Real-Time Streaming Graph Embedding Through Local Actions

Xi Liu
Texas A&M University
xiliu.tamu@gmail.com

Ping-Chun Hsieh
Texas A&M University
pingchun.hsieh@tamu.edu

Nick Duffield
Texas A&M University
duffieldng@tamu.edu

Rui Chen
Samsung Research America
rui.chen1@samsung.com

Muhe Xie
Samsung Research America
muhexie@gmail.com

Xidao Wen
University of Pittsburgh
xidao.wen@pitt.edu

ABSTRACT

Recently, considerable research attention has been paid to graph embedding, a popular approach to construct representations of vertices in latent space. Due to the curse of dimensionality and sparsity in graphical datasets, this approach has become indispensable for machine learning tasks over large networks. The majority of the existing literature has considered this technique under the assumption that the network is static. However, networks in many applications, including social networks, collaboration networks, and recommender systems, nodes, and edges accrue to a growing network as streaming. A small number of very recent results have addressed the problem of embedding for dynamic networks. However, they either rely on knowledge of vertex attributes, suffer high-time complexity or need to be re-trained without closed-form expression. Thus the approach of adapting the existing methods designed for static networks or dynamic networks to the streaming environment faces non-trivial technical challenges.

These challenges motivate developing new approaches to the problems of streaming graph embedding. In this paper, we propose a new framework that is able to generate latent representations for new vertices with high efficiency and low complexity under specified iteration rounds. We formulate a constrained optimization problem for the modification of the representation resulting from a stream arrival. We show this problem has no closed-form solution and instead develop an online approximation solution. Our solution follows three steps: (1) identify vertices affected by newly arrived ones, (2) generating latent features for new vertices, and (3) updating the latent features of the most affected vertices. The new representations are guaranteed to be feasible in the original constrained optimization problem. Meanwhile, the solution only brings about a small change to existing representations and only slightly changes the value of the objective function. Multi-class classification and clustering on five real-world networks demonstrate that our model can efficiently update vertex representations and simultaneously achieve comparable or even better performance compared with model retraining.

ACM Reference Format:
Xi Liu, Ping-Chun Hsieh, Nick Duffield, Rui Chen, Muhe Xie, and Xidao Wen. 2019. Real-Time Streaming Graph Embedding Through Local Actions. In *Companion Proceedings of the 2019 World Wide Web Conference (WWW '19 Companion), May 13–17, 2019, San Francisco, CA, USA.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3308560.3316585

1 INTRODUCTION

Recently graph embedding, also known as graph (a.k.a network) representation learning, has received considerable research attention. That is due to the fact that many real-world problems in complex systems can be modeled as machine learning tasks over large graphs. Direct representations of a vertex in such graphs are usually by its adjacency vector and thus suffer from the curse of dimensionality and sparsity. The idea of graph embedding is to learn a low-dimensional and dense representation for vertices in latent space. The latent representations are learned with the objective of preserving the structural information of the original graph into the geometric relationships among vertices' vector representations [1]. The learned vertex representations are regarded as informative feature inputs to various machine learning tasks. Graph embedding has been proven to be a useful tool for many machine learning tasks, such as vertex classification [2], community detection [3], and link reconstruction [4].

Prior studies have proposed several prominent graph embedding methods [2, 5–9] (see Section 2 for a careful review). Unfortunately, they are subject to three limitations. First, these methods have focused on *static* graphs. However, the majority of real-world networks are naturally dynamic and continuously growing. New vertices, as well as their partial edges, form in a streaming fashion. Such networks are normally referred to as "streaming networks" [10]. Typical examples include social networks and collaboration networks. Those methods ignore the dynamic nature and are unable to update the vertices' embeddings in accordance with networks' evolution. Second, those methods are *transductive*. They require that all vertices in a graph be present during training in order to generate their embeddings and thus cannot generate embeddings for unseen vertices. In streaming networks that constantly encounter new vertices, the *inductive* capability is essential to support diverse machine learning applications. Third, the time complexity of retraining in those methods usually increases linearly with the number of vertices. This makes simple adaptations of the above methods through retraining computationally expensive, let alone the uncertainty of convergence. Indeed, the few very recent works [11–16] adapted from the above methods require either prior knowledge of new vertices' attributes to be inductive or require retraining on the new graph. This presents a challenge for many high-throughput production machine learning systems that need to generate the representations of new vertices in real time. In fact, a streaming network's structure may not change substantially within a short period of time, and retraining over the entire graph is usually unnecessary.

To overcome the aforementioned limitations, we propose a novel efficient real-time representation learning framework for streaming graphs. In this framework, a constrained optimization model is formulated to preserve temporal smoothness and structural proximity in streaming representations. We show that the model belongs to quadratic optimization with orthogonal constraints, which in general has no closed-form solution. Therefore, we propose an online approximation algorithm that is able to inductively generate embeddings for newly arrived vertices and has a closed-form expression. In the online algorithm, we divide the task of streaming graph embedding into three sub-tasks: identifying original vertices that are affected most by the new vertices, calculating the embeddings of the new vertices and adjusting the embeddings of the affected original vertices. Since the change of a streaming graph within a short time period, compared with the entire network, is small, the algorithm only updates the representations of a small proportion of vertices. Moreover, such an update does not require to retrain a model or wait for convergence, has low space and time complexity and thus our method is particularly suitable for high-throughput production machine learning systems.

Contributions. Our research contributions are as follows:

(1) We propose a novel online representation learning framework for streaming graphs based on a constrained optimization model. Our model simultaneously takes into consideration temporal smoothness and structural proximity. Our framework is able to calculate representations of unseen vertices without knowing their attributes.

(2) We devise an approximation algorithm that is able to generate representations in real time for vertices arriving in a streaming manner. This algorithm is highly efficient. In particular, it does not require retraining on the entire network or additional rounds of gradient descent. Moreover, we prove that the generated representations are still feasible in the original optimization problem.

(3) We conduct extensive experiments on five real-world data sets to validate the effectiveness and efficiency of our model in both a supervised learning task (i.e., multi-class classification) and an unsupervised learning task (i.e., clustering). The results demonstrate that the proposed framework can achieve comparable or even better performance than those achieved by retraining.

2 RELATED WORK

Static Network Embedding. Recent developments in modeling practical problems in complex systems by machine learning tasks on large graphs have highlighted the need for graph embedding. In graph embedding, each vertex is mapped to a low-dimensional vector, while preserving a graph's structural information. Current studies in this direction can be roughly categorized by different types of structural information preserved in the mapping. LINE [5] and SDNE [6] preserve the first- and second-order proximities, with the difference that SDNE uses highly non-linear functions to model the mapping. Inspired by recent advances in natural language processing, DeepWalk [7] and *node2vec* [2] preserve higher-order proximities by maximizing the conditional probability of observing the contexts of a vertex given its representations. The crucial difference lies in that *node2vec* follows a biased approach to sample contexts. *struct2vec* [8] proposes to preserve the structural identity between nodes in the representation. To achieve this goal, it first creates a new graph based on the structural identity similarity between nodes and then follows a similar method to DeepWalk on the created graph. A very recent method GraphWave [9] makes use of wavelet diffusion patterns by treating the wavelets from the heat wavelet diffusion process as distributions.

Table 1: Comparison between our method and existing ones

Paper	Time complexity	Need retraining	Need attributes
[1, 13]	-	✓	✓
[17]	$O(k^2(\lvert\mathcal{V}_t\rvert + k))$	×	✓
[18]	$O\big(\lvert\mathcal{V}_t\rvert^2\big)$	×	✓
[19]	$O(\lvert\mathcal{V}_t\rvert k^2 + k^4)$	✓	×
[14]	$O(\lvert\mathcal{V}_t\rvert)$	✓	×
Ours	$O(\beta)$	×	×

Dynamic Network Embedding. Most of the aforementioned studies have focused on static and fixed networks. However, the majority of real-world networks evolve over time and continuously grow. New vertices as well as new edges form in a stream fashion. There are several studies on techniques able to generate embeddings for dynamic graphs. We compare them with our method in terms of three aspects in Table 1: for embedding update at time t (1) the time complexity, (2) whether retraining is needed, and (3) whether vertex attributes is needed. From Table 1 we observe that the time complexity of update at time t is $O(\beta)$, where β is the average degree of graph at time t and can be much smaller than $|\mathcal{V}_t|$, the total number of vertices. Furthermore, the discussion of complexity in [17–19] is assumes a sparse adjacency matrix sparsity, which our methods do not require. Our method has closed-form expression and does not need to re-train the model for embedding updates, and thus does not depend on convergence during training. Finally, our method is able to generate embeddings for new vertices without vertex attributes.

3 PROBLEM FORMULATION

For a streaming graph, we consider new vertices and edges to arrive every with time interval between $t_0 + i\Delta t$ and time $t_0 + (i + 1)\Delta t$ for $i \in \{0, 1, \ldots\}$, where t_0 is the initial time and Δt is the interval width. We will use t_i as a shorthand of $t_0 + i\Delta t$. The number of vertices and their edges that arrive within any Δt can be arbitrary. Let $\mathcal{G}_{t_i} = (\mathcal{V}_{t_i}, \mathcal{E}_{t_i})$ denote the graph consisting of vertices $\mathcal{V}_{t_i}$ and edges $\mathcal{E}_{t_i}$ formed before time t_i. Let $\Delta\mathcal{V}_{t_i}$ and $\Delta\mathcal{E}_{t_i}$ be the vertices and their edges formed between time t_i and t_{i+1}. For any time t_i, adding the vertices $\Delta\mathcal{V}_{t_i}$ and the edges $\Delta\mathcal{E}_{t_i}$ to the graph $\mathcal{G}_{t_i}$ leads to the new graph $\mathcal{G}_{t_{i+1}}$ at time t_{i+1}. For example, consider the "Observed" rectangle in Figure 1. Adding the vertices v_4, v_5, v_6 and their edges (depicted by dashed lines) formed between time t_0 and t_1 to $\mathcal{G}_{t_0}$ leads to $\mathcal{G}_{t_1}$. Let $\mathbf{f}_v^{(t_i)} \in \mathbb{R}^k$ be the embedding of vertex $v \in \mathcal{V}_{t_i}$, where the embedding dimension $k \ll |\mathcal{V}_{t_i}|$. Then at any time t_i, the collection of embeddings of vertices arrived before t_i is denoted by $\{\mathbf{f}_v^{(t_i)}\}_{v\in\mathcal{V}_{t_i}}$. Our objective is to generate embeddings

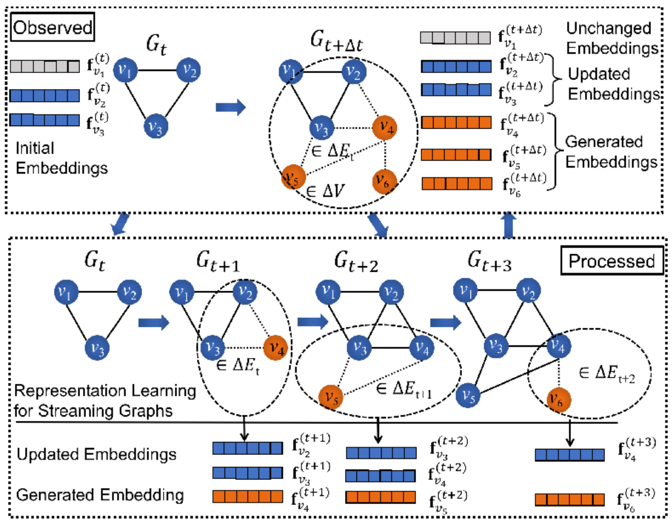


Figure 1: An illustrated example of the proposed embedding technique for streaming graphs with $D = 1$. Gray squares represent unchanged embeddings, blue ones represent updated embeddings for influenced vertices and orange ones represent generated embeddings for new vertices. To be clear, $\mathbf{f}_{v_2}^{(t+\Delta t)} = \mathbf{f}_{v_2}^{(t+1)}$, $\mathbf{f}_{v_3}^{(t+\Delta t)} = \mathbf{f}_{v_3}^{(t+2)}$, $\mathbf{f}_{v_4}^{(t+\Delta t)} = \mathbf{f}_{v_4}^{(t+3)}$, $\mathbf{f}_{v_5}^{(t+\Delta t)} = \mathbf{f}_{v_5}^{(t+2)}$, and $\mathbf{f}_{v_6}^{(t+\Delta t)} = \mathbf{f}_{v_6}^{(t+3)}$.

for the new vertices with a *real-time*, *low-complexity* and *efficient* approach. Now we can formally define the real-time representation learning problem for streaming graphs as follows, using notations described in Table 2.

Definition 1. [Streaming graph embedding] *Consider a graph $\mathcal{G}_{t_0} = (\mathcal{V}_{t_0}, \mathcal{E}_{t_0})$, possibly empty, at initial time t_0. Starting from $i = 0$, a set of vertices $\Delta\mathcal{V}_{t_i}$ and associated edges $\Delta\mathcal{E}_{t_i}$ form in graph $\mathcal{G}_{t_i}$ between time t_i and t_{i+1} and results in a new graph $\mathcal{G}_{t_{i+1}}$ at time t_i. At any time t_{i+1} with $i \in \mathbb{N}$, (1) generate representations $\{\mathbf{f}_v^{(t_{i+1})}\}_{v\in\Delta\mathcal{V}_{t_i}}$ for new vertices $\Delta\mathcal{V}_{t_i}$, and (2) update the representations $\{\mathbf{f}_v^{(t_i)}\}_{v\in\mathcal{V}_{t_i}}$ to $\{\mathbf{f}_v^{(t_{i+1})}\}_{v\in\mathcal{V}_{t_i}}$ for existing vertices $\mathcal{V}_{t_i}$.*

4 METHODS

In this section, we first provide a brief introduction to representation learning for static graphs based on spectral theory and then present our model for streaming graphs. We show that our model belongs to the class of quadratic optimization problem with orthogonality constraints, which has no general closed-form solution. We proposed an approximate solution that has low-complexity, high efficiency and being real-time. The approximated solution is composed of three steps: (1) identify vertices influenced most by the arrival of the new vertices, (2) generate representations of the new vertices, and (3) adjust the representations of the influenced vertices. The approximated solution is inspired by the line-search method on the Stiefel manifold and influence the diffusion process.

4.1 Static Graph Representation Learning

Consider a static graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = \{v_1, v_2, .., v_{|\mathcal{V}|}\}$. Each edge in $\mathcal{E}$ is represented by its two ends, i.e., (v_i, v_j). The

Notations	Descriptions or Definitions
$\mathcal{G}_{t_i}$	Graph consisting of vertices and edges formed before time t_i
$\mathcal{V}_{t_i}$	Vertices formed before time t_i
$\Delta\mathcal{V}_{t_i}$	Vertices formed between time t_i and t_{i+1}
$\mathcal{E}_{t_i}$	Edges formed before time t_i
$\Delta\mathcal{E}_{t_i}$	Edges formed between time t_i and t_{i+1}
k	Embedding dimension
D	Depth of influence $D = \{1, 2, \ldots\}$
$\mathbf{f}_v^{(t_i)}$	$\mathbb{R}^k$ representation of vertex v at time t_i
$\mathbf{A}_{t_i}$	Adjacency matrix of graph $\mathcal{G}_{t_i}$
$\mathbf{D}_{t_i}$	Diagonal matrix of graph $\mathcal{G}_{t_i}$
$\mathbf{L}_{t_i}$	Laplacian matrix of graph $\mathcal{G}_{t_i}$
$\{\lambda_j^{(t_i)}\}_{j=1}^{\lvert\mathcal{V}_{t_i}\rvert}$	Eigenvalues of $\mathbf{D}_{t_i}^{-1}\mathbf{L}_{t_i}$ in ascending order $\lambda_1 \le \lambda_2 \le \ldots \le \lambda_{\lvert\mathcal{V}_{t_i}\rvert}$
$\mathbf{x}_j^{(t_i)}$	$\mathbb{R}^{\lvert\mathcal{V}_{t_i}\rvert}$ eigenvector corresponding to λ_j
$\mathcal{I}_{t_i}(m)$	Set of vertices influenced by vertex m in $\mathcal{G}_{t_i}$
$p_{uv}^{(t+1)}$	Probability that v influences u in graph $\mathcal{G}_{t+1}$
$\lvert\cdot\rvert$	Cardinality of a set
$\lVert\cdot\rVert$	The l_2 norm
$\lVert\cdot\rVert_F$	The Frobenius norm
$tr(\cdot)$	Trace of a matrix

Table 2: Notations and Symbols.

target of spectral theory based graph representation [20] is to keep the representations of two vertices close if they are connected, a reflection of *graph homophily*. Denote the adjacency matrix of $\mathcal{G}$ by $\mathbf{A}$, where $\mathbf{A}(i,j) = 1$ when $(v_i, v_j) \in \mathcal{E}$ and $\mathbf{A}(i,j) = 0$ otherwise. For graph $\mathcal{G}$, this target can be modelled as the optimization problem below:

$$\min_{\mathbf{F}} \mathcal{L}(\mathbf{F}) = \frac{1}{2}\sum_{i,j=1}^{|\mathcal{V}|} \mathbf{A}(i,j)\left\|\mathbf{f}_{v_i} - \mathbf{f}_{v_j}\right\|^2 \quad s.t.\ \mathbf{F}^\top\mathbf{F} = \mathbf{I}_{k\times k}, \tag{1}$$

where the matrix of embeddings $\mathbf{F} \in \mathbb{R}^{|\mathcal{V}|\times k}$ is:

$$\mathbf{F} = \left[(\mathbf{f}_{v_1})^\top, (\mathbf{f}_{v_2})^\top, \cdots, (\mathbf{f}_{v_{|\mathcal{V}|}})^\top\right]^\top \tag{2}$$

Denote the diagonal matrix of $\mathcal{G}$ by $\mathbf{D}$ and denote its element in the i-th row and j-th column by $\mathbf{D}(i,j)$. Then the Laplacian matrix $\mathbf{L} = \mathbf{D} - \mathbf{A}$, where $\mathbf{D}(i,i) = \sum_{j=1}^{|\mathcal{V}|}\mathbf{A}(i,j)$ and $\mathbf{D}(i,j) = 0$ for $i \neq j$. Belkin et al. [21] show that Eq. (1) can be solved by finding the top-k eigenvectors of the following generalized eigen-problem: $\mathbf{L}\mathbf{x} = \lambda\mathbf{D}\mathbf{x}$. Let $\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_{|\mathcal{V}|}$ be the eigenvectors of the corresponding eigenvalues $0 = \lambda_1 \le \lambda_2 \le \ldots \le \lambda_{|\mathcal{V}|}$. It is easy to verify that $\mathbf{1}$ is the only corresponding eigenvector for eigenvalue λ_1. Then the matrix of embeddings can be obtained by $\mathbf{F} = [\mathbf{x}_2, \mathbf{x}_3, ..., \mathbf{x}_{k+1}]$. The time complexity of calculating $\mathbf{F}$ can be as high as $O(k|\mathcal{V}|^2)$ without any sparsity assumption [22].

4.2 Dynamic Graph Representation Learning

For simplicity of presentation, our explanation for dynamic graph representation learning focuses on the case where only one vertex along with part of its edges is added to a graph each time. In fact, with a solution able to handle a single vertex addition at a time, the addition of multiple vertices can be solved by sequentially processing multiple single vertex additions. This is illustrated by the "Processed" rectangle in Figure 1. Processing the addition of

$\{v_4, v_5, v_6\}$ in a batch can be decomposed into the sequential processing of adding v_4 at $t+1$, v_5 at $t+2$ and v_6 at $t+3$, where $t+i$ simply indicates the virtual order of processing and does not have any practical meaning. Therefore, in below discussion, suppose initially at time $t_0 = 1$, the graph is empty and starting from $t_0 = 1$, there is vertex arrival between t and $t+1$ for any $t \geq t_0$. Also suppose $\Delta t = 1$. Then we denote the single vertex and part of its edges that arrive at time t by v_t and $\Delta\mathcal{E}_t$, respectively. Then we have, $\mathcal{V}_t = \{v_1, v_2, ..., v_{t-1}\}$ and $\mathcal{E}_t = \bigcup_{i=1}^{t-1} \Delta\mathcal{E}_i$.

To solve the problem defined in Definition 1, we propose an optimization problem that needs to be solved at time $t = 2, 3, ...$ The objective function of the optimization problem is designed based on two key properties of the graph streams: *temporal smoothness* and *graph homophily*. First, since only one vertex and its edges arrive per time, the dynamic graph will evolve smoothly, most of the representations of the same vertices at two consecutive time steps should be close. This property is referred to as *temporal smoothness*. This property has also been observed and shown to be helpful to improve representation performance in [13]. Suppose that we are at time $t+1$. Then, this property can be modeled by minimizing the following objective function at any time $t+1$:

$$\mathcal{L}_s^{(t+1)}(\mathbf{F}_{t+1}) := \sum_{v_i \in \mathcal{V}_t} \left\| \mathbf{f}_{v_i}^{(t+1)} - \mathbf{f}_{v_i}^{(t)} \right\|^2, \tag{3}$$

which is the sum of squared ℓ_2-norm representation difference for the same vertices in two consecutive graph snapshots $\mathcal{G}_t$ and $\mathcal{G}_{t+1}$.

Second, the target of representation learning suggests that connected vertices should be embedded to close points in the latent representation space. This property is referred to as *graph homophily*. This property has been reflected in the objective function and constraints of the optimization in Eq. (1). Thus, they should be kept for the new graph $\mathcal{G}_{t+1}$. Formally, this property can be modeled by minimizing the following objective function at time $t+1$:

$$\mathcal{L}_h^{(t+1)}(\mathbf{F}_{t+1}) := \frac{1}{2} \sum_{i,j=1}^{|\mathcal{V}_{t+1}|} \mathbf{A}_{t+1}(i,j) \left\| \mathbf{f}_{v_i}^{(t+1)} - \mathbf{f}_{v_j}^{(t+1)} \right\|^2. \tag{4}$$

To take into account these two properties, we include both $\mathcal{L}_s^{(t+1)}$ and $\mathcal{L}_h^{(t+1)}$ in the final objective function and retain the constraint given in Eq. (1). The optimization problem to solve at time $t+1$ can be summarized as follows.

$$\begin{aligned} \min_{\mathbf{F}_{t+1}} \mathcal{L}^{(t+1)}(\mathbf{F}_{t+1}) &= \gamma_s^{(t+1)} \mathcal{L}_s^{(t+1)}(\mathbf{F}_{t+1}) + \gamma_h^{(t+1)} \mathcal{L}_h^{(t+1)}(\mathbf{F}_{t+1}) \\ s.t.\ \mathbf{F}_{t+1}^\top \mathbf{F}_{t+1} &= \mathbf{I}_{k\times k}, \end{aligned} \tag{5}$$

where, the matrix of embeddings $\mathbf{F}_{t+1} \in \mathbb{R}^{|\mathcal{V}_{t+1}|\times k}$ and $\gamma_s^{(t+1)} = 1/|\mathcal{V}_{t+1}|$ and $\gamma_h^{(t+1)} = 1/(4|\mathcal{E}_{t+1}|)$ are trade-off terms for the temporal smoothness loss functions $\mathcal{L}_s^{(t+1)}$ and graph homophily loss function $\mathcal{L}_h^{(t+1)}$. It is straightforward to observe that $\mathcal{L}_h^{(t+1)}(\mathbf{F}_{t+1})$ is convex in $\mathbf{F}_{t+1}$. Since $\mathcal{L}_s^{(t+1)}(\mathbf{F}_{t+1})$ can be expressed by

$$\begin{aligned} \mathcal{L}_s^{(t+1)}(\mathbf{F}_{t+1}) &= \sum_{v_i \in \mathcal{V}_t} \left\| \mathbf{f}_{v_i}^{(t+1)} - \mathbf{f}_{v_i}^{(t)} \right\|^2 = \left\| \mathbf{J}_{t+1}\mathbf{F}_{t+1} - \mathbf{F}_t \right\|_F^2 \\ &= tr\Big((\mathbf{J}_{t+1}\mathbf{F}_{t+1} - \mathbf{F}_t)^\top (\mathbf{J}_{t+1}\mathbf{F}_{t+1} - \mathbf{F}_t) \Big), \end{aligned} \tag{6}$$

where $\mathbf{J}_{t+1} \in \mathbb{R}^{|\mathcal{V}_t|\times|\mathcal{V}_{t+1}|}$ is:

$$\mathbf{J}_{t+1} := \begin{bmatrix} 1 & 0 & \dots & 0 & 0 \\ 0 & 1 & \dots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \dots & 1 & 0 \end{bmatrix}_{|\mathcal{V}_t|\times|\mathcal{V}_{t+1}|}, \tag{7}$$

$\mathcal{L}_s^{(t+1)}(\mathbf{F}_{t+1})$ is also convex. Therefore $\mathcal{L}^{(t+1)}(\mathbf{F}_{t+1})$ the objective function in Eq. (5) is convex in $\mathbf{F}_{t+1}$.

Since the constraints in Eq. (5) are orthogonality constraints, the optimization problem to solve is a general formed quadratic optimization problem under orthogonality constraints. The space defined by the orthogonal constraints is Stiefel manifold. The problem with such format has been widely studied and concluded with no closed-form solution. The state-of-the-art solution is to learn the solution through Riemann gradient approach [23] or line-search method on the Stiefel manifold [24], whose convergence analysis has attracted extensive research attention very recently. However, they are not suitable for streaming setting, because waiting for convergence brings in time uncertainty and gradient-based methods possess unsatisfied time complexity.

4.3 Approximated Algorithm in Graph Streams

Motivated by the aforementioned limitations, we propose an approximated solution that satisfies the low-complexity, efficiency and real-time requirement in a streaming setting. The proposed approximated solution is inspired by an observation of the line-search method. The basic idea of the line-search method for the optimization problem is to search the optimal solution in the tangent space of the Stiefel manifold. We observed that the line-search method based on the polar decomposition-based retraction updates the representation of a vertex through a linear summation of other representations in iterations [24]. In our problem, that means:

$$\mathbf{F}_{t+1}^{(i+1)} = \left(\mathbf{F}_{t+1}^{(i)} + \alpha_i \Gamma^{(i)} \right) \left[\mathbf{I}_{k\times k} + \alpha_i^2 (\Gamma^{(i)})^\top \Gamma^{(i)} \right]^{-1/2}, \tag{8}$$

where α_i is the step size, $\Gamma^{(i)}$ is the search direction in the tangent space of the Stiefel manifold at iteration i, and $\mathbf{F}_{t+1}^{(i)}$ is the matrix of embedding at iteration i. This inspires us to generate new embedding for a vertex from a linear summation of original embedding for other vertices. Meanwhile, the temporal smoothness in the problem indicates that the embeddings of most vertices would not change a lot. Therefore, to reduce the summation complexity, in the approximated solution, we propose to only update the embeddings of vertices that are influenced by the new vertex. We summarize the steps of the approximated solution as follows: (1) identify vertices influenced most by the arrival of the new vertices, (2) generate embeddings of the new vertex, and (3) adjust the embeddings of the influenced vertices.

The task for the first step can be summarized as: given a vertex, identify the set of vertices that are influenced by it. Similar problems have been widely discussed in the field "influence propagation" and "information diffusion" (see [25] for a survey). A marriage between this field and graph representation learning has been shown very successful in a few recent works [9, 26] for static graphs. Therefore, we apply the Weighted Independent Cascade Model (WICM), one of the most widely used models in this field, to model the influence spread. Suppose the influence is spread through multiple rounds when vertex v first becomes influenced at round j, it is given a *single* chance to influence a currently uninfluenced neighbor u at round $j+1$. It succeeds with a probability $p_{uv}^{(t+1)}$. The outcome is independent of the history and of v's influence to other vertices. $p_{uv}^{(t+1)}$ is the probability that v influences u in graph $\mathcal{G}_{t+1}$ and can be estimated through

$$p_{uv}^{(t+1)} := \frac{1}{\sum_{i \in \mathcal{V}_{t+1}} \mathbf{A}_{t+1}(i,u)}, \tag{9}$$

where the denominator is the in-degree of vertex u in graph $\mathcal{G}_{t+1}$. If u has multiple already-influenced neighbors other than v, their attempts are sequenced in an arbitrary order. The new influenced vertices will also have a single chance to influence their neighbors in next round. Denote by D total number of rounds determined by us. The set of influence vertices is determined using Algorithm 1.

Algorithm 1: Influenced vertices identification

```
Input: Graph $\mathcal{G}_{t+1}$, influenced rounds $D$, new vertex $v_t$
Output: $\mathcal{I}_{t+1}(v_t)$ set of vertices influenced by new vertex $v_t$
1  $k = 0, R_k = \{v_t\}$;
2  while $k \leq D$ do
3      $k = k+1, R_{k+1} = \varnothing, \mathcal{I}_{t+1}(v_t) = \varnothing$;
4      for $v \in R_k$ do
5          for $u \in \mathcal{N}_{t+1}(v)$ do
6              Draw $r \sim Bernoulli(p_{uv})$;
7              if $r = 1$ then
8                  $R_{k+1} = R_{k+1} \bigcup \{u\}$;
9              end
10         end
11     end
12     $\mathcal{I}_{t+1}(v_t) = \mathcal{I}_{t+1}(v_t) \bigcup R_{k+1}$;
13 end
14 return $\mathcal{I}_{t+1}(v_t)$
```

There are two benefits in computing $\mathcal{I}_{t+1}(v_t)$ using Algorithm 1. First, by adjusting the value of D we can control the time complexity. When $D = 1$, Algorithm 1 will stop after all neighbors of v_t are visited. This is where $O(\beta)$ comes from. Second, the influence from new vertex v_t is not equal among already-arrived vertices. It is reasonable to hope the representation of a vertex influenced less by v_t has a smaller chance to be updated than those influenced more by v_t. This has already been handled by WICM. Compared to vertices close to the v_t, those far-away are less likely to be included in $\mathcal{I}_{t+1}(v_t)$ because to be included, all the outcomes in line 7 of Algorithm 1 must be 1 along an influence spreading path. We also note that $\mathcal{I}_{t+1}(v_t)$ can be stored and thus computed incrementally. All these benefits enable high efficiency in the streaming setting.

After identifying influenced vertices, following the idea inspired by the line-search method, we generate the embedding for a new vertex through a linear summation of the influenced vertices' embeddings and adjust the original embeddings of influenced vertices. This is detailed in Algorithm 2. The quantity α_{t+1} is determined in a way that the orthogonal constraints in Eq. (5) are satisfied:

$$\alpha_{t+1} := 1 - \sqrt{1 - 1/|\mathcal{I}_{t+1}(v_t)|}. \tag{10}$$

Algorithm 2: Representation generation and update

```
Input: Graph $\mathcal{G}_t$, newly arrived vertex $v_t$, newly arrived
       edges $\Delta\mathcal{E}_t$, matrix of embeddings $\mathbf{F}_t$
Output: Updated matrix of embeddings $\mathbf{F}_{t+1}$
1  Update graph: $\mathcal{V}_{t+1} \leftarrow \mathcal{V}_t \cup \{v_t\}$ and $\mathcal{E}_{t+1} \leftarrow \mathcal{E}_t \cup \Delta\mathcal{E}_t$;
2  Calculate representation for new vertex $v_t$ by:
3  $\mathbf{f}_{v_t}^{(t+1)} = \frac{1}{|\mathcal{I}_{t+1}(v_t)|} \sum_{u \in \mathcal{I}_{t+1}(v_t)} \mathbf{f}_u^{(t)}$;
4  Adjust representations for already-arrived vertices:
5  for $u \in \mathcal{V}_t$ do
6      $\mathbf{f}_u^{(t+1)} = \begin{cases} \mathbf{f}_u^{(t)} - \alpha_{t+1}\mathbf{f}_{v_t}^{(t+1)} & u \in \mathcal{I}_{t+1}(v_t) \\ \mathbf{f}_u^{(t)} & o.w. \end{cases}$
7  end
8  return $\mathbf{F}_{t+1}$
```

Algorithm 2 ensures that the embedding of a vertex is generated when it arrives and will be updated when it is influenced by some vertices that come after it. That makes a connection between vertices that arrive at different orders and preserves the temporal pattern in later update. Since the algorithm will only update the embeddings of influenced vertices, different from those solutions that suffer time uncertainty from retraining, the proposed algorithm guarantees to output $\mathbf{F}_{t+1}$ after $|\mathcal{I}_{t+1}(v_t)|$ operations. Therefore, the time complexity of Algorithm 2 is $O\big(|\mathcal{I}_{t+1}(v_t)|\big)$ and is expected to have small variance in running time. The value of $|\mathcal{I}_{t+1}(v_t)|$ can be controlled through changing value of D. Thus this algorithm enables trade-off between complexity and performance in the streaming setting. As discussed in Section 2, it can be as low as $O(\beta)$ with β denoting the average degree of the graph and $D = 1$.

5 EXPERIMENTS

In this section, we conduct experiments of both multi-class vertex classification and network clustering on five data sets to evaluate the effectiveness and efficiency of the proposed method. We use the indices of vertices as their arriving order and generate their arrived edges at each time randomly to model the streaming scenario. We evaluate our method in terms of the performance of the learning tasks and the running time to generate vertex embeddings. The experiments are structured to answer the following questions:

- Effectiveness: compared to state-of-the-art retraining based approaches, how does the proposed approach perform in supervised learning and unsupervised learning in the streaming setting?
- Efficiency: compared to state-of-the-art retraining based approaches, how fast is the proposed solution able to generate new embeddings?
- Scalability and stability: how stable and scalable is the proposed solution in different-scale networks?

5.1 Data Sets

We use the following five real data sets to validate the propose framework. All of them are publicly available and have been widely used in previous research of static and dynamic graph embedding. The statistics of the datasets are summarized in Table 3.

- **Blog** was collected from the BlogCatalog website, which manages bloggers and their posted blogs. Bloggers follow each other to form network edges. Bloggers categorize their blogs under predefined classes, which are taken as the ground truth of class labels.

- **CiteSeer** is a literature citation network for the selected papers indexed in CiteSeer. Papers are considered as vertices. The paper citation relations are considered as the links in the network and papers are classified into the following six classes: Agents, Artificial Intelligence, Database, Information Retrieval, Machine Learning and Human-Computer Interaction.
- **Cora** also represents a citation network, whose vertices represent publications from 7 classes: Case Based, Genetic Algorithms, Neural Networks, Probabilistic Methods, Reinforcement Learning, Rule Learning, and Theory. Each link is a citation relationship between the two publications.
- **Flickr** was collected from Flickr, an image sharing website hosting images uploaded by users. Users in Flickr interact with others to form edges. User can subscribe different interest groups, which correspond to the class labels. The interest groups, for instance, are "black and white photos".
- **Wiki** contains 2,405 documents from 17 classes and 17,981 links between them.

Table 3: Dataset statistics

Dataset	# of vertices	# of edges	# of classes
Blog	5,196	171,743	6
CiteSeer	3,312	4,732	6
Cora	2,708	5,429	7
Flickr	7,575	239,738	9
Wiki	2,405	17,981	17

5.2 Comparison with Baseline Methods

We compare our approach with the following four graph embedding algorithms. Since they are designed for static graph embedding, the retraining based utility of them has achieved similar performance in graph embedding tasks for dynamic graphs compared to dynamic methods. Many studies on dynamic graph embedding have used them as baseline methods. For instance, *node2vec* in [19, 27–29], NetMF in [17, 19], DeepWalk in [17, 19, 29]. Except for those already tested in existing works, we also compare our solution with a new framework *struct2vec*. For each baseline, a combination of their hyper parameters are tested and the one achieving the best performance is reported as their performance. To be fair, our solution use the same values for the shared hyper parameters. In the following, we refer to "walk length" as wl, "window size" as ws, and representation dimensions as d. The values of hyper parameters for baselines are obtained through grid search of different combinations: $d \in \{10, 20, ..., 200\}$, $wl \in \{10, 20, 30, 40\}$, $ws \in \{3, 5, 7, 10\}$. where the finally chosen values are $d = 90$, $wl = 10$, $ws = 7$.

- **NetMF** [30] obtains graph embeddings through explicitly factorizing the closed-form matrices It has been shown to outperform LINE [5] on several benchmark data sets.
- **DeepWalk** [7] learns graph embeddings by preserving higher-order structural proximity between vertices in the latent space. A pair of vertices are considered similar if they are close in truncated random walks.
- **node2vec** is equipped with biased random walk to provide a trade-off between BFS and DFS. Compared to DeepWalk, it has a more flexible strategy to explore neighborhoods. The ranges of its unique hyper parameters are experimented with: $p \in \{0.5, 1, 1.5, 2\}$, and $q \in \{0.5, 1, 1.5, 2\}$, where $p = 1$, $q = 1$ and the number of walks is 10 are finally reported.
- **struct2vec** [8] learns embeddings by preserving the structural similarities between vertices in the embedded space, where a hierarchical multi-layer graph is used to encode vertex structural similarities at different scales.

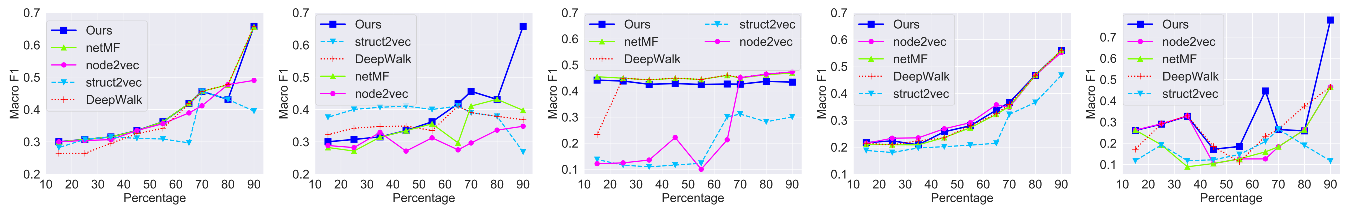
5.3 Supervised Tasks - Vertex Classification

To evaluate the effectiveness of the proposed model, we first compare the performance of our method with different baseline methods on the vertex classification task. The vertex embeddings are fed into a one-vs-rest logistic regression classifier with L2 regularization. In our context, the training percentage (say 20%) specifies an initial portion of arriving vertices for which the offline spectral theory based method in (1) to generate the initial embeddings and then follow Algorithm 2 to learn embeddings for vertices arriving thereafter. For a vertex that arrives in the testing phase, only its embedding obtained upon arrival is used in testing, although later this may be updated, e.g., when there is arrival of its neighbors. As comparison, baseline methods are retrained on all already arrived vertices to generate embeddings for new vertices. Correspondingly, the classifier for the our method will be trained on the embeddings for the first 20% arrived vertices and tested on the remaining 80% arrived vertices. In other words, the classifiers, the baseline methods and the proposed method must be trained and tested on same percentage split of arrival vertices.

5.3.1 Discussion of Effectiveness. We use Micro-F_1 and Macro-F_1 as the evaluation metrics. Figure 2 and Figure 3 compare the Macro-F_1 and Micro-F_1 performance in testing, respectively, with varying percentages of data used for training. We observe that overall, the performance of both the proposed method and the baseline methods improves as the percentage of training increases. This is because as the larger percentage of data is used for training, the embeddings obtained by the proposed methods and retraining based baseline methods are more similar to those by static methods. It can be also observed that our solution achieves almost the same or even slightly better Micro-$F1$ and Macro-F_1 under varying percentages. For example, on CiteSeer, Cora and Wiki data set, our method slightly outperforms baseline methods for most percentages. On Blog and Flickr data set, our method achieves almost the same Micro-F_1 and Macro-F_1 for most percentages.

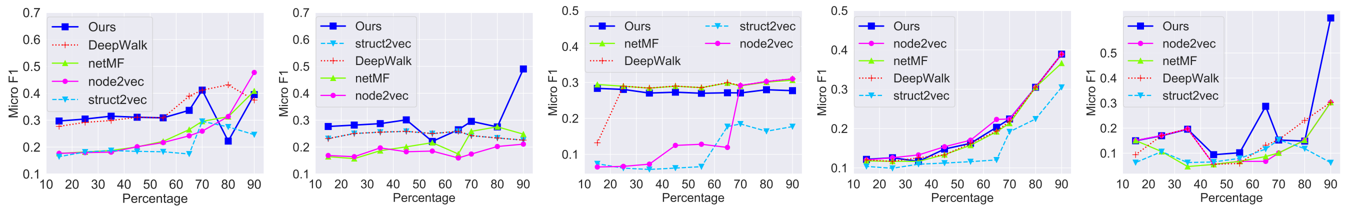
The observation that retraining based baseline methods sometimes are slightly worse than the proposed method indicates that when a graph is highly incomplete, embeddings generated by retraining based baseline methods that considered global structural information may not be as reliable as the proposed method that only considers local updates. This is because the global information baseline methods are highly biased particularly when only a small proportion of the whole graph has arrived. This effect slightly degrades the baseline performance in some cases.

5.3.2 Discussion of Efficiency. We empirically evaluate the running times of different methods in Figure 4. Note that the y-axis is in log scale. The running times for the proposed method count both the times to generate training vertices and testing vertices.



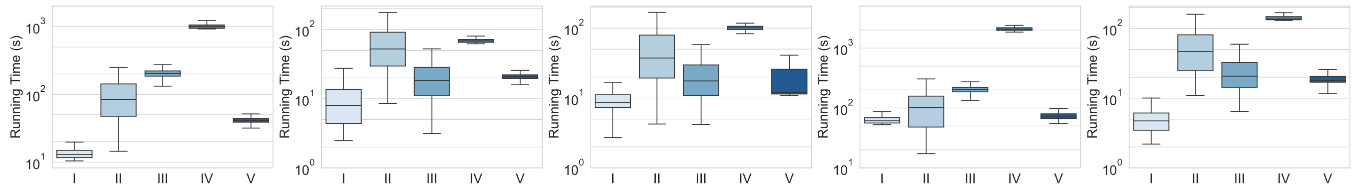
(a) Blog (b) CiteSeer (c) Cora (d) Flickr (e) Wiki

Figure 2: Comparison of vertex multi-class classification performance in Macro-F_1 with $D = 1$.



(a) Blog (b) CiteSeer (c) Cora (d) Flickr (e) Wiki

Figure 3: Comparison of vertex multi-class classification performance in Micro-F_1 with $D = 1$.



(a) Blog (b) CiteSeer (c) Cora (d) Flickr (e) Wiki

Figure 4: Comparison of running time in seconds (I-Ours, II-DeepWalk, III-Node2Vec, IV-Struct2Vec, V-NetMF).

Table 4: Comparison of performance on clustering %

	Blog		CiteSeer		Wiki		Cora		Flickr	
	Completeness	NMI	Completeness	NMI	Completeness	NMI	Completeness	NMI	Completeness	NMI
Ours	16.48	**26.71**	16.48	22.46	16.48	26.71	**34.50**	**34.62**	**16.53**	**20.44**
DeepWalk	**17.77**	20.01	**17.77**	20.01	11.78	11.72	34.62	34.30	16.46	17.31
netMF	0.55	0.67	0.25	0.27	6.85	7.17	6.99	7.49	1.91	2.05
node2vec	16.78	22.93	17.55	**22.93**	**16.60**	**27.77**	31.77	31.60	15.82	21.65
struct2vec	4.35	6.44	2.34	2.44	3.57	4.74	9.80	7.94	7.23	8.04

Since the variation of training percentages influence the running times, we collect running times over all training percentages. The running times for the baseline methods count the times to generate embeddings for all vertices. We observe that, in general when same number of embeddings are generated, the running time of our solution is much smaller. If we compare them under generation at each t, the running time of our solution will be further shorter because our method has lower time complexity as discussed in Section 2. Meanwhile, along with the statistics in Table 3, we note the increased running times of the proposed method is not as large as other baseline methods when the network size increases. That empirically demonstrates scalability of the proposed method. This is because that all retraining baseline methods need to wait for convergence of retraining, whose uncertainty increases as the network size increases, while our method only need to update over its neighbors and guarantee to stop after certain steps.

5.4 Unsupervised Tasks - Network Clustering

Next, we assess the effectiveness of different vertex representations on an unsupervised learning task - network clustering. Since the variation of training percentages influence the performance, we compare the average clustering performance over all training percentages. We use the same embeddings used in vertex classification task. Thus our method's running time is also illustrated in Figure 4. We perform K-means clustering based on the embeddings generated by our method and different baselines. K is chosen to equal the number of classes in each data set. K-means algorithm is repeated 10 times and the average results are reported since K-means may converge to the local minima due to different initialization. We use normalized mutual information (NMI) and completeness score as the performance metrics. They help quantify how close the clustering results are to the ground-truth class belongings. The computation of the two evaluation metrics can be expressed below:

$$NMI := \frac{2I(C;K)}{H(C)+H(K)}, \quad Completeness := 1 - \frac{H(K|C)}{H(K)},$$

where C denotes the class assignment, K denotes the clustering assignment, $I(X;Y) = \sum_{y \in \mathcal{Y}} \sum_{x \in \mathcal{X}} p(x,y) logp(x,y)/p(x)p(y)$ is the mutual information between random variable X and Y and $H(X) = -\sum_{x \in \mathcal{X}} p(x) logp(x)$ is the entropy. The results are summarized in Table 4. Again it can be seen that our method achieves comparable or slightly better performance. For example, our method achieves slightly better performance on Cora and Flickr. As a reminder, Flickr network is the largest among all. Please refer Section 5.3.1 for discussion of the reasons.

6 CONCLUDING REMARKS

We proposed an efficient online representation learning framework for graph streams, in which new vertices and edges arrive as a stream. The framework is inspired by incrementally approximating the solution to a constructed constrained optimization problem, which preserves temporal smoothness and structural proximity in resultant representations. Our approximating solution has closed form, high efficiency, and low complexity, and remains feasible under orthogonality constraints. To validate the effectiveness of our model and learning algorithm, we conducted experiments on five real-world networks for both supervised and unsupervised learning tasks (multi-class classification and clustering) with four baseline methods. Experimental results demonstrate that compared with several state-of-the-art techniques, our approach achieves comparable performance to that of retraining the entire graph with substantially less running time.

ACKNOWLEDGMENTS

This work was supported in part by the National Science Foundation under Grant No. IIS-1848596

REFERENCES

[1] William L Hamilton, Rex Ying, and Jure Leskovec. Representation learning on graphs: Methods and applications. *IEEE Data Engineering Bulletin*, 2017.
[2] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 855–864, 2016.
[3] Ye Li, Chaofeng Sha, Xin Huang, and Yanchun Zhang. Community detection in attributed graphs: An embedding approach. In *Proceedings of the 32nd AAAI Conference on Artificial Intelligence*, pages 338–345, 2018.
[4] Zhu Cao, Linlin Wang, and Gerard de Melo. Link prediction via subgraph embedding-based convex matrix completion. In *Proceedings of the 32nd AAAI Conference on Artificial Intelligence*, pages 2803–2810, 2018.
[5] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. Line: Large-scale information network embedding. In *Proceedings of the 24th International Conference on World Wide Web*, pages 1067–1077, 2015.
[6] Daixin Wang, Peng Cui, and Wenwu Zhu. Structural deep network embedding. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1225–1234, 2016.
[7] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 701–710, 2014.
[8] Leonardo FR Ribeiro, Pedro HP Saverese, and Daniel R Figueiredo. struc2vec: Learning node representations from structural identity. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 385–394, 2017.
[9] Claire Donnat, Marinka Zitnik, David Hallac, and Jure Leskovec. Learning structural node embeddings via diffusion wavelets. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1320–1329, 2018.
[10] Shiyu Chang, Yang Zhang, Jiliang Tang, Dawei Yin, Yi Chang, Mark A Hasegawa-Johnson, and Thomas S Huang. Positive-unlabeled learning in streaming networks. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 755–764, 2016.
[11] Lun Du, Yun Wang, Guojie Song, Zhicong Lu, and Junshan Wang. Dynamic network embedding: An extended approach for skip-gram based network embedding. In *Proceedings of the 17th International Joint Conferences on Artificial Intelligence*, pages 2086–2092, 2018.
[12] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*, pages 1024–1034, 2017.
[13] Xi Liu, Muhe Xie, Xidao Wen, Rui Chen, Yong Ge, Nick Duffield, and Na Wang. A semi-supervised and inductive embedding model for churn prediction of large-scale mobile games. In *IEEE International Conference on Data Mining (ICDM)*, 2018.
[14] Jianxin Ma, Peng Cui, and Wenwu Zhu. Depthlgp: Learning embeddings of out-of-sample nodes in dynamic networks. In *Proceedings of the 32nd AAAI Conference on Artificial Intelligence*, pages 370–377, 2018.
[15] Giang Hoang Nguyen, John Boaz Lee, Ryan A Rossi, Nesreen K Ahmed, Eunyee Koh, and Sungchul Kim. Continuous-time dynamic network embeddings. In *Companion of the The Web Conference 2018 on The Web Conference 2018*, pages 969–976. International World Wide Web Conferences Steering Committee, 2018.
[16] Qixiang Wang, Shanfeng Wang, Maoguo Gong, and Yue Wu. Feature hashing for network representation learning. In *IJCAI*, pages 2812–2818, 2018.
[17] Jundong Li, Harsh Dani, Xia Hu, Jiliang Tang, Yi Chang, and Huan Liu. Attributed network embedding for learning in a dynamic environment. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, pages 387–396, 2017.
[18] Ling Jian, Jundong Li, and Huan Liu. Toward online node classification on streaming networks. *Data Mining and Knowledge Discovery*, 32(1):231–257, 2018.
[19] Dingyuan Zhu, Peng Cui, Ziwei Zhang, Jian Pei, and Wenwu Zhu. High-order proximity preserved embedding for dynamic networks. *IEEE Transactions on Knowledge and Data Engineering*, 2018.
[20] Palash Goyal and Emilio Ferrara. Graph embedding techniques, applications, and performance: A survey. *Knowledge-Based Systems*, 151:78–94, 2018.
[21] Mikhail Belkin and Partha Niyogi. Laplacian eigenmaps and spectral techniques for embedding and clustering. In *Advances in Neural Information Processing Systems*, pages 585–591, 2002.
[22] Johann Paratte and Lionel Martin. Fast eigenspace approximation using random signals. *EPFL-ARTICLE*, 2017.
[23] Zhiqiang Xu and Xin Gao. On truly block eigensolvers via riemannian optimization. In *International Conference on Artificial Intelligence and Statistics*, pages 168–177, 2018.
[24] Huikang Liu, Weijie Wu, and Anthony Man-Cho So. Quadratic optimization with orthogonality constraints: explicit łojasiewicz exponent and linear convergence of line-search methods. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning*, pages 1158–1167, 2016.
[25] Sancheng Peng, Yongmei Zhou, Lihong Cao, Shui Yu, Jianwei Niu, and Weijia Jia. Influence analysis in social networks: a survey. *Journal of Network and Computer Applications*, pages 17–32, 2018.
[26] Yuan Zhang, Tianshu Lyu, and Yan Zhang. Cosine: Community-preserving social network embedding from information diffusion cascades. In *Proceedings of the 32nd AAAI Conference on Artificial Intelligence*, pages 2620–2627, 2018.
[27] Franco Manessi, Alessandro Rozza, and Mario Manzo. Dynamic graph convolutional networks. *arXiv preprint arXiv:1704.06199*, 2017.
[28] Rakshit Trivedi, Mehrdad Farajtbar, Prasenjeet Biswal, and Hongyuan Zha. Representation learning over dynamic graphs. *arXiv preprint arXiv:1803.04051*, 2018.
[29] Le-kui Zhou, Yang Yang, Xiang Ren, Fei Wu, and Yueting Zhuang. Dynamic network embedding by modeling triadic closure process. In *Proceedings of the 32nd AAAI Conference on Artificial Intelligence*, pages 571–578, 2018.
[30] Jiezhong Qiu, Yuxiao Dong, Hao Ma, Jian Li, Kuansan Wang, and Jie Tang. Network embedding as matrix factorization: Unifying deepwalk, line, pte, and node2vec. In *Proceedings of the 11th ACM International Conference on Web Search and Data Mining*, pages 459–467, 2018.