VIETNAM NATIONAL UNIVERSITY HANOI
UNIVERSITY OF ENGINEERING AND TECHNOLOGY

HOANG THI LINH

ADAPTIVE SAMPLING TOWARDS EFFICIENT REPRESENTATION LEARNING WITH GRAPH TRANSFORMER IN DYNAMIC GRAPHS

MASTER'S THESIS OF COMPUTER SCIENCE

Hanoi - 2023

VIETNAM NATIONAL UNIVERSITY, HANOI
UNIVERSITY OF ENGINEERING AND TECHNOLOGY

HOANG THI LINH

ADAPTIVE SAMPLING TOWARDS EFFICIENT REPRESENTATION LEARNING WITH GRAPH TRANSFORMER IN DYNAMIC GRAPHS

Major: Computer Science
Code: 8480101.01

MASTER'S THESIS OF COMPUTER SCIENCE

SUPERVISOR: Dr. Viet Cuong Ta

Hanoi - 2023

AUTHORSHIP

I hereby declare that the work contained in this thesis is of my own and has not been previously submitted for a degree or diploma at this or any other higher education institution. To the best of my knowledge and belief, the thesis contains no materials previously published or written by another person except where due reference or acknowledgment is made.

Signature:. . . . . .

SUPERVISOR'S APPROVAL

I hereby approve that the thesis in its current form is ready for committee examination as a requirement for the Master of Computer Science degree at the University of Engineering and Technology.

Signature:. . . . . .

ACKNOWLEDGEMENT

I have never thought that one day I will pursue higher education before. I would like to thank all the people who have inspired and supported me to accomplish a new page in my life. Without them, my accomplishment would be impossible.

First of all, I would like to sincerely thank my advisor Viet Cuong Ta for guiding me throughout my research experience. Cuong has been my role model since I joined the lab. I learn so much about formulating the research questions and methodology. His insightful feedback pushed me to sharpen my thinking and brought my work to a higher level. Apart from research, as I grow more senior in the lab, I even have the opportunity to teach with him, contribute to research grant proposals, and advise junior students in the lab. Gradually he has imparted me with the necessary skills to become a successful researcher and get me ready for the next step in my career.

During my master's student time, I have a chance to work as a research assistant at the Human-Machine Interaction Laboratory (HMI Lab) and I am also grateful to all members. Professor Thanh Ha always gave me meaningful advice that I found incredibly helpful to my research and life. Dr. Duyen and Dr. Chau usually engaged me to continue my studies. I want to thank other young lab members. Thuong and Giang who often eat outside and chat with me, helped me get over the hardest times when I was stressed. I have collaborated with Tuan Dung, he is always patient with me, I learned through him many skills that will help me a lot in the future. Hai Dang shared with me a lot of interest in graph learning, he encouraged me to believe in myself and become more confident.

I extend my heartfelt gratitude to the esteemed members of the Information Technology Faculty for their invaluable guidance, support, and mentorship throughout this journey.

I would like to express my gratitude to the Vingroup Innovation Foundation for their generous support in granting me a one-year master's scholarship. Their invaluable assistance has enabled me to pursue my academic aspirations and advance my education in Computer Science.

Thanks to all my friends whom I met in graduate classes who shared the same passion for computer science with me: Dat, Khanh, Phi, Lam, Nhat, Khoa, and many others that I cannot mention. I have valuable experience being in a team with you to explore the beauty of computer science.

In my academic journey, I found some interesting people that I want to thank because they influenced and motivated me a lot. I met Viet Anh at KSE 2022, he gave me an interesting view of my work about over-smoothing on graph neural networks from the optimization perspective and helped me to realize that I need to build a strong philosophy. At LoG 2022, I have a chance to discuss the explainable of GNNs with Valentina and Chirag Varun. I had also a wonderful time in the 2022 Fatima Fellowship program to investigate the unbiased sampler for training the GNNs model under mentor Balin with another mentee Omar.

Besides my passion for the field that I followed, I want to thank my close friends: Linh Linh, Le, Ngat, Quyet, Trang, Oanh, and Hang. We have several hanging out together, whenever I felt blue they were always there to listen and keep me positive. Especially for Linh Linh - my roommate, I never forget the funny times when we played Bang Bang, raced to the top of Duolingo together, and some weird meals I made for you.

Finally, I want to express my gratitude to my family. My sister and brother, I have sought out advice from you and loved you with all my heart. My parents always support me with unconditional love even though they don't know much about my field. Dad and Mom, I am so proud of you!

Publications related to this thesis

1. T. L. Hoang and V. C. Ta, ”Effect of Cluster-based Sampling on the Over-smoothing Issue in Graph Neural Network”, KSE 2022.

2. T. L. Hoang and V. C. Ta, “Dynamic-GTN: Learning an Node Efficient Embedding in Dynamic Graph with Transformer”, PRICAI 2022 (presented at the best-paper session).

3. V. C. Ta, T. L. Hoang, N. S. Doan, V. T. Nguyen, D. N. Nguyen, T. T. T Pham, and N. N. Dang, ”Tabnet efficiency for facies classification and learning feature embedding from well log data”, Petroleum Science and Technology 2023, Journal ISI Q2.

4. T. L. Hoang and V. C. Ta, Balancing Structure and Position Information in Graph Transformer Network with a Learnable Node Embedding”, submitted to Journal Expert Systems With Applications ISI Q1, under reviewer.

ABSTRACT

Graphs are widely used to represent data in various domains, such as social networks, recommendation systems, and biological networks. However, real-world graphs are often dynamic, meaning that they evolve over time, which poses significant challenges for representation learning methods. This thesis proposes a novel approach to address the computational challenges of learning representations of dynamic graphs using graph transformers and adaptive sampling techniques called ASDGT. The key idea of the ASDGT method is to selectively sample a subset of nodes and edges to update the graph representations, rather than updating all nodes and edges at each time step. The proposed adaptive sampling method is based on the idea that nodes that change frequently are more important to learn, and therefore should be sampled more frequently. The thesis's contributions include the development of a novel approach for representation learning in dynamic graphs using adaptive sampling techniques and graph transformers. The proposed method produces more efficient and scalable representations of dynamic graphs, which can be applied to various domains. The thesis also provides insights into the importance of adaptive sampling techniques for efficient representation learning in dynamic graphs. We evaluated the proposed method and baselines on some popular real-world datasets, and the results demonstrate the effectiveness of ASDGT in various tasks, including node classification and link prediction.

Keywords: *Dynamic Graph, Adaptive Sampling, Graph Transformer*

Contents

List of Figures

List of Tables

List of Acronyms

Acronym	Definition
ASDGT	Adaptive Sampling Dynamic Graph Transformer
DGT	Dynamic Graph Transformer
GCN	Graph Convolutional Network
GNNs	Graph Neural Network
GRL	Graph Representation Learning
GT	Graph Transformer
MAB	Multi-armed Bandit
NS	Neighbor Sampling
VE	Variation Edges
VN	Variation Neighborhoods

Chapter 1

Introduction

"Est modus in rebus, sunt certi
denique fines, quos ultra citraque
nequit consistere retum."

Horace

1.1. Motivation

Graph data structures are ubiquitous in modeling complex relationships and interactions across various domains, including social networks [1, 2], recommendation systems [1, 3, 4], biological networks, knowledge graphs, and more. The inherent structure of graphs captures intricate connections and dependencies that traditional machine learning models struggle to represent effectively. With the advent of deep learning, there has been significant progress in developing models specifically tailored to handle graph-structured data, enabling more accurate and insightful analysis.

Deep learning on graphs has witnessed remarkable advancements in recent years. Traditional deep learning architectures, such as convolutional neural networks (CNN) [5] and recurrent neural networks (RNN) [6], are not directly applicable to graph data due to the irregular and variable connections between nodes. To address this challenge, researchers have introduced novel architectures and techniques designed specifically for graph data representation and processing. As a result, Graph Neural Networks (GNN)

have gained a lot of attention. The main idea of GNN is to find a mapping of the nodes in the graph to a latent space, which preserves several key properties of the graphs. Given that every single node has a certain influence on its neighbors, node embedding is created by GNN based on a message-passing mechanism to aggregate information from the neighborhood nodes, which can be used for downstream tasks such as node classification, edge prediction, or graph classification.

One such model is the Graph Transformer [7], which combines the power of attention mechanisms with GNN. The Graph Transformer utilizes self-attention mechanisms to capture both global and local dependencies between nodes, allowing for more expressive and flexible graph representations.

However, as the scale and complexity of real-world graph data continue to grow, there is an increasing need for scalable models capable of handling dynamic graphs. Dynamic graphs arise in scenarios where nodes and edges evolve over time, such as social networks with evolving connections or knowledge graphs with updated relationships. Figure 1.1 shows the example of the existence of the dynamic graph on the e-commerce system where nodes representing products and customers, as well as edges indicating purchase interactions, evolve over multiple time steps. Existing models lack the ability to adapt to such dynamic settings, thereby limiting their applicability in real-world scenarios.

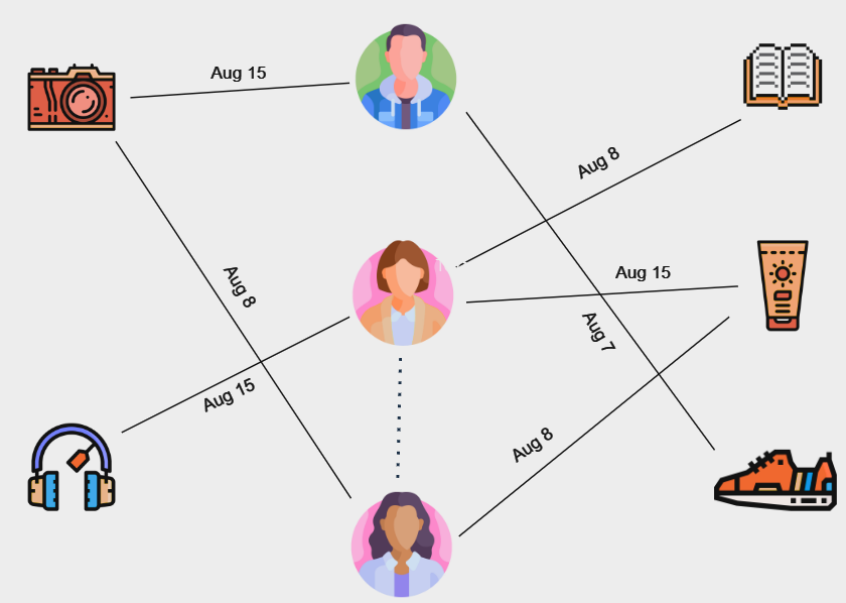

Figure 1.1: An illustration of dynamic user-item graph on the e-commerce system

To address this challenge, the development of the Dynamic Graph Transformer (DGT) [8] becomes crucial. The DGT aims to extend the Graph Transformer model to handle dynamic graph data, enabling the modeling of time-varying interactions and capturing temporal dependencies. By incorporating the temporal dimension into graph learning, the DGT facilitates more accurate and comprehensive analysis of dynamic graph data, thereby enabling tasks such as dynamic community detection, temporal link prediction, and evolving node classification.

By leveraging the power of deep learning on dynamic graphs, the DGT offers numerous advantages. Firstly, it enables the modeling of temporal dynamics in complex systems, capturing how node attributes and connections evolve over time. Secondly, it allows for the identification of temporal patterns and trends within the dynamic graph, providing valuable insights into the evolution of relationships and behaviors. Thirdly, it facilitates the prediction of future states and interactions within the dynamic graph, enabling proactive decision-making and intervention in various applications.

The DGT also holds promise in several domains. In social networks, it can help identify influential individuals and evolving communities, analyze information diffusion patterns, and predict temporal interactions and collaborations. In recommendation systems, it can capture the temporal preferences and dynamics of users, improving the accuracy of personalized recommendations over time. In biological networks, it can uncover gene regulatory dynamics, predict protein-protein interactions, and identify temporal patterns in disease progression.

In summary, the combination of graph data, deep learning techniques, and models like the Graph Transformer has revolutionized the analysis and understanding of complex systems. However, the lack of scalable models for dynamic graphs remains a challenge. This thesis aims to address this gap by developing the Dynamic Graph Transformer and exploring its applications in dynamic graph analysis. By doing so, it contributes to the advancement of deep learning on dynamic graph data, enabling more accurate and scalable analysis of evolving real-world networks and opening up new avenues for research and applications in various domains.

1.2. Contributions

In this thesis, we focus on the development and evaluation of the Adaptive Sampling Dynamic Graph Transformer (ASDGT) with a particular emphasis on its adaptive sampling design for efficient and scalable dynamic graph representation. Our primary contribution lies in demonstrating the improved computational efficiency, enhanced scalability, and contextual relevance and adaptability achieved through the integration of adaptive sampling in the dynamic graph learning framework.

- **Improved Computational Efficiency:** Adaptive sampling significantly reduces the computational complexity of the Transfomer-based model by focusing on a subset of the graph. This enables the model to process and learn from large-scale graphs efficiently, without being constrained by memory limitations. The reduced computational burden facilitates faster training and inference, making the ASDGT a practical solution for dynamic graph analysis.
- **Enhanced Scalability:** The adaptive sampling design allows the model to handle graphs of varying sizes. By selecting a subset of nodes and edges, the model can represent and learn from a portion of the graph at a time, enabling scalability to large-scale graphs that may exceed memory capacity. The ASDGT's ability to handle massive graphs contributes to its applicability in real-world scenarios.
- **Contextual Relevance and Adaptability:** Adaptive sampling enables our model to adapt to changes in the graph structure over time. By dynamically selecting relevant components, the model captures the contextual relevance and adaptability of the graph. This ensures that the graph representation remains up-to-date and meaningful, providing accurate insights and predictions in dynamic graph analysis tasks.

The improved computational efficiency achieved through adaptive sampling allows the ASDGT to process dynamic graphs more efficiently, reducing training and inference times. This efficiency enables researchers and practitioners to analyze large-scale graphs without being limited by computational resources. The enhanced scalability further extends the applicability of the ASDGT, making it suitable for real-world scenarios involving massive graph-structured data.

Moreover, the contextual relevance and adaptability offered by adaptive sampling contribute to the model's ability to capture the changing dynamics of the graph. By adaptively selecting nodes and edges based on their importance or relevance to the current context, the DGT can effectively represent the evolving graph structures. This adaptability ensures that the graph representation remains accurate and meaningful, even in the presence of dynamic changes.

1.3. Outlines

This thesis is structured as follows:

- Chapter 1: Introduction. This chapter provides the rationale behind utilizing graph data and emphasizes the significance of sampling techniques in the context of improving the capacity of DGT.
- Chapter 2: Background. This chapter provides a concise overview of the relevant literature and related work in the field. It serves as a foundation for understanding the current state of research and lays the groundwork for the subsequent chapters. The chapter covers key concepts, methodologies, and approaches that have been explored by researchers in the domain, offering valuable insights into the existing knowledge landscape. By examining the related work, we gain a comprehensive understanding of the advancements, challenges, and gaps in the field, which in turn informs the direction and contributions of this thesis.
- Chapter 3: Methodology. This chapter presents a detailed description of the methodology employed in this thesis—an adaptive sampler designed specifically for DGT called ASDGT. The adaptive sampler addresses the challenge of effectively sampling nodes and edges from large-scale graphs to enhance the training process of Graph Transformer models.
- Chapter 4: Experiments. This chapter presents the experimental setup, training details, and the obtained results of the proposed adaptive sampler for Graph Transformers. It aims to provide a comprehensive analysis of the performance and effectiveness of the methodology.

- Chapter 5: Conclusion. This chapter serves as the conclusion of the thesis, summarizing the key findings, contributions, and implications of the research conducted on the adaptive sampler for Graph Transformers. It also outlines potential directions for future work and areas that merit further investigation.

Chapter 2

Background

In order to understand the foundations and context of this thesis, it is essential to establish the preliminary concepts and methodologies that form the basis of our research. In this section, we will provide a comprehensive overview of the key preliminaries, including dynamic graph-structured data, the Transformer model, and adaptive sampling techniques. By establishing a solid understanding of these concepts, we lay the background for the development and evaluation of the Dynamic Graph Transformer (DGT) with its adaptive sampling design. This thesis aims to leverage these preliminaries to address the challenges associated with efficient and scalable dynamic graph representation and to contribute novel insights and methodologies to the field of graph analysis and representation.

2.1. Graph-related Concepts and Notions

2.1.1. Static Graph Modeling

Definition 2.1 (Stactic graph) *A graph $G = (V, E)$ where V is a finite set of nodes (or vertices) and E is the set of edges (or link), each being an unordered pair $(u, v) \in V \times V, u \neq v$.*

Definition 2.2 (Adjacency Matrix) *Matrix $A \in \mathbb{R}^{|V| \times |V|}$ is called an adjacency matrix and contains the connectivity information of each node. The $A_{i,j}$ represents the*

connection between two nodes v_i and v_j.

$$A_{i,j} = \begin{cases} 1, & \text{if } v_i \text{ is adjacent to } v_j \\ 0, & \text{otherwise} \end{cases}$$

Definition 2.3 (Node degree) *The degree of a vertice v is the number of edges attached to each vertex denoted by $deg(v)$.*

Definition 2.4 (Degree matrix) *The degree matrix D for a graph G is a $n \times n$ diagonal matrix defined as:*

$$D_{i,j} = \begin{cases} deg(v_i) & \text{if } i = j \\ 0, & \text{otherwise} \end{cases}$$

Definition 2.5 (Laplacian Matrix) *For a graph $G = (V, E)$ with A as its adjacency matrix, its Laplacian matrix is defined as $L = D - A$ where D is a diagonal degree matrix.*

Definition 2.6 (Weighted Graph) *A weighted graph is a graph whose edges are assigned values, known as weights.*

For example, in network flow modeling, the vertices can symbolize source and destination IP addresses, and the weight assigned to each edge may denote either the duration of a flow or the quantity of data transmitted during that flow.

2.1.2. Dynamic Graph Modeling

Definition 2.7 (Dynamic Graph) *We represent a dynamic graph as a sequence of non-decreasing chronological interactions $G = \{(u_1, v_1, t_1), (u_2, v_2, t_2), \cdots\}$ with $0 \leq t_1 \leq t_2 \leq \cdots$, where $u_i, v_i \in N$ denote the source node and destination node of the i-th link at timestamp t_i. N is the set of all the nodes. Each node $u \in N$ can be associated with node feature $\boldsymbol{x}_u \in \mathbb{R}^{d_N}$, and each interaction (u, v, t) has link feature $\boldsymbol{e}^t_{u,v} \in \mathbb{R}^{d_E}$, where d_N and d_E denote the dimensions of the node feature and link feature. If the graph is non-attributed, we simply set the node feature and link feature to zero vectors, i.e., $\boldsymbol{x}_u = 0$ and $\boldsymbol{e}^t_{u,v} = 0$.*

Definition 2.8 (Problem Formalization) *Given the source node u, destination node v, timestamp t, and historical interactions before t, i.e., $\{(u', v', t') \mid t' < t\}$, representation learning on dynamic graph aims to design a model to learn time-aware representations $\boldsymbol{h}_u^t \in \mathbb{R}^d$ and $\boldsymbol{h}_v^t \in \mathbb{R}^d$ for u and v with d as the dimension. We validate the effectiveness of the learned representations via two classic tasks in dynamic graph learning: (i) dynamic link prediction, which predicts whether u and v are connected at t; (ii) dynamic node classification, which infers the state of u or v at t.*

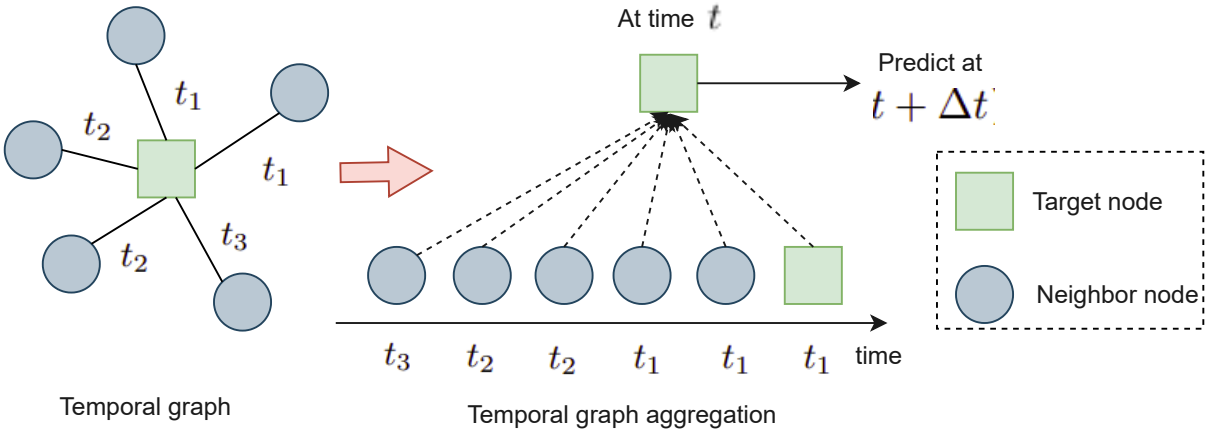

Figure 2.1: An example of node classification on dynamic graph

2.1.3. Graph Coarsening

Graph Coarsening [9, 10] reduced the number of nodes of the graph by clustering them into super-nodes, see the example in Figure 2.2. Given a graph $G = (V, E)$ the graph coarsening algorithm aims to construct an appropriate coarser graph $\hat{G} = (\hat{V}, \hat{E})$ that contains certain properties of G. $\hat{G}$ is obtained from the original graph by calculating a partition $C_1, C_2 ..., C_{|\hat{V}|}$ of V. Each cluster C_i corresponds to a super-node in $\hat{G}$ The partition can also be characterized by a matrix $\hat{P} \in \{0,1\}^{|V| \times |\hat{V}|}$, with $\hat{P}_{ij} = 1$ if and only if node v_i in G belongs to cluster C_i. Its normalized version can be defined by $P \triangleq \hat{P} D^{-\frac{1}{2}}$, where D is a $|V| \times |\hat{V}|$ diagonal matrix with $|C_i|$ as its i-th diagonal entry. The feature matrix and weighted adjacency matrix of $\hat{G}$ are defined by $\hat{X} \triangleq P^T X$ and $\hat{A} \triangleq P^T A P$. After Graph Coarsening, the number of nodes/edges in $\hat{G}$ is significantly smaller than that of G. The coarsening rate can be defined as $c = \frac{|\hat{V}|}{|V|}$.

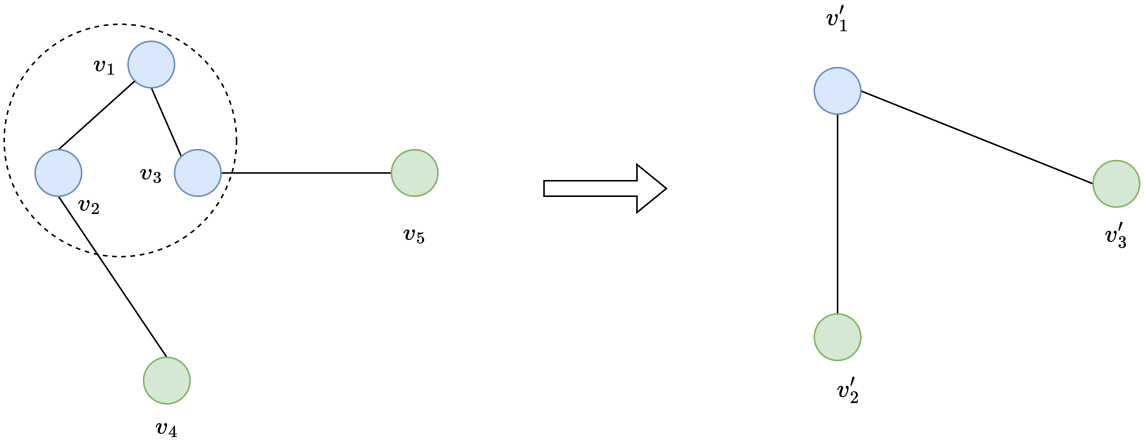

Figure 2.2: The example of graph coarsen on toy graph

2.2. Deep Learning on Graph

2.2.1. Graph Representation Learning

Graph Representation Learning (GRL) [11–15] methods aim at learning low-dimensional continuous vector representations for graph-structured data, also called embeddings as are represented in 2.9:

Definition 2.9 (Graph Embedding [15]) *Given a graph $G = (V, E)$, a graph embedding is a mapping $f : v_i \rightarrow z_i \in \mathbb{R}^d$, $\forall i \in [n]$ such that $d \ll |V|$ and the function f preserves some proximity measure defined on graph G.*

In other words, we want to map nodes into latent space, where geometric relations in this latent space correspond to relationships in the original graph. For example, the distance between embeddings should represent a metric evaluating the similarity between the corresponding members of the network. With the two nodes u and v the good embedding satisfied as:

$$similarity(u, v) \approx z_v^T z_u \tag{2.1}$$

where, z_u, z_v is the embedding of nodes u, v respectively.

A good network embedding should also possess the following characteristics: (1) Adaptability: The learned embeddings should work for networks that are constantly

evolving without repeating the learning process all over again. (2) Scalability: The embeddings should be generated in a reasonable period of time for large-scale networks. (3) Low dimensionality: The embeddings should be low-dimensional since when labeled data is scarce; low-dimensional models generalize better and converge faster.

One of the popular techniques for network embedding is the random walk-base method, exemplified by influential techniques like DeepWalk [16], node2vec [17], and LINE [18]. These methods leverage the concept of random walks, simulating exploratory journeys through the network's nodes, to generate embeddings that encode the structural intricacies of the graph. DeepWalk and node2vec treat these random walks as sentences, applying language modeling principles to transform traversal sequences into meaningful embeddings. In contrast, the LINE method focuses on optimizing both first-order (node co-occurrence) and second-order (neighborhood similarity) proximities, resulting in embeddings that encapsulate local and global graph characteristics. The optimization objective for LINE can be formulated as:

$$\mathcal{L}_{LINE} = \sum_{(i,j)\in E} \left(\log \sigma \left(\mathbf{v}_i^T \mathbf{v}_j \right) + k \cdot \mathbb{E}_{j \sim P_n(j)} \left[\log \sigma \left(-\mathbf{v}_i^T \mathbf{v}_j \right) \right] \right)$$

where σ represents the sigmoid function, $\mathbf{v}_i$ is the embedding of node i, and $P_n(j)$ is the negative sampling distribution.

Besides the shadow embedding base random walk, some methods are grounded in GNN with notable instances including GraphSAGE [19] and Graph Attention Networks (GAT) [20]. These approaches harness the expressive power of GNN architectures to aggregate and propagate information across nodes, facilitating the generation of context-rich embeddings that consider both the connectivity and attributes of nodes in the network. As GNNs have gained prominence across diverse graph-related tasks, techniques like GraphSAGE and GAT emphasize the incorporation of neural network paradigms to capture intricate dependencies, leading to embeddings that seamlessly encapsulate the topological and feature-based attributes of nodes within the graph. This integration of neural networks in the representation learning process has propelled GNN-based methods to the forefront of research in the field. The aggregation process in GraphSAGE can be defined as:

$$\mathbf{h}_v^{(l)} = \text{AGGREGATE}^{(l)} \left(\left\{ \mathbf{h}_u^{(l-1)}, \forall u \in \mathcal{N}(v) \right\} \right)$$

where $\mathbf{h}_v^{(l)}$ is the embedding of node v at layer l, and $\mathcal{N}(v)$ denotes the neighbors of node v.

2.2.2. Transformer for Graphs

Transformer [21] architecture, developed by Vaswani et al. in 2021, was first observed in the domain of machine translation. In the paper, the authors introduce the Transformer as a novel encoder-decoder structure of multiple self-attention blocks. Given $X \in \mathbb{R}^{n \times d}$ as the input of each Transformer layer, where n is the number of tokens, d is the dimension of each token, then one block layer can be a function $g : \mathbb{R}^{n \times d} \to \mathbb{R}^{n \times d}$ with $g(X) = Z$ defined by:

$$A_{att} = \frac{1}{\sqrt{d}} XQ(XK)^T \tag{2.2}$$

$$\tilde{X} = Softmax(A_{att})(XV), \tag{2.3}$$

$$M = LayerNorm(\tilde{X}O + X), \tag{2.4}$$

$$F = \sigma(MW_1 + b_1)W_2 + b_2, \tag{2.5}$$

$$Z = LayerNorm(M + F), \tag{2.6}$$

where $Softmax(.)$ refers to the row-wise softmax function, $LayerNorm(.)$ refers to the layer normalization function [22], σ denotes the activation function, in our work, we use the ReLU function as the activation function. $Q, K, V, O \in \mathbb{R}^{d \times d}, W_1 \in \mathbb{R}^{d \times d_g}, b_1 \in \mathbb{R}^{d_g}, W_2 \in \mathbb{R}^{d_g \times d}, b_2 \in \mathbb{R}^d$ are trainable parameters in the layer.

Additionally, it is customary to incorporate multiple attention heads to expand the self-attention mechanism into multi-head self-attention. Specifically, the matrices Q, K, V are decomposed into H heads, where $Q^{(h)}, K^{(h)}, V^{(h)} \in \mathbb{R}^{d \times d_h}$ with $d = \sum d_h$ and $\tilde{X}^{(h)} \in \mathbb{R}^{n \times d_h}$ from attention heads are concatenated to obtained $\tilde{X}$:

$$A_{att}^{(h)} = \frac{1}{\sqrt{d}} XQ^{(h)}(XK^{(h)})^T, \tag{2.7}$$

$$\tilde{X} = ||(SoftMax({A_{att}}^{(h)})XV^{(h)}) \tag{2.8}$$

Applying the transformative power of the Transformer architecture to graph data has ushered in a new era of graph representation learning. Originally designed for natural language processing, the Transformer's self-attention mechanism has proven adaptable

to capturing complex relationships present within graphs. In this paradigm, each node in the graph is treated as a token, and attention mechanisms allow nodes to weigh the importance of their neighboring nodes based on learned relationships. One popular adaptation of the Transformer for graph data is the Graph Transformer (GT) [7]. In GT, the self-attention mechanism enables each node to gather information from its neighbors, creating context-rich embeddings that preserve both local and global graph structures.

The GT architecture offers a distinctive set of advantages when compared to traditional GCN and GAT. The Graph Transformer excels at capturing long-range dependencies across nodes in the graph through its self-attention mechanism, making it well-suited for tasks involving global graph structures. Unlike GCNs and GATs that predominantly aggregate information from immediate neighbors, the Graph Transformer's non-local aggregation capability enables it to consider information from all nodes concurrently, allowing for a more holistic perspective. The adaptive nature of the self-attention mechanism in GT allows nodes to dynamically adjust the importance of different neighbors during aggregation, leading to fine-grained and context-aware feature representations. Moreover, GT is well-equipped to handle heterogeneous graphs, integrating diverse attributes and edge types seamlessly. Additionally, its parallelizable nature facilitates efficient computation on modern hardware, enhancing scalability for large-scale graphs and distributed training scenarios. These distinctive strengths position GT as a potent contender in the domain of graph representation learning

2.2.3. Representation Learning on Dynamic Graphs

The existing modeling approaches are roughly divided into two categories based on how the dynamic graph is constructed: discrete-time methods and continuous-time methods.

Discrete-time Methods: This category of methods deals with a sequence of discretized graph snapshots that coarsely approximate a time-evolving graph. DynAERNN [23] is an auto-encoding method that minimizes reconstruction loss and learns incremental node embeddings from previous time steps. DySAT [24] computes dynamic embeddings by employing structural attention layers on each snapshot, followed by temporal attention layers to capture temporal variations among snapshots, as inspired by the self-attention mechanism. VGRNN [25] utilizes variational techniques, this model introduces latent random variables to collaboratively model hidden states in

graph recurrent neural networks. It adeptly captures changes in both graph topology and node attributes. EvolveGCN [26] recently leverages RNNs to regulate the GCN model (i.e., network parameters) at each time step to capture the dynamism in the evolving network parameters.

Continuous-time Methods: Methods in this category operate directly on time-evolving graphs without time discretization and focus on designing various temporal aggregators to extract information. The dynamic graphs are represented as a series of chronological interactions with precise timestamps JODIE [27] see two coupled RNNs to update dynamic node embeddings based on each interaction. DyRep [28] employs a joint learning approach, DyRep updates node representations in response to events involving the nodes, effectively capturing dynamic changes in both network topology and node activities. Recent work such as TGAT [29] and TGNs [30] use a different graph creation technique, namely a time-recorded multi-graph, which allows for more than one interaction (edge) between two nodes. A single TGAT layer is used to collect one-hop neighborhoods, similar to the encoding process in static models (e.g. GraphSAGE [19]). By stacking numerous layers, the TGAT model can capture high-order topological information. TGNs generalize TGAT's aggregation and use a node-wise memory to keep track of long-term dependencies.

2.3. Sampling for Training Graph Neural Networks

Sampling strategies play a pivotal role in training GNN models by dictating the nodes or edges used for each iteration. This section delves into the diverse realm of sampling techniques designed to enhance the efficiency and effectiveness of GNN training. We explore a spectrum of methodologies ranging from node-wise and layer-wise sampling approaches to a deeper investigation of node sampling as a bandit problem. These techniques offer crucial insights into harnessing sampling as a strategic tool in refining the performance and convergence of GNN.

2.3.1. Node-wise Sampling Methods

Node-wise sampling method is one of the fundamental sampling methods. The common idea between node-wise sampling methods is that they employ the sampling process on each node and sample neighbors based on specific probability.

$$SN(v) = Sampling^{(k)}(N(v), P) \tag{2.9}$$

where $P \sim Uniform(0, M)$ or $P \sim Metric(v)$. For random sampling, the probability P obeys distribution, and M denotes the limit of neighbors to be sampled of node v. While, in non-random sampling, the probability P is non-uniform and is proportional to particular metrics of node. GraphSAGE [19] and VR-GCN [31] are the most popular in this category. Though successfully achieving a comparable convergence rate, the memory complexity is higher and the time complexity remains the same, NS scheme alleviates the memory bottleneck of GCN, the GCN, there exists redundant computation under NS since the embedding calculation for each node in the same layer is independent, thus the complexity grows exponentially when the number of layers increases.

2.3.2. Layer-wise Sampling Methods

Layer-wise sampling method is the improvement of node-wise sampling method and generally samples a certain number of nodes together in one sampling step. In this category of sampling methods, since multiple nodes are jointly sampled in each layer, the time cost of the sampling process is significantly reduced by avoiding the exponential extension of neighbors. More importantly, a novel view that one can interpret loss and graph convolutions as integral transformations of embedding computation to reduce sampling variance by leveraging importance sampling technique is introduced in the works [32, 33].

The forward propagation of GCN is as:

$$h^{(l+1)}(v) = \sigma\left(\int \hat{S}(u, v) h^{(l)} W^{(l)} dP(u)\right) \tag{2.10}$$

where $\sigma(.)$ denotes a specific activation function, and $\hat{A}(u, v)$ is normalized adjacency matrix. $h^{(l)}(u)$ and $W^{(l)}$ simplify the hidden feature vector of u and weight matrix in

the $l-$th layer, respectively. And P is the sampling distribution to represent various sampling. In layer-wise sampling methods, one node is sampled under the influence of other nodes in the current layer, and the impact of other nodes in the form of importance sampling as

$$h^{(l+1)}(v) = \sigma\left(\hat{A}(v,u)\mathbb{E}_{q(u)}\left[\frac{p(u \mid v) \times h^{(l)}(u)}{q(u)}\right] W^{(l)}\right) \tag{2.11}$$

where $q(u)$ denotes the probability of sampling node u under the condition that all nodes in the current layer are given. In this situation, to accelerate the forward propagation in 2.11, we specify a concrete instance and apply Monte-Carlo approximation to the expectation $\mu_q(v) = \mathbb{E}_{q(u)}\left[p(u \mid v) \cdot h^{(l)}(u) / \ q(u)\right]$:

$$\hat{\mu}_q(v_i) = \frac{1}{n}\sum_{j=1}^{n} \frac{p\left(\hat{u}_j \mid v_i\right) \times h^{(l)}\left(\hat{u}_j\right)}{q\left(\hat{u}_j \mid v_1, \ldots, v_n\right)}. \tag{2.12}$$

Here, $\hat{\mu}_q(v_i)$ is the approximate expectation where $\hat{u}_j \sim q\left(\hat{u}_j \mid v_1, \ldots, v_n\right) . q\left(\hat{u}_j \mid v_1, \ldots, v_n\right)$ is the specified format of $q(u)$ that denotes the probability of sampling node u_j given nodes $v_1, v_2, \ldots, v_n$ in the current layer. Next up, the target for optimizing sampling is to reduce the variance of $\hat{\mu}_q(v_i)$ as far as possible, that is, finding an optimal sampling probability $q^*\left(u_j\right)$ to minimize $\mathrm{Var}_q\left(\hat{\mu}_q(v_i)\right)$.

While it is true that IS (Importance Sampling) outperforms uniform sampling and that layer-wise sampling successfully reduces both time and memory complexities, it is important to acknowledge a significant limitation of this sampling strategy. This limitation results in a large variance in the approximate embedding.

2.3.3. Nodes Sampling as Bandit Problem

The majority of prior investigations pertaining to dynamic graph neighbor sampling have operated under the assumption of a static environment, where the sampling preference of the target node remains constant over time. However, real-world scenarios frequently exhibit dynamic characteristics owing to the ever-changing interests of users. In instances where existing algorithms fail to discern impending shifts in the underlying sampling policy, their outcomes ultimately deviate towards sub-optimal conclusions that rely on outdated observations.

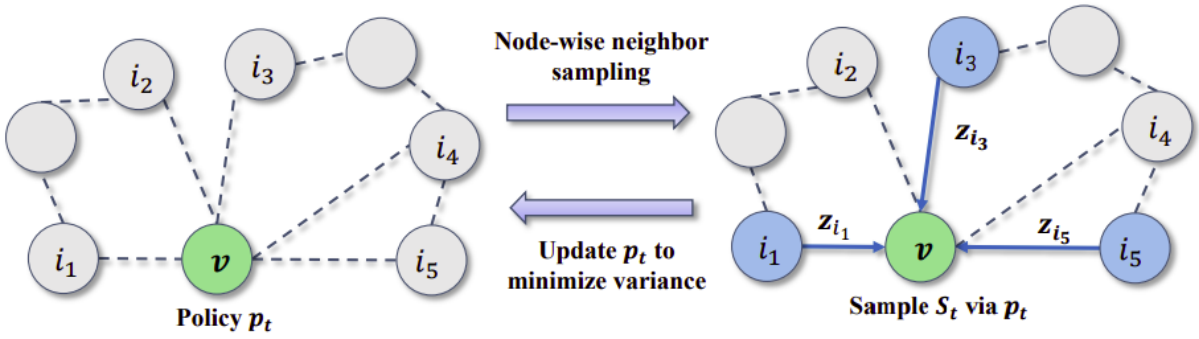

Figure 2.3: Formulate node-wise neighbor sampling as a MAB problem

Multi-armed bandit (MAB) [27] is a powerful framework for algorithms that make decisions over time under uncertainty. To deploy the bandit algorithm on the graph sampling problem, we consider the target node and its neighbors. Then we can naturally cast the sampling policy as a bandit problem.

In the realm of the standard k-armed bandit problem, the process of neighborhood sampling on a graph can be likened to a recurring game between the sampling policy and the optimal policy. Each stage consists of two steps: Firstly, the optimal policy selects k rewards, denoted as $r_1, \ldots, r_k \in [0, 1]$ for each neighbor. Secondly, the target node v_i chooses a neighbor $v_j \in N_i$ without any knowledge of the reward distribution of the optimal policy and subsequently observes the reward r_j. Previous literature [34] has extensively explored the standard k-armed bandit problem on graphs.

In the context of dynamic scenarios, the application of the adversarial bandit approach could present a more robust alternative to the conventional bandit methodology when it comes to optimizing graph neighbor sampling policies. The unique challenges posed by dynamic graphs make it crucial to leverage the inherent uncertainty and adaptability of the adversarial bandit framework for better policy optimization. Unlike the conventional bandit setup, the adversarial bandit framework takes into account the evolving preferences and changes in the graph's structure over time, making it well-suited for the task of sampling optimal neighbors in dynamic graph scenarios.

In the adversarial bandit framework, we consider a scenario where there are multiple arms (neighbor candidates) and a time horizon extending to T. At each discrete

time step t, the system encounters a target node v_i along with its corresponding d-dimensional contextual representation vector h_i. A subset $N_i \subseteq [K]$ of neighboring nodes is provided for selection. Each neighboring node v_j comes with its own contextual representation vector h_j. Figure 2.3 from [35] illustrates to formulating node-wise neighbor sampling as a MAB can be applied to adversarial bandit algorithms.

Within this framework, the sampling policy's objective is to strategically select a neighboring node v_j for the given target node v_i at each time step. Subsequently, the process generates a stochastic reward $r_{ij}(t)$, reflecting the quality of the chosen neighbor. This reward plays a pivotal role in updating and adapting the sampling policy's parameter settings, allowing it to dynamically adjust to changing graph dynamics.

The adversarial bandit approach brings an element of adaptability to the sampling policy, allowing it to respond effectively to the dynamic nature of the graph. By continuously optimizing the selection of neighboring nodes in response to changing contextual information and evolving preferences, the adversarial bandit framework holds the potential to provide more accuracy and efficiency in dynamic graph scenarios.

Chapter 3

Methodology

In this chapter, we present the novel adaptive sampling methods employed in the context of improving the capacity of the DGT model called ASDGT, specifically focusing on temporal network analysis. These adaptive sampling techniques leverage contextual bandit algorithms to intelligently select subsets of nodes and edges from dynamic graphs. By incorporating contextual information, the DGT model can effectively capture the temporal dynamics of the network. Furthermore, we provide a theoretical analysis of the regret associated with the proposed adaptive sampling methods, offering valuable insights into their performance and optimality.

We illustrate the ASDGT in Figure 3.1. The input graph passes to three modules parallel, they are Positional Encoding, Adaptive Sampling, and Graph Coarsening. Then, node sequences construct by nodes result in adaptive sampling, super-nodes from coarser graph, and global nodes are input for Transformer layers.

3.1. Adaptive Sampling

In this section, we introduce our proposed adaptive sampling methodology, which plays a crucial role in enhancing the efficiency and effectiveness of the graph transformer-based model. Our Adaptive Node Sampling module aims to choose the batch of the most informative nodes by the multi-armed bandit mechanism. Because the changes vary on dynamic graphs, the contributions of node nodes to learning performance can be time-sensitive. We use the adversarial setting in the multi-armed bandit problem due

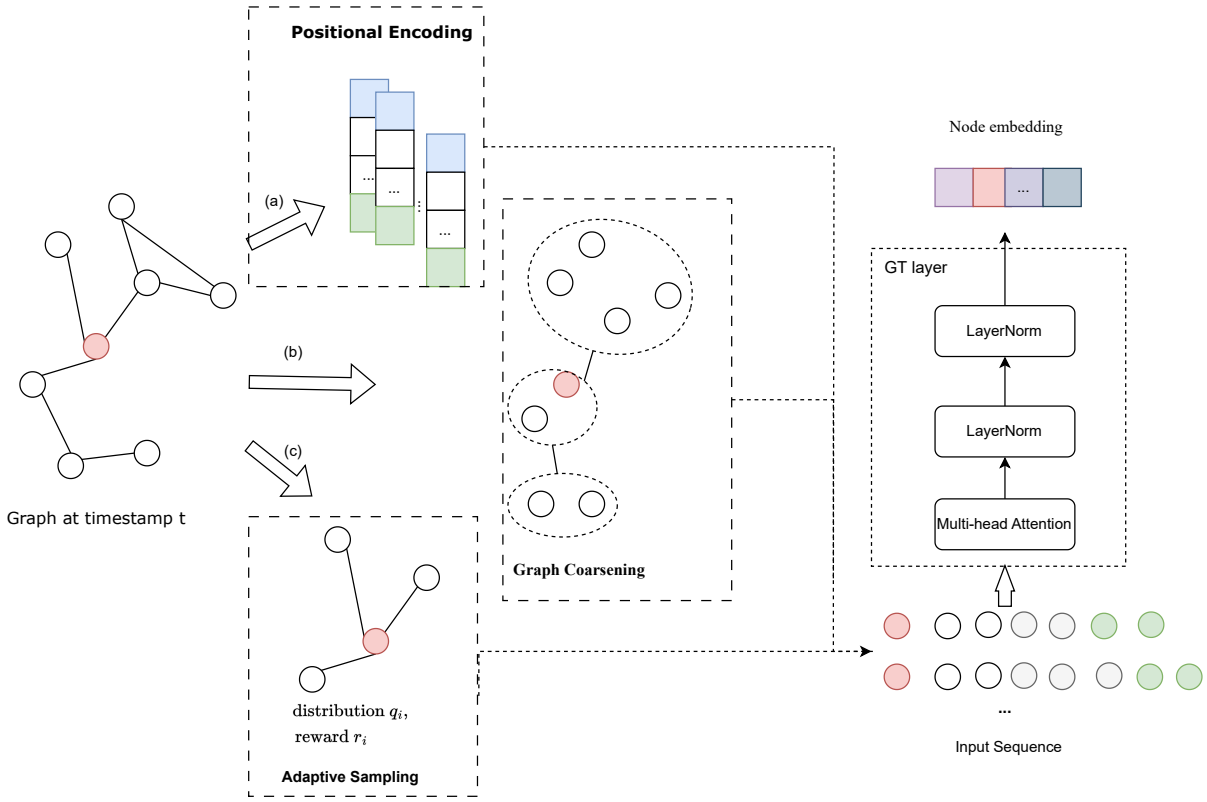

Figure 3.1: The ASDGT architecture for learning node embedding.

to the rewards which are associated with the model training process are not independent random variables across the interations.

The adaptive sampling process begins with the initialization of the dynamic graph. Recall the dynamic graph that we introduced before $G = (V, E, T)$, where V represents the set of nodes, E represents the set of edges, and T represents the set of time steps. The initial state of the graph includes its structure, node features, edge weights, and any other relevant information.

To efficiently represent nodes in the dynamic graph, the DGT model generates an initial node embedding vector for each node. Let h_t^v denote the node embedding vector for node v at time step t. The node embedding vectors capture the essence of the node's characteristics in a lower-dimensional space. These embeddings can be obtained using techniques such as GCN or GAT.

The DGT model learns a policy that maps the current node embedding vector to a probability distribution over nodes and edges. This policy guides the sampling process,

determining which nodes and edges should be included in the subgraph. Let $P(V_s|\{h\}_t)$ denote the probability distribution over nodes in the subgraph and $P(E_s|h_t)$ denote the probability distribution over edges in the subgraph given the node embedding vector h_t.

To optimize the policy, adversarial bandit algorithms such as Exp3 [36] or Online Mirror Descent can be employed. These algorithms strike a balance between exploration and exploitation based on the observed rewards. The policy learning process can be formulated as follows:

$$\text{Maximize} \quad \sum_{t \in T} \mathrm{R}(t) \tag{3.1}$$

$$\text{Subject to} \quad P(V_s|h_t), P(E_s|h_t) \tag{3.2}$$

where $\mathrm{R}(t)$ represents the reward obtained at time step t based on the selected subgraph.

$$R(t) = \frac{1}{N} \sum_{v \in V} A_{t,v} \cdot ||H_{t,v}|| \tag{3.3}$$

The reward function is used to evaluate the attention-based performance in our algorithm. The reward $R(t)$ is determined by summing up the contributions from all nodes in the graph, represented by the summation symbol $\sum_{v \in V}$. For each node v, the contribution is computed as the product of the attention score $A_{t,v}$ and the magnitude of the node embedding $||H_{t,v}||$. The attention score $A_{t,v}$ represents the relevance or importance of node v in the attention mechanism at time step t taken from the Transformer layer as from the equation 2.7. It is a measure of how much attention should be given to that particular node based on its characteristics and its relationship with other nodes in the graph.

By using this reward function, we aim to quantify the performance of the attention mechanism in our algorithm solely based on the attention scores and node embeddings. The reward value R_t provides an indication of the quality and effectiveness of the attention mechanism in selecting relevant nodes at time step t. This measure can be utilized to optimize and evaluate the attention-based aspects of our algorithm.

Algorithm 3.1: Extent EXP3 Algorithm for node-wise sampling

Input: Number of nodes K, Learning rate η, Exploration rate γ

Output: Action probabilities p

Initialize weight vector $w = (1, 1, \ldots, 1)$;

for *each iteration* **do**

  Compute the probability distribution $p = (p_1, p_2, \ldots, p_K)$ using the softmax function: $p_k = (1 - \gamma) \cdot \frac{w_k}{\sum_{i=1}^{K} w_i} + \frac{\gamma}{K}$;

  Choose an action a based on the distribution p;

  Receive reward r;

  Update the weights using the EXP3 update rule: $w_a = w_a \cdot e^{\eta \cdot \frac{r}{p_a}}$;

  $w_k = w_k \cdot e^{-\eta \cdot \frac{r}{K \cdot p_k}}$, for $k \neq a$;

end

It is important to note that the reward function 3.3 is specifically tailored to our algorithm and the problem it addresses. Its usage allows us to assess attention-based performance and make informed decisions regarding the attention mechanism's behavior and effectiveness based on the provided attention scores and node embeddings.

After the sampling decision is made, the DGT model extracts a subgraph from the dynamic graph. This subgraph represents a localized view of the graph at a specific time step, encompassing the chosen nodes and edges. Let $G_s = (V_s, E_s)$ denote the subgraph extracted from the dynamic graph G.

To adapt to the changing dynamics of the graph, the DGT model updates the node embedding vectors of the selected nodes in the subgraph. This update incorporates information from the current time step and the previous state of the graph, allowing the node embeddings to reflect the evolving nature of the graph accurately.

Algorithm 3.2: Adaptive Sampling Iterative Process

Input: Dynamic graph $G = (V, E, T)$, Initial node embeddings H_0, Number of time steps T

Output: Updated node embeddings H_t for each time step t

Initialize;
$t \leftarrow 1$;
$H \leftarrow H_0$;
while $t \leq T$ **do**
    // Update Node Embeddings;
    for *each node* v *in* V **do**
        Calculate the updated node embedding vector H_t^v using the previous embedding H_{t-1}^v, the neighborhood N_v, and the subgraph edges E_s;
        Update H_t^v in the node embeddings H;
    end
    // Learn and Update Policy;
    Train the policy based on the updated node embeddings H_t to obtain the probability distributions $P(V_s|H_t)$ and $P(E_s|H_t)$;
    // Sampling Decision;
    Sample nodes V_s from the probability distribution $P(V_s|H_t)$;
    Sample edges E_s from the probability distribution $P(E_s|H_t)$;
    // Subgraph Extraction;
    Extract the subgraph $G_s = (V_s, E_s)$ from the dynamic graph G based on the sampled nodes and edges;
    $t \leftarrow t + 1$;
end
Output the updated node embeddings H_t for each time step t;

3.2. Dynamic Graph Transformer

In this section, we introduce the Dynamic Graph Transformer, a variant of the Graph Transformer that is specifically designed to model dynamic or temporal graphs. The Dynamic Graph Transformer extends the traditional Graph Transformer

by incorporating temporal information and capturing temporal dependencies within the graph structure. The Dynamic Graph Transformer takes the token from adaptive sampling and global nodes to learn the embedding.

3.2.1. Temporal Sequence Input

Consider a temporal sequence S as a result of sampling by the adaptive sampler, super-nodes from graph coarse, and the global nodes. The global nodes help the model capture better the long-range dependencies. This sequence captures the dynamic nature of the graph, allowing us to track nodes and their interactions over time.

3.2.2. Embedding Update Process

The update process involves several key steps:

1. Node Embedding Initialization: Each node v_i is assigned an initial embedding $H_i^{(0)}$.

2. Positional Encoding: To incorporate temporal information, positional encodings are added to the node embeddings. These encodings allow the model to differentiate between nodes based on their positions in the sequence. The positional encoding $PE_{i,t}$ for node v_i at time step t can be calculated as follows:

$$PE_{i,t} = \begin{bmatrix} \sin(t/\tau) \\ \cos(t/\tau) \end{bmatrix}$$

   Here, τ is a hyperparameter controlling the scale of the positional encoding.

3. Multi-Head Self-Attention: The embeddings $H_i^{(0)}$ are transformed using multi-head self-attention mechanisms. This allows nodes to attend to other nodes within their temporal context. The attended embeddings are given by:

$$H_i^{(att)} = \text{MultiHeadAttention}(H_i^{(0)} + PE_{i,t})$$

4. Temporal Aggregation: The attended embeddings are then aggregated across the temporal dimension. This aggregation process captures the evolving graph structure by considering the interactions of nodes across different time steps. The aggregated embedding $H_i^{(agg)}$ can be computed as:

$$H_i^{(agg)} = \sum_{t=1}^{T} H_i^{(att)} \cdot W_t$$

where W_t is a learnable weight matrix for time step t.

5. Feedforward Layers: The aggregated embeddings are then passed through feedforward neural layers to capture complex relationships and further refine node representations:

$$H_i^{(out)} = \text{Feedforward}(H_i^{(agg)})$$

6. Temporal Update: The final updated node embedding $H_i^{(out)}$ encapsulates both the inherent attributes of the node and its evolving interactions within the dynamic graph.

3.2.3. Advantages and Limitations

The ASDGT offers several advantages in modeling dynamic graphs:

Temporal Dependency Modeling: ASDGT excels in capturing temporal dependencies within the graph structure, allowing it to effectively model dynamic or evolving graphs. By considering the temporal dimension of the graph, the ASDGT can capture how the relationships between nodes evolve over time. It leverages self-attention and multi-head attention mechanisms to learn the importance of different nodes and their interactions at different time steps. This enables the ASDGT to adapt its representations to changes in the graph structure over time, ensuring that the learned node embeddings capture both the static and dynamic aspects of the graph. Through graph convolution operations performed over time, the ASDGT can effectively encode the temporal evolution of node representations, capturing how the influence of nodes and their connections change over time. This temporal dependency modeling allows the

DGT to handle dynamic or evolving graphs and capture the inherent dynamics present in real-world systems, such as social networks, financial transactions, or biological processes.

Flexible Graph Representation: The ASDGT harnesses the power of self-attention and multi-head attention mechanisms to capture intricate relationships and dependencies between nodes, providing a flexible and adaptable representation of the graph. These mechanisms enable the model to assign varying levels of importance to different nodes based on their contextual relevance and capture complex patterns of interactions within the graph. By leveraging these attention mechanisms, the ASDGT can learn a rich and expressive representation that encapsulates the diverse and nuanced aspects of the graph structure. This flexibility in graph representation empowers the model to effectively handle diverse graph types and capture the intricacies of real-world systems, such as social networks, knowledge graphs, or biological networks.

Scalability: An important advantage of the ASDGT is its ability to efficiently model large dynamic graphs that consist of a significant number of nodes and time steps. The architecture of the model is designed in such a way that it can handle the computational challenges associated with large-scale graphs. By leveraging optimized algorithms and parallel processing techniques, the ASDGT can effectively process and analyze the complex connectivity patterns of large graphs, ensuring scalability without compromising performance. This scalability feature enables the model to be applied to various domains and real-world applications where the graphs are extensive and evolve over time, such as social media networks, online recommendation systems, or financial transaction networks. ASDGT's efficient handling of large dynamic graphs makes it a valuable tool for researchers and practitioners to gain insights and make informed decisions in complex networked systems.

However, it is important to consider the limitations of the Dynamic Graph Transformer:

- Computational Complexity: The ASGT may introduce additional computational complexity, especially when modeling large dynamic graphs with a high number of time steps.
- Data Availability: The effectiveness of the ASDGT relies on the availability of

temporal graph data and labeled examples for training.

In the next section, we will discuss the experimental setup and evaluation of ASDGT on various dynamic graph datasets.

3.3. Optimization and Inference

3.3.1. Node classification

In the training and inference, we sample S input sequences for each center node and use the center node representation from the final Transformer layer $h_c^{(s)}$ for prediction. Note that the computational complexity is controllable by choosing a suitable number of sampled nodes. A MLP (Multi-Layer Perceptron) is used to predict the node class:

$$\widetilde{\boldsymbol{y}}^{(s)} = f_{MLP}\left(\boldsymbol{h}_{c(t)}^{(s)}\right) \tag{3.4}$$

where $\widetilde{\boldsymbol{y}}_c \in R^{C\times 1}$ stands for the classification result, C stands for the number of classes. In the training process, we optimize the average cross entropy loss of labeled training nodes V_L :

$$\mathcal{L} = -\frac{1}{S}\sum_{v_i\in V_L}\sum_{s=1}^{S}\boldsymbol{y}_i^T \log \widetilde{\boldsymbol{y}}_i^{(s)}, \tag{3.5}$$

where $\boldsymbol{y}_i \in R^{C\times 1}$ is the ground truth label of center node v_i. In the inference stage, we take a bagging aggregation to improve accuracy and reduce variance:

$$\widetilde{\boldsymbol{y}}_i = \frac{1}{S}\sum_{s=1}^{S}\widetilde{\boldsymbol{y}}_i^{(s)} \tag{3.6}$$

3.3.2. Link Prediction

In the task of time-aware link prediction, we aim to predict the existence or absence of links between pairs of nodes in a temporal network. To achieve this, we employ a time-aware graph neural network model.

In both the training and inference phases in the link prediction task as (u, v), we also sample S nodes from center u and v and utilize the representations of the nodes from

the final Transformer layer, denoted as $h_{u(t)}^{(s)}$ and $h_{v(t)}^{(s)}$, respectively. To make predictions, we concatenate these representations and pass the resulting vector through a Multi-Layer Perceptron (MLP):

$$\widetilde{\boldsymbol{y}}^{(s)} = f_{\text{MLP}}([\boldsymbol{h}_{u(t)}^{(s)}, \boldsymbol{h}_{v(t)}^{(s)}]) \tag{3.7}$$

Here, $\widetilde{\boldsymbol{y}}_{ij} \in R^{C\times 1}$ represents the classification result for the link between nodes v_i and v_j at time step t, where C is the number of classes. During the training process, we optimize the average cross-entropy loss for the labeled training links E_L:

$$\mathcal{L} = -\frac{1}{S} \sum_{(v_i,v_j)\in E_L} \sum_{s=1}^{S} \boldsymbol{y}_{ij}^{T} \log \widetilde{\boldsymbol{y}}_{ij}^{(s)} \tag{3.8}$$

Here, $\boldsymbol{y}_{ij} \in R^{C\times 1}$ represents the ground truth label of the link between nodes v_i and v_j.

3.4. Regret Analysis

In the context of the bandit-based adaptive sampling approach, regret analysis provides a theoretical framework to quantify the performance of the algorithm by measuring the cumulative difference between the rewards obtained and the rewards that would have been achieved by always selecting the best possible nodes for labeling. The regret analysis allows us to understand how well the algorithm performs compared to an ideal oracle that knows the true labels of all nodes in the graph.

The intuition behind regret analysis is to compare the performance of the bandit-based adaptive sampling algorithm with an optimal strategy. The optimal strategy is defined as always selecting the nodes that would maximize the cumulative rewards if we had perfect knowledge of the true labels. The regret is then calculated as the difference between the cumulative rewards obtained by the algorithm and the rewards that would have been achieved by the optimal strategy.

Let $R(t)$ denote the cumulative reward obtained by the algorithm up to time step t, and $R^*(t)$ denote the cumulative reward that would have been achieved by the optimal strategy up to time step t. The regret $L(t)$ at time step t is defined as:

$$L(t) = R^*(t) - R(t) \tag{3.9}$$

The goal of the regret analysis is to bound the cumulative regret $L(T)$ for a given time horizon T. By bounding the regret, we can assess the performance of the bandit-based adaptive sampling algorithm and analyze its efficiency in selecting informative nodes.

Proof

The proof of regret bounds typically involves analyzing the exploration and exploitation trade-off of the bandit-based adaptive sampling algorithm. It typically involves establishing upper bounds on the expected rewards obtained by the algorithm and lower bounds on the rewards that would have been achieved by the optimal strategy.

The proof starts by considering the expected reward obtained by the algorithm at each time step t. Let $\mathbb{E}[R(t)]$ denote the expected reward at time step t, and $\mathbb{E}[R^*(t)]$ denote the expected reward that would have been achieved by the optimal strategy at time step t. The regret at time step t, denoted as $L(t)$, can be defined as:

$$L(t) = \mathbb{E}[R^*(t)] - \mathbb{E}[R(t)] \tag{3.10}$$

To analyze the regret bounds, we need to consider the exploration and exploitation phases of the algorithm separately.

During the exploration phase, the algorithm explores different nodes to gather information about their labels. Let $N(t)$ denote the set of nodes selected for labeling up to time step t. The expected reward obtained during the exploration phase can be bounded as:

$$\mathbb{E}[R(t)] \leq \sum_{n \in N(t)} \mathbb{E}[r_n(t)] \tag{3.11}$$

where $E[r_n(t)]$ is the expected reward obtained by selecting node n at time step t. The exploration phase aims to gather sufficient information to estimate the true labels of the nodes accurately.

During the exploitation phase, the algorithm exploits the gathered knowledge to select the most informative nodes. Let $S(t)$ denote the set of nodes selected for labeling during the exploitation phase up to time step t. The expected reward obtained during the exploitation phase can be bounded as:

$$\mathbb{E}[R(t)] \geq \sum_{n \in S(t)} \mathbb{E}[r_n(t)] \tag{3.12}$$

where $\mathbb{E}[r_n(t)]$ is the expected reward obtained by selecting node n at time step t. The exploitation phase aims to maximize the cumulative rewards by selecting nodes with high expected rewards based on the available information.

The difference between the expected rewards obtained by the algorithm and the expected rewards achieved by the optimal strategy is:

$$\Delta = \mathbb{E}[R(a)] - \mathbb{E}[R(a^*)] \tag{3.13}$$

By applying concentration inequalities, we can derive high probability bounds on the regret.

$$\text{Regret} \leq C\sqrt{2T\log(K)} \tag{3.14}$$

The specific steps of the proof depend on the characteristics of the bandit algorithm, the exploration-exploitation strategy employed, and the assumptions made about the underlying graph structure. The proof typically involves analyzing the exploration phase's information-gathering ability and the exploitation phase's ability to select informative nodes.

The proof establishes that the cumulative regret $L(T)$ grows sublinearly with respect to the time horizon T and the properties of the graph. This sublinear growth indicates that as the number of time steps increases, the algorithm's performance approaches that of the optimal strategy, demonstrating its effectiveness in selecting informative nodes for labeling.

Chapter 4

Experiments

This chapter represents the experimental setup for evaluating the performance of the proposed method to compare with other popular baselines.

4.1. Experimental setup

4.1.1. Dataset

For our experimental analysis, we have selected a comprehensive set of datasets that encompass a wide range of temporal dynamics across diverse domains. Our dataset collection includes the Wikipedia temporal dataset, which captures the temporal evolution of articles; the Reddit temporal dataset [27], enabling the study of temporal patterns in online discussions; the ENRON temporal dataset [37], providing insights into communication dynamics within a corporate environment; the UCI temporal dataset [38], encompassing various domains and offering temporal information for research purposes; and the MOOC temporal dataset [27], capturing temporal aspects of Massive Open Online Courses, such as enrollment, participation, and learning behaviors over time. This diverse selection of datasets allows us to explore temporal phenomena, understand social dynamics, and analyze temporal patterns across multiple contexts, providing valuable resources for our experimental investigations.

- Wikipedia [27]: The Wikipedia temporal dataset offers a valuable resource for

conducting research on the temporal dynamics of Wikipedia articles. This dataset captures the changes made to articles over time, providing detailed information about edits, revisions, and associated timestamps. It consists of edits on Wikipedia pages over one month. Editors and wiki pages are modeled as nodes, and the timestamped posting requests are edges.

- Reddit [27]: The Reddit temporal dataset serves as a valuable resource for studying the temporal dynamics of discussions and user interactions on the popular online forum. This dataset captures the evolution of Reddit threads over time, providing researchers with detailed information about post edits, comments, and associated timestamps. It contains subreddits posted spanning one month, where the nodes are users or posts and the edges are the timestamped posting requests.

- SocialEvo [39]: The SocialEvo temporal dataset is a valuable resource for studying the temporal dynamics of social networks and human interactions in online platforms. This dataset captures the evolution of various social media platforms from a mobile phone proximity network, such as Twitter or Facebook, over time, providing researchers with detailed information about user activities, interactions, and associated timestamps. Key aspects of the SocialEvo temporal dataset include user profiles, posts, comments, likes, shares, and other forms of social engagement. Additionally, metadata such as timestamps, user demographics, and network structures may be included, enabling researchers to analyze temporal patterns, community dynamics, information diffusion, sentiment analysis, and other aspects of social behavior over time.

- ENRON [37]: The ENRON temporal dataset is a valuable resource for studying the temporal dynamics of communication within the ENRON corporation, which was one of the largest energy companies in the United States. This dataset captures the email communications exchanged among ENRON employees over a specific period. It provides researchers with detailed information about the sender, recipient, timestamps, and content of each email. The ENRON temporal dataset offers a unique opportunity to analyze the patterns of communication, identify key individuals or groups, study information flow, and investigate temporal aspects such as email activity over time.

- UCI [38]: The UCI temporal dataset refers to a collection of temporal data as a Facebook-like network, made available by the University of California, Irvine (UCI). UCI is known for providing a vast array of datasets for research and educational purposes across various domains. The UCI temporal dataset encompasses diverse domains such as machine learning, data mining, social sciences, and more. Each dataset within the UCI temporal collection contains temporal information, capturing the evolution of data over time. These datasets often include attributes such as timestamps, sequential data, or time-series observations. Researchers can leverage the UCI temporal dataset to explore temporal patterns, conduct longitudinal analyses, develop predictive models, and investigate other time-dependent phenomena.

- MOOC [27]: The MOOC temporal dataset refers to a collection of data capturing the temporal aspects of Massive Open Online Courses (MOOCs). MOOCs are online educational platforms that offer courses to a large number of participants from diverse backgrounds. The MOOC temporal dataset typically includes information about course enrollment, participation, and interactions over time. It may encompass data such as timestamps of student dropout. Each edge is a student accessing a content unit and has 4 features.

The detailed statistics of each dataset are described in Table 1. We evaluate the efficiency of our model output embedding in both transductive and inductive settings.

More specifically, we split the data by time for training, validating, and testing. We use the first 70% interaction to train, the next 15% to evaluate, and the final 15% to test. For example, on Reddit dataset consists of four weeks of posts created by users on subreddits, in a week the models take the first 5 days of data of the week to train, the next day to evaluate, and the last day to test. The fixed evaluation period is selected at a week duration. Because our proposed model can learn continuously, and the duration could be changed freely.

4.1.2. Baselines

We compare our method against a variety of strong baselines on the task of edge prediction and node classification on dynamic graphs. These baselines can be

Table 4.1: The detail of the datasets used in our experiments.

Dataset	Wikipedia	Reddit	SocialEvo	ENRON	UCI	MOOC
Edges	157474	672447	352180	2099520	59835	411749
Nodes	9227	10984	74	184	1899	7145
Edge feature dim	172	172	0	0	0	4
Timespan	30 days	30 days	30 days	1316 days	193days	30 days

categorized into two groups. (1) Continuous-based methods: TGAT [29], JODIE [27], TGN [30], and DyRep [28]. (2) Snapshot-based methods: DynAERNN [23], VGRNN [25], EvolveGCN [26], DySAT [24], and ASTGN [40].

- DynAERNN [23]: This model focuses on learning the temporal transitions within the network by employing a deep architecture consisting of both dense and recurrent layers.
- VGRNN [25]: A variational model that introduces extra latent random variables. These variables work collaboratively to model the hidden states within a graph recurrent neural network. This approach effectively captures changes in both the topology and node attributes in dynamic graphs.
- EvolveGCN [26]: This model adeptly captures the dynamic nature of graph sequences by employing an RNN to evolve the parameters of GCN. It achieves this without the need for traditional node embedding approaches, effectively adapting the GCN model to the temporal dimension.
- DySAT [24]: This model computes dynamic embeddings by employing structural attention layers on each snapshot, followed by temporal attention layers to capture temporal variations among snapshots, as inspired by the self-attention mechanism.
- DyRep [28]: This model takes a joint approach to learning the dynamic changes in network topology alongside the activities between nodes. It achieves this by updating the representation of node v_i following an event involving v_i.
- JODIE [27]: Utilizing an RNN model to continually refresh the node memory of interconnected nodes. Additionally, it incorporates an attention architecture to extract information from the node memory and produce node embedding vectors.

- TGAT [29]: This method involved employing the self-attention mechanism as a foundational building block and introducing a novel functional time encoding technique.
- TGN [30]: Introducing a novel model that synergizes memory modules with graph-based operators. This model demonstrated remarkable performance improvements over its predecessors.
- ASTGN [40]: This method employs an adaptive sampling for link prediction on graph snapshots.

4.1.3. Setting of ASDGT and baselines

ASDGT is implemented by the PyTorch framework, and the codes for the baselines are published by the author on GitHub. We conduct experiments on a GPU RTX 3090 with all baselines and our model with 10 random seeds. The models are training with Adam Optimizer. We select the search space of hyper-parameters as described in Table 4.2. .

Table 4.2: Hyperparameter used in ASDGT and the baselines

Hyperparameter	Values
Learning rate	{0.01, 0.001, 0.0001, 0.0005}
Dropout	{0.1, 0.2, 0.3, 0.4, 0.5}
Batch size	{256, 512, 1024}
Number epochs	$\leq$ 1000
Attention head H	8
Number of layer L	{2, 3, 4, 5,5 6}
Number sample nodes	{5, 10, 15, 20, 25}
Number of global nodes	{1, 2, 3}
Number of super-nodes	{0, 3, 6, 9}

Evaluation Metrics

In evaluating the performance of our node classification model, we utilize the accuracy metric. Accuracy measures the proportion of correctly classified nodes and provides an

overall assessment of the model's ability to classify instances correctly.

For node classification with N as the total number of nodes in the dataset. We assign a predicted label $\hat{y}_i$) to each node i based on its features and the learned model. The true label of node i is denoted as y_i. The accuracy is then calculated as:

$$\text{Accuracy} = \frac{1}{N} \sum_{i=1}^{N} \mathbb{1}(\hat{y}_i = y_i) \tag{4.1}$$

where $\mathbb{1}(\hat{y}_i = y_i)$ is an indicator function that evaluates to 1 if $\hat{y}_i$ is equal to y_i, and 0 otherwise. By considering the accuracy metric, we can assess the node classification model's performance in terms of correctly predicting the labels of nodes. A higher accuracy indicates a better classification performance, while a lower accuracy suggests the model's limitations in correctly assigning labels to nodes.

In evaluating the performance of our link prediction model, we utilize multiple metrics, including AUC (Area Under the ROC Curve) and Average Precision (AP). AUC measures the model's ability to rank positive and negative links correctly, regardless of the threshold used to classify the links. Additionally, AP is a widely used metric for link prediction. AP measures the precision-recall trade-off by calculating the average precision across all possible recall levels.

For link prediction, let M represent the total number of link predictions made by the model. We assign a predicted label $\hat{y}_{ij}$ (1 for the presence of a link, 0 for the absence of a link) to each link prediction e_{ij} based on the learned model. The true label of the link prediction e_{ij} is denoted as y_{ij} (1 for the presence of a link, 0 for the absence of a link).

To calculate AUC, we first construct the Receiver Operating Characteristic (ROC) curve by varying the classification threshold and plotting the True Positive Rate (TPR) against the False Positive Rate (FPR). TPR represents the proportion of correctly predicted positive links out of all actual positive links, and FPR represents the proportion of incorrectly predicted positive links out of all actual negative links. AUC is then calculated as the area under the ROC curve:

$$\text{AUC} = \int_0^1 \text{TPR}(\text{FPR}^{-1}(t)), dt \tag{4.2}$$

To calculate Average Precision (AP), we first compute precision and recall values at different classification thresholds. Precision (P) is the proportion of correctly predicted positive links out of all predicted positive links and recall (R) is the proportion of correctly predicted positive links out of all actual positive links. The precision-recall curve is then constructed by varying the classification threshold, and AP is calculated as the area under the precision-recall curve:

$$\text{AP} = \sum_{k=1}^{n} (R_k - R_{k-1}) \times P_k \tag{4.3}$$

where n is the number of points on the precision-recall curve, and R_k and P_k are the recall and precision values at the k-th point on the curve, respectively.

By considering AUC and AP as evaluation metrics, we can assess the link prediction model's performance in terms of correctly predicted links and the precision-recall trade-off. Accuracy provides an overall measure of correct link predictions, while AP offers a more nuanced evaluation capturing the quality of the predicted links across different recall levels.

4.2. Results

4.2.1. Node classification task

Table 4.3 shows the performances of our proposed ASDGT model alongside baseline methods for node future classification. The results underscore the substantial performance advantage of our model over the baselines. This superiority serves as a testament to the Graph Transformer's adeptness at unraveling intricate long-range dependencies, especially when complemented by the adaptive sampling strategy. The confluence of the Graph Transformer's capabilities and adaptive sampling's informed node selection amplifies the model's proficiency in capturing temporal nuances and contextual cues, ultimately leading to exceptional node future classification accuracy. This reinforcement not only attests to the efficacy of our approach but also highlights the inherent synergy between the model's architecture and the sampling mechanism, further advancing the field of dynamic graph representation learning.

Table 4.3: The performance of our model (Accuracy - %) and baselines on node classification task

Method	Model	MOOC	Wikipedia	Reddit
Discrete-time	EvolveGCN [2020]	70.26 ± 0.5	63.41 ± 0.3	81.77 ± 1.2
	DySAT [2019]	72.11 ± 0.5	61.79 ± 0.3	74.82 ± 1.2
Continuous-time	Jodie [2019]	73.39 ± 2.1	61.23 ± 2.5	84.35 ± 1.2
	TGAT [2020]	74.23 ± 1.2	65.43 ± 0.7	83.12 ± 0.7
	DyRep [2020]	75.12 ± 0.7	62.79 ± 2.3	84.82 ± 2.2
	TGN [2020]	77.47 ± 0.8	67.11 ± 0.9	87.41 ± 0.3
	ASDGT (ours)	78.13 ± 0.9	69.74 ± 1.3	89.03 ± 0.3

4.2.2. Link prediction task

Table 4.4: The performance of our proposed model and baselines on link prediction task of Wikipedia, Reddit, and MOOC dataset.

Dataset	Wikipedia	Reddit	MOOC
Metric	AUC (%) ↑	AUC (%) ↑	AUC (%) ↑
DynAERNN [2020]	71.00 ± 1.1	83.37 ± 1.4	89.34 ± 0.2
JODIE [2019]	88.43 ± 0.7	87.71 ± 0.4	90.50 ± 0.1
DyRep [2019]	77.40 ± 1.2	67.36 ± 1.1	90.49 ± 0.3
VGRNN [2019]	72.20 ± 0.6	51.89 ± 0.2	90.03 ± 0.4
EvolveGCN [2020]	60.48 ± 0.5	58.42 ± 0.4	50.36 ± 0.8
TGAT [2020]	95.47 ± 0.1	96.65 ± 0.1	72.09 ± 0.3
TGN-attn [2020]	95.72 ± 2.1	96.12 ± 1.1	81.64 ± 1.3
ASTGN [2022]	96.92 ±0.2	98.97 ± 0.1	92.75 ± 0.8
ASDGT (ours)	97.45 ± 0.3	99.02 ± 0.1	93.23 ± 0.4

Tables 4.4 and 4.5 present a comprehensive overview of the results attained from our proposed model ASDGT, juxtaposed with the performance exhibited by various baseline methods, in the domain of temporal link prediction. A thorough examination of the comparative metrics reveals a consistent trend where our model significantly outperforms the baseline alternatives by a considerable margin. Notably, the advancements introduced by ASDGT coupled with the integration of adaptive

Table 4.5: The performance of our proposed model and baselines of link prediction task of UCI, SocialEvo, and Enron dataset.

Dataset	UCI	SocialEvo	Enron
Metric	AP (%) ↑	AP (%) ↑	AP (%) ↑
EvolveGCN [2020]	76.63 ± 0.2	56.90 ± 0.6	57.37 ± 0.2
TGAT [2020]	60.25 ± 0.2	57.37 ± 0.6	60.36 ± 0.7
TGN-attn [2020]	64.21 ± 0.2	58.11 ± 1.2	62.47 ± 1.7
JODIE [2019]	78.02 ± 0.2	60.01 ± 1.1	63.10 ± 1.3
DyRep [2019]	79.76 ± 0.1	62.02 ± 1.7	67.28 ± 1.3
ASTGN [2022]	81.52 ± 0.2	62.41 ± 1.0	69.12 ± 1.0
ASDGT (ours)	82.34 ± 0.2	65.12 ± 0.3	71.33 ± 0.5

sampling manifest as a driving factor behind the observed performance boost. It is particularly intriguing to note that even in comparison to the highly adept temporal attention model ASTGN, our model showcases a notable edge, carving out a performance improvement in the range of $1-3\%$. This trend underscores the robustness and efficacy of our adaptive sampling strategy, which seamlessly augments the prowess of the Dynamic Graph Transformer across diverse datasets.

These findings echo the core tenets of our proposed methodology, wherein adaptive sampling not only lends itself to computational efficiency but also significantly enriches the model's predictive capabilities. This augmentation stands as a testament to the versatility and adaptability of our approach across a spectrum of datasets. Furthermore, the consistent performance enhancement across varying datasets accentuates the inherent robustness of our model and adaptive sampling scheme. It is evident that the symbiotic relationship between ASDGT and adaptive sampling yields improvements that transcend dataset-specific characteristics, offering a promising avenue for application in dynamic graph scenarios. In essence, these results highlight the symbiotic synergy of our proposed ASDGT model and the adaptive sampling mechanism, which collectively spearhead advancements in dynamic graph representation learning and temporal link prediction

4.3. Ablation study

4.3.1. Effectiveness of graph coarsening algorithm

In order to comprehensively explore the influence of the number of sampled nodes on each target node within dynamic graphs, we conducted an experimental analysis across three distinct datasets: Wikipedia, UCI, and EnRon. The results of this investigation are depicted in Figure 4.1. Notably, we observed that the choice of the number of sampled nodes significantly affects the performance of our proposed method. Strikingly, for all three datasets, the AUC/AP values exhibited a prominent trend toward optimization when the number of sampled nodes was set at 10. This suggests a critical point at which the trade-off between effective information utilization and computational efficiency is most favorable. While smaller numbers of sampled nodes led to diminished performance due to inadequate information capture, larger numbers introduced complexities that resulted in only marginal improvements. This consistent trend across diverse datasets underscores the generalizability of the observation and points to 10 sampled nodes as a strong candidate for achieving optimal representation learning outcomes.

In the ASDGT model, we use graph coarsen to compact the large graph into smaller ones, which helps the transformers can be easily adapt with limited computation resources. Here, we evaluate ANS-GT with different coarsening algorithms and different coarsening rates. Specifically, the considered coarsening algorithms include Variation Neighborhoods (VN) [9], Variation Edges (VE) [9], and Algebraic (JC) [10]. We vary the coarsening rate from 0.01 to 0.50. In Table 4.6 we show the performances of these coarsening methods on the node classification task (Wikipedia) with AUC and the link prediction task (Enron) with AP. Note that the value $c = 1.0$ denotes the use of a full graph. We can see that the graph coarsening is helpful to ASDGT in learning efficient representation, it can be demonstrated by the result of link prediction and node classification.

4.3.2. Effectiveness of adaptive sampling

In order to comprehensively explore the influence of the number of sampled nodes on each target node within dynamic graphs, we conducted an experimental analysis across

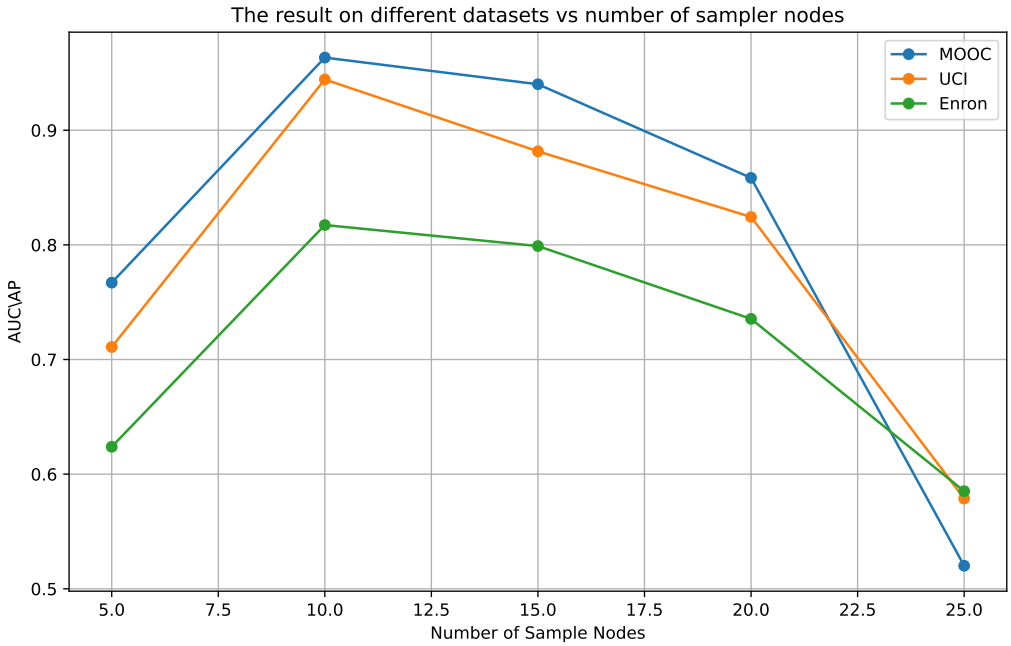

Figure 4.1: The performance of ASDGT with the number of sampled nodes

Table 4.6: Sensitivity analysis of coarsening algorithms and coarsening rate on the node classification task (Wikipedia) and the link prediction task (Enron) without adaptive sampling.

Dataset	Method	$c = 0.01$	$c = 0.05$	$c = 0.10$	$c = 0.50$	$c = 1.00$
Wikipedia	VN	88.60	92.23	88.14	91.01	93.26
	VE	87.95	88.13	88.30	87.32	87.22
	JC	88.49	88.20	87.46	87.36	89.14
Enron	VN	70.72	69.45	70.10	68.83	69.08
	VE	69.20	69.66	67.51	68.94	68.06
	JC	69.15	69.85	69.92	70.16	69.09

three distinct datasets: Wikipedia, UCI, and EnRon. The results of this investigation are depicted in Figure 4.1. Notably, we observed that the choice of the number of sampled nodes significantly affects the performance of our proposed method. Strikingly, for all three datasets, the AUC/AP values exhibited a prominent trend toward optimization when the number of sampled nodes was set at 10. This suggests a critical point at which

the trade-off between effective information utilization and computational efficiency is most favorable. While smaller numbers of sampled nodes led to diminished performance due to inadequate information capture, larger numbers introduced complexities that resulted in only marginal improvements. This consistent trend across diverse datasets underscores the generalizability of the observation and points to 10 sampled nodes as a strong candidate for achieving optimal representation learning outcomes.

Furthermore, our analysis extends to a comprehensive comparison of the effectiveness of adaptive sampling on ASDGT when juxtaposed with two alternative training strategies: the utilization of 1-hop neighbor sampling and the original training on the full graph. This investigation is conducted with a focus on evaluating both time and memory complexities, critical considerations in real-world applications. In this context, it's imperative to highlight that our decision not to include a comparison with other existing sampling methods is rooted in the unique nuances of our proposed approach. The examined sampling methods, often tailored to graph neural networks and predominantly static settings, possess divergent objectives that make direct comparisons inappropriate within the scope of this study. As such, we refrain from drawing comparisons that may lead to unfounded conclusions or skew the evaluation process. Instead, our focus remains on illuminating the potency of adaptive sampling in the dynamic graph scenario, offering an unbiased analysis that prioritizes fairness and comprehensiveness in the assessment of training strategies

An intriguing aspect of our analysis involves a direct comparison between 1-hop neighbor sampling and the proposed adaptive sampling strategy. Figures 4.2 and 4.3 provide illuminating insights into the effectiveness of both approaches. As depicted in Figure 4.2, the adaptive sampling consistently outperforms the 1-hop neighbor sampling in terms of AUC values across various iterations. This trend accentuates the ability of adaptive sampling to capture more informative and discriminative interactions, thereby enhancing the model's classification prowess. Figure 4.3 further underscores the advantages of adaptive sampling. Remarkably, adaptive sampling not only showcases superior performance but also yields a substantial reduction in the number of sampled nodes compared to 1-hop neighbor sampling. This reduction amounts to approximately four times fewer sampled nodes, a remarkable feat that can significantly alleviate computational burdens without compromising model accuracy.

These findings collectively advocate for the adoption of the adaptive sampling strategy as a judicious choice, offering dual benefits in terms of computational efficiency and predictive performance.

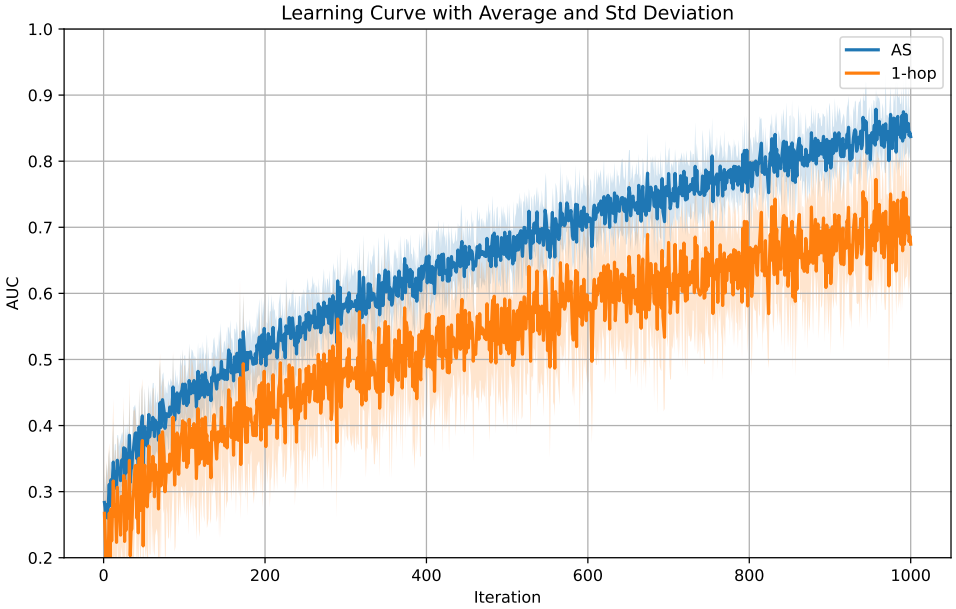

Figure 4.2: The training AUC curves of adaptive sampling and 1-hop neighbor on MOOC node classification. The results are conducted with 10 random seeds and take the average and std that are plotted in this figure.

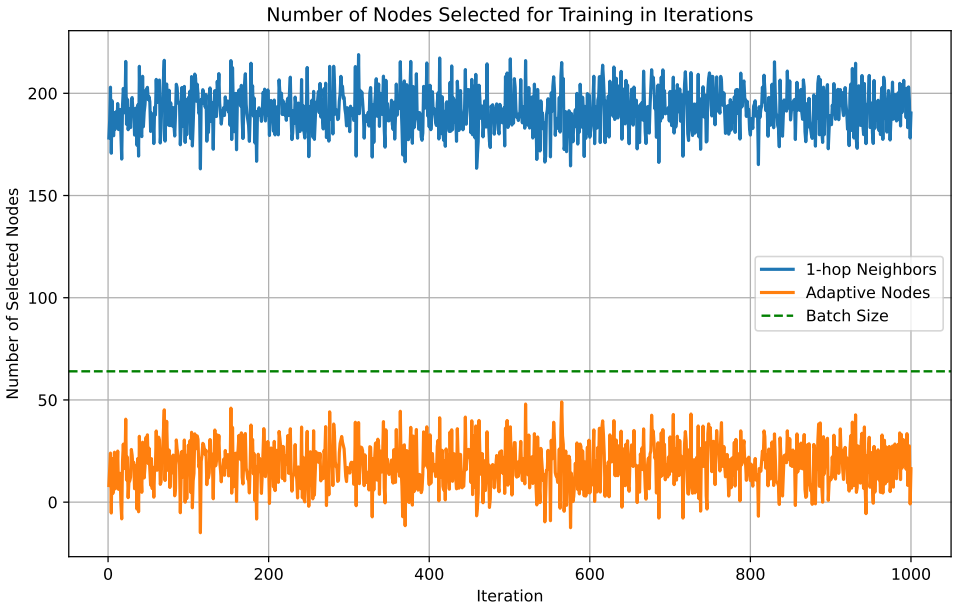

Figure 4.3: The number nodes are sampled during the training with MOOC dataset

To assess the computational efficiency of these models, we measured and compared the time required for training on four diverse datasets. The table 4.7 provides a comparison of computational times for training various models on different datasets, with the goal of understanding their efficiency. The models are evaluated on four different datasets: MOOC, UCI, Wikipedia, and Reddit. Among the models, ASDGT stands out as the most computationally efficient, consistently outperforming the other models across all datasets. For example, on the MOOC dataset, ASDGT achieves a remarkable training time of seconds per batch, significantly faster than other models such as DyRep, TGAT, and TGN-attn. Similarly, on the UCI, Wikipedia, and Reddit datasets, ASDGT consistently maintains its efficiency advantage. It can be see that ASDGT demonstrates superior computational efficiency compared to the baseline models, making it a promising choice for training graph-based models, particularly in scenarios where computational resources are limited or efficiency is a critical factor.

Table 4.7: The comparison of the computational time for training of the proposed model and baseline. The average time is reported per batch with lower is better.

Model	MOOC	UCI	Wikipedia	Reddit
$\|V\|$	7,144	1,899	9,227	10,984
$\|E_{temp}\|$	411,749	59,835	157,474	672,447
DyRep [2019]	227.9	205.2	217.3	222.6
TGAT [2020]	229.8	220.4	236.5	237.0
TGN-attn [2020]	242.2	225.3	260.2	250.8
JODIE [2020]	182.1	178.5	187.3	191.1
ASTGN [2022]	–	158.1	190.7	212.4
ASDGT (ours)	92.1	80.2	102.4	120.3

4.3.3. Effectiveness of ASDGT in long period

Time Projection: Our proposed model projects the embedding to capture temporal information and predicts the future embedding at a time. After a short duration Δ_t the node i's projected embedding is updated to as follow:

$$h_{i(t+\Delta t)} = (1 + w) * h_{i(t)} \tag{4.4}$$

where w is time-context vector is converted from Δt by using a linear layer: $w = W_p \Delta t$. The vector $(1 + w)$ works as a temporal attention vector to scale the past node embedding.

We test the accuracy of our proposed model by varying the time projecting window Δt. The node classification task results on the Reddit dataset of our model and other baselines are shown in Table 4.8. In general, it is more difficult to predict for a long period updating time Δt than the short one. While all of the tested models drop accuracy, our model still achieves the best accuracies. At the longest $\Delta t = 7$, the proposed ASDGT achieves around 85.36% accuracy. The second highest accuracy is the TGN with 82.53% accuracy. These outcomes reinforce our model's adaptability to extended time horizons, affirming its capability to navigate the complexities of temporal dynamics and offering promising insights into its applicability across real-world scenarios.

Table 4.8: The accuracy of node classification task on Reddit dataset by varying the time projection $\Delta t(days)$ of different models

Model	$\Delta t = 1$	$\Delta t = 3$	$\Delta t = 5$	$\Delta t = 7$
EvolveGCN [2020]	81.77 ± 1.2	70.39 ± 0.7	71.22 ± 0.5	74.07 ± 0.5
DySAT [2020]	82.32 ± 0.7	75.13 ± 0.5	74.05 ± 0.4	71.39 ± 0.5
Jodie [2019]	84.35 ± 1.2	81.71 ± 0.8	81.13 ± 0.5	79.38 ± 0.7
TGAT [2020]	83.12 ± 0.7	84.46 ± 0.5	83.18 ± 0.7	78.59 ± 1.2
DyRep [2019]	84.82 ± 2.2	80.33 ± 0.5	81.05 ± 0.5	79.77 ± 1.1
TGN [2020]	87.41 ± 0.3	87.58 ± 0.5	86.11 ± 0.3	82.53 ± 0.5
ASDGT (ours)	89.03 ± 0.3	88.11 ± 0.2	86.43 ± 0.5	85.36 ± 0.7

Chapter 5

Conclusion

In this thesis, we investigate adaptive sampling strategies for enhancing representation learning within the context of dynamic graphs using graph transformers. Our investigation has shed light on the critical role of adaptively selecting nodes during training to capture both short- and long-range dependencies, thereby improving the model's efficacy in capturing dynamic interactions. Through comprehensive analysis and experimentation, we've showcased the potential of our proposed adaptive sampling technique to yield significant enhancements in performance, resulting in improved accuracy and convergence. This research marks a substantial stride forward in the field of representation learning, addressing the challenges posed by dynamic graph data and offering new avenues for refining graph-based machine learning models.

Future Work: As this thesis opens a direction in the domain of adaptive sampling and dynamic graph modeling, several exciting avenues remain for further exploration and refinement. The following areas present promising directions for future research:

Robustness and Generalization: In this thesis, we design experiments that are slightly different from previous methods. In the future, we will evaluate other settings such as in the related works to confirm the robustness of our proposed method. While our approach has shown promising results across various datasets, further investigation is necessary to assess its robustness and generalization capabilities across diverse dynamic graphs. Adapting the technique to real-world scenarios, which often exhibit noise and uncertainty, could be a valuable avenue of exploration.

Fine-Tuning Sampling Strategies: Our current work has introduced an adaptive sampling approach, but further investigation into more sophisticated sampling strategies could offer even more significant improvements in capturing intricate graph dynamics. Exploring strategies that adaptively balance the sampling of different node types, such as hubs and peripheral nodes, could contribute to a richer understanding of complex interactions.

Temporal Variability: Our study has concentrated on the interactions of nodes in the dynamic graph, but the temporal variability of edges and nodes also warrants consideration. Investigating how the temporal evolution of individual nodes and edges affects their importance in adaptive sampling could lead to more accurate and nuanced representation learning.

Model Integration: Incorporating adaptive sampling into other graph-based models beyond the Graph Transformer architecture could provide insights into its broader applicability and potential improvements in various tasks. Integrating it into more complex graph neural networks or combining it with other graph mining techniques could lead to even more powerful models.

Scalability: As graph data continues to grow in scale, the scalability of adaptive sampling techniques becomes increasingly crucial. Exploring methods to optimize the computational efficiency of adaptive sampling could enable its application to larger and more complex dynamic graphs.

In conclusion, this thesis has illuminated the significance of adaptive sampling in dynamic graphs and its potential for enhancing representation learning using the Graph Transformer model. As we step into the future, the pursuit of the aforementioned research avenues promises to contribute to the advancement of adaptive sampling techniques, enriching our understanding of dynamic graph dynamics, and ultimately, bolstering the performance of graph-based machine learning models.

Bibliography

[1] W. Fan, Y. Ma, Q. Li, Y. He, E. Zhao, J. Tang, and D. Yin, "Graph neural networks for social recommendation," in *The world wide web conference*, pp. 417–426, 2019.

[2] X. He, K. Deng, X. Wang, Y. Li, Y. Zhang, and M. Wang, "Lightgcn: Simplifying and powering graph convolution network for recommendation," in *Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval*, pp. 639–648, 2020.

[3] R. Qiu, J. Li, Z. Huang, and H. Yin, "Rethinking the item order in session-based recommendation with graph neural networks," in *Proceedings of the 28th ACM international conference on information and knowledge management*, pp. 579–588, 2019.

[4] T. L. Hoang, T. D. Pham, and V. C. Ta, "Improving graph convolutional networks with transformer layer in social-based items recommendation," in *2021 13th International Conference on Knowledge and Systems Engineering (KSE)*, pp. 1–6, IEEE, 2021.

[5] L. Atlas, T. Homma, and R. Marks, "An artificial neural network for spatio-temporal bipolar patterns: Application to phoneme classification," in *Neural Information Processing Systems*, 1987.

[6] L. R. Medsker and L. Jain, "Recurrent neural networks," *Design and Applications*, vol. 5, no. 64-67, p. 2, 2001.

[7] V. P. Dwivedi and X. Bresson, "A generalization of transformer networks to graphs," *arXiv preprint arXiv:2012.09699*, 2020.

[8] T.-L. Hoang and V.-C. Ta, "Dynamic-gtn: Learning an node efficient embedding in dynamic graph with transformer," in *PRICAI 2022: Trends in Artificial Intelligence* (S. Khanna, J. Cao, Q. Bai, and G. Xu, eds.), (Cham), pp. 430–443, Springer Nature Switzerland, 2022.

[9] A. Loukas, "Graph reduction with spectral and cut guarantees.," *J. Mach. Learn. Res.*, vol. 20, no. 116, pp. 1–42, 2019.

[10] D. Ron, I. Safro, and A. Brandt, "Relaxation-based coarsening and multiscale graph organization," *Multiscale Modeling & Simulation*, vol. 9, no. 1, pp. 407–423, 2011.

[11] P. Cui, X. Wang, J. Pei, and W. Zhu, "A survey on network embedding," *IEEE transactions on knowledge and data engineering*, vol. 31, no. 5, pp. 833–852, 2018.

[12] W. L. Hamilton, R. Ying, and J. Leskovec, "Representation learning on graphs: Methods and applications," 2017.

[13] D. Zhang, J. Yin, X. Zhu, and C. Zhang, "Network representation learning: A survey," *IEEE transactions on Big Data*, vol. 6, no. 1, pp. 3–28, 2018.

[14] H. Cai, V. W. Zheng, and K. C.-C. Chang, "A comprehensive survey of graph embedding: Problems, techniques, and applications," *IEEE transactions on knowledge and data engineering*, vol. 30, no. 9, pp. 1616–1637, 2018.

[15] P. Goyal and E. Ferrara, "Graph embedding techniques, applications, and performance: A survey," *Knowledge-Based Systems*, vol. 151, pp. 78–94, 2018.

[16] B. Perozzi, R. Al-Rfou, and S. Skiena, "Deepwalk: Online learning of social representations," in *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 701–710, 2014.

[17] A. Grover and J. Leskovec, "node2vec: Scalable feature learning for networks," in *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 855–864, 2016.

[18] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei, "Line: Large-scale information network embedding," in *Proceedings of the 24th international conference on world wide web*, pp. 1067–1077, 2015.

[19] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," *Advances in neural information processing systems*, vol. 30, 2017.

[20] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Lio, Y. Bengio, *et al.*, "Graph attention networks," *stat*, vol. 1050, no. 20, pp. 10–48550, 2017.

[21] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[22] J. L. Ba, J. R. Kiros, and G. E. Hinton, "Layer normalization," *arXiv preprint arXiv:1607.06450*, 2016.

[23] P. Goyal, S. R. Chhetri, and A. Canedo, "dyngraph2vec: Capturing network dynamics using dynamic graph representation learning," *Knowledge-Based Systems*, vol. 187, p. 104816, 2020.

[24] A. Sankar, Y. Wu, L. Gou, W. Zhang, and H. Yang, "Dysat: Deep neural representation learning on dynamic graphs via self-attention networks," in *Proceedings of the 13th international conference on web search and data mining*, pp. 519–527, 2020.

[25] E. Hajiramezanali, A. Hasanzadeh, K. Narayanan, N. Duffield, M. Zhou, and X. Qian, "Variational graph recurrent neural networks," *Advances in neural information processing systems*, vol. 32, 2019.

[26] A. Pareja, G. Domeniconi, J. Chen, T. Ma, T. Suzumura, H. Kanezashi, T. Kaler, T. Schardl, and C. Leiserson, "Evolvegcn: Evolving graph convolutional networks for dynamic graphs," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 34, pp. 5363–5370, 2020.

[27] S. Kumar, X. Zhang, and J. Leskovec, "Predicting dynamic embedding trajectory in temporal interaction networks," in *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pp. 1269–1278, 2019.

[28] R. Trivedi, M. Farajtabar, P. Biswal, and H. Zha, "Dyrep: Learning representations over dynamic graphs," in *International conference on learning representations*, 2019.

[29] D. Xu, C. Ruan, E. Korpeoglu, S. Kumar, and K. Achan, "Inductive representation learning on temporal graphs," *arXiv preprint arXiv:2002.07962*, 2020.

[30] E. Rossi, B. Chamberlain, F. Frasca, D. Eynard, F. Monti, and M. Bronstein, "Temporal graph networks for deep learning on dynamic graphs," *arXiv preprint arXiv:2006.10637*, 2020.

[31] R. Ye, X. Li, Y. Fang, H. Zang, and M. Wang, "A vectorized relational graph convolutional network for multi-relational network alignment.," in *IJCAI*, pp. 4135–4141, 2019.

[32] J. Chen, T. Ma, and C. Xiao, "Fastgcn: fast learning with graph convolutional networks via importance sampling," *arXiv preprint arXiv:1801.10247*, 2018.

[33] D. Zou, Z. Hu, Y. Wang, S. Jiang, Y. Sun, and Q. Gu, "Layer-dependent importance sampling for training deep and large graph convolutional networks," *Advances in neural information processing systems*, vol. 32, 2019.

[34] Z. Liu, Z. Wu, Z. Zhang, J. Zhou, S. Yang, L. Song, and Y. Qi, "Bandit samplers for training graph neural networks," *Advances in Neural Information Processing Systems*, vol. 33, pp. 6878–6888, 2020.

[35] Q. Zhang, D. Wipf, Q. Gan, and L. Song, "A biased graph neural network sampler with near-optimal regret," *Advances in Neural Information Processing Systems*, vol. 34, pp. 8833–8844, 2021.

[36] P. Auer, N. Cesa-Bianchi, Y. Freund, and R. E. Schapire, "The nonstochastic multiarmed bandit problem," *SIAM journal on computing*, vol. 32, no. 1, pp. 48–77, 2002.

[37] J. Shetty and J. Adibi, "The enron email dataset database schema and brief statistical report," *Information sciences institute technical report, University of Southern California*, vol. 4, no. 1, pp. 120–128, 2004.

[38] P. Panzarasa, T. Opsahl, and K. M. Carley, "Patterns and dynamics of users' behavior and interaction: Network analysis of an online community," *Journal of the American Society for Information Science and Technology*, vol. 60, no. 5, pp. 911–932, 2009.

[39] A. Madan, M. Cebrian, S. Moturu, K. Farrahi, *et al.*, "Sensing the" health state" of a community," *IEEE Pervasive Computing*, vol. 11, no. 4, pp. 36–45, 2011.

[40] W. Jiang, X. Wang, P. Cui, B. Huang, Y. Yang, and Q. Fan, "Adaptive sampling temporal graph network," in *2022 4th International Conference on Advances in Computer Technology, Information Science and Communications (CTISC)*, pp. 1–8, IEEE, 2022.