Quality-Aware Streaming Network Embedding with Memory Refreshing

Hsi-Wen Chen[1], Hong-Han Shuai[2], Sheng-De Wang[1], and De-Nian Yang[3,4](✉)

[1] Department of Electrical Engineering, National Taiwan University, Taipei, Taiwan
{r06921045,sdwang}@ntu.edu.tw
[2] Department of Electrical and Computer Engineering, National Chiao Tung University, Hsinchu, Taiwan
hhshuai@nctu.edu.tw
[3] Institute of Information Science, Academia Sinica, Taipei, Taiwan
dnyang@iis.sinica.edu.tw
[4] Research Center for Information Technology Innovation (CITI), Academia Sinica, Taipei, Taiwan

Abstract. Static network embedding has been widely studied to convert sparse structure information into a dense latent space. However, the majority of real networks are continuously evolving, and deriving the whole embedding for every snapshot is computationally intensive. To avoid recomputing the embedding over time, we explore streaming network embedding for two reasons: 1) to efficiently identify the nodes required to update the embeddings under multi-type network changes, and 2) to carefully revise the embeddings to maintain transduction over different parts of the network. Specifically, we propose a new representation learning framework, named *Graph Memory Refreshing (GMR)*, to preserve both global types of structural information efficiently. We prove that GMR maintains the consistency of embeddings (crucial for network analysis) for isomorphic structures better than existing approaches. Experimental results demonstrate that GMR outperforms the baselines with much smaller time.

Keywords: Network embedding · Streaming data mining

1 Introduction

Low-dimensional vector representation of nodes in large-scale networks has been widely applied to a variety of domains, such as social media [13], molecular structure [7], and transportation [9]. Previous approaches, e.g., DeepWalk [13], LINE [16], and SDNE [20], are designed to reduce the sparse structure information to a dense latent space for node classification [13], link prediction [16], and network visualization [21]. However, the above embedding schemes were not designed for evolutionary networks. Current popular networks tend to evolve with time, e.g., the average number of friends increases from 155 in 2016 and to

338 in 2018 [8]. Ephemeral social networks, like Snapchat for short-term conversations, may disappear within weeks. However, retraining the whole embedding for each snapshot is computationally intensive for a massive network. Therefore, streaming network embedding is a desirable option to quickly update and generate new embeddings in a minimum amount of time.

Different from dynamic network embeddings [12,21] that analyze a sequence of networks to capture the temporal patterns, *streaming network embedding*[1] aims to update the network embedding from the changed part of the network to find the new embedding. Efficient streaming network embedding has the following four main challenges. 1) *Multi-type change.* Dynamic changes of networks with insertions and deletions of nodes and edges are usually frequent and complex. It is thus important to derive the new embedding in minimum time to timely reflect the new network status. 2) *Evaluation of affected nodes.* Updating the embeddings of only the nodes neighboring to the changed part ignores the ripple effect on the remaining nodes. It is crucial to identify the nodes required to update the embeddings and ensure that the nodes with similar structures share similar embeddings. 3) *Transduction.* When a network significantly changes, it is difficult to keep the local proximity between the changed part and the remaining part of the network. It is also important to reflect the change in the global structure. 4) *Quality guarantee.* For streaming embeddings based on neural networks (usually regarded as a black box), it is challenging to provide theoretical guarantees about the embedding quality.

To effectively address the above challenges, this paper proposes a new representation learning approach, named *Graph Memory Refreshing (GMR).* GMR first derives the new embedding of the changed part by decomposing the loss function of Skip-Gram to support multi-type changes. It carefully evaluates the ripple-effect area and ensures the correctness by proposing a globally structure-aware selecting strategy, named *hierarchical addressing*, to efficiently identify and update those affected nodes with beam search to avoid the overfitting problem. To effectively support streaming data, our idea is to interpret the update of embeddings as the memory networks with two controllers, a *refreshing gate* and *percolation gate*, to tailor the embeddings from the structural aspect and maintain the transduction. GMR then updates the embeddings according to the streaming information of the new network and the stored features (i.e., memory) of the current network to avoid recomputing the embedding of the whole network. Moreover, GMR aims to both preserve the global structural information and maintain the embeddings of isomorphic structures, i.e., ensuring that the nodes with similar local structures share similar embeddings. This property is essential to ensure the correctness of network analysis based on network embeddings [18]. We theoretically prove that GMR preserves the consistency of embeddings for isomorphic structures better than that of the existing approaches. The contributions of this paper are summarized as follows.

[1] In *streaming* data mining [10], the incoming data stream, instead of the whole dataset, is employed to update the previous mining results efficiently.

- GMR explores streaming network embedding with quality guarantees. The hierarchical addressing, refreshing gate, and percolation gate efficiently find and update the affected nodes under multi-type changes.
- We prove that GMR embedding preserves isomorphic structures better than the existing approaches. According to our literature review, this is the first theoretical analysis for streaming network embedding.
- Experimental results show that GMR outperforms the baselines by at least 10.5% for link prediction and node classification with a much shorter time.

2 Related Work

Static network embedding has attracted a wide range of attention. Laplacian Eigenmaps [1] and IsoMaps [17] first constructed the adjacency matrix and then solved the matrix factorization, but the adjacency matrix was not scalable for massive networks. After Skip-Gram [11] was demonstrated to be powerful for representation learning, DeepWalk [13] and node2vec [5] employed random walks to learn network embedding, while LINE [16] and SDNE [20] were able to preserve the first-order and second-order proximity. GraphSAGE [6] and GAT [19] generated node representations in an inductive manner, by mapping and aggregating node features from the neighborhood.

In addition, a recent line of research proposed to learn the embeddings from a sequence of networks over time for finding temporal behaviors [12,21]. However, these approaches focused on capturing the temporal changes rather than the efficiency since they recomputed the embeddings of the whole network, instead of updating only the changed part. Another line of recent research studied the dynamic embedding without retraining. However, the SVD-based approach [22] was more difficult to support large-scale networks according to [5]. Besides, [10] only supported the edge insertion and ignored edge deletion, whereas the consistency of the embeddings for globally isomorphic structures was not ensured. Compared with the above research and [3], the proposed GMR is the only one that provides a theoretical guarantee on the embedding quality (detailed later). It also more accurately preserves both the global structural information and the consistency of the embeddings.

3 Problem Formulation

In this section, we present the definitions for streaming network embeddings.

Definition 1 (*Streaming Networks*). A dynamic network $\mathcal{G}$ is a sequence of networks $\mathcal{G} = \{G_1, \cdots, G_T\}$ over time, where $G_t = (V_t, E_t)$ is the network snapshot at timestamp t. $\Delta G_t = (\Delta V_t, \Delta E_t)$ represents the streaming network with the changed part ΔV_t and ΔE_t as the sets of vertices and edges inserted or deleted between t and $t + 1$.

Definition 2 (*Streaming Network Embeddings*). Let $z_{i,t}$ denote the streaming network embedding that preserves the structural property of $v_i \in G_t$ at timestamp t. The streaming network embeddings are derived by $\Phi^s = (\phi^s_1, \cdots, \phi^s_{t+1}, \cdots, \phi^s_T)$, where ϕ^s_{t+1} updates the node embedding $z_{i,t+1}$ at timestamp $t+1$ according to $\mathbf{z}_t$ and ΔG_t, i.e., $z_{i,t+1} = \phi^s_{t+1}(\mathbf{z}_t, \Delta G_t)$, where $\mathbf{z}_t = \{z_{i,t} | \forall v_i \in V_t\}$.

In other words, the inputs of the streaming network function are the embedding in the current time and the changed part of the network. In contrast, for [12,21], given a dynamic network $\mathcal{G}$, the embedding is derived by a sequence of functions $\Phi = (\phi_1, \cdots, \phi_{t+1}, \cdots, \phi_T)$, where ϕ_{t+1} maps the node v_i to the d-dimensional embedding $z_{i,t+1}$ at timestamp $t+1$, i.e., $z_{i,t+1} = \phi_{t+1}(v_i, G_{t+1})$. Therefore, the inputs are the whole networks in the current and next time. In the following, we present the problem studied in this paper.

Definition 3 (*Quality-aware Multi-type Streaming Network Embeddings*). Given a streaming network with ΔV_t and ΔE_t as the sets of the vertices and edges inserted or deleted between t and $t+1$, the goal is to find the streaming network embedding and derive the corresponding embedding quality to ensure that the nodes with similar structures share similar embeddings.

Later in Sect. 5, we formally present and theoretically analyze the quality of the embedding with a new metric, named *isomorphic retaining score*. Moreover, we prove that the proposed GMR better preserves the structures than other state-of-the-art methods in Theorems 1.

4 Graph Memory Refreshing

In this section, we propose Graph Memory Refreshing (GMR) to support multi-type embedding updates, to identify the affected nodes required to update the embeddings by hierarchical addressing, and to ensure that the nodes with similar structures share similar embeddings. To effectively support streaming data, we leverage the controllers (refreshing and percolation gates) of *memory networks* [4] to refresh the memory (update the embedding) according to the current state (the current embedding) and new input (streaming network).

4.1 Multi-type Embedding Updating

For each node v_i, the Skip-Gram model predicts the context nodes $v_j \in N(v_i)$ and maximizes the log probability,

$$\sum_{v_i \in V} \sum_{v_j \in N(v_i)} \log p(v_j | v_i). \tag{4.1}$$

However, it is computationally intensive to derive the above probabilities for all nodes. Therefore, the probabilities are approximated by negative sampling [11],

$$\sum_{(v_i, v_j) \in E} \sigma(z_i^T z_j) + \sum_{v_i \in V} \mathbb{E}_{v_j \sim P_N(v_i)}[\sigma(-z_i^T z_j)], \tag{4.2}$$

where $\sigma(x) = 1/(1 + e^{-x})$ is the sigmoid function, z_i and z_j are respectively the embedding vectors of v_i and v_j, and $P_N(v_i)$ is the noise distribution for negative sampling. The two terms respectively model the observed neighborhoods and the negative samples (i.e., node pairs without an edge) drawn from distribution $P_N(v_i)$. However, Eq. (4.2) focuses on only the edge insertion. To support the edge deletion, the second part in Eq. (4.2) is revised to consider unpaired negative samples and the deletion as follows,

$$\sum_{(v_i,v_j)\in E} \sigma(z_i^T z_j) + \sum_{v_i \in V} \mathbb{E}_{v_j \sim P_N(v_i)}[\sigma(-z_i^T z_j)] + \alpha \sum_{(v_i,v_j)\in D} \sigma(-z_i^T z_j), \quad (4.3)$$

where D is the set of deleted edges, and α is required to be set greater than 1 because the samples from D usually provide more information than the unpaired negative samples $P(v_i)$.[2] Note that node deletion is handled by removing all incident edges of a node, while adding a node with new edges is regarded as the edge insertion.[3]

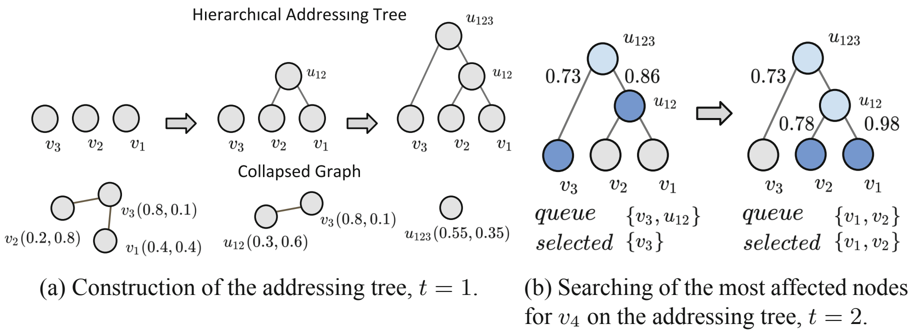

Fig. 1. Example of hierarchical addressing.

4.2 Hierarchical Addressing

For streaming network embedding, previous computationally intensive approaches [4] find the embeddings of *all nodes* by global addressing. A more efficient way is updating only the *neighboring nodes* of the changed part with local addressing [10]. However, the ripple-effect area usually has an arbitrary shape (i.e., including not only the neighboring nodes). Therefore, instead of extracting the neighboring nodes with heuristics, *hierarchical addressing* systematically transforms the original network into a search tree that is aware of the global

[2] Equation (4.3) is introduced as the general form for the Skip-Gram model under the multi-type change, and GMR only samples the insertions/deletions from streaming network ΔG_t at time stamp t for updating the embeddings.

[3] The new node embedding is initialized by the average of its neighborhood [10] and then updated by maximizing Eq.(4.3).

structure for the efficient identification of the affected nodes to update their embeddings.

Hierarchical addressing has the following advantages: 1) *Efficient.* It can be regarded as a series of binary classifications (on a tree), whereas global addressing and local addressing belong to multi-class classification (on the candidate list). Therefore, the time complexity to consider each node in ΔV_t is reduced from $O(|V_t|)$ (i.e., pairwise comparison) to $O(k \log(|V_t|))$, where k is the number of search beams (explained later). 2) *Topology-aware.* It carefully examines the graph structure to evaluate the proximity and maintain the isomorphic structure, i.e., ensuring that the nodes with similar structures share similar embeddings. This property is essential for the correctness of network analysis with network embeddings [18].

Specifically, hierarchical addressing first exploits graph coarsening to build an addressing tree for the efficient search of the affected nodes. Graph coarsening includes both first-hop and second-hop collapsing: first-hop collapsing preserves the first-order proximity by merging two adjacent nodes into a supernode; second-hop collapsing aggregates the nodes with a common neighbor into a supernode, where the embedding of the supernode is averaged from its child nodes [2]. Second-hop collapsing is prioritized because it can effectively compress the network into a smaller tree.

The network is accordingly transformed into an addressing tree with each node $v \in V_t$ as a leaf node. Afterward, for each node $v_i \in \Delta V_t$, we search for the node $v_j \in V_t$ sharing the highest similarity with v_i as the first affected node for v_i by comparing their cosine similarity [4] along the addressing tree. For each node in the tree, if the left child node shares a greater similarity to v_i, the search continues on the left subtree; otherwise, it searches the right subtree. The similarity search ends when it reaches the leaf node with the highest similarity to v_i, and any node in V_t (not only the neighbors of v_i) is thereby allowed to be extracted. In other words, hierarchical addressing enables GMR to extract the affected nodes located in different locations of the network (not necessary to be close to v_i), whereas previous approaches [3,10,21] update only the neighboring nodes of v_i. Afterward, hierarchical addressing extracts the top-1 result for all nodes in ΔV_t as the initially affected nodes (more will be included later), where the nodes with the similarity smaller than a threshold h are filtered. To prevent over-fitting in a local minimum, hierarchical addressing can also extract the top-k results at each iteration with the beam search.[4]

Figure 1 presents an example of hierarchical addressing with the dimension of embeddings as 2. At timestamp $t = 1$ (Fig. 1(a)), we construct the addressing tree by first merging nodes v_1 and v_2 into supernode u_{12} through second-hop collapsing. The embedding of u_{12} is $0.5 \cdot (0.4, 0.4) + 0.5 \cdot (0.2, 0.8) = (0.3, 0.6)$. Afterward, v_3 merges u_{12} into u_{123} through first-hop collapsing, and u_{123} is the root of the tree. At $t = 2$ (Fig. 1(b)), if a new node v_4 is linked to v_1 with the

[4] For each node, the k search beams iteratively examine their child nodes (e.g., total $2k$ nodes) and maintain only the top-k child nodes with the highest similarity in a queue. Any leaf node reached by a beam will be included in the top-k results.

embedding as $(0.3, 0.2)$, we identify the affected nodes with bream search ($k = 2$) and start from the root u_{123}. First, we insert v_3 and u_{12} into the search queue with the size as 2 since $k = 2$, to compare the similarity of v_4 with that of v_3 and u_{12}. Both u_{12} and v_3 are then popped out from the queue because v_1 and v_2 have higher similarity i.e., the top-2 results (0.78 and 0.98), compared with 0.73 for v_3.

4.3 Refresh and Percolate

After identifying the nodes required to update the embeddings by hierarchical addressing, a simple approach is to update the embeddings of those affected nodes with a constant shift [6,20]. However, a streaming network with a topology change on only a subset of nodes usually leads to different shifts for the nodes in distinct locations. Moreover, updating only the nodes extracted from hierarchical addressing is insufficient to ensure consistency of embeddings for the nodes with similar structures when the embeddings are tailored independently.

To effectively support streaming data, inspired by the gating mechanism in GRU [4], we parameterize the update of the embedding according to the current embedding and incoming streaming network. Specifically, GMR decomposes the update procedure into two controller gates: a *refreshing gate* g_r and *percolation gate* g_p. For each node v_j selected in hierarchical addressing for each $v_i \in \Delta V_t$, the refreshing gate first updates the embedding of v_j according the new embedding of v_i, and the percolation gate then updates the embedding for every neighbor v_k of v_j from the new embedding of v_j. The refreshing gate quantifies the embedding update for v_j from an incoming stream (i.e., one-to-one update), while the percolation gate transduces the embedding of v_j to its neighborhoods (i.e., one-to-many update) to preserve better local structure. The two gates are the cornerstones to maintain isomorphic structure, as proved later in the Theorem 1.

To update the embeddings of v_j, i.e., updating $z_{j,t+1}$ from $z_{j,t}$, we first define a shared function a_r to find the refreshing coefficient ρ_r, which represents the correlation between the embedding of v_j and the new embedding of v_i, i.e., $\rho_r = a_r(z_{i,t+1}, z_{j,t})$. The refreshing gate selects the correlation function [19] as the shared function a_r to extract the residual relation [19] between the two embeddings, instead of directly adopting a constant shift as was done in previous work. Here $\boldsymbol{a_r} \in \mathbb{R}^{2d}$ is a shift projection, and ρ_r is derived by $\boldsymbol{a_r}^T[z_{i,t+1}||z_{j,t}]$, where $||$ is the vector concatenation operation. After this, we regulate refreshing coefficient ρ_r into $[0, 1]$ by a sigmoid function $g_r = \sigma(\rho_r)$ to provide a non-linear transformation. Therefore, g_r quantifies the extent that $z_{i,t+1}$ affects $z_{j,t}$,

$$z_{j,t+1} \leftarrow g_r z_{i,t+1} + (1 - g_r) z_{j,t}. \tag{4.4}$$

Thereafter, the percolation gate revises the embedding of the neighbor nodes of v_j to ensure the consistency of the embeddings for the nodes with similar structures. The percolation gate learns another sharable vector $\boldsymbol{a_p} \in \mathbb{R}^{2d}$ and

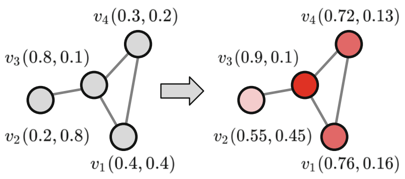

Fig. 2. Example of percolation gate.

finds the percolation coefficient $\rho_p = \boldsymbol{a_p}^T[z_{j,t+1}||z_{k,t}]$, to quantify the extent that v_j affects v_k. Similarly, we regulate ρ_p by $g_p = \sigma(\rho_p)$ to update $z_{k,t}$ as follows,

$$z_{k,t+1} \leftarrow g_p z_{j,t+1} + (1 - g_p) z_{k,t}. \tag{4.5}$$

Therefore, when the refreshing and percolation gates are 0, the streaming network is ignored. In contrast, when both gates become 1, the previous snapshot embedding is dropped accordingly. In summary, the refreshing and percolation gates act as decision makers to learn the impact of the streaming network on different nodes. For the percolation gate, when node v_j is updated, the percolation gate tailors the embedding of each $v_k \in N_1(v_j)$,[5] by evaluating the similarity of v_j and v_k according to the embeddings z_k and z_j. If v_j and v_k share many common neighbors, the percolation value of (v_j, v_k) will increase to draw z_k and z_j closer to each other. The idea is similar for the refreshing gate. Note that $\boldsymbol{a_r}$ and $\boldsymbol{a_p}$ are both differentiable and can be trained in an unsupervised setting by maximize the objective Eq. (4.3). The unsupervised loss can also be replaced or augmented by a task-oriented objective (e.g., cross-entropy loss) when labels are provided. We alternatively update the embeddings (i.e., $z_{i,t}$ and $z_{j,t}$) and the correlation parameters (i.e., $\boldsymbol{a_r}$ and $\boldsymbol{a_p}$) to achieve better convergence.

Figure 2 illustrates an example of updating the node v_3. After the embedding of v_3 updated from $(0.8, 0.1)$ to $(0.9, 0.1)$, GMR uses the percolation gate to transduce the embedding to the neighborhood nodes (i.e., v_1, v_2, and v_4) to preserve the local structure. Since v_1 shares more common neighbors (v_4) with v_3 than v_2 (none), the values of percolation gate for v_1 and v_2 are 0.8 and 0.5, respectively. The embeddings of node v_1 and v_2 become $(0.76, 0.16) = 0.2 \cdot (0.4, 0.4) + 0.8 \cdot (0.9, 0.1)$ and $(0.55, 0.45) = 0.5 \cdot (0.2, 0.8) + 0.5 \cdot (0.9, 0.1)$ through the percolation gate from v_3, respectively. Therefore, relative distance between $\|z_3 - z_2\|$ and $\|z_3 - z_1\|$ can be maintained.

5 Theoretical Analysis

The quality of network embedding can be empirically evaluated from the experiment of network analysis, e.g., link prediction [16] and node classification [13], since the network embedding algorithm is unsupervised learning without knowing the ground truth. In contrast, when the network analysis task is unknown

[5] $N_1(.)$ represents the set of first-hop neighborhoods.

a priori, it is important to theoretically analyze the quality of network embedding. To achieve this goal, we first define the isomorphic pairs and prove that the embeddings of isomorphic pairs are the same in GMR. This property has been regarded as a very important criterion to evaluate the quality of network embedding [18], because the nodes with similar structures are necessary to share similar embeddings. Moreover, the experimental results in Sect. 6 manifest that a higher quality leads to better performance on task-oriented metrics.

Definition 4 (*Isomorphic Pair*). Any two different nodes v_i and v_j form an isomorphic pair if the sets of their first-hop neighbors $N_1(.)$ are the same.

Lemma 1 *If (v_i, v_j) and (v_j, v_k) are both isomorphic pairs, (v_i, v_k) is also an isomorphic pair.*

Proof: According to Definition 4, (v_i, v_j) and (v_j, v_k) are both isomorphic pairs, indicating that $N_1(v_i) = N_1(v_j)$ and $N_1(v_j) = N_1(v_k)$. Therefore, $N_1(v_i)$ is equal to $N_1(v_j)$, and thus (v_i, v_k) is also an isomorphic pair. □

Lemma 2. *The embeddings z_i and z_j are the same after GMR converges if and only if (v_i, v_j) is an isomorphic pair.*

Proof: We first prove the sufficient condition. If (v_i, v_j) is an isomorphic pair with $z_i \neq z_j$, the probability of v_i to predict the context nodes is not to equal to that of v_j (Eq. (4.1)). Therefore, there exists a better solution that makes z_i and z_j be equal, contradicting the condition that the algorithm has converged. For the necessary condition, if $z_i = z_j$ but (v_i, v_j) is not an isomorphic pair, since the probabilities are equal and the algorithm has converged, $N(v_i)$ should be identical to $N(v_j)$ for Eq. (4.1), contradicting that (v_i, v_j) is not an isomorphic pair. The lemma follows. □

As proved in [14], the network embedding algorithms can be unified into the factorization of the affinity matrix. Therefore, nodes with the same first-hop neighborhood have the same embedding when the decomposition ends.

Based on Lemma 2, we define the isomorphic retaining score as follows.

Definition 5 (*Isomorphic Retaining Score*). The isomorphic retaining score, denoted as S_t, is the summation of the cosine similarity over every isomorphic pair in G_t, $S_t \in [-1, 1]$. Specifically,

$$S_t = \frac{1}{|\xi_t|} \sum_{(v_i, v_j) \in \xi_t} s_{ij,t}, \tag{5.1}$$

where $s_{ij,t}$ is the cosine similarity between $z_{i,t}$ and $z_{j,t}$, and ξ_t is the set of isomorphic pairs in G_t. In other words, the embeddings of any two nodes v_i and v_j with the same structure are more consistent to each other if $s_{ij,t}$ is close to 1 [18]. Experiment results in the next section show that higher isomorphic retaining scores lead to better performance of 1) the AUC score for link prediction and 2) the Macro-F1 score for node classification.

The following theorem proves that GMR retains the isomorphic structure better than other Skip-Gram-based approaches, e.g., [5,13,16], under edge insertion. Afterward, the time complexity analysis is presented.

Theorem 1. *GMR outperforms other Skip-Gram-based models regarding the isomorphic retaining score under edge insertion after each update by gradient descent.*

Proof: Due to the space constraint, Theorem 1 is proved in the online version.[6] □

Time Complexity. In GMR, the initialization of the addressing tree involves $O(|V_1|)$ time. For each t, GMR first updates the embeddings of ΔV_t in $O(|\Delta V_t|\log(|\Delta V_t|))$ time. After this, hierarchical addressing takes $O(k|\Delta V_t|\log(|V_t|))$ time to identify the affected nodes. Notice that it requires $O(|\Delta V_t|\log(|V_t|)$ time to update the addressing tree. To update the affected nodes, the refreshing and percolation respectively involve $O(1)$ and $O(d_{max})$ time for one affected node, where d_{max} is the maximum node degree of the network. Therefore, updating all the affected nodes requires $O(kd_{max}|\Delta V_t|)$. Therefore, the overall time complexity of GMR is $O(kd_{max}|\Delta V_t| + k|\Delta V_t|\log(|V_t|))$, while retraining the whole network requires $O(|V_t|\log(|V_t|))$ time at each timestamp. Since k is a small constant, $d_{max} \ll |V_t|$, and $|\Delta V_t| \ll |V_t|$, GMR is faster than retraining.

6 Experiments

To evaluate the effectiveness and efficiency of GMR, we compare GMR with the state-of-the-art methods on two tasks, i.e., link prediction and node classification. For the baselines, we compare GMR with 1) Full, which updates the whole network with DeepWalk [13]; 2) change [3], which only takes the changed part as the samples with DeepWalk;[7] 3) GraphSAGE [6], which derives the embeddings from graph inductive learning; 4) SDNE [20], which extends the auto-encoder model to generate the embeddings of new nodes from the embeddings of neighbors; 5) CTDNE [12], which performs the biased random walk on the dynamic network;[8] and 6) DNE [3], which updates only one affected node; 7) SLA [10], which handles only node/edge insertion; 8) DHPE [22], which is an SVD method based on matrix perturbation theory. The default α, h, k, d, batch size, and learning rate are 1, 0.8, 3, 64, 16, and 0.001, respectively. Stochastic gradient descent (SGD) with Adagrad is adopted to optimize the loss function.

[6] The online version is presented in https://bit.ly/2UUeO7B.

[7] The setting follows OpenNE: https://github.com/thunlp/OpenNE.

[8] For fair comparison, SDNE only takes the adjacency matrix of current stream as the input feature. CTDNE only samples from the latest 50 streams instead of the whole network.

6.1 Link Prediction

For link prediction, three real datasets [15] for streaming networks are evaluated: *Facebook* (63,731 nodes, 1,269,502 edges, and 736,675 timestamps), *Yahoo* (100,001 nodes, 3,179,718 edges, and 1,498,868 timestamps), and *Epinions* (131,828 nodes, 841,372 edges, and 939 timestamps).[9] The concatenated embedding $[z_i||z_j]$ of pair (v_i, v_j) is employed as the feature to predict the link by logistic regression.[10]

Table 1. Experiment results of link prediction.

	Facebook			Yahoo			Epinions		
	AUC	S	sec	AUC	S	sec	AUC	S	sec
GMR	0.7943	0.94	3325	0.7674	0.93	3456	0.9294	0.92	3507
FULL	0.8004	0 95	66412	0.7641	0.95	72197	0.9512	0.96	61133
CHANGE	0.6926	0.79	2488	0.6326	0.82	2721	0.8233	0.84	2429
GraphSAGE	0.6569	0.77	4094	0.6441	0.79	5117	0.8158	0.85	4588
SDNE	0.6712	0.81	7078	0.6585	0.83	7622	0.8456	0.88	6799
CTDNE	0.7091	0.85	4322	0.6799	0.84	5136	0.8398	0.90	5097
DNE	0.7294	0.87	2699	0.6892	0.86	2843	0.8648	0.92	2613
SLA	0.7148	0.86	2398	0.6910	0.86	2438	0.8598	0.91	2569
DHPE	0.7350	0.88	3571	0.7102	0.88	3543	0.8458	0.90	3913

Table 1 reports the AUC [5], isomorphic retaining score S in Eq. (5.1), and running time of different methods.[11] The results show that the proposed GMR achieves the best AUC among all streaming network embedding algorithms. Compared with other state-of-the-art baselines, GMR outperforms other three baselines in terms of AUC by at least 17.1%, 15.7% and 11.3% on *Facebook*, *Yahoo* and *Epinions*, respectively. Besides, GMR is close to that of FULL(1.7% less on *Facebook*, 0.6% more on *Yahoo* and 2.2% less on *Epinions*), but the running time is only 4.7%. Moreover, GraphSAGE has relatively weak performance since it cannot preserve the structural information without node features. The running time of SDNE is 2.1× greater than that of GMR due to the processing of the deep structure, while the AUC of SDNE is at least 12.5% less than that of GMR on all datasets.

[9] *Facebook* and *Epinions* contain both the edge insertion and deletion, represented by "i j -1 t" for removing edge (i, j) at timestamp t. *Yahoo* lacks deletion since it is a message network.

[10] For link prediction, at time t, we predict the new edges for time $t+1$ (excluding the edges incident to the nodes arriving at time $t+1$).

[11] For FULL, due to high computational complexity in retraining the networks for all timestamps, we partition all timestamps into 50 parts [23] with the network changes aggregated in each part.

Compared to other streaming network embedding methods (e.g., DNE, SLA, and DHPE), GMR achieves at least 10.8% of improvement because the embeddings of other methods are updated without considering the global topology. In contrast, GMR selects the affected nodes by globally structure-aware hierarchical addressing, and the selected nodes are not restricted to the nearby nodes. Furthermore, GMR outperforms baselines regarding the isomorphic retraining score since it percolates the embeddings to preserve the structural information. Note that the isomorphic retaining score S is highly related to the AUC with a correlation coefficient of 0.92, demonstrating that it is indeed crucial to ensure the embedding consistency for the nodes with similar structures.

6.2 Node Classification

For node classification, we compare different approaches on *BlogCatalog* [16] (10,132 nodes, 333,983 edges, and 39 classes), *Wiki* [5] (2,405 nodes, 17,981 edges, and 19 classes), and *DBLP* [22] (101,253 nodes, 223,810 edges, 48 timestamps, and 4 classes). *DBLP* is a real streaming network by extracting the paper citation network of four research areas from 1970 to 2017. *BlogCatalog* and *Wiki* are adopted in previous research [3] to generate the streaming networks.[12] The learned embeddings are employed to classify the nodes according to the labels. Cross-entropy is adopted in the loss function for classification with logistic regression. We randomly sample 20% of labels for training and 80% of

Table 2. Experiment results of node classification.

	BlogCatalog			Wiki			DBLP		
	F1	S	sec	F1	S	sec	F1	S	sec
GMR	0.2059	0.90	1998	0.4945	0.92	199	0.7619	0.93	7638
Full	0.2214	0.91	37214	0.5288	0.93	3811	0.7727	0.94	149451
Change	0.1651	0.71	1237	0.3597	0.79	122	0.6841	0.86	5976
GraphSAGE	0.1558	0.81	2494	0.3419	0.82	173	0.6766	0.86	11410
SDNE	0.1723	0.83	2795	0.3438	0.84	266	0.6914	0.87	16847
CTDNE	0.1808	0.84	2923	0.4013	0.85	301	0.7171	0.88	9115
DNE	0.1848	0.86	1547	0.4187	0.86	141	0.7302	0.90	6521
SLA	0.1899	0.87	1399	0.3998	0.85	149	0.7110	0.88	6193
DHPE	0.1877	0.87	2047	0.4204	0.86	215	0.7311	0.90	8159

[12] The streaming network $\mathcal{G} = \{G_1, ..., G_T\}$ is generated from the original network by first sampling half of the original network as G_1. For each timestamp t, ΔG_t is constructed by sampling 200 edges (not in G_{t-1}) from the original network and adding them (and the corresponding terminal nodes) to G_{t-1}, whereas 100 edges of G_{t-1} are deleted.

labels for testing, and the average results from 50 runs are reported.[13] Table 2 demonstrates that GMR outperforms Change by 27.1% regarding Macro-F1 [13], and it is close to Full but with 20.7× speed-up. The Macro-F1 scores of GraphSAGE and SDNE are at least 40% worse than that of GMR, indicating that GraphSAGE and SDNE cannot adequately handle multi-type changes in dynamic networks. Moreover, GMR achieves better improvement on *BlogCatalog* than on *DBLP*, because the density (i.e., the average degree) of *BlogCatalog* is larger, enabling hierarchical addressing of GMR to exploit more structural information for updating multiple nodes. For *DBLP*, GMR also achieves the performance close to Full.

It is worth noting that the isomorphic retaining score S is also positively related to Macro-F1. We further investigate the percentages of isomorphic pairs with the same label on different datasets. The results manifest that 88%, 92% and 97% of isomorphic pairs share the same labels on *BlogCatalog*, *Wiki*, and *DBLP*, respectively. Therefore, it is crucial to maintain the consistency between isomorphic pairs since similar embeddings of isomorphic pairs are inclined to be classified with the same labels.

7 Conclusion

In this paper, we propose GMR for streaming network embeddings featuring the hierarchical addressing, refreshing gate, and percolation gate to preserve the structural information and consistency. We also prove that the embeddings generated by GMR are more consistent than the current network embedding schemes under insertion. The experiment results demonstrate that GMR outperforms the state-of-the-art methods in link prediction and node classification. Moreover, multi-type updates with the beam search improve GMR in both task-oriented scores and the isomorphic retaining score. Our future work will extend GMR to support multi-relations in knowledge graphs.

References

1. Belkin, M., Niyogi, P.: Laplacian eigenmaps and spectral techniques for embedding and clustering. In: Advances in NIPS, pp. 585–591 (2002)
2. Chen, H., Perozzi, B., Hu, Y., Skiena, S.: HARP: hierarchical representation learning for networks. In: Thirty-Second AAAI (2018)
3. Du, L., Wang, Y., Song, G., Lu, Z., Wang, J.: Dynamic network embedding: an extended approach for skip-gram based network embedding. In: IJCAI, pp. 2086–2092 (2018)
4. Graves, A., Wayne, G., Danihelka, I.: Neural turing machines. arXiv preprint arXiv:1410.5401 (2014)
5. Grover, A., Leskovec, J.: node2vec: scalable feature learning for networks. In: Proceedings of the 22nd ACM SIGKDD, pp. 855–864 (2016)

[13] For a new node, only its embedding derived after the arrival is employed in the testing.

6. Hamilton, W., Ying, Z., Leskovec, J.: Inductive representation learning on large graphs. In: Advances in NIPS, pp. 1024–1034 (2017)
7. Jaeger, S., Fulle, S., Turk, S.: Mol2vec: unsupervised machine learning approach with chemical intuition. J. Chem. Inf. Model. **58**(1), 27–35 (2018)
8. Leskovec, J., Krevl, A.: SNAP datasets: Stanford large network dataset collection, June 2014
9. Li, J., Chen, C., Tong, H., Liu, H.: Multi-layered network embedding. In: Proceedings of the 2018 SIAM ICDM, pp. 684–692 (2018)
10. Liu, X., Hsieh, P.C., Duffield, N., Chen, R., Xie, M., Wen, X.: Real-time streaming graph embedding through local actions. In: Proceedings of the WWW, pp. 285–293 (2019)
11. Mikolov, T., Sutskever, I., Chen, K., Corrado, G.S., Dean, J.: Distributed representations of words and phrases and their compositionality. In: Advances in NIPS, pp. 3111–3119 (2013)
12. Nguyen, G.H., Lee, J.B., Rossi, R.A., Ahmed, N.K., Koh, E., Kim, S.: Continuous-time dynamic network embeddings. In: Proceedings of the WWW, pp. 969–976 (2018)
13. Perozzi, B., Al-Rfou, R., Skiena, S.: DeepWalk: online learning of social representations. In: Proceedings of the 20th ACM SIGKDD, pp. 701–710 (2014)
14. Qiu, J., Dong, Y., Ma, H., Li, J., Wang, K., Tang, J.: Network embedding as matrix factorization: unifying DeepWalk, LINE, PTE, and node2vec. In: Proceedings of the WSDM, pp. 459–467 (2018)
15. Rossi, R., Ahmed, N.: The network data repository with interactive graph analytics and visualization. In: Twenty-Ninth AAAI (2015)
16. Tang, J., Qu, M., Wang, M., Zhang, M., Yan, J., Mei, Q.: LINE: large-scale information network embedding. In: Proceedings of the WWW, pp. 1067–1077 (2015)
17. Tenenbaum, J.B., De Silva, V., Langford, J.C.: A global geometric framework for nonlinear dimensionality reduction. Science **290**(5500), 2319–2323 (2000)
18. Tsitsulin, A., Mottin, D., Karras, P., Müller, E.: VERSE: versatile graph embeddings from similarity measures. In: Proceedings of the the WWW, pp. 539–548 (2018)
19. Veličković, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., Bengio, Y.: Graph attention networks. arXiv preprint arXiv:1710.10903 (2017)
20. Wang, D., Cui, P., Zhu, W.: Structural deep network embedding. In: Proceedings of the 22nd ACM SIGKDD, pp. 1225–1234 (2016)
21. Zhou, L., Yang, Y., Ren, X., Wu, F., Zhuang, Y.: Dynamic network embedding by modeling triadic closure process. In: Thirty-Second AAAI (2018)
22. Zhu, D., Cui, P., Zhang, Z., Pei, J., Zhu, W.: High-order proximity preserved embedding for dynamic networks. Trans. Knowl. Data Eng. **30**, 2134–2144 (2018)
23. Zoghi, M., Tunys, T., Ghavamzadeh, M., Kveton, B., Szepesvari, C., Wen, Z.: Online learning to rank in stochastic click models. In: Proceedings of the ICML, pp. 4199–4208 (2017)