Time-Capturing Dynamic Graph Embedding for Temporal Linkage Evolution

Yu Yang, Jiannong Cao, *Fellow, IEEE*, Milos Stojmenovic, Senzhang Wang, *Member, IEEE*, Yiran Cheng, Chun Lum, and Zhetao Li, *Member, IEEE*

Abstract—Dynamic graph embedding learns representation vectors for vertices and edges in a graph that evolves over time. We aim to capture and embed the evolution of vertices' temporal connectivity. Existing work studies the vertices' dynamic connection changes but neglects the time it takes for edges to evolve, failing to embed temporal linkage information into the evolution of the graph. To capture vertices' temporal linkage evolution, we model dynamic graphs as a sequence of snapshot graphs, appending the respective timespans of edges (ToE). We co-train a linear regressor to embed ToE while inferring a common latent space for all snapshot graphs by a matrix-factorization-based model to embed vertices' dynamic connection changes. Vertices' temporal linkage evolution is captured as their moving trajectories within the common latent representation space. Our embedding algorithm converges quickly with our proposed training methods, which is very time efficient and scalable. Extensive evaluations on several datasets show that our model can achieve significant performance improvements, i.e., 22.98 percent on average across all datasets, over the state-of-the-art baselines in the tasks of vertex classification, static and time-aware link prediction, and ToE prediction.

Index Terms—Dynamic graph embedding, graph evolution, edge timespan, graph mining

1 Introduction

Graphs are one of the most widely used data representations which naturally exist in the real world in the form of social networks, biological networks, information diffusion networks, road networks etc. Static graphs represent immutable connections among vertices; however, many real world applications of graphs are dynamic and evolve over time. Vertices could join quickly or slowly, leave at their own pace, and even re-join the graph, thereby making their connections dynamically malleable over time. Efficiently extracting meaningful knowledge from the evolution of vertex connections in dynamic graphs is an open research problem in graph mining.

Dynamic graph embedding draws from and builds upon the great success of graph representation learning, also referred to as graph embedding or network embedding [1]. Dynamic graph embedding captures and encodes the evolution of vertex properties and connections as low dimensional representation vectors in order to benefit downstream machine learning applications. Existing works model the dynamic graph as either a sequence of static snapshot graphs [2], [3], [4], [5], [6], [7], [8], [9], [10] or neighborhood formation sequence sampled from the temporal random walk [11], [12], [13]. These approaches merely capture the sequential changes of static graph structure throughout the snapshot graph sequence as well as the sequential linkage evolution among vertices for embedding. However, the time it takes for vertex connections to evolve is also dynamic and it is neglected by the above approaches. Here, we tackle the problem of embedding the temporal linkage evolution of vertices in a dynamic graph, while simultaneously preserving their dynamic connection changes and timespans of edge formation (ToE).

ToE preserves important duration of edge formation information as well as the temporal dependencies of vertices while the dynamic graphs evolve. For example, in a dynamic transaction network, buyers could appear at any time to trade with sellers and disappear afterward, thereby forming an edge. The ToE in this case represents how long the buyer takes to complete the transaction after the seller posts a sell order, which carries important trading behavior and may be used to form trading strategies. Cautious traders may prefer to spend a significant amount of time looking for the best price of an item. Thus, the edges they construct may have a relatively long ToE. Other traders may complete a transaction as soon as goods appear on the market, therefore resulting in a significantly shorter ToE. It is possible for buyers to complete the transaction with one of multiple sell orders posted by the same seller at different times, in which the ToE serves as discriminative information. Should ToE be neglected and merely reduced to the dynamic connectivity

- *Yu Yang, Jiannong Cao, Yiran Cheng, and Chun Lum are with the Department of Computing, Hong Kong Polytechnic University, Hong Kong. E-mail: csyyang@comp.polyu.edu.hk, jiannong.cao@polyu.edu.hk, {cyr20040123, kimlumchun}@gmail.com.*
- *Milos Stojmenovic is with the Singidunum University, Beograd 160622, Serbia. E-mail: mstojmenovic@singidunum.ac.rs.*
- *Senzhang Wang is with the School of Computer Science and Engineering, Central South University, Changsha 410083, China. E-mail: szwang@csu.edu.cn.*
- *Zhetao Li is with the Hunan International Scientific and Technological Cooperation Base of Intelligent Network, Key Laboratory of Hunan Province for Internet of Things and Information Security, and College of Computer Science, Xiangtan University, Xiangtan 411105, China. E-mail: liztchina@hotmail.com.*

Manuscript received 18 Aug. 2020; revised 9 May 2021; accepted 27 May 2021. Date of publication 0 . 0000; date of current version 0 . 0000.
(Corresponding author: Yu Yang.)
Recommended for acceptance by P. Cui.
Digital Object Identifier no. 10.1109/TKDE.2021.3085758

changes among vertices, the above trading patterns and strategies would be totally lost.

There are two major challenges in jointly embedding the dynamic linkage evolution and ToE for preserving the temporal evolutionary patterns of a dynamic graph. The first challenge is capturing and learning the structural evolutionary patterns of a dynamic graph from their local dynamic instances, which is the snapshot graph, in an interpretable manner. The vertices' connections and ToEs in every snapshot graph are highly dynamic, therefore making it difficult to reconstruct the global evolution process of a dynamic graph from the snapshot graph sequence in an interpretable manner. Another challenge is preserving the temporal dependency among vertices while embedding ToE. If the ToEs are aggregated for each vertex directly and appended with other vertex attributes, which is a common approach for embedding vertex attributes in static graphs [14], [15], the temporal dependency among vertices will gradually be lost due to information loss through aggregation [16]. Therefore, the embedding algorithm should maximally prevent vanishing temporal dependency while embedding ToE.

To address the above challenges, we first model the dynamic graph as a sequence of snapshot graphs with ToE for every edge. We then propose a matrix factorization based **T**ime **C**apturing **D**ynamic **G**raph **E**mbedding algorithm named *TCDGE*, which infers a common latent space for capturing the structural and temporal evolution of the dynamic graph and encodes them into representation vectors. Our approach differs from TNE [2], which embeds the snapshot graphs into separate latent spaces, since we learn a common latent space from every snapshot graph for representing the vertices' dynamic connections. When vertex connections evolve, their projected positions in the latent space will change accordingly. Therefore, vertices' moving trajectories within the common latent space reflects their evolutionary patterns in the dynamic graph.

In order to embed ToE into the representations and preserve the temporal dependencies among vertices, we first concatenate the representation of every two vertices that form an edge as features for representing their temporal dependency. We then regard the ToE as discriminative information and co-train a linear regressor using the above features while learning the common latent space. The optimization algorithm we present in this paper is generic for any linear regressor such as the LASSO regression, the ridge regression, and the elastic net regression. Finally, vertices' temporal dependency and ToE will be gradually embedded into the representations as well as the latent space during co-training.

To overcome the bottleneck of time efficiency in factorizing large-scale matrices, we optimize the latent space and the representation of the vertices by a projected gradient approach. Meanwhile, we propose a singular value decomposition (SVD) based approach to initialize our embedding algorithm. It not only helps boost the convergence speed for our algorithm but also prevents it from converging to a meaningless local optimal. Inspired by negative sampling, we introduce negative samples to co-train the regression model. Negative samples are constructed as any two vertices without any edges between them and we set their ToE to zero. This indicates that these two vertices have no temporal dependencies in the snapshot graph. Consequently, our TCDGE algorithm is very time efficient and scalable, even though the model is complicated with high-order polynomials.

Our contributions are highlighted as follows:

- We propose a matrix factorization based dynamic graph embedding algorithm to embed the temporal linkage evolution by learning a common latent space for capturing the global evolutionary patterns throughout the sequence of snapshot graphs while co-training a linear regressor, i.e., LASSO, to embed ToE for preserving vertices' temporal dependency. Our approach differs from end-to-end embedding algorithms, which usually are black boxes, by interpretively capturing vertices' temporal linkage evolution as their moving trajectories within the latent space.
- We initialize our embedding algorithm by an SVD-based method and introduce negative samples to co-train the linear regressor. Thus, our embedding algorithm is very time efficient and scalable.
- We propose a new task, namely time-aware link prediction, to validate the effectiveness of dynamic graph embedding algorithms in preserving the temporal dynamics.
- We conduct experiments on three public datasets over four machine learning applications. The experimental results show that our model achieves performance improvements of 17.00, 22.91 and 11.88 percent, respectively, over the state-of-the-art baselines in vertex classification, ToE prediction, static and time-aware link prediction.

The remainder of this paper is organized as follows. Related work is reviewed in the next section, followed by the problem definition in Section 3. We present the intuition of capturing the temporal linkage evolution in Section 4 and propose TCDGE and the optimization algorithm in Section 5. Experimental results will be reported in Section 6 before we conclude the paper in the last section.

2 Related Work

Starting with DeepWalk [1], numerous static graph embedding methods have been proposed to encode the graph structure and attributes such as high-order proximities [17], [18], vertices' centrality [19], vertex and edge attributes [15], [16], text semantics [14], [20], and communities [21], [22]. In addition to embedding a single homogeneous graph, EOE [23] and HWNN [24] infer a common latent space for respectively embedding coupled heterogeneous graphs and hypergraph. In addition to these unsupervised methods, there are several works focusing on task specific graph representation learning [25], [26]. It simultaneously train a discriminator or classifier using the labels of edges or vertices while learning the embeddings. The discriminator serves as a supervisor to make the final learned representation robust enough for discriminating the labels in specific applications. We borrow the idea of learning discriminative information while embedding the graph structure to co-train a linear regressor for encoding the ToE and temporal dependencies of vertices into the final representations.

In dynamic graph embedding, the main issue becomes handling the dynamic evolving nature of vertices and edges, and encoding their evolutionary patterns. Existing works learn the structural differences of a graph at different timestamps by either matrix factorization or deep learning approaches. For matrix factorization approaches, TNE [2] is a pioneering work that factorizes the consecutive snapshot graphs into different latent spaces with a temporal smoothness regularization. TMF [3] learns the first-order neighborhood information while factorizing the adjacency matrices of snapshot graphs. DHPE [5] employs the generalized SVD to preserve the high-order proximities and Timers [27] explores the timing of restarting SVD to overcome the error accumulation while embedding the dynamic graph. However, they fail to preserve the global structural evolution of the whole dynamic graph over time. In addition, none of them embed temporal information of vertices and edges like ToE with the structural evolution.

There exist deep learning methods that capture the specific evolution process in dynamic graphs. DynamicTriad [4] models the triad closure process when a graph evolves. HTNE [12] models the neighborhood formation sequences as a Hawkes Process with a time-aware weights. EPNE [28] learns the periodic linkage evolution patterns by causal convolutions. However, these specific dynamic processes merely exist in some particular graphs. For example, the triad closure process is not common in other networks except social networks, thus leading to poor performance. We embed temporal linkage evolution without pre-assuming any dynamic processes and give an interpretation about what happens in the latent space when the dynamic graph evolves over time.

There are also methods that approach the graph evolution process by incrementally appending out-of sample vertices or edges into the existing in-sample graph like DepthLGP [10], GraphSAGE [8], MVC-DNE [29], etc. DynGEN [6] adopts auto-encoders to incrementally handle the growing graph and its extended version Dyngraph2Vec [7] trains a LSTM to capture the evolution throughout snapshot graphs. DySAT [30] employs the self-attention mechanism to capture the structure difference throughout the snapshot graph sequence instead of using LSTM. DynGraphGAN [9] learns long-term structural evolution via adversarial training. However, none of them model the ToE and temporal dependencies of vertices, thereby failing to preserve the complete evolutionary pattern of the dynamic graph in both structural and temporal domains, which is one of the main contributions of our work.

3 Problem Definition

In this section, we give a complimentary definition of dynamic graphs and then properly formulate the dynamic graph embedding problem.

Definition 1 Dynamic Graph. *A dynamic graph* $G = \{G_{t_1}, G_{t_2}, \ldots, G_{t_n}\}$ *is a sequence of directed or undirected snapshot graphs* G_t*, where* $G_t = (V_t, E_t, W_t)$ *is a snapshot graph at time* $t \in \{t_1, t_2, \ldots, t_n\}$. V_t *is a subset of the vertex set* $V = \{v_1, v_2, \ldots, v_m\}$. *The edge* $e_{i,j}^{t,\delta} = (v_i^t, v_j^{t'}, \delta) \in E_t$ *in* G_t *represents the connection between an upcoming vertex* v_i^t *joining at time* t *and an existing vertex* $v_j^{t'}$ *appearing at time* t'*, where* $i, j \in \{1, 2, \ldots, m\}$, $t' \leq t$ *and* $\delta = t - t'$ *is the ToE of* $e_{i,j}^{t,\delta}$. *Each edge* $e_{i,j}^{t,\delta}$ *is associated with an edge weight* $w_{i,j}^{t,\delta} \in W_t$.

Since $V_t \subseteq V$ for any t, the network structure in G_t evolves over time which also leads to G evolving. At time t, the edge $e_{i,j}^{t,\delta}$ links the upcoming vertex v_i^t to an existing one $v_j^{t'}$ which joins the graph at time t'. The temporal dependency among vertices is reflected by the ToE δ. It is possible for v_i^t to form edges with the same vertex appearing at different times $v_j^{t'}$ and $v_j^{t''}$. These two edges link the same vertex pair but have different ToE δ, which gives the dynamic graph the ability to distinguish the edges between the same pair of vertices but established at two different timestamps.

Definition 1 provides a generic description of the dynamic graph. When $t_n = 1$, the dynamic graph G degenerates into a static graph. If we assume $t = t' + 1$ for all edges, G becomes a continuous-time dynamic graph defined in [11]. If we assume $t' = t$, G becomes a structure evolving dynamic snapshot graph sequence which is adopted by most of the approaches in dynamic graph embedding literature [2], [3], [4], [5], [6], [7], [8], [9], [10]. When we assume $t' = t$, $V_t \subseteq V_{t+1}$ and $E_t \subseteq E_{t+1}$, G becomes a growing graph, where the vertices and edges are only appended to the graph but not removed. Our definition of a dynamic graph is generic and captures both the structure and temporal dynamics.

Definition 2 Dynamic Graph Embedding. *Given a dynamic graph* $G = \{G_{t_1}, G_{t_2}, \ldots, G_{t_n}\}$ *and assuming that the maximum number of vertices* m *is known, the objective is to learn a mapping function* $f : v \mapsto r_v \in \mathbb{R}^k$ *for* $\forall v \in V$ *such that* r_v *preserves the temporal linkage evolution of vertex* v *in terms of the dynamic connection changes and temporal dependency, where* k *is a positive integer indicating the dimension of the representation* r_v.

4 Capturing the Evolution of Dynamic Graphs

In this section, we introduce the intuitions of capturing the evolution of a dynamic graph and interpret what happens in the latent representation space when the dynamic graph evolves.

Since each snapshot graph G_t is an instance of the dynamic graph G at time t, the dynamic change throughout the snapshot graph sequence exactly reflects the evolution of G. From a vertex point of view, this evolution process consists of the sequential changes of vertices' connections with their corresponding ToE. Embedding the dynamic graph G becomes inferring a latent space H with k dimensions that maximizes the retention of vertices' temporal connections and attributes. When projecting the snapshot graph G_t into the latent space H, every vertex in G_t obtains a response vector r_t, which is its embedding, showing its position in H. If vertices have similar connectivity and ToE, they should be close to each other in H, which means the distance between their embeddings is small.

When either vertices' connections evolve or their ToE changes, resulting from the evolution of the dynamic graph,

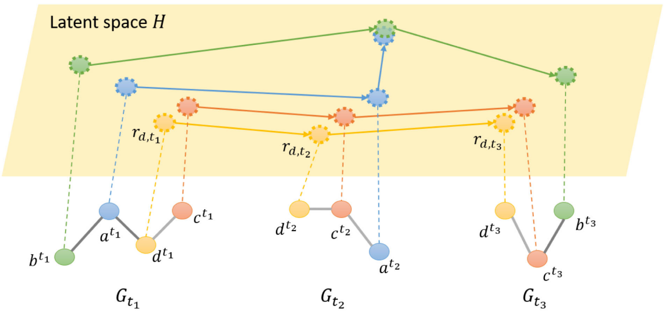


Fig. 1. An illustration of the evolution of a dynamic graph. a, b, c, and d are vertices in a dynamic graph $G = \{G_{t_1}, G_{t_2}, G_{t_3}\}$, where their edge colors represents different ToE.

TABLE 1
Notations for Time Capturing Dynamic Graph Embedding

Symbols	Description
G_t	A snapshot graph at time t, $t = t_1, t_2, \dots, t_n$
m	The maximum number of vertices in G
$A_t \in \mathbb{R}^{m\times m}$	The adjacency matrix of G_t
$M_t \in \mathbb{R}^{m\times m}$	The high-order proximity matrix of G_t
$M_t(u) \in \mathbb{R}^{m\times 1}$	The high-order proximity vector of vertex u at t
$H \in \mathbb{R}^{m\times k}$	The inferred latent representation space
$W_t \in \mathbb{R}^{k\times m}$	The learned representation matrix at t
$W_t(u) \in \mathbb{R}^{k\times 1}$	The representation vector of vertex u at t
$y^t_{u,v} \in \mathbb{R}$	The ToE of an edge linked vertices u and v at t
$x \in \mathbb{R}^{(2k+1)\times 1}$	The learned coefficients of a linear regressor

their embeddings will change accordingly, therefore causing their position in H to move. The trajectory of every vertex in H carries its evolution process throughout the snapshot graph sequence. An example of our idea is shown in Fig. 1, where vertices c and d have similar connectivity as well as ToE among their connections so that their embeddings in the latent space H should be close to each other, and their moving trajectories are also similar. Since the silent vertices disconnect from any existing vertices, they should be projected to the same position in H no matter which snapshot graph they leave. The connectivities of vertices a and c are different in the three snapshot graphs resulting in different temporal linkage evolutions, which leads to their moving trajectories being far away from each other. Finally, the embedding of any vertex v that preserves its temporal linkage evolution is obtained by Eq. (1), and represents its moving trajectory in H, where $r_{v,t} \in \mathbb{R}^k$ is its learned representation from the snapshot graph G_t, and T is a transpose operator

$$r_v = [r^T_{v,t_1}, r^T_{v,t_2}, \dots, r^T_{v,t_n}]^T. \tag{1}$$

In the next section, we will propose our dynamic graph embedding model and an optimization algorithm to efficiently infer the latent space H for embedding vertices' temporal linkage evolution.

5 Embedding Temporal Linkage Evolution

In this section, we present the details of our proposed time capturing dynamic graph embedding (TCDGE) model for encoding vertices' temporal linkage evolution as representations. Plus, we illustrate the optimization algorithm and training procedure to efficiently train the TCDGE model.

5.1 Time Capturing Dynamic Graph Embedding Model

Before introducing our TCDGE model to solve the challenges, we first list the notations that will be used in the remainder of this paper in Table 1.

The representations of vertices in a latent space should reconstruct the original dynamic graph reasonably well with the inverted latent space projector. Thus, we minimize the quadratic reconstruction loss under non-negative constraints for inferring the common latent space H and encode the representations of vertices in each snapshot graph G_t

$$\arg\min_{H,W_t} \frac{1}{2}\sum_{t=1}^{n} \|G_t - HW_t\|_F^2 \quad s.t. \quad \forall W_t \geq 0, \quad H \geq 0. \tag{2}$$

Adjacency matrices are commonly used to capture the linkage information among vertices in a graph. However, the adjacency matrices of real world graphs are usually very sparse such as those for information networks, transaction networks, etc., which introduces bias into machine learning algorithms and leads to imprecise results [31]. Additionally, the adjacency matrix only captures 1-step direct connections of vertices and is weak at representing the high-order neighborhood structure of the graph. One common approach to overcome this issue is to extract the high-order proximities of a graph from its adjacency matrix [14], [18]. In this paper, we employ high-order proximity matrix M_t of G_t as input, where $M_t = \hat{A}_t + \hat{A}_t{}^2 + \cdots + \hat{A}_t{}^m$ and $\hat{A}_t$ is the 1-step probability transition matrix obtained from the adjacency matrix A_t after a column-wise normalization. If a vertex leaves G_t, meaning that it has no connection with any existing vertices at t, we define it as a silent vertex and all elements in its corresponding A_t column are zero. Consequently, its corresponding vector in M_t is also a zero vector leading the optimally learned representations to also be zero vectors. Finally, the evolving structure of G is preserved in a sequence of high-order proximity matrices M_t. By factorizing them, we infer a common latent space H and encode the structural dynamics of the dynamic graph into the representation W_t.

In solving the second challenge and further capturing the temporal dynamics, which are the temporal dependencies of every pair of vertices carried by the ToE of their linked edges, the objective is to embed the ToE into the representations W_t while factorizing M_t. We regard every single edge as a data sample to encode their ToE individually, which is different from treating all edges in a snapshot graph as a matrix M_t for embedding the graph structure. Inspired by discriminative embedding [25], [26], we treat ToE as "supervised" information to co-train a linear regressor while encoding the representations W_t. In other words, we employ the ToE to guide the embedding process and transfer it into the learned representations. Specifically, we constrain the learned W_t such that it should have the ability to simultaneously reconstruct the graph structure and accurately predict the ToE using the co-trained regressor x. Given the ToE of an edge connecting two vertices u and v, we concatenate their representation vectors $W_t(u)$ and $W_t(v)$ together as the feature of their corresponding edge to co-train a linear regressor for estimating its ToE as follows:

$$J_{tp} = \sum_{t=1}^{n}\sum_{u,v}\Big(y_{u,v}^t - \Big[W_t(u)^T \quad W_t(v)^T \quad 1\Big]x\Big)^2 + \alpha\phi(x), \tag{3}$$

where $\phi(x)$ is a regularization of x and $\alpha > 0$ is a regression parameter. 1 is a constant for linear regression. J_{tp} is a LASSO regressor when $\phi(x) = \|x\|_1$, and it becomes a ridge regressor or an elastic net regressor if $\phi(x)$ is $\|x\|_2^2$ or $\|x\|_2^2 + \alpha'\|x\|_1$ respectively.

The temporal dependencies of u and v are embedded into their corresponding representations by the co-trained regressor since the representations of both source and target vertices are involved to regress the ToE of the edge they formed. If there does not exist any edges between u and v at time t, we set the corresponding $y_{u,v}^t = 0$, indicating that there is no temporal dependency between these two vertices at time t. When the dynamic graphs are undirected, we let $y_{u,v}^t = y_{v,u}^t$ so that every pair of vertices corresponds to the same ToE no matter how we concatenate their representation $W_t(u)$ and $W_t(v)$. In addition, if u appears multiple times in the dynamic graph, such as a seller that posts multiple selling announcements at different times in a dynamic transaction network that we mentioned in Section 1, ToE is exactly the unique discriminative information for the new coming vertex v, identifying which u it connects to. Such temporal dependencies between u and v are accurately preserved by our co-trained regressor which adopts the concatenation of their representations $W_t(u)$ and $W_t(v)$ as a feature to regress their corresponding ToE.

The co-trained linear regressor x allows our approach to identify the exact source vertex by estimating the ToE when performing link prediction. Although existing approaches can achieve the same goal by training an extra discriminator using well learned representations, their performance is not satisfactory due to the absence of discriminative information, such as ToE, for identifying the source vertex while learning the embeddings (please refer to the experimental results in Section 6.5). Therefore, the learned representation W_t has the ability to reconstruct the dynamic graph structure and preserve the temporal dependencies of vertices by approximating the ToE of every edge.

Lastly, we assume that the graph evolves smoothly instead of being totally reconstructed at every time step. Thus, we penalize vertices's sharp changes of position in the latent space by minimizing the ℓ_2 distance between representations in two consecutive snapshot graphs

$$\begin{aligned} J_{sm} = &\sum_{t=1}^{n}\sum_{u}\Big(1 - W_t(u)^T W_t(u)\Big)^2 \\ &+ \sum_{t=2}^{n}\sum_{u}\Big(1 - W_t(u)^T W_{t-1}(u)\Big)^2. \end{aligned} \tag{4}$$

In order to maintain stability when factorizing H and W_t from M_t, we employ quadratic regularizations $J_{reg} = \|H\|_F^2 + \sum_{t=1}^n \|W_t\|_F^2$ to prevent H and W_t from becoming sparse rapidly. Therefore, the overall TCDGE model is

$$\mathop{\arg\min}_{H\geq 0, W_t \geq 0, x} \frac{1}{2}\sum_{t=1}^{n}\|M_t - HW_t\|_F^2 + \frac{\lambda_1}{2}J_{reg} + \frac{\lambda_2}{2}J_{sm} + \frac{\lambda_3}{2}J_{tp}, \tag{5}$$

where $\lambda_1 > 0$, $\lambda_2 > 0$, and $\lambda_3 > 0$ are model parameters. It co-trains a linear regressor to embed the ToE y, which carries the timespan of edges and temporal dependencies of vertices, into the representation W_t while encoding the high-order proximities by factorizing M_t for simultaneously preserving the structural dynamics. Since the ToE is an attribute of the dynamic graph and naturally exists, our proposed TCDGE is still an unsupervised representation learning approach.

5.2 Optimization Algorithm

In this subsection, we will explain how the optimization problem (5) was solved in detail. We aim to find the optimal latent space H, the representations of vertices W_t and the regression coefficient x. It is suitable to use an alternating directions method to solve this optimization problem by fixing H and x to solve W_t followed by fixing W_t to update H and x.

5.2.1 Optimizing Vertex Presentation W_t

Since $W_t(u)$ and $W_t(v)$ are a part of W_t, it is difficult to handle the integrated vector $[W_t(u)^T, W_t(v)^T, 1]$ in J_{tp} when solving for W_t. Thus, we let $x = [x_u^T, x_v^T, x_0]^T$, where $x_u \in \mathbb{R}^{k\times 1}$, $x_v \in \mathbb{R}^{k\times 1}$, and $x_0 \in \mathbb{R}$, and rewrite J_{tp} as

$$J_{tp} = \sum_{t=1}^{n}\sum_{u,v}\Big(y_{u,v}^t - W_t(u)^T x_u - W_t(v)^T x_v - x_0\Big)^2 + \alpha\phi(x). \tag{6}$$

We obtain the objective function of optimizing W_t as Eq. (7), which is a fourth-order polynomial and is non-convex

$$\begin{aligned} \mathop{\arg\min}_{W_t\geq 0} &\frac{1}{2}\sum_{t=1}^{n}\|M_t - HW_t\|_F^2 + \frac{\lambda_1}{2}\sum_{t=1}^{n}\|W_t\|_F^2 + \frac{\lambda_2}{2}J_{sm} \\ &+ \frac{\lambda_3}{2}\sum_{t=1}^{n}\sum_{u,v}\Big(y_{u,v}^t - W_t(u)^T x_u - W_t(v)^T x_v - x_0\Big)^2. \end{aligned} \tag{7}$$

Therefore, we adopt a block coordinate descent approach to solve W_t. When updating $W_t(u)$ for each vertex u at time t, we fix the H, x, and $W_t(v)$ of all the other vertices v at time t as well as all the representations W that are not at time t. Consequently, the W_t problem becomes a convex optimization problem as shown in Eq. (8)

$$\begin{aligned} \mathop{\arg\min}_{W_t(u)\geq 0} f(W_t(u)) = &\mathop{\arg\min}_{W_t(u)\geq 0}\frac{1}{2}\|M_t(u) - HW_t(u)\|_2^2 + \frac{\lambda_1}{2}\|W_t(u)\|_2^2 \\ &+ \frac{\lambda_2}{2}\Big(\Big(1 - W_t(u)^T W_t(u)\Big)^2 + \Big(1 - W_t(u)^T W_{t-1}(u)\Big)^2\Big) \\ &+ \frac{\lambda_3}{2}\sum_{v}\Big(y_{u,v}^t - W_t(u)^T x_u - W_t(v)^T x_v - x_0\Big)^2 \\ &+ \frac{\lambda_3}{2}\Big(y_{u,u}^t - W_t(u)^T x_u - W_t(u)^T x_v - x_0\Big)^2. \end{aligned} \tag{8}$$

If we choose to ignore situations where vertices can link to themselves (self-links), the last term in Eq. (8) could be removed. We use the projected gradient methods [32] to solve this convex optimization problem and obtain the updating function of $W_t(u)$

$$W_t(u) = \max\{W_t(u) - \beta \bigtriangledown f(W_t(u)), 0\}, \tag{9}$$

where $\beta > 0$ is the learning rate and the gradient $\bigtriangledown f(W_t(u))$ satisfies

$$\begin{aligned} \bigtriangledown f(W_t(u)) &= H^T H W_t(u) - H^T M_t(u) + \lambda_1 W_t(u) \\ -\lambda_2 \Big(W_t(u)\Big(1 - W_t(u)^T W_t(u)\Big) &+ W_{t-1}(u)\Big(1 - W_t(u)^T W_{t-1}(u)\Big)\Big) \\ &- \lambda_3 \sum_v \Big(y^t_{u,v} - W_t(u)^T x_u - W_t(v)^T x_v - x_0\Big) x_u \\ &- \lambda_3 \Big(y^t_{u,u} - W_t(u)^T (x_u + x_v) - x_0\Big)(x_u + x_v). \end{aligned} \tag{10}$$

To ensure a sufficient decrease of Eq. (9) and to speed up convergence, we update the learning rate β with a scaling factor θ to make the new $W_t(u)$ satisfy

$$f(W_t^{i+1}(u)) - f(W_t^i(u)) \leq \sigma_1 \bigtriangledown f(W_t^i(u))^T \big(W_t^{i+1}(u) - W_t^i(u)\big), \tag{11}$$

where i is the number of iterations and σ_1 is a tolerance. With the proof by Bertsekas in [33], there always exists a $\beta > 0$ that satisfies the rule (11) and every limit point of $\{W_t^i(u)\}_{i=1}^{\infty}$ is a stationary point of the bound-constrained optimization problem (8) [32]. After optimizing every $W_t(u)$ for every vertex in G_t, an optimal W_t is obtained. The pseudo code for solving W_t is presented in Algorithm 1.

Algorithm 1. The Projected Gradient Algorithm of Solving W_t

Input: $M_t, H, x, W_t^0, y^t, \lambda_1, \lambda_2, \lambda_3, 0 < \theta < 1, 0 < \sigma_1 < 1$
Output: W_t

```
1:  repeat
2:    for u = 1, 2, ..., m do
3:      β^0 = 0.01
4:      for i = 1, 2, ... do
5:        β^i = β^{i-1}
6:        if β^i satisfies Eq. (11) then
7:          repeat
8:          β^i = β^i / θ
9:          until β^i does not satisfy Eq. (11)
10:       else
11:         repeat
12:         β^i = β^i · θ
13:         until β^i satisfies Eq. (11)
14:       end if
15:       Update W_t(u) by using Eq. (9)
16:     end for
17:   end for
18: until converge.
19: return W_t
```

5.2.2 Optimizing Common Latent Space H

When fixing W_t and x, the H optimization problem can be addressed by solving

$$\arg\min_H h(H) = \arg\min_{H \geq 0} \frac{1}{2} \sum_{t=1}^{n} \|M_t - HW_t\|_F^2 + \frac{\lambda_1}{2} \|H\|_F^2. \tag{12}$$

This is also a convex bound-constrained optimization problem that is again solvable using the projected gradient method, which is similar to the approach we employed in solving W_t. The updating function of H is

$$H = \max\{H - \beta \bigtriangledown h(H), 0\}, \tag{13}$$

where the gradient of $h(H)$ is

$$\bigtriangledown h(H) = \sum_{t=1}^{n} (HW_t - M_t) W_t^T + \lambda_1 H. \tag{14}$$

When optimizing H, we adopt the same learning rate updating strategy in solving W_t here to ensure sufficient decent under the condition (15). σ_2 is the tolerance and i is the number of iterations

$$h(H^{i+1}) - h(H^i) \leq \sigma_2 \bigtriangledown h(H^i)^T \big(H^{i+1} - H^i\big). \tag{15}$$

5.2.3 Co-Training Linear Regressor for Embedding ToE

Fixing H and W_t for all t to optimize x is a standard linear regression problem. When rewriting the J_{tp} in Eq. (3) in matrix form, we obtain the objective function of optimizing x by Eq. (16), where $Z = [Z_1^T, Z_2^T, \ldots, Z_n^T]^T$ and $y = [y_1^T, y_2^T, \ldots, y_n^T]^T$, which is a standard linear regression problem

$$\arg\min_x \frac{\lambda_3}{2} J_{tp} = \arg\min_x \frac{\lambda_3}{2} \|y - Zx\|_2^2 + \frac{\alpha\lambda_3}{2} \alpha\phi(x). \tag{16}$$

Z_t for $t = 1, \ldots, n$ contains the concatenated features of any pair of vertices in the snapshot graph G_t as showed in Eq. (17) and $y_t \in \mathbb{R}^{m^2}$ is the corresponding ToE. The standard algorithm can be directly applied to solve the linear regression problem with different regularization $\phi(x)$ and finally get x

$$Z_t = \begin{bmatrix} W_t(1)^T & W_t(1)^T & 1 \\ W_t(1)^T & W_t(2)^T & 1 \\ \vdots & \vdots & \vdots \\ W_t(1)^T & W_t(m)^T & 1 \\ W_t(2)^T & W_t(1)^T & 1 \\ \vdots & \vdots & \vdots \\ W_t(2)^T & W_t(m)^T & 1 \\ \vdots & \vdots & \vdots \\ W_t(m)^T & W_t(m)^T & 1 \end{bmatrix} \in \mathbb{R}^{m^2 \times (2k+1)}. \tag{17}$$

Since the connections of vertices usually evolve very frequently in a dynamic graph, which leads to substantial changes to the concatenated edge features but only has a slight impact on ToE, LASSO is very robust for embedding the ToE and unlikely to overfit. In the remainder of this paper, we specifically employ the LASSO regressor, letting $\phi(x) = \|x\|_1$, for illustration. To obtain the optimal LASSO regressor x, we first let $g(x) = \frac{\lambda_3}{2}\|y - Zx\|_2^2$, and then compute its gradient by $\nabla g(x) = \lambda_3(Z^T Zx - Z^T y)$. Lastly, we employ the FISTA algorithm [34] to solve the LASSO problem and obtain an optimal x with

$$x = S_{\frac{\alpha\lambda_3}{2}}(x - \gamma\nabla g(x)), \tag{18}$$

where $S(\cdot)$ is a soft-threshold calculator. $\gamma = 1/\lambda_{max}(Z^T Z)$ where $\lambda_{max}(Z^T Z)$ is the maximum eigenvalue of $Z^T Z$ which is the smallest Lipschitz constant of $\nabla g(x)$. The computational complexity of the FISTA algorithm is only $O(1/m^2)$ [34] which solves the LASSO very efficiently.

5.3 Efficient Training Procedure and Convergence

Although the projected gradient algorithm and the FISTA algorithm are efficient for matrix factorization and LASSO regression respectively, there exist two bottlenecks in further improving the training efficiency and making TCDGE converge faster. One bottleneck is the initialization of W_t and H to make them close to the optimal point for reducing the training time while preventing them from sticking into meaningless local optima. The other bottleneck is that very large-scale training samples make the FISTA algorithm very time-consuming in computing the gradient $\nabla g(x)$. Training sets containing too many edges with zero ToE impair the training precision of linear regressor as well. Here, we present an initialization approach using singular value decomposition (SVD) and an efficient FISTA training procedure to address the above efficiency bottlenecks.

5.3.1 Initialization of W_t and H by SVD

The TCDGE algorithm cannot be initialized by randomly generated H^0 and W_t^0. Usually, M_t is a sparse matrix but randomly generated H^0 and W_t^0 are all dense matrices. It will make either or both H and W_t become zero matrices after a few iterations. Thus, the algorithm stops at a local optimum and outputs meaningless results.

To avoid reaching the zero local optimal point, the initialized H^0 and W_t^0 should meet the requirement $\|M_t - H^0W_t^0\|_F^2 \leq \|M_t\|_F^2$ [32]. Therefore, we adopt SVD to initialize H^0 and W_t^0 as follows. First, we decompose every M_t and obtain its left-singular matrix U_t, singular value matrix I_t, and right-singular matrix S_t. Then, we select the rectangular diagonal sub-matrix from I_t corresponding to the top k singular values, and the first k columns from U_t and S_t denoted as $I_{t,k}$, $U_{t,k}$ and $S_{t,k}$. Finally, we initialize H^0 and W_t^0 by

$$H^0 = \frac{1}{n}\sum_{t=1}^{n} U_{t,k} \quad and \quad W_t^0 = I_{t,k}S_{t,k}^T. \tag{19}$$

Using SVD to initialize our embedding algorithm prevents it from being stuck in the zero local optimum and allows it to pursue meaningful results.

5.3.2 Efficient Linear Regressor Training With Negative Sampling

To capture all of the temporal dependencies among the m vertices in a dynamic graph consisting of n snapshot graphs, $m^2 \times n$ training samples in Z are used to co-train the linear regressor in every time step. Because of the high dimensionality of Z, computing the gradient $\nabla g(x)$ is very time-consuming. Meanwhile, many vertices usually do not connect to each other in real cases. Thus, edges with zero ToE are much more common than nonzero ToE edges, which causes imbalance issues and impairs the precision of the co-trained regression model.

Inspired by negative sampling [35], we mark all edges with nonzero ToE as positive samples and randomly choose a set of zero ToE edges, following a uniform distribution, as negative samples to jointly train the regressor. Different from deep learning models that just select a very small number of negative samples based on the label difference for training, we restrict the number of negative samples to half of the number of positive ones because negative samples in our model indicate vertices having no temporal dependency which is one of the most important pieces of information that should be learned by the regressor.

After negative sampling, the training samples in Z are dramatically reduced and positive samples become majorities, therefore saving the computational cost in calculating $\nabla g(x)$ and preventing the regressor from being dominated by the negative samples, which makes it converge quickly and precisely. We have tried selecting negative samples based on a probability distribution that is proportional or inversely proportional to the vertex degree but the experimental results show that this is rarely much different from following the uniform distribution.

5.3.3 Convergence and Stop Criteria

The overall work flow of the TCDGE algorithm is presented in Algorithm 2, which essentially is a block-wise coordinate descent algorithm. Therefore, its convergence can be guaranteed according to the proof of convergence of block-wise coordinate descent [36]. Both algorithms for optimizing W_t and H stop when they meet the condition in Eqs. (20) and (21), which ensures the optimization outputs are close to a stationary point [32]. ϵ is a very small positive number. For the jth element a_j in vector a, $p(\cdot)$ equals the gradient at a_j if $a_j > 0$ else $p(\cdot)$ equals the negative gradient at a_j

$$\|p(\bigtriangledown h(H^i))\|_2 \leq \epsilon\| \bigtriangledown h(H^1)\|_2 \tag{20}$$

$$\|p(\bigtriangledown f(W_t^i(u)))\|_2 \leq \epsilon\| \bigtriangledown f(W_t^1(u))\|_2. \tag{21}$$

The FISTA algorithm for embedding the ToE stops when the residual of x is less than a small positive number ϵ'.

Algorithm 2. The TCDGE Algorithm

Input: M_t, y, Z, x^0, τ^0, λ_1, λ_2, λ_3, α, $0 < \theta < 1$, $0 < \sigma_1 < 1$, $0 < \sigma_2 < 1$
Output: H, W_t, x

Initialize H^0 and W_t^0 by Eq. (19)
Initialize x by the FISTA algorithm
repeat
    for $t = 1, 2, \ldots, n$ **do**
        Update W_t by Algorithm 1
    end for
    Update H by the projected gradient algorithm
    Update x by the FISTA algorithm
until converge.
return H, W_t, x

TABLE 2
Statistics of Datasets

Dataset	$\lvert V\rvert$	$\lvert E\rvert$	$\lvert G_t\rvert$	Mean ToE	Std ToE	#Classes
UCI Messages	1899	22640	7	0.7387 (days)	2.1762	-
Transaction	5881	35592	11	1.4637 (months)	1.9303	2
Co-authorship	10374	60101	5	1.3834 (years)	1.0414	3

6 Experimental Results and Analysis

In this section, we conduct extensive experiments to showcase the effectiveness and efficiency of the TCDGE algorithms in the data mining tasks of vertex classification, ToE prediction, static link prediction, and time-aware link prediction.

6.1 Experimental Setting

6.1.1 Datasets

Three public real-world datasets are considered when validating the performance of TCDGE on data mining applications, whose statistics are presented in Table 2.

UCI Messages[1] [37] is an online communication network of students. A vertex represents a student that has sent or received messages. The ToE is the communication time interval between a pair of students. The communication lasts 7 months so that a dynamic graph containing 7 snapshot communication graphs has been built for capturing their dynamic communication behaviors.

Transaction[2] [38] is a bitcoin transaction network. A vertex is a trader who buys and sells bitcoins and an edge forms while two traders complete a transaction. The ToE is the time interval between buying and selling. Each snapshot graph carries the transactions in a 6 month period. Since bitcoin traders are anonymous, there is a need to maintain a record of their reputation to prevent transactions with fraudulent and risky traders. Traders rate each other's trustworthiness on a scale of -10 (total distrust) to +10 (total trust) with a step of 1 after completing each transaction, so that we label traders whose average score is above 1 as trustworthy while the rest are deemed untrustworthy. Finally, we obtain 1092 untrustworthy traders and 4789 trustworthy ones.

We derive a *Co-authorship*[3] network for publications from 2010 to 2014 in three research areas including networking (NW), data mining (DM) and artificial intelligence (AI) from the DBLP. A vertex is an author and two authors form an edge when they coauthor a paper. The ToE indicates the time interval between co-authorship. We deem researchers that have coauthored with not less than 6 other authors and at least coauthored with one of them twice in that period. The snapshot graphs represent the co-authorship in every year. We label the vertices by their research areas which they published most in. Finally, we obtains 3405 authors in NW, 2909 authors in DM, and 4060 authors in AI.

6.1.2 Baseline Methods

We benchmark our TCDGE algorithm to 7 state-of-the-art methods listed below using their published codes.

- *DeepWalk*[4] [1] is a static graph embedding algorithm that employs skip-gram to encode linkage relationships among vertices searched by the random walk. We tested the combination of hyper parameters given window sizes $ws \in \{5, 8, 10\}$, walk lengths $wl \in \{10, 20, 30, 40\}$, and numbers of walks $nw \in \{20, 40, 60\}$, and report the best results.
- *Temporal Network Embedding (TNE)*[5] [2] is a matrix factorization based dynamic graph embedding method that encodes the structure evolving patterns in different latent spaces. We tested the hyper parameter $\lambda \in \{0.01, 0.1, 1, 10\}$, and report the best results.
- *Timers*[6] [27] is an incremental SVD approach for dynamic graph embedding which overcomes the error accumulation issues by restarting SVD when the error margin exceeds a threshold. We use the default parameter settings $\theta = 0.17$.
- *DynamicTriad*[7] [4] preserves the triad closure process while embedding the structural evolution. We tested all combinations of hyper parameters $\beta_0, \beta_1 \in \{0.01, 0.1, 1, 10\}$, and report the best results.
- *GraphSAGE*[8] [8] is a graph convolutional network approach for embedding the structural evolution of a dynamic graph. We train a two layer model with respective neighborhood sample sizes 25 and 10, as described in the original paper. We test different aggregators including GCN, mean, mean-pooling, and LSTM and report the performance of the best performing aggregator in each dataset.
- *DynGEN*[9] [6] adopts a deep auto-encoder to embed the structure changes throughout the snapshot graph sequence. We train a two layer model and adopt the default parameter settings that are recommended by the authors.
- *DynG2vecAERNN*[9] [7] is an extension of DynGEN which first adopts a deep neutral network to encode the structure of each snapshot graph, and then employs an LSTM to embed the sequential evolution of every vertex throughout the snapshot graphs. A two layer model is trained with the default parameter setting as described in the original paper.

In order to verify the effectiveness of learning the common latent space H to capture the linkage evolution, we experiment with our TCDGE without embedding ToE by setting $\lambda_3 = 0$, namely TCDGE-noToE. Meanwhile, we test another variant TCDGE, namely TCDGE-wgToE, that adopts the ToE as weights of the adjacency matrix of each snapshot graph but does not co-train any regression model, thus verifying the effectiveness of our co-training approach.

6.1.3 Evaluation Metrics

We employ micro-F1 and macro-F1 scores as evaluation metrics for the task of vertex classification as seen below:

1. http://konect.uni-koblenz.de/networks/opsahl-ucsocial
2. https://snap.stanford.edu/data/soc-sign-bitcoin-otc.html
3. http://projects.csail.mit.edu/dnd/DBLP/
4. https://github.com/phanein/deepwalk
5. https://github.com/linhongseba/Temporal-Network-Embedding
6. https://github.com/ZW-ZHANG/TIMERS
7. https://github.com/luckiezhou/DynamicTriad
8. https://github.com/williamleif/GraphSAGE
9. https://github.com/palash1992/DynamicGEM

$$\text{Micro-F1} = \frac{2\sum_i \mathrm{TP}_i}{\sum_i (2\mathrm{TP}_i + \mathrm{FP}_i + \mathrm{FN}_i)} \tag{22}$$

$$\text{Macro-F1} = \frac{1}{c}\sum_i \frac{2\mathrm{TP}_i}{2\mathrm{TP}_i + \mathrm{FP}_i + \mathrm{FN}}, \tag{23}$$

where TP_i, FP_i, and FN_i are the true positive, false positive, and false negative results of the ith predicted class, respectively. The macro-F1 score is the mean of the class-wise F1 score that is sensitive to the performance in classifying each individual class. The micro-F1 score measures the overall classification performance regardless of the accuracy in individual classes. Higher micro-F1 and macro-F1 scores indicate better vertex classification performance.

We evaluate the performance of ToE prediction by measuring the Root Mean Square Error (RMSE) between the predicted ToE and the ground truth as

$$\mathrm{RMSE} = \sqrt{\frac{\sum_{y\in\mathcal{S}_{test}}(y-\hat{y})^2}{|\mathcal{S}_{test}|}}, \tag{24}$$

where y denotes the real ToE in the test set $\mathcal{S}_{test}$ and $\hat{y}$ is the predicted one. $|\mathcal{S}_{test}|$ is the number of test samples in $\mathcal{S}_{test}$. The smaller the RMSE, the more accurate the ToE prediction.

In link prediction, we employ the average area under the curve (AUC) of the receiver operating characteristic (ROC) curve as the performance metric. The higher the AUC, the better the link prediction performance.

6.1.4 Parameter Setting

The experiments have been conducted with $k = 45$ as the dimension of the representation vector for both our method and all baselines in all testing datasets. For the parameters of TCDGE, we set the scaling factor $\theta = 0.5$, tolerance $\sigma_1 = \sigma_2 = 0.01$. We co-train a LASSO regressor for our TCDGE with initial regression parameter $\alpha = 1$ Since W_t will be updated at each time step, making Z change dynamically, the regression parameter α cannot be fixed. Otherwise, the LASSO cannot adequately fit the ToE by using the new Z at each time. In addition, the training error of LASSO will gradually accumulate so that the reconstruction error of the overall embedding model will progressively increase, thus leading to poor embedding results. We adopt θ to dynamically update α 10 times using the same updating strategies in the projected gradient algorithm for learning the best LASSO regressor x at each time. Finally, we report the best results by testing the combination of model parameters given $\lambda_1 \in \{0.001, 0.01, 1\}$ and $\lambda_2, \lambda_3 \in \{0.0001, 0.001, 0.01, 0.1, 1\}$ for the data mining tasks presented in the following subsections. All experiments are conducted on a standard workstation with 2 Intel Xeon Gold 6128 CPUs and 64GB RAM, and are implemented in MATLAB.

6.2 Vertex Classification

Vertex classification aims to identify the unique label of vertices using their learned representations in the dynamic graph G. We first learn the representation of vertices in every snapshot graph G_t. Then, concatenate the representations W_t together by Eq. (1) for classification. A support vector machine (SVM) with a Gaussian kernel is trained by using these features to classify their corresponding labels. It tests the embedding algorithms' ability to capture the global graph evolutionary patterns in G for all timestamps. Since the UCI messages dataset does not contain vertex labels, we compare the classification performance in both bitcoin transactions and co-authorship datasets. We repeat the 5-fold cross-validation on both datasets 10 times and compare the average performance in macro-F1 and micro-F1 scores. We did not adopt any extra methods to handle the issues of unbalanced labels in the bitcoin transaction dataset but straightforwardly train the SVM for testing the actual performance of our TCDGE algorithm in the case of label unbalanced classification. The results are shown in Table 3.

TABLE 3
Vertex Classification Results

	Transaction		Co-authorship	
	Micro-F1	Macro-F1	Micro-F1	Macro-F1
DeepWalk	0.8855	0.7229	0.5645	0.5684
TNE	0.8673	0.6777	0.4221	0.4064
Timers	0.7810	0.4896	0.3358	0.2965
GraphSAGE	0.6205	0.6256	0.4793	0.4537
DynamicTriad	0.8652	0.6737	0.5243	0.5189
DynGEN	0.8316	0.6680	0.5150	0.5065
DynG2vecAERNN	0.8140	0.4721	0.4681	0.5398
TCDGE-noToE	0.8621	0.7390	**0.6921**	**0.7220**
TCDGE-wgToE	0.8625	0.7402	0.6701	0.6885
TCDGE	**0.9032**	**0.7725**	0.6728	0.7079

In the bitcoin transaction dataset, our TCDGE algorithm achieves the best performance, and outperforms the best baseline by 2.00 percent in micro-F1 scores and by 4.36 percent in macro-F1 scores. In the co-authorship dataset, TCDGE and its variants, TCDGE-noToE and TCDGE-wgToE, dramatically outperform all 7 other baseline methods. This indicates that capturing the moving trajectories of vertices in the common latent space, learned throughout the snapshot graph sequence by our proposed approach, embeds the evolution of a dynamic graph better than the baseline methods. In addition, the temporal evolution patterns captured by our approach work much better than the baselines in unbalanced label classification.

In the co-authorship dataset, TCDGE-noToE performs the best. This may be because the standard deviation of its ToE is relatively small meaning that the time intervals of co-authoring papers are not as significant as who they co-author with over time for classifying their research areas. Therefore, purely capturing the linkage evolution may be good enough for classifying authors' research areas from their co-authorship, and our TCDGE and TCDGE-wgToE achieve close performance, yet slightly worse than TCDGE-noToE but still much better than the baselines.

When people trade bitcoins, the time interval between transactions becomes important for measuring traders' trading behavior and strategies, which results in higher standard deviation of ToEs. Since our TCDGE algorithm successfully embeds both structural evolution and the temporal information of edges at the same time, it achieves the highest macro

TABLE 4
Average RMSE of ToE Prediction

	UCI Messages	Transaction	Co-authorship
DeepWalk	2.5934	2.1084	1.0395
TNE	2.5335	2.0932	1.0377
Timers	2.1536	2.1074	1.0428
GraphSAGE	2.1471	2.1004	1.0392
DynamicTriad	2.3329	2.3485	1.3425
DynGEN	2.4160	2.4053	1.0395
DynG2vecAERNN	2.1640	1.9756	0.9891
TCDGE-noToE	2.1957	2.0920	1.0371
TCDGE-wgToE	2.1595	2.0659	1.0452
TCDGE	**2.1419**	**1.7798**	**0.8967**

and micro F1 scores and dramatically outperforms the traditional models which merely capture the linkage information. Although TCDGE-wgToE leverages the ToE as weights of the adjacency matrices for embedding, the temporal dependency among vertices gradually diminishes during embedding due to the aggregation throughout the snapshot graph sequence, which is consistent with the conclusion drawn in [16]. However, co-training the LASSO regressor has the ability to better preserve the temporal dependency among vertices and encode it into the final representation. Therefore, embedding the temporal dependency together with the structural evolution among vertices into a common latent space makes the learned representation vectors preserve the global structural and temporal evolutionary patterns from the whole dynamic graph, which is more discriminative and leads to better classification results.

6.3 ToE Prediction

The objective of ToE prediction is to estimate the ToE of an edge given the representation of its source and target vertices for testing how effectively the learned representations capture temporal information. The experiment is conducted under the leave-one-snapshot-graph-out cross-validation setting. Since our model co-trains a LASSO regressor simultaneously with the representation learning, a snapshot graph is selected for testing at each round and we use the rest of the snapshot graphs to train our model until every snapshot graph serves as the testing graph once. When testing the baseline methods, we first generate all the representations from every snapshot graph, and then employ them to further train a LASSO regressor under the same cross-validation setting. We repeat each experiment 10 times and report the average RMSE.

The ToE prediction results are presented in Table 4. Our method achieves a 12.85 percent lower RMSE on average against all baseline methods and outperforms the best baseline by 6.50 percent indicating that the temporal dynamics are preserved by our proposed co-training approach, which results in much lower ToE prediction errors than the baseline approaches that ignore it. The representations learned by our TCDGE carry both structural evolution of the dynamic graph and its ToE such that it is more effective when discriminating temporal information than those approaches that purely embed the graph structure, which leads to better performance in ToE prediction.

TABLE 5
Average AUC of Static Link Prediction

	UCI Messages	Transaction	Co-authorship
DeepWalk	0.6619	0.9028	0.5977
TNE	0.6524	0.8264	0.5861
Timers	0.4943	0.4938	0.5156
GraphSAGE	0.5091	0.5624	0.5890
DynamicTriad	0.5187	0.4197	0.5950
DynGEN	0.6028	0.5826	0.4874
DynG2vecAERNN	0.4977	0.5218	0.4949
TCDGE-noToE	0.7003	0.9194	0.6044
TCDGE-wgToE	0.6969	0.9187	0.6037
TCDGE	**0.7314**	**0.9248**	**0.6142**

6.4 Static Link Prediction

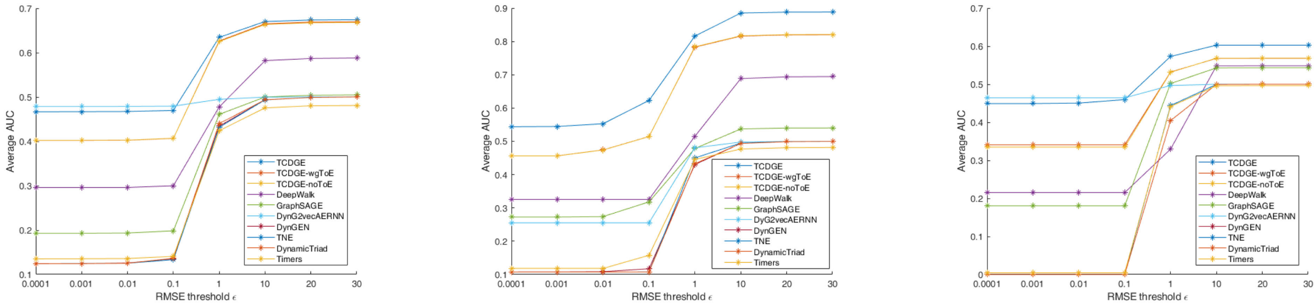
Static link prediction aims to predict whether a pair of vertices will form an edge at time $t+1$, given their embeddings learned at t. This task ignores the joining time of source vertices, which is widely adopted by the existing work to test the performance of learned embeddings. Here we employ the cosine distance to measure the similarity of two vertices in the latent space and calculate the probability of forming a new edge by the sigmoid function. We predict the links in snapshot graph G_{t+1} by using the representation W_t under the same experimental settings as those of [2]. The performance is measured by the average AUC for predicting G_2 to G_n.

The results are reported in Table 5. Overall, our proposed TCDGE algorithm outperforms all baselines by 27.56 percent on average with respect to the AUC, and achieves 2.22 percent higher AUC than the best baseline method TCDGE-noToE on average in all three datasets. The baseline approaches only learn from the linkage information. However, our TCDGE algorithm not only learns the evolving patterns of who the vertices link to, but also embeds how they link by capturing their ToEs and temporal dependency such that the edges between the same pair of vertices but established at two different timestamps can be distinguished. Therefore, our TCDGE algorithm achieves better static link prediction performance in terms of higher AUC than all baselines.

6.5 Time-Aware Link Prediction

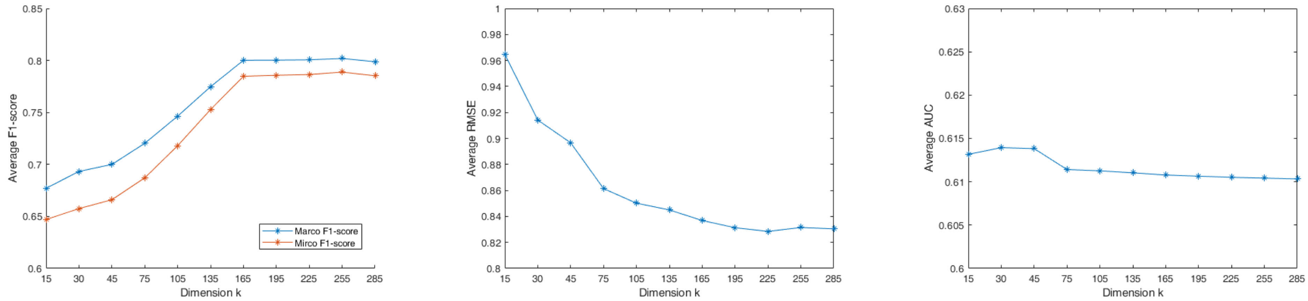
Time-aware link prediction is a unique application for dynamic graph embedding, which aims to identify the joining time of existing vertices on top of the static link prediction. It performs two tasks at the same time. One is to predict whether a pair of vertices will form an edge at time $t+1$ when given their representations at time t. The other is to predict the joining time of existing vertex to identify the unique one since it can join the dynamic graph several times. Specifically, data mining applications such as predicting which sell order will be completed by a buyer, predicting the future victims of fraud and when the fraud will happen, recommending items at an appropriate time, etc., can all be abstracted as time-aware link prediction applications.

Since the joining time of an existing vertex is equal to the difference between the ToE and the joining time of an



(a) Results for the UCI message dataset (b) Results for the Transaction dataset (c) Results for the Co-authorship dataset

Fig. 2. Average AUC of time-aware link prediction with varying threshold ϵ.



(a) Average F1-score of vertex classification (b) Average RMSE of ToE prediction (c) Average AUC of static link prediction

Fig. 3. Testing the hyperparameter k in the co-authorship dataset.

upcoming vertex, predicting the joining time of existing vertices at the time when the upcoming one joins the dynamic graph is the same as predicting the ToE of the edge they form. Thus, we predict the ToE instead of the actual joining time of existing vertices in this experiment.

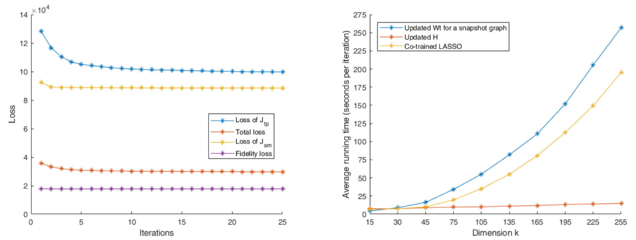
We conduct the experiment under the one-snapshot-graph-ahead cross-validation setting in which a snapshot graph G_t $(t > 1)$ is selected for testing at each round and we use the snapshot graph sequence $\{G_1, \ldots, G_{t-1}\}$ to train our model until every snapshot graph except G_1 serves as the testing graph once. Since none of the baselines can achieve the two goals in time-aware link prediction simultaneously, we employ the same cross-validation setting to obtain baselines' representations and then further train a LASSO regressor to predict the ToE. We adopt the same approach used in static link prediction to determine whether there exists an edge connecting a pair of vertices here.

A temporal link has been correctly predicted if and only if the model correctly predicts that a pair of vertices formed an edge and the RMSE of ToE prediction for this edge is less than a threshold ϵ. To test how the prediction accuracy of ToE affects time-aware link prediction, we perform time-aware link prediction in three datasets and test the threshold ϵ from 0.0001 to 30. The experiment repeats 10 times for each threshold and the average AUC are reported in Fig. 2.

Our TCDGE performs the best in all three testing datasets when $\epsilon > 0.1$. It also achieves the highest AUC in the bitcoin transaction dataset and dramatically outperforms other baselines except DynG2vecAERNN in the remaining two datasets when $\epsilon \leq 0.1$. DynG2vecAERNN works better in ToE prediction than other baselines (refer to Table 4) but is comparatively much worse in link prediction (refer to Table 5) such that it achieves relatively high AUC with small ϵ but it cannot correctly predict more temporal links when relaxing the threshold ϵ. Although our TCDGE performs slightly worse than DynG2vecAERNN with small ϵ, it becomes the best of all when $\epsilon = 1$, and AUC increases slowly when $\epsilon > 1$. This indicates that the RMSE of ToE prediction for most temporal edges predicted by our TCDGE is less than 1. Consequently, our LASSO co-training approach preserves the temporal dynamics well while embedding the ToE, therefore resulting in superior performance in time-aware link prediction.

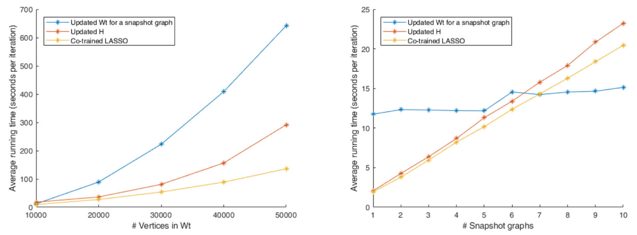
6.6 Parameter Sensitivity Analysis

The TCDGE defined by Eq. (5) is dependent on regularizer weights λ_1, λ_2, λ_3 and a hyperparameter k which is the dimension of the latent representation space as well as the dimension of the learned embeddings. The selection of λ_1, λ_2 and λ_3 highly depends on the input data and the selection approach has been illustrated in Section 6.1.4. Therefore, we conduct sensitivity analysis on the hyperparameter k from 15 to 285 in vertex classification, ToE prediction, and static link prediction. The co-authorship dataset is adopted here because the scale is relatively large compared to the other two datasets and the number of vertices in the three categories are almost balanced, which is more common in daily life. We fix $\lambda_1 = 0.0001$, $\lambda_2 = \lambda_3 = 0.01$ and only vary k at each time. As shown in Fig. 3, when k increases, both F1-scores in vertex classification increase almost linearly and gradually converge. The RMSE of ToE prediction decreases exponentially and converges with increasing k. The average AUC of static link prediction is not sensitive to the dimension of the representations. Since all results



(a) Convergence curves (b) Training time with varying k

Fig. 4. Convergence and training efficiency of TCDGE in the co-authorship dataset.



(a) Training time with varying the number of vertices (b) Training time with varying the number of snapshot graphs

Fig. 5. Scalability test results of TCDGE in the synthesized dataset.

eventually converge to the best case when the k is high enough, our TCDGE is not sensitive to the dimension of the common latent space k.

6.7 Convergence and Training Efficiency

We demonstrate the convergence of our TCDGE algorithm in the co-authorship dataset which has the highest number of vertices. The loss of the objective function in Eq. (5), fidelity term $\frac{1}{2}\sum_{t=1}^{n}\|M_t - HW_t\|_F^2$, the LASSO regressor J_{tp} in Eq. (3) when $\phi(x) = \|x\|_1$, and temporal smoothness regularization J_{sm} in Eq. (4) are shown in Fig. 4a.

Our model converges in very few iterations because of the initialization set by the SVD and the effectiveness of the projected gradient method for solving W_t and H. Our initialization approach not only prevents our TCDGE from being stuck in the zero local optimum, but also generates an approximation of M_t upon initialization, which already decreases the loss of fidelity. In the projected gradient method, a scaler θ is employed to search for a good learning rate to ensure the sufficient decrease of the gradient, thereby boosting the convergence speed of the overall TCDGE algorithm.

Fig. 4b shows the average running time of each iteration while training our TCDGE by varying the hyperparameter k. As k increases, the running time for updating H at each iteration hardly increases. The running time of updating W_t and co-training x almost linearly grows with increasing k. It takes less than 1 minutes to finish updating the representation for over ten thousand vertices when $k <= 105$ and less than 5 minutes when $k = 255$.

Although our TCDGE model looks complex, it converges quickly in terms of a small number of iterations and a very short running time for encoding the representation W_t, learning the common latent space H, and co-training the LASSO regressor x, demonstrating the effectiveness of the projected gradient method and the our proposed efficient training procedure.

6.8 Scalability of TCDGE

We synthesize two datasets on top of the co-authorship dataset to test the scalability of our TCDGE. One is to fix the number of snapshot graphs but augment the number of vertices in every snapshot graph by sampling vertices and edges in the other snapshot graphs as new vertices and edges of the current graph. This tests the scalability of the project gradient approach for updating W_t and the LASSO co-training. The experimental results are shown in Fig. 5a. As the number of vertices in the snapshot graph increases, the running time for updating W_t grows almost linearly. The running time for co-training LASSO and updating H becomes slightly longer, but still much slower than the growth rate of updating W_t.

The other synthesized dataset fixes the number of vertices in every snapshot graph but augments the number of snapshot graphs to test the scalability of learning the common latent space H. We divide the vertices of each existing snapshot graph into 5-folds based on the degree of vertices. We take a fold from each existing snapshot graph without duplication to synthesize a new snapshot graph. In Fig. 5b, the experimental results indicates that the running time of learning the common latent space grows linearly. Consequently, our TCDGE algorithm has very good scalability although the embedding model is complicated with high-order polynomials.

7 Conclusion

We generically model a dynamic graph as a sequence of snapshot graphs appended with ToE for every edge, which captures both the graph structure and temporal dependency among vertices. A time capturing dynamic graph embedding model is proposed to embed the global evolutionary patterns of the dynamic graph, which preserves every vertex's temporal linkage evolution as its moving trajectories within the inferred common latent representation space. The experimental results show that our method can achieve significant performance improvements over existing state-of-the-art approaches and it is very efficient and scalable.

Acknowledgments

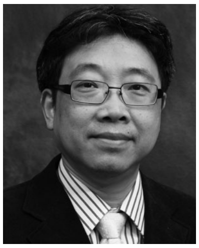
This work was supported in part by the Hong Kong RGC Collaborative Research Fund (CRF) under Grant C5026-18G, in part by the Hong Kong RGC Theme-based Research Scheme (TRS) under Grant T41-603/20-R, in part by the Fundamental Research Funds for the Central Universities under Grant NZ2020014, in part by the Guanddong Basic and Applied Basic Research Foundation under Grant 2021A1515012239, in part by the National Natural Science Foundation of China under Grant 62032020, and in part by the Hunan Science and Technology Planning Project under Grant 2019RS3019.

References

[1] B. Perozzi, R. Al-Rfou, and S. Skiena, "DeepWalk: Online learning of social representations," in *Proc. 20th ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2014, pp. 701–710.

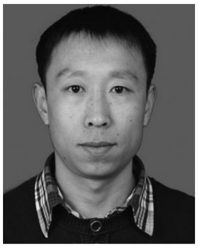
[2] L. Zhu, D. Guo, J. Yin, G. Ver Steeg, and A. Galstyan, "Scalable temporal latent space inference for link prediction in dynamic social networks," *IEEE Trans. Knowl. Data Eng.*, vol. 28, no. 10, pp. 2765–2777, Oct. 2016.
[3] W. Yu, C. C. Aggarwal, and W. Wang, "Temporally factorized network modeling for evolutionary network analysis," in *Proc. Tenth ACM Int. Conf. Web Search Data Mining*, 2017, pp. 455–464.
[4] L.-k. Zhou, Y. Yang, X. Ren, F. Wu, and Y. Zhuang, "Dynamic network embedding by modeling triadic closure process," in *Proc. AAAI Conf. Artif. Intell.*, 2018, pp. 571–578.
[5] D. Zhu, P. Cui, Z. Zhang, J. Pei, and W. Zhu, "High-order proximity preserved embedding for dynamic networks," *IEEE Trans. Knowl. Data Eng.*, vol. 30, no. 11, pp. 2134–2144, Nov. 2018.
[6] P. Goyal, N. Kamra, X. He, and Y. Liu, "DynGEM: Deep embedding method for dynamic graphs," 2018, *arXiv:1805.11273*.
[7] P. Goyal, S. R. Chhetri, and A. Canedo, "dyngraph2vec: Capturing network dynamics using dynamic graph representation learning," *Knowl. Based Syst.*, vol. 187, 2020, Art. no. 104816.
[8] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2017, pp. 1025–1035.
[9] Y. Xiong, Y. Zhang, H. Fu, W. Wang, Y. Zhu, and S. Y. Philip, "DynGraphGAN: Dynamic graph embedding via generative adversarial networks," in *Proc. Int. Conf. Database Syst. Adv. Appl.*, 2019, pp. 536–552.
[10] J. Ma, P. Cui, and W. Zhu, "DepthLGP: Learning embeddings of out-of-sample nodes in dynamic networks," in *Proc. AAAI Conf. Artif. Intell.*, 2018, 370–377.
[11] G. H. Nguyen, J. B. Lee, R. A. Rossi, N. K. Ahmed, E. Koh, and S. Kim, "Continuous-time dynamic network embeddings," in *Proc. 3rd Int. Workshop Learn. Representations Big Netw.*, 2018, pp. 969–976.
[12] Y. Zuo, G. Liu, H. Lin, J. Guo, X. Hu, and J. Wu, "Embedding temporal network via neighborhood formation," in *Proc. 24th ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2018, pp. 2857–2866.
[13] H. Peng *et al.*, "Dynamic network embedding via incremental skip-gram with negative sampling," *Sci. China Inf. Sci.*, vol. 63, no. 10, pp. 1–19, 2020.
[14] C. Yang, Z. Liu, D. Zhao, M. Sun, and E. Y. Chang, "Network representation learning with rich text information," in *Proc. 24th Int. Conf. Artif. Intell.*, 2015, pp. 2111–2117.
[15] D. Zhang, J. Yin, X. Zhu, and C. Zhang, "User profile preserving social network embedding," in *Proc. 24th Int. Conf. Artif. Intell.*, 2017, pp. 3378–3384.
[16] P. Goyal, H. Hosseinmardi, E. Ferrara, and A. Galstyan, "Capturing edge attributes via network embedding," 2018, *arXiv:1805.03280*.
[17] A. Grover and J. Leskovec, "node2vec: Scalable feature learning for networks," in *Proc. 22nd ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2016, pp. 855–864.
[18] S. Cao, W. Lu, and Q. Xu, "GraRep: Learning graph representations with global structural information," in *Proc. 24th ACM Int. Conf. Inf. Knowl. Manage.*, 2015, pp. 891–900.
[19] H. Chen, H. Yin, T. Chen, Q. V. H. Nguyen, W.-C. Peng, and X. Li, "Exploiting centrality information with graph convolutions for network representation learning," in *Proc. 35th IEEE Int. Conf. Data Eng.*, 2019, pp. 590–601.
[20] L. Xu, X. Wei, J. Cao, and P. S. Yu, "On exploring semantic meanings of links for embedding social networks," in *Proc. World Wide Web Conf.*, 2018, pp. 479–488.
[21] X. Wang, P. Cui, J. Wang, J. Pei, W. Zhu, and S. Yang, "Community preserving network embedding," in *Proc. AAAI Conf. Artif. Intell.*, 2017, pp. 203–209.
[22] J. Wang, J. Cao, W. Li, and S. Wang, "CANE: Community-aware network embedding via adversarial training," *Knowl. Inf. Syst.*, vol. 63, no. 2, pp. 411–438, 2021.
[23] L. Xu, X. Wei, J. Cao, and P. S. Yu, "Embedding of embedding: Joint embedding for coupled heterogeneous networks," in *Proc. Tenth ACM Int. Conf. Web Search Data Mining*, 2017, pp. 741–749.
[24] X. Sun *et al.*, "Heterogeneous hypergraph embedding for graph classification," in *Proc. 14th ACM Int. Conf. Web Search Data Mining*, 2021, pp. 725–733.
[25] J. Li, J. Zhu, and B. Zhang, "Discriminative deep random walk for network classification," in *Proc. 54th Annu. Meeting Assoc. Comput. Linguistics*, 2016, pp. 1004–1013.
[26] D. Zhang, J. Yin, X. Zhu, and C. Zhang, "Collective classification via discriminative matrix factorization on sparsely labeled networks," in *Proc. 25th ACM Int. Conf. Inf. Knowl. Manage.*, 2016, pp. 1563–1572.
[27] Z. Zhang, P. Cui, J. Pei, X. Wang, and W. Zhu, "Timers: Error-bounded SVD restart on dynamic networks," in *Proc. AAAI Conf. Artif. Intell.*, 2018, pp. 224–231.
[28] J. Wang, Y. Jin, G. Song, and X. Ma, "EPNE: Evolutionary pattern preserving network embedding," in *Proc. 24th Eur. Conf. Artif. Intell.*, 2020, pp. 1603–1610.
[29] D. Yang, S. Wang, C. Li, X. Zhang, and Z. Li, "From properties to links: Deep network embedding on incomplete graphs," in *Proc. ACM Conf. Inf. Knowl. Manage.*, 2017, pp. 367–376.
[30] A. Sankar, Y. Wu, L. Gou, W. Zhang, and H. Yang, "Dynamic graph representation learning via self-attention networks," 2018, *arXiv:1812.09430*.
[31] S. Greenland, M. A. Mansournia, and D. G. Altman, "Sparse data bias: A problem hiding in plain sight," *BMJ*, vol. 352, 2016, Art. no. 352:i1981.
[32] C.-J. Lin, "Projected gradient methods for nonnegative matrix factorization," *Neural Comput.*, vol. 19, pp. 2756–2779, 2007.
[33] D. Bertsekas, "On the Goldstein-Levitin-Polyak gradient projection method," *IEEE Trans. Autom. Control*, vol. 21, no. 2, pp. 174–184, Apr. 1976.
[34] 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.
[35] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Proc. Adv. Neural Inf. Process. Syst.*, 2013, pp. 3111–3119.
[36] P. Tseng, "Convergence of a block coordinate descent method for nondifferentiable minimization," *J. Optim. Theory Appl.*, vol. 109, no. 3, pp. 475–494, 2001.
[37] T. Opsahl and P. Panzarasa, "Clustering in weighted networks," *Social Netw.*, vol. 31, no. 2, pp. 155–163, 2009.
[38] S. Kumar, F. Spezzano, V. Subrahmanian, and C. Faloutsos, "Edge weight prediction in weighted signed networks," in *Proc. 16th IEEE Int. Conf. Data Mining*, 2016, pp. 221–230.

Yu Yang received the bachelor of computer science degree from the Xi'an University of Science and Technology in 2012 and the MEng degree in pattern recognition and intelligence system from Shenzhen University in 2015. He is currently working toward the PhD degree with the Department of Computing, The Hong Kong Polytechnic University. His research interests include spatio-temporal data analysis, representation learning, and image processing.

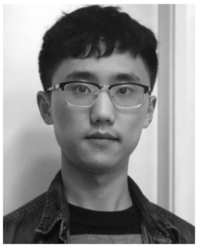


Jiannong Cao (Fellow, IEEE) received the MSc and PhD degrees in computer science from Washington State University, Pullman, WA, USA, in 1986 and 1990, respectively. He is currently the chair professor with the Department of Computing, The Hong Kong Polytechnic University, Hong Kong. His current research interests include parallel and distributed computing, mobile computing, and big data analytics. He was a member of the editorial boards of several international journals, a reviewer of international journals or conference proceedings, and an Organizing or Program Committee member for many international conferences.

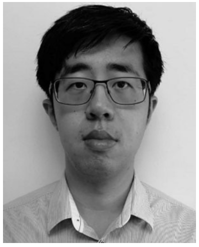
Milos Stojmenovic received the PhD degree in computer science from the University of Ottawa, Canada, in 2008. He is currently a professor with Singidunum University, Belgrade, Serbia. He has authored or coauthored more than 55 scientific contributions in books and peer-reviewed conferences and journals. His research interests include computer vision, machine learning and sensor networks, where he researches and develops software for the automated detection of objects in images, deep Learning for prediction of future events, and routing simulations in ad hoc sensor networks. He is currently on the editorial boards of two journals and has an H-index of 18. He is currently a visiting fellow with Hong Kong Polytechnic University Riga Technical University.



Senzhang Wang (Member, IEEE) received the MSc degree from Southeast University, Nanjing, China, in 2009 and the PhD degree in computer science from Beihang University, Beijing, China, in 2015. He is currently a professor with the School of Computer Science and Engineering, Central South University, Changsha. He has authored or coauthored more than ten papers on the top international journals and conferences such as Knowledge and Information Systems, ACM SIGKDD Conference on Knowledge Discovery and Data Mining, and AAAI Conference on Artificial Intelligence. His current research interests include data mining and social network analysis.



Yiran Cheng received the BSc degree from the Department of Computing from The Hong Kong Polytechnic University in 2020. From 2017 to 2019, he was an undergraduate research assistant with the Internet and Mobile Computing Laboratory, The Hong Kong Polytechnic University. His research interests include data mining and parallel computing.



Chun Lum received the BS degree from the Department of Computing from the Hong Kong Polytechnic University in 2018. His research interests include data analytics and machine learning.

Zhetao Li (Member, IEEE) received the BEng degree in electrical information engineering from Xiangtan University in 2002, the MEng degree in pattern recognition and intelligent system from Beihang University in 2005, and the PhD degree in computer application technology from Hunan University in 2010. He is currently a professor with the College of Computer Science, Xiangtan University. From December 2013 to December 2014, he was a postdoc in wireless network with Stony Brook University. His research interests include data analytics, wireless communication, and multimedia signal processing.