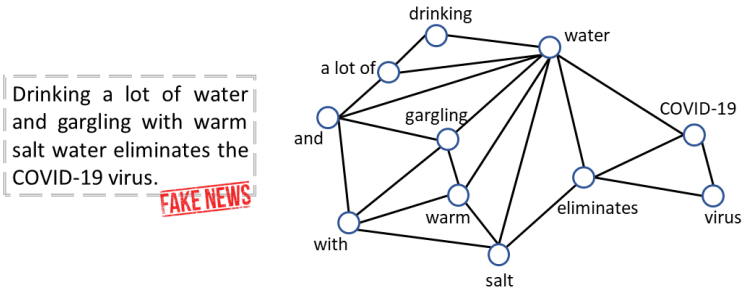


Figure 3.3: A Fake News Article and its Word Graph.

Comprehensive and Explainable Disinformation Detection Disinformation dissemination has a huge negative impact on our society, and many factors may hinder the identification of disinformation. For example, messages on social networks are usually short and fast propagated, which requires that the predictive model should have an efficient training process and adaptable structures for the evolving environment. Hence, we design the fake news case study with the proposed SDG model. Our objective is news article word graph classification. To build a word graph (i.e., Figure 3.3), a word stands for a node, an edge is established if two words co-occur in a window of q text units [120]. Then, each node has a word embedding vector from the pre-trained Bert model [121], and the word graph feature aggregates all node embeddings through an average pooling layer. Also, each word graph is associated with a label indicating fake news or not.

Two main challenges hinder disinformation-related research. Firstly, a characteristic of the spread and creation of disinformation is the co-existence of multiple types of heterogeneous features, which introduces ambiguous factors that could potentially camouflage disinformation from real information, i.e., even a single word can have different semantic meanings in different contexts [122, 123]. For example,“Apple" in the food corpus means a kind of fruit, while it stands for the company in the high-tech corpus. Secondly, interpretability is another key to understanding the logic of prediction systems to further build human-machine trust in results and improve model accuracy [124, 125, 126]. While pioneering state-of-the-art

algorithms have been proposed to detect disinformation, many of them are black-box in nature and lack interpretable mechanisms for explaining why information has been flagged as false. Take fake news as an example, not every sentence in fake news is false. It is critical to specify guidelines for improving the existing model in an explainable way. For instance, to distinguish the importance of different sentences in determining the detection decision [124]. As such, there is a critical need for new technology that can support disinformation detection both accurately and efficiently (at an early stage before widespread propagation), and in a way that is understandable by both domain experts and the general public.

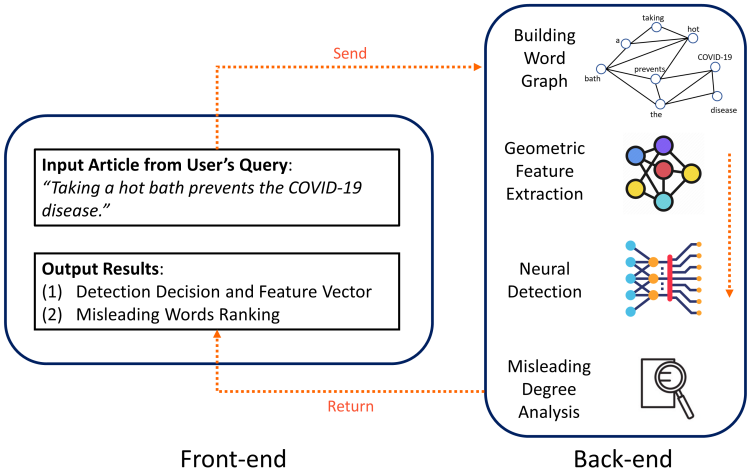


Figure 3.4: System Architecture of DISCO.

To this end, we propose a comprehensive computational framework for disinformation based on the SDG graph neural network, named DISCO. It models the heterogeneity of disinformation with graph machine learning techniques, such that the heterogeneity can be positively leveraged to solve challenges for detection and explanation. To be specific, *for the heterogeneity*, DISCO leverages large-scale pre-trained (from general corpus) language models in a transfer learning manner, which means involving a simple projection head could make DISCO achieve high detection accuracy in specific domains like politics news; *for the explanation*, DISCO involves dynamics in SDG, such that the word and sentence importance towards the detection variance can be figured out by graph augmentations (i.e., masking nodes and edges in article graphs). We also develop an online demo for visualizing the detection and the explanation result of DISCO, and we design the real-world experiment with 48,000+ fake and real news articles, and DISCO could outperform than baseline algorithms.

The online demo of DISCO has two main parts, front-end, and back-end, as shown in Figure 3.4. The front-end (1) accepts and passes the user open query (i.e., a suspect piece of information) to the back-end, and (2) receives and shows the detection decision and

misleading word rankings from the back-end. The back-end is supported by the graph machine learning techniques, which are responsible for identifying the input information and making the corresponding explanation by (1) building the word graph for the input article (2) extracting the geometric feature of that graph; (3) predicting whether the input article is real or fake; and (4) ranking each word in that text based on the misleading degree. Then the back-end returns all results to the front-end for users.

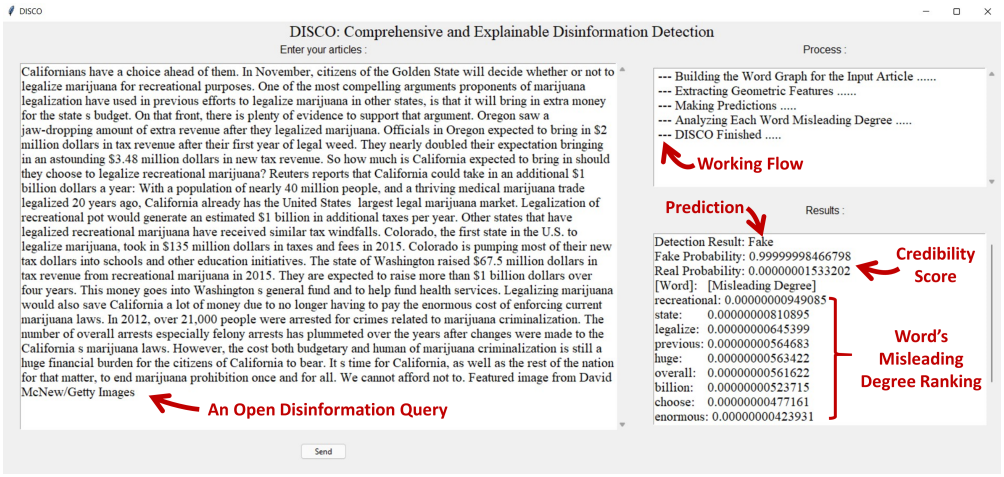
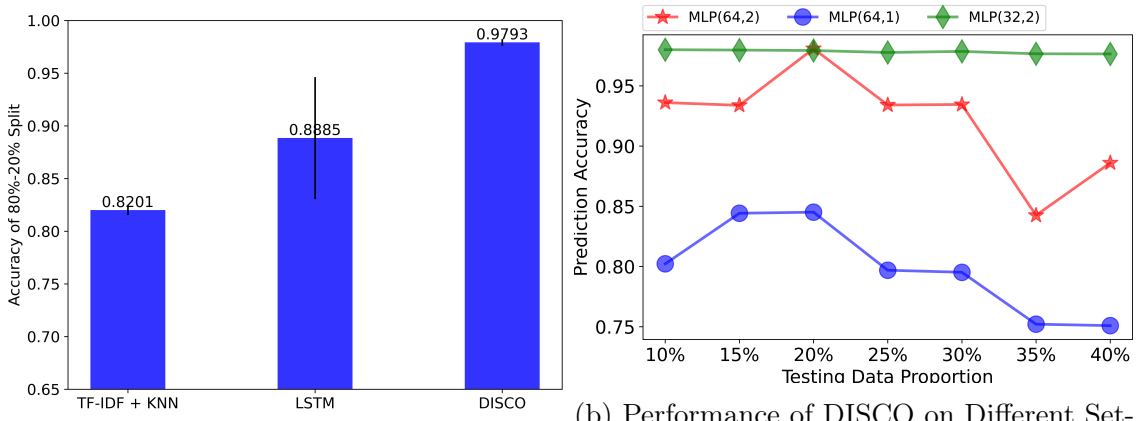


Figure 3.5: User Interface of DISCO. The left-hand side is the text area to receive open queries from users, the upper right-hand side is the real-time process monitor of DISCO, and the lower right-hand side is the prediction and explanation result.

We choose the fake real news data set from [127, 128]. The fake news articles consist of 23,481 items, ranging from Mar 31, 2015 to Feb 19, 2018. As for real news articles, 21,417 items are involved from Jan 13, 2016 to Dec 31, 2017. We report the fake news detection accuracy in Figure 3.6a. For each bar in Figure 3.6a. Also, as shown in Figure 3.6b, we can observe that (1) even the sample neural detection module can achieve competitive performance under the DISCO modeling, for MLP (hidden dim $= 32$, # layers $= 2$) achieving the most accurate and stable predictions.

Summary Graph Neural Networks (GNNs) have achieved state-of-the-art performance in many high-impact applications such as fraud detection, information retrieval, and recommender systems due to their powerful representation learning capabilities. Some nascent efforts have been concentrated on simplifying the structures of GNN models, in order to reduce the computational complexity. However, the dynamic nature of these applications requires GNN structures to evolve over time, which has been largely overlooked so far. To bridge this gap, we propose a simplified and dynamic graph neural network model, called SDG. It is efficient, effective, and provides interpretable predictions. In particular, in SDG,



(a) Detection Accuracy

(b) Performance of DISCO on Different Settings.

Figure 3.6: Accuracy and Robustness of DISCO Model.

we replace the traditional message-passing mechanism of GNNs with the designed dynamic propagation scheme based on the personalized PageRank tracking process. We conduct extensive experiments and ablation studies to demonstrate the effectiveness and efficiency of our proposed SDG. We also design a case study on fake news detection to show the interpretability of SDG.

3.3 ACHIEVING ROBUST NODE REPRESENTATION VIA NATURAL DYNAMICS

3.3.1 Overview and Motivation

Graph representation learning algorithms provide node representations for many graph mining tasks such as node classification, link prediction, clustering, and anomaly detection. Classical graph embedding algorithms such as DeepWalk [129], LINE [130] and node2vec [131] capture topology information like local neighborhood connectivity pattern, structural role, and other high-order proximities to represent each node. Moreover, in order to obtain robust representations, many graph embedding algorithms have been proposed by leveraging the principle of the generative adversarial net (GAN) [132], such as ANE [133] and GraphGAN [134]. To this end, the generative model tries to fit the underlying connectivity distribution of the network and then produces fake samples (i.e., fake nodes, fake relations or fake representations) to fool the discriminative model, while the discriminative model tries to distinguish the generated fake samples from the ground truth samples.

Traditional graph embedding algorithms like [129, 130, 131] and state-of-the-art GAN-based graph embedding algorithms like [133, 134] only focus on the single network scheme.

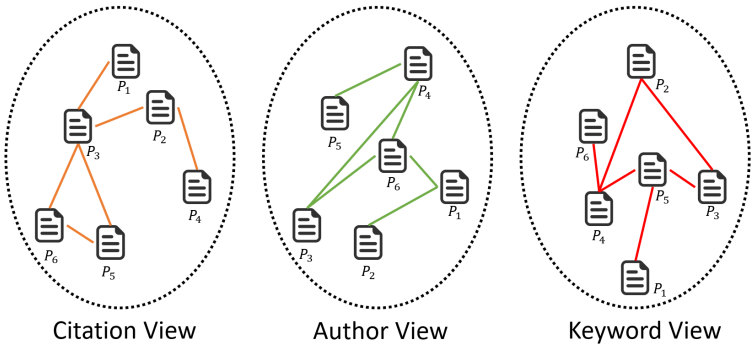


Figure 3.7: An illustrative multi-view network example of 3 views with 6 papers. In the citation view, an edge connects two papers if one paper cites the other. In the author view, an edge exists if two papers share at least one common author. In the keyword view, an edge exists if two papers share at least one common keyword.

In the multi-view scenario (e.g., Figure 3.7), the same set of nodes can have different connectivity patterns in different views. Take the scientific papers for example, in the view of citations, two papers are connected with an edge if there is a reference record. However, in the common-author view, these two papers are not connected if they do not share at least one common author. Also, in the time-evolving graph, each graph snapshot can be regarded as a view. Jointly embedding the structural information from multiple views can lead to a comprehensive node representation and remove noise and bias from a single view [135, 136]. Recently, with the advances of graph neural networks, many multi-view graph embedding frameworks [137, 138] have been proposed to obtain the attributed node representation.

Most, if not all, existing multi-view graph embedding algorithms follow the nature of the view-collaboration mechanism [135] to integrate specific node representations from each specific view, in order to generate the comprehensive node representations. We propose a view-adversarial framework for multi-view graph embedding named VANE, which consists of two adversarial games [119]. The first adversarial game ensures the comprehensiveness of node representations: the feature extractor aims to extract comprehensive node representations containing view-invariant structural information to fool the view discriminator, while the view discriminator aims to distinguish which node representation comes from which view by discriminating view-dependent structural information. The second adversarial game ensures the robustness of node representations: the generator tries to fit the distribution of the extracted node representations to generate fake node representations, and the node representation discriminator tries to discriminate fake representations from real representations; in the meanwhile, the feature extractor tries to provide robust node representations which are hard to fit.

3.3.2 Problem Definition

A view is defined as a single type of edge. A multi-view graph with k views is defined as $\mathcal{G} = (\mathcal{V}, \mathcal{E}_1, \ldots, \mathcal{E}_k)$, where $\mathcal{V}$ is the set of nodes, and $\mathcal{E}_i$ is the set of edges in the i-th view. Thus, we formally define the *Multi-View Graph Embedding* as follows.

Problem 3.2. Multi-View Graph Embedding

Input: the multi-view graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}_1, \ldots, \mathcal{E}_k)$.

Output: the robust node representations $\{\mathbf{x}_v\}_{v \in \mathcal{V}} \in \mathbb{R}^d$ with $d \ll |\mathcal{V}|$, which are consistent across k different views.

3.3.3 Proposed Method

In this section, we introduce the proposed VANE model from the overview to the details of two adversarial games.

The proposed VANE framework is shown in Figure 3.8. It consists of two adversarial games. To obtain comprehensive node representations, the first adversarial game involves the feature extractor F and the view discriminator D_S. The feature extractor F tries to extract the view-invariant node representation across different views, while the view discriminator D_S tries to discriminate the view-dependent structural information (i.e., subgraph representation) from node representations. To obtain robust node representations, the second adversarial game involves the node embedding model F_N, the node representation generator G, and the node representation discriminator D_N. Challenged by fake node representations from the generator G, the node embedding model F_N tries to provide robust node representations for the node representation discriminator D_N.

The First Adversarial Game The first adversarial game consists of the feature extractor F and the view discriminator D_S. In the feature extractor F, the node embedding model F_N is instantiated by the regular embedding layer which is a fully-connected layer for one-hot node vectors. The subgraph embedding model F_S is instantiated by an LSTM [139] layer to aggregate a sequence of node representation vectors into a subgraph representation vector. The view discriminator D_S is a multi-class classifier and instantiated by a multilayer perceptron to take the subgraph representation vector as the feature and the index of view where the subgraph comes from as the label.

We first preprocess all input views into node sequences by random walks where the node sequence S is denoted as $S = \{v_1, v_2, \ldots, \}$. Then we input node sequences into the feature

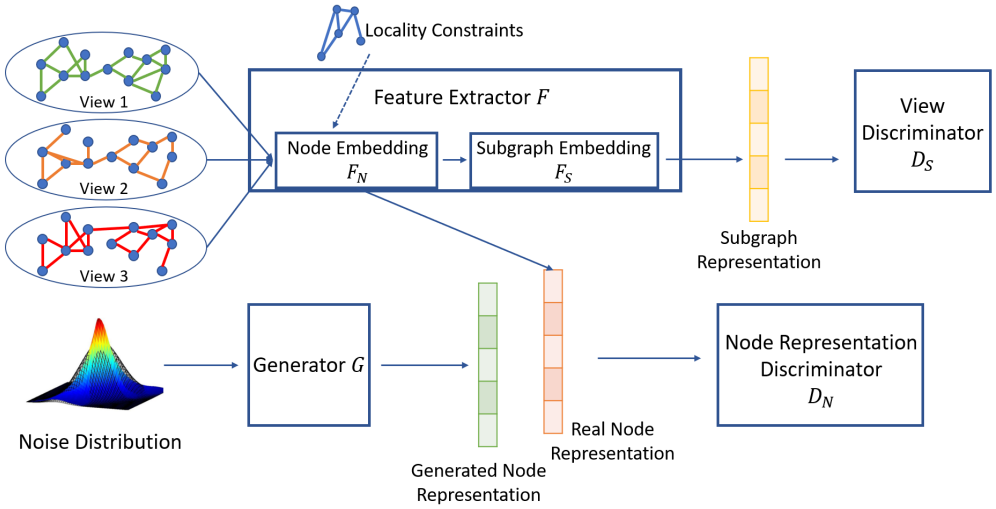


Figure 3.8: The framework of VANE.

extractor F individually. Constructed by F_N and F_S, the feature extractor F is a shared neural network by all node sequences. In F_N, the one-hot vector of each node v_i in the sequence S is represented by the node embedding vector $\mathbf{x}_{v_i}$. After that, the subgraph embedding model F_S aggregates the sequence of node embedding vectors into a subgraph embedding vector denoted as $F(S)$. During the first adversarial game, F tries to extract indistinguishable (i.e., view-invariant) subgraph embedding vectors to fool the view discriminator D_S, while D_S tends to distinguish the view-specific information from the subgraph representations to discriminate the source view.

Suppose we have k views in the given multi-view network. For the i-th view discrimination $(1 \leq i \leq k)$, the distribution of subgraph representation $F(S)$ from the i-th view denotes as $F(S) \sim p_i(F(S))$, the distribution of node sequence $F(S)$ from views other than the i-th view defines as $\bar{p}_i(F(S)) = \frac{1}{k-1}\sum_{m \neq i} p_m(F(S))$, and $D_S(F(S))$ represents the probability that $F(S)$ came from the i-th view rather than other views. Thus, the objective J_S for the i-th view subgraph representation and discrimination states as follows.

$$\begin{aligned} &\min_F \max_{D_S} J_S(D_S, F) \\ =&\min_F \max_{D_S} \mathbb{E}_{F(S)\sim p_i(F(S))}[\log(D_S(F(S)))] \\ &+ \mathbb{E}_{F(S)\sim \bar{p}_i(F(S))}[\log(1 - D_S(F(S)))] \end{aligned} \tag{3.20}$$

where the feature extractor F and the view discriminator D_S aim to play the first adversarial game converging to the equilibrium as stated in [140], $p_1(F(S)) = p_2(F(S)) = \cdots = p_k(F(S))$, which suggests that the probability distribution of the subgraph representation

$F(S)$ from each view equals to each other.

To further improve the ability of our model to capture the local topology information of the node sequence from each specific view, we add the cosine similarity based locality constraints on the node embedding model F_N with minimizing the following objective J_{LC}.

$$\min_{F_N} J_{LC}(F_N(v_i), F_N(v_j)) = \min_{F_N} \mathbb{E}_{v_i,v_j \in S}[1 - \cos(F_N(v_i), F_N(v_j))] \tag{3.21}$$

where node v_i and v_j come from the same node sequence S.

The Second Adversarial Game To enhance the robustness of extracted node representations from the first adversarial game, the second adversarial minimax game involves three players: the node representation generator G, the node embedding model F_N, and the node representation discriminator D_N. The generator G and node representation discriminator D_N are instantiated by multilayer perceptron classifiers.

During the second game, the generator G generates fake node representations to fit the distribution of node representations produced by F_N, and F_N tries to provide robust node representations that are hard to fit to help the discriminator D_N tell the fake node embedding vectors. The objective function J_N of the second adversarial game is described as follows.

$$\begin{aligned} &\min_{G} \max_{D_N} \max_{F_N} J_N(D_N, G, F_N) \\ =&\min_{G} \max_{D_N} \max_{F_N} \mathbb{E}_{F_N(v) \sim p_{data}(F_N(v))}[\log(D_N(F_N(v)))] \\ &+ \mathbb{E}_{\mathbf{z} \sim p_{\mathbf{z}}(\mathbf{z})}[\log(1 - D_N(G(\mathbf{z})))] \end{aligned} \tag{3.22}$$

where $F_N(v)$ is the real node representation, and $G(\mathbf{z})$ is the generated fake node representation from the noise vector $\mathbf{z}$.

Optimization We summarize the training procedure of the VANE framework in Alg. 3.1, where two adversarial games are executed alternatively.

In Alg. 3.1, we first randomly initialize the parameter θ of each component of the VANE framework. In Step 3 and Step 4, we prepare the node sequence along with the ground truth view label and generate the fake node representation. Steps 5-7 execute the first adversarial game for comprehensive node representations by updating the feature extractor F and the view discriminator D_S. Steps 8-10 execute the second adversarial game for robust node representations, where the generator G and the discriminator D_N compete with each other to force the node representation model F_N to provide robust (i.e. hard to fit) node representations. In practice, the training procedure for each data sample (S, y) is

Algorithm 3.1: Stochastic Training Procedure for VANE

Require:

multi-view network $\mathcal{G} = (\mathcal{V}, \mathcal{E}_1, \ldots, \mathcal{E}_k)$, r node sequences sampled from each view, max iteration T, noise vector $\mathbf{z}$.

Ensure:

embedding vector $\mathbf{x}_v$ for each node $v \in \mathcal{V}$.

for t in T iterations **do**

  for node sequence S in $k \times r$ shuffled sequence samples **do**

    Form data samples (S, y), y is the true label indicating the source view where S comes from.

    Use generator G to generate a fake node embedding vector $G(\mathbf{z})$ from the noise vector $\mathbf{z}$.

    Update the node embedding model F_N by descending its gradient w.r.t. Eq. (3.21) with S.

    Update the feature extractor F by descending its gradient w.r.t. Eq. (3.20) with S and y by fixing θ_{D_S}.

    Update the view discriminator D_S by ascending its gradient w.r.t. Eq. (3.20) with S and y by fixing θ_F.

    Update the feature extractor F_N by ascending its gradient w.r.t. Eq. (3.22) by fixing θ_{D_N}.

    Update the generator G by descending its gradient w.r.t. Eq. (3.22) with $G(\mathbf{z})$ by fixing θ_{D_N}.

    Update the node representation discriminator D_N by ascending its gradient w.r.t. Eq. (3.22) with $G(\mathbf{z})$ and $\mathbf{x}_v$ (i.e., $F_N(S)$) by fixing θ_F and θ_G.

  end for

end for

independent with each other, thus we utilize the mini-batch gradient descent to optimize the VANE framework in parallel.

3.3.4 Experimental Evaluation

Datasets *Aminer* [141] is an academic literature dataset, which contains 27,734 papers as nodes. We observe two types of views: (1) Citation view, where an edge represents for a reference record between two papers; (2) Common-author view, where an edge represents that two papers share at least one common author. For the mentioned two views, we have 111,819 and 525,623 edges.

Twitter-Rugby [142] is a collection of 850 rugby-related Twitter users. We observe three views: (1) Follow view, where an edge stands for a user following another user; (2) Mention view, where an edge stands for a user mentioning another user in his/her Tweet; (3) Retweet

view, where an edge stands for a user retweeting another user's Tweet. For the above three views, we have 22,861, 21,660, and 9,627 edges, respectively.

Baseline Methods We extract node sequences by the random walk [129] and the biased random walk [131], such that two versions of VANE are named as VANE-RW and VANE-BRW. We compare VANE with single-view methods like DeepWalk [129], node2vec [131] and GraphGAN [134], and multi-view methods like MNE [136] and MVE [135]. For single-view algorithms, we combine all individual views into a combined view, where the edge exists if it exists in any specific view. Moreover, MNE is designed to generate node embeddings for each view.

Node Classification and Link Prediction based on Domain-Invariant Representation *Node Classification.* In the Aminer, we adopt venues as node labels and node embeddings as node features. We shuffle the dataset and sample 90% as the training set for a k-NN classifier and test on the rest 10%. We train 10 times and report the mean and standard deviation of accuracy in Table 3.6. Since MNE cannot get embedding results on an Intel i7 CPU, 64GB RAM machine within 96 hours, we don't report it. In the Twitter-Rugby, we adopt node embeddings as node features and user geo-locations as node labels. We use the same classifier setting in the Aminer and report results in Table 3.7.

Link Prediction. For each dataset, we remove 10% of the shared common links (i.e., the edge exists in every view) and learn node embeddings on the truncated graph. We denote removed links as the positive samples, and the links never appeared as negative samples. We sample the same number of negative samples and positive samples and use the cosine similarity to measure the proximity between two nodes. In the Aminer, there are many missing links between research communities. To generate quality negative samples, we conduct link prediction tasks only on Bioinformatics venue papers. In the Twitter-Rugby, since all users share the characteristic of rugby enthusiast, we use the whole dataset. The link prediction performances on two datasets are shown in Table 3.6 and Table 3.7.

Our VANE model outperforms baselines on both tasks. An intuitive explanation is that: compared with single-view algorithms [129, 131, 134], representations of the VANE framework are more comprehensive across views to support the view adversarial game; and comparing with the multiplex algorithms [135, 136], representations of the VANE framework are more robust by involving the generator. To further verify our guess, we design the following ablation study.

Table 3.6: Performance on the Aminer Dataset

Methods	View	Accuracy (%)	
		Node Classification	Link Prediction
DeepWalk	Citation	78.03±0.72	95.99
	Common-Author	72.72±0.77	96.29
	Combined	74.94±0.54	97.28
node2vec	Citation	78.05±0.57	97.73
	Common-Author	73.69±0.68	95.58
	Combined	74.88±0.91	97.85
GraphGAN	Citation	74.29±1.20	88.93
	Common-Author	72.07±1.11	89.57
	Combined	71.69±1.06	90.21
MNE	Citation	N/A	54.25
	Common-Author	N/A	52.32
MVE	All	80.16±0.42	72.82
VANE-RW	All	78.84±0.63	97.62
VANE-BRW	All	**80.79±0.80**	**98.53**

3.3.5 Summary

Graph representation learning has demonstrated effective empirical performance for various network mining tasks such as node classification, link prediction, clustering, and anomaly detection. However, most of these algorithms focus on the single-view network scenario. From a real-world perspective, one individual node can have different connectivity patterns in different networks. For example, one user can have different relationships on Twitter, Facebook, and LinkedIn due to varying user behaviors on different platforms. In this case, jointly considering the structural information from multiple platforms (i.e., multiple views) can potentially lead to more comprehensive node representations, and eliminate noises and bias from a single view. Therefore, we propose a view-adversarial framework to generate comprehensive and robust multi-view network representations named VANE, which is based on two adversarial games. The first adversarial game enhances the comprehensiveness of the node representation by discriminating the view information that is obtained from the subgraph induced by neighbors of that node. The second adversarial game improves the robustness of the node representation with the challenging of fake node representations from the generative adversarial net. We conduct extensive experiments on downstream tasks with real-world multi-view networks, which shows that our proposed VANE framework significantly outperforms other baseline methods.

Table 3.7: Performance on Twitter-Rugby Dataset

Methods	View	Accuracy (%)	
		Node Classification	Link Prediction
DeepWalk	Follow	70.95±2.56	50.30
	Mention	69.64±5.46	50.27
	Retweet	73.78±5.18	52.24
	Combined	66.47±2.85	50.03
node2vec	Follow	79.52±4.42	65.45
	Mention	79.64±3.47	62.94
	Retweet	81.83±4.31	52.18
	Combined	80.59±2.75	60.61
GraphGAN	Follow	76.15±1.92	53.97
	Mention	71.95±2.74	51.88
	Retweet	39.20±2.42	50.21
	Combined	72.44±1.69	55.41
MNE	Follow	85.66±2.87	56.37
	Mention	84.70±3.45	74.66
	Retweet	85.06±3.42	76.15
MVE	All	83.76±4.90	68.85
VANE-RW	All	82.89±2.38	69.40
VANE-BRW	All	**90.60±2.57**	**85.36**

CHAPTER 4: EMPOWERING SUBGRAPH-LEVEL GRAPH INTELLIGENCE VIA NATURAL AND ARTIFICIAL DYNAMICS

In this chapter, based on subgraph-level graph intelligence and knowledge discovery, we have three sections introducing how natural dynamics help increase link prediction efficiency and accuracy (Section 4.1), how natural + artificial dynamics help access local high-order graph clustering (Section 4.2), and how natural + artificial dynamics help achieve fair high-order graph partitioning (Section 4.3).

4.1 INCREASING LINK PREDICTION EFFICIENCY AND ACCURACY VIA NATURAL DYNAMICS

4.1.1 Overview and Motivation

Link prediction is an important research topic in the graph deep learning community [143], especially in complex temporal networks [144, 145]. Learning from the time-evolving topological structures and time-evolving node and edge features, and determining whether the link (i.e., edge or interaction between two nodes) exists at a certain timestamp are challenging but have attracted much research interests like dynamic protein-protein interactions [146, 147], recommender systems [148], social network analysis [149], and many more.

Recently, with the success of the attention mechanism [74], graph transformer techniques have been proposed for the structured data to obtain the outperformance in many graph applications, like GraphGPS [64] for graph classification, TokenGT [71] for node and graph classification, and DyGFormer [150] for link prediction. However, to preserve the long-distance relation into the representation vectors, graph transformers usually need high time complexity due to the dense attention or relational attention [31, 67], i.e., each pair of nodes need to be attended with $O(n^2)$ time complexity, which can even approach $O(n^3)$ in a more complex relational setting [151].

The above complexity analysis places the burden on the temporal link prediction task from at least two aspects [37]. First, to make the current link prediction, the decision models need to consider the interaction history, and the corresponding spatial-temporal structure will force the attention mechanism to become more complex to attend to more interactions. Second, besides the spatial-temporal structure, the node and edge features are also evolving over time, the prolonged features also increase the size of neural architectures. Promisingly, some nascent work [152] proposed to only use Multilayer Perceptrons (MLPs) to accomplish the temporal link prediction. Although [152] does not meet the superiority performance

of the graph transformers [150], it paves the way for exploring the ability of MLPs in the temporal link prediction task.

Thus, we propose a simple temporal link prediction model, named SimpleTLP. In SimpleTLP, we propose two novel techniques: *Learnable Positional Encoding* Module (LPE) and *Node-Link-Positional Encoder* based on Discrete Fourier Transform. With those proposed techniques, even MLPs can achieve better performance than graph transformers in the temporal link prediction task. To verify this, we first prove that the proposed positional encoding method in SimpleTLP can preserve the temporal graph topology from the spatial-temporal spectral viewpoint. Moreover, we design the comprehensive experiential setting that contains 13 benchmark datasets, 9 SOTA baseline algorithms, 2 learning settings (transductive and inductive), and 3 sampling strategies, where our SimpleTLP performs the best across the board.

4.1.2 Preliminary

Temporal Graph Snapshot A snapshot of a temporal graph G is a collection of temporal interactions (i.e., edges) at time t, which is defined as $G^t = \{(u, v, t) \big| u \in V^t; v \in V^t; (u, v, t) \in E^t\}$. Each event (u, v, t) denotes an interaction between nodes u and v that occurs at time t, and the node and edge sets of G^t are denoted as V^t and E^t respectively. Additionally, for node $u \in V^t$, we denote the node feature as $\mathbf{s}_u^t \in \mathbb{R}^{d_N}$, and the edge features associated with an event (u, v, t) as $\mathbf{e}_{u,v}^t \in \mathbb{R}^{d_E}$, where d_N and d_E are the dimensions of the node and edge features.

Temporal Neighbors The temporal neighborhood of a node u at time t, denoted as $\mathcal{N}_u^t$, is the set of all interactions that involve u at time t. Formally, $\mathcal{N}_u^t$ can be mathematically expressed as $\mathcal{N}_u^t = \{(u, v, t) \big| v \in V^t; (u, v, t) \in E^t\}$.

Time Encoder A time encoder is a function to obtain the vector representation of time t [27, 152, 153]. Let d_T be the dimension of the time encoding vector, we define our time encoding function as $f_T : \mathbb{R}^+ \to \mathbb{R}^{d_T}$. In SimpleTLP, for $t \in \mathbb{R}^+$, $f_T(t) = \cos(t \cdot \omega)$, where $\omega \in \mathbb{R}^{d_T}$ with the ith entry computed as $\omega_i = \alpha^{-(i-1)/\beta}$, and $\alpha, \beta \in \mathbb{R}$ are hyper-parameters. In order to enhance the ability of d_T in distinguishing all timestamps, given that t_{max} is the maximum timestamp being considered, α and β should be selected in such a way that $t_{max} \cdot \alpha^{-(i-1)/\beta} \to 0$ as i goes towards d_T. Moreover, α, β, and ω are fixed during the training process of SimpleTLP.

Positional Encoding for Graphs Originated from the transformer [74], positional encoding is proposed to add node feature information to support the attention mechanism by indicating the relative position of a node within the input graph [154], and the common positional encoding method is to compute the Laplacian eigenvector for each node [64, 65]. Therefore, if two nodes are close to each other, their positional encodings should also be close.

Mathematically, given an n-node (static) graph $G = (V, E)$ and its adjacency matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$, the normalized Laplacian of G is computed as $\mathbf{L} = \mathbf{I}_n - Ph.D.^{-1/2}\mathbf{A}Ph.D.^{-1/2}$, where $\mathbf{I}_n$ is the $n \times n$ identity matrix, and $Ph.D.$ is the diagonal degree matrix of G. Let the eigen-decomposition of $\mathbf{L}$ be $\mathbf{L} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^\top$, where $\mathbf{\Lambda} = \text{diag}(\lambda_1, \lambda_2, \ldots, \lambda_n)$ is a diagonal matrix consisting of $\mathbf{L}$'s eigenvalues and $\mathbf{U}$ consists n eigenvectors of $\mathbf{L}$ as its columns. Let $\mathbf{u}_i \in \mathbb{R}^n$ be the ith eigenvector of $\mathbf{L}$ and its jth entry denoted as $\mathbf{u}_{i,j}$. Then, the positional encoding of node $i \in V$ is defined as $\mathbf{P}_i = [\mathbf{u}_{1,i}, \ldots, \mathbf{u}_{n,i}]^\top \in \mathbb{R}^n$. In practice, the dimension d_P of positional encoding is usually set to be less than n for computational efficiency [58], i.e., $d_P \ll n$ and $\mathbf{P}_i = [\mathbf{u}_{1,i}, \ldots, \mathbf{u}_{d_P,i}]^\top \in \mathbb{R}^{d_P}$.

Discrete Fourier Transform To pave the way for learnable positional encoding, we need to first introduce the basics of Discrete Fourier Transform (DFT) [155]. In brief, DFT is designed to convert a finite-length sequence of samples into another sequence in the frequency domain (with the same length). Formally, given a sequence with length of N, $\{\mathbf{x}_j\}_{j=1}^N$, the DFT operation $\mathcal{F}(\{\mathbf{x}_j\}_{j=1}^N)$ converts $\{\mathbf{x}_j\}_{j=1}^N$ to the complex-valued sequence $\{\mathbf{X}_j\}_{j=1}^N$ as $\mathbf{X}_j = \sum_{k=1}^N \mathbf{x}_k e^{-i2\pi \frac{j}{N} k}$. Also, to re-construct the original sequence $\{\mathbf{x}_j\}_{j=1}^N$, inverse DFT (IDFT) $\mathcal{F}^{-1}(\{\mathbf{X}_j\}_{j=1}^N)$ is defined as $\mathbf{x}_j = \sum_{k=1}^N \mathbf{X}_k e^{i2\pi \frac{j}{N} k}$.

4.1.3 Proposed Method: SimpleTLP

To begin with, we focus on a general research question, i.e., *given two nodes u and v, how can we predict whether the link (interaction or edge) exists between u and v at a certain timestamp t, effectively and efficiently*?

To answer this question, we propose a simple temporal link prediction model named SimpleTLP. It retrieves the interaction history of two nodes and decides whether the current link exists or not. Without heavy neural architectures like dense attention [64] or relational attention [151], SimpleTLP only depends on two of our proposed simple yet effective techniques, i.e., *Learnable Positional Encoding Module* (LPE) and *Node-Link-Positional Encoder*, such that SimpleTLP can preserve the spatial-temporal topology rigorously and outperform attention-based state-of-the-art link prediction methods.

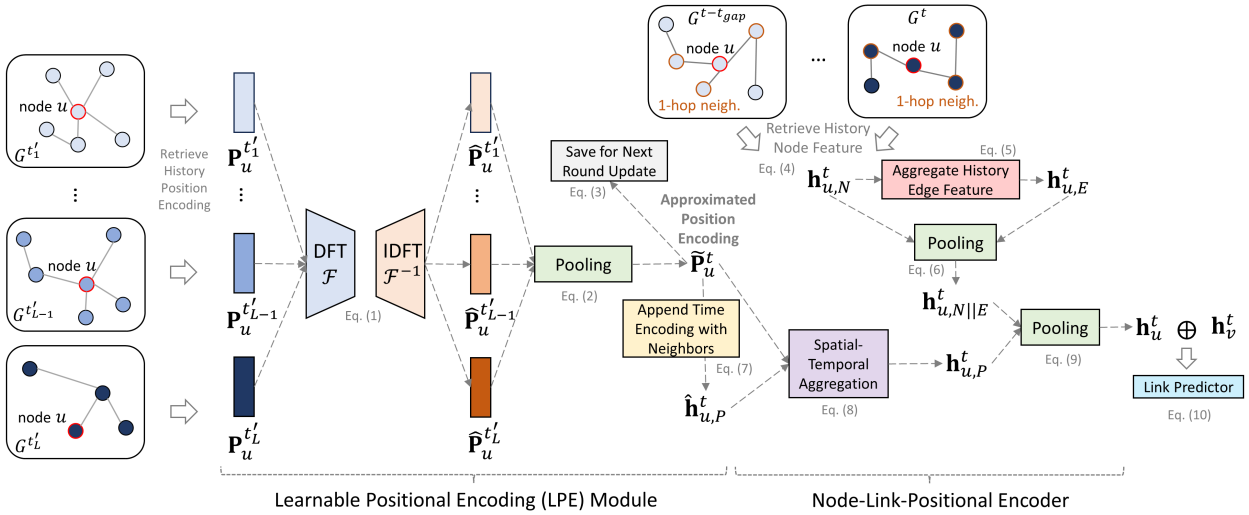


Figure 4.1: Overall Framework of SimpleTLP

The overall framework of SimpleTLP is shown in Figure 4.1 about how to retrieve and learn from history to make the current decision with two proposed techniques. Next, we will introduce the details of the proposed techniques, i.e., *Learnable Positional Encoding Module* and *Node-Link-Positional Encoder*.

Learnable Positional Encoding (LPE) Module The main idea of Learnable Positional Encoding (LPE) is to leverage historical positional encodings to determine whether the link exists at the current timestamp. However, learning from the history data in an effective and efficient way and only through positional encodings is challenging, because state-of-the-art methods usually need a complex attention mechanism [150, 156].

Motivated by that, we wish to only use position encodings to carry informative and useful knowledge to record what had happened in the past (e.g., what interactions occurred and what the node and edge features looked like around the interaction). Existing simple hand-crafted positional encodings can be efficient but tend to induce suboptimal performance [27, 152].

Thus, we propose a "*learnable*" positional encoding module, enabling it to reflect complex and non-linear knowledge. As shown in Figure 4.1, in our SimpleTLP, LPE estimates (or approximates) the positional encoding at current time t by applying the Fourier Transform (with learnable parameters) on the past positional encodings. It is worth noting that obtaining the positional encoding through LPE (1) records the complex spatial-temporal information, (2) and avoids computing the Laplacian eigendecomposition for each timestamp from scratch, which is computationally expensive (e.g., $O(|V|^3)$ [65] or $O(|E|^{3/2})$ [154])

Because of no peeking at the current graph topology structure and features, our LPE is an

iterative updating module. Next, we introduce LPE in two folds: (1) given the past positional encodings, how LPE approximates the current positional encoding (that will be used to determine links by *Node-Link-Positional Encoder*, and (2) how to store the approximate positional encodings and update/optimize them in the future round with current topology structure and features observed.

Approximating current position encoding. First of all, we denote the positional encoding of u to be learned for time t as $\mathbf{P}_u^t \in \mathbb{R}^{d_P}$. Suppose there are L distinct timestamps prior to t, i.e., $t'_1, \ldots, t'_L$, and we can retrieve the already learned $\{\mathbf{P}_u^{t'_1}, \ldots, \mathbf{P}_u^{t'_L}\}$ that is regarded as a L-length sequence. Then, in this LPE module, we leverage the Discrete Fourier Transform as a way to capture the temporal dependencies among $\mathbf{P}_u^{t'_1}, \ldots, \mathbf{P}_u^{t'_L}$ and derive an estimated positional encoding of u at time t denoted by $\widetilde{\mathbf{P}}_u^t \in \mathbb{R}^{d_P}$.

To be specific, we start by applying Discrete Fourier Transform (DFT) on the sequence $\{\mathbf{P}_u^{t'}\}_{t'=t'_1}^{t'_L}$ to obtain the corresponding sequence in the frequency domain. Then, we employ a complex-valued learnable filter, $\mathbf{W}_{filter} \in \mathbb{C}^{d_P \times L}$, to filter out noises of the sequence in the frequency domain, and finally transform back to the original domain. The process can be mathematically expressed as

$$\{\widehat{\mathbf{P}}_u^{t'_j}\}_{j=1}^{L} = \mathcal{F}^{-1}(\mathbf{W}_{filter} \odot \mathcal{F}(\{\mathbf{P}_u^{t'_j}\}_{j=1}^{L})) \tag{4.1}$$

Then, we obtain the approximate positional encoding $\widetilde{\mathbf{P}}_u^t$ by applying sum-pooling with learnable weights on $\{\widehat{\mathbf{P}}_u^{t'_j}\}_{j=1}^{L}$ by

$$\widetilde{\mathbf{P}}_u^t = \begin{bmatrix} \widehat{\mathbf{P}}_u^{t'_1} & \ldots & \widehat{\mathbf{P}}_u^{t'_L} \end{bmatrix} \mathbf{W}_p^{sum} \tag{4.2}$$

where $\mathbf{W}_p^{sum} \in \mathbb{R}^{L \times 1}$. Note that the approximate positional encoding $\widetilde{\mathbf{P}}_u^t$ will be sent to a Node-Link-positional Encoder (introduced in the next paragraph) for making predictions for current time t.

Storing approximate positional encoding for future retrieval and optimization. Since SimpleTLP does not peek at the current topology structure and features when making the prediction for now, i.e., only using approximate positional encodings $\widetilde{\mathbf{P}}_u^t$, we want to clarify how the current approximate positional encodings get updated and optimized in the next round.

After making the current link predictions at time t, SimpleTLP can then "peek" (or "reveal") the ground-truth links at time t, such that it can update the approximation $\widetilde{\mathbf{P}}_u^t$ into updated $\mathbf{P}_u^t$ for the future retrieval and predictions.

Then, we need to update the positional encoding of u at t. Again, we sample the most

K recent interactions, *including those that occur at time t*. Suppose they are indexed as $\{(\bar{u}_1, u, \bar{t}_1), \ldots, (\bar{u}_K, u, \bar{t}_K)\}$ where $\bar{t}_1 \leq \cdots \leq \bar{t}_K \leq t$. Leveraging the approximate positional encodings from $\bar{u}_1, \ldots, \bar{u}_K$, $\mathbf{P}_u^t$ is derived as

$$\mathbf{P}_u^t = \widetilde{\mathbf{P}}_u^t + tanh(\mathbf{W}_P^{(self)}\widetilde{\mathbf{P}}_u^t + \mathbf{W}_P^{(2)}\text{RELU}(\mathbf{W}_P^{(1)}\widehat{\mathbf{q}}_u^t)) \in \mathbb{R}^{d_P} \tag{4.3}$$

where $\widehat{\mathbf{q}}_u^t = \sum_{j=1}^{K}[(f_T(t-\bar{t}_j))^\top \,||\, (\widetilde{\mathbf{P}}_{\bar{u}_j}^t)^\top]^\top \in \mathbb{R}^{d_T+d_P}$, f_T is the time encoder, and $\mathbf{W}_P^{(1)} \in \mathbb{R}^{d_P\times(d_P+d_T)}, \mathbf{W}_P^{(2)} \in \mathbb{R}^{d_P\times d_P}, \mathbf{W}_P^{(self)} \in \mathbb{R}^{d_P\times d_P}$ are learnable MLP parameters.

At the first time t_0, we denote the initial positional encoding as the Laplacian of the initial graph snapshot G^{t_0}. Formally, let $\mathbf{L}$ be the Laplacian of G^{t_0} and the eigen-decomposition of $\mathbf{L}$ be $\mathbf{U}\Lambda\mathbf{U}^T$, then $\mathbf{P}_u^{t_0} = [\mathbf{u}_{1,u}, \ldots, \mathbf{u}_{d_P,u}]^T \in \mathbb{R}^{d_P}$, where $\mathbf{u}_i$ is the i-th eigenvector of $\mathbf{L}$.

Node-Link-Positional Encoder For node u, the Node-Link-Positional Encoder is designed to aggregate its historical node and edge features with the positional encodings of its recent 1-hop neighbors prior to time t, and, eventually, derive a compact representation associated with u that encodes all the information.

In general, Node-Link-Positional Encoder works as follows. We first use $\mathbf{h}_{u,N}^t$ and $\mathbf{h}_{u,E}^t$ to denote the node and link encodings of recent 1-hop neighbors of node u. Then, we apply MLP transformations on $\mathbf{h}_{u,N}^t$ and $\mathbf{h}_{u,E}^t$ to obtain an encoding, denoted by $\mathbf{h}_{u,N\|E}^t$, which represents the combination of node-link information. For adding positional information, we use $\mathbf{h}_{u,P}^t$ to denote the aggregated positional encoding of recent 1-hop neighbors, and further transform them by MLP layers. Eventually, $\mathbf{h}_{u,N\|E}^t$ and $\mathbf{h}_{u,P}^t$ are combined by a 1-layer MLP to derive a temporal representation at time t for u, denoted by $\mathbf{h}_u^t$, which will be sent together with $\mathbf{h}_v^t$ to the link predictor to determine whether the link exists between nodes u and v at time t.

The detailed computational procedures of the Node-Link-Positional Encoder are explained below.

Firstly, the node encoding of recent 1-hop neighbors of u can be obtained by aggregating features of nodes that interact with u in the time interval $[t-t_{gap}; t]$, where t_{gap} is a dataset-dependent hyper-parameter. Mathematically, this process can be described as

$$\mathbf{h}_{u,N}^t = \mathbf{s}_u^t + \text{Mean}(\{\mathbf{s}_{u'}^{t'} \big| (u', u, t') \in \mathcal{N}_u^{t-t_{gap}} \cup \cdots \cup \mathcal{N}_u^t\}; t' \in [t-t_{gap}, t]) \in \mathbb{R}^{d_N} \tag{4.4}$$

where $\mathbf{s}_u^t$ is the input node feature, and $\text{Mean}(\cdot)$ is the mean pooling operation.

We derive the link encoding for u by first extracting the K most recent interactions involving u that occurred before time t. For each interaction, we then combine its edge

features and the relative time differences. Finally, we further apply MLP transformations to these combined representations to obtain the link encoding.

Then, we derive the link encoding for u by extracting the most K recent interactions involving u that occur prior to time t, combine the edge features and the time encoding of these interactions, and further apply MLP transformations on these representations. Specifically, suppose the most K recent interactions of u are indexed as $\{(u_1, u, t'_1), \ldots, (u_K, u, t'_K)\}$, where $t'_1 \leq \cdots \leq t'_K < t$. Let $\mathbf{H}^t_{u,E} \in \mathbb{R}^{K \times (d_T + d_E)}$ be a matrix, whose k^{th} row is defined as $[\mathbf{H}^t_{u,E}]_k = [(f_T(t - t'_k))^\top \,||\, (\mathbf{e}^t_{u,u'_k})^\top] \in \mathbb{R}^{1 \times (d_T + d_E)}$, where $[\cdot||\cdot]$ denotes concatenation. If u does not involve at least K interactions before t then $\mathbf{H}^t_{u,E}$ is zero-padded so that the matrix has K rows. We first apply a 1-layer MLP on the last dimension of $\mathbf{H}^t_{u,E}$, sum-pool over K rows of $\mathbf{H}^t_{u,E}$ with learnable weights to obtain a vector of dimension $d_T + d_E$, and finalize the encoding with an additional 1-layer MLP transformation as

$$\mathbf{h}^t_{u,E} = \mathbf{W}^{(2)}_{link}(\text{RELU}((\mathbf{H}^t_{u,E}\mathbf{W}^{(1)}_{link})^\top \mathbf{W}^{sum}_{link})) \in \mathbb{R}^{d_T + d_E} \tag{4.5}$$

where $\mathbf{W}^{(1)}_{link}, \mathbf{W}^{(2)}_{link} \in \mathbb{R}^{(d_T + d_E) \times (d_T + d_E)}$, and $\mathbf{W}^{sum}_{link} \in \mathbb{R}^{K \times 1}$.

After that, $\mathbf{h}^t_{u,N\|E}$ is obtained by a 1-layer MLP transformation as

$$\mathbf{h}^t_{u,N\|E} = \mathbf{W}_{N\|E}([(\mathbf{h}^t_{u,N})^\top \,||\, (\mathbf{h}^t_{u,E})^T])^\top \in \mathbb{R}^{d_N} \tag{4.6}$$

where $\mathbf{W}_{N\|E} \in \mathbb{R}^{d_N \times (d_N + d_T + d_E)}$.

Next, for historical positional information, we apply sum aggregation over positional information from the aforementioned most recent K interactions involving u as

$$\widehat{\mathbf{h}}^t_{u,P} = \sum_{j=1}^{K} [(f_T(t - t'_j))^\top \,||\, (\widetilde{\mathbf{P}}^t_{u_j})^\top]^\top \in \mathbb{R}^{d_T + d_P} \tag{4.7}$$

where f_T is the time encoder. Then, $\mathbf{h}^t_{u,P}$ is obtained through the neighborhood-aggregated positional encoding $\widehat{\mathbf{h}}^t_{u,P}$ and $\widetilde{\mathbf{P}}^t_u$ as

$$\mathbf{h}^t_{u,P} = \widetilde{\mathbf{P}}^t_u + tanh(\mathbf{W}^{(self)}_P \widetilde{\mathbf{P}}^t_u + \mathbf{W}^{(2)}_P \text{RELU}(\mathbf{W}^{(1)}_P \widehat{\mathbf{h}}^t_{u,P})) \in \mathbb{R}^{d_P} \tag{4.8}$$

where $\mathbf{W}^{(1)}_P \in \mathbb{R}^{d_P \times (d_P + d_T)}, \mathbf{W}^{(2)}_P \in \mathbb{R}^{d_P \times d_P}, \mathbf{W}^{(self)}_P \in \mathbb{R}^{d_P \times d_P}$, and $tanh$ is the hyperbolic tangent function.

Finally, the temporal representation at time t of u, $\mathbf{h}^t_u$, is derived as

$$\mathbf{h}^t_u = \mathbf{W}[(\mathbf{h}^t_{u,N\|E})^\top \,||\, (\mathbf{h}^t_{u,P})^\top]^\top \in \mathbb{R}^{d_N} \tag{4.9}$$

where $\mathbf{W} \in \mathbb{R}^{d_N \times (d_N + d_P)}$.

Optimization With the two embeddings $\mathbf{h}_u^t$ and $\mathbf{h}_v^t$, we can now let SimpleTLP decide whether a link exists between nodes u and v at time t and train SimpleTLP against the ground truth connections at t.

Link predictor. Given node pair (u, v) and timestamp t, SimpleTLP's goal is to predict whether there is a link between (u, v) at time t. Specifically, the Link Predictor would give the probability $\hat{y}$ $(0 \leq \hat{y} \leq 1)$ of whether u and v interact at time t by

$$\hat{y} = sigmoid(\mathbf{W}^{(2)}\text{RELU}(\mathbf{W}^{(1)}[(\mathbf{h}_u^t)^\top \,||\, (\mathbf{h}_v^t)^\top]^\top)) \in \mathbb{R} \tag{4.10}$$

where *sigmoid* denotes the sigmoid function mapping all real values to the range $(0, 1)$, and $\mathbf{W}^{(1)} \in \mathbb{R}^{(2d_N) \times d_N}, \mathbf{W}^{(2)} \in \mathbb{R}^{d_N \times 1}$ are 1-layer MLPs.

Loss functions. To begin with, we point out the necessary components of the objective loss function for the training process of SimpleTLP, give formal definitions for those components, and derive the final loss function used in the training process of SimpleTLP.

Given B positive (i.e., existing edges) samples $(u_1^{(pos)}, v_1^{(pos)}, t_1), \ldots, (u_B^{(pos)}, v_B^{(pos)}, t_B)$ and B negative (i.e., non-existing edges) samples $(u_1^{(neg)}, v_1^{(neg)}, t_1), \ldots, (u_B^{(neg)}, v_B^{(neg)}, t_B)$, and the ground truth label for the positive samples should be 1, and 0 for negative samples, as the link prediction can be regarded as a binary classification problem, we then employ the binary cross-entropy loss function to obtain the link prediction loss $\mathcal{L}_{lp}$ as

$$\mathcal{L}_{lp} = \frac{-1}{2B}\left[\left(\sum_{i=1}^{B} \log\left(\hat{y}_{u_i^{(pos)}, v_i^{(pos)}}\right)\right) + \left(\sum_{i=1}^{B} (1 - \log\left(\hat{y}_{u_i^{(neg)}, v_i^{(neg)}}\right)\right)\right] \tag{4.11}$$

where $\hat{y}_{u_i^{(pos)}, v_i^{(pos)}}$ denotes the predicted probability of link existing between $u_i^{(pos)}, v_i^{(pos)}$ at time t_i, as defined in Eq. 4.10, and $\hat{y}_{u_i^{(neg)}, v_i^{(neg)}}$ is the predicted probability of link existing between $u_i^{(neg)}, v_i^{(neg)}$ at time t_i. Thus, ideally, $\hat{y}_{u_i^{(pos)}, v_i^{(pos)}}$ should be close to 1, and $\hat{y}_{u_i^{(neg)}, v_i^{(neg)}}$ should be close to zero.

In addition, we introduce the positional encoding loss $\mathcal{L}_{pe}$ to assess the learned positional encoding, defined in Eq. 4.2. Specifically, we would expect $\widetilde{\mathbf{P}}^{t_i}_{u_i^{(pos)}}$ be close to $\widetilde{\mathbf{P}}^{t_i}_{v_i^{(pos)}}$, as there is a link between $u_i^{(pos)}, v_i^{(pos)}$ at time t_i. Thus, the difference $||\widetilde{\mathbf{P}}^{t_i}_{u_i^{(pos)}} - \widetilde{\mathbf{P}}^{t_i}_{v_i^{(pos)}}||_2$ (where $||\cdot||_2$ denotes the 2-norm of a real-valued vector) should be small, so that the learned positional encoding correctly reflects the nature of positional encoding. Moreover, in order to avoid SimpleTLP giving similar positional encodings to all nodes to minimize the aforementioned 2-norm difference of positive samples, we add another constraint, i.e.,

maximizing the difference $\alpha_{neg}||\widetilde{\mathbf{P}}^{t_i}_{u_i^{(neg)}} - \widetilde{\mathbf{P}}^{t_i}_{v_i^{(neg)}}||_2$, where α_{neg} is a positive scaling factor. As $(u_1^{(neg)}, v_1^{(neg)}, t_1), \ldots, (u_B^{(neg)}, v_B^{(neg)}, t_B)$ are not all negative samples, so we let $\alpha_{neg} < 1$ to avoid SimpleTLP emphasizing the negativity of these B negative samples. Therefore, we define the positional encoding loss as

$$\mathcal{L}_{pe} = \frac{1}{B}\left[\left(\sum_{i=1}^{B} ||\widetilde{\mathbf{P}}^{t_i}_{u_i^{(pos)}} - \widetilde{\mathbf{P}}^{t_i}_{v_i^{(pos)}}||_2\right) - \alpha_{neg}\left(\sum_{i=1}^{B} ||\widetilde{\mathbf{P}}^{t_i}_{u_i^{(neg)}} - \widetilde{\mathbf{P}}^{t_i}_{v_i^{(neg)}}||_2\right)\right] \tag{4.12}$$

Finally, the overall loss function is

$$\mathcal{L} = (1 - \alpha_{pe})\mathcal{L}_{lp} + \alpha_{pe}\mathcal{L}_{pe} \tag{4.13}$$

where $\alpha_{pe} \in (0, 1)$ is the weight for $\mathcal{L}_{pe}$.

4.1.4 Experimental Evaluation

Datasets and Baselines We assess the ability of SimpleTLP's in performing Link Prediction with 13 datasets covering various domains and collected by [156]: *Wikipedia, Reddit, MOOC, LastFM, Enron, Social Evo., UCI, Flights, Can. Parl., US Legis., UN Trade, UN Vote,* and *Contact*. Details about the dataset statistics are shown in Table 4.1. We compare SimpleTLP with 8 state-of-the-art baselines, including JODIE [157], DyRep [158], TGAT [153], TGN [159], CAWN [27], TCL [160], GraphMixer [152], and DyGFormer [150]. Across all 13 datasets, training/validation/testing sets are following the benchmark [150] by chronological splits with ratios 70%/15%/15%.

Table 4.1: Statistics of the datasets.

Datasets	Domains	#Nodes	#Links	#N&L Feat	Bipartite	Duration	Unique Steps	Time Granularity
Wikipedia	Social	9,227	157,474	– & 172	True	1 month	152,757	Unix timestamps
Reddit	Social	10,984	672,447	– & 172	True	1 month	669,065	Unix timestamps
MOOC	Interaction	7,144	411,749	– & 4	True	17 months	345,600	Unix timestamps
LastFM	Interaction	1,980	1,293,103	– & –	True	1 month	1,283,614	Unix timestamps
Enron	Social	184	125,235	– & –	False	3 years	22,632	Unix timestamps
Social Evo.	Proximity	74	2,099,519	– & 2	False	8 months	565,932	Unix timestamps
UCI	Social	1,899	59,835	– & –	False	196 days	58,911	Unix timestamps
Flights	Transport	13,169	1,927,145	– & 1	False	4 months	122	days
Can. Parl.	Politics	734	74,478	– & 1	False	14 years	14	years
US Legis.	Politics	225	60,396	– & 1	False	12 congresses	12	congresses
UN Trade	Economics	255	507,497	– & 1	False	32 years	32	years
UN Vote	Politics	201	1,035,742	– & 1	False	72 years	72	years
Contact	Proximity	692	2,426,279	– & 1	False	1 month	8,064	5 minutes

Evaluation Metrics and Settings Following existing works [150, 153, 159, 159] in evaluating models for link prediction, we assess SimpleTLP under two settings: **transductive** setting and **inductive** setting. Under the transductive setting, models will predict the link occurrence in future timestamps between nodes that had been observed during the training process, while inductive setting involves predicting future links between unseen nodes in the training process. For each setting, we use two evaluation metrics: Average Precision (AP) and Area Under the Receiver Operating Characteristic Curve (AUC-ROC). In addition, following [156], for more robust evaluation, each baseline is assessed with three negative sampling strategies (NSS): *random, historical,* and *inductive*. Specifically, random NSS samples, any possible node pairs, historical NSS samples, negative edges from the edges that occurred in past timestamps but are absent in the present time, and inductive NSS samples with edges that are unseen during the training time. We refer readers to [156] for more details regarding these three sampling strategies. The model with the best performance, in terms of metric scores AP and AUC-ROC, on the validation set will be selected for testing. We run SimpleTLP five times with different random seeds from 0 to 4 and report the average metric score. Results from all other baselines are also obtained in the same manner [150].

Comparison of Empirical Results For each dataset, we report the average metric score and the standard deviation on the testing set across 5 runs with different seeds. Due to the limited space, we present the empirical performance of all baseline methods on AP and AUC-ROC metrics with *random* negative sampling strategy for transductive setting in Table. 4.2 and inductive setting in Table. 4.3. We report the corresponding results for *historical* negative sampling strategy in Table. 4.4, Table. 4.5 and for *inductive* negative sampling strategy in Table. 4.6, Table. 4.7. In all aforementioned tables, the best and second-best results are highlighted with **Red** and **Blue**, respectively.

For the random negative sampling strategy, as suggested in Table. 4.2 and Table. 4.3, SimpleTLP achieves competitive results over 13 datasets and outperforming most of the baselines with average rankings close to 1 under both settings. Moreover, the second-best results of SimpleTLP closely approach the corresponding best results by a small gap. Notably, on some datasets, such as *UN Trade, UN Vote*, the difference between the second-best and SimpleTLP's performance is substantial, suggesting that SimpleTLP makes great improvements over existing methods.

Regarding the historical and inductive sampling strategy, SimpleTLP also attains competitive results for both settings with the second-best average ranking under AP metric of *transductive* setting with historical negative sampling strategy, and the best ranking over other settings. Notably, as suggested in Table. 4.4, 4.5, 4.6, and 4.7, the best result from

Table 4.2: AP and ROC-AUC for *transductive setting* temporal link prediction with *random negative sampling strategy*

Metric	Datasets	JODIE	DyRep	TGAT	TGN	CAWN	TCL	GraphMixer	DyGFormer	SimpleTLP
AP	Wikipedia	96.50 ± 0.14	94.86 ± 0.06	96.94 ± 0.06	98.45 ± 0.06	98.76 ± 0.03	96.47 ± 0.16	97.25 ± 0.03	**99.03 ± 0.02**	**99.34 ± 0.04**
	Reddit	98.31 ± 0.14	98.22 ± 0.04	98.52 ± 0.02	98.63 ± 0.06	99.11 ± 0.01	97.53 ± 0.02	97.31 ± 0.01	**99.22 ± 0.01**	**99.37 ± 0.04**
	MOOC	80.23 ± 2.44	81.97 ± 0.49	85.84 ± 0.15	**89.15 ± 1.60**	80.15 ± 0.25	82.38 ± 0.24	82.78 ± 0.15	**87.52 ± 0.49**	86.94 ± 0.34
	LastFM	70.85 ± 2.13	71.92 ± 2.21	73.42 ± 0.21	77.07 ± 3.97	86.99 ± 0.06	67.27 ± 2.16	75.61 ± 0.24	**93.00 ± 0.12**	**96.06 ± 0.30**
	Enron	84.77 ± 0.30	82.38 ± 3.36	71.12 ± 0.97	86.53 ± 1.11	89.56 ± 0.09	79.70 ± 0.71	82.25 ± 0.16	**92.47 ± 0.12**	**93.96 ± 0.36**
	Social Evo.	89.89 ± 0.55	88.87 ± 0.30	93.16 ± 0.17	**93.57 ± 0.17**	84.96 ± 0.09	93.13 ± 0.16	93.37 ± 0.07	**94.73 ± 0.01**	92.22 ± 0.25
	UCI	89.43 ± 1.09	65.14 ± 2.30	79.63 ± 0.70	92.34 ± 1.04	95.18 ± 0.06	89.57 ± 1.63	93.25 ± 0.57	**95.79 ± 0.17**	**96.67 ± 0.47**
	Flights	95.60 ± 1.73	95.29 ± 0.72	94.03 ± 0.18	97.95 ± 0.14	98.51 ± 0.01	91.23 ± 0.02	90.99 ± 0.05	**98.91 ± 0.01**	**98.94 ± 0.10**
	Can. Parl.	69.26 ± 0.31	66.54 ± 2.76	70.73 ± 0.72	70.88 ± 2.34	69.82 ± 2.34	68.67 ± 2.67	77.04 ± 0.46	**97.36 ± 0.45**	**98.24 ± 0.11**
	US Legis.	75.05 ± 1.52	75.34 ± 0.39	68.52 ± 3.16	**75.99 ± 0.58**	70.58 ± 0.48	69.59 ± 0.48	70.74 ± 1.02	71.11 ± 0.59	**76.74 ± 0.60**
	UN Trade	64.94 ± 0.31	63.21 ± 0.93	61.47 ± 0.18	65.03 ± 1.37	65.39 ± 0.12	62.21 ± 0.03	62.61 ± 0.27	**66.46 ± 1.29**	**75.84 ± 2.08**
	UN Vote	63.91 ± 0.81	62.81 ± 0.80	52.21 ± 0.98	**65.72 ± 2.17**	52.84 ± 0.10	51.90 ± 0.30	52.11 ± 0.16	55.55 ± 0.42	**73.40 ± 0.03**
	Contact	95.31 ± 1.33	95.98 ± 0.15	96.28 ± 0.09	96.89 ± 0.56	90.26 ± 0.28	92.44 ± 0.12	91.92 ± 0.03	**98.29 ± 0.01**	**98.16 ± 0.09**
	Avg. Rank	6.00	6.61	6.38	3.30	5.15	7.53	6.07	**2.30**	**1.61**
ROC-AUC	Wikipedia	96.33 ± 0.07	94.37 ± 0.09	96.67 ± 0.07	98.37 ± 0.07	98.54 ± 0.04	95.84 ± 0.18	96.92 ± 0.03	**98.91 ± 0.02**	**99.48 ± 0.03**
	Reddit	98.31 ± 0.05	98.17 ± 0.05	98.47 ± 0.02	98.60 ± 0.06	99.01 ± 0.01	97.42 ± 0.02	97.17 ± 0.02	**99.15 ± 0.01**	**99.49 ± 0.03**
	MOOC	83.81 ± 2.09	85.03 ± 0.58	87.11 ± 0.19	**91.21 ± 1.15**	80.38 ± 0.26	83.12 ± 0.18	84.01 ± 0.17	87.91 ± 0.58	**89.28 ± 0.28**
	LastFM	70.49 ± 1.66	71.16 ± 1.89	71.59 ± 0.18	78.47 ± 2.94	85.92 ± 0.10	64.06 ± 1.16	73.53 ± 0.12	**93.05 ± 0.10**	**97.52 ± 0.17**
	Enron	87.96 ± 0.52	84.89 ± 3.00	68.89 ± 1.10	88.32 ± 0.99	90.45 ± 0.14	75.74 ± 0.72	84.38 ± 0.21	**93.33 ± 0.13**	**95.98 ± 0.23**
	Social Evo.	92.05 ± 0.46	90.76 ± 0.21	94.76 ± 0.16	**95.39 ± 0.17**	87.34 ± 0.08	94.84 ± 0.17	95.23 ± 0.07	**96.30 ± 0.01**	94.33 ± 0.21
	UCI	90.44 ± 0.49	68.77 ± 2.34	78.53 ± 0.74	92.03 ± 1.13	93.87 ± 0.08	87.82 ± 1.36	91.81 ± 0.67	**94.49 ± 0.26**	**97.62 ± 0.23**
	Flights	96.21 ± 1.42	95.95 ± 0.62	94.13 ± 0.17	98.22 ± 0.13	98.45 ± 0.01	91.21 ± 0.02	91.13 ± 0.01	**98.93 ± 0.01**	**99.38 ± 0.04**
	Can. Parl.	78.21 ± 0.23	73.35 ± 3.67	75.69 ± 0.78	76.99 ± 1.80	75.70 ± 3.27	72.46 ± 3.23	83.17 ± 0.53	**97.76 ± 0.41**	**98.97 ± 0.06**
	US Legis.	82.85 ± 1.07	82.28 ± 0.32	75.84 ± 1.99	**83.34 ± 0.43**	77.16 ± 0.39	76.27 ± 0.63	76.96 ± 0.79	77.90 ± 0.58	**83.89 ± 0.43**
	UN Trade	69.62 ± 0.44	67.44 ± 0.83	64.01 ± 0.12	69.10 ± 1.67	68.54 ± 0.18	64.72 ± 0.05	65.52 ± 0.51	**70.20 ± 1.44**	**83.17 ± 1.28**
	UN Vote	68.53 ± 0.95	67.18 ± 1.04	52.83 ± 1.12	**69.71 ± 2.65**	53.09 ± 0.22	51.88 ± 0.36	52.46 ± 0.27	57.12 ± 0.62	**79.59 ± 0.02**
	Contact	96.66 ± 0.89	96.48 ± 0.14	96.95 ± 0.08	97.54 ± 0.35	89.99 ± 0.34	94.15 ± 0.09	93.94 ± 0.02	**98.53 ± 0.01**	**98.72 ± 0.06**
	Avg. Rank	6.21	6.71	6.42	3.5	5.14	7.28	5.85	**2.28**	**1.57**

Table 4.3: AP and ROC-AUC for *inductive setting* temporal link prediction with *random negative sampling strategy*.

Metric	Datasets	JODIE	DyRep	TGAT	TGN	CAWN	TCL	GraphMixer	DyGFormer	SimpleTLP
AP	Wikipedia	94.82 ± 0.20	92.43 ± 0.37	96.22 ± 0.07	97.83 ± 0.04	98.24 ± 0.03	96.22 ± 0.17	96.65 ± 0.02	**98.59 ± 0.03**	**99.15 ± 0.04**
	Reddit	96.50 ± 0.13	96.09 ± 0.11	97.09 ± 0.04	97.50 ± 0.07	98.62 ± 0.01	94.09 ± 0.07	95.26 ± 0.02	**98.84 ± 0.02**	**98.02 ± 0.09**
	MOOC	79.63 ± 1.92	81.07 ± 0.44	85.50 ± 0.19	**89.04 ± 1.17**	81.42 ± 0.24	80.60 ± 0.22	81.41 ± 0.21	86.96 ± 0.43	**88.49 ± 0.24**
	LastFM	81.61 ± 3.82	83.02 ± 1.48	78.63 ± 0.31	81.45 ± 4.29	89.42 ± 0.07	73.53 ± 1.66	82.11 ± 0.42	**94.23 ± 0.09**	**96.25 ± 0.43**
	Enron	80.72 ± 1.39	74.55 ± 3.95	67.05 ± 1.51	77.94 ± 1.02	86.35 ± 0.51	76.14 ± 0.79	75.88 ± 0.48	**89.76 ± 0.34**	**89.17 ± 0.88**
	Social Evo.	**91.96 ± 0.48**	90.04 ± 0.47	91.41 ± 0.16	90.77 ± 0.86	79.94 ± 0.18	91.55 ± 0.09	91.86 ± 0.06	**93.14 ± 0.04**	91.71 ± 0.32
	UCI	79.86 ± 1.48	57.48 ± 1.87	79.54 ± 0.48	88.12 ± 2.05	92.73 ± 0.06	87.36 ± 2.03	91.19 ± 0.42	**94.54 ± 0.12**	**94.60 ± 0.41**
	Flights	94.74 ± 0.37	92.88 ± 0.73	88.73 ± 0.33	95.03 ± 0.60	97.06 ± 0.02	83.41 ± 0.07	83.03 ± 0.05	**97.79 ± 0.02**	**97.43 ± 0.15**
	Can. Parl.	53.92 ± 0.94	54.02 ± 0.76	55.18 ± 0.79	54.10 ± 0.93	55.80 ± 0.69	54.30 ± 0.66	55.91 ± 0.82	**87.74 ± 0.71**	**92.25± 0.24**
	US Legis.	54.93 ± 2.29	57.28 ± 0.71	51.00 ± 3.11	**58.63 ± 0.37**	53.17 ± 1.20	52.59 ± 0.97	50.71 ± 0.76	54.28 ± 2.87	**61.98 ± 1.31**
	UN Trade	59.65 ± 0.77	57.02 ± 0.69	61.03 ± 0.18	58.31 ± 3.15	**65.24 ± 0.21**	62.21 ± 0.12	62.17 ± 0.31	64.55 ± 0.62	**76.80 ± 2.01**
	UN Vote	56.64 ± 0.96	54.62 ± 2.22	52.24 ± 1.46	**58.85 ± 2.51**	49.94 ± 0.45	51.60 ± 0.97	50.68 ± 0.44	55.93 ± 0.39	**75.09 ± 0.34**
	Contact	94.34 ± 1.45	92.18 ± 0.41	95.87 ± 0.11	93.82 ± 0.99	89.55 ± 0.30	91.11 ± 0.12	90.59 ± 0.05	**98.03 ± 0.02**	**97.25 ± 0.07**
	Avg. Rank	5.69	6.84	6.30	4.69	4.69	6.79	6.15	**2.15**	**1.69**
ROC-AUC	Wikipedia	94.33 ± 0.27	91.49 ± 0.45	95.90 ± 0.09	97.72 ± 0.03	98.03 ± 0.04	95.57 ± 0.20	96.30 ± 0.04	**98.48 ± 0.03**	**99.30 ± 0.03**
	Reddit	96.52 ± 0.13	96.05 ± 0.12	96.98 ± 0.04	97.39 ± 0.07	98.42 ± 0.02	93.80 ± 0.07	94.97 ± 0.05	**98.71 ± 0.01**	**98.49 ± 0.06**
	MOOC	83.16 ± 1.30	84.03 ± 0.49	86.84 ± 0.17	**91.24 ± 0.99**	81.86 ± 0.25	81.43 ± 0.19	82.77 ± 0.24	87.62 ± 0.51	**90.63 ± 0.21**
	LastFM	81.13 ± 3.39	82.24 ± 1.51	76.99 ± 0.29	82.61 ± 3.15	87.82 ± 0.12	70.84 ± 0.85	80.37 ± 0.18	**94.08 ± 0.08**	**97.66 ± 0.21**
	Enron	81.96 ± 1.34	76.34 ± 4.20	64.63 ± 1.74	78.83 ± 1.11	87.02 ± 0.50	72.33 ± 0.99	76.51 ± 0.71	**90.69 ± 0.26**	**92.30 ± 0.57**
	Social Evo.	93.70 ± 0.29	91.18 ± 0.49	93.41 ± 0.19	93.43 ± 0.59	84.73 ± 0.27	93.71 ± 0.18	94.09 ± 0.07	**95.29 ± 0.03**	**94.09 ± 0.24**
	UCI	78.80 ± 0.94	58.08 ± 1.81	77.64 ± 0.38	86.68 ± 2.29	90.40 ± 0.11	84.49 ± 1.82	89.30 ± 0.57	**92.63 ± 0.13**	**95.88 ± 0.17**
	Flights	95.21 ± 0.32	93.56 ± 0.70	88.64 ± 0.35	95.92 ± 0.43	96.86 ± 0.02	82.48 ± 0.01	82.27 ± 0.06	**97.80 ± 0.02**	**98.46 ± 0.07**
	Can. Parl.	53.81 ± 1.14	55.27 ± 0.49	56.51 ± 0.75	55.86 ± 0.75	58.83 ± 1.13	55.83 ± 1.07	58.32 ± 1.08	**89.33 ± 0.48**	**95.06 ± 0.11**
	US Legis.	58.12 ± 2.35	61.07 ± 0.56	48.27 ± 3.50	**62.38 ± 0.48**	51.49 ± 1.13	50.43 ± 1.48	47.20 ± 0.89	53.21 ± 3.04	**66.18 ± 1.19**
	UN Trade	62.28 ± 0.50	58.82 ± 0.98	62.72 ± 0.12	59.99 ± 3.50	67.05 ± 0.21	63.76 ± 0.07	63.48 ± 0.37	**67.25 ± 1.05**	**83.95 ± 1.76**
	UN Vote	58.13 ± 1.43	55.13 ± 3.46	51.83 ± 1.35	**61.23 ± 2.71**	48.34 ± 0.76	50.51 ± 1.05	50.04 ± 0.86	56.73 ± 0.69	**82.01 ± 0.21**
	Contact	95.37 ± 0.92	91.89 ± 0.38	96.53 ± 0.10	94.84 ± 0.75	89.07 ± 0.34	93.05 ± 0.09	92.83 ± 0.05	**98.30 ± 0.02**	**97.99 ± 0.07**
	Avg. Rank	5.69	6.84	6.30	4.30	5.00	7.00	6.30	**2.23**	**1.30**

SimpleTLP surpasses the second-best results by a substantial margin.

Table 4.4: AP and ROC-AUC for *transductive setting* temporal link prediction with *historical negative sampling strategy*.

Metric	Datasets	JODIE	DyRep	TGAT	TGN	CAWN	TCL	GraphMixer	DyGFormer	SimpleTLP
AP	Wikipedia	83.01 ± 0.66	79.93 ± 0.56	87.38 ± 0.22	86.86 ± 0.33	71.21 ± 1.67	89.05 ± 0.39	**90.90 ± 0.10**	82.23 ± 2.54	**99.10 ± 0.51**
	Reddit	80.03 ± 0.36	79.83 ± 0.31	79.55 ± 0.20	81.22 ± 0.61	80.82 ± 0.45	77.14 ± 0.16	78.44 ± 0.18	**81.57 ± 0.67**	**93.25 ± 1.13**
	MOOC	78.94 ± 1.25	75.60 ± 1.12	82.19 ± 0.62	**87.06 ± 1.93**	74.05 ± 0.95	77.06 ± 0.41	77.77 ± 0.92	85.85 ± 0.66	**96.52 ± 1.58**
	LastFM	74.35 ± 3.81	74.92 ± 2.46	71.59 ± 0.24	76.87 ± 4.64	69.86 ± 0.43	59.30 ± 2.31	72.47 ± 0.49	**81.57 ± 0.48**	**94.88 ± 1.35**
	Enron	69.85 ± 2.70	71.19 ± 2.76	64.07 ± 1.05	73.91 ± 1.76	64.73 ± 0.36	70.66 ± 0.39	**77.98 ± 0.92**	75.63 ± 0.73	**95.42 ± 0.53**
	Social Evo.	87.44 ± 6.78	93.29 ± 0.43	**95.01 ± 0.44**	94.45 ± 0.56	85.53 ± 0.38	94.74 ± 0.31	94.93 ± 0.31	**97.38 ± 0.14**	90.19 ± 2.01
	UCI	75.24 ± 5.80	55.10 ± 3.14	68.27 ± 1.37	80.43 ± 2.12	65.30 ± 0.43	80.25 ± 2.74	**84.11 ± 1.35**	**82.17 ± 0.82**	80.76 ± 1.88
	Flights	66.48 ± 2.59	67.61 ± 0.99	**72.38 ± 0.18**	66.70 ± 1.64	64.72 ± 0.97	70.68 ± 0.24	71.47 ± 0.26	66.59 ± 0.49	**82.02 ± 2.39**
	Can. Parl.	51.79 ± 0.63	63.31 ± 1.23	67.13 ± 0.84	68.42 ± 3.07	66.53 ± 2.77	65.93 ± 3.00	74.34 ± 0.87	**97.00 ± 0.31**	**75.03 ± 1.76**
	US Legis.	51.71 ± 5.76	**86.88 ± 2.25**	62.14 ± 6.60	74.00 ± 7.57	68.82 ± 8.23	80.53 ± 3.95	81.65 ± 1.02	**85.30 ± 3.88**	62.66 ± 3.54
	UN Trade	61.39 ± 1.83	59.19 ± 1.07	55.74 ± 0.91	58.44 ± 5.51	55.71 ± 0.38	55.90 ± 1.17	57.05 ± 1.22	**64.41 ± 1.40**	**80.00 ± 1.55**
	UN Vote	**70.02 ± 0.81**	69.30 ± 1.12	52.96 ± 2.14	**69.37 ± 3.93**	51.26 ± 0.04	52.30 ± 2.35	51.20 ± 1.60	60.84 ± 1.58	61.43 ± 1.07
	Contact	95.31 ± 2.13	**96.39 ± 0.20**	96.05 ± 0.52	93.05 ± 2.35	84.16 ± 0.49	93.86 ± 0.21	93.36 ± 0.41	**97.57 ± 0.06**	80.24 ± 1.98
	Avg. Rank	5.85	5.31	5.54	4.23	7.77	5.92	4.46	**2.92**	**3.0**
ROC-AUC	Wikipedia	80.77 ± 0.73	77.74 ± 0.33	82.87 ± 0.22	82.74 ± 0.32	67.84 ± 0.64	85.76 ± 0.46	**87.68 ± 0.17**	78.80 ± 1.95	**99.21 ± 0.48**
	Reddit	80.52 ± 0.32	80.15 ± 0.18	79.33 ± 0.16	**81.11 ± 0.19**	80.27 ± 0.30	76.49 ± 0.16	77.80 ± 0.12	80.54 ± 0.29	**96.40 ± 0.59**
	MOOC	82.75 ± 0.83	81.06 ± 0.94	80.81 ± 0.67	**88.00 ± 1.80**	71.57 ± 1.07	72.09 ± 0.56	76.68 ± 1.40	87.04 ± 0.35	**97.69 ± 0.94**
	LastFM	75.22 ± 2.36	74.65 ± 1.98	64.27 ± 0.26	77.97 ± 3.04	67.88 ± 0.24	47.24 ± 3.13	64.21 ± 0.73	**78.78 ± 0.35**	**97.66 ± 1.96**
	Enron	75.39 ± 2.37	74.69 ± 3.55	61.85 ± 1.43	**77.09 ± 2.22**	65.10 ± 0.34	67.95 ± 0.88	75.27 ± 1.14	76.55 ± 0.52	**95.79 ± 0.33**
	Social Evo.	90.06 ± 3.15	93.12 ± 0.34	93.08 ± 0.59	**94.71 ± 0.53**	87.43 ± 0.15	93.44 ± 0.68	94.39 ± 0.31	**97.28 ± 0.07**	93.60 ± 2.27
	UCI	**78.64 ± 3.50**	57.91 ± 3.12	58.89 ± 1.57	77.25 ± 2.68	57.86 ± 0.15	72.25 ± 3.46	77.54 ± 2.02	76.97 ± 0.24	**90.59 ± 1.60**
	Flights	68.97 ± 1.87	69.43 ± 0.90	**72.20 ± 0.16**	68.39 ± 0.95	66.11 ± 0.71	70.57 ± 0.18	70.37 ± 0.23	68.09 ± 0.43	**92.84 ± 1.21**
	Can. Parl.	62.44 ± 1.11	70.16 ± 1.70	70.86 ± 0.94	73.23 ± 3.08	72.06 ± 3.94	69.95 ± 3.70	79.03 ± 1.01	**97.61 ± 0.40**	**87.22 ± 1.85**
	US Legis.	67.47 ± 6.40	**91.44 ± 1.18**	73.47 ± 5.25	83.53 ± 4.53	78.62 ± 7.46	83.97 ± 3.71	85.17 ± 0.70	**90.77 ± 1.96**	76.64 ± 3.38
	UN Trade	68.92 ± 1.40	64.36 ± 1.40	60.37 ± 0.68	63.93 ± 5.41	63.09 ± 0.74	61.43 ± 1.04	63.20 ± 1.54	**73.86 ± 1.13**	**87.67 ± 1.80**
	UN Vote	**76.84 ± 1.01**	**74.72 ± 1.43**	53.95 ± 3.15	73.40 ± 5.20	51.27 ± 0.33	52.29 ± 2.39	52.61 ± 1.44	64.27 ± 1.78	72.92 ± 1.73
	Contact	**96.35 ± 0.92**	96.00 ± 0.23	95.39 ± 0.43	93.76 ± 1.29	83.06 ± 0.32	93.34 ± 0.19	93.14 ± 0.34	**97.17 ± 0.05**	90.99 ± 1.80
	Avg. Rank	4.77	5.08	6.31	3.77	7.70	6.46	5.08	**3.31**	**2.54**

Table 4.5: AP and ROC-AUC for *inductive setting* temporal link prediction with *historical negative sampling strategy*.

Metric	Datasets	JODIE	DyRep	TGAT	TGN	CAWN	TCL	GraphMixer	DyGFormer	SimpleTLP
AP	Wikipedia	68.69 ± 0.39	62.18 ± 1.27	84.17 ± 0.22	81.76 ± 0.32	67.27 ± 1.63	82.20 ± 2.18	**87.60 ± 0.30**	71.42 ± 4.43	**98.45 ± 0.88**
	Reddit	62.34 ± 0.54	61.60 ± 0.72	63.47 ± 0.36	64.85 ± 0.85	63.67 ± 0.41	60.83 ± 0.25	64.50 ± 0.26	**65.37 ± 0.60**	**85.74 ± 2.46**
	MOOC	63.22 ± 1.55	62.93 ± 1.24	76.73 ± 0.29	77.07 ± 3.41	74.68 ± 0.68	74.27 ± 0.53	74.00 ± 0.97	**80.82 ± 0.30**	**95.17 ± 2.78**
	LastFM	70.39 ± 4.31	71.45 ± 1.76	76.27 ± 0.25	66.65 ± 6.11	71.33 ± 0.47	65.78 ± 0.65	**76.42 ± 0.22**	76.35 ± 0.52	**92.96 ± 1.08**
	Enron	65.86 ± 3.71	62.08 ± 2.27	61.40 ± 1.31	62.91 ± 1.16	60.70 ± 0.36	67.11 ± 0.62	**72.37 ± 1.37**	67.07 ± 0.62	**89.37 ± 1.01**
	Social Evo.	88.51 ± 0.87	88.72 ± 1.10	93.97 ± 0.54	90.66 ± 1.62	79.83 ± 0.38	**94.10 ± 0.31**	94.01 ± 0.47	**96.82 ± 0.16**	85.13 ± 1.05
	UCI	63.11 ± 2.27	52.47 ± 2.06	70.52 ± 0.93	70.78 ± 0.78	64.54 ± 0.47	**76.71 ± 1.00**	**81.66 ± 0.49**	72.13 ± 1.87	75.97 ± 1.13
	Flights	61.01 ± 1.65	62.83 ± 1.31	64.72 ± 0.36	59.31 ± 1.43	56.82 ± 0.57	64.50 ± 0.25	**65.28 ± 0.24**	57.11 ± 0.21	**71.54 ± 2.12**
	Can. Parl.	52.60 ± 0.88	52.28 ± 0.31	56.72 ± 0.47	54.42 ± 0.77	57.14 ± 0.07	55.71 ± 0.74	55.84 ± 0.73	**87.40 ± 0.85**	**68.91 ± 1.63**
	US Legis.	52.94 ± 2.11	**62.10 ± 1.41**	51.83 ± 3.95	**61.18 ± 1.10**	55.56 ± 1.71	53.87 ± 1.41	52.03 ± 1.02	56.31 ± 3.46	49.51 ± 1.09
	UN Trade	55.46 ± 1.19	55.49 ± 0.84	55.28 ± 0.71	52.80 ± 3.19	55.00 ± 0.38	**55.76 ± 1.03**	54.94 ± 0.97	53.20 ± 1.07	**79.09 ± 1.86**
	UN Vote	61.04 ± 1.30	60.22 ± 1.78	53.05 ± 3.10	**63.74 ± 3.00**	47.98 ± 0.84	54.19 ± 2.17	48.09 ± 0.43	52.63 ± 1.26	**61.89 ± 1.28**
	Contact	90.42 ± 2.34	89.22 ± 0.66	**94.15 ± 0.45**	88.13 ± 1.50	74.20 ± 0.80	90.44 ± 0.17	89.91 ± 0.36	**93.56 ± 0.52**	79.08 ± 2.24
	Avg. Rank	6.15	6.23	4.77	5.23	6.85	4.62	4.23	**3.92**	**3.0**
ROC-AUC	Wikipedia	61.86 ± 0.53	57.54 ± 1.09	78.38 ± 0.20	75.75 ± 0.29	62.04 ± 0.65	79.79 ± 0.96	**82.87 ± 0.21**	68.33 ± 2.82	**98.59 ± 0.82**
	Reddit	61.69 ± 0.39	60.45 ± 0.37	64.43 ± 0.27	64.55 ± 0.50	**64.94 ± 0.21**	61.43 ± 0.26	64.27 ± 0.13	64.81 ± 0.25	**92.04 ± 1.87**
	MOOC	64.48 ± 1.64	64.23 ± 1.29	74.08 ± 0.27	77.69 ± 3.55	71.68 ± 0.94	69.82 ± 0.32	72.53 ± 0.84	**80.77 ± 0.63**	**96.85 ± 1.59**
	LastFM	68.44 ± 3.26	68.79 ± 1.08	69.89 ± 0.28	66.99 ± 5.62	67.69 ± 0.24	55.88 ± 1.85	70.07 ± 0.20	**70.73 ± 0.37**	**96.34 ± 1.68**
	Enron	65.32 ± 3.57	61.50 ± 2.50	57.84 ± 2.18	62.68 ± 1.09	62.25 ± 0.40	64.06 ± 1.02	**68.20 ± 1.62**	65.78 ± 0.42	**92.14 ± 0.70**
	Social Evo.	88.53 ± 0.55	87.93 ± 1.05	91.87 ± 0.72	92.10 ± 1.22	83.54 ± 0.24	93.28 ± 0.60	**93.62 ± 0.35**	**96.91 ± 0.09**	90.08 ± 2.91
	UCI	60.24 ± 1.94	51.25 ± 2.37	62.32 ± 1.18	62.69 ± 0.90	56.39 ± 0.10	70.46 ± 1.94	**75.98 ± 0.84**	65.55 ± 1.01	**84.63 ± 1.23**
	Flights	60.72 ± 1.29	61.99 ± 1.39	63.38 ± 0.26	59.66 ± 1.04	56.58 ± 0.44	**63.48 ± 0.23**	63.30 ± 0.19	56.05 ± 0.21	**85.60 ± 1.47**
	Can. Parl.	51.62 ± 1.00	52.38 ± 0.46	58.30 ± 0.61	55.64 ± 0.54	60.11 ± 0.48	57.30 ± 1.03	56.68 ± 1.20	**88.68 ± 0.74**	80.73 ± 1.09
	US Legis.	58.12 ± 2.94	**67.94 ± 0.98**	49.99 ± 4.88	**64.87 ± 1.65**	54.41 ± 1.31	52.12 ± 2.13	49.28 ± 0.86	56.57 ± 3.22	55.74 ± 1.53
	UN Trade	58.73 ± 1.19	57.90 ± 1.33	59.74 ± 0.59	55.61 ± 3.54	60.95 ± 0.80	**61.12 ± 0.97**	59.88 ± 1.17	58.46 ± 1.65	**86.26 ± 1.78**
	UN Vote	65.16 ± 1.28	63.98 ± 2.12	51.73 ± 4.12	**68.59 ± 3.11**	48.01 ± 1.77	54.66 ± 2.11	45.49 ± 0.42	53.85 ± 2.02	**72.98 ± 1.22**
	Contact	90.80 ± 1.18	88.88 ± 0.68	**93.76 ± 0.41**	88.84 ± 1.39	74.79 ± 0.37	90.37 ± 0.16	90.04 ± 0.29	**94.14 ± 0.26**	88.69 ± 2.24
	Avg. Rank	5.92	6.85	5.08	5.31	6.38	4.85	4.54	**3.77**	**2.31**

Table 4.6: AP and ROC-AUC for *transductive setting* temporal link prediction with *inductive negative sampling strategy.*

Metric	Datasets	JODIE	DyRep	TGAT	TGN	CAWN	TCL	GraphMixer	DyGFormer	SimpleTLP
AP	Wikipedia	75.65 ± 0.79	70.21 ± 1.58	87.00 ± 0.16	85.62 ± 0.44	74.06 ± 2.62	86.76 ± 0.72	**88.59 ± 0.17**	78.29 ± 5.38	**98.66 ± 0.94**
	Reddit	86.98 ± 0.16	86.30 ± 0.26	89.59 ± 0.24	88.10 ± 0.24	**91.67 ± 0.24**	87.45 ± 0.29	85.26 ± 0.11	91.11 ± 0.40	**98.15 ± 0.48**
	MOOC	65.23 ± 2.19	61.66 ± 0.95	75.95 ± 0.64	77.50 ± 2.91	73.51 ± 0.94	74.65 ± 0.54	74.27 ± 0.92	**81.24 ± 0.69**	**97.49 ± 0.99**
	LastFM	62.67 ± 4.49	64.41 ± 2.70	71.13 ± 0.17	65.95 ± 5.98	67.48 ± 0.77	58.21 ± 0.89	68.12 ± 0.33	**73.97 ± 0.50**	**93.92 ± 1.10**
	Enron	68.96 ± 0.98	67.79 ± 1.53	63.94 ± 1.36	70.89 ± 2.72	75.15 ± 0.58	71.29 ± 0.32	75.01 ± 0.79	**77.41 ± 0.89**	**98.98 ± 0.53**
	Social Evo.	89.82 ± 4.11	93.28 ± 0.48	94.84 ± 0.44	**95.13 ± 0.56**	88.32 ± 0.27	94.90 ± 0.36	94.72 ± 0.33	**97.68 ± 0.10**	90.53 ± 1.88
	UCI	65.99 ± 1.40	54.79 ± 1.76	68.67 ± 0.84	70.94 ± 0.71	64.61 ± 0.48	**76.01 ± 1.11**	**80.10 ± 0.51**	72.25 ± 1.71	74.08 ± 1.84
	Flights	69.07 ± 4.02	70.57 ± 1.82	**75.48 ± 0.26**	71.09 ± 2.72	69.18 ± 1.52	74.62 ± 0.18	74.87 ± 0.21	70.92 ± 1.78	**80.75 ± 1.51**
	Can. Parl.	48.42 ± 0.66	58.61 ± 0.86	68.82 ± 1.21	65.34 ± 2.87	67.75 ± 1.00	65.85 ± 1.75	69.48 ± 0.63	**95.44 ± 0.57**	**73.79 ± 1.26**
	US Legis.	50.27 ± 5.13	**83.44 ± 1.16**	61.91 ± 5.82	67.57 ± 6.47	65.81 ± 8.52	78.15 ± 3.34	79.63 ± 0.84	**81.25 ± 3.62**	57.67 ± 3.10
	UN Trade	60.42 ± 1.48	60.19 ± 1.24	60.61 ± 1.24	61.04 ± 6.01	**62.54 ± 0.67**	61.06 ± 1.74	60.15 ± 1.29	55.79 ± 1.02	**80.96 ± 1.12**
	UN Vote	**67.79 ± 1.46**	67.53 ± 1.98	52.89 ± 1.61	**67.63 ± 2.67**	52.19 ± 0.34	50.62 ± 0.82	51.60 ± 0.73	51.91 ± 0.84	59.42 ± 3.73
	Contact	93.43 ± 1.78	94.18 ± 0.10	**94.35 ± 0.48**	90.18 ± 3.28	89.31 ± 0.27	91.35 ± 0.21	90.87 ± 0.35	**94.75 ± 0.28**	80.15 ± 1.48
	Avg. Rank	6.92	6.54	4.46	4.77	5.92	5.0	4.77	**3.54**	**3.08**
ROC-AUC	Wikipedia	70.96 ± 0.78	67.36 ± 0.96	81.93 ± 0.22	80.97 ± 0.31	70.95 ± 0.95	82.19 ± 0.48	**84.28 ± 0.30**	75.09 ± 3.70	**98.78 ± 0.91**
	Reddit	83.51 ± 0.15	82.90 ± 0.31	87.13 ± 0.20	84.56 ± 0.24	**88.04 ± 0.29**	84.67 ± 0.29	82.21 ± 0.13	86.23 ± 0.51	**98.51 ± 0.40**
	MOOC	66.63 ± 2.30	63.26 ± 1.01	73.18 ± 0.33	77.44 ± 2.86	70.32 ± 1.43	70.36 ± 0.37	72.45 ± 0.72	**80.76 ± 0.76**	**98.31 ± 0.59**
	LastFM	61.32 ± 3.49	62.15 ± 2.12	63.99 ± 0.21	65.46 ± 4.27	67.92 ± 0.44	46.93 ± 2.59	60.22 ± 0.32	69.25 ± 0.36	**97.08 ± 1.47**
	Enron	70.92 ± 1.05	68.73 ± 1.34	60.45 ± 2.12	71.34 ± 2.46	**75.17 ± 0.50**	67.64 ± 0.86	71.53 ± 0.85	74.07 ± 0.64	**95.79 ± 0.33**
	Social Evo.	90.01 ± 3.19	93.07 ± 0.38	92.94 ± 0.61	**95.24 ± 0.56**	89.93 ± 0.15	93.44 ± 0.72	94.22 ± 0.32	**97.51 ± 0.06**	93.73 ± 2.22
	UCI	64.14 ± 1.26	54.25 ± 2.01	60.80 ± 1.01	64.11 ± 1.04	58.06 ± 0.26	70.05 ± 1.86	**74.59 ± 0.74**	65.96 ± 1.18	**84.09 ± 1.61**
	Flights	69.99 ± 3.10	71.13 ± 1.55	**73.47 ± 0.18**	71.63 ± 1.72	69.70 ± 0.75	72.54 ± 0.19	72.21 ± 0.21	69.53 ± 1.17	**92.32 ± 1.51**
	Can. Parl.	52.88 ± 0.80	63.53 ± 0.65	72.47 ± 1.18	69.57 ± 2.81	**72.93 ± 1.78**	69.47 ± 2.12	70.52 ± 0.94	**96.70 ± 0.59**	85.80 ± 1.51
	US Legis.	59.05 ± 5.52	**89.44 ± 0.71**	71.62 ± 5.42	78.12 ± 4.46	76.45 ± 7.02	82.54 ± 3.91	84.22 ± 0.91	**87.96 ± 1.80**	70.81 ± 3.72
	UN Trade	66.82 ± 1.27	65.60 ± 1.28	66.13 ± 0.78	66.37 ± 5.39	**71.73 ± 0.74**	67.80 ± 1.21	66.53 ± 1.22	62.56 ± 1.51	**87.92 ± 1.61**
	UN Vote	**73.73 ± 1.61**	**72.80 ± 2.16**	53.04 ± 2.58	72.69 ± 3.72	52.75 ± 0.90	52.02 ± 1.64	51.89 ± 0.74	53.37 ± 1.26	70.70 ± 3.58
	Contact	**94.47 ± 1.08**	94.23 ± 0.18	94.10 ± 0.41	91.64 ± 1.72	87.68 ± 0.24	91.23 ± 0.19	90.96 ± 0.27	**95.01 ± 0.15**	90.68 ± 1.59
	Avg. Rank	6.15	6.31	5.31	4.69	5.69	5.38	5.08	**3.77**	**2.62**

Table 4.7: AP and ROC-AUC for *inductive setting* temporal link prediction with *inductive negative sampling strategy.*

Metric	Datasets	JODIE	DyRep	TGAT	TGN	CAWN	TCL	GraphMixer	DyGFormer	SimpleTLP
ind	Wikipedia	68.70 ± 0.39	62.19 ± 1.28	84.17 ± 0.22	81.77 ± 0.32	67.24 ± 1.63	82.20 ± 2.18	**87.60 ± 0.29**	71.42 ± 4.43	**98.45 ± 0.88**
	Reddit	62.32 ± 0.54	61.58 ± 0.72	63.40 ± 0.36	64.84 ± 0.84	63.65 ± 0.41	60.81 ± 0.26	64.49 ± 0.25	**65.35 ± 0.60**	**85.76 ± 2.46**
	MOOC	63.22 ± 1.55	62.92 ± 1.24	76.72 ± 0.30	77.07 ± 3.40	74.69 ± 0.68	74.28 ± 0.53	73.99 ± 0.97	**80.82 ± 0.30**	**95.17 ± 2.78**
	LastFM	70.39 ± 4.31	71.45 ± 1.75	76.28 ± 0.25	69.46 ± 4.65	71.33 ± 0.47	65.78 ± 0.65	**76.42 ± 0.22**	76.35 ± 0.52	**92.96 ± 1.08**
	Enron	65.86 ± 3.71	62.08 ± 2.27	61.40 ± 1.30	62.90 ± 1.16	60.72 ± 0.36	67.11 ± 0.62	**72.37 ± 1.38**	67.07 ± 0.62	**89.36 ± 1.01**
	Social Evo.	88.51 ± 0.87	88.72 ± 1.10	93.97 ± 0.54	90.65 ± 1.62	79.83 ± 0.39	**94.10 ± 0.32**	94.01 ± 0.47	**96.82 ± 0.17**	85.13 ± 1.05
	UCI	63.16 ± 2.27	52.47 ± 2.09	70.49 ± 0.93	70.73 ± 0.79	64.54 ± 0.47	**76.65 ± 0.99**	**81.64 ± 0.49**	72.13 ± 1.86	75.97 ± 1.13
	Flights	61.01 ± 1.66	62.83 ± 1.31	64.72 ± 0.37	59.32 ± 1.45	56.82 ± 0.56	64.50 ± 0.25	**65.29 ± 0.24**	57.11 ± 0.20	**69.14 ± 1.19**
	Can. Parl.	52.58 ± 0.86	52.24 ± 0.28	56.46 ± 0.50	54.18 ± 0.73	57.06 ± 0.08	55.46 ± 0.69	55.76 ± 0.65	**87.22 ± 0.82**	**69.59 ± 1.45**
	US Legis.	52.94 ± 2.11	**62.10 ± 1.41**	51.83 ± 3.95	**61.18 ± 1.10**	55.56 ± 1.71	53.87 ± 1.41	52.03 ± 1.02	56.31 ± 3.46	49.51 ± 1.09
	UN Trade	55.43 ± 1.20	55.42 ± 0.87	55.58 ± 0.68	52.80 ± 3.24	54.97 ± 0.38	**55.66 ± 0.98**	54.88 ± 1.01	52.56 ± 1.70	**79.12 ± 1.84**
	UN Vote	61.17 ± 1.33	60.29 ± 1.79	53.08 ± 3.10	**63.71 ± 2.97**	48.01 ± 0.82	54.13 ± 2.16	48.10 ± 0.40	52.61 ± 1.25	**61.88 ± 1.27**
	Contact	90.43 ± 2.33	89.22 ± 0.65	**94.14 ± 0.45**	88.12 ± 1.50	74.19 ± 0.81	90.43 ± 0.17	89.91 ± 0.36	**93.55 ± 0.52**	79.09 ± 1.25
	Avg. Rank	6.15	6.38	4.62	5.15	6.85	4.62	4.23	**4.0**	**3.0**
ind	Wikipedia	61.87 ± 0.53	57.54 ± 1.09	78.38 ± 0.20	75.76 ± 0.29	62.02 ± 0.65	79.79 ± 0.96	**82.88 ± 0.21**	68.33 ± 2.82	**98.59 ± 0.82**
	Reddit	61.69 ± 0.39	60.44 ± 0.37	64.39 ± 0.27	64.55 ± 0.50	**64.91 ± 0.21**	61.36 ± 0.26	64.27 ± 0.13	64.80 ± 0.25	**92.05 ± 1.87**
	MOOC	64.48 ± 1.64	64.22 ± 1.29	74.07 ± 0.27	77.68 ± 3.55	71.69 ± 0.94	69.83 ± 0.32	72.52 ± 0.84	**80.77 ± 0.63**	**96.85 ± 1.59**
	LastFM	68.44 ± 3.26	68.79 ± 1.08	69.89 ± 0.28	66.99 ± 5.61	67.68 ± 0.24	55.88 ± 1.85	70.07 ± 0.20	**70.73 ± 0.37**	**96.34 ± 1.68**
	Enron	65.32 ± 3.57	61.50 ± 2.50	57.83 ± 2.18	62.68 ± 1.09	62.27 ± 0.40	64.05 ± 1.02	**68.19 ± 1.63**	65.79 ± 0.42	**92.14 ± 0.70**
	Social Evo.	88.53 ± 0.55	87.93 ± 1.05	91.88 ± 0.72	92.10 ± 1.22	83.54 ± 0.24	93.28 ± 0.60	**93.62 ± 0.35**	**96.91 ± 0.09**	90.08 ± 3.91
	UCI	60.27 ± 1.94	51.26 ± 2.40	62.29 ± 1.17	62.66 ± 0.91	56.39 ± 0.11	70.42 ± 1.93	**75.97 ± 0.85**	65.58 ± 1.00	**84.63 ± 1.22**
	Flights	60.72 ± 1.29	61.99 ± 1.39	63.40 ± 0.26	59.66 ± 1.05	56.58 ± 0.44	**63.49 ± 0.23**	63.32 ± 0.19	56.05 ± 0.22	**83.87 ± 1.30**
	Can. Parl.	51.61 ± 0.98	52.35 ± 0.52	58.15 ± 0.62	55.43 ± 0.42	60.01 ± 0.47	56.88 ± 0.93	56.63 ± 1.09	**88.51 ± 0.73**	**81.26 ± 3.92**
	US Legis.	58.12 ± 2.94	**67.94 ± 0.98**	49.99 ± 4.88	**64.87 ± 1.65**	54.41 ± 1.31	52.12 ± 2.13	49.28 ± 0.86	56.57 ± 3.22	55.74 ± 1.53
	UN Trade	58.71 ± 1.20	57.87 ± 1.36	59.98 ± 0.59	55.62 ± 3.59	60.88 ± 0.79	**61.01 ± 0.93**	59.71 ± 1.17	57.28 ± 3.06	**86.28 ± 1.76**
	UN Vote	65.29 ± 1.30	64.10 ± 2.10	51.78 ± 4.14	**68.58 ± 3.08**	48.04 ± 1.76	54.65 ± 2.20	45.57 ± 0.41	53.87 ± 2.01	**72.98 ± 1.22**
	Contact	90.80 ± 1.18	88.87 ± 0.67	**93.76 ± 0.40**	88.85 ± 1.39	74.79 ± 0.38	90.37 ± 0.16	90.04 ± 0.29	**94.14 ± 0.26**	88.69 ± 3.25
	Avg. Rank	5.92	6.77	5.0	5.31	6.38	4.85	4.62	**3.85**	**2.31**

4.1.5 Summary

Link prediction is of key importance in many real-world applications like social network analysis and recommender systems. To achieve state-of-the-art performance, many recent works tend to adapt the attention mechanism to the structured data for link prediction, in which dense or relational attention is often unaffordable to large-scale structured data. Moreover, in the realistic setting, the time-evolving topological and feature information can bring more challenging questions to the efficiency and effectiveness of structured attention mechanisms. We discern that the attention mechanism may not always be as necessary as expected for temporal graph representation learning, at least not for temporal link prediction tasks. Formally, we discover that simple positional encoding plus MLP can achieve SOTA performance than graph transformers. Hence, we propose a simple temporal link prediction model, named SimpleTLP. In detail, for SimpleTLP, we propose the Fourier Transform on temporal graphs for learning informative positional encodings, prove they can preserve the temporal graph topology from the spatial-temporal spectral viewpoint, and demonstrate its empirical outperformance in a comprehensive setting of 13 datasets and 9 baseline algorithms in both transductive and inductive settings with 3 different sampling strategies.

4.2 LOCAL HIGH-ORDER GRAPH CLUSTERING VIA NATURAL + ARTIFICIAL DYNAMICS

4.2.1 Overview and Motivation

Local clustering aims to identify a qualified local cluster near a given seed node. Previously, most existing local clustering methods [161, 162] take the connectivity of low-order structures (i.e., edges) as the clustering metric. In other words, they largely ignore the connection of high-order subgraph patterns, i.e., motifs [163] (Figure 4.2(a) illustrates a few motifs), which might be of key importance in real applications [164]. For example, triangle motifs may help identify frequent flight patterns in air traffic networks [165]; star motifs may indicate synthetic identities in social networks [166]; loop motifs may be associated with money laundering in financial networks [167]; user-specified complex protein-pair motifs can be helpful for better understanding the functional organization of the proteome in protein-protein interaction networks [168]. To address this problem, local motif clustering algorithms [167, 169] have been proposed to find a cluster with densely connected motifs in the neighborhood of the seed node. Figure 4.2(b) illustrates the difference between the traditional edge-based local clustering method [162] and the emerging local motif clustering

method [170], by setting the triangle as the target motif being preserved from cut, the local cluster identified by the local motif clustering method is much more densely connected by the triangle motifs.

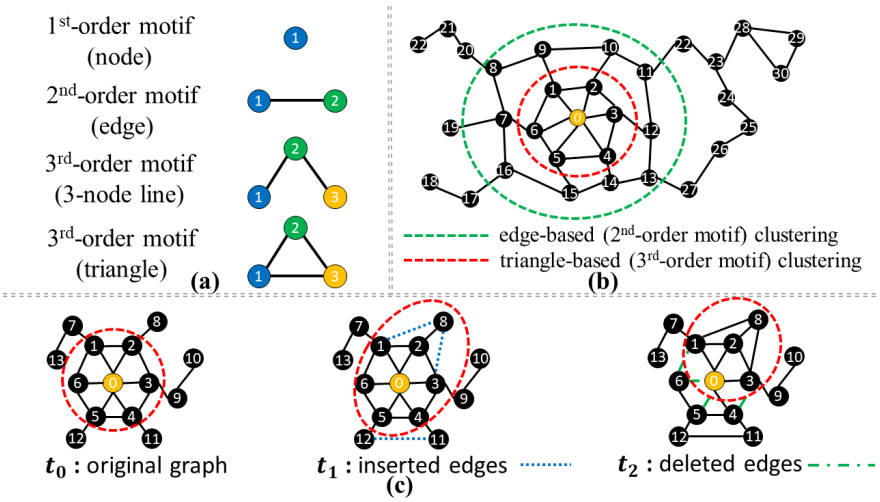


Figure 4.2: An overview of local motif clustering on time-evolving graphs. Part (a) illustrates the different order motif structures; Part (b) shows the difference between second-order and third-order local clustering on a network example given the yellow seed node; Part (c) shows the evolution pattern of the triangle-based local cluster (red dash line) on the time-evolving graphs for inserted edges and deleted edges.

However, most existing local motif clustering techniques [169, 170] are designed for the static setting. On time-evolving graphs, repeatedly applying these techniques at each timestamp from scratch would be computationally prohibitive. To bridge this gap, we propose a novel local motif clustering framework for time-evolving graphs named L-MEGA. Figure 4.2(c) illustrates the triangle-based local cluster at each timestamp as the graph evolves over time. The core of L-MEGA is approximately tracking the stationary distribution of high-order random walk [171] among timestamps, then input the tracked stationary distribution into a vector-based sweep cut procedure [161] to get a local cluster with a small motif conductance score. L-MEGA consists of three parts, and we design speeding-up strategies in each part. The first part is updating transition tensors for representing the transition probability for high-order random walk process [172], the second part is tracking the stationary probability distribution at each timestamp, and the third part is using the tracked stationary distribution to incrementally get a qualified local cluster.

4.2.2 Problem Definition

A time-evolving graph can be naturally presented as a collection of graph snapshots over time, i.e., $\widetilde{G} = \{G^{(0)}, G^{(1)}, \ldots, G^{(T)}\}$, where $G^{(t)} = (V^{(t)}, E^{(t)})$, $t = 1, \ldots, T$. We let $V^{(t)}$ and $E^{(t)}$ denote the sets of nodes and edges in $G^{(t)}$, $\mathbf{d}^{(t)} \in \mathbb{R}^{n^{(t)}}$ denote the degree vector of $G^{(t)}$, and $d^{(t)}(v)$ denote the specific degree of vertex $v \in V^{(t)}$. Following [173], we assume the number of nodes is fixed, i.e., an inserted (or deleted) node at timestamp t is regarded as an existed dangling node at all previous timestamps (or all future timestamps), such that the size of adjacency matrix $A^{(t)}$ of each snapshot $G^{(t)}$ are consistent over time. On top of that, we introduce the notion of updated edge set $\Delta E^{(t)} = \Delta E^{(t)}_{+} \cup \Delta E^{(t)}_{-}$ to simplify the description of our algorithm, where $\Delta E^{(t)}_{+} = \{E^{(t+1)} \setminus E^{(t)}\}$ and $\Delta E^{(t)}_{-} = \{E^{(t)} \setminus E^{(t+1)}\}$ are the sets of inserted edges and deleted edges at timestamp t to transform graph $G^{(t)}$ to $G^{(t+1)}$.

In the context of classic local motif clustering [167, 169], we aim to identify a local dense cluster C which sits near the given seed node v and preserves rich user-defined motifs $\mathbb{N}$. We let k denote the order of the user-defined motif $\mathbb{N}$, and k refers to the number of vertices involved in the motif $\mathbb{N}$. For example, an edge is a second-order motif, a three-node line and a triangle can be classified as the third-order motif. The quality of the identified motif cluster is measured by motif conductance [167, 169], small motif conductance indicates a good partition. For any local cluster C and user-defined motif $\mathbb{N}$, the corresponding motif conductance $\Phi(C, \mathbb{N})$ is defined as

$$\Phi(C, \mathbb{N}) = \frac{cut(C, \mathbb{N})}{min\{\mu(C, \mathbb{N}), \mu(\bar{C}, \mathbb{N})\}} \tag{4.14}$$

where $cut(C, \mathbb{N})$ denotes the number of motif structures broken due to the partition of the graph G into local cluster C and its complement $\bar{C}$, $\mu(C, \mathbb{N})$ and $\mu(\bar{C}, \mathbb{N})$ denote the number of motif structures $\mathbb{N}$ within C and $\bar{C}$, respectively. Here we generalize the local motif clustering problem to the dynamic setting. Given a time-evolving graph $\widetilde{G}$, our goal is minimizing motif conductance to obtain a good local cluster $C^{(t)}$ over time.

$$\min_{C^{(1)}, \ldots, C^{(T)}} \sum_{t=1}^{T} \Phi(C^{(t)}, \mathbb{N}) \tag{4.15}$$

With the above notations, we formally define our problem as below.

Problem 4.1. Local Motif Clustering on Time-Evolving Graphs

Input: (i) a user-defined motif $\mathbb{N}$, (ii) a sequence of snapshots of the time-evolving graph $\widetilde{G} = \{G^{(0)}, G^{(1)}, \ldots, G^{(T)}\}$, (iii) the seed node v, and (iv) the motif conductance upper

bound ϕ.

Output: a local cluster $C^{(t)}$ near seed node v such that $\Phi(C^{(t)}, \mathbb{N}) \leq \phi$ at each timestamp $t \in \{1, 2, \ldots, T\}$.

4.2.3 Preliminary

PageRank and Multilinear PageRank Given a random walk on the graph, the PageRank framework [174] models a stochastic process with a unique stationary distribution, which is the PageRank vector solving the following linear system:

$$\mathbf{x} = \alpha \mathbf{P}\mathbf{x} + (1 - \alpha)\mathbf{u} \tag{4.16}$$

where $\mathbf{x} \in \mathbb{R}^n$ is the PageRank vector, $\mathbf{P} \in \mathbb{R}^{n \times n}$ is a stochastic column matrix encoding the transition probability for the random walk, and $\mathbf{u} \in \mathbb{R}^n$ is the stochastic vector.

Multilinear PageRank framework [171] is proposed based on the spacey random walk [172] which is a high-order random walk developed on the high-order Markov Chain stochastic process. For example, in the second-order Markov Chain, the next state of the discrete-time stochastic process only depends on the past two states; the probability is expressed $Prob(S_{t+1}|S_t, S_{t-1})$. With the rank-1 approximation [175] of high-order Markov Chain, authors in [171] encode $(k-1)^{th}$-order Markov Chain into a kth-order probability transition tensor, and then propose the multilinear PageRank. For example, in modeling third-order data, the stationary distribution (i.e., multilinear PageRank vector) solves the following equation:

$$\mathbf{x} = \alpha \bar{\mathbf{P}}(\mathbf{x} \otimes \mathbf{x}) + (1 - \alpha)\mathbf{u} \tag{4.17}$$

where $\mathbf{x} \in \mathbb{R}^n$ is the multilinear PageRank vector, $\otimes$ denotes the Kronecker product, the third-order transition tensor $\bar{\mathbf{P}} \in \mathbb{R}^{n \times n \times n}$ encodes the transition probability for second-order Markov Chain (random walk), and $\mathbf{u} \in \mathbb{R}^n$ is the stochastic vector.

Note that, the order of data is different from the order of Markov Chain (random walk). In general, k^{th}-order motif data correspond to the $(k-1)^{th}$-order Markov chain. Encoding k^{th}-order motif data into the kth-order probability transition tensor (i.e., $(k-1)^{th}$-order Markov Chain) will be introduced in the next subsection. Moreover, for the tensor-vector multiplication operation, we define the one-mode unfolding operation of tensors. For example, a three-mode (or third-order) tensor $\bar{\mathbf{P}} \in \mathbb{R}^{n \times n \times n}$ is unfolded as the matrix $\mathbf{P} = [\bar{P}(:,:,1), \bar{P}(:,:,2), \ldots, \bar{P}(:,:,n)]$, where $\mathbf{P} \in \mathbb{R}^{n \times n^2}$ and $\bar{P}(:,:,i)$ is the i-th $n \times n$ block of the tensor $\bar{\mathbf{P}}$.

Approximated PageRank for Local Clustering The problem of identifying a local cluster with minimum conductance has been proven to be NP-complete [176]. To mitigate the intractable complexity, a series of PageRank-based approximation methods [161, 162] have been developed in the past. The key idea behind these approaches is to model the stochastic process of random walks on graphs and then conduct a vector-based sweep-cut procedure on the obtained PageRank vector to identify local clusters.

Solving the PageRank vector (i.e., stationary distribution) through the power iteration method [174] is time-consuming, which iterates Eq. 4.16 until convergence requiring time complexity $O(m)$, where m is the number of edges of the graph. Instead, using the approximated PageRank vector, PageRank-Nibble [161] could obtain a local cluster with conductance ϕ in time complexity $O(2^b \log^3 m/\phi^2)$, where b is the constant parameter. First, PageRank-Nibble [161] solves the approximated PageRank vector through the push operation. Push operation maintains a residual vector to record the divergence between the approximated PageRank vector and the PageRank vector. After sufficient iterations of moving probability mass to the approximated PageRank vector from the residual vector, the push operation obtains the converged approximated stationary distribution as the clustering indicator vector. Second, PageRank-Nibble feeds the approximated PageRank into the sweep cut procedure to obtain the qualified local cluster satisfying statistical conditions.

More recently, a surge of research interest in motif clustering has been observed in the data mining community [167, 169, 177, 178, 179]. To indicate a local cluster preserving rich user-defined high-order motifs, (approximated) multilinear PageRank vector [171] has been proven to be effective in many high-order clustering applications [167, 177, 178] as the clustering indicator vector.

Inspired by local motif clustering method [167], we extend the statistical conditions of sweep cut [161] to high-order and dynamic settings, such that we can identify a qualified motif cluster on the approximated multilinear PageRank vector at each timestamp. For notation simplicity, we ignore the superscript of timestamps below.

(C.1) $\Phi(S_j(\mathbf{x}), \mathbb{N}) \leq \phi$ /*motif conductance check*/

(C.2) $\lambda_j(\mathbf{x}) \leq (2/3)\mu(V)$ /*upper-bound volume check*/

(C.3) $2^b \leq \lambda_j(\mathbf{x})$ /*lower-bound volume check*/

Condition (C.1) guarantees that the returned local cluster $C = S_j(\mathbf{x})$ has a low motif conductance. $S_j(\mathbf{x})$ denotes the sweep cut (or sweep set) of $\mathbf{x}$ where $\mathbf{x}$ is the approximated multilinear PageRank vector and $S_j(\mathbf{x}) = \{\pi(1), \ldots, \pi(j)\}$ is the set of top j vertices v

that maximize $x(v)/d(v)$, where $\mathbf{d}$ is the degree vector, π is the permutation of nodes as $\frac{x(\pi(i))}{d(\pi(i))} \geq \frac{x(\pi(i+1))}{d(\pi(i+1))}$.

Condition (C.2) and Condition (C.3) limit the volume of the returned local cluster $S_j(\mathbf{x})$. The volume is defined as the summation of vertex degrees, $\lambda_j(\mathbf{x}) = \sum_{v \in S_j(\mathbf{x})} d(v)$ and $\mu(V) = \sum_{v \in V} d(v)$. $b \in [1, \log_2 m]$ is the parameter that controls the lower bound of volume, and m is the number of edges.

During the sweep cut procedure, j starts from 1 to the number of nodes, if a sweep cut $S_j(\mathbf{x})$ satisfies the conditions (C.1) to (C.3), then we suppose $S_j(\mathbf{x})$ is a qualified motif-preserving local cluster.

4.2.4 Proposed Method

Time-Evolving Motif Representation Nonzero entries in the adjacency matrix indicate the existence of edges (second-order motif), and the probability transition matrix derived from the adjacency matrix provides the matrix representation of the first-order Markov Chain stochastic process. To explore high-order motif data, we follow the indicator and transition tensor to represent the high-order motif existence and the transition probability of high-order Markov Chain [167, 177, 178]. Furthermore, we generalize the idea to the dynamic setting and propose the following *dynamic indicator tensor* and *dynamic transition tensor*.

Definition 4.1 (Dynamic Indicator Tensor)**.** At each timestamp t, given an undirected and unweighted graph $G^{(t)} = (V^{(t)}, E^{(t)})$, and the user-defined k^{th}-order structure $\mathbb{N}$, the k-mode dynamic indicator tensor $\bar{\mathbf{I}}^{(t)}$ is defined as follows.

$$\bar{I}^{(t)}(v_1, v_2, \ldots, v_k) = \begin{cases} 1 & \{v_1, v_2, \ldots, v_k\} \subseteq V^{(t)} \text{ and form } \mathbb{N} \\ 0 & \text{otherwise} \end{cases} \tag{4.18}$$

Definition 4.2 (Dynamic Transition Tensor)**.** At each timestamp t, given an undirected and unweighted graph $G^{(t)} = (V^{(t)}, E^{(t)})$, the kth-order motif $\mathbb{N}$, and the k-mode dynamic indicator tensor $\bar{\mathbf{I}}^{(t)}$, the corresponding k-mode dynamic transition tensor $\bar{\mathbf{P}}^{(t)}$ can be computed as follows.

$$\bar{P}^{(t)}(v_1, v_2, \ldots, v_k) = \frac{\bar{I}^{(t)}(v_1, v_2, \ldots, v_k)}{\sum_{v_1=1}^{n^{(t)}} \bar{I}^{(t)}(v_1, v_2, \ldots, v_k)} \tag{4.19}$$

At each timestamp t, the dynamic transition tensor $\bar{\mathbf{P}}^{(t)}$ encodes the transition probability with respect to k^{th}-order motif $\mathbb{N}$ based on $(k-1)^{th}$-order Markov Chain, such that $\sum_{v_1=1}^{n^{(t)}} \bar{P}^{(t)}(v_1, v_2, \ldots, v_k) = 1$ and $\bar{P}^{(t)}(v_1, v_2, \ldots, v_k) = Prob(S_{t+1} = v_1 | S_t = v_2, \ldots, S_{t-k+2} =$

v_k) [175], where S_t denotes a discrete time stochastic process on the state space. With the k-mode transition tensor $\bar{\mathbf{P}}^{(t)}$ modelling k^{th}-order motif structure, at time t, the multilinear PageRank [171] vector $\mathbf{x}^{(t)} \in \mathbb{R}^{n^{(t)}}$ satisfies the following equation:

$$\mathbf{x}^{(t)} = \alpha \bar{\mathbf{P}}^{(t)} \underbrace{(\mathbf{x}^{(t)} \otimes \ldots \otimes \mathbf{x}^{(t)})}_{(k-1 \text{ terms})} + (1-\alpha)\mathbf{u} \tag{4.20}$$

where $\otimes$ denotes the Kronecker product, and $\mathbf{u} \in \mathbb{R}^{n^{(t)}}$ is the stochastic vector inherited from last timestamp $t-1$. Stochastic vector $\mathbf{u}$ encodes a probability distribution over $n^{(t)}$ nodes, where $\sum_{i=1}^{n^{(t)}} u(i) = 1$ and its element associated with the seed node v_{seed} has a large personalized value, e.g., 1.

Push operation [161] is an efficient method for solving the approximated PageRank vector for local clustering problems. It is widely adopted for tracking the PageRank vector evolution on dynamic graphs [180, 181]: first, the approximated PageRank vector and the corresponding residual vector from the last timestamp are usually used as the initial vectors for the current timestamp; then an iterative procedure of reducing the probability distribution of the initial residual vector is performed till convergence.

We aim to extend the push operation to obtain the multilinear PageRank vector. Here, we first define the residual vector $\mathbf{r}^{(t)}$ of the multilinear PageRank vector $\mathbf{x}^{(t)}$ at timestamp t as follows.

$$\mathbf{r}^{(t)} = \alpha \bar{\mathbf{P}}^{(t)} \underbrace{(\mathbf{x}^{(t)} \otimes \ldots \otimes \mathbf{x}^{(t)})}_{(k-1 \text{ terms})} + (1-\alpha)\mathbf{u} - \mathbf{x}^{(t)} \tag{4.21}$$

Dynamic graphs change over time due to updated edges. With the vectors $\mathbf{x}^{(t)}$ and $\mathbf{r}^{(t)}$, tensor $\bar{\mathbf{P}}^{(t)}$, and local cluster $C^{(t)}$ from timestamp t, we aim to efficiently obtain $\mathbf{x}^{(t+1)}$ to feed into the sweep cut procedure (i.e., Conditions (C.1) to (C.3)), in order to get local cluster $C^{(t+1)}$. To obtain the multilinear PageRank vector $\mathbf{x}^{(t+1)}$ for timestamp $t+1$, we need to solve the following two problems efficiently, which will be elaborated in the next two paragraphs.

Transition Tensor Update via Edge Filtering To obtain the transition tensor $\bar{\mathbf{P}}^{(t+1)}$, we start with $\bar{\mathbf{P}}^{(t)}$ from the previous timestamp, and update it based on the updated edge set $\Delta E^{(t)}$. Furthermore, we propose an *edge filtering model* to identify the subset of edges in $\Delta E^{(t)}$ that have little or no impacts on the resulting local cluster, which further speeds up the tensor computation. For example, some updated edges (inserted or deleted in the current timestamp) are "far-away" from the seed node. Therefore, their appearance (or disappearance) will not affect the local cluster identified at the last timestamp. The proposed

edge filtering model is able to identify and filter these edges without affecting the generated local cluster at the current timestamp. To this end, we define the "far-away" updated edges as follows.

Definition 4.3 ("Far-away" Updated Edge)**.** At time t, given a kth-order motif $\mathbb{N}$, a multilinear PageRank vector $\mathbf{x}^{(t)}$, a degree vector $\mathbf{d}^{(t)}$ and an identified local cluster $C^{(t)} = S_j(\mathbf{x}^{(t)})$, an updated edge $e = (v_1, v_2)$ is "far-away" if it satisfies the following conditions:

$$\frac{x^{(t)}(v_1)}{d^{(t)}(v_1)} < \frac{x^{(t)}(\pi(j+1))}{d^{(t)}(\pi(j+1))},\ \frac{x^{(t)}(v_2)}{d^{(t)}(v_2)} < \frac{x^{(t)}(\pi(j+1))}{d^{(t)}(\pi(j+1))} \tag{4.22}$$

$$\frac{\gamma + x^{(t)}(\pi(j+1))}{d^{(t)}(\pi(j+1))} < \frac{x^{(t)}(\pi(j))}{d^{(t)}(\pi(j))} \tag{4.23}$$

$$D_{ist}(v_1, C^{(t)}) > k-1,\ D_{ist}(v_2, C^{(t)}) > k-1 \tag{4.24}$$

where π is the permutation that follows $\frac{x^{(t)}(\pi(i))}{d^{(t)}(\pi(i))} \geq \frac{x^{(t)}(\pi(i+1))}{d^{(t)}(\pi(i+1))}$, as stated in condition (C.1); the local cluster $C^{(t)}$ is formed by vertices $S_j(\mathbf{x}^{(t)}) = \{\pi(1), \ldots, \pi(j)\}$ from the sweep cut; $D_{ist}(v, C^{(t)})$ denotes the shortest distance from the node v to reach any node within the local cluster $C^{(t)}$; and γ denotes the largest probability mass increment that updated edge $e = (v_1, v_2)$ adds to node $\pi(j+1)$, which can potentially change the ranking of node $\pi(j+1)$ at timestamp $t+1$. Depending on whether the updated edge $e = (v_1, v_2)$ is an inserted edge or a deleted edge, it can be shown that γ has the following values:

$$\gamma = \begin{cases} \frac{x^{(t)}(v_1)\ +\ x^{(t)}(v_2)}{\min(d^{(t)}(v_1),\ d^{(t)}(v_2))} & \text{if } (v_1, v_2) \text{ is an inserted edge} \\ \frac{x^{(t)}(v_1)}{d^{(t)}(v_1)-1} + \frac{x^{(t)}(v_2)}{d^{(t)}(v_2)-1} & \text{if } (v_1, v_2) \text{ is a deleted edge} \end{cases} \tag{4.25}$$

We provide Lemma 4.1 which implies that even considering one "far-away" updated edge, the ranking of the first j entries $\{\pi(1), \ldots, \pi(j)\}$ of permutation π at time t remains still at time $t+1$; and sweep cut procedure will still terminate at the same index j at time $t+1$ to return sweep set $S_j(\mathbf{x}^{(t+1)})$ as the local cluster $C^{(t+1)}$. Therefore, one "far-away" updated edge is neglectable.

Lemma 4.1. If updated edge $e = (v_1, v_2)$ is a "far-away" updated edge, $\mathbf{x}^{(t)}$ is the multilinear PageRank vector at time t, $\mathbf{x}^{(t+1)}$ is the approximated multilinear PageRank vector after sufficient spacey random walk steps starting from vector $\mathbf{x}^{(t)}$ on the new edge set $E^{(t+1)} = \{E^{(t)} \cup e\}$, then $S_j(\mathbf{x}^{(t)}) = S_j(\mathbf{x}^{(t+1)})$; and if the local cluster $C^{(t)} = S_j(\mathbf{x}^{(t)})$ is returned by sweep cut procedure through conditions (C.1) to (C.3) regrading the parameter set $B = \{\phi, b\}$, $\mu(C^{(t)}, \mathbb{N}) < \mu(\bar{C}^{(t)}, \mathbb{N})$ when the updated edge e is inserted (or $\mu(C^{(t)}, \mathbb{N}) > \mu(\bar{C}^{(t)}, \mathbb{N})$

when the updated edge e is deleted), then sweep cut procedure will still terminate at the same j index to return local cluster $C^{(t+1)} = S_j(\mathbf{x}^{(t+1)})$ at time $t+1$ regarding the same parameter set B.

Proof. If an updated edge $e = (v_1, v_2)$ is a "far-away" updated edge, then according to Eq. 4.22, we have

$$\frac{x^{(t)}(v_1)}{d^{(t)}(v_1)} < \frac{x^{(t)}(\pi(j+1))}{d^{(t)}(\pi(j+1))}, \ \frac{x^{(t)}(v_2)}{d^{(t)}(v_2)} < \frac{x^{(t)}(\pi(j+1))}{d^{(t)}(\pi(j+1))} \tag{4.26}$$

hence, we know that the "far-away" update $e = (v_1, v_2)$ can only occur on the complement of the local cluster (i.e., $\bar{C}^{(t)}$), furthermore, only on the nodes who rank after $j+1$ in the permutation π at time t. Also, from Eq. 4.24, v_1 and v_2 are at least $k-1$ hops away from the local cluster $C^{(t)}$ which means v_1 and v_2 are not directly connected with any nodes in $C^{(t)}$. Because permutation π is nonincreasing, when a "far-away" update occurs on the complement $\bar{C}^{(t)}$, node $\pi(j+1)$ has the largest potential to break through and rank higher than node $\pi(j)$ at time $t+1$ to change the structure of the local cluster $C^{(t)}$. If node $\pi(j+1)$ fails to rank higher than node $\pi(j)$ at time $t+1$, then any nodes behind $\pi(j+1)$ will also fail.

Next, we need to prove that adding the probability mass increment of the "far-away" updated edge $e = (v_1, v_2)$ on node $\pi(j+1)$ is not enough to make node $\pi(j+1)$ rank higher than node $\pi(j)$ at time $t+1$.

In Eq. 4.23, γ is formed as the largest probability mass increment that v_1 and v_2 can contribute to node $\pi(j+1)$. Based on Eq. 4.23, we have

$$\frac{\gamma + x^{(t)}(\pi(j+1))}{d^{(t)}(\pi(j+1))} < \frac{x^{(t)}(\pi(j))}{d^{(t)}(\pi(j))} \tag{4.27}$$

which states that v_1 and v_2 are incapable to help the node $\pi(j+1)$ to rank higher than $\pi(j)$ at time $t+1$. Because the permutation π is nonincreasing which implies that nodes rank lower than $\pi(j+1)$ at time t have also no chance to rank higher than $\pi(j)$ at time $t+1$. So far, the first part of Lemma 4.1 is proved that the first j entries of permutation π remains between two consecutive timestamps, $S_j(\mathbf{x}^{(t)}) = S_j(\mathbf{x}^{(t+1)})$.

Moreover, the reason why the constructed γ represents the largest contributed probability mass increment from nodes v_1 and v_2 to node $\pi(j+1)$ is stated below.

At timestamp t, assume node $\pi(j+1)$ is reachable from the "far-away" updates v_1, then the most efficient channel to transmit the probability mass from node v_1 to node $\pi(j+1)$ is that v_1 is directly connected with node $\pi(j+1)$. Then, in the kth-order setting, the

$(k-1)^{th}$-order Markov Chain is modeled as follows.

$$Prob(S_{t+1} = \pi(j+1)|S_t = v_1, S_{t-1} = v_i, \ldots, S_{t-k+2} = v_i) \tag{4.28}$$

where S_t denotes a discrete time stochastic process on the state space $1, \ldots, n$, v_i is the node (i.e., state of the stochastic process) such that $i \in \{1, \ldots, n\}$.

We reduce the $(k-1)^{th}$-order Markov Chain to a first-order Markov Chain by omitting the tail from S_{t-1} to S_{t-k+2}, which is modeled as follows.

$$\begin{aligned} &Prob(S_{t+1} = \pi(j+1)|S_t = v_1) \geq \\ &Prob(S_{t+1} = \pi(j+1)|S_t = v_1, S_{t-1} = v_i, \ldots, S_{t-k+2} = v_i) \end{aligned} \tag{4.29}$$

Then, we can take the $Prob(S_{t+1} = \pi(j+1)|S_t = v_1) = \frac{1}{d^{(t)}(v_1)}$ which means the next random walk step starting from node v_1 will reach node $\pi(j+1)$ with probability $\frac{1}{d^{(t)}(v_1)}$. Therefore, node v_1 can transmit probability mass $\frac{x^{(t)}(v_1)}{d^{(t)}(v_1)}$ to node $\pi(j+1)$. Analogously, node v_2 can transmit probability mass $\frac{x^{(t)}(v_2)}{d^{(t)}(v_2)}$ to node $\pi(j+1)$ respectively.

Case 1. "Far-away" updated edge $e = (v_1, v_2)$ is an inserted edge.

$$\frac{x^{(t)}(v_1)}{d^{(t)}(v_1)} + \frac{x^{(t)}(v_2)}{d^{(t)}(v_2)} \leq \frac{x^{(t)}(v_1) + x^{(t)}(v_2)}{\min(d^{(t)}(v_1), d^{(t)}(v_2))} \tag{4.30}$$

where the left-hand-side is the summation of the transmitted probability mass from node v_1 and v_2 before the insertion; the right-hand side is the maximum transmitted probability mass after the insertion by taking v_1 and v_2 as one node and selecting the less transition loss channel (i.e., minimum of degree). Therefore, in the insertion scenario, we construct γ as $\frac{x^{(t)}(v_1)+x^{(t)}(v_2)}{\min(d^{(t)}(v_1),d^{(t)}(v_2))}$.

Case 2. "Far-away" updated edge $e = (v_1, v_2)$ is a deleted edge.

$$\frac{x^{(t)}(v_1)}{d^{(t)}(v_1)} + \frac{x^{(t)}(v_2)}{d^{(t)}(v_2)} < \frac{x^{(t)}(v_1)}{d^{(t)}(v_1) - 1} + \frac{x^{(t)}(v_2)}{d^{(t)}(v_2) - 1} \tag{4.31}$$

where the left-hand-side is the summation of the transmitted probability mass from node v_1 to $\pi(j+1)$ and the transmitted probability mass from node v_2 to $\pi(j+1)$ before the deletion; the right-hand-side is the summation of the transmitted probability mass from v_1 to $\pi(j+1)$ and from v_2 to $\pi(j+1)$ after the deletion. Due to the deletion, the probability mass increment of node $\pi(j+1)$ after one random walk is actually $(\frac{x^{(t)}(v_1)}{d^{(t)}(v_1)-1} + \frac{x^{(t)}(v_2)}{d^{(t)}(v_2)-1}) - (\frac{x^{(t)}(v_1)}{d^{(t)}(v_1)} + \frac{x^{(t)}(v_2)}{d^{(t)}(v_2)})$, we enlarge it by forming γ as $\frac{x^{(t)}(v_1)}{d^{(t)}(v_1)-1} + \frac{x^{(t)}(v_2)}{d^{(t)}(v_2)-1}$.

For Lemma 4.1, we have proved that one "far-away" update has no impact on sweep

set $S_j(\mathbf{x}^{(t)}) = S_j(\mathbf{x}^{(t+1)})$, i.e., the ranking of first j nodes of permutation π between two consecutive timestamps does not change. Now, we only need to prove that node $\pi(j)$ is still the first node to satisfy conditions (C.1) to (C.3) regarding the same parameter set $B = \{\phi, b\}$.

Proof of condition (C.1)

At time t, we have the local cluster $C^{(t)} = S_j(\mathbf{x}^{(t)})$ such that $\Phi(S_j(\mathbf{x}^{(t)}), \mathbb{N}) \leq \phi$. According to Eq. 4.14, we have

$$\Phi(S_j(\mathbf{x}^{(t)}), \mathbb{N}) = \frac{cut(S_j(\mathbf{x}^{(t)}))}{\min(\mu(S_j(\mathbf{x}^{(t)}), \mathbb{N}), \mu(\bar{S}_j(\mathbf{x}^{(t)}), \mathbb{N}))} \leq \phi \tag{4.32}$$

According to Eq. 4.24, the "far-away" updated edge $e = (v_1, v_2)$ is not directly connected with the local cluster $C^{(t)}$ and at least $k-1$ hops away from $C^{(t)}$, in other words, Eq. 4.24 denies any new k^{th}-order motif $\mathbb{N}$ involved into the cut identified at time t. Then, we can derive that $cut(S_j(\mathbf{x}^{(t)}) = cut(S_j(\mathbf{x}^{(t+1)})$. Because Eq. 4.22 limits the "far-away" update can only occur on the complement of the local cluster, i.e., $\bar{S}_j(\mathbf{x}^{(t)})$, such that if $\mu(S_j(\mathbf{x}^{(t)}), \mathbb{N}) < \mu(\bar{S}_j(\mathbf{x}^{(t)}), \mathbb{N})$ at time t, then a "far-away" inserted edge at time $t+1$ will have

$$\min(\mu(S_j(\mathbf{x}^{(t+1)}), \mathbb{N}), \mu(\bar{S}_j(\mathbf{x}^{(t+1)}), \mathbb{N})) = \mu(S_j(\mathbf{x}^{(t)}), \mathbb{N}) \tag{4.33}$$

and if $\mu(S_j(\mathbf{x}^{(t)}), \mathbb{N}) > \mu(\bar{S}_j(\mathbf{x}^{(t)}), \mathbb{N})$ at time t, then an "far-away" deleted edge at time $t+1$ will have

$$\min(\mu(S_j(\mathbf{x}^{(t+1)}), \mathbb{N}), \mu(\bar{S}_j(\mathbf{x}^{(t+1)}), \mathbb{N})) = \mu(\bar{S}_j(\mathbf{x}^{(t)}), \mathbb{N}) \tag{4.34}$$

Thus, according to the motif conductance score of the sweep cut procedure, node $\pi(j)$ is still the first node satisfying condition (C.1) at time $t+1$.

Proof of conditions (C.2) and (C.3)

Conditions (C.2) and (C.3) limit the volume of the local cluster $C^{(t)}$ should not be too small or too big, which is bounded by $2^b \leq \lambda_j(\mathbf{x}^{(t)}) \leq (2/3)\mu(V^{(t)})$.

Since Eq. 4.22 allows "far-away" updates can only occur on the complement $\bar{C}^{(t)}$, the volume of the local cluster $C^{(t)}$ is not changed, which means $\lambda_j(\mathbf{x}^{(t)}) = \lambda_j(\mathbf{x}^{(t+1)})$. Because we assume the size of updated edge set $\Delta E^{(t)}$ is much smaller than the whole graph $G^{(t)}$, not to mention the size of "far-away" updated edges, therefore, $\mu(V^{(t)}) \approx \mu(V^{(t+1)})$. Thus, node $\pi(j)$ is still the first node satisfying conditions (C.2) and (C.3) at time $t+1$ is proved.

To summarize, we have proved that node $\pi(j)$ is still the first node satisfying conditions (C.1) to (C.3) in the sweep cut procedure regarding the same parameter set $B = \{\phi, b\}$.

QED.

Based on the above analysis, we are now ready to present the main theorem with respect to the properties of multiple "far-away" updated edges.

Theorem 4.1. Let Γ denote the summation of probability mass increments of sampled "far-away" inserted (or deleted) edges $\{e_1, \ldots, e_m\}$, $\Gamma = \sum_{i=1}^{m} \gamma_i$, where γ_i of e_i is obtained by Eq. 4.25. If

$$\frac{\Gamma + x^{(t)}(\pi(j+1))}{d^{(t)}(\pi(j+1))} < \frac{x^{(t)}(\pi(j))}{d^{(t)}(\pi(j))} \tag{4.35}$$

and $\mu(C^{(t)}, \mathbb{N}) < \mu(\bar{C}^{(t)}, \mathbb{N})$ (or $\mu(C^{(t)}, \mathbb{N}) > \mu(\bar{C}^{(t)}, \mathbb{N})$), then $C^{(t+1)}$ for timestamp $t+1$ is the same as $C^{(t)}$.

According to Theorem 4.1, "far-away" updated edges can be safely filtered before updating transition tensor $\bar{\mathbf{P}}^{(t)}$. This greatly saves the running time, as shown below.

Proof. A single "far-away" updated edge has no impact on the evolution of the local cluster has been proved by the Lemma 4.1. Then, Lemma 4.1 and Eq. 4.35 limit that the summation of probability mass increments of sampled "far-away" inserted (or deleted) edges set $\{e_1, \ldots, e_m\}$ could also not improve node $\pi(j+1)$ rank higher than node $\pi(j)$ at time $t+1$; and $\mu(C^{(t)}, \mathbb{N}) < \mu(\bar{C}^{(t)}, \mathbb{N})$ (or $\mu(C^{(t)}, \mathbb{N}) > \mu(\bar{C}^{(t)}, \mathbb{N})$) indicates that the node $\pi(j)$ is still the first node to satisfy conditions (C.1) to (C.3) of sweep cut procedure at time $t+1$. QED.

Claim 4.1. For a kth-order motif $\mathbb{N}$ ($k \geq 3$), the time complexity of updating transition tensor $\bar{\mathbf{P}}^{(t)}$ in terms of one single updated edge is $O(\frac{k(k-1)}{2} \cdot d_{max}^{k-2})$, where d_{max} is the max degree of the graph $G^{(t)}$.

Let n and m denote the numbers of nodes and edges of $G^{(t)}$, respectively. Based on Claim 4.1, in the worst case (node degree $d_{max} \approx n$), the time complexity of updating transition tensor $\bar{\mathbf{P}}^{(t)}$ of one updated edge is $O(n^{k-2})$. However, the edge filtering operations Eq. 4.22 and 4.23 cost $O(1)$, and Eq. 4.24 costs $O(n+m\log n)$ [182]. Therefore, our proposed edge filtering model identifies a "far-away" updated edge in $O(n + m\log n)$ time instead of taking it into consideration for updating the transition tenor in $O(n^{k-2})$ time.

We summarize the proposed edge filtering model within the updating process of the transition tensor in Alg. 4.1. Step 1 separates the updated edges into inserted edges and deleted edges; Step 2 checks whether these edges are "far-away" edges according to Theorem 4.1; and Steps 3-5 (or Steps 7-9) filter "far-away" inserted (or deleted) edges before updating the transition tensor.

The edge filtering model depends on the multilinear PageRank vector (i.e., stationary distribution). As we solve the approximated multilinear PageRank vector at each timestamp,

Algorithm 4.1: Transition Tensor Update with Edge Filtering

Require:
local cluster $C^{(t)}$, transition tensor $\bar{\mathbf{P}}^{(t)}$, updated edge set $\Delta E^{(t)}$.

Ensure:
transition tensor $\bar{\mathbf{P}}^{(t+1)}$.

Separate updated edge set $\Delta E^{(t)}$ into inserted edge set ΔE_+ and deleted edge set ΔE_-.
if $\mu(C^{(t)}, \mathbb{N}) < \mu(\bar{C}^{(t)}, \mathbb{N})$ **then**
  for edge e in ΔE_+ **do**
    Sample "far-away" edge set $E_F = \{e_1, \ldots, e_m\}$ which satisfies the conditions of Eq. 4.22, 4.23, 4.24, and 4.35.
  end for
else
  for edge e in ΔE_- **do**
    Sample "far-away" edge set $E_F = \{e_1, \ldots, e_m\}$ which satisfies the conditions of Eq. 4.22, 4.23, 4.24, and 4.35.
  end for
  Update $\bar{\mathbf{P}}^{(t+1)}$ from $\bar{\mathbf{P}}^{(t)}$ based on edge set $(\Delta E^{(t)} \setminus E_F)$.
  Save filtered edge set E_F for next timestamp updates.
end if

therefore, after the long-term updating transition tensor via edge filtering model based on the approximated multilinear PageRank vector, there may be accumulated errors in the proposed L-MEGA framework. To ensure the accuracy of the transition tensor, we need to construct the transition tensor from scratch after certain timestamps.

Motif Push Operation With the updated transition tensor $\bar{\mathbf{P}}^{(t+1)}$, we also need to update the multilinear PageRank vector $\mathbf{x}^{(t+1)}$ accordingly. To this end, we initialize $\mathbf{x}^{(t+1)}$ with $\mathbf{x}^{(t)}$, and compute the initial residual vector $\mathbf{r}^{(t+1)}$ through Eq. 4.20 and 4.21 as follows.

$$\mathbf{r}^{(t+1)} = \mathbf{r}^{(t)} + \alpha(\bar{\mathbf{P}}^{(t+1)} - \bar{\mathbf{P}}^{(t)})(\mathbf{x}^{(t)} \otimes \ldots \otimes \mathbf{x}^{(t)}) \tag{4.36}$$

Next, we aim to gradually decrease the residual vector in order to obtain the updated multilinear PageRank vector. To this end, given a user-defined high-order motif, we propose an iterative *motif push operation* to track the multilinear PageRank vector. We use subscript as iteration index. Therefore, $\mathbf{x}_0^{(t+1)} = \mathbf{x}^{(t)}$, and $\mathbf{r}_0^{(t+1)}$ is set to Eq. 4.36. Motif push operation repeatedly updates the multilinear PageRank vector $\mathbf{x}_{v+1}^{(t+1)}$ and its residual vector $\mathbf{r}_{v+1}^{(t+1)}$ as follows.

$$\mathbf{x}_{v+1}^{(t+1)} = \mathbf{x}_v^{(t+1)} + r_v^{(t+1)}(i)\mathbf{e} \tag{4.37}$$

$$\mathbf{r}_{v+1}^{(t+1)} = \mathbf{r}_v^{(t+1)} - r_v^{(t+1)}(i)\mathbf{e} + \alpha r_v^{(t+1)}(i)\bar{\mathbf{P}}^{(t+1)}(\mathbf{e} \otimes \mathbf{x}_{v-1}^{(t+1)} \ldots \otimes \mathbf{x}_{v-k+2}^{(t+1)}) \tag{4.38}$$

where $r_v^{(t+1)}(i)$ is the largest entry of $\mathbf{r}_v^{(t+1)}$, $\mathbf{e} \in \mathbb{R}^{n^{(t)}}$ is the basis vector with the i-th entry equals to 1, and the other entries equal to 0.

Intuitively, motif push operation iteratively extracts the largest probability mass $r_v^{(t+1)}(i)$ from the residual vector $\mathbf{r}_v^{(t+1)}$, and adds it to the approximated multilinear PageRank vector $\mathbf{x}_v^{(t+1)}$, such that the residual vector $\mathbf{r}^{(t+1)} \to \mathbf{0}$ after sufficient iterations. During each iteration of the motif push operation, the effect of increasing the i-th entry of the vector $\mathbf{x}_v^{(t+1)}$ should also be considered, as it produces the additive divergence between $\mathbf{r}_{v+1}^{(t+1)}$ and $\mathbf{x}_{v+1}^{(t+1)}$ according to Eq. 4.20. Therefore, we adopt the lazy updating rule of HOSPLOC [167] (i.e., Eq. 4.39) to measure such additive divergence, which works as follows.

$$\mathbf{x}_{v+1}^{(t+1)} = \mathbf{P}^{(t+1)}(\mathbf{x}_v^{(t+1)} \otimes \mathbf{x}_{v-1}^{(t+1)} \otimes \ldots \otimes \mathbf{x}_{v-k+2}^{(t+1)}) \tag{4.39}$$

Adding vector $r_v^{(t+1)}(i)\mathbf{e}$ on vector $\mathbf{x}_v^{(t+1)}$ (i.e., Eq. 4.37) produces additive divergence $\alpha \mathbf{r}_v^{(t+1)}(i)\mathbf{P}^{(t+1)}(\mathbf{e} \otimes \mathbf{x}_{v-1}^{(t+1)} \ldots \otimes \mathbf{x}_{v-k+2}^{(t+1)})$ derived from Eq. 4.20 and 4.39, which should be added back to residual vector $\mathbf{r}_v^{(t+1)}$ as the third item of the right-hand-side of Eq. 4.38.

Motif push operation terminates until $\mathbf{r}^{(t+1)} = \mathbf{0}$ at timestamp $t+1$. In practice, we define a threshold ϵ such that at the v-th iteration, if the largest entry $r_v^{(t+1)}(i) \leq \epsilon$, then motif push operation terminates. The following theorem shows the efficiency of the proposed push operation with such a threshold.

Theorem 4.2 (Efficiency of motif push operation)**.** At timestamp $t+1$, given a motif push operation threshold ϵ, the motif push operation will terminate in $\frac{1}{(1-\alpha)\epsilon}$ number of iterations with $\|\mathbf{r}^{(t+1)}\|_\infty \leq \epsilon$, and each iteration has polylogarithmic time complexity $O(\frac{1}{\xi^k})$ with respect to the number of edges, where $\xi \propto \frac{1}{\log_2(\mu(V^{(t+1)}))}$, $\mu(V^{(t+1)})$ is the volume of graph $G^{(t+1)}$, k is the order of motif $\mathbb{N}$.

Proof. We define the potential δ_v of $\mathbf{r}_v^{(t+1)}$ at the v-th iteration of the motif push operation as follows.

$$\delta_v = \|\mathbf{r}_v^{(t+1)}\|_1 \tag{4.40}$$

According to the Cauchy-Schwarz inequality, in Eq. 4.38, we then can have the third term $\|\alpha r_v^{(t+1)}(i)\mathbf{P}^{(t+1)}(\mathbf{e} \otimes \mathbf{x}_{v-1}^{(t+1)} \ldots \otimes \mathbf{x}_{v-k+2}^{(t+1)})\|_1 \leq \alpha r_v^{(t+1)}(i)$. Thus, the v-th iteration reduces δ_0 by at least $(1-\alpha)r_v^{(t+1)}(i)$. Hence, after an adequate number of iterations, the motif push operation makes the final residual vector $\mathbf{r}^{(t+1)} \to \mathbf{0}$. Furthermore, for each motif push operation iteration, we have

$$\delta_{v+1} \leq \delta_v - (1-\alpha)r_v^{(t+1)}(i) \leq \delta_v - (1-\alpha)\epsilon \tag{4.41}$$

since $\delta_v \geq 0$ and maximum of δ_v is $\delta_0 \leq 1$, hence, motif push operation will terminate in at most $\frac{1}{(1-\alpha)\epsilon}$ number of iterations. The compelling complexity of each iteration is the lazy updating rule (i.e., Eq. 4.39) which runs in polylogarithmic time $O(\frac{1}{\xi^k})$ with respect to the number of edges [167], where $\xi \propto \frac{1}{\log_2(\mu(V^{(t+1)}))}$, $\mu(V^{(t+1)})$ is the volume of graph $G^{(t+1)}$, k is the order of motif $\mathbb{N}$. QED.

Incremental Sweep Cut After we have obtained the approximated multilinear PageRank vector $\mathbf{x}^{(t+1)}$ for timestamp $t+1$, we then use the sweep cut procedure to compute permutation π, which leads to sweep set $S_j(\mathbf{x}^{(t+1)})$ as local cluster $C^{(t+1)}$. However, in the sweep cut procedure, checking condition (C.1) is costly: each iteration of the sweep cut algorithm requires $O(n^2)$ for computing $\Phi(S_j(\mathbf{x}^{(t+1)}), \mathbb{N})$ with the support of the motif weighted matrix from [169, 179], where n is the number of nodes of graph $G^{(t+1)}$. Therefore, we aim to leverage the temporal information for identifying shared computation between two consecutive timestamps. In this way, we can reduce the number of iterations required in the sweep cut procedure. To this end, we first introduce some properties of the local cluster as follows.

Claim 4.2. With the local cluster $C^{(t)}$ identified by the sweep cut procedure through conditions (C.1) to (C.3), if $\mu(C^{(t)}, \mathbb{N}) < \mu(\bar{C}^{(t)}, \mathbb{N})$, and updated edge set $\Delta E^{(t)}$ only contains inserted edges on $\bar{C}^{(t)}$ after edge filtering, then $|C^{(t+1)}| \geq |C^{(t)}|$.

From Claim 4.2, when the updated edge set only consists of inserted edges on the complement of the local cluster identified by the sweep cut procedure, and the motif volume of the local cluster is smaller than its complement, then the local cluster can only expand or remain the same at the next timestamp. Hence, it is safe to conclude that the first q shared nodes of $\pi^{(t)}$ and $\pi^{(t+1)}$ are included in the same local cluster.

Based on the above analysis, we propose an incremental sweep cut to speed up the computation. With $S_j(\mathbf{x}^{(t+1)})$, we need to determine the value of j to return the local cluster $C^{(t+1)}$ in the sweep cut procedure. Here we use $j^{(t)}$ and $j^{(t+1)}$ to denote the values of j at timestamps t and $t+1$ respectively. In the original sweep cut procedure, $j^{(t+1)}$ increases from 1. However, in incremental sweep cut, if the condition of Claim 4.2 is satisfied, we first compare permutation $\pi^{(t)}$ and $\pi^{(t+1)}$ to find the largest index q such that

$$\pi^{(t)}(1) = \pi^{(t+1)}(1)\ ,\ \ \ldots\ ,\ \pi^{(t)}(q) = \pi^{(t+1)}(q)\ ,\ \text{and}\ q \leq j^{(t)} \tag{4.42}$$

then $j^{(t+1)}$ would start from $q+1$ instead of 1. In this way, incremental sweep cut reduces q iterations where each iteration costs polynomial time complexity $O(n^2)$. However, incre-

mental sweep cut cannot deal with deletion updates. Because deleted edges may keep some shared ranking but the local cluster shrinks, which means the length of the shared ranking is larger than the size of the optimal local cluster. If $j^{(t+1)}$ still iterates from $q+1$, we may miss the denser local cluster.

Algorithm L-MEGA algorithm includes three major proposed techniques for speeding up the computation. First, in updating transition tensor $\bar{\mathbf{P}}^{(t+1)}$, instead of considering every updated edge which costs $O(|V^{(t+1)}|^{k-2})$, the edge filtering model filters out some "far-away" edges in $O(|V^{(t+1)}| + |E^{(t+1)}| \log |V^{(t+1)}|)$ time. Second, the motif push operation approximates a multilinear PageRank vector in a constant number of iterations, and each iteration costs polylogarithmic time with respect to the number of edges (Theorem 4.2). Third, incremental sweep cut reduces q iterations ($O(|V^{(t+1)}|^2)$ time complexity each iteration) compared with the original sweep cut procedure. Moreover, the returned cluster $C^{(t+1)}$ is bounded, which means if there is a cluster $\hat{C}^{(t+1)}$ whose motif conductance $\Phi(\hat{C}^{(t+1)}, \mathbb{N}) \leq \frac{1}{2c_1(l+2)}$ and $\frac{x^{(t+1)}(\pi(j))}{d^{(t+1)}(\pi(j))} \geq \frac{1}{c_1(l+2)2^b}$, then $\mu(\hat{C}^{(t+1)} \cap C^{(t+1)}) \geq 2^{(b-1)}$ [167], b and c_1 are two constants, l is proportional to the volume of the graph.

As shown in Alg. 4.2, L-MEGA receives as input the high-order motif $\mathbb{N}$, the seed node v_{seed}, and the updated edge set ΔE. Notice that some inputs (e.g., $\mathbf{x}^{(0)}$, $\bar{\mathbf{P}}^{(0)}$, $C^{(0)}$) could be obtained from existing static algorithms, e.g., [167]. Then it incrementally tracks the local cluster as the graph evolves over time, and outputs the local clusters associated with each timestamp. First, Step 2 updates the transition tensor $\bar{\mathbf{P}}^{(t+1)}$ with the edge filtering model (Alg.4.1), which paves the way for calculating the initial approximate multilinear PageRank vector $\mathbf{x}_0^{(t+1)}$ and residual vector $\mathbf{r}_0^{(t+1)}$ in Step 3. Then Steps 6–8 call motif push operation to converge the initial solution fit the probability distribution of graph $G^{(t+1)}$ to obtain approximated multilinear PageRank vector $\mathbf{x}^{(t+1)}$. Steps 9–13 check whether the updated edge set $\Delta E^{(t)}$ meets the conditions of the incremental sweep cut, then set q with different values for the incremental sweep cut algorithm (i.e., Steps 14–22).

4.2.5 Experimental Evaluation

Datasets The real-world temporal graphs are summarized in Table 4.8, which are used to compare the effectiveness of the proposed L-MEGA algorithm with other state-of-the-art methods. We employ four real-world temporal graphs. Alpha and OTC networks [183, 184] are rating networks from BitcoinAlpha and BitcoinOTC platforms, where an edge is a rating record between two Bitcoin traders. Call network [185] records human mobile phone call events among a set of core users. Contact network [185] is a human contact network where

nodes represent humans and edges between them represent contacts in the physical world. Moreover, to compare the efficiency scalability on different density level temporal networks, we generate two kinds of synthetic networks: control the edge density (i.e., 0.6%) and increase the number of vertices (i.e., from $1,000$ to $5,000$); control the number of vertices (i.e., $2,000$) and increase the edge density (i.e., from 0.5% to 0.9%). The advantage of using synthetic networks is that we can investigate the scalability of efficiency of the proposed L-MEGA framework between the growth of time consumed and the growth of the size of networks.

Baselines We compare L-MEGA with different categories of state-of-the-art clustering algorithms. Static edge-based clustering algorithms (Nibble [162]), dynamic edge-based clustering algorithms (TPPR [180], ISC [186]), static motif-based clustering algorithms (HOSPLOC [167], MAPPR [169]), and our L-MEGA stands for the dynamic motif-based clustering method. Moreover, we provide the self-ablation comparison. L-MEGA-1 replaces the updating transition tensor of L-MEGA with the re-building the tensor from scratch, L-MEGA-2 replaces the tracking approximated multilinear PageRank vector of L-MEGA with solving it by power iteration, and L-MEGA-3 replaces the incremental sweep cut with the original sweep cut procedure.

Performance Comparison *Evaluation Metrics*: We set the triangle as the motif $\mathbb{N}$, triangle is the widely adopted motif for high-order clustering in existing work [167, 169, 177, 178, 179] for balancing the validity of experiments with the heavy load of time complexity $O(n^k)$, also triangles have the fundamental role in understanding the social network and community structures [187, 188, 189]. Moreover, we use three metrics: (1) *conductance*, which measures the general quality of a cut on the graph to indicate the compactness of the cut, and it is computed by setting the undirected edge as the motif $\mathbb{N}$ in Eq. 4.14; (2) *third-order conductance*, which could be computed with Eq. 4.14 by setting the undirected triangles as the motif $\mathbb{N}$; (3) *triangle density*, which computes the ratio of triangles included in the returned local cluster.

Quantitive Results: The clustering effectiveness and consumed time of each baseline are shown in Table 4.9, where we select different seed nodes for representing the average and the standard deviation of the first local minimum conductance score and its associated triangle density under the same parameter ϕ of the sweep cut. For the static algorithm, the clustering result is solely obtained at the tenth timestamp, and for the dynamic clustering algorithms, the clustering result is tracked from the first timestamp to the tenth timestamp. Because the updating and tracking process of the L-MEGA framework has accumulated errors, if L-MEGA performs well at the tenth timestamp, it also performs well at previous timestamps.

From Table 4.9, we observe that our proposed L-MEGA achieves comparable performance with the baseline methods across real-world dynamic networks in terms of the three aforementioned metrics. For example, L-MEGA performs slightly better than static algorithms like HOSPLOC in both conductance and third-order conductance. An intuitive explanation is that the last time tracked approximated multilinear PageRank was closer to the stationary distribution than the approximated multilinear PageRank solved by HOSPLOC from scratch. Comparing L-MEGA with L-MEGA-1, we know that building transition tensor occupies major time consumption of the motif-based clustering algorithms, and our updating transition tensor algorithm provides acceptable accuracy and reduces time complexity to a large extent. Comparing L-MEGA with L-MEGA-2, we know that the multilinear PageRank tracking algorithm also reduces the consumed time and provides adequate accuracy. Comparing L-MEGA with L-MEGA-3, we observe that incremental sweep cut also saves time and ensures preciseness. Observing the time consumption of L-MEGA in the Alpha network and Call Network, we find running L-MEGA on the Call dataset is faster. An explanation is that the growth of the consumed time of L-MEGA can not relate to the size of the graph but to the dynamics between two consecutive timestamps.

4.2.6 Summary

Graph motifs are subgraph patterns that occur in complex networks, which are of key importance for gaining deep insights into the structure and functionality of the graph. Motif clustering aims at finding clusters consisting of dense motif patterns. It is commonly used in various application domains, ranging from social networks to collaboration networks, from market-basket analysis to neuroscience applications. More recently, local clustering techniques have been proposed for motif-aware clustering, which focuses on a small neighborhood of the input seed node instead of the entire graph. However, most of these techniques are designed for static graphs and may render sub-optimal results when applied to large time-evolving graphs. To bridge this gap, we propose a novel framework, Local Motif Clustering on Time-Evolving Graphs (L-MEGA), which provides the evolution pattern of the local motif cluster in an effective and efficient way. The core of L-MEGA is approximately tracking the temporal evolution of the local motif cluster via novel techniques such as edge filtering, motif push operation, and incremental sweep cut. Furthermore, we theoretically analyze the efficiency and effectiveness of these techniques on time-evolving graphs. Finally, we evaluate the L-MEGA framework via extensive experiments on both synthetic and real-world temporal networks.

4.3 FAIR HIGH-ORDER GRAPH PARTITIONING VIA NATURAL + ARTIFICIAL DYNAMICS

4.3.1 Overview and Motivation

Nowadays, for the applications that require fair clustering, research interests are devoted to designing various fair objectives [190, 191, 192, 193, 194, 195, 196], to ensure the fairness for the social good [125, 197, 198, 199, 200, 201, 202, 203]. For example, based on relational data, the fairness constraints are established to guide the spectral clustering such that the demographics of each detected community are proportional to the whole distribution [190]. But the majority of existing graph clustering algorithms consider the fairness constraints in terms of the low-order connectivity patterns (e.g., edges) while overlooking the higher-order patterns (e.g., k-cliques). Actually, clique density is a key factor in many graph clustering applications. For example, k-clique preserved clusters (especially $k = 3$) play a fundamental role in understanding the social network structures [187, 204, 205, 206], as well as in identifying protein complexes and discovering new modules [207, 208]. Therefore, designing fairness constraints (e.g., proportional demographics) for high-order (e.g., clique-preserving) clustering has a huge application potential but largely remains nascent. For example, in the citation network or co-author network, the high-order clusters can be used for the expert team formation [209], but the dense connections usually occur at the intra-subject level. Without proportional demographics, the formed team could not handle interdisciplinary tasks requiring diverse backgrounds. Another example is triangle-preserved communities, which are suited for the community-driven recommendations [210]. However, when these communities are tasked with voting, without proportional demographics, the voices of different groups, especially minority groups, can barely be heard.

When we focus on clique-preserving dense clusters, it is not clear how to guarantee fairness at the same time. Furthermore, when the graph structure evolves [10, 37, 38, 211] , it can be even more challenging to ensure the clustering compactness and demographic parity simultaneously (e.g., evolving structures can break previously obtained fairness). Figure 4.3 illustrates the difficulty of achieving demographic-fair and triangle-preserving clustering on evolving graphs: even if the initial high-order clustering is demographically fair, as structure evolves, the fairness can be broken if the cluster compactness is the only objective along with time. Moreover, the straightforward application of (high-order) spectral clustering methods [34, 212, 213] can be computationally prohibitive, and the additional fairness and frequently updated graph structures may further render the existing computational resources inadequate. The core problem is how to maintain fair and dense clusters effectively and

efficiently when the underlying structure is evolving.

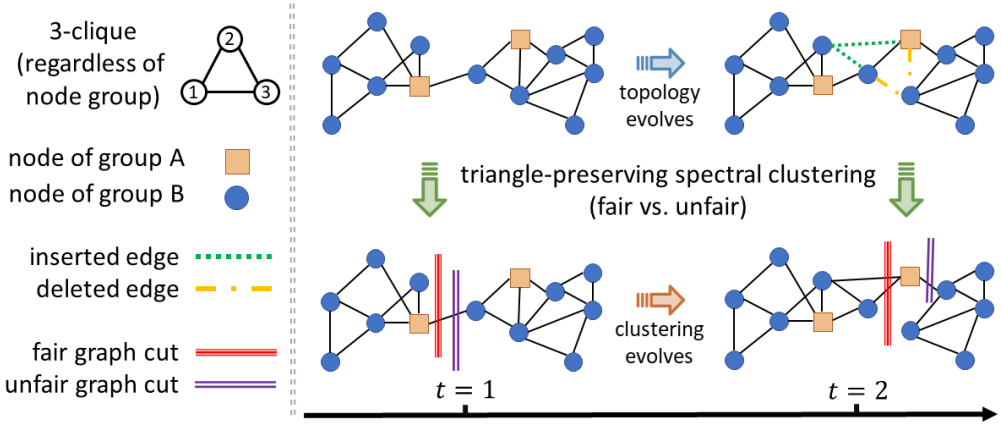


Figure 4.3: Evolving Structure breaks the Fairness during the Continuous Clustering. At $t = 0$, the unfair cut (purple) aligns with the fair cut (red); then, at $t = 1$, unfair communities are discovered for only considering the cluster compactness but ignoring the demographic fairness.

Therefore, we need to model the two separate objectives (i.e., demographic fairness and clustering compactness) in a unified spectral clustering framework. Then, efficient and effective techniques are indispensable for adapting this framework to the dynamic setting. We start from the static setting and propose a fairness-aware clique-preserving spectral clustering method, which preserves clique connections from graph cuts and guides the clustering results to be fair, i.e., the demographics of each cluster are close to the entire graph. Then, to adapt to the dynamic setting, we propose two core components: (1) *Laplacian Update via Edge Filtering and Searching*, and (2) *Eigen-Pairs Update with Singularity Avoided*. Thus, demographic-fair and clique-dense clusters under new arrival graph structures can be incrementally tracked instead of solving them from scratch. Finally, we combine the proposed techniques into an end-to-end clustering framework named F-SEGA.

4.3.2 Preliminary and Problem Definition

Cliques A k-clique is a complete subgraph consisting of k nodes, and each pair of nodes is connected with an edge. For instance, an edge is a 2-clique, and a triangle is a 3-clique.

Clique-Preserving Normalized Cut Normalized cut ($Ncut$) [34] measures the compactness of the resulting clusters regarding edge connections. A small $Ncut$ indicates a good partition where many edges are preserved in clusters. Here, we define the Clique-Preserving

$Ncut$ ($CPNcut$) to cover any k-cliques.

$$CPNcut(C_1, \ldots, C_q, \mathbb{N}) = \sum_{i=1}^{q} \frac{cut(C_i, V \setminus C_i, \mathbb{N})}{\mu(C_i, \mathbb{N})} \tag{4.43}$$

where $\mathbb{N}$ denotes the target k-clique to be preserved from partitions, $cut(C, V \setminus C_i, \mathbb{N})$ is the number of broken cliques for partitioning graph G into cluster C_i and its complement $V \setminus C_i$, and $\mu(C_i, \mathbb{N})$ denotes the number of instances of cliques within C_i. When $k = 2$, the defined $CPNcut$ is $Ncut$; when $k \geq 3$, $CPNcut$ measures the compactness of high-order clique connections.

Demographic Fairness Constraints A clustering is fair if the demographics of each cluster are close to the demographics of the whole graph [190, 214], which is expressed as follows.

$$\forall s \in \{1, \ldots, h\}, \ \frac{|V_s \bigcap C_i|}{|C_i|} = \frac{|V_s|}{|V|} \tag{4.44}$$

where V_s stands for the set of nodes of the s-th group when the entire set of nodes $V = \bigcup_{s=1}^{h} V_s$ has h different groups in total, and C_i is the i-th cluster produced by a clustering method.

Our goal is to partition the temporal graph $\mathcal{G}$ while largely preserving user-specified k-cliques from fair cuts through all given timestamps. To be specific, at timestamp t, we aim to partition graph $G^{(t)}$ into q disjoint clusters $\bigcup_{i=1}^{q} C_i^{(t)} = V$ and $C_i^{(t)} \cap C_j^{(t)} = \emptyset$ with $i \neq j \in \{1, 2, \ldots, q\}$ via the following objective.

$$\min_{C_i^{(t)}} \sum_{t=1}^{T} CPNcut(C_1^{(t)}, \ldots, C_q^{(t)}, \mathbb{N}) \tag{4.45}$$

s.t.

$$\forall s \in \{1, \ldots, h\} : \frac{|V_s \bigcap C_i^{(t)}|}{|C_i^{(t)}|} = \frac{|V_s|}{|V|}, t \in \{1, \ldots, T\} \tag{4.46}$$

Our problem can be formally defined as follows.

Problem 4.2. Fairness-Aware Clique-Preserving Clustering of Temporal Graphs

Input: (i) a temporal graph $\mathcal{G} = \{G^{(1)}, \ldots, G^{(T)}\}$, (ii) the number of clusters q, and (iii) a k-clique $\mathbb{N}$.

Output: clusters $\{C_1^{(t)}, \ldots C_q^{(t)}\}$ for $t \in \{1, \ldots, T\}$ satisfying clique-dense and demographic-fair objectives (Eq. 4.45 and Eq. 4.46).

4.3.3 Proposed Method

We start with the static setting and explain how to obtain the fairness-aware clique-preserving clustering. Then we adapt this static method to the dynamic setting via two components: Laplacian Update via Edge Filtering and Searching, and Eigen-Pairs Update with Singularity Avoided.

Clique-Preserving Spectral Clustering Traditional spectral clustering (e.g., [34]) is a special case of minimizing $CPNcut$ if the target clique $\mathbb{N}$ is set to be an edge ($k = 2$). To accommodate high-order ($k \geq 3$) clique patterns, $CPNcut$ (i.e., Eq. 4.43) is instantiated as follows.

$$\begin{aligned} CPNcut(C_1, \ldots, C_q, \mathbb{N}) &= \sum_{l=1}^{q} \frac{cut(C_l, V \setminus C_l, \mathbb{N})}{\mu(C_l, \mathbb{N})} \quad \text{w.r.t} \\ cut(C_l, V \setminus C_l, \mathbb{N}) &= \sum_{i \in C_l, j \in V \setminus C_l} W(i,j) \ , \\ \mu(C_l, \mathbb{N}) &= \sum_{i \in C_l, j \in V} W(i,j) \end{aligned} \tag{4.47}$$

where $\mathbf{W} \in \mathbb{R}^{n \times n}$ is the clique-weighted adjacency matrix, n is the number of nodes, and $W(i,j)$ is the number of instances of clique $\mathbb{N}$ containing edge (i,j). For example, if $k = 3$ and edge (i,j) makes up 4 different triangles, then $W(i,j) = 4$.

For a clustering result $\bigcup_{l=1}^{q} C_l$, if we represent it by a node-cluster assignment matrix $\mathbf{H} \in \mathbb{R}^{n \times q}$ as follows,

$$H(i,l) = \begin{cases} 1/\sqrt{\mu(C_l, \mathbb{N})} & i \in C_l \\ 0 & \text{otherwise} \end{cases} \tag{4.48}$$

then we will get $CPNcut(C_1, \ldots, C_q, \mathbb{N}) = Tr(\mathbf{H}^\top \mathbf{L} \mathbf{H})$, where $\mathbf{H}^\top$ is the transpose of matrix $\mathbf{H}$, $Tr(\cdot)$ denotes the trace of a matrix, $\mathbf{L} = \mathbf{D} - \mathbf{W} \in \mathbb{R}^{n \times n}$ is the Laplacian matrix, $\mathbf{D}$ is the clique-weighted degree matrix, i.e., $\mathbf{D} = diag(\mathbf{We})$, and $\mathbf{e}$ is the vector with all ones[4].

Therefore, the problem of minimizing $CPNcut$ equals to the trace minimization problem, i.e., $\min CPNcut(C_1, \ldots, C_q, \mathbb{N}) = \min Tr(\mathbf{H}^\top \mathbf{L} \mathbf{H})$, with $\mathbf{H}$ defined in Eq. 4.48. Since this minimization problem is discrete and proven NP-complete [34], we relax this problem by allowing the entries of matrix $\mathbf{H}$ to take any real values. Then the trace minimization

[4] $diag(\cdot)$ is defined as the standard diagonalize operation.

objective is rewritten as follows.

$$\min_{\mathbf{H}\in\mathbb{R}^{n\times q}} Tr(\mathbf{H}^\top \mathbf{L}\mathbf{H}) \text{ s.t. } \mathbf{H}^\top \mathbf{D}\mathbf{H} = \mathbf{I} \tag{4.49}$$

Substituting $\mathbf{H} = \mathbf{D}^{-1/2}\mathbf{T}$ with $\mathbf{T} \in \mathbb{R}^{n\times q}$, we have

$$\min_{\mathbf{T}\in\mathbb{R}^{n\times q}} Tr(\mathbf{T}^\top \mathbf{D}^{-1/2}\mathbf{L}\mathbf{D}^{-1/2}\mathbf{T}) \text{ s.t. } \mathbf{T}^\top \mathbf{T} = \mathbf{I} \tag{4.50}$$

Then the trace minimization problem (i.e., Eq. 4.50) for finding the optimal $\mathbf{T}$ can be solved by Rayleigh-Ritz theorem (Section 5.2.2(6) in [215]), which computes the q smallest eigenvalues of $\mathbf{D}^{-1/2}\mathbf{L}\mathbf{D}^{-1/2}$ and stores the corresponding eigenvectors as columns of the optimal $\mathbf{T}$. To infer the clustering result from $\mathbf{H} = \mathbf{D}^{-1/2}\mathbf{T}$, we can apply the K-means algorithm [216] on the rows of $\mathbf{H}$.

Fairness Constraints on Clique-Preserving Clusters To make the clique-preserving spectral clustering demographic fair, we need to add the fairness constraint (i.e., Eq. 4.46) to the clique-preserving clustering. We first derive Eq. 4.46 as follows.

$$\forall s \in \{1,\ldots,h\} : \ \frac{|V_s \bigcap C_l|}{|C_l|} = \frac{|V_s|}{|V|} \tag{4.51}$$

$$\Leftrightarrow \forall s \in \{1,\ldots,h-1\} : \sum_{i=1}^{n} \left(f_s(i) - \frac{|V_s|}{n} \right) H(i,l) = 0$$

where $\mathbf{f}_s \in \mathbb{R}^n$ is the group-membership vector: if node i belongs to the group s, then $f_s(i) = 1$; otherwise, $f_s(i) = 0$. Eq. 4.51 can be proven by replacing $H(i,l)$ with Eq. 4.48. With the fairness constraint, i.e., Eq. 4.51, the objective of clique-preserving spectral clustering (i.e., Eq. 4.49) can be extended as follows.

$$\min_{\mathbf{H}\in\mathbb{R}^{n\times q}} Tr(\mathbf{H}^\top \mathbf{L}\mathbf{H}) \text{ s.t. } \mathbf{H}^\top \mathbf{D}\mathbf{H} = \mathbf{I} \text{ and } \mathbf{F}^\top \mathbf{H} = \mathbf{0} \tag{4.52}$$

where $\mathbf{F} \in \mathbb{R}^{n\times(h-1)}$ is the group-membership matrix with column vectors $\mathbf{f}_s - \frac{|V_s|}{n} \cdot \mathbf{e}$, and $\mathbf{0}$ stands for the zero matrix.

To solve Eq. 4.52 by Rayleigh-Ritz theorem [215], we substitute $\mathbf{H} = \mathbf{Z}\mathbf{Q}^{-1}\mathbf{X}$ for $\mathbf{Z} \in \mathbb{R}^{n\times(n-h+1)}$ with $\mathbf{Q} \in \mathbb{R}^{(n-h+1)\times(n-h+1)}$

$$\min_{\mathbf{X}\in\mathbb{R}^{(n-h+1)\times q}} Tr(\mathbf{X}^\top \mathbf{Q}^{-1}\mathbf{Z}^\top \mathbf{L}\mathbf{Z}\mathbf{Q}^{-1}\mathbf{X}) \text{ s.t. } \mathbf{X}^\top \mathbf{X} = \mathbf{I} \tag{4.53}$$

where $\mathbf{Z}$ is the matrix whose columns are the orthonormal basis of the nullspace of $\mathbf{F}^\top$, and $\mathbf{Q}^2 = \mathbf{Z}^\top\mathbf{D}\mathbf{Z}$. Thus, $\mathbf{Z}$ and $\mathbf{Q}$ are directly solvable from $\mathbf{F}$ and $\mathbf{D}$. Similarly, we first need to compute q smallest eigenvalues[5] of matrix $\mathbf{Q}^{-1}\mathbf{Z}^\top\mathbf{L}\mathbf{Z}\mathbf{Q}^{-1}$, and store the corresponding eigenvectors as columns of the optimal $\mathbf{X}$; then we infer a clique-preserving and demographic-fair clustering from $\mathbf{H} = \mathbf{Z}\mathbf{Q}^{-1}\mathbf{X}$ by K-means. For notation clarity, we denote the matrix $\mathbf{M}$ as follows and name it *fairness-constrained clique-weighted Laplacian matrix*.

$$\mathbf{M} = \mathbf{Q}^{-1}\mathbf{Z}^\top\mathbf{L}\mathbf{Z}\mathbf{Q}^{-1} \in \mathbb{R}^{(n-h+1)\times(n-h+1)} \tag{4.54}$$

Challenges from the Dynamic Setting To extend the unified objective (i.e., Eq. 4.52) to the dynamic setting, two challenges need to be addressed when the input graph is evolving over time.

- Updated edges change the graph structure. Instead of rebuilding $\mathbf{M}^{(t+1)}$ from scratch every single timestamp, we need to identify the unchanged structures and update the outdated structures inside $\mathbf{M}^{(t)}$ for efficiency.
- Instead of solving eigenvalues and eigenvectors (i.e., eigen-pairs) of $\mathbf{M}^{(t+1)}$ from scratch every single timestamp, we need to compute them in a fast manner. Moreover, we also need to eliminate accumulated tracking errors, if any.

Laplacian Update via Edge Filtering and Searching When the new graph arrives, updated edge set $\Delta E^{(t)}$ changes the fairness-constrained Laplacian matrix $\mathbf{M}^{(t)}$ into $\mathbf{M}^{(t+1)}$. In Section 3.1, $\mathbf{M} = \mathbf{Q}^{-1}\mathbf{Z}^\top\mathbf{L}\mathbf{Z}\mathbf{Q}^{-1}$, where $\mathbf{Z}$ and $\mathbf{Q}$ depend on the member-group matrix $\mathbf{F}$. Since we assume the demographic group membership of each node remains the same, the Laplacian matrix $\mathbf{L}^{(t)} = \mathbf{D}^{(t)} - \mathbf{W}^{(t)}$ is the changing part of $\mathbf{M}^{(t)}$. To update $\mathbf{L}^{(t)}$, we focus on updating $\mathbf{W}^{(t+1)}$, i.e., clique-weighted adjacency matrix.

Updating the corresponding matrices between two consecutive timestamps determines the time complexity of many temporal graph algorithms [11, 12, 13, 158, 217]. According to [13], in the worst case, updating our $\mathbf{W}^{(t)}$ for one single updated edge will cost $O(n^{k-2})$, where n is the number of nodes in the graph $\mathcal{G}$, and k is the number of nodes in the target clique $\mathbb{N}$. However, not every updated edge will change the previous spectral clustering result, and we call such an edge *insensitive updated edge*, otherwise, we regard it as a *sensitive updated edge*. Hence, we aim to filter *insensitive updated edges* other than involving them in the update of $\mathbf{W}^{(t)}$. First, we model insensitive updated edges as follows.

[5]We assume $q \leq n - h + 1$ for valid solutions.

Definition 4.4 (Insensitive Updated Edge)**.** At timestamp t, given the clustering result $\bigcup_{l=1}^{q} C_l^{(t)}$, an updated edge $e = (i,j) \in \Delta E^{(t)}$ is *insensitive* if it satisfies the following conditions:

$$\begin{aligned} &i,j \in C_l^{(t)},\ e=(i,j) \in \Delta E_+^{(t)}, \\ &D_{ist}(i, V \setminus C_l^{(t)}) > 1,\ D_{ist}(j, V \setminus C_l^{(t)}) > 1 \end{aligned} \tag{4.55}$$

or

$$\begin{aligned} &i \in C_l^{(t)}, j \in V \setminus C_l^{(t)},\ e=(i,j) \in \Delta E_-^{(t)}, \\ &D_{ist}(i, V \setminus C_l^{(t)}) \leq 1 \end{aligned} \tag{4.56}$$

where $D_{ist}(v, C^{(t)})$ denotes the shortest distance (i.e., the number of hops) from node v to reach any node within the cluster $C^{(t)}$. The condition $D_{ist}(i, V \setminus C_l^{(t)}) > 1$ indicates that node i does not induce any k-clique on the clustering boundary.

Note that the insensitive edge is independent of its nodes group membership, and it is only related to graph structures. Intuitively, Eq. 4.55 can be understood as an *"inserted intra-cluster edge"*, and Eq. 4.56 can be understood as a *"deleted inter-cluster edge"*; whereas a sensitive edge is an *"inserted inter-cluster edge"* or a *"deleted intra-cluster edge"*. It is easy to prove that a single insensitive updated edge will not change the previous spectral clustering (i.e., the optimal partition $\bigcup_{l=1}^{q} C_l^{(t)}$ still achieves the optimal $CPNcut$ ratio at $t+1$). Therefore, comparing with updating $\mathbf{W}^{(t)}$ for one insensitive edge $e = (i,j)$ costing $O(n^{k-2})$, our proposed edge filtering operation (i.e., Eq. 4.55 and Eq. 4.56) only costs $O(max(D^{(t)}(i,i), D^{(t)}(j,j)))$ for identifying that insensitive edge, filtering it out for the current time update, and saving it for the future timestamps. However, even if $\Delta E^{(t)}$ only contains insensitive updated edges, multiple insensitive edges may change the spectral clustering structure under extreme circumstances. In the proof of Proposition 4.1, we analyze the extreme scenario when many insensitive updated edges affect the previously identified optimal clustering.

Proposition 4.1. Assuming that the clustering $\bigcup_{l=1}^{q} C_l^{(t)}$ is obtained by minimizing $CPNcut$ ratio under the fairness constraint at time t and $\Delta E^{(t)}$ only contains insensitive updated edges, the extreme cases exists (e.g., when the insensitive added edges are heavily localized) that the structure $\bigcup_{l=1}^{q} C_l^{(t)}$ is not guaranteed to have the minimal $CPNcut$ ratio at time $t+1$.

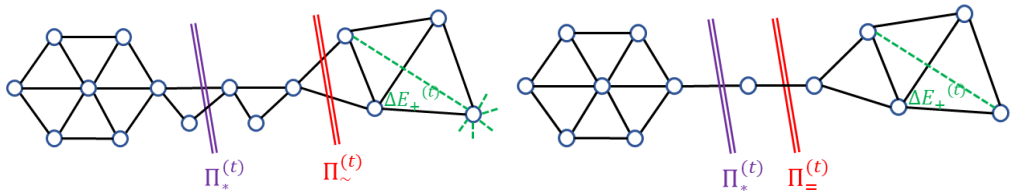
Proof. To prove Proposition 4.1, we need to show the **optimal graph cuts** $\Pi_*^{(t)}$ at time t may not achieve the optimized $CPNcut$ ratio at $t+1$, which implies the **tie graph cuts**

$\Pi_{=}^{(t)}$ and **inferior graph cuts** $\Pi_{\sim}^{(t)}$ (w.r.t $CPNcut$ ratio) may be chosen as the optimum for time $t+1$. To give a relatively full analysis, we first discuss that in a general case that $\Pi_{*}^{(t)}$ is still the optimum at $t+1$, and give some extreme cases that it will not be the optimum.

For an **insensitive inserted edge**, according to Eq. 4.47, it will only increase or remain the volume of each cluster of $\Pi_{*}^{(t)}$ and remain the number of broken cliques. Therefore, the $CPNcut$ ratio of $\Pi_{*}^{(t)}$ remains or in-depth minimizes under the fairness constraint.

- However, extreme cases can happen that $\Pi_{\sim}^{(t)}$ can be the optimum at $t+1$ for only insensitive inserted edges. For example, as shown in Figure 4.4a (for 3-clique), ΔE only contains insensitive inserted edges w.r.t $\Pi_{*}^{(t)}$, but when the huge number of insensitive updated edges arrives intensively and locally on the right complement of $\Pi_{\sim}^{(t)}$, the volume of the right cluster will increase dramatically and $\Pi_{\sim}^{(t)}$ can be $\Pi_{*}^{(t+1)}$. In the real world, the graph is usually sparse, and each time update is also much smaller compared to the volume of cluster [13], such that we based on this assumption and infer $\Pi_{\sim}^{(t)}$ will not be $\Pi_{*}^{(t+1)}$.

- Extreme cases can also happen that $\Pi_{=}^{(t)}$ is qualified for $\Pi_{*}^{(t+1)}$ when the inserted insensitive edges w.r.t $\Pi_{*}^{(t)}$ are also insensitive to $\Pi_{=}^{(t)}$ as shown in Figure 4.4b (for 3-clique). In this scenario, temporal smoothness [218] will choose $\Pi_{*}^{(t)}$ as $\Pi_{*}^{(t+1)}$, i.e., the previously selected graph cuts have a preference when tied.



(a) Extreme Case I. (b) Extreme Case II.

Figure 4.4: Inserted Insensitive Edges Analysis.

For a **deleted insensitive edge**, if that inter-cluster deleted edge occupies s k-cliques $\mathbb{N}$, then s k-cliques will disappear and the volume of graph $G^{(t)}$, $\mu(G^{(t)}, \mathbb{N})$, will decrease $s \cdot \frac{k(k-1)}{2}$, because the weights of edges in each of s k-cliques are decreased. According to Eq. 4.47, for a cluster $C_l^{(t)}$, $cut(C_l, V \setminus C_l, \mathbb{N})$ will decrease at most $\eta \cdot s \cdot \frac{k(k-1)}{2}$ and $\mu(C_l, \mathbb{N})$ will decrease at most $(1-\eta) \cdot s \cdot \frac{k(k-1)}{2}$, $\eta = \frac{\kappa}{k} \in (0,1)$, and $\kappa \in \{1, \ldots, k-1\}$. If $\eta \geq 0.5$, since $\mu(C_l, \mathbb{N}) > cut(C_l, V \setminus C_l, \mathbb{N})$, the $CPNcut$ ratio of $\Pi_{*}^{(t)}$ will decrease at $t+1$. Even $\eta \to 0$, we consider $\mu(C_l, \mathbb{N}) \gg cut(C_l, V \setminus C_l, \mathbb{N})$ then the $CPNcut$ ratio of $\Pi_{*}^{(t)}$ will remain at $t+1$. Thus, the $CPNcut$ ratio of $\Pi_{*}^{(t)}$ remains or in-depth minimizes under the fairness constraint.

- An extreme case as shown in in the Figure 4.5, $\Pi_{=}^{(t)}$ achieves the same $CPNcut$ ratio (k=3) as $\Pi_{*}^{(t)}$ in terms of insensitive deleted edges. An insensitive deleted edge to $\Pi_{*}^{(t)}$ is also insensitive to $\Pi_{=}^{(t)}$, which means $\Pi_{=}^{(t)}$ has the change to be $\Pi_{*}^{(t+1)}$. If that happens, temporal smoothness [218] will still choose $\Pi_{*}^{(t)}$ as $\Pi_{*}^{(t+1)}$, i.e., the previous selected graph cuts have preference when tied.

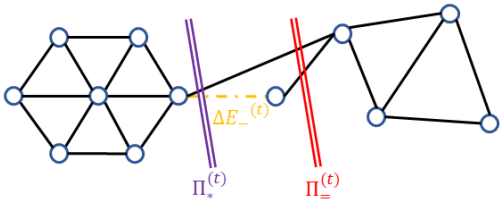


Figure 4.5: Extreme Case III.

QED.

Proposition 4.1 suggests that even if $\Delta E^{(t)}$ only consists of insensitive updated edges, the previous optimal clustering may not remain optimal when those updates are centralized on a local area of graph $G^{(t)}$. However, in practice, the graph itself and the corresponding updates can usually be sparse [38], and we can assume the extreme cases (e.g., updates are densely localized) in Proposition 4.1 are rare and obvious against the sparse background. Hence, if extreme cases happen, we may relatively easily observe them (e.g., by identifying the distance among updates or comparing the conductance of local updates with the previously obtained conductance of the entire graph) and re-run the static algorithm from scratch. Thus, in the dynamic setting, we can filter $\Delta E^{(t)}$ when it only contains insensitive updated edges and save the filtered edges for future updates. But when $\Delta E^{(t)}$ contains a single sensitive updated edge, even if the majority of $\Delta E^{(t)}$ is insensitive, the whole $\Delta E^{(t)}$ need to be involved into the updating process. Because any sensitive edge will change the previous clustering structure, which will make the currently considered insensitive edges invalid.

After edge filtering, we can then update $\mathbf{W}^{(t)}$ efficiently based on edge searching. First, we start from each inserted (or deleted) edge and then incrementally search edges that compose instances (or disappearance) of k-cliques. After that, we add the newly found cliques to $\mathbf{W}^{(t)}$. To be specific, the searching process is instanced by the enumeration of k-cliques containing the node i of the updated edge $e = (i, j)$ into the enumeration of $(k-1)$-cliques in the subgraph induced by the neighbors of node i, until $(k-1) = 2$. We illustrate the process for $k = 3$ in Alg. 4.3 and analyze how a k-clique is decomposed for the fast enumeration in Proposition 4.2. In Steps 6-8, enumerating an appeared (or disappeared) triangle due to the updated edge (i, j) is decomposed into the enumeration of any edge containing node j

in the subgraph induced by the 1-hop neighbors of node i. After detecting an appearing (or disappearing) triangle, Step 9 feeds each involved node into Eq. 4.57 to update $\mathbf{W}^{(t)}$.

$$W^{(t)}(u,v) = W^{(t)}(u,v) + \gamma \tag{4.57}$$

where u and v denote any two nodes which cause the appearance (or disappearance) of the k-clique $\mathbb{N}$; $\gamma = 1$ (or -1) if the updated edge is an inserted (or deleted) edge. With the recursion rule of decomposing a k-clique into a $(k-1)$-clique [219], the searching time complexity is bounded by the arboricity of the current graph as shown in Proposition 4.2.

Proposition 4.2. Given the graph $G^{(t)}$ and set $\Delta E^{(t)}$ containing sensitive updated edges, updating matrix $\mathbf{W}^{(t)}$ in terms of user-specified k-clique ($k \geq 2$) costs time complexity $O(k\alpha^{k-2}m^{(t)})$, where α is the arboricity of graph $G^{(t)}$, and $m^{(t)}$ is the number of edges in graph $G^{(t)}$.

Proof. We first prove that the proposed Fast Update of Graph Laplacian method updates Laplacian matrix $\mathbf{L}^{(t)}$ in terms of the 3-clique requires $O(\alpha m^{(t)})$ time. Then we generalize the proof to arbitrary k. In the outermost for-loop of Alg. 4.3, node i denotes the node with the larger degree in the updated edge (i,j). Step 3 and Step 9 require $O(d^{(t)}(i))$ time, and Step 4 requires $O(\sum_{(i,j)\in\Delta E^{(t)}} d^{(t)}(j))$ time, where $d^{(t)}(u)$ denotes the standard (i.e., edge-based) degree of the node u and $d^{(t)}(u) = \sum_{v\in V^{(t)}} A^{(t)}(u,v)$. Therefore, for the total running time $O(updating)$, we have

$$\begin{aligned}
O(updating) &= \sum_{i\in V_{\Delta E^{(t)}}} O(\ d^{(t)}(i) + \sum_{(i,j)\in\Delta E^{(t)}} d^{(t)}(j)) \\
&\leq \sum_{i\in V^{(t)}} O(\ d^{(t)}(i) + \sum_{(i,j)\in\Delta E^{(t)}} d^{(t)}(j)) \\
&\leq O(m^{(t)}) + O\ (\ \sum_{(i,j)\in\Delta E^{(t)}} d^{(t)}(j)) \\
&\leq O(m^{(t)}) + O\ (\ \sum_{(u,v)\in\Delta E^{(t)}} \min\{d^{(t)}(u),\ d^{(t)}(v)\})
\end{aligned} \tag{4.58}$$

where $V_{\Delta E^{(t)}}$ stands of the set of nodes in $\Delta E^{(t)}$. We assume that $|\Delta E^{(t)}| < m^{(t)}$, then according to [219], we have

$$\sum_{(u,v)\in\Delta E^{(t)}} \min\{d^{(t)}(u),\ d^{(t)}(v)\} \leq 2\alpha m^{(t)} \tag{4.59}$$

where α is the arboricity of the graph $G^{(t)}$ and stands for the minimum number of edge-

disjoint spanning forests of $G^{(t)}$. Note that the bounded time complexity is independent of the number of updated edges due to the summation. Thus, we have the total running time $O(updating) \leq O(\alpha m^{(t)})$. With the recursion rule of listing k-clique [219], we can detect the appearance (or disappearance) of k-cliques at each timestamp by recursively applying Alg. 4.1 until k-1=2, and the total time complexity is bounded by $O(k\alpha^{k-2}m^{(t)})$. QED.

Eigen-Pairs Update with Singularity Avoided Next, to infer clustering, we need q smallest eigenvalues $\lambda_1, \ldots, \lambda_q$ and the corresponding eigenvectors $\mathbf{u}_1, \ldots, \mathbf{u}_q$ (i.e., eigen-pairs) of the updated $\mathbf{M}^{(t+1)}$. We denote $\mathbf{\Lambda} \in \mathbb{R}^q$ as the column vector consisting of q smallest eigenvalues, i.e., $\Lambda(i) = \lambda_i$, and $\mathbf{U} \in \mathbb{R}^{(n-h+1)\times q}$ as the matrix consisting of the corresponding q eigenvectors, i.e., $U(:, j) = \mathbf{u}_j$. However, computing eigen-pairs $(\mathbf{\Lambda}^{(t+1)}, \mathbf{U}^{(t+1)})$ is costly, typically requiring $O(n^3)$ time [34], where n is the number of nodes at time $t+1$. Therefore, we aim to track them instead of solving them every single time. To be specific, based on matrix perturbation theory in [220], when $\mathbf{M}^{(t)}$ changes into $\mathbf{M}^{(t+1)}$, the change of eigen-pairs is expressed as follows.

$$(\mathbf{M}^{(t)} + \Delta\mathbf{M})(\mathbf{u}_i^{(t)} + \Delta\mathbf{u}_i) = (\lambda_i^{(t)} + \Delta\lambda_i)(\mathbf{u}_i^{(t)} + \Delta\mathbf{u}_i) \tag{4.60}$$

where $\lambda_i^{(t)}$ and $\mathbf{u}_i^{(t)}$ are the i-th eigenvalue and eigenvector of $\mathbf{M}^{(t)}$, $i \in \{1, \ldots, q\}$. Then we can expand Eq. 4.60 and obtain Eq. 4.61.

$$\mathbf{M}^{(t)}\Delta\mathbf{u}_i + \Delta\mathbf{M}\mathbf{u}_i^{(t)} + \Delta\mathbf{M}\Delta\mathbf{u}_i = \lambda_i^{(t)}\Delta\mathbf{u}_i + \Delta\lambda_i\mathbf{u}_i^{(t)} + \Delta\lambda_i\Delta\mathbf{u}_i^{(t)} \tag{4.61}$$

Eigenvalue Update. Based on Eq. 4.61, we multiply $\mathbf{u}_i^{(t)\top}$ on both sides. Suppose matrix $\mathbf{M}^{(t)}$ is near symmetric, and the eigenvector has unit length, then we get the following equation.

$$\mathbf{u}_i^{(t)\top}\Delta\mathbf{M}\mathbf{u}_i^{(t)} + \mathbf{u}_i^{(t)\top}\Delta\mathbf{M}\Delta\mathbf{u}_i = \Delta\lambda_i + \mathbf{u}_i^{(t)\top}\Delta\lambda_i\Delta\mathbf{u}_i^{(t)} \tag{4.62}$$

According to the assumption of [220] that $\Delta\lambda_i \ll \lambda_i$ and $\Delta\mathbf{u}_i \ll \mathbf{u}_i$, then $\mathbf{u}_i^{(t)\top}\Delta\mathbf{M}\Delta\mathbf{u}_i$ and $\mathbf{u}_i^{(t)\top}\Delta\lambda_i\Delta\mathbf{u}_i^{(t)}$ in Eq. 4.62 are safely omitted during the computation of $\Delta\lambda_i$. Therefore, we get eigenvalue update equation as follows.

$$\lambda_i^{(t+1)} = \lambda_i^{(t)} + \Delta\lambda_i, \quad \text{s.t.} \quad \Delta\lambda_i = \mathbf{u}_i^{(t)\top}\Delta\mathbf{M}\mathbf{u}_i^{(t)} \tag{4.63}$$

Eigenvector Update. Since we assume the dimensions of $\mathbf{M}^{(t)}$ and $\mathbf{M}^{(t+1)}$ are the same, the vector $\Delta\mathbf{u}_i$ can be expressed by the weighted sum of current eigenvectors as $\Delta\mathbf{u}_i = \sum_{j=1}^{q} \beta_{ji}\ \mathbf{u}_j^{(t)}$. Next, we need to estimate each weight β_{ji} to get $\Delta\mathbf{u}_i$, and finally obtain

$\mathbf{u}_i^{(t+1)} = \mathbf{u}_i^{(t)} + \Delta\mathbf{u}_i$.

To solve for each β_{ji}, again starting from Eq. 4.61, we replace $\Delta\mathbf{u}_i$ with $\sum_{j=1}^{q} \beta_{ji}\ \mathbf{u}_j^{(t)}$ and multiply any eigenvector $\mathbf{u}_p^{(t)}$ $(1 \leq p \leq q,\ p \neq i)$ on both sides. Then, based on the orthogonality of eigenvectors, we rearrange terms and get the following equation.

$$X(:,i)^{(t)} - \mathbf{B}^{(t)}\mathbf{b}_i + \mathbf{X}^{(t)}\mathbf{b}_i = \mathbf{0} \tag{4.64}$$

where $\mathbf{X}^{(t)} = \mathbf{U}^{(t)\top}\Delta\mathbf{M}\mathbf{U}^{(t)}$. $\mathbf{B}^{(t)} \in \mathbb{R}^{q\times q}$ is the diagonal matrix with $B^{(t)}(j,j) = \lambda_i^{(t)} + \Delta\lambda_i - \lambda_j^{(t)}$, and λ_i is from Eq. 4.63. $\mathbf{b}_i \in \mathbb{R}^q$ is the weight vector to compute Δu_i, i.e., $\mathbf{b}_i(j) = \beta_{ji}$. Rearranging Eq. 4.64, we have the weight computation equation below.

$$\mathbf{b}_i = (\mathbf{B}^{(t)} - \mathbf{X}^{(t)})^{-1} X(:,i)^{(t)} \tag{4.65}$$

With $\mathbf{b}_i$, we can then compute $\mathbf{u}_i^{(t+1)}$ as follows.

$$\mathbf{u}_i^{(t+1)} = \mathbf{u}_i^{(t)} + \Delta\mathbf{u}_i, \quad \text{s.t.} \quad \Delta\mathbf{u}_i = \sum_{j=1}^{q} \mathbf{b}_i(j)\mathbf{u}_j^{(t)} \tag{4.66}$$

However, in Eq. 4.65, when a certain eigenvalue does not change, then $\mathbf{B}^{(t)}$ is a singular matrix. Since the graph and its update are usually sparse, $\mathbf{B}^{(t)} - \mathbf{X}^{(t)}$ is also highly likely to be singular, then an unchanged eigenvalue will stop other eigenvectors' update. To address this issue, we aim to update eigenvector $\mathbf{u}_i$ independent of its $\Delta\lambda_i$. Again, starting from Eq. 4.61 and omitting trivial terms $\Delta\mathbf{M}\Delta\mathbf{u}_i$ and $\Delta\lambda_i\Delta\mathbf{u}_i^{(t)}$, we obtain the approximation solution $\mathbf{M}^{(t)}\Delta\mathbf{u}_i + \Delta\mathbf{M}\mathbf{u}_i^{(t)} = \lambda_i^{(t)}\Delta\mathbf{u}_i + \Delta\lambda_i\mathbf{u}_i^{(t)}$, where we replace $\Delta\mathbf{u}_i$ with $\sum_{j=1}^{q} \beta_{ji}\ \mathbf{u}_j^{(t)}$ and get the weight β_{ji} as follows.

$$\beta_{ji} = \frac{\mathbf{u}_j^{(t)\top}\Delta\mathbf{M}\mathbf{u}_i^{(t)}}{\lambda_i^{(t)} - \lambda_j^{(t)}} \tag{4.67}$$

Then, the alternative eigenvector update operation is as follows.

$$\mathbf{u}_i^{(t+1)} = \mathbf{u}_i^{(t)} + \Delta\mathbf{u}_i, \quad \text{s.t.} \quad \Delta\mathbf{u}_i = \sum_{j=1}^{q} \frac{\mathbf{u}_j^{(t)\top}\Delta\mathbf{M}\mathbf{u}_i^{(t)}}{\lambda_i^{(t)} - \lambda_j^{(t)}}\mathbf{u}_j^{(t)} \tag{4.68}$$

Note that, compared with Eq. 4.66, Eq. 4.68 is less accurate by omitting two mentioned trivial terms, but the matrix singularity issue is avoided. Now, we are ready to present Alg 4.4. to track eigenvalues and eigenvectors of $\mathbf{M}^{(t+1)}$. In Alg. 4.4, Steps 2–3 update the eigenvalue based on Eq. 4.63, Steps 5–12 update the eigenvector according to Eq. 4.66 if

singularity issue does not happen, otherwise Steps 13–18 update the eigenvector according to Eq. 4.68. The total time complexity of Alg. 4.4 is bounded, as shown in Proposition 4.3.

Proposition 4.3. Given the perturbation matrix $\Delta\mathbf{M}$ from time t to $t+1$, using Alg. 4.2 to track $(\mathbf{\Lambda}^{(t+1)}, \mathbf{U}^{(t+1)})$ costs time complexity $O(q^4 + nq^2)$, where q stands for the number of tracked eigenvalues, and n is the number of nodes.

Proof. First of all, for Step 1 of Alg. 4.4, we have

$$\mathbf{u}_i^{(t)\prime} \Delta\mathbf{M} \mathbf{u}_j^{(t)} = \sum_{(s,r)} \Delta M(s,r)\ u_i^{(t)}(s)\ u_j^{(t)}(r) \tag{4.69}$$

where (s,r) stands for the non-zero entry of $\Delta\mathbf{M}$. We denote the number of non-zero entries of $\Delta\mathbf{M}$ as o, then Step 1 costs $O(q^2 o)$ time complexity. Then, compared with the second for-loop (i.e., Steps 14–17), it is easy to prove that the dominant part of Alg. 4.2 is the first for-loop (i.e., Steps 5–12) [221], where the inverse operation of Step 7 costs $O(q^3)$, and the multiplication operation of Step 11 costs $O(nq)$. Therefore, the first for-loop costs $O(q^4 + nq^2)$. Since $o < n$, Alg. 4.4 will cost $O(q^4 + nq^2)$ in the worst case. QED.

Algorithm We combine all proposed techniques and introduce the end-to-end clustering framework, F-SEGA, in Alg. 4.5. Given the user-specified k-clique $\mathbb{N}$, the desired number of clusters q, the group-membership matrix $\mathbf{F}$, the clique-weighted adjacency matrix $\mathbf{W}^{(0)}$ at $t = 0$, and the updated edge set $\{\Delta E^{(0)}, \ldots, \Delta E^{(T-1)}\}$, F-SEGA algorithm outputs the clique-preserving and fairness-aware clustering associated with each timestamp t. In Alg. 4.5, the initial clustering at $t = 0$ can be obtained through the proposed static algorithm. Then, Step 2 updates Laplacian matrix $\mathbf{L}^{(t)}$ into $\mathbf{L}^{(t+1)}$ to obtain $\mathbf{M}^{(t+1)}$ through Eq. 4.54. After that, Step 3 tracks eigen-pairs of the obtained $\mathbf{M}^{(t+1)}$ by leveraging the perturbation between $\mathbf{M}^{(t+1)}$ and $\mathbf{M}^{(t)}$ in Alg. 4.4. Finally, Step 4 returns the clique-preserving and fair clustering result.

Theorem 4.3 (Time Complexity of F-SEGA)**.** The time complexity of the proposed F-SEGA algorithm is bounded by $O(k\alpha^{k-2}m^{(t)} + q^4 + q^2 n)$ at each timestamp t, where k is the number of nodes in user-defined clique $\mathbb{N}$, α is the arboricity of graph $G^{(t)}$, and $m^{(t)}$ is the number of edges in graph $G^{(t)}$.

Proof. F-SEGA consists of two sequential parts, i.e., Laplacian update and eigen-pairs update. Therefore, the time complexity is easy to be proved by Proposition 4.2 and Proposition 4.3. QED.

4.3.4 Experimental Evaluation

Datasets *HighSchool-2011* [222] stores dynamic human contacts, where 126 nodes denote students, and the 28,561 timestamped edges denote the face-to-face contacts between students during 4 days. The groups of students include males and females. *HighSchool-2013* [223] is another dynamic human contact graph of 327 high school students during 5 days. The nodes denote students grouped by males and females, and the 188,509 timestamped edges denote contacts. *PrimarySchool* [224] is a dynamic interaction graph of 232 students and 10 teachers grouped by males and females, and 125,773 timestamped edges denote the contacts among them. *Hospital* [225] is a dynamic human contact graph among patients and healthcare workers in a hospital ward for 5 days. 75 nodes in the Hospital graph are divided into four groups: patients, nurses, medical doctors, and administrative staff. 32,424 timestamped edges denote the contacts. Due to different time granularities (e.g., seconds and months) and different durations (e.g., 4 days and 10 years) among real-world temporal graphs, we unify all timestamps into 11 snapshots, where the snapshot $G^{(0)}$ occupies $80\% - 90\%$ of the entire observed graph in terms of the volume (i.e., number of edges), and each updated edge set $\Delta E^{(t)}, t \in \{0, \ldots, 9\}$ contains updated edges with the number of $1\% - 2\%$ volume of the whole observed graph. We have four metrics for the evaluation: (1) $Ncut$, which measures the compactness of clustering through broken edges, low $Ncut$ score indicates that clusters are densely connected by edges; (2) $CPNcut$, which measures the compactness of clusters in terms of broken k-cliques, in experiments we set $k = 3$, low $CPNcut$ score indicates that clusters are densely connected by k-cliques; (3) Time, which records the consumed time of each algorithm; (4) Average Balance [190], which measures the fairness of clustering by calculating the demographics of each cluster, higher balance $bal(\cdot)$ indicates a cluster is fairer. F-SEGA achieves good overall performance in terms of high-order density, demographic fairness, and time complexity. Taking HighSchool-2011 dataset as an example shown in Figure 4.6, F-SEGA is the closest one to the comprehensiveness, i.e., the line $(x = y = z)$.

Baselines We include spectral clustering algorithms from 3 aspects, i.e., **fair and unfair**, **low-order and high-order**, and **static and dynamic**. SC [34] is the standard (i.e., edge-based) spectral clustering aiming for minimizing $Ncut$. TripSC [220] tracks eigen-pairs of Laplacian matrix to meet the dynamic spectral clustering baseline. MSC [213] stands for the motif spectral clustering, which is a high-order algorithm proposed to minimize the number of broken motifs due to the graph cuts. Different from cliques, motifs do not have to be fully connected. FSC [190] represents the fair spectral clustering, which adds the fairness

constraints on the standard spectral clustering to make the demographics of each cluster close to the whole graph. For static baselines, we report the clustering result that is solely obtained on the last snapshot. *For dynamic baselines, we report the tracked clustering result until the last snapshot, whose tracking is started from the first snapshot.*

Performance Comparison *Evaluation Metrics.* (1) $Ncut$, which measures the compactness of clustering through broken edges, low $Ncut$ score indicates that clusters are densely connected by edges; (2) $CPNcut$, which measures the compactness of clusters in terms of broken k-cliques, in experiments we set $k = 3$, low $CPNcut$ score indicates that clusters are densely connected by k-cliques; (3) Time, which records the consumed time of each algorithm; (4) Average Balance [190], which measures the fairness of clustering by calculating the demographics of each cluster, higher balance $bal(\cdot)$ indicates a cluster is fairer. $Ncut$ and $CPNcut$ are realized by Eq. 4.43 by changing the order of $\mathbb{N}$, and Average Balance is realized as follows.

$$\frac{1}{|C_l|}\sum_{l} bal(C_l) = \frac{1}{|C_l|}\sum_{l} \min_{s\neq s'\in\{1,\ldots,h\}} \frac{|V_s \cap C_l|}{|V_{s'} \cap C_l|} \in [0,1] \tag{4.70}$$

Quantitative Analysis. The six settings (i.e., three datasets with a small number of clusters $q = \{5, 6, 7\}$ and a large number of clusters $q = \{14, 15, 16\}$) in Table 4.10 reflects two scenarios in the real world. *First, the initial distribution of the input graph is not demographic fair.* As shown in HighSchool-2011(Large), when all other baselines find roughly similar sub-optimal compact and fair clustering, our F-SEGA identifies a fairer (i.e., higher average balance) clustering while sacrificing very little compactness (e.g., competitive low $CPNcut$ ratio). In Highschool-2011(Small) and HighSchool-2013(Large), the input data is not fairly distributed either. Our F-SEGA still achieves very competitive fairness scores with only a little decreases in the compactness metric. *Second, the initial distribution of the input graph is already demographic fair.* For example, no matter with Highschool-2013(Small), PrimarySchool(Small), or PrimarySchool(Large), the initial distribution is fair as all baselines fall into the same performance level. Despite some random tracking errors, our F-SEGA still achieves competitive performance in a very efficient manner.

F-SEGA achieves good overall performance in terms of high-order density, demographic fairness, and time complexity. Taking HighSchool-2011 dataset as an example shown in Figure 4.6, F-SEGA is the closest one to the comprehensiveness, i.e., the line $(x = y = z)$.

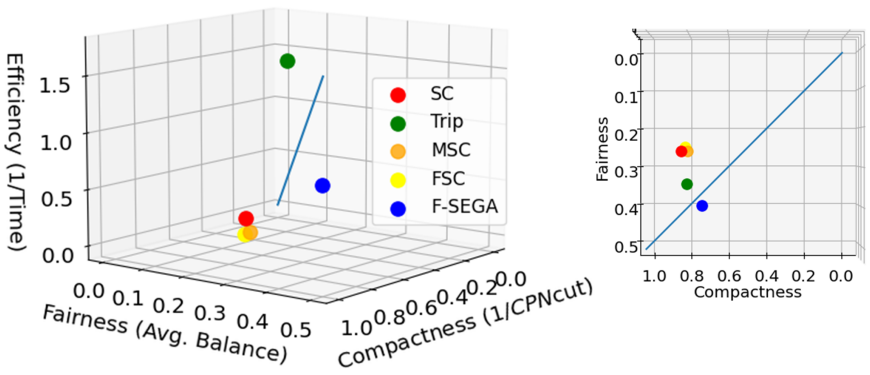


Figure 4.6: Comprehensiveness in HighSchool-2011.

4.3.5 Summary

With the widespread development of algorithmic fairness, there has been a surge of research interest that aims to generalize the fairness notions from the attributed data to the relational data (graphs). The vast majority of existing work considers the fairness measure in terms of the low-order connectivity patterns (e.g., edges), while overlooking the higher-order patterns (e.g., k-cliques) and the dynamic nature of real-world graphs. For example, preserving triangles from graph cuts during clustering is the key to detecting compact communities; however, if the clustering algorithm only pays attention to triangle-based compactness, then the returned communities lose the fairness guarantee for each group in the graph. Furthermore, in practice, when the graph (e.g., social networks) topology constantly changes over time, one natural question is how can we ensure the compactness and demographic parity at each timestamp efficiently. To address these problems, we start from the static setting and propose a spectral method that preserves clique connections and incorporates demographic fairness constraints in returned clusters at the same time. To make this static method fit for the dynamic setting, we propose two core techniques, *Laplacian Update via Edge Filtering and Searching* and *Eigen-Pairs Update with Singularity Avoided*. Finally, all proposed components are combined into an end-to-end clustering framework named F-SEGA, and we conduct extensive experiments to demonstrate the effectiveness, efficiency, and robustness of F-SEGA.

Algorithm 4.2: Local Motif Clustering on Time-Evolving Graphs (L-MEGA)

Require:
the kth-order motif $\mathbb{N}$, the transition tensor $\bar{\mathbf{P}}^{(0)}$, the initial multilinear PageRank vector $\mathbf{x}^{(0)}$ and initial residual $\mathbf{r}^{(0)}$, the initial local cluster $C^{(0)}$, the updated edge set $\{\Delta E^{(0)}, \ldots, \Delta E^{(T-1)}\}$, the motif push operation threshold ϵ, the motif conductance upperbound ϕ, and parameters b, c_1.

Ensure:
clusters $C^{(1)}, \ldots, C^{(T)}$.

for $t = 0 : T-1$ **do**
  Update transition tensor $\bar{\mathbf{P}}^{(t+1)}$ using Alg. 4.1.
  Initialize $\mathbf{x}_0^{(t+1)} = \mathbf{x}^{(t)}$, and $\mathbf{r}_0^{(t+1)}$ through Eq. 4.36.
  Initialize $\mathbf{x}_v^{(t+1)} = \mathbf{x}_0^{(t+1)}$, $\mathbf{r}_v^{(t+1)} = \mathbf{r}_0^{(t+1)}$, $v = 1, \ldots, k$.
  Set $v = k$.
  while the largest entry $r_v^{(t+1)}(i)$ of $\mathbf{r}_v^{(t+1)} \geq \epsilon$ **do**
    Get pushed $\mathbf{x}_{v+1}^{(t+1)}, \mathbf{r}_{v+1}^{(t+1)}$ through Eq. 4.37 and 4.38.
  end while
  if $\Delta E^{(t)}$ only contains inserted edges on $\bar{C}^{(t)}$ **then**
    Find the largest index q through Eq. 4.42.
  else
    Let $q = 0$.
  end if
  /*Incremental Sweep Cut*/
  for $j = q+1 : n^{(t+1)}$ **do**
    if there exists j such that:
      (C.1) $\Phi(S_j(\mathbf{x}^{(t+1)}), \mathbb{N}) \leq \phi$
      (C.2) $\lambda_j(\mathbf{x}^{(t+1)}) \leq (2/3)\mu(V)$
      (C.3) $2^b \leq \lambda_j(\mathbf{x}^{(t+1)})$ **then**
      Return $C^{(t+1)} = S_j(\mathbf{x}^{(t+1)})$ and quit.
    end if
  end for
  Return $C^{(t+1)} = S_q(\mathbf{x}^{(t+1)})$.
end for

Table 4.8: Statistics of Real-world Networks

Network	Category	$\lvert V\rvert$	$\lvert E\rvert$	Time Span
Alpha	Rating	3,783	14,124	62 months
OTC	Rating	5,881	21,492	62 months
Call	Communication	6,809	7,967	4 months
Contact	Interaction	10,972	44,517	3 months

Table 4.9: Comparison of Local Motif Clustering Effectiveness and Efficiency

Methods	Alpha				OTC			
	conductance	third-order conductance	triangle density	time	conductance	third-order conductance	triangle density	time
Nibble	0.4909 ± 0.0060	0.4555 ± 0.0454	0.2355 ± 0.1033	18.4073 ± 5.9853	0.4963 ± 0.0045	0.5091 ± 0.0941	0.1582 ± 0.1076	63.1869 ± 34.2154
TPPR	0.4923 ± 0.0089	0.4994 ± 0.1188	0.1613 ± 0.0934	12.4094 ± 5.7653	0.4970 ± 0.0021	0.5751 ± 0.1106	0.1524 ± 0.1320	39.1307 ± 19.3550
ISC	0.3334 ± 0.0000	1.0000 ± 0.0000	0.0000 ± 0.0000	56.6376 ± 0.0000	0.5999 ± 0.0000	0.5656 ± 0.0000	0.1908 ± 0.0000	195.5490 ± 0.0000
MAPPR	0.4947 ± 0.0008	0.5852 ± 0.0104	0.0712 ± 0.0030	43.0597 ± 2.9107	0.4890 ± 0.0015	0.5404 ± 0.0023	0.0904 ± 0.0001	207.5004 ± 1.1757
HOSPLOC	0.4915 ± 0.0080	0.4816 ± 0.0576	0.1891 ± 0.0859	237.6121 ± 12.5513	0.4957 ± 0.0041	0.5080 ± 0.0722	0.2000 ± 0.1172	753.3742 ± 51.6812
L-MEGA	0.4712 ± 0.0586	$\mathbf{0.4097 \pm 0.0278}$	0.2561 ± 0.1008	$\mathbf{8.2032 \pm 4.8534}$	0.4652 ± 0.0074	$\mathbf{0.4102 \pm 0.0620}$	0.2946 ± 0.0719	$\mathbf{32.4308 \pm 46.8278}$
L-MEGA-1	0.4728 ± 0.0102	0.4676 ± 0.0344	0.2490 ± 0.0736	241.4762 ± 13.3320	0.4733 ± 0.0074	0.4622 ± 0.0547	0.2578 ± 0.0961	778.5583 ± 33.4156
L-MEGA-2	0.4944 ± 0.0036	0.4369 ± 0.0428	0.3819 ± 0.0737	14.8578 ± 4.0788	0.4860 ± 0.0013	0.4750 ± 0.0175	0.5318 ± 0.0141	32.6827 ± 1.6759
L-MEGA-3	0.4712 ± 0.0586	0.4097 ± 0.0278	0.2561 ± 0.1008	11.4955 ± 4.2939	0.4652 ± 0.0074	0.4102 ± 0.0620	0.2946 ± 0.0719	45.5937 ± 45.6706
Methods	Call				Contact			
	conductance	third-order conductance	triangle density	time	conductance	third-order conductance	triangle density	time
Nibble	0.0792 ± 0.0309	0.5675 ± 0.4809	0.0249 ± 0.0384	13.5155 ± 2.7236	0.3536 ± 0.0925	0.2878 ± 0.1857	0.0017 ± 0.0015	33.7139 ± 0.1147
TPPR	0.1910 ± 0.1399	0.5589 ± 0.4442	0.0274 ± 0.0420	8.3268 ± 2.1605	0.2643 ± 0.1323	0.2221 ± 0.1382	0.0025 ± 0.0023	25.0759 ± 0.1416
ISC	0.5893 ± 0.0000	0.5270 ± 0.0000	0.1700 ± 0.0000	27.3982 ± 0.0000	0.4765 ± 0.0000	0.5252 ± 0.0000	0.0035 ± 0.0000	1351.1732 ± 0.0000
MAPPR	0.5957 ± 0.0042	0.4401 ± 0.0291	0.2219 ± 0.1869	2938.3853 ± 81.2163	0.3317 ± 0.0573	0.2790 ± 0.0753	0.0006 ± 0.0003	88.6153 ± 0.2981
HOSPLOC	0.1652 ± 0.0485	0.2981 ± 0.3721	0.0296 ± 0.0416	768.4879 ± 1.1554	0.2646 ± 0.1346	0.2308 ± 0.1559	0.0034 ± 0.0051	3443.8829 ± 0.2193
L-MEGA	0.1542 ± 0.0544	0.2866 ± 0.3823	0.0395 ± 0.0448	$\mathbf{1.1316 \pm 0.9816}$	0.2438 ± 0.1676	$\mathbf{0.1614 \pm 0.1443}$	0.0042 ± 0.0052	$\mathbf{3.4496 \pm 3.0660}$
L-MEGA-1	0.1542 ± 0.0544	0.2866 ± 0.3823	0.0395 ± 0.0448	754.2537 ± 0.8341	0.2028 ± 0.1349	0.2172 ± 0.1921	0.0004 ± 0.0005	3492.1936 ± 3.2966
L-MEGA-2	0.2333 ± 0.1839	$\mathbf{0.2713 \pm 0.3883}$	0.0239 ± 0.0180	17.2451 ± 0.9250	0.2511 ± 0.1121	0.2166 ± 0.1304	0.0003 ± 0.0004	48.1245 ± 1.1528
L-MEGA-3	0.1542 ± 0.0544	0.2866 ± 0.3823	0.0395 ± 0.0448	1.2502 ± 1.0725	0.2438 ± 0.1676	0.1614 ± 0.1443	0.0042 ± 0.0052	3.5800 ± 3.1314

Algorithm 4.3: Laplacian Update via Edge Filtering and Searching

Require:
updated edge set $\Delta E^{(t)}$, matrices $\mathbf{A}^{(t)}$, $\mathbf{W}^{(t)}$, and $\mathbf{D}^{(t)}$
Ensure:
matrices $\mathbf{A}^{(t+1)}$, $\mathbf{W}^{(t+1)}$, $\mathbf{D}^{(t+1)}$, and $\mathbf{L}^{(t+1)}$

if each $e \in \Delta E^{(t)}$ satisfies Eq. 4.55 or Eq. 4.56 **then**
  Save $\Delta E^{(t)}$ for next timestamp updates
else
  for $e = (i, j) \in \Delta E^{(t)}$ **do** /*Denote i with the larger degree*/
    Update adjacency matrix $\mathbf{A}^{(t)}$ for edge (i, j)
    Mark all the nodes adjacent to i based on $\mathbf{A}^{(t)}$
    for each node r adjacent to j **do**
      if node r is marked **then**
        Input each node pair of i, j, r into Eq. 4.57
      end if
    end for
    Erase marks
  end for
end if

Algorithm 4.4: Eigen-Pairs Update with Singularity Avoided

Require:
eigen-pairs $(\mathbf{\Lambda}^{(t)}, \mathbf{U}^{(t)})$, perturbation matrix $\Delta\mathbf{M}$
Ensure:
eigen-pairs $(\mathbf{\Lambda}^{(t+1)}, \mathbf{U}^{(t+1)})$
Compute $\mathbf{X}^{(t)} = \mathbf{U}^{(t)\top}\Delta\mathbf{M}\mathbf{U}^{(t)}$
/*Eigenvalue Update*/
$\Delta\Lambda = diag(\mathbf{X}^{(t)})$
$\Lambda^{(t+1)} = \Lambda^{(t)} + \Delta\Lambda$
/*Eigenvector Update*/
Singularity = False
for $i = 1 : q$ **do**
  Form $B^{(t)}(j,j) = \lambda_i^{(t)} + \Delta\lambda_i - \lambda_j^{(t)}$ for $j = 1, \ldots, q$
  Compute $\mathbf{b}_i = (\mathbf{B}^{(t)} - \mathbf{X}^{(t)})^{-1} X(:,i)^{(t)}$
  if $(\mathbf{B}^{(t)} - \mathbf{X}^{(t)})$ is singular **then**
    Singularity = True; Break
  end if
  $\mathbf{u}_i^{(t+1)} = \mathbf{u}_i^{(t)} + \sum_{j=1}^{q} \mathbf{b}_i(j)\mathbf{u}_j^{(t)}$
end for
if Singularity == True **then**
  for $i = 1 : q$ **do**
    $\mathbf{u}_i^{(t+1)} = \mathbf{u}_i^{(t)}$
    $\mathbf{u}_i^{(t+1)} += \frac{\mathbf{u}_j^{(t)\top}\Delta\mathbf{M}\mathbf{u}_i^{(t)}}{\lambda_i^{(t)} - \lambda_j^{(t)}}\mathbf{u}_j^{(t)}$ for $j = 1, \ldots, q$ and $j \neq i$
  end for
end if

Algorithm 4.5: Fairness-Aware Clique-Preserving Spectral Clustering of Temporal Graphs (F-SEGA)

Require:
k-clique $\mathbb{N}$, number of clusters q, matrices $\mathbf{A}^{(0)}$, $\mathbf{W}^{(0)}$, $\mathbf{F}$, and updated edge set $\{\Delta E^{(0)}, \ldots, \Delta E^{(T-1)}\}$.
Ensure:
clusters $\{C_1^{(t)}, \ldots C_q^{(t)}\}$, where $t \in \{1, 2, \ldots, T\}$
for $t = 0 : T - 1$ **do**
  Update graph Laplacian matrix $\mathbf{L}^{(t)}$ into $\mathbf{L}^{(t+1)}$ through Alg. 4.3 to obtain $\mathbf{M}^{(t+1)}$ by Eq. 4.54.
  Track $(\mathbf{\Lambda}^{(t+1)}, \mathbf{U}^{(t+1)})$ through Alg. 4.4 by leveraging $\mathbf{M}^{(t)}$ and $\mathbf{M}^{(t+1)}$.
  Set $\mathbf{U}^{(t+1)}$ as $\mathbf{X}^{(t+1)}$ in Eq. 4.53 to form $\mathbf{H}^{(t+1)}$ for obtaining clustering $\{C_1^{(t+1)}, \ldots C_q^{(t+1)}\}$
end for

Table 4.10: Comparison of Effectiveness and Efficiency

Data	HighSchool-2011 (Small Number of Clusters)				HighSchool-2011 (Large Number of Clusters)			
Method \ Metric	$Ncut$	$CPNcut$	Avg. Balance	Time (cs)	$Ncut$	$CPNcut$	Avg. Balance	Time (cs)
SC	3.1389 ± 0.8599	3.0331 ± 0.9046	0.4596 ± 0.0454	9.5270 ± 2.4491	11.7701 ± 1.0063	11.7578 ± 0.9956	0.2679 ± 0.0245	29.7407 ± 4.3385
TripSC	3.9756 ± 1.0791	3.9507 ± 1.1274	0.4519 ± 0.0669	4.7160 ± 0.2390	12.6620 ± 1.2201	12.4800 ± 1.0953	0.3458 ± 0.0268	5.8290 ± 0.0238
MSC	3.1443 ± 0.8973	2.9554 ± 0.8900	0.3888 ± 0.0850	17.0819 ± 2.1950	12.1444 ± 0.9648	12.1707 ± 0.9624	0.2686 ± 0.0503	47.5360 ± 4.0390
FSC	3.4110 ± 0.7931	3.3047 ± 0.8312	0.4457 ± 0.0185	23.8289 ± 2.3470	11.8866 ± 0.9955	11.9571 ± 1.0047	0.2473 ± 0.0228	55.2460 ± 4.5900
F-SEGA	4.4525 ± 0.9885	4.4435 ± 0.9947	0.6281 ± 0.0851	15.4022 ± 0.9090	13.4689 ± 1.5127	13.4726 ± 1.4659	0.4023 ± 0.0413	15.1860 ± 1.1020
Data	HighSchool-2013 (Small Number of Clusters)				HighSchool-2013 (Large Number of Clusters)			
Method \ Metric	$Ncut$	$CPNcut$	Avg. Balance	Time (cs)	$Ncut$	$CPNcut$	Avg. Balance	Time (cs)
SC	1.4866 ± 0.4334	0.6458 ± 0.2347	0.4708 ± 0.0135	33.2589 ± 2.3160	8.3264 ± 0.9598	6.9722 ± 0.9852	0.3596 ± 0.0276	55.7780 ± 5.3360
TripSC	1.7915 ± 0.2823	1.0755 ± 0.5641	0.4531 ± 0.0182	27.5309 ± 0.4920	12.4063 ± 0.7526	12.2279 ± 0.8568	0.4568 ± 0.1048	29.5580 ± 7.2800
MSC	1.4664 ± 0.4205	0.6483 ± 0.1829	0.4695 ± 0.0139	63.0641 ± 2.2970	8.4577 ± 1.0816	7.0812 ± 1.1676	0.3971 ± 0.0432	84.5730 ± 4.9880
FSC	1.5203 ± 0.4895	0.6620 ± 0.2860	0.5160 ± 0.0466	52.8459 ± 2.5430	8.3129 ± 0.9832	6.9370 ± 0.9918	0.3430 ± 0.0264	76.1430 ± 5.9880
F-SEGA	1.5296 ± 0.3493	0.6728 ± 0.1800	0.4620 ± 0.0058	23.1415 ± 0.1730	11.1068 ± 1.1258	10.4639 ± 1.1930	0.4481 ± 0.0827	23.2780 ± 0.5130
Data	PrimarySchool (Small Number of Clusters)				PrimarySchool (Large Number of Clusters)			
Method \ Metric	$Ncut$	$CPNcut$	Avg. Balance	Time (cs)	$Ncut$	$CPNcut$	Avg. Balance	Time (cs)
SC	3.2948 ± 0.7586	3.1690 ± 0.7504	0.7346 ± 0.0621	26.5813 ± 2.9490	7.6632 ± 0.9263	7.4345 ± 0.8838	0.6385 ± 0.0145	33.1950 ± 4.3430
TripSC	3.3368 ± 0.7193	3.1903 ± 0.7137	0.7090 ± 0.0319	17.1575 ± 0.2420	9.0180 ± 0.9117	8.8868 ± 0.9408	0.6235 ± 0.0432	17.2640 ± 0.4350
MSC	3.3528 ± 0.7852	3.1730 ± 0.7744	0.7098 ± 0.0232	74.5010 ± 2.7000	7.4506 ± 0.5170	7.4875 ± 0.8808	0.6775 ± 0.0331	90.0930 ± 4.0000
FSC	3.2830 ± 0.7488	3.1894 ± 0.7390	0.7430 ± 0.0566	49.4491 ± 3.2140	7.6606 ± 0.9217	7.4255 ± 0.8698	0.6747 ± 0.0331	67.6080 ± 4.4381
F-SEGA	3.4018 ± 0.7412	3.2126 ± 0.7205	0.6717 ± 0.0143	21.8349 ± 0.6260	8.7635 ± 1.8539	8.4676 ± 1.7872	0.6338 ± 0.0500	22.1440 ± 0.2928

CHAPTER 5: EMPOWERING GRAPH-LEVEL GRAPH INTELLIGENCE VIA NATURAL AND ARTIFICIAL DYNAMICS

In this chapter, based on graph-level graph intelligence and knowledge discovery, we have three sections introducing how natural dynamics help increase graph classification accuracy (Section 5.1), how natural dynamics help increase graph classification efficiency (Section 5.2), and how natural dynamics help to enhance the efficiency and accuracy of graph matching and alignment (Section 5.3).

5.1 INCREASING GRAPH CLASSIFICATION ACCURACY VIA NATURAL DYNAMICS

5.1.1 Overview and Motivation

Metric learning aims to learn a proper distance metric among data items in the input space, which reflects their underlying relationship. With the prevalence of graph data in many real-world applications, it is of key importance to design a good distance metric function for graph data, such that the output is small for similar (e.g., same class) graphs and large for dissimilar (e.g., different class) ones, as shown in Figure 5.1. Many downstream tasks on the graph data can benefit from such a graph metric. For example, it could lead to significantly improved classification accuracy for graph classification in many domains such as protein and drug discovery [226, 227], molecular property prediction [228, 229], and epidemic infectious pattern analysis [230, 231].

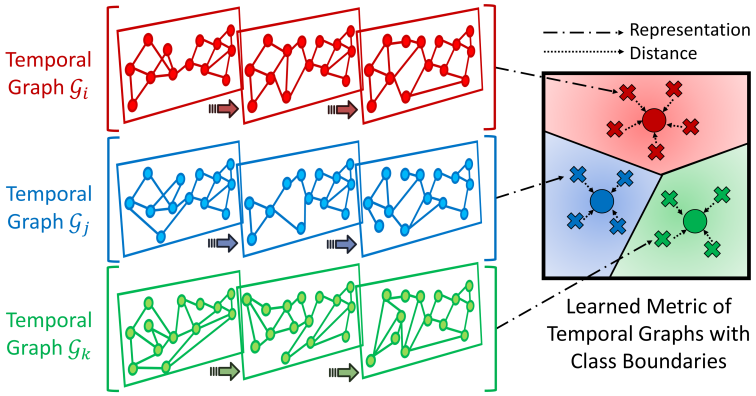


Figure 5.1: Metric Learning on Temporal Graphs $\mathcal{G}_i$, $\mathcal{G}_j$, $\mathcal{G}_k$, etc., from 3 Classes. In the right box (i.e., metric space), each 'X' denotes the representation of a single temporal graph, and each 'O' denotes the representation of a class.

Most graph metric learning methods [232, 233, 234, 235, 236] assume the input graph data

as static and ignore the evolution patterns of temporal graphs [11, 13, 80, 119, 237, 238, 239], which can also provide insights for identifying the graph property. To bridge this gap, we aim to learn a good metric over temporal graphs such that it is *accurate* (e.g., high temporal graph classification accuracy) and *comprehensive* (i.e., coordinating and adapting numerous seen and unseen classes appropriately in the space). Several metric learning works have been proposed for i.i.d data [101, 240, 241, 242, 243, 244]. However, the relational data such as graphs especially the time-evolving ones challenge metric learning in the following aspects: (1) When a graph evolves, different evolution patterns are observed from different angles, e.g., microscopic dynamics from continuous time and macroscopic dynamics from discrete time [37, 39, 41]. What patterns and their weights are dominating the temporal graph similarity and largely influencing the metric space formation? (2) Because labeling graph data is typically expensive and requires background knowledge [245, 246, 247, 248], each class may only have a few labeled graphs, and some new classes may appear at a later time. How could we leverage a few labeled graphs to learn an accurate metric, which could accommodate the arrival of future unseen classes?

To answer these questions, we propose the Temp-GFSM framework to learn a good metric over temporal graphs. Temp-GFSM is an end-to-end trainable framework, which consists of three components: (1) *Multi-Time Evolution* modeling, which preserves multi-scope evolution patterns of a single temporal graph for the downstream metric learning process; (2) *Multi-Time Attention* modeling, which receives the output from the previous component, and assigns different weights to different evolution patterns in order to improve the effectiveness of the learned metric; (3) *Temporal Graph Few-Shot Metric Learning* process with the bi-level optimization. With Temp-GFSM, on the one hand, the learned metric will be accurate for the classes observed so far; on the other hand, it could be fast adapted to unseen classes to ensure metric comprehensiveness. Tuning the whole metric learning process is based on the loss of misclassification, with only a few labels needed.

5.1.2 Problem Definition

We formally describe the problem of *Learning Metrics over Temporal Graphs* as follows.

Problem 5.1. Learning Metrics over Temporal Graphs

Input: a set of n temporal graphs $\mathcal{D} = \{\mathcal{G}_1, \ldots, \mathcal{G}_n\}$.

Output: the metric space $\mathbb{M}$ parameterized by θ, where similar (e.g., same class) temporal graphs are closer and dissimilar (e.g., different class) graphs are farther apart.

To solve the above problem, our desired metric $\mathbb{M}$ should be both ***accurate***, i.e., making similar graphs close and dissimilar ones far apart, and ***comprehensive***, i.e., being able to handle not only the classes seen so far, but also new classes appearing at a later time.

5.1.3 Proposed Method

Overview of Temp-GFSM As shown in Figure 5.2, **first**, real-world interactions are modeled into a temporal graph $\mathcal{G}$ by *Multi-Time Evolution* such that the temporal graph $\mathcal{G}$ contains continuous time tracing the microscopic evolution patterns and discrete time tracing the macroscopic evolution patterns. **Second**, each temporal graph $\mathcal{G}$ goes through the *Multi-Time Attention* and gets represented by an embedding vector $\mathbf{z}$, which embeds different-level dynamics in $\mathcal{G}$ with different attention weights. **Third**, different class vectors go into the meta-training to seek for an effective metric space $\mathbb{M}$. After that, $\mathbb{M}$ gets updated when unseen class joins the meta-testing phase.

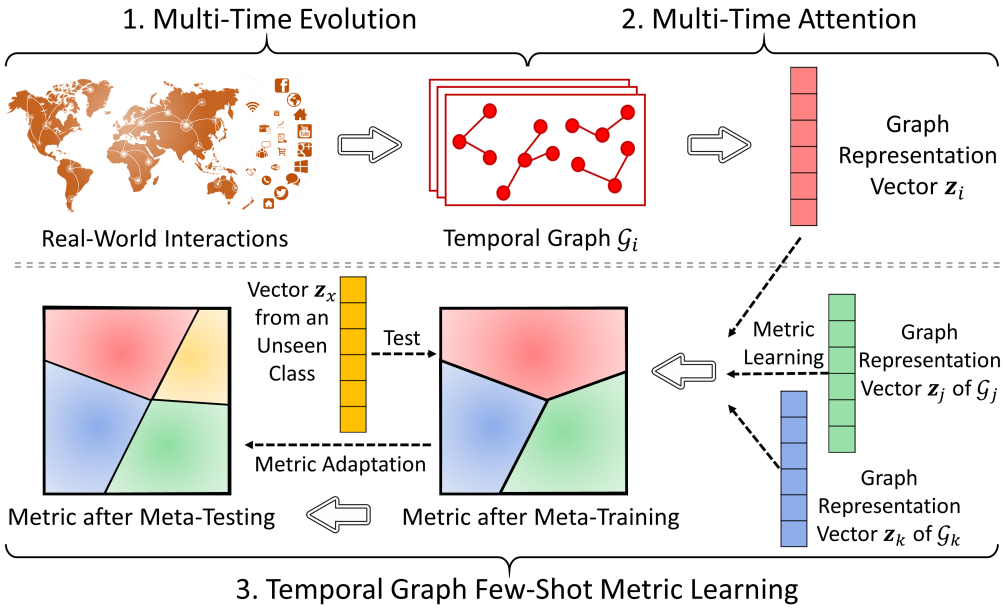


Figure 5.2: Overview of Temp-GFSM Framework.

In Temp-GFSM, the above three steps are combined in an end-to-end manner. Given the multi-time evolution modeled temporal graphs and their labels $\mathcal{D} = \{(\mathcal{G}_1, y_1), \ldots, (\mathcal{G}_n, y_n)\}$, we split $\mathcal{D}$ into training set $\mathcal{D}^{train}$ for meta-training and $\mathcal{D}^{test}$ for meta-testing, where $\mathcal{D}^{test}$ only has unseen graph labels from $\mathcal{D}^{train}$.

- At the meta-training stage, we first shuffle $\mathcal{D}^{train}$ to sample *graph metric learning tasks* $\mathcal{T}_j$ following the distribution $\mathcal{T}_j \sim \mathbb{P}(\mathcal{T})$. Controlled by θ_j, each $\mathcal{T}_j$ is instanced by a

N-way K-shot temporal graph classification task based on the graph representation $f_{\theta_j}(\mathcal{G}_i)$ (i.e., $\mathbf{z}_i$) in Euclidean space. To optimize θ_j, we again sample a support set $\mathcal{D}^{train}_{support}$ and a query set $\mathcal{D}^{train}_{query}$ within each $\mathcal{T}_j$, such that $\mathcal{D}^{train}_{support}$ is used to train f_{θ_j} to accurately predict the labels of graphs from $\mathcal{D}^{train}_{query}$.

- After we transfer the learned knowledge (i.e., θ_j) from each observed task $\mathcal{T}_j$ to the meta-learner (i.e., Θ), then at the meta-testing stage, we fine-tune Θ a few times by classifying unseen temporal graphs only in $\mathcal{D}^{test}_{support}$. Finally, we report the classification accuracy of fine-tuned Θ in query set $\mathcal{D}^{test}_{query}$.

Multi-Time Evolution Even in the same temporal graph, different modeling methods focus on different (partial) dynamic patterns. For example, a temporal graph can be modeled by an initial state G with a set of timestamped events O, and each event can be node/edge addition/deletion, which modeling is called continuous-time [37] or streaming [38]. Also, that temporal graph can be modeled as a sequence of time-respecting snapshots $G^{(1)}, G^{(2)}, \ldots$, and each $G^{(t)}$ has its own node set and edge set, which modeling is called discrete-time [37] or snapshot [38]. These two modeling methods have non-trivial complements [37, 38]: (1) **continuous-time** models rapid node/edge-level evolution, i.e., microscopic evolution [39], for example, protein molecule interactions in a cell [40]; (2) However, it loses control to represent the episodic and slowly-changing evolution patterns, which can be captured by **discrete-time modeling** (snapshot), i.e., macroscopic evolution [41], for example, the periodical metabolic cycles in that cell [40]. To the best of our knowledge, different evolution patterns in a single graph are currently not jointly considered for improving graph representation learning.

To bridge this gap, in our multi-time evolution modeling, each temporal graph $\mathcal{G}$ is defined as $\mathcal{G} = (V, E, t_e, t_s)$. V stands for the set of nodes, where each one ever exists in the whole life time of $\mathcal{G}$. E is the set of temporal edges, where each edge is defined as $e = (v_i, v_j, t_e, t_s)$, where $t_e \in \{0, 1, \ldots, T_e\}$ is named as *edge timestamp* and $t_s \in \{0, 1, \ldots, T_s\}$ is named as *snapshot timestamp*. As shown in Figure 5.3, for the record $e = (v_i, v_j, t_e, t_s)$, at the edge level, (v_i, v_j, t_e) means the connection between v_i, v_j exists at time t_e; if it still exists in the future, t_e will increase correspondingly, and if it is deleted, then this tuple (v_i, v_j, t_e) does not exist, neither does (v_i, v_j, t_e, t_s). At the snapshot level, $e = (v_i, v_j, t_e, t_s)$ means that the event (v_i, v_j, t_e) happens in the interval of snapshot $\mathcal{S}^{(t_s)}$. According to [37], in our multi-time evolution modeling, the edge timestamp t_e is continuous, and snapshot timestamp t_s is discrete. Because each snapshot $\mathcal{S}^{(t_s)}$ at t_s is generated by aggregating the event at each t_e with a certain range (e.g., every 4 t_e in Figure 5.3).

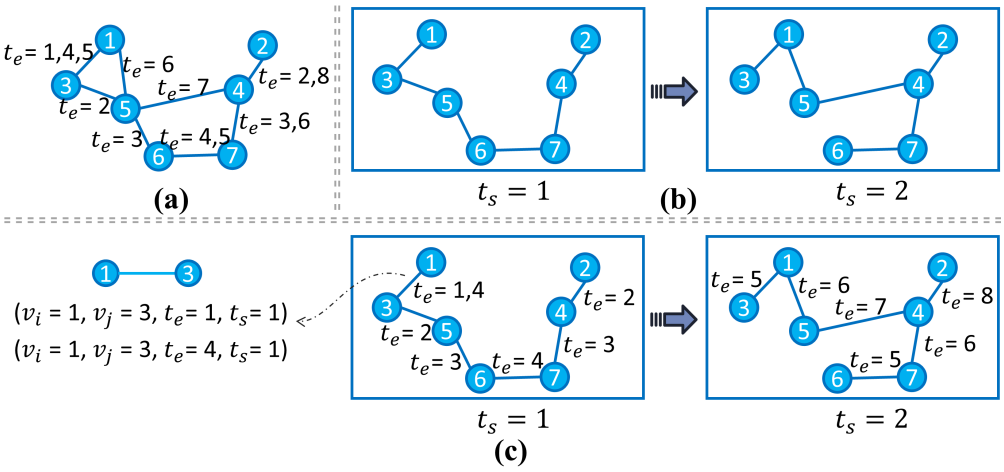


Figure 5.3: Part (a) shows a streaming graph with only edge timestamps t_e. Part (b) shows a snapshot-modeled graph with only snapshot timestamps t_s, where each t_s elapses every 4 t_e. Part (c) shows our multi-time evolution modeling with edge timestamps t_s and snapshot timestamps t_e.

Note that, two timestamps (t_e and t_s) are both real, and some temporal graphs have both naturally (furthermore, t_s do not need to be the equal interval in items of t_e like Figure 5.3). For example, in [40], each dynamic protein-protein interaction network (PPI) has 36 continuous observations (i.e., 36 edge timestamps), every 12 observations compose a metabolic cycle (i.e., 3 snapshot timestamps), and each cycle reflects 25 mins in the real world. Also, for those temporal graphs that only have edge timestamps, we can manually set the snapshot timestamp as a period hyperparameter (i.e., how many t_e compose a t_s) like Figure 5.3. Moreover, in experiments, we show how we set this hyperparameter to learn episodic or slowly-changed patterns to further help the learned metric utility (e.g., temporal graph classification). As for the data structure, we store each edge (v_i, v_j, t_e) in an array and each snapshot $\mathcal{S}^{(t_s)}$ by an adjacency matrix as $\mathbf{A}^{(t_s)} \in \mathbb{R}^{|V^{(t_s)}| \times |V^{(t_s)}|}$, i.e., $V^{(t_s)} \subseteq V$ and $|V^{(t_s)}| \neq |V^{(t_s+1)}|$ is allowable. For the notation clarity, we denote the node feature matrix $\mathbf{X} \in \mathbb{R}^{n \times m}$, such that the input node feature of temporal graph $\mathcal{G}$ is already time-aware, and $n = |V|$ and m denotes the dimension of features[6].

Multi-Time Attention Given the multiple evolution patterns of a temporal graph $\mathcal{G}$, the next step is to assign different weights to different evolution patterns, such that the dominating evolution pattern(s) can be discovered to make the learned metric accurate for downstream tasks. Hence, we propose the *Multi-Time Attention* scheme. As shown

[6]Our implementation is readily designed for evolving input features according to different timestamps, which means that $\mathbf{X}^{(t_s)} \in \mathbb{R}^{|V^{(t_s)}| \times m^{(t_s)}}$

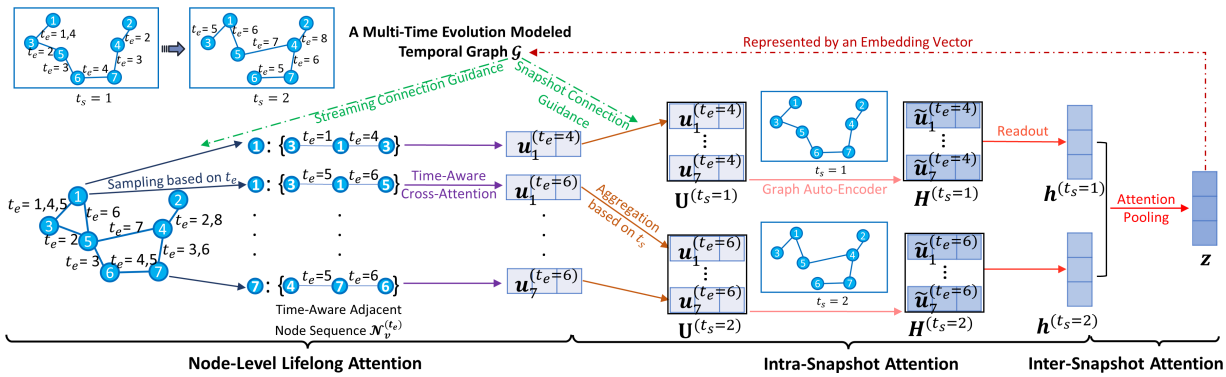


Figure 5.4: Multi-Time Attention Mechanism. A temporal graph $\mathcal{G}$ gets represented by the embedding vector $\mathbf{z}$.

in Figure 5.4, our proposed Multi-Time Attention has three sequential components, *Node-Level Lifelong Attention*, *Intra-Snapshot Attention*, and *Inter-Snapshot Attention*. Hence, a temporal graph $\mathcal{G}_i$ gets represented by an embedding vector $\mathbf{z}_i$ with its evolution patterns getting appropriate attention weights.

Node-Level Lifelong Attention. As shown in Figure 5.4, this mechanism first learns microscopic evolution patterns by the continuous timestamp t_e. To achieve this learning, we need to sample a time-aware adjacent node sequence $\mathcal{N}_v$ for each node v in the temporal graph $\mathcal{G}$, then we apply the node-level time attention mechanism on $\mathcal{N}_v$ to learn the time-aware embedding $\mathbf{u}_v$ for each node v.

The self-attention mechanism has become the key component for representing sequential data [74], which itself could not deal with ordering information without a positional encoding function to map discrete position indexes into a differentiable functional domain. Therefore, we need a time encoding function $\mathbb{K}$ for our node-level time attention mechanism, which could map every observed time interval of node connections into a continuous differentiable functional domain, i.e., $\mathbb{K} : [t_e - \Delta t, t_e] \to \mathbb{R}^d$. Another intuition of involving $\mathbb{K}$ is that, suppose node v_i connects with node v_j at edge timestamp $t_e - \Delta t$, when we need to represent node v_j at edge timestamp t_e, we wish the encoded node representation $\mathbf{u}_{v_j}^{(t_e)}$ to be time-aware by containing the temporal relation information of v_i (i.e., $\mathbb{K}(t_e - \Delta t, t_e)$). This time function $\mathbb{K}$ should be able to reflect the time relation between $\mathbf{u}_{v_i}^{(t_e - \Delta t)}$ and $\mathbf{u}_{v_j}^{(t_e)}$, and many previous works solve $\mathbb{K}$ with different kinds of kernel methods [153, 249, 250, 251, 252]. For example, in [153, 251],

$$\mathbb{K}(t_e - \Delta t, t_e) = \Psi(t_e - (t_e - \Delta t)) = \Psi(\Delta t) \tag{5.1}$$

and

$$\Psi(\Delta t) = \sqrt{\frac{1}{d}}[\cos \omega_1(\Delta t), \cos \omega_2(\Delta t), \ldots, \cos \omega_d(\Delta t)] \tag{5.2}$$

where $\Delta t = t_e - (t_e - \Delta t)$ denotes the input time interval, and $\{\omega_1, \ldots, \omega_d\}$ are learnable parameters.

With the above time encoding (i.e., Eq. 5.2), we are now ready to learn node representation $\mathbf{u}_v^{(t_e)}$. Intuitively, we set node v as the query node to query and aggregate attention weights from nodes in $\mathcal{N}_v^{(t_e)}$. To be more specific, after sampling temporal adjacent nodes $\mathcal{N}_v^{(t_e)}$ for the target (or query) node v, we apply a self-attention like mechanism on $\mathcal{N}_v^{(t_e)}$ and then stack cross-attention layers to aggregate the target node and its neighbour nodes' hidden representations, to learn the time-aware node representation $\mathbf{u}_v^{(t_e)}$.

Similar with the self-attention mechanism [74], we first use the multi-head cross-attention module and form, for each head k, queries $\mathbf{Q}$, keys $\mathbf{K}$ and values $\mathbf{V}$. Then the time-aware node representation of each head k, $\mathbf{a}_v^{k(t_e)} \in \mathbb{R}^r$, can be computed as follows.

$$\mathbf{a}_v^{k(t_e)} = \text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{r}})\mathbf{V} \in \mathbb{R}^r \tag{5.3}$$

where $\mathbf{Q} = [\mathbf{X}(v,:) \| \mathbb{K}(t_e, t_e)] \cdot \mathbf{W}_Q$ is generated by the target node v. $\mathbf{K} = \mathbf{N} \cdot \mathbf{W}_K$ and $\mathbf{V} = \mathbf{N} \cdot \mathbf{W}_V$ are generated by the adjacent node sequence with number $|\mathcal{N}_v^{(t_e)}|$. $\mathbf{N} \in \mathbb{R}^{|\mathcal{N}_v^{(t_e)}| \times (m+d)}$ is the matrix whose rows are $[\mathbf{X}(v',:) \| \mathbb{K}(t, t_e)] \in \mathbb{R}^{(m+d)}$ from the adjacent node sequence $\mathcal{N}_v^{(t_e)}$, and $\mathbf{W}_Q, \mathbf{W}_K, \mathbf{W}_V \in \mathbb{R}^{(m+d) \times r}$ are three learnable weight matrices with r denoting the dimension of the time-aware node representation vector $\mathbf{a}_v^{k(t_e)}$. We concatenate the output $\mathbf{a}_v^{k(t_e)}$ of each head k and feed it to a fully-connected feed-forward neural network, of which the parameters are $\mathbf{W}_0 \in \mathbb{R}^{k(m+d) \times r}$ and $\mathbf{b}_0 \in \mathbb{R}^r$, in order to generate the time-aware node representation $\mathbf{a}_v^{(t_e)}$. The aggregation of multi-head attention is as follows.

$$\mathbf{a}_v^{(t_e)} = ([\mathbf{a}_v^{1(t_e)} \| \ldots \| \mathbf{a}_v^{k(t_e)}])\mathbf{W}_0 + \mathbf{b}_0 \tag{5.4}$$

As mentioned above, one (multi-head) attention layer actually aggregates 1-hop neighbours temporal information according to the node adjacency sequence sampled by $\mathcal{N}_v^{(t_e)}$ Algorithm 4.1. Then, we propose to stack time-aware cross-attention layers to capture temporal information from multi-hop away neighbour nodes. Hence, in the l-th attention layer, the time-aware node representation $\mathbf{u}_v^{l(t_e)}$ can be computed as follows.

$$\mathbf{u}_v^{l(t_e)} = \text{ReLU}([\mathbf{a}_v^{l(t_e)} \| \mathbf{u}_v^{l-1(t_e)}]\mathbf{W}_1^l + \mathbf{b}_1^l)\mathbf{W}_2^l + \mathbf{b}_2^l \tag{5.5}$$

where $\mathbf{W}_1^l$, $\mathbf{b}_1^l$, $\mathbf{W}_2^l$, and $\mathbf{b}_2^l$ are parameters of the feed-forward neural network at layer l.

Also, When $l = 1$, $\mathbf{u}_v^{l-1(t_e)} = [\mathbf{X}(v,:) \| \mathbb{K}(t_e, t_e)]$.

Intra-Snapshot Attention. After we learn the time-aware node embedding $\mathbf{u}_v^{(t_e)}$ that follows the streaming pattern based on the edge timestamp t_e, we also want the node in the snapshot $\mathcal{S}^{(t_s)}$ follows the snapshot pattern of $\mathbf{A}^{(t_s)}$ w.r.t the snapshot timestamp t_s. Therefore, intra-snapshot time attention is proposed to add constraints on node embeddings $\mathbf{u}_v^{(t_e)}$ in terms of timestamp t_s by reconstructing $\mathbf{A}^{(t_s)}$ via a graph autoencoder.

Note that snapshot $\mathcal{S}^{(t_s)}$ may not have all nodes of the input temporal graph. Thus, we construct the snapshot feature matrix $\mathbf{U}^{(t_s)} \in \mathbb{R}^{|V^{(t_s)}| \times r}$ in which rows are time-aware node embedding vectors, i.e., for the edge (v_i, v_j, t_e, t_s), $\mathbf{U}^{(t_s)}(v_i,:) = \mathbf{u}_{v_i}^{(t_e)}$ and $\mathbf{U}^{(t_s)}(v_j,:) = \mathbf{u}_{v_j}^{(t_e)}$. If there is also another edge (v_i, v_j, t'_e, t_s), we will sample the most recent edge timestamp. For example, if $t'_e > t_e$, then $\mathbf{U}^{(t_s)}(v_i,:) = \mathbf{u}_{v_i}^{(t'_e)}$ and $\mathbf{U}^{(t_s)}(v_j,:) = \mathbf{u}_{v_j}^{(t'_e)}$. As shown in Figure 5.4, for snapshot $\mathcal{S}^{(t_s=1)}$, there exists two edges connecting nodes v_i=1 and v_j=3, i.e., (v_i=1, v_j=3, t_e=1, t_s=1) and (v_i=1, v_j=3, t_e=4, t_s=1). Then, for node v_i=1, vector $\mathbf{u}_1^{(t_e=4)}$ other than $\mathbf{u}_1^{(t_e=1)}$ goes into $\mathbf{U}^{(t_s=1)}(1,:)$. The reason we adopt the latest node embedding is that, according to the sampling strategy in Algorithm 5.1, the latest node embedding contains early node information.

To be specific, the time-aware adjacent node sampling is summarized in Algorithm 5.1. Starting from a user-initialized timestamp and a target node, Algorithm 5.1 captures its time-aware 1-hop neighbors, which actually form a star network surrounding the target node at a specific temporal scope. Note that, if a node v' connects the target node v at different timestamps, then each v' is a different node towards the target node v.

Algorithm 5.1: Sample Time-Aware Adjacent Node Sequence $\mathcal{N}_v^{(t_e)}$ for Node v at Edge Timestamp t_e

Require: target node v at t_e, temporal graph $\mathcal{G}$
Ensure:
time-aware adjacent node sequence $\mathcal{N}_v^{(t_e)}$
1: Initialize the starting timestamp t
2: **while** $t \leq t_e$ **do**
3: **if** edge (v', v, t) exists **then** ▷ existing connections before t_e
4: $\mathcal{N}_v^{(t_e)}$ appends $\mathbf{X}(v',:) \| \mathbb{K}(t, t_e)$ ▷ concatenation of $\mathbf{X}(v',:) \in \mathbb{R}^m$ and $\mathbb{K}(t, t_e) \in \mathbb{R}^d$
5: **end if**
6: t++
7: **end while**

Now, with adjacency matrix $\mathbf{A}^{(t_s)}$ and snapshot feature matrix $\mathbf{U}^{(t_s)}$, we next learn the latent intra-snapshot representation matrix $\mathbf{H}^{(t_s)}$. Particularly, we add a reconstruction loss to refine the time-aware node embedding vectors in matrix $\mathbf{U}^{(t_s)}$ to learn $\mathbf{H}^{(t_s)}$ via the graph

autoencoder model [253], and the snapshot reconstruction loss $\mathcal{L}_{rec}$ for the snapshot topology of $\mathcal{S}^{(t_s)}$ is defined as follows.

$$\mathcal{L}_{rec}(\mathbf{A}^{(t_s)}, \mathbf{U}^{(t_s)}) = \|\mathbf{A}^{(t_s)} - \hat{\mathbf{A}}^{(t_s)}\|_F \tag{5.6}$$

where $\hat{\mathbf{A}}^{(t_s)} = \sigma(\mathbf{H}^{(t_s)}\mathbf{H}^{(t_s)\top})$ is the reconstructed adjacency matrix computed by the sigmoid of the inner production of $\mathbf{H}^{(t_s)}$ and its transpose. $\mathbf{H}^{(t_s)} = \text{GNN}_{enc}(\mathbf{A}^{(t_s)}, \mathbf{U}^{(t_s)}) \in \mathbb{R}^{|V^{(t_s)}|\times q}$ denotes the intra-snapshot representation matrix, and $\|\cdot\|_F$ is the Frobenius norm. GNN_{enc} can be realized by GCN [25] or GAT [106]. Note that $\mathbf{H}^{(t_s)}$ and $\mathbf{U}^{(t_s)}$ share the same number of rows, but each column is updated from $\mathbf{u}_v^{(t_e)}$ to $\tilde{\mathbf{u}}_v^{(t_e)}$

Given the extracted intra-snapshot representation matrix $\mathbf{H}^{(t_s)}$, we apply a Readout function to get the intra-snapshot representation vector $\mathbf{h}^{(t_s)}$ for each snapshot timestmap t_s as follows.

$$\mathbf{h}^{(t_s)} = \text{Readout}(\mathbf{H}^{(t_s)}(v,:) \mid v \in \{1,\ldots,|V^{(t_s)}|\}) \in \mathbb{R}^q \tag{5.7}$$

where the Readout function could be a graph pooling layer like [254, 255] or any specifically designed attention pooling layers.

Inter-Snapshot Attention. After we obtain the intra-snapshot representation vector $\mathbf{h}^{(t_s)}$ for each snapshot timestamp t_s individually, we are not sure which one or ones should represent the temporal graph representation vector $\mathbf{z}$ to make it class-distinctive. For example, if a certain snapshot $\mathcal{S}$ is shared by different classes of temporal graphs, then that snapshot is less representative and we should decrease its weight during the snapshot representations aggregation process, to make different class temporal graph-level representation distinguishable. Therefore, we design an inter-snapshot time attention mechanism on vectors $\mathbf{h}^{(t_s)}$ to obtain the final temporal graph representation vector $\mathbf{z} \in \mathbb{R}^q$. Especially, the inter-snapshot time attention is realized by an attention pooling layer [256] with attention weight $\alpha^{(t_s)}$ for each snapshot time t_s. Moreover, that inter-snapshot time attention weight is parameterized by the learnable query vector $\mathbf{w}^{(t_s)} \in \mathbb{R}^q$. The temporal graph representation can then be calculated as follows.

$$\mathbf{z} = \sum_{t_s=1}^{T_s}(\alpha^{(t_s)}\mathbf{h}^{(t_s)}) \in \mathbb{R}^f,\ \alpha^{(t_s)} = \text{softmax}(\mathbf{w}^{(t_s)\top}\mathbf{h}^{(t_s)}) \tag{5.8}$$

Temporal Graph Few-Shot Metric Learning For using few-shot labeled data to learn the metric, we involve the above discussed representation learning process into the few-shot learning setting, i.e., *Bi-Level Optimization.* First, in each graph metric learning task $\mathcal{T}_i$ during the meta-training, *Prototype Generator* generates the prototype for each class, such

that each θ_i built on $\mathcal{D}^{train}_{support}$ can be in-depth optimized for the accurate classification on $\mathcal{D}^{train}_{query}$. Second, the meta-learner Θ transfers the knowledge from each θ_i of $\mathcal{T}_i$ in the meta-training phase to new tasks $\mathcal{T}_j$ in the meta-testing phase for the fast adaption on unseen classes.

Prototype Generator. After we encode a temporal graph $\mathcal{G}$ into a vector $\mathbf{z}$ as shown in Figure 5.1, we want the same class temporal graphs closer and different class graphs farther apart in metric $\mathbb{M}$, i.e., same class representation vectors are closer to their class prototype and farther from other class prototypes. Thus, in each graph metric learning task $\mathcal{T}_i$ with support set $\mathcal{D}^{train}_{support}$ and query set $\mathcal{D}^{train}_{query}$, we can first learn prototypes in $\mathcal{D}^{train}_{support}$ and then use them to predict graph class labels in $\mathcal{D}^{train}_{query}$. The construction of prototype $\mathbf{p}_k$ of class k in $\mathcal{D}^{train}_{support}$ is expressed as follows.

$$\mathbf{p}_k = \frac{1}{C_k} \sum_{j}^{C_k} (\mathbf{z}_j) \ , \ \mathcal{G}_j \in \mathcal{D}^{train}_{support} \text{ and } y_j = k \tag{5.9}$$

where $\mathcal{G}_j$ is a graph with label y_j, $\mathbf{z}_j$ is the embedding of $\mathcal{G}_j$, and C_k denotes the number of temporal graphs in class k in $\mathcal{D}^{train}_{support}$.

To help the class prototype distinctive to each other, we design the temporal graph classification loss $\mathcal{L}_{cls}$ in each graph metric learning task $\mathcal{T}_i$ to tune θ_i in $\mathcal{D}^{train}_{query}$.

$$\begin{gathered} \mathcal{L}_{cls} = -\sum_{j}^{C_k} \log \frac{\exp(-dist(\mathbf{z}_j, \mathbf{p}_k))}{\sum_{\bar{k}} \exp(-dist(\mathbf{z}_j, \mathbf{p}_{\bar{k}}))}, \\ \mathcal{G}_j \in \mathcal{D}^{train}_{query} \ \text{ and } \ y_j = k \end{gathered} \tag{5.10}$$

where $\mathbf{p}_k$ denotes the prototype of class k learned from $\mathcal{D}^{train}_{support}$, $\bar{k}$ denote the class other than k, $dist(\cdot)$ denotes Euclidean distance between two vectors, $\mathbf{z}_j$ is the representation vector of $\mathcal{G}_j$, and C_k denotes the number of class k graphs in $\mathcal{D}^{train}_{query}$.

Bi-Level Optimization. We have introduced the whole learning procedure with two loss functions $\mathcal{L}_{rec}$ (i.e., Eq. 5.6) and $\mathcal{L}_{cls}$ (i.e., Eq. 5.10). The entire loss is the weighted sum (i.e., by the balancing hyperparameter γ, $\mathcal{L}_{cls} + \gamma\mathcal{L}_{rec}$) for extracting knowledge θ_i from a single task $\mathcal{T}_i$. Next, we need to break through the knowledge transfer and adaption cross tasks given only few-shot examples. Here, we introduce a meta-learner to transfer the learned knowledge θ_i and tailor the globally shared knowledge Θ, the theory behind is that transferring shareable knowledge could obtain the fast convergence on unseen tasks [257, 258], and the bi-level optimization is able to find meta-learner Θ that could be fast converged in each graph metric learning task [259].

Algorithm 5.2: Meta-Training Process of Temp-GFSM

Require: graph metric learning task distribution $\mathbb{P}(\mathcal{T})$, step size hyperparameters α and β, loss balancing hyperparameter γ

1: Randomly initialize Θ ▷ Θ denotes all parameters
2: **while** not done **do**
3: Sample task $\mathcal{T}_i \sim \mathbb{P}(\mathcal{T})$ with its support set $\mathcal{D}^{train}_{support}$ and query set $\mathcal{D}^{train}_{query}$
4: **for** support set of each task $\mathcal{T}_i$ **do**
5: Compute $\mathbf{z}_j = f_\Theta(\mathcal{G}_j)$ ▷ $\mathcal{G}_j \in \mathcal{D}^{train}_{support}$
6: Construct $\mathbf{p}_k$ for each class k in $\mathcal{D}^{train}_{support}$ according to Eq. 5.9.
7: Evaluate snapshot reconstruction loss $\nabla_\Theta \mathcal{L}_{rec,\mathcal{T}_i}(f_\Theta)$ and classification loss $\nabla_\Theta \mathcal{L}_{cls,\mathcal{T}_i}(f_\Theta)$
8: Compute parameter $\theta_i \leftarrow \Theta - \alpha(\nabla_\Theta \mathcal{L}_{cls,\mathcal{T}_i}(f_\Theta) + \gamma \nabla_\Theta \mathcal{L}_{rec,\mathcal{T}_i}(f_\Theta))$
9: **end for**
10: Update $\Theta \leftarrow \Theta - \beta \nabla_\Theta \sum_{\mathcal{T}_i}(\mathcal{L}_{cls,\mathcal{T}_i}(f_{\theta_i}) + \gamma \mathcal{L}_{rec,\mathcal{T}_i}(f_{\theta_i}))$ ▷ On query set $\mathcal{D}^{train}_{query}$
11: **end while**

The detailed meta-training process is shown in Algorithm 5.2, and the detailed meta-testing process of Temp-GFSM is similar to Algorithm 5.2, i.e., after changing $\mathcal{D}^{train}_{support}$ into $\mathcal{D}^{test}_{support}$ and $\mathcal{D}^{train}_{query}$ into $\mathcal{D}^{test}_{query}$, the only difference is changing Step 10 of Algorithm 5.2 to directly report the accuracy based on θ_i instead of updating a new Θ.

To be specific, the meta-training phase is shown in Algorithm 5.2, we randomly initialize Θ in Step 1. Then, in each graph metric learning task $\mathcal{T}_i \sim \mathbb{P}(\mathcal{T})$, we obtain the temporal graph representation vector in Step 5 and build the class prototype for each class of the support set in Step 6. In Step 8, we tune Θ to get θ_i for current task $\mathcal{T}_i$. In Step 10, we aggregate the loss from each task $\mathcal{T}_i$ and fine-tune Θ to end the meta-training process. After that, we can use the fine-tuned Θ as the initialized parameter in the meta-testing stage for unseen graph metric learning tasks, aiming to the fast adaptation via only a few labeled samples. The meta-testing phase of Temp-GFSM is similar to Algorithm 5.2. After changing $\mathcal{D}^{train}_{support}$ into $\mathcal{D}^{test}_{support}$ and $\mathcal{D}^{train}_{query}$ into $\mathcal{D}^{test}_{query}$, the only difference is that Step 10 directly reports the accuracy based on θ_i instead of getting new Θ.

5.1.4 Empirical Evaluation

Datasets Our experiments include 12 temporal graph datasets (i.e., 12 classes) from the biological domain [40] and 6 temporal graph datasets (i.e., 12 classes) from the social network domain [260]. Each biological graph is a dynamic protein-protein interaction network, where each edge describes the protein-protein interaction of metabolic cycles of yeast cells. Each social graph is a human-contact relation network, where the edges between two individuals

stand for the online or offline contacts. The statistics of all network data are summarized in Table 5.1 and Table 5.2.

Table 5.1: Statistics of Biological Temporal Graph Data

Graph	#Classes	#Graphs	Total Nodes	Total Edges	Timestamps	Graph	#Classes	#Graphs	Total Nodes	Total Edges	Timestamps
Uetz	1	11	922	2,159	36	Ito	1	11	2,856	8,638	36
Ho	1	11	1,548	42,220	36	Gavin	1	11	2,541	140,040	36
Krogan-LCMS	1	11	2,211	85,133	36	Krogan-MALDI	1	11	2,099	78,297	36
Yu	1	11	1,163	3,602	36	Breitkreutz	1	11	869	39,250	36
Babu	1	11	5,003	111,466	36	Lambert	1	11	697	6,654	36
Tarassov	1	11	1,053	4,826	36	Hazbun	1	11	143	1,959	36

Table 5.2: Statistics of Social Temporal Graph Data

Graph (Online)	#Classes	#Graphs	Total Nodes	Total Edges	Timestamps	Graph (Offline)	#Classes	#Graphs	Total Nodes	Total Edges	Timestamps
Facebook	2	995	95,224	267,673	104	Infectious	2	200	10,000	91,944	48
Tumblr	2	373	19,811	74,520	89	HighSchool	2	180	9,418	98,066	203
DBLP	2	755	39,917	241,674	46	MIT	2	97	1,940	142,508	5,576

Baselines The selection of baseline algorithms includes two aspects, i.e., *graph kernel or graph metric learning* and *static or dynamic*. Graph kernel methods include Shortest Path [261], Neighborhood Hash [262], and Weisfeiler-Lehman Optimal Assignment [263]. Graph metric learning or graph representation learning algorithms include GL2Vec [264], Graph2Vec [265], tdGraphEmbed [266], TGAT [153], and CAW [27]. Among those, tdGraphEmbed is a dynamic algorithm that could take a temporal graph as input and output the graph embedding of each snapshot individually, and TGAT and CAW are dynamic graph representation learning algorithms but focus on the node-level representation learning. To enable graph kernel and metric learning methods the few-shot learning capability, we also include ProtoNet and its special case kNN method [101]. To enable static baselines to handle evolving graphs, we use Reduced Graph Representation [231] to map temporal graphs into dynamics-preserving static graphs. We use the mean pooling function for TGAT, CAW, and tdGraphEmbed when an aggregation is necessary.

Temporal Graph Classification First, in the biological dataset, given the 12 classes, we divide 8 classes into meta-training set $\mathcal{D}^{train}$ and 4 classes into meta-testing set $\mathcal{D}^{test}$. Note that $\mathcal{D}^{train}$ and $\mathcal{D}^{test}$ do not share any class label. We shuffle $\mathcal{D}^{train}$ and $\mathcal{D}^{test}$ 4 times for the cross-validation and report the average classification accuracy in Table 5.3. Second, in the social dataset, the amount of offline graph data is less than online data. Thus, we aim to investigate whether sufficient online temporal graph data could provide transferable knowledge to help the offline temporal graph classification. To this end, we select online networks (i.e., Facebook, Tumblr, and DBLP) for $\mathcal{D}^{train}$ and offline networks (i.e., Infectious,

HighSchool, and MIT) for $\mathcal{D}^{test}$. By comparing the performance in Table 5.4, we know that our Temp-GFSM achieves the best performance. An intuitive explanation is that online network patterns indeed share some knowledge that helps identify offline patterns. Also, our experiments show that increasing the number of shots during the meta-training is not always a good choice for improving the performance of meta-testing because intra-class variances may be amplified [267].

Table 5.3: Temporal Graph Classification Accuracy on Biological Temporal Graphs

Method \ Few-shot Setting		3 way - 5 shot	3 way - 3 shot	3 way - 2 shot	3 way - 1 shot
Graph Kernel	Weisfeiler-Lehman Opt + kNN	0.6833 ± 0.1486	0.5722 ± 0.1554	0.4958 ± 0.1843	0.3417 ± 0.1371
	Neighborhood Hash + kNN	0.6833 ± 0.1858	0.5972 ± 0.1222	0.5833 ± 0.2050	0.5500 ± 0.2082
	Shortest Path + kNN	0.5433 ± 0.1988	0.5306 ± 0.0728	0.5292 ± 0.0946	0.2333 ± 0.1217
Graph Metric Learning	GL2Vec + kNN	0.0717 ± 0.0900	0.0917 ± 0.0793	0.0333 ± 0.0471	0.0333 ± 0.0667
	Graph2Vec + kNN [7]	–	–	–	–
	TGAT + kNN	0.1200 ± 0.0960	0.1250 ± 0.0793	0.1208 ± 0.0865	0.0917 ± 0.0319
	CAW + kNN	0.0400 ± 0.0362	0.0435 ± 0.0441	0.0569 ± 0.0370	0.0528 ± 0.0210
	tdGraphEmbed + kNN	0.2167 ± 0.1736	0.1056 ± 0.0814	0.1500 ± 0.1800	0.0750 ± 0.0877
	GL2Vec + Protonet	0.7100 ± 0.0361	0.6625 ± 0.0407	0.6075 ± 0.0496	0.5750 ± 0.0537
	Graph2Vec + ProtoNet	0.3792 ± 0.0459	0.3958 ± 0.0731	0.3958 ± 0.0241	0.3958 ± 0.0798
	TGAT + ProtoNet	0.2417 ± 0.0500	0.3083 ± 0.0739	0.2917 ± 0.1167	0.2417 ± 0.0319
	CAW + ProtoNet	0.1496 ± 0.0104	0.2113 ± 0.0110	0.2404 ± 0.0117	0.2842 ± 0.0044
	tdGraphEmbed + ProtoNet	0.6562 ± 0.1882	0.6791 ± 0.1141	0.6271 ± 0.1159	0.4229 ± 0.0463
	Temp-GFSM	$\mathbf{0.7292 \pm 0.0682}$	$\mathbf{0.7917 \pm 0.1278}$	$\mathbf{0.7062 \pm 0.0762}$	$\mathbf{0.6833 \pm 0.0589}$

Table 5.4: Temporal Graph Classification Accuracy on Social Temporal Graphs

Method \ Few-shot Setting		3 way - 5 shot	3 way - 3 shot	3 way - 2 shot	3 way - 1 shot
Graph Kernel	Weisfeiler-Lehman Opt + kNN	0.3631 ± 0.0298	0.2941 ± 0.1023	0.2700 ± 0.1105	0.2133 ± 0.0388
	Neighborhood Hash + kNN	0.3938 ± 0.0371	0.3185 ± 0.1030	0.2178 ± 0.1500	0.3022 ± 0.1137
	Shortest Path + kNN	0.3996 ± 0.0317	0.3296 ± 0.1413	0.3556 ± 0.1139	0.3844 ± 0.0420
Graph Metric Learning	GL2Vec + kNN	0.2716 ± 0.0839	0.2637 ± 0.0482	0.1711 ± 0.0700	0.0000 ± 0.0000
	Graph2Vec + kNN	0.3360 ± 0.0352	0.3756 ± 0.0241	0.2400 ± 0.0149	0.1933 ± 0.0723
	TGAT + kNN	0.0289 ± 0.0096	0.0407 ± 0.0123	0.0333 ± 0.0068	0.0200 ± 0.0199
	CAW + kNN	0.0284 ± 0.0106	0.0378 ± 0.0178	0.0322 ± 0.0099	0.0333 ± 0.0192
	tdGraphEmbed + kNN	0.3600 ± 0.0208	0.3000 ± 0.1310	0.2767 ± 0.1185	0.2267 ± 0.0268
	GL2Vec + Protonet	0.3400 ± 0.0306	0.3822 ± 0.0290	0.2633 ± 0.0606	0.1933 ± 0.0723
	Graph2Vec + Protonet	0.3573 ± 0.0203	0.3711 ± 0.0279	0.2467 ± 0.0380	0.1933 ± 0.0723
	TGAT + ProtoNet	0.3227 ± 0.0171	0.3293 ± 0.0156	0.3243 ± 0.0110	0.3363 ± 0.0010
	CAW + ProtoNet	0.3340 ± 0.0113	0.3333 ± 0.0229	0.3380 ± 0.0155	0.3270 ± 0.0189
	tdGraphEmbed + ProtoNet	0.5083 ± 0.0121	0.4523 ± 0.0353	0.4670 ± 0.0199	0.3973 ± 0.0164
	Temp-GFSM	$\mathbf{0.6161 \pm 0.0139}$	$\mathbf{0.5931 \pm 0.0148}$	$\mathbf{0.6074 \pm 0.0164}$	$\mathbf{0.5605 \pm 0.0201}$

5.1.5 Summary

Graph metric learning methods aim to learn the distance metric over graphs such that similar (e.g., same class) graphs are closer and dissimilar (e.g., different class) graphs are farther apart. This is of critical importance in many graph classification applications such

[7] Cannot get the results within 48 hours.

as drug discovery and epidemics categorization. Most, if not all, graph metric learning techniques consider the input graph as static, and largely ignore the intrinsic dynamics of temporal graphs. However, in practice, a graph typically has heterogeneous dynamics (e.g., microscopic and macroscopic evolution patterns). As such, labeling a temporal graph is usually expensive and also requires background knowledge. To learn a good metric over temporal graphs, we propose a temporal graph metric learning framework, Temp-GFSM. With only a few labeled temporal graphs, Temp-GFSM outputs a good metric that can accurately classify different temporal graphs and be adapted to discover new subspaces for unseen classes. Each proposed component in Temp-GFSM answers the following questions: What patterns are evolving in a temporal graph? How to weigh these patterns to represent the characteristics of different temporal classes? And how to learn the metric with the guidance from only a few labels? Finally, the experimental results on real-world temporal graph classification tasks from various domains show the effectiveness of our Temp-GFSM.

5.2 INCREASING GRAPH CLASSIFICATION EFFICIENCY VIA NATURAL DYNAMICS

5.2.1 Overview and Motivation

Graphs, as a relational structure, model the complex relationships among entities and have attracted much research attention nowadays. To serve various applications, graph neural networks have been extensively studied for their representation learning ability. On the one hand, graph neural networks usually need to build complex neural architectures with hyperparameters to achieve their powerful expressive ability, which is typically a nonlinear process and hard to interpret [268, 269, 270, 271]. On the other hand, graph kernels enjoy the explicit formula and can be convex, leading to solid theoretical results, although their specific form is often hand-crafted and may not be powerful enough to support complicated application scenarios [272, 273]. Hence, *graph neural tangent kernel* (GNTK) [274] has been proposed to fuse graph neural networks and graph kernels, enjoying the benefits of both approaches, i.e., achieving the excellent representation ability while relying on simple computation processes.

However, in the real world, the graph topology and features are inevitably evolving over time, e.g., the user connections and interests in social networks. This temporal evolution brings new challenges to GNTK research as to how the similarity of temporal graphs is measured and how the corresponding kernel matrix is derived. To be more specific, how can we design a temporal graph neural tangent kernel, which not only *has a superior representa-*

tion ability than temporal graph neural networks [152, 159] but also *inherits the expression simplicity and analysis rigorousness of graph neural tangent kernels [274, 275]*?

Hence, we propose **Temp**oral **G**raph **N**eural **T**angent **K**ernel with **G**raphon-**G**uaranteed, or **Temp-G^{3}NTK**. *First*, we propose the kernel matrix computation formula for temporal graphs with time-evolving structures and time-evolving node features, and the corresponding kernel value can be used for classification tasks with generalization error bounded. This proposed kernel method addresses how to measure the similarity between temporal graphs to achieve the accuracy of graph neural networks but without complex neural computational procedures like gradient descent. *Second*, considering the property of temporal graphs, we also prove that when the temporal graph is growing, i.e., the number of nodes increases over time, our Temp-G^{3}NTK kernel will converge in the limit to the graphon NTK value. This result addresses the challenge of adapting graphon neural network [276] and graphon neural tangent kernel [275] to the temporal setting, and, more importantly, demonstrates that Temp-G^{3}NTK has the excellent potential to transfer to large-scale temporal graph data with robust performance. *Third*, in addition to the theoretical analysis, we also design extensive experiments for not only temporal graph classification but also temporal node classification, illustrating the effectiveness of the proposed Temp-G^{3}NTK compared with various state-of-the-art temporal graph representation learning and graph kernel methods.

5.2.2 Preliminary

To extend to the temporal setting, researchers usually model the temporal graph G as a *continuous-time dynamic graph* (CTDG) [37], which is represented as a stream of events, $G = \{(u, v, \mathbf{h}_u(t), \mathbf{h}_v(t), \mathbf{e}_{uv}(t), t)\}_{t=t_0}^{T}$, where an event $(u, v, \mathbf{h}_u(t), \mathbf{h}_v(t), \mathbf{e}_{uv}(t), t)$ indicates that at time t, an edge exists between node u and v, and $\mathbf{h}_u(t)$, $\mathbf{h}_v(t)$, and $\mathbf{e}_{uv}(t)$ are the features of u, v, and (u, v) at time t, respectively. To support different computation requirements, a CTDG G can also be transferred into a *discrete-time dynamic graph* (DTDG) [37], which is a collection of snapshots $G^{(t)}$. To be specific, a snapshot of G at any time $t \geq t_0$, is denoted as $G^{(t)}$, which can be obtained by sequentially updating the initial state of $G^{(t_0)}$ with the event stream, i.e., $G^{(t)} = \{(u, v, \mathbf{h}_u(\bar{t}), \mathbf{h}_v(\bar{t}), \mathbf{e}_{uv}(\bar{t}), \bar{t})\}_{\bar{t}=t_0}^{t}$ given $(t_0 \leq \bar{t} \leq t)$, and temporal graph equals to the last time snapshot, i.e., $G = G^{(T)}$ given the last timestamp is denoted by T.

Let $V^{(t)}, E^{(t)}$ be the sets of vertices and edges of $G^{(t)}$. We denote the temporal neighborhood of node v at time t as $\mathcal{N}^{(t)}(v) = \{(u, \bar{t}) : ((u, v, \mathbf{h}_u(\bar{t}), \mathbf{h}_v(\bar{t}), \mathbf{e}_{uv}\bar{t}) \in G^{(t)}\}$, i.e., a set of nodes u that are involved in an event with v at any time $\bar{t}$ $(t_0 \leq \bar{t} \leq t)$. Note that, in the rest of the paper, we use G to denote an entire temporal graph, and $G^{(t)}$ as a snapshot. For

simplicity, we denote $t_0 = 0$.

Temporal Graph Representation Learning Graph Neural Networks (GNN) are a family of neural architectures for graph representation learning. In general, most GNNs leverage a message-passing framework to compute a node v's representation $\mathbf{h}_v^{(l)}$ at the lth layer. In the static setting, $\mathbf{h}_v^{(l)}$ can be obtained by applying a neighborhood aggregation operator on $\mathbf{h}_u^{(l-1)}$, $\forall u \in \mathcal{N}(v)$, then transforming the aggregated neighborhood information. The graph-level representation can be obtained by applying a pooling function on representations of all nodes, e.g., the summation of all node representations.

To derive Temp-G^{3}NTK, our first step is to compute node representations at time t and then apply a pooling function over all nodes to obtain the graph-level (or snapshot-level) representation. Specifically, we obtain the node representation of v at time t, $\mathbf{h}_v(t)$, by aggregating information from its temporal neighborhood $\mathcal{N}^{(t)}(v)$ as

$$\mathbf{h}_v(t) = c \cdot \sum_{(u,\bar{t}) \in \mathcal{N}^{(t)}(v)} [\mathbf{t}_{enc}(t - \bar{t}) || \mathbf{e}_{uv}(t)] \tag{5.11}$$

where $\mathbf{t}_{enc} : \mathbb{R} \to \mathbb{R}^{d_t}$ is the time encoding function that can be instantiated [152] as $\mathbf{t}_{enc}(\Delta t) = \cos(\Delta t \mathbf{w})$, and $\mathbf{w} \in \mathbb{R}^{d_t}$ with its ith entry $[\mathbf{w}]_i = \alpha^{-(i-1)/\beta}$ and d_t is the dimension of the time representation vector. Operation $[\cdot||\cdot]$ denotes the concatenation. c is the scaling factor, and if we set $c = 1$, then Eq. 5.11 is simply the sum neighborhood aggregation; and if $c = |\frac{1}{|\mathcal{N}^{(t)}(v)|}|$, then Eq. 5.11 would be the average neighborhood aggregation. Note that if the edge features do not exist then we simply set $\mathbf{e}_{uv}(t) = 0$.

After aggregating information from node v's neighborhood at time t as Eq. 5.11, we can input $\mathbf{h}_v(t)$ into L layers of Multilayer Perceptrons (MLPs), where the representation of node v after the lth MLP projection is as

$$\mathbf{h}_v^{(l)}(t) = \sqrt{\frac{2}{d_l}} \sigma\big(\mathbf{W}^{(l)} \mathbf{h}_v^{(l-1)}(t)\big) \tag{5.12}$$

where d_l is the output dimension of the lth MLP layer, σ is a non-linear activation function that can be instantiated as the Rectified Linear Units (ReLU) function.

Furthermore, we denote the graph-level representation of $G^{(t)}$ as $\mathbf{h}_G(t) = \sum_{v \in V^{(t)}} \mathbf{h}_v^{(L)}(t)$.

Graph Neural Tangent Kernel Next, we provide some background on the infinite-width limit of a fully connected deep neural network f_{nn} and derive the definition of NTK and its properties on graphs.

Consider the following settings: given a training dataset of n samples $\{(\mathbf{x}_i, y_i)\}_{i=1}^n$, where $\mathbf{x}_i \in \mathbb{R}^d$ and its label denoted by $y_i \in \mathbb{R}$. Let $f_{nn}(\mathbf{x}, \theta)$ be the output of a fully-connected neural network, with parameters $\theta \in \mathbb{R}^p$ and $\mathbf{x}$ as the input. We train f_{nn} by minimizing the squared loss over the training dataset.

$$\ell(\theta) = \frac{1}{2}\sum_{i=1}^{n}(f_{nn}(\mathbf{x}_i, \theta) - y_i)^2 \tag{5.13}$$

Let $\mathbf{X} \in \mathbb{R}^{n \times d}$ (where $[\mathbf{X}]_i = \mathbf{x}_i$) and $\mathbf{y} \in \mathbb{R}^n$ (where $[\mathbf{y}_i] = y_i$), such that $f_{nn}(\mathbf{X}, \theta)$ would be the prediction of f_{nn}, with parameters θ, over all $\mathbf{x}_i$ of the training set.

Suppose that ℓ is minimized by gradient descent, so the output $f_{nn}(\mathbf{X}, \theta)$ evolves with respect to the training time τ as follows [277].

$$\frac{d\ f_{nn}(\mathbf{X}, \theta(\tau))}{d\tau} = -\mathbf{H}(\tau) \cdot (f_{nn}(\mathbf{X}, \theta(\tau)) - \mathbf{y}) \tag{5.14}$$

where $\theta(\tau)$ is the parameters θ being updated at training time τ based on gradient descent, and $\mathbf{H}(\tau)$ is a $n \times n$ positive definite matrix with its (i, j)-th entry as follows

$$\left\langle \frac{\partial f_{nn}(\mathbf{x}_i, \theta(\tau))}{\partial \theta}, \frac{\partial f_{nn}(\mathbf{x}_j, \theta(\tau))}{\partial \theta} \right\rangle \tag{5.15}$$

Existing works on over-parameterized neural networks [277, 278], [279, 280], and [281] have proven that for infinite-width neural networks, the matrix $\mathbf{H}(\tau)$ remains constant during training, and under random initialization of parameters, the matrix $\mathbf{H}(0)$ converges in probability to a certain deterministic kernel matrix $\mathbf{H}^*$, which is named as Neural Tangent Kernel [281]. Moreover, as proven in [277, 278], the prediction of a fully-trained sufficiently wide ReLU neural network is equivalent to the kernel regression predictor with the kernel matrix $\mathbf{H}^*$.

For the temporal setting, similar to the NTK and the infinite-width neural networks, let f_{temp} denote the aforementioned temporal GNN, and $f_{temp}(G^{(t)}, \mathbf{W})$ be the output of f_{temp} with the input $G^{(t)}$ and parameters $\mathbf{W}$. Given two temporal graphs G and G', at time t, the NTK value corresponds to infinite-width f_{temp}, i.e., in the limit that $d_l \to \infty$, where d_l is the output dimension stated in Eq. 5.12, $l \in [L]$, such that

$$K(G^{(t)}, G'^{(t)}) = \mathbb{E}_{\mathbf{W} \sim \mathcal{N}(0,1)} \left\langle \frac{\partial f(G^{(t)}, \mathbf{W})}{\partial \mathbf{W}}, \frac{\partial f(G'^{(t)}, \mathbf{W})}{\partial \mathbf{W}} \right\rangle \tag{5.16}$$

Then, in the rest of this paper, we can refer to this value $K(G^{(t)}, G'^{(t)})$ as the Temp-G^{3}NTK

value of G and G' at time t. Next, we are ready to introduce and prove our proposed Temp-G^{3}NTK.

5.2.3 Proposed Method

Given two temporal graphs G and G', we propose to compute the Temp-G^{3}NTK value at time t, i.e., $K(G^{(t)}, G'^{(t)})$, with L BLOCK operations [8].

In general, similar to [274, 277], we first recursively compute the node pairwise covariance matrix $\mathbf{\Sigma}^{(l)}$, its derivative $\dot{\mathbf{\Sigma}}^{(l)}$, and the node-pairwise kernel matrix $\mathbf{\Theta}^{(l)}$ that correspond to the lth BLOCK transformation. Finally, the Temp-G^{3}NTK value is obtained by the summation of all of the entries in the kernel matrix of the last BLOCK transformation, i.e., $\mathbf{\Theta}^{(L)}$.

To begin with, we initialize the node pairwise covariance matrix $\mathbf{\Sigma}$ and the kernel matrix $\mathbf{\Sigma}$ at entries u, u' ($u \in V^{(t)}, u' \in V'^{(t)}$) as the inner product of node representations of u, u' at time t, respectively,

$$\mathbf{\Theta}^{(0)}(G^{(t)}, G'^{(t)})_{uu'} = \mathbf{\Sigma}^{(0)}(G^{(t)}, G'^{(t)})_{uu'} = \mathbf{h}_u(t)^T \mathbf{h}'_u(t) \tag{5.17}$$

where $\mathbf{h}_u(t)$ and $\mathbf{h}'_u(t)$ are computed by Eq. 5.11.

Next, we need to compute $\mathbf{\Sigma}^{(l)}$ and $\mathbf{\Theta}^{(l)}$ that correspond to the lth BLOCK operator with ReLU activation function σ. As $\sigma(x) = \max(0, x)$, the derivative of σ is $\dot{\sigma}(x) = \mathbf{1}[x \geq 0]$, where $\mathbf{1}$ is the indicator vector[9]. For $l (1 \leq l \leq L)$, we first define an intermediate covariance matrix as

$$\mathbf{\Lambda}^{(l)}(G^{(t)}, G'^{(t)})_{uu'} = \begin{pmatrix} \mathbf{\Sigma}^{(l-1)}(G^{(t)}, G'^{(t)})_{uu'} & \mathbf{\Sigma}^{(l-1)}(G^{(t)}, G^{(t)})_{uu} \\ \mathbf{\Sigma}^{(l-1)}(G'^{(t)}, G^{(t)})_{u'u} & \mathbf{\Sigma}^{(l-1)}(G'^{(t)}, G'^{(t)})_{u'u'} \end{pmatrix} \tag{5.18}$$

and $\mathbf{\Lambda}^{(l)}(G^{(t)}, G'^{(t)})_{uu'} \in \mathbb{R}^{2\times 2}$.

As the covariance matrix $\mathbf{\Sigma}^{(l)}$represents the i.i.d centered Gaussian Processes of $h_u(t)$ and $h'_u(t)$ after transformed by l BLOCK operations, we can compute $\mathbf{\Sigma}^{(l)}$ and $\dot{\mathbf{\Sigma}}^{(l)}$ based on

[8] We follow the name of "BLOCK" in [274], which can be understood as an iterative transformation operation. The new version for temporal graphs is expressed in Eq. 5.18, Eq. 5.19, Eq. 5.20, and Eq. 5.21.

[9] with the pseudo derivative at 0.

the aforementioned intermediate covariance matrix as

$$\begin{aligned}\mathbf{\Sigma}^{(l)}(G^{(t)}, G'^{(t)})_{uu'} &= \mathbb{E}_{(a,b)\sim\mathcal{N}(0,\mathbf{\Lambda}^{(l)}(G^{(t)},G'^{(t)})_{uu'})}[\sigma(a)\cdot\sigma(b)] \\ &= \frac{\pi - \arccos(\mathbf{\Sigma}^{(l-1)}(G^{(t)}, G'^{(t)})_{uu'})}{2\pi} + \frac{\sqrt{1-(\mathbf{\Sigma}^{(l-1)}(G^{(t)}, G'^{(t)})_{uu'})^2}}{2\pi}\end{aligned} \tag{5.19}$$

with

$$\begin{aligned}\dot{\mathbf{\Sigma}}^{(l)}(G^{(t)}, G'^{(t)})_{uu'} &= \mathbb{E}_{(a,b)\sim\mathcal{N}(0,\mathbf{\Lambda}^{(l)}(G^{(t)},G'^{(t)})_{uu'})}[\dot{\sigma}(a)\cdot\dot{\sigma}(b)] \\ &= \frac{\pi - \arccos(\mathbf{\Sigma}^{(l-1)}(G^{(t)}, G'^{(t)})_{uu'})}{2\pi}\end{aligned} \tag{5.20}$$

Eq. 5.19 and Eq. 5.20 hold due to the closed form of the kernel function with ReLU activation $g(x,y) = \mathbb{E}_w[\sigma'(w^T x)\sigma'(w^T y)] = \left(\frac{1}{2} - \frac{\arccos x^T y}{2\pi}\right)$.

Then, the lth kernel matrix, $\mathbf{\Theta}^{(l)}$, is obtained as

$$\mathbf{\Theta}^{(l)}(G^{(t)}, G'^{(t)})_{uu'} = \mathbf{\Theta}^{(l-1)}(G^{(t)}, G'^{(t)})_{uu'} \cdot \dot{\mathbf{\Sigma}}^{(l)}(G^{(t)}, G'^{(t)})_{uu'} + \mathbf{\Sigma}^{(l-1)}(G^{(t)}, G'^{(t)})_{uu'} \tag{5.21}$$

Finally, the Temp-G^{3}NTK value of G, G' at time t is

$$K(G^{(t)}, G'^{(t)}) = \sum_{v\in V^{(t)}} \sum_{v'\in V'^{(t)}} \mathbf{\Theta}^{(L)}(G^{(t)}, G'^{(t)})_{vv'} \tag{5.22}$$

We perform summation over all entries, and we can obtain the graph embedding by applying a pooling function, e.g., sum pooling, on node-level representations.

The algorithm for computing the Temp-G^{3}NTK kernel is shown in Alg. 5.3.

5.2.4 Empirical Evaluation

Datasets Targeting temporal graph classification, we conduct experiments on one of the most advanced temporal graph benchmarks that have graph-level labels, i.e., TUDataset [260], the four datasets are INFECTIOUS, DBLP, FACEBOOK, and TUMBLR.

Problem Setting For each dataset above, we evaluate the temporal graph classification accuracy by conducting 5-fold cross-validation and then report the mean and standard deviation of test accuracy. To be specific, given a dataset of n temporal graphs $\{G_1, G_2, ..., G_n\}$ and their labels $\{y_1, y_2, ..., y_n\}$, and in all four datasets, label y_i of the temporal graph G_i is already time-aware, which means the value does not change with respect to time. Also, edge

Algorithm 5.3: Computing Temp-G^{3}NTK value between G, G' at time t

Require: node embeddings of G, G' at time t: $\mathbf{h}(t), \mathbf{h}'(t)$; number of BLOCK operations L
Ensure: Temp-G^{3}NTK value between G, G' at time t: K

$K = 0$
for $u \in V^{(t)}$ **do**
  for $u' \in V'^{(t)}$ **do**
    $\mathbf{\Theta}^{(0)}_{u,u'} \leftarrow \mathbf{h}_u(t)^T \mathbf{h}_{u'}(t)$;
    $\mathbf{\Sigma}^{(0)}_{u,u'} \leftarrow \mathbf{h}_u(t)^T \mathbf{h}_{u'}(t)$;
    for $l \in [1, \ldots, L]$ **do**
      $\mathbf{\Sigma}^{(l)}_{u,u'} \leftarrow \frac{\sqrt{1-\arccos\left(\mathbf{\Sigma}^{(l-1)}_{u,u'}\right)^2}}{2\pi}$
      $\dot{\mathbf{\Sigma}}^{(l)}_{u,u'} \leftarrow \frac{\pi-\arccos\left(\mathbf{\Sigma}^{(l-1)}_{u,u'}\right)}{2\pi}$
      $\mathbf{\Theta}^{(l)}_{u,u'} \leftarrow \mathbf{\Theta}^{(l-1)}_{u,u'} \cdot \dot{\mathbf{\Sigma}}^{(l-1)}_{u,u'} + \mathbf{\Sigma}^{(l-1)}_{u,u'}$
    end for
  end for
end for
for $u \in V^{(t)}$ **do**
  for $u' \in V'^{(t)}$ **do**
    $K \leftarrow K + \mathbf{\Theta}^{(L)}_{u,u'}$
  end for
end for

features are not provided in these four datasets, and we apply the Temp-G^{3}NTK formula with plain time encoding as stated in Eq. 5.11.

Baselines. We compare Temp-G^{3}NTK with a range of graph classification algorithms: (1) Graph Kernels, including WL-Subtree Kernel [282], Random Walk Kernel [283], and Shortest-Path Kernel [261]; (2) Graph Representation Learning methods, including Graph2Vec [265], NetLSD [233], GL2Vec [264]; and (3) Temporal Graph Representation Learning algorithms, including TGN [159] and GraphMixer [152].

Results The graph classification results are shown in Table 5.5, and the best test accuracy is highlighted in bold. Our method, Temp-G^{3}NTK, outperforms the other methods on all four datasets. In particular, the most notable gap between Temp-G^{3}NTK and the other methods lies in the FACEBOOK dataset, where Temp-G^{3}NTK gains 70% accuracy. In addition, as the label for each graph remains unchanged, we evaluate the performance of baseline algorithms on different timestamps until the end of the temporal graph, and report their highest accuracy score in Table 5.5.

Table 5.5: Comparison of Temporal Graph Classification Accuracy.

Method	infectious	dblp	facebook	tumblr
WL-Subtree [282]	0.600 ± 0.044	0.520 ± 0.068	0.650 ± 0.075	0.570 ± 0.121
Shortest Path [261]	0.670 ± 0.075	0.560 ± 0.049	0.560 ± 0.086	0.580 ± 0.143
Random Walk [283]	0.670 ± 0.073	0.530 ± 0.058	0.590 ± 0.093	0.580 ± 0.112
Graph2Vec [265]	0.565 ± 0.081	0.539 ± 0.031	0.538 ± 0.028	0.547 ± 0.071
NetLSD [233]	0.625 ± 0.061	0.558 ± 0.035	0.535 ± 0.011	0.552 ± 0.046
GL2Vec [264]	0.545 ± 0.051	0.562 ± 0.030	0.538 ± 0.031	0.558 ± 0.080
GraphMixer [152]	0.500 ± 0.000	0.563 ± 0.011	0.561 ± 0.023	0.509 ± 0.508
TGN [159]	0.520 ± 0.019	0.580 ± 0.003	0.559 ± 0.018	0.517 ± 0.025
Temp-G^{3}NTK	**0.740 ± 0.058**	**0.600 ± 0.063**	**0.700 ± 0.138**	**0.630 ± 0.068**

5.2.5 Summary

Graph Neural Tangent Kernel (GNTK) fuses graph neural networks and graph kernels, simplifies the process of graph representation learning, interprets the training dynamics of graph neural networks, and serves various applications like protein identification, image segmentation, and social network analysis. In practice, graph data carries complex information among entities that inevitably evolves over time, and previous static graph neural tangent kernel methods may be stuck in the sub-optimal solution in terms of both effectiveness and efficiency. As a result, extending the advantage of GNTK to temporal graphs becomes a critical problem. To this end, we propose the temporal graph neural tangent kernel, which not only extends the simplicity and interpretation ability of GNTK to the temporal setting but also leads to rigorous temporal graph classification error bounds. Furthermore, we prove that when the input temporal graph grows over time in the number of nodes, our temporal graph neural tangent kernel will converge in the limit to the graphon NTK value, which implies the transferability and robustness of the proposed kernel method, named Temporal Graph Neural Tangent Kernel with Graphon-Guaranteed or Temp-G^{3}NTK. In addition to the theoretical analysis, we also perform extensive experiments, not only demonstrating the superiority of Temp-G^{3}NTK in the temporal graph classification task, but also showing that Temp-G^{3}NTK can achieve very competitive performance in node-level tasks like node classification compared with various SOTA graph kernel and representation learning baselines.

5.3 ENHANCING GRAPH MATCHING AND ALIGNMENT EFFICIENCY AND ACCURACY VIA NATURAL DYNAMICS

5.3.1 Overview and Motivation

Knowledge graph alignment, i.e., searching node correspondence (or similarity) across knowledge graphs, has a wide application on web mining, social network analysis, question answering, protein-protein interaction prediction, etc [79, 80, 284, 285, 286, 286, 287, 288, 289, 290, 291, 292]. To be specific, given the adjacency matrices, node attributes, and edge attributes of two knowledge graphs, retrieving one-to-one node pair between two knowledge graphs is the main goal [284].

Static Alignment Solution using Personalized PageRank Given two graphs $\mathcal{G}_1 = \{\mathbf{A}_1, \mathbf{N}_1, \mathbf{E}_1\}$ and $\mathcal{G}_2 = \{\mathbf{A}_2, \mathbf{N}_2, \mathbf{E}_2\}$, where $\mathbf{A}$, $\mathbf{N}$, $\mathbf{E}$ stands for adjacency matrix, node attribute matrix (i.e., $\mathbf{N}(a, a)$ is attribute of node a), and edge attribute matrix (i.e., $\mathbf{E}(a, b)$ is attribute of edge ab), respectively. n_1 (or n_2) is the number of nodes in graph $\mathcal{G}_1$ (or $\mathcal{G}_2$). The similarity-based alignment solution $\mathbf{s} \in \mathbb{R}^{n_1 n_2}$ [284] can be rewritten in the following form as personalized PageRank.

$$\mathbf{s} = \alpha \widetilde{\mathbf{W}} \mathbf{s} + (1 - \alpha)\mathbf{h} \tag{5.23}$$

where $\widetilde{\mathbf{W}} = \mathbf{D}^{-\frac{1}{2}}\mathbf{N}(\mathbf{E} \odot (\mathbf{A_1} \otimes \mathbf{A_2}))\mathbf{N}\mathbf{D}^{-\frac{1}{2}}$. Suppose we have K node attributes and L edge attributes in total (i.e., $\mathbf{N}_i^k(a, a) = 1$ denotes that node a in $\mathcal{G}_i$ has the attribute k, and $\mathbf{E}_i^l(a, b) = 1$ denotes edge ab in $\mathcal{G}_i$ has attribute l), then the corresponding matrices are $\mathbf{N} = \sum_{k=1}^{K} \mathbf{N}_1^k \otimes \mathbf{N}_2^k$ and $\mathbf{E} = \sum_{l=1}^{L} \mathbf{E}_1^l \otimes \mathbf{E}_2^l$. Matrix $\mathbf{D} = diag(\sum_{k,k'=1}^{K} \sum_{l=1}^{L} (\mathbf{N}_1^k(\mathbf{E}_1^l \odot \mathbf{A}_1)\mathbf{N}_1^{k'}\mathbf{1}) \otimes (\mathbf{N}_2^k(\mathbf{E}_2^l \odot \mathbf{A}_2)\mathbf{N}_2^{k'}\mathbf{1}))$ is the diagonal degree matrix of $\mathbf{W} = \mathbf{N}(\mathbf{E} \odot (\mathbf{A}_1 \otimes \mathbf{A}_2))\mathbf{N}$.

Again, Eq. 5.23 is in the same form as the standard personalized PageRank formula [16], if we view the concatenated two knowledge graphs as one graph. Each entry of $\mathbf{s} \in \mathbb{R}^{n_1 n_2}$ represents the similarity score of a node pair across the two graphs. Then we can apply a heuristic greedy match algorithm [293] to obtain the final alignment based on the ranking, i.e., similarity vector $\mathbf{s}$.

Fully Dynamic Setting of Personalized PageRank In the real world, one challenge of knowledge graph alignment is that, the graphs are changing from time to time, but re-computing the alignment solution from scratch is time-consuming. There are many kinds of changes that can occur, making the graph alignment task even more complex. For instance, in the LinkedIn knowledge graph, a user can start following someone (edge insertion), change

his or her occupation (node attribute change), become CEO of a company from the employee (edge attribute change), or change his or her username (prior knowledge change). Good prior knowledge (or pre-knowledge) about pre-aligned anchors across knowledge graphs is important for the alignment task because it reduces the alignment search space for improbable nodes and strengthens the alignment between the nodes that seem to be from the same entity. For dynamic networks, the pre-knowledge is important to be dynamic as well, since the networks are able to change so many times that any starting static pre-knowledge will become distorted.

Therefore, we focus on the fully dynamic setting and seek for the personalized PageRank, i.e., each component is dependent on time, as follows.

$$\mathbf{v}^{(t)} = \alpha \mathbf{P}^{(t)} \mathbf{v}^{(t)} + (1 - \alpha)\mathbf{h}^{(t)} \tag{5.24}$$

The transition matrix of $\mathbf{P}^{(t)}$ is dependent on time because the underlying graph structure is evolving, which can be represented as $\{\mathcal{G}^{(1)}, \mathcal{G}^{(2)}, \ldots, \mathcal{G}^{(T)}\}$, $t \in \{1, 2, \ldots, T\}$. The stochastic vector $h^{(t)}$ is dependent on time because we allow end-users to change the interested seed nodes along with the evolving underlying structures. The varying $h^{(t)}$ is untouched by current PageRank tracking methods [13, 87, 103, 294, 295] but has a huge potential to serve real-world downstream tasks such as dynamic knowledge graph alignment. With varying $\mathbf{P}^{(t)}$ and $h^{(t)}$, we aim to track the exact solution $\mathbf{v}^{(t)}$ at each timestamp efficiently instead of solving it from scratch like the power iteration.

5.3.2 Problem Definition

Problem 5.2. Personalized PageRank in the Fully Dynamic Setting

Input: (i) a constant tolerance ϵ, (ii) a sequence of evolving graph structures, represented by transition matrices $\{\mathbf{P}^{(1)}, \mathbf{P}^{(2)}, \ldots, \mathbf{P}^{(T)}\}$, (iii) a sequence of varying end-user interests, represented by stochastic vectors $\{\mathbf{h}^{(1)}, \mathbf{h}^{(2)}, \ldots, \mathbf{h}^{(T)}\}$.

Output: a sequence of personalized PageRank vectors $\mathbf{v}^{(t)}$ whose error bound is not greater than ϵ, for $t \in \{1, 2, \ldots, T\}$.

5.3.3 Proposed Method

In the fully dynamic setting, the major difference from the partially dynamic setting is that we allow the stochastic vector to evolve, i.e., $\mathbf{h}^{(t)} \neq \mathbf{h}^{(t+1)}$, to reflect the varying

interests from end-users when the latent graph structure updates. However, each entry in the stochastic vector is a source node for random walks with restart [296], and we do not limit $\mathbf{h}^{(t)}$ to be a one-hot vector. How to track the multi-source personalized PageRank is complex, and how it relates to multiple single-source tracking is not clear until now.

Hence, we propose EvePPR to track the personalized PageRank vector in the fully dynamic setting and allow $\mathbf{h}^{(t)}$ to be a multi-source stochastic vector. For EvePPR, we prove that tracking multi-source personalized PageRank can be decomposed into tracking multiple single-source personalized PageRank vectors, by viewing each single-source tracking as an intermediate step of random walks. To be specific, for each single source node i, we store a personalized PageRank vector, i.e., $\mathbf{q}_i$, which is initially computed from the first transition matrix $\mathbf{P}^{(t=0)}$. Then, given a timestamp t when the single source node i changes, we retrieve $\mathbf{q}_i$ and update it (e.g., by OSP [87]), then we involve the updated $\mathbf{q}_i$ to the tracking process of $\mathbf{v}^{(t)}$. The detailed operations are summarized in Algorithm 5.4.

The Algorithm 5.4 is easy to understand and has two sequential parts, i.e., the pre-computing phase and the tracking phase. In Step 2–5, at time $t = 0$, we initialize a dictionary $\mathcal{M}$ for storing the current transition matrix $\mathbf{P}^{(t=0)}$ for each node $i \in \{1, 2, \ldots, n\}$. With which transition matrix, the single-source personalized PageRank $\mathbf{q}_i$ is computed by the power iteration method by setting each node i as the seed node with the one-hot vector $\mathbf{e}_i$ (i.e., the i-th entry equals to 1, others equal to 0). In the future, the transition matrix stored in $\mathcal{M}$ will be updated if the corresponding seed node is updated. In Step 8, we first call OSP [87] to compute an intermediate vector $\mathbf{v}_{\text{mid}}$ for time t, which is tracked essentially based on the outdated stochastic vector $\mathbf{h}^{(t-1)}$, because OSP is not able to track personalized PageRank vector when $\mathbf{h}$ evolves. After that, when we detect the evolving seed nodes in Step 9, we can then track multiple single-source PageRank vectors in Step 12, by retrieving its last active transition matrix in $\mathcal{M}$ and updating it in Step 13. Then, in Step 14, we add each single-source tracked vector to $\mathbf{v}_{\text{mid}}$, and finally obtain $\mathbf{v}^{(t)}$.

The theoretical analysis of exactness, time complexity, and error bound of the proposed EvePPR are expressed as follows.

Theorem 5.1 (Exactness of EvePPR)**.** At each timestamp t of the fully dynamic setting, EvePPR outputs the exact personalized PageRank vector $\mathbf{v}^{(t)}$ that satisfies $\mathbf{v}^{(t)} = \alpha\mathbf{P}^{(\mathrm{t})}\mathbf{v}^{(t)} + (1-\alpha)\mathbf{h}^{(t)}$.

Proof. Firstly, according to the power iteration for solving $\mathbf{q}_{\mathrm{i}}$ in the pre-computing phase,

$$\mathbf{q}_i = \alpha\mathbf{P}^{(0)}\mathbf{q}_i + (1-\alpha)\mathbf{e}_{\mathrm{i}} \iff \mathbf{q}_i = (1-\alpha)\sum_{k=0}^{\infty}(\alpha\mathbf{P}^{(0)})^k\mathbf{e}_i \tag{5.25}$$

Algorithm 5.4: EvePPR

Require: a sequence of transition matrices $\{\mathbf{P}^{(0)}, \mathbf{P}^{(1)}, \ldots, \mathbf{P}^{(T)}\}$, a sequence of stochastic vectors $\{\mathbf{h}^{(0)}, \mathbf{h}^{(1)}, \ldots, \mathbf{h}^{(T)}\}$, damping factor α; error tolerance ϵ

Ensure: a sequence of personalized PageRank vectors $\{\mathbf{v}^{(0)}, \mathbf{v}^{(1)}, \ldots, \mathbf{v}^{(T)}\}$

/* Pre-computing Phase */

Compute $\mathbf{v}^{(0)} = PowerIteration(\mathbf{P}^{(0)}, \mathbf{h}^{(0)}, \alpha, \epsilon)$

Construct dictionary $\mathcal{M}$ with n keys $\{1, 2, ..., n\}$

for iteration $i = 1$; $i \leq n$; i++ **do**

  $\mathbf{q}_i = PowerIteration(\mathbf{P}^{(0)}, \mathbf{e}_i, \alpha, \epsilon)$

  Set $\mathcal{M}[i] = \mathbf{P}^{(0)}$

end for

/* Tracking Phase */

for timestamp $t = 1$; $t \leq T$; t++ **do**

  $\mathbf{v}_{\text{mid}} = OSP(\mathbf{v}^{(t-1)}, \mathbf{P}^{(t-1)}, \mathbf{P}^{(t)}, \alpha, \epsilon)$

  $\Delta\mathbf{h} = \mathbf{h}^{(t)} - \mathbf{h}^{(t-1)}$

  for iteration $i = 1$; $i \leq n$; i++ **do**

    if $\Delta\mathbf{h}(i) \neq 0$ **then**

      $\mathbf{q}_i = OSP(\mathbf{q_i}, \mathcal{M}[i], \mathbf{P}^{(\mathrm{t})}, \alpha, \epsilon)$

      $\mathcal{M}[i] = \mathbf{P}^{(t)}$

      $\mathbf{v}_{\text{mid}} \ += \Delta\mathbf{h}(i)\mathbf{q}_i$

    end if

  end for

  $\mathbf{v}^{(\mathrm{t})} = \mathbf{v}_{\text{mid}}$

end for

Return: $\mathbf{v}^{(t)}$ for $t \in \{0, 1, \ldots, T\}$

Then, in the tracking phase for timestamp t, $\mathbf{v}_{\text{mid}}$ is first figured out as below,

$$\mathbf{v}_{\text{mid}} = OSP(\mathbf{v}, \mathbf{P}^{(t-1)}, \mathbf{P}^{(t)}, \alpha, \epsilon) = (1-\alpha)\sum_{k=0}^{\infty}(\alpha\mathbf{P}^{(t)})^k\mathbf{h}^{(t-1)} \tag{5.26}$$

and when a certain $\mathbf{q}_i$ is called, it will be updated as follows.

$$\mathbf{q}_i = OSP(\mathbf{q_i}, \mathcal{M}[i], \mathbf{P}^{(t)}, \alpha, \epsilon) = (1-\alpha)\sum_{k=0}^{\infty}(\alpha\mathbf{P}^{(t)})^k\mathbf{e}_i \tag{5.27}$$

Assume l seed nodes evolve, then the one-hot decomposition of $\Delta\mathbf{h} = \mathbf{h}^{(\mathrm{t})} - \mathbf{h}^{(\mathrm{t}-1)}$ is rewritten as follows.

$$\Delta\mathbf{h} = \sum_{j=1}^{l}\Delta\mathbf{h}(j)\mathbf{e}_j \tag{5.28}$$

Finally, $\mathbf{v}^{(t)}$ calculated by EvePPR is

$$\begin{aligned}
\mathbf{v}^{(\mathrm{t})} &= \mathbf{v}_{\mathrm{mid}} + \sum_{j=1}^{l} \Delta\mathbf{h}(j)\mathbf{q}_j \\
&= (1-\alpha)\sum_{k=0}^{\infty}(\alpha\mathbf{P}^{(t)})^k\mathbf{h}^{(t-1)} + \sum_{j=1}^{l}\Delta\mathbf{h}(j)((1-\alpha)\sum_{k=0}^{\infty}(\alpha\mathbf{P}^{(t)})^k\mathbf{e}_j) \\
&= ((1-\alpha)\sum_{k=0}^{\infty}(\alpha\mathbf{P}^{(t)})^k)(\mathbf{h}^{(t-1)} + \sum_{j=1}^{l}\Delta\mathbf{h}(j)\mathbf{e}_j) \\
&= (1-\alpha)\sum_{k=0}^{\infty}(\alpha\mathbf{P}^{(t)})^k\mathbf{h}^{(t)} \\
&= (1-\alpha)(\mathbf{I}-\alpha\mathbf{P}^{(t)})^{-1}\mathbf{h}^{(t)}
\end{aligned} \tag{5.29}$$

Hence, $\mathbf{v}^{(t)} = \alpha\mathbf{P}^{(\mathrm{t})}\mathbf{v}^{(t)} + (1-\alpha)\mathbf{h}^{(t)}$ gets proved for EvePPR. QED.

Theorem 5.2 (Time Complexity of EvePPR)**.** At timestamp t of the fully dynamic setting, EvePPR outputting personalized PageRank $\mathbf{v}^{(t)}$ costs $O(m(l+1)log_\alpha\epsilon)$, where m is the number of non-zero entries of $\mathbf{P}^{(t)}$, l is the number of non-zero entries of $\mathbf{h}^{(t)}$, ϵ is the tolerance.

Proof. In a single OSP calculation, when $\mathbf{P}^{(t-1)}$ is updated to $\mathbf{P}^{(t)}$, $\mathbf{x}^{(i)}_{\mathrm{offset}}$ will be computed iteratively through $\mathbf{x}^{(i)}_{\mathrm{offset}} = \alpha\mathbf{P}^{(t)}\mathbf{x}^{(i-1)}_{\mathrm{offset}}$, which takes $O(m)$, where m is the number of non-zero entries in $\mathbf{P}^{(t)}$. OSP stops when the error tolerance is achieved, i.e., $\|\mathbf{x}^{(i)}_{\mathrm{offset}}\|_1 = \|(\alpha\mathbf{P}^{(t)})^i\mathbf{x}^{(0)}_{\mathrm{offset}}\|_1 = \alpha^i\|(\mathbf{P}^{(t)})^i\mathbf{x}^{(0)}_{\mathrm{offset}}\|_1 \leq \epsilon$. In PageRank setting, we have $\|\mathbf{P}^{(t)}\|_1 \leq 1$, then $i = \log_\alpha \frac{\epsilon}{\|\mathbf{x}^{(0)}_{\mathrm{offset}}\|_1}$ suffices. $\|\mathbf{x}^{(0)}_{\mathrm{offset}}\|_1 = \|\alpha(\mathbf{P}^{(t)} - \mathbf{P}^{(t-1)})\mathbf{v}^{(t)}\|_1 \leq 2\alpha\|\mathbf{v}^{(t)}\|_1$ Therefore, the number of iterations in one OSP is in $O(\log_\alpha \epsilon)$, the time complexity of a single OSP is in $O(m\log_\alpha\epsilon)$. EvePPR calls OSP $l+1$ times, where l is the number of non-zero entries in $\Delta\mathbf{h} = \mathbf{h}^{(t)} - \mathbf{h}^{(t-1)}$. Therefore, the number of iterations in one EvePPR is in $O((l+1)\log_\alpha\epsilon)$, and the time complexity of EvePPR is in $O(m(l+1)\log_\alpha\epsilon)$. QED.

Theorem 5.3 (Error Bound of EvePPR in Practice)**.** At timestamp t in the fully dynamic setting, if OSP only works k iterations, the PageRank tracking error of EvePPR is bounded by $\frac{\alpha\epsilon}{1-\alpha}(1+\|\Delta\mathbf{h}\|_1)$, with $\Delta\mathbf{h} = \mathbf{h}^{(t)} - \mathbf{h}^{(t-1)}$ and ϵ is the tolerance.

Proof. If OSP terminates at the k-th iteration, then the remaining error $O(error\ of\ OSP)$

is expressed as follows.

$$
\begin{aligned}
O(error\ of\ OSP) &\leq \| \sum_{i=k+1}^{\infty} \mathbf{x}_{\text{offset}}^{(i)} \|_1 \leq \sum_{i=k+1}^{\infty} \alpha^{i-k} \|\mathbf{x}_{\text{offset}}^{(k)}\| \\
&\leq \sum_{i=1}^{\infty} \alpha^i \epsilon = \frac{\alpha\epsilon}{1-\alpha}
\end{aligned}
\tag{5.30}
$$

Then, the error bound of EvePPR is

$$
\begin{aligned}
O(error) &\leq O(error\ of\ OSP) + \sum_{j=1}^{l} |\Delta\mathbf{h}(j)|\, O(error\ of\ OSP) \\
&= \frac{\alpha\epsilon}{1-\alpha}(1 + \|\Delta\mathbf{h}\|_1)
\end{aligned}
\tag{5.31}
$$

where j denotes the index of non-zero entries in $\Delta\mathbf{h}$. QED.

5.3.4 Empirical Evaluation

PageRank Tracking Error in Fully Dynamic Setting After proving EvePPR theoretically, we now empirically verify both tracking effectiveness and efficiency with previous PageRank methods in a real-world dataset MathOverflow [297]. This widely-used network dataset is dynamics and has 24,818 nodes and 506,550 edges. We then choose OSP [87] and Gauss-Southwell [103] as two state-of-the-art baseline methods to track PageRank dynamically.

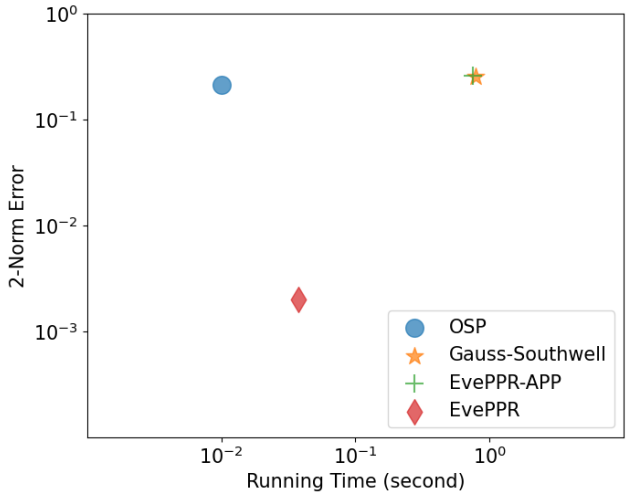


Figure 5.5: Tracking Error and Running Time of Different PageRank Algorithms in MathOverflow Network.

Starting from the $100,000$-th edge[10], we aggregate 20,000 edges for each timestamp. We calculate an exact single-source PageRank vector when the graph has the first $100,000$ edges. Then for each following timestamp, we update the transition matrix according to the newly added edges. As for the stochastic vector, we let the seed node pass sequentially (i.e., seed node is i at t and $i+1$ at $t+1$). We evaluate the L2-norm tracking error and running time of each method on each timestamp. After tracking 5 timestamps, the average tracking error and running time are shown in Figure 5.5. The lower L2-norm error stands for the higher tracking accuracy between two PageRank vectors (i.e., the tracked PageRank v.s. the ground-truth PageRank by power iterations), and the lower running time stands for higher computational efficiency. It can be seen that EvePPR outperforms Gauss-Southwell on both accuracy and efficiency, while EvePPR-APP can achieve similar performance to Gauss-Southwell. Compared to OSP, EvePPR outperforms OSP on the accuracy, with a slight sacrifice on running time.

Knowledge Graph Alignment in the Fully Dyanmics Setting *Datasets.* The real-world temporal networks (or graphs) [11] are summarized in Table 5.6, which are used to compare the effectiveness and efficiency of the proposed EvePPR and EvePPR-APP with other state-of-the-art baseline methods. We employ three real-world temporal networks after necessary preprocessing. MovieLens-1M [298] is a bipartite network containing one million movie ratings, where an edge between a user and a movie shows which level the user rates the movie. Bitcoin Alpha [298, 299] is a user-user trust/distrust network from the Bitcoin Alpha platform, where each edge is labeled from -10 to +10, indicating the trust level between users. WikiLens [298, 300] is also a bipartite rating network, where an edge represents a rating. To demonstrate the effectiveness of EvePPR and EvePPR-APP on node attribute changes and edge attribute changes, we extract synthetic temporal networks with such changes on each of the three real-world datasets mentioned.

Table 5.6: Statistics of Real-World Graphs

Graphs	Format	$\|V\|$	$\|E\|$	Time Span
MovieLens-1M	Bipartite	9,746	1,000,209	35 months
Bitcoin Alpha	Unipartite	3,783	24,186	64 months
WikiLens	Bipartite	5,437	26,937	46 months

Baselines. We choose several state-of-the-art methods for graph alignment as the base-

[10]In MathOverflow, edges from approximately the $300,000$-th are just identically repeating with different timestamps

[11]In experiments, we call "network" and "graph" interchangeably.

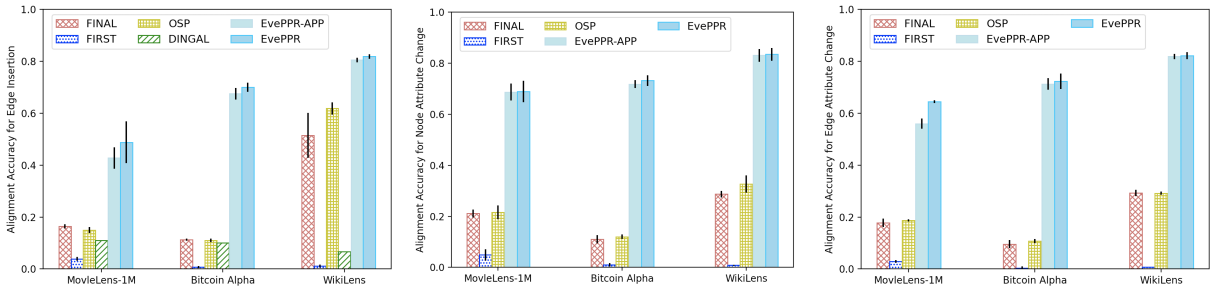
lines of our experiments. We compare static attributed network alignment algorithm (FINAL [284]), dynamic attributed subgraph matching algorithm based on eigenvalue decomposition (FIRST [301]), dynamic knowledge graph alignment based on graph neural networks (DINGAL [80]), dynamic random walk with restart based on offset score propagation (OSP [87]), together with our EvePPR and EvePPR-APP (the approximation version of EvePPR).

Extract the query graph. For each dataset, we extract an amount of nodes and the edges between them, i.e., a temporal subgraph of the original temporal network, to form the query network. The size of the extracted query graph of each dataset can be found in Table 5.7. Since we are actually tracking similarities between node pairs, the scale of personalized PageRank tracking is the multiplication of numbers of nodes in the original graph and the query graph, i.e., 877,140, 378,300, 815,550 for MovieLens-1M, Bitcoin Alpha, WikiLens respectively.

Table 5.7: Statistics of Extracted Query Graphs

Extracted Subgraphs	Format	$\|V\|$	$\|E\|$
MovieLens-1M	Bipartite	90	375
Bitcoin Alpha	Unipartite	100	423
WikiLens	Bipartite	150	553

Knowledge Graph Alignment Evaluation Metric. The network alignment task is, for a certain dataset, to find the best alignment of the extracted query graph in the original network. The alignment accuracy is defined to be the portion of nodes in the query graph whose top k alignment of its ranking list (i.e., a list consisting of nodes in the original graph, sorted by the similarity) contains the ground-truth [284]. In experiments, we set $k = 1$. Also, we set three dynamic network alignment scenarios as follows. In each scenario, we allow the query graph and the original graph to evolve simultaneously.



(a) Edge Insertion (b) Node Attribute Change (c) Edge Attribute Change

Figure 5.6: Alignment Accuracy under Three Dynamic Scenarios

- Edge Insertion: For MovieLens-1M and Bitcoin Alpha datasets, we take the first-occurred 90% of edges as the initial network to start tracking. For WikiLens, this ratio is 20%. When an edge is inserted in the original network and the insertion position is in the scope of the extracted query network, we also add that edge into the query network.
- Node Attribute Change: For each dataset, we randomly extract a sequence of nodes and change their attributes. We take the last timestamp networks in the Edge Insertion scenario as the starting networks, then apply node attribute changes with other properties (i.e., graph structures, edge attributes) unchanged.
- Edge Attribute Change: For each dataset, we randomly extract a sequence of edges and change their attributes. Similar to the Node Attribute Change scenario, we take the last time networks in the Edge Insertion scenario as the starting networks. If an edge to be changed is in both the original and query networks, we change the attribute of that edge in both networks.

When a change occurs in the extracted query network, we mark a timestamp and activate the tracking methods. The numbers of timestamps for the three datasets in the edge insertion scenario are 187, 39, and 33, respectively. For each timestamp, we first update the transition matrix and stochastic (prior knowledge) vector, and then apply the tracking methods. We evaluate the performance of EvePPR and the approximated version (EvePPR-APP) by comparing their alignment accuracy with other baselines. We uniformly choose some timestamps and calculate the mean and standard deviation of the accuracy at those timestamps. The experiment results are illustrated in Figure 5.6. First, both EvePPR and EvePPR-APP outperform the existing baseline methods on accuracy in all scenarios. Second, EvePPR-APP can achieve closely but not as accurately as EvePPR, validating that EvePPR can track exact personalized PageRank vector when the transition matrix and stochastic vector change simultaneously. In the following section, we will compare the efficiency of methods and show that EvePPR-APP is a good trade-off between accuracy and efficiency. Third, the accuracy of the node attribute change scenario and edge attribute change scenario is higher than the edge insertion scenario. This is because the attribute change scenarios start from the final network of edge insertion scenario, and hence they have more information to use.

5.3.5 Summary

Personalized PageRank has been proven as an effective solution in many applications such as web page search, recommendation, etc. However, in the real world, the setting

of personalized PageRank is usually dynamic like the evolving World Wide Web. On the one hand, the outdated PageRank solution can be sub-optimal for ignoring the evolution pattern. On the other hand, solving the solution from scratch at each timestamp causes costly computation complexity. Hence, we solve the Personalized PageRank effectively and efficiently in a fully dynamic setting, i.e., *every component in the Personalized PageRank formula is dependent on time*. To this end, we propose the EvePPR method that can track the exact personalized PageRank solution at each timestamp in the fully dynamic setting, and we theoretically and empirically prove the accuracy and time complexity of EvePPR. Moreover, we apply EvePPR to solve the dynamic knowledge graph alignment task, where a fully dynamic setting is necessary but complex. The experiments show that EvePPR outperforms the state-of-the-art baselines for similar node retrieval across graphs.

CHAPTER 6: EMPOWERING COMBINED-LEVEL GRAPH INTELLIGENCE VIA NATURAL AND ARTIFICIAL DYNAMICS

In this chapter, based on combined-level graph intelligence and knowledge discovery, we have two sections introducing how natural dynamics help discover graph structure in time series data to improve time series forecasting and time series anomaly detection (Section 6.1) and how natural + artificial dynamics help generator models to generate privacy-guaranteed and utility-preserved temporal heterogeneous graphs (Section 6.2).

6.1 FINE-GRAINED CAUSAL GRAPH STRUCTURE DISCOVERY VIA NATURAL DYNAMICS

6.1.1 Overview and Motivation

Structure learning in time series has recently gained much attention, such as [302, 303, 304, 305, 306, 307, 308, 309, 310, 311, 312, 313]. Among others, causal graphs as a directed acrylic graph structure provide more explicit and interpretable correlations between variables, thus enabling a better understanding of the underlying physical mechanisms and dynamic systems for time series [314, 315]. As a widely applied causal structure in time series understanding and explanation, Granger Causality [316, 317] discovers causal relations among variables in an autoregressive (or time-lagged) manner. The discovered Granger Causal structures can help many time series analysis tasks, like building parsimonious prediction models such as Earth System [318]. Moreover, real-world time series data can have many variables, and their causal relations can be even more complex, i.e., non-linear and instantaneous, which require complex causality discovery beyond the classic Granger model. Although some nascent non-linear (or neural) Granger models have been proposed [308, 309, 310, 319, 320, 321, 322, 323], how to effectively integrate instantaneous causal effects with neural Granger models has the great research potential [324, 325, 326, 327, 328] and remains largely underexplored [312, 323].

Motivated by the above analysis, we start from the tensor time series data as shown in Figure 6.1(a), in which the 3D structure contains higher dimensions than typical 2D multivariate time series data. For example, tensor time series can represent multivariate climate time series data (e.g., time plus temperature, wind, and atmospheric water content) with corresponding spatial information (e.g., longitude, latitude, and geocode). After that, we aim to build a comprehensive causality model for this tensor time series, which could not only capture non-linear and time-lagged causality (like the Granger model [310, 316]) but also

offset the ignored instantaneous causal effects at each timestamp, as shown in Figure 6.1(b). Our goal is to leverage the discovered comprehensive causality to understand the trend and latent distribution of the historical tensor time series and finally contribute to the analysis tasks like tensor time series forecasting and anomaly detection.

To this end, we first propose a comprehensive causal model named Time-Respecting Bayesian Network Augmented Neural Granger Causality, i.e., TBN Granger Causality. Theoretically, discovering TBN Granger Causality relies on a bi-level optimization. The inner optimization discovers a sequence of Bayesian Networks at each timestamp t respectively for representing the instantaneous causal effects among variables (i.e., which causality is responsible for the instantaneous feature generation). Then, the outer optimization realizes integrating time-respecting Bayesian Networks with time-lagged neural Granger causality in an autoregressive manner.

Second, to embed TBN Granger Causality into guiding the tensor time series analysis tasks like forecasting and anomaly detection, we propose an end-to-end deep generative model, called **T**ime-**A**ugmented **C**ausal Time **S**eries **A**nalysi**S** Model, i.e., TacSas. TacSas fits more real-world application scenarios (e.g., climate or transportation) by investigating how to capture good causal structures *without* the ground-truth structures guidance. Furthermore, TacSas is end-to-end, meaning that it can not only discover TBN Granger Causality from the observed time series but also seamlessly use the discovery to forecast future time series and detect possible anomalies.

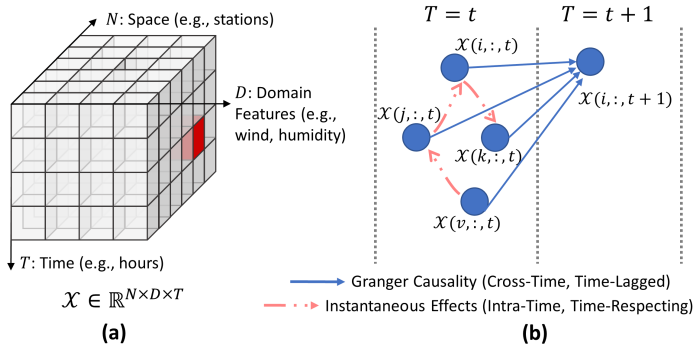


Figure 6.1: (a) Tensor Time-Series Data: The Red Cell Means the Possible Anomaly. (b) Visualization of (Neural) Granger Causality's Time-Lagged Property without Instantaneous Effects.

To evaluate TacSas, we first use the synthetic benchmark, Lorenz-96 [329], to verify that TacSas can indeed discover ground-truth causal structures with high accuracy. Then, we extend to the real-world setting and test if TacSas can utilize the discovered causality to conduct tensor time series forecasting and identify anomalies. We use four tensor time series

datasets from the hourly climate benchmark database ERA5 [330] and align them with the extreme weather database of NOAA[12] based on geoinformation and contribute a new benchmark for climate science. The results show that TacSas outperforms both state-of-the-art forecasting and detection baselines.

6.1.2 Preliminary

Tensor Time Series As shown in Figure 6.1(a), we have tensor time series data stored in $\mathcal{X} \in \mathbb{R}^{N \times D \times T}$. Note that a slice of $\mathcal{X}$, i.e., $\mathcal{X}(i, :, :) \in \mathbb{R}^{D \times T}$, $i \in \{1 \dots, N\}$, is typically denoted as the common multivariate time series data [304, 331]. In this way, tensor time series can be understood as multiple multivariate time series data. Such tensor time series data can usually be found in the real world. For example, in each element $\mathcal{X}(i, d, t)$ of the nationwide weather data $\mathcal{X}$, $i \in \{1 \dots, N\}$ can be the spatial locations (e.g., counties), $d \in \{1 \dots, D\}$ can be the weather features (e.g., temperature and humidity), and $t \in \{1 \dots, T\}$ can be the time dimension (e.g., hours). Throughout the paper, we use the calligraphic letter to denote a 3D tensor (e.g., $\mathcal{X}$) and the bold capital letter to denote a 2D matrix (e.g., $\mathbf{X}$).

Problem Definition We aim to discover and utilize comprehensive causal structures for tensor time-series analysis tasks, including forecasting and anomaly detection. To be more specific, given the tabular data $\mathcal{X} \in \mathbb{R}^{N \times D \times T}$ as shown in Figure 6.1, we aim to forecast the future data $\mathcal{X}' \in \mathbb{R}^{N \times D \times \tau}$, where τ is a forecasting window. Additionally, with the forecasted $\mathcal{X}'$, we also aim to detect if $\mathcal{X}'$ contains abnormal values.

6.1.3 Proposed Method

In this section, we introduce how TacSas discovers TBN Granger Causality in the historical tensor time series and utilizes it to guide tensor time series forecasting and anomaly detection. The overall framework of TacSas is shown in Figure 6.2.

The upper component of Figure 6.2 represents the data preprocessing part (i.e., converting raw input $\mathcal{X}$ to latent representation $\mathcal{H}$) of TacSas through a pre-trained autoencoder. The goal of this component is leveraging comprehensive causality (e.g., TBN Granger Causality) to achieve seamless forecasting and anomaly detection.

The lower component of Figure 6.2 shows how TacSas discovers TBN Granger Causality in the historical tensor time series (in the form of $\mathcal{H}$ other than $\mathcal{X}$) and generates future

[12] `https://www.ncdc.noaa.gov/stormevents/ftp.jsp`

tensor time series. In brief, the optimization of TacSas is bi-level. First, the inner optimization captures instantaneous effects among variables at each timestamp, respectively, which describes the inner-time feature generation. These causal structures are then stored in the form of a sequence of Bayesian Networks. Second, the outer optimization discovers the Neural Granger Causality among variables in a time window with the support of a sequence of Bayesian Networks (i.e., TBN Granger Causality). After introducing details, we derive the formal equation of TBN Granger Causality, Eq. 6.7.

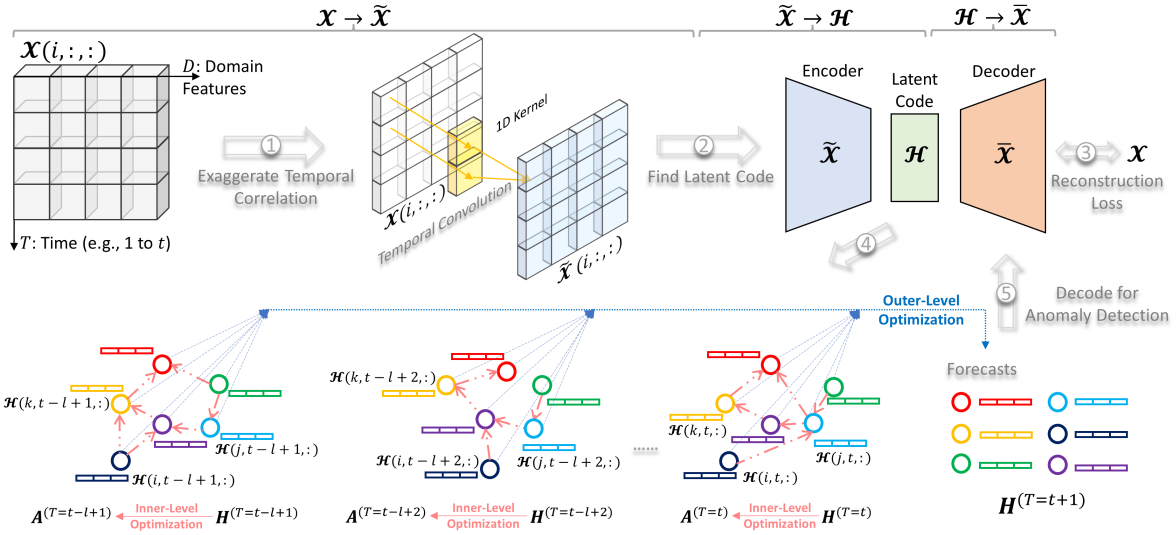


Figure 6.2: Working Flow of TacSas: Discovering and Utilizing the TBN Granger Causality through a Bi-Level Optimization for Tensor Time Series Forecasting and Anomaly Detection.

Inner Optimization of TacSas for Identifying Instantaneous Causal Relations in Time Series Generally speaking, the inner optimization produces a sequence of Bayesian Networks for each observed timestamp. At time t, the instantaneous causality is discovered based on input features $\mathcal{H}(:,:,t) = \mathbf{H}^{(t)} \in \mathbb{R}^{N \times H}$, and is represented by a directed acyclic graph $\mathcal{G}^{(t)} = (\mathbf{A}^{(t)} \in \mathbb{R}^{N \times N}, \mathbf{H}^{(t)} \in \mathbb{R}^{N \times H})$. To be specific, $\mathbf{A}^{(t)}$ is a weighted adjacency matrix of the Bayesian Network at time t, and each cell represents the coefficient of causal effects between variables u and $v \in \{1, \ldots, N\}$. The features (e.g., $\mathcal{H}(v,:,t)$) are transformed from the input raw features (e.g., $\mathcal{X}(v,:,t)$). The transformation is causality-agnostic but necessary for downstream time series analysis tasks.

The reasoning for discovering the instantaneous causal effects in the form of the Bayesian Network originates from a widely adopted assumption of causal graph learning [309, 312, 332, 333, 334]: there exists a ground-truth causal graph $\mathbf{S}^{(t)}$ that specifies instantaneous parents of variables to recover their value generating process. Therefore, in our inner optimization,

the goal is to discover the causal structure $\mathbf{S}^{(t)}$ at each time t by recovering the generation of input features $\mathbf{H}^{(t)}$. Specifically, given the observed $\mathbf{H}^{(t)}$, we aim to estimate a structure $\mathbf{A}^{(t)}$, through which a certain distribution $\mathbf{Z}^{(t)}$ could generate $\mathbf{H}^{(t)}$ for $t \in \{1, \ldots, T\}$. In this way, the instantaneous causal effects are discovered, and the corresponding structures are encoded in $\mathbf{A}^{(t)}$. The generation function is expressed as follows.

$$\sum_t log\mathcal{P}(\mathbf{H}^{(t)}) = \sum_t log \int \mathcal{P}(\mathbf{H}^{(t)}|\mathbf{Z}^{(t)})\mathcal{P}(\mathbf{Z}^{(t)})d\mathbf{Z}^{(t)} \tag{6.1}$$

where the generation likelihood $\mathcal{P}(\mathbf{H}^{(t)}|\mathbf{Z}^{(t)})$ also takes $\mathbf{A}^{(t)}$ as input. The complete formula is shown in Eq. 6.3.

For Eq 6.1, on the one hand, it is hard to get the prior distribution $\mathcal{P}(\mathbf{Z}^{(t)})$, which is highly related to the distribution of ground-truth causal graph distribution $\mathcal{P}(\mathbf{S}^{(t)})$ at time t [309]. On the other hand, for the generation likelihood $\mathcal{P}(\mathbf{H}^{(t)}|\mathbf{Z}^{(t)})$, the actual posterior $\mathcal{P}(\mathbf{Z}^{(t)}|\mathbf{H}^{(t)})$ is also intractable. Thus, we resort to the variational autoencoder (VAE) [335]. In this way, the actual posterior $\mathcal{P}(\mathbf{Z}^{(t)}|\mathbf{H}^{(t)})$ can be replaced by the variational posterior $\mathcal{Q}(\mathbf{Z}^{(t)}|\mathbf{H}^{(t)})$, and the prior distribution $\mathcal{P}(\mathbf{Z}^{(t)})$ is approximated by a Gaussian distribution. Furthermore, the inside encoder and decoder modules should take the structure $\mathbf{A}^{(t)}$ as the input. This design can be realized by various off-the-shelf variational graph autoencoders such as VGAE [253], etc. However, the inner optimization is coupled with the outer optimization, i.e., the instantaneous causality will be integrated with cross-time Granger causality to make inferences. The inner complex neural architectures and parameters may render the outer optimization module hard to train, especially when the outer module itself needs to be complex. Therefore, we extend the widely-adopted linear Structural Equation Model (SEM) [309, 312, 332, 333] to the time-respecting setting as follows.

For $\mathcal{Q}(\mathbf{Z}^{(t)}|\mathbf{H}^{(t)})$, the encoder equation is expressed as

$$\mathbf{Z}^{(t)} = (\mathbf{I} - {\mathbf{A}^{(t)}}^\top) f_{\theta_{enc}^{(t)}}(\mathbf{H}^{(t)}) \tag{6.2}$$

For $\mathcal{P}(\mathbf{H}^{(t)}|\mathbf{Z}^{(t)})$, the decoder equation is expressed as

$$\mathbf{H}^{(t)} = f_{\theta_{dec}^{(t)}}((\mathbf{I} - {\mathbf{A}^{(t)}}^\top)^{-1}\mathbf{Z}^{(t)}) \tag{6.3}$$

As analyzed above[13], $f_{\theta_{enc}^{(t)}}$ and $f_{\theta_{dec}^{(t)}}$ do not need complicated neural architectures. Therefore, we can use two-layer MLPs for them. Then, the objective function $\mathcal{L}_{DAG}^{(t)}$ for discovering

[13]The complete forms of $\mathcal{Q}(\mathbf{Z}^{(t)}|\mathbf{H}^{(t)})$ and $\mathcal{P}(\mathbf{H}^{(t)}|\mathbf{Z}^{(t)})$ are $\mathcal{Q}_{A^{(t)}}(\mathbf{Z}^{(t)}|\mathbf{H}^{(t)})$ and $\mathcal{P}_{A^{(t)}}(\mathbf{H}^{(t)}|\mathbf{Z}^{(t)})$, we omit the subscript $A^{(t)}$ for brevity.

the instantaneous causality at time t is expressed as follows, which corresponds to the inner optimization.

$$\begin{aligned}\min_{\theta_{enc}^{(t)},\theta_{dec}^{(t)},\mathbf{A}^{(t)}} \mathcal{L}_{DAG}^{(t)} = \; & D_{KL}(\mathcal{Q}(\mathbf{Z}^{(t)}|\mathbf{H}^{(t)})\|\mathcal{P}(\mathbf{Z}^{(t)})) \\ & -\mathbb{E}_{\mathcal{Q}(\mathbf{Z}^{(t)}|\mathbf{H}^{(t)})}[\log\mathcal{P}(\mathbf{H}^{(t)}|\mathbf{Z}^{(t)})] \\ \text{s.t.} \; \sum_t \mathrm{Tr}[(\mathbf{I}+\mathbf{A}^{(t)}\circ\mathbf{A}^{(t)})^N] - N = 0, \; & \text{for } t \in \{1,\ldots,T\}\end{aligned} \tag{6.4}$$

where the first term in $\mathcal{L}_{DAG}^{(t)}$ is the KL-divergence measuring the distance between the distribution of generated $\mathbf{Z}^{(t)}$ and the pre-defined Gaussian, and the second term is the reconstruction loss between the generated $\mathbf{Z}^{(t)}$ with the original input $\mathbf{H}^{(t)}$. Note that there is an important constraint, i.e., $\mathrm{Tr}[(\mathbf{I}+\mathbf{A}^{(t)}\circ\mathbf{A}^{(t)})^N]-N=0$, on $\mathbf{A}^{(t)} \in \mathbb{R}^{N\times N}$. $\mathrm{Tr}(\cdot)$ is the trace of a matrix, and $\circ$ denotes the Hadamard product. The meaning of the constraint is explained as follows. The constraint in Eq. 6.4, i.e., $\mathrm{Tr}[(\mathbf{I}+\mathbf{A}^{(t)}\circ\mathbf{A}^{(t)})^N]-N=0$ regularizes the acyclicity of $\mathbf{A}^{(t)}$ during the optimization process, i.e., the learned $\mathbf{A}^{(t)}$ should not have any possible closed-loops at any length.

Lemma 6.1. Let $\mathbf{A}^{(t)}$ be a weighted adjacency matrix (negative weights allowed). $\mathbf{A}^{(t)}$ has no N-length loops, if $\mathrm{Tr}[(\mathbf{I}+\mathbf{A}^{(t)}\circ\mathbf{A}^{(t)})^N]-N=0$.

The intuition is that there will be no k-length path from node u to node v on a binary adjacency matrix $\}(u,v)=0$. Compared with original acyclicity constraints in [333], our Lemma 6.1 gets rid of the λ condition. Then we can denote $\alpha(A^{(t)}) = \mathrm{Tr}[(\mathbf{I}+\mathbf{A}^{(t)}\circ\mathbf{A}^{(t)})^N]-N$ and use Lagrangian optimization for Eq. 6.4 as follows.

$$\begin{aligned}\min_{\theta_{enc}^{(t)},\theta_{dec}^{(t)},\mathbf{A}^{(t)}} \mathcal{L}_{DAG}^{(t)} = \; & D_{KL}(\mathcal{Q}(\mathbf{Z}^{(t)}|\mathbf{H}^{(t)})\|\mathcal{P}(\mathbf{Z}^{(t)})) \\ & -\mathbb{E}_{\mathcal{Q}(\mathbf{Z}^{(t)}|\mathbf{H}^{(t)})}[\log\mathcal{P}(\mathbf{H}^{(t)}|\mathbf{Z}^{(t)})] \\ & +\lambda\,\alpha(A^{(t)}) + \frac{c}{2}|\alpha(A^{(t)})|^2, \quad \text{for } t \in \{1,\ldots,T\}\end{aligned} \tag{6.5}$$

where λ and c are two hyperparameters, and larger λ and c enforce $\alpha(\mathbf{A}^{(t)})$ to be smaller.

Theorem 6.1. If the ground-truth instantaneous causal graph $\mathbf{S}^{(t)}$ at time t generates the features of variables following the normal distribution, then the inner optimization (i.e., Eq. 6.4) can identify $\mathbf{S}^{(t)}$ under the standard causal discovery assumptions [309].

Outer Optimization of TacSas for Integrating Instantaneous Causality with Neural Granger Given the inner optimization, Bayesian Networks can be obtained at each

timestamp t, which means that multiple instantaneous causalities are discovered. Thus, in the outer optimization, we integrate these evolving Bayesian Networks into Granger Causality discovery. First, the classic Granger Causality [316] is discovered in the form of the variable-wise coefficients across different timestamps (i.e., a time window) through the autoregressive prediction process. The prediction based on the linear Granger Causality [316] is expressed as follows.

$$\mathbf{H}^{(t)} = \sum_{l=1}^{L} \mathbf{W}^{(l)} \mathbf{H}^{(t-l)} + \mathbf{e}^{(t)} \tag{6.6}$$

where $\mathbf{H}^{(t)} \in \mathbb{R}^{N \times D}$ denotes the features of N variables at time t, $\mathbf{e}^{(t)}$ is the noise, and L is the pre-defined time lag indicating how many past timestamps can affect the values of $\mathbf{H}^{(t)}$. Weight matrix $\mathbf{W}^{(l)} \in \mathbb{R}^{N \times N}$ stores the cross-time coefficients captured by Granger Causality, i.e., matrix $\mathbf{W}^{(l)}$ aligns the variables at time $t-l$ with the variables at time t. To compute those weights, several linear methods are proposed, e.g., vector autoregressive model [317].

Facing non-linear causal relationships, neural Granger Causality discovery [310] is recently proposed to explore the nonlinear Granger Causality effects. The general principle is to represent causal weights $\mathbf{W}$ by deep neural networks. To integrate instantaneous effects with neural Granger Causality discovery, our TBN Granger Causality is expressed as follows.

$$\hat{H}(i,:)^{(t)} = f_{\Theta_i}[(\mathbf{A}^{(t-1)}, \mathbf{H}^{(t-1)}), \ldots, (\mathbf{A}^{(t-L)}, \mathbf{H}^{(t-L)})] \tag{6.7}$$

where L is the lag (or window size) in the Granger Causality, and i is the index of the i-th variable. f_{Θ_i} is a neural computation unit with all parameters denoted as Θ_i, whose input is an L-length time-ordered sequence of $(\mathbf{A}, \mathbf{H})$. And f_{Θ_i} is responsible for discovering the TBN Granger Causality for variable i at time t from all variables that occurred in the past time lag l. The choice of neural unit f_{Θ_i} is flexible, such as MLP and LSTM [310]. Different neural unit choices correspond to different causality interpretations. In our proposed TacSas model, we use graph recurrent neural networks [30].

In the outer optimization, to evaluate the prediction under the TBN Granger Causality, we use the mean absolute error (MAE) loss on the prediction and the ground truth, which is effective and widely applied to time-series forecasting tasks [302, 306].

$$\min_{\Theta_i, \mathbf{A}^{(t-1)}, \ldots, \mathbf{A}^{(t-l)}} \mathcal{L}_{pred} = \sum_i \sum_t |H(i,:)^{(t)} - \hat{H}(i,:)^{(t)}| \tag{6.8}$$

where $\Theta_i, \mathbf{A}^{(t-1)}, \ldots, \mathbf{A}^{(t-l)}$ are all the parameters for the prediction $\hat{H}(i,:)^{(t)}$ of variable i

at time t. The composition and update rules are expressed below.

For updating f_{Θ_i}, we employ the recurrent neural structure to fit the input sequence. Moreover, the sequential inputs also contain the structured data $\mathbf{A}$. Therefore, we use the graph recurrent neural architecture [302] because it is designed for directed graphs, whose core is a gated recurrent unit [336].

$$\begin{aligned}
\mathbf{R}^{(t)} &= \text{sigmoid}(\mathbf{W}_{\mathbf{R}*\mathbf{A}^{(t)}}[\mathbf{H}^{(t)} \oplus \mathbf{S}^{(t-1)}] + \mathbf{b}_R) \\
\mathbf{C}^{(t)} &= \tanh(\mathbf{W}_{\mathbf{C}*\mathbf{A}^{(t)}}[\mathbf{H}^{(t)} \oplus (\mathbf{R}^{(t)} \odot \mathbf{S}^{(t-1)})] + \mathbf{b}_C) \\
\mathbf{U}^{(t)} &= \text{sigmoid}(\mathbf{W}_{\mathbf{U}*\mathbf{A}^{(t)}}[\mathbf{H}^{(t)} \oplus \mathbf{S}^{(t-1)}] + \mathbf{b}_U) \\
\mathbf{S}^{(t)} &= \mathbf{U}^{(t)} \odot \mathbf{S}^{(t-1)} + (\mathbf{I} - \mathbf{U}^{(t)}) \odot \mathbf{C}^{(t)}
\end{aligned} \tag{6.9}$$

where $\mathbf{R}^{(t)}$, $\mathbf{C}^{(t)}$, and $\mathbf{U}^{(t)}$ are three parameterized gates, with corresponding weights $\mathbf{W}$ and bias $\mathbf{b}$. $\mathbf{H}^{(t)}$ is the input, and $\mathbf{S}^{(t)}$ is the hidden state. Gates $\mathbf{R}^{(t)}$, $\mathbf{C}^{(t)}$, and $\mathbf{U}^{(t)}$ share the similar structures. For example, in $\mathbf{R}^{(t)}$, the graph convolution operation for computing the weight $\mathbf{W}_{\mathbf{R}*\mathbf{A}^{(t)}}$ is defined as follows, and the same computation applies to gates $\mathbf{U}^{(t)}$ and $\mathbf{C}^{(t)}$.

$$\mathbf{W}_{\mathbf{R}*\mathbf{A}^{(t)}} = \sum_{k=0}^{K} \theta_{k,1}^R (\mathbf{D}_{out}^{(t)\,-1} \mathbf{A}^{(t)})^k + \theta_{k,2}^R (\mathbf{D}_{in}^{(t)-1} \mathbf{A}^{(t)\top})^k \tag{6.10}$$

where $\theta_{k,1}^R$, $\theta_{k,2}^R$ are learnable weight parameters; scalar k is the order for the stochastic diffusion operation (i.e., similar to steps of random walks); $\mathbf{D}_{out}^{(t)\,-1}\mathbf{A}^{(t)}$ and $\mathbf{D}_{in}^{(t)-1}\mathbf{A}^{(t)\top}$ serve as the transition matrices with the in-degree matrix $\mathbf{D}_{in}^{(t)}$ and the out-degree matrix $\mathbf{D}_{out}^{(t)}$; -1 and $\top$ are inverse and transpose operations.

For updating each of $\{\mathbf{A}^{(t-1)}, \ldots, \mathbf{A}^{(t-l)}\}$, we take $\mathbf{A}^{(t-l)}$ as an example to illustrate. The optimal $\mathbf{A}^{(t-l)}$ stays in the space of $\{0,1\}^{N\times N}$. To be specific, each edge $A^{(t-l)}(i,j)$ can be parameterized as $\theta_{i,j}^{(t-l)}$ following the Bernoulli distribution. However, $N^2 l$ is hard to scale, and the discrete variables are not differentiable. Therefore, we adopt the Gumbel reparameterization from [337, 338]. It provides a continuous approximation for the discrete distribution, which has been widely used in the graph structure learning [306, 339]. The general reparameterization form can be written as $A^{(t-l)}(i,j) = softmax(FC((H(i,:)^{(t-l)}||H(j,:)^{(t-l)}) + g)/\xi)$, where FC is a feedforward neural network, g is a scalar drawn from a Gumbel$(0,1)$ distribution, and ξ is a scaling hyperparameter. Different from [306, 339], in our setting, the initial structure input is constrained by the causality discovery, which originates from the inner optimization step. Hence, the structure learning in the outer optimization takes the

adjacency matrix from the inner optimization as the initial input, which is

$$A_{outer}^{(t-l)}(i,j) = softmax(A_{inner}^{(t-l)}(i,j) + \mathbf{g})/\xi) \tag{6.11}$$

where $A_{inner}^{(t)}(i,j)$ is the structure learned by our inner optimization through Eq. 6.4, $A_{outer}^{(t)}(i,j)$ is the updated structure, and $\mathbf{g}$ is a vector of i.i.d samples drawn from a Gumbel$(0,1)$ distribution. In outer optimization, Eq. 6.8 fine-tunes the evolving Bayesian Networks to make the intra-time causality fit the cross-time causality well. Note that, the outer optimization w.r.t. $\mathbf{A}^{(t)}$ may break the acyclicity, and another round of inner optimization may be necessary.

6.1.4 Empirical Evaluation

Datasets Our forecasting data (i.e., hourly tensor time series data) originates from climate domain benchmark ERA5 [330][14]. To be specific, we select four datasets covering 45 weather features (i.e., wind gusts, rain, etc.) from 238[15] counties in the United States of America during 2017–2020. Moreover, we choose thunderstorms as the anomaly pattern to be detected after forecasting. The thunderstorm record is identified in NOAA database[16] hourly and nationwide, i.e., 1 means a thunderstorm happens in the corresponding hour at a certain location, and 0 means no thunderstorm happens. We processed the geocode to align weather features in ERA5 with anomaly patterns in NOAA.

Baselines Besides the causality discovery baseline, the first category is for tensor time series forecasting: (1) GRU [336] is a classical sequence-to-sequence generation model. (2) DCRNN [302] is a graph convolutional recurrent neural network, of which the input graph structure is given, not causal, and static (i.e., shared by all timestamps). In this viewpoint, we let each node randomly distribute its unit weights to others. (3) GTS [306] is also a graph convolutional recurrent neural network that does not need the input graph but learns the structure based on the node features, but the learned structure is also shared by all timestamps and is not causal. To compare the performance of DCGNN [302] and GTS [306] with TacSas, causality is the control variable since we make all the rest (e.g., neural network type, number of layers, etc.) identical for them. The second category is for anomaly detection on tensor time series: (1) DeepSAD [340], (2) DeepSVDD [341], and (3) DROCC [342]. Since these three have no forecast abilities, we let them use the ground-

[14] `https://cds.climate.copernicus.eu/cdsapp\#!/home`

[15] 100 of 238 counties are top-ranked counties for the thunderstorm (anomaly label) frequency, and the rest are randomly selected.

[16] `https://www.ncdc.noaa.gov/stormevents/ftp.jsp`

truth observations, and our TacSas utilizes the forecast features during anomaly detection experiments. Also, these three baselines are designed for multi-variate time-series data, not tensor time-series. Thus, we flatten our tensor time series along the spatial dimension and report the average performance for these three baselines over all locations.

6.1.5 Causality Discovery

Lorenz-96 model [329] is a famous synthetic system of multivariate time-series, e.g., $\mathbf{X} \in \mathbb{R}^{P \times T}$ is a P-dimensional time series whose dynamics can be modeled as follows.

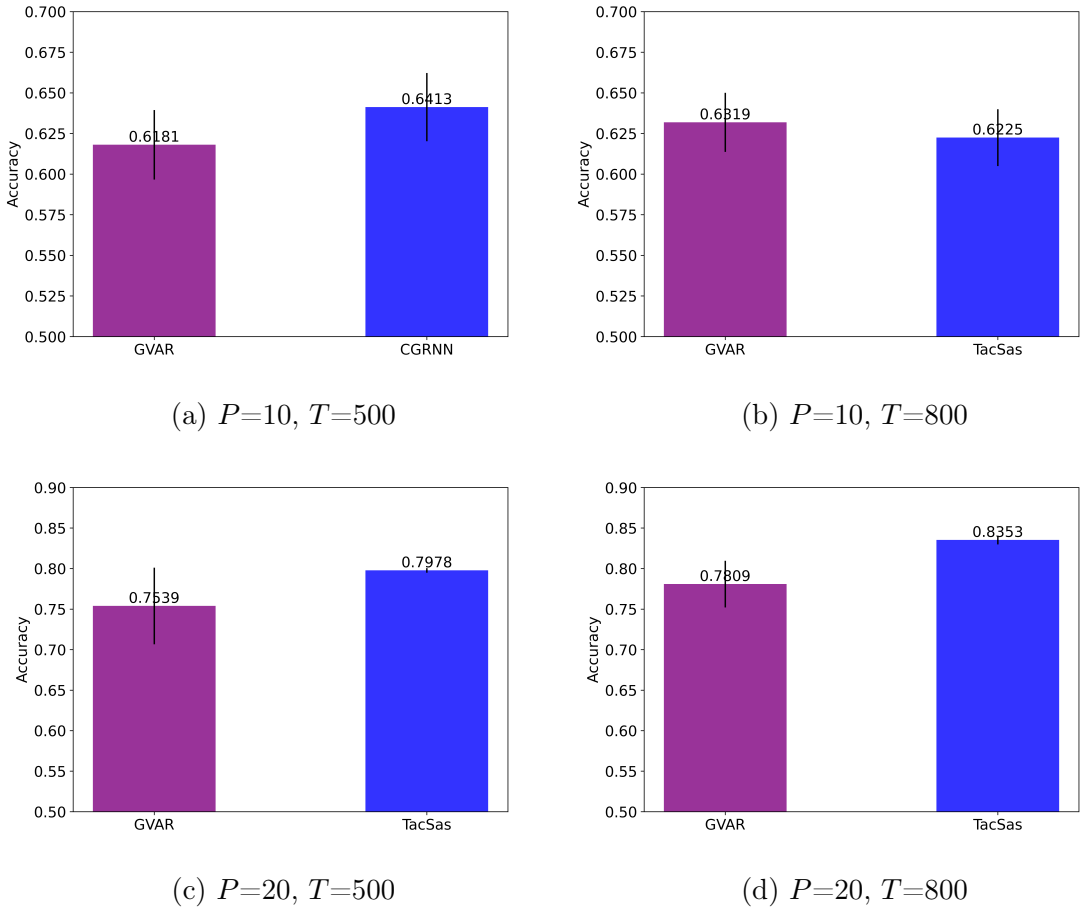
$$\frac{d\mathbf{X}(i,t)}{dt} = (\mathbf{X}(i+1,t) - \mathbf{X}(i-2,t))\mathbf{X}(i-1,t) - \mathbf{X}(i,t) + F, \text{ for } i \in \{1,2,\ldots,P\} \quad (6.12)$$

where $\mathbf{X}(0,t) = \mathbf{X}(P,t)$, $\mathbf{X}(-1,t) = \mathbf{X}(P-1,t)$, $\mathbf{X}(P+1,t) = \mathbf{X}(1,t)$, and F is the forcing constant determining the level of nonlinearity and chaos in the time series. With the above modeling, the corresponding ground-truth Granger causal structures can be simulated, involving multivariate, nonlinear, and sparse [310].

To generate the ground-truth causal structures, there are two parameters, i.e., the number of variables (i.e., P) and the number of timestamps (i.e., T). Therefore, we control these two parameters and report the accuracy of TacSas discovered causal structures against the ground-truth ones (i.e., 0/1 adjacency matrices), compared with the state-of-the-art causality discovery method GVAR [308]. The comparison is shown in Figure 6.3 after eight experiment trials with mean and variance computed, where we can observe our TacSas achieve the competitive accuracy of discovering the ground-truth causal structures. Also, by comparing Figure 6.3(a) and (b) (and Figure 6.3(c) and (d)), we can see that fixing the number of variables (i.e., P), increasing the time series length (i.e., T) may help discover the causality. And by comparing Figure 6.3(a) and (c) (and Figure 6.3(b) and (d)), we can see that fixing the time length (i.e., T), increasing the number of variables (i.e., P) may make the causality easier to be discovered.

6.1.6 Forecasting Performance

In Table 6.1, we present the forecasting performance in terms of mean absolute error (MAE) on the testing data of three algorithms, namely DCGNN [302], GTS [306], ST-SSL [343], our TacSas, and TacSas++ (i.e., TacSas with persistence forecast constraints). Here, we set the time window as 24, meaning that we use the past 24 hours tensor time series to forecast the future 24 hours in an autoregressive manner. Moreover, for baselines



(a) P=10, T=500

(b) P=10, T=800

(c) P=20, T=500

(d) P=20, T=800

Figure 6.3: Accuracy of Causality Discovery in Lorenz-96 with Varying Number of Variables (i.e., P) and Timestamps (i.e., T).

and TacSas, we set f_{Θ_i} in Eq. 6.7 shared by all weather variables to ensure the scalability, such that we do not need to train N recurrent graph neural networks for a single prediction. In Table 6.1, we can observe a general pattern that our TacSas outperforms the baselines with GTS performing better than DCGNN. For example, with 2017 as the testing data, our TacSas performs 39.44% and 36.16% better than DCRNN and GTS. An explanation is that the temporally fine-grained causal relationships can contribute more to the forecasting accuracy than non-causal directed graphs, since DCGNN, GTS, and our TacSas all share the graph recurrent manner. TacSas, however, discovers causalities at different timestamps, while DCGNN and GTS use feature similarity based connections. Moreover, ST-SSL achieves competitive forecasting performance via contrastive learning on time series. Motivated by contrastive manner, TacSas++ is proposed by persistence forecast constraints. That is, the current forecast of TacSas is further calibrated by its nearest time window (i.e., the last 24

Table 6.1: Forecasting Error (MAE, 10^{-2})

	ERA5-2017 (↓)	ERA5-2018 (↓)	ERA5-2019 (↓)	ERA5-2020 (↓)
GRU	1.8834 ± 0.0126	1.9764 ± 0.1466	1.6194 ± 0.2645	1.7859 ± 0.2324
DCRNN	0.0819 ± 0.0025	0.0797 ± 0.0049	0.0799 ± 0.0035	0.0826 ± 0.0033
GTS	0.0777 ± 0.0054	0.0766 ± 0.0029	0.0760 ± 0.0031	0.0742 ± 0.0021
TacSas	0.0496 ± 0.0017	0.0499 ± 0.0017	0.0502 ± 0.0016	0.0488 ± 0.0019
ST-SSL	0.0345 ± 0.0051	0.0330 ± 0.0018	0.0361 ± 0.0021	0.0348 ± 0.0020
TacSas++	**0.0271 ± 0.0004**	**0.0276 ± 0.0004**	**0.0282 ± 0.0003**	**0.0265 ± 0.0004**

Table 6.2: Anomaly Detection Performance (AUC-ROC)

	NOAA-2017 (↑)	NOAA-2018 (↑)	NOAA-2019 (↑)	NOAA-2020 (↑)
DeepSAD	0.5305 ± 0.0481	0.5267 ± 0.0406	0.5563 ± 0.0460	0.6420 ± 0.0054
DeepSVDD	0.5201 ± 0.0045	0.5603 ± 0.0111	**0.6784 ± 0.0112**	0.5820 ± 0.0205
DROCC	0.5319 ± 0.0661	0.5103 ± 0.0147	0.6236 ± 0.0992	0.5630 ± 0.1082
TacSas	**0.5556 ± 0.0010**	**0.5685 ± 0.0011**	0.6298 ± 0.0184	**0.6745 ± 0.0185**

hours in our setting). The persistence forecasting can be expressed as

$$\mathbf{X}^{(t)}_{TacSas++} = \alpha \mathbf{X}^{(t)}_{TacSas} + (1-\alpha)\mathbf{X}^{(t-\tau)} \quad \text{s.t. } \mathbf{X}^{(t)}_{TacSas} = \text{TacSas}(\mathbf{X}^{(t-\tau)}) \tag{6.13}$$

where τ is the time window, for example, in the experiments, $\tau = 24$h.

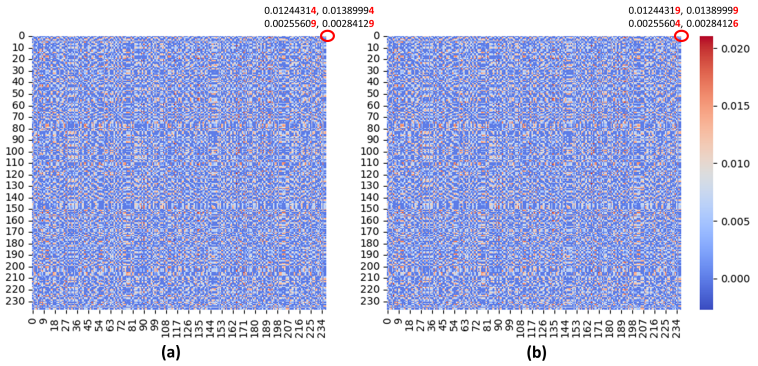


Figure 6.4: Time-Respecting Bayesian Networks of at the Same Hour of Two Consecutive Days.

To evaluate our explanation, we visualize causal connections at different times in Figure 6.4. Specifically, we show the Bayesian Network of 238 counties at the same hour on two consecutive days in the training data (i.e., May 1st and May 2nd, 2018). Interestingly, we can observe that two patterns in Figure 6.4 are almost identical at first glance. That could be the reason why DCRNN and GTS can perform well using the static structure. However, upon closer inspection, we find that these two are quite different to some extent if we zoom

in, such as, in the upper right corner. Although the values have a tiny divergence, their volume is quite large. In two matrices of Figure 6.4, the number of different cells is 28,509, and the corresponding percentage is $\frac{28509}{238\times238} \approx 0.5033$. We suppose that discovering those value-tiny but volume-big differences makes TacSas outperform, to a large extent.

6.1.7 Anomaly Detection

After forecasting, we can have the hourly forecast of weather features at certain locations, denoted as $\mathcal{X}'$. Then, we use the encoder-decoder model in Figure 6.2 to calculate the feature-wise generation probability using mean squared error (MSE) between $\mathcal{X}'$ and its generation $\bar{\mathcal{X}}'$. Thus, we can calculate the average of feature-wise generation probability as the condition of anomalies to identify if an anomaly weather pattern (e.g., a thunderstorm) happens in an hour in a particular location. In Table 6.2, we use the Area Under the ROC Curve (i.e., AUC-ROC) as the metric, repeat the experiments four times, and report the performance of TacSas with baselines.

From Table 6.2, we can observe that the detection module of TacSas achieves very competitive performance. An explanation is that, based on the anomalies distribution shown in Table 6.3, it can be observed that the anomalies are very rare events. Our generative manner could deal with the very rare scenario by learning the feature latent distributions instead of the (semi-)supervised learning manner. For example, the maximum frequency of occurrences of thunderstorms is 770 (i.e., Jun 2017), which is collected from 238 counties over $30 \times 24 = 720$ hours, and the corresponding percentage is $\frac{770}{238\times30\times24} \approx 0.45\%$.

Table 6.3: Statistics of Anomaly Weather Patterns (i.e., Thunderstorm Wind) Occurrence in 238 Selected Counties in the United States

Year	2017	2018	2019	2020	2021
Jan	26	3	2	41	7
Feb	53	6	9	50	8
Mar	85	16	26	63	62
Apr	93	44	140	170	60
May	245	207	263	175	218
Jun	770	302	348	331	452
Jul	306	291	457	453	701
Aug	294	269	415	354	435
Sep	61	80	122	29	123
Oct	32	32	82	60	55
Nov	20	22	9	114	11
Dec	5	15	11	8	58

6.1.8 Summary

Understanding the causal interaction of time series variables can help time series data analysis for many applications, such as climate forecasting and extreme weather alerts. However, these kinds of causal relationships are difficult to observe in real-world complex settings, such as spatial-temporal data from deployed sensor networks. To this end, we aim to discover fine-grained causal relations among spatial-temporal variables just by observing their value distributions, finally realizing a more accurate and reliable time series analysis. First, we design a conceptual fine-grained causal model, named *TBN Granger Causality*, which adds time-respecting Bayesian Networks to the previous time-lagged Neural Granger Causality in order to offset instantaneous effects. Second, we propose an end-to-end deep generative model, named *TacSas*, which discovers TBN Granger Causality to generate future time series data and detect possible anomalies during the generation. For evaluations, besides the causality discovery benchmark Lorenz-96, we also tested TacSas on climate benchmark ERA5 for forecasting and the extreme weather benchmark of NOAA for anomaly detection.

6.2 PRIVACY-GUARANTEED AND UTILITY-PRESERVED GRAPH GENERATION VIA NATURAL + ARTIFICIAL DYNAMICS

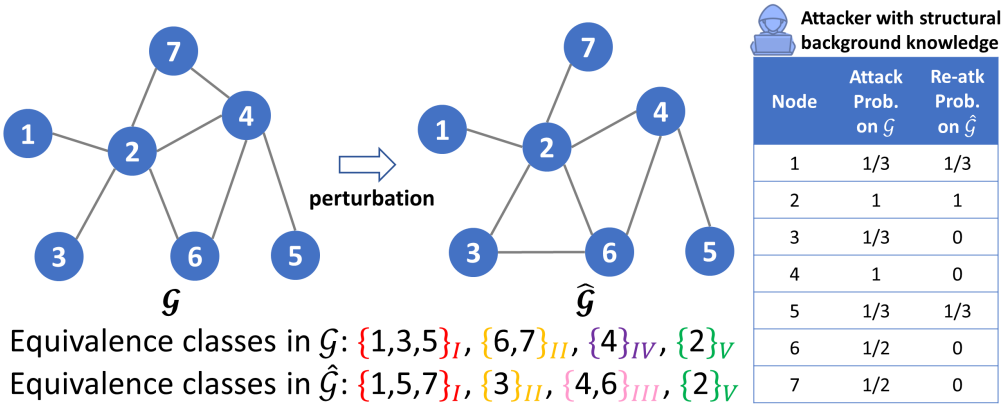
6.2.1 Overview and Motivation

Data privacy has become one of the most concerning topics in the era of big data and AI. As a ubiquitous data structure, graphs carry complex relational information among entities, and graph data generation techniques are critical to many applications, such as social network analysis and recommender systems [246, 344]. However, most of the graph generative models ignore the possibility that graph data can contain sensitive information, and only paying attention to fitting the distribution of graph data may generate privacy-leaking graph data, leaving a backdoor for attackers to reveal sensitive information like user identities in social networks [345].

Unlike texts and images, protecting privacy during relational graph data generation is more challenging, because the complex topology introduces additional structural information to expose the uniqueness of entities. For example, in a social network, the neighbors surrounding different users are typically different, which can be leveraged by attackers to easily locate the victim and reveal the identity by the isomorphism of the ego-network (i.e., 1-hop neighbors), called neighborhood relationships attacker queries [346]. In addition, vertex refinement attacker queries, hub fingerprint attacker queries, and subgraph attacker queries

have been proposed [347] to leverage structural background knowledge to locate victims.

Beyond that, the open world often involves heterogeneous and temporal graphs. First, a heterogeneous graph means the node type (and edge type) in a graph can be more than one [348, 349]. For example, in the citation network, the node types can include 'paper', 'author', and 'venue', and the edge types can have 'cite', 'write', and 'works at'. Second, graph structures (and features) can evolve over time. In a citation network, the structure evolves when new papers are published and new citation relationships are established. Generating temporal heterogeneous graphs provides high potential for real-world applications, but the realization is quite challenging. In other words, the complex and time-dependent structures and features of original graphs bring extra risk to expose privacy by making nodes more unique, which creates unprecedented challenges for graph generative models to preserve privacy when generating utilizable graph data.



Node	Attack Prob. on $\mathcal{G}$	Re-atk Prob. on $\hat{\mathcal{G}}$
1	1/3	1/3
2	1	1
3	1/3	0
4	1	0
5	1/3	1/3
6	1/2	0
7	1/2	0

Figure 6.5: A Toy Example of Attacker and Protector on a Static Homogeneous Graph: Attacker with Structural Background Knowledge vs. Graph Perturbation based Protection. The second column of the table shows the probability that an attacker can successfully identify the victim with background knowledge, and the third column shows the success probability with the same background knowledge on the perturbed graph.

To bridge this gap, we propose a Privacy-guaranteed and Utility-preserving Graph Generative model, called PUG. In brief, PUG relies on an adversarial learning manner, where PUG tries to explore the distribution of privacy-guaranteed graphs and utility-preserved graphs and then generates the synthetic graph that preserves both privacy and utility. To achieve that, the utilizable samples for adversarial learning can be from the original input graph data, and the key point is how to obtain the privacy-guaranteed samples, especially in the temporal heterogeneous setting.

Therefore, the graph perturbation provides a promising direction [347, 350, 351, 352, 353, 354]. A motivation case study is shown in Figure 6.5, where we demonstrate an attacker associated with the background knowledge of *node signature structural query* [353], i.e.,

nodes with the same structural information (e.g., node degree) establish the equivalence class and contain possible victims. The toy example in Figure 6.5 shows that one simple edge perturbation can decrease this attacker's confidence. For example, the attacker aims to reveal the node identity of node #6 and knows its degree is 2, then in the original graph $\mathcal{G}$, the attacker has $1/2$ probability of identifying node #6 correctly. But if graph $\mathcal{G}$ is perturbed into graph $\hat{\mathcal{G}}$, the attacker has 0 probability of identifying node #6, because node #6 is now transferred to another class for the new degree is 3.

Beyond the motivation case study, we need to prove a proposed perturbation algorithm on real temporal heterogeneous graphs can preserve privacy, for establishing privacy-guaranteed data samples for privacy-utility adversarial learning. Hence, in our PUG solution, we propose a bounded differential privacy algorithm for privacy guarantee in the temporal heterogeneous setting, and an effective privacy-utility adversarial learning method for generating temporal heterogeneous graphs. Also, we propose 6 metrics for measuring the utility and privacy of generated temporal heterogeneous graphs and conduct extensive attacking experiments showing the outperformance of PUG with baseline algorithms.

6.2.2 Preliminary and Problem Definition

First, we denote the perturbation operation as $\hat{}$, i.e., $\hat{\mathbf{A}}$ is the perturbed matrix of $\mathbf{A}$, the perturbation operation is related to privacy protection and is introduced in the next subsection.

Temporal Heterogeneous Graph Modeling A heterogeneous graph means the number of types of objects (i.e., nodes) is greater than 1 or the number of types of links (i.e., edges) is greater than 1 [355]. Moreover, the relationship between node type and edge type is deterministic, i.e., if two edges belong to the same type, then the two edges share the same starting node type as well as the ending node type [356]. *Hence, we use o to denote the node type and do not assign extra symbols for the edge type throughout the paper.* For modeling a temporal heterogeneous graph $\mathcal{G}$, we denote each edge as a triplet $((v_i, o_i), (v_j, o_j), t)$, where o_i is the node type of node v_i, and t is the timestamp that nodes v_i and v_j connects.

Problem 6.1. Privacy-Preserving Heterogeneous Temporal Graph Data Generation

Input: a temporal heterogeneous graph $\mathcal{G}$ of timestamped edges $((v_i, o_i), (v_j, o_j), t)$;

Output: a privacy-preserved temporal heterogeneous graph $\mathcal{G}^*$ that accurately captures the structural heterogeneity and temporal properties of the input graph $\mathcal{G}$.

6.2.3 Proposed Method

As shown in Figure 3.8, to generate a synthetic graph $\mathcal{G}^*$ that contains both privacy property and utility property, first, we need a privacy-preserving graph sample $\hat{\mathcal{G}}$ to provide the distribution of privacy-preserved structures. Second, we can then set the original graph $\mathcal{G}$ as the utilizable sample, and develop the adversarial generative model to fit the distribution of $\hat{\mathcal{G}}$ and $\mathcal{G}$ to generate a privacy-preserving and utility-preserving graph $\mathcal{G}^*$. Next, we first introduce the overall working flow of the PUG framework. Then, we introduce the details about how to perturb a temporal heterogeneous graph with privacy guaranteed. Finally, we introduce how to design the privacy-utility adversarial learning setting to generate viable temporal heterogeneous graphs.

Overview of PUG The overview of PUG is shown in Figure 6.6. The red arrows indicate the privacy-guaranteed graph creation, where a perturbation $\mathbb{M}$ is called to transfer the original graph $\mathcal{G}$ into $\hat{\mathcal{G}}$. Then, the blue and green arrows stand for the privacy-utility adversarial learning that involves three pillar modules. Discriminator $\mathbb{D}_{\text{prv}}$ is designed to guarantee the privacy of the generated graph, which aims to discriminate the generated graph $\mathcal{G}^*$ and the perturbed graph $\hat{\mathcal{G}}$. And Discriminator $\mathbb{D}_{\text{util}}$ is designed for preserving the graph utility, which aims to discriminate the generated graph $\mathcal{G}^*$ and the original graph $\mathcal{G}$. To optimize this adversarial training,

- We first pre-train $\mathbb{D}_{\text{util}}$ such that it can distinguish an original graph $\mathcal{G}$ with any other graph patterns.
- Then, in the iterations of adversarial training, we alternatively (1) train $\mathbb{D}_{\text{prv}}$ with fixed the Generator $\mathbb{G}$, and (2) train Generator $\mathbb{G}$ with fixed Discriminator $\mathbb{D}_{\text{prv}}$ and fixed Discriminator $\mathbb{D}_{\text{util}}$.

After adversarial training, the ability of Generator $\mathbb{G}$ will be improved, such that (1) Discriminator $\mathbb{D}_{\text{prv}}$ could not distinguish graph $\mathcal{G}^*$ and $\hat{\mathcal{G}}$ (i.e., privacy guaranteed), and (2) Discriminator $\mathbb{D}_{\text{util}}$ could not distinguish graph $\mathcal{G}^*$ and $\mathcal{G}$ (i.e., utility preserved). Under this optimization, Generator $\mathbb{G}$ can output the privacy-guaranteed and utility-preserved graphs.

Data-Driven Graph Structure Perturbation based on Differential Privacy Here, we propose the *Data-Driven Graph Structure Perturbation based on Differential Privacy* for perturbing the input graph $\mathcal{G}$ into $\hat{\mathcal{G}}$, which (1) not only randomizes the graph structural distribution bounded by edge-level differential privacy (2) but also preserves the temporal heterogeneous graph topology in terms of the degree distribution. The main advantage of

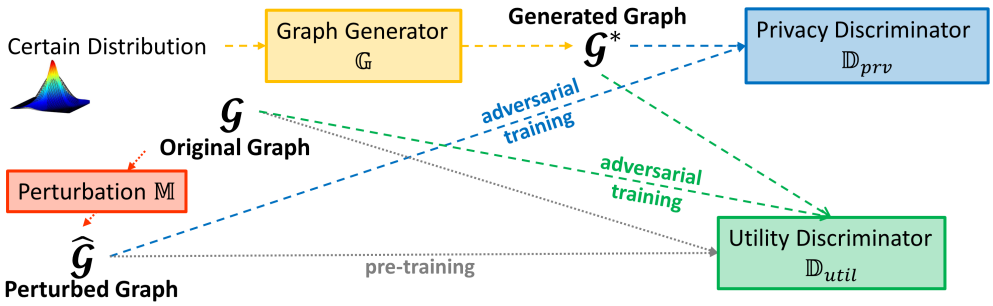


Figure 6.6: Overview of PUG Framework.

preserving the topology information includes maintaining the discrimination differently for the discriminator in adversarial training to avoid the training collapse.

Before we introduce the detailed operation, we need to first reformat the temporal heterogeneous graph within a time-respecting representation, i.e., representing all edges (regardless of edge types) that share the same timestamp into a snapshot. This time-respecting representation is more convenient for the operation illustration and theoretic analysis and meanwhile, will not influence the follow-up adversarial learning. The graph is represented as $\mathcal{G} = \{\mathcal{G}^{(1)}, \mathcal{G}^{(2)}, \ldots, \mathcal{G}^{(T)}\}$, where each snapshot has the adjacency matrix $\mathbf{A}^{(t)} \in \mathbb{R}^{n \times n}$, and n is the number of all appeared nodes in the lifetime of $\mathcal{G}$ such that $\mathbf{A}^{(t)}$ allows dangling nodes.

Definition 6.1 (Adjacent Temporal Snapshots)**.** At the timestamp t, two snapshots $\mathcal{G}^{(t)}$ and $\widetilde{\mathcal{G}}^{(t)}$ are adjacent snapshots, if and only if $\mathcal{G}^{(t)}$ and $\widetilde{\mathcal{G}}^{(t)}$ have the same set of nodes and differ in only one edge existence. For example, e_{ij} exists in $\mathcal{G}^{(t)}$ but not in $\widetilde{\mathcal{G}}^{(t)}$, or vice versa.

More precisely, based on the heterogeneous definition [356], the two node types determine the edge type, then the above definition also applies to heterogeneous graphs.

Definition 6.2 (Adjacent Temporal Graphs)**.** Two temporal graphs $\mathcal{G}$ and $\widetilde{\mathcal{G}}$ are adjacent, if and only if they have the same time length (i.e., from $t = 1$ to $t = T$) and only one pair of adjacent snapshots exists with the identical rest.

In the static setting, **edge-level differential privacy** [20, 357, 358] describes that: given two adjacent graphs G and $\widetilde{G}$ (i.e., differ in one edge existence), a randomized mechanism $\mathbb{M}$ satisfies the edge-level differential privacy under a constant budget $\varepsilon > 0$, if and only if

$$Pr[\mathbb{M}(\mathbf{A}) = \mathbf{S}] \leq Pr[\mathbb{M}(\widetilde{\mathbf{A}}) = \mathbf{S}] \times e^{\varepsilon} \tag{6.14}$$

where $\mathbf{A}$ denotes the adjacency matrix of G and $\widetilde{\mathbf{A}}$ denotes the adjacency matrix of $\widetilde{G}$.

The general idea of the above differential privacy is that two adjacent graphs (e.g., one edge difference between two graphs) are indistinguishable through the perturbation algorithm $\mathbb{M}$. Then, this perturbation algorithm $\mathbb{M}$ satisfies the differential privacy (e.g., edge-level differential privacy). The intuition is that the randomness of $\mathbb{M}$ does not make the small divergence produce a considerably different distribution, such that the randomness of $\mathbb{M}$ is not the cause of the privacy leak. Likewise, the temporal edge-level differential privacy can be defined as follows.

Definition 6.3 (Temporal Edge-Level Differential Privacy)**.** Given two adjacent temporal graphs $\mathcal{G} = \{\mathcal{G}^{(1)}, \mathcal{G}^{(2)}, \ldots, \mathcal{G}^{(T)}\}$ and $\widetilde{\mathcal{G}} = \{\widetilde{\mathcal{G}}^{(1)}, \widetilde{\mathcal{G}}^{(2)}, \ldots, \widetilde{\mathcal{G}}^{(T)}\}$, a temporal randomized mechanism $\mathbb{M} = \{\mathbb{M}^{(1)}, \mathbb{M}^{(2)}, \ldots, \mathbb{M}^{(T)}\}$ satisfies the temporal edge-level differential privacy under the budget $\varepsilon > 0$, if and only if

$$\begin{aligned} &Pr[\mathbb{M}^{(t)}(\mathbf{A}^{(t)}) \in \mathbf{S}^{(t)}] \leq Pr[\mathbb{M}^{(t)}(\widetilde{\mathbf{A}}^{(t)}) \in \mathbf{S}^{(t)}] \times e^{\varepsilon^{(t)}} \\ &\text{and } \sum_{t} \varepsilon^{(t)} \leq \varepsilon, \quad \text{for } t \in \{1, 2, \ldots, T\} \end{aligned} \tag{6.15}$$

Next, we introduce our temporal randomized mechanism $\mathbb{M} = \{\mathbb{M}^{(1)}, \mathbb{M}^{(2)}, \ldots, \mathbb{M}^{(T)}\}$, and give the proof under temporal edge-level differential privacy.

Perturbation Operations. For a snapshot $\mathcal{G}^{(t)}$ at time t, $\mathbb{M}^{(t)}$ contains two independent perturb operations, i.e., *absent edge flip* $\mathbb{M}_{+}^{(t)}$ and *existing edge flip* $\mathbb{M}_{-}^{(t)}$, proposed to be directly applied on the adjacency matrix $\mathbf{A}^{(t)}$ to obtain $\hat{\mathbf{A}}^{(t)}$. As the name suggests, the *absent edge flip* is responsible for adding an absent edge and the *existing edge flip* tries to delete an existing edge. These two perturbation operations are sequentially and interchangeably operated on the structure (i.e., adjacency matrix $\mathbf{A}^{(t)}$) of snapshot $\mathcal{G}^{(t)}$ to get the perturbed structure (i.e., adjacency matrix $\hat{\mathbf{A}}^{(t)}$) for the new snapshot $\hat{\mathcal{G}}^{(t)}$. Following this way, we apply $\mathbb{M}^{(t)} = (\mathbb{M}_{+}^{(t)}, \mathbb{M}_{-}^{(t)})$ to each snapshot $\mathcal{G}^{(t)}$ such that we can get the perturbed temporal graph $\hat{\mathcal{G}} = \{\hat{\mathcal{G}}^{(1)}, \hat{\mathcal{G}}^{(2)}, \ldots, \hat{\mathcal{G}}^{(T)}\}$. $\mathbb{M}_{+}^{(t)}$ and $\mathbb{M}_{-}^{(t)}$ are expressed below.

$$\mathbb{M}_{+}^{(t)} = \begin{cases} 1/e^{\varepsilon_{+}^{(t)}} & \text{Prob. of adding an absent edge} \\ 1 - 1/e^{\varepsilon_{+}^{(t)}} & \text{Prob. of ignoring that absent edge} \end{cases} \tag{6.16}$$

$$\mathbb{M}_{-}^{(t)} = \begin{cases} 1/e^{\varepsilon_{-}^{(t)}} & \text{Prob. of deleting an existing edge} \\ 1 - 1/e^{\varepsilon_{-}^{(t)}} & \text{Prob. of ignoring that existing edge} \end{cases} \tag{6.17}$$

where e is Euler's number, and an absent edge means $\mathbf{A}^{(t)}(i,j) = 0$. Then, given the privacy budget $\varepsilon_{+}^{(t)} > 0$, with the probability $1/e^{\varepsilon_{+}^{(t)}}$, $\mathbb{M}_{+}^{(t)}$ will flip $\mathbf{A}^{(t)}(i,j)$ from 0 to 1; Or $\mathbb{M}_{+}^{(t)}$ will

Algorithm 6.1: Graph Perturbation based on Differential Privacy

Require:
graph snapshot $\mathcal{G}^{(t)}$, privacy budgets $e^{\varepsilon_+^{(t)}}$ and $e^{\varepsilon_-^{(t)}}$
Ensure:
graph snapshot $\hat{\mathcal{G}}^{(t)}$

for a certain edge type **do**
/* Apply $\mathbb{M}_+^{(t)}$ */
  for edge in non-existing_set **do**
    Randomly generate a probability α
    if $\alpha \leq 1/e^{\varepsilon_+^{(t)}}$ **then**
      Add edge to set append_set
    end if
  end for
/* Apply $\mathbb{M}_-^{(t)}$ */
  for edge in existing_set **do**
    Randomly generate a probability α
    if $\alpha \leq 1/e^{\varepsilon_-^{(t)}}$ **then**
      Add edge to set del_set
    end if
  end for
end for
residual_set = set_diff(existing_set, del_set)
$\hat{\mathcal{G}}^{(t)}$ = set_union(residual_set, append_set)

do nothing to $\mathbf{A}^{(t)}(i,j)$ with the probability $1 - 1/e^{\varepsilon_+^{(t)}}$, similarly on $\varepsilon_-^{(t)} > 0$.

When sequentially executing $\mathbb{M}_+^{(t)}$ and $\mathbb{M}_-^{(t)}$ on a single snapshot graph $\mathcal{G}$, they are executed on different edge sets (i.e., existing edge set and absent edge set) and not overwrite each other, as shown in Alg. 6.1. In practice, given the high sparsity of real-world graphs, iterating through non-existing edges could lead to high time complexity. Therefore, we apply $\mathbb{M}_+^{(t)}$ by first sampling a larger edge set (contains mostly non-existing edges and a small portion of existing edges), then remove those existing edges, and sample the number of edges we need to append. In this way, the time complexity of Alg. 6.1 is approximately the number of edges to be sampled.

Theorem 6.2. The proposed perturbation operation for temporal heterogeneous graphs in Alg. 6.1 is an ϵ-edge-level differential privacy algorithm, which is bounded by $2 \cdot max(e^{\varepsilon_+^{(t)}}, e^{\varepsilon_-^{(t)}})$ at each t. (As proved below)

For the proposed perturbation operation, we first prove that edge-level differential privacy is guaranteed for each timestamp, and we then prove that temporal edge-level differential

privacy is guaranteed for the entire temporal graph. Denoting the perturbed adjacency matrix as $\mathbb{M}^{(t)}(\mathbf{A}^{(t)})$, the probability of the perturbed matrix equals a certain matrix $\mathbf{S}^{(t)}$ is

$$Pr[\mathbb{M}^{(t)}(\mathbf{A}^{(t)}) = \mathbf{S}^{(t)}] = \prod_{k} Pr[\mathbb{M}_k^{(t)}(\mathbf{A}^{(t)}) = \mathbf{S}_k^{(t)}] \tag{6.18}$$

where the $\mathbb{M}_k^{(t)}$ includes two randomization $\mathbb{M}_+^{(t)}$ and $\mathbb{M}_-^{(t)}$.

Then, given two snapshots $\mathcal{G}^{(t)}$ and $\widetilde{\mathcal{G}}^{(t)}$, the ratio of the probability of perturbing on $\mathbf{A}^{(t)}$ and $\widetilde{\mathbf{A}}^{(t)}$ and outputting the same output can be expressed as follows,

$$\begin{aligned} &\frac{Pr[\mathbb{M}^{(t)}(\mathbf{A}^{(t)}) = \mathbf{S}^{(t)}]}{Pr[\mathbb{M}^{(t)}(\widetilde{\mathbf{A}}^{(t)}) = \mathbf{S}^{(t)}]} \quad \text{s.t.} \quad \text{for } i,j \in \{1,2,\ldots,n\} \\ =&\prod_{v_i,v_j} \frac{Pr[\mathbb{M}_+^{(t)}(A(i,j)) = S(i,j)]}{Pr[\mathbb{M}_+^{(t)}(\widetilde{A}(i,j)) = S(i,j)]} \prod_{v_i,v_j} \frac{Pr[\mathbb{M}_-^{(t)}(A(i,j)) = S(i,j)]}{Pr[\mathbb{M}_-^{(t)}(\widetilde{A}(i,j) = S(i,j)]} \end{aligned} \tag{6.19}$$

which is a general formula over all possible edges. To be specific, given two adjacent snapshots $\mathcal{G}^{(t)}$ and $\widetilde{\mathcal{G}}^{(t)}$, we can denote the difference as an absent edge (u,v), e.g., $\mathcal{G}^{(t)}$ does not have it but $\widetilde{\mathcal{G}}^{(t)}$ does. Then, according to Alg. 6.1 and Eq. 6.19, the ratio Γ of the marginal probability of outputting the same output is below, where the first term represents that $\mathbb{M}_+^{(t)}$ successfully flips $A(u,v)^{(t)}$ and $\mathbb{M}_-^{(t)}$ can do nothing on the flipped $A(u,v)^{(t)}$ based on Alg. 6.1.

$$\begin{aligned} \Gamma = &\underbrace{\frac{Pr[A(u,v)^{(t)} = 0 \to 1]}{Pr[\widetilde{A}(u,v)^{(t)} = 1 \to 1]}}_{\mathbb{M}_+^{(t)}} \underbrace{\frac{Pr[A(u,v)^{(t)} = 1 \to 1]}{Pr[\widetilde{A}(u,v)^{(t)} = 1 \to 1]}}_{\mathbb{M}_-^{(t)}} + \\ &\underbrace{\frac{Pr[A(u,v)^{(t)} = 0 \to 0]}{Pr[\widetilde{A}(u,v)^{(t)} = 1 \to 1]}}_{\mathbb{M}_+^{(t)}} \underbrace{\frac{Pr[A(u,v)^{(t)} = 0 \to 0]}{Pr[\widetilde{A}(u,v)^{(t)} = 1 \to 0]}}_{\mathbb{M}_-^{(t)}} \\ = &\frac{1/e^{\varepsilon_+^{(t)}}}{1} \times \frac{1}{1 - 1/e^{\varepsilon_-^{(t)}}} + \frac{1 - 1/e^{\varepsilon_+^{(t)}}}{1} \frac{1}{1/e^{\varepsilon_-^{(t)}}} \\ = &\frac{e^{\varepsilon_-^{(t)}}}{e^{\varepsilon_+^{(t)}}} \times \frac{1}{e^{\varepsilon_-^{(t)}} - 1} + e^{\varepsilon_-^{(t)}} - \frac{e^{\varepsilon_-^{(t)}}}{e^{\varepsilon_+^{(t)}}} \\ < &\frac{e^{\varepsilon_-^{(t)}}}{e^{\varepsilon_+^{(t)}}} \times \frac{1}{e^{\varepsilon_-^{(t)}} - 1} + e^{\varepsilon_-^{(t)}} \\ < &\frac{1}{e^{\varepsilon_+^{(t)}}} + e^{\varepsilon_-^{(t)}} \end{aligned} \tag{6.20}$$

Then, we can derive Theorem 6.2 based on Eq. 6.20. Moreover, we have analyzed the

differential privacy scenario that the later mechanism can overwrite (e.g., $\mathbb{M}_{-}^{(t)}$ flips the edge that $\mathbb{M}_{+}^{(t)}$ has just flipped) in the appendix, and the bound is looser than in Eq. 6.20 (i.e., larger than $\frac{1}{e^{\varepsilon_{+}^{(t)}}} + e^{\varepsilon_{-}^{(t)}}$), which suggests that forbidding the overwrite is a safer way to protect privacy.

Data-Driven $\varepsilon^{(t)}$ Determination. In Alg. 6.1, the perturbation process may result in a privacy-preserving graph but a totally different topological distribution and thus hurts the utility of the original graph. Also, the dramatic change compared with the original graph is prone to collapse the adversarial training of the next stage.

Hence, we remedy the graph distortion caused by inserted and deleted edges, with respect to the degree distribution. To achieve that, from the data-driven angle, we give the relation between $\varepsilon_{+}^{(t)}$ and $\varepsilon_{-}^{(t)}$ for $\mathcal{G}^{(t)}$ based on the expectation of numbers of added edges and detected edges at each time t. For the notation clarity, the below equation is expressed in the homogeneous setting but can be easily extended to the heterogeneous graph by constraining node types.

$$
\begin{aligned}
m^{(t)} \frac{1}{e^{\varepsilon_{-}^{(t)}}} &\approx ((n^{(t)})^2 - m) \frac{1}{e^{\varepsilon_{+}^{(t)}}} \\
\varepsilon_{-}^{(t)} - \varepsilon_{+}^{(t)} &\approx ln \frac{m^{(t)}}{(n^{(t)})^2 - m^{(t)}}
\end{aligned}
\tag{6.21}
$$

where $n^{(t)}$ denotes the number of non-dangling nodes in $\mathcal{G}^{(t)}$, and $m^{(t)}$ denotes the number of edges in $\mathcal{G}^{(t)}$.

Privacy-Utility Adversarial Generation To integrate the privacy of perturbed graph $\hat{\mathcal{G}}$ and the utility of original graph $\mathcal{G}$, we propose a privacy-utility adversarial learning method. The method consists of three modules: privacy-guaranteed discriminator $\mathbb{D}_{\text{prv}}$, utility-preserving discriminator $\mathbb{D}_{\text{util}}$, and generator $\mathbb{G}$. Moreover, during the adversarial training, we use the sampled subgraphs (e.g., via random walks) of the input graph. To be specific, at each time t, if a node v_i has connections, we can use a heterogeneous random walk method [359] to get an l-th node sequence, as $\mathcal{W}_i^{(t)} = \{\{v_1^{(t)}, v_2^{(t)}, ..., v_l^{(t)}\}, \{o_1, o_2, ..., o_l\}\}$, where v_j denotes the node and o_j denotes the node type of v_j. Therefore, we also design an assembler for aggregating generated walks of generator $\mathbb{G}$ into a privacy-guaranteed and utility-preserved temporal heterogeneous graph $\mathcal{G}^*$.

Utility-preserving Discriminator $\mathbb{D}_{util}$. This discriminator is designed to distinguish generated graphs $\hat{\mathcal{G}}$ and original graphs $\mathcal{G}$, such that if the generator $\mathbb{G}$ can bypass discriminator $\mathbb{D}_{\text{util}}$, i.e., $\mathbb{D}_{\text{util}}$ cannot distinguish the generated walks sampled from real graphs, then the generated graph is 'close enough' to the real graph and preserves the utility. On the other

side, given a bunch of walks sampled from real graph $\{\mathcal{W}_1, \mathcal{W}_2, ..., \mathcal{W}_k\}$ and some generated walks $\{\mathcal{W}_1^*, \mathcal{W}_2^*, ..., \mathcal{W}_k^*\}$, discriminator $\mathbb{D}_{\text{util}}$ should have a confident bound for

$$\sum_{i=1}^{k} \mathbb{D}_{\text{util}}(\mathcal{W}_i) >> \sum_{i=1}^{k} \mathbb{D}_{\text{util}}(\mathcal{W}_i^*) \tag{6.22}$$

where we omit the superscript for time and the subscript for node index in $\mathcal{W}_i^{(t)}$, the current $\mathcal{W}_i$ means the i-th walk in the sampled walks that consists of multiple walks sampled for different nodes from different timestamps.

To achieve the above goal, we extend our discriminator $\mathbb{D}_{\text{util}}$ from bi-level self-attention [360] to the tri-level self-attention. On top of that, we first use a model-agnostic projection [361] to map node $v_i^{(t)}$ into an embedding vector $\mathbf{h}_i^{(t)}$. Then, with a sampled walk $\mathcal{W}_i^{(t)} = \{\{v_1^{(t)}, v_2^{(t)}, ..., v_l^{(t)}\}, \{o_1, o_2, ..., o_l\}\}$, the tri-level self-attention mechanism ATN is expressed as follows.

$$ATN^{time}(v_i^{(t)}, v_j^{(t)}) = \frac{(\mathbf{h}_i^{(t)}\mathbf{W}_Q^{time}) \odot (\mathbf{h}_j^{(t)}\mathbf{W}_K^{time})}{\sqrt{d_k}} \tag{6.23}$$

$$ATN^{node}(v_i, v_j) = \frac{(\mathbf{h}_i\mathbf{W}_Q^{node}) \odot (\mathbf{h}_j\mathbf{W}_K^{node})}{\sqrt{d_k}} \tag{6.24}$$

$$\mathbf{h}_i = \sum_{t \in \{1,...,T\}} \mathbf{h}_i^{(t)} \tag{6.25}$$

$$ATN^{type}(o_i, o_j) = \frac{(\mathbf{o}_i\mathbf{W}_Q^{type}) \odot (\mathbf{o}_j\mathbf{W}_K^{type})}{\sqrt{d_k}} \tag{6.26}$$

whereas $\mathbf{o}_i$ is the one-hot embedding of node type o_i. Matrices $\mathbf{W}_Q^{time}, \mathbf{W}_Q^{node}, \mathbf{W}_Q^{type}$ are the time-level, node-level, and type-level query weight matrices; $\mathbf{W}_K^{time}, \mathbf{W}_K^{node}, \mathbf{W}_K^{type}$ are the time-level, node-level, and type-level key weight matrices.

To integrate the attention in these three different levels, we define the pooling operation of the tri-level self-attention as

$$\mathbf{Z} = (\text{softmax}(\mathbf{A}^{time}) + \text{softmax}(\mathbf{A}^{node}) + \text{softmax}(\mathbf{A}^{type}))\mathbf{V} \tag{6.27}$$

where $\mathbf{V} = \mathbf{W}_V\mathbf{H}, H(v, :) = \mathbf{h}_v$, and $\mathbf{W}_V$ is the value weight matrix.

Finally, the multi-head attention layers are followed by a fully connected layer and a softmax function to output a score estimating the probability that the input walk is sampled from the real graph.

To train $\mathbb{D}_{\text{util}}$ for fulfilling the goal in Eq. 6.22, the objective function for optimizing $\mathbb{D}_{\text{util}}$

is to minimize the following loss.

$$\mathcal{L}_{\mathbb{D}_{\text{util}}}(\{\mathcal{W}\},\{\mathcal{W}^*\}) = \sum_{i}^{|\{\mathcal{W}\}|} \mathbb{D}_{\text{util}}(\mathcal{W}_i^*) - \mathbb{D}_{\text{util}}(\mathcal{W}_i) \tag{6.28}$$

where $\{\mathcal{W}\}$ and $\{\mathcal{W}^*\}$ are the mini-batches of sampled walks.

For realization, we choose to train $\mathbb{D}_{\text{util}}$ with randomly generated negative samples instead of generated ones from generator $\mathbb{G}$, whose operation actually allows us to detach $\mathbb{D}_{\text{util}}$ from the min-max manner [362] and pre-train $\mathbb{D}_{\text{util}}$. In this way, $\mathbb{D}_{\text{util}}$ acts like a good scoring function for the penalty in the adversarial training. Although involving $\mathbb{D}_{\text{util}}$ with the generator might lead to a higher utility, it will increase the difficulty of model tuning complexity for training collapse and inefficiency. Our experiments also corroborate that the pre-trained $\mathbb{D}_{\text{util}}$ is sufficiently good for graph generation.

Privacy-guaranteed Discriminator $\mathbb{D}_{prv}$. This discriminator aims to discriminate generated graphs $\mathcal{G}^*$ and perturbed graphs $\hat{\mathcal{G}}$. Again, if the discriminator $\mathbb{D}_{\text{prv}}$ cannot discriminate walks in the generated graph and walks sampled from the perturbed graph, the privacy of the generated graph can be guaranteed. On the contrary, a good $\mathbb{D}_{\text{prv}}$ should satisfy

$$\sum_{i=1}^{k} \mathbb{D}_{\text{prv}}(\hat{\mathcal{W}}_i) >> \sum_{i=1}^{k} \mathbb{D}_{\text{prv}}(\mathcal{W}_i^*) \tag{6.29}$$

where $\{\hat{\mathcal{W}}_1, \hat{\mathcal{W}}_2, ..., \hat{\mathcal{W}}_k\}$ are random walks sampled from $\hat{\mathcal{G}}$, and $\{\mathcal{W}_1^*, \mathcal{W}_2^*, ..., \mathcal{W}_k^*\}$ are walks generated by generator $\mathbb{G}$.

With similar intuition as $\mathbb{D}_{\text{util}}$, $\mathbb{D}_{\text{prv}}$ shares the same neural architecture with $\mathbb{D}_{\text{util}}$. The loss function for training $\mathbb{D}_{\text{prv}}$ is defined as

$$\mathcal{L}_{\mathbb{D}_{\text{prv}}}(\{\hat{\mathcal{W}}\},\{\mathcal{W}^*\}) = \sum_{i}^{|\{\hat{\mathcal{W}}\}|} \mathbb{D}_{\text{prv}}(\mathcal{W}_i^*) - \mathbb{D}_{\text{prv}}(\hat{\mathcal{W}}_i) \tag{6.30}$$

where $\mathbb{D}_{\text{prv}}$ and generator $\mathbb{G}$ are trained alternatively in Alg. 6.2.

Generator $\mathbb{G}$. The generator aims to generate synthetic graphs that are both privacy-guaranteed and utility-preserved. We first introduce how to sample nodes to generate temporal heterogeneous walks, then we introduce how the generated walks get constrained by the designed loss function to preserve both privacy and utility. In the next subsection, we introduce how the generated walks are aligned into a temporal heterogeneous graph by the assembler.

To begin with, we sample the node with node type and timestamp based on the perturbed

graph $\hat{\mathcal{G}}$, which could ensure higher privacy-preserving than just sampling from the original graph $\mathcal{G}$. The walk generation relies on the hidden states of recurrent neural architectures (e.g., LSTM [363]) to generate an L-th walk, i.e., the generator sequentially samples timestamped nodes with type based on the hidden state, the corresponding mechanism is described as follows. For the l-th step in walks, i.e., the l-th block in LSTM f_θ, for $l \in \{1, \ldots, L\}$, we have

$$\begin{aligned}(m_l, h_l) &= f_\theta(m_{l-1}, a_{l-1}) \\ o_l &\sim g_o(h_l) \ / * \text{ sample node type} * / \\ v_l &\sim g_v(h_l, o_l) \ / * \text{ sample timestamped node} * / \\ a_l &= g_c(o_l, v_l)\end{aligned} \tag{6.31}$$

where v_l is the l-th timestamped node, and o_l is the node type. h_l and m_l are the hidden state and memory state of LSTM, and a_l is the input for the next recurrent block. g_o is implemented with a fully connected layer and Gumbel Softmax, which aims to sample the node type based on the hidden state. g_v is implemented by a bag of fully connected layers, each one corresponding to one node type; g_v generates an expected timestamped node embedding $\mathbf{h}^{*(t)}$ for each node type; Given that, we can sample timestamped node embedding $\hat{\mathbf{h}}^{(t)}$ in $\hat{\mathcal{G}}$ based on the distance of embedding vectors, the detailed distance function is stated in Eq. 6.32. And $g_c = f_o(\text{onehot}(o_l)) + f_v(\hat{h}_{v_l}^{(t_{v_l})})$, f_o and f_v are both fully-connected layers, where t_{v_l} is the timestamp of node v_l.

In the above formula, for the initialization, $a_0 = 0$, $m_0 = f_0(z)$, where f_0 is a fully connected layer and z is sampled from normal distribution $z \sim \mathcal{N}(0, 1)$. To be specific, a timestamped node v_l with type o_l is sampled by the highest probability based on the vector distance as below

$$\frac{e^{-||h^{*(t)} - \hat{h}_{v_l}^{(t_{v_l})}||}}{\sum_{v \text{ s.t. } o(v)=o_l} e^{-||h^{*(t)} - \hat{h}_v^{(t_v)}||}} \tag{6.32}$$

where $o(v)$ means the node type of a node v.

Note that we do not enforce adjacent nodes to share the same timestamp to increase the diversity of the generated graph. In the next assembling step, edges connecting nodes from different timestamps will be added to both snapshots.

To generate the privacy-guaranteed and utility-preserving graph, the walks generated by $\mathbb{G}$ try to bypass both $\mathbb{D}_{\text{util}}$ and $\mathbb{D}_{\text{prv}}$. Therefore the loss function for training $\mathbb{G}$ is defined as

$$\mathcal{L}_{\mathbb{G}}(\{\mathcal{W}^*\}) = -\sum_{i=1}^{|\{\mathcal{W}^*\}|} (\mathbb{D}_{\text{util}}(\mathcal{W}_i^*) + \mathbb{D}_{\text{prv}}(\mathcal{W}_i^*)) \tag{6.33}$$

Algorithm 6.2: Privacy-Utility Adversarial Training

Require:
sampled walks $\{\mathcal{W}\}$ from $\mathcal{G}$, sampled walks $\{\hat{\mathcal{W}}\}$ from $\hat{\mathcal{G}}$, randomly sampled walks $\{\hat{\mathcal{W}}'\}$,

Ensure:
synthetic walks generator $\mathbb{G}$

1: **while** until not converge **do**
2: Initialize $\mathbb{D}_{\text{util}}$, $\mathbb{D}_{\text{prv}}$, $\mathbb{G}$
3: **for** batches $\mathcal{B}$ in $\{\mathcal{W}\}$, $\mathcal{B}'$ in $\{\mathcal{W}'\}$ **do**
4: Calculate $\mathcal{L}_{\mathbb{D}_{\text{util}}}(\mathcal{B}, \mathcal{B}')$ in Eq. 6.28
5: Update $\mathbb{D}_{\text{util}}$
6: **end for**
7: **end while**
8: **while** until not converge **do**
9: **for** batches $\hat{\mathcal{B}}$ in $\{\hat{\mathcal{W}}\}$ **do**
10: Initialize noise $z = \{z_i \sim \mathcal{N}(0, 1)\}$
11: $\mathcal{B}^* = \mathbb{G}(z)$
12: Calculate $\mathcal{L}_{\mathbb{D}_{\text{prv}}}(\hat{\mathcal{B}}, \mathcal{B}^*)$ in Eq. 6.30
13: Update $\mathbb{D}_{\text{prv}}$
14: Calculate $\mathcal{L}_{\mathbb{G}}(\mathcal{B}^*)$ in Eq. (6.30)
15: Update $\mathbb{G}$
16: **end for**
17: **end while**

The general training process is summarized in Alg. 6.2.

Claim 6.1. Since our utility discriminator $\mathbb{D}_{\text{util}}$ is pre-trained as shown in lines 3–6 of Alg. 2, and the generator $\mathbb{G}$ does not touch any real data as it is only listening to decisions of discriminators, our generative and adversarial game can be rewritten in the form of Eq. 2 and Eq. 3 of [364]. In [364], generative adversarial networks have been proved to have an outstanding generalization property that can be interpreted as constraining the input and output under differential privacy constraints.

Thus, we only need to make one extra assumption that the sampled temporal heterogeneous subgraphs are independent and identically distributed, and then we can reach a generalization bound of our adversarial learning from the perspective of differential privacy [364], which means the adversarial learning is also under differential privacy if the discriminator is well converged (i.e., stable). Moreover, that assumption is mild, since all subgraphs originate from a whole larger graph, we can assume they are identically distributed; because we independently sample each subgraph, we can also assume they are independently distributed.

Assembler. Here, we extend the assembling mechanism in [359] to assemble a temporal

heterogeneous graph. First, we generate sufficient numbers of walks from the generator. Then, we construct a scoring matrix $\mathbf{S}$ by counting the existing edges in the generated walks and find all meta-paths in the generated walks. To construct the generated graph, we start with sampling a node $v_1 \sim Pr(v)$ according to probability

$$Pr(v_1 = u) = \frac{\sum_{v \in \hat{\mathcal{G}}}(\mathbf{S}(u, v))}{\sum_{v_i \in \hat{\mathcal{G}}} \sum_{v_j \in \hat{\mathcal{G}}} \mathbf{S}(v_i, v_j)} \tag{6.34}$$

where $\mathbf{S}(u, v)$ counts how many connections between node u and v is the generated walks.

Then, we sample a meta-path according to meta-paths' frequencies. Following that meta-path $\mathbf{O} = (o_1, ...o_l)$, we sequentially sample the next node $v_{i+1}(i \geq 1)$ with node type o_{i+1} according to probabilities proportional to scoring matrix.

$$Pr(v_{i+1} = u) = \frac{\mathbf{1}(o(u) = o_{i+1})\mathbf{S}(v_i, u)}{\sum_{v \in \hat{\mathcal{G}} \wedge o(v) = o_{i+1}} \mathbf{S}(v_i, v)} \tag{6.35}$$

where $o(v)$ is the node type of node v.

Finally, we add all edges in the generated meta-path instances into the generated graph. Note that the timestamp of graphs naturally goes with the timestamp of nodes.

6.2.4 Empirical Evaluation

Datasets To test the performance, we utilize 4 real-world temporal heterogeneous graph datasets from academic citation graphs, online rating graphs, and online shopping graphs. The detailed statistics are summarized in Table 6.4. For the DBLP dataset, we retrieve the subgraph by constraining the paper venue as KDD.

Table 6.4: Temporal Heterogeneous Graph Datasets

	Type	#Nodes	#Node Types	#Edges	#Edge Types	Time
MovieLens 100k[365]	Rating	2,644	4	102,625	3	7 months
DBLP[366]	Citation	17,876	4	51,137	4	10 years
MovieLens 20M[365]	Rating	165,790	3	20,027,541	2	21 years
Taobao[17]	E-Commerce	1,009,827	3	2,932,288	2	10 hours

Baselines The comparison baselines include 4 kinds of generative models. HGEN [359] is a heterogeneous graph generation algorithm. TagGen [360] and TG-GAN [367] are temporal

[17] https://tianchi.aliyun.com/dataset/649

graph generation algorithms. EDGE [368] is designed for plain graph generation based on diffusion models. Our PUG stands for temporal and heterogeneous graph generation.

Table 6.5: Comprehensive Evaluation of Generated Temporal Heterogeneous Graphs, where e denotes the scientific notation, red denotes the first place, and blue denotes the second place.

Datasets	Methods	Cluster (↓)	TC (↓)	LCC (↓)	Degree (↓)	Meta-2 (↓)	Meta-3 (↓)	EO-Rate (↓)
ML-100k	HGEN [359]	$5.871e-02(6.971e-04)$	$1.000e+00(0.000e+00)$	$1.905e-01(0.000e+00)$	$1.395e-01(1.353e-03)$	$1.469e+00(1.850e-04)$	$1.289e+00(4.450e-05)$	$3.200e-02(2.008e-04)$
	TagGen [360]	$3.021e-01(1.561e-01)$	$1.000e+00(0.000e+00)$	$6.417e-01(2.265e-02)$	$1.570e-01(5.345e-03)$	$1.500e+00(1.306e-01)$	$1.310e+00(3.281e-03)$	$2.238e-01(1.159e-01)$
	TG-GAN [367]	$4.987e-01(7.305e-02)$	$1.000e+00(0.000e+00)$	$2.143e-01(2.381e-02)$	$2.097e-01(8.056e-02)$	$1.527e+00(2.535e-02)$	$1.308e+00(1.600e-05)$	$2.666e-02(4.729e-03)$
	EDGE [368]	$3.960e-03(5.544e-06)$	$1.000e+00(0.000e+00)$	$2.955e-01(1.663e-01)$	$1.569e-01(4.883e-04)$	$1.744e+00(6.850e-06)$	$1.313e+00(0.000e+00)$	$5.488e-03(1.143e-04)$
	PUG	$2.536e-03(5.673e-04)$	$1.532e+00(0.000e+00)$	$2.636e-01(7.055e-02)$	$3.547e-01(1.283e-02)$	$1.664e+00(1.922e-03)$	$1.313e+00(1.950e-05)$	$2.247e-02(5.652e-03)$
DBLP	HGEN [359]	$1.088e-08(4.790e-10)$	$4.270e-01(3.150e-02)$	$0.000e+00(0.000e+00)$	$1.191e-01(3.784e-05)$	$1.566e+00(5.550e-05)$	$9.121e-01(3.059e-04)$	$3.064e-04(1.623e-05)$
	TagGen [360]	$1.574e-02(1.437e-03)$	$1.000e+00(0.000e+00)$	$0.000e+00(0.000e+00)$	$3.339e-01(1.455e-01)$	$1.821e+00(2.987e-02)$	$9.193e-01(8.229e-03)$	$1.164e-01(2.001e-02)$
	TG-GAN [367]	$6.980e-03(6.585e-03)$	$1.000e+00(0.000e+00)$	$0.000e+00(0.000e+00)$	$1.496e-01(4.812e-02)$	$1.794e+00(7.134e-03)$	$9.155e-01(1.056e-02)$	$2.090e-03(7.383e-04)$
	EDGE [368]	- - - Generates Empty Graph - - -						
	PUG	$0.000e+00(0.000e+00)$	$1.192e-07(0.000e+00)$	$2.466e-02(2.344e-02)$	$1.265e-01(5.729e-02)$	$1.839e+00(1.179e-04)$	$9.145e-01(4.435e-04)$	$6.908e-05(3.402e-05)$
ML-20M	HGEN [359]	$2.217e-08(4.491e-09)$	$2.835e-01(8.390e-03)$	$2.552e-10(2.552e-10)$	$7.843e-02(8.729e-05)$	$1.091e+00(1.862e-04)$	$1.232e+00(1.530e-04)$	$4.266e-03(8.808e-05)$
	TagGen [360]	$7.934e-05(0.000e+00)$	$1.008e+00(0.000e+00)$	$0.000e+00(0.000e+00)$	$1.263e-01(0.000e+00)$	$1.532e+00(3.200e-06)$	$1.266e+00(0.000e+00)$	$5.105e-01(0.000e+00)$
	TG-GAN [367]	$2.154e-09(2.154e-09)$	$3.056e-02(3.056e-02)$	$2.215e-01(2.119e-01)$	$5.431e-01(9.484e-03)$	$1.328e+00(6.003e-03)$	$1.257e+00(6.818e-03)$	$2.339e-03(2.915e-04)$
	EDGE [368]	- - - OOM - - -						
	PUG	$0.000e+00(0.000e+00)$	$0.000e+00(0.000e+00)$	$5.168e-01(1.653e-01)$	$5.596e-01(1.225e-02)$	$1.081e+00(4.282e-03)$	$1.266e+00(1.000e-06)$	$2.305e-03(1.856e-04)$
Taobao	HGEN [359]	$0.000e+00(0.000e+00)$	$1.192e-07(0.000e+00)$	$0.000e+00(0.000e+00)$	$6.137e-04(1.382e-05)$	$1.233e+00(2.706e-02)$	$6.133e-01(0.000e+00)$	$1.127e-04(7.581e-06)$
	TagGen [360]	$0.000e+00(0.000e+00)$	$1.192e-07(0.000e+00)$	$0.000e+00(0.000e+00)$	$4.515e-04(4.168e-05)$	$1.540e+00(2.941e-03)$	$6.133e-01(0.000e+00)$	$2.082e-02(1.365e-04)$
	TG-GAN [367]	$0.000e+00(0.000e+00)$	$1.192e-07(0.000e+00)$	$4.982e-01(4.363e-02)$	$7.204e-04(5.501e-06)$	$1.526e+00(2.498e-04)$	$6.133e-01(0.000e+00)$	$5.021e-06(2.063e-06)$
	EDGE [368]	- - - OOM - - -						
	PUG	$0.000e+00(0.000e+00)$	$1.192e-07(0.000e+00)$	$5.637e-01(8.726e-02)$	$7.545e-04(6.616e-07)$	$1.493e+00(1.116e-03)$	$6.133e-01(5.000e-07)$	$4.367e-06(4.367e-06)$

Evaluation Metrics We propose 6 metrics measuring four different aspects of temporal heterogeneous graph generation quality, including plain graph utility, temporal graph utility, heterogeneous graph utility, and overall privacy.

Plain Graph Utility. We propose Node Degree Distribution by Maximum Mean Discrepancy Distance (i.e., MMD), named **Degree** in short, which is expressed as below.

$$Dist(\mathbf{d}_{ori}, \mathbf{d}_{gen}) = \frac{1}{T}\sum_{t=1}^{T} MMD(\mathbf{d}_{ori}^{(t)}, \mathbf{d}_{gen}^{(t)}) \tag{6.36}$$

where $\mathbf{d}_{ori}^{(t)}, \mathbf{d}_{gen}^{(t)}$ are degrees of nodes in the original and generated graph at time t, and MMD is averaged over all timestamps.

$$MMD(x, x') = ker(x, x) + ker(x', x') - 2ker(x, x') \tag{6.37}$$

where the kernel function $ker(\cdot)$ is $\sum_{j=1}^{k} e^{-\alpha_j||x-x'||^2}$, and k and α_j are constants. In the evaluation, we set $k = 1$ and $\alpha = 4$.

Temporal Graph Utility. We extend the Clustering Coefficient (i.e., **Cluster**), Size of the Largest Connected Component (i.e., **LCC**), and Triangle Count (i.e., **TC**) to the temporal setting, by calculating those metrics at each timestamp and using Maximum Mean Discrepancy Distance (MMD) between the sequences of metrics over time of the original graph $\mathcal{G}$

and the generated graph $\mathcal{G}^*$. Taking LCC as an example,

$$MMD([LCC(\mathcal{G}^{(1)}), LCC(\mathcal{G}^{(2)}), ...LCC(\mathcal{G}^{(T)})] \\ , [LCC(\mathcal{G}^{*(1)}), LCC(\mathcal{G}^{*(2)}), ..., LCC(\mathcal{G}^{*(T)})]) \quad (6.38)$$

Heterogeneous Graph Utility. For each timestamp, we count the number of length-2 and length-3 meta-path instances and get a probability distribution of meta-paths. Cross Entropy is applied for length-2 and length-3 respectively to measure distance, and the mean over time is taken to evaluate the overall performance. The corresponding metrics are named **Meta-2** and **Meta-3**. Specifically, we only preserve the meta-paths that exist in the original graph, and all other meta-paths in the generated graph are combined into one category. For example, in the MovieLens 100k dataset, we consider length-2 meta-path distributions over $\mathcal{M}_2 =$ ['user-occupation', 'movie-genre', 'user-movie', 'none of above'].

$$Dist(Pr_{\mathcal{G}}(\mathcal{M}), Pr_{\mathcal{G}^*}(\mathcal{M})) = \frac{1}{T}\sum_{t=1}^{T}\sum_{m\in\mathcal{M}}(-\log\frac{\exp(Pr_{\mathcal{G}^*}^{(t)}(m))}{\sum_{m\in\mathcal{M}}\exp(Pr_{\mathcal{G}^*}^{(t)}(m))}Pr_{\mathcal{G}}^{(t)}(m)) \quad (6.39)$$

where $Pr_{\mathcal{G}}(m)$ denotes the probability distribution of the meta-path m in the graph $\mathcal{G}$, and $\mathcal{M}$ is a collection of meta-path m.

Overall Privacy. **EO-Rate** is defined as the ratio of overlapped edges (i.e., in terms of existence and edge type) between the original graphs and the generated graph, which is used to analyze the privacy-preserving property of the generation, and is averaged over time.

$$\text{EO_Rate}(\mathcal{G}^*) = \frac{1}{T}\sum_{t=1}^{T}\frac{|edges(\mathcal{G}^{(t)})\bigcap edges(\mathcal{G}^{*(t)})|}{|edges(\mathcal{G}^{*(t)})|} \quad (6.40)$$

Main Results Experiment results are shown in Table 6.5, where we report the average and standard deviation of the generated temporal heterogeneous graphs with respect to the 6 proposed metrics. In general, our proposed PUG framework generates very competitively utilizable temporal heterogeneous graphs compared with the state-of-the-art baselines, especially in the large-scale datasets, DBLP, MovieLens 20M (i.e., ML-20M), and Taobao datasets. Also, we observed that the new proposed plain graph generative model, EDGE [368] can achieve good performance on small-scale datasets, but it relies on diffusion model for the generation process, which requires costly computational resources, especially high space complexity.

Attacker Experiment To further evaluate the privacy of our generated graphs, we perform an attack experiment on the DBLP dataset in addition to the EO-Rate metric. We consider the same attack scenario as Figure. 6.5, i.e., at time t, nodes sharing the same degree with the target victim are possible candidates. Mathematically, the successful attack probability (i.e., for accurately identifying the target) can be modeled as follows. At a certain timestamp t, the attack probability of node i is

$$p_i^{(t)} = \begin{cases} 0 & \mathbf{d}_{gen}^{(t)}[i] \neq \mathbf{d}_{ori}^{(t)}[i] \\ \frac{1}{\sum_{v \in \mathcal{G}^*} \mathbb{1}[\mathbf{d}_{gen}^{(t)}[v] == \mathbf{d}_{gen}^{(t)}[i]]} & \mathbf{d}_{gen}^{(t)}[i] = \mathbf{d}_{ori}^{(t)}[i] \end{cases} \tag{6.41}$$

where $\mathbb{1}$ is the indicator function, $\mathbf{d}_{gen}, \mathbf{d}_{ori}$ are node degree functions of the generated graph and original graph. Finally, we measure the model's privacy performance by the mean of $p_i^{(t)}$ over timestamps and all nodes. We compare the attack probabilities of graphs generated by PUG, HGEN, and a state-of-the-art differential-privacy-based graph generation method, DPGGAN [369], and get the following results; we also calculate the successful attack probability on the original graph for reference. The results are shown in Table. 6.6. We

Table 6.6: Graph Attack Results on DBLP

Graphs	Successful Attack Probability
Original	92.910
HGEN	8.662(0.132)
DPGGAN	8.261(0.134)
PUG(Ours)	**6.217(0.090)**

can observe that PUG-generated graphs give attackers the lowest confidence of a successful attack, and PUG is the most stable one with the smallest standard deviation.

Robustness of PUG training In this section, we discuss the convergence of the adversarial training part of PUG (lines 8 – 17 in Alg. 6.2). In practice, our generator $\mathbb{G}$ is updated every 2-5 iterations while privacy discriminator $\mathbb{D}_{prv}$ is updated every iteration. An illustration of the training process on the largest dataset (Taobao) is attached in Figure 6.7. In the training process, we first see a decrease of $\mathcal{L}_{\mathbb{D}_{prv}}$ to the lowest point, and then an increase of $\mathcal{L}_{\mathbb{D}_{prv}}$ as $\mathbb{G}$ gradually starting to outperform $\mathbb{D}_{prv}$.

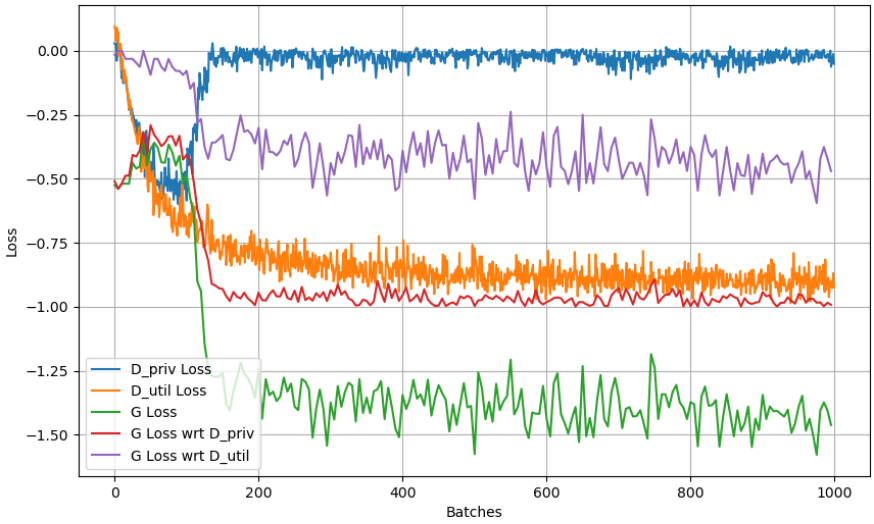


Figure 6.7: Training curve of generator/discriminators on Taobao

6.2.5 Summary

Graph data often contains sensitive information. Moreover, the complex graph structure introduces extra exposure risk for each node due to its more distinctive local topology than grid data (e.g., texts and images), which calls for well-defined protection techniques on graphs. To this end, we aim to design a privacy-guaranteed and utility-preserving graph generative model. Also, we take one step further by considering both temporal (i.e., structures and features are dependent on time) and heterogeneous (i.e., various node types and edge types) graphs. This makes privacy-preserving even more challenging because the abundant heterogeneous features and time dimension of the graph data are more prone to exposing sensitive information, although such information is critical for many real-world applications. We propose a **P**rivacy-guaranteed **U**tility-preserving Graph **G**enerative model, named PUG. More specifically, we propose a differential privacy algorithm to perturb the input temporal heterogeneous graph for protecting privacy, and then utilize both the perturbed graph and the original one in a generative adversarial setting for PUG to learn and generate privacy-guaranteed and utility-preserved graph data. We further propose 6 new metrics in the temporal setting to measure heterogeneous graph utility and privacy. Finally, based on temporal heterogeneous graph datasets with up to 1 million nodes and 20 million edges, the experiments show that PUG generates utilizable temporal heterogeneous graphs with privacy protection, compared with state-of-the-art baselines.

CHAPTER 7: CONCLUSION AND FUTURE WORK

This thesis proposed three general dynamics study directions, i.e., natural dynamics, artificial dynamics, and natural + artificial dynamics to empower graph AI and knowledge discovery from different aspects, e.g., node level, subgraph level, graph level, and combined level, to upgrade the corresponding performance in terms of effectiveness, efficiency, and trustworthiness.

Compared with the informative complex interactions and attributes from various aspects of nature and humans, the intelligent utilization and development of graph data is still in an obscure stage and waiting for urgent solutions. Thus, my future goal is to make graph data more applicable for various applications in different complex settings, with the theme *Unlocking the Power of Graph Intelligence*. To achieve this goal, I would continue to investigate the **natural dynamics** and **artificial dynamics** of graphs to (1) **disentangle and understand the structural complexity** and (2) **make necessary modifications** to further build next-generation graph AI systems.

Graph data integration is the base to support various graph AI developments by aggregating different kinds of graph data for extracting non-trivial, abundant, and useful knowledge. But its realization is very challenging, for example, unlike the independent and identically distributed (IID) data, the distribution of graph data from different domains can be quite different and not easy to be jointly leveraged, i.e., the size, the connection probability, and the attributes of different graphs place a barrier for distilling a consensus intelligence. Hence, under this direction (i.e., graph data integration), I design the follow-up of four sequential research agendas from establishing the theoretical foundation to extending to broad empirical applications step by step.

- **I. Theoretical Foundation: Graph Law, Distribution Convergence, and Dictionary Learning**. Graph data contains great potential, but its inherent complexity often gets discriminated against and exaggerated by different individual domains and hinders the development of ubiquitous advanced systems like large-scale foundation models. Hence, as the initial phase, I plan to find a statistical way to model graphs to break the inconsistency of different domains, such that different graphs can be described under the same language. The first sub-direction is *graph law*, which characterizes graphs with some parameters and their relations instead of the concrete matrices, i.e., the cross-domain graph representation complexities can be reduced to several shareable parameters, akin to Erdős–Rényi graphs (e.g., number of nodes and connection probability) but accounting for heterogeneity and temporality. In another

similar sub-direction, I aim to uncover the *graphon*, i.e., the general graph skeleton or prototype in the format of a converged matrix among various matrices, shared across diverse graph data. Last but not least, I also seek to create the *dictionary of graphs* analogous to NLP tokenization, which tokenized graphs into meaningful and shareable substructures [370]. Hence, we can break the non-IID and non-Euclidean constraints of graph data from different domains by modeling them in a unified manner, like texts and images, to enable a large-scale graph foundation model.

- **II. Practical Realization: Empowering and Exploiting Lightweight Graph Neural Architectures**. In addition to the first bullet point for cross-domain distribution, the volume of graph data also hinders the development of graph AI systems. Facing large-scale and complex graph data (e.g., long-range interaction, heterophily), the associated learning models often necessitate intricate neural architectures. Therefore, I would like to investigate how to restructure data to achieve a lightweight but effective graph learning model to avoid over-squashing, vanishing gradient, and over-smoothing problems. Also, hyperparameters of graph neural architectures are critical for the model performance but heavily rely on hand-crafted efforts. Therefore, I am also interested in designing auto ML for large-scale graph representation learning. To pave the way, I plan to continue my current study about integrating graph topological structure (and graph neural architecture) change with the reward of model performance by reinforcement learning.

- **III. Graph Foundation Model: Pre-training and Prompting**. The foundation models for computer vision and natural language have developed to a large extent recently. Unlike the grid data, the foundation model for graph data still needs a lot of effort for non-IID and non-Euclidean properties from different domains. After finishing the first two bullet points mentioned above, as a consequence, the establishment of large graph foundation models becomes possible. With the unified format of graph data representation and light but effective graph neural computational frameworks, I would like to first explore the possible large-scale graph foundation models (not limited to heterogeneous and temporal graph data) based on my experience on graph model pre-training across domains. Then, I aim to develop graph-driven in-context learning methods for improving the generalization and adaptation of trained large-scale graph foundation models through prompting, which dynamically modifies the graph research components from different domains.

- **IV. Multimodal Applications: Graphs with Large Language Models (LLMs)**.

In addition to solely developing foundation models on graphs, I also would like to leverage the representation ability of graphs to contribute to other application domains. To be specific, on the one hand, LLMs can produce a comprehensive understanding of knowledge for the public [371, 372, 373]. However for domain knowledge, knowledge graphs and taxonomy graphs usually have a better domain-specific reasoning and interpretation. Therefore, I aim to combine the conversational strengths of Large Language Models (LLMs) with domain-specific reasoning ability from graphs to create the next generation of conversable experts. On the other hand, for LLMs' inference ability, chain-of-thought is still a nascent in-context-learning manner for promoting the LLMs to achieve logical reasoning. Graph structures offer superior representation than linear chains, such that I would seek how to build *graph-of-thought* towards the next generation of multimodal LLMs.

REFERENCES

[1] C. Chen, Y. Wu, Q. Dai, H. Zhou, M. Xu, S. Yang, X. Han, and Y. Yu, "A survey on graph neural networks and graph transformers in computer vision: A task-oriented perspective," *CoRR*, vol. abs/2209.13232, 2022.

[2] L. Wu, Y. Chen, K. Shen, X. Guo, H. Gao, S. Li, J. Pei, and B. Long, "Graph neural networks for natural language processing: A survey," *Found. Trends Mach. Learn.*, vol. 16, no. 2, 2023.

[3] S. Wang, L. Hu, Y. Wang, X. He, Q. Z. Sheng, M. A. Orgun, L. Cao, F. Ricci, and P. S. Yu, "Graph learning based recommender systems: A review," in *IJCAI 2021*, 2021.

[4] T. Gaudelet, B. Day, A. R. Jamasb, J. Soman, C. Regep, G. Liu, J. B. R. Hayter, R. Vickers, C. Roberts, J. Tang, D. Roblin, T. L. Blundell, M. M. Bronstein, and J. P. Taylor-King, "Utilizing graph machine learning within drug discovery and development," *Briefings Bioinform.*, vol. 22, 2021.

[5] M. M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, and P. Vandergheynst, "Geometric deep learning: Going beyond euclidean data," *IEEE Signal Process. Mag.*, vol. 34, no. 4, pp. 18–42, 2017. [Online]. Available: https://doi.org/10.1109/MSP.2017.2693418

[6] M. M. Bronstein, J. Bruna, T. Cohen, and P. Velickovic, "Geometric deep learning: Grids, groups, graphs, geodesics, and gauges," *CoRR*, vol. abs/2104.13478, 2021. [Online]. Available: https://arxiv.org/abs/2104.13478

[7] D. Fu and J. He, "Natural and artificial dynamics in graphs: Concept, progress, and future," *Frontiers Big Data*, vol. 5, 2022.

[8] D. Fu, Z. Xu, H. Tong, and J. He, "Natural and artificial dynamics in gnns: A tutorial," in *Proceedings of the Sixteenth ACM International Conference on Web Search and Data Mining, WSDM 2023, Singapore, 27 February 2023 - 3 March 2023*. ACM, 2023. [Online]. Available: https://doi.org/10.1145/3539597.3572726 pp. 1252–1255.

[9] D. Fu, "Investigating natural and artificial dynamics in graph data mining and machine learning," in *Proceedings of the 32nd ACM International Conference on Information and Knowledge Management, CIKM 2023, Birmingham, United Kingdom, October 21-25, 2023*, I. Frommholz, F. Hopfgartner, M. Lee, M. Oakes, M. Lalmas, M. Zhang, and R. L. T. Santos, Eds. ACM, 2023. [Online]. Available: https://doi.org/10.1145/3583780.3616007 pp. 5173–5176.

[10] D. Fu, L. Fang, R. Maciejewski, V. I. Torvik, and J. He, "Meta-learned metrics over multi-evolution temporal graphs," in *KDD '22: The 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Washington, DC, USA, August 14 - 18, 2022*. ACM, 2022. [Online]. Available: https://doi.org/10.1145/3534678.3539313 pp. 367–377.

[11] D. Fu and J. He, "SDG: A simplified and dynamic graph neural network," in *SIGIR '21: The 44th International ACM SIGIR Conference on Research and Development in Information Retrieval, Virtual Event, Canada, July 11-15, 2021*. ACM, 2021. [Online]. Available: https://doi.org/10.1145/3404835.3463059 pp. 2273–2277.

[12] D. Fu, Y. Ban, H. Tong, R. Maciejewski, and J. He, "DISCO: comprehensive and explainable disinformation detection," in *Proceedings of the 31st ACM International Conference on Information & Knowledge Management, Atlanta, GA, USA, October 17-21, 2022*. ACM, 2022. [Online]. Available: https://doi.org/10.1145/3511808.3557202 pp. 4848–4852.

[13] D. Fu, D. Zhou, and J. He, "Local motif clustering on time-evolving graphs," in *KDD '20: The 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Virtual Event, CA, USA, August 23-27, 2020*, R. Gupta, Y. Liu, J. Tang, and B. A. Prakash, Eds. ACM, 2020. [Online]. Available: https://doi.org/10.1145/3394486.3403081 pp. 390–400.

[14] D. Fu, D. Zhou, R. Maciejewski, A. Croitoru, M. Boyd, and J. He, "Fairness-aware clique-preserving spectral clustering of temporal graphs," in *Proceedings of the ACM Web Conference 2023, WWW 2023, Austin, TX, USA, 30 April 2023 - 4 May 2023*. ACM, 2023. [Online]. Available: https://doi.org/10.1145/3543507.3583423 pp. 3755–3765.

[15] D. Fu and J. He, "DPPIN: A biological repository of dynamic protein-protein interaction network data," in *IEEE International Conference on Big Data, Big Data 2022, Osaka, Japan, December 17-20, 2022*. IEEE, 2022. [Online]. Available: https://doi.org/10.1109/BigData55660.2022.10020904 pp. 5269–5277.

[16] Z. Li, D. Fu, and J. He, "Everything evolves in personalized pagerank," in *Proceedings of the ACM Web Conference 2023, WWW 2023, Austin, TX, USA, 30 April 2023 - 4 May 2023*, Y. Ding, J. Tang, J. F. Sequeda, L. Aroyo, C. Castillo, and G. Houben, Eds. ACM, 2023. [Online]. Available: https://doi.org/10.1145/3543507.3583474 pp. 3342–3352.

[17] D. Fu, Z. Hua, Y. Xie, J. Fang, S. Zhang, K. Sancak, H. Wu, A. Malevich, J. He, and B. Long, "Vcr-graphormer: A mini-batch graph transformer via virtual connections," *CoRR*, vol. abs/2403.16030, 2024. [Online]. Available: https://doi.org/10.48550/arXiv.2403.16030

[18] D. Fu, Y. Zhu, H. Tong, K. Weldemariam, O. Bhardwaj, and J. He, "Generating fine-grained causality in climate time series data for forecasting and anomaly detection," in *ICML 2024 AI4Science Workshop*, 2024.

[19] D. Fu, J. He, H. Tong, and R. Maciejewski, "Privacy-preserving graph analytics: Secure generation and federated learning," *CoRR*, 2022.

[20] D. Fu, W. Bao, R. Maciejewski, H. Tong, and J. He, "Privacy-preserving graph machine learning from data to computation: A survey," *SIGKDD Explor.*, vol. 25, no. 1, pp. 54–72, 2023.

[21] W. Bao, J. Wu, and J. He, "BOBA: byzantine-robust federated learning with label skewness," in *International Conference on Artificial Intelligence and Statistics, 2-4 May 2024, Palau de Congressos, Valencia, Spain*, ser. Proceedings of Machine Learning Research, S. Dasgupta, S. Mandt, and Y. Li, Eds., vol. 238. PMLR, 2024. [Online]. Available: https://proceedings.mlr.press/v238/bao24a.html pp. 892–900.

[22] W. Bao, T. Wei, H. Wang, and J. He, "Adaptive test-time personalization for federated learning," in *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, Eds., 2023. [Online]. Available: http://papers.nips.cc/paper_files/paper/2023/hash/f555b62384279b98732204cb1a670a23-Abstract-Conference.html

[23] Y. Zhou, J. Wu, H. Wang, and J. He, "Adversarial robustness through bias variance decomposition: A new perspective for federated learning," in *Proceedings of the 31st ACM International Conference on Information & Knowledge Management, Atlanta, GA, USA, October 17-21, 2022*, M. A. Hasan and L. Xiong, Eds. ACM, 2022. [Online]. Available: https://doi.org/10.1145/3511808.3557232 pp. 2753–2762.

[24] J. Wu, W. Bao, E. A. Ainsworth, and J. He, "Personalized federated learning with parameter propagation," in *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, KDD 2023, Long Beach, CA, USA, August 6-10, 2023*, A. K. Singh, Y. Sun, L. Akoglu, D. Gunopulos, X. Yan, R. Kumar, F. Ozcan, and J. Ye, Eds. ACM, 2023. [Online]. Available: https://doi.org/10.1145/3580305.3599464 pp. 2594–2605.

[25] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *ICLR 2017*, 2017.

[26] J. Leskovec and C. Faloutsos, "Scalable modeling of real graphs using kronecker multiplication," in *ICML 2007*, 2007.

[27] Y. Wang, Y. Chang, Y. Liu, J. Leskovec, and P. Li, "Inductive representation learning in temporal networks via causal anonymous walks," in *ICLR 2021*, 2021.

[28] B. Bach, N. H. Riche, R. Fernandez, E. Giannisakis, B. Lee, and J.-D. Fekete, "Networkcube: bringing dynamic network visualizations to domain scientists," in *InfoVis 2015*, 2015.

[29] Y. Yang, K. Marriott, M. Butler, C. Goncu, and L. Holloway, "Tactile presentation of network data: Text, matrix or diagram?" in *CHI 2020*, 2020.

[30] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu, "A comprehensive survey on graph neural networks," *IEEE Trans. Neural Networks Learn. Syst.*, 2021.

[31] E. Min, R. Chen, Y. Bian, T. Xu, K. Zhao, W. Huang, P. Zhao, J. Huang, S. Ananiadou, and Y. Rong, "Transformer for graphs: An overview from architecture perspective," *CoRR*, vol. abs/2202.08455, 2022. [Online]. Available: https://arxiv.org/abs/2202.08455

[32] M. Chen, Z. Wei, Z. Huang, B. Ding, and Y. Li, "Simple and deep graph convolutional networks," in *ICML 2020*, 2020.

[33] J. Klicpera, A. Bojchevski, and S. Günnemann, "Predict then propagate: Graph neural networks meet personalized pagerank," in *ICLR 2019*, 2019.

[34] J. Shi and J. Malik, "Normalized cuts and image segmentation," *IEEE Trans. Pattern Anal. Mach. Intell.*, 2000.

[35] R. Andersen, F. R. K. Chung, and K. J. Lang, "Local graph partitioning using pagerank vectors," in *FOCS 2006*, 2006.

[36] W. Chiang, X. Liu, S. Si, Y. Li, S. Bengio, and C. Hsieh, "Cluster-gcn: An efficient algorithm for training deep and large graph convolutional networks," in *KDD 2019*, 2019.

[37] S. M. Kazemi, R. Goel, K. Jain, I. Kobyzev, A. Sethi, P. Forsyth, and P. Poupart, "Representation learning for dynamic graphs: A survey," *J. Mach. Learn. Res.*, vol. 21, pp. 70:1–70:73, 2020. [Online]. Available: http://jmlr.org/papers/v21/19-447.html

[38] C. C. Aggarwal and K. Subbian, "Evolutionary network analysis: A survey," *ACM Comput. Surv.*, 2014.

[39] J. Leskovec, L. Backstrom, R. Kumar, and A. Tomkins, "Microscopic evolution of social networks," in *KDD 2008*, 2008.

[40] D. Fu and J. He, "DPPIN: A biological repository of dynamic protein-protein interaction network data," *CoRR*, 2021.

[41] J. Leskovec, J. M. Kleinberg, and C. Faloutsos, "Graphs over time: densification laws, shrinking diameters and possible explanations," in *KDD 2005*, 2005.

[42] S. D. Kamvar, T. H. Haveliwala, C. D. Manning, and G. H. Golub, "Extrapolation methods for accelerating pagerank computations," in *WWW 2003*, 2003.

[43] L. Zheng, D. Fu, R. Maciejewski, and J. He, "Deeper-gxx: Deepening arbitrary gnns," *CoRR*, 2022.

[44] K. Ding, Z. Xu, H. Tong, and H. Liu, "Data augmentation for deep graph learning: A survey," *SIGKDD Explor.*, vol. 24, no. 2, pp. 61–77, 2022. [Online]. Available: https://doi.org/10.1145/3575637.3575646

[45] Z. Xu, B. Du, and H. Tong, "Graph sanitation with application to node classification," in *WWW '22: The ACM Web Conference 2022, Virtual Event, Lyon, France, April 25 - 29, 2022*, F. Laforest, R. Troncy, E. Simperl, D. Agarwal, A. Gionis, I. Herman, and L. Médini, Eds. ACM, 2022. [Online]. Available: https://doi.org/10.1145/3485447.3512180 pp. 1136–1147.

[46] Z. Xu, Y. Chen, M. Pan, H. Chen, M. Das, H. Yang, and H. Tong, "Kernel ridge regression-based graph dataset distillation," in *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, KDD 2023, Long Beach, CA, USA, August 6-10, 2023*, A. K. Singh, Y. Sun, L. Akoglu, D. Gunopulos, X. Yan, R. Kumar, F. Ozcan, and J. Ye, Eds. ACM, 2023. [Online]. Available: https://doi.org/10.1145/3580305.3599398 pp. 2850–2861.

[47] Z. Zeng, R. Qiu, Z. Xu, Z. Liu, Y. Yan, T. Wei, L. Ying, J. He, and H. Tong, "Graph mixup on approximate gromov–wasserstein geodesics," in *Forty-first International Conference on Machine Learning*.

[48] T. Wei, Y. You, T. Chen, Y. Shen, J. He, and Z. Wang, "Augmentations in hypergraph contrastive learning: Fabricated and generative," in *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, Eds., 2022. [Online]. Available: http://papers.nips.cc/paper_files/paper/2022/hash/0cd1eec0eeaf5ce1bf6d8875a7c1d095-Abstract-Conference.html

[49] X. Lin, J. Kang, W. Cong, and H. Tong, "Bemap: Balanced message passing for fair graph neural network," in *Learning on Graphs Conference, 27-30 November 2023, Virtual Event*, ser. Proceedings of Machine Learning Research, S. Villar and B. Chamberlain, Eds., vol. 231. PMLR, 2023. [Online]. Available: https://proceedings.mlr.press/v231/lin24a.html p. 37.

[50] R. Qiu, D. Wang, L. Ying, H. V. Poor, Y. Zhang, and H. Tong, "Reconstructing graph diffusion history from a single snapshot," in *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, KDD 2023, Long Beach, CA, USA, August 6-10, 2023*, A. K. Singh, Y. Sun, L. Akoglu, D. Gunopulos, X. Yan, R. Kumar, F. Ozcan, and J. Ye, Eds. ACM, 2023. [Online]. Available: https://doi.org/10.1145/3580305.3599488 pp. 1978–1988.

[51] R. Qiu, Z. Sun, and Y. Yang, "DIMES: A differentiable meta solver for combinatorial optimization problems," in *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, Eds., 2022. [Online]. Available: http://papers.nips.cc/paper_files/paper/2022/hash/a3a7387e49f4de290c23beea2dfcdc75-Abstract-Conference.html

[52] Y. Rong, W. Huang, T. Xu, and J. Huang, "Dropedge: Towards deep graph convolutional networks on node classification," in *ICLR 2020*, 2020.

[53] X. Liu, J. Cheng, Y. Song, and X. Jiang, "Boosting graph structure learning with dummy nodes," in *ICML 2022*, 2022.

[54] C. Yang, H. Wang, K. Zhang, L. Chen, and L. Sun, "Secure deep graph generation with link differential privacy," in *IJCAI 2021*, 2021.

[55] Z. Xu, R. Qiu, Y. Chen, H. Chen, X. Fan, M. Pan, Z. Zeng, M. Das, and H. Tong, "Discrete-state continuous-time diffusion for graph generation," *CoRR*, vol. abs/2405.11416, 2024. [Online]. Available: https://doi.org/10.48550/arXiv.2405.11416

[56] S. H. Khan, M. Naseer, M. Hayat, S. W. Zamir, F. S. Khan, and M. Shah, "Transformers in vision: A survey," *CoRR*, vol. abs/2101.01169, 2021. [Online]. Available: https://arxiv.org/abs/2101.01169

[57] T. Lin, Y. Wang, X. Liu, and X. Qiu, "A survey of transformers," *AI Open*, vol. 3, pp. 111–132, 2022. [Online]. Available: https://doi.org/10.1016/j.aiopen.2022.10.001

[58] V. P. Dwivedi and X. Bresson, "A generalization of transformer networks to graphs," *CoRR*, vol. abs/2012.09699, 2020. [Online]. Available: https://arxiv.org/abs/2012.09699

[59] J. Zhao, C. Li, Q. Wen, Y. Wang, Y. Liu, H. Sun, X. Xie, and Y. Ye, "Gophormer: Ego-graph transformer for node classification," *CoRR*, vol. abs/2110.13094, 2021. [Online]. Available: https://arxiv.org/abs/2110.13094

[60] W. Kuang, W. Zhen, Y. Li, Z. Wei, and B. Ding, "Coarformer: Transformer for large graph via graph coarsening," 2021.

[61] C. Ying, T. Cai, S. Luo, S. Zheng, G. Ke, D. He, Y. Shen, and T. Liu, "Do transformers really perform badly for graph representation?" in *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, M. Ranzato, A. Beygelzimer, Y. N. Dauphin, P. Liang, and J. W. Vaughan, Eds., 2021. [Online]. Available: https://proceedings.neurips.cc/paper/2021/hash/f1c1592588411002af340cbaedd6fc33-Abstract.html pp. 28 877–28 888.

[62] D. Kreuzer, D. Beaini, W. L. Hamilton, V. Létourneau, and P. Tossou, "Rethinking graph transformers with spectral attention," in *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, M. Ranzato, A. Beygelzimer, Y. N. Dauphin, P. Liang, and J. W. Vaughan, Eds., 2021. [Online]. Available: https://proceedings.neurips.cc/paper/2021/hash/b4fd1d2cb085390fbbadae65e07876a7-Abstract.html pp. 21 618–21 629.

[63] Z. Zhang, Q. Liu, Q. Hu, and C. Lee, "Hierarchical graph transformer with adaptive node sampling," in *NeurIPS*, 2022. [Online]. Available: http://papers.nips.cc/paper_files/paper/2022/hash/854a9ab0f323b841955e70ca383b27d1-Abstract-Conference.html

[64] L. Rampásek, M. Galkin, V. P. Dwivedi, A. T. Luu, G. Wolf, and D. Beaini, "Recipe for a general, powerful, scalable graph transformer," in *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, Eds., 2022. [Online]. Available: http://papers.nips.cc/paper_files/paper/2022/hash/5d4834a159f1547b267a05a4e2b7cf5e-Abstract-Conference.html

[65] J. Chen, K. Gao, G. Li, and K. He, "Nagphormer: A tokenized graph transformer for node classification in large graphs," in *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023. [Online]. Available: https://openreview.net/pdf?id=8KYeilT3Ow

[66] H. Shirzad, A. Velingker, B. Venkatachalam, D. J. Sutherland, and A. K. Sinop, "Exphormer: Sparse transformers for graphs," in *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, ser. Proceedings of Machine Learning Research, A. Krause, E. Brunskill, K. Cho, B. Engelhardt, S. Sabato, and J. Scarlett, Eds., vol. 202. PMLR, 2023. [Online]. Available: https://proceedings.mlr.press/v202/shirzad23a.html pp. 31 613–31 632.

[67] L. Müller, M. Galkin, C. Morris, and L. Rampásek, "Attending to graph transformers," *CoRR*, vol. abs/2302.04181, 2023. [Online]. Available: https://doi.org/10.48550/arXiv.2302.04181

[68] G. Mialon, D. Chen, M. Selosse, and J. Mairal, "Graphit: Encoding graph structure in transformers," *CoRR*, vol. abs/2106.05667, 2021. [Online]. Available: https://arxiv.org/abs/2106.05667

[69] Z. Wu, P. Jain, M. A. Wright, A. Mirhoseini, J. E. Gonzalez, and I. Stoica, "Representing long-range context for graph neural networks with global attention," in *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, M. Ranzato, A. Beygelzimer, Y. N. Dauphin, P. Liang, and J. W. Vaughan, Eds., 2021. [Online]. Available: https://proceedings.neurips.cc/paper/2021/hash/6e67691b60ed3e4a55935261314dd534-Abstract.html pp. 13 266–13 279.

[70] D. Chen, L. O'Bray, and K. M. Borgwardt, "Structure-aware transformer for graph representation learning," in *International Conference on Machine Learning, ICML 2022, 17-23 July 2022, Baltimore, Maryland, USA*, ser. Proceedings of Machine Learning Research, K. Chaudhuri, S. Jegelka, L. Song, C. Szepesvári, G. Niu, and S. Sabato, Eds., vol. 162. PMLR, 2022. [Online]. Available: https://proceedings.mlr.press/v162/chen22r.html pp. 3469–3489.

[71] J. Kim, D. Nguyen, S. Min, S. Cho, M. Lee, H. Lee, and S. Hong, "Pure transformers are powerful graph learners," in *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, Eds., 2022. [Online]. Available: http://papers.nips.cc/paper_files/paper/2022/hash/5d84236751fe6d25dc06db055a3180b0-Abstract-Conference.html

[72] D. Bo, C. Shi, L. Wang, and R. Liao, "Specformer: Spectral graph neural networks meet transformers," in *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023. [Online]. Available: https://openreview.net/pdf?id=0pdSt3oyJa1

[73] L. Ma, C. Lin, D. Lim, A. Romero-Soriano, P. K. Dokania, M. Coates, P. H. S. Torr, and S. Lim, "Graph inductive biases in transformers without message passing," in *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, ser. Proceedings of Machine Learning Research, A. Krause, E. Brunskill, K. Cho, B. Engelhardt, S. Sabato, and J. Scarlett, Eds., vol. 202. PMLR, 2023. [Online]. Available: https://proceedings.mlr.press/v202/ma23c.html pp. 23 321–23 337.

[74] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, I. Guyon, U. von Luxburg, S. Bengio, H. M. Wallach, R. Fergus, S. V. N. Vishwanathan, and R. Garnett, Eds., 2017. [Online]. Available: https://proceedings.neurips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html pp. 5998–6008.

[75] K. Xu, C. Li, Y. Tian, T. Sonobe, K. Kawarabayashi, and S. Jegelka, "Representation learning on graphs with jumping knowledge networks," in *ICML 2018*, 2018.

[76] Y. Qi, Y. Ban, and J. He, "Graph neural bandits," in *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, KDD 2023, Long Beach, CA, USA, August 6-10, 2023*, A. K. Singh, Y. Sun, L. Akoglu, D. Gunopulos, X. Yan, R. Kumar, F. Ozcan, and J. Ye, Eds. ACM, 2023. [Online]. Available: https://doi.org/10.1145/3580305.3599371 pp. 1920–1931.

[77] Y. Ban, Y. Yan, A. Banerjee, and J. He, "Ee-net: Exploitation-exploration neural networks in contextual bandits," in *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net, 2022. [Online]. Available: https://openreview.net/forum?id=X_ch3VrNSRg

[78] T. Wei, F. Feng, J. Chen, Z. Wu, J. Yi, and X. He, "Model-agnostic counterfactual reasoning for eliminating popularity bias in recommender system," in *KDD '21: The 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Virtual Event, Singapore, August 14-18, 2021*, F. Zhu, B. C. Ooi, and C. Miao, Eds. ACM, 2021. [Online]. Available: https://doi.org/10.1145/3447548.3467289 pp. 1791–1800.

[79] L. Liu, B. Du, H. Ji, C. Zhai, and H. Tong, "Neural-answering logical queries on knowledge graphs," in *KDD '21: The 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Virtual Event, Singapore, August 14-18, 2021*, F. Zhu, B. C. Ooi, and C. Miao, Eds. ACM, 2021. [Online]. Available: https://doi.org/10.1145/3447548.3467375 pp. 1087–1097.

[80] Y. Yan, L. Liu, Y. Ban, B. Jing, and H. Tong, "Dynamic knowledge graph alignment," in *AAAI 2021*, 2021.

[81] B. Jing, H. Tong, and Y. Zhu, "Network of tensor time series," in *WWW 2021*, 2021.

[82] D. Zhou, L. Zheng, D. Fu, J. Han, and J. He, "Mentorgnn: Deriving curriculum for pre-training gnns," in *Proceedings of the 31st ACM International Conference on Information & Knowledge Management, Atlanta, GA, USA, October 17-21, 2022*, M. A. Hasan and L. Xiong, Eds. ACM, 2022. [Online]. Available: https://doi.org/10.1145/3511808.3557393 pp. 2721–2731.

[83] K. Kong, J. Chen, J. Kirchenbauer, R. Ni, C. B. Bruss, and T. Goldstein, "GOAT: A global transformer on large-scale graphs," in *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, ser. Proceedings of Machine Learning Research, A. Krause, E. Brunskill, K. Cho, B. Engelhardt, S. Sabato, and J. Scarlett, Eds., vol. 202. PMLR, 2023. [Online]. Available: https://proceedings.mlr.press/v202/kong23a.html pp. 17 375–17 390.

[84] K. M. Choromanski, V. Likhosherstov, D. Dohan, X. Song, A. Gane, T. Sarlós, P. Hawkins, J. Q. Davis, A. Mohiuddin, L. Kaiser, D. B. Belanger, L. J. Colwell, and A. Weller, "Rethinking attention with performers," in *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021. [Online]. Available: https://openreview.net/forum?id=Ua6zuk0WRH

[85] W. Hu, M. Fey, M. Zitnik, Y. Dong, H. Ren, B. Liu, M. Catasta, and J. Leskovec, "Open graph benchmark: Datasets for machine learning on graphs," in *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, Eds., 2020. [Online]. Available: https://proceedings.neurips.cc/paper/2020/hash/fb60d411a5c5b72b2e7d3527cfc84fd0-Abstract.html

[86] Q. Wu, W. Zhao, Z. Li, D. P. Wipf, and J. Yan, "Nodeformer: A scalable graph structure learning transformer for node classification," in *NeurIPS*, 2022. [Online]. Available: http://papers.nips.cc/paper_files/paper/2022/hash/af790b7ae573771689438bbcfc5933fe-Abstract-Conference.html

[87] M. Yoon, W. Jin, and U. Kang, "Fast and accurate random walk with restart on dynamic graphs with guarantees," in *Proceedings of the 2018 World Wide Web Conference on World Wide Web, WWW 2018, Lyon, France, April 23-27, 2018*, P. Champin, F. Gandon, M. Lalmas, and P. G. Ipeirotis, Eds. ACM, 2018. [Online]. Available: https://doi.org/10.1145/3178876.3186107 pp. 409–418.

[88] F. Wu, A. H. S. Jr., T. Zhang, C. Fifty, T. Yu, and K. Q. Weinberger, "Simplifying graph convolutional networks," in *ICML 2019*, 2019.

[89] D. Bo, X. Wang, C. Shi, and H. Shen, "Beyond low-frequency information in graph convolutional networks," in *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021*. AAAI Press, 2021. [Online]. Available: https://doi.org/10.1609/aaai.v35i5.16514 pp. 3950–3957.

[90] J. Topping, F. D. Giovanni, B. P. Chamberlain, X. Dong, and M. M. Bronstein, "Understanding over-squashing and bottlenecks on graphs via curvature," in *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net, 2022. [Online]. Available: https://openreview.net/forum?id=7UmjRGzp-A

[91] D. A. Spielman, "Spectral graph theory and its applications," in *48th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2007), October 20-23, 2007, Providence, RI, USA, Proceedings*. IEEE Computer Society, 2007. [Online]. Available: https://doi.org/10.1109/FOCS.2007.66 pp. 29–38.

[92] U. Alon and E. Yahav, "On the bottleneck of graph neural networks and its practical implications," in *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021. [Online]. Available: https://openreview.net/forum?id=i80OPhOCVH2

[93] V. P. Dwivedi, L. Rampásek, M. Galkin, A. Parviz, G. Wolf, A. T. Luu, and D. Beaini, "Long range graph benchmark," in *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, Eds., 2022. [Online]. Available: http://papers.nips.cc/paper_files/paper/2022/hash/8c3c666820ea055a77726d66fc7d447f-Abstract-Datasets_and_Benchmarks.html

[94] Y. Yan, B. Jing, L. Liu, R. Wang, J. Li, T. F. Abdelzaher, and H. Tong, "Reconciling competing sampling strategies of network embedding," in *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, Eds., 2023. [Online]. Available: http://papers.nips.cc/paper_files/paper/2023/hash/15dc2344ea9bdc01ffb8bb2d692e4018-Abstract-Conference.html

[95] K. Karhadkar, P. K. Banerjee, and G. Montúfar, "Fosr: First-order spectral rewiring for addressing oversquashing in gnns," in *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023. [Online]. Available: https://openreview.net/pdf?id=3YjQfCLdrzz

[96] O. Platonov, D. Kuznedelev, M. Diskin, A. Babenko, and L. Prokhorenkova, "A critical look at the evaluation of gnns under heterophily: Are we really making progress?" in *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023. [Online]. Available: https://openreview.net/pdf?id=tJbbQfw-5wv

[97] Y. Yan, Y. Chen, H. Chen, M. Xu, M. Das, H. Yang, and H. Tong, "From trainable negative depth to edge heterophily in graphs," in *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, Eds., 2023. [Online]. Available: http://papers.nips.cc/paper_files/paper/2023/hash/de2d52c5cf2bea853ef39bb2e1535dde-Abstract-Conference.html

[98] Z. Xu, Y. Chen, Q. Zhou, Y. Wu, M. Pan, H. Yang, and H. Tong, "Node classification beyond homophily: Towards a general solution," in *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, KDD 2023, Long Beach, CA, USA, August 6-10, 2023*, A. K. Singh, Y. Sun, L. Akoglu, D. Gunopulos, X. Yan, R. Kumar, F. Ozcan, and J. Ye, Eds. ACM, 2023. [Online]. Available: https://doi.org/10.1145/3580305.3599446 pp. 2862–2873.

[99] J. Shi and J. Malik, "Normalized cuts and image segmentation," in *1997 Conference on Computer Vision and Pattern Recognition (CVPR '97), June 17-19, 1997, San Juan, Puerto Rico*. IEEE Computer Society, 1997. [Online]. Available: https://doi.org/10.1109/CVPR.1997.609407 pp. 731–737.

[100] G. Karypis and V. Kumar, "A fast and high quality multilevel scheme for partitioning irregular graphs," *SIAM J. Sci. Comput.*, vol. 20, no. 1, pp. 359–392, 1998. [Online]. Available: https://doi.org/10.1137/S1064827595287997

[101] J. Snell, K. Swersky, and R. S. Zemel, "Prototypical networks for few-shot learning," in *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, I. Guyon, U. von Luxburg, S. Bengio, H. M. Wallach, R. Fergus, S. V. N. Vishwanathan, and R. Garnett, Eds., 2017. [Online]. Available: https://proceedings.neurips.cc/paper/2017/hash/cb8da6767461f2812ae4290eac7cbc42-Abstract.html pp. 4077–4087.

[102] M. Yoon, J. Jung, and U. Kang, "TPA: fast, scalable, and accurate method for approximate random walk with restart on billion scale graphs," in *34th IEEE International Conference on Data Engineering, ICDE 2018, Paris, France, April 16-19, 2018*. IEEE Computer Society, 2018. [Online]. Available: https://doi.org/10.1109/ICDE.2018.00105 pp. 1132–1143.

[103] N. Ohsaka, T. Maehara, and K. Kawarabayashi, "Efficient pagerank tracking in evolving networks," in *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Sydney, NSW, Australia, August 10-13, 2015*, L. Cao, C. Zhang, T. Joachims, G. I. Webb, D. D. Margineantu, and G. Williams, Eds. ACM, 2015. [Online]. Available: https://doi.org/10.1145/2783258.2783297 pp. 875–884.

[104] W. Feng, Y. Dong, T. Huang, Z. Yin, X. Cheng, E. Kharlamov, and J. Tang, "GRAND+: scalable graph random neural networks," in *WWW '22: The ACM Web Conference 2022, Virtual Event, Lyon, France, April 25 - 29, 2022*, F. Laforest, R. Troncy, E. Simperl, D. Agarwal, A. Gionis, I. Herman, and L. Médini, Eds. ACM, 2022. [Online]. Available: https://doi.org/10.1145/3485447.3512044 pp. 3248–3258.

[105] D. Lim, F. Hohne, X. Li, S. L. Huang, V. Gupta, O. Bhalerao, and S. Lim, "Large scale learning on non-homophilous graphs: New benchmarks and strong simple methods," in *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, 2021. [Online]. Available: https://proceedings.neurips.cc/paper/2021/hash/ae816a80e4c1c56caa2eb4e1819cbb2f-Abstract.html pp. 20 887–20 902.

[106] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph attention networks," in *ICLR 2018*, 2018.

[107] E. Chien, J. Peng, P. Li, and O. Milenkovic, "Adaptive universal generalized pagerank graph neural network," in *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021. [Online]. Available: https://openreview.net/forum?id=n6jl7fLxrP

[108] H. Zeng, H. Zhou, A. Srivastava, R. Kannan, and V. K. Prasanna, "Graphsaint: Graph sampling based inductive learning method," in *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. [Online]. Available: https://openreview.net/forum?id=BJe8pkHFwS

[109] A. Bojchevski, J. Klicpera, B. Perozzi, A. Kapoor, M. Blais, B. Rózemberczki, M. Lukasik, and S. Günnemann, "Scaling graph neural networks with approximate pagerank," in *KDD 2020*, 2020.

[110] S. Abu-El-Haija, B. Perozzi, A. Kapoor, N. Alipourfard, K. Lerman, H. Harutyunyan, G. V. Steeg, and A. Galstyan, "Mixhop: Higher-order graph convolutional architectures via sparsified neighborhood mixing," in *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, ser. Proceedings of Machine Learning Research, K. Chaudhuri and R. Salakhutdinov, Eds., vol. 97. PMLR, 2019. [Online]. Available: http://proceedings.mlr.press/v97/abu-el-haija19a.html pp. 21–29.

[111] S. Suresh, V. Budde, J. Neville, P. Li, and J. Ma, "Breaking the limit of graph neural networks by improving the assortativity of graphs with local mixing patterns," in *KDD '21: The 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Virtual Event, Singapore, August 14-18, 2021*. ACM, 2021. [Online]. Available: https://doi.org/10.1145/3447548.3467373 pp. 1541–1551.

[112] Y. Yan, M. Hashemi, K. Swersky, Y. Yang, and D. Koutra, "Two sides of the same coin: Heterophily and oversmoothing in graph convolutional neural networks," in *IEEE International Conference on Data Mining, ICDM 2022, Orlando, FL, USA, November 28 - Dec. 1, 2022*. IEEE, 2022. [Online]. Available: https://doi.org/10.1109/ICDM54844.2022.00169 pp. 1287–1292.

[113] S. Luan, C. Hua, Q. Lu, J. Zhu, M. Zhao, S. Zhang, X. Chang, and D. Precup, "Is heterophily A real nightmare for graph neural networks to do node classification?" *CoRR*, vol. abs/2109.05641, 2021. [Online]. Available: https://arxiv.org/abs/2109.05641

[114] X. Li, R. Zhu, Y. Cheng, C. Shan, S. Luo, D. Li, and W. Qian, "Finding global homophily in graph neural networks when meeting heterophily," in *International Conference on Machine Learning, ICML 2022, 17-23 July 2022, Baltimore, Maryland, USA*, ser. Proceedings of Machine Learning Research, vol. 162. PMLR, 2022. [Online]. Available: https://proceedings.mlr.press/v162/li22ad.html pp. 13 242–13 256.

[115] J. Chen, T. Ma, and C. Xiao, "Fastgcn: Fast learning with graph convolutional networks via importance sampling," in *ICLR 2018*, 2018.

[116] X. He, K. Deng, X. Wang, Y. Li, Y. Zhang, and M. Wang, "Lightgcn: Simplifying and powering graph convolution network for recommendation," in *SIGIR 2020*, 2020.

[117] Y. Han, G. Huang, S. Song, L. Yang, H. Wang, and Y. Wang, "Dynamic neural networks: A survey," *CoRR*, 2021.

[118] Y. Ma, Z. Guo, Z. Ren, J. Tang, and D. Yin, "Streaming graph neural networks," in *SIGIR 2020*, 2020.

[119] D. Fu, Z. Xu, B. Li, H. Tong, and J. He, "A view-adversarial framework for multi-view network embedding," in *CIKM '20: The 29th ACM International Conference on Information and Knowledge Management, Virtual Event, Ireland, October 19-23, 2020*. ACM, 2020. [Online]. Available: https://doi.org/10.1145/3340531.3412127 pp. 2025–2028.

[120] R. Mihalcea and P. Tarau, "Textrank: Bringing order into text," in *EMNLP 2004*, 2004.

[121] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: pre-training of deep bidirectional transformers for language understanding," in *NAACL-HLT 2019*, 2019.

[122] P. K. Sarma, Y. Liang, and B. Sethares, "Domain adapted word embeddings for improved sentiment classification," in *ACL 2018*, 2018.

[123] P. K. Sarma, Y. Liang, and W. A. Sethares, "Shallow domain adaptive embeddings for sentiment analysis," in *EMNLP 2019*, 2019.

[124] K. Shu, L. Cui, S. Wang, D. Lee, and H. Liu, "defend: Explainable fake news detection," in *KDD 2019*, 2019.

[125] Y. Zhou, A. R. Nelakurthi, R. Maciejewski, W. Fan, and J. He, "Crowd teaching with imperfect labels," in *WWW 2020*, 2020.

[126] Y. Zhou, H. Wang, J. He, and H. Wang, "From intrinsic to counterfactual: On the explainability of contextualized recommender systems," *CoRR*, 2021.

[127] H. Ahmed, I. Traoré, and S. Saad, "Detection of online fake news using n-gram analysis and machine learning techniques," in *ISDDC 2017*, 2017.

[128] H. Ahmed, I. Traoré, and S. Saad, "Detecting opinion spams and fake news using text classification," *Secur. Priv.*, 2018.

[129] B. Perozzi, R. Al-Rfou, and S. Skiena, "Deepwalk: Online learning of social representations," in *KDD 2014*, 2014.

[130] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei, "LINE: large-scale information network embedding," in *WWW 2015*, 2015.

[131] A. Grover and J. Leskovec, "node2vec: Scalable feature learning for networks," in *KDD 2016*, 2016.

[132] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *NeurIPS 2014*, 2014.

[133] Q. Dai, Q. Li, J. Tang, and D. Wang, "Adversarial network embedding," in *AAAI 2018*, 2018.

[134] H. Wang, J. Wang, J. Wang, M. Zhao, W. Zhang, F. Zhang, X. Xie, and M. Guo, "Graphgan: Graph representation learning with generative adversarial nets," in *AAAI 2018*, 2018.

[135] M. Qu, J. Tang, J. Shang, X. Ren, M. Zhang, and J. Han, "An attention-based collaboration framework for multi-view network representation learning," in *CIKM 2017*, 2017.

[136] H. Zhang, L. Qiu, L. Yi, and Y. Song, "Scalable multiplex network embedding," in *IJCAI 2018*, 2018.

[137] Y. Sun, S. Wang, T. Hsieh, X. Tang, and V. G. Honavar, "MEGAN: A generative adversarial network for multi-view network embedding," in *IJCAI 2019*, 2019.

[138] Y. Cen, X. Zou, J. Zhang, H. Yang, J. Zhou, and J. Tang, "Representation learning for attributed multiplex heterogeneous network," in *KDD 2019*, 2019.

[139] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, 1997.

[140] P. Yang, Q. Tan, H. Tong, and J. He, "Task-adversarial co-generative nets," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD 2019, Anchorage, AK, USA, August 4-8, 2019*. ACM, 2019, pp. 1596–1604.

[141] J. Tang, J. Zhang, L. Yao, J. Li, L. Zhang, and Z. Su, "Arnetminer: extraction and mining of academic social networks," in *KDD 2008*, 2008.

[142] D. Greene and P. Cunningham, "Producing a unified graph representation from multiple social network views," in *WedSci 2013*, 2013.

[143] A. Kumar, S. S. Singh, K. Singh, and B. Biswas, "Link prediction techniques, applications, and performance: A survey," *Physica A: Statistical Mechanics and its Applications*, vol. 553, p. 124289, 2020.

[144] L. Lu and T. Zhou, "Link prediction in complex networks: A survey," *CoRR*, vol. abs/1010.0725, 2010. [Online]. Available: http://arxiv.org/abs/1010.0725

[145] V. Martínez, F. Berzal, and J. C. C. Talavera, "A survey of link prediction in complex networks," *ACM Comput. Surv.*, vol. 49, no. 4, pp. 69:1–69:33, 2017. [Online]. Available: https://doi.org/10.1145/3012704

[146] I. W. Taylor, R. Linding, D. Warde-Farley, Y. Liu, C. Pesquita, D. Faria, S. Bull, T. Pawson, Q. Morris, and J. L. Wrana, "Dynamic modularity in protein interaction networks predicts breast cancer outcome," *Nature biotechnology*, vol. 27, no. 2, pp. 199–204, 2009.

[147] G. D. Gupta, É. Coyaud, J. Gonçalves, B. A. Mojarad, Y. Liu, Q. Wu, L. Gheiratmand, D. Comartin, J. M. Tkach, S. W. Cheung et al., "A dynamic protein interaction landscape of the human centrosome-cilium interface," *Cell*, vol. 163, no. 6, pp. 1484–1499, 2015.

[148] Z. Huang, X. Li, and H. Chen, "Link prediction approach to collaborative filtering," in *Proceedings of the 5th ACM/IEEE-CS joint conference on Digital libraries*, 2005, pp. 141–142.

[149] N. N. Daud, S. H. Ab Hamid, M. Saadoon, F. Sahran, and N. B. Anuar, "Applications of link prediction in social networks: A review," *Journal of Network and Computer Applications*, vol. 166, p. 102716, 2020.

[150] L. Yu, L. Sun, B. Du, and W. Lv, "Towards better dynamic graph learning: New architecture and unified library," in *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, Eds., 2023. [Online]. Available: http://papers.nips.cc/paper_files/paper/2023/hash/d611019afba70d547bd595e8a4158f55-Abstract-Conference.html

[151] C. Diao and R. Loynd, "Relational attention: Generalizing transformers for graph-structured tasks," in *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023. [Online]. Available: https://openreview.net/pdf?id=cFuMmbWiN6

[152] W. Cong, S. Zhang, J. Kang, B. Yuan, H. Wu, X. Zhou, H. Tong, and M. Mahdavi, "Do we really need complicated model architectures for temporal networks?" in *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023. [Online]. Available: https://openreview.net/pdf?id=ayPPc0SyLv1

[153] D. Xu, C. Ruan, E. Körpeoglu, S. Kumar, and K. Achan, "Inductive representation learning on temporal graphs," in *ICLR*, 2020.

[154] V. P. Dwivedi, C. K. Joshi, A. T. Luu, T. Laurent, Y. Bengio, and X. Bresson, "Benchmarking graph neural networks," *J. Mach. Learn. Res.*, vol. 24, pp. 43:1–43:48, 2023. [Online]. Available: http://jmlr.org/papers/v24/22-0567.html

[155] D. Sundararajan, *The discrete Fourier transform: theory, algorithms and applications*. World Scientific, 2001.

[156] F. Poursafaei, S. Huang, K. Pelrine, and R. Rabbany, "Towards better evaluation for dynamic link prediction," in *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, Eds., 2022. [Online]. Available: http://papers.nips.cc/paper_files/paper/2022/hash/d49042a5d49818711c401d34172f9900-Abstract-Datasets_and_Benchmarks.html

[157] S. Kumar, X. Zhang, and J. Leskovec, "Predicting dynamic embedding trajectory in temporal interaction networks," in *KDD 2019*, 2019.

[158] R. Trivedi, M. Farajtabar, P. Biswal, and H. Zha, "Dyrep: Learning representations over dynamic graphs," in *ICLR 2019*, 2019.

[159] E. Rossi, B. Chamberlain, F. Frasca, D. Eynard, F. Monti, and M. M. Bronstein, "Temporal graph networks for deep learning on dynamic graphs," *CoRR*, vol. abs/2006.10637, 2020. [Online]. Available: https://arxiv.org/abs/2006.10637

[160] L. Wang, X. Chang, S. Li, Y. Chu, H. Li, W. Zhang, X. He, L. Song, J. Zhou, and H. Yang, "TCL: transformer-based dynamic graph modelling via contrastive learning," *CoRR*, vol. abs/2105.07944, 2021. [Online]. Available: https://arxiv.org/abs/2105.07944

[161] R. Andersen, F. R. K. Chung, and K. J. Lang, "Local graph partitioning using pagerank vectors," in *IEEE FOCS 2006*, 2006.

[162] D. A. Spielman and S. Teng, "A local clustering algorithm for massive graphs and its application to nearly linear time graph partitioning," *SIAM J. Comput.*, 2013.

[163] A. Paranjape, A. R. Benson, and J. Leskovec, "Motifs in temporal networks," in *ACM WSDM 2017*, 2017.

[164] R. Milo, S. Shen-Orr, S. Itzkovitz, N. Kashtan, D. Chklovskii, and U. Alon, "Network motifs: simple building blocks of complex networks," *Science*, 2002.

[165] M. Rosvall, A. Viamontes Esquivel, A. Lancichinetti, J. West, and R. Lambiotte, "Memory in network flows and its effects on spreading dynamics and community detection," *Nature communications*, 2014.

[166] C. J. Hoofnagle, "Identity theft: Making the known unknowns known," *Harv. JL & Tech.*, 2007.

[167] D. Zhou, S. Zhang, M. Y. Yildirim, S. Alcorn, H. Tong, H. Davulcu, and J. He, "A local algorithm for structure-preserving graph cut," in *ACM SIGKDD 2017*, 2017.

[168] K. Voevodski, S. Teng, and Y. Xia, "Spectral affinity in protein networks," *BMC Systems Biology*, 2009.

[169] H. Yin, A. R. Benson, J. Leskovec, and D. F. Gleich, "Local higher-order graph clustering," in *ACM SIGKDD 2017*, 2017.

[170] D. Zhou, S. Zhang, M. Y. Yildirim, S. Alcorn, H. Tong, H. Davulcu, and J. He, "A local algorithm for structure-preserving graph cut," in *KDD 2017*, 2017.

[171] D. F. Gleich, L. Lim, and Y. Yu, "Multilinear pagerank," *SIAM J. Matrix Anal. Appl.*, 2015.

[172] A. R. Benson, D. F. Gleich, and L. Lim, "The spacey random walk: A stochastic process for higher-order data," *SIAM Review*, 2017.

[173] H. Tong, S. Papadimitriou, P. S. Yu, and C. Faloutsos, "Proximity tracking on time-evolving bipartite graphs," in *SDM*, 2008.

[174] L. Page, S. Brin, R. Motwani, and T. Winograd, "The pagerank citation ranking: Bringing order to the web," Tech. Rep., 1999.

[175] W. Li and M. K. Ng, "On the limiting probability distribution of a transition probability tensor," *Linear and Multilinear Algebra*, 2014.

[176] J. Síma and S. E. Schaeffer, "On the np-completeness of some graph cluster measures," in *SOFSEM 2006*, 2006.

[177] A. R. Benson, D. F. Gleich, and J. Leskovec, "Tensor spectral clustering for partitioning higher-order network structures," in *SIAM SDM 2015*, 2015.

[178] T. Wu, A. R. Benson, and D. F. Gleich, "General tensor spectral co-clustering for higher-order data," in *NeurIPS 2016*, 2016.

[179] A. R. Benson, D. F. Gleich, and J. Leskovec, "Higher-order organization of complex networks," *Science*, 2016.

[180] N. Ohsaka, T. Maehara, and K. Kawarabayashi, "Efficient pagerank tracking in evolving networks," in *ACM SIGKDD*, 2015.

[181] H. Zhang, P. Lofgren, and A. Goel, "Approximate personalized pagerank on dynamic graphs," in *ACM SIGKDD 2016*, 2016.

[182] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische mathematik*, 1959.

[183] S. Kumar, F. Spezzano, V. S. Subrahmanian, and C. Faloutsos, "Edge weight prediction in weighted signed networks," in *IEEE ICDM 2016*, 2016.

[184] S. Kumar, B. Hooi, D. Makhija, M. Kumar, C. Faloutsos, and V. S. Subrahmanian, "REV2: fraudulent user prediction in rating platforms," in *ACM WSDM 2018*, 2018.

[185] R. A. Rossi and N. K. Ahmed, "The network data repository with interactive graph analytics and visualization," in *AAAI*, 2015. [Online]. Available: http://networkrepository.com

[186] H. Ning, W. Xu, Y. Chi, Y. Gong, and T. S. Huang, "Incremental spectral clustering by efficiently updating the eigen-system," *Pattern Recognition*, 2010.

[187] M. S. Granovetter, "The strength of weak ties," *American Journal of Sociology*, 1973.

[188] N. Durak, A. Pinar, T. G. Kolda, and C. Seshadhri, "Degree relations of triangles in real-world networks and graph models," in *ACM CIKM 2012*, 2012.

[189] A. Prat-Pérez, D. Dominguez-Sal, J. M. Brunat, and J.-L. Larriba-Pey, "Shaping communities out of triangles," in *ACM CIKM 2012*, 2012.

[190] M. Kleindessner, S. Samadi, P. Awasthi, and J. Morgenstern, "Guarantees for spectral clustering with fairness constraints," in *ICML*, 2019.

[191] X. Chen, B. Fain, L. Lyu, and K. Munagala, "Proportionally fair clustering," in *ICML*, 2019.

[192] L. Huang, S. H. Jiang, and N. K. Vishnoi, "Coresets for clustering with fairness constraints," in *NeurIPS*, 2019.

[193] S. K. Bera, D. Chakrabarty, N. Flores, and M. Negahbani, "Fair algorithms for clustering," in *NeurIPS*, 2019.

[194] P. Li, H. Zhao, and H. Liu, "Deep fair clustering for visual learning," in *CVPR*, 2020.

[195] J. Kang, J. He, R. Maciejewski, and H. Tong, "Inform: Individual fairness on graph mining," in *KDD*, 2020.

[196] J. Kang, T. Xie, X. Wu, R. Maciejewski, and H. Tong, "Infofair: Information-theoretic intersectional fairness," in *IEEE BigData*, 2022.

[197] H. Yoo, Z. Zeng, J. Kang, R. Qiu, D. Zhou, Z. Liu, F. Wang, C. Xu, E. Chan, and H. Tong, "Ensuring user-side fairness in dynamic recommender systems," in *Proceedings of the ACM on Web Conference 2024, WWW 2024, Singapore, May 13-17, 2024*, T. Chua, C. Ngo, R. Kumar, H. W. Lauw, and R. K. Lee, Eds. ACM, 2024. [Online]. Available: https://doi.org/10.1145/3589334.3645536 pp. 3667–3678.

[198] Z. Liu, R. Qiu, Z. Zeng, H. Yoo, D. Zhou, Z. Xu, Y. Zhu, K. Weldemariam, J. He, and H. Tong, "Class-imbalanced graph learning without class rebalancing," in *Forty-first International Conference on Machine Learning*.

[199] Z. Liu, W. Cao, Z. Gao, J. Bian, H. Chen, Y. Chang, and T. Liu, "Self-paced ensemble for highly imbalanced massive data classification," in *36th IEEE International Conference on Data Engineering, ICDE 2020, Dallas, TX, USA, April 20-24, 2020*. IEEE, 2020. [Online]. Available: https://doi.org/10.1109/ICDE48307.2020.00078 pp. 841–852.

[200] Z. Liu, R. Qiu, Z. Zeng, Y. Zhu, H. Hamann, and H. Tong, "Aim: Attributing, interpreting, mitigating data unfairness," *arXiv preprint arXiv:2406.08819*, 2024.

[201] Z. Liu, Z. Wei, E. Yu, Q. Huang, K. Guo, B. Yu, Z. Cai, H. Ye, W. Cao, J. Bian, P. Wei, J. Jiang, and Y. Chang, "IMBENS: ensemble class-imbalanced learning in python," *CoRR*, vol. abs/2111.12776, 2021. [Online]. Available: https://arxiv.org/abs/2111.12776

[202] L. Zheng, D. Zhou, H. Tong, J. Xu, Y. Zhu, and J. He, "Fairgen: Towards fair graph generation," *CoRR*, vol. abs/2303.17743, 2023. [Online]. Available: https://doi.org/10.48550/arXiv.2303.17743

[203] L. Zheng, Y. Zhu, and J. He, "Fairness-aware multi-view clustering," in *Proceedings of the 2023 SIAM International Conference on Data Mining, SDM 2023, Minneapolis-St. Paul Twin Cities, MN, USA, April 27-29, 2023*, S. Shekhar, Z. Zhou, Y. Chiang, and G. Stiglic, Eds. SIAM, 2023. [Online]. Available: https://doi.org/10.1137/1.9781611977653.ch96 pp. 856–864.

[204] I. Derényi, G. Palla, and T. Vicsek, "Clique percolation in random networks," *Physical Review Letters*, 2005.

[205] G. Palla, I. Derényi, I. Farkas, and T. Vicsek, "Uncovering the overlapping community structure of complex networks in nature and society," *nature*, 2005.

[206] G. Kossinets and D. J. Watts, "Empirical analysis of an evolving social network," *Science*, 2006.

[207] B. Adamcsek, G. Palla, I. J. Farkas, I. Derényi, and T. Vicsek, "CFinder: locating cliques and overlapping modules in biological networks," *Bioinformatics*, 2006.

[208] J. Wang, X. Peng, M. Li, and Y. Pan, "Construction and application of dynamic protein interaction network based on time course gene expression data," *Proteomics*, 2013.

[209] L. Li and H. Tong, "Computational approaches to the network science of teams," 2020.

[210] A. L. Traud, P. J. Mucha, and M. A. Porter, "Social structure of facebook networks," *CoRR*, 2011.

[211] D. Fu and J. He, "Natural and artificial dynamics in graphs: Concept, progress, and future," *Frontiers in Big Data*, 2022.

[212] T. Wu, A. R. Benson, and D. F. Gleich, "General tensor spectral co-clustering for higher-order data," in *NeurIPS*, 2016.

[213] A. R. Benson, D. F. Gleich, and J. Leskovec, "Higher-order organization of complex networks," *Science*, 2016.

[214] F. Chierichetti, R. Kumar, S. Lattanzi, and S. Vassilvitskii, "Fair clustering through fairlets," in *NeurIPS*, 2017.

[215] H. Lütkepohl, *Handbook of matrices*. Wiley Chichester, 1996, vol. 1.

[216] U. von Luxburg, "A tutorial on spectral clustering," *Stat. Comput.*, 2007.

[217] L. Huang, D. Yan, M. I. Jordan, and N. Taft, "Spectral clustering with perturbed data," in *NeurIPS*, 2008.

[218] Y. Chi, X. Song, D. Zhou, K. Hino, and B. L. Tseng, "Evolutionary spectral clustering by incorporating temporal smoothness," in *KDD*, 2007.

[219] N. Chiba and T. Nishizeki, "Arboricity and subgraph listing algorithms," *SIAM J. Comput.*, 1985.

[220] C. Chen and H. Tong, "Fast eigen-functions tracking on dynamic graphs," in *SDM*, 2015.

[221] C. Chen and H. Tong, "On the eigen-functions of dynamic graphs: Fast tracking and attribution algorithms," *Stat. Anal. Data Min.*, 2017.

[222] J. Fournet and A. Barrat, "Contact patterns among high school students," *PLoS ONE*, 2014.

[223] R. Mastrandrea, J. Fournet, and A. Barrat, "Contact patterns in a high school: A comparison between data collected using wearable sensors, contact diaries and friendship surveys," *PLOS ONE*, 2015.

[224] V. Gemmetto, A. Barrat, and C. Cattuto, "Mitigation of infectious disease at school: targeted class closure vs school closure." *BMC infectious diseases*, 2014.

[225] P. Vanhems, A. Barrat, C. Cattuto, J.-F. Pinton, N. Khanafer, C. Régis, B.-a. Kim, B. Comte, and N. Voirin, "Estimating potential infection transmission routes in hospital wards using wearable proximity sensors," *PLoS ONE*, 2013.

[226] B. Schölkopf, K. Tsuda, and J.-P. Vert, *Kernel Methods in Computational Biology*. MIT press, 2004.

[227] H. Dai, B. Dai, and L. Song, "Discriminative embeddings of latent variable models for structured data," in *ICML*, 2016.

[228] D. Duvenaud, D. Maclaurin, J. Aguilera-Iparraguirre, R. Gómez-Bombarelli, T. Hirzel, A. Aspuru-Guzik, and R. P. Adams, "Convolutional networks on graphs for learning molecular fingerprints," in *NeurIPS*, 2015.

[229] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, "Neural message passing for quantum chemistry," in *ICML 2017*, 2017.

[230] T. Derr, Y. Ma, W. Fan, X. Liu, C. C. Aggarwal, and J. Tang, "Epidemic graph convolutional network," in *WSDM*, 2020.

[231] L. Oettershagen, N. M. Kriege, C. Morris, and P. Mutzel, "Temporal graph kernels for classifying dissemination processes," in *SDM*, 2020.

[232] B. Shaw, B. Huang, and T. Jebara, "Learning a distance metric from a network," in *NeurIPS*, 2011.

[233] A. Tsitsulin, D. Mottin, P. Karras, A. M. Bronstein, and E. Müller, "Netlsd: Hearing the shape of a graph," in *KDD*, 2018.

[234] Y. Bai, H. Ding, S. Bian, T. Chen, Y. Sun, and W. Wang, "Simgnn: A neural network approach to fast graph similarity computation," in *WSDM*, 2019.

[235] Y. Li, C. Gu, T. Dullien, O. Vinyals, and P. Kohli, "Graph matching networks for learning the similarity of graph structured objects," in *ICML*, 2019.

[236] T. Yoshida, I. Takeuchi, and M. Karasuyama, "Learning interpretable metric between graphs: Convex formulation and computation with graph mining," in *KDD*, 2019.

[237] D. Zhou, L. Zheng, J. Xu, and J. He, "Misc-gan: A multi-scale generative model for graphs," *Frontiers Big Data*, 2019.

[238] D. Zhou, L. Zheng, J. Han, and J. He, "A data-driven graph generative model for temporal interaction networks," in *KDD 2020*, 2020.

[239] D. Fu, Y. Ban, H. Tong, R. Maciejewski, and J. He, "DISCO: comprehensive and explainable disinformation detection," *CoRR*, 2022.

[240] J. Goldberger, S. T. Roweis, G. E. Hinton, and R. Salakhutdinov, "Neighbourhood components analysis," in *NeurIPS*, 2004.

[241] R. Salakhutdinov and G. E. Hinton, "Learning a nonlinear embedding by preserving class neighbourhood structure," in *AISTATS*, 2007.

[242] O. Vinyals, C. Blundell, T. Lillicrap, K. Kavukcuoglu, and D. Wierstra, "Matching networks for one shot learning," in *NeurIPS*, 2016.

[243] B. N. Oreshkin, P. R. López, and A. Lacoste, "TADAM: task dependent adaptive metric for improved few-shot learning," in *NeurIPS*, 2018.

[244] K. R. Allen, E. Shelhamer, H. Shin, and J. B. Tenenbaum, "Infinite mixture prototypes for few-shot learning," in *ICML*, 2019.

[245] W. Hu, B. Liu, J. Gomes, M. Zitnik, P. Liang, V. S. Pande, and J. Leskovec, "Strategies for pre-training graph neural networks," in *ICLR 2020*, 2020.

[246] Z. Hu, Y. Dong, K. Wang, K. Chang, and Y. Sun, "GPT-GNN: generative pre-training of graph neural networks," in *KDD*, 2020.

[247] J. Qiu, Q. Chen, Y. Dong, J. Zhang, H. Yang, M. Ding, K. Wang, and J. Tang, "GCC: graph contrastive coding for graph neural network pre-training," in *KDD 2020*, 2020.

[248] M. Zitnik, R. Sosič, and J. Leskovec, "Prioritizing network communities," *Nature communications*, 2018.

[249] K. Zhou, H. Zha, and L. Song, "Learning triggering kernels for multi-dimensional hawkes processes," in *ICML*, 2013.

[250] N. Du, H. Dai, R. Trivedi, U. Upadhyay, M. Gomez-Rodriguez, and L. Song, "Recurrent marked temporal point processes: Embedding event history to vector," in *KDD*, 2016.

[251] D. Xu, C. Ruan, E. Körpeoglu, S. Kumar, and K. Achan, "Self-attention with functional time representation learning," in *NeurIPS*, 2019.

[252] Q. Zhang, A. Lipani, Ö. Kirnap, and E. Yilmaz, "Self-attentive hawkes process," in *ICML*, 2020.

[253] T. N. Kipf and M. Welling, "Variational graph auto-encoders," *CoRR*, vol. abs/1611.07308, 2016. [Online]. Available: http://arxiv.org/abs/1611.07308

[254] M. Zhang, Z. Cui, M. Neumann, and Y. Chen, "An end-to-end deep learning architecture for graph classification," in *AAAI 2018*, 2018.

[255] Z. Ying, J. You, C. Morris, X. Ren, W. L. Hamilton, and J. Leskovec, "Hierarchical graph representation learning with differentiable pooling," in *NeurIPS 2018*, 2018.

[256] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," in *ICLR*, 2015.

[257] J. Chauhan, D. Nathani, and M. Kaul, "Few-shot learning on graphs via super-classes based on graph spectral measures," in *ICLR*, 2020.

[258] N. Ma, J. Bu, J. Yang, Z. Zhang, C. Yao, Z. Yu, S. Zhou, and X. Yan, "Adaptive-step graph meta-learner for few-shot graph classification," in *CIKM*, 2020.

[259] C. Finn, P. Abbeel, and S. Levine, "Model-agnostic meta-learning for fast adaptation of deep networks," in *ICML*, 2017.

[260] C. Morris, N. M. Kriege, F. Bause, K. Kersting, P. Mutzel, and M. Neumann, "Tudataset: A collection of benchmark datasets for learning with graphs," *CoRR*, vol. abs/2007.08663, 2020. [Online]. Available: https://arxiv.org/abs/2007.08663

[261] K. M. Borgwardt and H. Kriegel, "Shortest-path kernels on graphs," in *Proceedings of the 5th IEEE International Conference on Data Mining (ICDM 2005), 27-30 November 2005, Houston, Texas, USA*. IEEE Computer Society, 2005. [Online]. Available: https://doi.org/10.1109/ICDM.2005.132 pp. 74–81.

[262] S. Hido and H. Kashima, "A linear-time graph kernel," in *ICDM*, 2009, pp. 179–188.

[263] N. M. Kriege, P. Giscard, and R. C. Wilson, "On valid optimal assignment kernels and applications to graph classification," in *NeurIPS*, 2016.

[264] H. Chen and H. Koga, "Gl2vec: Graph embedding enriched by line graphs with edge features," in *Neural Information Processing - 26th International Conference, ICONIP 2019, Sydney, NSW, Australia, December 12-15, 2019, Proceedings, Part III*, ser. Lecture Notes in Computer Science, T. Gedeon, K. W. Wong, and M. Lee, Eds., vol. 11955. Springer, 2019. [Online]. Available: https://doi.org/10.1007/978-3-030-36718-3_1 pp. 3–14.

[265] A. Narayanan, M. Chandramohan, R. Venkatesan, L. Chen, Y. Liu, and S. Jaiswal, "graph2vec: Learning distributed representations of graphs," *CoRR*, 2017.

[266] M. Beladev, L. Rokach, G. Katz, I. Guy, and K. Radinsky, "tdgraphembed: Temporal dynamic graph-level embedding," in *CIKM*, 2020.

[267] T. Cao, M. T. Law, and S. Fidler, "A theoretical analysis of the number of shots in few-shot learning," in *ICLR*, 2020.

[268] W. Ju, S. Yi, Y. Wang, Z. Xiao, Z. Mao, H. Li, Y. Gu, Y. Qin, N. Yin, S. Wang, X. Liu, X. Luo, P. S. Yu, and M. Zhang, "A survey of graph neural networks in real world: Imbalance, noise, privacy and OOD challenges," *CoRR*, vol. abs/2403.04468, 2024. [Online]. Available: https://doi.org/10.48550/arXiv.2403.04468

[269] J. Wu, J. He, and E. A. Ainsworth, "Non-iid transfer learning on graphs," in *Thirty-Seventh AAAI Conference on Artificial Intelligence, AAAI 2023, Thirty-Fifth Conference on Innovative Applications of Artificial Intelligence, IAAI 2023, Thirteenth Symposium on Educational Advances in Artificial Intelligence, EAAI 2023, Washington, DC, USA, February 7-14, 2023*, B. Williams, Y. Chen, and J. Neville, Eds. AAAI Press, 2023. [Online]. Available: https://doi.org/10.1609/aaai.v37i9.26231 pp. 10 342–10 350.

[270] H. Wang, Y. Mao, Y. Yan, Y. Yang, J. Sun, K. Choi, B. Veeramani, A. Hu, E. Bowen, T. Cody et al., "Evolunet: Advancing dynamic non-iid transfer learning on graphs," in *Forty-first International Conference on Machine Learning*.

[271] D. Zhou and J. He, "Rare category analysis for complex data: A review," *ACM Comput. Surv.*, vol. 56, no. 5, pp. 123:1–123:35, 2024. [Online]. Available: https://doi.org/10.1145/3626520

[272] N. M. Kriege, F. D. Johansson, and C. Morris, "A survey on graph kernels," *Appl. Netw. Sci.*, vol. 5, no. 1, p. 6, 2020. [Online]. Available: https://doi.org/10.1007/s41109-019-0195-3

[273] G. Nikolentzos, G. Siglidis, and M. Vazirgiannis, "Graph kernels: A survey," *J. Artif. Intell. Res.*, vol. 72, pp. 943–1027, 2021. [Online]. Available: https://doi.org/10.1613/jair.1.13225

[274] S. S. Du, K. Hou, R. Salakhutdinov, B. Póczos, R. Wang, and K. Xu, "Graph neural tangent kernel: Fusing graph neural networks with graph kernels," in *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, H. M. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. B. Fox, and R. Garnett, Eds., 2019. [Online]. Available: https://proceedings.neurips.cc/paper/2019/hash/663fd3c5144fd10bd5ca6611a9a5b92d-Abstract.html pp. 5724–5734.

[275] S. Krishnagopal and L. Ruiz, "Graph neural tangent kernel: Convergence on large graphs," in *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, ser. Proceedings of Machine Learning Research, A. Krause, E. Brunskill, K. Cho, B. Engelhardt, S. Sabato, and J. Scarlett, Eds., vol. 202. PMLR, 2023. [Online]. Available: https://proceedings.mlr.press/v202/krishnagopal23a.html pp. 17 827–17 841.

[276] L. Ruiz, L. F. O. Chamon, and A. Ribeiro, "Graphon neural networks and the transferability of graph neural networks," in *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, Eds., 2020. [Online]. Available: https://proceedings.neurips.cc/paper/2020/hash/12bcd658ef0a540cabc36cdf2b1046fd-Abstract.html

[277] S. Arora, S. S. Du, W. Hu, Z. Li, R. Salakhutdinov, and R. Wang, "On exact computation with an infinitely wide neural net," in *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, H. M. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. B. Fox, and R. Garnett, Eds., 2019. [Online]. Available: https://proceedings.neurips.cc/paper/2019/hash/dbc4d84bfcfe2284ba11beffb853a8c4-Abstract.html pp. 8139–8148.

[278] S. Arora, S. S. Du, W. Hu, Z. Li, and R. Wang, "Fine-grained analysis of optimization and generalization for overparameterized two-layer neural networks," in *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, ser. Proceedings of Machine Learning Research, K. Chaudhuri and R. Salakhutdinov, Eds., vol. 97. PMLR, 2019. [Online]. Available: http://proceedings.mlr.press/v97/arora19a.html pp. 322–332.

[279] S. S. Du, J. D. Lee, H. Li, L. Wang, and X. Zhai, "Gradient descent finds global minima of deep neural networks," in *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, ser. Proceedings of Machine Learning Research, K. Chaudhuri and R. Salakhutdinov, Eds., vol. 97. PMLR, 2019. [Online]. Available: http://proceedings.mlr.press/v97/du19c.html pp. 1675–1685.

[280] S. S. Du, X. Zhai, B. Póczos, and A. Singh, "Gradient descent provably optimizes over-parameterized neural networks," in *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. [Online]. Available: https://openreview.net/forum?id=S1eK3i09YQ

[281] A. Jacot, C. Hongler, and F. Gabriel, "Neural tangent kernel: Convergence and generalization in neural networks," in *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, S. Bengio, H. M. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds., 2018. [Online]. Available: https://proceedings.neurips.cc/paper/2018/hash/5a4be1fa34e62bb8a6ec6b91d2462f5a-Abstract.html pp. 8580–8589.

[282] N. Shervashidze, P. Schweitzer, E. J. van Leeuwen, K. Mehlhorn, and K. M. Borgwardt, "Weisfeiler-lehman graph kernels," *J. Mach. Learn. Res.*, 2011.

[283] S. V. N. Vishwanathan, K. M. Borgwardt, I. R. Kondor, and N. N. Schraudolph, "Graph kernels," *CoRR*, vol. abs/0807.0093, 2008. [Online]. Available: http://arxiv.org/abs/0807.0093

[284] S. Zhang and H. Tong, "FINAL: fast attributed network alignment," in *KDD 2016*, 2016.

[285] Q. Zhou, L. Li, X. Wu, N. Cao, L. Ying, and H. Tong, "Attent: Active attributed network alignment," in *WWW 2021*, 2021.

[286] Y. Yan, S. Zhang, and H. Tong, "BRIGHT: A bridging algorithm for network alignment," in *WWW 2021*, 2021.

[287] J. Gao, X. Huang, and J. Li, "Unsupervised graph alignment with wasserstein distance discriminator," in *KDD 2021*, 2021.

[288] Z. Zeng, S. Zhang, Y. Xia, and H. Tong, "PARROT: position-aware regularized optimal transport for network alignment," in *Proceedings of the ACM Web Conference 2023, WWW 2023, Austin, TX, USA, 30 April 2023 - 4 May 2023*, Y. Ding, J. Tang, J. F. Sequeda, L. Aroyo, C. Castillo, and G. Houben, Eds. ACM, 2023. [Online]. Available: https://doi.org/10.1145/3543507.3583357 pp. 372–382.

[289] Z. Zeng, B. Du, S. Zhang, Y. Xia, Z. Liu, and H. Tong, "Hierarchical multi-marginal optimal transport for network alignment," in *Thirty-Eighth AAAI Conference on Artificial Intelligence, AAAI 2024, Thirty-Sixth Conference on Innovative Applications of Artificial Intelligence, IAAI 2024, Fourteenth Symposium on Educational Advances in Artificial Intelligence, EAAI 2014, February 20-27, 2024, Vancouver, Canada*, M. J. Wooldridge, J. G. Dy, and S. Natarajan, Eds. AAAI Press, 2024. [Online]. Available: https://doi.org/10.1609/aaai.v38i15.29605 pp. 16 660–16 668.

[290] Y. Qi, Y. Ban, and J. He, "Neural bandit with arm group graph," in *KDD '22: The 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Washington, DC, USA, August 14 - 18, 2022*, A. Zhang and H. Rangwala, Eds. ACM, 2022. [Online]. Available: https://doi.org/10.1145/3534678.3539312 pp. 1379–1389.

[291] L. Zheng, B. Jing, Z. Li, H. Tong, and J. He, "Heterogeneous contrastive learning for foundation models and beyond," *CoRR*, vol. abs/2404.00225, 2024. [Online]. Available: https://doi.org/10.48550/arXiv.2404.00225

[292] Z. Li, Y. Ao, and J. He, "Sphere: Expressive and interpretable knowledge graph embedding for set retrieval," *CoRR*, vol. abs/2404.19130, 2024. [Online]. Available: https://doi.org/10.48550/arXiv.2404.19130

[293] G. Kollias, S. Mohammadi, and A. Grama, "Network similarity decomposition (NSD): A fast and scalable approach to network alignment," *IEEE Trans. Knowl. Data Eng.*, 2012.

[294] S. Park, W. Lee, B. Choe, and S. Lee, "A survey on personalized pagerank computation algorithms," *IEEE Access*, 2019.

[295] D. Mo and S. Luo, "Agenda: Robust personalized pageranks in evolving graphs," in *CIKM 2021*, 2021.

[296] H. Tong, C. Faloutsos, and J. Pan, "Fast random walk with restart and its applications," in *ICDM 2006*, 2006.

[297] A. Paranjape, A. R. Benson, and J. Leskovec, "Motifs in temporal networks," in *WSDM 2017*, 2017.

[298] J. Kunegis, "KONECT: the koblenz network collection," in *WWW 2013, Companion Volume*, 2013.

[299] S. Kumar, F. Spezzano, V. S. Subrahmanian, and C. Faloutsos, "Edge weight prediction in weighted signed networks," in *ICDM 2016*, 2016.

[300] D. Frankowski, S. K. Lam, S. Sen, F. M. Harper, S. Yilek, M. Cassano, and J. Riedl, "Recommenders everywhere: : the wikilens community-maintained recommender system," in *WIKISYM 2007*, 2007.

[301] B. Du, S. Zhang, N. Cao, and H. Tong, "FIRST: fast interactive attributed subgraph matching," in *KDD 2017*, 2017.

[302] Y. Li, R. Yu, C. Shahabi, and Y. Liu, "Diffusion convolutional recurrent neural network: Data-driven traffic forecasting," in *ICLR 2018*, 2018.

[303] Z. Wu, S. Pan, G. Long, J. Jiang, X. Chang, and C. Zhang, "Connecting the dots: Multivariate time series forecasting with graph neural networks," in *KDD '20: The 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Virtual Event, CA, USA, August 23-27, 2020*, R. Gupta, Y. Liu, J. Tang, and B. A. Prakash, Eds. ACM, 2020. [Online]. Available: https://doi.org/10.1145/3394486.3403118 pp. 753–763.

[304] H. Zhao, Y. Wang, J. Duan, C. Huang, D. Cao, Y. Tong, B. Xu, J. Bai, J. Tong, and Q. Zhang, "Multivariate time-series anomaly detection via graph attention network," in *20th IEEE International Conference on Data Mining, ICDM 2020, Sorrento, Italy, November 17-20, 2020*, C. Plant, H. Wang, A. Cuzzocrea, C. Zaniolo, and X. Wu, Eds. IEEE, 2020. [Online]. Available: https://doi.org/10.1109/ICDM50108.2020.00093 pp. 841–850.

[305] D. Cao, Y. Wang, J. Duan, C. Zhang, X. Zhu, C. Huang, Y. Tong, B. Xu, J. Bai, J. Tong, and Q. Zhang, "Spectral temporal graph neural network for multivariate time-series forecasting," in *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, Eds., 2020. [Online]. Available: https://proceedings.neurips.cc/paper/2020/hash/cdf6581cb7aca4b7e19ef136c6e601a5-Abstract.html

[306] C. Shang, J. Chen, and J. Bi, "Discrete graph structure learning for forecasting multiple time series," in *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021. [Online]. Available: https://openreview.net/forum?id=WEHSlH5mOk

[307] A. Deng and B. Hooi, "Graph neural network-based anomaly detection in multivariate time series," in *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021*. AAAI Press, 2021. [Online]. Available: https://doi.org/10.1609/aaai.v35i5.16523 pp. 4027–4035.

[308] R. Marcinkevics and J. E. Vogt, "Interpretable models for granger causality using self-explaining neural networks," in *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021. [Online]. Available: https://openreview.net/forum?id=DEa4JdMWRHp

[309] T. Geffner, J. Antorán, A. Foster, W. Gong, C. Ma, E. Kiciman, A. Sharma, A. Lamb, M. Kukla, N. Pawlowski, M. Allamanis, and C. Zhang, "Deep end-to-end causal inference," *CoRR*, vol. abs/2202.02195, 2022. [Online]. Available: https://arxiv.org/abs/2202.02195

[310] A. Tank, I. Covert, N. J. Foti, A. Shojaie, and E. B. Fox, "Neural granger causality," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 44, no. 8, pp. 4267–4279, 2022. [Online]. Available: https://doi.org/10.1109/TPAMI.2021.3065601

[311] G. Spadon, S. Hong, B. Brandoli, S. Matwin, J. F. R. Jr., and J. Sun, "Pay attention to evolution: Time series forecasting with deep graph-evolution learning," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 44, no. 9, pp. 5368–5384, 2022. [Online]. Available: https://doi.org/10.1109/TPAMI.2021.3076155

[312] W. Gong, J. Jennings, C. Zhang, and N. Pawlowski, "Rhino: Deep causal temporal relationship learning with history-dependent noise," in *The Eleventh International Conference on Learning Representations*, 2023. [Online]. Available: https://openreview.net/forum?id=i_1rbq8yFWC

[313] D. Wang, Y. Yan, R. Qiu, Y. Zhu, K. Guan, A. Margenot, and H. Tong, "Networked time series imputation via position-aware graph enhanced variational autoencoders," in *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, KDD 2023, Long Beach, CA, USA, August 6-10, 2023*, A. K. Singh, Y. Sun, L. Akoglu, D. Gunopulos, X. Yan, R. Kumar, F. Ozcan, and J. Ye, Eds. ACM, 2023. [Online]. Available: https://doi.org/10.1145/3580305.3599444 pp. 2256–2268.

[314] M. Kofinas, E. J. Bekkers, N. S. Nagaraja, and E. Gavves, "Latent field discovery in interacting dynamical systems with neural fields," *CoRR*, vol. abs/2310.20679, 2023. [Online]. Available: https://doi.org/10.48550/arXiv.2310.20679

[315] M. Kofinas, N. S. Nagaraja, and E. Gavves, "Roto-translated local coordinate frames for interacting dynamical systems," in *NeurIPS 2021*, 2021.

[316] C. W. Granger, "Investigating causal relations by econometric models and cross-spectral methods," *Econometrica: journal of the Econometric Society*, pp. 424–438, 1969.

[317] A. Arnold, Y. Liu, and N. Abe, "Temporal causal modeling with graphical granger methods," in *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Jose, California, USA, August 12-15, 2007*, P. Berkhin, R. Caruana, and X. Wu, Eds. ACM, 2007. [Online]. Available: https://doi.org/10.1145/1281192.1281203 pp. 66–75.

[318] J. Runge, P. Nowack, M. Kretschmer, S. Flaxman, and D. Sejdinovic, "Detecting and quantifying causal associations in large nonlinear time series datasets," *Science advances*, vol. 5, no. 11, p. eaau4996, 2019.

[319] M. Nauta, D. Bucur, and C. Seifert, "Causal discovery with attention-based convolutional neural networks," *Mach. Learn. Knowl. Extr.*, vol. 1, no. 1, pp. 312–340, 2019. [Online]. Available: https://doi.org/10.3390/make1010019

[320] C. Xu, H. Huang, and S. Yoo, "Scalable causal graph learning through a deep neural network," in *Proceedings of the 28th ACM International Conference on Information and Knowledge Management, CIKM 2019, Beijing, China, November 3-7, 2019*, W. Zhu, D. Tao, X. Cheng, P. Cui, E. A. Rundensteiner, D. Carmel, Q. He, and J. X. Yu, Eds. ACM, 2019. [Online]. Available: https://doi.org/10.1145/3357384.3357864 pp. 1853–1862.

[321] S. Khanna and V. Y. F. Tan, "Economy statistical recurrent units for inferring nonlinear granger causality," in *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. [Online]. Available: https://openreview.net/forum?id=SyxV9ANFDH

[322] H. Huang, C. Xu, S. Yoo, W. Yan, T. Wang, and F. Xue, "Imbalanced time series classification for flight data analyzing with nonlinear granger causality learning," in *CIKM '20: The 29th ACM International Conference on Information and Knowledge Management, Virtual Event, Ireland, October 19-23, 2020*, M. d'Aquin, S. Dietze, C. Hauff, E. Curry, and P. Cudré-Mauroux, Eds. ACM, 2020. [Online]. Available: https://doi.org/10.1145/3340531.3412710 pp. 2533–2540.

[323] R. Pamfil, N. Sriwattanaworachai, S. Desai, P. Pilgerstorfer, K. Georgatzis, P. Beaumont, and B. Aragam, "DYNOTEARS: structure learning from time-series data," in *The 23rd International Conference on Artificial Intelligence and Statistics, AISTATS 2020, 26-28 August 2020, Online [Palermo, Sicily, Italy]*, ser. Proceedings of Machine Learning Research, S. Chiappa and R. Calandra, Eds., vol. 108. PMLR, 2020. [Online]. Available: http://proceedings.mlr.press/v108/pamfil20a.html pp. 1595–1605.

[324] A. Moneta, D. Entner, P. O. Hoyer, and A. Coad, "Causal inference by independent component analysis: Theory and applications," *Oxford Bulletin of Economics and Statistics*, vol. 75, no. 5, pp. 705–730, 2013.

[325] B. Wild, M. Eichler, H.-C. Friederich, M. Hartmann, S. Zipfel, and W. Herzog, "A graphical vector autoregressive modelling approach to the analysis of electronic diary data," *BMC medical research methodology*, vol. 10, pp. 1–13, 2010.

[326] R. Dahlhaus and M. Eichler, "Causality and graphical models in time series analysis," *Oxford Statistical Science Series*, pp. 115–137, 2003.

[327] D. Malinsky and P. Spirtes, "Causal structure learning from multivariate time series in settings with unmeasured confounding," in *Proceedings of 2018 ACM SIGKDD Workshop on Causal Discovery, CD@KDD 2018, London, UK, 20 August 2018*, ser. Proceedings of Machine Learning Research, T. D. Le, K. Zhang, E. Kiciman, A. Hyvärinen, and L. Liu, Eds., vol. 92. PMLR, 2018. [Online]. Available: http://proceedings.mlr.press/v92/malinsky18a.html pp. 23–47.

[328] C. K. Assaad, E. Devijver, and É. Gaussier, "Discovery of extended summary graphs in time series," in *Uncertainty in Artificial Intelligence, Proceedings of the Thirty-Eighth Conference on Uncertainty in Artificial Intelligence, UAI 2022, 1-5 August 2022, Eindhoven, The Netherlands*, ser. Proceedings of Machine Learning Research, J. Cussens and K. Zhang, Eds., vol. 180. PMLR, 2022. [Online]. Available: https://proceedings.mlr.press/v180/assaad22a.html pp. 96–106.

[329] E. N. Lorenz, "Predictability: A problem partly solved," in *Proc. Seminar on predictability*, vol. 1, no. 1. Reading, 1996.

[330] H. Hersbach, B. Bell, P. Berrisford, G. Biavati, A. Horányi, J. Muñoz Sabater, J. Nicolas, C. Peubey, R. Radu, I. Rozum et al., "Era5 hourly data on single levels from 1979 to present," *Copernicus climate change service (c3s) climate data store (cds)*, vol. 10, no. 10.24381, 2018.

[331] Y. Su, Y. Zhao, C. Niu, R. Liu, W. Sun, and D. Pei, "Robust anomaly detection for multivariate time series through stochastic recurrent neural network," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD 2019, Anchorage, AK, USA, August 4-8, 2019*, A. Teredesai, V. Kumar, Y. Li, R. Rosales, E. Terzi, and G. Karypis, Eds. ACM, 2019. [Online]. Available: https://doi.org/10.1145/3292500.3330672 pp. 2828–2837.

[332] X. Zheng, B. Aragam, P. Ravikumar, and E. P. Xing, "Dags with NO TEARS: continuous optimization for structure learning," in *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, S. Bengio, H. M. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds., 2018. [Online]. Available: https://proceedings.neurips.cc/paper/2018/hash/e347c51419ffb23ca3fd5050202f9c3d-Abstract.html pp. 9492–9503.

[333] Y. Yu, J. Chen, T. Gao, and M. Yu, "DAG-GNN: DAG structure learning with graph neural networks," in *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, ser. Proceedings of Machine Learning Research, K. Chaudhuri and R. Salakhutdinov, Eds., vol. 97. PMLR, 2019. [Online]. Available: http://proceedings.mlr.press/v97/yu19a.html pp. 7154–7163.

[334] R. Guo, L. Cheng, J. Li, P. R. Hahn, and H. Liu, "A survey of learning causality with data: Problems and methods," *ACM Comput. Surv.*, vol. 53, no. 4, pp. 75:1–75:37, 2021. [Online]. Available: https://doi.org/10.1145/3397269

[335] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," in *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2014. [Online]. Available: http://arxiv.org/abs/1312.6114

[336] J. Chung, Ç. Gülçehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," *CoRR*, vol. abs/1412.3555, 2014. [Online]. Available: http://arxiv.org/abs/1412.3555

[337] E. Jang, S. Gu, and B. Poole, "Categorical reparameterization with gumbel-softmax," in *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. [Online]. Available: https://openreview.net/forum?id=rkE3y85ee

[338] C. J. Maddison, A. Mnih, and Y. W. Teh, "The concrete distribution: A continuous relaxation of discrete random variables," in *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. [Online]. Available: https://openreview.net/forum?id=S1jE5L5gl

[339] T. N. Kipf, E. Fetaya, K. Wang, M. Welling, and R. S. Zemel, "Neural relational inference for interacting systems," in *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, ser. Proceedings of Machine Learning Research, J. G. Dy and A. Krause, Eds., vol. 80. PMLR, 2018. [Online]. Available: http://proceedings.mlr.press/v80/kipf18a.html pp. 2693–2702.

[340] L. Ruff, R. A. Vandermeulen, N. Görnitz, A. Binder, E. Müller, K. Müller, and M. Kloft, "Deep semi-supervised anomaly detection," in *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. [Online]. Available: https://openreview.net/forum?id=HkgH0TEYwH

[341] L. Ruff, N. Görnitz, L. Deecke, S. A. Siddiqui, R. A. Vandermeulen, A. Binder, E. Müller, and M. Kloft, "Deep one-class classification," in *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, ser. Proceedings of Machine Learning Research, J. G. Dy and A. Krause, Eds., vol. 80. PMLR, 2018. [Online]. Available: http://proceedings.mlr.press/v80/ruff18a.html pp. 4390–4399.

[342] S. Goyal, A. Raghunathan, M. Jain, H. V. Simhadri, and P. Jain, "DROCC: deep robust one-class classification," in *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, ser. Proceedings of Machine Learning Research, vol. 119. PMLR, 2020. [Online]. Available: http://proceedings.mlr.press/v119/goyal20c.html pp. 3711–3721.

[343] J. Ji, J. Wang, C. Huang, J. Wu, B. Xu, Z. Wu, J. Zhang, and Y. Zheng, "Spatio-temporal self-supervised learning for traffic flow prediction," in *AAAI 2023*, 2023.

[344] X. Guo and L. Zhao, "A systematic survey on deep generative models for graph generation," *IEEE Trans. Pattern Anal. Mach. Intell.*, 2023.

[345] H. Jiang, J. Pei, D. Yu, J. Yu, B. Gong, and X. Cheng, "Applications of differential privacy in social network analysis: A survey," *IEEE Trans. Knowl. Data Eng.*, 2023.

[346] B. Zhou and J. Pei, "Preserving privacy in social networks against neighborhood attacks," in *ICDE*, 2008.

[347] M. Hay, G. Miklau, D. D. Jensen, D. F. Towsley, and P. Weis, "Resisting structural re-identification in anonymized social networks," *Proc. VLDB Endow.*, 2008.

[348] C. Shi, Y. Li, J. Zhang, Y. Sun, and P. S. Yu, "A survey of heterogeneous information network analysis," *IEEE Trans. Knowl. Data Eng.*, 2017.

[349] C. J. Yang, Y. Xiao, Y. Zhang, Y. Sun, and J. Han, "Heterogeneous network representation learning: A unified framework with survey and benchmark," 2022.

[350] E. Zheleva and L. Getoor, "Preserving the privacy of sensitive relationships in graph data," in *PinKDD*, 2007.

[351] K. Liu and E. Terzi, "Towards identity anonymization on graphs," in *SIGMOD*, 2008.

[352] X. Ying and X. Wu, "Randomizing social networks: a spectrum preserving approach," in *SDM 2008*, 2008.

[353] H. H. Nguyen, A. Imine, and M. Rusinowitch, "Anonymizing social graphs via uncertainty semantics," in *CCS 2015*, 2015.

[354] A. Hoang, B. Carminati, and E. Ferrari, "Privacy-preserving sequential publishing of knowledge graphs," in *ICDE*, 2021.

[355] Y. Sun, J. Han, X. Yan, P. S. Yu, and T. Wu, "Pathsim: Meta path-based top-k similarity search in heterogeneous information networks," *Proc. VLDB Endow.*, 2011.

[356] Y. Sun and J. Han, "Mining heterogeneous information networks: a structural analysis approach," *SIGKDD Explor.*, 2012.

[357] J. Blocki, A. Blum, A. Datta, and O. Sheffet, "The johnson-lindenstrauss transform itself preserves differential privacy," in *FOCS*, 2012.

[358] Z. Qin, T. Yu, Y. Yang, I. Khalil, X. Xiao, and K. Ren, "Generating synthetic decentralized social graphs with local differential privacy," in *CCS 2017*, 2017.

[359] C. Ling, C. Yang, and L. Zhao, "Deep generation of heterogeneous networks," in *ICDM*, 2021.

[360] D. Zhou, L. Zheng, J. Han, and J. He, "A data-driven graph generative model for temporal interaction networks," in *KDD*, 2020.

[361] R. Hussein, D. Yang, and P. Cudré-Mauroux, "Are meta-paths necessary?: Revisiting heterogeneous graph embeddings," in *CIKM*, 2018.

[362] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. C. Courville, and Y. Bengio, "Generative adversarial networks," *CoRR*, 2014.

[363] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, 1997.

[364] B. Wu, S. Zhao, C. Chen, H. Xu, L. Wang, X. Zhang, G. Sun, and J. Zhou, "Generalization in generative adversarial networks: A novel perspective from privacy protection," in *NeurIPS*, 2019.

[365] F. M. Harper and J. A. Konstan, "The movielens datasets: History and context," 2015.

[366] J. Tang, J. Zhang, L. Yao, J. Li, L. Zhang, and Z. Su, "Arnetminer: Extraction and mining of academic social networks," in *KDD*, 2008.

[367] L. Zhang, L. Zhao, S. Qin, D. Pfoser, and C. Ling, "TG-GAN: continuous-time temporal graph deep generative models with time-validity constraints," in *WWW*, 2021.

[368] X. Chen, J. He, X. Han, and L. Liu, "Efficient and degree-guided graph generation via discrete diffusion modeling," in *ICML*, 2023.

[369] C. Yang, H. Wang, K. Zhang, L. Chen, and L. Sun, "Secure deep graph generation with link differential privacy," *arXiv preprint arXiv:2005.00455*, 2020.

[370] Z. Zeng, R. Zhu, Y. Xia, H. Zeng, and H. Tong, "Generative graph dictionary learning," in *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, ser. Proceedings of Machine Learning Research, A. Krause, E. Brunskill, K. Cho, B. Engelhardt, S. Sabato, and J. Scarlett, Eds., vol. 202. PMLR, 2023. [Online]. Available: https://proceedings.mlr.press/v202/zeng23c.html pp. 40 749–40 769.

[371] L. Fang, L. Li, Y. Liu, V. I. Torvik, and B. Ludäscher, "KAER: A knowledge augmented pre-trained language model for entity resolution," *CoRR*, vol. abs/2301.04770, 2023. [Online]. Available: https://doi.org/10.48550/arXiv.2301.04770

[372] T. Wei, B. Jin, R. Li, H. Zeng, Z. Wang, J. Sun, Q. Yin, H. Lu, S. Wang, J. He, and X. Tang, "Towards unified multi-modal personalization: Large vision-language models for generative recommendation and beyond," *CoRR*, vol. abs/2403.10667, 2024. [Online]. Available: https://doi.org/10.48550/arXiv.2403.10667

[373] T. Wei, Z. Guo, Y. Chen, and J. He, "Ntk-approximating MLP fusion for efficient language model fine-tuning," in *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, ser. Proceedings of Machine Learning Research, A. Krause, E. Brunskill, K. Cho, B. Engelhardt, S. Sabato, and J. Scarlett, Eds., vol. 202. PMLR, 2023. [Online]. Available: https://proceedings.mlr.press/v202/wei23b.html pp. 36 821–36 838.