TGNet: Learning to Rank Nodes in Temporal Graphs

Qi Song*, Bo Zong†, Yinghui Wu*§, Lu-An Tang†, Hui Zhang†, Guofei Jiang†, Haifeng Chen†
Washington State University* NEC Laboratories America† Pacific Northwest National Laboratory§
{qi.song, yinghui}@wsu.edu* {bzong, ltang, huizhang, gfj, haifeng}@nec-labs.com†

ABSTRACT

Node ranking in temporal networks are often impacted by heterogeneous context from node content, temporal, and structural dimensions. This paper introduces **TGNet**, a deep learning framework for node ranking in heterogeneous temporal graphs. TGNet utilizes a variant of Recurrent Neural Network to adapt context evolution and extract context features for nodes. It incorporates a novel influence network to dynamically estimate temporal and structural influence among nodes over time. To cope with label sparsity, it integrates graph smoothness constraints as a weak form of supervision. We show that the application of TGNet is feasible for large-scale networks by developing efficient learning and inference algorithms with optimization techniques. Using real-life data, we experimentally verify the effectiveness and efficiency of TGNet techniques. We also show that TGNet yields intuitive explanations for applications such as alert detection and academic impact ranking, as verified by our case study.

KEYWORDS

Graph ranking, neural network, deep learning

ACM Reference Format:
Qi Song, Bo Zong, Yinghui Wu, Lu-An Tang, Hui Zhang, Guofei Jiang, Haifeng Chen. 2018. TGNet: Learning to Rank Nodes in Tempo-ral Graphs. In *The 27th ACM International Conference on Information and Knowledge Management (CIKM '18), October 22–26, 2018, Torino, Italy*. ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3269206.3271698

1 INTRODUCTION

Temporal graphs have been widely applied to model dynamic networks [9, 10, 27]. A temporal graph is a sequence of graph *snapshots*, where each snapshot (G, t) encodes a graph G occurs at an associated timestamp t. Emerging applications call for efficient predictive models that can effectively suggest and maintain the nodes with high priority in dynamic networks. The need for such models is evident in causal analysis [10], anomaly and attack detection [27], and link prediction in social networks [9].

Learning to rank node in static graphs has been studied [3, 8, 35, 39]. A common practice in prior work is to make use of latent features from graph structures to learn ranking functions. Learning node ranks in dynamic networks are nevertheless more involved than its counterpart in static networks. In particular, the node ranks can be influenced by rich context from *heterogeneous node information*, *network structures*, and *temporal perspectives*. Consider a real-world example from alert management [41].

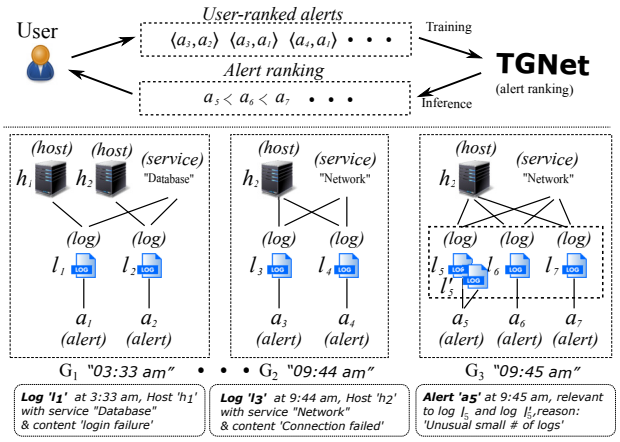


Figure 1: Alert ranking in a heterogeneous system alert network.

Example 1: Figure 1 illustrates a fraction of a real-world heterogeneous temporal information network from data center monitoring. The graph contains three snapshots G_1-G_3. Each snapshot contains four types of entities: `host`, `service` that a host can provide, `log` generated by services, and `alert` as suspicious system events. For example, an event "*unusual low amount of packages*" occurred at host h_2, indicated by its associated `log` l_5 and l_5', triggers an alert a_5 at 09 : 45 am that suggests a network failure. It is infeasible to manually inspect every alert to identify those with high priority, which are more likely to trigger diagnosis.

One may train a model to automatically rank and suggest alerts with high priority. This is nevertheless nontrivial for temporal graphs. (1) The node ranks can be determined by context features from both structural and temporal dimensions. For example, frequent linkages between "*login failures*" a_6 and a_7 to host h_2 in G_3 may suggest a host failure; alert a_5 ("*unusual small amount of logs*") should be ranked higher when it occurs *after* alerts " *Connection failed*" (a_3, a_4), suggesting a temporal consequence. (2) The model should cope with context features that bear constant changes over time. Moreover, labels provided by users over temporal data are typically sparse, covering a small fraction of nodes. Conventional models over static networks [3, 8, 35, 39] rely on structural features, and are unable to capture these context dynamics. □

CIKM '18, October 22–26, 2018, Torino, Italy

ACM ISBN 978-1-4503-6014-2/18/10…$15.00
https://doi.org/10.1145/3269206.3271698

Other applications, such as network attack detection [31] and author ranking [14], also require learning to rank nodes in temporal networks. While desirable, learning to rank nodes in temporal graphs is more challenging than its counterpart over static graphs. In particular, we observe the following challenges.

Context learning. Context features of a node are formed by certain activities related to this node. Such features offer us important sources of information to differentiate the roles that different nodes play in node ranking. In temporal graphs, context features are characterized from both structural and temporal dimensions. (1) Structural context captures the features in the neighborhood of nodes at each snapshot. For example, frequent occurrences of a certain type of alerts related to a host (*e.g.,* "*login failures*" a_6 and a_7 around host h_2 in G_3, Figure 1) may suggest a host failure. (2) Temporal context suggests the impact of historical or future events to current decisions. For example, we should suggest a higher investigation priority to the alert "*unusual small amount of logs*" (a_5) upon a scenario where it happens after the frequent occurrences of alerts "*Connection failed*" (a_3, a_4), compared with another scenario where we observe a_5 alone. As it is impractical to manually perform exhaustive feature engineering over heterogeneous graph, we need an automated approach to learn context features for node ranking.

Context dynamics. Context features of a node bear constant changes over time. For example, the context of host h_2 (Figure 1) at 03:33 am (normal condition) is different from its counterpart at 09:44 am (system failure). Such dynamics require effective and expressive models that can capture context evolution over time.

Label sparsity. Labels provided by users are typically sparse and cover a small fraction of nodes. For example, system admins and anti-malware software may only provide a few ranked examples over months of data with thousands of alerts. This calls for effective learning that can be generalized from sparsely labeled data.

Time cost. Efficient algorithms should be developed to support fast learning and ranking upon the arrival of new nodes. This requires effective optimization and provable performance guarantees for both learning and inference cost.

Learning node ranking has been studied in static graphs [3, 8, 35, 39]. These approaches focus on structural features, and are unable to capture context dynamics. The above challenges call for new framework that enables accurate node ranking in temporal graphs.

Contribution. This paper introduces TGNet, a novel deep learning framework that automates feature learning for node ranking in temporal graphs. TGNet tackles all the above challenges with contributions summarized as follows.

(1) TGNet learns context features from heterogeneous node attributes, network structures, and temporal correlation for more accurate node rank prediction. It utilizes two parameter-controlled network layers, namely, *structural propagation* and *temporal propagation* layers, to extract and exploit network structural and temporal context features, respectively (Section 3).

(2) We introduce *influence network*, a novel network layer utilized by TGNet to cope with context dynamics (Section 4). Unlike conventional methods that adopt fixed pairwise influence [4, 35, 38], influence network layer takes updated node context features as input, and dynamically estimates the amount of temporal and structural influence among the nodes.

(3) We show that TGNet can be efficiently learned over large temporal networks, by developing an end-to-end learning algorithm (Section 5). To address label sparsity, we impose a class of *graph smoothness constraints* as a weak form of supervision to address label sparsity. We also develop optimization strategies such as neighborhood sampling to reduce the learning cost.

(4) Using real-life datasets from system management, cybersecurity surveillance, academic networks, and social networks, we experimentally verify the performance of TGNet (Section 6). Our experimental study over real-world temporal networks verifies that TGNet outperforms conventional learning to rank models (such as random-walk based models) in accuracy with better generalization power, and incur no worse or smaller training cost. In particular, it achieves a high accuracy of 0.92 for ranking system alerts, and 0.87 for ranking suspicious packages in network security. TGNet also provides interpretable patterns that "explain" the ranking of nodes, as verified by our case study. These demonstrate practical applications of TGNet in large-scale dynamic networks.

2 RELATED WORK

Node ranking in static graphs. Node ranking has been studied for static graphs with unsupervised PageRank [36] or HITS [13]. These methods compute ranking scores via link-based analysis. Both graph structures and node attributes are exploited for unsupervised PageRank-based node ranking [20]. Supervised methods are also studied to exploit user preference. Tsoi et al. [33] studied quadratic programming to solve constrained PageRank with users' opinions. Agarwal et al. [3, 4] developed methods that learn parameters of Markovian walks in graphs that satisfy pairwise preference constraints between nodes. Backstrom et al. [8] investigated parameter learning in random walk with restart for link recommendation in online social networks. Wei et al. [35] studied parameter learning in heterogeneous graphs. In [5], a Laplacian smoothness constraint based method was developed. Rao et al. [29] proposed a label propagation based ranking algorithm. Hsu et al. [20] incorporated node attributes and proposed a Markov chain model to obtain the ranking. To cope with label sparsity, semi-supervised learning to rank [15, 40] enforces smoothness over ranking scores.

It is difficult to directly apply these methods for temporal graphs, due to context drifting and considerable tuning effort. Temporal context is not addressed in these works.

Node ranking in temporal graphs. Several recent methods have been proposed to update node ranks in dynamic networks [19, 28, 30, 39]. For example, Zhang et al. [39] proposed a method with bounded propagation cascade for forward push and reverse push. O'Madadhain et al. [28] proposed a framework to rank nodes derived from social events occurring over time. Yu et al. [37] proposed an information flow propagation model for ranking on heterogeneous networks. To capture how changes in external interest influence the importance of a node, Rossi et al. [30] proposed an evolving teleportation adaptation of the PageRank method. To improve the accuracy, evolution patterns [10, 38] are used to model

dynamics in node importance. To improve the efficiency, an adaptive ranking method is proposed [2] by taking into account only recent historical data.

Unlike existing methods that mainly focus on unsupervised settings, we develop a supervised method that automatically incorporates domain knowledge by labeled data, and integrates node attributes, network structures, and temporal correlation to improve ranking accuracy. Our model utilizes parameter-controlled graph propagation to jointly model structural and temporal context. We also develop end-to-end, efficient learning algorithm to guide parameter estimation, instead of using preset parameters [38].

Deep Learning on Static Graphs. Deep learning has been applied to prediction problems in static graphs. Notable examples include variants of graph neural networks [25] for prediction tasks in graph sequence data, and graph convolution neural networks [12]. These models are hard to be adapted for ranking in temporal networks, as they are designed for learning over static data. For example, while the model in [25] only deals with a fixed set of nodes in graph sequences, we propose influence network to adaptively update influence among dynamic sets of nodes that are allowed to join or leave at any time. Moreover, none of these models makes use of temporal information, which is a necessity for predicting node ranks in temporal networks. To the best of our knowledge, our work is the first attempt that exploits deep learning for node ranking in temporal graphs.

3 MODEL OVERVIEW

3.1 Learning to rank node

We start with the temporal graph model used in TGNet. We reserve bold lower-case letters for column vectors (*e.g.,* $\mathbf{a}$), normal lower-case letters for scalars (*e.g., a*), bold upper-case letters for matrices (*e.g.,* $\mathbf{A}$), and normal upper-case letters for graphs, sets, and functions (*e.g., A*).

Temporal Graph. A temporal graph is a sequence of snapshots $\mathcal{G} = \big(\langle (G_1, t_1), \ldots, (G_s, t_s)\rangle, V, F\big)$, where

(1) A *snapshot* $G_i = (V_i, E_i)$ is a graph with nonempty set of nodes V_i and edges E_i at timestamp t_i ($i \in [1, s]$);
(2) $V = \bigcup_{i\in[1,s]} V_i$ is the node set of temporal graph $\mathcal{G}$; and
(3) For each node $v \in V$, $F : V \to \mathbb{R}^{d_{in}}$ is a function that assigns a d_{in}-dimensional input feature vector to v. Here $F(\cdot)$ can specify domain-specific node features.

We consider heterogeneous temporal graphs with multiple types of nodes and edges. Fig. 1 depicts a heterogeneous temporal graph $\mathcal{G}$ with three snapshots. For node a_5, its feature vector $F(a_5)$ contains node type (*e.g.,* "alert"), and domain-specific features such as alert description (*e.g.,* "*Unusual small # of logs*") extracted from log analysis tools [7].

Ranking function. We define training data $\mathbb{G}$ as a pair $(\mathcal{G}, \mathcal{D})$, where (1) $\mathcal{G}$ is a temporal graph with node set V, and (2) the labeled data $\mathcal{D}$ is a set of ordered node pairs $\{\langle u, v\rangle \mid u \in V, v \in V\}$. Each pair $\langle u, v\rangle \in \mathcal{D}$ suggests that v is preferred over (ranked higher than) u. In practice, such data can be provided by domain experts.

Given a temporal graph $\mathcal{G}$ and a set of target nodes $V' \subseteq V$, a ranking function $g : V' \to \mathbb{R}$ assigns ranking scores to these nodes, such that for each node $v \in V'$, $g(v)$ is a ranking score that quantifies the preference to v: the higher, the more preferred v is. In practice, the targeted node set V' specifies the nodes of users' interests to be considered for ranking (*e.g.,* "alert" nodes in Fig. 1).

Symbol	Definition
$\mathcal{G}$	$\big(\langle (G_1, t_1), \ldots, (G_s, t_s)\rangle, V, F\big)$; temporal graph
G_i	a snapshot in $\mathcal{G}$ at timstamp t_i
V'	target nodes in V to be ranked
$\mathcal{D}$	labeled data
$\mathbf{x}_v$	input feature vector of node v
$\mathbf{h}_v$	hidden state vector of node v with length d_h
y_v	ranking score of node v
$\mathbf{W}_{in}$, $\mathbf{b}_{in}$	model parameters in initialization layer
$p_{u,v}$	structural influence factor from u to v
$\mathbf{w}_{si}$, b_{si}	model parameters in structural influence network
λ_{v,t_i,t_j}	temporal influence factor from t_i to t_j on node v
$\mathbf{w}_{ti}$, b_{ti}	model parameters in temporal influence network
$\mathbf{w}_{out}$, b_{out}	model parameters in output layer

Table 1: Notations

Problem statement. We study a general *learning to rank node* problem for heterogeneous temporal graphs. Given training data $\mathbb{G}$ consists of a temporal graph $\mathcal{G}$ and labeled data $\mathcal{D}$, a ranking function space $\mathbb{M}$, and an error function $J(\cdot)$ that measures the ranking error, the problem is to find an optimal function $g^* \in \mathbb{M}$ such that the ranking error $J(g^*, \mathbb{G})$ is minimized.

This problem is a departure from our familiar learning to rank for static data. It requires a feasible model that incorporates structural and temporal context, is adaptive to context dynamics, and is able to be generalized from sparse labels.

3.2 Overview of TGNet Model

We next introduce **TGNet**, a neural network based model for learning to rank nodes in temporal networks. In contrast to conventional models, it incorporates both structural and temporal context to predict node ranks in temporal graphs.

TGNet model. TGNet consists of four components (as illustrated in Figure 2): (1) Initialization layer, which projects input feature vector into hidden state space; (2) Structural propagation layer, which exchanges neighborhood information among the nodes in a snapshot for the construction of structural context features; (3) Temporal propagation layer, which propagates temporal influence between snapshots for temporal context features; and (4) Output layer, which transforms hidden states to ranking scores.

We introduce the details of each layer. Given a temporal graph $\mathcal{G}$ with $\langle (G_1, t_1), \ldots, (G_s, t_s)\rangle$ and target nodes V', TGNet sequentially processes snapshots from G_1 to G_s and infers node ranking scores for nodes in V'. Assume that (G_i, t_i) is the snapshot TGNet is processing at timestamp t_i.

Initialization layer. When $v \in V_i$, and G_i is the snapshot in which v first occurs, the initialization layer of TGNet performs feature selection on the input node features of v as:

$$\mathbf{h}_v^{(i,0)} = \tanh(\mathbf{W}_{in}^T \cdot \mathbf{x}_v + \mathbf{b}_{in}) \tag{1}$$

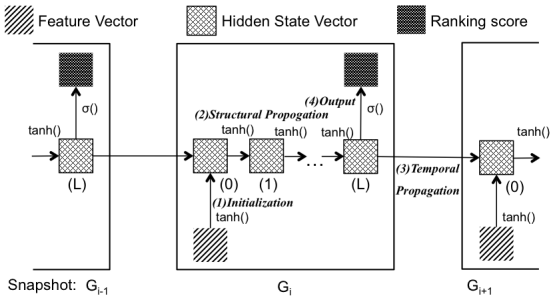


Figure 2: Basic components of TGNet

where $\mathbf{x}_v = F(v)$ is the input feature of v, $\tanh(\cdot)$ is hyperbolic tangent function that projects input feature vector into a d_h-dimensional feature space, $\mathbf{h}_v^{(i,0)}$ is the initial hidden state of node v, and $\mathbf{W}_{in}, \mathbf{b}_{in}$ are model parameters.

Intuitively, hidden states of a node v encode the context information "around" v in a temporal network. Hidden states are formed and updated by multiple layers of structural and temporal propagation as shown below.

Structural propagation layer. Structural neighborhood serves as a source of context features and provides valuable insight to decide node ranks [17, 34]. The goal of this layer is to collect structural neighborhood information for each individual node in G_i. TGNet materializes this process by iteratively performing local information propagation (for L times). For any $v \in V_i$ and $0 \le k < L$,

$$\mathbf{h}_v^{(i,k+1)} = \tanh\Big(\sum_{u \in N_v} p_{u,v}^{(i,k)}\, \mathbf{h}_u^{(i,k)}\Big), \tag{2}$$

where $\tanh(\cdot)$ is hyperbolic tangent function, $N_v = \{u \mid (u,v) \in E_i\} \cup \{v\}$ is a set of neighborhood nodes around v (including v), and $p_{u,v}^{(i,k)}$ depicts the amount of the influence from u to v in the k-th structural propagation at G_i.

By performing the propagation following Equation (2) up to L iterations, each node gradually accumulates context information in its neighborhood. In this process, $p_{u,v}^{(i,k)}$ filters out unnecessary neighborhood information, while keeping the essential context for node ranking prediction.

Temporal propagation layer. Context features also encode the influence from past events. Intuitively, the activities occurred earlier provide useful information for node rank estimation. When $v \in V_i$ and $v \in V_{i+1}$, we have

$$\mathbf{h}_v^{(i+1,0)} = \lambda_{v,t_i,t_{i+1}} \mathbf{h}_v^{(i,L)} \tag{3}$$

where $\lambda_{v,t_i,t_{i+1}}$ suggests how much the context information on node v at time t_i should be carried to t_{i+1}, $\mathbf{h}_v^{(i,L)}$ is the hidden state of v in G_i after structural propagation, and $\mathbf{h}_v^{(i+1,0)}$ is the initial hidden state of v in G_{i+1}.

In this layer, $\lambda_{v,t_i,t_{i+1}}$ serve as filters that only keep useful historical context information for node ranking.

Output layer. This layer generates ranking score estimation based on context features (*i.e.,* hidden states) on each individual nodes. For $v \in V'$, when it is the case that $i < s$, $v \in V_i$, $v \notin V_{i+1}$, or the case that $v \in V_s$, we have

$$\hat{y}_v = \sigma(\mathbf{w}_{out}^T \cdot \mathbf{h}_v^{(i,L)} + b_{out}) \tag{4}$$

where $\sigma(\cdot)$ is sigmoid function, $\mathbf{h}_v^{(i,L)}$ is the hidden state of the node v after structural propagation, and $\mathbf{w}_{out}, b_{out}$ are model parameters that control the projection from the hidden state space of the nodes to their ranking score space.

By default, TGNet generates the ranking score of a node v in the last snapshot it appears, given the intuition that more reliable decision can be made to rank v if all the relevant context of v is observed. For standalone nodes with no edges in any timestamps, their raking values are determined by their input feature vectors and temporal propagation, *i.e.,* they do not participate in structural propagation. Our method can be readily extended to allow TGNet to output ranking scores at user-specified timestamps.

4 INFLUENCE MODELING

In this section, we discuss influence modeling. With the belief that the context of a node is shaped by what has happened in the "history" of its neighborhood, the key of influence modeling is to quantify neighborhood impact.

"Node-centric" vs. "Edge-centric". Conventional methods [3, 4, 8, 20, 25, 35] adopts "edge-centric" influence model, where the parameters are associated to edges. Nevertheless, the assumption of fixed edge influence is hard to hold for context changes. Moreover, it is daunting for users to choose edge types with expected generalization power for unknown testing data.

In contrast, TGNet adopts an *influence network* layer, denoted as **InfNet**, which uses a "node-centric" approach to model influence. The layer **InfNet** is devised based on two intuitions: (1) *The influence between two nodes is conditioned by their contexts*; and (2) *The node context is determined by its hidden state*. For example, a `host` v that generates observed critical `alerts` in a short period (encoded by its hidden states) typically indicates that a future alert, when connected to v, should be ranked higher (more suspicious).

Consider a node v and its hidden state vectors $\mathbf{h}_v$. **InfNet** infers the influence that affects v as:

$$\text{inf}_v = \textbf{InfNet}\big(\mathbf{H}_v, \mathbf{h}_v, \mathbf{c}\big) \tag{5}$$

where (1) $\mathbf{H}_v$ includes neighborhood information of v (to be discussed) and (2) $\mathbf{c}$ is a vector encoding side information of interest (*e.g.,* elapsed time) when applicable.

Influence network explicitly addresses the three challenges as remarked earlier. (1) It captures influence that may be drifted along with the evolution of node contexts, as the influence from new nodes and edges is naturally encoded. (2) As node contexts are automatically constructed in TGNet, influence modeling is fully automated in accordance, minimizing the effort of user interference. (3) The number of parameters in influence network is *independent* of the size of the underlying graphs. This indicates the cost of model learning and inference tend to be insensitive to the growth of $\mathcal{G}$, suggesting good scalability of TGNet over large graphs.

The framework TGNet gears two separate influence networks in structural propagation and temporal propagation layer, respectively. We next introduce the details.

Structural influence. TGNet undertakes an intuition of "my neighbors decide who I am". In particular, structural influence is captured by $p_{u,v}^{(i,k)}$ in Equation (2). Suppose G_i is the snapshot under

processing. For $v \in V_i$ and $u \in N_v$, given $\mathbf{H}_v = [\mathbf{h}_u^{(i,k)}]_{u \in N_v}$ that includes hidden vectors of v's neighbors, we compute influence in v's neighborhood $\mathbf{p}_v^{(i,k)} = [p_{u,v}^{(i,k)}]_{u \in N_v}^T$ as follows.

$$\begin{aligned} \mathbf{p}_v^{(i,k)} &= \mathbf{InfNet}(\mathbf{H}_v, \mathbf{h}_v^{(i,k)}, \emptyset) \\ &= S(\mathbf{z}_v^{(i,k)}) \end{aligned} \tag{6}$$

where (1) $S(\cdot)$ is the softmax function, and (2) $\mathbf{z}_v^{(i,k)} = [z_{u,v}^{(i,k)}]_{u \in N_v}^T$, which is further derived by

$$z_{u,v}^{(i,k)} = \mathbf{w}_{si}^T \cdot \begin{bmatrix} \mathbf{h}_u^{(i,k)} \\ \mathbf{h}_v^{(i,k)} \end{bmatrix} + b_{si}, \tag{7}$$

where $\mathbf{w}_{si}, b_{si}$ are model parameters.

Model complexity. The size of $\mathbf{w}_{si}$ is linear to the number of dimensions in hidden state vectors. As $\mathbf{w}_{si}$ and b_{si} are shared among all edges, TGNet is able to provide structural influence estimation with a number of parameters, which is independent of the number of edges.

Temporal influence. Equation (3) represents temporal influence as $\lambda_{v,t_i,t_{i+1}}$, with an intuition that "what I am now" affects "what I will be". Unlike structural influence that is conditioned on the contexts of two adjacent nodes, temporal influence is determined by the context of a node itself and elapsed time. Suppose G_i is the snapshot under processing. When $v \in V_i$, and $v \in V_{i+1}$, we have

$$\begin{aligned} \lambda_{v,t_i,t_{i+1}} &= \mathbf{InfNet}(\emptyset, \mathbf{h}_v^{(i,L)}, \Delta t) \\ &= \sigma\big(\mathbf{w}_{ti}^T \cdot \begin{bmatrix} \mathbf{h}_v^{(i,L)} \\ \Delta t \end{bmatrix} + b_{ti}\big) \end{aligned} \tag{8}$$

where $\sigma(\cdot)$ is sigmoid function, $\Delta t = t_{i+1} - t_i$, and $\mathbf{w}_{ti}, b_{ti}$ are model parameters.

Model complexity. The size of $\mathbf{w}_{ti}$ is linear to the number of dimensions in hidden state vectors, and $\mathbf{w}_{ti}$, b_{ti} are shared among all graph snapshots of $\mathcal{G}$.

Inference Cost. Let the size of $\mathcal{G}$ (denoted as $|\mathcal{G}|$) be $|V| + |E|$, where E is the set of all the edges occurred in $\mathcal{G}$. Consider a newly arrived snapshot G_i at time t_i. The result below verifies the efficiency of TGNet in the inference cost.

Lemma 1: *Upon the arrival of each graph snapshot G_i,* TGNet *takes $O(|\mathcal{G}|)$ time to rank the targeted nodes in G_i.* □

We verify the inference cost as below. Let G_i contain $|V_i|$ nodes and $|E_i|$ edges. The inference cost for each G_i consists of: (1) the structural propagation cost in $O(L * d_h * |E_i|)$ time, for L rounds of propagation, and each visits edges in E_i and updates structural influence in $O(1)$ time; (2) the cost of temporal propagation is $O(|V_i|)$, as temporal influence is computed in $O(1)$ time; and (3) $O(|V_i|)$ time for output layer. In practice, L, d_h and d_{in} are relatively much smaller. (for example, $L \leq 5$ and $d_h \leq 10$ when accuracy converges in our experiments). TGNet thus takes linear time ($O(|\mathcal{G}|)$). This is desired for tasks that require online prediction.

Neighbor Sampling. To further reduce the inference cost for large graphs, TGNet introduces a *neighbor sampling* strategy to support fast structural inference. In each iteration of inference, we randomly sample a subset $D(v)$ of the neighbors $N(v)$ of each node v in G_i ($D(v) \subset N(v)$). We then use Monte-Carlo approximation as an unbiased estimator to estimate the hidden state value of v by accessing only the nodes in $D(v)$, instead of $N(v)$ as in Equation (2). Specifically, the propagation follows a revised Equation (2) as:

$$\tilde{\mathbf{h}}_v^{(i,k+1)} \approx \tanh\big(\frac{|N(v)|}{|D(v)|} \sum_{u \in D(v)} p_{u,v}^{(i,k)}\, \mathbf{h}_u^{(i,k)}\big) \tag{9}$$

The sampling reduces the cost of structural propagation by accessing only sampled nodes. Our model can be readily extended to incorporate other sampling methods that sparsify large networks [6]. We integrate neighbor sampling at runtime in each iteration to avoid additional overhead to inference cost.

5 PARAMETER LEARNING

We next investigate parameter estimation in TGNet. Given labeled data, we introduce an end-to-end training algorithm that jointly learns the model parameters.

Objective function. Let Θ be parameters in TGNet and $\mathbb{G} = (\mathcal{G}, \mathcal{D})$ be training data, where $\mathcal{G}$ is a temporal graph and $\mathcal{D}$ is labeled data which contain a set of labeled node pairs. Given Θ and $\mathbb{G}$, we define the learning objective function as

$$\begin{aligned} J(\Theta, \mathbb{G}) = & \sum_{\langle u,v \rangle \in \mathcal{D}} E(\Theta, u, v) && (\textit{Model error}) \\ & + \beta_1 R_1(\Theta) && (\textit{Model complexity}) \\ & + \beta_2 R_2(\Theta, \mathcal{G}) && (\textit{Graph smoothness}) \end{aligned}$$

where $E(\cdot)$ quantifies error made by Θ, $R_1(\cdot)$ and $R_2(\cdot)$ are two regularization terms, which penalizes model complexity, and encourages graph smoothness (discussed below), respectively; and β_1, β_2 are meta parameters that tune the impact from $R_1(\cdot)$ and $R_2(\cdot)$ to the objective value, respectively.

Loss function. Following the convention in node ranking [3, 4], given $\langle u, v \rangle \in \mathcal{D}$ (*i.e.,* $y_u < y_v$), the error made by Θ is calculated by the loss function below:

$$E(\Theta, u, v) = 1 - (\hat{y}_v - \hat{y}_u) \tag{10}$$

where $\hat{y}_u$ and $\hat{y}_v$ are estimated ranking scores via Θ. As ranking scores $\hat{y}_u, \hat{y}_v \in [0, 1]$, $E(\Theta, u, v) \geq 0$.

We employ $\mathcal{L}_2$ regularization to penalize model complexity. The term $R_1(\Theta)$ is defined as:

$$R_1(\Theta) = \sum_{\theta \in \Theta} \|\theta\|_2^2. \tag{11}$$

Graph smoothness. Labeled data $\mathcal{D}$ could be sparse in practice. To mitigate the risk of overfitting caused by label sparsity, TGNet applies graph smoothness constraints to penalize high dissimilarity between a node and its neighbors. Indeed, real-life applications suggest "birds of a feather" effect [40]. For example, a critical alert of users' interest suggests that the logs in its neighborhood could also be of interest to the users.

Based on this intuition, we formulate smoothness constraints in single snapshots as:

$$R_2(\Theta, \mathcal{G}) = \sum_{i=1}^{s} \sum_{(u,v) \in E_i} \|\mathbf{h}_u^{(i,L)} - \mathbf{h}_v^{(i,L)}\|_2^2 \tag{12}$$

where $\mathbf{h}_u^{(i,L)}$ is the hidden state of the node u in G_i after structural propagation. That is, we encourage low distance in hidden state space between a node and its neighbors in $\mathcal{G}$.

Algorithm: TGNet **learning**

Input: training data $\mathbb{G}$, learning rate α, threshold ϵ;
Output: Optimal Θ^* with respect to $J(\Theta, \mathbb{G})$

1. Initialize $\Theta^{(0)}$; $j = 0$;
2. **repeat**
3. $j = j + 1$
4. **for** $\theta \in \Theta$
/* Gradients calculation and parameter update*/
/* Equations (13) – (18) */
5. $\theta^{(j)} = \theta^{(j-1)} - \alpha \cdot \frac{\partial J(\Theta^{(j-1)}, \mathbb{G})}{\partial \theta}^T$;
6. **until** $|J(\Theta^{(j)}, \mathbb{G}) - J(\Theta^{(j-1)}, \mathbb{G})| < \epsilon$
7. $\Theta^* = \Theta^{(j)}$
8. **return** Θ^*

Figure 3: Algorithm sketch of TGNet **learning**

Algorithm. We now introduce the learning algorithm of TGNet. The TGNet learning algorithm, illustrated in Figure 3, follows back propagation through time framework. It has the following steps.

(1) Initialize Θ (line 1);
(2) Iteratively compute error resulting from current Θ and update Θ by their gradients (lines 2-5);
(3) The above step iterates until the objective function $J(\Theta, \mathbb{G})$ converges (line 6).

Given objective function J, for $\theta \in \Theta$:

$$\frac{\partial J}{\partial \theta} = \frac{1}{|\mathbb{G}|} \sum_{(\mathcal{G},\mathcal{D})} \sum_{\langle u,v\rangle \in \mathcal{D}} (\frac{\partial \hat{y}_u}{\partial \theta} - \frac{\partial \hat{y}_v}{\partial \theta}) + 2\beta_1 \theta^T \tag{13}$$
$$+ \frac{2\beta_2}{|\mathbb{G}|} \sum_{(\mathcal{G},\mathcal{D})} \sum_{i=1}^{s} \sum_{(u,v)\in E_i} (\mathbf{h}_u^{(i,L)} - \mathbf{h}_v^{(i,L)})^T (\frac{\partial \mathbf{h}_u^{(i,L)}}{\partial \theta} - \frac{\partial \mathbf{h}_v^{(i,L)}}{\partial \theta})$$

Below we highlight the key steps in the learning algorithms.

- *Output layer*. For $\theta \in \Theta$,

$$\frac{\partial \hat{y}_v}{\partial \theta} = \sigma'(\mathbf{w}_{out}^T \cdot \mathbf{h}_v^{(i,L)} + b_{out}) \cdot Z_{out}(\theta) \tag{14}$$

where

$$Z_{out}(\theta) = \begin{cases} \mathbf{h}_v^{(i,L)^T}, & \theta = \mathbf{w}_{out}; \\ 1, & \theta = b_{out}; \\ \mathbf{w}_{out}^T \cdot \frac{\partial \mathbf{h}_v^{(i,L)}}{\partial \theta}, & \text{otherwise.} \end{cases}$$

- *Structural propagation layer*. In this layer, the learning algorithm performs the computation recursively,

$$\frac{\partial \mathbf{h}_v^{(i,k)}}{\partial \theta} = tanh'(\sum_{u\in N_v} p_{u,v}^{(i,k-1)} \mathbf{h}_u^{(i,k-1)}) \tag{15}$$
$$\cdot \sum_{u\in N_v} \left(S'(\mathbf{w}_{si}^T \cdot \begin{bmatrix} \mathbf{h}_u^{(i,k-1)} \\ \mathbf{h}_v^{(i,k-1)} \end{bmatrix} + b_{si}) \right.$$
$$\left. \cdot \mathbf{h}_u^{(i,k-1)} \times Z_{sp}(\theta) + p_{u,v}^{(i,k-1)} \cdot \frac{\partial \mathbf{h}_u^{(i,k-1)}}{\partial \theta} \right)$$

where

$$Z_{sp}(\theta) = \begin{cases} \begin{bmatrix} \mathbf{h}_u^{(i,k-1)} \\ \mathbf{h}_v^{(i,k-1)} \end{bmatrix}^T + \mathbf{w}_{si}^T \cdot \begin{bmatrix} \frac{\partial \mathbf{h}_u^{(i,k-1)}}{\partial \theta} \\ \frac{\partial \mathbf{h}_v^{(i,k-1)}}{\partial \theta} \end{bmatrix}, & \theta = \mathbf{w}_{si}; \\ 1 + \mathbf{w}_{si}^T \cdot \begin{bmatrix} \frac{\partial \mathbf{h}_u^{(i,k-1)}}{\partial \theta} \\ \frac{\partial \mathbf{h}_v^{(i,k-1)}}{\partial \theta} \end{bmatrix}, & \theta = b_{si} \\ \mathbf{w}_{si}^T \cdot \begin{bmatrix} \frac{\partial \mathbf{h}_u^{(i,k-1)}}{\partial \theta} \\ \frac{\partial \mathbf{h}_v^{(i,k-1)}}{\partial \theta} \end{bmatrix}, & \text{otherwise.} \end{cases}$$

- *Temporal propagation layer*. If $v \in V_i$ and $v \in V_{i-1}$, for $\theta \in \{\mathbf{W}_{in}, \mathbf{b}_{in}, \mathbf{w}_{si}, b_{si}, \mathbf{w}_{ti}, b_{ti}\}$, the learning algorithm propagates error in this layer as follows.

$$\frac{\partial \mathbf{h}_v^{(i,0)}}{\partial \theta} = \lambda_{v,t_{i-1},t_i} \cdot \frac{\partial \mathbf{h}_v^{(i-1,L)}}{\partial \theta} \tag{16}$$
$$+ \sigma'(\mathbf{w}_{ti}^T \cdot \begin{bmatrix} \mathbf{h}_v^{(i-1,L)} \\ \Delta t \end{bmatrix} + b_{ti}) \cdot \mathbf{h}_v^{(i-1,L)} \times Z_{tp}(\theta)$$

where

$$Z_{tp}(\theta) = \begin{cases} \begin{bmatrix} \mathbf{h}_v^{(i-1,L)} \\ \Delta t \end{bmatrix}^T + \mathbf{w}_{ti}{}^T \cdot \begin{bmatrix} \frac{\partial \mathbf{h}_v^{(i-1,L)}}{\partial \theta} \\ 0 \end{bmatrix}, & \theta = \mathbf{w}_{ti}; \\ 1 + \mathbf{w}_{ti}{}^T \cdot \begin{bmatrix} \frac{\partial \mathbf{h}_v^{(i-1,L)}}{\partial \theta} \\ 0 \end{bmatrix}, & \theta = b_{ti}; \\ \mathbf{w}_{ti}{}^T \cdot \begin{bmatrix} \frac{\partial \mathbf{h}_v^{(i-1,L)}}{\partial \theta} \\ 0 \end{bmatrix}, & \text{otherwise.} \end{cases}$$

- *Initialization layer*. If $v \in V_i$ and $v \notin V_{i-1}$, error propagation moves into initialization layer as follows. Let $\mathbf{w}_d$ be the d-th column of $\mathbf{W}_{in}$, b_d be the d-th entry in $\mathbf{b}_{in}$, and r_{in} be the number of rows in $\mathbf{W}_{in}$.

$$\frac{\partial \mathbf{h}_v^{(i,0)}}{\partial \mathbf{w}_d} = (tanh'(\mathbf{w}_d^T \cdot \mathbf{x}_v + b_d) \cdot \mathbf{e}_d) \times \mathbf{x}^T, \tag{17}$$

where $\mathbf{e}_d$ is a $r_{in} \times 1$ standard basis vector for dimension d. Moreover,

$$\frac{\partial \mathbf{h}_v^{(i,0)}}{\partial \mathbf{b}_{in}} = tanh'(\mathbf{W}_{in}^T \cdot \mathbf{x}_v + b_{in}) \times \mathbf{1}^T, \tag{18}$$

where $\mathbf{1}$ is a $r_{in} \times 1$ vector with 1 in each individual entry.

Learning cost. We show that the learning of TGNet is feasible over large graphs. Consider the training data $\mathbb{G} = (\mathcal{G}, \mathcal{D})$, where $|\mathcal{D}|$ is the number of node pairs in $\mathcal{D}$. we show the following result.

Lemma 2: *The learning cost of* TGNet *is in* $O(|\mathcal{G}||\mathcal{D}|)$ *time.* □

To see this, we observe that (1) it takes in total $O(|\mathcal{D}|)$ time to compute the derivative of $\mathbf{w}_{out}$, and (2) it takes in total $O(c * |\mathcal{G}| * |\mathcal{D}|)$ time to compute the derivate for $\mathbf{w}_{si}$, $\mathbf{w}_{ti}$ and $\mathbf{w}_{in}$ as the training algorithm traverses $\mathcal{G}$ at most $|\mathcal{D}|$ times by backpropagation. Here c is a constant determined by model configuration, *i.e.,* parameters L and d_{in}. Indeed, following practical assumptions in cost on single core learning [11] (and verified in Section 6), the model parameters d_{in} and L are typically small in practice. Moreover, $|\mathcal{D}|$ is much smaller than $|\mathcal{G}|$ for sparse labelled data.

Dataset	# snapshots	total # nodes ($\|V\|$)	total # edges ($\|E\|$)	$\|\mathcal{D}\|$
SLD	19.4K	61.4K	258.1K	20K
IDS	66.7K	5.7M	11.5M	100K
MAG	12K	2.5M	16.2M	100K
APC	1.5K	2.1M	9.5M	100K

Table 2: Real-life temporal graphs

"Early terminating". A bottleneck of TGNet learning is due to the vanishing gradient in backpropagation. The gradients of loss tend to decrease as we keep moving backward, and the backpropagation steps with small gradients still take time but contribute little to the learning outcome. To reduce the cost incurred by these steps, we apply a second threshold ϵ' to the gradients of nodes to terminate their backpropagation early. That is, whenever the process sees a gradient of node v below ϵ' at a snapshot, TGNet stops backpropagation of v. The learning terminates early if all gradients are smaller than ϵ' at a snapshot.

This optimization is quite effective: it can reduce 47% of the learning cost on average over real-life data, with up to 3% loss of model accuracy, as verified in Section 6.

6 EXPERIMENTS

Using both real-world and synthetic datasets, we evaluate (1) the accuracy of TGNet, (2) the impact of model parameters which cannot be learned from end-to-end training and size of training data, and (3) efficiency of TGNet learning.

Dataset. We used four real-world datasets.

(1) SLD is a private system log dataset from NEC labs, including 30 days' system logs generated by a cluster of 16 hosts. Using log analysis tool NGLA [7], we obtained a graph with in total 19.4K snapshots, 61.4K nodes, 258.1K edges, and 20K user-ranked alert pairs by domain-experts. For SLD, we focus on ranking *alert* nodes.

(2) IDS (*ISCX Intrusion detection data*) [31] is a network intrusion dataset that consists of in total 66.7K snapshots, 5.7M nodes, 11.5M edges, and 100K user-ranked packet pairs. In this dataset, we are interested in *packet* node ranking: the higher one packet node is ranked, the more likely it indicates an attack.

(3) MAG (*Microsoft academic graph*) [32] is an academic network with 12k snapshots, 2.5M nodes (*e.g.,* authors, papers, institutions) and 16.2M edges. We sampled 100K author ranking pairs based on their H-index value, and focus on learning to rank *author* nodes.

(4) APC (*Amazon product co-purchasing network*) [24] is a network describing the the product information and related reviews. It contains 1.5k snapshots, 2.1 million nodes (*e.g.,* products, customers), and 9.5 million edges. We focus on learning to rank the salesrank value for each product and sampled 100k product pairs.

To create temporal graphs as snapshots, we select time scales that simulates update frequency of each datasets or the needs of domain experts. We set interval as 1 second for SLD and IDS, and one day for MAG and APC. The datasets are summarized in Table 2.

We also generated synthetic datasets to evaluate the learning efficiency of our models. Using sampling with replacement, we generated larger sets of temporal graphs from SLD, IDS, MAG and APC, respectively. Keeping the average size of snapshots, the total number of snapshots for a synthetic graph is increased up to 5 times of its real world counterpart.

	Influence model	Structural context	Temporal context	Node features	Smoothness
TGNet	node-centric	yes	yes	yes	yes
TGNet_BA	edge-centric	yes	yes	yes	no
TGNet_IN	node-centric	yes	yes	yes	no
SPR	edge-centric	yes	no	no	no
DPR	edge-centric	yes	no	yes	no
SVMRank	-	no	no	yes	-
LRT	-	no	no	yes	-
BGCM	-	no	no	yes	-

Table 3: Comparison of TGNet and baseline methods

Implementation. We implemented TGNet using Tensorflow [1]. We initialized parameters in TGNet following a general parameter initialization method from [23]: randomly drawn from a distribution with mean zero and standard deviation. By default, we set d_h (number of dimensions in hidden states) as 5, and the default value for L (number of iterations in structural propagation) is set as 3. We apply Adam method [22] with its default learning rate. For the best performance, we set $\beta_1 = 1, \beta_2 = 1, \epsilon = 0.0001$.

Ranking methods. We compared TGNet with seven baselines.

(1) *TGNet variants.* We develop two variants of TGNet, TGNet_IN and TGNet_BA, by removing graph smoothness constraints, and by removing both influence network and graph smoothness, respectively. Instead of influence network, TGNet_BA follows edge-centric methods [4] to model a uniform temporal influence with edge parameters, by a decay function $\lambda_{v,t_i,t_{i+1}} = e^{-\omega\Delta t}$ where ω is the parameter used by all temporal propagation.

(2) *PageRank variants.* SPR [4] uses supervised PageRank to rank nodes in static graphs, with edge-centric influence model (adopted by the variant TGNet_BA of TGNet). For a fair comparison, we merge all snapshots into a static graph as its input. We followed the setting in [4] and used Newton method to train edge weights. DPR [30] is an unsupervised PageRank algorithm that leverages dynamic teleportation to capture how changes in external interest influence the importance of a node.

(3) *SVM variants.* SVMRank [21] is a ranking algorithm that uses SVM to learn node ranks, where the ranking is determined only by learning from node features. We selected L1-norm and regularization parameters for SLD, IDS and MAG as 0.1, 2.0, and 0.5, respectively, such that SVMRank achieves the best performance.

Besides general ranking approaches, we also implemented two domain-specific methods in system alert management, to evaluate the effectiveness of TGNet in real-world applications.

(4) *Domain specific methods.* LRT is a ranking algorithm [18] that applies statistical testing to discover interesting alerts and packets. BGCM [16] is a statistical method that discovers critical alerts or suspicious packets by finding a non-trivial weighted combination of all atomic detectors. Neither of these approaches considers context features or dynamics.

The above algorithms are summarized in Table 3.

Features. For SLD, each node feature vector has 125 dimensions, including node type, alert- and log-related information. For IDS,

	BGCM	LRT	SVMRank	SPR	DPR	TGNet_BA	TGNet_IN	**TGNet**
SLD	0.35	0.72	0.74	0.78	0.56	0.88	0.90	**0.92**
IDS	0.32	0.59	0.70	0.72	0.67	0.77	0.85	**0.87**
MAG	-	-	0.56	0.69	0.61	0.79	0.86	**0.91**
APC	-	-	0.52	0.55	0.45	0.73	0.82	**0.88**

Table 4: Accuracy (NDCG@k) comparison

each feature vector has 9 dimensions, including attributes such as protocol type and packet length. Each feature vector in MAG has 45 dimensions, such as research topics and citations. For APC, each feature vector has 15 dimensions, including the category and review information. For synthetic datasets, the features are the same as their corresponding real-world counterparts.

Metric. We evaluated the accuracy of TGNet and its baselines by NDCG@k [26] (the top-k version of Normalized Discounted Cumulative Gain), which is the DCG value of top-k output ranking list normalized by the DCG value of top-k ground truth (optimal) ranking list. The value of k can be determined in needs of domain experts. For SLD, we set k based on the suggestion from our industry partner which is the number of interested alerts. For IDS, MAG, and APC, we consider the worst case and set k as the total number of packets, authors, and products, respectively, to test our model. More specifically, k is set as $29k$, 0.9 million, $55k$ and $549k$ for SLD, IDS, MAG and APC respectively.

To evaluate the accuracy, we performed 10-fold cross validation. Average results of 20 times of executions are reported. All the experiments were conducted on a machine powered by an Xeon processor with 2.3GHz, 128GB memory, and a NVIDIA K80 GPU.

We next report the details of our experimental results.

Exp-1: Model Accuracy. Table 4 reports the accuracy of TGNet and the baseline models. As LRT and BGCM are domain-specific for alert ranking, their results on MAG and APC are omitted. In all cases, it takes up to 50 iterations to train TGNet for SLD, and up to 100 iterations for IDS, MAG and APC. We find the following.

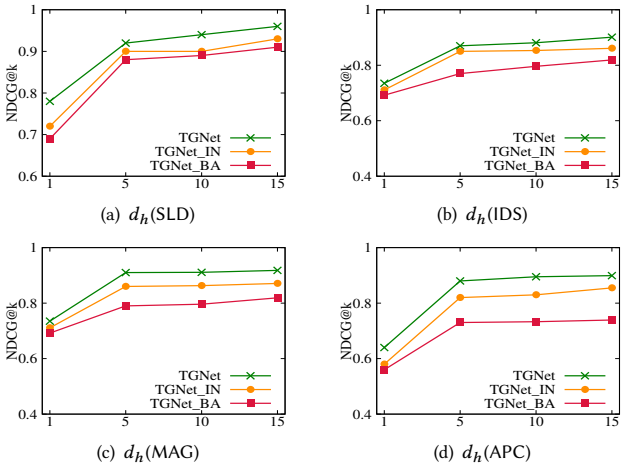
(1) TGNet reaches the best performance compared with its variants. Comparing TGNet_BA with TGNet_IN, influence networks bring up to 12.3% improvement, while graph smoothness constraints render another 7.3% gain of accuracy in NDCG@k.

(2) Compared with SVMRank, SPR, and DPR, TGNet improves the accuracy by 45%, 25% and 59% on average, respectively. Indeed, SVMRank only exploits input node features, SPR uses only structure features, and DPR does not leverage labeled data. These justify the need of both structural and temporal context in supervised node ranking for temporal networks. Interestingly, TGNet improves the accuracy the most for MAG, which indicates the importance of exploiting temporal and structural context jointly in determining the impact of authors in citation networks.

(3) TGNet outperforms domain-specific models LRT and BGCM for alert ranking. While both LRT and BGCM assume the quality of anomaly detectors is uniformly distributed, TGNet makes a decision based on extracted context features: the quality of an anomaly detector is conditioned by contexts. This key difference in TGNet brings in average 95% improvement of accuracy in NDCG@k.

We next perform an in-depth analysis for critical parameters.

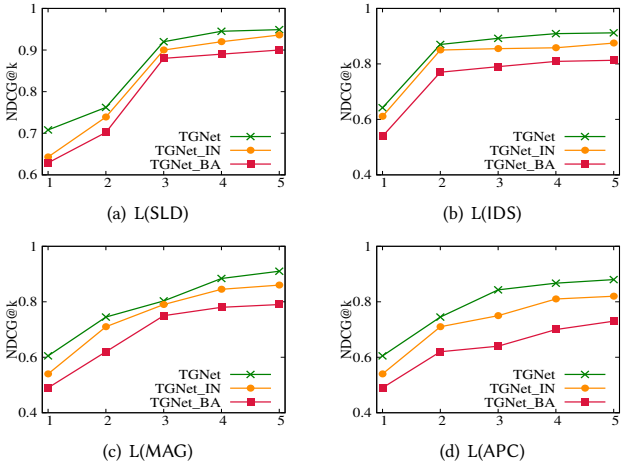
Impact of d_h. We evaluate the impact of d_h to the accuracy of TGNet in Figure 4, by varying it from 1 to 15. (1) The accuracy of all the



(a) d_h(SLD) (b) d_h(IDS) (c) d_h(MAG) (d) d_h(APC)

Figure 4: Accuracy vs. d_h (Number of hidden state dimensions)

three TGNet models increases when d_h becomes larger. Indeed, larger d_h indicates a better capability of encoding complex context features. (2) TGNet achieves good accuracy (NDCG $>$ 0.92 for SLD) with small d_h ($\leq$ 5), and outperforms its less enhanced variants.

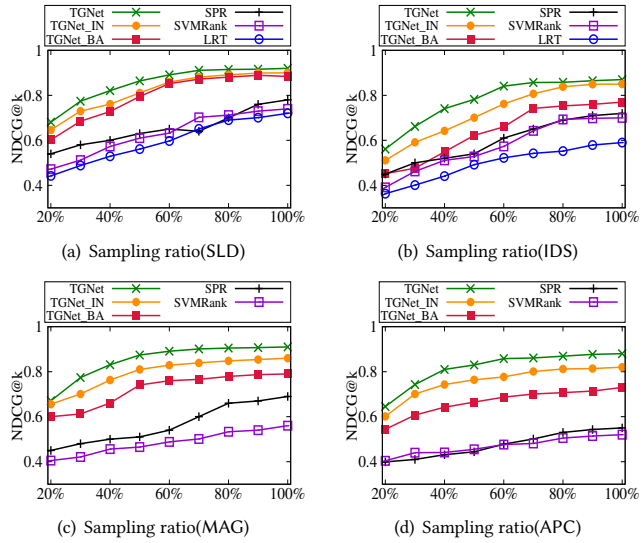


(a) L(SLD) (b) L(IDS) (c) L(MAG) (d) L(APC)

Figure 5: Accuracy vs. L (# of structural propagation iteration)

Impact of L. As shown in Fig. 5, the accuracy of TGNet and its variants TGNet_BA and TGNet_IN increases when L is larger. This indicates that collecting information from more hops of neighbors improves the ranking quality. Meanwhile, prediction performance tends to converge when L is small, with marginal improvement. In SLD, TGNet achieves reasonable accuracy(NDCG $>$ 0.93) with small L ($L \leq 4$).

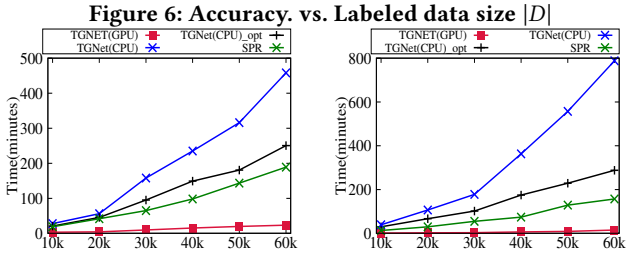
Impact of $|D|$. We also evaluate the impact of the size of labeled data $|D|$ to the accuracy of TGNet. We "sparsify" the labeled data with random sampling. Figure 6 shows the following. While all the methods have higher accuracy when more labeled data are given, TGNet needs the least amount of labeled data to achieve the highest accuracy due to its more enhanced influence network and smoothness constraints. For example, it achieves an accuracy of NDCG@k=0.83 (resp. NDCG@k=0.85) using 40% (resp. 60%) labeled data in SLD (resp. IDS), while SVMRank has accuracy up to 0.74 (resp. 0.7) using all the labeled data.



(a) Sampling ratio(SLD) (b) Sampling ratio(IDS)

(c) Sampling ratio(MAG) (d) Sampling ratio(APC)

Figure 6: Accuracy. vs. Labeled data size $|D|$



(a) Synthetic (SLD) (b) Synthetic (IDS)

Figure 7: Learning cost vs. Graph size (# of snapshots)

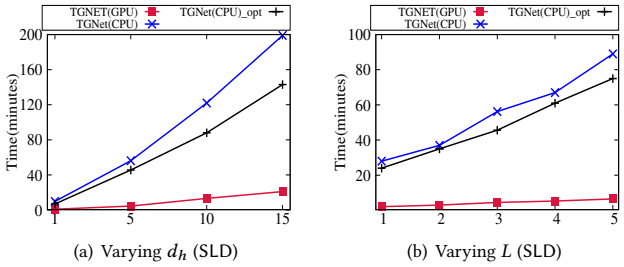
Exp-2: Efficiency. We next evaluate the efficiency and scalability of parameter learning of TGNet. The training of TGNet using GPU implementation is quite efficient. It takes up to 4.5, 54, 48 and 29.6 minutes to train TGNet over SLD, IDS, MAG and APC with 19.4k (15 days' data with 20k training pairs), 66.7k (20 hours' data with 100k training pairs), 12k (30 years' data with 100k training pairs) and 1.5k (5 years' data with 100k training pairs) snapshots, respectively. The training of TGNet is 10 times faster than that of SPR.

For a fair comparison to other methods, we "downgraded" TGNet to CPU-based implementations: TGNet(CPU)_opt and its variant TGNet(CPU) without optimization. We compare both with CPU-based SPR. By default, we use 20k of training node pairs to evaluate the impact of different parameters on learning cost.

Impact of Graph Size. Varying the number of snapshots for synthetic graphs from $10k$ to $60k$, we report the learning time in Fig. 7. (1) With optimizations (Section 5), it is feasible to train TGNet with CPUs over large graphs: TGNet(CPU)_opt incurs a comparable cost with SPR, while achieving much higher accuracy. (2) The optimization improves the learning efficiency better over larger temporal graphs. For example, for SLD, TGNet(CPU)_opt is twice faster than TGNet(CPU) over a temporal graph with 60K snapshots.

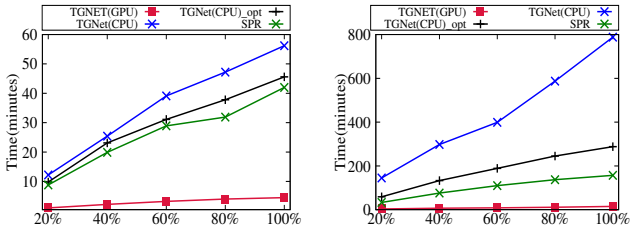
Impact of d_h. As shown in Figure 8(a), the learning cost of TGNet and its variants increases when d_h is larger, and the response time grows (almost) linearly with d_h.

Impact of L. Varying L from 1 to 5, the learning cost increases as we perform more iterations of structural propagations, and is not



(a) Varying d_h (SLD) (b) Varying L (SLD)

Figure 8: Learning cost vs. parameters



(a) Synthetic (SLD) (b) Synthetic (IDS)

Figure 9: Learning cost vs. Labeled data size $|D|$

very sensitive (Figure 8(b)). This is consistent with the impact of d_h, and verifies our cost analysis in Section 5. We observe that TGNet converges to high accuracy with small d_h and L on the employed datasets (see **Exp-1**).

Impact of $|D|$. The impact of the size of labeled data ($|D|$) is shown in Figure 9. Use random sampling, we select a subset of examples from the 20k training pairs for both SLD and IDS. While all methods take more time given more training examples, TGNet is the least sensitive due to early terminating, and requires much less examples to achieve high accuracy (see **Exp-1**).

Exp-3: Real-world Case Study. To evaluate the application of TGNet, we inspected and report the top ranked nodes over real-world datasets in the following use cases.

Alert ranking. Figure 10 shows a top ranked alert a_1, and a low ranked alert a_2, along with their contexts reported by TGNet over SLD. We can readily trace the structural propagation (solid arrows) associated with maximum structural influence among all iterations, and the temporal propagation (dashed arrows) associated with temporal influence. This yields intuitive context (shown in Figure 10).

(1) TGNet gathered the context of a_1 starting from the snapshot at 12:18:38, in which l_1 and l_2 suggest a potential issue in the SQLServer at h_1 as connection errors are reported. From 12:18:41 to 12:18:42, multiple user login requests also fail at the SQLServer at h_1. At 12:18:42, the evidence from different sources strongly suggest a service failure in the SQLServer at h_1, which needs attention. By capturing the strong correlation among logs that are most relevant to system admins' interest, TGNet decides to rank a_1 high.

(2) TGNet gives a_2 a low priority. During training, TGNet learns there is little correlation between what h_1 has experienced and logs from SQLServer Agent, and any alerts from SQLServer Agent rarely trigger system admins' interest.

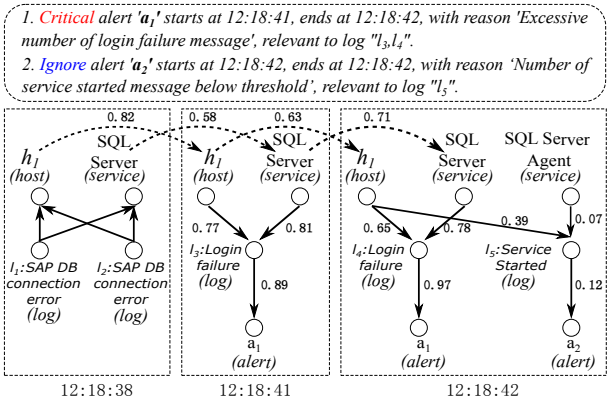


Figure 10: Contexts support alert ranking in SLD.

The two alerts, among other top ranked ones, have been validated by domain experts as critical events. This indicates the practical application of TGNet in system management. Other methods fail to rank these alerts as expected.

Author ranking. Using structural and temporal context, TGNet can suggest researchers with high academic influence with high ranking scores (*e.g.,* "Herbert A. Simon"). We found that SVMRank can not rank these authors correctly using node features alone. The authors ranked higher by SPR usually have more publications but with relatively low total citations. TGNet distinguishes such scenarios and ranks those authors lower with enhanced influence model.

7 CONCLUSION

In this paper, we introduce TGNet, a novel graph neural network that explicitly tackles challenges in node ranking for temporal graphs. We propose influence network to boost the capability of modeling context dynamics, coupled with graph smoothness constraints to cope with label sparsity. We develop an end-to-end learning algorithm for TGNet with optimization techniques. Our experimental study verifies that TGNet outperforms the baselines in terms of accuracy, and can be generalized from sparse labeled data. Our case study verifies that TGNet yields intuitive context of node ranks that can be interpreted.

Acknowledgments. This work is supported in part by NSF IIS-1633629 and an HIRP grant.

REFERENCES

[1] Martín Abadi et al. 2016. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467* (2016).
[2] Serge Abiteboul, Mihai Preda, and Gregory Cobena. 2003. Adaptive on-line page importance computation. In *WWW*. 280–290.
[3] Alekh Agarwal and Soumen Chakrabarti. 2007. Learning random walks to rank nodes in graphs. In *ICML*. 9–16.
[4] Alekh Agarwal, Soumen Chakrabarti, and Sunny Aggarwal. 2006. Learning to rank networked entities. In *KDD*. 14–23.
[5] Shivani Agarwal. 2006. Ranking on graph data. In *ICML*. 25–32.
[6] Nesreen K Ahmed, Jennifer Neville, and Ramana Kompella. 2014. Network sampling: From static to streaming graphs. *TKDD* 8, 2 (2014), 7.
[7] Nipun Arora. 2017. NGLA: Next Genration Log Analytics. http://www.nipunarora.net/project/ngla/. (2017).
[8] Lars Backstrom and Jure Leskovec. 2011. Supervised random walks: predicting and recommending links in social networks. In *WSDM*. ACM, 635–644.
[9] Shiyu Chang, Yang Zhang, Jiliang Tang, Dawei Yin, Yi Chang, Mark A Hasegawa-Johnson, and Thomas S Huang. 2016. Positive-unlabeled learning in streaming networks. In *KDD*. 755–764.
[10] Wei Cheng, Kai Zhang, Haifeng Chen, Guofei Jiang, Zhengzhang Chen, and Wei Wang. 2016. Ranking causal anomalies via temporal and dynamical analysis on vanishing correlations. In *KDD*. 805–814.
[11] Cheng-Tao Chu, Sang K Kim, Yi-An Lin, YuanYuan Yu, Gary Bradski, Kunle Olukotun, and Andrew Y Ng. 2007. Map-reduce for machine learning on multicore. In *NIPS*. 281–288.
[12] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. 2016. Convolutional neural networks on graphs with fast localized spectral filtering. In *NIPS*. 3844–3852.
[13] Hongbo Deng, Michael R Lyu, and Irwin King. 2009. A generalized co-hits algorithm and its application to bipartite graphs. In *KDD*. 239–248.
[14] Ying Ding, Erjia Yan, Arthur Frazho, and James Caverlee. 2009. PageRank for ranking authors in co-citation networks. *JASIST* 60, 11 (2009), 2229–2243.
[15] Bin Gao, Tie-Yan Liu, Wei Wei, Taifeng Wang, and Hang Li. 2011. Semi-supervised ranking on very large graphs with rich metadata. In *KDD*. 96–104.
[16] Jing Gao, Wei Fan, Deepak Turaga, Olivier Verscheure, Xiaoqiao Meng, Lu Su, and Jiawei Han. 2011. Consensus extraction from heterogeneous detectors to improve performance over network traffic anomaly detection. In *INFOCOM, 2011 Proceedings IEEE*. 181–185.
[17] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. 2017. Neural Message Passing for Quantum Chemistry. In *ICML*. 1263–1272.
[18] Guofei Gu, Alvaro A Cárdenas, and Wenke Lee. 2008. Principled reasoning and practical applications of alert fusion in intrusion detection systems. In *ASIACCS*. 136–147.
[19] Wentian Guo, Yuchen Li, Mo Sha, and Kian-Lee Tan. 2017. Parallel personalized pagerank on dynamic graphs. *VLDB* 11, 1 (2017), 93–106.
[20] Chin-Chi Hsu, Yi-An Lai, Wen-Hao Chen, Ming-Han Feng, and Shou-De Lin. 2017. Unsupervised Ranking using Graph Structures and Node Attributes. In *WSDM*. 771–779.
[21] Thorsten Joachims. 2006. Training linear SVMs in linear time. In *KDD*. 217–226.
[22] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).
[23] Yann A LeCun, Léon Bottou, Genevieve B Orr, and Klaus-Robert Müller. 2012. Efficient backprop. In *Neural networks: Tricks of the trade*. 9–48.
[24] Jure Leskovec, Lada A Adamic, and Bernardo A Huberman. 2007. The dynamics of viral marketing. *TWEB* 1, 1 (2007), 5.
[25] Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel. 2015. Gated graph sequence neural networks. *arXiv preprint arXiv:1511.05493* (2015).
[26] Tie-Yan Liu et al. 2009. Learning to rank for information retrieval. *Foundations and Trends® in Information Retrieval* 3, 3 (2009), 225–331.
[27] Emaad Manzoor, Sadegh M Milajerdi, and Leman Akoglu. 2016. Fast memory-efficient anomaly detection in streaming heterogeneous graphs. In *KDD*. 1035–1044.
[28] Joshua O'Madadhain and Padhraic Smyth. 2005. EventRank: A framework for ranking time-varying networks. In *LinkKDD*. 9–16.
[29] Delip Rao and David Yarowsky. 2009. Ranking and semi-supervised classification on large scale graphs using map-reduce. In *TextGraphs*. 58–65.
[30] Ryan A Rossi and David F Gleich. 2012. Dynamic pagerank using evolving teleportation. In *WAW*. 126–137.
[31] Ali Shiravi, Hadi Shiravi, Mahbod Tavallaee, and Ali A Ghorbani. 2012. Toward developing a systematic approach to generate benchmark datasets for intrusion detection. *computers & security* 31, 3 (2012), 357–374.
[32] Arnab Sinha, Zhihong Shen, Yang Song, Hao Ma, Darrin Eide, Bo-june Paul Hsu, and Kuansan Wang. 2015. An overview of microsoft academic service (mas) and applications. In *WWW*. 243–246.
[33] Ah Chung Tsoi, Gianni Morini, Franco Scarselli, Markus Hagenbuchner, and Marco Maggini. 2003. Adaptive ranking of web pages. In *WWW*. 356–365.
[34] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. 2018. Graph attention networks. *ICLR* (2018).
[35] Wei Wei, Bin Gao, Tie-Yan Liu, Taifeng Wang, Guohui Li, and Hang Li. 2016. A ranking approach on large-scale graph with multidimensional heterogeneous information. *IEEE transactions on cybernetics* 46, 4 (2016), 930–944.
[36] Wenlei Xie, David Bindel, Alan Demers, and Johannes Gehrke. 2015. Edge-weighted personalized pagerank: Breaking a decade-old performance barrier. In *KDD*. 1325–1334.
[37] Chen Yu, Ruidan Li, Dezhong Yao, Feng Lu, and Hai Jin. 2014. Temporal-based ranking in heterogeneous networks. In *NPC*. 23–34.
[38] Quan Yuan, Gao Cong, and Aixin Sun. 2014. Graph-based point-of-interest recommendation with geographical and temporal influences. In *CIKM*. 659–668.
[39] Hongyang Zhang, Peter Lofgren, and Ashish Goel. 2016. Approximate Personalized PageRank on Dynamic Graphs. In *KDD*. 1315–1324.
[40] Xiaojin Zhu, John Lafferty, and Ronald Rosenfeld. 2005. *Semi-supervised learning with graphs*. Ph.D. Dissertation. Carnegie Mellon University.
[41] Bo Zong, Yinghui Wu, Jie Song, Ambuj K Singh, Hasan Cam, Jiawei Han, and Xifeng Yan. 2014. Towards scalable critical alert mining. In *KDD*. 1057–1066.