On the Equivalence Between Temporal and Static Graph
Representations for Observational Predictions

Jianfei Gao\(^1\) and Bruno Ribeiro\(^1\)

\(^1\) Department of Computer Science, Purdue University, West Lafayette, In, U.S.A.

Abstract

In this work we formalize the (pure observational) task of predicting node attribute evolution in temporal graphs. We show that node representations of temporal graphs can be cast into two distinct frameworks: (a) The de-facto standard approach, which we denote \textit{time-and-graph}, where equivariant graph (e.g., GNN) and sequence (e.g., RNN) representations are intertwined to represent the temporal evolution of the graph; and (b) an approach that we denote \textit{time-then-graph}, where the sequences describing the node and edge dynamics are represented first (e.g., RNN), then fed as node and edge attributes into a (static) equivariant graph representation that comes after (e.g., GNN). In real-world datasets, we show that our \textit{time-then-graph} framework achieves the same prediction performance as state-of-the-art \textit{time-and-graph} methods. Interestingly, \textit{time-then-graph} representations have an expressiveness advantage over \textit{time-and-graph} representations when both use component GNNs that are not most-expressive (e.g., 1-Weisfeiler-Lehman GNNs). We introduce a task where this expressiveness advantage allows \textit{time-then-graph} methods to succeed while state-of-the-art \textit{time-and-graph} methods fail.

1 Introduction

Graph representation learning (GRPL) has recently focused on predicting node attribute evolution in temporal graphs. In most existing state-of-the-art works (Manessi et al., 2020; Pareja et al., 2020; Sankar et al., 2020; Goyal et al., 2020; Rossi et al., 2020), the temporal graph is described as a sequence of graph snapshots; these methods jointly learn an equivariant representation for each graph snapshot and how these node representations evolve over time. We call these \textit{time-and-graph} representations, which have dominated real-world applications in domains as diverse as social networks (Rossi et al., 2020), trading networks (Pareja et al., 2020), biological networks (Xu et al., 2019) and recommender systems (Sankar et al., 2020).

In this paper, we address the following question:

Q: Are \textit{time-and-graph} representations the only—and best—way to predict the observational (non-causal) evolution of nodes attributes in temporal graphs?

A: This work shows that there is another equally good—if not at times better—way to represent temporal graphs for observational tasks: \textit{Time-then-graph} representations, which first represent the temporal evolution of node and edge attributes, then use these representations to define a static graph representation.

Contributions. Our work introduces \textit{time-then-graph} representations and studies representation power over \textit{time-and-graph} and \textit{time-then-graph}. Our contributions are as follows:

1. \textit{Time-then-graph} representations are more expressive than \textit{time-and-graph} representations if the sequence representation is most expressive (e.g., GRU, LSTM) and the graph representation is a standard GNN (Gilmer et al., 2017) (more precisely, a 1-Weisfeiler-Lehman GNN (Morris et al., 2019; Xu et al., 2018)).

2. \textit{Time-then-graph} and \textit{time-and-graph} representations are equally expressive when both the temporal sequence and graph representations are most-expressive (e.g., the graph representations of Maron et al. (2019a); Murphy et al. (2019)).
Our experiments show that our time-then-graph methods outperform state-of-the-art time-and-graph methods in a hard synthetic task, and obtain equivalent performance in all real-world datasets. We also observe that most state-of-the-art time-and-graph methods perform roughly the same in all experiments.

In what follows we give an overview of our key insights.

2 Key Insights

Equivalence between time-and-graph and time-then-graph on data-driven observations. We now see why time-and-graph representations are equivalent in expressive power to time-then-graph representations. First, consider a temporal graph from an universe of nodes $V$ (all nodes that can appear in the graph over time). Let $A_{i,j,t}$ be the random variable that describes the edges of temporal graph at time $1 \leq t \leq T$, between nodes $i,j \in V$ (edges may also contain attributes). Let $X_{i,t}$ be the random variable describing attribute of node $i \in V$ at time $1 \leq t \leq T$ (including $i$ is not presented at time $t$). For edges and nodes not present at time $t$, their attributes are assumed to be null values (a special symbol). Based on the node equivariance, predicting the attribute of a specific node $v$ at time $T+1$ given the past, must follow the relation:

$$
\Pr[X_{v,T+1} | [X_{i,t}]_{i \in V}, [A_{i,j,t}]_{i,j \in V}]_{1 \leq t \leq T} = \Pr[X_{\pi(v),T+1} | [X_{\pi(i),t}]_{i \in V}, [A_{\pi(i),\pi(j),t}]_{i,j \in V}]_{1 \leq t \leq T},
$$

where $\pi$ is an arbitrary node permutation (permutation of $(1, \cdots, |V|)$ node indices), $A_{i,j,T} = [A_{i,j,1}, \cdots, A_{i,j,T}]$ and $X_{i,T} = [X_{i,1}, \cdots, X_{i,T}]$ integrate all past edge and node attributes as standalone attributes. In the first two equations, a temporal graph is regarded as a sequence of snapshots

$$
G_t = ([X_{i,t}]_{i \in V}, [A_{i,j,t}]_{i,j \in V}),
$$

at time $1 \leq t \leq T$, while in the last two equations, a temporal graph is regarded as a static graph

$$
G = ([X_{i,\leq T}]_{i \in V}, [A_{i,j,\leq T}]_{i,j \in V}),
$$

aggregating temporal attributes over all time steps per node and edge (see Figure 1).

Equation 1 shows that predicting the future node attributes of a temporal graph can be cast as a standard equivariant node prediction task on a static graph whose node and edge attributes include all past node and edge attributes, respectively.

We now can show the benefits of using time-then-graph over time-and-graph representations when the graph representation building block used is a GNN (Gilmer et al., 2017) (1-WL GNN (Morris et al., 2019; Xu et al., 2018)).
**Theorem (Informal).** If a GNN is the graph representation building block, and the temporal representation is most-expressive (e.g., GRU (Cho et al., 2014), LSTM (Hochreiter & Schmidhuber, 1997), Self-Attention (Vaswani et al., 2017)), then a time-then-graph representation is more expressive than a time-and-graph representation.

The above informal theorem statement will be later formalized in Theorem 1.

**Equivariant time-and-graph and time-then-graph representations are not suitable for link prediction.** Temporal (equivariant) node representations should not be used for link prediction: Although most of state-of-the-art methods achieve fairly good performance on temporal link prediction tasks, our work also points out that temporal (equivariant) node representations (e.g., node representations derived from GNNs) are not designed for this task. This is a consequence of the equivalence between time-and-graph and time-then-graph representations, which allow us to apply the static graph results of Srinivasan & Ribeiro (2019).

In the following sections, we will expand on each of the above insights. We first start with some essential preliminary definitions.

### 3 Preliminaries

#### 3.1 Representation and Expressiveness

In what follows we define the expressiveness of a neural network representation, which is a topic extensively covered by the graph neural network literature in particular (e.g., Maron et al. (2019b); Azizian & Lelarge (2020); Kileel et al. 2019; Morris et al. (2019); Murphy et al. (2019); Xu et al. (2018)).

**Definition 1 (Representation Family).** A representation family is a set of functions $\mathcal{F} = \{ f : U \mapsto \mathbb{R}^d \}$, where $U$ is the input domain and $d \geq 1$.

Different representation families focus on different domains. For example, $\mathcal{F}$ can be the set of RNN functions when $U = \mathbb{R}^{T \times p}$ is a sequence input; another example is $\mathcal{F}$, then domain $U = \mathbb{R}^{V \times V \times \mathbb{R}^{V \times V \times p}}$ where $V$ is an arbitrary node set and $p \geq 1$.

Kazemi et al. (2020) argues that the expressiveness of a representation function of a discrete input domain $U$ can be regarded as the ratio between the cardinality of its image and its domain. Based on this, we describe an identifiable set, which is used to define the expressiveness of representation functions.

**Definition 2 (Identifiable Set).** The identifiable set $\mathcal{I}(f, U)$ of a representation function $f$ with domain $U$ is a set s.t.: 1. It is a subset of $U$ that is $\mathcal{I}(f, U) \subseteq U$; 2. None of its elements have the same representations, that is $\forall u_1 \neq u_2 \in \mathcal{I}(f, U), f(u_1) \neq f(u_2)$; 3. It is defined over domain $U$, that is, $\forall u_1 \in U \setminus \mathcal{I}(f, U), \exists u_2 \in \mathcal{I}(f, U), f(u_1) = f(u_2)$.

In other words, $\mathcal{I}(f, U)$ is the maximal subset of $U$ that $f$ can differentiate any pair of elements in it.

There can be multiple identifiable sets for the same function and input space but, intuitively, they are all equivalent since there should be a bijection between any pair of identifiable sets. A proof is provided in Appendix B.1

**Definition 3 (More Expressive Representation).** Consider two representation families $\mathcal{F}_1$ and $\mathcal{F}_2$ of domain $U$. We say $\mathcal{F}_2 \preceq_U \mathcal{F}_1$, $\mathcal{F}_1$ is more or as equally expressive as $\mathcal{F}_2$, if and only if $\exists f_1 \in \mathcal{F}_1, \forall f_2 \in \mathcal{F}_2, |\mathcal{I}(f_2, U)| \leq |\mathcal{I}(f_1, U)|$. It is strictly more expressive, $\mathcal{F}_2 \prec_U \mathcal{F}_1$, if and only if $\exists f_1 \in \mathcal{F}_1, \forall f_2 \in \mathcal{F}_2, |\mathcal{I}(f_2, U)| < |\mathcal{I}(f_1, U)|$. If two representation families satisfies both $\mathcal{F}_2 \preceq_U \mathcal{F}_1$ and $\mathcal{F}_1 \preceq_U \mathcal{F}_2$, we say that they are equally expressive, $\mathcal{F}_2 \preceq_U \mathcal{F}_1$.

Generally speaking, more expressive means being able to differentiate more non-equivalent elements on domain $U$.

**Definition 4 (Most Expressive Representation).** A representation family $\mathcal{F}^+$ for $U$ is the most expressive representation family if and only if $\forall u_1, u_2 \in U$, $\exists f^+ \in \mathcal{F}^+, f^+(u_1) = f^+(u_2) \iff u_1 = u_2$. We denote such $f^+$ as the most expressive representation function.
Generally speaking, a most-expressive representation family has a function that is able to differentiate distinct elements of domain \( U \). Note that this depends on the domain.

**Lemma 1** (More Expressive Alternative). For two representation families \( F_1 \) and \( F_2 \) of domain \( U \), if for any function of \( F_2 \), there is an equivalent function on \( U \) in \( F_1 \), then \( F_2 \preceq_U F_1 \). To be more specific:

\[
\forall f_2 \in F_2, \exists f_1 \in F_1, \text{ such that } \forall u \in U, f_1(u) = f_2(u) \implies F_2 \preceq_U F_1.
\]

The above means that if \( F_1 \) can simulate any representation function in \( F_2 \) on domain \( U \), then it has at least the same expressiveness.

### 3.2 Expressiveness of Recurrent Neural Networks

In this work, we focus on sequences \([S_1, \ldots, S_T]\) \( \in \mathbb{R}^{T \times p} \) of node or edge attributes with a maximum time length of \( T \), whose elements are \( p \)-length vectors, where \( T \) and \( p \) are integers.

GRUs (Cho et al., 2014), LSTMs (Hochreiter & Schmidhuber, 1997), and Self-Attention (Vaswani et al., 2017) are a good ways to represent a temporal sequence. In this work we will generally refer to these methods as Recurrent Neural Network (RNN) representations. Strictly speaking, RNNs recursively encode the current and past observations as new representation of current time step (self-attention works a little differently by encoding the entire time sequence at once). The RNN representation function can be defined as

\[
H_t = \begin{cases} 
\text{RNNCell}(S_t, H_{t-1}), & 1 \leq t \leq T, \\
0, & t = 0
\end{cases}
\]

where \( \text{RNNCell} \) is learnable function (e.g., a RNN/LSTM/GRU cell) and \( 0 \) is a start-of-time indicator (e.g., a vector of zeros). The hidden state representation at the last step \( H_T \) is then the representation for the entire sequence \([S_1, \ldots, S_T]\). When we focus on the entire sequence representation, our RNN is defined as

\[
H_T = \text{RNN}([S_1, \ldots, S_T])
\]

In the case of self-attention, we consider \( H_T \) as obtained by a set representation instead. In our applications, we need only to focus only uni-directional RNN and Self-Attention with future masking. An overview of GRU, LSTM and Self-Attention for our applications is provided in Appendix C.1.

Siegelmann & Sontag (1995) has shown that RNN is a most expressive representation (universal Turing machine approximator) for arbitrary sequences in \( \mathbb{R}^{T \times p} \). Thus, in what follows we assume that we have the most expressive RNN representation function for \( \mathbb{R}^{T \times p} \). A most-expressive RNN representation function is always injective (Definition 4), thus there exists a decoder function translating most-expressive representations back to raw sequences.

### 3.3 Expressiveness of (Static) Graph Neural Networks

We now focus on static graph representations, where \( V \) is the set of nodes, and node attributes are \( p \)-length vectors.

**Definition 5** (Static Graph). A static graph can be defined as a tuple \((X, A)\), where \( X \in \mathbb{R}^{|V| \times p} \) are the node attributes and \( A \in \mathbb{R}^{|V| \times |V| \times p} \) are the edge attributes, with \( V \) a set of unique nodes in the graph. Here, \( p \geq 1 \) is the dimension of the observed node and edge attributes. If nodes or edges have no attributes, we pad with null items (e.g., zeros) to fit them into the definition. We denote the set of all such graphs as \( \mathcal{G}_{|V|, p} := \mathbb{R}^{|V| \times p} \times \mathbb{R}^{|V| \times |V| \times p} \).

In static graph representation learning, Message Passing Neural Network (MPNN) (Gilmer et al., 2017) is by far the most popular method due to its simplicity and efficacy (Kipf & Welling, 2016; Defferrard et al., 2016; Veličković et al., 2017). In this work, we use the term “GNN” to refer to all MPNN-type methods. A

\footnote{Chung et al. (2014) provides an alternative approach of the same power. We focus only on Cho et al. (2014) in this work.}
GNN (using mean readout) recursively encodes node representations at each layer (recursion) \( 1 \leq l \leq L \) as follows:

\[
M^{(l)}_i = \sum_{j \in \mathcal{N}(i)} g^{(l)}(Z^{(l-1)}_i, Z^{(l-1)}_j, A_{ij}), \tag{5}
\]

\[
Z^{(l)}_i = \begin{cases} 
  u^{(l)}(Z^{(l-1)}_i, M^{(l)}_i), & l > 0, \\
  X_i, & l = 0.
\end{cases} \quad \forall i \in \mathcal{V},
\]

where \( M^{(l)}_i \) is a message from the neighbors of node \( i \) at layer \( l \), \( X \) and \( A \) are the node and edge attributes, respectively, and \( Z^{(l)}_i \) is the representation output of node \( i \) at layer \( l \), \( g^{(l)} \) and \( u^{(l)} \) are learnable functions (often no-hidden-layer MLP), and \( \mathcal{N}(i) \) is the set of 1-hop neighbors of node \( i \). Sometimes, people are interested in whole graph representation instead of node representation, and a readout function is applied to get graph representation, e.g., sum readout \( \sum_{i \in \mathcal{V}} Z^{(L)}_i \). For ease of notation, we denote a stack of \( L \)-layer GNN operations by just a single function

\[
Z^{(L)} = \text{GNN}^L(X, A),
\]

where all the variables are the same as in Equation (5), and \( Z^{(L)} \) is the representation for all nodes at the last layer. The graph representation GNN\(^L(\cdot)\) is equivariant, i.e., a permutation of node IDs acting on the adjacency matrix and node attributes (e.g., node \( i \) changes its node id to \( i' = \pi(i) \)) impose the same permutation over the final GNN node representation vector \( Z^{(L)} \) (i.e., the representation \( Z^{(L)}_i \) is now \( Z^{(L)}_{i'} \)).

The above GNN architecture can be classified as 1-WL GNNs, due to its known relationship with the Weisfeller-Lehman test (Douglas, 2011), which also tells us GNNs have limited expressiveness (Morris et al., 2019; Maron et al., 2019a; Murphy et al., 2019) proposed methods to overcome the 1-WL expressiveness limitation, but these either considerably increase the time and memory costs of the algorithms, or are randomized algorithms with potentially high variance at test time. In this paper, we concentrate mainly on 1-WL GNNs.

## 4 Temporal Graph Representation

As described in Section 2, a temporal graph is a static graph with sequences of node and edge attributes:

**Definition 6 (Aggregated Temporal Graph).** An aggregated temporal graph over \( T \) time steps can be defined as a static graph \( \mathcal{G} = (\mathcal{X}_{\leq T}, \mathcal{A}_{\leq T}) \), where \( \mathcal{X}_{\leq T} \in \mathbb{R}^{\mathcal{V} \times T \times p} \) are the node attributes and \( \mathcal{A}_{\leq T} \in \mathbb{R}^{\mathcal{V} \times \mathcal{V} \times T \times p} \) are the edge attributes, with \( \mathcal{V} \) a set of all possible unique nodes in the temporal graph. Here, \( p \geq 1 \) is the dimension of the observed node and edge attributes (attributes may be filled by null if not present). We denote the set of all such aggregated temporal graphs as \( \mathbb{T}_{\mathcal{V}, T, p} := \mathbb{R}^{\mathcal{V} \times T \times p} \times \mathbb{R}^{\mathcal{V} \times \mathcal{V} \times T \times p} \).

As in Section 2, the universe of all possible nodes is \( \mathcal{V} \); we also similarly define the universe of all possible edges \( \mathcal{E} \). We call the graph in Definition 6 an aggregated temporal graph in order to differentiate it from equivalent temporal graph description (snapshots) in Equation (2). We now describe two different temporal graph representations.

**Time-and-graph Representation Family.** Time-and-graph representation family is the most common representation adopted in the literature (and the current state-of-the-art for temporal graph representation learning). The most widely used version of time-and-graph representations (e.g., Li et al. (2019); Chen et al. (2018); Seo et al. (2018)) treats the temporal graph as a sequence of snapshots:

\[
[\mathcal{G}_1, \cdots, \mathcal{G}_T] = [\mathcal{X}_{\leq 1}, \mathcal{A}_{\leq 1}, \cdots, \mathcal{X}_{\leq T}, \mathcal{A}_{\leq T}]
\]

and a GNN is applied to each snapshot \( \mathcal{G}_t = (\mathcal{X}_{\leq t}, \mathcal{A}_{\leq t}) \), \( 1 \leq t \leq T \), and the output of these GNNs are
combined with a sequence representation, giving the recursive architecture $T$:

$$H_{i,t} = \begin{cases} \text{RNNCell} \left( \left[ \text{GNN}_{\text{in}}^L(X_{i,t}, A_{i:t}) \right]_{i \in V} \right), & 1 \leq t \leq T, \\ \text{0}, & t = 0, \end{cases}$$

(6)

where RNNCell is a single time step representation of an arbitrary sequence representation function (e.g., Equation (4)), and 0 is the zero vector. The representation at the last step $H_{i,T}$ is regarded as the final representation of node $i$.

Other related work (e.g., Manessi et al. (2020); Sankar et al. (2020); Seo et al. (2018)) are special cases of the above time-and-graph architecture where, for $1 \leq t \leq T$,

$$H_{i,t} = \begin{cases} \text{RNNCell} \left( \left[ \text{GNN}_{\text{in}}^L(X_{i,t}, A_{i:t}) \right]_{i \in V}, H_{i,t-1} \right), & t > 0, \\ \text{0}, & t = 0. \end{cases}$$

(7)

Note that Equation (7) is a simplification of Equation (6) where the graph convolution is not applied to $H_{i,t-1}$. The above could also be called “graph-then-time”, but in the end, we prefer to refer to it as a special case of time-and-graph.

**Time-then-graph Representation Family.** This work proposes a time-then-graph framework, which is also an equivariant node representation for temporal graphs. Instead of applying a GNN over each snapshot, we represent the evolution of the node attributes using a sequence model. We also perform a sequence representation of the temporal edges. These new node and edge representations then become node and edge attributes in a new (static) graph, which is then represented by a GNN. This architecture is generally described as follows:

$$H_{i}^V = \text{RNN}^V(X_{i,T}), \quad \forall i \in V,$$

$$H_{i,j}^E = \text{RNN}^E(A_{i,j,T}), \quad \forall (i,j) \in E,$$

$$Z = \text{GNN}^L_{\text{agg}}(H^V, H^E).$$

(8)

The final output $Z$ is regarded as the representation of all nodes over all time steps. Although applying representation learning on an aggregated temporal graph $(X, A)$ has been performed by Rahman et al. (2018), their approach is a special case —rather than a general framework as in our case— and does not use GNNs or any other type of graph representation learning.

**Granger Causality and the Equivalence between Time-and-graph and Time-then-graph.** Effective data-driven predictions (e.g., predict $G_{T+1}$ given $G_T, \cdots, G_1$) can come from very different causal models that are equally effective at describing the conditional distribution $\text{Pr}[G_{T+1} | G_T, \cdots, G_1]$ (Peters et al. 2011). In some extrapolation tasks, time-and-graph shows an advantage over time-then-graph (see n-body simulation prediction (Battaglia et al. 2016; Xu et al. 2021)). This, however, does not mean that time-and-graph indeed holds any advantage over time-then-graph in pure data-driven observational tasks.

Temporal precedence is generally assumed essential to define causation, and it is one of the most important clues that people use to distinguish causal from other types of associations (Pearl 2009). Inspired by this intuition, Granger causality assumes no exogenous confounding variables to imply causality when past observed variables (e.g., $G_1, \cdots, G_T$) are able to make predictions about another observed variable in the future (e.g., $G_{T+1}$) (Eichler 2012).

While it is true that $G_{T+1}$ cannot cause $G_t$, $1 \leq t \leq T$, it is also true that another confounder (static) variable $U$ could cause all $\{G_t\}_{t \in \{1, \cdots, T+1\}}$ and produce the same exact conditional probabilities $\text{Pr}[G_{T+1} | G_T, \cdots, G_1]$ observed in the training/test data, $\forall t \in \{1, \cdots, T\}$. Ultimately, a time-then-graph representation is able to capture some information about a confounder $U$ and use it to predict $G_{T+1}$. Even when a time-then-graph is the wrong causal model for a task (e.g., COVID-19), it can show comparable predictive performance to the best time-and-graph method (see Section 7), as long as the task is observational.
and not an extrapolation task. Hence, for pure data-driven observational tasks, both time-then-graph and time-and-graph can provide the same predictive performance.

In the next section we compare the expressiveness of time-and-graph and time-then-graph representation family, and show a temporal graph classification task where today’s most widely used time-and-graph framework will fail.

5 Theorems

Theorem 1. [Temporal 1-WL Expressiveness] Time-then-graph representation family is strictly more expressive than time-and-graph representation family when the graph representation is a 1-WL GNN, i.e.,

\[ \text{time-and-1WL} \preceq_{\mathcal{V}_{|T,p}} \text{time-then-1WL}. \]

In Appendix B.2 we prove Theorem 1 by showing that we can always construct a time-then-graph neural network that outputs the same representations as a time-and-graph neural network. Thus, by Lemma 1, time-then-graph is as expressive as time-and-graph. Finally, we provide a task (Figure 2) where time-and-graph representation will fail while time-then-graph still works.

Figure 2: A task to distinguish two different CSL dynamics (see Section 7 for details). The two temporal graphs in figures (a) and (b) show different snapshots in the second time step (denote by \( C_{7,2} \) and \( C_{7,1} \)). Time-and-graph using 1-WL GNNs will fail since GNN outputs the same node representation for both snapshot \( C_{7,2} \) and \( C_{7,1} \) (see Murphy et al. (2019)). On the other hand, time-then-graph works since even the aggregated temporal graphs of (a) and (b) show different aggregate topologies (and edge attribute colors encoding the temporal dynamics), as shown in figure (c) that a GNN can easily distinguish.

Corollary 1. [Temporal Expressiveness] Using more expressive GNN (e.g., k-WL GNN of (Morris et al., 2019)) will improve the expressiveness of time-then-graph, and if a most-expressive GNN (GNN+) (e.g., Murphy et al. (2019) or Maron et al. (2019)) are used, time-then-graph and time-and-graph representation families will both be most-expressive.

\[ \text{time-then-kWL} \preceq_{\mathcal{V}_{|T,p}} \text{time-and-kWL}, \]

where \( A \preceq_C B \) means \( A \preceq C B \) and \( B \preceq C A \). A proof is provided in Appendix B.3. Finally, we show that equivariant temporal node representations cannot predict temporal links.

Theorem 2. [Temporal Link Prediction Impossibility] No equivariant temporal node representation (be in time-then-graph or time-and-graph family) is equipped to correctly predict temporal links.

Our proof relies on the temporal task in Figure 3, where neither time-then-graph nor time-and-graph family produce equivariant temporal node representations that can correctly predict links in the training data.
Figure 3: A simple example of two disconnected components (boreal forest and antarctic fauna) over two-step dynamics in a food chain system. The two disconnected components have the same dynamic because of similar prey and predator behaviors in the ecosystem. Extending Srinivasan & Ribeiro (2019) to temporal graphs, we can show that equivariant node representations of both time-then-graph and time-and-graph will output the same temporal representations for nodes Lynx and Orca. Hence, temporal link prediction based on these node representations will predict the same predator-prey edges for the Lynx and the Orca (e.g., Orcas eat Coyotes) which are incorrect predictions. See Appendix B.4 for more details.

6 Related Work

Temporal Graph Representation Surveys. A few surveys have summarized the time-and-graph literature in recent years. Skarding et al. (2020); Rossetti & Cazabet (2018) formalize existing temporal graph tasks into several categories. Kazemi et al. (2020); Skarding et al. (2020) also summarize state-of-the-art GNN-based temporal representation methods. Kazemi et al. (2020) also introduces the concept of expressiveness in temporal graph, however, neither of these studies consider the expressiveness of existing works or compare them on the same datasets. Holme (2015); Aouay et al. (2014); Holme & Saramäki (2012) focuses mostly on data-driven description (not representation learning) of temporal graphs, and studies temporal graph properties by statistics.

Time-and-graph. GCRN-M2 (Seo et al., 2018) is the first work, as far as we know, to adopt Equation (6) combining SpectralGCN (Defferrard et al., 2016) and LSTM (Hochreiter & Schmidhuber, 1997). There is a similar later work GC-LSTM (Chen et al., 2018) where adjacency matrix is also used as direct input besides convolution. LRGCN (Li et al., 2019) is another similar work applies a learnable projection on node representations.

Graph-then-time. Framework defined by Equation (7) is the most popular among both traditional and state-of-the-art works. Seo et al. (2018) proposes GCRN-M1 in the same paper of GCRN-M2 using the same graph and sequence modelings. Manessi et al. (2020) also proposes a quite similar work WD-GCN where a residual connection, as in He et al. (2016), is added to the GNN. Another state-of-the-art work DySAT (Sankar et al., 2020) exploits self-attention mechanism by using GAT (Veličković et al., 2017) and Self-Attention (Vaswani et al., 2017) as graph and sequence modelings. Of special interest is EvolveGCN (Pareja et al., 2020), which learns the temporal dynamics through the GCN parameters. EvolveGCN can also be regarded as graph-then-time since the evolving GCN parameters are RNN hidden states.

Time-then-graph. Although time-then-graph has been noticed by Holme (2015); Holme & Saramäki (2012) for a while, only limited works have tested its power. The closest work we can find is Rahman et al. (2018). It tested several ways of accumulating adjacency matrices. However, this work neither tried GNN nor analyzed the expressiveness. On the other hand, Rossi et al. (2020) is the state-of-the-art time-then-graph work different from ours. Instead of accumulation, they collect all historical edges and build a static multigraph, then GTC [Xu et al. (2020)] is used as multigraph representation.

Permutation-sensitive Temporal Graph Representations. There are several temporal graph works which applies non-equivariant representations. DyAE-RNN (Goyal et al., 2020) uses adjacency vector of
each node as augmented node feature; AdaNN-D \cite{Xu et al., 2019} uses the random walk vector based on adjacency matrix as augmented node feature; E-LSTM-D \cite{Chen et al., 2019} uses flattened adjacency matrix as input argument of LSTM; and GC-LSTM \cite{Chen et al., 2018} uses adjacency matrix as input argument of LSTM. All aforementioned works share the same problem that if we permute node indices, which results in a different adjacency matrix, their final node representation changes. This violates the equivariance hypothesis in Equation (1).

7 Experiments

In this section, we present a few architectures for our time-then-graph models, and compare against existing time-and-graph and time-then-graph baselines. We split each temporal graph dataset into folds, learn and tune the model on a set of folds and test the model on a different fold. Stratified folding is used to ensure label distribution is similar between training, validation and test splits. A detailed account of our data split strategy is provided in Appendix D.1.

7.1 Time-then-graph Configuration

We implement a time-then-graph representation (Equation (8)) as follows:
1. For RNN\(^V\), we use a GRU \cite{Cho et al., 2014}.
2. For RNN\(^E\), we use two different strategies: The first strategy (denoted as D) is based on exponential decay
\[
H_{i,j}^E = \sigma \left( \sum_{t=1}^{T} e^{\lambda(T-t+1)} A_{i,j,t} \right),
\]
where \(\sigma\) is sigmoid function and \(\lambda\) is the only learnable parameter; The second strategy (denoted as S) assigns learnable weights to each time step
\[
H_{i,j}^E = \sigma \left( \sum_{t=1}^{T} \theta_t A_{i,j,t} \right),
\]
where \(\sigma\) is sigmoid function and \(\theta\) is a learnable vector of length \(T\).
3. For the static GNN of time-then-graph, we simply use GCN \cite{Kipf & Welling, 2016} and GTC \cite{Xu et al., 2020}.

For readers unfamiliar with aforementioned GNN models, a brief formula abstract is provided in Appendix C.2.

Finally, by combining these different parts, our proposed time-then-graph models are GRU-D-GCN, GRU-D-GTC, GRU-S-GCN, and GRU-S-GTC.

7.2 Baselines

We compare our time-then-graph approach with four time-and-graph state-of-the-art baselines (namely, EvolveGCN \cite{Pareja et al., 2020}, GCRN-M2 \cite{Seo et al., 2018}, WD-GCN \cite{Manessi et al., 2020}, and DySAT \cite{Sankar et al., 2020}) and a time-then-graph baseline TGN \cite{Rossi et al., 2020}. Although TGN is indeed a mixture of time-and-graph and time-then-graph, we classify it as time-then-graph since time-and-graph is less expressive and can be regarded as a subset of time-then-graph. For readers not familiar with the above baselines, a short description of the methods is provided in Appendix C.3.

7.3 Hyperparameters

In all methods, we fix the number of GNN layers to \(L = 2\). The node embedding (final representation) has dimension \(d\), with \(d = 2, 6, 10, 16\) in the hyperparameter search. We found that 6 is enough for all datasets. Our hyperparameter grid search is detailed in the Appendix D.2.

We train all models by Adam optimizer \cite{Kingma & Ba, 2014} for 400 epochs with learning rates 0.1, 0.01 and 0.001 defined via hyperparameter search, and fixed weight decay of 0.005.

All hyperparameters are selected according to the best mean validation performance over all train/validation splits.
Table 1: Real-world Dataset Statistics. \(|V|\) is total number of nodes in the aggregated temporal graph. \(|E|\) is total number of edges in the aggregated temporal graph. \(T\) is number of time steps. \(p\) is the node attribute dimension. \(\text{Label}\) is number of different target labels. \(\max(\text{Label})/\min(\text{Label})\) is the proportion between maximum and minimum count of labels.

| Dataset    | \(|V|\)   | \(|E|\)   | \(T\) | \(p\) | \(\text{Label}\) | \(\max(\text{Label})/\min(\text{Label})\) |
|------------|-----------|-----------|-------|-------|-----------------|---------------------------------------------|
| Brain-10   | 5000      | 1761414   | 12    | 20    | 10              | 4.81                                        |
| DBLP-5     | 6606      | 32566     | 10    | 100   | 5               | 3.12                                        |
| COVID-10   | 3000      | 34284     | 14    | 11    | 10              | 82.75                                       |
| Reddit-4   | 8291      | 221089    | 10    | 20    | 4               | 1.00                                        |

7.4 Datasets

All our tasks are classification tasks predicting a future class, based on either a temporal graph (where both node attributes and edges evolve) or a dynamical process on top of a static graph (where only node attributes evolve). A 2-layer MLP is used as the upstream classifier for all tasks. Imbalanced label distributions are taken into consideration in training and evaluation. Table 1 shows statistics of real-world datasets.

Figure 4: ROCAUC Performance Boxplot: The white square is the mean performance of the model. The colored box reflects the first quartile, median, and third quartile from left to right. The x-axis is ROCAUC score weighted (ROCAUC Wtd) by inverse of label counts. Closer to right (larger ROCAUC scores) means better. The bottom four from GRU-D-GCN to GRU-S-GTC are our proposals.

DynamicCSL task. DynamicCSL is a synthetic temporal graph classification task. Each sample is a graph sequence of 8 time steps. At each step, the element is randomly picked from \(\{C_{19.2}, \cdots, C_{19.6}\}\). Here, \(C_{|V|, s}\) means a Circular Skip Link (CSL) graph with \(|V|\) nodes and skip length \(s\). (Also see Figure 2). The goal is to predict number of non-isomorphic CSLs in the temporal graph, e.g., the temporal graph with three snapshots \([C_{19.2}, C_{19.3}, C_{19.3}]\) has label “2”.

Real-world Datasets. All real-world datasets are temporal node classification tasks. Brain-10 is based on fMRI brain scans taken while the subjects were solving tasks. Nodes are voxels in the scan, and temporal edges are based on node activation status in the scan at a certain time interval. The goal is to predict the functionality category of each voxel (node). DBLP-5 is a temporal citation network from 5 related research areas. Each node represents an author, and each edge represents a citation. The goal is to predict the most interested area of each author at the future interval. COVID-10 is a simulation dataset by extended SIR model based on COVID-19 epidemic statistics before 9 Aug 2020. The model simulates the infected status transition for each person based on their contact networks. The latest infected status in an arbitrary interval will be collected as observed attributes at that time. The goal is to predict the infected status of each person at the next time interval given observed status. Reddit-4 is collected from the Reddit social network. Each node represents a post, and a connection is linked if two posts share similar keyword at given interval. The
goal is to predict the status of the post at the future interval.

7.5 Results

The ROCAUC results are provided in Figure 4. Accuracy score results and table of exact values are attached in Appendix D.3. Since some confidence intervals overlap, we also run a paired t-test to better compare the performance of the methods. The paired t-test results are provided in Figure 5 as heatmaps.

Figure 4 shows that all baselines (including the 1-WLGNN-based time-and-graph model) fail in the DynamicCSL task. Our 1-WLGNN-based time-then-graph models are the only ones that can perform the DynamicCSL task. Interestingly, TGN is time-then-graph baseline but it still does not perform well in the DynamicCSL task. This is because TGN only collects edges into a multigraph and does not assign them extra attributes, which is necessary for this task, where one must distinguish non-isomorphic CSLs across multiple time steps.

On the remaining real-world temporal node classification datasets, most of the methods have similar performance (except EvolveGCN, which consistently unperformed). Note that EvolveGCN is similar to a static graph representation method, thus, it struggles to capture node dynamics over time. Our time-then-graph methods are slightly better than the baseline time-and-graph methods on the Brain-10 and Reddit-4 datasets (per t-tests in Figure 5), and with comparable to the state-of-the-art in all other real-world datasets. We note in passing that the Reddit-4 prediction task is difficult because the nodes are only loosely connected and their temporal evolution seems mostly independent. Further analysis of our experiments can be found in Appendix D.3.

8 Conclusions

Although time-and-graph architectures today are the dominant representation methods for temporal graphs, our work showed that time-then-graph representations are an excellent alternative for observational (non-causal) tasks with an importantly upside: Time-then-graph family is more expressive than time-and-graph family if the graph representation uses the 1-WL GNN architecture. Our experiments confirm that our time-then-graph method achieves performance comparable to state-of-the-art time-and-graph methods on several real-world datasets, and clearly superior performance on a hard-to-represent synthetic task.

References

Saoussen Aouay, Sahma Jamoussi, Faiez Gargouri, and Ajith Abraham. Modeling dynamics of social networks: A survey. In 2014 6th International Conference on Computational Aspects of Social Networks, pp. 49–54. IEEE, 2014.
Waiss Azizian and Marc Lelarge. Characterizing the expressive power of invariant and equivariant graph neural networks. *arXiv preprint arXiv:2006.15646*, 2020.

Peter W Battaglia, Razvan Pascanu, Matthew Lai, Danilo Rezende, and Koray Kavukcuoglu. Interaction networks for learning about objects, relations and physics. *Advances in Neural Information Processing Systems*, 2016.

Jinyin Chen, Xuanheng Xu, Yangyang Wu, and Haibin Zheng. Ge-lstm: Graph convolution embedded lstm for dynamic link prediction. *arXiv preprint arXiv:1812.04206*, 2018.

Jinyin Chen, Jian Zhang, Xuanheng Xu, Chenbo Fu, Dan Zhang, Qingpeng Zhang, and Qi Xuan. E-lstm-d: A deep learning framework for dynamic network link prediction. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 2019.

Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. *arXiv preprint arXiv:1606.09375*, 2016.

Zulong Diao, Xin Wang, Dafang Zhang, Yingru Liu, Kun Xie, and Shaoyao He. Dynamic spatial-temporal graph convolutional neural networks for traffic forecasting. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pp. 890–897, 2019.

Brendan L Douglas. The weisfeiler-lehman method and graph isomorphism testing. *arXiv preprint arXiv:1101.5211*, 2011.

Michael Eichler. *Causal inference in time series analysis*. Wiley Online Library, 2012.

Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *International Conference on Machine Learning*, pp. 1263–1272. PMLR, 2017.

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

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

Ehsan Hajiramezanali, Arman Hasanzadeh, Nick Duffield, Krishna R Narayanan, Mingyuan Zhou, and Xiaoning Qian. Variational graph recurrent neural networks for traffic forecasting. *arXiv preprint arXiv:1908.09710*, 2019.

Zhen Han, Yunpu Ma, Yuyi Wang, Stephan Günnemann, and Volker Tresp. Graph hawkes neural network for forecasting on temporal knowledge graphs. In *Automated Knowledge Base Construction*, 2020.

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

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation, 9(8):1735–1780, 1997.

Petter Holme. Modern temporal network theory: a colloquium. *The European Physical Journal B*, 88(9):234, 2015.

Petter Holme and Jari Saramäki. Temporal networks. *Physics reports*, 519(3):97–125, 2012.
Woojeong Jin, He Jiang, Meng Qu, Tong Chen, Changlin Zhang, Pedro Szekely, and Xiang Ren. Recurrent event network: Global structure inference over temporal knowledge graph. *arXiv preprint arXiv:1904.05530*, 2019.

Seyed Mehran Kazemi, Rishab Goel, Kshitij Jain, Ivan Kobyzev, Akshay Sethi, Peter Forsyth, and Pascal Poupart. Representation Learning for Dynamic Graphs: A Survey. *Journal of Machine Learning Research*, 21(70):1–73, 2020.

Joe Kileel, Matthew Trager, and Joan Bruna. On the expressive power of deep polynomial neural networks. *arXiv preprint arXiv:1905.12207*, 2019.

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. *arXiv preprint arXiv:1609.02907*, 2016.

Lauri Kovanen, Márton Karsai, Kimmo Kaski, János Kertész, and Jari Saramäki. Temporal motifs in time-dependent networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2011(11):P11005, 2011.

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

Kai Lei, Meng Qin, Bo Bai, Gong Zhang, and Min Yang. Gcn-gan: A non-linear temporal link prediction model for weighted dynamic networks. In *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*, pp. 388–396. IEEE, 2019.

Jia Li, Zhichao Han, Hong Cheng, Jiao Su, Pengyun Wang, Jianfeng Zhang, and Lujia Pan. Predicting path failure in time-evolving graphs. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 1279–1289, 2019.

Yaguang Li, Rose Yu, Cyrus Shahabi, and Yan Liu. Diffusion convolutional recurrent neural network: Data-driven traffic forecasting. In *International Conference on Learning Representations*, 2018.

Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel. Gated graph sequence neural networks. *arXiv preprint arXiv:1511.05493*, 2015.

Xiaoke Ma, Penggang Sun, and Yu Wang. Graph regularized nonnegative matrix factorization for temporal link prediction in dynamic networks. *Physica A: Statistical mechanics and its applications*, 496:121–136, 2018.

Yao Ma, Ziyi Guo, Zhaocun Ren, Jiliang Tang, and Dawei Yin. Streaming graph neural networks. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 719–728, 2020.

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

Haggai Maron, Heli Ben-Hamu, Hadar Serviansky, and Yaron Lipman. Provably powerful graph networks. *arXiv preprint arXiv:1905.11136*, 2019a.

Haggai Maron, Ethan Fetaya, Nimrod Segol, and Yaron Lipman. On the universality of invariant networks. In *International conference on machine learning*, pp. 4363–4371. PMLR, 2019b.

Hongyuan Mei and Jason Eisner. The neural hawkes process: A neurally self-modulating multivariate point process. *arXiv preprint arXiv:1612.09328*, 2016.
Christopher Morris, Martin Ritzert, Matthias Fey, William L Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe. Weisfeiler and leman go neural: Higher-order graph neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pp. 4602–4609, 2019.

Ryan Murphy, Balasubramaniam Srinivasan, Vinayak Rao, and Bruno Ribeiro. Relational pooling for graph representations. In *International Conference on Machine Learning*, pp. 4663–4673. PMLR, 2019.

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, pp. 5363–5370, 2020.

Judea Pearl. *Causality*. Cambridge university press, 2009.

Jonas Peters, Joris Mooij, Dominik Janzing, and Bernhard Schölkopf. Identifiability of causal graphs using functional models. *UAI*, 2011.

Mahmudur Rahman, Tanay Kumar Saha, Mohammad Al Hasan, Kevin S Xu, and Chandan K Reddy. Dylin2vec: Effective feature representation for link prediction in dynamic networks. *arXiv preprint arXiv:1804.05755*, 2018.

Giulio Rossetti and Rémy Cazabet. Community discovery in dynamic networks: a survey. *ACM Computing Surveys (CSUR)*, 51(2):1–37, 2018.

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

Aravind Sankar, Yanhong Wu, Lianqou Wei, 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*, pp. 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*, pp. 362–373. Springer, 2018.

Jin Shang and Mingxuan Sun. Geometric hawkes processes with graph convolutional recurrent neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pp. 4878–4885, 2019.

Hava T Siegelmann and Eduardo D Sontag. On the computational power of neural nets. *Journal of computer and system sciences*, 50(1):132–150, 1995.

Uriel Singer, Ido Guy, and Kira Radinsky. Node embedding over temporal graphs. *arXiv preprint arXiv:1903.08889*, 2019.

Joakim Skarding, Bogdan Gabrys, and Katarzyna Musial. Foundations and modelling of dynamic networks using dynamic graph neural networks: A survey. *arXiv preprint arXiv:2005.07496*, 2020.

Balasubramaniam Srinivasan and Bruno Ribeiro. On the equivalence between positional node embeddings and structural graph representations. *arXiv preprint arXiv:1910.00452*, 2019.

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

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *arXiv preprint arXiv:1706.03762*, 2017.

Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.
Yun Xiong, Yao Zhang, Hanjie Fu, Wei Wang, Yangyong Zhu, and S Yu Philip. Dyngraphgan: Dynamic graph embedding via generative adversarial networks. In *International Conference on Database Systems for Advanced Applications*, pp. 536–552. Springer, 2019.

Da Xu, Chuanwei Ruan, Evren Korpeoglu, Sushant Kumar, and Kannan Achan. Inductive representation learning on temporal graphs. *arXiv preprint arXiv:2002.07962*, 2020.

Dongkuan Xu, Wei Cheng, Dongsheng Luo, Xiao Gu, Yamengand Liu, Jingchao Ni, Bo Zong, Haifeng Chen, and Xiang Zhang. Adaptive neural network for node classification in dynamic networks. In *2019 IEEE International Conference on Data Mining (ICDM)*, pp. 1402–1407. IEEE, 2019.

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826*, 2018.

Keyulu Xu, Jingling Li, Mozhi Zhang, Simon S Du, Ken-ichi Kawarabayashi, and Stefanie Jegelka. How neural networks extrapolate: From feedforward to graph neural networks. *ICLR*, 2021.

Bing Yu, Haoteng Yin, and Zhanxing Zhu. Spatio-temporal graph convolutional networks: A deep learning framework for traffic forecasting. *arXiv preprint arXiv:1709.04875*, 2017.

Ziwei Zhang, Peng Cui, Jian Pei, Xiao Wang, and Wenwu Zhu. Timers: Error-bounded svd restart on dynamic networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

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, 2018.
A Notation Table

Table 2 summarizes our notations.

| Notation | Meaning |
|----------|---------|
| v, i, j  | Random variable of node. |
| T        | Maximum number of time steps. |
| G        | Aggregated temporal graph. |
| V, E     | Node and edge sets of aggregated temporal graph over all time steps. |
| N(i)     | 1-hop neighbors of node i ∈ V in aggregated temporal graph over all time steps. |
| X_{t≤T}, A_{t≤T} | Node and edge attributes of aggregated temporal graph of all T time steps. |
| t        | Random variable of time step. |
| G_t      | Temporal graph snapshot at time step 1 ≤ t ≤ T. |
| V_t, E_t | Node and edge sets of temporal graph snapshot G_t. |
| N(i, t)  | 1-hop neighbors of node i ∈ V in temporal graph snapshot G_t. |
| X_{t, i}, A_{t, i} | Node and edge attributes of temporal graph snapshot G_t. |
| p, d     | Dimension of observational attributes and neural network representations (embeddings). |
| G_{V, t, p} | All (static) graphs on node set V whose node and edge attributes are vectors from \( \mathbb{R}^p \). |
| T_{V, t, p} | All temporal graphs on node set V over T time steps whose node and edge attributes are vectors from \( \mathbb{R}^p \). |
| U        | A generic domain. |
| u        | A generic random variable in domain U. |
| f, r, g, u | Learnable functions according to context, e.g., MLPs. |
| I(f, U)  | Identifiable set of f on domain U. |
| π        | An arbitrary node index permutation over \([1, \ldots, |V|]\). |
| \( \not\in \) | Relational operators of expressiveness on domain U. |
| S_t      | Sequence element at time step (position) t. |
| H        | Neural network representations (embeddings) given by RNN. |
| L        | Maximum number of GNN layers. |
| l        | Random variable of GNN layer index. |
| M, E, F  | Message outputs inside GNN. |
| H, I, J  | Neural network representations (embeddings) given by GNN. |
| C_{n, s} | Circular Skip Link graph \cite{Murphy2019} of n nodes with skip length s. |

B Proves

B.1 Identifiable Set Equivalence

Lemma 2 (Identifiable Set Equivalence). Suppose \( \mathbb{I}_1(f, U) \) and \( \mathbb{I}_2(f, U) \) are two identifiable sets of representation function f on domain U, then there exists a bijection between \( \mathbb{I}_1(f, U) \) and \( \mathbb{I}_2(f, U) \) based on matching between representations.

Proof. We show this by contradiction. Assume that there is no such bijection by matching representations between \( \mathbb{I}_1(f, U) \) and \( \mathbb{I}_2(f, U) \).

Without loss of generality, suppose there is an element \( u_1 \in \mathbb{I}_1(f, U) \) such that no element from \( \mathbb{I}_2(f, U) \) can match its representation, in other words, \( \exists u_1 \in \mathbb{I}_1(f, U), \forall u_2 \in \mathbb{I}_2(f, U), f(u_1) \neq f(u_2) \). This also implies \( u_1 \notin \mathbb{I}_2(f, U) \) otherwise \( u_1 \in \mathbb{I}_2(f, U) \) can match with \( u_1 \in \mathbb{I}_1(f, U) \) on representations. Thus, \( u_1 \in X - \mathbb{I}_2(f, U) \).

According to Definition 2 for \( u_1 \in X - \mathbb{I}_2(f, U) \), \( \exists u'_2 \in \mathbb{I}_2(f, U), f(u_1) = f(u'_2) \). This leads to a contradiction since we assume that no elements from \( \mathbb{I}_2(f, U) \) can match with \( u_1 \) on its representation \( (u_1) \), but \( u'_2 \in \mathbb{I}_2(f, U) \) did.
So, for any elements in \( \mathbb{I}_1(f, U) \), there must exists an element in \( \mathbb{I}_2(f, U) \) matching its representation. And, this mapping is an injection from \( \mathbb{I}_1(f, U) \) to \( \mathbb{I}_2(f, U) \) since \( \mathbb{I}_1(f, U) \) does not contain elements of the same representation based on Definition 2. The same applies flipping the roles of \( \mathbb{I}_2(f, U) \) and \( \mathbb{I}_1(f, U) \).

Hence, by the Schr¨ oder-Bernstein theorem, if there is an injection from \( \mathbb{I}_1(f, U) \) to \( \mathbb{I}_2(f, U) \) and an injection from \( \mathbb{I}_2(f, U) \) to \( \mathbb{I}_1(f, U) \), then there is a bijection between \( \mathbb{I}_1(f, U) \) and \( \mathbb{I}_2(f, U) \).

\[ \square \]

### B.2 Temporal 1-WL Expressiveness

**Theorem 1. [Temporal 1-WL Expressiveness]** Time-then-graph representation family is strictly more expressive than time-and-graph representation family when the graph representation is a 1-WL GNN, i.e.,

\[
\text{time-and-1WLGNN} \not\subseteq_{T|V,T,p} \text{time-then-1WLGNN}. 
\]

**Proof.** Roadmap: We first show that we can construct an equivalent time-then-1WLGNN representation function for any time-and-1WLGNN representation on any temporal graph domain \( T|V,T,p \). Thus, time-then-1WLGNN is as expressive as time-and-1WLGNN. Furthermore, we provide an instance where time-and-1WLGNN fails to distinguish two non-equivalent temporal graphs but time-then-1WLGNN can. This shows that it is strictly more expressive, in other words,

\[
\text{Time-and-1WLGNN} \not\subseteq_{T|V,T,p} \text{Time-then-1WLGNN}. 
\]

First, we will show that any time-and-graph representation function has an equivalent time-then-graph function. First, we start with the definition of time-and-graph:

**Time-and-graph.** The definition of time-and-graph is given by Equation (6):

\[
H_{i,t} = \begin{cases} 
\text{RNNCell} \left( \left[ \text{GNN}_{\text{in}}^L (\mathbf{X}_{i,t}, \mathbf{A}_{\cdot ; t}) \right]_{i \in V} \right), \\
\left[ \text{GNN}_{\text{rec}}^L (\mathbf{H}_{i,t-1}, \mathbf{A}_{\cdot ; t}) \right]_{i \in V}, & 1 \leq t \leq T, \\
0, & t = 0,
\end{cases}
\]

The above definition has two different graph neural networks \( \text{GNN}_{\text{in}}^L \) and \( \text{GNN}_{\text{rec}}^L \). These represent encoding raw node attributes at time \( t \) and encoding node historical representations before \( t \) given topology \( \mathbf{A}_{\cdot ; t} \) at snapshot \( t \), respectively. Also, we revisit the definition of a \( L \)-layer graph neural network \( \text{GNN}^L \) in Equation (5):

\[
\mathbf{M}_i^{(l)} = \sum_{j \in N(i)} g^{(l)} (\mathbf{Z}_i^{(l-1)}, \mathbf{Z}_j^{(l-1)}, \mathbf{A}_{i,j}),
\]

where

\[
\mathbf{Z}_i^{(l)} = \begin{cases} 
g^{(l)} (\mathbf{Z}_i^{(l-1)}, \mathbf{M}_i^{(l)}), & l > 0, \\
\mathbf{X}_i, & l = 0.
\end{cases}
\]

The final time-and-graph architecture is shown in Figure 6 and described in detail next:

1. **Description of \( \text{GNN}_{\text{in}}^L \):** This GNN represents the input features at time \( t \in \{1, \ldots, T\} \). We denote the \( \text{GNN}_{\text{in}}^L \) definition at each layer \( l \in \{1, \ldots, L\} \) as

\[
\mathbf{E}_{i,t}^{(l)} = \sum_{j \in N(i)} g_{\text{in}}^{(l)} (\mathbf{l}_{i,t}^{(l-1)}, \mathbf{E}_{j,t}^{(l-1)}, \mathbf{A}_{i,j,t}),
\]

where

\[
\mathbf{l}_{i,t}^{(l)} = \begin{cases} 
g_{\text{in}}^{(l)} (\mathbf{l}_{i,t}^{(l-1)}, \mathbf{E}_{i,t}^{(l)}), & l > 0, \\
\mathbf{X}_{i,t}, & l = 0,
\end{cases}
\]

(9)
and \( g_{in}(l), u_{in}(l) \) are arbitrary learnable functions (e.g., MLPs). \( l_{i,t}^{(l)} \) is internal (hidden) state of GNN\(_{in}^{(l)} \) at layer \( l \) for node \( i \) at time step \( t \). And \( l_{i,t}^{(0)} = X_{i,t} \) is a special initialization case. Finally, the output for node \( i \in V \) is \( [GNN_{in}^{(l)}(X_{i,t}, A_{i,:}, t)]_i := l_{i,t}^{(l)}. \)

2. Description of GNN\(_{rec}^L \): We denote the GNN\(_{rec}^{(l)} \) definition at each layer \( l \in \{1, \ldots, L\} \) as

\[
F_{i,t}^{(l)} = \sum_{j \in V(l)} g_{rec}^{(l)}(J_{i,t}^{(l-1)}, E_{j,t}, A_{i,j,t}),
\]

where

\[
J_{i,t}^{(l)} = \begin{cases} 
  u_{rec}^{(l)}(J_{i,t}^{(l-1)}, E_{i,t}), & l > 0, \\
  H_{i,t-1}, & l = 0,
\end{cases}
\tag{10}
\]

and \( g_{rec}, u_{rec} \) are arbitrary learnable functions (e.g., MLPs). \( J_{i,t}^{(l)} \) is internal (hidden) state of GNN\(_{rec}^{(l)} \) at layer \( l \) for node \( i \) at time step \( t \). And \( J_{i,t}^{(0)} = H_{i,t-1} \) is a special initialization case (\( H_{i,0} = 0 \) as defined earlier). Finally, the output for node \( i \in V \) is \( [GNN_{rec}^{(l)}(H_{i,t-1}, A_{i,:})]_i := J_{i,t}^{(L)}. \)

Next, for the sake of simplicity, we redefine \( H_{i,t} \) of Equation (6) as (see the definitions of \( I_{i,t}^{(L)} \) and \( J_{i,t}^{(L)} \) right above in Equations (9) and (10)):

\[
H_{i,t} = \begin{cases} 
  \text{RNNCell}\left(I_{i,t}^{(L)}, J_{i,t}^{(L)}\right), & t > 0, \\
  0, & t = 0.
\end{cases}
\tag{11}
\]

Equations (9) to (11) give a general description of existing architectures for \textit{time-and-graph} representation functions. Finally, \( H_{i,T} \) is the final temporal representation of node \( i \in V \) and can be used to predict its future node attributes.

\textbf{Time-then-graph.} Our next target is to construct a \textit{time-then-graph} representation function. We will show it can give the same output \( H_{i,T} \) for the same inputs \( [X_{i,t}]_{\forall i \in V, 1 \leq t \leq T}, [A_{i,j,t}]_{\forall (i, j) \in E, 1 \leq t \leq T} \).

Let’s first revisit the definition of \textit{time-then-graph} representation in Equation (8):

\[
\begin{align*}
H_{i}^Y &= \text{RNN}^Y(X_{i,:}^{\leq T}), \quad \forall i \in V, \\
H_{i,j}^E &= \text{RNN}^E(A_{i,j,:}^{\leq T}), \quad \forall (i, j) \in E, \\
Z &= \text{GNN}_{agg}^{TL}(H^Y, H^E).
\end{align*}
\]

\cite{Siegelmann1995} shows that with enough hidden neurons, an RNN can be a most-expressive sequence model (universal Turing machine approximator). Hence, since the input representation is perfect, we can equivalently think of the RNN output simply as a “copy” of the input, i.e., \( H^Y_i \) perfectly represents \( X_{i,:}^{\leq T} \) and \( H^E_{i,j} \) perfectly represents \( A_{i,j,:}^{\leq T} \).

Hence, a \textit{time-then-graph} representation can be described as a graph neural network applied over the aggregated node attribute and adjacency matrix inputs:

\[
Z' = \text{GNN}_{agg}^{TL}(X_{i,:}^{\leq T}, A_{i,:}^{\leq T}),
\]

which will we describe next as giving the same output \( Z' = H_{i,T} \) for any nodes \( i \in V \).

\textbf{The GNN\(_{tar}^{TL} \) representation.} The basic idea is straightforward: GNN\(_{tar}^{TL} \) has \( TL \) layers and emulates a \textit{time-and-graph} representation. Between layers \( l = \{ (\tau - 1) + 1, \ldots, \tau L \} \) layers, the GNN will focus on the node attributes \( X_{i,:}\) and edges \( A_{i,:}\), \( \tau = 1, \ldots, T \), from the complete sequences \( X_{i,:}^{\leq T} \) and \( A_{i,:}^{\leq T} \). Clearly, this GNN can learn to output \( H_{i,T} \), \( \forall i \in V \).

Let \( N(i) = \{ \forall j \in V | \exists \tau, 1 \leq \tau \leq T, A_{i,j,\tau} > 0 \} \) be the set of neighbors of \( i \in V \) in the aggregated adjacency matrix, and \( N_{\tau}(i) \) is the neighbor set of node \( i \) at snapshot \( A_{i,:}\). Inside our GNN we will denote the neighbors as \( \tilde{N}_{\tau}(i) \), which means neighbor set at snapshot \( \tau \).

In what follows, \( \| \) denotes a vector concatenation and \( l\% \) denotes

\[
l\% = \begin{cases} 
  l \mod L, & l \mod L \neq 0, \\
  L, & l \mod L = 0.
\end{cases}
\]
We also denote \( g_{\text{in}} \), \( u_{\text{in}} \), \( g_{\text{rec}} \), and \( u_{\text{rec}} \) as arbitrary learnable functions (much like the ones in Equations (9) and (10) at layer \( l \)). We use a prime superscript to variables that are the corresponding variables to the ones in Equations (9) to (11). We will construct \( g_{\text{tar}}^{(l)} \) and \( u_{\text{tar}}^{(l)} \) for \( \text{GNN}_{\text{tar}}^{T,L} \) by exploiting this new notations. Figures 7 and 8 illustrate the construction we will describe in Equations (12) and (13).

The output of layer \( l \in \{1, \cdots, TL\} \) of our \( \text{GNN}_{\text{tar}}^{T,L} \) is split into three parts: \( [X_{i,t \leq T} \| I_{i}^{(l)} \| J_{i}^{(l)}] \) based on their usage.

1. \( X_{i,t \leq T} \) is the attribute of node \( i \) in aggregated temporal graph, and it can also be regarded as the concatenation of all attributes of node \( i \) of all time steps \( 1 \leq t \leq T \);
2. \( I_{i}^{(l)}, J_{j}^{(l)} \) is used to achieve the same representations as \( I_{i,[\frac{l}{L}]}^{(l)}, J_{j,[\frac{l}{L}]}^{(l)} \) as in Equations (9) and (10), as shown later.

The learnable function \( g_{\text{tar}}^{(l)} \) takes the concatenation of three parts as input of node \( j \): a concatenation of node attributes over all \( T \) time steps, \( I_{j}^{(l-1)}, J_{j}^{(l-1)} \) that mimic \( I_{i,[\frac{l}{L}]}^{(l-1)}, J_{j,[\frac{l}{L}]}^{(l-1)} \) as in Equations (9) and (10). Then, \( g_{\text{tar}}^{(l)} \) outputs \( [E_{i}^{(l)} \| F_{i}^{(l)}] \) that mimic the representations of \( E_{i,[\frac{l}{L}]}^{(l)}, F_{i,[\frac{l}{L}]}^{(l)} \) as in Equations (9) and (10).

We also define the input of \( g_{\text{tar}}^{(l)} \) as the initialization status of our construction
\[
[X_{i,t \leq T} \| I_{i}^{(0)} \| J_{i}^{(0)}] = [X_{i,t \leq T} \| X_{i,1} \| 0],
\]
for any node \( i \).

In the following Equations (12) and (13) we give further details of functions \( g_{\text{tar}}^{(l)} \) and \( u_{\text{tar}}^{(l)} \) of \( \text{GNN}_{\text{tar}}^{T,L} \):

\[
\begin{align*}
\sum_{j \in \mathcal{N}(i)} g_{\text{tar}}^{(l)}(X_{j,t \leq T} \| I_{j}^{(l-1)} \| J_{j}^{(l-1)}, A_{i,j,t \leq T})
&= \left[ \sum_{j \in \mathcal{N}(i)} g_{\text{in}}^{(l)}(I_{i}^{(l-1)}, I_{j}^{(l-1)}, A_{i,j,[\frac{l}{L}]}), \right. \\
&\left. \sum_{j \in \mathcal{N}(i)} g_{\text{rec}}^{(l)}(J_{i}^{(l-1)}, J_{j}^{(l-1)}, A_{i,j,[\frac{l}{L}]}), \right] \\
&= [E_{i}^{(l)} \| F_{i}^{(l)}],
\end{align*}
\]

\[
\begin{align*}
u_{\text{tar}}^{(l)}([X_{i,t \leq T} \| I_{i}^{(l-1)} \| J_{i}^{(l-1)}, E_{i}^{(l)} \| F_{i}^{(l)}])
&= \begin{cases}
[X_{i,t \leq T} \| u_{\text{in}}^{(l)}(I_{i}^{(l-1)}, E_{i}^{(l)}), u_{\text{rec}}^{(l)}(J_{i}^{(l-1)}, F_{i}^{(l)})], & l \mod L \neq 0, \\
[X_{i,t \leq T} \| X_{i,\frac{l}{L}+1} \| \text{RNNCell}(u_{\text{in}}^{(l)}(I_{i}^{(l-1)}, E_{i}^{(l)}), u_{\text{rec}}^{(l)}(J_{i}^{(l-1)}, F_{i}^{(l)})), & l \mod L = 0,
\end{cases}
\end{align*}
\]

Finally, we are going to show the above construction can give the same output \( H_{i,T} \) as Equation (9) to Equation (11) for all nodes \( i \in \mathcal{V} \).

The GNN output given by Equations (12) and (13) can be the same output as \( H_{i,T} \) of Equations (9) to (11). We now show by induction that \( J_{i}^{(TL)} = H_{i,t} \) for any time step \( t \in \{1, \cdots, T\} \). First, at the first layer \((l \leftarrow 1)\) for any node \( i \in \mathcal{V} \). Based on Equation (13), we have initial input
\[
[X_{i,t \leq T} \| X_{i,1} \| 0].
\]
Then, we have the equivalent operations between Equations (12) and (13) and Equations (9) to (11) on the initial inputs.

\[ I_i^{(0)} = X_{i,1} = I_i^{(0)}_1, \quad \forall i \in \mathbb{V}, \]
\[ J_i^{(0)} = 0 = J_i^{(0)}_1, \quad \forall i \in \mathbb{V}. \]

Now, suppose that

\[ I_i^{(l-1)} = I_i^{(l-1)}_1, \quad \forall i \in \mathbb{V}, \]
\[ J_i^{(l-1)} = J_i^{(l-1)}_1, \quad \forall i \in \mathbb{V}, \] (14)

for \( 1 \leq l \leq TL \).

Next, we show that the output of Equation (12) is the same as outputs of Equations (9) and (10) given the condition in Equation (14).

\[ E_i^{(l)} = \sum_{j \in \mathcal{N}_i^{l-1} + i} g_{in}^{(l)}(i, j, A_{i,j,\frac{l}{l}}) \]
\[ = \sum_{j \in \mathcal{N}_i^{l-1} + i} g_{in}^{(l)}(i, j, A_{i,j,\frac{l}{l}}) \]
\[ = E_i^{(l)} \]
\[ F_i^{(l)} = \sum_{j \in \mathcal{N}_i^{l-1} + i} g_{rec}^{(l)}(i, j, A_{i,j,\frac{l}{l}}) \]
\[ = \sum_{j \in \mathcal{N}_i^{l-1} + i} g_{rec}^{(l)}(i, j, A_{i,j,\frac{l}{l}}) \]
\[ = F_i^{(l)} \] (15)

Next, we show that the output of Equation (13) is the same as outputs of Equation (9) and Equation (10) given the conditions in Equations (14) and (15). Pay attention that there are two branches in Equation (13), and we will show them one by one.

When \( l \) mod \( L \neq 0 \),

\[ I_i^{(l)} = u_{in}^{(l)}(i, E_i^{(l)}) \]
\[ = u_{in}^{(l)}(i, E_i^{(l)}) \]
\[ = I_i^{(l)}_{i,\frac{i+1}{L}} = I_i^{(l)}_{i,\frac{i}{L+1}} \] (16)

and

\[ J_i^{(l)} = u_{rec}^{(l)}(j, F_i^{(l)}) \]
\[ = u_{rec}^{(l)}(j, F_i^{(l)}) \]
\[ = J_i^{(l)}_{i,\frac{i}{L}} = J_i^{(l)}_{i,\frac{i+1}{L}} \] (17)

We can see that Equations (16) and (17) are indeed Equation (14) in our induction. When \( l \) mod \( L = 0 \),

\[ I_i^{(l)} = X_{i,\frac{i+1}{L}+1} = I_i^{(0)}_{i,\frac{i+1}{L}+1} \] (18)
and

\[ J_i^{(l)} = \text{RNNCell}(u_i^{(l)}(I_i^{(l-1)}, E_i^{(l)}), \]
\[ u_i^{(l)}(J_i^{(l-1)}, F_i^{(l)}) = \text{RNNCell}(u_i^{(l)}(I_i^{(l-1)}, E_i^{(l)}), \]
\[ u_i^{(l)}(J_i^{(l-1)}, F_i^{(l)})) = H_{i,[\frac{L}{i}, \frac{L}{i}]} = J_i^{(0)} \] (19)

We can see that Equations (18) and (19) also corresponds to Equation (14) in our induction.

We now continue by induction until layer \( l = TL \), where we stop, where Equation (19) yields

\[ J_i^{(TL)} = H_{i,T}. \]

This implies that we get the targeting \textit{time-and-graph} representation output from constructed (static) GNN output for an arbitrary node \( i \in \mathcal{V} \).

The above shows that any \textit{arbitrary time-and-graph} representation function relying on 1-WL GNNs (Equations (9) to (11)) can be emulated by a \textit{time-then-graph} representation function (Equations (12) and (13)), which outputs the same representations for the same temporal graph inputs. Thus, \textit{time-then-1WLGGNN} is at least the same expressive as \textit{time-and-1WLGGNN}.

Next, we show a specific task (illustrated in Figure 9) where \textit{time-and-1WLGGNN} is more expressive than \textit{time-and-1WLGGNN}.

We now propose a task, whose temporal graph is illustrated in Figure 9. The goal is to differentiate between two 2-step temporal graphs. Each snapshot is a Circular Skip Link (CSL) graph (see Murphy et al. (2019) with 7 nodes, denoted \( C_{7,s} \), where \( s \) denotes the number of nodes between two neighbors on the outer circle. These two temporal graphs differ in their second time step. From Murphy et al. (2019), if the CSL graphs are unattributed, any 1-WL GNN will output the same representations for both \( C_{7,1} \) and \( C_{7,2} \). We use \( A^{(a)} \) to represent the adjacency matrix of dynamics (a), and \( A^{(b)} \) for dynamics (b) in Figure 9. Note that \( X^{(a)} = X^{(b)} = 0 \) since the temporal graph is unattributed.

Hence, for a \textit{time-and-1WLGGNN} representation

\[ \text{GNN}^L_{in}(X^{(a)}, A^{(a)}) = \text{GNN}^L_{in}(X^{(b)}, A^{(b)}) \]

\[ = \text{GNN}^L_{in}(X^{(b)}, A^{(b)}) \]

\[ = \text{GNN}^L_{rec}(H^{(a)}, A^{(a)}) = \text{GNN}^L_{rec}(0, A^{(a)}) \]

\[ = \text{GNN}^L_{rec}(H^{(b)}, A^{(b)}) = \text{GNN}^L_{rec}(0, A^{(b)}) \]

Then, when we apply Equation (11) at the first time step, we will get the same hidden representation \( H_{i,1}^{(a)} \) and \( H_{i,1}^{(b)} \). Thus, at the second (also the last) time step, we also have

\[ \text{GNN}^L_{rec}(H_{i,1}^{(a)}, A_{i,2}^{(a)}) = \text{GNN}^L_{rec}(H_{i,1}^{(b)}, A_{i,2}^{(b)}) \]

This also results in the same hidden representation \( H_{i,2}^{(a)} \) and \( H_{i,2}^{(b)} \).

Thus, \textit{time-and-1WLGGNN} will output the same representation for the two different temporal graphs (a) and (b) in Figure 9.

Finally, \textit{time-then-1WLGGNN} will work directly on aggregated graph, illustrated by Figure 9(c). Thus, here, we can manually verify that a 1-WL GNN can distinguish these two graphs, implying that a \textit{time-then-1WLGGNN} representation can distinguish two temporal graphs that \textit{time-and-1WLGGNN} cannot distinguish.

Finally, we conclude:
1. The time-then-1WLGNN representation family is at least as expressive as the time-and-1WLGNN representation family;
2. The time-then-1WLGNN can distinguish temporal graphs not distinguished by time-and-1WLGNN. Thus, time-then-1WLGNN is strictly more expressive than time-and-1WLGNN. More precisely,
\[ \text{Time-and-1WLGNN} \preceq \text{Time-then-1WLGNN}, \]
concluding our proof.

B.3 Temporal Most Expressiveness

Corollary 1. [Temporal Expressiveness] Using more expressive GNN (e.g., k-WL GNN of [Morris et al., 2019]) will improve the expressiveness of time-then-graph, and if a most-expressive GNN (GNN+) (e.g., Murphy et al. (2019) or Maron et al. (2019a)) are used, time-then-graph and time-and-graph representation families will both be most-expressive.

\[ \text{time-then-1WLGNN} \preceq \text{time-then-kWLGNN} \]
\[ \preceq \text{time-then-GNN}^+ \preceq \text{time-and-GNN}^+, \]

Proof. In the proof of Theorem 1 we have noted that we can think the RNNs of time-then-graph as capable of perfectly copying the sequence into a representation, which ensures that the time-then-graph expressiveness is equivalent to the expressiveness of a (static) GNN whose node and edge attributes are their respective temporal sequences. In Morris et al. (2019), it is shown that 1-WL GNN \( \preceq G_{|V|,p}^k \)-WL GNN. And Murphy et al. (2019), Maron et al. (2019a) both define most-expressive graph representations—which we denote as GNN+. Using these known results we have \( k\text{-WL GNN} \preceq G_{|V|,p}^k \text{GNN}^+ \) respectively, which then yields

\[ \text{time-then-1WLGNN} \preceq \text{time-then-kWLGNN} \]
\[ \preceq \text{time-then-GNN}^+ \preceq \text{time-and-GNN}^+, \]

and time-then-GNN+ is also the most expressive representation family on \( T_{|V|,T,p} \).

On the other hand, if we have the most expressive graph representation GNN+, then we can have an injection from any non-isomorphic snapshots to unique representations. Since RNN is also the most expressive for finite-length sequences, there also exists an injection from finite-length sequence of snapshot representations to a final time-and-GNN+ representation. Combining these two injections together, we can always get a representation injection from temporal graph space \( T_{|V|,T,p} \), thus time-and-GNN+ is also the most expressive representation family.

Hence, both time-then-GNN+ and time-and-GNN+ are most-expressive representations of temporal attributed graphs with discrete time steps, i.e.,

\[ \text{time-then-1WLGNN} \preceq \text{time-then-kWLGNN} \]
\[ \preceq \text{time-then-GNN}^+ \preceq \text{time-and-GNN}^+. \]

B.4 Improper Link Prediction

Theorem 2. [Temporal Link Prediction Impossibility] No equivariant temporal node representation (be in time-then-graph or time-and-graph family) is equipped to correctly predict temporal links.

Proof. Our proof relies on a counterexample.

We start with the statement that an equivariant graph representation is a structural node representation, since a permutation acts on the nodes (see Srinivasan & Ribeiro (2019) for a description of the difference between structural and positional node representations).

Figure 10 shows a temporal graph with two disconnected components (blue and orange) in a food web with two isolated ecosystems. At each time step, the two disconnected components have the same topology. Then,
the sequences of snapshots of the aggregated temporal graphs will be the same in these two disconnected components. This means that the time-then-graph static graphs of the two disconnected components (land and sea) are the same at time $t_2$. We can now invoke Srinivasan & Ribeiro (2019) for static attributed graphs to declare that the Orca and Lynx will receive the same time-then-graph most-expressive node representation at time $t_2$. Finally, by Corollary [1] the most-expressive time-then-graph representation is as expressive as the most-expressive time-and-graph representation. Hence, the Orca and Lynx at step $t_2$ will also have the same temporal time-and-graph node representation.

Since Orca and Lynx have the same temporal node representations, the method will incorrectly predict predatory links between two isolated ecosystem. Thus, time-and-graph and time-then-graph temporal node representations are not designed to predict temporal links.

\[ \square \]

C A Description of Standard Neural Networks

C.1 RNN

In this subsection, we will briefly introduce GRU (Cho et al., 2014) and Self-Attention (Vaswani et al., 2017) used in this paper, for those not familiar with these architectures.

**GRU.** Gated Recurrent Unit (GRU) is defined as

\[
\begin{align*}
    u_t &= \sigma(W^u x_t + V^u h_{t-1} + b^u), \\
    r_t &= \sigma(W^r x_t + V^r h_{t-1} + b^r), \\
    c_t &= \phi(W^c x_t + V^c (r_t \odot h_{t-1}) + b^c), \\
    h_t &= (1 - u_t) \odot h_{t-1} + u_t \odot c_t,
\end{align*}
\]

where $x_t$ is an input vector at time step $t$, $h_{t-1}$ is historical (hidden) representation from previous time steps, $h_t$ is representation at current time step, and $\sigma, \phi, \odot$ are sigmoid, tanh and element-wise product, respectively. $W^u, W^r, W^c, V^u, V^r, V^c, b^u, b^r, b^c$ are learnable parameters and $u_t, r_t, c_t$ are some internal outputs which are called update gate, reset gate and cell state, respectively. Note that GRU is a RNN by definition.

Suppose the input is a $p$-dimension vector, and the representation is a $d$-dimension vector. The time cost of computing $u_t, r_t, c_t, h_t$ are

\[ d(p + 4d), d(p + 4d), d(p + 4d + 1), 4d, \]

respectively. Thus, the exact time cost of GRU on all $T$ time steps (for an arbitrary node or edge) is

\[ T(3d(p + 4d) + 5d), \]

which comes down to

\[ O(Td^2), \]

assuming that $p$ and $d$ are of the same magnitude. Since $d^2$ is of the same magnitude as number of learnable parameters $W$, the total cost is $O(TW)$.

**Self-Attention.** Self-Attention (Vaswani et al., 2017) with future masking is given by

\[
C_h = \text{softmax}\left( \frac{(XW^Q_h) (XW^K_h)^T \odot M}{\sqrt{p}} (XW^V_h) \right),
\]

and

\[
H = [C_1 \cdots |C_H]|W^O,
\]

where $X$ is a $T \times p$ matrix including attributes over all $T$ time steps, $M$ is $T \times T$ future masking matrix (upper triangular 0-or-1 matrix), $H$ is $T \times d$ hidden state representation matrix of all time steps, and
\( W^Q_h, W^K_h, W^V_h, W^O \), \( 1 \leq h \leq H \) are learnable parameters where \( H \) is an arbitrary number, a.k.a. number of multi-heads. \( C_h, 1 \leq h \leq H \) are internal outputs. An RNN view of such Self-Attention can be described as

\[
Q_{h,t} = (W^Q_h)^T x_t, \\
K_{h,t} = (W^K_h)^T x_t, \\
V_{h,t} = (W^V_h)^T x_t, \\
\alpha_{h,t} = \frac{\exp\left( \frac{1}{\sqrt{p}} Q_{h,t} \cdot K_{h,t} \right)}{\sum_{\tau=1} \exp\left( \frac{1}{\sqrt{p}} Q_{h,\tau} \cdot K_{h,\tau} \right)}, \\
C_{h,t} = \alpha_{h,t} V_{h,t}, \\
S_t = [S_{h,t-1} || Q_{1,t} || \cdots || Q_{H,t} || K_{1,t} || \cdots || K_{H,t}], \\
H_t = [C_{1,t}^T || \cdots || C_{H,t}^T] W^O,
\]

where \( x_t \) is the input vector at time step \( t \), \( S_t, H_t \) are hidden state representation passing through RNN recursions. \( H_t \) is the same as raw definition of Self-Attention while \( S_t \) is an additional memory recording past \( Q, K \) for the ease of computation for \( \alpha \).

For the time complexity analysis, suppose the input is a \( p \)-dimension vector, and the representation is a \( d \)-dimension vector. Our complexity analysis is based on raw form of Self-Attention. The time cost of computing \( C_h \) of an arbitrary \( h \) are

\[
\frac{1}{H} T(3dp) + 4T^2 + \frac{1}{H} T^2 d,
\]

and the total cost to achieve \( H \) is

\[
T(3dp) + 4HT^2 + T^2 d + Td^2,
\]

which can be summarized as

\[
O(Td^2 + T^2 d),
\]

assuming that \( p \) and \( d \) are of the same magnitude and \( H \) is an arbitrary constant. Since \( d^2 \) is of the same magnitude as number of learnable parameters \( W \) and \( T, d \) are of the same magnitude (around \( 10^3 \)) in experiments, it can be regarded as \( O(TW) \) in our case.

It is important to note that Self-Attention final representation \( H_T \) is invariant to sequence permutation (or \( H \) is equivariant) in its raw form. To make the sequence order matter, [Vaswani et al., 2017] proposes adding a positional encoding \( PE(t) \) —that varies with the sequence index \( 1 \leq t \leq T \) (time steps in our application)— to \( X_t \).

### C.2 (Static) GNN

We now briefly introduce GCN [Kipf & Welling, 2016], GTC [Xu et al., 2020] and GAT [Veličković et al., 2017], for those unfamiliar with these architectures.

**GCN.** GCN works as follows: Let

\[
z_i = W \sum_{j \in N(i)} \hat{A}_{i,j} x_j + b, \tag{22}
\]

where \( x_i \) is a \( p \)-dimension input of node \( i \), \( z_i \) is a \( d \)-dimension representation of node \( i \), and

\[
\hat{A}_{i,j} = \frac{1}{\sqrt{d_i d_j}} A_{i,j}
\]

is normalized edge weight where \( d_i, d_j \) are degree of node \( i \) and \( j \). It is assumed that self-loop edges are already included in adjacency matrix \( A \) in our experiments. The time complexity is

\[
O(|V|d^2 + (|V| + |E|)d),
\]
assuming that $p$ and $d$ are of the same magnitude. Since $d$ and $|\mathcal{V}|$ are relatively small w.r.t. to $|\mathcal{E}|$, the time
cost is indeed $\mathcal{O}(|\mathcal{E}|)$ in our case.

**GTC.** GTC (Xu et al. [2018]) is a special time-then-graph representation by arranging all edges and
neighbors as sequences and encode them by Self-Attention,

\[
\begin{align*}
\text{SN}_i &= \left[ \sum_{t=1}^{T} \left[ \mathbf{X}_{j,t} \| \mathbf{A}_{i,j,t} \right]_{j \in \mathcal{N}(i)} \right], \\
\mathbf{Z}_{i,T} &= \text{Self-Attention} (\text{SN}_i),
\end{align*}
\]

where $\mathbf{X}_{i,t}$ is attribute of node $i$ at time $t$, $\mathbf{A}_{i,j,t}$ is attribute of edge from $j$ to $i$ at time $t$, and $\mathbf{Z}_{i,T}$ is the final representation of node $i$. Indeed, each element of $\text{SN}_i$ is a concatenation of source node attribute $\mathbf{X}_{i,t}$, edge attribute $\mathbf{A}_{i,j,t}$, and observation time $t$. It is important to note that positional encoding of Self-Attention
is not used here to ensure that the node representations are permutation invariant, and essential ordering
features are provided by $t$ in $\text{SN}_i$. All edges will appear at most twice (from $j$ to $i$ and back) in all $\text{SN}_i$, $i \in \mathcal{V}$,
thus the worst case is when we have all edges in the same sequence. The complexity is then directly obtained
from the Self-Attention complexity: $\mathcal{O}(|\mathcal{E}|W)$, where $\mathcal{E}$ is the edge set given in Definition 6 and $W$ is number of learnable parameters of GTC.

**GAT.** GAT (Velickovic et al. [2017]) is GTC with $T = 1$. Thus, GAT can inherit the property from
GTC, and its complexity is also $\mathcal{O}(|\mathcal{E}|W)$ where $\mathcal{E}$ is the edge set of static graph and $W$ is number learnable parameters of GAT.

### C.3 Temporal Graph Baselines

In what follows we will introduce all baselines. We use the temporal graph notation in Definition 6, $\mathcal{V}$, $\mathcal{E}$ are
all nodes and edges over all $T$ time steps, and $E_t$ is edges at time step $t$. $X_{i,t}, A_{i,j,t}$ are node and edge
attributes over all $T$ time steps, and $X_{i,t}, A_{i,j,t}$ are attributes at time $t$. We use superscript $1 \leq l \leq L$ to represent different GNN layers ($L = 2$ in our experiments).

**EvolveGCN.** EvolveGCN (Pareja et al. [2020]) is a special temporal graph model whose GCN weights
change over time. Generally speaking, its GCN layer can be defined as

\[
\begin{align*}
\mathbf{W}_{i}^{(l)} &= \text{GRU}^{(l)}(\mathbf{X}_{i,t}, \mathbf{W}_{i}^{(l-1)}), \\
\mathbf{Z}_{i,t} &= \text{GCN}^{(l)}(\mathbf{X}_{i,t}, \mathbf{A}_{i,j,t} | \mathbf{W}_{t}^{(l)}),
\end{align*}
\]

where $\text{GCN} (\cdot | W)$ means using given parameter $W$ in GCN. The GRU of EvolveGCN is slightly modified in
Pareja et al. [2020] to accept matrix inputs instead of vectors. And based on this strategy, it can be further
classified as EvolveGCN-O and EvolveGCN-H. For both cases, the number of parameters are $\mathcal{O}(d^2)$. Thus, the complexity of EvolveGCN is

\[
\mathcal{O} \left( Td^2 + |\mathcal{E}| \right).
\]

**GCRN-M2.** GCRN-M2 (Seo et al. [2018]) can be described by Equation 6 with GRUs and GCNs as
their component neural networks. The complexity of it is

\[
\mathcal{O} \left( T|\mathcal{V}|d^2 + |\mathcal{E}| \right),
\]

since GRU will be applied on each of the nodes independently.

**WD-GCN.** WD-GCN (Manessi et al. [2020]) can be described by Equation 7 with GRUs and GCNs (adding residual) as their component neural networks. The GCN is slightly modified as

\[
\mathbf{Z}_{i,t} = \text{GCN}(\mathbf{X}_{i,t}, \mathbf{A}_{i,j,t}) + \mathbf{X}_{i,t} \mathbf{W}^\text{Residual}.
\]

This adds a linear transformation of inputs to representation outputs of GCN. And, the time complexity of it is

\[
\mathcal{O} \left( T|\mathcal{V}|d^2 + |\mathcal{E}|d^2 \right).
\]

**DySAT.** DySAT (Sankar et al. [2020]) can be described by Equation 7 with Self-Attentions and GATs
as its component architectures. Both of them have an order of $\mathcal{O}(d^2)$ parameters. The time complexity is

\[
\mathcal{O} \left( T|\mathcal{V}|d^2 + |\mathcal{E}|d^2 \right).
\]
TGN. TGN [Rossi et al., 2020] is a *time-then-graph* framework exploiting GTC. It first aggregates node attributes at each snapshot through the average

\[ S_{i,t} = \frac{1}{|N_i(i)|} \sum_{j \in N_i(i)} X_{j,t}, \]

then a GRU is applied on \( S_i \) for each node \( i \in V \) just as *time-and-graph*

\[ H_{i,t} = \text{GRU}(\{S_{i,1}, \cdots, S_{i,t}\}), \quad \forall i \in V, 1 \leq t \leq T. \]

Then, GTC (Equation (23)) is applied on the aggregated temporal graph with \( H \) as node attributes.

\[ \text{SN}_i = \left[ \left[ H_{j,t} \parallel A_{i,j,t} \right] \right]_{j \in N_i(i)}^{t=1}^{T} \]

\[ Z_{i,T} = \text{Self-Attention}(\text{SN}_i) \]

The time complexity of the *time-and-graph* part is

\[ O(T|V|d^2 + |E|), \]

and the time complexity of the GTC part is \( O(|E|d^2) \). The total time complexity is

\[ O(T|V|d^2 + |E|d^2). \]

### C.4 Complexity of Our Methods

Table 3: **Complexity Table.** The time complexity of all methods evaluated in our work (in big O notation). \( T \) is number of time steps, \( V, E \) are nodes and edges of aggregated temporal graph and \( d \) is representation (embedding) size. The number of learnable parameters \( W \) is assumed to have magnitude \( O(d^2) \). Also, \( d \) is assumed to have same magnitude as \( T \) in our experiments.

| Method     | Time Complexity              |
|------------|------------------------------|
| EvolveGCN  | \( O(Td^2 + |E|) \)         |
| GCRN-M2    | \( O(T|V|d^2 + |E|) \)      |
| WD-GCN     | \( O(T|V|d^2 + |E|) \)      |
| DySAT      | \( O(T|V|d^2 + |E|d^2) \)   |
| TGN        | \( O(T|V|d^2 + |E|d^2) \)   |
| GRU-D-GCN  | \( O(T|V|d^2 + |E|) \)      |
| GRU-D-GTC  | \( O(T|V|d^2 + |E|) \)      |
| GRU-S-GCN  | \( O(T|V|d^2 + |E|d^2) \)   |
| GRU-S-GTC  | \( O(T|V|d^2 + |E|d^2) \)   |

We also analyze the time complexity of our methods. The time cost of our methods can be divided into three parts.

1. The first part is modeling node temporal attributes by GRU, and the complexity is \( O(T|V|d^2) \).
2. The second part is modeling edge temporal attributes. This part applies a weighted summation to the existing edges and accumulate each of them at different time steps into a single edge. Thus, all existing edges are computed at once, and the time complexity is \( O(|E|) \).
3. The last part applies either GCN or GTC over the aggregated temporal graph with \( |V| \) nodes and \( |E| \) edges. The overall time complexity is given in Table 3 based on the different component architectures: GRU-D-GCN, GRU-D-GTC, GRU-S-GCN, and GRU-S-GTC.
We first divide 150 samples into 10 folds, each fold of which has 15 samples with uniform distribution across all datasets. Table 4: Real-world Dataset Results. The performance (ROCAUC and accuracy) mean and variance of each methods on all four real-world datasets are reports. We also report the real time cost (in seconds) per training epoch of all methods. All experiments are performed on a AMD Epyc 7662 64-core CPU. Our methods are always among the best ones all four real-world datasets (often having the highest average accuracy). Besides the outstanding performance and stability, our methods are also fast in training time, on par with the fastest baselines.

### D.3 Extra Results

Some real-world datasets are comprised of single large temporal graph. In these cases, we will divide all nodes \( V \) into 10 folds. Each fold will be held out as test split once. For the remaining nodes, the first 90% be used as training data, and the last 10% be used as validation data. Thus, we will have ten different dataset splits, and we evaluate all methods on test fold of each split.

### D.2 Hyperparameter Search

In our experiments, hyperparameter settings are as follows:

1. Number of GNN layers is fixed to \( L = 2 \);
2. Activation function is fixed to tanh;
3. Optimizer is fixed to Adam [Kingma & Ba, 2014];
4. Number of training epochs is fixed to 120 for DynamicCSL, and 400 for all real-world datasets;
5. The weight decay is fixed to 0.005;
6. The representation (embedding) sizes \( d = 2, 6, 10, 16 \);
7. The learning rates are 0.1, 0.01, 0.001.

Hyperparameters are selected according to mean ROCAUC performance on all validations. In our case, the hyperparameter to be searched are just representation (embedding) size \( d \) and learning rate. And the search result is provided in Figure [17]. We can see that we barely get improvement from representation size 6, and become nearly stable after representation size 10.

### D.1 Dataset Split

The DynamicCSL dataset has 150 different temporal graphs categorized by number of non-isomorphic CSLs. We first divide 150 samples into 10 folds, each fold of which has 15 samples with an uniform distribution of labels. Then, two consecutive folds will be held out as test split, the last fold in will be our validation data, with all the remaining folds used in training. We then have five different dataset splits, and evaluate all methods on test folds of each split.

Some real-world datasets are comprised of single large temporal graph. In these cases, we will divide all nodes \( V \) into 10 folds. Each fold will be held out as test split once. For the remaining nodes, the first 90% be used as training data, and the last 10% be used as validation data. Thus, we will have ten different dataset splits, and we evaluate all methods on test fold of each split.

### D.3 Extra Results

Figures [12] and [13] present accuracy results rather than the ROCAUC presented in Figures [4] and [5]. We also provide the mean and standard deviation report in Table [4]. We can see that the accuracy results and the ROCAUC scores reach the same conclusion: In DynamicCSL tasks, our proposals are the only methods that can correctly represent the input data and show between 80% and 100% accuracy, while all other state-of-the-art baselines fail at 20% accuracy (random accuracy). In real world tasks, all state-of-the-art baselines and our proposals have similar performance except EvolveGCN which is always the worst method across all datasets.
We note that the DynamicCSL results of TGN are surprisingly disappointing. As a time-then-graph representation model, one would expect it to perform better than random at the DynamicCSL task. The temporal graphs illustrated in Figure 9 help explain the reason. In TGN, time-then-graph is implemented by GTC where all edges connected to a node \( i \in V \) over all time steps are arranged as a sequence, as shown in Equation (23). However, this sequence just describes the edge and node attributes of the neighbors of \( i \). In the task illustrated in Figure 9 the graphs are unattributed, hence, all nodes \( i \in V \) will see the same sequence in both temporal graphs (a) and (b) (in what follows, 1 and 2 indicate the time steps):

\[
SN_i = [0|0|1, 0|0|1, 0|0|1, 0|0|1, 0|0|1, 0|0|2, 0|0|2, 0|0|2, 0|0|2],
\]

where 0 means node and edge attributes are zero (unattributed). Thus, the above encoding prevents GTC from distinguishing the two temporal CSL graphs in our task.

All methods on Reddit-4 present a higher ROCAUC variance than in other datasets. Reddit is one of the largest worldwide social media platforms, where posts on different reddit communities (subreddits) likely have little in common, creating a lot of temporal independence between posts different communities, making our temporal task harder.

E Related Works

In what follows we introduce other related works not included in the main paper due to space constraints. They are either no longer considered state-of-the-art methods and/or are not representation learning methods (neural networks). We will also introduce methods for temporal knowledge graph (TKG), which have a different set of assumptions than our work.

Other Time-and-graph Works. DynGEM (Goyal et al., 2018) is similar to EvolveGCN. It also only applies static representation functions independently on each snapshot. It differs from EvolveGCN in that previous steps directly inheriting parameters of current step as initialization for a faster training convergence. DCRNN (Li et al., 2018) is a time-and-graph application in traffic forecasting utilizing simple graph diffusion instead of GNN. GGNN (Li et al., 2015) is another time-and-graph application where node attributes do not change, thus it does not have the GNN \(^{\text{in}}\) part that most other time-and-graph methods have (see Equation (9)). There are also variational autoencoder works (Hajiramezanali et al., 2019) and generative adversarial works (Lei et al., 2019; Xiong et al., 2019), which are extensions of time-and-graph works introduced earlier, e.g., putting time-and-graph as the encoder of generative adversarial framework.

Permutation Robust. In all the works so far, including our work, implicitly assume that node indices are preserved throughout the time, e.g., the node with index 1 at time step 1 preserves its index at later time steps. However, it is possible that indexing is not preserved in some extreme applications (e.g., molecular dynamics). As far as we know, tNodeEmbed (Singer et al., 2019) is the only work addressing this task. It solves the problem by also learning the permutation matrices between consecutive time steps along with learning the temporal node representation.

Temporal Graph Methods using Engineered Features. There are also traditional temporal graph works using temporal features. For instance, DynTriad (Zhou et al., 2018) predicts the social connections by exploiting triadic closures (if a and b are connected, b and c are connected, it is likely that a and c will be connected in future). Kovanen et al. (2011) directly uses the motifs of aggregated temporal graph to predict links.

Tensor Decomposition. The adjacency matrix of aggregated temporal graph can be seen as a tensor (albeit, not permutation-invariant on the time axis). Hence, it is expected to see tensor decomposition applied to temporal graphs. DGCNN (Diao et al., 2019) is a recent work merging dynamic Lapacian matrix decomposition and SGCNN (Yu et al., 2017). Another work is TIMERS (Zhang et al., 2018), which models the Laplacian matrix evolution rather than the temporal graph dynamics directly. It does so by reusing decomposition results from previous temporal steps. GrNMF (Ma et al., 2018) uses a similar strategy to TIMERS. We note in passing that tensor decomposition generally assumes the time axis to be permutation invariant, that is, the order of events in time does not matter. In some applications this assumption will be fine, while in other applications it will simply not work.
Temporal Knowledge Graphs (TKG). There is host of works on temporal knowledge graphs (Han et al., 2020; Ma et al., 2020; Jin et al., 2019; Kumar et al., 2019; Shang & Sun, 2019; Trivedi et al., 2019). Here, it is commonly assumed that the temporal graph is continuous (with exact timestamps), and edge (event) comes one-by-one rather than as a whole-graph snapshot. Most of those works also require immediate reaction on an edge (event) observation, thus node representation is updated per edge observation by a RNN instead of a GNN aggregating all neighbors. Although our approach applies to TKGs, the continuous stream of edges would require a different edge-representation approach (e.g. cLSTM (Mei & Eisner, 2016)). Hence, we do not regard TKG as a too closely related task, but worthy of future work.
Figure 6: Illustration of a time-and-graph neural network architecture: \( X_{:,t} \) is all node attributes at snapshot \( t \) and \( A_{:,t} \) is all edge attributes at snapshot \( t \) for \( 1 \leq t \leq T \). \( \text{GNN}^{(l)}_{\text{in}} \) (blue) and \( \text{GNN}^{(l)}_{\text{rec}} \) (orange) are \( l \)-th layer (\( 1 \leq l \leq L \)) of graph neural networks \( \text{GNN}^L_{\text{in}} \) and \( \text{GNN}^L_{\text{rec}} \) as defined in Equation (6). Different color saturation means layer depth, or in other words, darker color means larger \( l \). \( \text{RNNCell} \) (green) is the recurrent neural network defined in Equation (6). \( H_{:,t} \) is the historical representation output of time-and-graph function at snapshot \( t \), and \( H_{:,T} \) will be the final representation output. \( I^{(l)}_{:,t} \) is the internal (hidden) states of \( \text{GNN}^{L}_{\text{in}} \) at layer \( l \). \( J^{(l)}_{:,t} \) is the internal (hidden) states of \( \text{GNN}^{L}_{\text{rec}} \) at layer \( l \).
Figure 7: **Illustration of l-th layer in GNN\textsuperscript{TL\textsubscript{tar}} when l mod L ≠ 0:** Yellow square is the illustration of Equations (12) and (13) when $t = \lceil \frac{l}{L} \rceil$. Operations (Squares) with the same colors inside it imply the same operations as in Figure 6. We can see that for $l \mod L \neq 0$, any arbitrary layer $l$ of the GNN\textsubscript{tar} is indeed processing layer $l\%$ of GNN\textsubscript{in} and GNN\textsubscript{rec} at snapshot $t = \lceil \frac{l}{L} \rceil$ in parallel.

Figure 8: **Illustration of l-th layer in GNN\textsuperscript{TL\textsubscript{tar}} when l mod L = 0:** Yellow square is the illustration of Equations (12) and (13) when $t = \lceil \frac{l}{L} \rceil$. Operations (Squares) with the same colors inside it imply the same operations as in Figure 6. We can see that for $l \mod L = 0$, besides the operations illustrated in the Figure 7, GNN\textsubscript{tar}(l) also replaces part of output, $I'(l)$, by node attributes $X_{\lceil \frac{l}{L} + 1 \rceil}$, and replaces the other part of output, $J'(l)$, by the output of Equation (11). This indeed is mirroring the operations after layer $L$ in Figure 6.
Figure 9: A task to distinguish two different Circular Skip Length (CSL) temporal graphs. In both (a) and (b) temporal graphs, we show two different unattributed temporal graphs. The two graphs in (c) show the aggregated temporal graphs of (a) and (b), where edge attributes are sequences of non-existing (0) and existing (1) edges at different time steps, denoted by different colors in the illustration. This task extends the static CSL task of Murphy et al. (2019) to the temporal domain.

Figure 10: An extension of the example in Srinivasan & Ribeiro (2019) to temporal graphs. Consider two disconnected components (boreal forest (blue) and antarctic fauna (orange)) over two-step dynamics in a food chain system. The two disconnected components have the same dynamic because of similar prey and predator behaviors in the ecosystem. The graphs have no edge or node attributes.
Figure 11:  **Representation (Embedding) Size Search:** For each representation size, only the best performance is reported from all learning rate options. Performance of different real-world datasets are rendered by different colors. The vertical bar at each point shows the standard deviation of that representation size on the corresponding datasets.

Figure 12:  **Accuracy Performance Boxplot:** The white square is the mean performance of the model. The colored box reflects the first quartile, median, and third quartile from left to right. The x-axis is accuracy score weighted (Accuracy) by inverse of label counts. Closer to right (larger accuracy scores) means better. The bottom four from GRU-D-GCN to GRU-S-GTC are our proposals.
Figure 13: **T-Test Result:** Red grid means row is better than column with more than 95% confidence, blue grid means row is worse than column with more than 95% confidence, and faded grid means t-test can not tell the difference with great confidence. More red grids in a row indicates the method is better.