FILDNE: A Framework for Incremental Learning of Dynamic Networks Embeddings

Piotr Bielak[a], Kamil Tagowski[a], Maciej Falkiewicz[a], Tomasz Kajdanowicz[a], Nitesh V. Chawla[a,b]

[a]*Department of Computational Intelligence, Wroclaw University of Science and Technology, Poland*
[b]*Department of Computer Science and Engineering, University of Notre Dame, Notre Dame, IN, USA*

Abstract

Representation learning on graphs has emerged as a powerful mechanism to automate feature vector generation for downstream machine learning tasks. The advances in representation on graphs have centered on both homogeneous and heterogeneous graphs, where the latter presenting the challenges associated with multi-typed nodes and/or edges. In this paper, we consider the additional challenge of evolving graphs. We ask the question whether the advances in representation learning for static graphs can be leveraged for dynamic graphs and how? It is important to be able to incorporate those advances to maximize the utility and generalization of methods. To that end, we propose the Framework for Incremental Learning of Dynamic Networks Embedding (FILDNE), which can utilize any existing static representation learning method for learning node embeddings, while keeping the computational costs low. FILDNE integrates the feature vectors computed using the standard methods over different timesteps into a single representation by developing a convex combination function and alignment mechanism. Experimental results on several downstream tasks, over seven real world data sets, show that FILDNE is able to reduce memory and computational time costs while providing competitive quality measure gains with respect to the contemporary methods for representation learning on dynamic graphs.

Keywords: Representation learning, Dynamic graph embedding, Incremental network embedding

1. Introduction

Learning embeddings from networks or graphs is pervasive with a sundry of applications across various fields, including social networks [33, 16, 18, 39], biological networks [26, 41], molecular networks [40, 42], spatial networks [38, 37], citation networks [33, 11, 12], transportation networks [13] and many others. These embeddings are generally learned in an unsupervised fashion, providing an automated way of discovering dynamic features and enabling several downstream inductive learning tasks (and in some cases transductive learning tasks as well).

The vast majority of graph embedding methods are devoted to so-called static networks, whose structure does not evolve over time. However, in most real-world scenarios, one has to deal with changes in the data, e.g., updates of node attributes and structural adjustments, like addition or removal of edges (links) and nodes. On the one hand, dynamics can infrequently occur, which can be easily addressed with static approaches. On the other hand, there are streams of temporal events that constitute constantly evolving networks. The latter case requires dedicated, computationally, and memory-wise efficient solutions. Thus, a key challenge remains: how to effectively learn network embeddings on dynamic networks? While recent works have addressed incremental learning of embeddings on dynamic graphs, in this paper, we consider a framework that encompasses existing methods for learning embeddings and enables them for a dynamic environment.
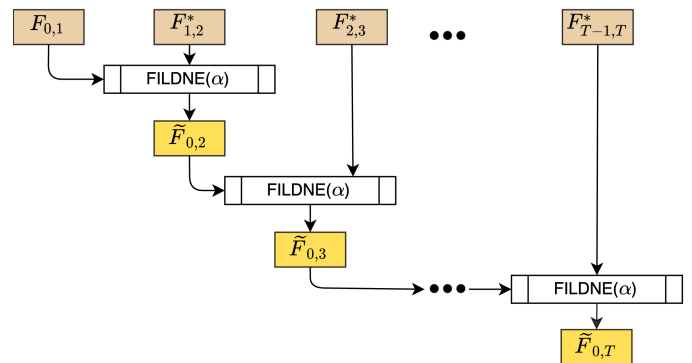

Figure 1: The FILDNE method applied on graph stream. At the beginning, FILDNE composes of two embeddings computed by the Base embedding method. Next, with each new snapshot, FILDNE composes the aligned version of the current embedding $F_{t-1,t}^*$ with the output of the previous iteration $\tilde{F}_{0,t}$.

Let us consider two conceptually different aspects of Dynamic Graph Embedding. The first is a naive one, where each new data batch triggers the computation of a new representation for historical data, completely disregarding previous feature vectors. The second one is an incremental learning paradigm, where both the time and storage costs are reduced by updating embeddings based on new temporal events. Incremental learning algorithms might specify a new objective function and could also be constrained to the types of problems that they could be applied to [6]. However, there is also a need for a framework that

arXiv:1904.03423v2 [stat.ML] 19 Nov 2020

is able to incorporate the existing works for embeddings and implement them in an incremental manner, allowing for time and space costs to be reduced while retaining the quality of the base method.

This work. In this paper, we propose FILDNE, for incremental learning of embeddings of dynamic network. Our contribution can be summarized as follows:

1. We provide a method that utilizes historical embeddings and incrementally enhances them based on batched event stream (non-overlapping snapshots for keeping the embeddings current; see Figure 2). We consider two variants: (a) the first one, FILDNE, recursively combines a pair of embeddings at each step using a hyper-parameter to steer the importance weighting (see Figure 1; (b) the second one, k-FILDNE, combines a vector of k embeddings at once using an automatic estimation method for importance weighting parameters.

2. The proposed method can work with any graph embedding method, including static and temporal graph embedding methods. Using a novel reference nodes selection scheme, our method performs an embedding alignment step that allows us to apply convex combination despite rotations and translations of embedding spaces. Moreover, our framework is designed to work in an unsupervised manner that does not require obtaining any additional class labels.

3. Through comprehensive empirical analyses that include link prediction, edge classification, and graph reconstruction tasks, we demonstrate that FILDNE allows reducing memory and computational time costs while being competitive when compared to other streaming methods in terms of embedding quality. Our hyper-parameter sensitivity study (see Figure 10) shows how the balance of importance of past and recent events (data batches) influences the performance of the representation.

The paper is organized as follows: in Section 2, we present an overview of the related work in the domain, whereas in Section 3, we introduce several formal definitions and notation. Next, in Section 4, we propose our Framework for Incremental Learning of Dynamic Networks Embeddings and provide its detailed description. Further, in Section 5 we report our extensive experiments results. Section 6 concludes our work and outlines the directions for future work.

2. Related work

Network embedding problem has attracted a lot of attention from the research community worldwide in recent years. Plenty of methods have been developed, each focused on a different aspect of network embedding, such as proximity, structure, attributes, learning paradigm, scalability, to name only a few [5, 8, 6]. In this section, we discuss several methods that are relevant to the scope of our paper. We summarize these in Table 1, where we adapt the taxonomy introduced in [5].

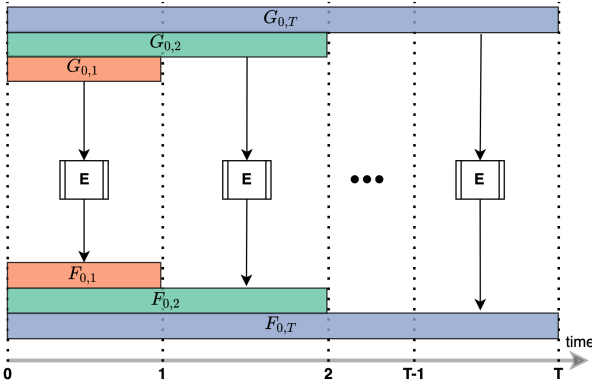### 2.1. Static network embedding

The topic is covered in various embedding method families, among which we first discuss matrix factorization based methods. **Locally Linear Embedding (LLE)** [31] and **Laplacian Eigenmaps (LE)** [1] both aim to map a high-dimensional data point space to low-dimensional one based on the neighbourhood of points (first-order proximity). LLE reconstructs a linear weight matrix, and in LE eigenvectors over graph Laplacian are computed. **Large-scale Information Network Embedding (LINE)** [33] extended these approaches by additionally preserving second-order proximities in the graph. Nodes with similar neighborhoods end up lying closer in the embedding space. **High Order Proximity preserved Embedding (HOPE)** [28] aims to sustain asymmetric proximities of nodes in the graph, in contrast to LINE, where proximities were symmetric.

The next group is methods which are based on random-walks. In **DeepWalk** [29] random-walks sampled over the graph are fed to the skip-gram model adapted from Word2Vec [24]. **Node2Vec** [16] was an improvement over DeepWalk, where authors introduced parameters p and q control both depth-first search and breadth-first search like behavior.
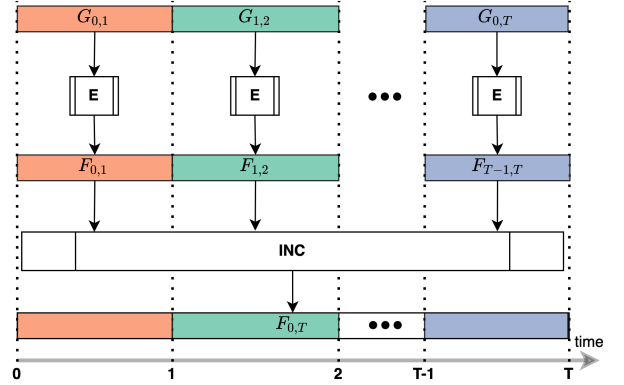
Another approach utilizes Graph Convolutional Networks architecture – **Deep Graph Infomax (DGI)** [36] is an unsupervised method, which relies on maximizing mutual information between patch representations (obtained by Graph Convolutional Network-based encoder layer), and corresponding high-level summaries of graphs (obtained by readout function). All of the approaches mentioned above are capable of processing static networks only and are not open.

2.2. Temporal and Dynamic Network Embedding

In Temporal Network Embedding methods, we aim to preserve the temporal properties of the network. On the other hand, Dynamic Network Embedding focuses on providing up-to-date embedding for evolving graphs. We can distinguish online approaches that update embedding with every new edge arrival and incremental approaches that process events in batches. The majority of the methods satisfy both of these objectives – temporal and dynamic. Here we give a brief overview of the most prominent ones. A popular trend found in the literature is to build upon the framework of random walks with the skip-gram model. **Continuous-Time Dynamic Network Embedding (CTDNE)** [27] introduced temporal walks that traverse edges according to their timestamps instead of performing random-walks statically. The original version of the method focused on the temporal aspect of embedding, while in the follow-up work, the authors introduce an online version of the algorithm [20] that produces a new portion of temporal random walk for upcoming events and then updates the model.

(a) Using a static or temporal embedding method (**E**) requires to fully retrain the method on every new batch using cumulative snapshots.

(b) Using an incremental embedding method (**INC**) it is possible to reuse the already computed embeddings of non-overlapping snapshots and combine those into an embedding which describes the full event stream history.

Figure 2: Comparison of applications of traditional static network embedding methods and incremental methods in dynamic network embedding task.

This direction is further extended in **Dynnode2vec** [22] architecture, where for each timestamp, a random-walk is sampled, only for nodes marked as evolving in terms of new edges. **Global Topology Preserving Dynamic Network Embedding (GloDyNE)** [17] also follows a similar schema with incrementally updating skip-gram model, but they differ in the method of selecting nodes to perform new random-walks. They partition the graph into k sub-networks for each timestamp, and for each sub-network, they randomly select one node based on calculated probability distribution within the sub-network. After obtaining the representative nodes list, they perform new random-walks over a new snapshot for representative nodes and update the skip-gram model. An interesting approach was introduced in **Online-Node2Vec** models [2]: StreamWalk and SecondOrder. StreamWalk follows a temporal random walk procedure like CTDNE but differs in the edge sampling scheme. In their second model SecondOrder, they use MinHash fingerprinting to approximate Jaccard node similarity instead of performing temporal random-walks that reduce the method's complexity. In **Weg2vec** [34], instead of embedding nodes, representations of events are learned by introducing a weighted neighborhood edge sampling strategy. Finally in **tNodeEmbed** [32] for each timestamp a new embedding is calculated using the Node2vec method. Embeddings are aligned between timestamps before they are fed as an input to the Long Short Term Memory (LSTM) model, which performs an end-to-end task.

In contrary to random-walk-based methods **Dyngraph2Vec** [14] extends Autoencoder (AE) architecture to capture to the evolving structure of temporal networks providing a purely neural-network-based approach. They present three variants of the model: dyngraphAE, dyngraphRNN, and dyngraphAERNN, which differ in how they represent input neighbor vector – dyngraphAE uses fully connected layers, dyngraphRNN uses LSTM layers, and DyngraphAERNN

uses fully connected layers followed by LSTM layers.

The authors of [35] propose a framework (further referred to as **LCF**) that is based on the linear combination of embeddings from consecutive snapshots. Building upon a similar paradigm, we provide an insightful framework with differences discussed in Section 4.5.

2.3. Network embedding alignment

Network embedding alignment problem arises when combining embeddings from subsequent runs or comparing representations from different graphs. [15, 7] identify cross-graph node similarities by jointly solving two optimization problems: they use the Sinkhorn algorithm to match node correspondence and find a linear transformation of one of the embeddings by solving Orthogonal Procrustes. [32] uses Orthogonal Procrustes to see a transformation matrix between embeddings of two consecutive timestamps (as we do). They use all common nodes between timestamps to form the matrix (which differs from our method). [21] uses alignment as an integral part of the model, improving the individual learning of embeddings. Embeddings are aligned at anchor nodes (that indicate the same users across two networks) and introduce soft-constraint for non-anchor nodes. Other approaches utilize generative adversarial networks [10, 43] to align embeddings. [43] solution is based on Wasserstein GAN to produce cross-lingual embedding mapping. [10] utilized GAN architecture in which they aim to obtain both sides' transformation using cycle consistency loss.

3

Table 1: Graph node embedding methods comparison. Methods marked in bold are the ones evaluated in our experiments.

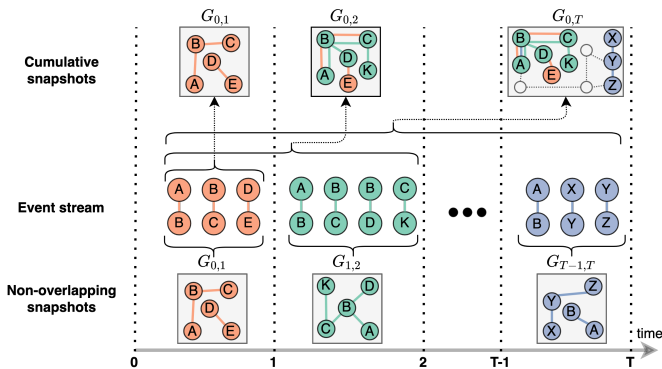| METHOD	STATIC	TEMPORAL	DYNAMIC	NETWORK ALLIGMENT	TAXONOMY
LLE'00 [31]	√	×	×	×	MATRIX FACTORIZATION
LE'03 [1]	√	×	×	×	MATRIX FACTORIZATION
LINE'15 [33]	√	×	×	×	EDGE RECONSTRUCTION
HOPE'16 [28]	√	×	×	×	MATRIX FACTORIZATION
DGI'19 [32]	√	×	×	×	GRAPH CONVOLUTIONAL NETWORKS
DEEPWALK'14 [29]	√	×	×	×	RANDOM WALK, SKIP-GRAM
NODE2VEC'16 [16]	√	×	×	×	RANDOM WALK, SKIP-GRAM
CTDNE'18, 19 [27, 20]	×	√	√	×	TEMPORAL RANDOM WALK, SKIP-GRAM
TNODEEMBED'19 [32]	×	√	√	√	NEURAL NETWORKS, RANDOM WALK, SKIP-GRAM
STREAMWALK'19 [20]	×	√	√	×	TEMPORAL RANDOM WALK
SECONDORDER'19 [20]	×	√	√	×	TEMPORAL RANDOM WALK
DYNGRAPH2VEC'20 [14]	×	√	√	×	NEURAL NETWORKS
DYNNODE2VEC'18 [22]	×	√	√	×	RANDOM WALK, SKIP-GRAM
WEG2VEC'20 [34]	×	√	×	×	TEMPORAL RANDOM WALK, SKIP-GRAM
GLODYNE'20 [17]	×	√	√	×	RANDOM WALK, SKIP-GRAM
LCF'20 [35]	×	√	√	√	DEPENDENT ON BASE METHOD
FILDNE'20 (OUR)	×	√	√	√	DEPENDENT ON BASE METHOD

Figure 3: Event streams can be saved as a series of cumulative or non-overlapping graph snapshots. The first hold the full history from the very beginning, but at the cost of a relatively high memory footprint. Contrary, the latter ones are restricted to a given time interval, hence requiring less memory.

3. Notation and problem definition

The notation introduced in these and all the following is summarized in Table 2.

Definition 1 (Static Network). A Network (Graph) is a pair $G = (V, E)$, where V is a set of vertices and $E = \{(u, v) : (u, v) \in V \times V\}$, is a set of edges connecting vertices. Both, the nodes and edges can posses assigned attributes. A special kind of vertices' attributes are timestamps, which lead to the next definition.

Definition 2 (Dynamic Network). A Dynamic Network (Graph) is a triple $G = (V, E, ts)$ where V and E are sets of vertices and edges respectively and $ts : E \to \mathcal{R}$ is a function assigning timestamp to each edge.

Working with such a network is inconvenient – whenever we want to check the state of the graph at a given time t

we have to iterate over E. The solution would be to store it as a snapshot $G_{0,t} = (V_{0,t}, E_{0,t}, ts)$, where $E_{0,t}$ is a set of edges with timestamps up to time t, while $V_{0,t}$ is the set of vertices associated with them. Further we arrange them in a sequence $[G_{0,1}, G_{0,2}, \ldots, G_{0,T}]$ of cumulative graphs, each associated with a time-index in the range $[1; T]$, where T denotes the maximal time-index. One might be interested in non-overlapping batches $[G_{0,1}, G_{1,2}, \ldots, G_{T-1,T}]$, where $G_{t,t+1}$ consists only of edges from $E_{t,t+1}$, created between t and $t + 1$.

For simplicity, we will mark snapshots with the end of interval whenever they cover a single time window, that is $G_{t-1,t} \equiv G_t$.

Dynamic Networks can be attributed in the same way as Static Networks are.

Definition 3 (Graph Stream). A Dynamic Network is defined for a limited time interval $[0; T]$ specified by the youngest snapshot's time-index. A Graph Stream expands this definition for a potentially infinite stream of events (each connecting two nodes and represented as an edge) forming an infinite sequence of cumulative graphs $[G_{0,1}, G_{0,2}, \ldots]$ or equivalently non-overlapping ones $[G_{0,1}, G_{1,2}, \ldots]$ (see Figure 3). The real-world applications of Graph Streams have to take resources limitation into account. Therefore the oldest history has to be forgotten or compressed.

A graph stream can be observed in an online (one edge at a time) or batched manner. In this paper, we would like to focus on the batched setting, remembering that the online setting can be interpreted as single-event batches.

Definition 4 (Static Network Embedding). The aim is to find a mapping $f_G : V \to R^d$, $d \ll |V|$ such that the topological (proximity or structural) similarity of vertices in a static G is preserved. The resulting embedding is marked as $F = f_G(V)$ and can be arranged as a $|V| \times d$

matrix, where each row denotes vector representation of a single node.

Definition 5 (Temporal Network Embedding). The aim is to find a mapping $f_{G_{0,T}} : V_{0,T} \to R^d$, $d \ll |V|$ such that the temporal topological [23] (proximity or structural) similarity of vertices in a dynamic $G_{0,T}$ is preserved. The resulting embedding is marked as $F_{0,T} = f_{G_{0,T}}(V_{0,T})$ and can be arranged as a $|V_{0,T}| \times d$ matrix, where each row denotes vector representation of a single node.

Definition 6 (Dynamic Network Embedding). As the network evolves one may be interested in evolving network embedding. We can distinguish two approaches – naive and incremental one (see Figure 2). In the latter setting, we reuse previously computed embeddings $(F_{t-1}, f^{t-1}, \ldots, F_1, f^1)$ to obtain a representation $F_{0,t}$ updated with the most recent snapshot G_t, i.e. $F_{0,t} = f^t(G_t, G_{t-1}, F_{t-1}, f^{t-1}, \ldots, F_1, f^1)$. The motivation for incremental paradigm is to reduce computational cost of naive approach by updating nodes' embeddings. The training objective is preservation of the topological properties in $G_{0,t}$.

Definition 7 (Matrix alignment). This can be seen as an instance of the <u>orthogonal Procrustes</u> problem [7]. Given matrices $A \in R^{n \times d}$ and $B \in R^{n \times d}$ with matching rows, we are interested in finding transformation matrix Q that satisfies

$$\underset{Q : Q^\intercal Q = I}{\operatorname{argmin}} ||BQ - A||_2^2. \tag{1}$$

The solution is easily found as $Q^* = UW^\intercal$, where $U\Sigma W^\intercal$ is the Singular Value Decomposition (SVD) of $B^\intercal A$.

Remark. For the completeness of the discussion on evolving networks, one should also consider such aspects as time attributes in the form of intervals, non-time attributes changing in time, and nodes appearing without an edge. However, these considerations go beyond the proposed method's scope, and we leave them as challenges for subsequent research.

4. Proposed Framework for Incremental Learning of Dynamic Networks Embeddings

The goal of Framework for Incremental Learning of Dynamic Networks Embeddings (FILDNE) is to find the embedding $F_{0,t}$ of the full graph $G_{0,t}$, without calculating it from all source data from period $[0, t]$, but based on already computed embeddings on historical data, i.e. $(F_{t-1}, \ldots, F_1)$ and the most recent snapshots G_{t-1} and G_t. Let us note that our method does not require the mapping functions $(f^{t-1}, \ldots, f^1)$. FILDNE consists of 3 consecutive steps that are repeated with each new data portion arrival $G_{t-1,t}$. The methods come in two versions – FILDNE and k-FILDNE — therefore, step 3. has two variants. The difference between FILDNE and k-FILDNE method is that the former works in a pairwise manner $F_{0,t} = \text{FILDNE}(G_t, G_{t-1}, F_{t-1})$, while the latter operates on a vector of past embeddings $F_{0,t} = \text{k-FILDNE}(G_t, G_{t-1}, F_{t-1}, \ldots, F_{t-k})$, where k is a parameter of the method.
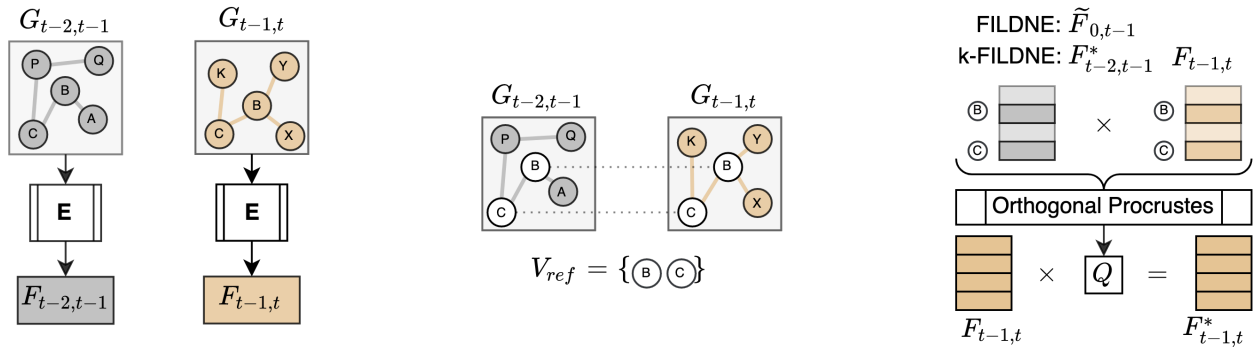
Table 2: Symbols and notations

Symbol	Object
G	network / graph
G_t	snapshot of dynamic graph with nodes and edges occurring between $t-1$ and t
G_{t_1,t_2}	snapshot of dynamic graph with nodes and edges occurring between t_1 and t_2
V	set of nodes
V_{t_1,t_2}	set of nodes in dynamic graph G_{t_1,t_2}
$V_{t-1 \cap t}$	set of common nodes between G_{t-1} and G_t
V_{ref}	set of reference nodes
E	set of edges
E_{t_1,t_2}	set of edges in dynamic graph G_{t_1,t_2}
T	maximal time-index in Dynamic Network
F	embedding matrix
$F(V')$	embedding matrix of a subset of nodes $V' \subseteq V$
F_t	embedding matrix for graph $G_{t-1,t}$
F_{t_1,t_2}	embedding matrix for graph G_{t_1,t_2}
F_{t_1,t_2}^*	aligned embedding for graph G_{t_1,t_2}
$\boldsymbol{F}_{k|t}^*$	vector of k aligned embeddings up to time t
$\tilde{F}_{0,t}$	embedding matrix generated by FILDNE
α	convex combination weight in FILDNE
$\boldsymbol{\alpha}$	vector of k convex combinations weights in k-FILDNE
$a(\cdot)$	activity function
$a_t^{(v)}$	v's activity in G_t
$s(\cdot, \cdot)$	scoring function
S	scores of common nodes $V_{t-1 \cap t}$

Step 1. Batch embedding. For each new snapshot $G_{t-1,t} \equiv G_t$, we apply some network embedding technique – called <u>Base Method</u> – and obtain $F_{t-1,t} \equiv F_t$ (see Figure 4a). The choice of the Base Method is entirely up to the user. It can be an instance of Static Network Embedding (e.g. node2vec, LINE, DGI) and Temporal Network Embedding (e.g. CTDNE), both handling additional node/edge attributes if such are observed.

We focus on transductive Base methods, for which inferring representation for unseen examples is not possible in no other way than a full re-training of the model. One could also use inductive graph embedding methods (such as GraphSAGE or GCN-based approaches), but these do not suit well to our problem setting (our framework if fact provides a way to inductive learning itself).

Step 2. Alignment. The random initialization and stochastic optimization adopted in embedding methods result in non-comparable realizations of representation vectors even for the same input data. The same applies when matching embeddings between two time steps, forcing us to run network embedding alignment before we go to the next step. The problem appears to be straightforward – arranging embeddings in the form of matrices with matching rows lets us apply matrix alignment techniques. However, the

(a) Batch embedding. Using Base method $\mathbf{E}$, the first step is to compute the embedding $F_{t-1,t}$ for the most recent non-overlapping snapshot.

(b) Reference node selection. Based on node activities $a(\cdot)$ from subsequent snapshots, a scoring function $s(\cdot, \cdot)$ is applied. A given node selection scheme uses those node scores, to obtain the reference nodes set V_{ref}.

(c) Embedding alignment. Solving the Orthogonal Procrustes problem for embeddings of reference nodes allows to obtain a transformation matrix Q, which is multiplied by $F_{t-1,t}$. FILDNE uses matrix $F_{0,t-1}(V_{ref})$ and k-FILDNE $F_{0,t-1}(V_{ref})$

(d) Single composition step of FILDNE. The previously composed embedding matrix $\tilde{F}_{0,t-1}$ is combined with the aligned embedding matrix $F_{t-1,t}^*$ using parameter α.

(e) Single composition step of k-FILDNE. The algorithm obtains the embedding $\tilde{F}_{0,t}$ by computing the dot product between the parameter vector $\boldsymbol{\alpha}$ and the vector of k aligned embeddings up to time t $\boldsymbol{F}_{k|t}^*$.

Figure 4: Overview of the proposed FILDNE algorithm in both variants.

nature of the problem to be solved should not be forgotten. Network evolution is not only about the appearance of new nodes but also about change in neighborhoods of existing ones. We are not interested in vertices, whose structural neighborhood topology was completely altered. Therefore, not all vectors should be considered when aligning embedding matrices. We introduce the concept of <u>reference nodes</u> (later discussed in Section 4.1) or anchor nodes [21] marked as V_{ref}. They are supposed to be (relatively) static over two neighboring batches. Having such a reference we can calculate the transformation matrix Q(as described in Definition 7) only on vectors representing those nodes and then apply in on the entire embedding matrix F_t what results in F_t^* (see Algorithm 1 and Figure 4c).

In case of the k-FILDNE method F_t is aligned to the previous embedding F_{t-1}^* which has been aligned in the former iteration. We store a vector of k aligned embeddings $\boldsymbol{F}_{k|t}^*$.

Algorithm 1: Embedding Alignment

Input: $F_{t-1}, F_t, \{(a_{t-1}^{(v)}, a_t^{(v)}) : v \in V_{t-1 \cap t}\}$,
      $\text{select}(\cdot, \cdot)$

1   $S = \{s(a_{t-1}^{(v)}, a_t^{(v)}) : v \in V_{t-1 \cap t}\}$ ▷ Calculate node scores
2   $V_{\text{ref}} = \text{select}(S, V_{t-1 \cap t})$ ▷ Obtain reference nodes
3   $U \Sigma W^\intercal = \text{SVD}(F_{t-1}(V_{\text{ref}})^\intercal F_t(V_{\text{ref}}))$
4   $Q = UW^\intercal$ ▷ Compute transformation matrix
5   **return** $\underline{F_t Q}$

Step 3. Embedding composition. In the FILDNE method (see Algorithm 2 and Figure 4d), at each iteration, we combine the previously composed embedding $\tilde{F}_{0,t-1}$ with its aligned embedding of the current snapshot $F_{t-1,t}^*$ using following <u>convex combination</u>:

$$\tilde{F}_{0,t} = \alpha \tilde{F}_{0,t-1} + (1 - \alpha) F_{t-1,t}^* \qquad (2)$$

At the beginning ($t = 2$) we use $\tilde{F}_{0,t-1} = F_{0,1}$. The α parameter can be selected using search methods or experts' knowledge. Whenever $\alpha > 0.5$, it means that the past embedding is more important than the recent data increment in the Graph Stream.

Algorithm 2: FILDNE

Input: $\tilde{F}_{0,t-1}, F_t, \{(a_{t-1}^{(v)}, a_t^{(v)}) : v \in V_{t-1 \cap t}\}$,
      $\text{select}(\cdot, \cdot), \alpha$

1   $F_t^* = \text{Embedding Alignment}(\tilde{F}_{0,t-1}, F_t, a_{t-1}, a_t, \text{select})$
2   $\tilde{F}_{0,t} = \alpha \tilde{F}_{0,t-1} + (1 - \alpha) F_t^*$
3   **return** $\underline{\tilde{F}_{0,t}}$

The k-FILDNE version (see Algorithm 3 and Figure 4e) builds upon two parameters, that is $k \in \mathbb{N}$, the number of

last embeddings combined by the algorithm, and $\boldsymbol{\alpha} \in \mathbb{R}^k$, a k-dimensional real-valued vector:

$$\boldsymbol{\alpha} = [\alpha_1, ..., \alpha_k]^\intercal \qquad (3)$$

which is constrained to $\{\boldsymbol{\alpha} : 0 \le \alpha_i \le 1, \sum_{i=1}^k \alpha_i = 1\}$ namely the $k - 1$-dimensional simplex. We propose a method to estimate $\boldsymbol{\alpha}$ what is described in Section 4.2. Such construction enables us to combine more than two embeddings at once, in opposite to FILDNE scenario where we always combine 2 embeddings.

Let $\boldsymbol{F}_{k|t}^*$ denote the vector of k embeddings up to time t:

$$\boldsymbol{F}_{k|t}^* = [\tilde{F}_{0,t-k+1}, F_{t-k+2}^*, ..., F_t^*]^\intercal. \qquad (4)$$

The embedding rule is defined as follows:

$$\tilde{F}_{0,t} = \boldsymbol{\alpha}^\top \cdot \boldsymbol{F}_{k|t}^* \qquad (5)$$

which is the dot product of $\boldsymbol{\alpha}$ and the sequence of aligned embeddings $\boldsymbol{F}_{k|t}^*$.

Algorithm 3: k-FILDNE

Input: $\boldsymbol{F}_{k-1|t-1}^*, F_t, \{(a_{t-1}^{(v)}, a_t^{(v)}) : v \in V_{t-1 \cap t}\}$,
      $\text{select}(\cdot, \cdot), G_t, \text{prior}$

1   $F_t^* = \text{Embedding Alignment}(F_{t-1}^*, F_t, a_{t-1}, a_t, \text{select})$
2   $\boldsymbol{F}_{k|t}^* = [\boldsymbol{F}_{k-1|t-1}^*; F_t^*]$
3   $\boldsymbol{\alpha} = \text{Alpha Estimation}(\boldsymbol{F}_{k|t}^*, G_t, \text{prior})$
4   $\tilde{F}_{0,t} = \boldsymbol{\alpha}^\top \cdot \boldsymbol{F}_{k|t}^*$
5   **return** $\underline{\tilde{F}_{0,t}}$

4.1. Reference nodes selection

To select appropriate reference nodes (see Figure 4b) we first introduce an <u>activity function</u>

$$a : V \to \mathbb{R} \qquad (6)$$

that for each node in the graph G assigns a scalar describing its behaviour. In our experiments we use <u>multi-degree</u> as the activity function. Activity is measured for common vertices $V_{t-1 \cap t}$ between two neighbouring snapshots.

The next step is to obtain a ranking of nodes best suited as a reference. To do so we apply a <u>scoring function</u> – in our case:

$$s(a_{t-1}^{(v)}, a_t^{(v)}) = |a_{t-1}^{(v)} - a_t^{(v)}| \left(\frac{\pi}{2} - \arctan(\max\{a_{t-1}^{(v)}, a_t^{(v)}\}) \right), \qquad (7)$$

where $a_{t-1}^{(v)}$ and $a_t^{(v)}$ are v's activities from neighboring snapshots. The resulting scores are **sorted in ascending order**. Finally, we are able to select a number of reference nodes based on the ranking. We propose the following schemes:

- **Percent** – based on the lowest score the top **p** percent of nodes is selected:

$$select(S, V) = V_{ref} \subseteq \text{sort}_S(V), \text{ s.t. } |V_{ref}| = \mathbf{p}|V|$$

- **Multiplier** – based on the lowest score the number of nodes is determined as **Md** (multiplier times the embedding size), but no more than $\mathbf{p_{max}}$ (maximum percent) of common vertices:

$$\text{select}(S, V) = V_{ref} \subseteq \text{sort}_S(V), \text{ s.t.}$$
$$|V_{ref}| = \min(\mathbf{Md}, \mathbf{p_{max}}|V|)$$

- **Threshold** – all nodes with score lower than a given threshold **th** are selected as reference:

$$\text{select}(S, V) = V_{ref} \subseteq V, \text{ s.t. } \forall_{v \in V_{ref}} S^{(v)} \leq \mathbf{th}$$

The methodology of determining reference nodes presented in this section is a generic solution. Other possible approaches include the usage of nodes' attributes or experts' knowledge.

4.2. Alpha estimation

A significant problem arises while using the k-FILDNE model with k parameters. Assuming that each parameter has the same number of considered values $|\Lambda|$, the model has a search space of size $O(|\Lambda|^k)$, i.e., it grows exponentially with each new dimension. Hence, we need to find a way to estimate the model parameters using an algorithm that is cheaper than a full search over the entire parameter space. We propose an algorithm that uses Bayesian inference with assumption of Dirichlet-Multinomial distribution (see Algorithm 4 and Figure 5). We want to estimate the parameters $\hat{\boldsymbol{\alpha}} = \{\hat{\alpha}_1, \ldots, \hat{\alpha}_k\}$. For the prior we use the Dirichlet distribution:

$$Dir(\boldsymbol{\alpha}|\boldsymbol{\beta}) = \frac{1}{B(\boldsymbol{\beta})} \prod_{i=1}^{k} \alpha_i^{\beta_i-1} \mathbb{1}_{\{S_{k-1}\}}(\boldsymbol{\alpha}), \qquad (8)$$

where $B(\cdot)$ is the beta function used for normalization purposes. $\boldsymbol{\beta}$ reflects the prior knowledge about the distribution and S_{k-1} denotes the $k-1$–dimensional simplex. We define two settings – <u>uniform</u>: $\beta_1 = \beta_2 = \ldots = \beta_k = 1$, where representations are equally important, and <u>increasing</u>: $\beta_1 < \beta_2 < \ldots < \beta_k$, where more recent embeddings are assumed to be more significant.

The likelihood function has the form:

$$p(\boldsymbol{D}|\boldsymbol{\alpha}) = \prod_{i=1}^{k} \alpha_i^{N_i} \qquad (9)$$

where $\boldsymbol{D} = [N_1, N_2, ..., N_k]$ is the vector of class occurrences from the link prediction experiment described below. Note that $N = \sum_{i=1}^{k} N_i$ is the total size of the sample.

Each of the embeddings $\tilde{F}_{0,t-k+1}, F_{t-k+2}^*, \ldots, F_t^*$ (see Equation 4) is a separate class in the Multinomial distribution. We take edges from the most recent snapshot G_t and split them into the train and test sets. Negative edges are sampled in both groups in numbers enabling class balance.

We fit a set of Logistic Regression classifiers with input vectors for each edge, built as Hadamard product of node embeddings, and the outputs denoting edge existence. If a link was correctly predicted with several embeddings, we randomly choose only a single representation. If none of them can provide the correct classification of the link, such an edge is removed from the sample. The class counts (correct predictions) $[N_1, N_2, ..., N_k]$ are measured on the test set.

Using the above assumptions we estimate the parameters $\hat{\boldsymbol{\alpha}}$ as the maximum a posteriori probability (MAP) of the Dirichlet-Multinomial model [25]:

$$\hat{\alpha}_j = \frac{N_j + \beta_j - 1}{N + \sum_{i=1}^{k} \beta_i - k} \qquad (10)$$

Algorithm 4: Alpha Estimation

Input: $\boldsymbol{F}_{k|t}^*, G_t$, prior

1 Generate link prediction dataset over graph G_t
2 Fit Logistic Regression classifiers on train set
3 Evaluate link prediction on test set and report correct predictions $[N_1, N_2, \ldots, N_k]$
4 Set $\boldsymbol{\beta}$ according to *prior* distribution
5 $\hat{\boldsymbol{\alpha}} = \left[\dfrac{N_1 + \beta_1 - 1}{N + \sum_{i=1}^{k} \beta_i - k}, \ldots, \dfrac{N_k + \beta_k - 1}{N + \sum_{i=1}^{k} \beta_i - k} \right]$
6 **return** $\hat{\boldsymbol{\alpha}}$

4.3. Missing embeddings: new and inactive nodes

In the FILDNE algorithm, the way we establish the embedding for new or disappearing nodes is trivial. In such a case at time t, we only have one out of two embeddings, and Eq. 2 cannot be applied. For new (previously unseen) nodes v, we use the its embedding from the most recent snapshot as the estimated embedding, i.e. $\tilde{F}_{0,t}(v) = F_{t-1,t}^*(v)$. Contrary, if a node does not appear in most recent graph snapshot, we use its previously estimated embedding, i.e. $\tilde{F}_{0,t}(v) = F_{0,t-1}^*(v)$. For the k-FILDNE method, the problem is more complex. The number of input embeddings k' varies between 1 and k. If there are all embeddings available, we simply apply Eq. 5, and if $k' = 1$, then we proceed with the same idea as for the FILDNE approach. Otherwise, we take the estimated $\hat{\boldsymbol{\alpha}}$ coefficients (see Section 4.2) corresponding to all k' available embeddings for a given node. Next, we normalize those values, so that they sum up to 1. Then, we apply Eq. 5 assuming that we only have k' components.

4.4. Complexity analysis

To estimate the time and space complexity of our algorithm, we consider the scenario of a single incremental step, i.e. the moment when we have $k-1$ past embeddings $[\tilde{F}_{0,t-k+1}, F_{t-k+2}^*, ..., F_{t-1}^*]^{\intercal}$ and a new graph snapshot G_t
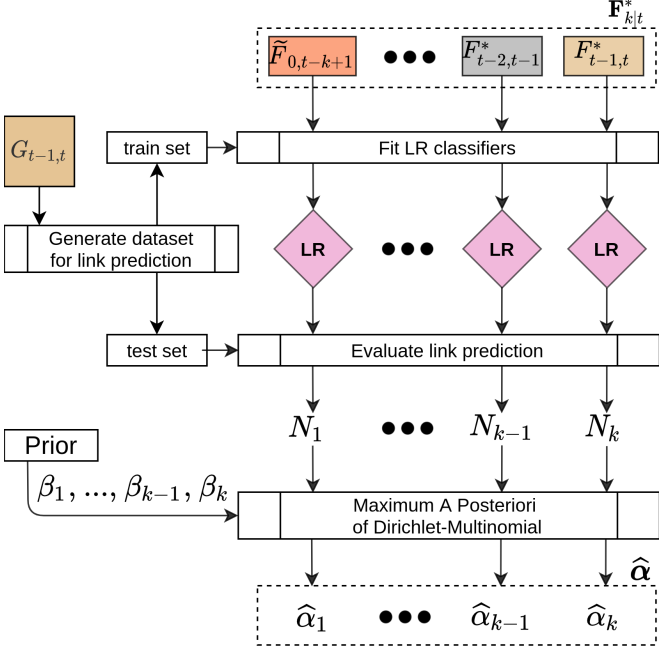
8

Figure 5: Alpha estimation. The newly arrived snapshot $G_{t-1,t}$ is prepared as train and test sets for link prediction. For each embedding in the vector $\boldsymbol{F}^*_{k|t}$ a logistic regression (LR) classifier is fitted on the train set and further evaluated on the test set. A poorly performing embedding should yield $\hat{\alpha}_i$ value that barely influence the final representation. This is achieved by Maximum A Posteriori estimation for the Dirichlet-Multinomial model fed with the classification results as described in Section 4.2.

appears. We use a random walk based embedding algorithm (Node2vec) and the k-FILDNE version. We denote $O_T(\cdot)$ and $O_S(\cdot)$ as the time and space complexities, respectively. Considering each algorithm step separately, we get:

(1) First, we run Node2vec:

$$O_T(\text{Node2vec}) = O(d|V_{t-1,t}|)$$

$$O_S(\text{Node2vec}) = O(\gamma\omega|E_{t-1,t}|),$$

where γ, ω are the number and length of random walks respectively.

(2) We obtain reference nodes by sorting all vertices $V_{t-1\cap t}$ by their activity score; it takes:

$$O_T(\text{sort}) = O(|V_{t-1\cap t}|log|V_{t-1\cap t}|)$$

$$O_S(\text{sort}) = O(|V_{t-1\cap t}|)$$

Next, we solve the Orthogonal Procrustes (OP) problem during embedding alignment:

$$O_T(\text{OP}) = O(d^3)$$

,

$$O_S(\text{OP}) = O(d^2)$$

.

(3a) Link prediction requires sampling of non-existing edges (in the same number as existing edges; Negative Sampling) takes:

$$O_T(\text{NS}) = O_S(\text{NS}) = O(|E_{t-1,t}|)$$

We train Logistic Regression using all k embeddings:

$$O_T(\text{LR}) = O(kd^2|E_{t-1,t}|)$$

$$O_S(\text{LR}) = O(kd|E_{t-1,t}|)$$

Finally we apply LR and aggregate the Class Counts, which takes:

$$O_T(\text{CC}) = O(k*|E_{t-1,t}|)$$

$$O_S(\text{CC}) = O(k)$$

(3b) The estimation of $\boldsymbol{\alpha}$ using Dirichlet-Multinomial model is done by simple division of k numbers, so it takes:

$$O_T(\text{dirichlet}) = O_S(\text{dirichlet}) = O(k)$$

(4) for the composition of all embeddings, we perform a Weighted Matrix Addition, which takes:

$$O_T(\text{WMA}) = O_S(\text{WMA}) =$$

$$= O(d|V_{0,1}| + \ldots + d|V_{t-1,t}|) = O(d|V|)$$

where $|V|$ is the number of vertices in the whole graph.

Therefore, the time and memory complexities of FILDNE equals the sum of all above steps' complexities. The size of the last snapshot $V_{t-1,t}, E_{t-1,t}$ is smaller than of the whole graph $V_{0,t}, E_{0,t}$: $|V_{t-1,t}| < |V_{0,t}| = |V|$, $|E_{t-1,t}| < |E_{0,t}| = |E|$. Let us note that the d and k hyper-parameters do not scale with the size of the network. Hence, we can approximate the total FILDNE complexity to: $O_T(FILDNE) \approx O(|V|\log|V| + |E|)$ and $O_S(FILDNE) \approx O(|V| + |E|)$.

4.5. FILDNE versus other Dynamic Graph Embedding methods

Let us now consider the differences between different approaches of dynamic embedding presented in the literature. First off, we aim at highlighting the differences in data requirements of each approach while computing embedding for new snapshot. The most unfavorable group of methods requires to store whole graph information from the very beginning to time t, i.e. $G_{0,t}$. These are **CTDNE**, **Online-Node2Vec**, **tNodeEmbed**, **dyngraph2vec**, **Dynnode2vec**. Additionally, tNodeEmbed method requires all intermediate embeddings $F_{0,1}, \ldots, F_{0,t-1}$. Another group of methods necessitate only the most recent graph snapshots, i.e. G_{t-1}, G_t (**GloDyne**). Our approach, in opposite to all previous, requires only one embedding $\tilde{F}_{0,t-1}$ (**FILDNE**) or k embeddings $\tilde{F}_{0,t-k+1}, \ldots, F^*_{t-2}, F^*_{t-1,t}$

9

(**k-FILDNE**) and the activity of nodes from G_{t-1}, i.e. $a_{t-1}^{(v)} \forall v \in G_{t-1}$.

Second off, we want to emphasize the difference in the easiness of methods' hyperparameters tuning. Such dynamic network embedding methods as **dyngraph2vec**, **tNodeEmbed** strongly rely on deep network architecture and all related optimization problems. **Online-Node2Vec** requires to specify time-dependent hyperparameters that are not very intuitive: time-decay and half-life, which must be set by an expert or through extensive searching. Our approach limits the number of hyperparameters to only two simple and intuitive: the number of reference nodes, combination weight α (FILDNE), or *prior* (k-FILDNE).

Most importantly, third off, our approach allows utilizing embeddings of choice (methods or already calculated vectors) in contrast to all other methods, which rely on specific embedding objectives. It enables our method to use additional network-related data, e.g., nodes' attributes (using, e.g., **DGI**).

The **LCF** method proposed in [35] does not fit to any of the above-mentioned criteria. The authors propose a similar framework to our FILDNE method. However, there are some non-negligible differences. In contrast to them, we employ a convex combination, a special case of the linear combination. Such an approach allows us to keep a nearly constant order of magnitude of embeddings vectors, whereas for linear combination with all coefficients equal to one (as proposed in [35]), the magnitude depends on the number of combined embeddings. Further, they introduce an exponential decay based approach where the most recent representation prevails the final embedding. Such an assumption is not flexible, whereas, in our proposed FILDNE methods, the combination parameters α are dynamically estimated from the data. Moreover, our experiments show that the assumption of exponential decay is not satisfied in the real world temporal networks (See Fig. 10). The authors do not neglect the fact that proximity-based embeddings are not comparable across timestamps and therefore apply embedding alignment. However, they do it naively by taking all the available reference nodes between snapshots. Our approach measures the stability of nodes' activities to select the most appropriate ones. Finally, the reported results are not compared to the other state-of-the-art Dynamic Graph Embedding methods. Furthermore, not commonly used accuracy metrics makes it impossible to compare with them directly.

5. Experiments

We evaluate the performance of our proposed algorithm in several experiments and compare the results against commonly used baseline methods as well as the Base methods in naive Dynamic Graph Embedding setting. Firstly, we train all methods and use the computed embeddings in a link prediction experiment, which should check whether these vector representations encode structural graph information, which can be used to distinguish connected nodes from non-connected ones. Next, for two datasets with edge labels, we use the same embeddings to perform edge classification – we measure the classification quality. We would like to preserve the distances between nodes in the resulting embedding space when learning representations on graphs. Hence, we compute metrics used in graph reconstruction (distortion, mAP). Finally, we perform a hyperparameter sensitivity study to show how they influence the model's performance.

5.1. Datasets

We conduct experiments on seven popular dynamic graph datasets downloaded from the Network Repository [30] – we summarize their statistics in Table 3. Selected datasets vary in the total time span between the first and last event (edges): from 2 days (*hypertext09*) up to about 5 years (*bitcoin-otc*). Three of them contain directed edges. Each dataset is used in link prediction and graph reconstruction tasks. Availability of edge labels in *bitcoin-alpha* and *bitcoin-otc* datasets allows us to perform edge classification. In preliminary experiments, we have checked several sizes of node embeddings for all networks and selected the best ones for each one (see Table 3). We note the differences in the characteristics of selected datasets. The two bitcoin networks exhibit a meager amount of intersecting nodes and edges across pairs of consecutive snapshots when compared to the others. The fb-forum, fb-messages, and hypertext09 graphs stay at a moderate level of variability, while enron-employees and radoslaw-email seem stable.

5.2. Experimental setup

In the following paragraphs, we explain how we have configured the experimental environment.

Base methods. We evaluate several state-of-the-art and representative node embedding algorithms from different method families, i.e. based on: random walks (**Node2vec** [16], **CTDNE** [27]), matrix factorization (**HOPE** [28], **LLE** [31], **LE** [1]), graph neural networks (**DGI** [36]) and general function optimization (**LINE** [33]).

These methods do not provide the ability to perform training incrementally. We use these methods to compute two kinds of embeddings:

- baseline embeddings to compare our proposed FILDNE method and other streaming/incremental ones against, i.e., the node representations are computed on the cumulative snapshots $G_{0,t}$ at a given time t;

- embeddings for non-overlapping graphs $G_{t,t+1}$, which are further consumed by our proposed FILDNE method.

Streaming methods. In terms of other incremental methods designed for graph streams embedding, we compare our method to: **tNodeEmbed** [32], two variants of **Online-Node2vec** [20] (**StreamWalk**, **SecondOrder**) and **dyngraph2vec** [14] (in the **AE-RNN** version).

Table 3: Statistics of graph datasets. $|\mathbf{V}|$ - no. of nodes, $|\mathbf{E}|$ - no. of temporal edges (events), $\mathbf{D}$ - density of the graph, **Directed** - is the graph directed or not. **LP, EC, GR** - dataset was used in **L**ink **P**rediction, **E**dge **C**lassification, **G**raph **R**econstruction tasks, respectively. **Avg. Jaccard Index** of nodes and edges computed as the mean of respective Jaccard Indexes across all consecutive snapshot pairs.

DATASET	$|V|$	$|E|$	D	TIMESPAN (DAYS)	DIRECTED	TASKS	EMBEDDING SIZE	AVG. JACCARD INDEX NODES	EDGES
ENRON-EMPLOYEES	151	50 572	4.466	1 138	$\times$	LP, GR	32	0.864	0.361
HYPERTEXT09	113	20 818	3.290	2	$\times$	LP, GR	32	0.816	0.142
RADOSLAW-EMAIL	167	82 927	2.991	271	$\checkmark$	LP, GR	32	0.903	0.430
FB-FORUM	899	33 720	0.084	164	$\times$	LP, GR	128	0.692	0.253
FB-MESSAGES	1 899	61 734	0.034	216	$\times$	LP, GR	128	0.516	0.093
BITCOIN-ALPHA	3 783	24 186	0.002	1 901	$\checkmark$	EC, LP, GR	32	0.217	0
BITCOIN-OTC	5 881	35 592	0.001	1 903	$\checkmark$	EC, LP, GR	128	0.213	0

Convergence issues. During the experiments, we found out that the LLE [31] method did not converge on three datasets (fb-messages, bitcoin-alpha, bitcoin-otc). We checked several hyperparameter settings of the underlying optimizer, but none of them fixed this issue. We decided not to report the results of this method for these datasets.

Data preprocessing. For each dataset, we apply the following preprocessing steps: (1) we take all edges and sort them by time in an ascending manner, (2) we split these edges into ten equally sized parts according to time, (3) we convert each edges chunk into a graph snapshot and call it $G_{t,t+1}$, where $t \in \{0, \ldots, 9\}$. In total, we obtain 10 non-overlapping graph snapshots. We also save cumulative graphs which accumulate all edges from the beginning, i.e. $G_{0,t}$, which contain all edges from snapshots $G_{0,1}, G_{1,2}, \ldots, G_{t-1,t}$.

Note that tNodeEmbed updates its internal model for every single timestamp. For the method to be comparable in our experimental scenario, we assume that each of the ten snapshots, as mentioned earlier, is equivalent to a single timestamp that is processed by tNodeEmbed. For this method, we also reuse already calculated Node2vec embeddings.

Embedding calculation. We train all of the above-mentioned methods and obtain two groups of embeddings: (1) *cumulative* $F_{0,t}$ for $t \in \{0, \ldots, 9\}$, which are computed in all $G_{0,t}$ using the *Base* and *Streaming methods*; (2) *non-overlapping* $F_{t,t+1}$ using the Base methods and further combined by our proposed FILDNE methods to obtain *cumulative* embeddings $\tilde{F}_{0,t}$. Unsupervised embeddings (all but tNodeEmbed) are trained once and used in all downstream tasks. tNodeEmbed is a supervised method, and the embeddings are trained for link prediction and edge classification individually. We repeat the training and evaluation procedure 30 times, reporting averaged statistics, to address the random initialization and stochastic optimization used in the methods (e.g., random walk generation in Node2vec and CTDNE).

Mean ranks. For each snapshot and dataset, we establish a ranking of methods based on the average of 30 runs. We summarise it as mean rank, which is the average ranking of methods. We report this score in each results table in the form of a separate column. Based on this score, we mark the three best methods in bold.

Fine-tuning. To provide a fair comparison of all methods, we decided to perform a hyperparameter search, with an equal number of 100 iterations, overall methods. We use Tree of Parzen Estimators (TPE) [4] (implemented in the HyperOPT [3] package) choosing most appropriate hyperparameters.

Reproducibility. To allow other researchers and developers to try out our proposed FILDNE model, we make our code available at `https://gitlab.com/fildne/fildne`. We also publish all experiments in the form of a DVC pipeline [19], so they can be easily reproduced.

Figure 6: Evaluation protocol. Link prediction (LP) and edge classification (EC) tasks are evaluated on the **next** snapshot $G_{t,t+1}$, whereas graph reconstruction (GR) task is evaluated on the **corresponding** graph snapshot $G_{0,t}$.

5.3. Link prediction

Setup. For each graph snapshot $G_{t,t+1}$, we generate a link prediction dataset. We split existing edges into a train (75%) and test (25%) dataset and mark as class 1. Then we sample the same number of non-existing edges from the graph (negative samples – class 0).

We implement **next snapshot prediction** evaluation scheme. The embedding methods are trained on cumulative snapshots, i.e. $G_{0,1}, G_{0,2}, \ldots, G_{0,t}$ and on non-overlapping ones, i.e. $G_{0,1}, G_{1,2}, \ldots, G_{t-1,t}$. The latter embeddings

are further combined by our proposed FILDNE method to obtain $\tilde{F}_{0,t}$). For each embedding $F_{0,t}$, we evaluate it on the **next** snapshot $G_{t,t+1}$ (see Figure 6). We follow well established protocol to provide link-prediction by means of classification if an edge exists [16]. Thus, we combine pair of node embeddings into edge features using **Hadamard** operator and feed **logistic regression** classifier.

Note that the Base methods do not provide any mechanism for obtaining embeddings for previously unseen nodes. We modify the sampling procedures mentioned above to consider edges (and negative edges) with nodes present in the appropriate embedding matrix.

Results. Experiments on the cumulative snapshots with Base methods are presented in Table 4. We see that LINE and Node2vec achieve the best results on all datasets. Based on the mean rank, we observe that the **3 best methods (LINE, Node2vec, CTDNE)** outperform other methods. Hence, in the other part of this article, we will focus only on these – using them as Base methods for our proposed FILDNE algorithm. Although, if the reader is interested in full results, we provide those in the Appendix B.

Based on the mean rank, we see in Table 5 that among the three best methods, there are two embeddings composed with FILDNE. The results clearly show that our method beats other streaming approaches in all cases besides the bitcoin-alpha dataset. Considering the three best methods, we observe that for fb-messages, enron-employees, and hypertext09, there are two FILDNE-based representations, whereas for bitcoin-otc and radoslaw-email our proposed method is placed in all three of them.

We notice that in the LINE Base method on bitcoin-otc and radoslaw-email, both our FILDNE methods result in high-performance gain compared to the vanilla Base algorithm. For other methods and datasets, we see comparable results. In general, we see that the gap between k-FILDNE and FILDNE is small.

5.4. Edge classification

Setup. This downstream task is defined similarly to the link prediction setup (see Figure 6); however, the dataset is built differently. We do not need to sample negative instances. We use the bitcoin-alpha and bitcoin-otc graphs, where each edge has a label assigned (besides the timestamp) – it represents the trust value of a transaction (values range from -10 to 10). We choose a threshold of 0 to define two classes (values below 0 – class 0, negative trust, untrusted; values equal or higher than 0 – class 1, positive trust). We use the same edge embeddings as in the link prediction task (all methods but tNodeEmbed, which is retrained in a supervised way for this problem explicitly) to train a logistic regression classifier. As the resulting datasets are mostly imbalanced, we set the "class_weight" argument to "balanced" to let the algorithm automatically determine appropriate class weights (we use the Scikit-learn implementation).

Results. We report results of edge classification in Table 6. The tNodeEmbed method outperforms others, but it is re-fitted in the same task, while the others reuse previously trained embeddings. Nevertheless, among the three best results based on the mean rank, two are obtained by FILDNE. For the bitcoin-otc dataset, we see that FILDNE improves AUC of edge classification for all Base methods. Analogously to link prediction (see Section 5.3), we see that FILDNE and k-FILDNE perform similarly as well as our method beats other streaming approaches (except tNodeEmbed).

Table 6: Comparison of FILDNE and other methods in edge classification task (AUC [%]). The presented values are the mean AUC scores over 8 graph snapshots and 30 methods' retrains. We also report the mean ranks of all methods in this experiment. Methods marked in bold are the 3 best methods based on the mean rank. Underlined values show the highest AUC score for each dataset individually.

	BITCOIN ALPHA	BITCOIN OTC	MEAN RANK
N2V	65.11	60.92	7.25
LINE	63.09	64.39	6.44
CTDNE	63.81	55.92	8.56
FILDNE(N2V)	61.57	65.47	6.56
FILDNE(LINE)	63.91	66.56	**5.00**
FILDNE(CTDNE)	59.28	57.85	8.56
K-FILDNE(N2V)	64.10	66.23	5.25
K-FILDNE(LINE)	63.92	66.57	**4.94**
K-FILDNE(CTDNE)	58.25	59.84	8.06
DYNGRAPH2VEC(AERNN)	56.40	56.14	10.12
TNODEEMBED	<u>71.88</u>	<u>69.32</u>	**3.12**
ONLINE-N2V(STREAMWALK)	56.73	56.70	9.25
ONLINE-N2V(SECONDORDER)	63.39	53.93	7.88

5.5. Graph reconstruction

Setup. Contrary to link prediction and edge classification tasks, we do not use here the next snapshot prediction (see Figure 6). The main goal in graph reconstruction problems is to tell how well the embedding represents the graph it was trained on, i.e. a given embedding $F_{0,t}$ is evaluated on its corresponding graph snapshot $G_{0,t}$ (in case of FILDNE we check how well the composed Base method embeddings reflect the original cumulative graph). Graphs are transformed to static ones in order to fulfill the graph reconstruction framework. We use the two metrics [9]:

- a local one – **mAP (mean Average Precision)**, which captures local graph properties; for any node and its embedding vector it checks how many of the nearest vectors (in the sense of euclidean norm) in the embedding space are actually first-order neighbours of this node:

$$mAP = \frac{1}{|V|} \sum_{v \in V} \frac{1}{\deg(v)} \sum_{i=1}^{|N_v|} \frac{|N_v \cap R_{v,w_i}|}{|R_{v,w_i}|}, \quad (11)$$

where $\deg(v)$ denotes the degree of v, $N_v = \{w_1, w_2, \ldots, w_{\deg(v)}\}$ – neighborhood of v, R_{v,w_i} – is the

Table 4: Comparison of <u>link prediction (AUC [%])</u> on cumulative graph embeddings $F_{0,t}$, $t \in \{1, \dots, 9\}$ across all Base methods. The presented values are the mean AUC over 8 snapshots $(G_{2,3}, \dots, G_{9,10})$ and 30 methods' retrains. We select the 3 best methods (LINE, Node2vec, CTDNE) based on the mean ranks and use those methods in further experiments. Underlined values show the highest AUC score for each dataset individually. Methods that did not converged are marked as "×".

	BITCOIN ALPHA	BITCOIN OTC	FB FORUM	FB MESSAGES	ENRON EMPLOYEES	HYPERTEXT09	RADOSLAW EMAIL	MEAN RANK
LINE	<u>66.08</u>	<u>54.69</u>	<u>79.03</u>	<u>70.75</u>	90.59	72.71	86.74	**1.89**
N2V	65.60	52.82	75.84	65.43	<u>92.28</u>	<u>73.76</u>	<u>90.15</u>	**1.91**
CTDNE	55.58	51.23	65.24	54.35	87.30	67.31	83.32	**3.46**
HOPE	59.16	49.71	53.72	51.16	58.79	53.89	88.71	4.64
DGI	55.53	50.26	54.34	53.09	59.76	57.23	70.88	4.91
LE	47.46	49.73	54.17	51.81	58.23	55.01	77.99	5.07
LLE	×	×	55.69	×	59.96	58.13	54.87	5.44

Table 5: Comparison of FILDNE and other methods in <u>link prediction task (AUC [%])</u>. The presented values are the mean AUC scores over 8 graph snapshots and 30 methods' retrains. We also report the mean ranks of all methods in this experiment. Methods marked in bold are the 3 best methods based on the mean rank. Underlined values show the highest AUC score for each dataset individually.

	BITCOIN ALPHA	BITCOIN OTC	FB FORUM	FB MESSAGES	ENRON EMPLOYEES	HYPERTEXT09	RADOSLAW EMAIL	MEAN RANK
N2V	65.60	52.82	75.84	65.43	92.28	73.76	90.15	**5.43**
LINE	66.08	54.69	<u>79.03</u>	70.75	90.59	72.71	86.74	5.77
CTDNE	55.58	51.23	65.24	54.35	87.30	67.31	83.32	9.61
FILDNE(N2V)	59.25	53.36	73.94	64.27	<u>92.43</u>	71.35	91.67	5.75
FILDNE(LINE)	58.28	<u>62.39</u>	74.11	<u>76.11</u>	92.00	<u>73.87</u>	93.42	**4.21**
FILDNE(CTDNE)	55.39	55.97	66.10	55.86	88.33	65.51	85.64	9.30
K-FILDNE(N2V)	62.22	53.93	75.95	65.09	91.43	71.71	91.66	5.55
K-FILDNE(LINE)	59.41	59.66	75.06	74.83	91.92	73.27	<u>93.62</u>	**4.48**
K-FILDNE(CTDNE)	53.52	53.38	61.24	57.01	87.85	64.44	85.54	10.00
DYNGRAPH2VEC(AERNN)	67.88	58.67	70.50	66.93	74.66	66.40	85.72	8.38
TNODEEMBED	68.68	48.40	55.96	60.97	83.57	55.79	76.87	9.61
ONLINE-N2V(STREAMWALK)	<u>69.59</u>	59.66	73.21	69.55	84.84	68.36	91.13	6.68
ONLINE-N2V(SECONDORDER)	68.70	58.33	69.34	75.31	87.59	72.44	89.41	6.23

smallest set of such points that contains w_i (that is, R_{v,w_i} is the smallest set of nearest points in the embedding space required to retrieve the ith neighbor of v).

- a global one – **distortion**, which compares the distances in the embedding space (euclidean norm) with the distances in the graph (shortest path lengths), the embedding distances are normalized to be within the range $[1; n]$, where n is the longest shortest path length (also called diameter of the graph).

$$D = \frac{1}{\binom{n}{2}} \left(\sum_{u,v \in U: u \neq v} \frac{|d_E(u,v) - d_G(u,v)|}{d_G(u,v)} \right), \quad (12)$$

where n is the number of nodes, d_G are the graph distances and d_E are the embedding distances.

Note that we are interested in higher mAP values and lower distortion values (with 100% and 0 as the ideal values, respectively). We also do not need an auxiliary classifier, like logistic regression for link prediction and edge classification. Using a given graph and its respective embedding, we compute the metrics.

Results. Table 7 summarizes the graph reconstruction task results as the mean Average Precision. Considering the mean ranks, we see both FILDNE and k-FILDNE models in the top three methods. One of them (k-FILDNE with Node2vec embeddings) achieves the best results for fb-forum, improving the pure Node2vec by approximately 10% percentage points. Overall, Node2vec significantly outperforms other methods on all the remaining datasets. We observe that the other streaming approaches perform poorly – they allocate themselves at the three last positions in the mean ranks.

We also examine distortion as a graph reconstruction measure (see Table 8) for which our proposed method improves the results of Base methods in most cases, or it stays at a comparable level. Contrary to mAP, the competitive streaming methods occupy two of the top three positions alongside FILDNE. Moreover, our method achieves the best performance on fb-forum and hypertext09 datasets.

5.6. Time and memory costs

Setup. In this experiment, we measured the time of computing embeddings using all of the considered methods, i.e.:

- for Base and streaming methods it is only the time needed to compute the embedding of the graph $G_{0,t}$ (either in batched or streaming manner);

- for both FILDNE and k-FILDNE we sum up the time of computing: node activities in graph $G_{t-1,t}$, embedding $F_{t-1,t}$ using a given Base method, the calibration procedure of this new embedding to previous

ones, alpha estimation (for the k-FILDNE) and the time of applying the FILDNE composition equation.

We also measure the peak (highest) memory consumption of the procedures mentioned above. Note that this also includes the reading of the graphs into memory, as well as previously saved models (for streaming methods) or calibrated embeddings and node activities (for both FILDNE methods).

The time measurements are performed while computing embedding in the main pipeline, so we have 30 measurements for each scenario. Meanwhile, the memory measurements are done in a separate branch of the pipeline, as it requires to probe the current memory usage with a relatively high frequency to obtain accurate results. It leads to a massive slowdown of the embedding algorithms, so eventually, we decided to perform five repetitions of these measurements.

For the tNodeEmbed method, we report calculation time and memory utilization measured on the link prediction task.

Results. From the time measurements reported in Table 9, it is visible that our proposed FILDNE method is the fastest for all datasets. The speedup ratio between the fastest FILDNE configuration and the next fastest competitive method ranges from approximately two to three times. Moreover, k-FILDNE is slower than the Basic version, due to the alpha estimation procedure, described in Section 4.2. However, the actual time difference is not significant – in most cases, k-FILDNE is about 1 second slower, what corresponds to a slowdown of $1\% - 10\%$. In the case of memory utilization (see Table 10), two FILDNE configurations are on average in the top three best-performing methods. It does not mean that our method always improves the Base method, especially on LINE, we can see no progress nor degradation. The rise of memory consumption is present for the hypertext09 dataset due to the small size of the network. We see that the Online-Node2vec methods are well optimized for memory, contrary to other streaming methods.

5.7. Gain results

5.7.1. Comparison to Base methods

For all of the experiments, we compare FILDNE results to the corresponding Base method, computing gain scores interpreted as a gain whenever above 1, and as a loss whenever below. To calculate these scores first, we average all 30 retrains of each method for each snapshot. Next, we divide the results of FILDNE by the ones obtained with the corresponding Base method. We compute the mean of those ratios and report it as the gain score (see Figure 7). Note that we are interested in lower values for distortion, time, and memory consumption, so we take the fraction's reciprocal. Each pair (dataset, Base method) can be viewed from the perspective of link prediction performance, edge classification performance, graph reconstruction quality,

Table 7: Comparison of FILDNE and other methods in graph reconstruction task (mAP [%]). The presented values are the mean mAP scores over 8 graph snapshots and 30 methods' retrains. We also report the mean ranks of all methods in this experiment. Methods marked in bold are the 3 best methods based on the mean rank. Underlined values show the highest mAP score for each dataset individually.

	BITCOIN ALPHA	BITCOIN OTC	FB FORUM	FB MESSAGES	ENRON EMPLOYEES	HYPERTEXT09	RADOSLAW EMAIL	MEAN RANK
N2V	65.99	70.18	23.16	67.44	70.46	55.95	42.12	**1.36**
LINE	10.83	10.81	6.42	9.90	45.35	41.90	39.75	7.52
CTDNE	35.65	47.24	7.85	22.84	55.15	54.80	31.42	4.61
FILDNE(N2V)	28.73	26.40	30.98	30.00	63.90	49.45	40.55	**3.55**
FILDNE(LINE)	14.70	18.02	12.16	7.12	41.71	38.24	38.38	7.61
FILDNE(CTDNE)	18.20	24.27	9.53	11.59	49.39	45.92	36.84	6.30
K-FILDNE(N2V)	33.46	29.30	33.17	34.66	64.21	53.05	40.92	**2.41**
K-FILDNE(LINE)	15.39	25.02	15.58	8.92	44.43	38.82	38.08	6.66
K-FILDNE(CTDNE)	16.85	27.46	13.93	20.39	55.10	48.48	37.26	4.98
DYNGRAPH2VEC(AERNN)	0.60	0.66	2.22	1.84	20.46	27.75	26.51	11.32
ONLINE-N2V(STREAMWALK)	1.08	1.83	2.22	1.25	25.85	28.03	29.60	10.70
ONLINE-N2V(SECONDORDER)	1.38	1.54	1.46	1.28	29.09	27.88	27.11	10.98

Table 8: Comparison of FILDNE and other methods in graph reconstruction task (distortion). The presented values are the mean distortion scores over 8 graph snapshots and 30 methods' retrains. We also report the mean ranks of all methods in this experiment. Methods marked in bold are the 3 best methods based on the mean rank. Underlined values show the lowest (best) distortion score for each dataset individually.

	BITCOIN ALPHA	BITCOIN OTC	FB FORUM	FB MESSAGES	ENRON EMPLOYEES	HYPERTEXT09	RADOSLAW EMAIL	MEAN RANK
N2V	0.81	1.29	0.61	0.93	0.66	0.38	0.57	8.41
LINE	0.87	0.94	0.50	0.59	0.65	0.33	0.80	7.52
CTDNE	0.93	1.70	0.76	0.89	0.81	0.36	0.65	10.12
FILDNE(N2V)	0.70	1.05	0.47	0.74	0.69	0.30	0.54	6.21
FILDNE(LINE)	0.65	0.65	0.41	0.54	0.67	0.31	0.70	5.43
FILDNE(CTDNE)	0.85	1.70	0.65	0.93	0.84	0.32	0.66	9.45
K-FILDNE(N2V)	0.64	1.04	0.44	0.72	0.63	0.30	0.67	5.59
K-FILDNE(LINE)	0.65	0.73	0.33	0.50	0.66	0.31	0.74	**5.05**
K-FILDNE(CTDNE)	0.83	1.65	0.46	0.75	0.68	0.30	0.66	6.84
DYNGRAPH2VEC(AERNN)	0.52	0.53	0.44	0.50	0.62	0.40	0.45	**4.20**
ONLINE-N2V(STREAMWALK)	0.64	0.80	0.52	0.53	0.47	0.39	0.56	5.39
ONLINE-N2V(SECONDORDER)	0.54	0.57	0.48	0.49	0.49	0.40	0.45	**3.79**

Table 9: Comparison of FILDNE and other methods in embeddings calculation time [s]. The presented values are the mean calculation time over 8 graph snapshots and 30 methods' retrains. We also report the mean ranks of all methods in this experiment. Methods marked in bold are the 3 best methods based on the mean rank. Underlined values show the lowest calculation time for each dataset individually.

	BITCOIN ALPHA	BITCOIN OTC	FB FORUM	FB MESSAGES	ENRON EMPLOYEES	HYPERTEXT09	RADOSLAW EMAIL	MEAN RANK
N2V	46.71	133.83	24.94	75.72	4.40	9.52	11.90	7.20
LINE	70.78	63.10	103.97	114.16	114.84	105.17	58.56	11.79
CTDNE	16.25	61.41	3.98	77.44	16.82	4.39	2.65	5.27
FILDNE(N2V)	7.87	22.26	8.44	27.62	2.35	4.79	4.65	**2.75**
FILDNE(LINE)	42.34	41.66	80.53	82.18	84.43	83.29	43.50	9.07
FILDNE(CTDNE)	7.84	18.11	1.91	33.67	8.69	2.63	0.68	**1.98**
K-FILDNE(N2V)	8.61	24.04	9.90	30.69	3.71	5.31	6.40	3.93
K-FILDNE(LINE)	43.10	43.20	81.74	85.02	85.67	83.81	45.09	10.11
K-FILDNE(CTDNE)	8.59	19.88	3.33	36.37	9.96	3.15	2.71	**3.20**
DYNGRAPH2VEC(AERNN)	144.71	341.14	25.54	84.10	15.89	13.35	15.47	7.82
TNODEEMBED	52.46	143.07	31.32	84.88	24.96	16.90	24.85	9.48
ONLINE-N2V(STREAMWALK)	67.76	112.97	27.44	52.09	130.27	12.50	212.59	9.43
ONLINE-N2V(SECONDORDER)	20.78	72.98	70.51	74.15	76.83	20.92	86.89	8.98

Table 10: Comparison of FILDNE and other methods in <u>max. memory utilization [MB]</u>. The presented values are the mean max. memory utilization during embeddings calculation over 8 graph snapshots and 30 methods' retrains. We also report the mean ranks of all methods in this experiment. Methods marked in bold are the 3 best methods based on the mean rank. Underlined values show the lowest memory utilization for each dataset individually.

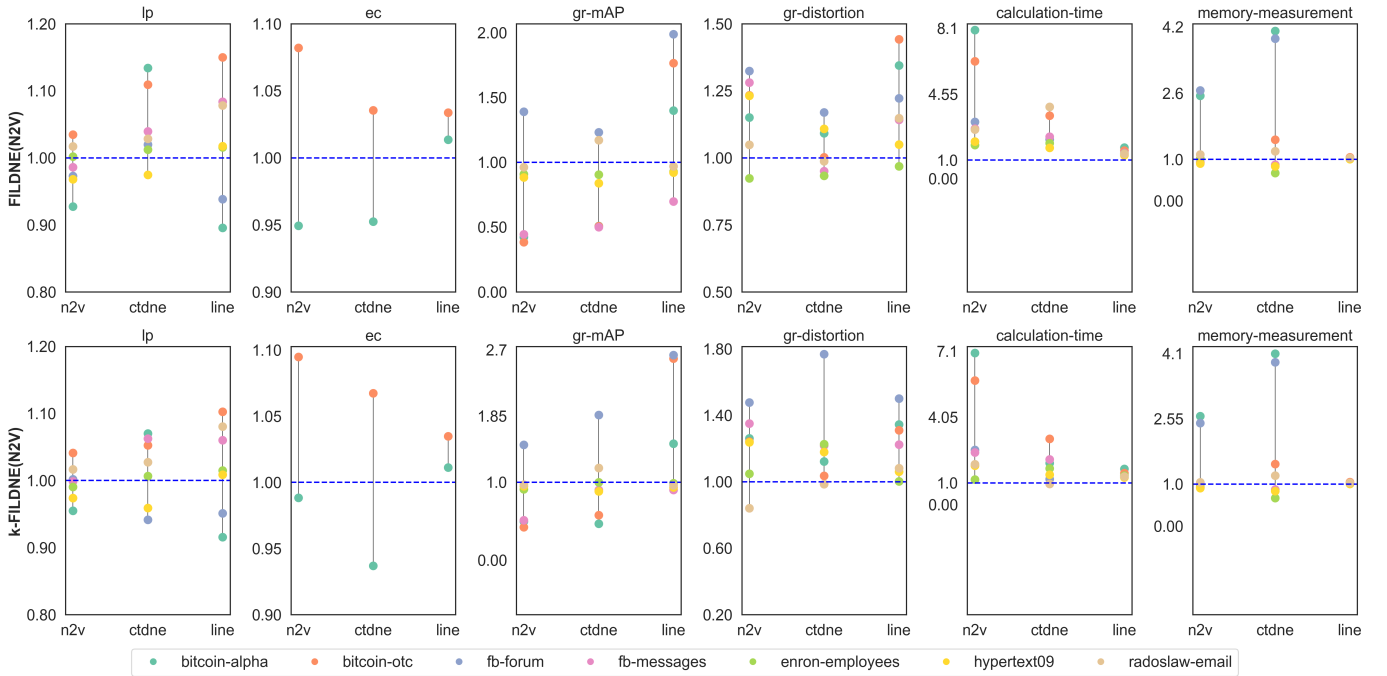|	BITCOIN ALPHA	BITCOIN OTC	FB FORUM	FB MESSAGES	ENRON EMPLOYEES	HYPERTEXT09	RADOSLAW EMAIL	MEAN RANK
N2V	580.09	317.12	552.29	360.35	212.27	<u>218.45</u>	227.75	3.71
LINE	917.17	947.08	1291.09	1309.96	1286.30	1277.53	920.00	12.16
CTDNE	1751.63	1262.22	1327.77	1113.66	344.43	320.13	361.23	9.07
FILDNE(N2V)	233.96	344.62	<u>207.13</u>	385.37	<u>206.60</u>	241.49	<u>203.49</u>	**2.23**
FILDNE(LINE)	888.59	899.76	1275.47	1279.96	1271.36	1269.56	889.90	10.45
FILDNE(CTDNE)	436.55	849.80	338.78	1270.43	514.53	384.26	300.85	6.80
K-FILDNE(N2V)	<u>224.82</u>	344.28	225.41	387.52	213.02	241.50	218.23	**2.62**
K-FILDNE(LINE)	888.82	901.10	1276.48	1281.39	1271.71	1269.49	889.21	10.66
K-FILDNE(CTDNE)	437.24	848.13	340.99	1272.02	514.99	385.14	300.77	7.09
DYNGRAPH2VEC(AERNN)	1373.12	1934.64	899.86	1023.55	793.38	774.93	816.70	10.20
TNODEEMBED	643.19	1225.00	774.82	827.67	624.12	598.00	618.93	8.54
ONLINE-N2V(STREAMWALK)	321.29	381.68	269.47	307.71	347.81	249.34	390.14	4.70
ONLINE-N2V(SECONDORDER)	260.86	<u>284.38</u>	258.63	<u>265.96</u>	253.60	246.26	259.93	**2.77**

Figure 7: Mean gain scores for FILDNE and k-FILDNE compared to corresponding Base methods. Points above the dashed line (score equal to 1) indicate a performance gain, whereas points below indicate loss.

memory consumption, and time required for computation. Such a perspective allows for a thorough inspection of our method's properties evaluated on real-world datasets.

In the case of link prediction, edge classification, and distortion, we either achieve comparable results (least gain score of 0.85) or improve the performance by a margin of 75%. We observe the most notable improvements for the calculation time, where we can speed up the computations by a factor of 8 for FILDNE (node2vec) and 7 for k-FILDNE (node2vec). Only for one of the considered cases, the speedup is below 1, i.e., 0.98 for k-FILDNE(ctdne) on radoslaw-email dataset. Memory measurements exhibit similar characteristics to other indicators – FILDNE performs

in most cases no worse than Base methods. In the best case, FILDNE allows reducing memory consumption up to 4 times. The maximum loss of about 30%, compared to vanilla ctdne is encountered with enron-employes, radoslaw-email, and fb-messages, which are small to medium-sized networks.

Results in mean Average Precision exhibit a slightly different nature of the FILDNE method. For some datasets (bitcoin-otc, fb-messages), we see a drop in the performance. At the same time, distortion in these cases remains at a decent level. We associate such behavior with applying the embedding alignment step, precisely how we choose reference nodes. As explored by us, multi-degree activity

function promotes choosing nodes with a similar degree but does not consider their neighborhood. Implementing dataset-specific activity and scoring functions could improve the results. However, one has to take into account the complexity of the function.

5.7.2. Comparison to Streaming methods

We selected Node2vec embeddings as the ones composed by our FILDNE method because three out of four competitive streaming approaches are built upon random walks. We compute the gain scores in an analogous way to Section 5.7.1. In terms of link prediction, our proposed FILDNE models achieve comparable results as the other streaming approaches. For edge classification, the tNodeEmbed presents slightly better results than other methods as it is a fine-tuned end-to-end edge classifier. Graph reconstruction measured by mean Average Precision shows the superiority of FILDNE by a factor of up to 50. Distortion gain scores vary between 0.55 and 1.3, what confirms FILDNE performance acceptable. We observe that other streaming methods tend to be slow compared to FILDNE method, which is reflected on the time gain scores – embeddings can be computed up to 30 (k-FILDNE) or 50 (FILDNE) times faster. The memory measurements are not surprising – we provide a comparable utilization to both Online-Node2vec models (StreamWalk and SecondOrder). Note that these were explicitly designed to reduce the memory footprint. At the same time, FILDNE significantly outperforms other streaming methods.

5.8. Hyperparameter sensitivity

In this experiment, we will demonstrate the influence of FILDNE's hyperparameters on downstream link prediction tasks in the last snapshot G_{10}. We use Node2vec as the Base method in all of the following experiments.

Reference node percentage. The first hyperparameter we evaluate is the percentage of reference nodes that are used in the alignment step. We proposed in Section 4.1 three selection schemes. Here we focus on the percentage scenario (with values: $1\%, 5\%, 10\%, \ldots, 95\%, 100\%$). We examine if each dataset has an individual percentage that yields the best results in downstream tasks. We plot the mean AUC values and standard deviations in link prediction tasks for each dataset (see Figure 9) with k-FILDNE (uniform prior) applied. A single AUC score for a given percentage value is computed as the mean over 30 experiment realizations. Finally, we fit a second-order polynomial (least sq. error) for visualizing the trend on a given dataset and mark in red the point with the highest AUC value.

There is an optimal reference nodes proportion within the exclusive range $(0\%, 100\%)$ if we observe a concave polynomial approximation. For more complex datasets, e.g., bitcoin-otc, that have a low node and edge Jaccard Index (see table 3) – representing high dynamics of the network and few common nodes in consecutive windows – it turns out that the best results are for 100 % ratio.

FILDNE alpha parameter. Next, we examine the impact of α parameter in FILDNE on AUC, exploring a range from 0 to 1 with a step size of 0.05. The results presented in Figure 10 for obtained for 30 experiment repetitions. We utilize the best ratio of reference nodes found in the previous experiment.

We hypothesized that we should consider both the historical and recent embedding information while constructing node vector representation. Our intuition is confirmed as we can observe different α for distinct datasets. In fb-forum, fb-messages, enron-employees, hypertext09, radoslaw-email, and bitcoin-otc, the past events are critical for performance. We see that greater α values indicate an increase in the embedding quality to a certain level, above which the performance decreases again. For bitcoin-alpha, we have to keep a subtle balance between the present and the past.

k-FILDNE prior. For k-FILDNE, we change the *prior* parameter and evaluate both possible values (uniform, increase). We noticed that the downstream task performance's change is not significant – both settings yield similar results. This shows that, after a certain number of observations (internal link prediction), the prior values become irrelevant – the likelihood computed from the actual data becomes more critical (the property that Maximum Likelihood Estimate is equal to Maximum A Posteriori in the limit).

5.9. Summary of the results

We provided an extensive experimental protocol that incorporated various network-related learning tasks, namely link prediction, edge classification, and graph reconstruction. We measured quality in each of those tasks and time/memory consumption on seven benchmark datasets. We can conclude that the experiments' results proved FILDNE superiority by reducing computation time (up to 50x) and memory consumption (up to 6x), achieving the same quality.

6. Conclusion and future work

In this paper, we proposed a Framework for Incremental Learning of Dynamic Networks Embedding (FILDNE). It utilizes timestep vectors obtained from any existing node embedding method and produces dynamic representation reducing the computational costs by working on batched data (non-overlapping graph snapshots). We showed experimentally in link prediction, edge classification, and graph reconstruction tasks on seven real-world datasets that FILNDE compared to static, dynamic, and temporal node embedding approaches reduces memory and computational time costs while providing competitive accuracy gains. Moreover, we conducted a hyperparameter sensitivity study and provided insights into how FILDNE's hyperparameters influence the vector representation quality. In terms of future work, we plan to address FILDNE's
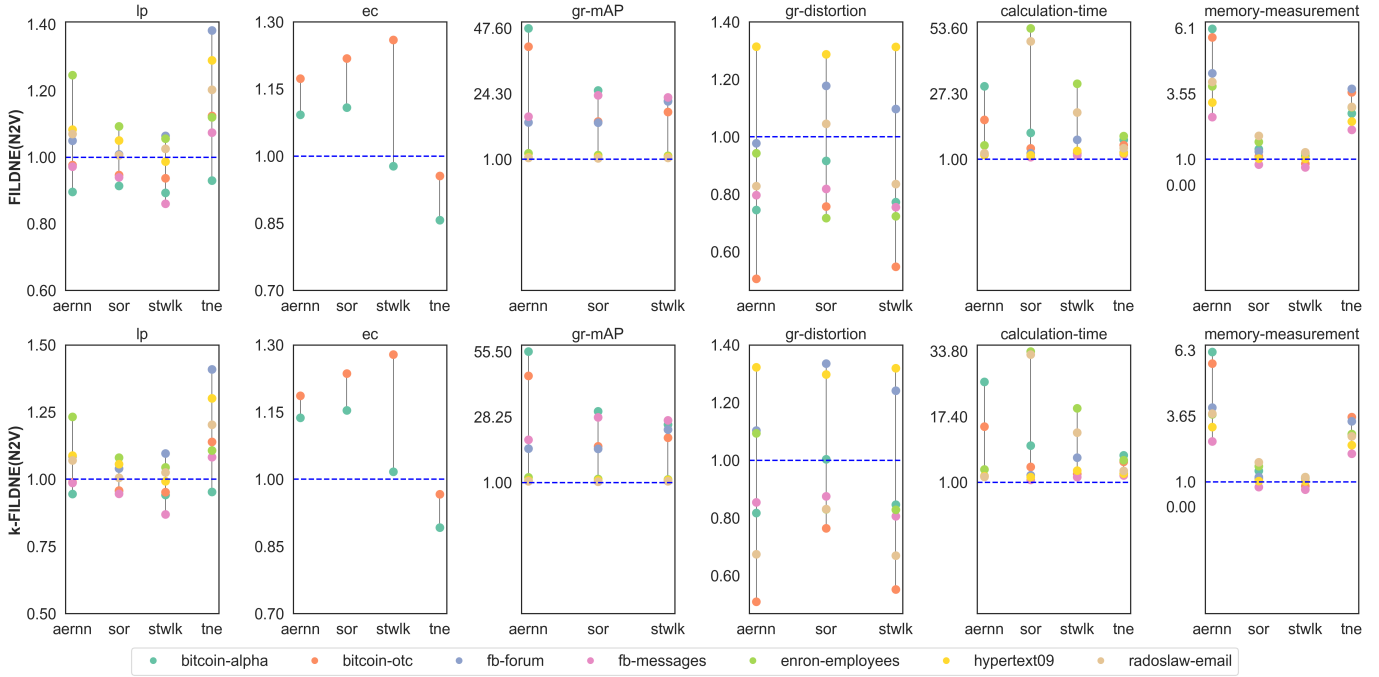
Figure 8: Mean gain scores for FILDNE and k-FILDNE (with node2vec as the Base method) compared to other streaming methods. Points above the dashed line (score equal to 1) indicate a performance gain, whereas points below indicate loss. Graph reconstruction results are not reported for tNodeEmbed, because the method performs an end-to-end task. We denote tNodeEmbed as `tne`, DynGraph2vec(AERNN) as `aernn` and both Online-Node2vec models as `stwlk` and `sor`.

Figure 9: Link prediction task results (AUC [%]) for different amounts of reference nodes (given as the mean fraction of common nodes between snapshots) The dashed line indicates the trend of these values (fitted using a 2nd order polynomial) and the red dot marks the highest AUC value. Each subtitle presents the name of the dataset and the coordinates of the highest value (percent, AUC).

limitations, namely: (1) it does not provide a mechanism for online learning (update after the single event) as our method requires batches (graph snapshots); (2) the calibration step requires an overlapping between consecutive snapshots in terms of nodes (there must exist common nodes) – one can expect a new way to calibrate such snapshots using nodes' structural similarity only; (3) we did not explore how our method works in an attributed environment (i.e., using both attributed graphs and embedding method suitable for such data); (4) in our experimental setup we decided to use snapshots of equal size (in terms of time intervals), but it might be required to extend that scenario.

Acknowledgments

Figure 10: Link prediction task results (AUC [%]) for different values of α. The dashed line indicates the trend of these values (fitted using a 2nd order polynomial) and the red dot marks the highest AUC value. Each subtitle presents the name of the dataset and the coordinates of the highest value (α, AUC)
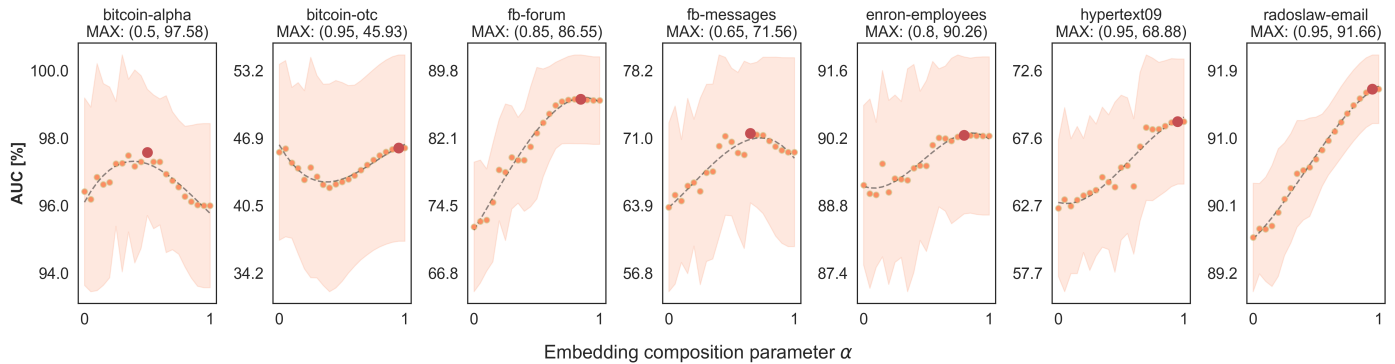
References

[1] Belkin, M., Niyogi, P.: Laplacian eigenmaps for dimensionality reduction and data representation. Neural computation **15**(6), 1373–1396 (2003)

[2] Béres, F., Kelen, D.M., Pálovics, R., Benczúr, A.A.: Node embeddings in dynamic graphs. Applied Network Science **4**(1) (dec 2019). https://doi.org/10.1007/s41109-019-0169-5

[3] Bergstra, J., Yamins, D., Cox, D.: Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. In: International conference on machine learning. pp. 115–123 (2013)

[4] Bergstra, J.S., Bardenet, R., Bengio, Y., Kégl, B.: Algorithms for hyper-parameter optimization. In: Advances in neural information processing systems. pp. 2546–2554 (2011)

[5] Cai, H., Zheng, V.W., Chang, K.C.C.: A Comprehensive Survey of Graph Embedding: Problems, Techniques, and Applications. IEEE Transactions on Knowledge and Data Engineering **30**(9), 1616–1637 (2018). https://doi.org/10.1109/TKDE.2018.2807452

[6] Chami, I., Abu-El-Haija, S., Perozzi, B., Ré, C., Murphy, K.: Machine learning on graphs: A model and comprehensive taxonomy. arXiv preprint arXiv:2005.03675 (2020)

[7] Chen, X., Heimann, M., Vahedian, F., Koutra, D.: Consistent network alignment with node embedding. arXiv preprint arXiv:2005.04725 (2020)

[8] Cui, P., Wang, X., Pei, J., Zhu, W.: A Survey on Network Embedding. IEEE Transactions on Knowledge and Data Engineering **31**(5), 833–852 (2019). https://doi.org/10.1109/TKDE.2018.2849727

[9] De Sa, C., Gu, A., Ré, C., Sala, F.: Representation tradeoffs for hyperbolic embeddings. Proceedings of machine learning research **80**, 4460 (2018)

[10] Derr, T., Karimi, H., Liu, X., Xu, J., Tang, J.: Deep adversarial network alignment. arXiv preprint arXiv:1902.10307 (2019)

[11] Dong, Y., Chawla, N.V., Swami, A.: metapath2vec: Scalable representation learning for heterogeneous networks. In: Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining. pp. 135–144 (2017)

[12] Gao, Y., Gong, M., Xie, Y., Zhong, H.: Community-oriented attributed network embedding. Knowledge-Based Systems **193**, 105418 (2020)

[13] Geng, X., Li, Y., Wang, L., Zhang, L., Yang, Q., Ye, J., Liu, Y.: Spatiotemporal multi-graph convolution network for ride-hailing demand forecasting. In: Proceedings of the AAAI Conference on Artificial Intelligence. vol. 33, pp. 3656–3663 (2019)

[14] Goyal, P., Chhetri, S.R., Canedo, A.: dyngraph2vec: Capturing network dynamics using dynamic graph representation learning. Knowledge-Based Systems **187**, 104816 (2020)

[15] Grave, E., Joulin, A., Berthet, Q.: Unsupervised alignment of embeddings with wasserstein procrustes. In: The 22nd International Conference on Artificial Intelligence and Statistics. pp. 1880–1890 (2019)

[16] Grover, A., Leskovec, J.: node2vec: Scalable feature learning for networks. In: Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining. pp. 855–864 (2016)

[17] Hou, C., Zhang, H., He, S., Tang, K.: Glodyne: Global topology preserving dynamic network embedding. arXiv preprint arXiv:2008.01935 (2020)

[18] Huang, F., Zhang, X., Xu, J., Li, C., Li, Z.: Network embedding by fusing multimodal contents and links. Knowledge-Based Systems **171**, 44–55 (2019)

[19] Kuprieiev, R., Petrov, D., Valles, R., Redzyński, P., da Costa-Luis, C., Schepanovski, A., Pachhai, S., Shcheklein, I., Orpinel, J., Santos, F., Rowlands, P., Sharma, A., Zhanibek, Hodovic, D., Grigorev, A., Earl, karajan1001, Dash, N., Vyshnya, G., nik123, maykulkarni, xliiv, Hora, M., Vera, Mangal, S., Baranowski, W., Wolff, C., Maslakov, A., Khamutov, A., Benoy, K.: Dvc: Data version control - git for data & models (Aug 2020). https://doi.org/10.5281/zenodo.3998731, https://doi.org/10.5281/zenodo.3998731

[20] Lee, J.B., Nguyen, G., Rossi, R.A., Ahmed, N.K., Koh, E., Kim, S.: Dynamic node embeddings from edge streams (2019)

[21] Liu, L., Cheung, W.K., Li, X., Liao, L.: Aligning users across social networks using network embedding. In: Ijcai. pp. 1774–1780 (2016)

[22] Mahdavi, S., Khoshraftar, S., An, A.: dynnode2vec: Scalable dynamic network embedding. In: 2018 IEEE International Conference on Big Data (Big Data). pp. 3762–3765. IEEE (2018)

[23] Marceau, D.J., Guindon, L., Bruel, M., Marois, C.: Building Temporal Topology in a GIS Database to Study the Land-Use Changes in a Rural-Urban Environment. Professional Geographer **53**(4), 546–558 (2001). https://doi.org/10.1111/0033-0124.00304

[24] Mikolov, T., Chen, K., Corrado, G.S., Dean, J.: Efficient estimation of word representations in vector space. CoRR **abs/1301.3781** (2013)

[25] Murphy, K.K.: Machine Learning: A Probabilistic Perspective. MIT Press (2012)

[26] Nelson, W., Zitnik, M., Wang, B., Leskovec, J., Goldenberg, A., Sharan, R.: To embed or not: network embedding as a paradigm in computational biology. Frontiers in genetics **10**, 381 (2019)

[27] Nguyen, G.H., Lee, J.B., Rossi, R.A., Ahmed, N.K., Koh, E., Kim, S.: Continuous-Time Dynamic Network Embeddings. pp. 969–976. Association for Computing Machinery (ACM) (2018). https://doi.org/10.1145/3184558.3191526

[28] Ou, M., Cui, P., Pei, J., Zhang, Z., Zhu, W.: Asymmetric transitivity preserving graph embedding. In: Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. vol. 13-17-August-2016, pp. 1105–1114. Association for Computing Machinery (aug 2016).

https://doi.org/10.1145/2939672.2939751

[29] Perozzi, B., Al-Rfou, R., Skiena, S.: Deepwalk: Online learning of social representations. In: Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining. pp. 701–710 (2014)

[30] Rossi, R., Ahmed, N.: The network data repository with interactive graph analytics and visualization. In: Twenty-Ninth AAAI Conference on Artificial Intelligence (2015)

[31] Roweis, S.T., Saul, L.K.: Nonlinear dimensionality reduction by locally linear embedding. science **290**(5500), 2323–2326 (2000)

[32] Singer, U., Guy, I., Radinsky, K.: Node embedding over temporal graphs. In: Proceedings of the 28th International Joint Conference on Artificial Intelligence. pp. 4605–4612. AAAI Press (2019)

[33] Tang, J., Qu, M., Wang, M., Zhang, M., Yan, J., Mei, Q.: Line: Large-scale information network embedding. In: Proceedings of the 24th international conference on world wide web. pp. 1067–1077 (2015)

[34] Torricelli, M., Karsai, M., Gauvin, L.: weg2vec: Event embedding for temporal networks. Scientific Reports **10**(1), 1–11 (2020)

[35] Trivedi, P., Büyükçakır, A., Lin, Y., Qian, Y., Jin, D., Koutra, D.: On structural vs. proximity-based temporal node embeddings (2020)

[36] Velickovic, P., Fedus, W., Hamilton, W.L., Liò, P., Bengio, Y., Hjelm, R.D.: Deep graph infomax. In: ICLR (Poster) (2019)

[37] Wu, Z., Pan, S., Long, G., Jiang, J., Zhang, C.: Graph wavenet for deep spatial-temporal graph modeling. arXiv preprint arXiv:1906.00121 (2019)

[38] Xu, D., Wei, C., Peng, P., Xuan, Q., Guo, H.: Ge-gan: A novel deep learning framework for road traffic state estimation. Transportation Research Part C: Emerging Technologies **117**, 102635 (2020)

[39] Xu, S., Liu, S., Feng, L.: Manifold graph embedding with structure information propagation for community discovery. Knowledge-Based Systems **208**, 106448 (2020)

[40] You, J., Liu, B., Ying, Z., Pande, V., Leskovec, J.: Graph convolutional policy network for goal-directed molecular graph generation. In: Advances in neural information processing systems. pp. 6410–6421 (2018)

[41] Yue, X., Wang, Z., Huang, J., Parthasarathy, S., Moosavinasab, S., Huang, Y., Lin, S.M., Zhang, W., Zhang, P., Sun, H.: Graph embedding on biomedical networks: methods, applications and evaluations. Bioinformatics **36**(4), 1241–1251 (2020)

[42] Zang, C., Wang, F.: Moflow: an invertible flow model for generating molecular graphs. In: Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. pp. 617–626 (2020)

[43] Zhang, Y., Li, Y., Zhu, Y., Hu, X.: Wasserstein gan based on autoencoder with back-translation for cross-lingual embedding mappings. Pattern Recognition Letters **129**, 311–316 (2020)

Appendices

A. Source code of used methods

We base our expirements on following methods implementations:

DynGraph2Vec - https://github.com/palash1992/DynamicGEM
DGI - https://github.com/PetarV-/DGI
N2V - https://github.com/eliorc/node2vec/
HOPE - https://github.com/palash1992/GEM
LLE - https://github.com/palash1992/GEM
SGE - https://scikit-learn.org/
OnlineN2V - https://github.com/ferencberes/online-node2vec
TNE - https://github.com/urielsinger/tNodeEmbed
FILDNE - code of our method will be published upon aceptance

B. All methods results (mean)

We provide comprehensive experimental results for all examined streaming approaches, Base methods and both FILDNE variants applied on these Base methods.

Table 11: Comparison of FILDNE and other methods in **link prediction task (AUC)**. The presented values are the mean AUC scores over 8 graph snapshots and 30 methods' retrains. We also report the mean ranks of all methods in this experiment. Methods marked in bold are the 3 best methods based on the mean rank. Underlined values show the highest AUC score for each dataset individually.

	BITCOIN ALPHA	BITCOIN OTC	FB FORUM	FB MESSAGES	ENRON EMPLOYEES	HYPERTEXT09	RADOSLAW EMAIL	MEAN RANK
N2V	65.60	52.82	75.84	65.43	92.28	73.76	90.15	**6.29**
LINE	66.08	54.69	<u>79.03</u>	70.75	90.59	72.71	86.74	6.48
CTDNE	55.58	51.23	65.24	54.35	87.30	67.31	83.32	11.79
HOPE	59.16	49.71	53.72	51.16	58.79	53.89	88.71	16.32
DGI	55.53	50.26	54.34	53.09	59.76	57.23	70.88	17.36
LE	47.46	49.73	54.17	51.81	58.23	55.01	77.99	18.02
LLE	×	×	55.69	×	59.96	58.13	54.87	19.69
FILDNE(N2V)	59.25	53.36	73.94	64.27	<u>92.43</u>	71.35	91.67	6.98
FILDNE(LINE)	58.28	<u>62.39</u>	74.11	<u>76.11</u>	92.00	<u>73.87</u>	93.42	**4.79**
FILDNE(CTDNE)	55.39	55.97	66.10	55.86	88.33	65.51	85.64	11.00
FILDNE(HOPE)	49.83	50.21	53.57	55.20	59.08	59.37	88.16	15.57
FILDNE(DGI)	53.98	51.48	55.57	53.32	63.56	59.89	68.78	15.77
FILDNE(LE)	49.39	50.59	55.27	49.15	62.12	58.03	80.72	16.79
FILDNE(LLE)	×	×	56.19	×	62.74	58.99	55.53	18.38
K-FILDNE(N2V)	62.22	53.93	75.95	65.09	91.43	71.71	91.66	6.52
K-FILDNE(LINE)	59.41	59.66	75.06	74.83	91.92	73.27	<u>93.62</u>	**5.21**
K-FILDNE(CTDNE)	53.52	53.38	61.24	57.01	87.85	64.44	85.54	12.25
K-FILDNE(HOPE)	53.60	49.50	53.35	54.08	53.30	58.36	78.47	17.39
K-FILDNE(DGI)	51.32	47.97	55.71	53.86	61.75	58.83	66.27	17.23
K-FILDNE(LE)	51.60	50.59	54.12	49.41	61.40	57.98	60.27	17.93
K-FILDNE(LLE)	×	×	55.84	×	61.33	57.73	56.53	19.44
DYNGRAPH2VEC(AERNN)	67.88	58.67	70.50	66.93	74.66	66.40	85.72	9.41
TNODEEMBED	68.68	48.40	55.96	60.97	83.57	55.79	76.87	13.93
ONLINE-N2V(STREAMWALK)	<u>69.59</u>	59.66	73.21	69.55	84.84	68.36	91.13	7.50
ONLINE-N2V(SECONDORDER)	68.70	58.33	69.34	75.31	87.59	72.44	89.41	6.77

Table 12: Comparison of FILDNE and other methods in **edge classification task (AUC)**. The presented values are the mean AUC scores over 8 graph snapshots and 30 methods' retrains. We also report the mean ranks of all methods in this experiment. Methods marked in bold are the 3 best methods based on the mean rank. Underlined values show the highest AUC score for each dataset individually.

	BITCOIN ALPHA	BITCOIN OTC	MEAN RANK
N2V	65.11	60.92	8.69
LINE	63.09	64.39	7.81
CTDNE	63.81	55.92	10.56
HOPE	57.76	43.04	17.25
DGI	57.58	59.39	12.00
LE	53.60	44.60	16.41
LLE	×	×	×
FILDNE(N2V)	61.57	65.47	7.75
FILDNE(LINE)	63.91	66.56	**6.19**
FILDNE(CTDNE)	59.28	57.85	10.94
FILDNE(HOPE)	58.89	50.77	15.03
FILDNE(DGI)	58.26	52.81	14.38
FILDNE(LE)	52.31	54.63	16.25
FILDNE(LLE)	×	×	×
K-FILDNE(N2V)	64.10	66.23	6.25
K-FILDNE(LINE)	63.92	66.57	**6.12**
K-FILDNE(CTDNE)	58.25	59.84	10.19
K-FILDNE(HOPE)	57.87	49.84	15.00
K-FILDNE(DGI)	58.29	54.61	12.94
K-FILDNE(LE)	47.80	49.87	18.50
K-FILDNE(LLE)	×	×	×
DYNGRAPH2VEC(AERNN)	56.40	56.14	13.62
TNODEEMBED	<u>71.88</u>	<u>69.32</u>	**3.62**
ONLINE-N2V(STREAMWALK)	56.73	56.70	12.81
ONLINE-N2V(SECONDORDER)	63.39	53.93	10.69

Table 13: Comparison of FILDNE and other methods in **graph reconstruction task (mAP)**. The presented values are the mean mAP scores over 8 graph snapshots and 30 methods' retrains. We also report the mean ranks of all methods in this experiment. Methods marked in bold are the 3 best methods based on the mean rank. Underlined values show the highest mAP score for each dataset individually.

	BITCOIN ALPHA	BITCOIN OTC	FB FORUM	FB MESSAGES	ENRON EMPLOYEES	HYPERTEXT09	RADOSLAW EMAIL	MEAN RANK
N2V	<u>65.99</u>	<u>70.18</u>	23.16	<u>67.44</u>	<u>70.46</u>	<u>55.95</u>	42.12	**1.84**
LINE	10.83	10.81	6.42	9.90	45.35	41.90	39.75	8.05
CTDNE	35.65	47.24	7.85	22.84	55.15	54.80	31.42	5.25
HOPE	0.51	0.64	2.43	1.39	13.20	28.10	27.06	18.32
DGI	0.66	0.66	2.55	1.45	12.44	27.11	26.91	18.48
LE	0.81	1.12	2.71	1.22	11.89	28.04	<u>76.53</u>	14.00
LLE	×	×	2.65	×	11.98	27.16	48.62	15.38
FILDNE(N2V)	28.73	26.40	30.98	30.00	63.90	49.45	40.55	**4.04**
FILDNE(LINE)	14.70	18.02	12.16	7.12	41.71	38.24	38.38	8.18
FILDNE(CTDNE)	18.20	24.27	9.53	11.59	49.39	45.92	36.84	6.86
FILDNE(HOPE)	0.59	0.67	2.50	1.46	12.90	27.42	27.55	18.27
FILDNE(DGI)	0.77	0.80	2.88	1.68	12.69	27.20	29.66	15.50
FILDNE(LE)	0.76	0.94	2.71	1.29	13.08	27.64	60.00	14.09
FILDNE(LLE)	×	×	2.58	×	11.21	28.12	26.34	19.34
K-FILDNE(N2V)	33.46	29.30	<u>33.17</u>	34.66	64.21	53.05	40.92	**2.89**
K-FILDNE(LINE)	15.39	25.02	15.58	8.92	44.43	38.82	38.08	7.23
K-FILDNE(CTDNE)	16.85	27.46	13.93	20.39	55.10	48.48	37.26	5.52
K-FILDNE(HOPE)	0.59	0.70	2.42	1.48	13.28	29.80	23.92	17.43
K-FILDNE(DGI)	0.93	0.94	3.24	2.06	12.72	27.76	29.04	14.04
K-FILDNE(LE)	0.73	0.90	2.81	1.40	12.84	29.05	41.80	13.79
K-FILDNE(LLE)	×	×	2.98	×	12.58	28.63	31.78	14.91
DYNGRAPH2VEC(AERNN)	0.60	0.66	2.22	1.84	20.46	27.75	26.51	17.07
ONLINE-N2V(STREAMWALK)	1.08	1.83	2.22	1.25	25.85	28.03	29.60	15.20
ONLINE-N2V(SECONDORDER)	1.38	1.54	1.46	1.28	29.09	27.88	27.11	16.04

Table 14: Comparison of FILDNE and other methods in graph reconstruction task (distortion). The presented values are the mean distortion scores over 8 graph snapshots and 30 methods' retrains. We also report the mean ranks of all methods in this experiment. Methods marked in bold are the 3 best methods based on the mean rank. Underlined values show the highest distortion score for each dataset individually.

	BITCOIN ALPHA	BITCOIN OTC	FB FORUM	FB MESSAGES	ENRON EMPLOYEES	HYPERTEXT09	RADOSLAW EMAIL	MEAN RANK
N2V	0.81	1.29	0.61	0.93	0.66	0.38	0.57	17.66
LINE	0.87	0.94	0.50	0.59	0.65	0.33	0.80	15.93
CTDNE	0.93	1.70	0.76	0.89	0.81	0.36	0.65	19.41
HOPE	0.58	0.60	0.36	0.48	0.48	0.34	0.44	8.93
DGI	<u>0.51</u>	0.55	0.36	0.61	0.49	0.34	<u>0.38</u>	7.38
LE	0.58	0.58	0.33	0.45	0.47	0.33	0.48	8.43
LLE	×	×	0.54	×	0.77	0.35	0.77	18.91
FILDNE(N2V)	0.70	1.05	0.47	0.74	0.69	0.30	0.54	14.29
FILDNE(LINE)	0.65	0.65	0.41	0.54	0.67	0.31	0.70	12.70
FILDNE(CTDNE)	0.85	1.70	0.65	0.93	0.84	0.32	0.66	18.54
FILDNE(HOPE)	0.56	0.60	0.40	0.48	0.48	0.34	0.42	8.89
FILDNE(DGI)	0.53	0.72	0.30	0.55	0.47	0.33	0.38	7.68
FILDNE(LE)	0.55	0.65	0.28	<u>0.38</u>	<u>0.41</u>	0.34	0.47	**6.96**
FILDNE(LLE)	×	×	0.46	×	0.88	0.35	0.81	19.00
K-FILDNE(N2V)	0.64	1.04	0.44	0.72	0.63	0.30	0.67	12.91
K-FILDNE(LINE)	0.65	0.73	0.33	0.50	0.66	0.31	0.74	11.61
K-FILDNE(CTDNE)	0.83	1.65	0.46	0.75	0.68	<u>0.30</u>	0.66	14.73
K-FILDNE(HOPE)	0.57	0.60	0.43	0.48	0.48	0.32	0.40	8.50
K-FILDNE(DGI)	0.52	0.72	0.30	0.51	0.49	0.33	0.42	**7.30**
K-FILDNE(LE)	0.55	0.61	<u>0.28</u>	0.38	0.43	0.32	0.43	**5.52**
K-FILDNE(LLE)	×	×	0.40	×	0.60	0.33	0.60	11.78
DYNGRAPH2VEC(AERNN)	0.52	<u>0.53</u>	0.44	0.50	0.62	0.40	0.45	10.54
ONLINE-N2V(STREAMWALK)	0.64	0.80	0.52	0.53	0.47	0.39	0.56	13.95
ONLINE-N2V(SECONDORDER)	0.54	0.57	0.48	0.49	0.49	0.40	0.45	10.20

Table 15: Comparison of FILDNE and other methods in embeddings calculation time (in seconds). The presented values are the mean calculation time over 8 graph snapshots and 30 methods' retrains. We also report the mean ranks of all methods in this experiment. Methods marked in bold are the 3 best methods based on the mean rank. Underlined values show the lowest calcualtion time for each dataset individually.

	BITCOIN ALPHA	BITCOIN OTC	FB FORUM	FB MESSAGES	ENRON EMPLOYEES	HYPERTEXT09	RADOSLAW EMAIL	MEAN RANK
N2V	46.71	133.83	24.94	75.72	4.40	9.52	11.90	17.91
LINE	70.78	63.10	103.97	114.16	114.84	105.17	58.56	22.50
CTDNE	16.25	61.41	3.98	77.44	16.82	4.39	2.65	15.86
HOPE	3.12	4.93	1.51	2.29	0.81	0.48	1.39	6.09
DGI	3.57	15.83	2.95	5.10	0.43	0.58	0.53	8.46
LE	5.34	13.81	1.71	2.64	0.29	0.25	0.38	5.52
LLE	×	×	4.22	×	0.28	0.25	0.66	7.09
FILDNE(N2V)	7.87	22.26	8.44	27.62	2.35	4.79	4.65	13.29
FILDNE(LINE)	42.34	41.66	80.53	82.18	84.43	83.29	43.50	19.79
FILDNE(CTDNE)	7.84	18.11	1.91	33.67	8.69	2.63	0.68	10.84
FILDNE(HOPE)	1.09	<u>1.52</u>	<u>1.02</u>	<u>1.35</u>	0.34	0.25	0.46	**2.71**
FILDNE(DGI)	<u>0.86</u>	4.59	2.10	2.91	0.34	0.51	0.38	5.09
FILDNE(LE)	1.24	2.01	1.45	1.68	0.25	0.19	<u>0.28</u>	**2.38**
FILDNE(LLE)	×	×	1.81	×	<u>0.16</u>	<u>0.11</u>	0.33	**2.34**
K-FILDNE(N2V)	8.61	24.04	9.90	30.69	3.71	5.31	6.40	14.50
K-FILDNE(LINE)	43.10	43.20	81.74	85.02	85.67	83.81	45.09	20.82
K-FILDNE(CTDNE)	8.59	19.88	3.33	36.37	9.96	3.15	2.71	13.61
K-FILDNE(HOPE)	1.78	3.50	2.24	4.14	1.42	0.75	1.91	8.21
K-FILDNE(DGI)	1.56	5.99	3.15	5.18	1.44	0.98	1.80	9.57
K-FILDNE(LE)	1.94	3.42	2.48	3.96	1.31	0.65	1.67	7.70
K-FILDNE(LLE)	×	×	2.80	×	1.29	0.63	1.74	9.34
DYNGRAPH2VEC(AERNN)	144.71	341.14	25.54	84.10	15.89	13.35	15.47	18.54
TNODEEMBED	52.46	143.07	31.32	84.88	24.96	16.90	24.85	20.20
ONLINE-N2V(STREAMWALK)	67.76	112.97	27.44	52.09	130.27	12.50	212.59	20.14
ONLINE-N2V(SECONDORDER)	20.78	72.98	70.51	74.15	76.83	20.92	86.89	19.70

Table 16: Comparison of FILDNE and other methods in in max. memory utilization (in MB). The presented values are the mean max. memory utilization during embeddings calculation over 8 graph snapshots and 30 methods' retrains. We also report the mean ranks of all methods in this experiment. Methods marked in bold are the 3 best methods based on the mean rank. Underlined values show the lowest memory utilization for each dataset individually.

	BITCOIN ALPHA	BITCOIN OTC	FB FORUM	FB MESSAGES	ENRON EMPLOYEES	HYPERTEXT09	RADOSLAW EMAIL	MEAN RANK
N2V	580.09	317.12	552.29	360.35	212.27	218.45	227.75	8.70
LINE	917.17	947.08	1291.09	1309.96	1286.30	1277.53	920.00	22.75
CTDNE	1751.63	1262.22	1327.77	1113.66	344.43	320.13	361.23	19.18
HOPE	669.68	938.47	300.07	384.13	258.63	253.09	257.59	13.05
DGI	501.36	776.38	418.67	504.65	344.67	338.79	352.69	16.16
LE	538.78	728.63	283.31	329.16	260.49	254.95	273.09	11.89
LLE	×	×	259.21	×	235.45	217.13	258.49	7.56
FILDNE(N2V)	233.96	344.62	<u>207.13</u>	385.37	<u>206.60</u>	241.49	<u>203.49</u>	4.75
FILDNE(LINE)	888.59	899.76	1275.47	1279.96	1271.36	1269.56	889.90	20.96
FILDNE(CTDNE)	436.55	849.80	338.78	1270.43	514.53	384.26	300.85	16.14
FILDNE(HOPE)	268.60	298.96	247.20	268.57	218.52	219.81	220.97	**4.73**
FILDNE(DGI)	342.13	437.32	381.80	401.09	323.55	323.18	328.49	13.36
FILDNE(LE)	247.31	<u>266.87</u>	242.43	<u>247.96</u>	220.95	219.35	220.60	**3.43**
FILDNE(LLE)	×	×	228.67	×	215.67	<u>213.14</u>	216.45	**2.62**
K-FILDNE(N2V)	<u>224.82</u>	344.28	225.41	387.52	213.02	241.50	218.23	5.38
K-FILDNE(LINE)	888.82	901.10	1276.48	1281.39	1271.71	1269.49	889.21	21.18
K-FILDNE(CTDNE)	437.24	848.13	340.99	1272.02	514.99	385.14	300.77	16.43
K-FILDNE(HOPE)	265.25	300.65	260.65	290.92	232.45	228.96	241.79	6.79
K-FILDNE(DGI)	344.94	442.24	392.31	405.24	333.50	329.50	343.56	14.32
K-FILDNE(LE)	252.32	267.62	261.33	265.87	233.62	224.25	238.77	5.61
K-FILDNE(LLE)	×	×	255.57	×	237.24	225.54	239.13	7.47
DYNGRAPH2VEC(AERNN)	1373.12	1934.64	899.86	1023.55	793.38	774.93	816.70	20.91
TNODEEMBED	643.19	1225.00	774.82	827.67	624.12	598.00	618.93	19.14
ONLINE-N2V(STREAMWALK)	321.29	381.68	269.47	307.71	347.81	249.34	390.14	11.59
ONLINE-N2V(SECONDORDER)	260.86	284.38	258.63	265.96	253.60	246.26	259.93	7.61