Large Scale Evolving Graphs with Burst Detection

Yifeng Zhao[1], Xiangwei Wang[2], Hongxia Yang[2], Le Song[3] and Jie Tang[1]

[1]Department of Computer Science and Technology, Tsinghua University
[2]DAMO Academy, Alibaba Group
[3]Ant Financial
zhao-yf16@mails.tsinghua.edu.cn, {florian.wxw,yang.yhx}@alibaba-inc.com,
le.song@antfin.com, jietang@tsinghua.edu.cn

Abstract

Analyzing large-scale evolving graphs are crucial for understanding the dynamic and evolutionary nature of social networks. Most existing works focus on discovering repeated and consistent temporal patterns, however, such patterns cannot fully explain the complexity observed in dynamic networks. For example, in recommendation scenarios, users sometimes purchase products on a whim during a window shopping. Thus, in this paper, we design and implement a novel framework called *BurstGraph* which can capture both recurrent and consistent patterns, and especially unexpected bursty network changes. The performance of the proposed algorithm is demonstrated on both a simulated dataset and a world-leading E-Commerce company dataset, showing that they are able to discriminate recurrent events from extremely bursty events in terms of action propensity.

1 Introduction

Dynamic networks, where edges and vertices arrive over time, are ubiquitous in various scenarios, (e.g., social media, security, public health, computational biology and user-item purchase behaviors in the E-Commerce platform [13; 19]), and have attracted significant research interests in recent years. An important problem over dynamic networks is burst detection – finding objects and relationships that are unlike the normal. There are many practical applications spanning numerous domains of burst detection, such as bursty interests of users in E-Commerce [7], cross-community relationships in social networks. Recently, the research community has focused on network embedding learning. One class of the network embedding methods represent nodes as single points in a low-dimensional latent space, which aims to preserve structural and content information of the network [17; 25]. Other classes include edge embedding and subgraph embedding [32; 36]. However, most existing network embedding methods mainly focus on the network structure, ignoring the bursty links appearing in the dynamic networks [17; 25; 24; 21; 37].

In social network dynamics, users may generate consistent temporal patterns by buying consumable goods, such as food and papers, to satisfy their recurrent needs; or purchasing durable products, such as cell phones and cars, to satisfy their longtime needs. However, in the real world, bursty links are very common in network evolution. For instance, in a social network, people will meet new friends or discover new interests if they are in a new environment; in an E-Commerce network, customers often do window shopping when they are exploring recommendation sections. Figure 1 illustrates the interest evolution of the young lady during shopping. However, existing works on modeling dynamic networks mostly focus on repeated and consistent patterns [10; 22; 31; 28; 35], and cannot well capture bursty links due to their sparsity. Such important bursty information is commonly viewed as noisy data in the general machine learning algorithms and ignored in modeling [9]. Furthermore, these bursty dynamics are hidden in other complex network dynamics, including the addition/removal of edges and the update of edge weights. It is challenging to design a framework to account for all these changes.

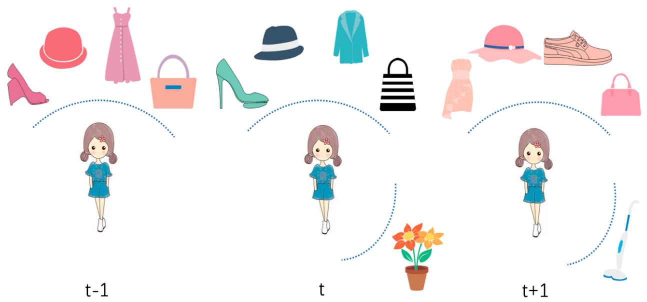

Figure 1: An illustrative example of the observed interest evolution of a young lady during shopping. The main interests of the lady are clothes and shoes, while there also exist burst interests, such as a mop.

To tackle the aforementioned challenges, we propose a novel framework with contributions summarized as follows:

- **Problem Formulation:** we formally define the problem of evolving graphs with bursty links. The key idea is to detect bursty links in dynamic graphs during their onset.
- **Algorithms:** we propose a novel framework for modeling evolving graphs with bursty links, namely *BurstGraph*. *BurstGraph* divides graph evolution into two

parts: vanilla evolution and bursty links' occurrences. For the sparsity of bursty links, a spike-and-slab distribution [2] is introduced as an approximation posterior distribution in the variational autoencoder (VAE)[14] framework, while vanilla evolution accepts the original framework of VAE. To fully exploit the dynamic information in graph evolution, we propose an RNN-based dynamic neural network by capturing graph structures at each time step. The cell of RNN maintains information of both vanilla and bursty evolution, which is updated over time.

The rest of this paper is organized as follows: Section 2 briefly reviews related work. Section 3 introduces the problem statement of evolving graphs with bursty links and presents the proposed framework. Experiments on both simulated dataset and real datasets are presented in Section 4 with discussions. Finally, Section 5 concludes the paper and visions the future work.

2 Related Work

Static Network Embedding. Recently, learning representations for networks has attracted considerable research efforts. Inspired by Word2Vec [15], [17; 20; 25] learn a node representation with its neighborhood contexts. As an adjacency matrix is used to represent the topology of a network, representative works, such as [21; 37], use matrix factorization to learn low-rank space for the adjacency matrix. Deep learning methods [27; 23] are proposed to introduce effective non-linear function learning in network embedding.

Dynamic Network Embedding. Actually, inductive static methods [17; 20] can also handle dynamic networks by making inference of the new vertices. [34] extends the skip-gram methods to update the original vertices' embedding. [35] focuses on capturing the triadic structure properties for learning network embedding. Considering both the network structure and node attributes, [28] focuses on updating the top eigenvectors and eigenvalues for the streaming network.

Burst Detection. Traditionally, burst detection is to detect an unexpectedly large number of events occurring within some time duration. There are two typical types of burst detection approaches, i.e., threshold-based [11] and state-based methods [4]. [3] studies fast algorithms using self-similarity to model bursty time series. [4] uses infinite-state automaton to model the burstiness and extract structure from text streams. However, the link building of evolving graphs is usually sparse and slow, where unexpected links are rare to occur simultaneously. Therefore, our definition of a burst is simply an unexpected behavior within a time duration, which is a straightforward definition adopted by many applications in the real world.

3 The Proposed Model

Existing dynamic network embedding methods mainly focus on the expansion of new vertices and new edges, but usually ignore the changes of vertices' attributes through time that lead to unexpected bursty network changes [6; 12]. To tackle this problem, we propose to learn low-dimensional representations of vertices to capture both vanilla and bursty evolution. Given a dynamic network $\{G_1, ..., G_T\}$, the dynamic network embedding is to learn a series of functions , where each function f_{ϕ_t} maps vertices in network G_t to low-dimensional vectors: $f_{\phi_t}(v_i) \to R^d$ and ϕ_ts are corresponding network parameters. We first summarize symbols and notations in Table 1 and use bold uppercase for matrices (e.g., $\boldsymbol{A}$), bold lowercase for vectors (e.g., $\boldsymbol{a}$), normal lowercase for scalars (e.g., a). $\mathbf{1}$ denotes a vector whose elements are all 1 and $\boldsymbol{I}$ denotes the identity matrix. The framework of the proposed model is illustrated in Figure 2. It mainly consists of two components, namely the original VAE for vanilla evolution and the spike-and-slab model to detect bursty links.

Table 1: Summary of Notation.

Notation	Description
G_t	network snapshot at time step t
$\boldsymbol{A}_t$	adjacency matrix for the network structure in G_t
$\boldsymbol{A}_{v,t}$	adjacency matrix for the vanilla evolution in G_t
$\boldsymbol{A}_{v_i,t}$	adjacency vector for vertex v_i in the vanilla evolution in G_t
$\boldsymbol{A}_{b,t}$	adjacency matrix for the bursty evolution in G_t
$\boldsymbol{A}_{b_i,t}$	adjacency vector for vertex v_i in the bursty evolution in G_t
$\boldsymbol{h}_t$	hidden variable of RNN-based VAE at time step t
$\boldsymbol{z}_t$	random Gaussian variable for vanilla evolution
$\boldsymbol{s}_t$	random spike-and-slab variable for bursty evolution
$\boldsymbol{c}_t, \boldsymbol{r}_{t,1}, \boldsymbol{r}_{t,2}$	composition variables of $\boldsymbol{s}_t$
λ, β	hyperparameters of our method
d	dimension of random variables
n	total number of nodes in G
T	number of time steps

VAE for Graph Evolution After detecting the bursty links at each discrete time t, we split the adjacency matrix of graph $\boldsymbol{A}_t$ into two parts: vanilla adjacency matrix $\boldsymbol{A}_{v,t}$ and burst adjacency matrix $\boldsymbol{A}_{b,t}$. To capture information of these two parts, we extend the framework of VAE and introduce a spike-and-slab distribution to simulate the sparsity of burst adjacency matrix. The loss function of VAE at each time step t is written as:

$$\begin{aligned} L_t =& \ln \prod_{i=1}^{n} p(\boldsymbol{A}_{i,t}|G_{i,t}) = \sum_{i=1}^{n} \ln \int_{\boldsymbol{Z}} \int_{\boldsymbol{S}} p(\boldsymbol{A}_{v_i,t}|\boldsymbol{z}_t) \\ & p(\boldsymbol{A}_{b_i,t}|\boldsymbol{s}_t) p(\boldsymbol{z}_t|G_{i,t}) p(\boldsymbol{s}_t|G_{i,t}) d\boldsymbol{z}_t d\boldsymbol{s}_t, \end{aligned} \tag{1}$$

where $G_{i,t}$ denotes the graph structure of vertex v_i at time step t. $\boldsymbol{z}_t$ and $\boldsymbol{s}_t$ are random variables in the VAE-based model for vanilla and bursty evolution respectively. The evidence lower bound (ELBO) of VAE [14] can be written as:

$$\begin{aligned} L_t =& E_{\boldsymbol{z}_t \sim q_\phi(\boldsymbol{z}_t|G_{i,t})} [\ln \frac{p_\theta(\boldsymbol{A}_{v_i,t}|\boldsymbol{z}_t) p(\boldsymbol{z}_t)}{q_\phi(\boldsymbol{z}_t|G_{i,t})}] \\ & + \lambda \cdot E_{\boldsymbol{s}_t \sim q_\phi(\boldsymbol{s}_t|G_{i,t})} [\ln \frac{p_\theta(\boldsymbol{A}_{b_i,t}|\boldsymbol{s}_t) p(\boldsymbol{s}_t)}{q_\phi(\boldsymbol{s}_t|G_{i,t})}], \end{aligned} \tag{2}$$

where importance weight λ is a hyperparameter. θ and ϕ are parameters of the encoder and decoder networks respectively. The approximate posterior distribution of random variable $\boldsymbol{z}_t$ follows a Gaussian distribution: $q_\phi(\boldsymbol{z}_t|G_{i,t}) \sim \mathcal{N}(\mu_0(G_{i,t}), \Sigma_0(G_{i,t}))$. $\mu_0(\cdot)$ and $\Sigma_0(\cdot)$ are the encoder networks, which can be any highly flexible functions such as neural networks [14]. We use GraphSAGE [29], a representative framework of graph convolutional network, to generate

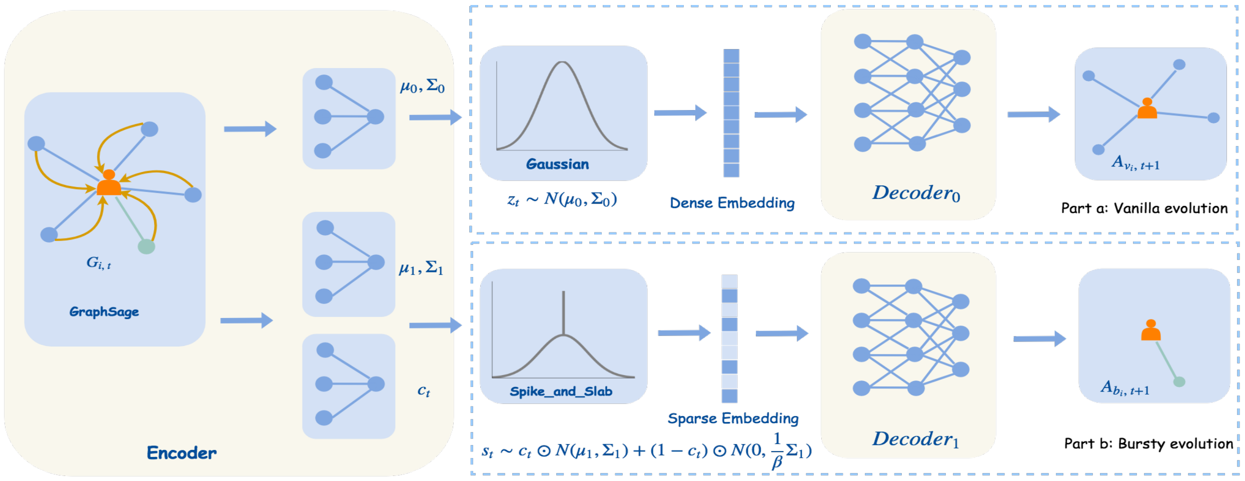

Figure 2: An illustration of the proposed framework *BurstGraph*. At time step t, the framework generates vanilla evolution and bursty evolution based on network structure G_t. Part a is an original VAE for vanilla evolution, where random variable $\boldsymbol{z}_t$ follows a Gaussian distribution. Part b is an extended VAE for bursty evolution, where random variable $\boldsymbol{s}_t$ follows a spike-and-slab distribution because of the sparsity of bursty links. The encoder for these two random variables $\boldsymbol{z}_t$ and $\boldsymbol{s}_t$ shares the same GraphSAGE to utilize the information from vertices and their neighbors.

representation from vertex attributes and its neighbours:

$$\begin{aligned} \boldsymbol{u}_{i,t} &= GraphSAGE(G_{i,t}) \\ \mu_0(G_{i,t}) &= f_{\mu_0}(\boldsymbol{u}_{i,t}) \\ \Sigma_0(G_{i,t}) &= f_{\Sigma_0}(\boldsymbol{u}_{i,t}), \end{aligned} \tag{3}$$

where $\mu_0(G_{i,t})$ and $\Sigma_0(G_{i,t})$ share the same GraphSAGE network to learn representation from topology structure and attributes of vertex v_i. In our paper, $f_{\mu_0}(\cdot)$ and $f_{\Sigma_0}(\cdot)$ are both two fully-connected layers where hidden layers are activated by RELU function.

Similar to the original framework of VAE [14], the prior of random variable $\boldsymbol{z}_t$ follows a standard Gaussian distribution. This is no longer suitable in the case of rare and discrete bursty links. In our paper, the approximate posterior distribution of random variables $\boldsymbol{s}_t$ for bursty links is set to follow a spike-and-slab distribution [2]:

$$\begin{aligned} \boldsymbol{c}_t|G_{i,t} &\overset{iid}{\sim} \text{Bernoulli}(\psi(G_{i,t})) \\ \boldsymbol{r}_{t,1}|G_{i,t} &\sim \mathcal{N}(\boldsymbol{0}, \frac{1}{\beta}\Sigma_1(G_{i,t})) \\ \boldsymbol{r}_{t,2}|G_{i,t} &\sim \mathcal{N}(\mu_1(G_{i,t}), \cdot\Sigma_1(G_{i,t})) \\ \boldsymbol{s}_t &= (\boldsymbol{1} - \boldsymbol{c}_t) \odot \boldsymbol{r}_{t,1} + \boldsymbol{c}_t \odot \boldsymbol{r}_{t,2}, \end{aligned} \tag{4}$$

where $\mu_1(\cdot)$ and $\Sigma_1(\cdot)$ are the encoder networks, with the same neural network structure settings as $\mu_0(\cdot)$ and $\Sigma_0(\cdot)$ in Equation (3), respectively. $\psi(\cdot)$ is also an encoder network which is two fully-connected layers activated by sigmoid function. The value for $\boldsymbol{\beta} > 0$ is predefined with usual setting $\boldsymbol{\beta} = 100$ for more flexbilities [5]. Therefore, $\boldsymbol{r}_{t,1}$ follows a spike distribution while $\boldsymbol{r}_{t,2}$ is a slab distribution. For easy implementation, the prior distribution $p^*(\boldsymbol{s}_t)$ of $\boldsymbol{s}_t$ is set to: $p^*(\boldsymbol{s}_t) \sim (\boldsymbol{1} - \boldsymbol{\alpha}) \cdot \mathcal{N}(\boldsymbol{0}, \frac{1}{100}\boldsymbol{I}) + \boldsymbol{\alpha} \cdot \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$, where $\boldsymbol{\alpha} = \{\alpha_i\}$ and each α_i is drawn from a Bernoulli distribution: $\{\alpha_i\} \overset{iid}{\sim} \text{Bernoulli}(\frac{1}{2})$. The regularization loss of spike-and-slab variable $\boldsymbol{s}_t$ in ELBO can be written as:

$$\begin{aligned} &-D_{KL}(q(\boldsymbol{s}_t|G_{i,t})||p(\boldsymbol{s}_t)) \\ =&E_{q(\boldsymbol{s}_t|G_{i,t})}[\ln\frac{p(\boldsymbol{c}_t)}{q(\boldsymbol{c}_t|G_{i,t})} + \ln\frac{p(\boldsymbol{r}_{t,1})}{q(\boldsymbol{r}_{t,1}|G_{i,t})} + \ln\frac{p(\boldsymbol{r}_{t,2})}{q(\boldsymbol{r}_{t,2}|G_{i,t})}] \\ =&-D_{KL}(q(\boldsymbol{c}_t|G_{i,t})||p(\boldsymbol{c}_t)) - D_{KL}(q(\boldsymbol{r}_{t,1}|G_{i,t})||p(\boldsymbol{r}_{t,1})) \\ &\quad - D_{KL}(q(\boldsymbol{r}_{t,2}|G_{i,t})||p(\boldsymbol{r}_{t,2})) \\ =&-\ln 2 - \psi(G_{i,t})\ln\psi(G_{i,t}) - (1 - \psi(G_{i,t}))\ln(1 - \psi(G_{i,t})) \\ &\quad + \frac{1}{2}(1 + \ln(\Sigma_1(G_{i,t})) - \Sigma_1(G_{i,t}) - \beta\cdot\mu_1^2(G_{i,t})) \\ &\quad + \frac{1}{2}(1 + \ln\Sigma_1(G_{i,t}) - \Sigma_1(G_{i,t}) - \mu_1^2(G_{i,t})), \end{aligned} \tag{5}$$

By using the reparameterization trick, the random variables like $\boldsymbol{z}_t$, $\boldsymbol{c}_t$, $r_{t,1}$ and $r_{t,2}$ can be rewritten as:

$$\begin{aligned} \boldsymbol{z}_t &= \mu_0(G_{i,t}) + \boldsymbol{\gamma}_0 \cdot \Sigma_0^{\frac{1}{2}}(G_{i,t}) \\ \boldsymbol{c}_t &= \sigma(\ln\boldsymbol{\epsilon} - \ln(\boldsymbol{1} - \boldsymbol{\epsilon}) + \ln\psi(G_{i,t}) + \ln(\boldsymbol{1} - \psi(G_{i,t}))) \\ \boldsymbol{r}_{t,1} &= \boldsymbol{\gamma}_1 \cdot (\frac{1}{\beta}\cdot\Sigma_1(G_{i,t}))^{\frac{1}{2}} \\ \boldsymbol{r}_{t,2} &= \mu_1(G_{i,t}) + \boldsymbol{\gamma}_2 \cdot \Sigma_1^{\frac{1}{2}}(G_{i,t}), \end{aligned} \tag{6}$$

where $\boldsymbol{\epsilon}$ follows the uniform distribution $\boldsymbol{\epsilon} \sim \mathcal{U}(\boldsymbol{0}, \boldsymbol{1})$. $\boldsymbol{\gamma}_0$, $\boldsymbol{\gamma}_1$ and $\boldsymbol{\gamma}_2$ all follow the standard Gaussian distribution: $\boldsymbol{\gamma}_0, \boldsymbol{\gamma}_1, \boldsymbol{\gamma}_2 \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$. The decoder network $f_s(\boldsymbol{s}_t) = \sigma(\boldsymbol{W}_s \cdot \boldsymbol{s}_t)$ is a transition function to reconstruct the bursty links $\boldsymbol{A}_{b_i,t}$ in time t. For vanilla evolution, the framework are similar with the original VAE, where the prior of random variable $\boldsymbol{z}_t$ follows a standard Gaussian distribution and the decoder network $f_z(\boldsymbol{z}_t)$ is a fully-connected network activated by sigmoid function to reconstruct the vanilla connection $\boldsymbol{A}_{v_i,t}$ at time t.

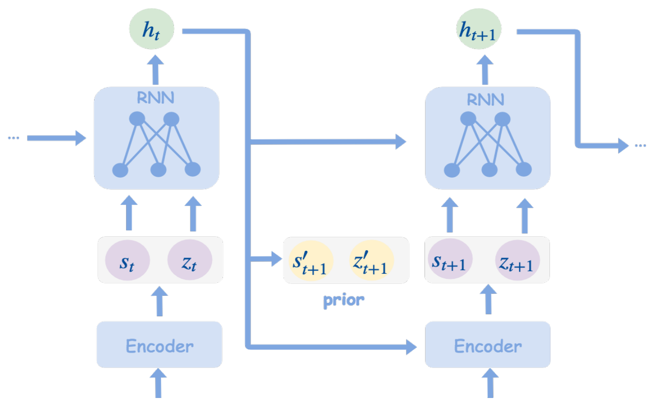

Figure 3: An illustration of the RNN structure in *BurstGraph*, the hidden variable $\boldsymbol{h}_t$ is updated by last hidden variable $\boldsymbol{h}_{t-1}$, random variables $\boldsymbol{z}_t$ and $\boldsymbol{s}_t$. $\boldsymbol{h}_0$ is set to a zero vector. The prior $\boldsymbol{s}'_t$ and $\boldsymbol{z}'_t$ depend on $\boldsymbol{h}_{t-1}$. The initial prior $\boldsymbol{s}'_0$ follows the distribution $p^*(\boldsymbol{s}_t)$ mentioned before, while the initial prior $\boldsymbol{z}'_t$ follows a standard Gaussian distribution.

Learning Representations For Dynamic Network. In the evolving network settings, the goal is to learn the evolving changes from a set of sequential graphs $G = \{G_1, G_2, ..., G_T\}$. To take the temporal structure of the sequential graphs into account, we introduce an RNN structure into our framework as shown in Figure 3. The prior of the latent random variables is not set to follow a standard distribution, but depends on the last hidden variable $\boldsymbol{h}_{t-1}$:

$$\begin{aligned}\boldsymbol{z}_t|\boldsymbol{h}_{t-1} &\sim \mathcal{N}(\mu_0(\boldsymbol{h}_{t-1}), \Sigma_0(\boldsymbol{h}_{t-1}))\\ \boldsymbol{r}_{t,1}|\boldsymbol{h}_{t-1} &\sim \mathcal{N}(\boldsymbol{0}, \frac{1}{\beta}\cdot\Sigma_1(\boldsymbol{h}_{t-1}))\\ \boldsymbol{r}_{t,2}|\boldsymbol{h}_{t-1} &\sim \mathcal{N}(\mu_1(\boldsymbol{h}_{t-1}), \Sigma_1(\boldsymbol{h}_{t-1}))\\ \boldsymbol{c}_t|\boldsymbol{h}_{t-1} &\overset{iid}{\sim} \text{Bernoulli}(\psi(\boldsymbol{h}_{t-1})),\end{aligned} \tag{7}$$

In the similar way, the approximate posterior distribution of these random variables will depend on both the current network snapshot $G_{i,t}$ and the last hidden variable $\boldsymbol{h}_{t-1}$:

$$\begin{aligned}\boldsymbol{z}_t|G_{i,t}, \boldsymbol{h}_{t-1} &\sim \mathcal{N}(\mu_0(G_{i,t}, \boldsymbol{h}_{t-1}), \Sigma_0(G_{i,t}, \boldsymbol{h}_{t-1}))\\ \boldsymbol{r}_{t,1}|G_{i,t}, \boldsymbol{h}_{t-1} &\sim \mathcal{N}(\boldsymbol{0}, \frac{1}{\boldsymbol{\beta}}\cdot\Sigma_1(G_{i,t}, \boldsymbol{h}_{t-1}))\\ \boldsymbol{r}_{t,2}|G_{i,t}, \boldsymbol{h}_{t-1} &\sim \mathcal{N}(\mu_1(G_{i,t}, \boldsymbol{h}_{t-1}), \Sigma_1(G_{i,t}, \boldsymbol{h}_{t-1}))\\ \boldsymbol{c}_t|G_{i,t}, \boldsymbol{h}_{t-1} &\overset{iid}{\sim} \text{Bernoulli}(\psi(G_{i,t}, \boldsymbol{h}_{t-1})),\end{aligned} \tag{8}$$

In the same way of RNN [1], the hidden variable $\boldsymbol{h}_t$ is updated based on the previous hidden variable $\boldsymbol{h}_{t-1}$, the random variables $\boldsymbol{z}_t$ and $\boldsymbol{s}_t$:

$$\boldsymbol{h}_t = f_h(\boldsymbol{h}_{t-1}, \boldsymbol{z}_t, \boldsymbol{s}_t),$$

where $f_h(\cdot)$ is also a fully-connected network to transform the hidden variables.

Training. We adopt Adam optimizer [16] to optimize the objective and also introduce dropout [18] with weight penalties into our proposed model. As expected, we penalize L1-norm of weight W_s to induce the sparsity of the output. It is worth to note that all the parameters of our proposed model are shared along dynamic graphs over time interval $(1, T)$. GraphSAGE in encoder network also shares a random features input for each vertice over time interval $(1, T)$.

The edge evolution is always sparse and unbalanced, which brings trouble to identify the positive instances to achieve better performance. Instead of traditional sigmoid cross entropy loss function, we use inter-and-intra class balanced loss in our model:

$$L = \underbrace{\frac{1}{2}(\frac{L^{nod}_{pos}}{n^{nod}_{pos}} + \frac{L^{nod}_{neg}}{n^{nod}_{neg}})}_{inter-class\ loss} + \underbrace{\frac{1}{2}(\frac{L^{cls}_{pos}}{n^{cls}_{pos}} + \frac{L^{cls}_{neg}}{n^{cls}_{neg}})}_{intra-class\ loss},$$

where L_{pos} and L_{neg} are the cross entropy losses for positive and negative samples, respectively. Similarly, L^{nod} and L^{cls} are the cross entropy losses for labels in each node and class. Similar to the setting of $loss$, n_{pos} and n_{neg} define the number of positive and negative samples, while n^{nod} and n^{cls} define the number of labels in each node and class, respectively.

Table 2: Statistics of datasets.

dataset	#vertices	#edges	#classes	#time
Simulate	20,118	527,268	118	6
A+-Small	19,091	479,441	213	6
A+-Large	25,432	3,745,819	7,419	6

4 Experiment

In this section, we evaluate our model in the dynamic setting from the performance on the multi-class link prediction.

Dataset. We first use the simulated data to verify the effectiveness of the model, then we apply *BurstGraph* to a real challenging dataset from a world-leading E-Commerce company.

- **Simulated Dataset:** We generate a sequence of synthetic bipartite graph snapshots of 21K vertices and 527K edges that share similar characteristics as real data. We assume that each vertex has two types (vanilla/burst) of representations. More specifically, we divide the generation process into two parts: the first part is to generate hyperparameters for each vertex to ensure the independence; the second part is to generate links by sampling two representations of vertices from Gaussian distributions with fixed hyperparameters. First, for each vertex, we sample the hyperparameters μ_0, σ_0^2 from the uniform distributions $\mu_0 \sim \mathcal{U}(0, 2.5)$ and $\sigma_0^2 \sim \mathcal{U}(0, 0.01)$ for the vanilla evolution, and μ_1, σ_1^2 from $\mu_1 \sim \mathcal{U}(-2.5, 0)$ and $\sigma_1^2 \sim \mathcal{U}(0, 1)$ for the bursty evolution respectively. To make sure that bursty links are sparse, the variance σ_1^2 is set to be bigger than σ_0^2. We then generate the links in each snapshot as follows: we first determine the link types of each vertex pair with probability drawn from Bernoulli(0.1) for bursty links and vanilla links otherwise. We then generate the links for each vertex pair. According to their link types, we resample the representations of these two vertices from corresponding Gaussian distributions: $\mathcal{N}(\mu_0, \sigma_0^2)$ or $\mathcal{N}(\mu_1, \sigma_1^2)$. A fixed weight

Table 3: Results(%) comparison of different embedding methods. We use bold to highlight winners.

Model	Simulated				A+-Small				A+-Large			
	vanilla evolution		bursty evolution		vanilla evolution		bursty evolution		vanilla evolution		bursty evolution	
	Micro	Macro	Micro	Macro	Micro	Macro	Micro	Macro	Micro	Macro	Micro	Macro
DeepWalk	73.5	71.2	71.7	69.8	79.0	74.7	71.9	70.2	80.7	70.4	68.1	64.6
GraphSAGE	88.3	87.4	70.8	69.2	74.6	74.2	67.8	67.8	76.1	72.8	69.7	68.5
TNE	75.9	74.3	69.1	68.3	77.6	72.1	70.4	68.2	79.9	73.9	69.1	67.2
CTDNE	72.2	69.6	70.9	68.7	79.3	75.0	72.2	70.4	80.5	70.0	67.8	64.3
BurstGraph	**91.5**	**91.4**	**78.8**	**78.4**	**80.4**	**78.4**	**75.7**	**74.8**	**81.4**	**77.7**	**73.3**	**70.8**

W is employed to transform these two representations into a single value. In all simulated graph snapshots, we set a threshold to truncate around 4% of vertex pairs as links. Results are illustrated in Figure 4.

- **A+ Dataset:** We collect this dataset from a world-leading E-Commerce company A+[1] with two types of nodes, user and item. The bursty link between user and item is defined as if the user has no interaction with the item or similar items in the same category during the last 15 days according to business needs. The other links are viewed as vanilla links. With dynamic graph setting, we split the whole graph into a sequence of graph snapshots with the same time interval (e.g., 1 day). The dataset with sampled items is denoted as A+-Small dataset.

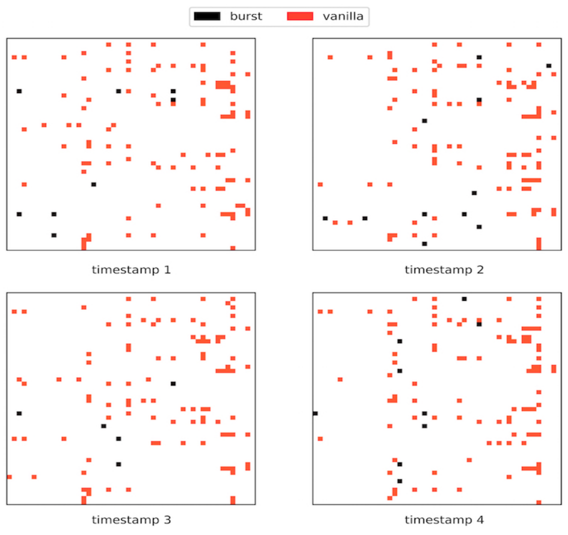

Figure 4: Adjacency matrix heatmap of sub-graphs with four time stamps of the simulated dataset. Each row represents a user, and each column represents an item. The red points represent the vanilla links between users and items, and the black points represent the bursty links. Compared to the bursty evolution, vanilla evolution is more regular and consistent over time.

The statistics of the above datasets are shown in Table 2. In each dataset, graph snapshot G_t consists of all the nodes and interactions that appear at time step t. In our experimental setting, we hide a set of edges from the original graph and train on the remaining graph. The test dataset contains 10% randomly selected vertices, while the negative links are also randomly selected with the same number of positive links for each link type (vanilla/burst).

[1]To preserve anonymity, we use A+ to represent the company.

Baseline Methods We compare our model against the following network embedding algorithms.

- **DeepWalk:** DeepWalk[2] [17] is a representative embedding method for static network. DeepWalk generates truncated random walks and uses Skip-gram algorithm [15] to learn latent representations by treating walks as the equivalent of sentences.
- **GraphSAGE:** GraphSAGE[3] [30] is a general inductive framework of network embedding, which leverages topological structure and attribute information of vertices to efficiently generate vertex embedding. Besides, GraphSAGE can still maintain a good performance even with random features input.
- **CTDNE:** CTDNE [33] is a continuous-time dynamic network embedding method. CTDNE generates temporal random walk as the context information of each vertex and uses Skip-gram algorithm to learn latent representations. In this method, the time of each edge is simply valued according to the number of its snapshot.
- **TNE:** TNE[4] [26] is a dynamic network embedding algorithm based on matrix factorization. Besides, this method holds a temporal smoothness assumption to ensure the continuity of the embeddings in evolving graphs.

It is worthy to mention, DeepWalk and GraphSAGE are static network embedding methods. To facilitate the comparison between our method and these relevant baselines, these two methods are trained with the whole graph, which includes all graph snapshots. In the following, we evaluate the performance of these methods on three datasets in terms of Micro-F1 and Macro-F1.

Comparison Results. Table 3 shows the overall performance of different methods on three datasets. Our model *BurstGraph* is able to consistently outperform all sorts of baselines in various datasets. Next we compare and analyze results on vanilla evolution and burst links, respectively. For the vanilla evolution, *BurstGraph* has a performance gain of +1.7% in terms of Micro-F1 and +3.7% in terms of Macro-F1

[2]https://github.com/phanein/deepwalk

[3]https://github.com/williamleif/GraphSAGE

[4]https://github.com/linhongseba/Temporal-Network-Embedding

on average. For the bursty evolution, *BurstGraph* outperforms other methods +4.8% in terms of Micro-F1 and +5.3% in terms of Macro-F1 averagely. These results show that splitting evolving graphs into vanilla and bursty evolution not only benefits for the performance of the bursty evolution, but also benefits for that of the vanilla evolution. Moreover, compared to other baselines, the performance of *BurstGraph* is quite robust over the three datasets. Notice that DeepWalk, CTDNE and TNE perform poorly on the simulated dataset. One potential reason could be that the simulated generation process may be easier for message passing algorithms (e.g., GraphSAGE) compared to matrix factorization based methods (e.g., DeepWalk, CTDNE or TNE).

Parameter Analysis. We investigate the sensitivity of different hyperparameters in *BurstGraph* including importance weight λ and random variable dimension d. Figure 5 shows the performance of *BurstGraph* on A+-Small dataset when altering the importance weight λ and variable dimension d, respectively. From part a), we can see that the performance of vanilla link prediction is stable when changing the importance weight λ. The performance of burst link prediction rises with the increase of importance weight λ and converges slowly when importance weight is larger than 1. From part b), we can conclude that the performance of *BurstGraph* is relatively stable within a large range of variable dimension, and the performance decreases when the variable dimension is either too small or too large.

Visualization of vertex representations. We visualize the embedding vectors of sampled vertices in the simulated dataset and A+-Small dataset learned by *BurstGraph*. We project the embedding vectors to a 2-dimensional space with t-SNE method [8]. As shown in Figure 6, embeddings from the vanilla evolution can be clearly separated from embeddings of the bursty evolution. More specifically, the embeddings of vertexes in the vanilla evolution evenly spread out in the space, while the embeddings of vertexes in the bursty evolution gather in a relatively small area. The reason could be that the vertex embedding in the vanilla evolution is highly depended on its attributes, while the vertex embedding in the bursty evolution is another way around because of sparsity.

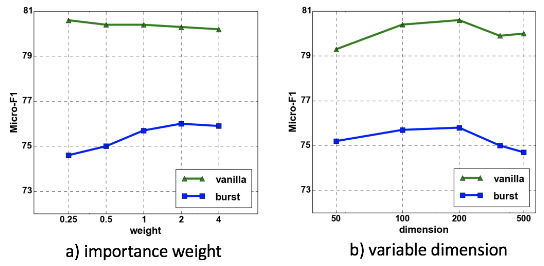

Figure 5: The performance of *BurstGraph* on A+-Small dataset with increasing importance weight λ (left) or variable dimension d (right).

5 Conclusion

In this work, we propose a novel approach for evolving graphs and assume that the evolving of edges in a sequence of graph snapshots can be split into two parts: vanilla and bursty evolution. In addition, these two parts follow two different prior distributions, the vanilla evolution follows a Gaussian distribution and the burst evolution follows a spike-and-slab distribution. Experiment results on real-world datasets show the benefits of our model on bursty links prediction in evolving graphs. However, there still exist limitations in our model. First, only bursty links are considered in our framework. However, there exist other bursty objects, e.g., vertices and communities, which should also be taken into account. We plan to extend our approach to support these bursty objects in the future. Second, we plan to propose a new time series model that supports continuous inputs rather than discretized graph snapshots.

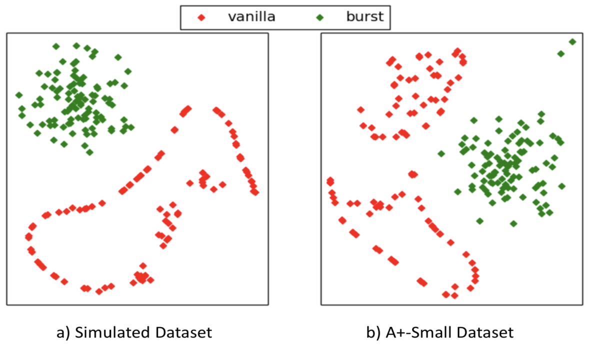

Figure 6: 2D visualization on embeddings (100 randomly selected vertices) for vanilla evolution and bursty evolution. This visualizes the embeddings from vanilla and bursty evolution on Simulated dataset (left) and A+-Small dataset (right). Embeddings from vanilla evolutions are spread out while embeddings from bursty evolutions concentrate in a relatively small area.

References

[1] Jordan, Michael I. Attractor dynamics and parallelism in a connectionist sequential machine. pages, 531–546. Hillsdale, NJ: Erlbaum, 1986.

[2] Mitchell, Toby J and Beauchamp, John J. Bayesian variable selection in linear regression. *Journal of the American Statistical Association*, 83(404):1023–1032, 1988.

[3] Wang, Mengzhi and Madhyastha, Tara and Chan, Ngai Hang and Papadimitriou, Spiros and Faloutsos, Christos. Data mining meets performance evaluation: Fast algorithms for modeling bursty traffic. pages, 507–516. IEEE, 2002.

[4] Kleinberg, Jon. Bursty and hierarchical structure in streams. *Data Mining and Knowledge Discovery*, 7(4):373–397, 2003.

[5] Ishwaran, Hemant and Rao, J Sunil and others. Spike and slab variable selection: frequentist and bayesian strategies. *The Annals of Statistics*, 33(2):730–773, 2005.

[6] Hill, Shawndra and Agarwal, D.K. and Bell, R. and Volinsky, C. Building an effective representation for dynamic networks. pages, 584–608. Taylor & Francis, 2006.

[7] Parikh, Nish and Sundaresan, Neel. Scalable and near real-time burst detection from ecommerce queries. pages, 972–980. ACM, 2008.

[8] Maaten, Laurens van der and Hinton, Geoffrey. Visualizing data using t-sne. *Journal of machine learning research*, 9(Nov):2579–2605, 2008.

[9] Chandola, Varun and Banerjee, Arindam and Kumar, Vipin. Anomaly detection: A survey. *ACM Comput. Surv.*, 41(3):15:1–15:58, July 2009.

[10] Fu, Wenjie and Song, Le and Xing, Eric P. Dynamic mixed membership blockmodel for evolving networks. pages, 329–336. ACM, 2009.

[11] Heard, Nicholas A and Weston, David J and Platanioti, Kiriaki and Hand, David J and others. Bayesian anomaly detection methods for social networks. *The Annals of Applied Statistics*, 4(2):645–662, 2010.

[12] Angel, Albert and Sarkas, Nikos and Koudas, Nick and Srivastava, Divesh. Dense subgraph maintenance under streaming edge weight updates for real-time story identification. *Proc. VLDB Endow.*, 5(6):574–585, 2012.

[13] Akoglu, Leman and Faloutsos, Christos. Anomaly, event, and fraud detection in large network datasets. pages, 773–774. ACM, 2013.

[14] Kingma, Diederik P and Welling, Max. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[15] Mikolov, Tomas and Sutskever, Ilya and Chen, Kai and Corrado, Greg S and Dean, Jeff. Distributed representations of words and phrases and their compositionality. pages, 3111–3119, 2013.

[16] Kingma, Diederik P and Ba, Jimmy. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[17] Perozzi, Bryan and Al-Rfou, Rami and Skiena, Steven. Deepwalk: Online learning of social representations. pages, 701–710. ACM, 2014.

[18] Srivastava, Nitish and Hinton, Geoffrey and Krizhevsky, Alex and Sutskever, Ilya and Salakhutdinov, Ruslan. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[19] Akoglu, Leman and Tong, Hanghang and Koutra, Danai. Graph based anomaly detection and description: a survey. volume, pages, 626–688. ACM, 2015.

[20] Tang, Jian and Qu, Meng and Wang, Mingzhe and Zhang, Ming and Yan, Jun and Mei, Qiaozhu. Line: Large-scale information network embedding. pages, 1067–1077. International World Wide Web Conferences Steering Committee, 2015.

[21] Ou, Mingdong and Cui, Peng and Pei, Jian and Zhang, Ziwei and Zhu, Wenwu. Asymmetric transitivity preserving graph embedding. pages, 1105–1114. ACM, 2016.

[22] Dai, Hanjun and Wang, Yichen and Trivedi, Rakshit and Song, Le. Deep coevolutionary network: Embedding user and item features for recommendation. *arXiv preprint arXiv:1609.03675*, 2016.

[23] Cao, Shaosheng and Lu, Wei and Xu, Qiongkai. Deep neural networks for learning graph representations. pages, 1145–1152, 2016.

[24] Dai, Hanjun and Dai, Bo and Song, Le. Discriminative embeddings of latent variable models for structured data. pages, 2702–2711, 2016.

[25] Grover, Aditya and Leskovec, Jure. node2vec: Scalable feature learning for networks. pages, 855–864. ACM, 2016.

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

[27] Wang, Daixin and Cui, Peng and Zhu, Wenwu. Structural deep network embedding. pages, 1225–1234. ACM, 2016.

[28] Li, Jundong and Dani, Harsh and Hu, Xia and Tang, Jiliang and Chang, Yi and Liu, Huan. Attributed network embedding for learning in a dynamic environment. pages, 387–396. ACM, 2017.

[29] Hamilton, Will and Ying, Zhitao and Leskovec, Jure. Inductive representation learning on large graphs. pages, 1024–1034, 2017.

[30] Hamilton, William L. and Ying, Rex and Leskovec, Jure. Inductive representation learning on large graphs. In *NIPS*, 2017.

[31] Trivedi, Rakshit and Dai, Hanjun and Wang, Yichen and Song, Le. Know-evolve: Deep temporal reasoning for dynamic knowledge graphs. pages, 3462–3471. JMLR. org, 2017.

[32] Dong, Yuxiao and Chawla, Nitesh V and Swami, Ananthram. metapath2vec: Scalable representation learning for heterogeneous networks. pages, 135–144. ACM, 2017.

[33] Nguyen, Giang Hoang and Lee, John Boaz and Rossi, Ryan A and Ahmed, Nesreen K and Koh, Eunyee and Kim, Sungchul. Continuous-time dynamic network embeddings. pages, 969–976. International World Wide Web Conferences Steering Committee, 2018.

[34] Du, Lun and Wang, Yun and Song, Guojie and Lu, Zhicong and Wang, Junshan. Dynamic network embedding: An extended approach for skip-gram based network embedding. pages, 2086–2092, 2018.

[35] Zhou, Lekui and Yang, Yang and Ren, Xiang and Wu, Fei and Zhuang, Yueting. Dynamic network embedding by modeling triadic closure process. 2018.

[36] Cao, Zhu and Wang, Linlin and de Melo, Gerard. Link prediction via subgraph embedding-based convex matrix completion. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[37] Qiu, Jiezhong and Dong, Yuxiao and Ma, Hao and Li, Jian and Wang, Kuansan and Tang, Jie. Network embedding as matrix factorization: Unifying deepwalk, line, pte, and node2vec. pages, 459–467. ACM, 2018.