Temporal SIR-GN: Efficient and Effective Structural Representation Learning for Temporal Graphs

Janet Layne
Boise State University, Boise, ID, USA
janetlayne@boisestate.edu

Justin Carpenter
Boise State University, Boise, ID, USA
justincarpenter836@u.boisestate.edu

Edoardo Serra
Boise State University, Boise, ID, USA
edoardoserra@boisestate.edu

Francesco Gullo
UniCredit, Rome, Italy
gullof@acm.org

ABSTRACT

Node representation learning (NRL) generates numerical vectors (embeddings) for the nodes of a graph. Structural NRL specifically assigns similar node embeddings for those nodes that exhibit similar structural roles. This is in contrast with its proximity-based counterpart, wherein similarity between embeddings reflects spatial proximity among nodes. Structural NRL is useful for tasks such as node classification where nodes of the same class share structural roles, though there may exist a distant, or no path between them.

Athough structural NRL has been well-studied in static graphs, it has received limited attention in the temporal setting. Here, the embeddings are required to represent the evolution of nodes' structural roles over time. The existing methods are limited in terms of efficiency and effectiveness: they scale poorly to even moderate number of timestamps, or capture structural role only tangentially.

In this work, we present a novel unsupervised approach to structural representation learning for temporal graphs that overcomes these limitations. For each node, our approach clusters then aggregates the embedding of a node's neighbors for each timestamp, followed by a further temporal aggregation of all timestamps. This is repeated for (at most) d iterations, so as to acquire information from the d-hop neighborhood of a node. Our approach takes linear time in the number of overall temporal edges, and possesses important theoretical properties that formally demonstrate its effectiveness.

Extensive experiments on synthetic and real datasets show superior performance in node classification and regression tasks, and superior scalability of our approach to large graphs.

PVLDB Reference Format:
Janet Layne, Justin Carpenter, Edoardo Serra, and Francesco Gullo. Efficient and Effective Structural Representation Learning for Temporal Graphs. PVLDB, 16(9): 2075-2089, 2023.
doi:10.14778/3598581.3598583

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/janetlayne2/Temporal-SIR-GN.

Proceedings of the VLDB Endowment, Vol. 16, No. 9 ISSN 2150-8097.
doi:10.14778/3598581.3598583

1 INTRODUCTION

Graphs, i.e., sets of entities (*nodes*) linked to one other (via *edges*), have become a ubiquitous model for representing real-world data from a plethora of domains [2, 13, 18, 33]. *Graph representation learning* (or *graph embedding*) automates the task of assigning elements of a graph (e.g., nodes, edges, subgraphs, entire graphs) numerical vectors – termed embeddings or representations[1] – such that the *similarity* between those elements in the graph corresponds to the *similarity* between their embeddings [10, 29, 84, 97]. *Node representation learning* (NRL) is the term used when embeddings are generated specifically for the graph nodes.

Importantly, the notion of *similarity* in NRL is not fixed; approaches can largely be understood as capturing either *node proximity* or *structural properties* in their representations. *Proximity-based approaches* [9, 24, 52, 68, 73, 91, 96] preserve the information about connections between nodes, assigning similar representations for nodes close in the graph in terms of d-hop reachability, co-occurence in a random walk, Personalized PageRank, etc. Conversely, *structural-role similarity* is concerned with information about nodes' neighborhood structure (**Figures 1–(I)–(II)**).[2]

NRL has been employed in several downstream tasks, including node classification, link prediction, clustering, graph visualization, graph alignment, and graph summarization [10, 57, 97]. Either methodology (proximity-based or structural) is useful in certain circumstances. As an example, structural approaches are useful for node classification, when the node labels are not determined by proximity/homophily, rather by isomorphic local subgraph structures. In contrast, tasks such as link prediction may benefit from use of proximity-based methods, where connections between nodes are preserved in node representations [28, 30, 54, 56, 57].

A *temporal graph* is one whose edges change over time. It is a sequence of *graph snapshots* representing the nodes and edges at specific timestamps. Temporal graphs have received considerable attention regarding a variety of problems [7, 20, 67, 82, 85], including NRL [31]. Structural NRL for temporal graphs yields embeddings that encode the *temporal evolution* of the (role played by the) nodes [31]. This temporal evolution may lead to different

[1] We use the two terms interchangeably throughout the paper.
[2] Nevertheless, the literature lacks in uniformity on this terminology. The term "structural" may refer to notions other than that of our interest. E.g., "*global structural information*" or similar terms are used [9, 63], which correspond to higher-order spatial proximity (i.e., proximity based on nodes' d-hop neighborhoods), rather than the actual structural role discussed in this work..

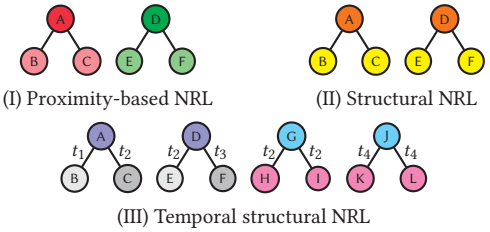


Figure 1: Illustration of various kinds of node representation learning (NRL). Similarity in nodes' colors denotes similarity in nodes' embeddings. (I) Proximity-based NRL recognizes {A, B, C} as similar, as they are physically close in the graph. The same holds for {D, E, F}. (II) Nodes {A, D} are part of different connected components. Nonetheless, structural NRL recognizes them as similar, as they have isomorphic 1-hop neighborhoods (or, as formally explained in Section 4.2, there is an *automorphism* that maps A to D, and vice versa). The same holds for {B, C, E, F}. (III) Edges are labeled with timestamps. For simplicity, here is one timestamp per edge, but, in general, an edge can be assigned multiple timestamps. Timestamps here come with no particular constraints: they can be any positive real number such that $t_1 < t_2 < t_3 < t_4$. Temporal structural NRL recognizes {A, D} as similar, as they have isomorphic 1-hop neighborhoods in all the timestamps (or, as formally explained in Section 4.2, there is a *temporal automorphism* that maps A to D, and vice versa). Likewise, {G, J} are recognized as similar, as they both have two neighbors in one timestamp, and no neighbors in the other timestamps. Similar considerations hold for {B, E}, {C, F}, and {H, I, K, L}. Conversely, if the temporal aspect is not considered, i.e., timestamps on edges are ignored, very different (structural) embeddings would be yielded: {A, D, G, J} would wrongly have identical embeddings, and {B, C, E, F, H, I, K, L} as well.

structural roles than the static case (cf. **Figure 1–(III)**). Thus, static NRL approaches cannot easily be adapted to temporal graphs.

Applications. A major application of temporal structural NRL is on any node classification task where node interactions change over time, and node labels are not homophily-driven, rather they depend on structural properties of the graph. For instance, classifying nodes (users) of financial transaction graphs as (non-)fraudulent typically relies on the the patterns of transactions issued by a user over time [66]. Embeddings encoding the structural temporal evolution of a graph are thus well-suited for such a fraud-detection task.

Similarly, in social-trust networks where nodes (users) express (dis)trust opinions vs. other nodes over time, the goal is to classify a user as trustful or not. This classification heavily relies on the temporal structural patterns of the rating received by a user [34].

The biological domain has plenty of applications that can benefit from temporal structural NRL. For instance, in dynamic protein-interaction networks, classifying a protein (node) as, e.g., uncharacterized/verified depends on the structural patterns of temporal interactions between that protein and the others in the network [19].

Further applications of temporal structural NRL include tasks other than node classification as well. For instance, the prediction of properties (e.g., centralities) that depend on time-varying structural characteristics of a graph, or problems like entity resolution, alignment and summarization in temporal graphs [40, 72, 86].

Motivation. To the best of our knowledge, the prominent work that may handle temporal structural NRL (at least to some extent) is that by Liu *et al.* [41]. It employs a temporal graph neural network (GNN) that is based on the notion of k-core, i.e., the maximal subgraph whose nodes have degree at least k [4]. Apart from Liu *et al.*, there exist several other temporal-GNN-based methods that can potentially (be adapted to) capture structural roles, though they are not specifically designed for structural NRL [12, 22, 36, 38, 39, 44, 46, 50, 60, 62, 64, 78, 79, 87, 88, 98]. All such methods (including Liu *et al.*'s one) have *limited efficiency*, as they employ computationally-expensive models, and *limited effectiveness*, due to the use of loss functions not ideally suited for structural NRL.

Contributions. In this work, we tackle the problem of structural NRL in temporal graphs. The goal is to generate a *single* embedding for every node that encodes the temporal evolution of that node's structural role. The target structural roles are defined based on the adaptation of graph isomorphism to the temporal setting.

Proposed method. We design Temporal SIR-GN, a novel *unsupervised* approach that improves upon efficiency and effectiveness of the state of the art. Temporal SIR-GN is inspired by SIR-GN [30], a recently-introduced efficient and effective method for structural NRL in static graphs. Temporal SIR-GN adopts SIR-GN's idea of iteratively clustering and aggregating the representations of a node's neighbors, which in turn emulates the well-established *Weisfeiler-Lehman* isomorphism test [81]. The main difference between our approach and basic SIR-GN is that the aggregation of nodes' representations must now consider the temporal dimension. We accomplish this by computing the expected number of transitions from a cluster C to any other cluster C' in the temporal sequence of embeddings. A major challenge of this temporal aggregation is that its naïve computation takes quadratic time in the number of timestamps. We thus derive a factorization that converts the quadratic cost to linear, without losing exactness. Clustering and temporal aggregation are repeated for (at most) d iterations, so as to explore d levels of depth of the temporal structure around a node.

Benefits of the proposed method include: (*i*) it takes linear time in the number of temporal edges, while the existing methods are slower, both in terms of theoretical time complexity, and especially in practice, due to their many additional maintenance costs; (*ii*) it keeps in main memory one embedding per node, as opposed to the state of the art, which typically needs to keep in memory one embedding for every node and every timestamp; (*iii*) it is backed by a theoretical analysis that formally shows how it preserves key temporal-structural information; (*iv*) it employs no sequence-learning models: besides enabling efficiency, this makes the method lightweight and easy-to-implement.

Summary and roadmap. To summarize, in this work, we:

- Tackle the problem of temporal structural NRL (**Section 2**).
- Devise Temporal SIR-GN, a novel unsupervised approach to temporal structural NRL that overcomes existing limitations of efficiency and effectiveness (**Section 3**).
- Show how to perform temporal aggregation in Temporal SIR-GN in linear time in the number of timestamps (**Section 3.4**).
- Prove theoretical properties about how Temporal SIR-GN preserves temporal structural-role information (**Section 4**).
- Design testbeds to assess a method in temporal structural NRL (**Section 5**). This is a contribution of per-se interest.
- Extensively test Temporal SIR-GN on both synthetic and real datasets. Results attest its high efficiency and effectiveness in classification and regression tasks (**Sections 5.1–5.4**).

Section 6 concludes the paper and discusses ideas for future work.

2 PRELIMINARIES AND BACKGROUND

Let $\mathcal{G} = (V, T, \mathcal{E})$ be a *temporal graph*, where V is a set of *nodes*, $T \subseteq \mathbb{R}_0^+$ is a *finite* set of *timestamps* (where a timestamp is a positive real number), $\mathcal{E} \subseteq V \times V \times T$ is a set of *temporal edges*, i.e., the set $\{(u, v, t)\}$ of all node pairs $u, v \in V$ and timestamps $t \in T$ such that an edge exists between u and v in t. Given a timestamp $t \in T$, $E_t = \{u, v \in V \mid (u, v, t) \in \mathcal{E}\}$ and $V_t = \{u \in V \mid \exists (u, v) \in E_t\}$ denote the set of *static* edges and nodes existing in t, respectively, and $G_t = (V_t, E_t)$ is the *(graph) snapshot* of t, i.e., the *static* graph corresponding to the projection of G in t. Let also $T(u) = \{t \in T \mid u \in V_t\}$ denote the timestamps in which $u \in V$ exists, and $\mathcal{T} = |\sum_{u \in V} T(u)|$. Hereinafter, we assume $\mathcal{G}$ to be *undirected*. However, handling directed graphs is an easy extension (cf. **Section 3.5**). The main notations used in the paper are summarized in **Table 1**.

This temporal graph model is general enough to have edges arbitrarily (dis)appear over time, and be present in multiple timestamps.

2.1 Problem statement

We focus on a *non-diachronic* objective, i.e., generating a *single* embedding for each node that encodes the evolution of that node's structural role over the *whole* temporal graph. This differs from a *diachronic* objective [21], where computation of the final embeddings requires producing and *materializing* an intermediate embedding for all timestamps. The problem addressed in this work is:

Problem 1 (**Temporal Structural NRL**). Given a temporal graph $\mathcal{G} = (V, T, \mathcal{E})$, and a natural number $h \in \mathbb{N}^+$, compute a real-valued matrix $R \in \mathbb{R}^{|V| \times h}$, where every row $R_{[u]}$ corresponds to the *embedding* (or *representation*) of node u, for all $u \in V$. Each $R_{[u]}$ encodes the temporal evolution of the *structural role* of u in $\mathcal{G}$.

We require the temporal structural role in Problem 1 to express the fact that similar embeddings are assigned to nodes whose local surrounding subgraphs (e.g., d-hop neighborhoods) are as isomorphic as possible. Isomorphism here is intended not only for nodes and edges, but for the *temporal dimension* as well. A more detailed yet formal discussion on the target structural roles is in **Section 3.1**.

2.2 State of the art and limitations

Existing approaches to temporal structural NRL are based on temporal GNNs, and most employ sequence-learning models. To the best of our knowledge, the prominent existing method that is (in part) suited for temporal structural NRL is Liu *et al.*'s CTGCN [41]. It consists of a double-sequence-learning architecture, where Recurrent Neural Networks (RNNs) are nested into a Long Short-Term Memory (LSTM). The latter has one cell per timestamp, and every cell is composed (among others) of multiple RNNs. Each RNN processes the k-cores of a graph snapshot. Multiple RNNs are stacked into every LSTM cell, to capture d-hop neighborhood information. The use of k-cores makes CTGCN able to capture structural roles, at least to some extent. In fact, two nodes of the same (highest-order) k-core intuitively have structurally similar neigborhoods, even if they are far away in the graph. However, being part of the same k-core is not always a signal of similar structural role: e.g., if the neighbors that make two nodes belong to the same k-core are in turn part of very different (highest-order) k-cores.

Other temporal-GNN-based approaches, though not explicitly conceived for structural NRL, can potentially be adapted to it. In fact, they generate embeddings by iteratively aggregating the embeddings of a node's neighbors: this process may capture local isomorphisms, hence structural roles. Many approaches of this kind have been devised [12, 22, 36, 39, 44, 46, 50, 60, 62, 64, 78, 79, 87, 88, 98]. They all share the same general design principle: GNNs yield individual embeddings for every graph snapshot, and all these embeddings are then aggregated over time (e.g., via a sequence-learning model). The differences between the various methods lie in the design and combination of the individual building blocks.

Running time limitations. The time complexity of all the above methods is mostly due to the processing of every graph snapshot via a GNN, which overall takes $\Omega(d \times h \times (|V| \times |T| + \sum_{t \in T} E_t)) = \Omega(d \times h \times (|V| \times |T| + |\mathcal{E}|))$ time. $\Omega(\cdot)$ is used here because it is a lower bound, as several "hidden" steps are not included in it, such as computing the loss function (which may be expensive, e.g., for an unsupervised graph-reconstruction loss), or the internal steps of a sequence-learning model (e.g., handling the internal parameters of every cell of an LSTM). In this regard, Liu *et al.*'s CTGCN comes with a specific additional (non-negligible) k_{max} factor, that is the maximum number of k-cores in a snapshot ($k_{max} = O(|V|)$).

Conversely, our method takes $O(d \times (|\mathcal{E}| \times \sqrt{h} + \mathcal{T} \times h + |V| \times h\sqrt{h}))$ time (cf. **Section 4.1**). This gives a theoretical speed-up that is considerable for large $|T|$, as in this case $\mathcal{T} \ll |V| \times |T|$, and $\sqrt{h} \ll |T|$. In practice, the speed-up is much more evident (cf. **Section 5.3**), due to the aforementioned occult costs of the existing methods.

Storage space limitations. Excluding the input graph, the above methods typically require $O(|T| \times |V| \times h)$ space, as an embedding for every node and every timestamp has to be materialized and kept in memory (e.g., during backpropagation). In contrast, our Temporal SIR-GN method needs $O(|V| \times h)$ space. This corresponds to an $O(|T|)$ improvement, which is particularly appreciable when the number of timestamps is relatively high.

Effectiveness limitations. The notion of k-core in Liu *et al.*'s CTGCN [41] allows for (implicitly) capturing structural roles. However, the two loss functions of CTGCN are not ideally suited for structural NRL. CTGCN's first loss function is defined as the distance between nodes' embeddings and nodes' features (transformed by neural-network layers). That loss is claimed to be structural-role-preserving, but it comes with an important conceptual limitation: it enforces the embeddings of any two nodes to be similar merely if their features are similar, *no matter the graph topology*. At the same time, CTGCN's second loss is based on graph reconstruction, whose use makes the method biased towards proximity.

Similarly, the other existing temporal-GNN-based approaches [12, 38, 39, 44, 46, 50, 60, 62, 64, 78, 87, 88] employ either supervised losses defined based on nodes' labels or unsupervised losses based on graph reconstruction. Both those losses are prone to learn spatial proximity. Particularly, supervised losses enforce a node's embedding to be close to the embeddings of its majority-label neighbors. One might utilize general structural-role-aware losses in those architectures. Unfortunately, designing a loss of this kind is hard. To our knowledge, the only existing attempt is aforementioned Liu *et al.*'s one, which has the previously-discussed downsides.

Table 1: Main notations used in this paper

General notations

$\mathcal{G}=(V,T,\mathcal{E})$	Temporal graph (V: vertices; T: timestamps; $\mathcal{E}$: temporal edges)
$G_t=(V_t,E_t)$	Graph snapshot of timestamp t
$T(u)$	Set of timestamps in which node u exists
$\mathcal{T}$	$\sum_{u\in V}\lvert T(u)\rvert$
d	Depth of exploration (i.e., max iterations of Temporal SIR-GN)
h	Dimensionality of the node embeddings
R	Matrix containing the node embeddings (representations)

Notations from the proposed T-SIRGN (all vectors are row vectors)

$M_{[x]}$	For any matrix M, the row of M corresponding to node x
α	Parameter to modulate temporal effect in the node representations
CR	Matrix containing the node representations from the current iteration
D	Matrix containing nodes' *description vectors* (Def. 3.3)
$nRep$	Number of distinct node representations (from the previous iteration)
c	Number of clusters of node representations
CC	Centers of the clusters of node representations
Γ_u	Vector of squared Euclidean distances from node u to cluster centers
N_u^t	*Neighborhood description vector* (Def. 3.4)
CF_u	*Cluster frequency vector* (Def. 3.5)
CT_u	*Cluster transition matrix* (Def. 3.7)
Z_u^t	Auxiliary vector to speed-up the computation of CF_u
$nbr(u,t)$	Set of neighbors of node u at timestamp t
KMeans()	Function executing K-Means clustering algorithm
Distance()	Function computing distances to cluster centers
MinMax()	Function computing min-max normalization of a matrix

2.3 Other related works

Proximity-based NRL in static graphs has its roots in the context of matrix factorization [5, 59, 70]. A modern reinterpretation of NRL, starting from the first decade of 2000s, has comprised methods aimed at preserving d-hop reachability, co-occurrence in a random walk, and Personalized PageRank [9, 24, 52, 68, 73, 91, 92, 96].

Structural NRL in static graphs includes approaches based on attributed random walks [3], diffusion wavelets [14], Gaussian embedding [51], structural identity [54], graphlets [56, 58], hybrid methods [69], and SIR-GN [30], the precursor of our approach.

Graph Neural Networks (GNNs) have been widely employed in NRL [6, 25, 32, 61, 75, 76, 83, 89]. GNNs yield embeddings by iteratively aggregating the embeddings of a node's neighbors. As such, they have the potential of capturing structural-role similarity. Nevertheless, major obstacles for GNNs to be truly structural-role-aware are the neighborhood-sampling trick, and the loss functions that are not appropriate for structural NRL (cf. **Section 2.2**).

For a more comprehensive overview of the vast literature on NRL in static graphs, we refer to [8, 10, 29, 57, 84, 97].

Proximity-based NRL in temporal graphs [15] includes methods that enforce embedding alignment between consecutive snapshots [15, 65, 90, 101], or *decompose* the adjacency matrices of the snapshots [43, 93, 95], or approaches based on *temporal random walks* [27, 45, 47, 48, 53, 94], *temporal point processes* [16, 42, 71, 99, 100, 102], *causal anonymous walks* (for edge embedding) [80].

3 PROPOSED METHOD: TEMPORAL SIR-GN

3.1 Design principles

Target structures and desiderata. A principled way to characterize structural roles in the static setting is via the notion of *graph isomorphism* [28, 89]: nodes are recognized as structurally similar based on how much their surrounding subgraphs are isomorphic. For this reason, here we identify our target temporal structural patterns by adapting graph isomorphism to the temporal setting.

Definition 3.1 (Isomorphism, subgraph isomorphism, automorphism). An *isomorphism* between graphs $G_1 = (V_1, E_1)$, $G_2 = (V_2, E_2)$ is a *permutation function* $F\colon V_1 \to V_2$, i.e., a function that assigns to each node $u_1 \in V_1$ one and only one node $u_2 \in V_2$, such that $(F(u_1), F(v_1)) \in E_2$ if and only if $(u_1, v_1) \in E_1$. A *subgraph isomorphism* from G to G' is an isomorphism between G and a subgraph of G'. An *automorphism* in G is an isomorphism between G and G itself. Nodes u, u' of G are said *automorphic* if there exists an automorphism in G mapping u to u' (and vice versa).

Based on the above definition, nodes u and u' are recognized as automorphic if they share identical degree, and all their k-hop neighbors share identical degree, for all $k = 1, \ldots, k_{max}$ (k_{max} is the maximum number of hops possible from both nodes). An automorphism for the toy graph in **Figure 1-(I)** is $F(\mathsf{A})=\mathsf{D}$, $F(\mathsf{B})=\mathsf{E}$, $F(\mathsf{C})=\mathsf{C}$, $F(\mathsf{D})=\mathsf{A}$, $F(\mathsf{E})=\mathsf{B}$, $F(\mathsf{F})=\mathsf{F}$.

Definition 3.2 (Temporal isomorphism, subgraph isomorphism, automorphism). A *temporal isomorphism* between temporal graphs $\mathcal{G}_1=(V_1,T_1,\mathcal{E}_1), \mathcal{G}=(V_2,T_2,\mathcal{E}_2)$ is a permutation function $\mathcal{F}\colon V_1 \to V_2$ such that, for every $u\in V_1$, there exists $\Delta_u \in (-\infty,+\infty)$ such that $(\mathcal{F}(u), \mathcal{F}(v), t+\Delta_u) \in \mathcal{E}_2$ if and only if $(u,v,t)\in\mathcal{E}_1$. A *temporal subgraph isomorphism* from $\mathcal{G}$ to $\mathcal{G}'$ is a temporal isomorphism between $\mathcal{G}$ and a temporal subgraph of $\mathcal{G}'$. A *temporal automorphism* in $\mathcal{G}$ is a temporal isomorphism between $\mathcal{G}$ and $\mathcal{G}$ itself. Nodes u, u' of $\mathcal{G}$ are said *temporally-automorphic* if there exists a temporal automorphism in $\mathcal{G}$ mapping u to u' (and vice versa).

Let us elaborate on the definition of temporal automorphism. Similar considerations hold for temporal (subgraph) isomorphism. Temporal automorphism extends the notion of automorphism to the temporal setting by allowing automorphism to occur *across* graph snapshots. Δ_u is the temporal shift between snapshots across which an automorphism should hold in order to have a temporal automorphism. Specifically, if $\Delta_u = 0$, for having a temporal automorphism $\mathcal{F}$ that maps u to u', there must exist an automorphism F_t that maps u to u' in the same snapshot occurring at timestamp t, for all t. Instead, if $\Delta_u > 0$ (resp., $\Delta_u < 0$) the automorphism mapping u to u' is required across the snapshot at timestamp t and the snapshot occurring an amount $|\Delta_u|$ of time after (resp., before) t, for all t. For instance, assuming $t_1=1, t_2=2, t_3=3, t_4=4$, a temporal automorphism for the graph in **Figure 1-(III)** is $\mathcal{F}(\mathsf{A})=\mathsf{D}$, $\mathcal{F}(\mathsf{B})=\mathsf{E}$, $\mathcal{F}(\mathsf{C})=\mathsf{F}$, $\mathcal{F}(\mathsf{D})=\mathsf{A}$, $\mathcal{F}(\mathsf{E})=\mathsf{B}$, $\mathcal{F}(\mathsf{F})=\mathsf{C}$, $\mathcal{F}(\mathsf{G})=\mathsf{J}$, $\mathcal{F}(\mathsf{H})=\mathsf{K}$, $\mathcal{F}(\mathsf{I})=\mathsf{L}$, $\mathcal{F}(\mathsf{J})=\mathsf{G}$, $\mathcal{F}(\mathsf{K})=\mathsf{H}$, $\mathcal{F}(\mathsf{L})=\mathsf{I}$; with $\Delta_\mathsf{A}=\Delta_\mathsf{B}=\Delta_\mathsf{C}=1$, $\Delta_\mathsf{D}=\Delta_\mathsf{E}=\Delta_\mathsf{F}=-1$, $\Delta_\mathsf{G}=\Delta_\mathsf{H}=\Delta_\mathsf{I}=2$, $\Delta_\mathsf{J}=\Delta_\mathsf{K}=\Delta_\mathsf{L}=-2$. As a key difference to the static setting, though there exists an automorphism (ignoring timestamps) mapping B to C, no temporal automorphism exists that maps those two nodes to one another, because they are *temporally structurally different* from the perspective of their common neighbor A (i.e., B comes after C in time).

Whenever any two nodes u and v are temporally-automorphic, they are temporally structurally identical to each other. This is a limit case, for which a desirable requirement is to have identical embeddings produced for u and v. More generally, the closer two nodes are to be temporally automorphic, the more structurally

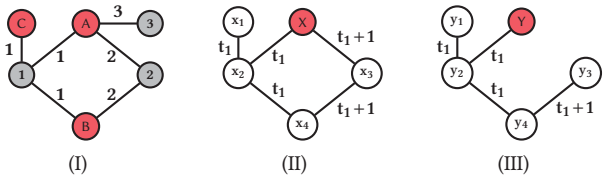


Figure 2: **(I) Toy temporal graph $\mathcal{G}$. For simplicity, $\mathcal{G}$ has one timestamp per edge (though, in general, edges may have multiple timestamps). Lettered nodes (A, B, C) are the ones of interest. Intuitively, A is closer to be temporally-automorphic to B than C, as A and B share two temporal neighbors, while A and C only one. This can be better observed with the maximal temporally isomorphic subgraphs in (II)–(III). (II) $\mathcal{G}_{\mathbf{AB}}$: maximal temporal graph that is temporally subgraph isomorphic to $\mathcal{G}$, and such that there exist two temporal subgraph isomorphisms $\mathcal{F}_1$, $\mathcal{F}_2$ that map the same node of $\mathcal{G}_{\mathbf{AB}}$ to A and B, respectively. t_1 is any positive real number. $\mathcal{F}_1$, $\mathcal{F}_2$ are: $\mathcal{F}_1(\mathbf{X})=\mathbf{A}$, $\mathcal{F}_1(x_1)=\mathbf{C}$, $\mathcal{F}_1(x_2)=1$, $\mathcal{F}_1(x_3)=2$, $\mathcal{F}_1(x_4)=\mathbf{B}$; $\mathcal{F}_2(\mathbf{X})=\mathbf{B}$, $\mathcal{F}_2(x_1)=\mathbf{C}$, $\mathcal{F}_2(x_2)=1$, $\mathcal{F}_2(x_3)=2$, $\mathcal{F}_2(x_4)=\mathbf{A}$. (III) $\mathcal{G}_{\mathbf{AC}}$: same as (II), but for nodes A and C. The temporal subgraph isomorphisms $\mathcal{F}_1$, $\mathcal{F}_2$ in this case are: $\mathcal{F}_1(\mathbf{Y})=\mathbf{A}$, $\mathcal{F}_1(y_1)=\mathbf{C}$, $\mathcal{F}_1(y_2)=1$, $\mathcal{F}_1(y_3)=2$, $\mathcal{F}_1(y_4)=\mathbf{B}$; $\mathcal{F}_2(\mathbf{Y})=\mathbf{C}$, $\mathcal{F}_2(y_1)=\mathbf{A}$, $\mathcal{F}_2(y_2)=1$, $\mathcal{F}_2(y_3)=2$, $\mathcal{F}_2(y_4)=\mathbf{B}$. Δ is $1-t_1$ for all the nodes and temporal subgraph isomorphisms.**

similar they are (cf **Figure 2**): we take this as our main desideratum in designing an algorithm for the Temporal Structural NRL problem. In **Section 4**, we show that our algorithm possesses theoretical guarantees for the limit case of temporally-automorphic nodes, while it comes with empirical evidence in the general case.

Algorithm rationale. The proposed Temporal SIR-GN method resembles the approach in SIR-GN [30], a method for structural NRL in static graphs that has been shown to achieve high effectiveness and efficiency. The logic underlying SIR-GN emulates the *Weisfeiler-Lehman* (WL) algorithm [26, 81], a popular method designed (among others) to test for graph isomorphism. WL compares structural representations generated for nodes in separate graphs. These representations are computed by iteratively updating the current representations via aggregation of additional layers of nodes' neighborhoods. Representations are stored as a multiset that is then used as a hash for unique structures (referred to as colors). Updating is performed until the number of unique hashes is unchanged. SIR-GN capitalizes on the aptitude of WL to capture structural information, but with important modifications. First, SIR-GN clusters node representations to control the overall representation size; then the probability of membership in each cluster is calculated for each node representation. Second, rather than aggregating neighbors via multisets and hashing, each neighbor's representation is summed to form nodes' updated representations. This sum aggregation generates a node representation at iteration i wherein each component of the vector corresponds to the expected number of i-hop neighbors of the node that are in a specific structural cluster.

Temporal SIR-GN can be viewed as the temporal version of SIR-GN. The idea of emulating SIR-GN, and, in turn, WL appears natural in order to identify temporal structures resembling (sub)graph isomorphism. Clustering and neighbor aggregation in Temporal SIR-GN are (mostly) borrowed from SIR-GN. A major novelty lies in the *temporal aggregation*, which is not present in SIR-GN, as it handles static graphs. This is a technically challenging step, as it in principle requires a pairwise comparison between timestamps. In the following, we show how to overcome this quadratic explosion.

Algorithm 1 Temporal SIR-GN

Input: Temporal graph $\mathcal{G}=(V, T, \mathcal{E})$; natural numbers d, $c>0$; real number $\alpha \geq 0$
Output: Matrix $R \in \mathbb{R}^{|V|\times(c^2+c)}$ containing the embeddings of all the nodes in V

$i=0$; $nRep=0$; initialize a matrix $D^0 \in \mathbb{R}^{|V|\times c}$ to $1/c$
R^0 = TemporalAggregation($\mathcal{G}$, c, D^0, α)
while $i<d \wedge nRep < |\{R^i_{[u]} \mid u \in V\}|$ **do**
  $nRep = |\{R^i_{[u]} \mid u \in V\}|$
  D^i = ClusteringNodeDescription(V, R^i, c)
  R^{i+1} = TemporalAggregation($\mathcal{G}$, c, D^i, α)
  $i = i+1$
end while
$R = R^{i-1}$, if $nRep \geq |\{R^i_{[u]} \mid u \in V\}|$; otherwise, $R = R^i$

function ClusteringNodeDescription(V, R, c)
  Initialize a matrix $D \in \mathbb{R}^{|V|\times c}$ to 0 ▷ Def. 3.3
  RN = MinMax(R)
  CC = KMeans(RN, c) ▷ Clustering step
  for all $u \in V$ **do** ▷ Node description loop
    Γ_u = Distance($RN_{[u]}$, CC)
    $D_{[u]} = (\max(\Gamma_u) - \Gamma_u)/(\max(\Gamma_u) - \min(\Gamma_u))$
    $D_{[u]} = D_{[u]}/\text{sum}(D_{[u]})$
  end for
  return D
end function

function TemporalAggregation($\mathcal{G}$, c, D, α)
  Initialize matrix $CR \in \mathbb{R}^{|V|\times(c^2+c)}$ to 0
  for all $u \in V$ **do**
    Let $[t_1, \ldots, t_{|T(u)|}]$ be $T(u)$ sorted in ascending order
    Initialize matrix $CT_u \in \mathbb{R}^{c\times c}$ and vector $Z_u^{t_{|T(u)|}} \in \mathbb{R}^c$ to 0
    $N_u^{t_{|T(u)|}} = \sum_{v\in nbr(u,\, t_{|T(u)|})} D_{[v]}$; $CF_u = N_u^{t_{|T(u)|}}$
    for all a from $|T(u)|-1$ to 1 **do** ▷ Temporal aggregation loop
      $N_u^{ta} = \sum_{v\in nbr(u,ta)} D_{[v]}$ ▷ Neighbor aggregation (Def. 3.4)
      $CF_u = CF_u + N_u^{ta}$ ▷ Def. 3.5
      $Z_u^{ta} = e^{\frac{-(t_{a+1}-t_a)}{\alpha}}\left(N_u^{ta+1} + Z_u^{ta+1}\right)$ ▷ Lemma 3.8
      $CT_u = CT_u + (N_u^{ta})^\top Z_u^{ta}$ ▷ Def. 3.7; Lemma 3.9
    end for
    $CR_{[u]}$ = concatenate(flatten(CT_u), CF_u)
  end for
  return CR
end function

3.2 Main loop

The pseudocode of Temporal SIR-GN is shown in **Algorithm 1**, while **Table 1** summarizes its main notations, and **Figure 3** provides an example of its execution. The algorithm takes as input a temporal graph $\mathcal{G}=(V,T,\mathcal{E})$, and three parameters (all explained in more detail during the description of the algorithm):

- $d \in \mathbb{N}^+$: an upper bound on the number of iterations.
- $c \in \mathbb{N}^+$: number of clusters of node representations, which determines the dimensionality h of the embeddings ($h=c^2+c$).
- $\alpha \in \mathbb{R}^+$ modulates the impact of the temporal aggregation.

The suggested default parameters are $\alpha=1$ and $d=\infty$, so as to let the method run until the stopping criterion is met. Parameter c is set so that the resulting c^2+c embedding dimensionality is the closest to the desired h. Specifically, one can set c to either the largest integer such that $c^2+c \leq h$, or the smallest integer such that $c^2+c \geq h$, and use standard tricks if $c^2+c \neq h$. If $c^2+c<h$, the embeddings can be padded with zeros. If $c^2+c>h$, dimensionality reduction techniques can be employed (as done, e.g., in [14]).

The main principle of Temporal SIR-GN is to identify c clusters of nodes' temporal structural roles, and let the representation of

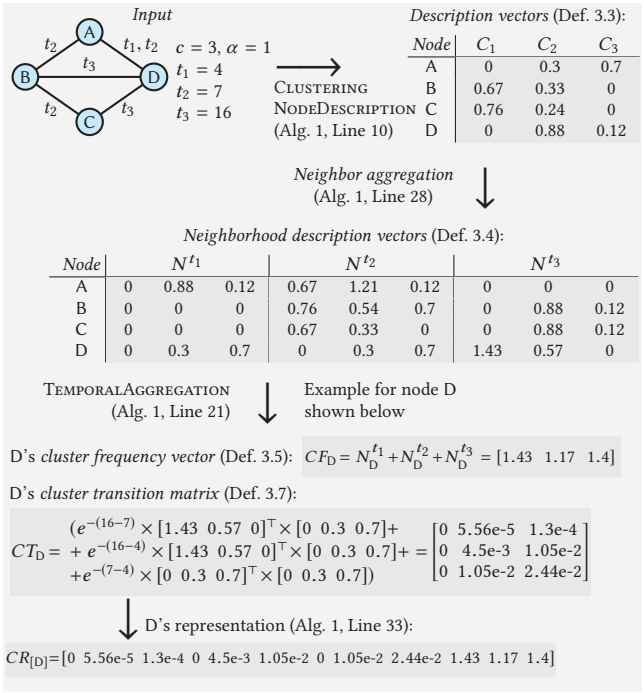


Figure 3: Run-through example of a single iteration of Algorithm 1.

a node reflect how well its k-hop neighborhood ($k \leq d$) complies with those clusters. To accomplish this, the first step consists in computing what we term nodes' *description vectors*:

Definition 3.3 (Description vector). Given c clusters of node representations, the *description vector* $D_{[u]}$ of a node $u \in V$ is a c-dimensional vector, where each component j represents the probability that u's representation belongs to cluster j.

After they are initialized to $1/c$ (Line 1), at each iteration of the main loop (Line 3) description vectors are (*i*) updated in terms of the new clustering of node representations (Line 5), and (*ii*) temporally aggregated, so as to form the actual node representations of the current iteration (Line 6). Clustering and temporal aggregation are executed for the lesser of either the user-input d iterations or until a stopping criterion is met. The stopping criterion is defined as with the WL algorithm, wherein the current number of unique node representations (*nRep*) is no longer increasing. Note that *nRep* is expected to increase iteration after iteration, because of increasing heterogeneity in the clusters, and, in turn, higher variance in the description vectors. This is in accordance with WL, and it is an opposite phenomenon to, e.g., the classic *over-smoothing* issue in GNNs [11]. The representations yielded at iteration i (R^i) are input to iteration $i+1$. Once the stopping criterion is met at iteration $k \leq d$, the representation of a node expresses the temporal evolution of that node's structure measured out to its k-hop neighborhood. Next, we describe clustering and temporal aggregation.

3.3 Clustering and node description

The ClusteringNodeDescription function (Line 10) first partitions the current node representations into c clusters (Line 13). Min-max normalization (Line 12) is performed beforehand, as a common preliminary step in clustering. As a clustering algorithm, we employ K-Means. This can be, however, replaced with any other algorithm that produces c cluster centers in the form of numerical vectors. Then, the new description vectors are computed (Line 14). Specifically, each component of the description vector $D_{[u]}$ of node u equals to the squared Euclidean distance from u to any cluster center (Line 15). These distances are in turn converted to the probabilities of membership in the various clusters (Lines 16–17).

3.4 Temporal aggregation

The TemporalAggregation function (Line 21) first computes the *neighborhood description vectors* (Line 28):

Definition 3.4 (Neighborhood description vector). The *neighborhood description vector* N_u^t of a node $u \in V$ at timestamp $t \in T(u)$ is a c-dimensional vector, where each component j is the expected number of neighbors of u at timestamp t whose representation belongs to cluster j of node representations. That is, $N_u^t = \sum_{v \in nbr(u,t)} D_{[v]}$.

The algorithm then computes the *cluster frequency vector* (Line 29):

Definition 3.5 (Cluster frequency vector). The *cluster frequency vector* CF_u of a node $u \in V$ is a c-dimensional vector, where each component j is the expected number of times cluster j appears in u's neighborhood over all the timestamps. That is, $CF_u = \sum_{t \in T(u)} \sum_{v \in nbr(u,t)} D_{[v]} = \sum_{t \in T(u)} N_u^t$.

CF_u will be part of the ultimate node representations (see below). However, it contains solely information aggregated over time. We thus complement CF_u with the *cluster transition matrix* CT_u, which keeps track of the *temporal transitions* τ_{jl} among clusters, occurring within the neighborhood of u:

Definition 3.6 (Cluster temporal transition). Given clusters j and l of node representations, a *cluster temporal transition* τ_{jl} between j and l within the neighborhood of a node $u \in V$ is the expected number of times j is observed to come before l in time in u's neighborhood. That is, $\tau_{jl} = \sum_{t,t' \in T(u), t' > t} \sum_{v \in nbr(u,t)} \sum_{v' \in nbr(u,t')} D_{[v]}[j] \times D_{[v']}[l] = \sum_{t,t' \in T(u), t' > t} N_u^t[j] \times N_u^{t'}[l]$.

The rationale of the above definition is as follows. For timestamps t, t', the expected number of times cluster j is observed in u's neighborhood at timestamp t *and* cluster l is observed in u's neighborhood at t' is $N_u^t[j] \times N_u^{t'}[l]$. As τ_{jl} is the expected number of times j is observed to come before l in time in general, here is the sum of $N_u^t[j] \times N_u^{t'}[l]$ over all $t' > t$. Intuitively, τ_{jl} expresses how often a structural pattern (cluster) j within a node's neighbors gets to another pattern l in the future. As such, cluster temporal transitions capture the temporal evolution of structural patterns.

To smooth the contribution of distant timestamps, we include a *time elapse* term $e^{-(t'-t)} \in [0,1]$, which can be interpreted as the probability that a cluster temporal transition occurs from t to t'. Additionally, we use a parameter $\alpha \geq 0$, explained below. This leads to the following ultimate definition:

Definition 3.7 (Cluster transition matrix). Given a real value $\alpha \geq 0$, the *cluster transition matrix* CT_u of a node $u \in V$ is a $(c \times c)$-dimensional matrix, where every $[j,l]$ component corresponds to the cluster temporal transition τ_{jl}, weighted by $e^{\frac{-(t'-t)}{\alpha}}$:

$$CT_u = \sum_{t,t' \in T(u),\ t' > t} e^{\frac{-(t'-t)}{\alpha}} (N_u^t)^\top N_u^{t'}. \quad (1)$$

Ultimate node representations. The cluster transition matrix CT_u is flattened (by concatenating its rows), and further concatenated to the cluster frequency vector CF_u. This forms the final node representation (embedding) $CR_{[u]}$ of node u at the current iteration of Temporal SIR-GN (Line 33). Specifically, $CR_{[u]}$ is a $(c^2 + c)$-dimensional vector, where the first c^2 components represent the expected number of temporal transitions from each cluster of node representations to each other cluster, within u's neighborhood. The remaining c components represent the overall expected number of times each cluster appears in u's neighborhood.

A large or small α makes $e^{\frac{-(t'-t)}{\alpha}}$ close to 1 or 0, respectively. The first case is equivalent to have no time elapse term at all. The second case makes $CT_u = 0$: this way the ultimate node representations will contain temporally-flattened information only (due to CF_u).

Linear time temporal aggregation. A naïve computation of Equation (1) takes quadratic time in the number $|T(u)|$ of timestamps in which a node u exists. This may lead to unaffordable running time for even moderate number of timestamps. Here, we show how to shorten this computation to linear. Let $[t_1, \ldots, t_{|T(u)|}]$ be the timestamps in $T(u)$ sorted in ascending order. Also, for any $t \in T(u)$, let Z_u^t be a c-dimensional auxiliary vector defined as:

$$Z_u^t = \begin{cases} 0, & \text{if } t = t_{|T(u)|}, \\ \sum_{t' \in T(u),\ t' > t} e^{\frac{-(t'-t)}{\alpha}} N_u^{t'}, & \text{if } t < t_{|T(u)|}. \end{cases} \quad (2)$$

The following lemma shows how to compute Z_u^t incrementally:

Lemma 3.8. *For every $a = 1, \ldots, T(u) - 1$, it holds that* $Z_u^{t_a} = e^{\frac{-(t_{a+1}-t_a)}{\alpha}} \left(N_u^{t_{a+1}} + Z_u^{t_{a+1}} \right)$.

Proof.

$$\begin{aligned} Z_u^{t_a} &= \textstyle\sum_{b=a+1,\ldots,|T(u)|} e^{\frac{-(t_b-t_a)}{\alpha}} N_u^{t_b} && \{\text{Eq. (2)}\} \\ &= e^{\frac{-(t_{a+1}-t_a)}{\alpha}} N_u^{t_{a+1}} + \textstyle\sum_{b=a+2,\ldots,|T(u)|} e^{\frac{-(t_b-t_a+t_{a+1}-t_{a+1})}{\alpha}} N_u^{t_b} \\ &= e^{\frac{-(t_{a+1}-t_a)}{\alpha}} N_u^{t_{a+1}} + e^{\frac{-(t_{a+1}-t_a)}{\alpha}} \textstyle\sum_{b=a+2,\ldots,|T(u)|} e^{\frac{-(t_b-t_{a+1})}{\alpha}} N_u^{t_b} \\ &= e^{\frac{-(t_{a+1}-t_a)}{\alpha}} \left(N_u^{t_{a+1}} + Z_u^{t_{a+1}} \right). \end{aligned}$$ □

The next further lemma shows how to express CT_u in terms of Z_u^t:

Lemma 3.9. *It holds that $CT_u = \sum_{t \in T(u)} (N_u^t)^\top Z_u^t$.*

Proof.

$$\begin{aligned} CT_u &= \textstyle\sum_{t, t' \in T(u),\ t' > t} e^{\frac{-(t'-t)}{\alpha}} (N_u^t)^\top N_u^{t'} && \{\text{Eq. (1)}\} \\ &= \textstyle\sum_{t \in T(u)} (N_u^t)^\top \sum_{t' \in T(u),\ t' > t} e^{\frac{-(t'-t)}{\alpha}} N_u^{t'} \\ &= \textstyle\sum_{t \in T(u)} (N_u^t)^\top Z_u^t. && \{\text{Eq. (2)}\} \end{aligned}$$ □

Given these lemmas, it is easily observed that Temporal SIR-GN performs a sound linear time computation of CT_u:

Theorem 3.10. *Lines 30–31 of Algorithm 1 soundly compute CT_u.*

Proof. This section of the algorithm processes all the timestamps in $T(u)$ is descending order. This way, $Z_u^{t_a}$ can be computed from $Z_u^{t_{a+1}}$ (Line 30), according to Lemma 3.8 (starting from $Z_u^{t_{|T(u)|}} = 0$, Line 25). CT_u is then computed according to Lemma 3.9 (Line 31). Note that, to compute CT_u, timestamps can be processed in any order, including the descending one used here. □

3.5 Extensions

We discuss here preliminary ideas to handle alternative settings.

Directed graphs. Separately generate representations as with the undirected method for each node's in and out edges, and concatenate both into a single representation.

Node labels/attributes. Concatenate them to the embeddings at each iteration (use one-hot encoding, if needed). This way, they can exert influence on the clustering, and, as such, on the embeddings.

Inductive setting. This setting refers to computing a "model" that can be used to yield embeddings for unseen nodes, or, in the most general case, for the nodes of an entire new temporal graph $\hat{\mathcal{G}}$. In the context of Temporal SIR-GN, the model corresponds to the vector $\widehat{CC}$ of cluster centers that have been produced at the end of a training execution of the algorithm on a temporal graph other than $\hat{\mathcal{G}}$. To get the node embeddings of $\hat{\mathcal{G}}$, it suffices to run Algorithm 1 by keeping cluster centers in Line 13 fixed and set to $\widehat{CC}$.

Time-interval representations. By default, Temporal SIR-GN generates embeddings that are representative of all the temporal snapshots $\{G_t\}_{t \in T}$ of the input temporal graph $\mathcal{G} = (V, T, \mathcal{E})$. To have embeddings specific for a time interval (or a set of timestamps) $T' \subseteq T$, one can simply take the temporal graph $\mathcal{G}' = (V, \mathcal{E}', T')$, $\mathcal{E}' = \{(u, v, t) \in \mathcal{E} \mid t \in T'\}$ composed of all the snapshots corresponding to timestamps in T', and run the algorithm on $\mathcal{G}'$.

4 ALGORITHM ANALYSIS

In this section, we analyze the proposed Temporal SIR-GN algorithm, from both a theoretical and an empirical point of view.

4.1 Computational complexity

Time complexity. The ClusteringNodeDescription function (Line 10) runs K-Means on the rows of matrix $RN \in \mathbb{R}^{|V| \times (c^2+c)}$, with number of clusters set to c. This takes $O(|V| \times c^3)$ time (by reasonably assuming the number of K-Means iterations is a constant). Then (Line 14), it computes distances between node representations and cluster centers, plus some normalization of the resulting vectors. This again takes $O(|V| \times c^3)$ time, which corresponds to the overall time complexity of ClusteringNodeDescription.

The runtime of the TemporalAggregation function is dominated by the steps at Line 28 and 31. In the former, neighbor aggregation is performed, which takes, for a node u and timestamp t, $O(c \times |nbr(u, t)|)$ time, as it sums up a number $|nbr(u, t)|$ of c-dimensional vectors. This is repeated for every node u and every timestamp $t \in T(u)$, which leads to overall $O(\sum_{u \in V} \sum_{t \in T(u)} c \times |nbr(u, t)|) = O(|\mathcal{E}| \times c)$ time. The step at Line 31 aggregates $(c \times c)$-dimensional matrices for every node $u \in V$ and over all timestamps $t \in T(u)$, thus it takes $O(\mathcal{T} \times c^2)$ time. As a result, the overall time complexity of TemporalAggregation is $O(|\mathcal{E}| \times c + \mathcal{T} \times c^2)$.

Altogether, ClusteringNodeDescription and TemporalAggregation take $O(|\mathcal{E}| \times c + \mathcal{T} \times c^2 + |V| \times c^3)$. Considering that those functions are executed for at most d iterations in the main

loop of the algorithm (Line 3), and that $c = O(\sqrt{h})$, then the ultimate time complexity of Temporal SIR-GN can be expressed as $O(d \times (|\mathcal{E}| \times \sqrt{h} + \mathcal{T} \times h + |V| \times h\sqrt{h}))$. If d and h are fixed (i.e., they are constant), this simplifies to $O(|\mathcal{E}|)$, since $\mathcal{T} = O(|\mathcal{E}|)$.

Space complexity. Besides the input graph, the largest data structures that the algorithm needs to keep in memory at each iteration i are matrices (i.e., R and RN) of dimensionality $|V| \times (c^2 + c)$. Thus, the overall space complexity is $O(|\mathcal{E}| + |V| \times c^2) = O(|\mathcal{E}| + |V| \times h)$.

4.2 Theoretical properties

Temporal SIR-GN comes with theoretical guarantees if the input temporal graph exhibits a temporal automorphism (**Definition 3.2**). Specifically, as a first theoretical property, we show that the temporal aggregation step of Temporal SIR-GN guarantees equal output node representations if equal description vectors for any two temporally-automorphic nodes are used. This result is stated in the following **Theorem 4.2**, and makes use of the next auxiliary lemma, which states that the neighbors of temporally-automorphic nodes must be in turn temporally-automorphic:

Lemma 4.1. *Let $\mathcal{F}$ be a temporal automorphism in a temporal graph $\mathcal{G}(V, T, \mathcal{E})$ such that, for $u, v \in V$, $\mathcal{F}(u) = v$ with a certain Δ_u. It holds that $\forall t \in T(u), \forall x \in nbr(u, t), \exists y \in nbr(v, t + \Delta_u)\colon \mathcal{F}(x) = y$.*

Proof. We prove the lemma by contradiction. Assume $\exists t' \in T(u), x' \in nbr(u, t')\colon \mathcal{F}(x') = z, z \notin nbr(v, t' + \Delta_u)$. Then, by definition of temporal automorphism, given edge (u, x', t'), there must exist edge $(\mathcal{F}(u), \mathcal{F}(x'), t' + \Delta_u) = (v, z, t' + \Delta_u)$. This means that $z \in nbr(v, t' + \Delta_u)$, which contradicts the assumption. □

Theorem 4.2. *Let there be a temporal graph $\mathcal{G} = (V, T, \mathcal{E})$, if the TemporalAggregation function of Algorithm 1 (Line 21) receives in input a matrix D such that for any two temporally-automorphic nodes $u, v \in V$ it holds that $D_{[u]} = D_{[v]}$, then $CR_{[u]} = CR_{[v]}$.*

Proof. If $D_{[u]} = D_{[v]}$ for temporally-automorphic nodes u and v (hypothesis), then, by Lemma 4.1, the neighbor aggregation at Line 28 for each will result in identical vectors $N_u^{t_u} = N_v^{t_u + \Delta_u}$, for all $t_u \in T(u)$. A further straightforward consequence of Lemma 4.1 is that $\forall t_u \in T(u), \exists t_v \in T(v)\colon t_v = t_u + \Delta_u$, and vice versa. This consequence along with identical $N_u^{t_u}$, $N_v^{t_u + \Delta_u}$ vectors leads to identical summation over all the timestamps of each neighbor aggregation (the cluster frequency vector, Line 29), i.e., $CF_u = \sum_{t_u \in T(u)} N_u^{t_u} = \sum_{t_u \in T(u)} N_v^{t_u + \Delta_u} = \sum_{t_v \in T(v)} N_v^{t_v} = CF_v$.

Then, cluster transition matrices are computed as in Definition 3.7. As (from above) all $t_u \in T(u)$ differ from $t_v \in T(v)$ by Δ_u, then $e^{\frac{-(t'_u - t_u)}{\alpha}} = e^{\frac{-(t'_u - t_u + \Delta_u - \Delta_u)}{\alpha}} = e^{\frac{-(t'_v - t_v)}{\alpha}}$. Coupling this with the above consequence of identical $N_u^{t_u}$, $N_v^{t_u + \Delta_u}$ leads to $CT_u = \sum_{t_u, t'_u \in T(u),\ t'_u > t_u} e^{\frac{-(t'_u - t_u)}{\alpha}} (N_u^{t_u})^\top N_u^{t'_u} =$
$= \sum_{t_u + \Delta_u, t'_u + \Delta_u \in T(u),\ t'_u > t_u} e^{\frac{-(t'_u - t_u + \Delta_u - \Delta_u)}{\alpha}} (N_u^{t_u + \Delta_u})^\top N_u^{t'_u + \Delta_u} =$
$= \sum_{t_v, t'_v \in T(v),\ t'_v > t_v} e^{\frac{-(t'_v - t_v)}{\alpha}} (N_v^{t_v})^\top N_v^{t'_v} = CT_v$. The theorem follows as $CR_{[u]} = (\text{flatten}(CT_u)\ CF_u) = (\text{flatten}(CT_v)\ CF_v) = CR_{[v]}$. □

A second property proved below regards the overall embeddings yielded by Temporal SIR-GN, which are guaranteed to be equal for temporally-automorphic nodes:

Theorem 4.3. *Given a temporal graph $\mathcal{G}$, for any two temporally-automorphic (see Definition 3.2) nodes u, u' in $\mathcal{G}$, the embeddings $R_{[u]}$ and $R_{[u']}$ computed by Algorithm 1 are equal.*

Proof. We apply a proof by induction. The base case consists in showing that any two temporally-automorphic nodes u and u' have identical initial representations $R^0_{[u]} = R^0_{[u']}$. In this regard, note that D^0 vector is initialized with a constant (Line 1); then, clearly, $D^0_{[u]} = D^0_{[u']}$. R^0 is the output of the temporal aggregation with D^0 in input: then, $R^0_{[u]}$ and $R^0_{[u']}$ must be equal by Theorem 4.2.

Now, we assume that the theorem is true for iteration i, and prove it for iteration $i+1$. This can be accomplished by noticing that equal representations $R^i_{[u]}$ and $R^i_{[u']}$ for temporally-automorphic nodes u and u' lead to equal description vectors $D^i_{[u]}$ and $D^i_{[u']}$ (Line 5). In fact, regardless of the specific cluster centers, the distance between $R^i_{[u]}$, $R^i_{[u']}$ and all those centers are the same, which means that $D^i_{[u]}$ and $D^i_{[u']}$ are the same too. Also, by Theorem 4.2, equal $D^i_{[u]}$ and $D^i_{[u']}$ lead to equal $R^{i+1}_{[u]}$ and $R^{i+1}_{[u']}$ (Line 6).

The theorem now follows by simply observing that the final embeddings $R_{[u]}$ and $R_{[u']}$ are the ones produced in the last iteration, which must be equal like the other iterations. □

Theorem 4.3 has two important consequences. The first is that for nodes with identical temporal structures, Temporal SIR-GN generates identical representations. This is vital to effectively capture temporal structural roles. Note that **Theorem 4.3** provides a sufficient condition. Deriving a necessary condition too is hard, as it would correspond to having found a polynomial-time algorithm for the problem of Graph Isomorphism, which is still a crucial open question in theoretical computer science [23].

The second consequence is that Temporal SIR-GN guarantees *time invariance*: nodes with identical neighborhood structures and identical *intervals* between timestamps have identical representations, no matter if the absolute timestamps are similar. Time invariance allows a temporal NRL model to capture similarity between events that are close in structure, but occur at different times.

4.3 Empirical properties

Although Temporal SIR-GN exhibits theoretical properties for the limit case of temporally-automorphic nodes, deriving formal guarantees for the general case is not easy. This goes beyond the scope of this work, and we defer it to the future. Instead, here we provide empirical intuitions of why our algorithm is generally well-designed for the target Temporal Structural NRL problem.

First of all, we remark that a connection between the desideratum of complying with temporal structural patterns that resemble a notion of graph isomorphism is the fact that Temporal SIR-GN emulates the WL isomorphism test. In fact, inspired by WL, the temporal aggregation in Temporal SIR-GN yields an embedding vector where part of the components correspond to the (expected) number a certain temporal structural pattern is exhibited in the neighbors of a node. These are complemented with novel components representing the (expected) number of temporal transitions among patterns. Intuitively, the closer two nodes are to be temporally automorphic,

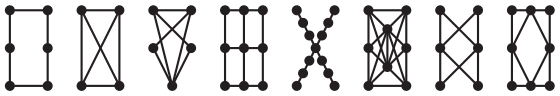


Figure 4: Static graph patterns used as a basis for the temporal graph patterns underlying the synthetic datasets.

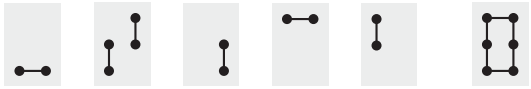


Figure 5: Example of temporal graph pattern underlying the synthetic datasets, derived from "building" the left-most static pattern in Figure 4 edge-by-edge over a sequence of timestamps. Aggregating the temporal edges from the $[t_1, t_5]$ interval leads to the static pattern at hand.

the more they share such structural patterns, then the more similar the components of their corresponding embedding vectors.

More in concrete, consider the following experiment. Given a temporal graph $\mathcal{G} = (V, T, \mathcal{E})$ and $\epsilon \in (0, 1]$, let $\mathcal{G}^\epsilon$ be the graph resulting from the addition of a number $\epsilon|\mathcal{E}|$ of random temporal edges $(u, v, t) \notin \mathcal{E}, u, v \in V, t \in T$ to $\mathcal{G}$. We generate $\mathcal{G}^{\epsilon_i}$, for $i = 0, \ldots, 5$, with $\epsilon_0 = 0, \epsilon_1 = 0.1, \ldots, \epsilon_5 = 0.5$. Every $\mathcal{G}^{\epsilon_i}$ is built adding random edges on top of $\mathcal{G}^{\epsilon_{i-1}}$. We compute Temporal SIR-GN's embeddings of $\mathcal{G}$ and all $\mathcal{G}^{\epsilon_i}$ and measure the average distance $\bar{d}_{\epsilon_i}$ between the embedding of every node in $\mathcal{G}$ and the "replica" of that node in $\mathcal{G}^{\epsilon_i}$, for all ϵ_i. The rationale is as follows. Between $\mathcal{G}$ and $\mathcal{G}^{\epsilon_0}$ there is a temporal isomorphism, as they are identical graphs. Then, $\bar{d}_{\epsilon_0} = 0$ is expected. From ϵ_1 on, the temporal isomorphism progressively disappears, due to the increasing addition of random edges. Thus, the desideratum here is to observe $\bar{d}_{\epsilon_i} < \bar{d}_{\epsilon_i+1}$, for all $i = 0, \ldots, 4$. As shown in the following table (for the real dataset DPPIN, cf. **. Section 5**), this is actually the case:

$\bar{d}_{\epsilon_0}$, $\epsilon_0 = 0$	$\bar{d}_{\epsilon_1}$, $\epsilon_1 = 0.1$	$\bar{d}_{\epsilon_2}$, $\epsilon_2 = 0.2$	$\bar{d}_{\epsilon_3}$, $\epsilon_3 = 0.3$	$\bar{d}_{\epsilon_4}$, $\epsilon_4 = 0.4$	$\bar{d}_{\epsilon_5}$, $\epsilon_5 = 0.5$
0	1.959	3.117	6.799	8.623	10.480

Finally, as a specific example where Temporal SIR-GN meets the desideratum that embedding similarity reflects to which extent the corresponding nodes are temporally automorphic, consider the graph in **Figure 2**. The (12-dimensional) embeddings produced by Temporal SIR-GN for the lettered nodes of that graph are:

$R_{[A]}$	0.2	0	0.163	0	0	0	0.275	0	0.233	1.449	0	1.551
$R_{[B]}$	0.065	0	0.074	0	0	0	0.108	0	0.121	0.848	0	1.151
$R_{[C]}$	0	0	0	0	0	0	0	0	0	0.378	0	0.622

The distances between the embedding of node A and the embeddings of the other nodes are $|R_{[A]} - R_{[B]}| = 1.5 < |R_{[A]} - R_{[C]}| = 2.87$. These distances comply with the size of the corresponding maximal temporally isomorphic subgraphs (**Figure 2–(I)-(II)**).

5 EXPERIMENTS

In this section, we empirically evaluate efficiency and effectiveness of the proposed Temporal SIR-GN (for short, T-SIRGN), and compare it to the state of the art. Efficiency is evaluated in terms of the runtime needed to generate the embeddings. Effectiveness is evaluated by using the generated embeddings in a couple of downstream machine-learning tasks, namely *node classification* and *regression*.

Table 2: Dataset characteristics. $|V|$: #nodes; $|T|$: #timestamps; $|\mathcal{E}|$: #temporal edges; $|E|$: #non-temporal edges (i.e., #node pairs sharing at least one temporal edge); $\mathcal{T}$: $\sum_{u \in V} |T(u)|$, where $T(u)$ is the set of timestamps in which node u exists; #distinct node labels (i.e., classes).

Dataset	$\lvert V\rvert$	$\lvert T\rvert$	$\lvert\mathcal{E}\rvert$	$\lvert E\rvert$	$\mathcal{T}$	*#Labels*
Synth0.0	20 280	28	27 768	27 768	54 912	24
Synth0.1	20 280	100	29 796	29 796	58 852	24
Synth0.2	20 280	100	31 824	31 824	62 792	24
Synth0.3	20 280	100	33 852	33 852	66 708	24
BrazilAir	39 300	31	354 420	354 415	446 836	12
EUAir	119 700	61	1 978 350	1 978 319	2 529 289	12
USAir	348 110	101	4 487 670	4 433 165	6 246 184	12
Hospital [1, 74]	75	9 453	32 424	1 139	50 645	4
HS [1, 17]	180	11 273	45 047	2 239	79 578	5
Bitcoin [34, 35]	5 881	35 592	35 592	35 592	71 184	2
DPPIN [19]	905	36	4 826	1 758	4 462	2
GDELT [98]	16 682	170 522	191M	191M	65M	80
Facebook [77]	4 117	10	8 029	5 143	10 226	–
AS [37]	6 828	100	1 947 704	17 364	475 765	–
UCIMsg [49]	1 899	7	22 663	13 838	4 558	–

Datasets. We experiment with synthetic and real datasets, whose characteristics are shown in Table 2 and described below.

Synthetic benchmark datasets. Four synthetic benchmark datasets were generated (Synth0.0–Synth0.3). We started from the 8 static graph structural patterns in **Figure 4** (popular in the structural NRL literature [30]), and we used them as a basis for creating *temporal patterns* that ultimately compose the synthetic datasets. Specifically, we first sort the set E of edges in a static pattern at random, so as to yield a sequence $e_1, \ldots, e_{|E|}$. Then, every $e_i = (u_i, v_i)$ is assigned a timestamp t_i that is sampled from the set $T = \{1, \ldots, 100\}$. The result is a temporal edge (u_i, v_i, t_i). Timestamp sampling is *with replacement*, so that the same timestamp can be assigned multiple edges. An example of this process is illustrated in **Figure 5**. For every static pattern, we considered 3 different random orderings of its edges, and associated every ordering to a sequence of timestamps. As a result, every static pattern yields 3 different temporal patterns, for a total of 24 temporal patterns. Nodes' class labels are defined by letting a temporal pattern be representative of the (temporal) structural role of that pattern's nodes. Hence, each pattern was assigned a different label, which was in turn used as a label for all the nodes of that pattern. A node may possibly be part of multiple patterns. However, we ensured a single label per node by setting an ordering (at random) among patterns, and associating a node to the label of the first pattern in the ordering that node appears in.

We repeated this process 104 times, setting different identities for all the nodes of the generated temporal edges. This led to the Synth0.0 dataset. We created subsequent datasets from it, each with additional noise in the form of randomly generated temporal edges between already existing nodes. Specifically, Synth0.1 corresponds to $0.1 \times |\mathcal{E}_{0.0}|$ random temporal edges added, where $\mathcal{E}_{0.0}$ are the temporal edges in Synth0.0, and so on. Note that the addition of noise makes it possible that in Synth0.1–Synth0.3 datasets the same pair of nodes is connected by an edge in multiple timestamps.

Temporally-adapted real datasets. We took three popular static real datasets from the air traffic domain (that are available, among others, from [54]), and converted them to temporal versions, namely BrazilAir, EUAir, and USAir. In these datasets, nodes, edges, and labels represent airports, air traffic, and airport designations as high to low traffic, respectively. Similar to our synthetic datasets, each original static graph was used as a base structure, but temporally constructed according to a time sequence. We used 3 time sequences, and the node classes correspond to the original 4 classes, along with the corresponding sequence, for a total of 12 ultimate classes. This was repeated for each temporal structure 100 times.

Real labeled temporal datasets. These are real temporal graphs with class labels on nodes. Hospital [1, 74] contains (RFIDs) contacts in a hospital ward in Lyon, France during Dec 6th-10th, 2010, in 20-second intervals. Node labels identify a node as a patient, medical doctor, nurse, administrative. HighSchool (HS) [1, 17] contains contacts in 5 classes in a high school in Marseilles, France during 7 days in Nov 2012, in 20-second intervals, with node labels corresponding to the class of a student. Bitcoin [34, 35] is a who-trusts-who network of traders on the Bitcoin OTC platform (we ignore edge weights). Timestamps represent the time of rating. Node labels correspond to trustworthy/untrustworthy users. DPPIN [19] consists of the protein-protein interactions of yeast cells through 12 stages of 3 metabolic cycles, for a total of 36 timestamps. Node labels identify proteins as uncharacterized/verified. GDELT [98] is a graph derived from the GDELT 2.0 Event DB, comprised of a record information taken every 15 mins from news sources over 2016 to 2020. Nodes are actors, edges are events. Node labels correspond to the country where the actor was present during that event.

Real unlabeled temporal datasets. Facebook [77] is a 3-month subset of Facebook user interaction from a New Orleans community. The original Facebook dataset had 9 984 snapshots, most with a single edge only. In our experiments, we used a more meaningful version of the dataset where we aggregate consecutive snapshots into 10 uniformly-sized bins. AutonomousSystems (AS) [37] is a communication network from Border Gateway Protocol logs. UCIMsg [49] is a directed graph (we ignore edge directionality) of messages between users of an online community at University of California Irvine. As with Facebook, the original UCIMsg had 59 811 snapshots, most with one edge only. We again here aggregated snapshots into 7 uniformly-sized bins (as suggested in [41]).

Competitors. We involve the following state-of-the-art methods.

Liu *et al.*'s CTGCN [41], the most direct competitor, in three variants: U-CTGCN-S (unsupervised, structural loss), U-CTGCN-C (unsupervised, connectivity-preserving loss), S-CTGCN-C (supervised, connectivity-preserving loss).

DynGem [22], GCRN [62], TGAT [88], TGN [55] as representatives of temporal-GNN-based NRL. As for GCRN, we assess both the supervised (S-GCRN) and unsupervised (U-GCRN) versions. DynGem, TGAT, TGN are solely unsupervised.

TIMERS [95], as a representative of proximity-based temporal NRL approaches. This method is also strictly unsupervised.

DGI [76], NWR [69], SIR-GN [30] are tested as representatives of (different classes of) static NRL approach: DGI is a GNN-based method; NWR is a "hybrid" method, which combines structural and proximity-based NRL; SIR-GN (the precursor of our T-SIRGN) is a purely structural method. These methods are run on the flattened input temporal graph (i.e., a static graph where an edge is drawn between any two nodes if they share at least one temporal edge).

Within this category of competitor, we also include a version of our T-SIRGN, termed StructuralShifted-T-SIRGN (for short, SS-TSIRGN), where we let α approach 0. This leads to embeddings that reflect the temporally-flattened structural aspect only (i.e., due to cluster frequency vector CF, cf. **Section 3.4**). As such, SS-TSIRGN corresponds to a static structural NRL method that is run on a *weighted* flattened version of the input temporal graph (where edge weights are the number of timestamps in which that edge appears).

For CTGCN, TGAT, TGN, DGI, NWR, SIR-GN, we use the official public implementations [30, 41, 55, 69, 76, 88]. For the remaining competitors, we use the implementations in the CTGCN repository.

Parameters. Unless otherwise specified, all the competitors are tested using their default/suggested parameters. In T-SIRGN (and SS-TSIRGN), a large d is used, so as to let it run until the stopping criterion is met (cf. **Section 3.2**), while α appropriate to every dataset and experiment was chosen (details reported case by case). The size h of the output embeddings is set to 128 for all the methods. For our T-SIRGN (and SS-TSIRGN), this corresponds to $c = 10$.

Assessment. For node classification (**Section 5.1**), we train a classifier using the embeddings as feature vectors and the node labels as a target variable to be predicted. We tried *Extra Trees*, *XGBoost*, *MLP* classifiers. Unless otherwise specified, the results refer to *Extra Trees*. We measure *accuracy* (*Acc*) and $F1$ (both $\in [0\%, 100\%]$, higher values meaning better performance) by 5-fold cross-validation.

For regression, (**Section 5.2**), *PageRank* (*PR*), *degree centrality* (*DC*), *hubs and authorities* (*HITS*), *betweenness centrality* (*BC*), and *eigenvector centrality* (*EC*) metrics are computed for every node and snapshot, then summed over all timestamps, to have temporally-aggregated scores for every node. We train a regressor (*Random Forest*) using the embeddings as feature vectors, and each aggregated score as a target variable to be predicted (one regressor per metric). The performance is measured in terms of *coefficient of determination* ($r^2 \in (-\infty, 1]$, higher values corresponding to better performance) and *mean squared error* ($MSE \in [0, +\infty)$, lower values corresponding to better performance), by 5-fold cross-validation.

Testing environment. For timed experiments, all methods were run on a single machine equipped with an Intel 9900k 5GHz CPU, 64GB RAM, and an Nvidia RTX 3090 GPU with 24GB of memory.

5.1 Node classification

Table 3 shows the node classification results. On all the synthetic datasets (Synth0.0–Synth0.3; results refer to *XGBoost* classifier), T-SIRGN outperforms all other methods. T-SIRGN reaches perfect *Acc* and $F1$ on the noise-free Synth0.0. From Synth0.1 on, noise in terms of random temporal edges is added. Thus, T-SIRGN expectedly shows an incremental performance decrease as the noise increases.

T-SIRGN is the best performer on the temporally-adapted datasets too. Note that these datasets are much larger than the synthetic ones. As such, DynGem and TGAT were not able to run in reasonable time (i.e., within 48 hours) on two of them, while TGN was unable to run on all such datasets due to memory constraints.

T-SIRGN consistently outperforms all static NRL methods too (i.e., DGI, NWR, SIR-GN, SS-TSIRGN). This demonstrates that our

Table 3: Node classification of our T-SIRGN vs. its competitors. *Accuracy* (*Acc* ∈ [0%, 100%]) and $F1 \in [0\%, 100\%]$ assessment criteria (higher values mean better performance). Best results in bold, second best in italic.

(a) Synthetic and temporally-adapted datasets

Method	Synth0.0		Synth0.1		Synth0.2		Synth0.3		BrazilAir		EUAir		USAir	
	Acc	*F1*	*Acc*	*F1*	*Acc*	*F1*	*Acc*	*F1*	*Acc*	*F1*	*Acc*	*F1*	*Acc*	*F1*
DynGem	8	2	8	2	8	2	7	2	16	11	–	–	–	–
TIMERS	8	2	8	2	7	1	8	3	10	2	9	2	9	2
U-GCRN	6	4	6	4	6	4	6	4	12	12	11	10	11	10
S-GCRN	6	3	7	5	8	6	9	8	8	6	9	9	11	11
U-CTGCN-S	8	2	8	2	8	2	8	2	16	9	14	7	12	6
U-CTGCN-C	17	15	8	6	7	6	7	6	*33*	*33*	8	8	12	12
S-CTGCN-C	17	16	7	6	9	8	10	9	44	45	21	22	11	11
TGAT	*93*	*93*	*82*	*83*	*73*	*74*	*68*	*69*	*51*	*51*	–	–	–	–
TGN	9	5	9	5	8	5	7	5	–	–	–	–	–	–
DGI	28	24	25	22	20	18	16	15	23	23	16	16	17	16
NWR	33	30	31	30	25	25	25	25	26	26	25	25	10	10
SIR-GN	30	26	44	42	35	35	26	26	32	32	29	29	29	29
SS-TSIRGN	30	27	42	40	33	33	26	26	31	31	*28*	*28*	*25*	*25*
T-SIRGN	**100**	**100**	**88**	**89**	**80**	**81**	**71**	**72**	**80**	**81**	**74**	**74**	**45**	**45**

(b) Real labeled datasets

Method	Hospital		HS		Bitcoin		DPPIN	
	Acc	*F1*	*Acc*	*F1*	*Acc*	*F1*	*Acc*	*F1*
DynGem	39	14	23	7	57	36	98	50
TIMERS	41	17	24	10	66	65	98	50
U-GCRN	35	19	22	22	56	49	98	50
U-CTGCN-S	41	17	24	10	57	36	98	50
U-CTGCN-C	35	20	14	13	57	49	98	50
TGAT	**75**	**58**	38	38	*81*	*81*	97	49
TGN	45	33	42	41	66	65	98	49
DGI	35	23	28	26	70	69	98	49
NWR	35	23	29	27	65	64	97	49
SIR-GN	*55*	37	44	42	80	80	97	49
SS-TSIRGN	*35*	*30*	*48*	*46*	80	80	97	49
T-SIRGN	52	*42*	**48**	**46**	**85**	**85**	**98**	**59**

datasets are effective in testing for a method's ability to capture not only structural, but temporal structural information.

The superiority of T-SIRGN is confirmed on the real datasets. T-SIRGN is the best performer on Bitcoin and DPPIN. In this regard, note that DPPIN is highly unbalanced, with the majority label spanning the 98% of the labels. For DPPIN, thus, the *Acc* measure is not really meaningful. What matters is *F1*, in terms of which T-SIRGN outperforms its competitors by at least 9 percentage points.

GDELT dataset. As for GDELT, we provide here a separate discussion, as the experiment was slightly different due to the time-varying nature of the node labels. Specifically, the graph from 2018–19 (spanning 14k nodes, 91M temporal edges, 69k timestamps) was used as a training set to compute a T-SIRGN's model (cf. **Section 3.5**, "**Inductive setting**"). Then, for each month of 2020, we (*i*) computed embeddings based on trained T-SIRGN's model, (*ii*) trained a classifier (*Extra Trees*) with those embeddings, and (*iii*) measured *F1* by a temporal 80/20 train/test split. The average *F1* over all months is 12.95%. This value is higher than the state-of-the-art one (11.9%) reported by Zhou *et al.* [98] for a similar experiment. We remark that this is a classification task with 80 classes, thus even an improvement of one percentage point is relevant. Zhou *et al.* [98] provide a framework on which to run temporal GNNs in faster time. They involve both TGAT and TGN, and TGN demonstrated the highest performance on GDELT, at *F1* = 11.9% (while other methods clustered around *F1* = 10-11%).

Table 4: Regression of our T-SIRGN vs. its competitors. *PageRank* (*PR*), *degree centrality* (*DC*), *hubs and authorities* (*HITS*), *betweenness centrality* (*BC*), and *eigenvector centrality* (*EC*) metrics. *Coefficient of determination* ($r^2 \in (-\infty, 1]$, higher values mean better performance), and *mean squared error* (*MSE* $\in [0, +\infty)$, lower values mean better performance) assessment criteria. Best results in bold, second best in italic.

	Method	*PR*		*DC*		*HITS*		*BC*		*EC*	
		r^2	*MSE*	r^2	*MSE*	r^2	*MSE*	r^2	*MSE*	r^2	*MSE*
Facebook	DynGem	-9.583	0.150	-6.195	0.109	-1.782	0.028	-1.316	0.0575	-1.407	0.0571
	TIMERS	-5.70	0.146	-3.551	0.107	-1.172	0.0263	-0.847	0.0555	-0.924	0.0555
	U-GCRN	-5.070	0.147	-3.243	0.109	-2.080	0.0288	-1.867	0.0601	-1.862	0.0596
	U-CTGCN-S	-9.04	0.145	-5.610	0.107	-0.574	0.0257	-0.667	0.0536	-0.607	0.0522
	U-CTGCN-C	-3.217	0.140	-2.126	0.102	-0.451	0.0268	-0.137	0.0489	-0.147	0.0496
	TGAT	0.82	2.49e-3	0.728	2.44e-3	-0.229	**4.85e-4**	0.113	4.47e-3	-0.031	2.49e-3
	TGN	-0.104	3.05e-2	-0.0807	2.04e-2	-0.873	1.56e-3	-0.289	6.24e-3	-0.141	7.47e-3
	SS-TSIRGN	*0.912*	*1.26e-3*	**0.971**	**2.64e-4**	*0.0379*	7.93e-3	*0.306*	*3.46e-3*	*0.229*	*2.64e-3*
	T-SIRGN	**0.922**	**1.09e-3**	*0.967*	*3.29e-4*	**0.112**	*7.02e-3*	**0.419**	**2.96e-3**	**0.358**	**2.28e-3**
UCIMsg	DynGem	0.267	0.0505	0.0296	0.0791	0.0055	0.0513	-0.265	0.0479	0.0423	0.0667
	TIMERS	0.307	0.0509	0.0831	0.0786	0.154	0.0498	0.175	0.0453	0.076	0.0789
	U-GCRN	0.0853	0.0549	0.0193	0.0775	0.136	0.0496	-0.667	0.0538	0.204	0.0618
	U-CTGCN-S	0.371	0.0488	0.0512	0.0793	0.146	0.0496	0.0447	0.0454	0.135	0.065
	U-CTGCN-C	0.48	0.0442	0.411	0.0622	0.403	0.0421	-0.165	0.0479	0.556	0.047
	TGAT	0.425	3.103e-3	0.424	3.92e-3	0.29	3.31e-3	0.049	2.99e-3	0.391	3.1e-3
	TGN	-0.185	5.16e-3	-0.161	7.6e-3	-0.214	4.19e-3	-0.29	4.05e-3	-0.117	6.39e-3
	SS-TSIRGN	*0.538*	**2.11e-3**	*0.878*	*8.0e-4*	*0.454*	**1.95e-3**	**0.369**	**2.17e-3**	*0.720*	**1.55e-3**
	T-SIRGN	**0.559**	*2.43e-3*	**0.887**	**7.74e-4**	**0.468**	*2.0e-3*	*0.241*	*2.22e-3*	**0.723**	*1.6e-3*
AS	DynGem	-0.618	0.006	-0.63	0.006	-0.66	0.006	-2.446	0.006	-0.207	0.01
	TIMERS	-0.777	0.007	-0.702	0.007	-0.708	0.007	-8.55	0.006	-0.057	0.009
	U-GCRN	-143.3	0.011	-231.6	0.011	-179	0.011	-19622	0.01	-3.756	0.0153
	U-CTGCN-S	-0.07	5.45e-3	-0.0839	0.0054	-0.081	0.0054	-0.1872	0.005	-1.624	0.0136
	U-CTGCN-C	-0.786	0.007	-0.784	0.006	-0.824	0.007	-12.27	0.006	-0.748	0.012
	TGAT	-0.216	9.33e-4	-0.0491	8.05e-4	0.0752	8.83e-4	–	–	0.0937	9.44e-4
	TGN	–	–	–	–	–	–	–	–	–	–
	SS-TSIRGN	*0.925*	*6.45e-5*	**0.963**	*4.94e-5*	**0.952**	**4.78e-5**	**0.807**	*6.9e-5*	**0.926**	**5.06e-5**
	T-SIRGN	**0.933**	**6.24e-5**	*0.956*	**4.49e-5**	**0.952**	*4.82e-5*	*0.769*	**6.43e-5**	*0.9*	*6.19e-5*

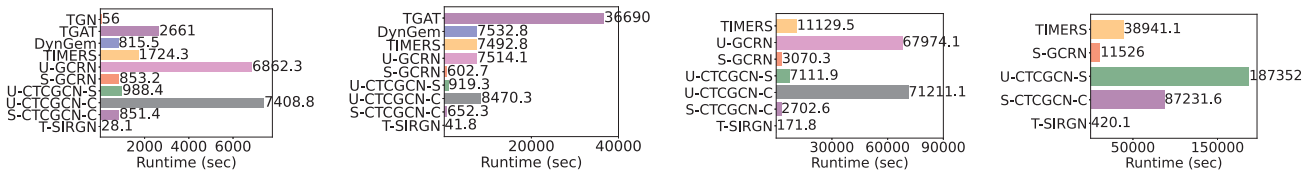
Temporal SIR-GN took about 30 minutes on the training set. None of the (implementations we used for the) selected competitors could run on GDELT on our hardware. Zhou *et al.* report training times on GDELT (for 2016–18) of 8 500 (TGAT) and 900 (TGN) seconds *for a single epoch*, which are quite a lot, and they are anyway achieved for implementations within their efficient framework.

5.2 Regression

Table 4 shows the results in the regression task. Note that static NRL methods are not included here, as they run on a static version of the graph, which differ from the temporal graphs such that comparison is not sensible. In general, T-SIRGN achieves r^2 close to one and/or *MSE* close to zero, resulting the best performer in most cases. In the few cases where T-SIRGN is the second highest performer, the structurally shifted version of our method (SS-TSIRGN) is the highest (or TGAT, in just one case). This complies with the design principles of this experiment: SS-TSIRGN emulates temporal aggregation in its execution, thus it is not surprising that it is good at predicting scores aggregated over time.

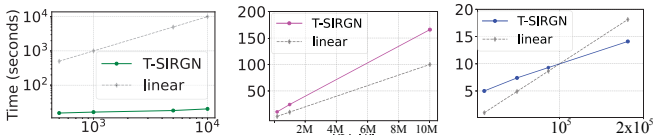
5.3 Efficiency

Running times. In **Figure 6**, we show the runtimes of our T-SIRGN and all its competitors (but SS-TSIRGN, as it is a variant of T-SIRGN, thus it runs comparably to it, and the static NRL methods, for which the comparison here is not meaningful, as these run on static yet much smaller versions of the temporal graph with flattened timestamps). We report results on the Synth0.1, BrazilAir, EUAir, and USAir datasets, representing a full range of graph sizes, total timestamps, and timestamps per node.



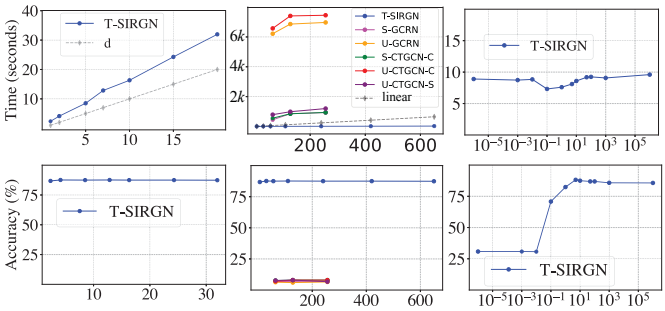
(a) Synth0.1 (b) BrazilAir (c) EUAir (d) USAir

Figure 6: Runtime of the proposed T-SIRGN and its competitors on several datasets. Actual time in seconds displayed on each bar.



(a) $|V|$ (b) $|\mathcal{E}|$ (c) $\mathcal{T}$

Figure 7: Effect of number of nodes ($|V|$), temporal edges ($|\mathcal{E}|$), and $\mathcal{T}$ on the running time of the proposed T-SIRGN, on random temporal graphs of varying size and number of snapshots.



(a) Depth d (b) Embedding size h (c) Parameter α

Figure 8: Runtime and accuracy (Acc) effects of parameter choice on the Synth0.1 dataset.

Runtimes were determined on a single machine (cf. beginning of **Section 5**). The GNN-based methods (DynGem, GCRN, CTGCN) run on a GPU, while the TIMERS implementation we use in our experiments is a CPU multi-threaded one. For the S-CTGCN-C method, memory constraints required usage of the CPU rather than the GPU on USAir. TGN and TGAT were both run on a GPU, and memory constraints prevented the completion of TGN for any airline dataset. DynGem, U-GCRN, and U-CTGCN-C could not terminate within 52 hours on USAir, thus we do not report their results. For TGAT, the same happened on both EUAir and USAir. Our T-SIRGN was tested here using a *single-threaded* CPU implementation. Thus, it is under adverse conditions with respect to the competitors. Despite that, it shows exceedingly shorter runtimes, sometimes greater than two orders of magnitude faster than others.

Scalability. We also determined the effects of nodes $|V|$, temporal edges $|\mathcal{E}|$, and timestamps per node ($\mathcal{T}$) on T-SIRGN's runtime on random temporal graphs (using $\alpha = 10$, $d = 5$). $|V|$ was varied in a graph with $|\mathcal{E}| = 100k$, and $\mathcal{T} = 200k$, while $|\mathcal{E}|$ was varied in a graph with $|V| = 1k$ and $\mathcal{T} = 100k$. As shown in **Figure 7(a)–(b)**, T-SIRGN's runtime increases sub-linearly in $|V|$. Also, the combined effect of temporal edges is roughly linear, which is consistent with time complexity analysis. Remarkably, T-SIRGN handles 10M temporal edges in less than 3 minutes: this confirms its high efficiency.

We also isolated the effect of $\mathcal{T}$, using graphs with a fixed number of nodes and edges ($|V| = 1k$ and $|\mathcal{E}| = 100k$). The number of timestamps was varied such that $\mathcal{T}$ increased while the number of temporal edges remained fixed. **Figure 7(c)**, shows the contribution of $\mathcal{T}$ to be linear, also consistent with our theoretical complexity.

5.4 Parameter analysis

We tested iterations (d), embedding size (h), and α on the Synth0.1 dataset, and plotted against runtime and accuracy (**Figure 8**). We focused on our T-SIRGN, and, for parameter h, on its closest competitors GCRN and CTGCN (in all variants). We did not involve any competitors for α as it is a parameter of T-SIRGN only. Also, manipulating d in the GNN-based competitors is not trivial, as it corresponds to the number of GNN layers (it is often hardcoded).

Figure 8-(a) shows T-SIRGN's runtime is linear in d (consistent with its time complexity), and maximal accuracy requires a small d.

The trend of T-SIRGN's runtime is roughly proportional to the square root of h (**Figure 8-(b)**). Accuracy is fairly stable, with the only exception of an expected (slight) decrease when h is very low.

Varying α (**Figure 8-(c)**) impacts accuracy, but not runtimes. A very small α leads to consistently lower accuracy, which complies with the fact that α close to zero leads to a version of T-SIRGN (SS-TSIRGN) that considers temporally-flattened information only.

6 CONCLUSION

This paper presents Temporal SIR-GN, a novel method for structural representation learning in temporal graphs, which overcomes efficiency and effectiveness limitations of existing methods. Temporal SIR-GN performance are attested both theoretically and experimentally, by an extensive evaluation on synthetic and real data.

Future work includes deriving further theoretical properties, experimenting with the settings in **Section 3.5** and with more tasks/applications, and investigating how to handle our target temporal structural patterns with temporal-GNN-based approaches.

ACKNOWLEDGMENTS

This research was funded by a National Centers of Academic Excellence in Cybersecurity grant (H98230-22-1-0300), which is part of the National Security Agency.

REFERENCES

[1] Alan A. Barrat, Ciro Cattuto, Jean-François Pinton, and Wouter Van den Broeck. 2008. SocioPatterns. http://www.sociopatterns.org/.

[2] Charu C. Aggarwal and Haixun Wang (Eds.). 2010. *Managing and Mining Graph Data*. Advances in Database Systems, Vol. 40. Springer.

[3] Nesreen K. Ahmed, Ryan A. Rossi, John Boaz Lee, Theodore L. Willke, Rong Zhou, Xiangnan Kong, and Hoda Eldardiry. 2022. Role-Based Graph Embeddings. *IEEE Transactions on Knowledge and Data Engineering (TKDE)* 34, 5 (2022), 2401–2415.

[4] Vladimir Batagelj and Matjaz Zaversnik. 2011. Fast algorithms for determining (generalized) core groups in social networks. *Advances in Data Analysis and Classification (ADAC)* 5, 2 (2011), 129–145.

[5] Mikhail Belkin and Partha Niyogi. 2001. Laplacian Eigenmaps and Spectral Techniques for Embedding and Clustering. In *Proc. of Conf. on Advances in Neural Information Processing Systems (NIPS)*. 585–591.

[6] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. 2014. Spectral Networks and Locally Connected Networks on Graphs. In *Proc. of Int. Conf. on Learning Representations (ICLR)*.

[7] Jaewook Byun, Sungpil Woo, and Daeyoung Kim. 2020. ChronoGraph: Enabling temporal graph traversals for efficient information diffusion analysis over time. In *Proc. of IEEE Int. Conf. on Data Engineering (ICDE)*. 2026–2027.

[8] Hongyun Cai, Vincent W. Zheng, and Kevin Chen-Chuan Chang. 2018. A Comprehensive Survey of Graph Embedding: Problems, Techniques, and Applications. *IEEE Transactions on Knowledge and Data Engineering (TKDE)* 30, 9 (2018), 1616–1637.

[9] Shaosheng Cao, Wei Lu, and Qiongkai Xu. 2015. GraRep: Learning Graph Representations with Global Structural Information. In *Proc. of Int. Conf. on Information and Knowledge Management (CIKM)*. ACM, 891–900.

[10] Ines Chami, Sami Abu-El-Haija, Bryan Perozzi, Christopher Ré, and Kevin Murphy. 2020. Machine Learning on Graphs: A Model and Comprehensive Taxonomy. *CoRR* abs/2005.03675 (2020).

[11] Deli Chen, Yankai Lin, Wei Li, Peng Li, Jie Zhou, and Xu Sun. 2020. Measuring and Relieving the Over-Smoothing Problem for Graph Neural Networks from the Topological View. In *Proc. of AAAI Conf. on Artificial Intelligence (AAAI)*. 3438–3445.

[12] Jinyin Chen, Xueke Wang, and Xuanheng Xu. 2022. GC-LSTM: graph convolution embedded LSTM for dynamic network link prediction. *Applied Intelligence (APIN)* 52, 7 (2022), 7513–7528.

[13] Michele Coscia. 2021. The Atlas for the Aspiring Network Scientist. *CoRR* abs/2101.00863 (2021).

[14] Claire Donnat, Marinka Zitnik, David Hallac, and Jure Leskovec. 2018. Learning Structural Node Embeddings via Diffusion Wavelets. In *Proc. of ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining (KDD)*, Yike Guo and Faisal Farooq (Eds.). 1320–1329.

[15] Lun Du, Yun Wang, Guojie Song, Zhicong Lu, and Junshan Wang. 2018. Dynamic Network Embedding : An Extended Approach for Skip-gram based Network Embedding. In *Proc. of Int. Joint Conf. on Artificial Intelligence (IJCAI)*. 2086–2092.

[16] Nan Du, Hanjun Dai, Rakshit Trivedi, Utkarsh Upadhyay, Manuel Gomez-Rodriguez, and Le Song. 2016. Recurrent Marked Temporal Point Processes: Embedding Event History to Vector. In *Proc. of ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining (KDD)*. 1555–1564.

[17] Julie Fournet and Alain Barrat. 2014. Contact Patterns among High School Students. *PLOS ONE* 9, 9 (2014), 1–17.

[18] Alan M. Frieze, Aristides Gionis, and Charalampos E. Tsourakakis. 2013. Algorithmic techniques for modeling and mining large graphs (AMAzING). In *Proc. of ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining (KDD)*. 1523.

[19] Dongqi Fu and Jingrui He. 2021. DPPIN: A Biological Dataset of Dynamic Protein-Protein Interaction Networks. *CoRR* abs/2107.02168 (2021).

[20] Swapnil Gandhi and Yogesh Simmhan. 2020. An Interval-centric Model for Distributed Computing over Temporal Graphs. In *Proc. of IEEE Int. Conf. on Data Engineering (ICDE)*. 1129–1140.

[21] Rishab Goel, Seyed Mehran Kazemi, Marcus A. Brubaker, and Pascal Poupart. 2020. Diachronic Embedding for Temporal Knowledge Graph Completion. In *Proc. of AAAI Conf. on Artificial Intelligence (AAAI)*. 3988–3995.

[22] Palash Goyal, Nitin Kamra, Xinran He, and Yan Liu. 2018. DynGEM: Deep Embedding Method for Dynamic Graphs. *CoRR* abs/1805.11273 (2018).

[23] Martin Grohe and Daniel Neuen. 2021. Recent advances on the graph isomorphism problem. In *Surveys in Combinatorics*. 187–234.

[24] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable Feature Learning for Networks. In *Proc. of ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining (KDD)*. 855–864.

[25] William L. Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive Representation Learning on Large Graphs. In *Proc. of Conf. on Advances in Neural Information Processing Systems (NIPS)*. 1024–1034.

[26] Ningyuan Teresa Huang and Soledad Villar. 2021. A Short Tutorial on The Weisfeiler-Lehman Test And Its Variants. In *Proc. IEEE Int. Conf. on Acoustics, Speech and Signal Processing (ICASSP)*. 8533–8537.

[27] Shixun Huang, Zhifeng Bao, Guoliang Li, Yanghao Zhou, and J. Shane Culpepper. 2020. Temporal Network Representation Learning via Historical Neighborhoods Aggregation. In *Proc. of IEEE Int. Conf. on Data Engineering (ICDE)*. 1117–1128.

[28] Junchen Jin, Mark Heimann, Di Jin, and Danai Koutra. 2022. Toward Understanding and Evaluating Structural Node Embeddings. *ACM Transactions on Knowledge Discovery from Data (TKDD)* 16, 3 (2022), 58:1–58:32.

[29] Wei Jin, Yao Ma, Yiqi Wang, Xiaorui Liu, Jiliang Tang, Yukuo Cen, Jiezhong Qiu, Jie Tang, Chuan Shi, Yanfang Ye, Jiawei Zhang, and Philip S. Yu. 2021. Graph Representation Learning: Foundations, Methods, Applications and Systems. In *Proc. of ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining (KDD)*. 4044–4045.

[30] Mikel Joaristi and Edoardo Serra. 2021. SIR-GN: A Fast Structural Iterative Representation Learning Approach For Graph Nodes. https://github.com/mjoaristi/SIR-GN. *ACM Transactions on Knowledge Discovery from Data (TKDD)* 15, 6 (2021), 100:1–100:39.

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

[32] Thomas N. Kipf and Max Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *Proc. of Int. Conf. on Learning Representations (ICLR)*.

[33] Danai Koutra and Christos Faloutsos. 2017. *Individual and Collective Graph Mining: Principles, Algorithms, and Applications*. Morgan & Claypool Publishers.

[34] Srijan Kumar, Bryan Hooi, Disha Makhija, Mohit Kumar, Christos Faloutsos, and VS Subrahmanian. 2018. Rev2: Fraudulent user prediction in rating platforms. In *Proc. of Int. Conf. on Web Search and Data Mining (WSDM)*. 333–341.

[35] Srijan Kumar, Francesca Spezzano, VS Subrahmanian, and Christos Faloutsos. 2016. Edge weight prediction in weighted signed networks. In *Proc. IEEE ICDM Conf.* 221–230.

[36] Srijan Kumar, Xikun Zhang, and Jure Leskovec. 2019. Predicting Dynamic Embedding Trajectory in Temporal Interaction Networks. In *Proc. of ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining (KDD)*. 1269–1278.

[37] Jure Leskovec and Andrej Krevl. 2014. SNAP Datasets: Stanford Large Network Dataset Collection. http://snap.stanford.edu/data.

[38] Yiming Li, Yanyan Shen, Lei Chen, and Mingxuan Yuan. 2023. Zebra: When Temporal Graph Neural Networks Meet Temporal Personalized PageRank. *Proc. of the VLDB Endowment (PVLDB)* 16, 6 (2023), 1332–1345.

[39] Yaguang Li, Rose Yu, Cyrus Shahabi, and Yan Liu. 2018. Diffusion Convolutional Recurrent Neural Network: Data-Driven Traffic Forecasting. In *Proc. of Int. Conf. on Learning Representations (ICLR)*.

[40] Shangsong Liang, Shaowei Tang, Zaiqiao Meng, and Qiang Zhang. 2021. Cross-Temporal Snapshot Alignment for Dynamic Networks. *IEEE Transactions on Knowledge and Data Engineering (TKDE)* (2021).

[41] Jingxin Liu, Chang Xu, Chang Yin, Weiqiang Wu, and You Song. 2020. K-Core based Temporal Graph Convolutional Network for Dynamic Graphs. https://github.com/jhljx/CTGCN. *IEEE Transactions on Knowledge and Data Engineering (TKDE)* (2020).

[42] Yuanfu Lu, Xiao Wang, Chuan Shi, Philip S. Yu, and Yanfang Ye. 2019. Temporal Network Embedding with Micro- and Macro-dynamics. In *Proc. of Int. Conf. on Information and Knowledge Management (CIKM)*. 469–478.

[43] Jing Ma, Qiuchen Zhang, Jian Lou, Li Xiong, and Joyce C. Ho. 2021. Temporal Network Embedding via Tensor Factorization. In *Proc. of Int. Conf. on Information and Knowledge Management (CIKM)*. 3313–3317.

[44] Yao Ma, Ziyi Guo, Zhaochun Ren, Jiliang Tang, and Dawei Yin. 2020. Streaming Graph Neural Networks. In *Proc. of Int. ACM SIGIR Conf. on Research and Development in Information Retrieval (SIGIR)*. 719–728.

[45] Sedigheh Mahdavi, Shima Khoshraftar, and Aijun An. 2018. dynnode2vec: Scalable Dynamic Network Embedding. In *Proc. of IEEE Int. Conf. on Big Data*. 3762–3765.

[46] Franco Manessi, Alessandro Rozza, and Mario Manzo. 2020. Dynamic graph convolutional networks. *Pattern Recognition* 97 (2020).

[47] Giang Hoang Nguyen, John Boaz Lee, Ryan A. Rossi, Nesreen K. Ahmed, Eunyee Koh, and Sungchul Kim. 2018. Dynamic Network Embeddings: From Random Walks to Temporal Random Walks. In *Proc. of IEEE Int. Conf. on Big Data*. 1085–1092.

[48] Giang Hoang Nguyen, John Boaz Lee, Ryan A. Rossi, Nesreen K. Ahmed, Eunyee Koh, and Sungchul Kim. 2018. Continuous-Time Dynamic Network Embeddings. In *Proc. of World Wide Web Conf. (WWW)*. 969–976.

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

[50] Aldo Pareja, Giacomo Domeniconi, Jie Chen, Tengfei Ma, Toyotaro Suzumura, Hiroki Kanezashi, Tim Kaler, Tao B. Schardl, and Charles E. Leiserson. 2020. EvolveGCN: Evolving Graph Convolutional Networks for Dynamic Graphs. In *Proc. of AAAI Conf. on Artificial Intelligence (AAAI)*. 5363–5370.

[51] Yulong Pei, Xin Du, Jianpeng Zhang, George Fletcher, and Mykola Pechenizkiy. 2020. struc2gauss: Structural role preserving network embedding via Gaussian embedding. *Data Mining and Knowledge Discovery (DAMI)* 34, 4 (2020), 1072–1103.

[52] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. DeepWalk: online learning of social representations. In *Proc. of ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining (KDD)*. 701–710.

[53] Zhenyu Qiu, Wenbin Hu, Jia Wu, Weiwei Liu, Bo Du, and Xiaohua Jia. 2020. Temporal Network Embedding with High-Order Nonlinear Information. In *Proc. of AAAI Conf. on Artificial Intelligence (AAAI)*. 5436–5443.

[54] Leonardo Filipe Rodrigues Ribeiro, Pedro H. P. Saverese, and Daniel R. Figueiredo. 2017. *struc2vec*: Learning Node Representations from Structural Identity. In *Proc. of ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining (KDD)*. 385–394.

[55] Emanuele Rossi, Ben Chamberlain, Fabrizio Frasca, Davide Eynard, Federico Monti, and Michael Bronstein. 2020. Temporal Graph Networks for Deep Learning on Dynamic Graphs. https://github.com/twitter-research/tgn. In *ICML 2020 Workshop on Graph Representation Learning*.

[56] Ryan A. Rossi, Nesreen K. Ahmed, Eunyee Koh, Sungchul Kim, Anup Rao, and Yasin Abbasi-Yadkori. 2020. A Structural Graph Representation Learning Framework. In *Proc. of Int. Conf. on Web Search and Data Mining (WSDM)*. 483–491.

[57] Ryan A. Rossi, Di Jin, Sungchul Kim, Nesreen K. Ahmed, Danai Koutra, and John Boaz Lee. 2020. On Proximity and Structural Role-based Embeddings in Networks: Misconceptions, Techniques, and Applications. *ACM Transactions on Knowledge Discovery from Data (TKDD)* 14, 5 (2020), 63:1–63:37.

[58] Ryan A. Rossi, Rong Zhou, and Nesreen K. Ahmed. 2020. Deep Inductive Graph Representation Learning. *IEEE Transactions on Knowledge and Data Engineering (TKDE)* 32, 3 (2020), 438–452.

[59] Sam T. Roweis and Lawrence K. Saul. 2000. Nonlinear Dimensionality Reduction by Locally Linear Embedding. *Science* 290, 5500 (2000), 2323–2326.

[60] Aravind Sankar, Yanhong Wu, Liang Gou, Wei Zhang, and Hao Yang. 2020. DySAT: Deep Neural Representation Learning on Dynamic Graphs via Self-Attention Networks. In *Proc. of Int. Conf. on Web Search and Data Mining (WSDM)*. 519–527.

[61] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. 2009. The Graph Neural Network Model. *IEEE Transactions on Neural Networks and Learning Systems (TNNLS)* 20, 1 (2009), 61–80.

[62] Youngjoo Seo, Michaël Defferrard, Pierre Vandergheynst, and Xavier Bresson. 2018. Structured Sequence Modeling with Graph Convolutional Recurrent Networks. In *Proc. of Int. Conf. on Neural Information Processing (ICONIP)*. 362–373.

[63] Blake Shaw and Tony Jebara. 2009. Structure preserving embedding. In *Proc. of Int. Conf. on Machine Learning (ICML)*. 937–944.

[64] Min Shi, Yu Huang, Xingquan Zhu, Yufei Tang, Yuan Zhuang, and Jianxun Liu. 2021. GAEN: Graph Attention Evolving Networks. In *Proc. of Int. Joint Conf. on Artificial Intelligence (IJCAI)*. 1541–1547.

[65] Uriel Singer, Ido Guy, and Kira Radinsky. 2019. Node Embedding over Temporal Graphs. In *Proc. of Int. Joint Conf. on Artificial Intelligence (IJCAI)*. 4605–4612.

[66] Michele Starnini, Charalampos E. Tsourakakis, Maryam Zamanipour, André Panisson, Walter Allasia, Marco Fornasiero, Laura Li Puma, Valeria Ricci, Silvia Ronchiadin, Angela Ugrinoska, Marco Varetto, and Dario Moncalvo. 2021. Smurf-Based Anti-money Laundering in Time-Evolving Transaction Networks. In *Proc. of Europ. Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML PKDD)*. 171–186.

[67] Yahui Sun, Shuai Ma, and Bin Cui. 2022. Hunting Temporal Bumps in Graphs with Dynamic Vertex Properties. In *Proc. of ACM Int. Conf. on Management of Data (SIGMOD)*. 874–888.

[68] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. 2015. LINE: Large-scale Information Network Embedding. In *Proc. of World Wide Web Conf. (WWW)*. 1067–1077.

[69] Mingyue Tang, Pan Li, and Carl Yang. 2022. Graph Auto-Encoder via Neighborhood Wasserstein Reconstruction. https://github.com/mtang724/NWR-GAE. In *Proc. of Int. Conf. on Learning Representations (ICLR)*.

[70] Joshua B. Tenenbaum, Vin de Silva, and John C. Langford. 2000. A Global Geometric Framework for Nonlinear Dimensionality Reduction. *Science* 290, 5500 (2000), 2319–2323.

[71] Rakshit Trivedi, Mehrdad Farajtabar, Prasenjeet Biswal, and Hongyuan Zha. 2019. DyRep: Learning Representations over Dynamic Graphs. In *Proc. of Int. Conf. on Learning Representations (ICLR)*.

[72] Ioanna Tsalouchidou, Francesco Bonchi, Gianmarco De Francisci Morales, and Ricardo Baeza-Yates. 2020. Scalable Dynamic Graph Summarization. *IEEE Transactions on Knowledge and Data Engineering (TKDE)* 32, 2 (2020), 360–373.

[73] Anton Tsitsulin, Marina Munkhoeva, Davide Mottin, Panagiotis Karras, Ivan V. Oseledets, and Emmanuel Müller. 2021. FREDE: Anytime Graph Embeddings. *Proc. of the VLDB Endowment (PVLDB)* 14, 6 (2021), 1102–1110.

[74] Philippe Vanhems, Alain Barrat, Ciro Cattuto, Jean-François Pinton, Nagham Khanafer, Corinne Régis, Byeul-a Kim, Brigitte Comte, and Nicolas Voirin. 2013. Estimating Potential Infection Transmission Routes in Hospital Wards Using Wearable Proximity Sensors. *PLOS ONE* 8, 9 (2013), 1–9.

[75] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2018. Graph Attention Networks. In *Proc. of Int. Conf. on Learning Representations (ICLR)*.

[76] Petar Velickovic, William Fedus, William L. Hamilton, Pietro Liò, Yoshua Bengio, and R. Devon Hjelm. 2019. Deep Graph Infomax. https://github.com/PetarV-/DGI. In *Proc. of Int. Conf. on Learning Representations (ICLR)*.

[77] Bimal Viswanath, Alan Mislove, Meeyoung Cha, and Krishna P. Gummadi. 2009. On the Evolution of User Interaction in Facebook. In *Proc. of ACM Workshop on Online Social Networks (WOSN)*. 37–42.

[78] Junshan Wang, Guojie Song, Yi Wu, and Liang Wang. 2020. Streaming Graph Neural Networks via Continual Learning. In *Proc. of Int. Conf. on Information and Knowledge Management (CIKM)*. 1515–1524.

[79] Xuhong Wang, Ding Lyu, Mengjian Li, Yang Xia, Qi Yang, Xinwen Wang, Xinguang Wang, Ping Cui, Yupu Yang, Bowen Sun, and Zhenyu Guo. 2021. APAN: Asynchronous Propagation Attention Network for Real-time Temporal Graph Embedding. In *Proc. of ACM Int. Conf. on Management of Data (SIGMOD)*. 2628–2638.

[80] Yanbang Wang, Yen-Yu Chang, Yunyu Liu, Jure Leskovec, and Pan Li. 2021. Inductive Representation Learning in Temporal Networks via Causal Anonymous Walks. In *Proc. of Int. Conf. on Learning Representations (ICLR)*.

[81] B. Weisfeiler and A. A. Lehman. 1968. The reduction of a graph to canonical form and the algebra which appears therein. *Nauchno-Technicheskaya Informatsia* 2, 9 (1968), 12–16.

[82] Huanhuan Wu, James Cheng, Silu Huang, Yiping Ke, Yi Lu, and Yanyan Xu. 2014. Path Problems in Temporal Graphs. *Proc. of the VLDB Endowment (PVLDB)* 7, 9 (2014), 721–732.

[83] Jun Wu, Jingrui He, and Jiejun Xu. 2019. DEMO-Net: Degree-specific Graph Neural Networks for Node and Graph Classification. In *Proc. of ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining (KDD)*. 406–415.

[84] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. 2021. A Comprehensive Survey on Graph Neural Networks. *IEEE Transactions on Neural Networks and Learning Systems (TNNLS)* 32, 1 (2021), 4–24.

[85] Wenwen Xia, Yuchen Li, Jianwei Tian, and Shenghong Li. 2021. Forecasting Interaction Order on Temporal Graphs. In *Proc. of ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining (KDD)*. 1884–1893.

[86] Chengjin Xu, Fenglong Su, and Jens Lehmann. 2021. Time-aware Graph Neural Network for Entity Alignment between Temporal Knowledge Graphs. In *Proc. of Conf. on Empirical Methods in Natural Language Processing (EMNLP)*. 8999–9010.

[87] Dongkuan Xu, Wei Cheng, Dongsheng Luo, Xiao Liu, and Xiang Zhang. 2019. Spatio-Temporal Attentive RNN for Node Classification in Temporal Attributed Graphs. In *Proc. of Int. Joint Conf. on Artificial Intelligence (IJCAI)*. 3947–3953.

[88] Da Xu, Chuanwei Ruan, Evren Körpeoglu, Sushant Kumar, and Kannan Achan. 2020. Inductive representation learning on temporal graphs. https://github.com/StatsDLMathsRecomSys/Inductive-representation-learning-on-temporal-graphs. In *Proc. of Int. Conf. on Learning Representations (ICLR)*.

[89] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. 2019. How Powerful are Graph Neural Networks?. In *Proc. of Int. Conf. on Learning Representations (ICLR)*.

[90] Yonghui Xu, Shengjie Sun, Huiguo Zhang, Chang'an Yi, Yuan Miao, Dong Yang, Xiaonan Meng, Yi Hu, Ke Wang, Huaqing Min, Hengjie Song, and Chuanyan Miao. 2022. Time-Aware Graph Embedding: A Temporal Smoothness and Task-Oriented Approach. *ACM Transactions on Knowledge Discovery from Data (TKDD)* 16, 3 (2022), 56:1–56:23.

[91] Cheng Yang, Maosong Sun, Zhiyuan Liu, and Cunchao Tu. 2017. Fast Network Embedding Enhancement via High Order Proximity Approximation. In *Proc. of Int. Joint Conf. on Artificial Intelligence (IJCAI)*. 3894–3900.

[92] Renchi Yang, Jieming Shi, Xiaokui Xiao, Yin Yang, Juncheng Liu, and Sourav S. Bhowmick. 2020. Scaling Attributed Network Embedding to Massive Graphs. *Proc. of the VLDB Endowment (PVLDB)* 14, 1 (2020), 37–49.

[93] Wenchao Yu, Wei Cheng, Charu C. Aggarwal, Haifeng Chen, and Wei Wang. 2017. Link Prediction with Spatial and Temporal Consistency in Dynamic Networks. In *Proc. of Int. Joint Conf. on Artificial Intelligence (IJCAI)*. 3343–3349.

[94] Wenchao Yu, Wei Cheng, Charu C. Aggarwal, Kai Zhang, Haifeng Chen, and Wei Wang. 2018. NetWalk: A Flexible Deep Embedding Approach for Anomaly Detection in Dynamic Networks. In *Proc. of ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining (KDD)*. 2672–2681.

[95] Ziwei Zhang, Peng Cui, Jian Pei, Xiao Wang, and Wenwu Zhu. 2018. TIMERS: Error-Bounded SVD Restart on Dynamic Networks. In *Proc. of AAAI Conf. on Artificial Intelligence (AAAI)*. 224–231.

[96] Ziwei Zhang, Peng Cui, Xiao Wang, Jian Pei, Xuanrong Yao, and Wenwu Zhu. 2018. Arbitrary-Order Proximity Preserved Network Embedding. In *Proc. of ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining (KDD)*. 2778–2786.

[97] Ziwei Zhang, Peng Cui, and Wenwu Zhu. 2022. Deep Learning on Graphs: A Survey. *IEEE Transactions on Knowledge and Data Engineering (TKDE)* 34, 1

(2022), 249–270.

[98] Hongkuan Zhou, Da Zheng, Israt Nisa, Vasileios Ioannidis, Xiang Song, and George Karypis. 2022. TGL: A General Framework for Temporal GNN Training on Billion-Scale Graphs. *Proc. of the VLDB Endowment (PVLDB)* 15, 8 (2022), 1572–1580.

[99] Le-kui Zhou, Yang Yang, Xiang Ren, Fei Wu, and Yueting Zhuang. 2018. Dynamic Network Embedding by Modeling Triadic Closure Process. In *Proc. of AAAI Conf. on Artificial Intelligence (AAAI)*. 571–578.

[100] Dingyuan Zhu, Peng Cui, Ziwei Zhang, Jian Pei, and Wenwu Zhu. 2018. High-Order Proximity Preserved Embedding for Dynamic Networks. *IEEE Transactions on Knowledge and Data Engineering (TKDE)* 30, 11 (2018), 2134–2144.

[101] Linhong Zhu, Dong Guo, Junming Yin, Greg Ver Steeg, and Aram Galstyan. 2016. Scalable Temporal Latent Space Inference for Link Prediction in Dynamic Social Networks. *IEEE Transactions on Knowledge and Data Engineering (TKDE)* 28, 10 (2016), 2765–2777.

[102] Yuan Zuo, Guannan Liu, Hao Lin, Jia Guo, Xiaoqian Hu, and Junjie Wu. 2018. Embedding Temporal Network via Neighborhood Formation. In *Proc. of ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining (KDD)*. 2857–2866.