Subset Node Representation Learning over Large Dynamic Graphs

Xingzhi Guo*
xingzguo@cs.stonybrook.edu
Stony Brook University
Stony Brook, USA

Baojian Zhou*
baojian.zhou@cs.stonybrook.edu
Stony Brook University
Stony Brook, USA

Steven Skiena
skiena@cs.stonybrook.edu
Stony Brook University
Stony Brook, USA

ABSTRACT

Dynamic graph representation learning is a task to learn node embeddings over dynamic networks, and has many important applications, including knowledge graphs, citation networks to social networks. Graphs of this type are usually large-scale but only a small subset of vertices are related in downstream tasks. Current methods are too expensive to this setting as the complexity is at best linear-dependent on both the number of nodes and edges.

In this paper, we propose a new method, namely Dynamic Personalized PageRank Embedding (DynamicPPE) for learning a target subset of node representations over large-scale dynamic networks. Based on recent advances in local node embedding and a novel computation of dynamic personalized PageRank vector (PPV), DynamicPPE has two key ingredients: 1) the per-PPV complexity is $O(m\bar{d}/\epsilon)$ where $m, \bar{d}$, and ϵ are the number of edges received, average degree, global precision error respectively. Thus, the per-edge event update of a single node is only dependent on $\bar{d}$ in average; and 2) by using these high quality PPVs and hash kernels, the learned embeddings have properties of both locality and global consistency. These two make it possible to capture the evolution of graph structure effectively.

Experimental results demonstrate both the effectiveness and efficiency of the proposed method over large-scale dynamic networks. We apply DynamicPPE to capture the embedding change of Chinese cities in the Wikipedia graph during this ongoing COVID-19 pandemic [1]. Our results show that these representations successfully encode the dynamics of the Wikipedia graph.

CCS CONCEPTS

• **Information systems** → **Data mining**; • **Computing methodologies** → **Learning latent representations**.

KEYWORDS

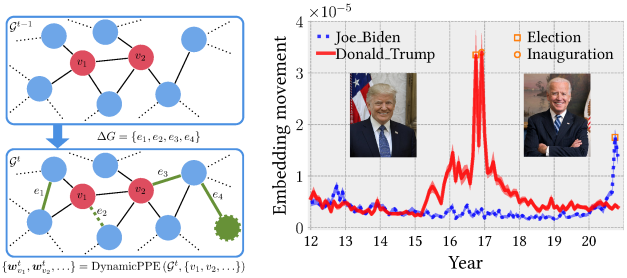
Dynamic graph embedding; Representation learning; Personalized PageRank; Knowledge evolution

*Both authors contributed equally to this research
[1] https://en.wikipedia.org/wiki/COVID-19_pandemic

KDD '21, August 14–18, 2021, Virtual Event, Singapore.

ACM ISBN 978-1-4503-8332-5/21/08…$15.00
https://doi.org/10.1145/3447548.3467393

ACM Reference Format:
Xingzhi Guo, Baojian Zhou, and Steven Skiena. 2021. Subset Node Representation Learning over Large Dynamic Graphs. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '21), August 14–18, 2021, Virtual Event, Singapore.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3447548.3467393



(a) Dynamic graph model (b) An application of DynamicPPE

Figure 1: (a) The model of dynamic network in two consecutive snapshots. (b) An application of DynamicPPE to keep track embedding movements of interesting Wikipedia articles (vertices). We learn embeddings of two presidents of the United States on the whole English Wikipedia graph from 2012 monthly, which cumulatively involves 6.2M articles (nodes) and 170M internal links (edges). The *embedding movement* between two time points is defined as $1 - \cos(\boldsymbol{w}_v^t, \boldsymbol{w}_v^{t+1})$ where $\cos(\cdot,\cdot)$ is the cosine similarity. The significant embedding movements may reflect big social status changes of Donald_Trump and Joe_Biden [2] in this dynamic Wikipedia graph.

1 INTRODUCTION

Graph node representation learning aims to represent nodes from graph structure data into lower dimensional vectors and has received much attention in recent years [12, 15, 16, 23, 29, 31, 37]. Effective methods have been successfully applied to many real-world applications where graphs are large-scale and static [43]. However, networks such as social networks [4], knowledge graphs [20], and citation networks [7] are usually time-evolving where edges and nodes are inserted or deleted over time. Computing representations of all vertices over time is prohibitively expensive because only a small subset of nodes may be interesting in a particular application. Therefore, it is important and technical challenging to efficiently

learn dynamic embeddings for these large-scale dynamic networks under this typical use case.

Specifically, we study the following dynamic embedding problem: We are given a subset $S = \{v_1, v_2, \ldots, v_k\}$ and an initial graph $\mathcal{G}^t$ with $t = 0$. Between time t and $t + 1$, there are edge events of insertions and/or deletions. The task is to design an algorithm to learn embeddings for k nodes with time complexity independent on the number of nodes n per time t where $k \ll n$. This problem setting is both technically challenging and practically important. For example, in the English Wikipedia graph, one need focus only on embedding movements of articles related to political leaders, a tiny portion of whole Wikipedia. Current dynamic embedding methods [10, 28, 47, 49, 50] are not applicable to this large-scale problem setting due to the lack of efficiency. More specifically, current methods have the *dependence issue* where one must learn all embedding vectors. This dependence issue leads to per-embedding update is linear-dependent on n, which is inefficient when graphs are large-scale. This obstacle motivates us to develop a new method.

In this paper, we propose a dynamic personalized PageRank embedding (DynamicPPE) method for learning a subset of node representations over large-sale dynamic networks. DynamicPPE is based on an effective approach to compute dynamic PPVs [45]. There are two challenges of using Zhang et al. [45] directly: 1) the quality of dynamic PPVs depend critically on precision parameter ϵ, which unfortunately is unknown under the dynamic setting; and 2) The update of per-edge event strategy is not suitable for batch update between graph snapshots. To resolve these two difficulties, first, we adaptively update ϵ so that the *estimation error* is independent of n, m, thus obtaining high quality PPVs. Yet previous work does not give an estimation error guarantee. We prove that the time complexity is only dependent on $\bar{d}$. Second, we incorporate a batch update strategy inspired from [13] to avoid frequent per-edge update. Therefore, the total run time to keep track of k nodes for given snapshots is $O(k\bar{d}m)$. Since real-world graphs have the sparsity property $\bar{d} \ll n$, it significantly improves the efficiency compared with previous methods. Inspired by *InstantEmbedding* [30] for static graph, we use hash kernels to project dynamic PPVs into embedding space. Figure 1 shows an example of successfully applying DynamicPPE to study the dynamics of social status in the English Wikipedia graph. To summarize, our contributions are:

(1) We propose a new algorithm DynamicPPE, which is based on the recent advances of local network embedding on static graph and a novel computation of dynamic PPVs. DynamicPPE effectively learns PPVs and then projects them into embedding space using hash kernels.
(2) DynamicPPE adaptively updates the precision parameter ϵ so that PPVs always have a provable estimation error guarantee. In our subset problem setting, we prove that the time and space complexity are all linear to the number of edges m but independent on the number of nodes n, thus significantly improve the efficiency.
(3) Node classification results demonstrate the effectiveness and efficiency of the proposed. We compile three large-scale datasets to validate our method. As an application, we showcase that learned embeddings can be used to detect the changes of Chinese cities during this ongoing COVID-19 pandemic articles on a large-scale English Wikipedia.

The rest of this paper is organized as follows: In Section 2, we give the overview of current dynamic embedding methods. The problem definition and preliminaries are in Section 3. We present our proposed method in Section 4. Experimental results are reported in Section 5. The discussion and conclusion will be presented in Section 6. Our code and created datasets are accessible at https://github.com/zjlxgxz/DynamicPPE.

2 RELATED WORK

There are two main categories of works for learning embeddings from the dynamic graph structure data. The first type is focusing on capturing the evolution of dynamics of graph structure [49]. The second type is focusing on both dynamics of graph structure and features lie in these graph data [38]. In this paper, we focus on the first type and give the overview of related works. Due to the large mount of works in this area, some related works may not be included, one can find more related works in a survey [22] and references therein.

Dynamic latent space models The dynamic embedding models had been initially explored by using latent space model [18]. The dynamic latent space model of a network makes an assumption that each node is associated with an d-dimensional vector and distance between two vectors should be small if there is an edge between these two nodes [32, 33]. Works of these assume that the distance between two consecutive embeddings should be small. The proposed dynamic models were applied to different applications [17, 34]. Their methods are not scalable from the fact that the time complexity of initial position estimation is at least $O(n^2)$ even if the per-node update is $\log(n)$.

Incremental SVD and random walk based methods Zhang et al. [48] proposed TIMERS that is an incremental SVD-based method. To prevent the error accumulation, TIMERS properly set the restart time so that the accumulated error can be reduced. Nguyen et al. [28] proposed continuous-time dynamic network embeddings, namely CTDNE. The key idea of CTDNE is that instead of using general random walks as DeepWalk [29], it uses temporal random walks contain a sequence of edges in order. Similarly, the work of Du et al. [10] was also based on the idea of DeepWalk. These methods have time complexity dependent on n for per-snapshot update. Zhou et al. [49] proposed to learn dynamic embeddings by modeling the triadic closure to capture the dynamics.

Graph neural network methods Trivedi et al. [38] designed a dynamic node representation model, namely DyRep, as modeling a latent mediation process where it leverages the changes of node between the node's social interactions and its neighborhoods. More specifically, DyRep contains a temporal attention layer to capture the interactions of neighbors. Zang and Wang [44] proposed a neural network model to learn embeddings by solving a differential equation with ReLU as its activation function. [24] presents a dynamic embedding, a recurrent neural network method, to learn the interactions between users and items. However, these methods either need to have features as input or cannot be applied to large-scale dynamic graph. Kumar et al. [24] proposed an algorithm to

[2]Two English Wikipedia articles are accessible at https://en.wikipedia.org/wiki/Donald_Trump and https://en.wikipedia.org/wiki/Joe_Biden.

learn the trajectory of the dynamic embedding for temporal interaction networks. Since the learning task is different from ours, one can find more details in their paper.

3 NOTATIONS AND PRELIMINARIES

Notations We use $[n]$ to denote a ground set $[n] := \{0, 1, \ldots, n-1\}$. The graph snapshot at time t is denoted as $\mathcal{G}^t\left(\mathbb{V}^t, \mathbb{E}^t\right)$. The degree of a node v is $d(v)^t$. In the rest of this paper, the average degree at time t is $\bar{d}^t$ and the subset of target nodes is $S \subseteq \mathbb{V}^t$. Bold capitals, e.g. $\boldsymbol{A}, \boldsymbol{W}$ are matrices and bold lower letters are vectors $\boldsymbol{w}, \boldsymbol{x}$. More specifically, the embedding vector for node v at time t denoted as $\boldsymbol{w}_v^t \in \mathbb{R}^d$ and d is the embedding dimension. The i-th entry of $\boldsymbol{w}_v^t$ is $w_v^t(i) \in \mathbb{R}$. The embedding of node v for all T snapshots is written as $\boldsymbol{W}_v = [\boldsymbol{w}_v^1, \boldsymbol{w}_v^2, \ldots, \boldsymbol{w}_v^T]^\top$. We use n_t and m_t as the number of nodes and edges in $\mathcal{G}^t$ which we simply use n and m if time t is clear in the context.

Given the graph snapshot $\mathcal{G}^t$ and a specific node v, the personalized PageRank vector (PPV) is an n-dimensional vector $\boldsymbol{\pi}_v^t \in \mathbb{R}^n$ and the corresponding i-th entry is $\pi_v^t(i)$. We use $\boldsymbol{p}_v^t \in \mathbb{R}^n$ to stand for a calculated PPV obtained from a specific algorithm. Similarly, the corresponding i-th entry is $p_v^t(i)$. The teleport probability of the PageRank is denoted as α. The *estimation error* of an embedding vector is the difference between true embedding $\boldsymbol{w}_v^t$ and the estimated embedding $\hat{\boldsymbol{w}}_v^t$ is measure by $\|\cdot\|_1 := \sum_{i=1}^n \left|w_v^t(i) - \hat{w}_v^t(i)\right|$.

3.1 Dynamic graph model and its embedding

Given any initial graph (could be an empty graph), the corresponding dynamic graph model describes how the graph structure evolves over time. We first define the dynamic graph model, which is based on Kazemi and Goel [22].

Definition 1 (Simple dynamic graph model [22]). *A simple dynamic graph model is defined as an ordered of snapshots $\mathcal{G}^0, \mathcal{G}^1, \mathcal{G}^2, \ldots, \mathcal{G}^T$ where $\mathcal{G}^0$ is the initial graph. The difference of graph $\mathcal{G}^t$ at time $t = 1, 2, \ldots, T$ is $\Delta G^t(\Delta\mathbb{V}^t, \Delta\mathbb{E}^t) := G^t \backslash G^{t-1}$ with $\Delta\mathbb{V}^t := \mathbb{V}^t \backslash \mathbb{V}^{t-1}$ and $\Delta\mathbb{E}^t := \mathbb{E}^t \backslash \mathbb{E}^{t-1}$. Equivalently, $\Delta\mathcal{G}^t$ corresponds to a sequence of edge events as the following*

$$\Delta\mathcal{G}^t = \left\{e_1^t, e_2^t, \ldots, e_{m'}^t\right\}, \tag{1}$$

where each edge event e_i^t has two types: insertion or deletion, i.e, $e_i^t = (u, v, \text{event})$ where $\text{event} \in \{\text{Insert}, \text{Delete}\}$ [3].

The above model captures evolution of a real-world graph naturally where the structure evolution can be treated as a sequence of edge events occurred in this graph. To simplify our analysis, we assume that the graph is undirected. Based on this, we define the subset dynamic representation problem as the following.

Definition 2 (Subset dynamic network embedding problem). *Given a dynamic network model $\left\{\mathcal{G}^0, \mathcal{G}^1, \mathcal{G}^2, \ldots, \mathcal{G}^T\right\}$ define in Definition 1 and a subset of target nodes $S = \{v_1, v_2, \ldots, v_k\}$, the subset dynamic network embedding problem is to learn dynamic embeddings of T snapshots for all k nodes S where $k \ll n$. That is, given any node $v \in S$, the goal is to learn embedding matrix for each node $v \in S$, i.e.*

$$\boldsymbol{W}_v := [\boldsymbol{w}_v^1, \boldsymbol{w}_v^2, \ldots, \boldsymbol{w}_v^T]^\top \text{ where } \boldsymbol{w}_v^t \in \mathbb{R}^d \text{ and } v \in S. \tag{2}$$

[3] The node insertion can be treated as inserting a new edge and then delete it and node deletion is a sequence of deleting its edges.

3.2 Personalized PageRank

Given any node v at time t, the personalized PageRank vector for graph $\mathcal{G}^t$ is defined as the following

Definition 3 (Personalized PageRank (PPR)). *Given normalized adjacency matrix $\boldsymbol{W}_t = \boldsymbol{D}_t^{-1}\boldsymbol{A}_t$ where $\boldsymbol{D}_t$ is a diagonal matrix with $D_t(i,i) = d(i)^t$ and $\boldsymbol{A}_t$ is the adjacency matrix, the PageRank vector $\boldsymbol{\pi}_s^t$ with respect to a source node s is the solution of the following equation*

$$\boldsymbol{\pi}_s^t = \alpha * \mathbf{1}_s + (1-\alpha)\boldsymbol{\pi}_s^t \boldsymbol{W}^t, \tag{3}$$

where $\mathbf{1}_s$ is the unit vector with $1_s(v) = 1$ when $v = s$, 0 otherwise.

There are several works on computing PPVs for static graph [1, 2, 5]. The idea is based a local push operation proposed in [2]. Interestingly, Zhang et al. [45] extends this idea and proposes a novel updating strategy for calculating dynamic PPVs. We use a modified version of it as presented in Algorithm 1.

Algorithm 1 ForwardPush [45]

Input: $\boldsymbol{p}_s, \boldsymbol{r}_s, \mathcal{G}, \epsilon, \beta = 0$
while $\exists u, r_s(u) > \epsilon d(u)$ **do**
  Push(u)
while $\exists u, r_s(u) < -\epsilon d(u)$ **do**
  Push(u)
return $(\boldsymbol{p}_s, \boldsymbol{r}_s)$
procedure Push(u)
  $p_s(u)$ += $\alpha r_s(u)$
  for $v \in \text{Nei}(u)$ **do**
    $r_s(v)$ += $(1-\alpha)r_s(u)(1-\beta)/d(u)$
  $r_s(u) = (1-\alpha)r_s(u)\beta$

Algorithm 1 is a generalization from Andersen et al. [2] and there are several variants of forward push [2, 5, 26], which are dependent on how β is chosen (we assume $\beta = 0$). The essential idea of forward push is that, at each Push step, the frontier node u transforms her α residual probability $r_s(u)$ into estimation probability $p_s(u)$ and then pushes the rest residual to its neighbors. The algorithm repeats this push operation until all residuals are small enough [4]. Methods based on local push operations have the following invariant property.

Lemma 4 (Invariant property [19]). *ForwardPush has the following invariant property*

$$\pi_s(u) = p_s(u) + \sum_{v \in V} r_s(v)\pi_v(u), \forall u \in \mathbb{V}. \tag{4}$$

The local push algorithm can guarantee that the each entry of the estimation vector $p_s(v)$ is very close to the true value $\pi_s(v)$. We state this property as in the following

Lemma 5 ([2, 45]). *Given any graph $\mathcal{G}(\mathbb{V}, \mathbb{E})$ with $\boldsymbol{p}_s = \mathbf{0}, \boldsymbol{r}_s = \mathbf{1}_s$ and a constant ϵ, the run time for ForwardLocalPush is at most $\frac{1-\|\boldsymbol{r}_s\|_1}{\alpha\epsilon}$ and the estimation error of $\pi_s(v)$ for each node v is at most ϵ, i.e. $|p_s(v) - \pi_s(v)|/d(v)| \leq \epsilon$*

[4] There are two implementation of forward push depends on how frontier is selected. One is to use a first-in-first-out (FIFO) queue [11] to maintain the frontiers while the other one maintains nodes using a priority queue is used [5] so that the operation cost is $O(1/\epsilon\alpha)$ instead of $O(\log n/\epsilon\alpha)$.

The main challenge to directly use forward push algorithm to obtain high quality PPVs in our setting is that: 1) the quality $\boldsymbol{p}_s$ return by forward push algorithm will critically depend on the precision parameter ϵ which unfortunately is unknown under our dynamic problem setting. Furthermore, the original update of per-edge event strategy proposed in [45] is not suitable for batch update between graph snapshots. Guo et al. [13] propose to use a batch strategy, which is more practical in real-world scenario where there is a sequence of edge events between two consecutive snapshots. This motivates us to develop a new algorithm for dynamic PPVs and then use these PPVs to obtain high quality dynamic embedding vectors.

4 PROPOSED METHOD

To obtain dynamic embedding vectors, the general idea is to obtain dynamic PPVs and then project these PPVs into embedding space by using two kernel functions [30, 42]. In this section, we present our proposed method DynamicPPE where it contains two main components: 1) an adaptive precision strategy to control the estimation error of PPVs. We then prove that the time complexity of this dynamic strategy is still independent on n. With this quality guarantee, learned PPVs will be used as proximity vectors and be "projected" into lower dimensional space based on ideas of *Verse* [39] and *InstantEmbedding* [30]. We first show how can we get high quality PPVs and then present how use PPVs to obtain dynamic embeddings. Finally, we give the complexity analysis.

4.1 Dynamic graph embedding for single batch

For each batch update ΔG^t, the key idea is to dynamically maintain PPVs where the algorithm updates the estimate from $\boldsymbol{p}_v^{t-1}$ to $\boldsymbol{p}_v^t$ and its residuals from $\boldsymbol{r}_v^{t-1}$ to $\boldsymbol{r}_v^t$. Our method is inspired from Guo et al. [13] where they proposed to update a personalized contribution vector by using the local reverse push [5]. The proposed dynamic single node embedding, DynamicSNE is illustrated in Algorithm 2. It takes an update batch $\Delta\mathcal{G}^t$ (a sequence of edge events), a target node s with a precision ϵ^t, estimation vector of s and residual vector as inputs. It then obtains an updated embedding vector of s by the following three steps: 1) It first updates the estimate vector $\boldsymbol{p}_s^t$ and $\boldsymbol{r}_s^t$ from Line 2 to Line 9; 2) It then calls the forward local push method to obtain the updated estimations, $\boldsymbol{p}_s^t$; 3) We then use the hash kernel projection step to get an updated embedding. This projection step is from *InstantEmbedding* where two universal hash functions are defined as $h_d : \mathbb{N} \to [d]$ and $h_{\text{sgn}} : \mathbb{N} \to \{\pm 1\}$ [6]. Then the hash kernel based on these two hash functions is defined as $H_{h_{\text{sgn}},h_d}(\boldsymbol{x}) : \mathbb{R}^n \to \mathbb{R}^d$ where each entity i is $\sum_{j\in h_d^{-1}(i)} x_j h_{\text{sgn}}(j)$. Different from random projection used in RandNE [47] and FastRP [6], hash functions has $O(1)$ memory cost while random projection based method has $O(dn)$ if the Gaussian matrix is used. Furthermore, hash kernel keeps unbiased estimator for the inner product [42].

[5]One should notice that, for undirected graph, PPVs can be calculated by using the invariant property from the contribution vectors. However, the invariant property does not hold for directed graph. It means that one cannot use reverse local push to get a personalized PageRank vector directly. In this sense, using forward push algorithm is more general for our problem setting.

[6]For example, in our implementation, we use MurmurHash https://github.com/aappleby/smhasher

In the rest of this section, we show that the time complexity is $O(m\bar{d}/\epsilon)$ in average and the estimation error of learned PPVs measure by $\|\cdot\|_1$ can also be bounded. Our proof is based on the following lemma which follows from Guo et al. [13], Zhang et al. [45].

Lemma 6. *Given current graph $\mathcal{G}^t$ and an update batch $\Delta\mathcal{G}^t$, the total run time of the dynamic single node embedding, DynamicSNE for obtaining embedding $\boldsymbol{w}_s^t$ is bounded by the following*

$$T_t \leq \frac{\|\boldsymbol{r}_s^{t-1}\|_1 - \|\boldsymbol{r}_s^t\|_1}{\alpha\epsilon^t} + \sum_{u\in\Delta\mathcal{G}^t} \frac{2-\alpha}{\alpha}\frac{p_s^{t-1}(u)}{d(u)} \tag{5}$$

Proof. We first make an assumption that there is only one edge update (u, v, event) in $\Delta\mathcal{G}^t$, then based Lemma 14 of [46], the run time of per-edge update is at most:

$$\frac{\|\boldsymbol{r}_s^{t-1}\|_1 - \|\boldsymbol{r}_s^{t-1}\|_1}{\alpha\epsilon^t} + \frac{\Delta_t(s)}{\alpha\epsilon^t}, \tag{6}$$

where $\Delta_t(s) = \frac{2-\alpha}{\alpha}\frac{p_s^{t-1}(u)}{d(u)}$. Suppose there are k edge events in $\Delta\mathcal{G}^t$. We still obtain a similar bound, by the fact that forward push algorithm has monotonicity property: the entries of estimates $p_s^t(v)$ only increase when it pushes positive residuals (Line 2 and 3 of Algorithm 1). Similarly, estimates $p_s^t(v)$ only decrease when it pushes negative residuals (Line 4 and 5 of Algorithm 1). In other words, the amount of work for per-edge update is not less than the amount of work for per-batch update. Similar idea is also used in [13]. □

Algorithm 2 DynamicSNE$(\mathcal{G}^t, \Delta\mathcal{G}^t, s, \boldsymbol{p}_s^{t-1}, \boldsymbol{r}_s^{t-1}, \epsilon^t, \alpha)$

1: **Input:** graph $\mathcal{G}^t, \Delta\mathcal{G}^t$, target node s, precision ϵ, teleport α
2: **for** $(u, v, \text{op}) \in \Delta G^t$ **do**
3: **if** op == Insert(u, v) **then**
4: $\Delta_p = p_s^{t-1}(u)/(d(u)^t - 1)$
5: **if** op == Delete(u, v) **then**
6: $\Delta_p = -p_s^{t-1}(u)/(d(u)^t + 1)$
7: $p_s^{t-1}(u) \leftarrow p_s^{t-1}(u) + \Delta_p$
8: $r_s^{t-1}(u) \leftarrow r_s^{t-1}(u) - \Delta_p/\alpha$
9: $r_s^{t-1}(v) \leftarrow r_s^{t-1}(v) + \Delta_p/\alpha - \Delta_p$
10: $\boldsymbol{p}_s^t = \text{ForwardPush}(\boldsymbol{p}_s^{t-1}, \boldsymbol{r}_s^{t-1}, \mathcal{G}^t, \epsilon^t, \alpha)$
11: $\boldsymbol{w}_s^t = \boldsymbol{0}$
12: **for** $i \in \{v : p_s^t(v) \neq 0, v \in \mathbb{V}^t\}$ **do**
13: $w_s^t(h_d(i))$ += $h_{\text{sgn}}(i) \max\left(\log\left(p_s^t(i)n^t\right), 0\right)$

Theorem 7. *Given any graph snapshot $\mathcal{G}^t$ and an update batch ΔG^t where there are m_t edge events and suppose the precision parameter is ϵ^t and teleport probability is α, DynamicSNE runs in $O(m_t/\alpha^2 + m_t\bar{d}^t/(\epsilon\alpha^2) + m_t/(\epsilon\alpha))$ with $\epsilon^t = \epsilon/m_t$*

Proof. Based lemma 6, the proof directly follows from Theorem 12 of [46]. □

The above theorem has an important difference from the previous one [45]. We require that the precision parameter will be small enough so that $\|\boldsymbol{p}_s^t - \boldsymbol{\pi}_s^t\|_1$ can be bound (will be discussed later). As we did the experiments in Figure 2, the fixed epsilon will make the embedding vector bad. We propose to use a dynamic precision

parameter, where $\epsilon^t \sim O(\epsilon'/m_t)$, so that the ℓ_1-norm can be properly bounded. Interestingly, this high precision parameter strategy will not cause too high time complexity cost from $O(m/\alpha^2)$ to $O(m\bar{d}/(\epsilon\alpha^2))$. The bounded estimation error is presented in the following theorem.

Theorem 8 (Estimation error). *Given any node s, define the estimation error of PPVs learned from DynamicSNE at time t as $\|\boldsymbol{p}_s^t - \boldsymbol{\pi}_s^t\|_1$, if we are given the precision parameter $\epsilon^t = \epsilon/m_t$, the estimation error can be bounded by the following*

$$\|\boldsymbol{p}_s^t - \boldsymbol{\pi}_s^t\|_1 \leq \epsilon, \tag{7}$$

where we require $\epsilon \leq 2$ [7] and ϵ is a global precision parameter of DynamicPPE independent on m_t and n_t.

Proof. Notice that for any node u, by Lemma 5, we have the following inequality

$$|p_s^t(u) - \pi_s^t(u)| \leq \epsilon d^t(u).$$

Summing all these inequalities over all nodes u, we have

$$\begin{aligned}\|\boldsymbol{p}_s^t - \boldsymbol{\pi}_s^t\|_1 &= \sum_{u\in\mathbb{V}^t} \left|p_s^t(u) - \pi_s^t(u)\right| \\ &\leq \sum_{u\in\mathbb{V}^t} \epsilon^t d^t(u) = \epsilon^t m_t = \frac{\epsilon}{m_t} m_t = \epsilon.\end{aligned}$$

□

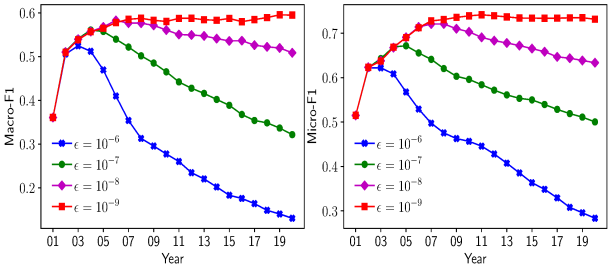


Figure 2: ϵ as a function of year for the task of node classification on the English Wikipedia graph. Each line corresponds to a fixed precision strategy of DynamicSNE. Clearly, when the precision parameter ϵ decreases, the performance of node classification improves.

The above theorem gives estimation error guarantee of $\boldsymbol{p}_s^t$, which is critically important for learning high quality embeddings. First of all, the dynamic precision strategy is inevitable because the precision is unknown parameter for dynamic graph where the number of nodes and edges could increase dramatically over time. To demonstrate this issue, we conduct an experiments on the English Wikipedia graph where we learn embeddings over years and validate these embeddings by using node classification task. As shown in Figure 2, when we use the fixed parameter, the performance of node classification is getting worse when the graph is getting bigger. This is mainly due to the lower quality of PPVs. Fortunately, the adaptive precision parameter ϵ/m_t does not make the run time increase dramatically. It only dependents on the average degree $\bar{d}^t$. In practice, we found $\epsilon = 0.1$ are sufficient for learning effective embeddings.

[7] By noticing that $\|\boldsymbol{p}_s^t - \boldsymbol{\pi}_s^t\|_1 \leq \|\boldsymbol{p}_s^t\|_1 + \|\boldsymbol{\pi}_s^t\|_1 \leq 2$, any precision parameter larger than 2 will be meaningless.

4.2 DynamicPPE

Our finally algorithm DynamicPPE is presented in Algorithm 3. At every beginning, estimators $\boldsymbol{p}_s^t$ are set to be zero vectors and residual vectors $\boldsymbol{r}_s^t$ are set to be unit vectors with mass all on one entry (Line 4). The algorithm then call the procedure DynamicSNE with an empty batch as input to get initial PPVs for all target nodes [8] (Line 5). From Line 6 to Line 9, at each snapshot t, it gets an update batch $\Delta\mathcal{G}^t$ at Line 7 and then calls DynamicSNE to obtain the updated embeddings for every node v.

Algorithm 3 DynamicPPE($\mathcal{G}_0, S, \epsilon, \alpha$)

1: **Input:** initial graph $\mathcal{G}^0$, target set S, global precision ϵ, teleport probability α
2: $t = 0$
3: **for** $s \in S := \{v_1, v_2, \ldots, v_k\}$ **do**
4: $\quad \boldsymbol{p}_s^t = \boldsymbol{0}, \quad \boldsymbol{r}_s^t = \boldsymbol{1}_s$
5: $\quad$ DynamicSNE($\mathcal{G}^0, \emptyset, s, \boldsymbol{p}_s^t, \boldsymbol{r}_s^t, 1/m_0, \alpha$)
6: **for** $t \in \{1, 2, \ldots, T\}$ **do**
7: $\quad$ read a sequence of edge events $\Delta\mathcal{G}^t := \mathcal{G}^t \backslash \mathcal{G}^{t-1}$
8: $\quad$ **for** $s \in S := \{v_1, v_2, \ldots, v_k\}$ **do**
9: $\quad\quad \boldsymbol{w}_s^t$ = DynamicSNE($\mathcal{G}^t, \Delta\mathcal{G}^t, s, \boldsymbol{p}_s^{t-1}, \boldsymbol{r}_s^{t-1}, \epsilon/m_t, \alpha$)
10: **return** $\boldsymbol{W}_s^t = [\boldsymbol{w}_s^1, \boldsymbol{w}_s^2, \ldots, \boldsymbol{w}_s^T], \forall s \in S$.

Based on our analysis, DynamicPPE is an dynamic version of *InstantEmbedding*. Therefore, DynamicPPE has two key properties observed in [30]: *locality* and *global consistency*. The embedding quality is guaranteed from the fact that *InstantEmbedding* implicitly factorizes the proximity matrix based on PPVs [39].

4.3 Complexity analysis

Time complexity The overall time complexity of DynamicPPE is the k times of the run time of DynamicSNE. We summarize the time complexity of DynamicPPE as in the following theorem

Theorem 9. *The time complexity of DynamicPPE for learning a subset of k nodes is $O(k\frac{m_t}{\alpha^2} + k\frac{m_t\bar{d}^t}{\alpha^2} + \frac{m_t}{\epsilon} + kT\min\{n, \frac{m}{\epsilon\alpha}\})$*

Proof. We follow Theorem 7 and summarize all run time together to get the final time complexity. □

Space complexity The overall space complexity has two parts: 1) $O(m)$ to store the graph structure information; and 2) the storage of keeping nonzeros of $\boldsymbol{p}_s^t$ and $\boldsymbol{r}_s^t$. From the fact that local push operation [2], the number of nonzeros in $\boldsymbol{p}_s^t$ is at most $\frac{1}{\epsilon\alpha}$. Thus, the total storage for saving these vectors are $O(k\min\{n, \frac{m}{\epsilon\alpha}\})$. Therefore, the total space complexity is $O(m + k\min(n, \frac{m}{\epsilon\alpha}))$.

Implementation Since learning the dynamic node embedding for any node v is independent with each other, DynamicPPE is are easy to parallel. Our current implementation can take advantage of multi-cores and compute the embeddings of S in parallel.

[8] For the situation that some nodes of S has not appeared in $\mathcal{G}^t$ yet, it checks every batch update until all nodes are initialized.

5 EXPERIMENTS

To demonstrate the effectiveness and efficiency of DynamicPPE, in this section, we first conduct experiments on several small and large scale real-world dynamic graphs on the task of node classification, followed by a case study about changes of Chinese cities in Wikipedia graph during the COVID-19 pandemic.

5.1 Datasets

We compile the following three real-world dynamic graphs, more details can be found in Appendix C.

Enwiki20 English Wikipedia Network We collect the internal Wikipedia Links (WikiLinks) of English Wikipedia from the beginning of Wikipedia, January 11th, 2001, to December 31, 2020 [9]. The internal links are extracted using a regular expression proposed in [8]. During the entire period, we collection 6,151,779 valid articles[10]. We generated the WikiLink graphs only containing edge insertion events. We keep all the edges existing before Dec. 31 2020, and sort the edge insertion order by the creation time. There are 6,216,199 nodes and 177,862,656 edges during the entire period. Each node either has one label (*Food, Person, Place,...*) or no label.

Patent (US Patent Graph) The citation network of US patent[14] contains 2,738,011 nodes with 13,960,811 citations range from year 1963 to 1999. For any two patents u and v, there is an edge (u, v) if the patent u cites patent v. Each patent belongs to six different types of patent categories. We extract a small weakly-connected component of 46,753 nodes and 425,732 edges with timestamp, called *Patent-small*.

Coauthor We extracted the co-authorship network from the Microsoft Academic Graph [35] dumped on September 21, 2019. We collect the papers with less than six coauthors, keeping those who has more than 10 publications, then build undirected edges between each coauthor with a timestamp of the publication date. In addition, we gather temporal label (e.g.: *Computer Science, Art, ...*) of authors based on their publication's field of study. Specifically we assign the label of an author to be the field of study where s/he published the most up to that date. The original graph contains 4,894,639 authors and 26,894,397 edges ranging from year 1800 to 2019. We also sampled a small connected component containing 49,767 nodes and 755,446 edges with timestamp, called *Coauthor-small*.

Academic The co-authorship network is from the academic network [36, 49] where it contains 51,060 nodes and 794,552 edges. The nodes are generated from 1980 to 2015. According to the node classification setting in Zhou et al. [49], each node has either one binary label or no label.

5.2 Node Classification Task

Experimental settings We evaluate embedding quality on binary classification for Academic graph (as same as in [49]), while using multi-class classification for other tasks. We use balanced logistic regression with ℓ_2 penalty in on-vs-rest setting, and report the Macro-F1 and Macro-AUC (ROC) from 5 repeated trials with 10% training ratio on the labeled nodes in each snapshot, excluding dangling nodes. Between each snapshot, we insert new edges and keep the previous edges.

[9] We collect the data from the dump https://dumps.wikimedia.org/enwiki/20210101/

[10] A valid Wikipedia article must be in the 0 namespace

We conduct the experiments on the aforementioned small and large scale graph. In small scale graphs, we calculate the embeddings of all nodes and compare our proposed method (**DynPPE.**) against other state-of-the-art models from three categories[11]. 1) Random walk based static graph embeddings (**Deepwalk**[12] [29], **Node2Vec**[13] [12]); 2) Random Projection-based embedding method which supports online update: **RandNE**[14] [47]; 3) Dynamic graph embedding method: **DynamicTriad** (**DynTri.**) [15] [49] which models the dynamics of the graph and optimized on link prediction.

Table 1: Node classification task on the *Academic, Patent Small, Coauthor Small* graph on the final snapshot

		Academic		Patent Small		Coauthor Small	
		F1	AUC	F1	AUC	F1	AUC
$d = 128$							
Static method	Node2Vec	0.833	**0.975**	0.648	0.917	0.477	**0.955**
	Deepwalk	**0.834**	**0.975**	**0.650**	**0.919**	**0.476**	0.950
Dynamic method	DynTri.	0.817	0.969	0.560	0.866	0.435	0.943
	RandNE	0.587	0.867	0.428	0.756	0.337	0.830
	DynPPE.	0.808	0.962	0.630	0.911	0.448	0.951
$d = 512$							
Static method	Node2Vec	0.841	**0.975**	0.677	0.931	0.486	0.955
	Deepwalk	**0.842**	**0.975**	0.680	0.931	0.495	0.955
Dynamic method	DynTri.	0.811	0.965	0.659	0.915	0.492	0.952
	RandNE	0.722	0.942	0.560	0.858	0.493	0.895
	DynPPE.	**0.842**	0.973	**0.682**	**0.934**	**0.509**	**0.958**

Table 2: Total CPU time for small graphs (in second). RandNE-I is with orthogonal projection, the performance is slightly better, but the running time is significantly increased. RandNE-II is without orthogonal projection.

	Academic	Patent Small	Coauthor Small
Deepwalk[16]	498211.75	181865.56	211684.86
Node2vec	4584618.79	2031090.75	1660984.42
DynTri.	247237.55	117993.36	108279.4
RandNE-I	12732.64	9637.15	8436.79
RandNE-II	1583.08	9208.03	177.89
DynPPE.	18419.10	3651.59	21323.74

Table 1 shows the classification results on the final snapshot. When we restrict the dimension to be 128, we observe that static methods outperform all dynamic methods. However, the static methods independently model each snapshot and the running time grows with number of snapshots, regardless of the number of edge changes. In addition, our proposed method (DynPPE.) outperforms other dynamic baselines, except that in the *academic* graph, where DynTri. is slightly better. However, their CPU time is 13 times

[11] Appendix D shows the hyper-parameter settings of baseline methods

[12] https://pypi.org/project/deepwalk/

[13] https://github.com/aditya-grover/node2vec

[14] https://github.com/ZW-ZHANG/RandNE/tree/master/Python

[15] https://github.com/luckiezhou/DynamicTriad

[16] We ran static graph embedding methods over a set of sampled snapshots and estimate the total CPU time for all snapshots.

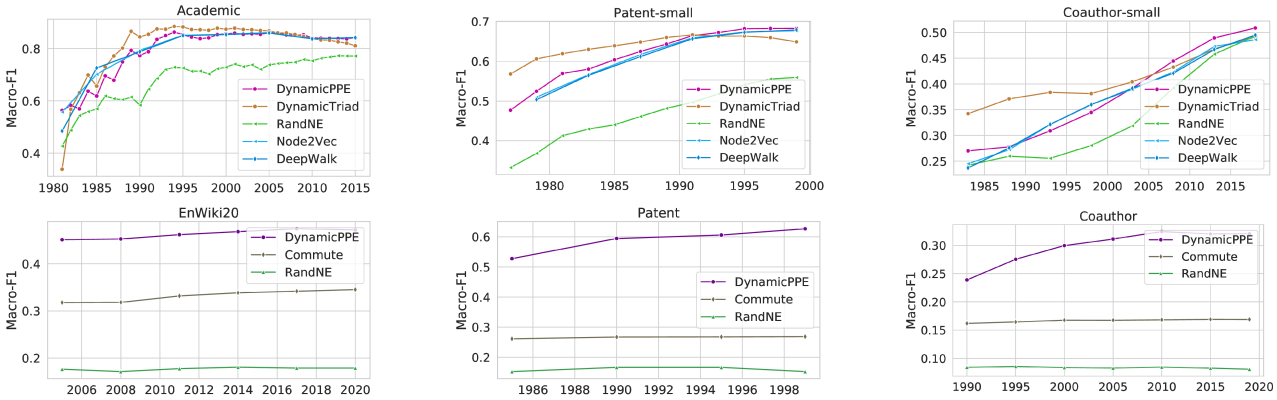


Figure 3: Macro-F1 scores of node classification as a function of time. The results of the small and large graphs are on the first and second row respectively (dim=512). Our proposed methods achieves the best performance on the last snapshot when all edges arrived and performance curve matches to the static methods as the graph becomes more and more complete.

more than ours as shown in Table 2. According to the Johnson-Lindenstrauss lemma[9, 21], we suspect that the poor result of RandNE is partially caused by the dimension size. As we increase the dimension to 512, we see a great performance improvement of RandNE. Also, our proposed method takes the same advantage, and outperform all others baselines in F1-score. Specifically, the increase of dimension makes hash projection step (Line 12-13 in Algorithm 2) retain more information and yield better embeddings. We attach the visualization of embeddings in Appendix.E.

The first row in the Figure 3 shows the F1-scores in each snapshot when the dimension is 512. We observe that in the earlier snapshots where many edges are not arrived, the performance of DynamicTriad [49] is better. One possible reason is that it models the dynamics of the graph, thus the node feature is more robust to the "missing" of the future links. While other methods, including ours, focus on an online feature updates incurred by the edge changes without explicitly modeling the temporal dynamics of a graph. Meanwhile, the performance curve of our method matches to the static methods, demonstrating the embedding quality of the intermediate snapshots is comparable to the state-of-the-art.

Table 2 shows the CPU time of each method (Dim=512). As we expected, static methods is very expensive as they calculate embeddings for each snapshot individually. Although our method is not blazingly fast compared to RandNE, it has a good trade-off between running time and embedding quality, especially without much hyper-parameter tuning. Most importantly, it can be easily parallelized by distributing the calculation of each node to clusters.

We also conduct experiment on large scale graphs (*EnWiki20, Patent, Coauthor*). We keep track of the vertices in a subset containing $|S| = 3,000$ nodes randomly selected from the first snapshot in each dataset, and similarly evaluate the quality of the embeddings of each snapshot on the node classification task. Due to scale of the graph, we compare our method against **RandNE** [47] and an fast heuristic **Algorithm 4**. Our method can calculate the embeddings for a subset of useful nodes only, while other methods have to calculate the embeddings of all nodes, which is not necessary under our scenario detailed in Sec. 5.3. The second row in Figure 3 shows that our proposed method has the best performance.

Table 3: Total CPU time for large graphs (in second)

	Enwiki20	Patent	Coauthor
Commute	6702.1	639.94	1340.74
RandNE-II	47992.81	6524.04	20771.19
DynPPE.	1538215.88	139222.01	411708.9
DynPPE (Per-node)	512.73	46.407	137.236

Table 3 shows the total CPU time of each method (Dim=512). Although total CPU time of our proposed method seems to be the greatest, the average CPU time for one node (as shown in row 4) is significantly smaller. This benefits a lot of downstream applications where only a subset of nodes are interesting to people in a large dynamic graph. For example, if we want to monitor the weekly embedding changes of a single node (e.g., the city of Wuhan, China) in English Wikipedia network from year 2020 to 2021, we can have the results in roughly 8.5 minutes. Meanwhile, other baselines have to calculate the embeddings of all nodes, and this expensive calculation may become the bottleneck in a downstream application with real-time constraints. To demonstrate the usefulness of our method, we conduct a case study in the following subsection.

5.3 Change Detection

Thanks to the contributors timely maintaining Wikipedia, we believe that the evolution of the Wikipedia graph reflects how the real world is going on. We assume that when the structure of a node greatly changes, there must be underlying interesting events or anomalies. Since the structural changes can be well-captured by graph embedding, we use our proposed method to investigate whether anomalies happened to the Chinese cities during the COVID-19 pandemic (from Jan. 2020 to Dec. 2020).

Changes of major Chinese Cities We target nine Chinese major cities (*Shanghai, Guangzhou, Nanjing, Beijing, Tianjin,* ***Wuhan****, Shenzhen, Chengdu, Chongqing)* and keep track of the embeddings

Table 4: The top cities ranked by the z-score along time. The corresponding news titles are from the news in each time period. $d(v)$ is node degree, $\Delta d(v)$ is the degree changes from the previous timestamp.

Date	City	$d(v)$	$\Delta d(v)$	Z-score	Top News Title
1/22/20	Wuhan	2890	54	2.210	NBC: "New virus prompts U.S. to screen passengers from Wuhan, China"
2/2/20	Wuhan	2937	47	1.928	WSJ: "U.S. Sets Evacuation Plan From Coronavirus-Infected Wuhan"
2/13/20	Hohhot	631	20	1.370	Poeple.cn: "26 people in Hohhot were notified of dereliction of duty for prevention and control, and the director of the Health Commission was removed" (Translated from Chinese).
2/24/20	Wuhan	3012	38	2.063	USA Today: "Coronavirus 20 times more lethal than the flu? Death toll passes 2,000"
3/6/20	Wuhan	3095	83	1.723	Reuters: "Infelctions may drop to zero by end-March in Wuhan: Chinese government expert"
3/17/20	Wuhan	3173	78	1.690	NYT: "Politicians Use of 'Wuhan Virus' Starts a Debate Health Experets Wanted to Avoid"
3/28/20	Zhangjiakou	517	15	1.217	"Logo revealed for Freestyle Ski and Snowboard World Championships in Zhangjiakou"
4/8/20	Wuhan	3314	47	2.118	CNN:"China lifts 76-day lockdown on Wuhan as city reemerges from conronavirus crisis"
4/19/20	Luohe	106	15	2.640	Forbes: "The Chinese Billionaire Whose Company Owns Troubled Pork Processor Smithfield Foods"
4/30/20	Zunhua	52	17	2.449	XINHUA: "Export companies resume production in Zunhua, Hebei"
5/11/20	Shulan	88	46	2.449	CGTN: "NE China's Shulan City to reimpose community lockdown in 'wartime' battle against COVID-19"

in a 10-day time window. From our prior knowledge, we expect that *Wuhan* should greatly change since it is the first reported place of the COVID-19 outbreak in early Jan. 2020.

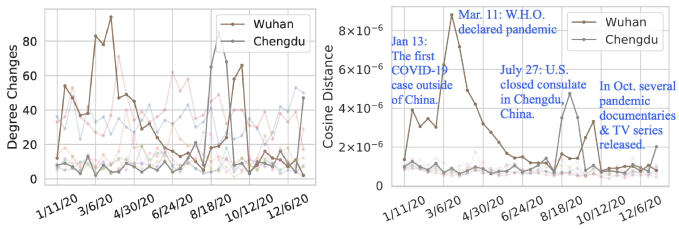


Figure 4: The changes of major Chinese cities in 2020. *Left*:Changes in Node Degree. *Right*: Changes in Cosine distance

Figure.4(a) shows the node degree changes of each city every 10 days. The curve is quite noisy, but we can still see several major peaks from *Wuhan* around 3/6/20 and *Chengdu* around 8/18/20. When using embedding changes [17] as the measurement, Figure.4 (b) provides a more clear view of the changed cities. We observe strong signals from the curve of *Wuhan*, correlating to the initial COVID-19 outbreak and the declaration of pandemic [18]. In addition, we observed an peak from the curve of *Chengdu* around 8/18/20 when U.S. closed the consulate in Chengdu, China, reflecting the U.S.-China diplomatic tension[19].

Top changed city along time We keep track of the embedding movement of 533 Chinese cities from year 2020 to 2021 in a 10-day time window, and filter out the inactive records by setting the threshold of degree changes (e.g. greater than 10). The final results are 51 Chinese cities from the major ones listed above and less famous cities like *Zhangjiakou, Hohhot, Shulan, ...*

Furthermore, we define the z-score of a target node u as $Z_t(u)$ based on the embedding changes within a specific period of time from $t-1$ to t.

$$Z_t(u|u \in S) = \frac{Dist(w_u^t, w_u^{t-1}) - \mu}{\sigma}$$

$$\mu = \frac{1}{|S|}\sum_{u' \in S} Dist(w_{u'}^t, w_{u'}^{t-1}), \quad \sigma = \sqrt{\frac{1}{|S|}\sum_{u' \in S}(Dist(w_{u'}^t, w_{u'}^{t-1}) - \mu)^2}$$

In Table 4, we list the highest ranked cities by the z-score from Jan.22 to May 11, 2020. In addition, we also attach the top news titles corresponding to the city within each specific time period. We found that Wuhan generally appears more frequently as the situation of the pandemic kept changing. Meanwhile, we found the appearance of *Hohhot* and *Shulan* reflects the time when COVID-19 outbreak happened in those cities. We also discovered cities unrelated to the pandemic. For example, *Luohe*, on 4/19/20, turns out to be the city where the headquarter of the organization which acquired Smithfield Foods (as mentioned in the news). In addition, *Zhangjiakou*, on 3/28/20, is the city, which will host World Snowboard competition, released the Logo of that competition.

6 DISCUSSION AND CONCLUSION

In this paper, we propose a new method to learn dynamic node embeddings over large-scale dynamic networks for a subset of interesting nodes. Our proposed method has time complexity that is linear-dependent on the number of edges m but independent on the number of nodes n. This makes our method applicable to applications of subset representations on very large-scale dynamic graphs. For the future work, as shown in Trivedi et al. [38], there are two dynamics on dynamic graph data, *structure evolution* and dynamics of *node interaction*. It would be interesting to study how one can incorporate dynamic of node interaction into our model. It is also worth to study how different version of local push operation affect the performance of our method.

ACKNOWLEDGMENTS

This work was partially supported by NSF grants IIS-1926781, IIS-1927227, IIS-1546113 and OAC-1919752.

[17] Again, the embedding movement $\text{Dist}(\cdot,\cdot)$ is defined as $1 - \cos(\cdot,\cdot)$
[18] COIVD-19: https://www.who.int/news/item/27-04-2020-who-timeline---covid-19
[19] US Consulate: https://china.usembassy-china.org.cn/embassy-consulates/chengdu/

REFERENCES

[1] Reid Andersen, Christian Borgs, Jennifer Chayes, John Hopcraft, Vahab S Mirrokni, and Shang-Hua Teng. 2007. Local computation of PageRank contributions. In *International Workshop on Algorithms and Models for the Web-Graph*. Springer, 150–165.

[2] Reid Andersen, Fan Chung, and Kevin Lang. 2006. Local graph partitioning using pagerank vectors. In *2006 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS'06)*. IEEE, 475–486.

[3] Reid Andersen, Fan Chung, and Kevin Lang. 2007. Using pagerank to locally partition a graph. *Internet Mathematics* 4, 1 (2007), 35–64.

[4] Tanya Y Berger-Wolf and Jared Saia. 2006. A framework for analysis of dynamic social networks. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*. 523–528.

[5] Pavel Berkhin. 2006. Bookmark-coloring algorithm for personalized pagerank computing. *Internet Mathematics* 3, 1 (2006), 41–62.

[6] Haochen Chen, Syed Fahad Sultan, Yingtao Tian, Muhao Chen, and Steven Skiena. 2019. Fast and Accurate Network Embeddings via Very Sparse Random Projection. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*. 399–408.

[7] Colin B Clement, Matthew Bierbaum, Kevin P O'Keeffe, and Alexander A Alemi. 2019. On the Use of ArXiv as a Dataset. *arXiv preprint arXiv:1905.00075* (2019).

[8] Cristian Consonni, David Laniado, and Alberto Montresor. 2019. WikiLinkGraphs: A complete, longitudinal and multi-language dataset of the Wikipedia link networks. In *Proceedings of the International AAAI Conference on Web and Social Media*, Vol. 13. 598–607.

[9] Sanjoy Dasgupta and Anupam Gupta. 1999. An elementary proof of the Johnson-Lindenstrauss lemma. *International Computer Science Institute, Technical Report* 22, 1 (1999), 1–5.

[10] Lun Du, Yun Wang, Guojie Song, Zhicong Lu, and Junshan Wang. 2018. Dynamic Network Embedding: An Extended Approach for Skip-gram based Network Embedding.. In *IJCAI*, Vol. 2018. 2086–2092.

[11] David F Gleich. 2015. PageRank beyond the Web. *Siam Review* 57, 3 (2015), 321–363.

[12] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*. 855–864.

[13] Wentian Guo, Yuchen Li, Mo Sha, and Kian-Lee Tan. 2017. Parallel personalized pagerank on dynamic graphs. *Proceedings of the VLDB Endowment* 11, 1 (2017), 93–106.

[14] Bronwyn H Hall, Adam B Jaffe, and Manuel Trajtenberg. 2001. *The NBER patent citation data file: Lessons, insights and methodological tools*. Technical Report. National Bureau of Economic Research.

[15] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. In *Advances in neural information processing systems*. 1024–1034.

[16] William L Hamilton, Rex Ying, and Jure Leskovec. 2017. Representation learning on graphs: Methods and applications. *arXiv preprint arXiv:1709.05584* (2017).

[17] Peter Hoff. 2007. Modeling homophily and stochastic equivalence in symmetric relational data. *Advances in neural information processing systems* 20 (2007), 657–664.

[18] Peter D Hoff, Adrian E Raftery, and Mark S Handcock. 2002. Latent space approaches to social network analysis. *Journal of the american Statistical association* 97, 460 (2002), 1090–1098.

[19] Sungryong Hong, Bruno C Coutinho, Arjun Dey, Albert-L Barabási, Mark Vogelsberger, Lars Hernquist, and Karl Gebhardt. 2016. Discriminating topology in galaxy distributions using network analysis. *Monthly Notices of the Royal Astronomical Society* 459, 3 (2016), 2690–2700.

[20] Guoliang Ji, Shizhu He, Liheng Xu, Kang Liu, and Jun Zhao. 2015. Knowledge graph embedding via dynamic mapping matrix. In *Proceedings of the 53rd annual meeting of the association for computational linguistics and the 7th international joint conference on natural language processing (volume 1: Long papers)*. 687–696.

[21] William B Johnson and Joram Lindenstrauss. 1984. Extensions of Lipschitz mappings into a Hilbert space. *Contemporary mathematics* 26, 189-206 (1984), 1.

[22] Seyed Mehran Kazemi and Rishab Goel. 2020. Representation Learning for Dynamic Graphs: A Survey. *Journal of Machine Learning Research* 21, 70 (2020), 1–73.

[23] Thomas N. Kipf and Max Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *International Conference on Learning Representations (ICLR)*.

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

[25] TG Lei, CW Woo, JZ Liu, and F Zhang. 2003. On the Schur complements of diagonally dominant matrices.

[26] Peter Lofgren. 2015. *Efficient Algorithms for Personalized PageRank*. Ph.D. Dissertation. Stanford University.

[27] Peter Lofgren, Siddhartha Banerjee, and Ashish Goel. 2015. Bidirectional PageRank Estimation: From Average-Case to Worst-Case. In *Proceedings of the 12th International Workshop on Algorithms and Models for the Web Graph-Volume 9479*. 164–176.

[28] Giang Hoang Nguyen, John Boaz Lee, Ryan A Rossi, Nesreen K Ahmed, Eunyee Koh, and Sungchul Kim. 2018. Continuous-time dynamic network embeddings. In *Companion Proceedings of the The Web Conference 2018*. 969–976.

[29] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. 701–710.

[30] Ștefan Postăvaru, Anton Tsitsulin, Filipe Miguel Gonçalves de Almeida, Yingtao Tian, Silvio Lattanzi, and Bryan Perozzi. 2020. InstantEmbedding: Efficient Local Node Representations. *arXiv preprint arXiv:2010.06992* (2020).

[31] Ryan A Rossi, Brian Gallagher, Jennifer Neville, and Keith Henderson. 2013. Modeling dynamic behavior in large evolving graphs. In *Proceedings of the sixth ACM international conference on Web search and data mining*. 667–676.

[32] Purnamrita Sarkar and Andrew W Moore. 2005. Dynamic social network analysis using latent space models. *Acm Sigkdd Explorations Newsletter* 7, 2 (2005), 31–40.

[33] Purnamrita Sarkar and Andrew W Moore. 2005. Dynamic social network analysis using latent space models. In *Proceedings of the 18th International Conference on Neural Information Processing Systems*. 1145–1152.

[34] Purnamrita Sarkar, Sajid M Siddiqi, and Geogrey J Gordon. 2007. A latent space approach to dynamic embedding of co-occurrence data. In *Artificial Intelligence and Statistics*. 420–427.

[35] Arnab Sinha, Zhihong Shen, Yang Song, Hao Ma, Darrin Eide, Bo-June Hsu, and Kuansan Wang. 2015. An overview of microsoft academic service (mas) and applications. In *Proceedings of the 24th international conference on world wide web*. 243–246.

[36] Jie Tang, Jing Zhang, Limin Yao, Juanzi Li, Li Zhang, and Zhong Su. 2008. Arnetminer: extraction and mining of academic social networks. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*. 990–998.

[37] Rakshit Trivedi, Mehrdad Farajtabar, Prasenjeet Biswal, and Hongyuan Zha. 2019. DyRep: Learning Representations over Dynamic Graphs. In *International Conference on Learning Representations*.

[38] Rakshit Trivedi, Mehrdad Farajtabar, Prasenjeet Biswal, and Hongyuan Zha. 2019. DyRep: Learning Representations over Dynamic Graphs. In *International Conference on Learning Representations*.

[39] Anton Tsitsulin, Davide Mottin, Panagiotis Karras, and Emmanuel Müller. 2018. Verse: Versatile graph embeddings from similarity measures. In *Proceedings of the 2018 World Wide Web Conference*. 539–548.

[40] Laurens Van der Maaten and Geoffrey Hinton. 2008. Visualizing data using t-SNE. *Journal of machine learning research* 9, 11 (2008).

[41] Ulrike Von Luxburg, Agnes Radl, and Matthias Hein. 2014. Hitting and commute times in large random neighborhood graphs. *The Journal of Machine Learning Research* 15, 1 (2014), 1751–1798.

[42] Kilian Weinberger, Anirban Dasgupta, John Langford, Alex Smola, and Josh Attenberg. 2009. Feature hashing for large scale multitask learning. In *Proceedings of the 26th annual international conference on machine learning*. 1113–1120.

[43] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L Hamilton, and Jure Leskovec. 2018. Graph convolutional neural networks for web-scale recommender systems. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 974–983.

[44] Chengxi Zang and Fei Wang. 2020. Neural dynamics on complex networks. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 892–902.

[45] Hongyang Zhang, Peter Lofgren, and Ashish Goel. 2016. Approximate personalized pagerank on dynamic graphs. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 1315–1324.

[46] Hongyang Zhang, Peter Lofgren, and Ashish Goel. 2016. Approximate Personalized PageRank on Dynamic Graphs. *arXiv preprint arXiv:1603.07796* (2016).

[47] Ziwei Zhang, Peng Cui, Haoyang Li, Xiao Wang, and Wenwu Zhu. 2018. Billion-scale network embedding with iterative random projection. In *2018 IEEE International Conference on Data Mining (ICDM)*. IEEE, 787–796.

[48] Ziwei Zhang, Peng Cui, Jian Pei, Xiao Wang, and Wenwu Zhu. 2018. TIMERS: Error-Bounded SVD Restart on Dynamic Networks. In *Proceedings of the 32nd AAAI Conference on Artificial Intelligence*. AAAI.

[49] Lekui Zhou, Yang Yang, Xiang Ren, Fei Wu, and Yueting Zhuang. 2018. Dynamic network embedding by modeling triadic closure process. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 32.

[50] Linhong Zhu, Dong Guo, Junming Yin, Greg Ver Steeg, and Aram Galstyan. 2016. Scalable temporal latent space inference for link prediction in dynamic social networks. *IEEE Transactions on Knowledge and Data Engineering* 28, 10 (2016), 2765–2777.

A PROOF OF LEMMAS

To prove Lemma 4, we first introduce the property of uniqueness of PPR $\boldsymbol{\pi}_s$ for any s.

Proposition 10 (Uniqueness of PPR [3]). *For any starting vector $\mathbf{1}_s$, and any constant $\alpha \in (0, 1]$, there is a unique vector $\boldsymbol{\pi}_s$ satisfying (3).*

Proof. Recall the PPR equation

$$\boldsymbol{\pi}_s = \alpha \mathbf{1}_s + (1-\alpha)\boldsymbol{D}^{-1}\boldsymbol{A}\boldsymbol{\pi}_s.$$

We can rewrite it as $(\boldsymbol{I}-(1-\alpha)\boldsymbol{D}^{-1}\boldsymbol{A})\boldsymbol{\pi}_s = \alpha\mathbf{1}_s$. Notice the fact that matrix $\boldsymbol{I} - (1-\alpha)\boldsymbol{D}^{-1}\boldsymbol{A}$ is strictly diagonally dominant matrix. To see this, for each $i \in \mathbb{V}$, we have $1-(1-\alpha)\sum_{j\neq i}|1/d(i)| = \alpha > 0$. By [25], strictly diagonally dominant matrix is always invertible. □

Proposition 11 (Symmetry property [27]). *Given any undirected graph $\mathcal{G}$, for any $\alpha \in (0, 1)$ and for any node pair (u, v), we have*

$$d(u)\pi_u(v) = d(v)\pi_v(u). \tag{8}$$

Proof of Lemma 7. Assume there are T iterations. For each forward push operation $t = 1, 2, \ldots T$, we assume the frontier node is u_t, the run time of one push operation is then $d(u_t)$. For total T push operations, the total run time is $\sum_{i=1}^{T} d(u_i)$. Notice that during each push operation, the amount of $\|\boldsymbol{r}_s^{t-1}\|_1$ is reduced at least $\epsilon\alpha d(u_t)$, then we always have the following inequality

$$\epsilon\alpha d(u_t) < \|\boldsymbol{r}_s^{t-1}\|_1 - \|\boldsymbol{r}_s^{t}\|_1$$

Apply the above inequality from $t = 1, 2$, to T, we will have

$$\epsilon\alpha \sum_{t=1}^{T} d(u_t) \leq \|\boldsymbol{r}_s^0\| - \|\boldsymbol{r}_s^T\|_1 = 1 - \|\boldsymbol{r}_s\|_1, \tag{9}$$

where $\boldsymbol{r}_s$ is the final residual vector. The total time is then $O(1/\epsilon\alpha)$. To show the estimation error, we follow the idea of [26]. Notice that, the forward local push algorithm always has the invariant property by Lemma 4, that is

$$\pi_s(u) = p_s(u) + \sum_{v\in V} r_s(v)\pi_v(u), \forall u \in \mathbb{V}. \tag{10}$$

By proposition 11, we have

$$\begin{aligned}
\pi_s(u) &= p_s(u) + \sum_{v\in V} r_s(v)\pi_v(u), \forall u \in \mathbb{V} \\
&= p_s(u) + \sum_{v\in V} r_s(v)\frac{d(u)}{d(v)}\pi_u(v), \forall u \in \mathbb{V} \\
&\leq p_s(u) + \sum_{v\in V} \epsilon d(u)\pi_u(v), \forall u \in \mathbb{V} = p_s(u) + \epsilon d(u),
\end{aligned}$$

where the first inequality by the fact that $r_s(v) \leq \epsilon d(v)$ and the last equality is due to $\|\pi_u\|_1 = 1$. □

Proposition 12 ([45]). *Let $\mathcal{G} = (V, E)$ be undirected and let t be a vertex of V, then $\sum_{x\in V}\frac{\pi_x(t)}{d(t)} \leq 1$.*

Proof. By using Proposition 11, we have

$$\sum_{x\in V}\frac{\pi_x(t)}{d(t)} = \sum_{x\in V}\frac{\pi_t(x)}{d(x)} \leq \sum_{x\in V}\pi_t(x) = 1.$$

□

B HEURISTIC METHOD: COMMUTE

We update the embeddings by their pairwise relationship (resistance distance). The commute distance (i.e. resistance distance) $C_{uv} = H_{uv} + H_{vu}$, where rescaled hitting time H_{uv} converges to $1/d(v)$. As proved in [41], when the number of nodes in the graph is large enough, we can show that the commute distance tends to $1/d_v + 1/d_u$.

Algorithm 4 Commute

Input: An graph $\mathcal{G}^0(\mathcal{V}^0, \mathcal{E}^0)$ and embedding $\boldsymbol{W}^0$, dimension d.
Output: $\boldsymbol{W}^T$
for $e^t(u, v, t) \in \{e^1(u^1, v^1, t_1), \ldots, e^T(u^T, v^T, t_T)\}$ **do**
  Add e^t to $\mathcal{G}^t$
  If $u \notin V^{t-1}$ **then**
    generate $\boldsymbol{w}_u^t = \mathcal{N}(\mathbf{0}, 0.1 \cdot \boldsymbol{I}_d)$ or $\mathcal{U}(-0.5, 0.5)/d$
  If $v \notin V^{t-1}$ **then**
    generate $\boldsymbol{w}_v^t = \mathcal{N}(\mathbf{0}, 0.1 \cdot \boldsymbol{I}_d)$ or $\mathcal{U}(-0.5, 0.5)/d$
  $\boldsymbol{w}_u^t = \frac{d(u)}{d(u)+1}\boldsymbol{w}_u^{t-1} + \frac{1}{d(u)}\boldsymbol{w}_v^t$
  $\boldsymbol{w}_v^t = \frac{d(v)}{d(v)+1}\boldsymbol{w}_v^{t-1} + \frac{1}{d(v)}\boldsymbol{w}_u^t$
Return $\boldsymbol{W}_T$

One can treat the Commute method, i.e. Algorithm 4, as the first-order approximation of RandNE [47]. The embedding generated by RandNE is given as the following

$$\boldsymbol{U} = \left(\alpha_0\boldsymbol{I} + \alpha_1\boldsymbol{A} + \alpha_2\boldsymbol{A}^2 + \ldots + \alpha_q\boldsymbol{A}^q\right)\boldsymbol{R}, \tag{11}$$

where $\boldsymbol{A}$ is the normalized adjacency matrix and $\boldsymbol{I}$ is the identity matrix. At any time t, the dynamic embedding of node i of Commute is given by

$$\begin{aligned}
\boldsymbol{w}_i^t &= \frac{d(u)}{d(u)+1}\boldsymbol{w}_i^{t-1} + \frac{1}{d(u)}\boldsymbol{w}_v^t \\
&= \frac{1}{d(u)+1}\boldsymbol{w}_i^0 + \frac{1}{|\mathcal{N}(i)|}\sum_{j\in\mathcal{N}(i)}\frac{1}{d(u)}\boldsymbol{w}_v^t
\end{aligned}$$

C DETAILS OF DATA PREPROCESSING

In this section, we describe the preprocessing steps of three datasets. **Enwiki20**: In Enwiki20 graph, the edge stream is divided into 6 snapshots, containing edges before 2005, 2005-2008, ..., 2017-2020. The sampled nodes in the first snapshot fall into 5 categories. **Patent**: In full patent graph, we divide edge stream into 4 snapshots, containing patents citation links before 1985, 1985-1990,..., 1995-1999. In node classification tasks, we sampled 3,000 nodes in the first snapshot, which fall into 6 categories. In patent small graph, we divide into 13 snapshots with a 2-year window. All the nodes in each snapshot fall into 6 categories.**Coauthor graph**: In full Coauthor graph, we divide edge stream into 7 snapshots (before 1990, 1990-1995, ..., 2015-2019). The sampled nodes in the first snapshot fall into 9 categories. In Coauthor small graph, the edge stream is divided into 9 snapshots (before 1978, 1978-1983,..., 2013-2017). All the nodes in each snapshot have 14 labels in common.

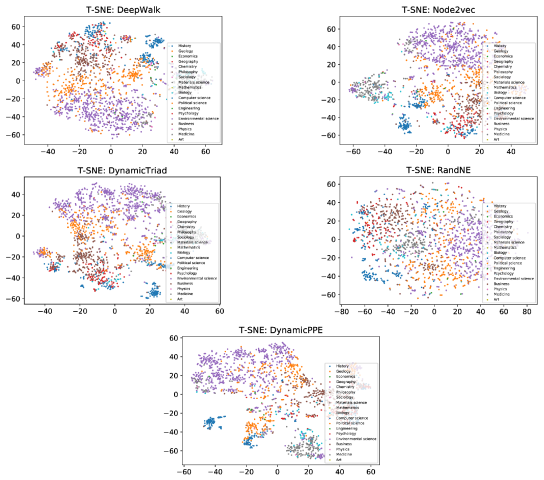


Figure 6: We randomly select 2,000 nodes from Coauthor-small graph and visualize their embeddings using T-SNE[40]

.

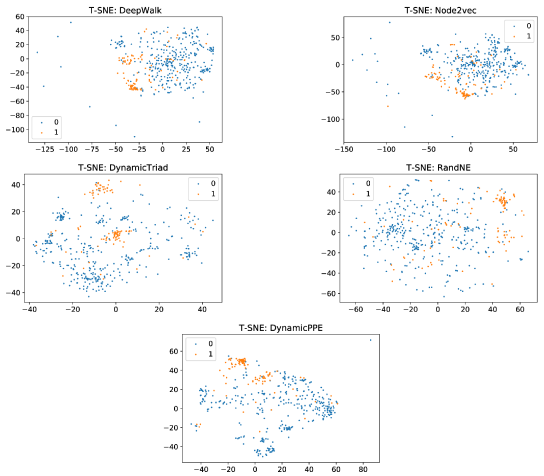


Figure 7: We select all labeled nodes from Academic graph and visualize their embeddings using T-SNE[40]

.

D DETAILS OF PARAMETER SETTINGS

Deepwalk: number-walks=40, walk-length=40, window-size=5
Node2Vec: Same as Deepwalk, p = 0.5, q = 0.5
DynamicTriad: iteration=10, beta-smooth=10, beta-triad=10. Each input snapshot contains the previous existing edges and newly arrived edges.
RandNE: q=3, default weight for node classification $[1, 1e2, 1e4, 1e5]$, input is the transition matrix, the output feature is normalized (l-2 norm) row-wise.
DynamicPPE: $\alpha = 0.15$, $\epsilon = 0.1$, projection method=hash. our method is relatively insensitive to the hyper-parameters.
Infrastructure: 64 CPUs with 16 cores on each (Intel(R) Xeon(R) Silver 4216 CPU @ 2.10GHz) with 376 GB memory.

E VISUALIZATIONS OF EMBEDDINGS

We visualize the embeddings of small scale graphs using T-SNE[40] in Fig.5,6,7.

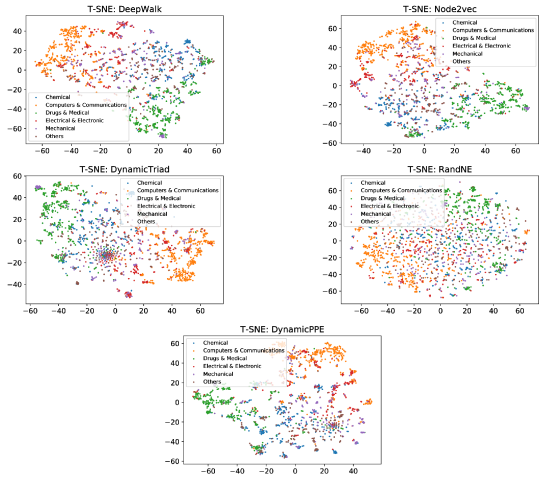


Figure 5: We randomly select 2,000 nodes from patent-small and visualize their embeddings using T-SNE[40]

.