Proceedings of the Twentieth International AAAI Conference on Web and Social Media (ICWSM 2026)

Unsupervised Detection of Persistent Communities in Dynamic Networks with Network Embeddings

Ferdinand Le Coz[1], **Guillaume Cabanac**[1], **Julien Figeac**[2]
[1]University of Toulouse, IRIT UMR 5505, Toulouse, France
[2]Sciences Po Toulouse, UTOPI UMR 5133, Toulouse, France
ferdinand.lecoz@irit.fr, guillaume.cabanac@univ-tlse3.fr, julien.figeac@cnrs.fr

Abstract

Many real-world phenomena can be modeled using dynamic networks organized according to an evolving community structure. While traditional community discovery algorithms rely on structural information, recent years have seen a shift in focus towards methods of embedding nodes in dynamic graphs that incorporate temporal information through timestamps and contextual information through features. This endeavour is particularly evident in Graph Neural Networks (GNNs). In this paper, we address the importance of node embeddings and features in understanding network dynamics. First, we propose a general framework for extracting dynamic communities from graph embeddings without prior knowledge of these communities. We then design a synthetic dynamic graph generator that produces network data and features for GNN training. Experiments on synthetic dynamic networks demonstrate that informative features are essential for ensuring the performance of GNNs. By comparing several embedding algorithms on different scenarios, we demonstrate that our method effectively captures the network's underlying dynamics at both the macro and node levels. Furthermore, we demonstrate that GNN-based embeddings are more effective at capturing global network dynamics. They significantly outperform conventional embedding or community detection algorithms when it comes to detecting changes in dynamics at the node level. Additionally, we tested our method on a real-world dataset of social media interactions, achieving promising results for network visualization.

Introduction

Traditional methods for studying graph properties and detecting communities often assume a static structure and are therefore not well suited for analysing temporal graphs. As the collection of data with temporal information becomes easier (via social media APIs, news article databases, blogs, etc.) and new algorithms to directly encode temporal graphs emerge, there is a need to explore the possibility offered by these algorithms to uncover dynamic communities. The evolving structure of online communities is of great interest for studying phenomena such as opinion dynamics in social media (Perozzi, Al-Rfou, and Skiena 2014; Ramaciotti Morales et al. 2022) or the formation of academic research fields (Zhou et al. 2021; Kong et al. 2019).

Since the notion of community or node similarity itself is not well defined, a variety of algorithms used to study graphs implicitly or explicitly assume a notion of similarity between nodes, depending on their specific area of interest (Rossetti and Cazabet 2019). Some consider a graph with nodes as a document composed of words (Narayanan et al. 2017). Other methods use Singular Value Decomposition or Graph Laplacian and instead consider message passing: the flow of information within a graph (Nadakuditi and Newman 2012; Belkin and Niyogi 2003).

In recent years, graph representation learning, and in particular a branch called Graph Neural Networks (GNNs), has received a great deal of attention. Algorithms emerging from this field offer the possibility of incorporating node and link features to produce embeddings at the level of nodes, links or the whole graph (Wu et al. 2021). For node embedding, they rely on a message passing mechanism to aggregate the information carried by these features in the neighborhood of nodes (Kipf and Welling 2017).

The wealth of algorithms capable of generating graph embeddings (Ju et al. 2024) motivated us to design a method to develop unsupervised dynamic community discovery on these representations independently of the algorithm used to produce them. To the best of our knowledge, this problem has been little studied using Graph Neural Networks in a dynamic setting. Communities in real-world dynamic networks experience a continuous temporal evolution. We posit that the persistence of instances of similar communities is key to understanding the structure of dynamic networks. Indeed, considering communities at each timestep only obscures the intrinsic dynamic nature of real networks.

We start with an overview of existing work in section **Background and Related Work**. In section **Contributions** we first present our research questions and a method for tracking similar instances of communities using graph embeddings over different timesteps in subsection **Method: Dynamic Community Discovery**. In order to evaluate the proposed method, we describe a procedure to generate synthetic temporal graphs with known community dynamics and given scenarios to act as ground-truth for validation in **Synthetic Dynamic Networks**. Experiments are performed in the **Experiments on Synthetic Data** to investigate our research question. Finally, we test our method on data coming from X/Twitter in **Experiments on Real-World Data**.

Background and Related Work

Dynamic Community Discovery

There is a rich literature on the problem of finding communities in static graphs. Each method focuses on one of the many definitions of what a "good community" might be (Yang and Leskovec 2015). The most common algorithms are Louvain (Blondel et al. 2008) or Infomap (Rosvall and Bergstrom 2008). They consider respectively modularity (i.e., the intra community vs. the extra community connectivity) or the flow of information (i.e., the cost of moving in the graph). They both have the advantage of not requiring any parameters. Other methods assume a Stochastic Block Model Stucture (SBM) (Holland, Laskey, and Leinhardt 1983) to find communities as Decelle et al. (2011). Finally some clustering algorithms rely on the use of graph spectra (Nadakuditi and Newman 2012).

On dynamic graphs, it is customary to consider that the temporal evolution of the network plays a crucial role in the definition of the communities themselves, since the community associated to a node can vary over time. (Rossetti and Cazabet 2019) give the following definition for the task of Dynamic Community Discovery:
Given a dynamic network $\mathcal{G}$, a Dynamic Community D is defined as a set of distinct (node, periods) pairs: $D = (v_1, P_1), (v_2, P_2), ..., (v_n, P_n)$, with $P_n = ((t_{s0}, t_{e0}), (t_{s1}, t_{e1})...(t_{sN}, t_{eN}))$, with $t_{s*} \leq t_{e*}$. Dynamic Community Discovery aims to identify the set D of all the dynamic communities in $\mathcal{G}$. The partitions described by D can be neat as well as overlapping.

Static clustering algorithms are often used for dynamic networks. They are run on several snapshots of the dynamic network, and then a technique is applied to attribute persistent labels to the nodes. Greene, Doyle, and Cunningham (2010) offers a procedure for producing such persistent labels. The procedure consists of associating pairs of communities across the timesteps. This is done by considering cluster $C_{i,t}$ at time t and its most similar counterpart cluster $C_{j,t+1}$ at time $t + 1$ using Jaccard similarity score (Palla, Barabasi, and Vicsek 2007). There are several ways of matching or detecting dynamic communities using Instant-optimal, Temporal trade-off or Cross-Time strategies (Rossetti and Cazabet 2019). In Falkowski and Spiliopoulou (2007) they use a three-step procedure to perform a multi-step matching on communities found at each timestep to find instances of similar communities. It takes advantage of the global temporality of the network as well as the instantaneous structure of a snapshot. This method is defined as follows:

1. In the first step, communities are found in each snapshot using a static community detection algorithm.
2. Then, communities at each timesteps are linked based on a chosen similarity measure (Typically the Jaccard coefficient). This creates a community Survival Graph.
3. Finally, a community detection algorithm is run on the community Survival Graph. This produces persistent communities across several snapshots.

Graph Embedding and Community Detection

These last few years, a lot of academic effort has been concentrated on the problem of static network embedding. A variety of algorithms have been developed to produce a vector representation for each node of the network so that the resulting embedding of the graph encapsulates its structural properties. The embedding can be achieved through Single Value Decomposition (Zhu et al. 2018), Random Walks (Perozzi, Al-Rfou, and Skiena 2014; Grover and Leskovec 2016), Graph Spectral Theory (Belkin and Niyogi 2003) or through Deep learning algorithms as Veličković et al. (2018). However, the availability of graph-structured data and breakthroughs in the field of Graph Neural Networks increased the attention on dynamic network embedding. These algorithms can be classified according to whether they consider time in a continuous or discrete manner and put more emphasis on the structural or temporal aspect (Xue et al. 2022). Some of these methods are relying on traditional approaches modified for the temporal setting like Dynnode2vec (Mahdavi, Khoshraftar, and An 2018) or Continuous-Time Dynamic Network Embedding (Nguyen et al. 2018). The Deep Learning oriented ones uses Recurrent Neural Network architecture like JODIE (Kumar, Zhang, and Leskovec 2019) or TGAT (Xu et al. 2020). They can also rely on methods that emerged from Graph Neural Networks such as TGN (Rossi et al. 2020) or EvolveGCN (Pareja et al. 2020).

A number of studies have addressed the issue of using graph embeddings for community detection such as Tandon et al. (2021) or Kojaku et al. (2024). They often rely on applying an Euclidean clustering algorithm on the produced embeddings. However, as stated by Longa et al. (2023), few works have considered the possibility of using GNNs designed for temporal graphs to perform unsupervised community detection.

Contributions

Research Questions

The purpose of our study is to investigate the use of graph embeddings for dynamic community discovery. It aims to validate their ability to detect changes in the community structure at both the macro and node level. In addition we want to report on the relative performance of static versus dynamic embedding algorithms. The experiment questions the ability of a method using embedding algorithms to:

1. Detect the fundamental events that characterise the evolution of dynamic communities (Palla, Barabasi, and Vicsek 2007): Birth, Death, Merging, Splitting, Expansion, Contraction.
2. Detect the change of community at the node level over the lifetime of the network.
3. Measure the impact of embedding algorithm properties on performance when:
   - Adding temporality to structural information,
   - Varying the informativeness of features in GNN training.

To answer these research questions, we design a method to provide persistent labels to nodes in a dynamic network. A synthetic dynamic network generator is proposed to test this method and investigate our research questions.

Method: Dynamic Community Discovery

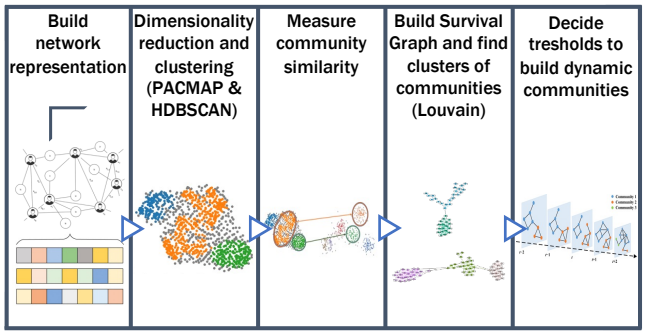


Figure 1. Workflow of the Dynamic Community Discovery Method

Graph embedding algorithms are able to encapsulate structural, and possibly temporal or contextual information relevant to the phenomena under scrutiny in a series of embeddings E_t with $t \in \{t_1, \ldots, t_n\}$. Our approach builds on this family of algorithms to discover dynamic communities (Figure 1). Drawing inspiration from (Falkowski and Spiliopoulou 2007), Graph Representation Learning and the use of Deep Learning embeddings for downstream tasks, we introduce a method for detecting relevant dynamic communities in social networks by leveraging the power of graph embedding algorithms.

The goal is, given a temporal network represented by a dynamic graph $\mathcal{G}$ and a decomposition of the temporality in n timesteps $\{t_1, \ldots, t_n\}$ to identify a set of k dynamic communities $D = \{D_1, \ldots, D_k\}$ that are present in the network across several timesteps.

We consider a dynamic network $\mathcal{G}$, which can be represented either as a continuous-time graph or a discrete-time graph (Xue et al. 2022). This graph, along with the relevant information, is then processed into a series of embeddings E_t with $t \in \{t_1, t_n\}$ across the n timesteps suiting the phenomenon under scrutiny. The first option is to use a continuous-time graph embedding algorithm (typically a RNN-based Neural Network) and then producing the embeddings at each $t \in \{t_1, \ldots, t_n\}$ (by feeding the data up to time t to the trained neural network for instance). The second considers the graph $\mathcal{G}$ as a series of timestep graphs $\{G_{t_1}, \ldots, G_{t_n}\}$ representing the evolving network at successive snapshots. Then, a static graph embedding algorithm is applied at each timestep t or a discrete-time dynamic embedding algorithm produces the embeddings.

We apply a clustering algorithm to each step of the series of graph embeddings E_t with $t \in \{t_1, ..., t_n\}$, to find all the k_t static communities $C_t^1, ..., C_t^{k_t}$ at time t. A graph which we call **Survival Graph** is initialized with all these communities as its nodes.

Algorithm n. 1: Persistent Community Detection Based on Embeddings.

Given a temporal graph $\mathcal{G}$, a set of timestep $\{t_1, ..., t_n\}$, a period threshold $\tau_{periods}$ and a similarity threshold $\tau_{similarity}$:

1. Apply the embedding algorithm to $\mathcal{G}$ to produce the series of embeddings $E = \{E_{t_1}, ..., E_{t_n}\}$.
2. Apply the clustering algorithm to each embedding E_t to produce the series of communities $C = \{C_{t_1}, ..., C_{t_n}\}$.
3. For all $\mathcal{C}_{t_1}^i, \mathcal{C}_{t_2}^j \in C$ so that $t_1 \neq t_2$, if $Similarity(\mathcal{C}_{t_1}^i, \mathcal{C}_{t_2}^j) \geq \tau_{similarity}$ and $|t_1 - t_2| \leq \tau_{timestep}$ then add a link between $\mathcal{C}_{t_1}^i$ and $\mathcal{C}_{t_2}^j$ in the Survival Graph.
4. Apply the static community detection on the Survival Graph. The "Super Communities" obtained this way are the Dynamic Communities $D = \{D_1, ..., D_k\}$.
5. Finally, each node is labeled to match its community at time t to the Dynamic Community D_t it belongs to at that timestep.

Following the clustering phase, we consider two user-defined parameters to find persistent instances of communities:

- $\tau_{timestep}$ controls the temporal window in which it is relevant for two clusters to be considered as similar with respect to the temporality of the network.
- $\tau_{similarity}$ is the threshold above which we link two clusters of nodes across time.

To track similar communities over time, we then define a commonly accepted measure of similarity between clusters. Given two communities $\mathcal{C}_{t_1}^i$ and $\mathcal{C}_{t_2}^j$ detected by the clustering algorithm at times t_1 and t_2:

$$Similarity\left(\mathcal{C}_{t_1}^i, \mathcal{C}_{t_2}^j\right) = \begin{cases} \frac{|\mathcal{C}_{t_1}^i \cap \mathcal{C}_{t_2}^j|}{\min(|\mathcal{C}_{t_1}^i|, |\mathcal{C}_{t_2}^j|)} & \text{if } |t_1 - t_2| \leq \tau_{timestep} \\ 0 & \text{otherwise} \end{cases}$$

If $Similarity(\mathcal{C}_{t_1}^i, \mathcal{C}_{t_2}^j) \geq \tau_{similarity}$, then a link is created between the two communities in the **Survival Graph** with a weight equals to the value of the similarity. A static community detection algorithm is run on this **Survival Graph** to find new "Super Communities" which are the persistent instances of similar clusters detected on the embeddings at each timestep. These k communities of the **Survival Graph** found in this way are the dynamic communities $D = \{D_1, ..., D_k\}$.
Finally at each timestep t, we assign to each node the label corresponding to the dynamic community D_i to which it belongs at this timestep. At each timestep and for each node, a single label is assigned, since each step community $\mathcal{C}_t^i$ can be associated to one and only one of the dynamic communities in D.

A representation of the pipeline can be found in Figure 1

and the pseudocode algorithm in Algorithm n. 1.

Synthetic Dynamic Networks

No comprehensive benchmark test the evolution of node community over time based on ground truth and produces data used for GNN training: interactions labeled with timestamps. Hence, we propose to generate a dynamic graph with:

1. A controllable community structure.
2. A dynamic interaction process within the network with control over temporality.
3. The possibility for nodes to change community over time.
4. Simulation scenarios such as merging or splitting of communities.
5. Options to generate features for nodes or links according to the community structure.

We drew inspiration from the code and motivating example proposed by Goyal, Chhetri, and Canedo (2020) who use a Stochastic Block Model (Holland, Laskey, and Leinhardt 1983) to generate communities. This relies on a block matrix B of size $N \times N$ determining the probability of an edge within or across communities, with N the number of communities in the network. We adapted this for our purposes: setting the probability Matrix B (that can evolve over time) to control the likelihood of interactions between nodes based on their respective communities. Since every interaction in the network is directed, the resulting temporal graph is also directed. Given a number of nodes n_{nodes}, an initial number of communities N, and the probability matrix B, the network will evolve over a user-defined number of timesteps n_{ts}, creating new interactions based on the underlying SBM structure.

The number of nodes to be perturbed at each timestep $n_{perturb}$ as well as the communities to be perturbed can be specified. These 2 parameters drive the evolution of some nodes that will migrate from their original communities to an other. A node undergoing a community change is called a *migrating node*, and then a *perturbed node* once the migration is over

Finally, the generator can be provided with the following parameters: the timesteps $\{t^1_{event}, ..., t^l_{event}\}$ at which a splitting or merging of communities occurs as well as the proportions of each community to be merged or split, a model for the temporal simulation, and parameters for the generation of features.

The workflow of the Synthetic Dynamic Network generator is summarized in Algorithm n. 2.

Algorithm n. 2: Synthetic Dynamic Network Simulation

Initialization

Given B, N, n_{nodes}: An initial graph is created. Random possible interactions between nodes are sampled and they are added to the graph with a probability given by B. If a temporal model is provided, then a timestamp associated with each event is sampled according to the model. If not, the associated timestep is the order of occurrence.

At each Step $t \in \{1, \dots, n_{ts}\}$

If $t \in \{t^1_{event}, \dots, t^l_{event}\}$, **then** the matrix B is changed according to the event and a new graph is sampled through a procedure specific to the event. The *migrating* and *perturbed nodes* are kept.

Otherwise:

1. The *migrating nodes* from the previous timestep are integrated into their new community and become *perturbed nodes*. $n_{perturb}$ new *migrating nodes* are selected from the perturbed communities.
2. These new *migrating nodes* see their probability of interaction with the community they are about to join increase, but still remain lower than if they were already members of this community.
3. A new graph is sampled: each pair of nodes will interact with a probability that depends on their respective communities through B. Timestamps are associated with each interaction as during the initialization.

Feature Generation

GNNs require features for training; to evaluate their impact on the produced embeddings, informative features must be generated alongside the synthetic dynamic graph. Textual information often plays a crucial role for social scientists in assessing the position and behavior of entities within a social network. Furthermore, recent work suggests that GNNs are suited and designed to use Bag of Words style features ,(Purchase, Zhao, and Mullins 2023), the decision was therefore taken to employ this formalism.

A Bag of Words (BoW) vector is a representation of a text document as a vector of word frequencies, based on a predefined vocabulary. Each position in the vector corresponds to a word in the vocabulary, and the value at that position indicates how many times the word appears in the document.

Our model generates BoW features based on the underlying community structure:

At timestep t, the size of a generated feature is divided into N_t blocks of equal size, one block per community present at timestep t. If a node belongs to community i, then the values of its i-th block of features are sampled according to a binomial law with parameter p_{in}. The other components are sampled according to a binomial law with parameter p_{out}. In this way, the generated feature reflects the community to which the node belongs when it starts an interaction. Furthermore, the informativeness of the features can be controlled by tuning the parameters p_{in} and p_{out}. By analogy with textual data, this feature generation accounts for the variety of vo-

cabularies used within different communities.

Experiments on Synthetic Data

Experimental Setup

We chose to compare the proposed method using the embeddings produced by different algorithms. As a static embedding algorithm, we use **Random Walks** (Grover and Leskovec 2016). The non-GNN dynamic embedding algorithms tested are **Dynnode2vec** (Mahdavi, Khoshraftar, and An 2018) and the **Continuous Time Dynamic Network Embedding** method (referred here as **CTDNE**) (Nguyen et al. 2018). We trained 3 GNN algorithms. One is static and named **Graph Attention Network** (namely **GAT**) (Veličković et al. 2018) and the other two are dynamic : Temporal Graph Network **Temporal Graph Network** (namely **TGN**) (Rossi et al. 2020) and **Evolve GCN-O** (namely **evolveGCN**) (Pareja et al. 2020). The proposed method is compared with two conventional dynamic community discovery methods based on modularity: the original method proposed by (Falkowski and Spiliopoulou 2007) (referred here as **FS**) and the algorithm proposed by (Greene, Doyle, and Cunningham 2010) (referred here as **GDC**)

A dimensionality reduction algorithm is applied to all the embeddings produced, reducing their dimension to 2. More information on the choice of dimensionality reduction can be found in **Appendix II) Impact of Dimensionality**. In the 4 forthcoming experiments, we used PACMAP (Wang et al. 2021) with its default parameters i.e. *n_neighbors* = 10, *MN_ratio* = 0.5 and *FP_ratio* = 2.0.
HDBSCAN (McInnes, Healy, and Astels 2017) was the clustering algorithm used in all the experiments with the following parameters: *min_cluster_size* = 50, *min_samples* = 10, *cluster_epsilon_selection* = 0.1, *method* = leaf. A parameter sensitivity analysis of the HDBSCAN algorithm and its impact on the proposed method can be found in **Appendix I) Parameter Sensitivity Analysis of the Clustering Algorithm**.
Finally, we chose $\tau_{timestep} = 2$, $\tau_{similarity} = 0.15$. We kept detected communities that existed over at least min_instances = 2 timesteps.

Let us report on the 3 scenarios we explored with the simulated networks:

1. A *fixed* community scenario: $n_{perturb}$ nodes change of community at each timestep.
2. A *merging* community scenario: two communities merge into a single one.
3. A *new* community scenario: a fraction of the nodes belonging to the initial communities form a new community.

We ran the dynamic network simulation process for all scenarios with a Block Model Matrix B set to have an intracommunity interaction probability of 0.1 and an intercommunity interaction probability of 0.01. The initial number of communities was N = 3 with a number of nodes n_{nodes} = 500. The simulations were run for n_{ts} = 25 steps with $n_{perturb}$ = 5 nodes migrating from community 1 to any other randomly selected community at each timestep. Both events (merging and new community) start from the second timestep onwards. In the *merging* scenario, all nodes from communities 0 and 2 merge into a new one. The *new* community is set to be created with 100 nodes, each community (0, 1 and 2) contributing for $\frac{1}{3}$ of the nodes. The resulting simulations for the *fixed*, the *merging*, and the *new* community scenarios are comprised of, respectively, 258,551, 412,703, and 217,542 interactions over 25 steps among 500 nodes distributed over, respectively, 3, 2 (after timestep 2) and 4 (after timestep 2) communities.

Code and data (including the generated graphs and features) needed to reproduce the experiments of this section are available at https://github.com/ferdinandlc/Long_clustering.

Graph Neural Network Training and Impact of Features

We benchmarked three Graph Neural Networks: TGN, GAT and evolveGCN. They learn a graph representation by being trained on structural data and features. The models are trained on 4 different sets of generated features. The results obtained assess the impact of the informativeness of the features. Based on these results, we also selected the models to compare with other embedding algorithms.

The TGN models are trained in an unsupervised manner using node features as well as interaction timestamps and features. Evolve-GCN models are trained in a semi-supervised manner using node features and interaction timestamps. GAT on the other hand is a static embedding algorithm trained in a semi-supervised manner using node features, thus we train a model for each graph snapshot G_t. For GAT and evolveGCN, we provide labels for 50 randomly selected nodes to use them as train subjects mimicking the manual labeling of a part of the network.

To study the influence of the informativeness of the features on the embedding produced, we generate 4 sets of features:

- Set 1: Feature vectors of size 64 with all components equal to 1.
- Set 2: Feature vectors of size 64 generated according to the procedure described previously with p_{in} = 0.4 and p_{out} = 0.3
- Set 3: Feature vectors of size 64 generated according to the procedure described previously with p_{in} = 0.5 and p_{out} = 0.1
- Set 4: Feature vectors of size 64 generated according to the procedure described previously with p_{in} = 0.9 and p_{out} = 0.01

Each model is trained 10 times on each set of generated features and the best model according to the training metric, validation accuracy, is kept for comparison. The experiments are run on the New Community scenario to investigate how they perform on a complex dynamic case.

Figure 2 shows for TGN,GAT and evolveGCN the quality of the dynamic communities obtained through the method

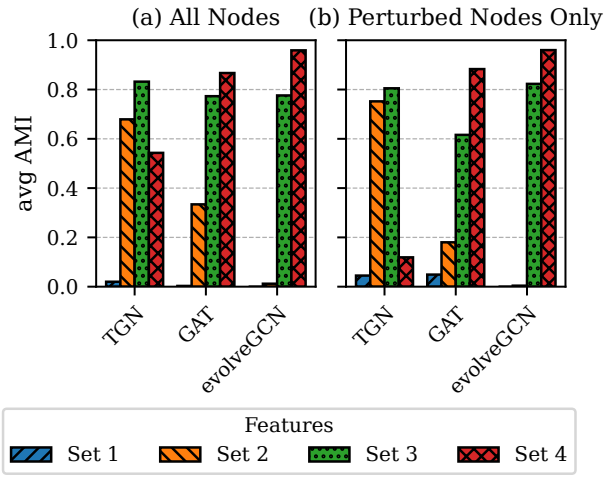


Figure 2. average AMI values of TGN, GAT and evolveGCN models trained on each Set of features across all nodes labels (a) and only on perturbed nodes (b)

described previously between 4 trained models. The metric used is the Adjusted Mutual Information between the ground truth communities and the ones discovered by our method. We observe a clear difference between the models trained with informative BoW features and the models trained with vectors of 1 as features (Set 1). While the latter seems to produce communities at random (AMI close or equal to 0), the TGN model is able to successfully represent the network community structure with Set 2. All models trained with the features of Set 3 and Set 4 effectively capture the underlying structure of the network. The accuracy of the GAT and evolveGCN model increase as the features become more informative about the communities. The models trained on Set 3 and Set 4 perform better in terms of AMI. For TGN, the features of Set 4 may cause overfitting, and Set 3 shows the best results.

The GAT and evolveGCN models perform better as the features become more discriminative. Set 1 describes each community as using the same vocabulary while Set 4 accounts for very distinct ones. For each algorithm, the models trained on Set 3 have an average number of detected communities closer to the true number of communities in the network (i.e. 4 after step 2) as shown in Table 1.

Algorithm	Set 1	Set 2	Set 3	Set 4
TGN	2.36	3.12	**3.48**	3.20
GAT	1.76	2.68	**3.12**	3.08
evolveGCN	1.00	1.80	**3.84**	3.68

Table 1. Average number of communities detected across timesteps for TGN, GAT and evolveGCN models trained on each Set of features

All algorithms display good results when trained on Set 3. Completely distinct vocabularies (i.e. set 4) is not a likely hypothesis for a real social network. Therefore, we keep the parameters of Set 3 to generate the features in the next experiments.

These results are consistent when tested only over the perturbed nodes as shown in subfigure (b) of Figure 2 These metrics highlight that the method effectively classifies the nodes that change their communities over time independently of the macro event (i.e. the creation of a new community).

Results for the Proposed Scenarios

We trained the TGN, GAT and evolveGCN models with features generated using the parameters of set 3 described in the previous section. We focus on the average metrics, Adjusted Mutual Information (AMI) and Fowlkes-Mallows Index (FMI), over the 25 timesteps for all nodes on the *merging* community and *new* community scenarios. On the *fixed* community scenario, we further investigate the ability of this method and embedding algorithms to correctly classify nodes that change their community over the lifetime of the network.

Algorithm	avg AMI	avg FMI	avg NBCOM
TGN BoW	**0.832**	**0.903**	3.48
GAT BoW	0.773	0.827	3.12
evolveGCN BoW	0.776	0.852	**3.84**
dynnode2vec	0.793	0.857	3.68
CTDNE	0.768	0.831	3.64
RandomWalk	0.732	0.798	3.36
FS	0.756	0.826	3.00
GDC	0.764	0.833	3.08

Table 2. Average metrics across timesteps for the *new* community scenario

Algorithm	avg AMI	avg FMI	avg NBCOM
TGN BoW	**0.790**	0.885	2.60
GAT BoW	0.601	0.841	2.44
evolveGCN BoW	0.693	0.857	2.76
dynnode2vec	0.699	**0.903**	**2.00**
CTDNE	0.556	0.778	2.68
RandomWalk	0.585	0.817	2.52
FS	0.616	0.863	2.20
GDC	0.606	0.854	2.24

Table 3. Average metrics across timesteps for the *merging* community scenario

Table 2 and Table 3 show that TGN consistently produces better results with the proposed methods for the task of classifying the nodes in dynamic communities. However, it is not the most accurate embedding algorithm in terms of finding the true number of communities. In particular, we observe that the dynamic community discovery methods using modularity instead of embedding algorithms (i.e FS and GDC) fail to detect the emergence of a new community as shown in Table 2. It also appears that all algorithms struggle more with the *merging* community scenario. A possible explanation is that the initial structure makes it more difficult

to represent the two communities merging into one. On the other hand the proportions of each of the 3 communities that fusion into a new one quickly form a more coherent structure (in terms of cohesiveness) as the merging of 3 smaller sets of nodes.

Algorithm	avg AMI	avg FMI	avg NBCOM
TGN BoW	**0.942**	**0.973**	3.12
GAT BoW	0.843	0.907	**3.00**
evolveGCN BoW	0.887	0.939	3.12
dynnode2vec	0.831	0.906	**3.00**
CTDNE	0.795	0.877	**3.00**
RandomWalk	0.799	0.880	**3.00**
FS	0.826	0.898	**3.00**
GDC	0.826	898	**3.00**

Table 4. Average metrics across timesteps for the *fixed* community scenario

Table 4 shows that the TGN model outperforms the other algorithms when used with the proposed method in the *fixed* community scenario, followed by evolveGCN. This can be explained by the fact that perturbed nodes are better represented in the embeddings these two GNNs produces, and thus better classified, as shown in Table 5.

Algorithm	avg AMI	avg FMI
TGN BoW	0.805	0.867
GAT BoW	0.609	0.736
evolveGCN BoW	**0.886**	**0.911**
dynnode2vec	0.445	0.577
CTDNE	0.378	0.545
RandomWalk	0.351	0.535
FS	0.451	0.625
GDC	0.451	0.625

Table 5. Average metrics across timesteps for the *fixed* community scenario over perturbed nodes

From these observations, we conclude that our method is capable of classifying in an unsupervised manner the nodes over time, even with a complex scenario. The embeddings produced by TGN outperform the other algorithm in all scenarios in terms of average Adjusted Mutual Information. We observe that the proposed method used with GNN-based embedding algorithms and are consistently more accurate on the perturbed nodes. This suggests that informative features do have an impact on the quality of the embeddings produced, especially at the micro-level of the network dynamic.
The method also approximates the true number of communities. Dynnode2vec effectively represent the macro-structure of communities in all scenarios. Overall, the best performing models are the three GNNs and dynnode2vec. These are all dynamic embedding algorithms that take temporality into account to produce the embeddings. On the other hand, Modularity (with FS and GDC) performs consistently well and should be considered when applying the method without having to train models. A comparison of our method against a more intuitive strategy to match communities across timesteps can be found in **Appendix III) Comparison of our Method with Simple Matching Strategy**.

Parameter Sensitivity Analysis

A further sensitivity analysis is conducted. The Sobol method was executed with $N_{total} = 512\times(2\times3+2) = 4096$ model evaluations for each algorithm on each scenario, evaluated on Adjusted Mutual Information. 512 is the base sample size and 3 is the number of parameters i.e. $\tau_{similarity}$, $\tau_{timesteps}$ and min_instances. The mean First-Order Sobol index (S_1), Mean Total-Order Sobol Index (S_T) and the standard deviation of the rank of the parameter important across all test case (Rank StdDev) are reported in Table 6. The mean value and standard deviation of each parameter over the set of parameters achieving 95% of the best AMI is also reported in Table 7.

Parameter	mean S_1	mean S_T	Rank StdDev
$\tau_{similarity}$	**0.433**	**0.640**	0
$\tau_{timesteps}$	0.123	0.468	0.4
min_instances	0.107	0.240	0.4

Table 6. Parameter sensitivity of the proposed method averaged over all scenarios and algorithms

Parameter	mean value	value StdDev
$\tau_{similarity}$	0.482	0.273
$\tau_{timesteps}$	13	6.40
min_instances	13	6.28

Table 7. Mean value and standard deviation of each parameter for evaluations achieving 95% of the best AMI

We observe that the method exhibits stronger sensitivity to the parameter $tau_{similarity}$, followed by $\tau_{timesteps}$. From the mean and Standard Deviation of the sets of parameters achieving 95% of the best AMI, a wide range of parameters values appears to yield good results. 43% of the evaluations conducted achieve at least 95% of the best AMI value. The parameters used in this paper for the previous study offer a good trade-off, as the average difference between the results presented and the best possible AMI for each algorithm is **0.03** (in absolute AMI value).
Thus, we observe that the method can be used with a wide range of parameter values and achieve convincing performance. We chose here to test our method with loose parameters to account for having no information about the real community structure and dynamic. It still performs well on average with more conservative parameters (i.e. higher $\tau_{similarity}$). This is expected as the scenarios tested here do not exhibit both frequent and abrupt changes in the community structure.

Experiments on Real World Data

Dataset And Feature Preparation

For the second evaluation, we apply the previously proposed method and algorithms to a real network extracted from X/Twitter during the 2017 French presidential elections (available at https://zenodo.org/records/5535333, Fraisier et al. (2018)). This dataset features 22,854 accounts that were manually labeled. 6 different labels represent the political affiliations of the accounts.

We focus on the dynamic social network created by the retweets among 19,483 of the labeled accounts between the 1st of February and the 24th of April 2017. The resulting graph consists of 2,592,037 interactions among the labeled accounts during the core of the 2017 presidential campaign. Despite the static nature of the node labels, the objective is to assess the extent to which the proposed method and the embeddings produced by the algorithms previously studied are able to represent a real-world network of political debates.

To train the TGN, GAT and evolveGCN models, we used two different feature generation frameworks. The first of these relies on detecting a specific vocabulary for each party in the dataset. TF-IDF was run on the messages produced by the supporters of each party separately to identify the relevant terms used. We manually review the resulting vocabularies and remove any words that are meaningless or too specific. This results in a global vocabulary of 157 words (approximately 30 per party). Each message (or interaction) is represented by a vector of presence/absence of words within this vocabulary. The node feature of an account is the average of the Bag of Word feature vectors of all messages published by an account within the studied time period. On the other hand, we produce feature vectors using an LLM to represent the message. We passed the messages through a pre-trained French Language Model (Camembert, Martin et al. (2020)) and jointly reduced the dimension of the vectors to 64. The node feature associated with each account is the average of all its message embeddings within the timeframe.

Experimental Setup

The dimensionality reduction algorithm was applied to all produced embeddings to reduce their dimension to 2. In all experiments, we used PACMAP (Wang et al. 2021) with the following parameters: *n_neighbors* = 10, *MN_ratio* = 0.5 and *FP_ratio* = 2.0

The clustering algorithm that was used in all the experiments was HDBSCAN (McInnes, Healy, and Astels 2017) with the following parameters: *min_cluster_size* = 200, *min_samples* = 10, *cluster_epsilon_selection* = 0.2, *method* = leaf.

Finally, we chose $\tau_{timestep}$ = 2, $\tau_{similarity}$ = 0.2 and to keep only the Dynamic Community that existed over at least min_instances = 5 timesteps.

Results

The results on the X/Twitter dataset can be found in Table 8. We observe that TGN exhibit better performance than the other algorithms when trained with BoW features. On the other hand, it is a matter of interest to note that both GAT and evolveGCN embeddings perform poorly while the training metrics were satisfying (0.728 validation accuracy for evolveGCN and 0.715 for GAT). Modularity-based methods (FS and GDC) perform well. It can easily be understood as in this case as the network labels are static and a change of community for active users is unlikely over the course of a presidential election. The metrics used are also independent of the absolute value of the labels. However, Modularity-based methods produce a large number of communities, which can be partially reduced by the persistent label attribution leveraged by the FS method.

Algorithm	avg AMI	mean FMI	avg NBCOM
TGN BoW	**0.478**	**0.601**	10.66
TGN LLM	0.357	0.396	7.33
GAT BoW	0.204	0.342	**6.11**
GAT LLM	0.231	0.389	3.0
evolveGCN BoW	0.161	0.330	4.88
evolveGCN LLM	0.129	0.311	7.66
dynnode2vec	0.366	0.543	4.0
CTDNE	0.441	0.569	5.0
RandomWalk	0.437	0.536	4.66
FS	0.463	0.579	45.22
GDC	0.455	0.566	92.22

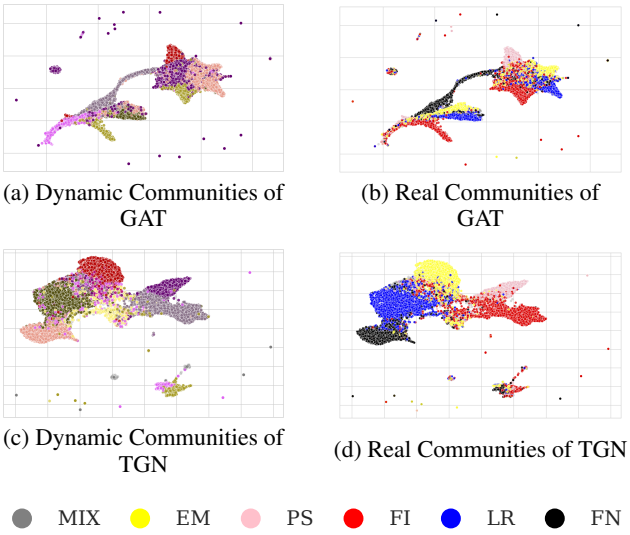
Table 8. Average metrics of the embedding algorithms on the X/Twitter dataset against the ground truth labels

Algorithm	SM-N ($10e^{-5}$)
TGN BoW	**3.89**
TGN LLM	2.81
GAT BoW	2.91
GAT LLM	3.63
evolveGCN BoW	3.38
evovleGCN LLM	2.70
dynnode2vec	3.63
CT	2.04
RandomWalk	2.23
FS	3.69
GDC	1.43

Table 9. SM-N scores for partition over the X/Twitter dataset

To investigate the stability of the labels at the node level, we further compute the SM-N of the partitions. This smoothness metric is defined as the inverse of the number of node membership changes (in absolute value) (Cazabet, Boudebza, and Rossetti 2021). The results are shown in Table 9. The most stable labels at the node level are obtained with TGN trained with BoW features, followed by FS, Dynnode2vec and GAT trained with LLM features.

Another interesting aspect of using node embeddings is that they can be used to provide a (time-evolving) representation of the users in a two-dimensional space. Figure 3 shows the user representation in the embedding space at the same given time during the campaign for TGN and GAT trained with BoW features. On the left side, nodes are colored according to their detected persistent community. On the right side, nodes are colored according to their ground-truth label (i.e. the party they stand for). We can see that the



(a) Dynamic Communities of GAT

(b) Real Communities of GAT

(c) Dynamic Communities of TGN

(d) Real Communities of TGN

Figure 3. Plots of the detected dynamic communities (left) vs the real communities (right) of TGN and GAT models trained with BoW features at time = 6 weeks. Underneath the figure are the color label for the real political communities affiliation (right side of the figure).

users are effectively represented in space in regard with their true party affiliation. The spatial representation of the parties also makes sense, especially for TGN embeddings. The different political parties on the left (Parti Socialiste (PS) in pink, France Insoumise (FI) in red) and on the right (Les Républicains (LR) in blue, Front National (FN) in black) are close to each other. We can also witness that the detected dynamic communities represented at this time during the campaign match with the real party affiliations.

Interestingly, the visualization and metrics provide mixed results. The average AMI values are quite low, and the proposed method only performs better than Modularity-based methods when TGN embeddings are used. The other GNN-based embeddings (GAT and evolveGCN) perform poorly. On the other hand, GNN-based models are able to represent in the structure of the network in a way that can be interpreted from a political perspective as shown in Figure 3. Overall, TGN appears as the most interesting embedding both for dynamic community discovery and visualization purposes in online social networks that exhibit an interaction modality similar to X/Twitter. This is also to be expected as the TGN model was initially created as a recommendation system for X/Twitter.

Discussion

This study was motivated by two broader questions. The first considers the relevance of graph embeddings. Given the wealth algorithms developed in recent years, we wanted to investigate the extent to which they produce "good" embeddings at the node level. By this we mean the ability of graph embedding algorithms to represent closely in the embedding space nodes that play a similar role in the network. The other question that derives from the first one is to examine their use in the study of social phenomena. These considerations must take into account the fact that these embedding algorithms were not designed to serve the interests of social sciences.

In answer to the first question, we demonstrated that GNN-based algorithms can achieve a higher average AMI than most of the other embedding algorithms tested on synthetic data. However, the cost and complexity of training a GNN must be considered. Traditional Modularity-based methods produce good results, are easier to use and allow for a much more direct interpretation of the communities they detect. Furthermore, the performance of GNNs is highly dependent on their training and the features provided. The ability of GNNs to leverage features still gives us some hope regarding the second question. In our experiments, we have seen that BoW features often outperform LLM features in terms of performance in our experiments. The visualizations they produce are easier to interpret in light of our knowledge of the networks. This paves the way for considering features produced by social scientists with a priori knowledge of the phenomenon under scrutiny. In this way, deep learning models can, in a sense, be biased by the knowledge of experts.

We also believe that the interpretation of results, especially visualization, should be done together with social scientist whose fields of research encompass the situation under investigation. Graph embeddings and other downstream tasks can then act as a first approach or an investigation tool to consider network-related research questions.

Limitations

We acknowledge that our study has several limitations. First, the comparison with other recent baselines for community detection such as the ones presented in (Li et al. 2023), (Sobolevsky and Belyi 2022) or (Guo, Wang, and Zhang 2014). As dynamic community discovery methods tend to assume certain paradigms and are designed for different use cases, there is no consensus on the current best practice. Therefore, we decided to focus on comparing our method with others that make similar design choices. This approach enables our study to investigate whether node embeddings can compete with structural methods, such as modularity, as the basis for a community detection algorithm..

Another limitation is that we had to design a specific synthetic network generation process. This was necessary to ensure that the same data and features could be used by several algorithms while , while also enabling GNNs to be trained on continuous-time events. However, this approach prevents comparison with commonly used benchmarks such as the LFR benchmark (Lancichinetti, Fortunato, and Radicchi 2008).

It would also be interesting to study how the embeddings produced become unstable over time. In particular, we

should consider its impact on the performance of dynamic community discovery.

Ultimately, this study has to be considered as a preliminary investigation into exploiting recent literature on graph embeddings in a dynamic setting. In the future, we hope to contribute to aligning some methods that emphasise the temporality of complex networks with the research interests of social scientists.

Conclusion

In this paper, we proposed a method for detecting and tracking dynamic communities from network embeddings. We also described a procedure to generate both dynamic networks with scenarios and embeddings that match the dynamic evolution of the network. The following conclusions were drawn from our study on synthetic data: GNN embeddings can recover the macro-structure of dynamic communities, producing results equivalent to those of established methods based on modularity. It has been demonstrated that certain GNN models consistently outperforms Modularity-based methods for unsupervised community detection. Furthermore, utilising GNN embeddings facilitates the effective detection of community change at the node level. This renders it a potentially valuable instrument for the identification of outlier behaviors. Our experiment highlighted the importance of features to produce informative embeddings when using GNN-based algorithms.

We also conducted an evaluation on real-world X/Twitter data. GNN-based algorithms such as GAT and evolveGCN struggle to represent the underlying community structure in the embeddings they produce. Despite the static nature of the communities in this case, the TGN model exhibited the best performance, achieving an average AMI of 0.478. The experiments also suggest the usefulness of graph embedding algorithms for network representation, especially as an initial approach when the nature of the network is unknow, or to identify nodes with unusual community affiliation behavior. Further study of the knowledge or hints that can be extracted from these representations would be interesting, for example in areas of the embedding space where communities overlap or when new communities emerge.

Our method proposed a way to use GNN embeddings for unsupervised dynamic community discovery, an area that has received little attention yet. It being independent of the underlying embedding algorithm also allows this method to eventually benefit from further breakthroughs in the field of Graph Representation.

References

Belkin, M.; and Niyogi, P. 2003. Laplacian Eigenmaps for Dimensionality Reduction and Data Representation. 1373–1396.

Blondel, V.; Guillaume, J.-L.; Lambiotte, R.; and Lefebvre, E. 2008. Fast Unfolding of Communities in Large Networks. *Journal of Statistical Mechanics Theory and Experiment*, 10: P10008.

Cazabet, R.; Boudebza, S.; and Rossetti, G. 2021. Evaluating community detection algorithms for progressively evolving graphs. *Journal of Complex Networks*, 8(6): cnaa027.

Decelle, A.; Krzakala, F.; Moore, C.; and Zdeborova, L. 2011. Asymptotic analysis of the stochastic block model for modular networks and its algorithmic applications. *Physical review. E, Statistical, nonlinear, and soft matter physics*, 84: 066106.

Falkowski, T.; and Spiliopoulou, M. 2007. Data Mining for Community Dynamics. *KI.*, 21(3): 23–29.

FORCE11. 2020. The FAIR Data principles. https://force11.org/info/the-fair-data-principles/.

Fraisier, O.; Cabanac, G.; Pitarch, Y.; Besançon, R.; and Boughanem, M. 2018. #Élysée2017fr: The 2017 French Presidential Campaign on Twitter. *International AAAI Conference on Web and Social Media*, 12(1).

Gebru, T.; Morgenstern, J.; Vecchione, B.; Vaughan, J. W.; Wallach, H.; Iii, H. D.; and Crawford, K. 2021. Datasheets for datasets. *Communications of the ACM*, 64(12): 86–92.

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

Greene, D.; Doyle, D.; and Cunningham, P. 2010. Tracking the Evolution of Communities in Dynamic Social Networks. *2010 International Conference on Advances in Social Network Analysis and Mining, ASONAM 2010*, 176–183.

Grover, A.; and Leskovec, J. 2016. node2vec: Scalable Feature Learning for Networks. New York, NY, USA: Association for Computing Machinery. ISBN 9781450342322.

Guo, C.; Wang, J.; and Zhang, Z. 2014. Evolutionary community structure discovery in dynamic weighted networks. *Physica A: Statistical Mechanics and its Applications*, 413: 565–576.

Holland, P. W.; Laskey, K. B.; and Leinhardt, S. 1983. Stochastic blockmodels: First steps. *Social Networks*, 5(2): 109–137.

Ju, W.; Fang, Z.; Gu, Y.; Liu, Z.; Long, Q.; Qiao, Z.; Qin, Y.; Shen, J.; Sun, F.; Xiao, Z.; Yang, J.; Yuan, J.; Zhao, Y.; Wang, Y.; Luo, X.; and Zhang, M. 2024. A Comprehensive Survey on Deep Graph Representation Learning. *Neural Networks*, 173: 106207.

Kipf, T. N.; and Welling, M. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *International Conference on Learning Representations*.

Kojaku, S.; Radicchi, F.; Ahn, Y.-Y.; and Fortunato, S. 2024. Network community detection via neural embeddings. *Nature Communications*, 15(1): 9446.

Kong, X.; Shi, Y.; Yu, S.; Liu, J.; and Xia, F. 2019. Academic social networks: Modeling, analysis, mining and applications. *Journal of Network and Computer Applications*, 132(C): 86–103.

Kumar, S.; Zhang, X.; and Leskovec, J. 2019. Predicting Dynamic Embedding Trajectory in Temporal Interaction Networks. In *Proceedings of the 25th ACM SIGKDD Interna-*

tional Conference on Knowledge Discovery & Data Mining, KDD '19, 1269–1278. ACM.

Lancichinetti, A.; Fortunato, S.; and Radicchi, F. 2008. Benchmark graphs for testing community detection algorithms. *Physical Review E*, 78: 046110.

Li, X.; Zhen, X.; Qi, X.; Han, H.; Zhang, L.; and Han, Z. 2023. Dynamic community detection based on graph convolutional networks and contrastive learning. *Chaos, Solitons Fractals*, 176: 114157.

Longa, A.; Lachi, V.; Santin, G.; Bianchini, M.; Lepri, B.; Lio, P.; Scarselli, F.; and Passerini, A. 2023. Graph Neural Networks for Temporal Graphs: State of the Art, Open Challenges, and Opportunities. arXiv:2302.01018.

Mahdavi, S.; Khoshraftar, S.; and An, A. 2018. dynnode2vec: Scalable Dynamic Network Embedding. *IEEE International Conference on Big Data*, 3762–3765.

Martin, L.; Muller, B.; Suarez, O.; Pedro, J.; Dupont, Y.; Romary, L.; de la Clergerie, E.; Seddah, D.; and Sagot, B. 2020. CamemBERT: a Tasty French Language Model. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, 7203–7219. Association for Computational Linguistics.

McInnes, L.; Healy, J.; and Astels, S. 2017. HDBSCAN: Hierarchical density based clustering. *The Journal of Open Source Software*, 2.

Nadakuditi, R. R.; and Newman, M. E. J. 2012. Graph Spectra and the Detectability of Community Structure in Networks. *Physical Review Letters*, 108(18): 188701.

Narayanan, A.; Mahinthan, C.; Venkatesan, R.; Chen, L.; Liu, Y.; and Jaiswal, S. 2017. graph2vec: Learning Distributed Representations of Graphs. arXiv:1707.05005.

Nguyen, G. H.; Lee, J. B.; Rossi, R. A.; Ahmed, N. K.; Koh, E.; and Kim, S. 2018. Continuous-time dynamic network embeddings. In *Companion proceedings of the the web conference 2018*, 969–976.

Palla, G.; Barabasi, A.-L.; and Vicsek, T. 2007. Quantifying social group evolution. *Nature*, 446(7136): 664–7.

Pareja, A.; Domeniconi, G.; Chen, J.; Ma, T.; Suzumura, T.; Kanezashi, H.; Kaler, T.; Schardl, T.; and Leiserson, C. 2020. EvolveGCN: Evolving Graph Convolutional Networks for Dynamic Graphs. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34: 5363–5370.

Perozzi, B.; Al-Rfou, R.; and Skiena, S. 2014. DeepWalk: online learning of social representations. KDD '14, 701–710. ACM.

Purchase, S.; Zhao, Y.; and Mullins, R. D. 2023. Revisiting Embeddings for Graph Neural Networks. arXiv:2209.09338.

Ramaciotti Morales, P.; Cointet, J.-P.; Zolotoochin, G.; Fernandez Peralta, A.; Iñiguez, G.; and Pournaki, A. 2022. Inferring attitudinal spaces in social networks. *Social Network Analysis and Mining*, 13.

Rossetti, G.; and Cazabet, R. 2019. Community Discovery in Dynamic Networks: a Survey. *ACM Comput. Surv.*, 51: 1–37.

Rossi, E.; Chamberlain, B.; Frasca, F.; Eynard, D.; Monti, F.; and Bronstein, M. 2020. Temporal Graph Networks for Deep Learning on Dynamic Graphs. arXiv:2006.10637.

Rosvall, M.; and Bergstrom, C. T. 2008. Maps of random walks on complex networks reveal community structure. *Proceedings of the National Academy of Sciences*, 105(4): 1118–1123.

Sobolevsky, S.; and Belyi, A. 2022. Graph neural network inspired algorithm for unsupervised network community detection. *Applied Network Science*, 7(1): 63.

Tandon, A.; Albeshri, A.; Thayananthan, V.; Alhalabi, W.; Radicchi, F.; and Fortunato, S. 2021. Community detection in networks using graph embeddings. *Physical Review E*, 103(2): 372–381.

Veličković, P.; Cucurull, G.; Casanova, A.; Romero, A.; Liò, P.; and Bengio, Y. 2018. Graph Attention Networks. In *International Conference on Learning Representations*.

Wang, Y.; Huang, H.; Rudin, C.; and Shaposhnik, Y. 2021. Understanding How Dimension Reduction Tools Work: An Empirical Approach to Deciphering t-SNE, UMAP, TriMap, and PaCMAP for Data Visualization. *Journal of Machine Learning Research*, 22(201): 1–73.

Wu, Z.; Pan, S.; Chen, F.; Long, G.; Zhang, C.; and Yu, P. S. 2021. A Comprehensive Survey on Graph Neural Networks. *IEEE Transactions on Neural Networks and Learning Systems*, 32(10): 4–24.

Xu, D.; Ruan, C.; Korpeoglu, E.; Kumar, S.; and Achan, K. 2020. Inductive Representation Learning on Temporal Graphs. arXiv:2002.07962.

Xue, G.; Zhong, M.; Li, J.; Chen, J.; Zhai, C.; and Kong, R. 2022. Dynamic network embedding survey. *Neurocomputing*, 472(C): 212–223.

Yang, J.; and Leskovec, J. 2015. Defining and evaluating network communities based on ground-truth. *Knowledge and Information Systems*, 42(1): 181–213.

Zhou, X.; Liang, W.; Wang, K. I.-K.; Huang, R.; and Jin, Q. 2021. Academic Influence Aware and Multidimensional Network Analysis for Research Collaboration Navigation Based on Scholarly Big Data. *IEEE Transactions on Emerging Topics in Computing*, 9(1): 246–257.

Zhu, D.; Cui, P.; Zhang, Z.; Pei, J.; and Zhu, W. 2018. High-Order Proximity Preserved Embedding for Dynamic Networks. *IEEE Transactions on Knowledge and Data Engineering*, 30(11): 2134–2144.

Paper Checklist

1. For most authors...

   (a) Would answering this research question advance science without violating social contracts, such as violating privacy norms, perpetuating unfair profiling, exacerbating the socio-economic divide, or implying disrespect to societies or cultures? Yes

   (b) Do your main claims in the abstract and introduction accurately reflect the paper's contributions and scope? Yes

(c) Do you clarify how the proposed methodological approach is appropriate for the claims made? es, see the Contribution Section for more details.

(d) Do you clarify what are possible artifacts in the data used, given population-specific distributions?

(e) Did you describe the limitations of your work? Yes, in the Discussion Section.

(f) Did you discuss any potential negative societal impacts of your work? Yes, we are aware of the potential misuse of our work as it provides insights on community detection. However, with our method it seems unrealistic to use it for user profiling at large scale. The results and methods are limited to Computational Social Science approach of studying and comparing communities.

(g) Did you discuss any potential misuse of your work? Yes, see our previous answer.

(h) Did you describe steps taken to prevent or mitigate potential negative outcomes of the research, such as data and model documentation, data anonymization, responsible release, access control, and the reproducibility of findings? The X/Twitter dataset used and the algorithms are publicly available.

(i) Have you read the ethics review guidelines and ensured that your paper conforms to them? Yes

2. Additionally, if your study involves hypotheses testing...

(a) Did you clearly state the assumptions underlying all theoretical results?

(b) Have you provided justifications for all theoretical results?

(c) Did you discuss competing hypotheses or theories that might challenge or complement your theoretical results?

(d) Have you considered alternative mechanisms or explanations that might account for the same outcomes observed in your study?

(e) Did you address potential biases or limitations in your theoretical framework?

(f) Have you related your theoretical results to the existing literature in social science?

(g) Did you discuss the implications of your theoretical results for policy, practice, or further research in the social science domain?

3. Additionally, if you are including theoretical proofs...

(a) Did you state the full set of assumptions of all theoretical results?

(b) Did you include complete proofs of all theoretical results?

4. Additionally, if you ran machine learning experiments...

(a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? All the parameters are given and the X/Twitter data are public. This is part of a Phd research and all code/methods are to be published by the end of the thesis. We gave the DOI where we are uploading code and data to reproduce the synthetic experiments.

(b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? Yes

(c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? Yes

(d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? Most of the calculations were ran on a personal computer with an 2,6 GHz Intel Core i7 CPU or a general-purpose GPU. No extra computational cost have been involved.

(e) Do you justify how the proposed evaluation is sufficient and appropriate to the claims made? Yes

(f) Do you discuss what is "the cost" of misclassification and fault (in)tolerance? All the classification used here comes either from synthetic data or from previous work that manually labeled the data with experts.

5. Additionally, if you are using existing assets (e.g., code, data, models) or curating/releasing new assets, **without compromising anonymity**...

(a) If your work uses existing assets, did you cite the creators? Yes

(b) Did you mention the license of the assets? No, but we complied with usage requirements.

(c) Did you include any new assets in the supplemental material or as a URL? Yes

(d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? No, we only use public data and publicly available code.

(e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? Yes, we anonymized the data and only provide embeddings obtained from the original messages.

(f) If you are curating or releasing new datasets, did you discuss how you intend to make your datasets FAIR (see FORCE11 (2020))?

(g) If you are curating or releasing new datasets, did you create a Datasheet for the Dataset (see Gebru et al. (2021))?

6. Additionally, if you used crowdsourcing or conducted research with human subjects, **without compromising anonymity**...

(a) Did you include the full text of instructions given to participants and screenshots?

(b) Did you describe any potential participant risks, with mentions of Institutional Review Board (IRB) approvals?

(c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation?

(d) Did you discuss how data is stored, shared, and deidentified?

Appendix

I) Parameter Sensitivity Analysis of the Clustering Algorithm

Similar to the results presented in subsection **Parameter Sensitivity Analysis**, we present here a sensitivity analysis of the parameters of the clustering algorithm used in this article, HDBSCAN. These evaluations were conducted varying the HDBSCAN parameters but keeping the same method parameters as described in the **Experimental Setup** subsection of the **Experiments on Synthetic Data** section. The Sobol method was again executed with $N_{total} = 4096$ evaluations for each algorithm on each scenario, evaluated on average Adjusted Mutual Information. There are 3 parameters : min_cluster_size, min_samples and cluster_selection_epsilon. We report separately the results of the analysis on non-GNN algorithms (dynnode2vec, CTDNE and RandomWalk) and the results on GNN-based algorithms (TGN, GAT and evolveGCN). For each category of algorithms, the mean first-order Sobol Index (S_1), mean total-order Sobol Index (S_T) are reportedbut as the rank of each parameter remained stable, it is not reported. The mean value and standard deviation of each parameter over the set of parameters achieving 95% of the best AMI are also reported.

The results for non-GNN algorithms are reported in Table 10 and Table 11.

The results for all GNN-based algorithms are reported in Table 12 and Table 13.

Parameter	mean S_1	mean S_T
min_cluster_size	0.347	0.787
min_samples	0.080	0.550
cluster_selection_epsilon	0.034	0.254

Table 10. Parameter sensitivity of HDBSCAN over all non-GNN algorithms

Parameter	mean value	value StdDev
min_cluster_size	35	20
min_samples	45	22
cluster_selection_epsilon	0.484	0.284

Table 11. Mean value and standard deviation of HDBSCAN parameters for evaluation achieving 95% of the best AMI over all non-GNN algorithms

Parameter	mean S_1	mean S_T
min_cluster_size	0.345	0.697
min_samples	0.288	0.659
cluster_selection_epsilon	0.047	0.192

Table 12. Parameter sensitivity of HDBSCAN over all GNN-based algorithms

For non-GNN algorithms, min_cluster_size is the main responsible for variation in theaverage AMI value. On the other end, GNN-based algorithms are sensitive to changes in both in min_cluster_size and min_samples. It can be explained by the fact that non-GNN algorithms tend to produce embeddings where clusters of points are further apart. Therefore, min_samples is an important parameter for separating clusters in areas of the embedding space where points from different communities overlap. Nevertheless, both categories of algorithm can achieve 95% of the best average AMI with a wide range of parameters. More than half of the evaluations for each algorithm achieve 95% of the best average AMI. We conclude that, even if some parameter tuning is required to maximize this metric, a reasonable choice of parameters will always achieve good results in terms of AMI.

Parameter	mean value	value StdDev
min_cluster_size	38	20
min_samples	44	22
cluster_selection_epsilon	0.506	0.288

Table 13. Mean value and standard deviation of HDBSCAN parameters for evaluation achieving 95% of the best AMI over all GNN-based algorithms

II) Impact of Dimensionality

In this section we examine the choice of dimensionality reduction applied to the produced embeddings. Clustering algorithms are known to perform better on low-dimensional data. As stated in the "Frequently Asked Questions" section of the documentation of HDBSCAN : "While HDBSCAN can perform well on low to medium dimensional data the performance tends to decrease significantly as dimension increases." (https://hdbscan.readthedocs.io/en/latest/faq.html). This can be seen in the example presented in Table 14 :

ndims = 64	ndims = 32	ndims = 16	ndims = 2
0.773	0.797	0.795	0.805

Table 14. average AMI value of the TGN model for different dimensions of embeddings on the *new community* scenario

As for the TGN, the best value in terms of average AMI value is generally obtained with ndims=2. This comfort us in our decision of reducing the dimensionality to 2. Furthermore, reducing the dimension of embeddings to 2 enables easy visualization of the node embeddings and their evolution through time, as shown in Figure 3.

III) Comparison of our Method with Simple Matching Strategy

In this section, we compare the proposed method with a simple matching strategy inspired by (Greene, Doyle, and Cunningham 2010). In the simple matching strategy, a Jaccard coefficient is computed between each pair of clusters $\mathcal{C}_t^i$ and $\mathcal{C}_{t+1}^j$ found in adjacent snapshots. However, instead of using a survival graph, two communities are matched if

(a) Metrics on *new* community scenario

(b) Metrics on *merging* community scenario

(c) Metrics on *fixed* community scenario

Figure 4. Metrics for TGN, GAT, RandomWalk and evolveGCN of simple matching strategy (red) against the proposed method (blue)

i) the Jaccard coefficient exceed a given threshold similar to $\tau_{similarity}$, ii) $\mathcal{C}_t^i$ is the most similar community at t for $\mathcal{C}_{t+1}^j$ and iii) $\mathcal{C}_{t+1}^j$ is the most similar community at $t+1$ for $\mathcal{C}_t^i$. Adjusted Mutual Information (AMI), Adjusted Rand Index (ARI), Fowlkes-Mallows Index (FMI), Homogeneity and Completeness are computed for the resulting communities against ground truth. The results for TGN, GAT, RandomWalk and evolveGCN are shown in Figure 4. We chose to represent RandomWalk because it consistently performs worst against the simple matching strategy using our method.

We observe that, for these algorithms, our proposed method is not better overall as RandomWalk performs slightly better with the simple matching strategy. However, TGN, GAT and evolveGCN perform better when using a strategy that allows community matching across multiple timesteps. We also observe a substantial improvement in performance with our method on the *merging* community scenario, where all algorithms struggle as shown in subsection **Results for the proposed scenarios**. This motivated us to propose our method, as we believe that a matching strategy using a broader number of steps of evolution simultaneously improves performance when performing community detection using embeddings produced with GNN-based algorithms.