Scalable Temporal Latent Space Inference for Link Prediction in Dynamic Social Networks

Linhong Zhu, *Member, IEEE*, Dong Guo, Junming Yin, Greg Ver Steeg, and Aram Galstyan

Abstract—We propose a temporal latent space model for link prediction in dynamic social networks, where the goal is to predict links over time based on a sequence of previous graph snapshots. The model assumes that each user lies in an unobserved latent space, and interactions are more likely to occur between similar users in the latent space representation. In addition, the model allows each user to gradually move its position in the latent space as the network structure evolves over time. We present a global optimization algorithm to effectively infer the temporal latent space. Two alternative optimization algorithms with local and incremental updates are also proposed, allowing the model to scale to larger networks without compromising prediction accuracy. Empirically, we demonstrate that our model, when evaluated on a number of real-world dynamic networks, significantly outperforms existing approaches for temporal link prediction in terms of both scalability and predictive power.

Index Terms—Latent space model, link prediction, non-negative matrix factorization, social network analysis

1 Introduction

Understanding and characterizing the processes driving social interactions is one of the fundamental problems in social network research. A particular instance of this problem, known as *link prediction*, has recently attracted considerable attention in various research communities. Besides academic interest (see [1], [2] for a survey of different methods), link prediction has many important commercial applications, e.g., recommending friends in an online social network such as Facebook and suggesting potential hires in a professional network such as LinkedIn.

In this work we focus on the temporal link prediction problem: Given a sequence of graph snapshots $G_1, \ldots, G_t$ from time 1 to t, how do we predict links in future time $t+1$? To perform link prediction in a network, one needs to construct models for link probabilities between pairs of nodes. Recent research interests in link prediction have focused on latent space modeling of networks. That is, given the observed interactions between nodes, the goal is to infer the position of each node in some latent space, so that the probability of a link between two nodes depends on their positions in that space. Latent space modeling allows us to naturally incorporate the well-known homophily effect [3] (birds of a feather flock together). Namely, each dimension of the latent space characterizes an unobservable homogeneous attribute, and shared attributes tend to create a link in a network. We illustrate this concept with an example shown in Fig. 1.

- *L. Zhu, G. Ver Steeg, and A. Galstyan are with the Information Sciences Institute, University of Southern California, Los Angeles, CA 90089. E-mail: {linhong, gregv, galstyan}@isi.edu.*
- *D. Guo is with the Department of Computer Science, University of Southern California, Los Angeles, CA 90089. E-mail: dongguo@usc.edu.*
- *J. Yin is with the Department of Management Information Systems, University of Arizona, Tucson, AZ 85721. E-mail: junmingy@email.arizona.edu.*

Manuscript received 25 Feb. 2016; revised 25 June 2016; accepted 27 June 2016. Date of publication 13 July 2016; date of current version 7 Sept. 2016. Recommended for acceptance by L.B. Holder.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TKDE.2016.2591009

Example 1. Assume that we have observed some interactions during a political campaign. An example of a latent space is shown in Fig. 1, where most observed links were formed between users with similar political attitudes. Based on the learned latent space, we could then predict that Bob is more likely to interact with Alice than Kevin in a political campaign.

Various approaches, including Bayesian inference [4], [5], multidimensional scaling (MDS) [6], matrix factorization [7], [8], [9], [10], [11], [12], and mixed-membership model [13], have been proposed to infer the static latent space representation of the observed network. Nevertheless, most of these studies were focusing on static graphs, where the latent positions of the nodes are fixed. In social networks, "There is nothing permanent except change"; the network itself is dynamic and changing with time [14], [15]. In addition, the latent position of each node might also be evolving over time [6], [16], [17], [18], [19]. A naive approach to extend static latent space modeling for dynamic networks is to model each node as a single latent representation and then update its position whenever the network evolves. Unfortunately, this approach tends to overfit on the current time step, leading to abrupt transitions and poor incorporation of historical information. Therefore, we are interested in inferring temporal latent positions for all the nodes in a dynamic network. The underlying problem can be stated as follows: *Given a sequence of graph snapshots $G_1, \ldots, G_t$, how can we infer the temporal latent space so that at each time point links are more likely to be formed between nodes that are closer in the temporal latent space?* With the inferred temporal latent space, links can be accurately predicted at future times.

Unfortunately, we still have a limited understanding of how the latent space is evolving for temporal social networks. In this work we propose a *Temporal Latent Space Model for Dynamic Link Prediction*. Our model is based on the intuition that nodes can move smoothly in the latent space over time, and that large moves are less likely [6], [19]. As illustrated in Fig. 2, in time $t = 1$, Bob is biased towards

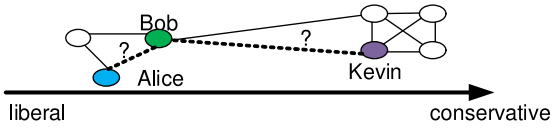


Fig. 1. An example of observed interactions among Alice, Bob, and Kevin, and their positions in a simplified one-dimension latent space. Alice is very liberal, Bob is biased towards being liberal, and Kevin is very conservative. However, all of their profile information as well as the dimension label are unobservable.

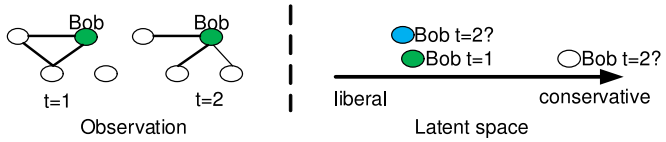


Fig. 2. An example of temporal latent positions of user Bob. With the temporal smoothness assumption, Bob is more likely to be in the blue position than the white position in time $t = 2$.

liberal; it is unlikely that Bob will suddenly move to very conservative in time $t = 2$. In addition to temporal smoothness, our model imposes a constraint on the dimensionality of the latent space and assumes that the dimension of latent space is much smaller than the number of nodes. With the dimensionality constraint, the online link prediction using our model is efficient in both computational time and storage cost (see related work for more details). In addition, varying the dimension of latent space offers an opportunity to fine-tune the compromise between computational cost and solution quality. The higher dimension leads to a more accurate latent space representation of each node, but also yields higher online computational cost.

One of the most widely used methods for inferring low-rank latent space in networks is via matrix factorization with the "block-coordinate gradient descent (BCGD)" algorithm [20], [21], which has been successfully applied in community detection for static networks [22]. However, the introduced additional temporal smoothness term in the objective function creates new challenges for the BCGD approach. Moreover, while there is a lot of work on studying convergence of BCGD, it is unknown whether the new objective function satisfies these criteria. We prove that the global BCGD algorithm proposed in this work has a quadratic convergence rate.

The global BCGD algorithm is computationally expensive and not applicable for truly large-scale problems. Therefore, for the temporal link prediction problem with latent space models, a significant gap remains between theoretical capabilities and practical applications. Towards this end, in this work we further introduce two new variants of BCGD: a local BCGD algorithm and an incremental BCGD algorithm. In a local BCGD algorithm, instead of using all historical graph snapshots to jointly infer temporal latent space at all timestamps, it sequentially infers the temporal latent space at each time position with a single graph snapshot and previous temporal latent space—thus significantly reducing computational cost. In addition, as we stated earlier, the latent positions of nodes are changing smoothly over time, not suddenly. To make use of this temporal smoothness, we develop an incremental BCGD algorithm to adaptively infer changes in latent positions based only on the changes in interactions. As shown in Fig. 2, when Bob receives new interactions, his latent position in time $t = 2$ requires an update which, however, can be performed efficiently leveraging his previous latent position in time $t = 1$.

In summary, the contributions of this work are:

1) We propose a temporal latent space model for dynamic networks to model the temporal link probability of node pairs, which can be further used to accurately recover or predict the formation of links.
2) We address algorithmic issues in learning the temporal latent space representation of nodes by developing the standard global BCGD algorithm. We also provide a set of theoretical results for the standard global BCGD algorithm.
3) We develop two fast BCGD algorithms: a local BCGD algorithm and an incremental BCGD algorithm. Furthermore, we illustrate that the proposed incremental BCGD algorithm only requires a conditional update on a small set of affected nodes, and is therefore significantly faster than all existing approaches.
4) We conduct experiments over large real-life graphs. The experimental results show that the proposed approaches—global BCGD, local BCGD, and incremental BCGD—achieve good predictive quality in both sparse and dense networks with average AUC scores of 0.81, 0.79 and 0.78 respectively. In addition, it takes only less than half an hour for the incremental BCGD algorithm to learn the temporal latent spaces of massive networks with millions of nodes and links.

The remainder of the paper is organized as follows: We first introduce our problem formulation in Section 2, and then present the proposed global BCGD algorithm and two fast BCGD algorithms in Sections 3 and 4 respectively. In Section 5, we conduct an experimental study to evaluate the effectiveness and efficiency of our approach. Related work is discussed in Section 6. We conclude this work and present future work in Section 7.

2 Problem Formulation

A graph G is denoted as (V, E), where V is the set of nodes and $E \subseteq V \times V$ is the set of (directed or undirected) interactions. In this work we first focus on undirected graphs, in which case its matrix representation is symmetric. We use u and v to denote individual nodes, and t and τ to denote timestamps. Let $G_\tau = (V_\tau, E_\tau)$ be a time-dependent network snapshot recorded at time τ, where V_τ is the set of nodes and $E_\tau \subseteq V_\tau \times V_\tau$ is the set of interactions. In addition, denote by ΔV_τ and ΔE_τ the sets of vertices and interactions to be introduced (or removed) at time τ, and let $\Delta G_\tau = (\Delta V_\tau, \Delta E_\tau)$ denote the change in the whole network.

Dynamic Social Network. A dynamic network G is a sequence of network snapshots within a time interval and evolving over time: $G = (G_1, \ldots, G_t)$.

Temporal Latent Space and Its Model. Let Z_τ be the low-rank k-dimension temporal latent space representation for node set V_τ at time τ. For each individual u at time τ, we use a row vector $Z_\tau(u)$ to denote its temporal latent space

representation and a scalar $Z(u,c)$ to denote its position in cth dimension. Inspired by the dynamic community model [15], [23], we impose the following assumptions on the temporal latent space:

1) *Temporal smoothness*: Individuals change their latent positions gradually over time.
2) *Network embedding*: Two nodes that are close to each other in the interaction network, in terms of the network distance, are also close to each other in the temporal latent space in terms of latent space distance.
3) *Latent Homophily*: Two members who are close to each other in latent space interact with one another more frequently than two faraway members.

Within our model, we assume that the probability of a link between two nodes depends only on their latent positions. In principle, given a dynamic graph $G = (G_1, \ldots, G_t)$, we need Z_{t+1} to predict future graph G_{t+1}. However, since Z_{t+1} is not available, in our model we assume it can be approximated by $Z_1, \ldots, Z_t$. Therefore, given a dynamic graph $G = (G_1, \ldots, G_t)$, we infer $Z_1, \ldots, Z_t$ based on $G_1, \ldots, G_t$, use $Z_1, \ldots, Z_t$ to approximate Z_{t+1}, and finally predict G_{t+1}.

With the above definitions, we focus on the following problem:

Problem 1 (Temporal Latent Space Inference). *Given a dynamic social network $G = (G_1, G_2, \ldots, G_t)$, we aim to find a k-dimension latent space representation at each timestamp Z_τ that minimizes the quadratic loss with temporal regularization*

$$\begin{aligned} &\arg\min_{Z_1,\ldots,Z_t} \sum_{\tau=1}^{t} \|G_\tau - Z_\tau Z_\tau^T\|_F^2 + \lambda \sum_{\tau=1}^{t} \sum_{u} (1 - Z_\tau(u) Z_{\tau-1}(u)^T) \\ &\text{subject to}: \forall u, \tau, Z_\tau \geq 0, Z_\tau(u) Z_\tau(u)^T = 1, \end{aligned} \tag{1}$$

where λ is a regularization parameter, and the term $(1 - Z_\tau(u) Z_{\tau+1}(u)^T)$ penalizes node u for suddenly changing its latent position. Note that when computing the quadratic loss $\|G_\tau - Z_\tau Z_\tau^T\|_F^2$, we ignore all of the diagonal entries.

In the above model, the latent representation of each node corresponds to a point on the surface of a unit hypersphere. Note that this is different from mixed membership stochastic block models [13] where nodes are mapped onto simplex. In practice, we find that sphere modeling gives us a clearer boundary between linked pairs and non-linked pairs when we project all pairs of nodes into the latent space. In addition, we impose the constraints $Z_\tau \geq 0$, not only because the non-negativity establishes the duality between our modeling and non-negative matrix factorization, but also because it gives latent space an intuitive parts-based interpretation, as suggested by Lee and Seung [24]. In the facial image example [24], with non-negative constraints, each dimension of latent space corresponds to a part of the face, such as nose, eye and ear; while without non-negative constraints, each dimension of latent space corresponds to a blur representation of the entire face, of which it is very difficult to interpret. Similarly, in the social network domain, with the non-negative constraints, each dimension of latent space corresponds to a part of users' attributes such as current city, hometown, personality and so on. It is more intuitive to represent each user as an additive mixture of attributes (current city: A, hometown: B, personality: openness to experience), rather than represent each user as a combination of different representatives.

Link Prediction. Given that we have inferred $Z_1, \ldots, Z_t$ by optimizing Eq. (1), our goal is to predict the adjacency matrix G_{t+1} at the next timestamp $t+1$. The most natural estimator is the conditional expectation: $Y_{t+1} = E[G_{t+1} \mid Z_1, \ldots, Z_t]$. By assuming that the temporal dynamics of latent positions is Markovian and satisfies $E[Z_{t+1} \mid Z_1, \ldots, Z_t] = Z_t$ and $\mathrm{cov}[Z_{t+1}^T \mid Z_1, \ldots, Z_t] = D_t$ (a diagonal matrix), as well as $Z_{t+1} Z_{t+1}^T$ an unbiased estimate of G_{t+1}, we obtain

$$\begin{aligned} Y_{t+1} &= E[G_{t+1} \mid Z_1, \ldots, Z_t] \\ &= E[E[G_{t+1} \mid Z_1, \ldots Z_t, Z_{t+1}] \mid Z_1, \ldots, Z_t] \\ &= E[Z_{t+1} Z_{t+1}^T \mid Z_1, \ldots, Z_t] \\ &= \mathrm{var}[Z_{t+1}^T \mid Z_1 \ldots Z_t] \\ &\quad + E[Z_{t+1} \mid Z_1 \ldots Z_t] E[Z_{t+1} \mid Z_1 \ldots Z_t]^T \\ &= D_t + Z_t Z_t^T. \end{aligned}$$

However, as we ignore the diagonal entries of the adjacency matrix (the graph does not contain self-loops) and because the conditional covariance matrix D_t is assumed to be diagonal, we will use the off-diagonal of $Z_t Z_t^T$ to predict G_{t+1}.

Generalization. Although in this work, we use $Z_t Z_t^T$ to predict G_{t+1}, without losing generality, the predicted graph at $t+1$ can be formulated as: $Y_{t+1} = \Phi(f(Z_1, \ldots, Z_t))$, where Φ is the link function and f is the temporal function. Learning or selecting the best link function Φ and temporal function f is beyond the scope of this work. For example, we could apply nonparametric approaches [25] to automatically learn the link function Φ. Additionally, though this work focuses on undirected graphs, our model can be generalized to directed and weighted graphs. Specifically, this can be done by introducing another matrix B, which represents the weighted mapping from one dimension to another dimension in the latent space. With the matrix B, the term $G_\tau - Z_\tau Z_\tau^T$ in Eq. (1) is now generalized to $G_\tau - Z_\tau B Z_\tau^T$. If B is symmetric, the underlying graph is undirected; otherwise, it is directed.

Problem Complexity. Vavasis recently proved that non-negative matrix factorization (NMF) is NP-hard [26]. In addition, with the separability condition, that is, for each dimension c in the latent space Z, there is a node u such that $Z(u,c) \geq p$ and $Z(u,c') = 0$ for $c' \neq c$, there exists a polynomial time exact algorithm to solve the non-negative matrix factorization problem [27]. Unfortunately, in our modeling, there is no guarantee that the latent space Z satisfies the separability condition. In addition, even if there exists a latent space Z that satisfies the separability condition, Recht et al. [28] pointed out that an exact algorithm is still too computationally expensive for large-scale data. Therefore, in the following we focus on the block coordinate gradient descent approach to obtain an approximate solution to Problem 1.

3 A Standard BCGD Algorithm

In this section we present the details of the standard BCGD algorithm that provide a local optimal solution to Problem 1.

With a partial observed graph structure, the objective function in Eq. (1) can be decomposed into a linked part and a non-linked part. That is,

$$\begin{aligned}\arg\min_{Z_1,\ldots,Z_t} &\sum_{\tau=1}^{t}\sum_{u,v\in E_\tau}(G_\tau(u,v)-Z_\tau(u)Z_\tau(v)^T)^2\\ &+\sum_{\tau=1}^{t}\sum_{u,v\notin E_\tau}(Z_\tau(u)Z_\tau(v)^T)^2\\ &+\lambda\sum_{\tau=1}^{t}\sum_{u}(1-Z_\tau(u)Z_{\tau-1}(u)^T)\\ \text{subject to: } &\forall\, u,\tau,\ Z_\tau\geq 0, Z_\tau(u)Z_\tau(u)^T=1.\end{aligned} \tag{2}$$

Unfortunately, the decomposed objective function in Eq. (2) for Problem 1 is still a fourth-order polynomial and non-convex. Therefore, we then adopt a block coordinate descent approach to solve Eq. (2). We update $Z_\tau(u)$ for each node u at time τ by fixing both latent positions $Z_\tau(v)$ of all other nodes v at time τ as well as all the temporal latent positions other than at time τ. Proceeding in this way, each step of block coordinate descent is solving a convex problem.

For each node u at time τ, we focus on optimizing the following problem:

$$\begin{aligned}&\arg\min_{Z_\tau(u)\geq 0} J(Z_\tau(u)),\ \text{where } J(Z_\tau(u))=\\ &\sum_{v\in N(u)}(G_\tau(u,v)-Z_\tau(u)Z_\tau(v)^T)^2+\sum_{v\notin N(u)}(Z_\tau(u)Z_\tau(v)^T)^2\\ &+\lambda(1-Z_{\tau+1}(u)Z_\tau(u)^T)+\lambda(1-Z_\tau(u)Z_{\tau-1}(u)^T)\}.\end{aligned} \tag{3}$$

In the following, we use the projected gradient descent algorithm to find the approximation solution with a non-negativity constraint. With the gradient descent optimization algorithm, for each node u at timepoint τ, we could iteratively update $Z_\tau(u)$ in each iteration $r+1$ with the following rule:

$$Z_\tau^{(r+1)}(u)=Z_\tau^{(r)}(u)-\alpha\nabla_{Z_\tau(u)}J(Z_\tau^{(r)}(u)),$$

where α is the step size.

Step Size. Nesterov's gradient method [29], [30], [31] iteratively updates the step size using the Lipschitz constant. According to the following lemma, we show that our step size can also be iteratively updated using the Lipschitz constant.

Lemma 1. *The gradient of the objective function $J(Z_\tau(u))$ is Lipschitz continuous, and the Lipschitz constant L is equal to $2\sqrt{n^2-2n+k}$, where n is the number of nodes in a graph, and k is the number of dimensions.*

Proof. A detailed proof is in Supplement A, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TKDE.2016.2591009. □

With the Lipschitz constant and $\nabla_{Z_\tau(u)}J(Z_\tau^{(r)}(u))$ shown in Eq. (12) in Supplement A, available online, each node u at timepoint τ can be computed via the update rule stated in the following Lemma.

Lemma 2. *The latent position of each node u at timepoint τ can be iteratively computed with the following update rule:*

$$\begin{aligned}Z_\tau^{(r+1)}(u)=max((1+2\alpha)Z_\tau^{(r)}(u)+\alpha\lambda(Z_{\tau-1}^{(r)}(u)+Z_{\tau+1}^{(r)}(u))\\ +2\alpha\sum_{v\in N(u)}G_\tau(u,v)Z_\tau^{(r)}(v)-2\alpha Z_\tau^{(r)}(u)Z_\tau^{(r)^T}Z_\tau^{(r)},0),\end{aligned} \tag{4}$$

where $\alpha=\frac{a_{r+1}+a_r-1}{a_{r+1}L}$, $N(u)$ denotes the set of u's neighbors, L is the Lipschitz constant, $d(u)$ is the degree of node u, and

$$a_r=\begin{cases}1 & if\ r=0,\\ \frac{1+\sqrt{4a_{r-1}^2+1}}{2} & if\ r>0.\end{cases}$$

Proof. A detailed proof is in Supplement B, available online. □

Note that when updating $Z_\tau(u)$ according to Lemma 2, it requires the latent positions of all the neighbors of u. For nodes with extra-large degrees, the above update will become time-consuming. One interesting direction in the future is to apply stochastic gradient descent update rules to nodes with large degrees. That is, instead of using the latent positions of all of u's neighbors, each iteration randomly selects one of u's neighbors v, and leverages v's latent positions to update the latent position of u.

We summarize our global block coordinate descent approach to solve Eq. (2) in Algorithm 1. In the following, we provide additional theoretical properties of the proposed BCGD algorithm, including convergence analysis and complexity analysis.

Algorithm 1. The Global BCGD Algorithm for Jointly Inferring Temporal Latent Space

Input: Graphs $\{G_1,\ldots,G_t\}$ and latent space dimension k
Output: latent space $\{Z_1,\ldots,Z_t\}$ and prediction Y_{t+1}
1: Nonnegative initial guess for $\{Z_1,\ldots,Z_t\}$
2: **Repeat**
3: **for** each time τ from 1 to t
4: **for** each u in graph
5: update $Z_\tau(u)$ by Eq. (4)
6: normalize $Z_\tau(u)$
7: **until** $\{Z_1,\ldots,Z_t\}$ converges.
8: **return** $Y_{t+1}=\Phi([Z_1,\ldots,Z_t])$ and $\{Z_1,\ldots,Z_t\}$

3.1 Theoretical Analysis

Convergence Rate. Since our algorithm uses Nesterov's gradient method to determine the step size (see Lemma 2), we can conclude that our algorithm achieves the convergence rate $O(\frac{1}{r^2})$ as stated in the following theorem.

Theorem 1. *Given the sequence generated by Algorithm 1, for each node u at timepoint τ, we have $J(Z_\tau^{(r)}(u))-J(Z_\tau^*(u))\leq\frac{2L\|Z_\tau^I(u)-Z_\tau^*(u)\|_F^2}{(r+2)^2}$, where $Z_\tau^*(u)$ is the optimum solution for $Z_\tau(u)$ with respect to the subproblem in Eq. (3), $Z_\tau(u)^I$ is the initialization for $Z_\tau(u)$, L is the Lipschitz constant in Lemma 1,*

TABLE 1
Time Complexity Analysis for Algorithm 1, n is the Number of Nodes, m_τ is the Number of Edges in Graph G_τ, $d(u)$ is the Degree of Node u, k is the Number of Dimensions, and T is the Number of Timestamps

	Sparse	Dense
Initialize	$O(\sum_\tau (n+m_\tau)k)$	$O(n^2Tk)$
Update $Z_\tau(u)$	$O(d(u)k)$	$O(nk)$
Convergence	$O(\sum_\tau (n+m_\tau k)$	$O(n^2Tk)$

and r is the iteration number (not the total number of iterations).

Proof. The proof is shown in Supplement C, available online. □

Local Error Bound. The solution returned by Algorithm 1 is a local optimum of the objective in Eq. (2). Unfortunately, it is not trivial to assess how this locally optimal solution compares to the global optimum of the objective in Eq. (2). This is because the objective function in Eq. (2) is non-convex, and thus arriving at the global optimum via local iterations is not guaranteed. Only if the input matrix G is separable or near-separable [27], [28], we are able to achieve the global optimum or a global error bound.

Computational Complexity. Table 1 summarizes the cost of each operation in an iteration for both sparse and dense graphs. Fortunately, Gibson et al. [32] concluded that real-world networks are generally globally very sparse, where most of nodes have a relatively small number of neighbors. Therefore, for such real-world sparse networks, the total cost of a single iteration of BCGD should be $O(\sum_\tau (n+m_\tau)k)$, which is *linear in the number of edges and nodes*. Assume that the total number of iterations is R, then the time complexity is bounded by $O(Rk\sum_\tau (n+m_\tau))$. Since in Algorithm 1, we need to store all the input matrices and the output temporal latent space matrices, the storage complexity is bounded by $O(\sum_\tau (nk+m_\tau))$.

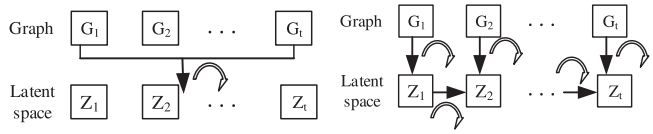
4 Fast BCGD Algorithms

Unfortunately, the standard BCGD is very expensive in both computation and storage since it requires all historical graph snapshots $\{G_1, \ldots, G_t\}$ to jointly and cyclicly update all temporal latent space $\{Z_1, \ldots, Z_t\}$. In this section we aim to further reduce the time complexity of the BCGD algorithm so that it scales with big data. In contrast to the joint inference in BCGD, we propose a sequential inference algorithm and an incremental inference algorithm, both of which utilize the locality and temporal information. The proposed two faster BCGD algorithms are as efficient as the fast independent inference (i.e., infer Z_τ from G_τ only) and as effective as the global joint inference (BCGD).

As illustrated in Fig. 3, we first introduce the local BCGD algorithm, which sequentially infers each temporal latent space Z_τ from a single graph snapshot G_τ and partial prior knowledge $Z_{\tau-1}$.

4.1 Local BCGD Algorithm

Specifically, at each timestamp τ, we aim to optimize the following local objective function to compute Z_τ:



(a) BCGD (b) Local BCGD

Fig. 3. BCGD versus local BCGD.

$$\begin{aligned}\arg\min_{Z_\tau} &\sum_{u,v\in E_\tau} (G_\tau(u,v) - Z_\tau(u)Z_\tau(v)^T)^2 \\ &+ \sum_{u,v\notin E_\tau} (Z_\tau(u)Z_\tau(v)^T)^2 \\ &+ \lambda \sum_{u\in V_\tau} (1 - Z_\tau(u)Z_{\tau-1}(u)^T).\end{aligned} \tag{5}$$

Using the same BCGD approach, we iteratively update the latent position $Z_\tau(u)$ of each node u by fixing the latent positions of all the other nodes. This leads to the following update rules for $Z_\tau(u)$ in the $(r+1)$th iteration:

$$\begin{aligned}Z_\tau^{(r+1)}(u) =& \max((1+2\alpha)Z_\tau^{(r)}(u) + \alpha\lambda Z_{\tau-1}(u) + 2\alpha \sum_{v\in N(u)} \\ & G_\tau(u,v)Z_\tau^{(r)}(v) - 2\alpha Z_\tau^{(r)}(u)Z_\tau^{(r)^T}Z_\tau^{(r)}, 0),\end{aligned} \tag{6}$$

where $\alpha = \frac{a_{r+1}+a_r-1}{a_{r+1}L}$, L is the Lipschitz constant, and a_r is defined in Lemma 2.

Algorithm 2. The Local BCGD Algorithm for Sequentially Inferring Temporal Latent Space

Input: Graphs $\{G_1, \ldots, G_t\}$ and latent space dimension k
Output: Y_{t+1} and latent space $\{Z_1, \ldots, Z_t\}$
1: Nonnegative initial guess for Z_1
2: **for** each τ from 1 to t
3: Initial Z_τ based on $Z_{\tau-1}$
4: **repeat**
5: **for** each u in graph G_τ
6: update $Z_\tau(u)$ by Eq. (6) and normalize it
7: **until** Z_τ converges.
8: **return** $Y_{t+1} = \Phi([Z_1, \ldots, Z_t])$ and $\{Z_1, \ldots, Z_t\}$

We summarize the proposed local BCGD algorithm in Algorithm 2. Note that in the global BCGD algorithm, for each iteration we jointly update Z_1, Z_2 until Z_t and then iterate back to Z_1 in the next iteration. This cyclic update schema is very expensive; while in the local BCGD algorithm, as shown in Lines 4-7, we sequentially compute Z_τ by a single inner iteration. That is, we iteratively update Z_τ until Z_τ converges and then move to the computation of temporal latent space $Z_{\tau+1}$. This local sequential update schema greatly reduces the computational cost in practice, as we analytically show in the following.

Complexity Analysis. The cost of each operation in Algorithm 2 remains the same as that of Algorithm 1, as reported in Table 1. Thus, the total computational cost is $O(\sum_\tau (n+m_\tau)R_\tau k)$. In practice, since we leverage prior knowledge $Z_{\tau-1}$ to locally update Z_τ, it converges much faster than the global BCGD algorithm, and thus the local BCGD algorithm is more efficient than the global BCGD algorithm, as also verified in our experiments. In addition, in the in-memory

algorithm where we put all the data into memory, the global BCGD algorithm requires at least $O(\sum_\tau(nk + m_\tau))$ memory cost; while the local BCGD only requires storage of a single graph snapshot and two latent space matrices representation. Therefore, the memory cost of the local BCGD algorithm is bounded by $O(nk + \max_\tau m_\tau)$.

4.2 Incremental BCGD Algorithm

In this section we further study how to infer or maintain temporal latent space incrementally with graph changes (nodes and edges insertion and deletion). Instead of recomputing the temporal latent space Z_τ with the entire graph snapshot G_τ and the prior knowledge $Z_{\tau-1}$, our dynamic update strategy is to adjust Z_τ incrementally from $Z_{\tau-1}$ as the network structure changes. Specifically, we take advantage of what we already learned in previous snapshots, as well as the temporal transition for inferring the dynamics in the current timestamp.

Overview of Incremental BCGD. The overview of this algorithm is as follows: First, we identify nodes whose local neighborhood has changed between two consecutive snapshots, including cases where an existing node adds or deletes links, or a node is added to or removed from the network. With the matrix representation, we model all the four updates towards G_t in terms of row-wise updates on matrix representation. For example, a newly added node with degree d is modeled as an update from a row with all-zero entries to a row with d non-zero entries. For nodes whose local neighborhood has not changed, we assume that their initial temporal latent positions do not change either. For nodes whose neighborhood has changed, we initialize their new temporal latent positions based on their new neighbors' latent space positions. Next, we iteratively perform a conditioned latent position update for each affected node (i.e., a candidate set of nodes whose latent position might change) and an update for the affected node set until there are no more changes in the temporal latent position of each node. The entire process is outlined in Algorithm 3.

Algorithm 3. The Incremental BCGD Algorithm for Inferring Temporal Latent Space

Input: Graphs $\{G_1, \ldots, G_t\}$ and latent space dimension k
Output: Y_{t+1} and latent space $\{Z_1, \ldots, Z_t\}$
1: Nonnegative initial guess for Z_1
2: **for** each time stamp τ from 1 to t
3: **for** each u in graph G_τ
4: **if** $G_\tau(u)$ is not updated
5: $Z_\tau(u) = Z_{\tau-1}(u)$
6: **else**
7: Initialize $Z_\tau(u)$ using Eq. (7)
8: affected node set $S = \Delta V_\tau$
9: **repeat**
10: **for** each u in S
11: update $Z_\tau(u)$ by Eq. (6) and normalize it
12: update affected node set S with Algorithm 4
13: **until** Z_τ converges or S is empty.
14: **return** $Y_{t+1} = \Phi([Z_1, \ldots, Z_t])$ and $\{Z_1, \ldots, Z_t\}$

Initialize Updated Nodes. In our algorithm, the four update operations are handled simply by comparing whether $G_\tau(u)$ is identical to $G_{\tau-1}(u)$. For each updated node u, we initialize its latent position based on the probability of seeing its updated neighbors' latent positions. Specifically, for each node u and the dimension c of the latent space at time τ, the position of u in dimension c is computed using the following equation:

$$Z_\tau(u, c) = \frac{\sum_{v \in N(u)} G_\tau(u, v) Z_\tau(v, c)}{\sum_{q \in N(u)} G_\tau(u, q)}. \quad (7)$$

The initialization of latent position for an updated node u follows the notion of "latent homophily" introduced earlier: The latent position of the node u is as close as possible to those of its network neighbors.

Identifying Affected Nodes. Our dynamic update strategy can be viewed as an extra conditional update by which only nodes affected accept a new latent position. Unfortunately, the set of affected nodes for which the latent positions need to be updated is not limited to only the set of recently changed nodes and/or their network neighbors. Therefore, how could we identify the set of affected nodes?

The overall idea of our affected nodes identification is outlined as follows. Initially, the set of affected nodes is identical to the set of updated nodes (Line 8 in Algorithm 3). Next, after performing each iteration of a conditional update with Eq. (6) in Line 11 in Algorithm 3, some old affected nodes are no longer required to be updated since their latent positions have converged. On the other hand, the effects of the update could be further propagated from old affected nodes to their neighborhood. The details of our affected nodes update procedure are presented in Algorithm 4.

Algorithm 4. Updating Affected Nodes

Input: $Z_{\tau-1}$, Z_τ, $S^{\texttt{old}}$
Output: A set of affected nodes S
1: $S = S^{\texttt{old}}$
2: **for** each u in S^{old}
3: **if** $\forall c, |Z_\tau(u, c) - Z_{\tau-1}(u, c)| < \delta$
4: $S = S \setminus \{u\}$
5: **for** each $w \in N(u)$
6: **if** $|Z_\tau(u)Z_\tau(w)^T - Z_{\tau-1}(u)Z_{\tau-1}(w)^T| \geq \zeta$
7: $S = S \cup \{w\}$
8: **return** S

Complexity Analysis. The total computational cost of Algorithm 3 is $O(k \sum_\tau \sum_r |S_r + N(S_r)|)$, where r denotes the iteration number of the conditional update process outlined in Lines 9-13 in Algorithm 3, S_r denotes the set of affected nodes and $N(S_r)$ denotes the neighborhood of affected nodes. When $\tau = 1$, $|S_r|$ is equal to the number of nodes n, and $|N(S_r)|$ is equal to the number of edges in the first graph snapshot m_1. For the in-memory storage cost, compared with Algorithm 2, Algorithm 3 requires additional storage cost about ΔG_τ. Therefore, the in-memory storage cost of Algorithm 3 is $O(nk + \max_\tau(m_\tau + \Delta m_\tau))$.

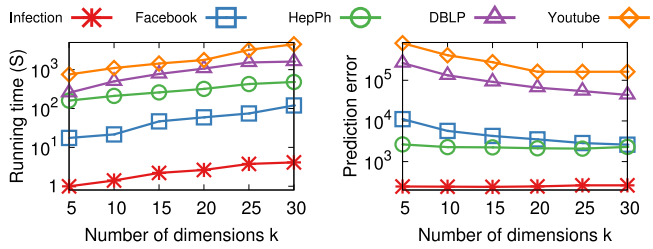
5 Experiments

5.1 Dataset and Evaluation

We use five real temporal datasets in our experiments, which are obtained from the Koblenz Large Network Collection [33]. The statistics of each data set are reported in

TABLE 2
Statistics of Data Sets, Where Volume Denotes the Total Number of Interactions, and the Number of Edges Denotes the Number of Distinct Interactions

Data	# nodes	volume	# edges	# snapshots
Infection [34]	410	17,298	2,765	8
Facebook [35]	63,731	817,035	183,412	5
HepPh [36]	28,093	4,596,803	3,148,447	9
DBLP [37]	1,314,050	18,986,618	10,724,828	11
YouTube [38]	3,223,589	9,375,374	9,375,374	7



(a) Inferring time (s) (b) Prediction errors

Fig. 4. Effect of dimensionality k to proposed approaches.

Table 2. Note that in our experimental setting, each graph snapshot G_τ consists of all the nodes and the interactions that appear within the time interval $[\tau - 1, \tau]$.

Evaluation Metrics. We evaluate the performance of the proposed approaches from three aspects: the effect of parameters, the efficiency of both offline latent space inference and online link prediction, and the accuracy of online link prediction. We use total running time and memory consumption to evaluate the efficiency of both offline latent space inference and online link prediction. We use prediction error to evaluate the inference accuracy. Given the training graph $G_1, \ldots, G_t$, prediction error is defined as $\frac{1}{t-1}\sum_{\tau=2}^{t}\|G_\tau - Z_{\tau-1}Z_{\tau-1}^T\|_F$. Therefore, a smaller prediction error indicates better inference accuracy.

For link prediction accuracy, we use Area Under Curves (both Receiver Operating Characteristic (ROC) and Precision-Recall (PR) curves), termed as $\texttt{AUC}_{\texttt{ROC}}$ and $\texttt{AUC}_{\texttt{PR}}$. The ROC curve is created by plotting the true positive rate versus false positive rate; while the PR curve is created by plotting precision versus recall at various threshold settings. Thus, AUC score evaluates the overall ranking yielded by the algorithms with a larger AUC indicating better link prediction performance. In the following we only report the experiment results based on $\texttt{AUC}_{\texttt{ROC}}$; the experiment results based on $\texttt{AUC}_{\texttt{PR}}$ are reported in the Supplement, available online.

Test Pair Generation. Given the training graph $G_1, \ldots, G_t$, we perform link prediction for the test graph G_{t+1}. We provide two different setups to generate test pairs: (1) *all links*: In this setup, we focus on how well the methods predict links in the test graph, no matter whether those links are repeated links or new links. Toward this goal, we randomly generate 100,000 linked and an equal number of non-linked pairs from G_{t+1} as test pairs. (2) *new links*: In this setup, we focus on how well the methods predict the emergence of new links and deletion of existing links. Again, we randomly generate 10,000 linked and an equal number of non-linked pairs from ΔG_{t+1}.

Comparable Approaches. We compare our approaches (standard BCGD $\texttt{BCGD}_G$, local BCGD $\texttt{BCGD}_L$ and incremental BCGD $\texttt{BCGD}_I$) with a set of state-of-the-art approaches, which are summarized in Table 3 and Section 6. For the offline latent space inference time comparison, we compare our approaches with global optimization approach DMMSB [39], local optimization approaches NMFR [40], Hott [28], BIGCLAM [22], PTM [5] and incremental optimization approach LabelRT [41]. We compare online link prediction times of our approaches to that of high-dimension latent space approaches BIGCLAM and LabelRT, and popular graph heuristic AA [42]. For the Hott, NMFR and DMMSB, the total running time for their approaches is the same as that of our BCGD approaches. This is because the online prediction time cost is proportional to the number of dimensions k, and k is set to 20 for all the low-dimension constraint latent space approaches including our BCGD approaches, Hott, NMFR, PTM and DMMSB.

Configurations. The results are reported as an average over 10 runs. If the maximum number of iterations is required as an input; we set it to 100. All experiments are conducted on a single machine with 8G memory and i7 2.7 GHZ CPU.

5.2 Effect of Parameters

This set of experiments aims to determine the impact of parameters on the performance of the proposed approaches and/or the comparable approaches.

The Effect of Temporal Smoothness. While fixing all the other parameters (e.g., $k = 20$), we first study the effect of

TABLE 3
A Summary of State-of-the-Art Approaches, Where *Time Series* Denotes Using a Sequence of Graph Snapshots as Inputs, *Aggregated* Denotes Using a Single Aggregated Static Graph Snapshot as an Input, *Low* Denotes Low Dimensional Latent Space, and *High* Denotes without Dimensionality Constraint

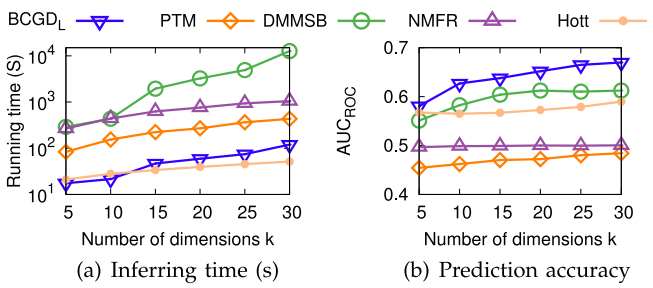
	Latent space dimension			Input graph		Scalability		Accuracy	
	low	high	NA	aggregated	time-series	inference	prediction	all links	new links
PTM [5], NMFR [40]	√			√			√	√	
Hott [28]	√			√		√	√	√	
LabelRT [41]		√		√	√				√
BIGCLAM [22]		√		√		√			
AA [42]			√	√		NA		√	
$\texttt{BCGD}_G$, DMMSB [39]	√			√	√		√	√	√
$\texttt{BCGD}_L$, $\texttt{BCGD}_I$	√				√	√	√	√	√

TABLE 4
Effect of the Smoothing Parameter λ

Prediction error by $\texttt{BCGD}_G$			
λ	0	$[0.0001, 1]$	10
Infection	228	225 ± 1.26	288
Facebook	3,436	3,312±16	3,410
Hepph	2,043	1,697±7.9	1,769
DBLP	80,566	65,878±429	65,926
Youtube	195,023	161,239±726	162,282
Prediction error by $\texttt{BCGD}_L$			
λ	0	$[0.0001, 1]$	10
Infection	250	243±3.5	249
Facebook	3,659	3,485±36	3,525
Hepph	2,566	2,106±51	2,172
DBLP	81,633	65,957±18	66,471
Youtube	198,646	162,287±357	163,675

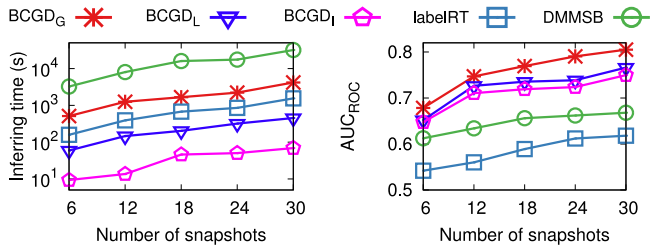
the smoothing parameter λ. The effect of parameter λ might be related to the inference algorithms, thus we evaluate the performance of both global BCGD and local BCGD when varying λ. We do not plot the running time since we notice that the running time is insensitive to the parameter λ. We vary λ from 0.0001 to 10 with logarithmic scale and report the prediction errors shown in Table 4. We also report the prediction errors for $\lambda = 0$. Note that when λ is 0, the update rules stated in Eqs. (4) and (6) become identical. $\texttt{BCGD}_G$ differs from $\texttt{BCGD}_L$ since they have different ways to proceed with initialization and convergence, see Lines 2-7 in Algorithms 1 and 2. Clearly, either absence of temporal smoothness ($\lambda = 0$) or strong temporal smoothness ($\lambda = 10$) leads to higher prediction errors. While λ varies from 0.0001 to 1, there are no significant differences in terms of prediction errors. Therefore, in the following experiments, we simply fix λ as 0.0001 for $\texttt{BCGD}_G$ and 0.01 for $\texttt{BCGD}_L$.

The Effect of Dimensionality. Next, we fix all the other parameters and analyze the effect of k. Since the effect of number of dimensions k is lightly correlated with inference algorithms, here we choose the representative local BCGD algorithm. We vary k from 5 to 30, and report both the running time of temporal latent space inference and prediction error in Fig. 5. The overall trends indicate that the running time increases with number of dimensions k, while prediction error decreases with number of dimensions k. However, increasing number of dimensions does not necessarily lead to the decrease of prediction error. For example, for a Facebook dataset, when the number of dimensions k is increased from 25 to 30, the prediction error is increased. In



(a) Inferring time (s) (b) Prediction accuracy

Fig. 5. Effect of dimensionality k to other approaches on Facebook dataset.



(a) Inferring time (s) (b) Prediction accuracy

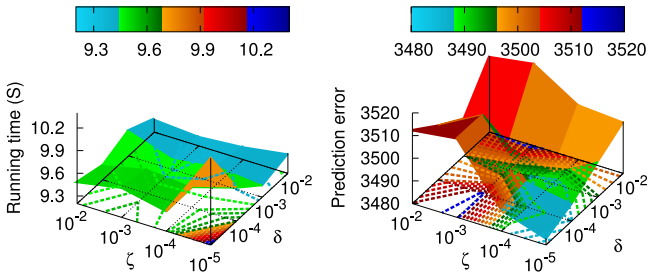
Fig. 6. Effect of number of snapshots to proposed and other approaches on Facebook dataset.

order to balance the efficiency and effectiveness, we opt to fix $k = 20$ for the proposed approach in all of the following experiments. A nonparametric approach that automatically selects the best dimension k will be an interesting future direction.

In order to perform a fair comparison, we also examine the effect of k to other comparable approaches including DMMSB, PTM, NMFR, and Hott. LabelRT and BIGCLAM are not included because their best dimension k is automatically computed. Furthermore, since several baselines DMMSB, PTM, NMFR are not scalable and can not support running on the two large datasets DBLP and YouTube, we simply choose the relatively largest dataset, Facebook, on which all of the approaches are able to be tested. The comparison of running time and $\texttt{AUC}_{\texttt{ROC}}$ on new links is reported in Figs. 5a and 5b respectively. Clearly, the overall trends of other approaches are very similar to those of $\texttt{BCGD}_L$: The inference time is growing linearly or even quadratically with the dimensionality k, while the prediction accuracy also improves with k. Therefore, we use the same value of $k = 20$ for other comparable approaches as well.

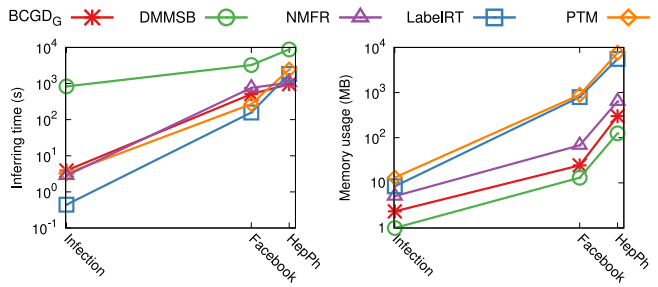
The Effect of Time Intervals. In order to understand the proposed model more deeply, we also study how the value of time interval τ impacts prediction performance. Specifically, we choose the Facebook dataset, vary the value of time interval τ, and create five different sequences of graph snapshots. As reported in Figs. 6a and. 6b, both the inference time and prediction accuracy increase with number of snapshots. The inference time increases because a larger number of snapshots mean more frequent model updates per shorter time interval. Meanwhile, shorter time interval leads to less change of graphs, which indicates that it is more reliable to forecast links at time $t + 1$ based on the current observation at time t. Therefore, the prediction accuracy also improves with number of snapshots. Again, the overall trends hold for both proposed and other approaches. Therefore, we simply fix the number of snapshots as reported in Table 2.

The Effect of Lazy Update. Finally, we evaluate the effect of two parameters ζ and δ that are related to the incremental BCGD algorithm. Since parameters ζ and δ are correlated with each other, we vary both of them and report the total inference time and prediction error on the Facebook dataset in Fig. 7. The results verify that when setting both ζ and δ to smaller values, the predictive accuracy is improved, while the total running time is also increased. However, Fig. 7a shows that when we reduce ζ and δ from 10^{-2} to 10^{-4}, the running time almost remains the same. The results verified that we can safely set ζ and δ to relatively small values without losing too much efficiency. We repeated this set of



(a) Inferring time (s) (b) Prediction errors

Fig. 7. Effect of parameters ζ and δ on Facebook dataset (best viewed in color).



(a) Inferring time (s) (b) Memory usage (MB)

Fig. 9. Inference efficiency comparison with DMMSB, PTM, LabelRT, and NMFR.

experiments on all the other datasets. Then, using curve fitting to capture how the best value of ζ and δ can be approximated by parameters such as n, m, T, and k, we noticed that a good value for ζ and δ can be achieved around $\sqrt{-\log(1-1/n)}$ and $2\zeta/k$. Therefore, in the following experiments, without specification, we set ζ as $\sqrt{-\log(1-1/n)}$ and δ as $2\zeta/k$.

5.3 Evaluation of Efficiency

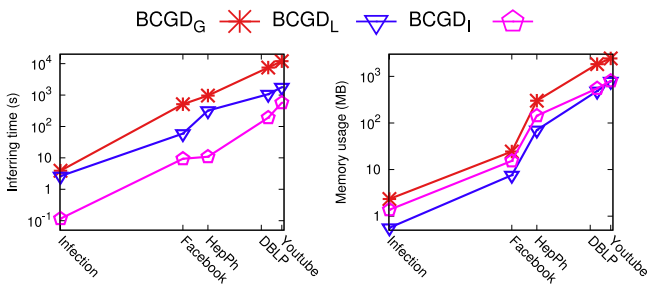
In this section we first compare the proposed global BCGD BCGD$_G$, local BCGD BCGD$_L$, and incremental BCGD BCGD$_I$ with other latent space inferring approaches in terms of inference time and memory consumption. We then subsequently compare BCGD approaches with two auto-dimensionality latent space approaches BIGCLAIM and LabelRT and graph heuristic AA in terms of online link prediction time and memory consumption.

Offline Inference Efficiency. We first report the total running time and memory consumption of our BCGD approaches in Fig. 8. Clearly, BCGD$_G$ is neither memory efficient nor time efficient. Both BCGD$_I$ and BCGD$_L$ are very time and memory efficient: BCGD$_I$ is more efficient in running time, while BCGD$_L$ is more efficient in memory usage. This is because BCGD$_I$ requires additional storage for graph changes ΔG; but it adaptively updates latent space with the graph changes and thus is very time efficient.

In the following we group other baselines into two classes to facilitate results visualization. We first compare those memory- or time-inefficient approaches DMMSB, PTM, LabelRT and NMFR with our BCGD$_G$ and report the results in Fig. 9. Among them, PTM and LabelRT require much larger memory consumption and thus fail to process the two large graph DBLP and YouTube due to their memory bottlenecks. NMFR is more memory efficient than PTM and LabelRT, but it still can not handle the two large graphs. In addition, global optimization algorithm DMMSB takes too long to process the two large graph DBLP and Youtube due to running time bottleneck. Fig. 9a shows that on Infection and Facebook data, LabelRT $\gg_t$ PTM $\gg_t$ {NMFR, BCGD$_G$} $\gg_t$ DMMSB, where $A \gg_t B$ denotes that A is faster than B on average. On HepPh data, {NMFR, BCGD$_G$} $\gg_t$ {LabelRT, PTM} $\gg_t$ DMMSB. Clearly, our global optimization algorithm BCGD$_G$ is at least 5 times faster than the other global approach, DMMSB, and is comparable to inefficient local approach NMFR. For small graphs, it is not surprising that the most efficient algorithm is the incremental approach LabelRT since it incrementally maintains and updates latent spaces. Unfortunately, it consumes large amounts of memory, and becomes much slower than our approach BCGD$_G$ for graph HepPh. In addition, all of these approaches are much slower than our fast BCGD algorithms BCGD$_L$ and BCGD$_I$, each of which takes less than 400 seconds to finish learning temporal latent space for HepPh.

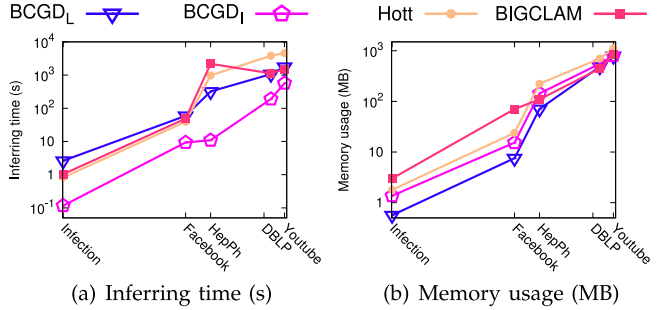
We further compare our fast BCGD algorithms with state-of-the-arts scalable approaches Hott and BigCLAM. BCGD$_L$ is comparable to Hott and BigCLAM. In addition, the incremental approach BCGD$_I$ is consistently more efficient than alternative approaches Hott and BigCLAM. Because we noticed from Fig. 10b that the memory usage of these four approaches is similar, we conclude that our two fast BCGD algorithms are both memory- and time-efficient.

Online Prediction Efficiency. We now verify that the low-dimension latent space approach is very efficient for online link prediction. Here we choose AA as a representative graph heuristic approach for link predication, not only because it achieves good accuracy [25], but also because it is fast to compute compared to other graph heuristics [43]. However, AA is still much slower than latent space-based



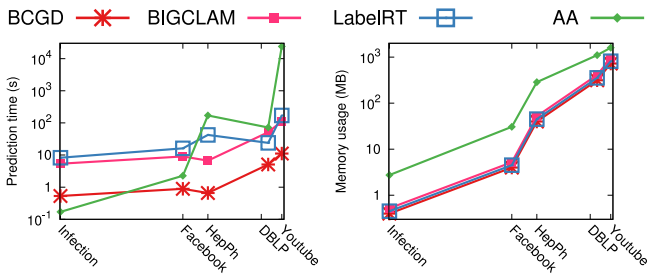
(a) Inferring time (s) (b) Memory usage (MB)

Fig. 8. Inference efficiency of BCGD algorithms.



(a) Inferring time (s) (b) Memory usage (MB)

Fig. 10. Inference efficiency comparison with scalable baselines Hott and BIGCLAM.



(a) Prediction time (s) (b) Memory usage (MB)

Fig. 11. Online prediction time and memory comparison for 20,000 node pairs. Note that BCGD represents all of the three proposed algorithms ($\texttt{BCGD}_G$, $\texttt{BCGD}_L$, and $\texttt{BCGD}_I$) since the online prediction time and memory consumption of these three are the same.

approaches in online link probability computation. As shown in Fig. 11, on the HepPh dataset, for 20,000 node pairs, it takes more than 150 seconds to compute the AA scores; while all the low-dimension latent space-based approaches are able to finish online link probability computation in less than one second. In addition, latent space-based link prediction approaches are also more efficient than AA in terms of memory consumption, see Fig. 11b. To predict whether two nodes are linked, latent space-based approaches only need to read the latent positions of two nodes; while AA requires the neighborhood information of two nodes. Finally, Fig. 11a also intuitively supports the reason why we use low-dimension constraint. With low-dimension constraint ($k = 20$), our BCGD approach obtains at least four times speed-up than auto-dimensionality approaches LabelRT and BigCLAM (sometimes k can be more than two hundred) in online link prediction.

5.4 Evaluation of Link Prediction Accuracy

In this section we quantitatively evaluate the quality of learned latent spaces in terms of their predictive power. We give an overall comparison of all the approaches in terms of AUC_{ROC} on predicting all links and report the results in Fig. 12. Here, $G_{\texttt{pre}}$ denotes the simple baseline using G_t to predict G_{t+1}. Clearly, for all link prediction accuracy, we have {$\texttt{BCGD}_G$, $\texttt{BCGD}_L$, $\texttt{BCGD}_I$, PTM, DMMSB} $\gg_a$ {Hott, NMFR} $\gg_a$ {BIGCLAM, LabelRT, AA, $G_{\texttt{pre}}$}, where $A \gg_a B$ denotes that on average, A is more accurate than B in terms of predictive power. BigCLAM and LabelRT have poor AUC scores because they only output a single, hard assignment of nodes to each dimension, which provides no means to control the trade-off between true and false positive rates. AA did not perform well on Facebook and YouTube because it can not capture the deleted links prediction; AA still gives each unlinked node pairs a high score based on their topological similarity on aggregated graph from G_1 to G_t. Our methods perform much better than G_{pre} due to: 1) Z_t encodes not only the link information of G_t, but also the temporal information from G_1 to G_t; 2) Z_t is a good estimate for Z_{t+1}; 3) the reconstruction error between $Z_t Z_t^T$ and G_t reflects the possible future changes. Given any pair (u, v), if the reconstruction error between $Z_t(u)Z_t(v)^T$ and $G_t(u, v)$ is large, then it indicates that the proximity of the pair (u, v) is changing over time.

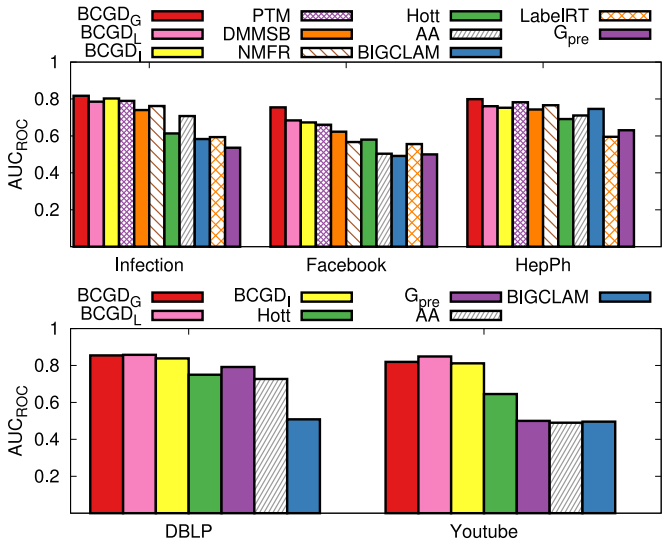


Fig. 12. All links prediction accuracy comparison.

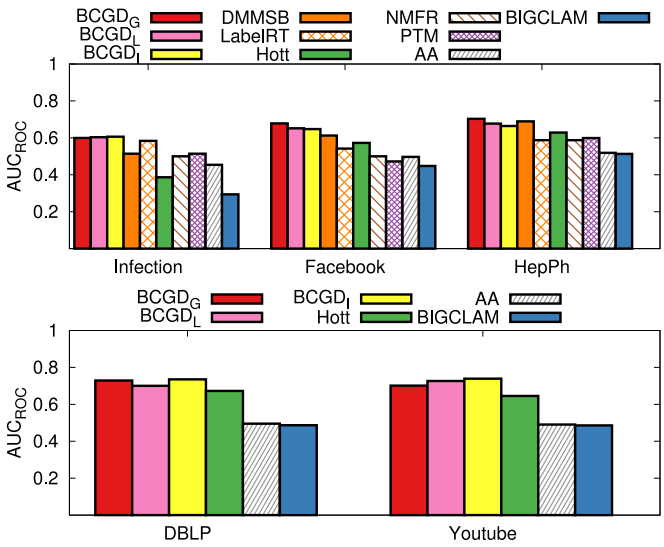


Fig. 13. New links prediction accuracy comparison.

We next evaluate the link prediction performance of all the approaches on new links. The results are plotted in Fig. 13. As expected, the AUC values are much lower for the new links. For the new links prediction, we have {$\texttt{BCGD}_G$, $\texttt{BCGD}_L$, $\texttt{BCGD}_I$, LabelRT, DMMSB} $\gg_a$ {Hott, NMFR} $\gg_a$ {AA, BIGCLAM}. It is not surprising that all the temporal link prediction approaches perform better than static link prediction approaches.

Fig. 13 also reflects an interesting phenomenon: our fast BCGD algorithms $\texttt{BCGD}_L$ and $\texttt{BCGD}_I$ not only improve the efficiency bottleneck of latent space inference, but also achieve comparable prediction accuracy with the global optimization approach $\texttt{BCGD}_G$. For the new links prediction, they even outperform global approaches on both dataset Infection and YouTube. This is because both $\texttt{BCGD}_L$ and $\texttt{BCGD}_I$ utilize local structures and local invariance, which significantly enhances local and temporal information encoding in the temporal latent space. In addition, we notice that on average, the incremental approach $\texttt{BCGD}_I$ is able to obtain higher accuracy than the local approach $\texttt{BCGD}_L$. Although both $\texttt{BCGD}_L$ and $\texttt{BCGD}_I$ utilize local invariance, $\texttt{BCGD}_I$ is able to exploit the local structure and local invariance in an adaptive way with graph changes, which leads to

a better performance. Because $\texttt{BCGD}_I$ is more scalable than all the other alternative approaches (as shown in Fig. 10a). We conclude that $\texttt{BCGD}_I$ is a practical solution for link prediction on large-scale dynamic networks.

6 Related Work

Recently, link prediction has attracted significant attention, and various methods have been proposed for different types of networks. In the following, we first give a brief overview of related work in link prediction. Particularly, we focus on two categories: graph-based heuristics and latent space-based approaches. Next, we present some related work in inferring latent positions of nodes in a latent space.

6.1 Link Prediction

The link prediction problem in static networks has been extensively studied. We refer readers to two recent survey papers [1], [2] and the related work section of the paper [44] for an exhaustive introduction to this thriving field. Among these existing works, one of the most popular categories of methods is graph-based heuristics. Graph-based heuristics [2], [42], [45], [46], [47] model the link probability between two nodes as a function of their topological similarity. The topological similarity can be defined based on local proximity such as common neighbors [42], triad closures [45], or global proximity such as weighted shortest-paths [46]. Here we highlight Adamic and Adar (*AA*) [42] who proposed a degree-weighted common-neighbor heuristic that works well in collaboration networks.

Unfortunately, in online link prediction, the computation of topological similarity metrics, especially for those metrics that require path computation or random walk over the entire network, is time-consuming if computed on-the-fly [43]. On the other hand, one can precompute all pairs of topological similarity scores in advance, leading to a quadratic storage cost. To strike a balance between online computation cost and offline storage cost, recent research interests in link prediction have been directed towards temporal link prediction with latent space modeling [4]. For example, Fu et al. [39] extended the mixed membership block model to allow a linear Gaussian trend in the model parameters (*DMMSB*). Sewell and Chen [48] proposed a model which embeds longitudinal network data as trajectories in a latent euclidean space, and the probability of link formation between two nodes depends on their euclidean distance and popularity. In this work we do not take the popularity of each node into consideration since we assume that the popularity of each node is automatically captured by the latent space modeling: a popular node is surrounded by many other nodes in the latent space. Furthermore, our model penalizes sudden and large changes in the latent positions.

Dunlavy et al. [17] developed a tensor-based latent space modeling to predict temporal links. Unfortunately, in online prediction, it requires the tensor product of vectors, which is much more computationally expensive than the dot product of vectors in the proposed approach. In addition, their approaches require very large storage costs since they need to put the entire tensor into memory. Recently, there have been several more efficient approaches to conduct tensor decomposition. For instance, Huang et al. [49] proposed to conduct tensor decomposition with singular value decomposition to find temporal communities in a GPGPU setting.

6.2 Inferring Latent Space

Recent work has explored static latent space modeling for social network analysis. For example, Hoff et al. [4] developed a class of models where the probability of a relation between actors depends on the positions of individuals in an unobserved social space. They make inference for the latent space with the maximum likelihood estimation and Bayesian posterior inference. Obtaining this posterior is, however, often infeasible in large-scale graphs. Yin et al. [5] proposed an efficient stochastic variational inference algorithm and a parsimonious triangular model to infer the latent spaces of large networks (*PTM*). Sarkar and Moore [6] first generalized the classical multidimensional scaling to get an initial estimate of the positions in the latent space, and then applied nonlinear optimization directly to obtain the latent space representation.

Matrix factorization approaches are also applied to observed networks to learn the low-dimension latent space representation. Yang et al. [40] propose a multiplicative algorithm with graph random walk to solve the symmetric graph factorization problem (*NMFR*). However, their approach does not scale well due to the high computation cost in each iteration. Additionally, the effectiveness of their approach decreases as the density of the input graph increases. Yang and Leskovec [22] proposed a matrix factorization approach over networks to learn the static latent space representation by maximizing the link likelihood (*BIGCLAM*). Their approach is very scalable; nevertheless, the learned latent space is of high dimension, which leads to expensive online link prediction computation. HottTopixx (*Hott* [28]) uses a new approach for NMF with low-rank constraint which is highly parallel and allows it to run on very large datasets. Unfortunately, their approach is designed to factorize the traditional rectangle matrices, and scalable approaches in symmetric graph factorization are much less studied than rectangle matrix factorization such as Hott. In this work we apply symmetric matrix factorization approaches directly on the observed networks to infer low-rank latent spaces. We propose a block coordinate gradient descent algorithm, which is more scalable than NMFR and performs well, regardless of the topology of input graphs.

In addition to matrix factorization approaches, recently Xie et al. [41] (LabelRT) proposed a label propagation-based approach that incrementally detects communities in dynamic networks. In this work we propose an incremental BCGD algorithm to incrementally detect latent positions at each time step with conditional updates on a set of affected nodes.

7 Conclusion and Future Work

In this work we propose a scalable approach for link prediction with a temporal latent space model, which assumes two nodes are more likely to be linked if they are close to each other in their latent space. In addition, our temporal latent space model prefers smoothly evolving by penalizing

frequent changes in latent positions for each node. With respect to the proposed model, we developed three different inference algorithms, $\texttt{BCGD}_G$, $\texttt{BCGD}_L$ and $\texttt{BCGD}_I$ to learn the temporal latent space via non-negative matrix factorization approaches. We also provide a set of theoretical analyses characterizing their performance guarantees. We conducted a set of experiments on large networks with millions of nodes and links to verify both the efficiency and predictive quality of all the proposed approaches.

Our model still has limitations. First, the temporal smoothness assumption may not hold in some circumstance. For example, in a road network, the latent position of each node changes significantly from rush hour to a non-rush hour. And even in social networks, where temporal smoothness assumptions typically hold, external events may cause significant shifts to the network that reflect abrupt changes in latent node positions. We plan to extend the proposed model to support temporal nonlinear transitions. Second, we plan to propose a new continuous time model that supports continuous inputs rather than discretized graph snapshots. Moreover, we plan to extend our approach to generalized cases including directed and weighted graphs. In addition to model improvement, another interesting direction is to further improve the efficiency. Note that in our block coordinate gradient descent approach, the latent position update for a node u can be conducted simultaneously with another node v if they do not share the same neighborhood. This property can be leveraged for very good parallelization in the future.

Acknowledgments

The authors are very grateful to Prof. Dacheng Tao, Prof. Kun Zhang, Prof. Rong Ge, Li Han, and Dingxiong Deng for their insightful discussions. This research was supported in part by DARPA grant No. W911NF1210034.

References

[1] M. A. Hasan and M. J. Zaki, "A survey of link prediction in social networks," in *Social Network Data Analytics*. New York, NY, USA: Springer, 2011, pp. 243–275.

[2] D. Liben-Nowell andJ. Kleinberg, "The link prediction problem for social networks," in *Proc. 12th Int. Conf. Inf. Knowl. Manage.*, 2003, pp. 556–559.

[3] M. Mcpherson, L. Smith-Lovin, and J. M. Cook, "Birds of a feather: Homophily in social networks," *Annu. Rev. Sociology*, vol. 27, pp. 415–444, 2001.

[4] P. D. Hoff, A. E. Raftery, and M. S. Handcock, "Latent space approaches to social network analysis," *J. Amer. Statistical Assoc.*, vol. 97, pp. 1090–1098, 2002.

[5] J. Yin, Q. Ho, and E. P. Xing, "A scalable approach to probabilistic latent space inference of large-scale networks," in *Proc. Adv. Neural Inf. Process. Syst. Conf.*, 2013, pp. 422–430.

[6] P. Sarkar and A. W. Moore, "Dynamic social network analysis using latent space models," *ACM SIGKDD Explorations Newslett.*, vol. 7, no. 2, pp. 31–40, 2005.

[7] A. K. Menon and C. Elkan, "Link prediction via matrix factorization," in *Proc. Eur. Conf. Mach. Learn. Knowl. Discovery Databases*, 2011, pp. 437–452.

[8] G.-J. Qi, C. C. Aggarwal, and T. Huang, "Link prediction across networks by biased cross-network sampling," in *Proc. 29th Int. Conf. Data Eng.*, 2013, pp. 793–804.

[9] S. Gao, L. Denoyer, and P. Gallinari, "Temporal link prediction by integrating content and structure information," in *Proc. 20th ACM Int. Conf. Inf. Knowl. Manage.*, 2011, pp. 1169–1174.

[10] J. Ye, H. Cheng, Z. Zhu, and M. Chen, "Predicting positive and negative links in signed social networks by transfer learning," in *Proc. 22nd Int. Conf. World Wide Web*, 2013, pp. 1477–1488.

[11] Y. Zhang, M. Zhang, Y. Liu, S. Ma, and S. Feng, "Localized matrix factorization for recommendation based on matrix block diagonal forms," in *Proc. 22nd Int. Conf. World Wide Web*, 2013, pp. 1511–1520.

[12] D. Erdős, R. Gemulla, and E. Terzi, "Reconstructing graphs from neighborhood data," *ACM Trans. Knowl. Discovery Data*, vol. 8, no. 4, pp. 23:1–23:22, Oct. 2014.

[13] E. M. Airoldi, D. M. Blei, S. E. Fienberg, and E. P. Xing, "Mixed membership stochastic blockmodels," *J. Mach. Learn. Res.*, vol. 9, pp. 1981–2014, 2008.

[14] P. Gupta, et al., "Real-time twitter recommendation: Online motif detection in large dynamic graphs," *Proc. VLDB Endowment*, vol. 7, no. 13, pp. 1379–1380, 2014.

[15] C. Tantipathananandh, T. Berger-Wolf, and D. Kempe, "A framework for community identification in dynamic social networks," in *Proc. 13th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2007, pp. 717–726.

[16] J. Sun, C. Faloutsos, S. Papadimitriou, and P. S. Yu, "Graphscope: Parameter-free mining of large time-evolving graphs," in *Proc. 13th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2007, pp. 687–696.

[17] D. M. Dunlavy, T. G. Kolda, and E. Acar, "Temporal link prediction using matrix and tensor factorizations," *ACM Trans. Knowl. Discovery Data*, vol. 5, no. 2, pp. 10:1–10:27, 2011.

[18] L. Zhu, A. Galstyan, J. Cheng, and K. Lerman, "Tripartite graph clustering for dynamic sentiment analysis on social media," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2014, pp. 1531–1542.

[19] J. Zhang, C. Wang, J. Wang, and J. X. Yu, "Inferring continuous dynamic social influence and personal preference for temporal behavior prediction," *Proc. VLDB Endowment*, vol. 8, no. 3, pp. 269–280, 2014.

[20] M. W. Berry, M. Browne, A. N. Langville, V. P. Pauca, and R. J. Plemmons, "Algorithms and applications for approximate nonnegative matrix factorization," *Comput. Statist. Data Anal.*, vol. 52, no. 1, pp. 155–173, 2007.

[21] P. Tseng and S. Yun, "A coordinate gradient descent method for nonsmooth separable minimization," *Math. Program.*, vol. 117, no. 1/2, pp. 387–423, 2009.

[22] J. Yang and J. Leskovec, "Overlapping community detection at scale: A nonnegative matrix factorization approach," in *Proc. 6th ACM Int. Conf. Web Search Data Mining*, 2013, pp. 587–596.

[23] C. Tantipathananandh and T. Y. Berger-Wolf, "Finding communities in dynamic social networks," in *Proc. 11th IEEE Int. Conf. Data Mining*, 2011, pp. 1236–1241.

[24] D. D. Lee and H. S. Seung, "Learning the parts of objects by nonnegative matrix factorization," *Nature*, vol. 401, pp. 788–791, 1999.

[25] P. Sarkar, D. Chakrabarti, and M. I. Jordan, "Nonparametric link prediction in dynamic networks," in *Proc. 29th Int. Conf. Mach. Learn.*, 2012, pp. 1687–1694.

[26] S. A. Vavasis, "On the complexity of nonnegative matrix factorization," *J. Optimization*, vol. 20, no. 3, pp. 1364–1377, 2009.

[27] S. Arora, R. Ge, R. Kannan, and A. Moitra, "Computing a nonnegative matrix factorization—provably," in *Proc. 44th Annu. ACM Symp. Theory Comput.*, 2012, pp. 145–162.

[28] B. Recht, C. Re, J. A. Tropp, and V. Bittorf, "Factoring nonnegative matrices with linear programs," in *Proc. Adv. Neural Inf. Process. Syst. Conf.*, 2012, pp. 1223–1231.

[29] Y. Nesterov, *Introductory Lectures on Convex Optimization : A Basic Course*. Berlin, Germany: Kluwer, 2004.

[30] N. Guan, D. Tao, Z. Luo, and B. Yuan, "NeNMF: An optimal gradient method for nonnegative matrix factorization," *IEEE Trans. Signal Process.*, vol. 60, no. 6, pp. 2882–2898, Jun. 2012.

[31] A. Beck and M. Teboulle, "A fast iterative shrinkage-thresholding algorithm for linear inverse problems," *SIAM J. Imaging Sci.*, vol. 2, no. 1, pp. 183–202, 2009.

[32] D. Gibson, R. Kumar, and A. Tomkins, "Discovering large dense subgraphs in massive graphs," in *Proc. 31st Int. Conf. Very Large Data Bases*, 2005, pp. 721–732.

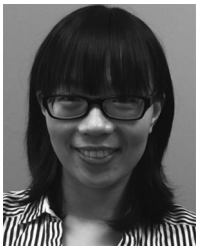
[33] J. Kunegis, "KONECT—The Koblenz network collection," in *Proc. 22nd Int. Conf. World Wide Web*, 2013, pp. 1343–1350.

[34] L. Isella, J. Stehlé, A. Barrat, C. Cattuto, J.-F. Pinton, and W. V. den Broeck, "What's in a crowd? analysis of face-to-face behavioral networks," *J. Theoretical Biol.*, vol. 271, no. 1, pp. 166–180, 2011.

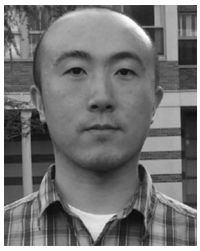
[35] B. Viswanath, A. Mislove, M. Cha, and K. P. Gummadi, "On the evolution of user interaction in Facebook," in *Proc. 2nd ACM Workshop Online Social Netw.*, 2009, pp. 37–42.

[36] J. Leskovec, J. Kleinberg, and C. Faloutsos, "Graph evolution: Densification and shrinking diameters," *ACM Trans. Knowl. Discovery Data*, vol. 1, no. 1, pp. 1–40, 2007.

[37] "DBLP network dataset—KONECT," May 2015. [Online]. Available: http://konect.uni-koblenz.de/networks/dblp_coauthor
[38] A. Mislove, "Online social networks: Measurement, analysis, and applications to distributed information systems," Ph.D. dissertation, Department of Computer Science, Rice University, Houston, TX, 2009.
[39] W. Fu, L. Song, and E. P. Xing, "Dynamic mixed membership blockmodel for evolving networks," in *Proc. 26th Annu. Int. Conf. Mach. Learn.*, 2009, pp. 329–336.
[40] Z. Yang, T. Hao, O. Dikmen, X. Chen, and E. Oja, "Clustering by nonnegative matrix factorization using graph random walk," in *Proc. Adv. Neural Inf. Process. Syst.*, 2012, pp. 1088–1096.
[41] J. Xie, M. Chen, and B. K. Szymanski, "LabelRankT: Incremental community detection in dynamic networks via label propagation," *CoRR*, vol. abs/1305.2006, 2013.
[42] L. A. Adamic and E. Adar, "Friends and neighbors on the Web," *Social Netw.*, vol. 25, pp. 211–230, 2001.
[43] S. Cohen and N. Cohen-Tzemach, "Implementing link-prediction for social networks in a database system," in *Proc. ACM SIGMOD Workshop Databases Social Netw.*, 2013, pp. 37–42.
[44] L. Zhu and K. Lerman, "A visibility-based model for link prediction in social media," in *Proc. ASE/IEEE Conf. Social Comput.*, 2014.
[45] P. Symeonidis, E. Tiakas, and Y. Manolopoulos, "Transitive node similarity for link prediction in social networks with positive and negative links," in *Proc. 4th ACM Conf. Recommender Syst.*, 2010, pp. 183–190.
[46] L. Katz, "A new status index derived from sociometric analysis," *Psychometrika*, vol. P-18, no. 1, pp. 39–43, Mar. 1953.
[47] G. Jeh and J. Widom, "SimRank: A measure of structural-context similarity," in *Proc. 8th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2002, pp. 538–543.
[48] D. K. Sewell and Y. Chen, "Latent space models for dynamic networks," *J. Amer. Statistical Assoc.*, vol. 110, no. 512, pp. 1646–1657, 2015.
[49] F. Huang, U. N. Niranjan, M. U. Hakeem, P. Verma, and A. Anandkumar, "Fast detection of overlapping communities via online tensor methods on GPUs," *CoRR*, vol. abs/1309.0787, 2013.



Linhong Zhu received the BS degree from the University of Science and Technology of China, in 2006, and the PhD degree in computer engineering from Nanyang Technological University, Singapore, in 2011. She is a computer scientist with USC's Information Sciences Institute. Prior to that, she worked as a scientist-I in the Data Analytics Department, Institute for Infocomm Research, Singapore. Her research interests include large-scale graph analytics with applications to social network analysis, social media analysis, and predictive modeling. She is a member of the IEEE and the ACM.



Dong Guo received the BS degree in physics from the University of Science and Technology of China, Hefei, China, in 2007, and entered the PhD program in computer science at the University of Southern California in 2011. His doctoral projects consisted of studying the data-driven representation learning algorithms and applying machine pipeline in industrial data analysis.

Junming Yin received the PhD degree in computer science from UC Berkeley in 2011. He is an assistant professor with the Department of Management Information Systems, University of Arizona. Prior to that, he was a lane fellow in the Lane Center of Computational Biology, Carnegie Mellon University. His research focus is on statistical machine learning and its applications in business intelligence, digital marketing, healthcare systems, and computational biology.

Greg Ver Steeg received the PhD degree in physics from Caltech, in 2009, and since then he has focused on using ideas from information theory to understand complex systems like human behavior, biology, and language. He is a research assistant professor in computer science at USC's Information Sciences Institute. His work has been recognized with an AFOSR Young Investigator Award.

Aram Galstyan is a research director for data science and machine learning at USC's Information Sciences Institute and a research associate professor in the USC Viterbi School of Engineering's Computer Science Department. His current research focuses on different aspects of machine learning, with applications in computational social sciences, forecasting, and bioinformatics.