K-Truss Based Temporal Graph Convolutional Network for Dynamic Graphs

Hongxi Li LIHONGXI@M.SCNU.EDU.CN
Zuxuan Zhang ZUXUANZHANG@M.SCNU.EDU.CN
Dengzhe Liang DZLIANG@M.SCNU.EDU.CN
School of Artificial Intelligence, South China Normal University, Foshan, 528225, China

Yuncheng Jiang* JIANGYUNCHENG@M.SCNU.EDU.CN
School of Artificial Intelligence, South China Normal University, Foshan, 528225, China
School of Computer Science, South China Normal University, Guangzhou, 510631, China

Editors: Berrin Yanıkoğlu and Wray Buntine

Abstract

Learning latent representations of nodes in graphs is important for many real-world applications, such as recommender systems, traffic prediction and fraud detection. Most of the existing research on graph representation learning has focused on static graphs. However, many real-world graphs are dynamic and their structures change over time, which makes learning dynamic node representations challenging. We propose a novel k-truss based temporal graph convolutional network named TTGCN to learn potential node representations on dynamic graphs. Specifically, TTGCN utilizes a novel truss-based graph convolutional layer named TrussGCN to capture the topology and hierarchical structure information of graphs, and combines it with a temporal evolution module to capture complex temporal dependencies. We conduct link prediction experiments on five different dynamic graph datasets. Experimental results demonstrate the superiority of TTGCN for dynamic graph embedding, as it consistently outperforms several state-of-the-art baselines in the link prediction task. In addition, our ablation experiments demonstrate the effectiveness of adopting TrussGCN in a dynamic graph embedding method.

Keywords: Graph Neural Networks, Dynamic Graphs, Representation Learning

1. Introduction

Learning the relationships between nodes in a graph is important for many real-world applications, such as recommender systems (Yang et al., 2022), traffic prediction (Zhao et al., 2019) and fraud detection (Guo et al., 2022). Graph representation learning aims to learn low-dimensional dense vector representations of nodes in a graph, where the learned embedding space can reconstruct the original graph and support inferences about the graph (Cui et al., 2019). Most of the current research on graph representation learning has focused on static graphs, where nodes and edges remain fixed. However, many real-world graphs are dynamic, with structures changing over time, and are often represented as a series of graph snapshots. Each snapshot stores data for a specific time interval, such as a day or a week. For example, in a social network, relationships between users evolve over time

* Corresponding Author

with users adding or removing friends, or joining different groups. These changes lead to the addition or removal of nodes and edges, and also affect the properties and behaviors of nodes in the social network. Therefore, it is very important to learn the representation of dynamic structures. Dynamic graph representation learning methods need not only to preserve topological structure information of graphs, but also to capture their temporal evolution at the same time.

Existing dynamic graph representation learning methods fall into two main categories. Temporal smoothing methods ensure the continuity and stability of embeddings across adjacent time steps (Goyal et al., 2018; Zhou et al., 2018). Recurrent methods integrate the graph neural network with a recurrent structure to generate dynamic node embeddings (Pareja et al., 2020; Yang et al., 2021; Liu et al., 2022), where the former captures the structural information of the graph and the latter captures the temporal information of the graph to summarize its historical evolutionary behavior by retaining and updating the hidden state. Temporal smoothing methods can reduce the effects of noise and outliers in dynamic graphs, enabling better capture of the evolutionary trends of dynamic graphs. However, it is difficult to model temporal dynamics with temporal smoothing methods when nodes present significantly different evolutionary behaviors. In comparison, recurrent methods are more widely applied due to their ability to capture intricate temporal dependencies, but they also face some challenges. On the one hand, graph neural networks usually aggregate information from neighboring nodes to generate embeddings for the target node. A typical example is graph convolutional network (GCN) (Kipf and Welling, 2017), which has been successfully applied in many graph-based applications. However, a one-layer GCN can only retain first-order proximity and may suffer from oversmoothing when the number of GCN layers increases, which limits the ability of GCN to capture global graph structure information. On the other hand, most methods only set the recurrent architecture on top of the graph neural network, which ignores the temporal evolutionary behavior of the lower-level node embeddings for multi-layer graph neural networks. In addition, the recurrent architecture has difficulties in modeling long-term dependencies and scalability. This motivates us to rethink whether it is possible to extend graph neural network methods to be able to capture more complex topological information of graphs and simultaneously learn rich evolutionary patterns and regularities.

To tackle the aforementioned problems, in this work, we propose a novel k-truss based temporal graph convolutional network named TTGCN to learn potential node representations on dynamic graphs. Since truss decomposition can naturally extract nested subgraphs at different scales, and each subgraph reflects important properties of the network such as connectivity and centrality, we design a novel truss-based graph convolutional network named TrussGCN to capture the hierarchical structure information in each snapshot. Then, we integrate the TrussGCN with advanced sequence models to further capture the temporal dependencies. To better capture the intricate temporal dependencies under different tasks and data features, we provide two different structures here. The hierarchical recurrent architecture differs from the general recurrent architecture. For a multilayer graph neural network, it models the temporal dynamics of the node embedding in each layer by retaining and updating the hidden state of each layer. The temporal self-attention layer is more efficient in capturing distant temporal dependencies due to its ability to extract context from all past graph snapshots and adaptively assign weights to historical representations.

We conduct link prediction experiments on five datasets with different sizes. The results indicate that TTGCN has significant gains compared to several state-of-the-art baselines, which further demonstrates that TTGCN can effectively learn the latent representations of nodes in large-scale dynamic graphs. We also demonstrate the superiority of using the TrussGCN module in dynamic graph embedding models through an ablation experiment. In summary, the main contributions of this paper are as follows:

- We propose a novel truss-based graph convolutional network which can effectively capture the multi-scale topological structure information of the graph.
- We propose a novel k-truss based temporal graph convolutional network, which can effectively capture both the intricate topology and the temporal dependencies of graphs. To the best of our knowledge, this is the first study of a dynamic graph representation learning method based on k-truss decomposition strategy.
- Extensive experiments on different real-world graphs demonstrate the superiority of TTGCN, as it consistently outperforms state-of-the-art methods on all datasets.

2. Related Works

Our work is mainly related to dynamic graph representation learning. Dynamic graphs are generally defined in two ways: (1) discrete time dynamic graphs, which are represented as a collection of graph snapshots at different time steps; and (2) continuous time dynamic graphs, which are denoted as graph updating event streams. The continuous time dynamic graph approach is effective in retaining temporal information. However, it may not always be feasible, as dynamic graphs often lack fine-grained timestamps. In contrast, the discrete time dynamic graph approach can be utilized in all graphs with timestamps by creating appropriate snapshots. Hence, we primarily focus on representation learning for discrete time dynamic graphs. For a systematic and detailed review, readers could refer to (Zhu et al., 2022) and (Barros et al., 2021).

Existing dynamic graph representation learning methods fall into two main categories. Temporal smoothing methods ensure continuity and stability of embeddings across adjacent time steps. For instance, DynGEM (Goyal et al., 2018) utilizes autoencoders to incrementally generate embeddings for the current time step based on the embedding of the graph snapshot from the previous time step. However, these methods have difficulty modeling temporal dependencies effectively when nodes exhibit highly different evolutionary behaviors. Recurrent methods combine the static graph neural network with a recurrent architecture to generate dynamic node embeddings. The former captures the structural information of graphs, while the latter captures the temporal information of graphs to summarize its historical evolutionary behavior by retaining and updating the hidden state. GCRN (Seo et al., 2018) provides two different architectures to capture the spatial and temporal information of dynamic networks by integrating the GCN model with an LSTM. Similarly, HTGN (Yang et al., 2021) projects the dynamic graph into hyperbolic space, and generates dynamic node representations utilizing hyperbolic GNN and hyperbolic GRU. CTGCN (Liu et al., 2022) uses a k-core based GCN to learn node embeddings, which are then fed into an RNN to capture evolving behaviors. On the other hand, EvolveGCN (Pareja et al., 2020) is a variant of GCN, which utilizes an RNN to dynamically evolve the parameters of GCN to capture the structural evolution of graphs.

However, the recurrent methods face challenges in modeling long-term dependencies and have some limitations in scalability. Therefore, some methods explore similar ideas but leverage other sequence models to capture complex temporal evolutionary behavior. DySAT (Sankar et al., 2020) employs a joint self-attention mechanism across spatial relationships and temporal evolution to learn dynamic node embeddings. GraphTCN (Wang et al., 2021) utilizes a 1-D convolutional neural network to capture temporal information and generate dynamic node representations. ROLAND (You et al., 2022) proposes a framework to extend static GNNs to dynamic graphs by updating the hierarchical node states with update modules.

3. Preliminaries

In this section, we first formally present the problem of representation learning for dynamic graphs. Then, we briefly introduce the principle of GCN.

3.1. Dynamic Graph Representation Learning

A dynamic graph can be represented as $G = (V, E)$, where V denotes the set of nodes of the graph and E denotes the set of edges of the graph. Each edge in the edge set is associated with a timestamp τ and can have additional edge features such as weight. The dynamic graph representation learning approach splits the dynamic graph into a series of graph snapshots $G = \{G_0, ..., G_T\}$, where T denotes the total number of graph snapshots. Each graph snapshot is a static undirected graph $G_t = (V, E_t)$, containing a shared set of nodes V and a set of edges $E_t = \{e \in E \mid \tau = t\}$. The objective of dynamic graph representation learning is to learn a latent representation for each node at each time step, and the learned representation preserves the topology of the graph and the temporal evolutionary behavior up to the current time step t such as edge joining or removal.

3.2. Graph Convolutional Network

Graph convolutional network (GCN) has achieved superior performance in many tasks. To learn the representation of each node, it leverages the connectivity between nodes to propagate and aggregate neighbor information through the graph's topology. Formally, a multi-layer GCN follows the layer-wise propagation rule:

$$\mathbf{H}^{(l)} = \sigma(\hat{\mathbf{A}}\mathbf{H}^{(l-1)}\mathbf{W}^{(l-1)}) \tag{1}$$

where $\mathbf{H}^{(l)}$ denotes the node representations of the l-th layer, $\mathbf{H}^{(0)} = \mathrm{X}$ is the input feature matrix, $\mathbf{W}^{(l-1)}$ denotes a trainable weight matrix of the (l-1)-th layer, $\sigma(\cdot)$ denotes a nonlinear activation function, and $\hat{\mathbf{A}}$ is a normalized adjacency matrix defined as:

$$\hat{\mathbf{A}} = \tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}} \tag{2}$$

where $\tilde{\mathbf{D}}$ denotes a diagonal matrix with $\tilde{\mathbf{D}}_{ii} = \sum_j \tilde{\mathbf{A}}_{ij}$, $\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}$ denotes the adjacency matrix with added self-connections, $\mathbf{I}$ is the identity matrix.

Since a one-layer GCN can only retain one-hop neighbor information, a multi-layer GCN is usually required to retain multi-hop neighbor information in the graph. However, as the

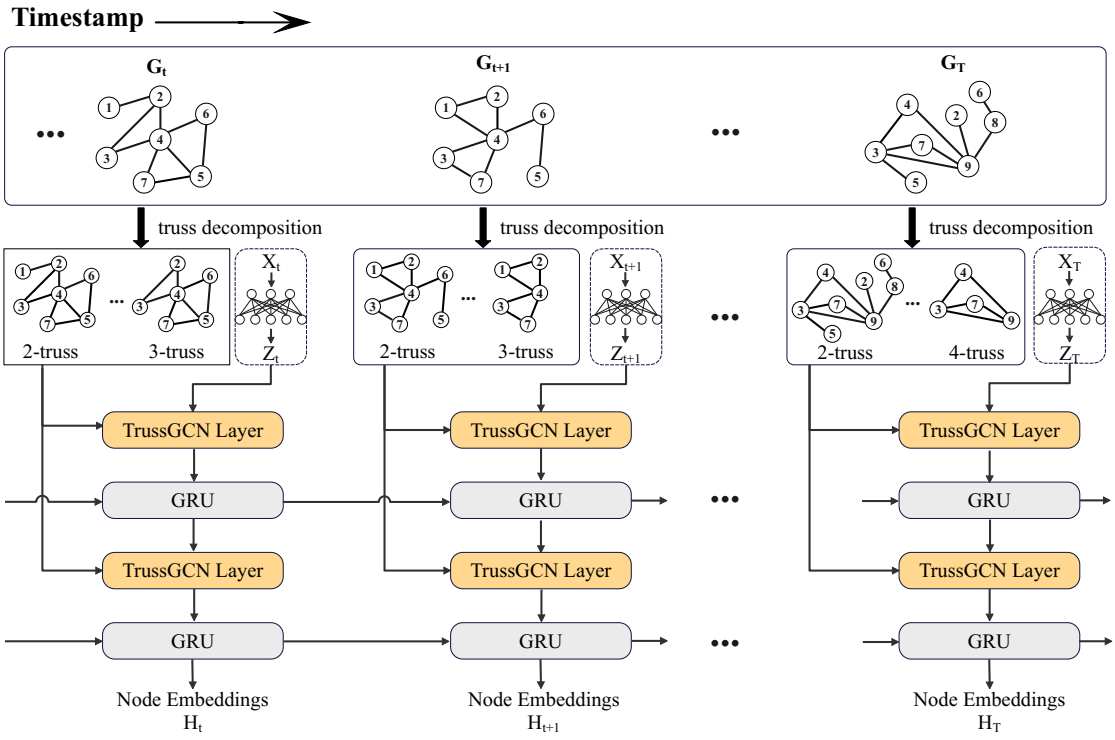

Figure 1: The overall architecture of the k-truss based temporal graph convolutional network(TTGCN-H).

number of layers increases, the GCN suffers from the problem of oversmoothing, which severely hinders the ability of the GCN to capture the rich contextual information in the graph. Therefore, it is critical to choose the appropriate number of layers when building a GCN model.

4. K-Truss Based Temporal GCN

We propose a k-truss based temporal graph convolutional network (TTGCN), which consists of the TrussGCN module and the temporal evolution module. Specifically, we provide two different temporal evolution modules, i.e., hierarchical recurrent architecture and temporal self-attention layer, and the corresponding TTGCN models are named TTGCN-H and TTGCN-S. Here, we present the overall framework of the TTGCN-H, as depicted in Fig. 1. We will elaborate the details of each module in the next paragraphs.

4.1. Truss-based GCN Module

We propose a novel truss-based graph convolutional network named TrussGCN to preserve the topological structure information in graphs. By utilizing k-truss subgraphs, TrussGCN is able to naturally capture the inherent multi-scale structural informations of the graph.

Here, we commence by presenting the formal definition of *support* and *k-truss* (Zheng et al., 2017).

A *triangle* is a cyclic structure of length 3. We denote a triangle consisting of three nodes u, v, $w \in V$ by $\triangle_{uvw}$. Based on the definition of triangle, we present the definition of the support of edges as follows:

Definition 1 (*Support*)).*For each edge* $e = (u, v)$ *in the edge set* E *of a graph* G*, its support is defined as* $sup(e, G) = |\{\triangle_{uvw} : w \in V\}|$.

The support of each edge in a graph G is determined by the number of triangles in G that contain that edge. Next, we present the definition of k-truss.

Definition 2 (*K-truss*).*Given a graph* G*, a subgraph* T_k *of the graph* G *is k-truss* $(k \geq 2)$ *if each edge* e *in the subgraph is contained in at least* $(k - 2)$ *triangles, that is,* $\forall e \in E_{T_k}$, $sup(e, T_k) \geq (k - 2)$.

From the above definitions, we can discover some important properties of k-truss. Firstly, k-truss captures a dense subgraph within nodes tightly connecting to each other, where each node has degree no less than k-1 (Cohen, 2008). The dense subgraph reflects important properties of the graph, such as connectivity and centrality. An analogous classical approach is k-core (Nikolentzos et al., 2018). Compared to k-core, k-truss is more consistent with the basic structure of the graph. This is because there are only simple edge connections in k-core, while k-truss is defined based on triangles, which are referred to as the basic building blocks of the graph (Wang and Cheng, 2012). Secondly, a nested structure is formed between the k-truss subgraphs of a graph G. Specifically, each k-truss subgraph is a subset of the k-truss subgraphs with smaller k values. Formally, let $T = \{T_2, T_3, \cdots, T_{k_{max}}\}$ be the k-truss set of a graph G, then, the relationship between the k-truss subgraphs is denoted as:

$$T_{k_{max}} \subseteq \cdots \subseteq T_3 \subseteq T_2 = G \tag{3}$$

where k_{max} is the maximum truss number of G.

Many problems of computing non-hierarchical dense subgraphs (Tsourakakis, 2015; Gao et al., 2018; Chen et al., 2021) are NP-hard. In contrast, we can obtain all k-truss subgraphs of G by using truss decomposition algorithm with algorithmic time complexity of $\mathcal{O}(m^{1.5})$, where m is the number of edges in G (Che et al., 2020). This allows k-truss to be applied efficiently in large-scale graphs (Kabir and Madduri, 2017). In addition, k-truss decomposition can be regarded as a graph partitioning strategy. A similar graph partitioning strategy is used in ClusterGCN (Chiang et al., 2019), which divides the graph into several subgraphs by a graph clustering algorithm to improve memory and computational efficiency. Here, we aim to extend the GCN to capture the abundant hierarchical structure information in the graph.

Based on the nested subgraph structure obtained by truss decomposition, we leverage all subgraphs to propagate the node features at different scales in the graph. Then, we aggregate the information of all subgraphs to capture the multi-scale topological structure information of the graph. The structure of the proposed TrussGCN is depicted in Fig. 2.

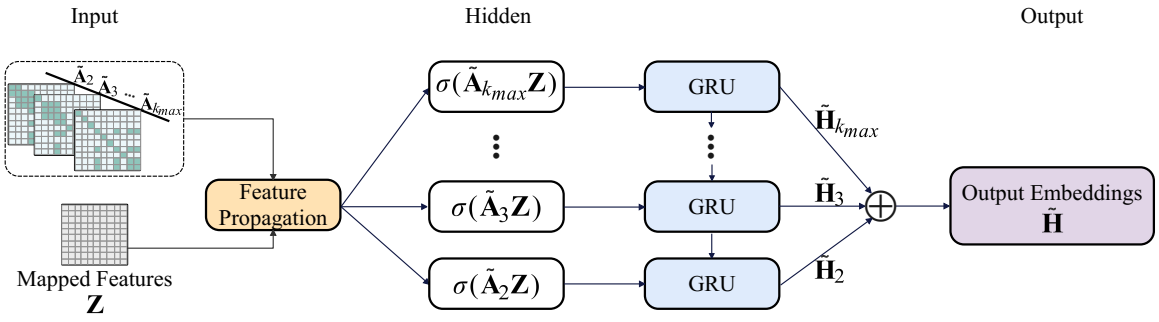

Figure 2: The architecture of the TrussGCN layer.

Similar to GCN, we defined the feature propagation rule of each k-truss subgraph as:

$$\mathbf{Z}'_k = \sigma(\tilde{\mathbf{A}}_k \mathbf{Z}) \tag{4}$$

where $\tilde{\mathbf{A}}_k = \mathbf{A}_k + \mathbf{I}$ is the extended adjacency matrix of the k-truss subgraph that incorporates self-connections for all nodes to ensure all k-truss adjacency matrices have the same dimension, $\mathbf{Z}$ denotes the feature transformation matrix obtained from the input feature matrix $\mathbf{X}$ by linear mapping, that is, $\mathbf{Z} = \mathbf{X}\mathbf{W} + \mathrm{b}$, and $\mathbf{Z}'_k$ denotes the hidden node representation of the k-truss subgraph. Notably, we share $\mathbf{Z}$ on all k-truss subgraphs, which reduces the complexity of the model.

After feature propagation on each subgraph, a simple and effective way to obtain the final embedding is to sum all the $\mathbf{Z}'_k$ directly. We will validate the effectiveness of this approach in the later experiment section. Furthermore, we suppose that there is a potential structural evolution relationship between the k-truss subgraphs of each graph. We employ the Gated Recurrent Unit(GRU) (Cho et al., 2014) to model the potential temporal dynamics between k-truss subgraphs, which is defined as:

$$\tilde{\mathbf{H}}_n = \mathrm{GRU}(\mathbf{Z}'_{k_{max}+2-n}, \tilde{\mathbf{H}}_{n-1}), \quad n = 2, \cdots, k_{max} \tag{5}$$

where $\tilde{\mathbf{H}}_1 = 0$ is a zero matrix. Finally, the output embeddings $\tilde{\mathbf{H}}$ of the TrussGCN is obtained by aggregating all $\tilde{\mathbf{H}}_n$ using an aggregation function:

$$\tilde{\mathbf{H}} = \sum_{n=2}^{k_{max}} \tilde{\mathbf{H}}_n \tag{6}$$

It is worth noting that, similar to GCN, a one-layer TrussGCN can only capture the information of one-hop neighbors, while a multi-layer TrussGCN can expand the receptive field and capture richer contextual information in the graph.

4.2. Temporal Evolution Module

To capture the intricate graph evolution behavior under different tasks and data characteristics, we provide two structures, a hierarchical recurrent architecture and a temporal

self-attention layer, to model the temporal dependencies between node representations at different time steps. We will present the details of each of these two structures.

For a multi-layer TrussGCN, the hierarchical recurrent architecture sets a GRU after each TrussGCN layer. Each GRU maintains a hidden state vector, and updates this hidden state vector at each time step to remain temporal dependency of the node embedding at each layer. Specifically, at time step t, the hidden state vector $\mathbf{H}_t^{(l)}$ of the l-th layer is affected by both the output representation $\tilde{\mathbf{H}}_t^{(l)}$ of the l-th TrussGCN layer at the current time step t and the historical hidden state $\mathbf{H}_{t-1}^{(l)}$ of that layer, denoted as:

$$\mathbf{H}_t^{(l)} = \mathrm{GRU}^{(l)}(\mathbf{H}_{t-1}^{(l)}, \tilde{\mathbf{H}}_t^{(l)}) \tag{7}$$

$$\tilde{\mathbf{H}}_t^{(l)} = \mathrm{TrussGCN}_t^{(l)}(\mathbf{A}_t, \mathbf{H}_t^{(l-1)}) \tag{8}$$

where $\mathbf{A}_t$ is the adjacency matrix of G_t, $\mathbf{H}_0^{(l)} = 0$ is a zero matrix, $\mathbf{H}_t^{(0)} = \mathbf{X}_t$ is the input feature matrix of G_t, $t = 1, \cdots, T$, and $l = 1, \cdots, L$.

On the other hand, we build a temporal self-attention layer on top of the TrussGCN. The temporal self-attention layer is easier to take into condideration long-term dependencies and is computationally more efficient for long sequences as it can operate in parallel across time steps. We add the position embedding into the output representation of the top TrussGCN layer and use it as the input representation of the temporal self-attention layer. Formally, let $\mathbf{X}_v = \{\mathbf{x}_v^1, \mathbf{x}_v^2, \cdots, \mathbf{x}_v^T\}$ denote the input representation of a node v packed together across time. To compute the output representation of the node v at t, we use the scaled dot-product form of attention (Vaswani et al., 2017), where the input representation is mapped into queries, keys and values through the weight matrices $\mathbf{W}_q$, $\mathbf{W}_k$ and $\mathbf{W}_v$. To maintain the auto-regressive property, we enable each time step to attend all previous time steps, including itself. The temporal self-attention layer is defined as:

$$\alpha_v^{ij} = (\frac{((\mathbf{X}_v\mathbf{W}_q)(\mathbf{X}_v\mathbf{W}_k)^T)_{ij}}{\sqrt{d_k}} + M_{ij}) \tag{9}$$

$$\mathbf{H}_v = \mathbf{A}_v(\mathbf{X}_v\mathbf{W}_v), \quad A_v^{ij} = \frac{exp(\alpha_v^{ij})}{\sum_{k=1}^{T} exp(\alpha_v^{ik})} \tag{10}$$

where α_v^{ij} denotes the attention score from time step i to j, d_k denotes the dimensions of both the query and key vectors, $\mathbf{M}$ is a mask matrix defined as:

$$M_{ij} = \begin{cases} 0, & i \le j \\ -\infty, & otherwise \end{cases}$$

When $M_{ij} = -\infty$, $A_v^{ij} = 0$ is a zero matrix, which turns off the attention from time step i to j. $\mathbf{A}_v$ is the attention weight matrix calculated by the multiplicative attention function. Then, the output representation $\mathbf{H}_v$ is fed into a position-wise feedforward layer to generate the final node representation.

It is worth noting that if more intricate temporal information is required, similar to the hierarchical recurrent architecture, we can extend the temporal self-attention approach to

capture the temporal dynamics of the node embedding at each layer. In addition, considering the structural evolution between k-truss subgraphs across time steps is also a method to enhance the module. However, these extension methods could have implications for the computational efficiency of the model.

4.3. Objective Function

In order for the learned representation to capture the temporal evolving behavior during temporal evolution, our objective function maintains the local connections surrounding a node across multiple time steps. At each time step t, we use two TrussGCN layers to preserve the 2-order proximity information in the graph snapshot. Specifically, we utilize an unsupervised loss function to incentivize nodes co-occurring in fixed-length random walks to possess similar representations.

The overall objective of the TTGCN can be described as follows:

$$\mathcal{L} = \sum_{t=1}^{T} \sum_{v \in V} \Big(\sum_{u \in \mathcal{N}_w^t(v)} -\log(\sigma(< \mathbf{h}_u^t, \mathbf{h}_v^t >)) - Q \cdot \sum_{u' \in P_n^t(v)} \log(1 - \sigma(< \mathbf{h}_{u'}^t, \mathbf{h}_v^t >)) \tag{11}$$

where $\mathcal{N}_w^t(v)$ represents the nodes that appear alongside v in fixed-length random walks at graph snapshot t, the negative sampling ratio Q is a modifiable hyper-parameter utilized to balance the positive and negative samples, $< . >$ denotes the dot product operation, and $P_n^t(v)$ represents a negative sampling distribution of the graph snapshot G_t, which is usually defined as a function of degree.

5. Experiments

In this section, extensive experiments are conducted to evaluate the proposed TTGCN model. We propose the following research questions to guide our experiments:

- **RQ$_1$**. How does TTGCN perform compared to state-of-the-art dynamic graph representation learning methods?
- **RQ$_2$**. What does each component of TTGCN contribute?

5.1. Experimental Setup

5.1.1. Datasets

We conduct experiments on five different datasets: (1) UCI dataset contains a collection of private messages sent on an online message network among students at the University of California, Irvine[1]. (2) Facebook dataset is a social network consisting of Facebook users, where users are represented as nodes and friendships between users are represented as edges between these nodes[2]. (3) The Autonomous Systems (AS) dataset consists of traffic flows

1. http://konect.cc/networks/opsahl-ucsocial/
2. http://networkrepository.com/fb-wosn-friends.php

Table 1: Dataset statistics.(k_{max}: maximal k-truss number.)

	UCI	**Facebook**	**AS**	**Enron**	**Math**
$\mathbf{k_{max}}$	6	7	11	18	11
# **Nodes**	1899	60370	6828	87036	24740
# **Total Edges**	59835	607487	1947704	530284	323357
# **Snapshots**	7	27	100	38	77

between routers that make up the Internet[3]. (4) Enron dataset is an email communication network from the Enron Corporation[4]. (5) Math is a communication network from the stack exchange website Math Overflow[5]. Table 1 provides detailed statistics of the above-mentioned datasets.

5.1.2. Baselines

We compare the performance of our TTGCN-H and TTGCN-S models with 7 state-of-the-art dynamic GNNs: (1) GCRN provides two different architectures to generate dynamic node representations. We choose the first architecture of GCRN, which captures the topology with the GCN model and then inputs the obtained node embeddings into an LSTM to capture the temporal dependence of the graph (Seo et al., 2018). (2) DynGEM utilizes autoencoders to incrementally generate embeddings for the current time step based on the embedding of the graph snapshot from the previous time step (Goyal et al., 2018). (3) EvolveGCN is a variant of GCN, which utilizes an RNN to dynamically evolve the parameters of GCN to capture the structural evolution of graphs (Pareja et al., 2020). (4) HTGN projects the dynamic graph into hyperbolic space, and leverages hyperbolic GNN and hyperbolic GRU to capture the spatial information and temporal information simultaneously (Yang et al., 2021). (5) ROLAND extends static GNNs to dynamic graphs by recursively updating hierarchical node states utilizing update modules (You et al., 2022). (6) CTGCN-C and (7) CTGCN-S use a k-core based GCN to learn node embeddings, which are then fed into an RNN to capture evolving behaviors (Liu et al., 2022).

5.1.3. Parameter Settings

In our experiments, we use one linear layer and two TrussGCN layers in both TTGCN-H and TTGCN-S methods. Both models obtain the original node feature matrix through one-hot encoding, and the final node embedding dimension is set to 128. We adopt the Adam optimizer (Kingma and Ba, 2014), and the weight decay and learning rate are set to 5×10^{-4} and 1×10^{-3}, respectively.

3. http://snap.stanford.edu/data/as-733.html
4. http://networkrepository.com/ia-enron-email-dynamic.php
5. http://snap.stanford.edu/data/sx-mathoverflow.html

Table 2: Average AUC scores of all timestamps for link prediction.

Dataset	UCI	Facebook	AS	Enron	Math
GCRN	0.8175	0.6829	0.8621	0.8891	0.8239
EvolveGCN	0.8528	0.7042	0.9036	0.8682	0.7599
HTGN	0.8227	0.7587	0.8934	0.8048	0.7639
DynGEM	0.9022	0.7923	0.9530	0.8901	0.8926
CTGCN-S	0.9106	0.8284	0.9628	0.9041	0.9232
ROLAND	0.9345	0.8360	*0.9650*	0.9119	0.9381
CTGCN-C	0.9272	0.8815	0.9332	0.9685	0.9687
TTGCN-H	**0.9659**	**0.9327**	0.9623	**0.9830**	*0.9786*
TTGCN-S	*0.9551*	*0.9184*	**0.9736**	*0.9708*	**0.9820**

5.2. Link Prediction

We perform link prediction tasks to evaluate the proposed TTGCN-H and TTGCN-S models. For $t = 1, \cdots, T$, we use the node embedding information at each time step t to predict whether an edge is present in the next snapshot $(t+1)$. To obtain the set of labeled edges, we randomly sample edges in snapshot $(t+1)$ as positive samples and sample the same number of node pairs without edge connectivity to generate negative samples. Then, we follow the experimental setup in the (Liu et al., 2022) to apply the Hadamard operation to the embedding vectors of the node pairs in the edge set to compute the edge feature vectors and train a logistic regression (LR) classifier to classify the positive and negative edge samples. We use the area under the curve (AUC) to evaluate candidate models and report the average AUC score as the measure of prediction performance.

We report the results of link predictions on different datasets in Table 2, with the best result for each dataset in bold and the next best result in italics. The results show that both our proposed TTGCN-H and TTGCN-S models significantly outperform all comparison methods on all datasets, which indicates that our proposed models are able to effectively preserve the hierarchical structure information in the graph and capture intricate temporal trends at the same time. We also observe that TTGCN-H and TTGCN-S each have advantages on different datasets. This suggests that the two models can cope with sequence data with different characteristics and hence model the temporal dynamics more effectively. In addition, we argue that the TrussGCN module can help the TTGCN model achieve better performance in the link prediction task by effectively capturing the topological and hierarchical properties of the graph at different scales, which is one of the reasons why TTGCN performs well in the link prediction task. We will follow up with an ablation experiment to further validate the effectiveness of the TrussGCN module.

5.3. Parameter Sensitivity

The proposed TTGCN models contain some parameters that may affect the performance. To study these parameters, we conduct experiments on the UCI dataset. We first analyze

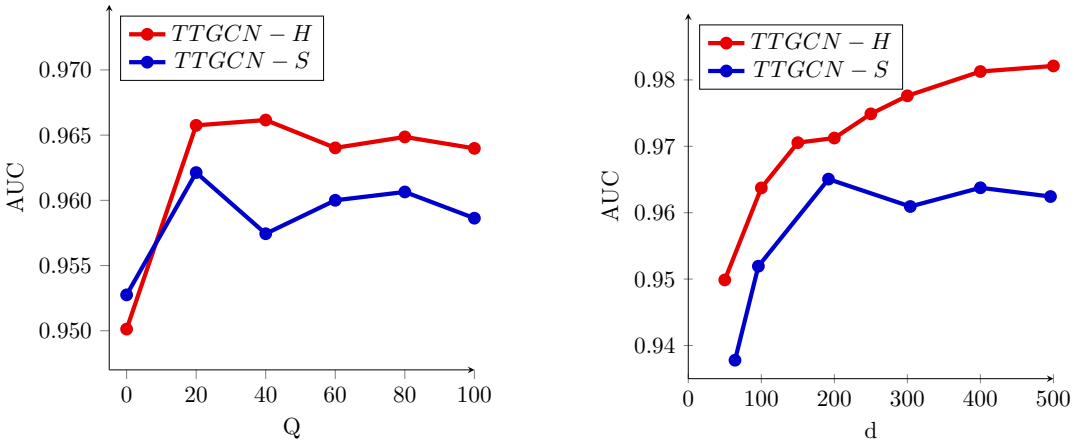

Figure 3: Sensitivity analysis of the hyperparameter Q and the embedding dimension d on the UCI dataset.

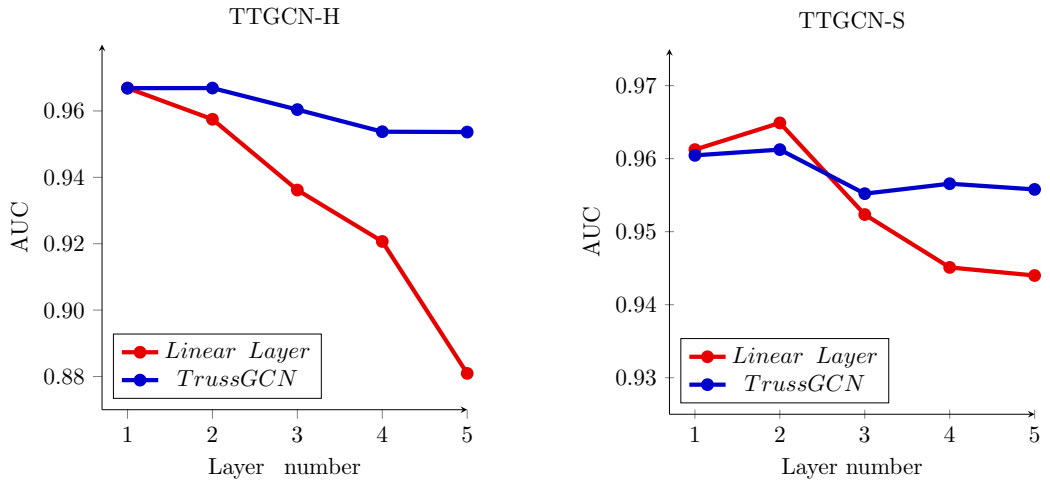

Figure 4: Average AUC scores with respect to the number of linear layers and TrussGCN layers on the UCI dataset.

the impact of hyperparameter Q (defined in *Eq.* 11) and embedding dimension d on the link prediction performance of TTGCN-H and TTGCN-S.

As shown in Fig. 3, we find that both TTGCN-H and TTGCN-S are sensitive to the hyperparameter Q. The performance of TTGCN-H and TTGCN-S increases as Q increases until Q reaches a certain value, and decreases when Q is too large. We also observe that the performance of TTGCN-S increases with the embedding dimension d and remains largely stable after reaching 200. In contrast, TTGCN-H is more sensitive to d, and its link prediction performance increases with increasing d.

We also test the effect of the number of linear layers and TrussGCN layers on the link prediction performance of TTGCN-H and TTGCN-S on the UCI dataset. The results are shown in Fig. 4. We can observe that both models are robust to changes in the number of

Table 3: Average AUC scores of all timestamps for ablation study.

Method	UCI	Facebook	AS	Enron	Math
GCN-H	0.8443	0.8404	0.7687	0.9319	0.9255
TTGCN-Simple	0.9294	0.9170	0.9065	0.9779	0.9774
TrussGCRN	0.9367	0.8795	0.9407	0.9711	0.9701
TTGCN-H	**0.9659**	**0.9327**	**0.9623**	**0.9830**	**0.9786**

TrussGCN layers, and the performance of both TTGCN-H and TTGCN-S is largely stable when the number of TrussGCN layers increases. In addition, we observe that both TTGCN-H and TTGCN-S are very sensitive to the number of linear layers. TTGCN-H is compatible with only one linear layer, and its performance decreases significantly when the number of linear layers increases. TTGCN-S reaches its best performance when the number of linear layers reaches 2. Then, if the number of linear layers continues to increase, its performance also decreases significantly. In conclusion, our experiments show that selecting appropriate values for these parameters is essential for achieving optimal performance of TTGCN-H and TTGCN-S.

5.4. Ablation Study

We conduct ablation experiments to further validate the effectiveness of the main components of our proposed model. We obtain the TTGCN-H variants as follows:

- GCN-H : Replace the TrussGCN module in TTGCN-H with GCN.
- TTGCN-Simple : Simplified TTGCN by omitting the GRU in each TrussGCN layer.
- TrussGCRN : Replace the graph evolving module in TTGCN-H by only building RNN on top of the TrussGCN.

We conduct five repetitions of each experiment and present the average AUC scores for the link prediction task. The results are shown in Table 3. It can be observed that the TTGCN-Simple and the TTGCN-H consistently outperform the GCN-H on five datasets. This indicates that the k-truss decomposition strategy can capture important structural properties at different scales of graphs, which is crucial to improve the link prediction performance of the model. Furthermore, it can be observed that the TTGCN-H consistently outperforms the TTGCN-Simple, which demonstrates that modeling the latent subgraph evolution process between k-truss subgraphs can effectively help the model improve its performance. In addition, we also observe that the TTGCN-H performs better than the TrussGCRN, suggesting that capturing the lower-level evolution information of the model is equally important to help the model better capture intricate temporal dependencies and improve performance of the model.

6. Conclusions

In this work, we propose a novel k-truss based temporal graph convolutional network named TTGCN for dynamic graph representation learning. Specifically, TTGCN learns dynamic node representations by integrating the novel truss-based graph convolutional network with

the temporal evolution module to capture the intricate topology and temporal dependencies of graphs. Experiments on five real-world graphs indicate significant performance improvements of TTGCN over state-of-the-art dynamic graph embedding baselines. In future work, we will develop new graph representation learning models based on k-truss decomposition strategy and explore continuous time dynamic graph learning to include finer-grained temporal changes.

References

Claudio DT Barros, Matheus RF Mendonça, Alex B Vieira, and Artur Ziviani. A survey on embedding dynamic graphs. *ACM Computing Surveys (CSUR)*, 55(1):1–37, 2021.

Yulin Che, Zhuohang Lai, Shixuan Sun, Yue Wang, and Qiong Luo. Accelerating truss decomposition on heterogeneous processors. *Proceedings of the VLDB Endowment*, 13 (10):1751–1764, 2020.

Jiejiang Chen, Shaowei Cai, Shiwei Pan, Yiyuan Wang, Qingwei Lin, Mengyu Zhao, and Minghao Yin. Nuqclq: an effective local search algorithm for maximum quasi-clique problem. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 12258–12266, 2021.

Wei-Lin Chiang, Xuanqing Liu, Si Si, Yang Li, Samy Bengio, and Cho-Jui Hsieh. Cluster-gcn: An efficient algorithm for training deep and large graph convolutional networks. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 257–266, 2019.

Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder–decoder for statistical machine translation. In *Conference on Empirical Methods in Natural Language Processing*, pages 1724–1734, 2014.

Jonathan Cohen. Trusses: Cohesive subgraphs for social network analysis. *National Security Agency Technical Report*, 16(3.1), 2008.

Peng Cui, Xiao Wang, Jian Pei, and Wenwu Zhu. A survey on network embedding. *IEEE Transactions on Knowledge and Data Engineering*, 31(5):833–852, 2019.

Jian Gao, Jiejiang Chen, Minghao Yin, Rong Chen, and Yiyuan Wang. An exact algorithm for maximum k-plexes in massive graphs. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence*, pages 1449–1455, 2018.

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

Xingzhi Guo, Baojian Zhou, and Steven Skiena. Subset node anomaly tracking over large dynamic graphs. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 475–485, 2022.

Humayun Kabir and Kamesh Madduri. Shared-memory graph truss decomposition. In *2017 IEEE 24th International Conference on High Performance Computing (HiPC)*, pages 13–22. IEEE, 2017.

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

Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *5th International Conference on Learning Representations*, 2017.

Jingxin Liu, Chang Xu, Chang Yin, Weiqiang Wu, and You Song. K-core based temporal graph convolutional network for dynamic graphs. *IEEE Transactions on Knowledge and Data Engineering*, 34(8):3841–3853, 2022.

Giannis Nikolentzos, Polykarpos Meladianos, Stratis Limnios, and Michalis Vazirgiannis. A degeneracy framework for graph similarity. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence*, pages 2595–2601, 2018.

Aldo Pareja, Giacomo Domeniconi, Jie Chen, Tengfei Ma, Toyotaro Suzumura, Hiroki Kanezashi, Tim Kaler, Tao Schardl, and Charles Leiserson. Evolvegcn: Evolving graph convolutional networks for dynamic graphs. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 5363–5370, 2020.

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

Youngjoo Seo, Michaël Defferrard, Pierre Vandergheynst, and Xavier Bresson. Structured sequence modeling with graph convolutional recurrent networks. In *International Conference on Neural Information Processing*, pages 362–373. Springer, 2018.

Charalampos E. Tsourakakis. The k-clique densest subgraph problem. In *Proceedings of the 24th International Conference on World Wide Web*, pages 1122–1132, 2015.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, page 6000–6010, 2017.

Chengxin Wang, Shaofeng Cai, and Gary Tan. Graphtcn: Spatio-temporal interaction modeling for human trajectory prediction. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, pages 3450–3459, 2021.

Jia Wang and James Cheng. Truss decomposition in massive networks. *Proceedings of the VLDB Endowment*, 5:812–823, 2012.

Menglin Yang, Min Zhou, Marcus Kalander, Zengfeng Huang, and Irwin King. Discrete-time temporal network embedding via implicit hierarchical learning in hyperbolic space. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 1975–1985, 2021.

Zhen Yang, Ming Ding, Bin Xu, Hongxia Yang, and Jie Tang. Stam: A spatiotemporal aggregation method for graph neural network-based recommendation. In *Proceedings of the ACM Web Conference 2022*, pages 3217–3228, 2022.

Jiaxuan You, Tianyu Du, and Jure Leskovec. Roland: graph learning framework for dynamic graphs. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 2358–2366, 2022.

Ling Zhao, Yujiao Song, Chao Zhang, Yu Liu, Pu Wang, Tao Lin, Min Deng, and Haifeng Li. T-gcn: A temporal graph convolutional network for traffic prediction. *IEEE Transactions on Intelligent Transportation Systems*, 21(9):3848–3858, 2019.

Zibin Zheng, Fanghua Ye, Rong-Hua Li, Guohui Ling, and Tan Jin. Finding weighted k-truss communities in large networks. *Information Sciences*, 417:344–360, 2017.

Lekui Zhou, Yang Yang, Xiang Ren, Fei Wu, and Yueting Zhuang. Dynamic network embedding by modeling triadic closure process. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, pages 571–578, 2018.

Yuecai Zhu, Fuyuan Lyu, Chengming Hu, Xi Chen, and Xue Liu. Encoder-decoder architecture for supervised dynamic graph learning: A survey. *arXiv preprint arXiv:2203.10480*, 2022.