Understanding the Benefits of Forgetting when Learning on Dynamic Graphs

Julien Tissier and Charlotte Laclau ✉

Univ. Lyon, UJM Saint-Etienne
CNRS, Lab Hubert Curien UMR 5516
42023, Saint-Etienne, France
{julien.tissier,charlotte.laclau}@univ-st-etienne.fr

Abstract. In order to solve graph-related tasks such as node classification, recommendation or community detection, most machine learning algorithms are based on node representations, also called embeddings, that allow to capture in the best way possible the properties of these graphs. More recently, learning node embeddings for dynamic graphs attracted significant interest due to the rich temporal information that they provide about the appearance of edges and nodes in the graph over time. In this paper, we aim to understand the effect of taking into account the static and dynamic nature of graph when learning node representations and the extent to which the latter influences the success of such learning process. Our motivation to do this stems from empirical results presented in several recent papers showing that static methods are sometimes on par or better than methods designed specifically for learning on dynamic graphs. To assess the importance of temporal information, we first propose a similarity measure between nodes based on the time distance of their edges with an explicit control over the decay of forgetting over time. We then devise a novel approach that combines the proposed time distance with static properties of the graph when learning temporal node embeddings. Our results on 3 different tasks (link prediction, node and edge classification) and 6 real-world datasets show that finding the right trade-off between static and dynamic information is crucial for learning good node representations and allows to significantly improve the results compared to state-of-the-art methods.

Keywords: Node vectors · Embedding · Dynamic graph

1 Introduction

Data in the form of graphs have become ubiquitous for describing complex information or structures from a large variety of domains of application such as a social network where users can *follow* and communicate with each other; webpages *linking* to other webpages; a group of cities connected by roads or rails; protein-protein interaction network to study genetic interactions in biology. In all these examples, a graph is defined by a set of entities (named *nodes* or *vertices*) and a set of pairs of related nodes (named *edges* or *links*). For instance, in

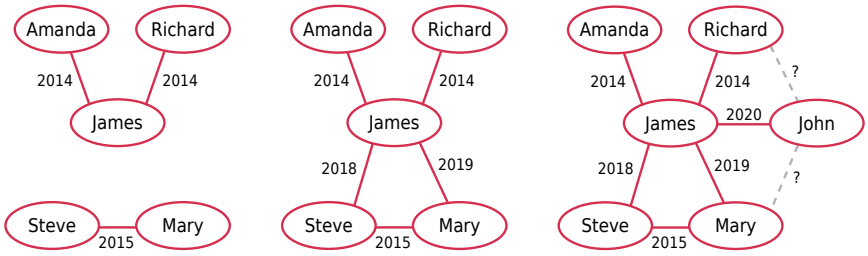


Fig. 1: A dynamic graph representing working relationships.

the case of a social network, the nodes represent users and the edges can be any relation between two nodes such as sending messages to one another or being friends. Many different problems can be solved with graph modelization: searching for the most relevant webpages given a query, being able to predict whether two people will start a relationship [33] or finding who should collaborate together [3,8]. A common approach to solve these kinds of problems is to associate each node of the graph with an *embedding*, a numeric vector reflecting the properties of this node such as its neighborhood, and more generally the structure of the overall graph. Node embeddings are then fed into a downstream machine learning model which is trained and optimized for a given task, e.g., link prediction [5,10] or node classification [4]. Several methods have been proposed to learn node embeddings directly from a graph [27,9,12,14]. Most of these methods are designed for *static* graphs, where there is no temporal information about the relations between nodes. However, most of real-world problems are represented by *dynamic* or *time-evolving* graphs where edges are ordered in time and nodes can be added or removed. Therefore, not using the temporal information during training prevents from capturing the evolution of the interactions between nodes inside their embeddings and can lead to poor predictions.

To illustrate this, let us consider a group of people and their work relationships where our task is to suggest a new collaboration to John based on Figure 1. One may see that a good suggestion here for John's new collaboration is Mary as she is the most recent collaboration (2019) of John's only connection James. However, a static method that doesn't take into account the information about the temporal evolution of this graph (ie, years over edges) will suggest Amanda, Richard, Steve and Mary to John as they are connected to his sole connection James. On the other hand, a dynamic method will take into account the temporal information *over all timestamps* even though only timestamp Figure 1b provides helpful information in this case, while Figure 1a is uninformative to this task. While for one uninformative timestamp this may not lead to a failure of the model, real-world graphs may have thousands of timestamps and attributing

the same importance to all of them may have a dramatic effect on the overall performance. Ideally, one would like to have a method that allows to control the forgetting and its speed when learning node representations in dynamic graphs to take into account only relevant information contained in them.

This paper addresses the problem of learning node representations in dynamic graphs where the temporal information is encoded inside their embeddings. We use the intuition presented above to provide a model with a way to forget the past timestamps with an explicit control over the speed of this forgetting. The main contributions are: **(1)** a novel approach to compute similarities between nodes based on static or dynamic information, suitable for both continuous and discrete dynamic graphs; **(2)** a model that learns nodes embeddings using the computed similarities and generates vectors that reflect the temporal characteristics of the graph; **(3)** an evaluation on 6 real-world datasets and 3 different tasks showing that a good trade-off between static and dynamic parts of the graph lead to the best performance in most cases.

2 Related work

2.1 Node embeddings in static graphs

In a static graph, all nodes and edges exist at the same time and no new edges appear over time. In this context, the goal of a node embedding method is to learn a function that takes a network as an input and maps each node to a low-dimensional vector. The learned vectors should reflect the structure of the network and the relations between nodes, *i.e.*, similar nodes in the graph have similar vectors. In [27], authors simulate random walks from one node to another using the edges and optimize node embeddings such that nodes that co-occur often in random walks of fixed length should be close in the embedding space. `Node2Vec` [9] uses a similar approach to build walks in the graph but it selects the next node based on a biased sampling. The generated paths are then fed into a `Word2Vec` model [21]. In the same vein, [1] propose to extend another word embedding model, namely `GloVe` [26], to learn node representations. Other methods factorize the adjacency matrix to learn a vector representation for each node with either SGD or SVD [2,23] , or train a model that learns how to combine the features of a node and its neighborhood [11]. For more details on this topic, we refer the interested reader to [12].

2.2 Node embeddings in dynamic graphs

In dynamic graphs, edges and nodes can appear (or disappear) over time. They can be separated into two categories: *time-continuous graphs*, where each change in the graph happens at a specific time t, and *discrete graphs* where a batch of changes happens during a time interval. The former can be transformed into the latter by grouping together all the changes that happen during $[t, t + T]$ where T is the duration of the interval, but not the other way around. A naive way to

learn node embeddings from a dynamic graph would be to use static methods on the final state of the graph, but the temporal information would not be captured in this case.

Several temporal node embedding techniques directly follows `Node2Vec` [35,22]. `CTDNE` generates paths in a time-continuous graph where the order of visited nodes respects the order of appearance of edges [22]. `tNodeEmbed` uses `Node2Vec` on each interval of a dynamic graph, aligns the node embeddings and passes them into a LSTM to obtain a unique vector for each node [30]. `DynGEM` trains autoencoders to reconstruct the adjacency matrix of each interval but initializes their weights with the weights of the autoencoder trained on the previous interval. The embedding of a node is the latent layer of the autoencoder after the final interval [7]. This method was extended in `dyngraph2vec` [6] where the adjacency matrices of multiple previous intervals are passed into the autoencoder. Finally, recent approaches such as `EvolveGCN` [25] or `GAEN` [29] learn embeddings at each timestep with Graph Convolution Networks or Attention models, and combine it with RNN or GRU to capture the graph evolution.

Although these methods operate on dynamic graphs, they do not take into account the activity history of an edge, *i.e.*, how often an edge appeared in the past and whether or not an edge has recently been used between two nodes. With the example in Figure 1, learning embeddings with those methods would make the vector of James close to the vectors of Richard, Mary and John, but the vectors of Mary and John should be closer because their relation with James is more recent. That is, since new edges appear over time, they should have more weight during training. The method we propose uses time difference between edges to select and weight more importantly the most recent edges during training.

Another drawback of these approaches is that they only focus on the temporal aspect of the graph and do not allow one to balance between static and dynamic structural aspects. As a result, they can be outperformed by purely static approaches on graphs where the dynamics do not carry as much information as the original structure of the graph. To tackle this problem, `JODIE` was proposed by [19]. This model focuses on bipartite graphs and learns two embeddings (a static and a dynamic one) for each node separately using RNNs. In this article, we are interested in non-bipartite graphs and we devise an objective function that allows one to learn node embeddings using both the static and the temporal information simultaneously, with an explicit control on the weight given to each part during the training.

3 Learning node embeddings using time distance

Below, we present the learning setup considered in this paper and our general framework that learns node embeddings by taking into account both static and dynamic information of a graph. This latter is then equipped with a forgetting mechanism that attributes more relevance to recent events.

3.1 Problem setup

Given a graph $G = (\mathcal{V}, \mathcal{E})$ where $\mathcal{V} = v_1, \cdots, v_n$ is the set of vertices ($|\mathcal{V}| = n$) and $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ is the set of edges, we aim to learn a function that maps vertices $v \in \mathcal{V}$ into a d-dimensional vector space, with $d \ll n$. This mapping function outputs a node embedding vector denoted by $z \in \mathbb{R}^d, \forall v \in \mathcal{V}$. $Z \in \mathbb{R}^{(n \times d)}$ is the matrix storing all node embeddings. In the context of dynamic graphs, we further assume that each edge $e_{ij} \in \mathcal{E}$ is characterized by both static and temporal attributes: the static attribute, denoted by a_{ij}, corresponds to the number of edges which occurred between a pair of nodes (v_i, v_j); the temporal attribute, denoted by $t_{ij} = t_{ij}^{(1)}, \cdots, t_{ij}^{(a_{ij})}$, is a list containing the timestamps associated with each link.

3.2 General framework

Our first goal is to design a general framework for learning node embeddings from both static and temporal attributes simultaneously. Given two nodes (v_i, v_j), we propose to minimize the error between their similarity given by the dot product of their embeddings and their static $sim_{\mathcal{S}} : \mathcal{V} \times \mathcal{V} \to \mathbb{R}$ and temporal $sim_{\mathcal{T}} : \mathcal{V} \times \mathcal{V} \to \mathbb{R}$ similarities in the original graph. As for each edge associated to a pair of nodes, these nodes can either be seen as the source or as the target node (*i.e.*, either e_{ij} or e_{ji}), we learn two vectors for each node and store them in two matrices, Z and $\tilde{Z}$. When the graph is undirected, both matrices are equivalent; when the graph is directed, the two embedding vectors allow one to differentiate edges for which a given node is the source node from edges for which it is a target node. In the end, we take the average of these embeddings to obtain one single vector for each node. Putting it all together, we consider the following optimization objective:

$$J = \sum_{i,j=1}^{n} \lambda \left[z_i^T \cdot \tilde{z}_j - \log(sim_{\mathcal{S}}(v_i, v_j) \right]^2 + (1 - \lambda) \left[z_i^T \cdot \tilde{z}_j - \log(sim_{\mathcal{T}}(v_i, v_j)) \right]^2, \tag{1}$$

where λ is a hyperparameter allowing us to control the balance between the static and the temporal component in the final node embeddings. Since the *log* function is not defined for 0, we discard the zero-values of $sim_{\mathcal{S}}(v_i, v_j)$ and $sim_{\mathcal{T}}(v_i, v_j)$ to be used in the objective function. If two nodes have a static or a temporal similarity of 0, it means there is no edge between them and therefore their vectors should not be trained to be moved closer. We now proceed to defining the last two missing ingredients $sim_{\mathcal{S}}(v_i, v_j)$ and $sim_{\mathcal{T}}(v_i, v_j)$.

3.3 Static and temporal similarities between nodes

To illustrate our proposal for static and dynamic similarity functions, we use a running example given in Figure 2 throughout this section. The latter is given by a small dynamic graph composed of 7 nodes ($n = 7$). In this dynamic graph,

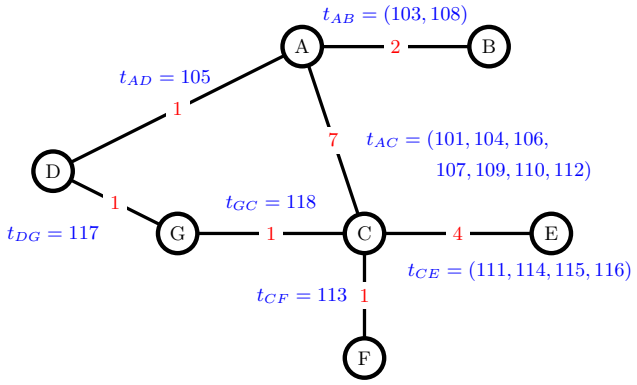


Fig. 2: Example of a temporal graph. In red, the number of edges between nodes. In blue, the times at which they appear.

an edge between two nodes can appear multiple times (*e.g.*, two people can send several emails to the other one). For instance, between A and B, there are 2 edges, that appear at time $t = 103$ and $t = 108$ (blue numbers). Red numbers in Figure 2 correspond to the number of edges between two related nodes (which does not depend on the time of appearance of edges). We are now ready to define $sim_{\mathcal{S}}(v_i, v_j)$ and $sim_{\mathcal{T}}(v_i, v_j)$.

Static similarities Given the abundance of different node embedding techniques for static graphs, one can define $sim_{\mathcal{S}}(v_i, v_j)$ in many different ways. In this work, we propose to define a similarity based on the normalized adjacency matrix similar to the LINE embedding model [31] where $sim_{\mathcal{S}}(v_i, v_j)$ denotes the probability of going from a node v_i to a node v_j in a random walk. These probabilities rely on both first and second-order proximity statistics and are computed using the static edge attributes. We have, $\forall i, j = 1, \cdots, n$:

$$sim_{\mathcal{S}}(v_i, v_j) = \begin{cases} \frac{a_{ij}}{a_i} & \text{if } e_{ij} \in \mathcal{E} \\ \sum_k \frac{a_{ik}}{a_i} \cdot \frac{a_{kj}}{(a_k - a_{ik})} & \text{if } e_{ij} \notin \mathcal{E} \cap \exists v_k : e_{ik}, e_{kj} \in \mathcal{E} \\ 0 & \text{otherwise,} \end{cases} \tag{2}$$

where a_{ij} is the number of edges that occur between a pair of nodes v_i and v_j; $a_i = \sum_j a_{ij}$. This similarity can be easily computed for either directed or undirected graphs. One should note that the role of the first term is to capture first-order proximity between two nodes in the graph (*i.e.*, the existence of an edge between two nodes) while the second term captures second-order proximity for pair of nodes separated by a distance of 2 in the graph.

In Figure 2, we have $s_{CA} = 7/13$ and $s_{AC} = 7/10$. One should note that static similarities are not symmetric. For nodes with a distance of 2, we have $s_{CB} = s_{CA} \times s_{AB} = 14/39$. When several paths are available to join v_i and v_j, we sum the probabilities of all paths (so $s_{CD} = 10/39$).

We want to stress out that the first part of the objective function is versatile (see [32]), as one can define the static similarity using other popular methods such as, for instance, Personalized PageRank [24] or SimRank [13].

Temporal similarities As explained above, the temporal similarity should take into account the information from different timestamps of a dynamic graph but also allow the model to forget the past that became irrelevant. To this end, we propose to define $sim_{\mathcal{T}}$ as a function of the time delta between the most recent and the other edges of a node and its neighbors. Formally, it is defined as:

$$sim_{\mathcal{T}}(v_i, v_j) = \begin{cases} \sum_k f(\Delta_{t_{ij}^{(k)}}) & \text{if } e_{ij} \in \mathcal{E} \\ 0 & \text{otherwise,} \end{cases} \tag{3}$$

where f is a decreasing function, allowing us to give more weight to recent edges (*i.e.*, when Δ is small) and $\Delta_{t_{ij}} = \max_j(t_{ij}) - t_{ij}$ (*i.e.*, the time difference between the timestamp t_{ij} and the most recent timestamp among all edges starting from node v_i). Choosing f to be a decreasing function indicates that we assume that as times passes, the strength of the relation between two nodes becomes weaker. We believe that for social networks, or co-citation networks this is a reasonable assumption. However for applications such as Protein-Protein interaction where this assumption might not be ideal, one can easily relax this condition and choose a function f that suits best their need.

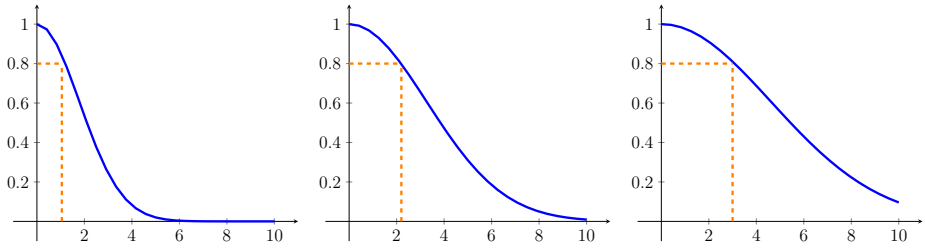
Going back to our example, blue numbers in Figure 2 are the times at which edges appear between the nodes. A model learning node embeddings using only static information would make the vector of C more similar to A than to E because it has more edges with A ($7 > 4$). However, edges between C and E are more recent than between C and A ($t = 114, 115, 116$ vs. $t = 110, 112$). The intuition behind temporal similarities is to bring closer the vectors of nodes having the most recent interactions. In Figure 2, we have $d_{CE} = f(\Delta_{111}) + f(\Delta_{114}) + f(\Delta_{115}) + f(\Delta_{116})$. The most recent edge of C appears at $t = 118$, so $\Delta_{111} = 118 - 111 = 7$ and therefore $d_{CE} = f(7) + f(4) + f(3) + f(2)$. One should note that temporal similarities are also not symmetric. Indeed, for d_{EC}, the most recent edge of E appears at $t = 116$, so we would have $\Delta_{111} = 116 - 111 = 5$.

In the following, we choose f to be a survival function of the form:

$$f : x \to e^{[-\alpha * (x/x_{max})^2]},$$

where α is a hyperparameter that controls the decay rate of this weight function and x_{max} is the maximum value that can be passed to f (*i.e.*, the largest time distance). This function models the probability for a given relation (i.e. an edge) to survive past a certain time, here represented by time-steps (see Figure 3). For high values of α this probability decreases faster to 0, than for small values of α. In that case, our model strongly favors the short-term rather than the long-term dynamic of the graph, a reasonable assumption when dealing with a graph presenting important structural changes between two time-steps. On the other hand, as α decreases, our model will take into account all the edges from

the past, hence capturing long-term dynamics. This particular setting will suit graphs presenting a smooth evolution over time, with all edges being relevant at any time. In the asymptotic limit, this function can be “flat" enough to behave as a dynamic model that takes into account all timestamps of a dynamic graph.



(a) $\alpha = 25$ (b) $\alpha = 5.58$ (c) $\alpha = 6$

Fig. 3: Survival function $f : x \rightarrow exp^{[-\alpha * (x/x_{max})^2]}$ for different values of α and with $x_{max} = 16$. (b) satisfies the Pareto's law meaning that 20% most recent edges have a weight among the highest 80%, i.e., $f(0.2 * \Delta_{t_{i_{max}}}) = 0.8$.

3.4 Complexity Analysis

For $sim_{\mathcal{T}}$, we have to compute at most $|\mathcal{E}|$ values (corresponding to the non-zero entries of the adjacency matrix) because only direct edges. For $sim_{\mathcal{S}}$ we have to compute on average $|\mathcal{V}| \times \kappa^2$ values where κ is the average node degree in the graph (so each node can reach on average κ^2 other nodes with a distance of 2). Therefore, the complexity of our model is $\mathcal{O}(|\mathcal{V}| \times \kappa^2 + |\mathcal{E}|)$ because it iterates only over the non-zero similarities.

4 Experiments

4.1 Datasets

We evaluate our hypothesis about the importance of combining both static and dynamic information for learning good node embeddings on 6 real-world datasets representing dynamic graphs: messages sent between people (Radoslaw[1] [28], ENRON[2] [16]), links between webpages (Subreddit[3] [17]), network of routers

[1] https://networkrepository.com/ia-radoslaw-email.php
[2] https://networkrepository.com/ia-enron-employees.php
[3] https://snap.stanford.edu/data/soc-RedditHyperlinks.html

(Autonomous Systems[4] [20]), rating of Bitcoin users (BTC-Alpha[5] and BTC-OTC[6] [18]). As explained in Subsection 2.2, dynamic graphs can be continuous or discrete. We use both types of graphs in our experiments to demonstrate that our model works regardless of the nature of the dynamic graph. Table 1 reports statistics about each dataset. For some datasets, we use a smaller version because some baselines were not able to run on the full version (for Subreddit, we consider only nodes with at least 10 edges; for Autonomous Systems (AS), we use only the 100 first steps).

Table 1: Statistics about the datasets used for experiments. ($*$) indicates datasets that have been shrunk. $\overline{deg}(v_i)$ (resp. $\overline{C_i}$ coef.) is the average degree (resp. clustering coefficient) of all nodes.

Dataset	Nodes	Edges	Type	$\overline{deg}(v_i)$	$\overline{C_i}$ coef.
Radoslaw	167	82,876	Continuous	992.5	0.592
ENRON	150	47,088	Continuous	627.8	0.521
Subreddit ($*$)	6,340	223,457	Continuous	70.5	0.364
Auto. Sys. ($*$)	3,569	561,139	Discrete	314.5	0.257
BTC-Alpha	3,783	24,186	Continuous	12.8	0.177
BTC-OTC	5,881	35,592	Continuous	12.1	0.178

4.2 Evaluation tasks

Link predictions. This task consists in predicting if a link between two nodes exists or not in the graph. We follow the same protocol as in [15]. For each edge (u, v) in $\mathcal{T}$, the set of all unique test edges, we generate the list $\mathcal{N}_v$ (resp. $\mathcal{N}_u$) of negative edges obtained by replacing v (resp. u) by another node from the graph such that the negative edge does not exist. Then, the cosine similarity between the embeddings of u and v is computed, as well as its rank r_v (resp. r_u) against the cosine similarities of all edges in $\mathcal{N}_v$ (resp. $\mathcal{N}_u$).

The Mean Reciprocal Rank (MRR) is the mean of the inverse of r_v and r_u for all edges (u, v) in $\mathcal{T}$:

$$MRR = \frac{1}{2 \times |\mathcal{T}|} \sum_{(u,v) \in \mathcal{T}} \left(\frac{1}{r_u} + \frac{1}{r_v} \right).$$

In addition to the MRR, we also compute Hits@K metrics (the percentages of ranks r_v or r_u which are less than K).

[4] https://snap.stanford.edu/data/as-733.html
[5] https://snap.stanford.edu/data/soc-sign-bitcoin-alpha.html
[6] https://snap.stanford.edu/data/soc-sign-bitcoin-otc.html

Node classification. This task only applies to the Subreddit dataset. It consists in predicting the correct label of nodes in the graph. Plenty of graphs with labeled nodes exist in the static configuration (*i.e.* when there is no evolution in the network over time) but dynamic graphs with node class information are almost non-existent. To overcome this problem, we generate labels for each node in the Subreddit dataset. It contains 6,340 nodes. We use an automatic method to generate the labels for similar nodes with a clustering algorithm and manually verify that related subreddits are within the same cluster. Each node in this dataset represents a subreddit, the name of a topic-specific discussion forum (*e.g.* chess, Olympics). Using the property of word embeddings to encapsulate semantic information, we generate the word embedding of each subreddit name using Fasttext library[7]. Since subreddits such as "skiing" or "skateboarding" are words related to the same semantic field, their respective word embeddings should be similar thanks to the Fasttext learning scheme. We then use a K-Means clustering algorithm to group together similar word embeddings, thus grouping related subreddit. We try different values for K, from 20 to 80. When K is too low, there are not enough clusters and unrelated subreddits fall into the same category when they should not. When K is too high, related subreddits are often separated into different groups. We find that using 50 clusters is a good trade-off. Each node is then associated with the ID of the group it belongs to. Table 2 shows some examples of subreddits with the same label. For the node classification task, we train a logistic regression classifier on the learned node embeddings (the embeddings learned from our dynamic graph method, not those generated with Fasttext) to predict their corresponding class. We report the accuracy.

Table 2: Examples of subreddits with the Autona for the node classification

Label = 3	Label = 17	Label = 32	Label = 41	Label = 43
altcoin	judo	blizzard	albania	amazon
bitcoin	olympics	bloodborne	finland	ebay
bitcoinmining	skateboarding	counterstrike	italy	netflix
cryptocurrency	skiing	halo	poland	silkroad
dogecoin	swimming	overwatch	spain	spotify
ethereum	tennis	streetfighter	usa	tumblr

Edge classification. This task only applies to the Bitcoin datasets and consists in predicting the class of edges. In BTC-Alpha and BTC-OTC, an edge indicates that one user of the Bitcoin trading platform rated another user on a scale from -10 to +10. We generate a binary label for each edge: 0 if the rating is negative, 1 otherwise. We compute an embedding for each edge by averaging the embeddings of the involved nodes. We train a linear regression classifier on

[7] https://fasttext.cc/

the edge embeddings to predict their corresponding binary label. The dataset is unbalanced, about 90% of edges have a label of 1. We report the F1 score.

4.3 Training settings

Each graph is split into a train (75%) and a test set (25%) according to timestamps. The same train and test sets are used for all models. We train vectors of 20 dimensions for small datasets (ENRON, Radoslaw) and 100 dimensions otherwise (same as in [7]). Our hyperparameters are tuned with a grid search to maximize the MRR on the train set, with α ranging from 2 to 60 with steps of 3, λ ranging from 0 to 1 with steps of 0.05 and the number of epochs ranging from 20 to 300 by steps of 20. All experiments are done with an Intel Xeon E3-1246 CPU, a NVIDIA Titan X GPU and 32 GB of RAM.

4.4 Baselines

Our model uses both static and temporal information during training. Therefore, we compare it against methods that learn node embeddings from static graphs (`Node2Vec`, `GraphSage` [11]) and other methods from dynamic graphs (`CTDNE`, `tNodeEmbed`, `DynGEM`, `dyngraph2vec` (AERNN version), `EvolveGCN`). We set a walk length of 40 for `Node2Vec` and `CTDNE`, and a length of 5 for `GraphSage`. We generate 10 walks per node for these methods. Other hyperparameters are set as indicated in their respective papers. For `GraphSage`, we use one-hot vectors as a replacement for node feature vectors when they are not present, as advised by the authors. We do not use `DynamicTriad` [34] as a baseline because it has been reported to have lower scores than `tNodeEmbed` and `dyngraph2vec`. Baselines are trained on the same machine as our model.

5 Results and analysis of the model

In this section[8], our goal is to answer two following questions:

Q1 *Does learning from both static and dynamic information lead to better node embeddings?*
To this end, we compare our method with several static and dynamic node embedding methods on 6 real-world datasets and on 3 different tasks.

Q2 *What insights such framework can provide about the graphs on which it is applied?*
To answer this question, we analyze the optimal values of λ and α revealing the importance of static component for learning node embeddings and the effect of forgetting when taking into account temporal information.

The proposed method has both a static and a dynamic component. We first compare its results against dynamic methods as we are working with dynamic graphs, and then against static methods. Thereafter, we analyze the role of each component depending on the dataset and we evaluate its complexity.

[8] Code to reproduce our results and access datasets can be found here: https://github.com/laclauc/DynSimilarity

Table 3: MRR and Hits@K metrics on 5 datasets for our method and other baselines on a link prediction task. Bold and underline results indicate respectively the best and the second best value for each metric.

	Radoslaw			ENRON			Subreddit			AS		
	MRR	Hits@5	Hits@50	MRR	Hits@5	Hits@50	MRR	Hits@5	Hits@50	MRR	Hits@5	Hits@50
Static												
Node2Vec	.088	15.82%	62.18%	<u>.247</u>	<u>52.65%</u>	<u>91.27%</u>	<u>.123</u>	<u>21.07%</u>	<u>37.04%</u>	**.221**	**44.94%**	**70.12%**
GraphSage	.100	17.42%	71.12%	.230	48.85%	86.24%	.100	15.60%	31.21%	.146	27.79%	56.04%
Dynamic												
CTDNE	.111	19.28%	<u>82.46%</u>	.235	48.32%	86.80%	.112	18.11%	31.34%	<u>.215</u>	<u>42.53%</u>	<u>69.88%</u>
tNodeEmbed	.169	33.93%	71.22%	.228	47.24%	86.35%	.105	16.32%	31.29%	.020	2.24%	6.19%
DynGEM	.123	16.93%	55.62%	.177	34.18%	74.44%	.080	9.29%	11.56%	.077	8.37%	11.10%
dyngraph2vec	<u>.180</u>	<u>36.51%</u>	81.02%	.145	27.46%	72.05%	.088	9.69%	17.79%	.031	3.84%	8.19%
EvolveGCN	.097	16.96%	64.43%	.124	22.54%	67.01%	.053	6.02%	10.40%	.017	1.64%	4.48%
Our	**0.329**	**69.33%**	**96.71%**	**0.298**	**62.24%**	**92.16%**	**0.132**	**22.54%**	**37.58%**	0.146	28.47%	50.80%

5.1 Against dynamic graph methods

Table 3 reports the scores of all modelsfor the task of link prediction. The MRR is the average of the inverse rank of a true edge against false random edges. Higher scores indicate that a model is able to better differentiate true edges against negative ones. On 3 out of 5 datasets, our method strongly outperforms all the other dynamic methods in terms of MRR. The Hits@K metrics in Table 3 indicate the percentage of true edges whose rank is among the first K when compared to several hundreds negative edges. We observe similar results as for the MRR. For instance, on Radoslaw, our method is able to retrieve almost 70% of true edges in the top 5 while other methods can only retrieve 36% at best. This means that our method ranks the majority of true edges with a much better rank than the other methods (top 5 vs. top 50), which is useful in a recommendation task. Note that Radoslaw and ENRON are the two datasets with the highest clustering coefficient and average degree (Table 1). The temporal information is crucial in these datasets because a change in the network topology spreads faster than for other datasets as they both represent email communications between people of a company, and are well suited to evaluate dynamic methods [5]. Our method outperforms other dynamic methods on those graphs, demonstrating that it is well appropriate for graphs where temporal information is important.

Results on the two other tasks are reported in Table 4. For node classification, the proposed model improves the classification accuracy over the other dynamic baselines. The results notably show an important improvement over the auto-encoder and GCN-based approaches, with results almost 4 times better than EvolveGCN for instance. Finally, for the edge classification task, we outperform all dynamic baselines on both Bitcoin datasets by an important margin.

Table 4: Results on a node classification task (accuracy of correct predictions among 50 classes) and an edge binary classification task (F1 score).

	Node classif.	Edge classif.	
	Subreddits	BTC-Alpha	BTC-OTC
Static methods			
Node2Vec	18.24%	0.254	0.290
GraphSage	19.67%	0.293	0.312
Dynamic methods			
CTDNE	19.21%	0.192	0.251
tNodeEmbed	22.20%	0.173	0.280
DynGEM	7.03%	0.080	0.321
dyngraph2vec	8.24%	0.398	0.273
EvolveGCN	6.04%	0.361	0.268
Our	**22.86%**	**0.429**	**0.360**

5.2 Against static graph methods

Table 3 reports the scores of static baselines (`Node2Vec` and `GraphSage`). Our method outperforms `GraphSage` and `Node2Vec` on 4 out of 5 graphs in terms of MRR (*e.g.* 0.329 vs. 0.100 on Radoslaw), and Hits@5 (*e.g.* 62.24% vs. 52.65% on Subreddit). It is on par on AS against `GraphSage` but looses against `Node2Vec`.

One should note that AS and Subreddits are the biggest graphs among the datasets, and most of the edges do not vary over time. Therefore, their behavior is similar to a static graph, which explains why `Node2Vec` is only slightly under or better than our dynamic method for them. Our best results for these datasets are obtained when the dynamic component of our model driven by 1-λ is almost zero , validating the assumption that they have a static behavior. However, our method doing better on the other datasets demonstrates that the dynamic component in our model is important to learn embeddings when the dataset evolves greatly over time, which `Node2Vec` cannot achieve. Our conclusions on node and edge classification are similar to the one made against dynamic approaches.

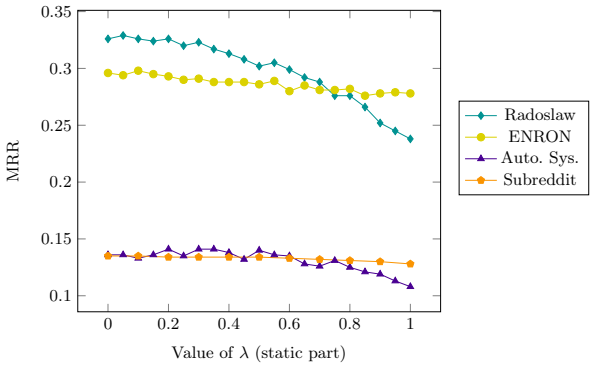
Overall, the obtained results demonstrate that our model is able to produce node embeddings capturing both the dynamic and the static aspects of an evolving graph. They are versatile, as we obtain consistently good results for various tasks, including link prediction, node classification and edge classification. Our approach is also robust to various graphs topology as we either outperform all other baselines or rank second at worst on all datasets. This shows the benefit brought by the two components of our objective function.

5.3 Influence of the hyperparameters of the model

We are now ready to address the second question. Our model naturally provide a way to gain insights regarding the dynamic of the graphs, through its two hyperparameters λ and α. Our model has both a static and a dynamic component,

	λ	Decay rate (α)
Radoslaw	0.05	2.634
ENRON	0.10	4.063
Subreddit	0.70	4.063
Auto. Sys.	0.40	30.099
BTC-Alpha	0.25	4.063
BTC-OTC	0.35	5.579

(a)



(b)

Fig. 4: Evolution of MRR versus static coefficient used in our model.

each one with its own coefficient in the objective function (resp. λ and (1-λ)). The two coefficients are inter-dependent: increasing one decreases the other. Therefore, they behave as a cursor that one can set to favour the static or the dynamic part of the method. We notice in our experiments that the value of λ must be selected depending on the nature of the dataset. Figure 4(b) shows the evolution of the MRR according to the value of λ on some datasets. We can see that when λ increases, the MRR drops for most datasets. Indeed, a value of λ close to 1 means that the dynamic part in our objective function is almost non-existent, which is detrimental for dynamic graphs. Unsurprisingly, the highest decrease is on Radoslaw, two datasets with a strong evolution over time that therefore require a large dynamic component in the model to capture temporal information. For Subreddit, the drop is smaller because this dataset does not vary a lot over time, so the static part is more important in this case. This is confirmed by Figure 4(a), which reports the λ and α hyperparameters that gives the best scores for each dataset. Radoslaw needs a dynamic oriented model ((1-λ) close to 1) while Subreddit needs a dominant static part ($\lambda = 0.7$) to achieve good results. We also notice that a large value of α makes the temporal similarities more focused on very short-term interactions while a smaller α allows to consider a longer history of previous events.

5.4 Training times

Table 5 reports the time required by each method to train on all 6 datasets. Due to its simplicity (no complex architectures like RNN/LSTM nor convolution networks), our method only needs 40 minutes, which makes it the fastest among all the dynamic methods (*e.g.* 6h36 for `DynGEM`, 24h01 for `dyngraph2vec`).

Table 5: Times required to train all models on the 6 graphs. Minimum time is highlighted by boldface.

	Node2vec	GraphSage	tNodeEmbed	CTDNE	DynGEM	dyngraph2vec	EvolveGCN	Our
Training time	48:53	14:43:27	1:34:08	4:00:56	6:36:10	24:01:18	5:33:11	**40:36**

6 Conclusion

This paper studies the importance of combining both static and dynamic information when learn node embeddings in dynamic graphs. It introduces a novel temporal similarity measure between nodes based on time distance of edges and a model that uses it in addition to static similarities to learn embeddings that reflect the structure of the dynamic graph. This method allows one to emphasize either the static or the dynamic component of the model to adapt to different kinds of graphs. It obtains better scores than other dynamic methods on 6 real-world datasets for various tasks thus suggesting that fully dynamic approaches may be too rigid for efficient learning in dynamic graphs. Further research directions of this work are many. First, we would like to explore new types of node similarities to train on different specific graphs such as bipartite graphs or knowledge-based graphs. tasks. We also plan to investigate how to integrate temporal node attributes into this framework by leveraging adapted Graph Neural Networks (GNN) architectures [36].

References

1. Brochier, R., Guille, A., Velcin, J., Global vectors for node representations. In: WWW. pp. 2587–2593. ACM (2019)
2. Cao, S., Lu, W., Xu, Q.: Grarep, Learning graph representations with global structural information. In: CIKM. pp. 891–900 (2015)
3. Chuan, P.M., Ali, M., Khang, T.D., Dey, N., et al., Link prediction in co-authorship networks based on hybrid content similarity metric. Applied Intelligence **48**(8), 2470–2486 (2018)
4. Dalmia, A., Gupta, M., Towards interpretation of node embeddings. In: Companion Proceedings of the The Web Conference 2018. pp. 945–952 (2018)
5. De Winter, S., Decuypere, T., Mitrović, S., Baesens, B., De Weerdt, J., Combining temporal aspects of dynamic networks with node2vec for a more efficient dynamic link prediction. In: ASONAM. pp. 1234–1241. IEEE (2018)
6. Goyal, P., Chhetri, S.R., Canedo, A., dyngraph2vec: Capturing network dynamics using dynamic graph representation learning. KBS **187**, 104816 (2020)
7. Goyal, P., Kamra, N., He, X., Liu, Y. Dyngem: Deep embedding method for dynamic graphs. arXiv preprint arXiv:1805.11273 (2018)
8. Goyal, P., Sapienza, A., Ferrara, E.: Recommending teammates with deep neural networks. In: Hypertext and Social Media, pp. 57–61 (2018)
9. Grover, A., Leskovec, J.: node2vec: Scalable feature learning for networks. In: KDD. pp. 855–864 (2016)

10. Haghani, S., Keyvanpour, M.R.: A systemic analysis of link prediction in social network. Artificial Intelligence Review **52**(3), 1961–1995 (2019)
11. Hamilton, W., Ying, Z., Leskovec, J.: Inductive representation learning on large graphs. In: NeurIPS (2017)
12. Hamilton, W.L., Ying, R., Leskovec, J.: Representation learning on graphs: Methods and applications. IEEE Data Eng. Bull. **40**(3), 52–74 (2017)
13. Jeh, G., Widom, J.: Simrank: A measure of structural-context similarity. In: Proceedings KDD. p. 538–543 (2002)
14. Kazemi, S.M., Goel, R., Jain, K., Kobyzev, I., Sethi, A., Forsyth, P., Poupart, P.: Representation learning for dynamic graphs: A survey. JMLR **21**(70), 1–73 (2020)
15. Kazemi, S.M., Poole, D.: Simple embedding for link prediction in knowledge graphs. In: NeurIPS (2018)
16. Klimt, B., Yang, Y.: Introducing the enron corpus. In: CEAS (2004)
17. Kumar, S., Hamilton, W.L., Leskovec, J., Jurafsky, D.: Community interaction and conflict on the web. In: WWW. pp. 933–943 (2018)
18. Kumar, S., Spezzano, F., Subrahmanian, V., Faloutsos, C.: Edge weight prediction in weighted signed networks. In: ICDM. pp. 221–230. IEEE (2016)
19. Kumar, S., Zhang, X., Leskovec, J.: Predicting dynamic embedding trajectory in temporal interaction networks. In: KDD. p. 1269–1278 (2019)
20. Leskovec, J., Kleinberg, J., Faloutsos, C.: Graphs over time: densification laws, shrinking diameters and possible explanations. In: KDD. pp. 177–187 (2005)
21. Mikolov, T., Chen, K., Corrado, G., Dean, J.: Efficient estimation of word representations in vector space. arXiv preprint arXiv:1301.3781 (2013)
22. Nguyen, G.H., Lee, J.B., Rossi, R.A., Ahmed, N.K., Koh, E., Kim, S.: Continuous-time dynamic network embeddings. In: WWW (2018)
23. Ou, M., Cui, P., Pei, J., Zhang, Z., Zhu, W.: Asymmetric transitivity preserving graph embedding. In: KDD. pp. 1105–1114 (2016)
24. Page, L., Brin, S., Motwani, R., Winograd, T.: The pagerank citation ranking: Bringing order to the web. Technical Report 1999-66, Stanford InfoLab (1999)
25. Pareja, A., Domeniconi, G., Chen, J., Ma, T., Suzumura, T., Kanezashi, H., Kaler, T., Schardl, T., Leiserson, C.: Evolvegcn: Evolving graph convolutional networks for dynamic graphs. In: AAAI. pp. 5363–5370 (2020)
26. Pennington, J., Socher, R., Manning, C.D.: Glove: Global vectors for word representation. In: EMNLP. pp. 1532–1543. ACL (2014)
27. Perozzi, B., Al-Rfou, R., Skiena, S.: Deepwalk: Online learning of social representations. In: KDD. pp. 701–710 (2014)
28. Rossi, R.A., Ahmed, N.K.: The network data repository with interactive graph analytics and visualization. In: AAAI (2015), http://networkrepository.com
29. Shi, M., Huang, Y., Zhu, X., Tang, Y., Zhuang, Y., Liu, J.: Gaen: Graph attention evolving networks. In: IJCAI. pp. 1541–1547 (2021)
30. Singer, U., Guy, I., Radinsky, K.: Node embedding over temporal graphs. In: IJCAI. pp. 4605–4612 (2019)
31. Tang, J., Qu, M., Wang, M., Zhang, M., Yan, J., Mei, Q.: Line: Large-scale information network embedding. In: Proceedings of WWW. p. 1067–1077 (2015)
32. Tsitsulin, A., Mottin, D., Karras, P., Müller, E.: Verse: Versatile graph embeddings from similarity measures. In: Proceedings of WWW. p. 539–548 (2018)
33. Wang, P., Xu, B., Wu, Y., Zhou, X.: Link prediction in social networks: the state-of-the-art. Science China Information Sciences **58**(1), 1–38 (2015)
34. Zhou, L., Yang, Y., Ren, X., Wu, F., Zhuang, Y.: Dynamic network embedding by modeling triadic closure process. In: AAAI (2018)

35. Haddad, M., Bothorel, C., Lenca, P., Bedart, D.:TemporalNode2vec: Temporal Node Embedding in Temporal Networks. In: Complex Networks (2019)
36. Rossi, E., Chamberlain, B., Frasca, F., Eynard, D., Monti, F., Bronstein, M. :Temporal Graph Networks for Deep Learning on Dynamic Graphs. In:arxiv