Triangle Completion Time Prediction using Time-conserving Embedding

Vachik S. Dave[1]✉ and Mohammad Al Hasan[1]

Indiana University Purdue University Indianapolis, IN, USA
vsdave@iupui.edu, alhasan@cs.iupui.edu

Abstract. A triangle is an important building block of social networks, so the study of triangle formation in a network is critical for better understanding of the dynamics of such networks. Existing works in this area mainly focus on triangle counting, or generating synthetic networks by matching the prevalence of triangles in real-life networks. While these efforts increase our understanding of triangle's role in a network, they have limited practical utility. In this work we undertake an interesting problem relating to triangle formation in a network, which is, to predict the time by which the third link of a triangle appears in a network. Since the third link completes a triangle, we name this task as **T**riangle **C**ompletion **T**ime **P**rediction (*TCTP*). Solution to *TCTP* problem is valuable for real-life link recommendation in social/e-commerce networks, also it provides vital information for dynamic network analysis and community generation study.
An efficient and robust framework (*GraNiTE*) is proposed for solving the *TCTP* problem. *GraNiTE* uses neural networks based approach for learning a representation vector of a triangle completing edge, which is a concatenation of two representation vectors: first one is learnt from graphlet based local topology around that edge and the second one is learnt from time-preserving embedding of the constituting vertices of that edge. A comparison of the proposed solution with several baseline methods shows that the mean absolute error (MAE) of the proposed method is at least one-forth of that of the best baseline method.

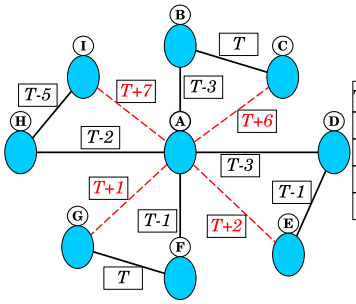
Keywords: Time prediction · embedding method · edge centric graphlets.

1 Introduction

It is a known fact that the prevalence of triangles in social networks is much higher than their prevalence in a random network. It is caused predominantly by the social phenomenon that friends of friends are typically friends themselves. A large number of triangles in social networks is also due to the "small-world network" property [23], which suggests that in an evolving social network, new links are formed between nodes that have short distance between themselves. Leskovec et al. [17] have found that depending on the kinds of networks, 30 to 60 percent of new links in a network are created between vertices which are only two-hops apart, i.e., each of these links is the third edge of a new triangle in the

network. High prevalence of triangles is also observed in directed networks, such as, trust networks, and follow-follower networks—social balance theory [1] can be attributed for such as observation.

There exist a number of works which study triangle statistics and their distribution in social networks. The majority among these works are focused on triangle counting; for a list of such works, see the survey [14] and the references therein. A few other works investigate how different network models perform in generating synthetic networks whose clustering coefficients match with those of real-life social networks [20]. Huang et al. [16] have analyzed the triad closure patterns and provided a graphical model to predict triad closing operation. Durak et al. [11] have studied the variance of degree values among the nodes forming a triangle in networks arising from different domains. These works are useful for discovering the local network context in which triangles appear, but they do not tell us whether local context can be used to predict when a triangle will appear. In this work, we fill this void by building a prediction model which uses local context of a network to predict *when a triangle will appear?* One of the similar time prediction problem in directed networks i.e. reciprocal link time prediction (RLTP), is studied and solved by V. Dave et al. [5, 6], where they used existing survival analysis models with socially motivated topological features. However, never designed features that incorporate time information. Also the proposed model utilizes the ordering and difference between the edge creation times in an innovative way, which significantly boost the accuracy of triangle completion time prediction.



Top recommendations for "User A"	
1	User G *(T+1)*
2	User E *(T+2)*
3	User C *(T+6)*
4	User I *(T+7)*

Fig. 1: Simple illustration of the utility of *TCTP* problem for providing improved friend recommendation. In this figure, user A is associated with 4 triangles, whose predicted completion times are noted as label on the triangles' final edges (red dotted lines). The link recommendation order for A at a time T, based on the earliest triangle completion time, is shown in the table on the right.

The knowledge of triangle completion time is practically useful. For instance, given that the majority of new links in a network complete a triangle, the knowledge—whether a link will complete a triangle in a short time—can be used to improve the performance of a link prediction model [15]. Specifically, by utilizing this knowledge, a link prediction model can assign a different prior probability of link formation when such links would complete a triangle in near

future. Besides, link creation time is more informative than a value denoting the chance of link formation. Say, an online social network platform wants to recommend a friend; it is much better for the platform if it recommends a member who is likely to accept the friend request in a day or two than recommending another who may accept the friend request after a week or few weeks (as illustrated in the Figure 1). In the e-commerce domain, a common product recommendation criterion is recommending an associated item (say, $item_2$) of an item (say, $item_1$) that a user u has already purchased. Considering a user-item network, in which $item_1 - item_2$ is a triangle's first edge, $item_1 - u$ is the triangle's second edge, the *TCTP* task can be used to determine the best time interval for recommending the user u the $item_2$, whose purchase will complete the $u - item_1 - item_2$ triangle. Given the high prevalence of triangle in real-life networks, the knowledge of triangle completion time can also improve the solution of various other network tasks that use triangles, such as, community structure generation [2], designing network generation models [17], and generating link recommendation [9].

In this work, we propose a novel framework called *GraNiTE* [1] for solving the *T*riangle *C*ompletion *T*ime *P*rediction (*TCTP*) task. *GraNiTE* is a network embedding based model, which first obtains latent representation vectors for the triangle completing edges; the vectors are then fed into a traditional regression model for predicting the time for the completion of a triangle. The main novelty of *GraNiTE* is the design of an edge representation vector learning model, which embeds edges with similar triangle completion time in close proximity in the latent space. Obtaining such embedding is a difficult task because the creation time of an edge depends on both local neighborhood around the edge and the time of the past activities of incident nodes. So, existing network embedding models [12, 7, 8] which utilize the neighborhood context of a node for learning its representation vector cannot solve the *TCTP* problem accurately. Likewise, existing network embedding models for dynamic networks [3, 26, 18, 25] are also ineffective for predicting the triangle completion time, because such embedding models dynamically encode network growth patterns, not the edge creation time.

To achieve the desired embedding, *GraNiTE* develops a novel supervised approach which uses local graphlet frequencies and the edge creation time. The local graphlet frequencies around an edge is used to obtain a part of the embedding vector, which yields a time-ordering embedding. Also, the edge creation time of a pair of edges is used for learning the remaining part of the embedding vector, which yields a time-preserving embedding. Combination of these two brings edges with similar triangle completion time in close proximity of each other in the embedding space. Both the vectors are learned by using a supervised deep learning setup. Through experiments [2] on five real-world datasets, we show that *GraNiTE* reduces the mean absolute error (MAE) to one-forth of the MAE value of the best competing method while solving the *TCTP* problem.

[1] *GraNiTE* is an anagram of the bold letters in **Gra**phlet and **N**ode based **Ti**me-conserving **E**mbedding.

[2] Code and data for the experiments are available at https://github.com/Vachik-Dave/GraNiTE_solving_triangle_completion_time_prediction

The rest of the paper is organized as follows. In Section 2, we define the *TCTP* problem formally. In Section 3, we show some interesting observation relating to triangle completion time on five real-world datasets. The *GraNiTE* framework is discussed in Section 4. In Section 5, we present experimental results which validate the effectiveness of our model over a collection of baseline models. Section 6 concludes the work.

2 Problem statement

2.1 Notations.

Throughout this paper, scalars are denoted by lowercase letters (e.g., n). Vectors are represented by boldface lowercase letters (e.g., $\mathbf{x}$). Bold uppercase letters (e.g., $\mathbf{X}$) denote matrices, the i^{th} row of a matrix $\mathbf{X}$ is denoted as $\mathbf{x}_i$ and j^{th} element of the vector $\mathbf{x}_i$ is represented as x_i^j. $\|\mathbf{X}\|_F$ is the Frobenius norm of matrix $\mathbf{X}$. Calligraphic uppercase letter (e.g., $\mathcal{X}$) is used to denote a set and $|\mathcal{X}|$ is used to denote the cardinality of the set $\mathcal{X}$.

2.2 Problem formulation.

Given, a time-stamped network $G = (\mathcal{V}, \mathcal{E}, \mathcal{T})$, where $\mathcal{V}$ is a set of vertices, $\mathcal{E}$ is a set of edges and $\mathcal{T}$ is a set of time-stamps. There also exists a mapping function $\tau : \mathcal{E} \rightarrow \mathcal{T}$, which maps each edge $e = (u, v) \in \mathcal{E}$ to a time-stamp value, $\tau(e) = t_{uv} \in \mathcal{T}$ denoting the creation time of the edge e. A triangle formed by the vertices $a, b, c \in \mathcal{V}$ and the edges $(a, b), (a, c), (b, c) \in \mathcal{E}$ is represented as Δ_{abc}. If exactly one of the three edges of a triangle is missing, we call it an open triple. Say, among the three edges above, (a, b) is missing, then the open triple is denoted as Λ_{ab}^{c}. We use Δ for the set of all triangles in a graph.

Given an open triple Λ_{uv}^{w}, the objective of *TCTP* is to predict the time-stamp (t_{uv}) of the missing edge (u, v), whose presence would have formed the triangle Δ_{uvw}. But, predicting the future edge creation time from training data is difficult as the time values of training data are from the past. So we make the prediction variable an invariant of the absolute time value by considering the interval time from a reference time value for each triangle, where reference time for a triangle is the time-stamp of the second edge in creation time order. For example, for the open triple Λ_{uv}^{w} the reference time is the latter of the time-stamps t_{wu}, and t_{wv}. Thus the interval time (target variable) that we want to predict is the time difference between t_{uv} and the reference time, which is $\max(t_{wu}, t_{wv})$. The interval time is denoted by I_{uvw}; mathematically, $I_{uvw} = t_{uv} - \max(t_{wu}, t_{wv})$. Then the predicted time for the missing edge creation is $t_{uv} = I_{uvw} + max(t_{wu}, t_{wv})$.

Predicting the interval time from a triangle specific reference time incurs a problem, when a single edge completes multiple (say k) open triples, we call such an edge a k-triangle edge. For such a k-triangle edge, ambiguity arises regarding the choice of triples (out of k triples), whose second edge should be used

for the reference time—for each of the reference time, a different prediction can be obtained. We solve this problem by using a weighted aggregation approach, a detailed discussion of this is available in Section 4.4 "Interval time prediction".

3 Dataset study

The problem of predicting triangle completion time has not been addressed in any earlier works, so before embarking on the discussion of our prediction method, we like to present some observations on the triangle completion time in five real-world datasets.

Among these datasets, BitcoinOTC [3] is a trust network of Bitcoin users, Facebook [4] and Digg-friend [4] are online friendship networks, Epinion [4] is an online trust network and DBLP [4] is a co-authorship network. Overall information, such as the number of vertices ($|\mathcal{V}|$), edges ($|\mathcal{E}|$), time-stamps ($|\mathcal{T}|$) and triangles ($|\Delta|$) for these datasets are provided in table 1. Note that, we preprocess these graphs to remove duplicate edges and edges without valid time-stamps, which leads to removal of disconnected nodes.

Table 1: Statistics of datasets (* $\mathcal{T}$ in years for DBLP)

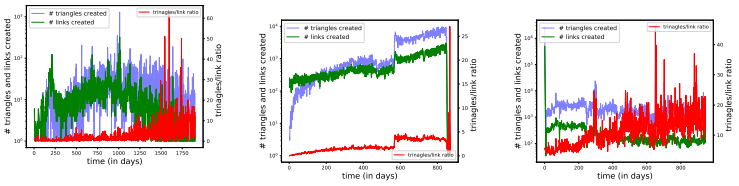
Datasets	$\lvert\mathcal{V}\rvert$	$\lvert\mathcal{E}\rvert$	$\lvert\mathcal{T}\rvert$ (days)	$\lvert\Delta\rvert$
BitcoinOTC	5,881	35,592	1,903	33,493
Facebook	61,096	614,797	869	1,756,259
Epinion	131,580	711,210	944	4,910,076
DBLP	1,240,921	5,068,544	23*	11,552,002
Digg-friend	279,374	1,546,540	1,432	14,224,759

3.1 Study of triangle generation rate.

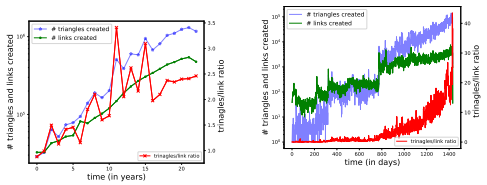
As network grows over time, so do the number of edges and the number of triangles. In this study our objective is to determine whether there is a temporal correlation between the growth of edges and the growth of triangles in a network. To observe this behavior, we plot the number of new edges (green line) and the number of new triangles (blue line) (y-axis) over different time values (x-axis); Figure 2 depicts five plots, one for each dataset. The ratio of newly created triangle count to newly created link count is also shown (red line).

Trend in the plots is similar; as time passes, the number of triangles and the number of links created at each time stamp steadily increase (except Epinion dataset), which represents the fact that the network is growing. Interestingly, triangle to link ratio also increases with time. This happens because as a network gets more dense, the probability that a new edge will complete one or more

[3] http://snap.stanford.edu/data/
[4] http://konect.uni-koblenz.de/

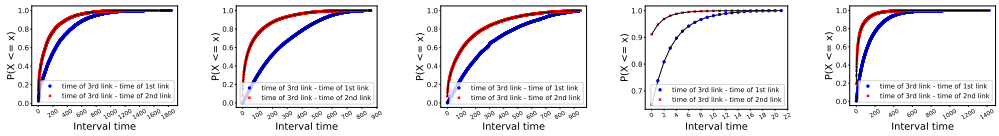


(a) BitcoinOTC dataset (b) Facebook dataset (c) Epinion dataset



(d) DBLP dataset (e) Digg-friend dataset

Fig. 2: Frequency of new edges (green line) and new triangles (blue line) created over time. Ratio of newly created triangle to the newly created link frequency is shown in red line. Y-axis labels on the left show frequency of triangles and link, and the y-axis labels on right show the triangle to link ratio value.



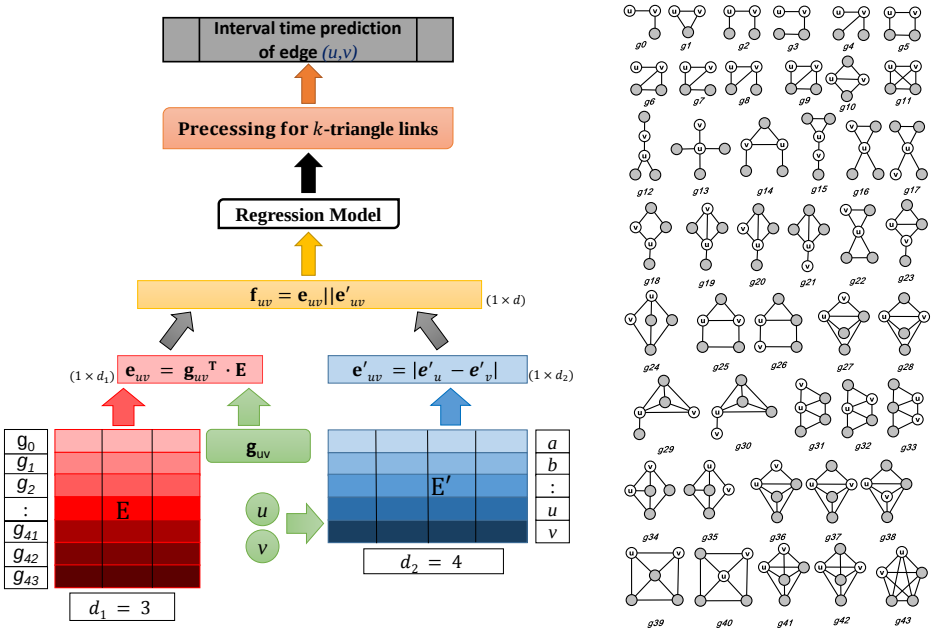
(a) BitcoinOTC dataset (b) Facebook dataset (c) Epinion dataset (d) DBLP dataset (e) Digg-friend dataset

Fig. 3: Plots of cumulative distribution function (CDF) for interval times

triangles increases. This trend is more pronounced in Digg-friend and DBLP networks. Especially, in Digg-friend network, each link contributes around 20 triangles during the last few time-stamps. On the other hand, for BitcoinOTC, Facebook and Epinion datasets, the triangle to link ratio increases slowly. For Facebook dataset, after slow and steady increase, we observe a sudden hike in all three values around day 570. After investigation, we discovered that, it is a consequence of a newly introduced recommendation feature by "Facebook" in 2008. This feature, exploits common friends information which leads to create many links completing multiple open triples.

3.2 Interval time analysis.

For solving *TCTP*, we predict interval time between the triangle completing edge and the second edge in time order. In this study, we investigate the distribution



(a) Interval time prediction for edge (u, v) using Proposed *GraNiTE*.

(b) Local graphlets for given edge (u, v)

Fig. 4: Proposed *GraNiTE* and local graphlets

of the interval time by plotting the cumulative distribution function (CDF) of the interval time for all the datasets (blue lines in the plots in Figure 3). For comparison, these plots also show the interval time between triangle completing edge and the first edge (red lines).

From Figure 3, we observe that for all real-world datasets the interval time between the third link and the second link creation follows a distribution from exponential family; which means most of the third links are created very soon after the generation of the second link. This observation agrees with the social balance theory [1]. As per this theory, triangles and individual links are balanced structures while an open triple is an imbalanced structure. All real-world networks (such as social networks) try to create a balanced structure by closing an open triple as soon as possible; which is validated in Figure 3 as the red curve quickly reaches to 1.0 compared to the ascent of the blue curve.

4 *GraNiTE* Framework

GraNiTE framework first obtains a latent representation vector for an edge such that edges with similar interval time have latent vectors which are in close proximity. Such a vector for an edge is learned in a supervised fashion via two

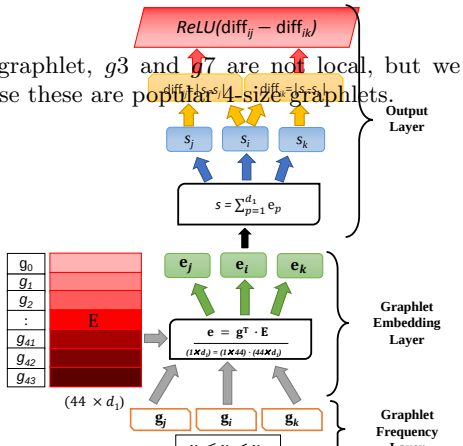
kinds of edge embeddings: first, a graphlet-based edge embedding, which embeds the local graphlets into embedding space such that their embedding vectors capture the information of edge ordering based on the interval time. So, we call the edge representation obtained from the graphlet-based embedding method *time-ordering embedding*. Second, a node-based edge embedding that learns node embedding such that proximity of a pair of nodes preserves the interval time of the triangle completing edge. We call the node-based edge embedding *time-preserving embedding*. Concatenation of these two vectors gives the final edge representation vector, which is used to predict a unique creation time for a given edge.

The overall architecture of *GraNiTE* is shown in Figure 4a. Here nodes u, v and local graphlet frequency vector of edge (u, v) are inputs to the *GraNiTE*. $\mathbf{E}$ and $\mathbf{E}'$ are graphlet embedding and node embedding matrices, respectively. For an edge (u, v), corresponding time-ordering embedding $\mathbf{e}_{uv} \in \mathbb{R}^{d_1}$ and time-preserving embedding $\mathbf{e}'_{uv} \in \mathbb{R}^{d_2}$ are concatenated to generate final feature vector $\mathbf{f}_{uv} = \mathbf{e}_{uv}||\mathbf{e}'_{uv} \in \mathbb{R}^{d(=d_1+d_2)}$. This feature vector $\mathbf{f}_{uv}$ is fed to a regression model that predicts interval time for (u, v). Lastly, we process the regression model output to return a unique interval time for (u, v), in case this edge completes multiple triangles. In the following subsections, we describe graphlet-based time-ordering embedding and node-based time-preserving embedding.

4.1 Graphlet-based Time-ordering Embedding.

In a real world network, local neighborhood of a vertex is highly influential for a new link created at that vertex. In existing works, local neighborhood of a vertex is captured through a collection of random walks originating from that vertex [19], or by first-level and second level neighbors of that node [21]. For finding local neighborhood around an edge we can aggregate the local neighborhood of its incident vertices. A better way to capture edge neighborhood is to use local graphlets (up to a given size), which provide comprehensive information of local neighborhood of an edge [4]. For an edge (u, v), a graphical structure that includes nodes u, v and a subset of direct neighbors of u and/or v is called a local graphlet for the edge (u, v). Then, a vector containing the frequencies of (u, v)'s local graphlets is a quantitative measure of the local neighborhood of this edge. In Figure 4b, we show all local graphlets of an edge (u, v) up to size-5, which we use in our time-ordering embedding task. To calculate frequencies of these local graphlets, we use E-CLoG algorithm [4], which is very fast and parallelizable algorithm because graphlet counting process is independent for each edge. After counting frequencies of all 44 graphlets [5], we generate normalized graphlet frequency (NGF), which is an input to our supervised embedding model.

Graphlet frequencies mimic edge features which are highly informative to capture the local neighborhood of



[5] Note that, by strict definition of local graphlet, $g3$ and $g7$ are not local, but we compute their frequencies anyway because these are popular 4-size graphlets.

an edge. For instance, the frequency of g_1 is the common neighbor count between u and v, frequency of g_5 is the number of 2-length paths, and frequency of g_{43} is the number of five-size cliques involving both u and v. These features can be used for predicting link probability between the vertex pair u and v. However, these features are not much useful when predicting the interval time of an edge. So, we learn embedding vector for each of the local graphlets, such that edge representation built from these vectors captures the ordering among the edges based on their interval times, so that they are effective for solving the *TCTP* problem. In the following subsection graphlet embedding model is discussed.

Learning Model. The embedding model has three layers: graphlet frequency layer, graphlet embedding layer and output layer. As shown in the Figure 5, graphlet frequency layer takes input, graphlet embedding layer calculates edge embedding for the given set of edges using graphlet embedding matrix and graphlet frequencies, and the output layer calculates our loss function for the embedding, which we optimize by using adaptive gradient descent. The loss function implements the time-ordering objective. Given, three triangle completing edges i, j and k and their interval times, y_i, y_j, and y_k, such that $y_i \leq y_j \leq y_k$, our loss function enforces that the distance between the edge representation vectors of i and j is smaller than the distance between the edge representation vectors of i and k. Thus, the edges which have similar interval time are being brought in a close proximity in the embedding space.

Training data for this learning model is the normalized graphlet frequencies (NGF) of all training instances (triangle completing edges with known interval values), which are represented as $\mathbf{G} \in \mathbb{R}^{m \times g_n}$, where m is the number of training instances and g_n is equal to 44 representing different types of local graphlets. Each row of matrix $\mathbf{G}$ is an NGF for a single training instance i.e. if i^{th} element corresponds to the edge (u, v), $\mathbf{g}_i$ $(= \mathbf{g}_{uv})$ is its normalized graphlet frequency. The target values (interval time) of m training instances are represented as vector $\mathbf{y} \in \mathbb{R}^m$. Now, the layers of the embedding model (Figure 5) are explained below:

Graphlet frequency layer: In input layer we feed triples of three sampled data instances i, j and k, such that $y_i \leq y_j \leq y_k$ with their NGF i.e. $\mathbf{g}_i$ $\mathbf{g}_j$, and $\mathbf{g}_k$.

Graphlet embedding layer: This model learns embedding vectors for each local graphlets, represented with the embedding matrix $\mathbf{E} \in \mathbb{R}^{g_n \times d_1}$, where d_1 is the (user-defined) embedding dimension. For any data instance i in training data $\mathbf{G}$, corresponding time-ordering edge representation $\mathbf{e}_i \in \mathbb{R}^{d_1}$ is obtained by vector to matrix multiplication i.e. $\mathbf{e}_i = \mathbf{g}_i^T \cdot \mathbf{E}$. In the embedding layer, for input data instances i, j and k, we calculate three time-ordering embedding vectors $\mathbf{e}_i$, $\mathbf{e}_j$ and $\mathbf{e}_k$ using this vector-matrix multiplication.

Output layer: This layer implements our loss function. For this, first we calculate the score of each edge representation using vector addition i.e. for $\mathbf{e}_i$ the score is $s_i = \Sigma_{p=1}^{d_1} e_i^p$. After that, we pass the score difference between instances i and j (diff_{ij}) and the score difference between i and k (diff_{ik}) to an activation function. The activation function in this layer is ReLU, whose output we minimize. The objective function after regularizing the graphlet embedding matrix is as below:

$$\mathscr{O}_g = \underset{\mathbf{E}}{\boldsymbol{min}} \sum_{\forall (i,j,k) \in T_{ijk}} ReLU(\text{diff}_{ij} - \text{diff}_{ik}) + \lambda_g \cdot \|\mathbf{E}\|_F^2 \tag{1}$$

where, $\text{diff}_{ij} = |s_i - s_j|$, λ_g is a regularization constant and T_{ijk} is a training batch of three qualified edge instances from training data.

4.2 Time-preserving Node Embedding.

This embedding method learns a set of node representation vectors such that the interval time of an edge is proportional to the l_1 norm of incident node vectors. If an edge has higher interval time, the incident node vectors are pushed farther, if the edge have short interval time, the incident node vectors are close to each other in latent space. Thus, by taking the l_1 norm of node-pairs, we can obtain an embedding vector of an edge which is interval time-preserving and is useful for solving the *TCTP* problem. As depicted in the Figure 6, this embedding method is composed of three layers: input layer, node & edge embedding layer, and time preserving output layer. Functionality of each layer is discussed below:

Input layer: For this embedding method, input includes two edges, say (u, v) and (x, y) with their interval times, y_{uv} and y_{xy}. The selection of these two edges is based on the criterion that $y_{uv} > y_{xy}$.

Node & edge embedding layer: In this layer, we learn embedding matrix $\mathbf{E}' \in \mathbb{R}^{|\mathcal{V}| \times d_2}$, where d_2 is (user-defined) embedding dimension. From the embedding matrix $\mathbf{E}'$, we find

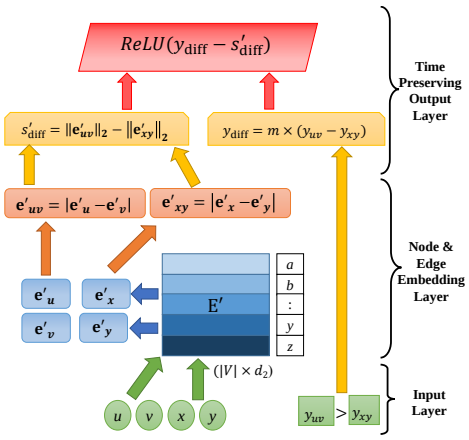


Fig. 6: Learning of the node embedding matrix using two edges (node-pairs).

node embedding for a set of 4 nodes incident to the edges (u, v) and (x, y). For any node u, node embedding vector is $\mathbf{e}'_u \in \mathbb{R}^{d_2}$ i.e. u^{th} element of matrix $\mathbf{E}'$. From the node embedding vectors $\mathbf{e}'_u$ and $\mathbf{e}'_v$, we calculate corresponding time-preserving edge embedding vector for (u, v). The time-preserving edge embedding is defined as l_1-distance between the node embedding vectors, i.e. $\mathbf{e}'_{uv} = |\mathbf{e}'_u - \mathbf{e}'_v| \in \mathbb{R}^{d_2}$.

Time-preserving output layer: The objective of this embedding is to preserve the interval time information into embedding matrix, such that time-preserving edge vectors are proportional to their interval time. For that, we calculate an edge score using l_2-norm of an edge embedding, i.e. (u, v) edge score $s'_{uv} = \|\mathbf{e}'_{uv}\|_2$. We design the loss function such that edge score difference $s'_{\text{diff}} = s'_{uv} - s'_{xy}$ between edges (u, v) and (x, y) is proportional to their interval time difference $y_{uv} - y_{xy}$. The objective function of the embedding is

$$\mathscr{O}_n = \min_{\mathbf{E}'} \sum_{\forall (u,v),(x,y) \in T_{uv,xy}} ReLU(y_{\text{diff}} - s'_{\text{diff}}) + \lambda_n \cdot \|\mathbf{E}'\|_F^2 \tag{2}$$

where, $y_{\text{diff}} = m \times (y_{uv} - y_{xy})$, λ_n is a regularization constant, $T_{uv,xy}$ is a training batch of edge pairs, and m is a scale factor.

4.3 Model inference and optimization.

We use mini-batch adaptive gradient decent (AdaGrad) to optimize the objective functions (Equations 1 and 2) of both embedding methods. Mini-batch AdaGrad is a modified mini-batch gradient decent approach, where learning rate of each dimension is different based on gradient values of all previous iterations [10]. This independent adaption of learning rate for each dimension is especially well suited for graphlet embedding method as graphlet frequency vector is mostly a sparse vector which generates sparse edge embedding vectors. For time-preserving node embedding, independent learning rate helps to learn the embedding vectors more efficiently such that two node can maintain its proximity in embedding space proportional to interval time.

For mini-batch AdaGrad, first we generate training batch, say T, from training instances. For each mini-batch, we uniformly choose training instances that satisfy the desired constrains: for graphlet embedding, a training instance consists of three edges i, j and k, for which $y_i \leq y_j \leq y_k$ and for time-preserving node embedding, a training instance is an edge pair, $i = (u, v)$ and $j = (x, y)$, such that, $y_i \leq y_j$. During an iteration, AdaGrad updates each embedding vector, say $\mathbf{e}$, corresponding to all samples from training batch using following equation:

$$e_i^{t+1} = e_i^t - \alpha_i^t \times \frac{\partial \mathscr{O}}{\partial e_i^t} \tag{3}$$

where, e_i^t is an i^{th} element of vector $\mathbf{e}$ at iteration t. Here we can see that at each iteration t, AdaGrad updates embedding vectors using different learning rates α_i^t for each dimension.

For time complexity analysis, given a training batch T, the total cost of calculating gradients of objective functions ($\mathscr{O}_g$ and $\mathscr{O}_n$) depends on the dimension of embedding vector i.e. $\Theta(d_i)$, $d_i \in \{d_1, d_2\}$. Similarly, calculating learning rate and updating embedding vector also costs $\Theta(d_i)$. In graphlet embedding, we need to perform vector to matrix multiplication, which costs $\Theta(44 \times d_1)$. Hence, total cost of the both embedding methods is $\Theta(44 \times d_1 + d_2) = \Theta(d_1 + d_2)$. As time complexity is linear to embedding dimensions, both embedding methods are very fast in learning embedding vectors even for large networks.

4.4 Interval time prediction.

We learn both time-ordering graphlet embedding matrix and time-preserving node embedding matrix from training instances. We generate edge representation for test instances from these embedding matrices, as shown in Figure 4a. This edge representation is fed to a traditional regression model (we have used Support Vector Regression) which predicts an interval time. However, predicting the interval time of a k-triangle link poses a challenge, as any regression model predicts multiple (k) creation times for such an edge. The simplest approach to overcome this issue is to assign the mean of k predictions as the final predicted value for the k-triangle link. But, as we know mean is highly sensitive to outliers especially for the small number of samples (mostly $k \in [2, 20]$), so using a mean value does not yield the best result. From the discussion in Section 3.2 , we know that triangle interval time follows exponential distribution. Hence we use exponential decay $W(I_{uvw}) = w_0 \cdot \exp(-\lambda \cdot I_{uvw})$ as a weight of each prediction, where λ is a decay constant and w_0 is an initial value. We calculate weighted mean which serves as a final prediction value for a k-triangle link.

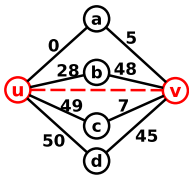


Fig. 7: (u, v) as 4-triangle link

In Figure 7, we show a toy graph with creation time of each link and (u, v) is a 4-triangle link. Let's assume our model predicts 4 interval times $(40, 3, 1, 1)$ corresponding to four open triples $(\Lambda_{uv}^a, \Lambda_{uv}^b, \Lambda_{uv}^c, \Lambda_{uv}^d)$ respectively. Hence, we have 4 predicted creation times i.e. $(5 + 40 = 45, 51, 50, 51)$ for link (u, v). So, the final prediction for the edge (u, v) is calculated by using the equation below:

$$\widehat{t_{uv}} = \frac{W(40)\times 45 + W(3)\times 51 + W(1)\times 50 + W(1)\times 51}{W(40)+W(3)+W(1)+W(1)}$$

5 Experiments and results

We conduct experiments to show the superior performance of the proposed *GraNiTE* in solving the *TCTP* problem. No existing works solve the *TCTP* problem, so we build baseline methods from two

approaches described as below:

The first approach uses features generated directly from the network topology.

1. **Topo. Feat.** (Topological features) This method uses traditional topological features such as common neighbor count, Jaccard coefficient, preferential attachment, adamic-adar, Katz measure with five different β values $\{0.1, 0.05, 0.01, 0.005, 0.001\}$. These features are well-known for solving the link prediction task [15]. We generate topological features for an edge (last edge of triangle) from the snapshot of the network when the second link of the triangle appears; triangle interval time is also computed from that temporal snapshot.
2. **Graphlet Feat.** In this method we use local graphlet frequencies of an edge (last edge of triangle) as a feature set for the time prediction task. These graphlet frequencies are also calculated from the temporal snapshot of the network as mentioned previously in Topo. Feat.

The second approach uses well known network embedding approaches.

3. **LINE** [21]: LINE embeds the network into a latent space by leveraging both first-order and second-order proximity of each node.
4. **Node2vec** [12]: Node2vec utilizes Skip-Gram based language model to analyze the truncated biased random walks on the graph.
5. **GraphSAGE** [13]: It presents an inductive representation learning framework that learns a function and generates embeddings by sampling and aggregating features from a node's local neighborhood.
6. **AROPE** [24]: AROPE is a matrix decomposition based embedding approach, which preserves different higher-order proximity for different input graphs and it provides global optimal solution for a given order.
7. **VERSE** [22]: It is a versatile node embedding method that preserves specific node similarity measure(s) and also captures global structural information.

5.1 Experiment settings.

For this experiment, we divide the time-stamps of each dataset into three chronologically ordered partitions with the assumption that initial partition is network growing period, which spans from the beginning up to 50% of total time-stamps. The second partition, which spans from 50% to 70% of the total time-stamps, is the train period, and finally, from 70% till the end is the test period. We select the edges completing triangles during the train period as training instances and the edges completing triangles during the test period as test instances. We also retain 5% of test instances for parameter tuning. Note that, this experiment setting is not suitable for dynamic network embedding methods, so we cannot compare with them.

There are a few user defined parameters in the proposed *GraNiTE*. For both embedding approaches, we fix the embedding dimensions as 50, i.e. $d_1 = d_2 = 50$. Hence, final embedding dimension is $d = 100$ as discussed in Section 4 "*GraNiTE*

Framework". Similarly, regularization rates for both embedding methods are set as $\lambda_g = \lambda_n = 1e-5$. Initial learning rate for AdaGrad optimization is set as 0.1. The training batch size is 100 and the number of epochs is set to 50. For time preserving node embedding, the scale factor is set to 0.01 i.e. $m = 0.01$. Additionally, for predicting time of k-triangle links, decay constant (λ) and initial weight (w_0) are set to 1.0 for calculating exponential decay weights. Lastly, we use support vector regression (SVR) with linear kernel and penalty $C = 1.0$ as a regression method for *GraNiTE* and for all competing methods. For fair comparison, SVR is identically configured for all methods.

For all competing embedding methods the embedding dimensions are set as 100, same size of our feature vector ($d = 100$). We grid search the different tuning parameters to find the best performance of these embedding methods. We select learning rate from set $\{0.0001, 0.001, 0.01, 0.1\}$ for all methods. For Node2vec, we select walk bias factors p and q from $\{0.1, 0.5, 1.0\}$ and number of walks per node is selected from $\{5, 10, 15, 20\}$. For AROPE, the order of proximity is selected from set $\{1, 2, 3, 4, 5\}$. For VERSE, we select personalized pagerank parameter α from set $\{0.1, 0.5, 0.9\}$.

5.2 Comparison results.

We evaluate the models using mean absolute error (MAE) over two groups of interval times: 1-month ($\leq$ 30 days) and 2-months (31 to 60 days) for all datasets, except DBLP, for which the two intervals are 0-2 years and 3-7 years. Instances that have higher than 60 days of interval time are outlier instances, hence they are excluded. Besides, for real-life social network applications, predicting an interval value beyond two months is probably not very interesting. Within 60 days, we show results in two groups: 1-month, and 2-month, because some of the competing methods work well for one group, but not the other.

Comparison results for all five datasets are shown in Table 2, where each column represents a prediction method. Rows are grouped into five, one for each dataset; each dataset group has three rows: small interval ($\leq 30d$), large interval (30-60d) and Average (Avg.) over these two intervals. Results of our proposed method (*GraNiTE*) is shown in the last column; besides MAE, in this column we also show the percentage of improvement of *GraNiTE* over the best of the competing methods(underlined). The best results in each row is shown in bold font.

We can observe from the table that the proposed *GraNiTE* performs the best for all the datasets considering the average. The improvements over the competing methods, at a minimum, 19.34% for the BitcoinOTC dataset, and, to the maximum, 76.8% for the Epinion dataset. If we consider short and long intervals ($\leq 30d$ and 30-60d) independently, *GraNiTE* performs the best in all datasets, except BitcoinOTC dataset. However, notice that for BitcoinOTC dataset, although Node2vec performs the best for large interval times, for small interval times its performance is extremely poor (almost thrice MAE compared to *GraNiTE*). Similarly, LINE performs the best for small interval times and incurs huge

Table 2: Comparison experiment results using MAE for interval times in 1^{st} (≤ 30 *days*) and 2^{nd}-month (31-60 *days*). [for DBLP dataset: 0-2 years and 3-7 years]. For *GraNiTE*, % improvement over the best competing method (underlined) is shown in brackets.

Dataset		Topo. Feat.	Graphlet Feat.	LINE	Node2vec	GraphSAGE	AROPE	VERSE	*GraNiTE*
Bitcoin-OTC	$\leq 30d$	17.22	17.7	**8.86**	26.68	11.99	28.62	25.81	9.08 (−2.48%)
	31-60d	21.92	18.29	34.03	**16.55**	28.84	21.59	20.56	19.95 (−20.54%)
	Avg.	19.57	17.995	21.445	21.615	20.415	25.105	23.185	**14.515** (19.34%)
Facebook	$\leq 30d$	7.78	7.93	8.36	7.95	8.37	7.93	7.98	**5.64** (27.51%)
	31-60d	32.04	30.9	31.98	32.87	31.96	32.55	32.73	**13.65** (55.83%)
	Avg.	19.91	19.415	20.17	20.41	20.165	20.24	20.355	**9.645** (50.32%)
Epinion	$\leq 30d$	15.88	14.31	12.52	17.09	13.79	14.3	19.85	**3.28** (73.8%)
	31-60d	22.02	24.82	25.18	20.17	23.45	23.22	17.9	**5.36** (70.06%)
	Avg.	18.95	19.565	18.85	18.63	18.62	18.76	18.875	**4.32** (76.8%)
DBLP	$\leq 30d$	0.526	0.525	0.527	0.527	0.526	0.526	0.5267	**0.449** (14.48%)
	31-60d	3.623	3.618	3.624	3.623	3.623	3.624	3.623	**0.969** (73.22%)
	Avg.	2.0745	2.0715	2.0755	2.075	2.0745	2.075	2.0748	**0.709** (65.77%)
Digg-friends	$\leq 30d$	6.75	6.25	6.03	7.73	5.95	7.37	6.95	**2.13** (64.2%)
	31-60d	41.06	37.34	38.77	32.66	38.85	34.34	34.75	**9.76** (70.12%)
	Avg.	23.905	21.795	22.4	20.195	22.4	20.855	20.85	**5.945** (70.56%)

error for large interval times. Only *GraNiTE* shows consistently good results for both small and large interval ranges over all the datasets.

Another observation is that, for all datasets, results of large interval times (31-60 days) is worse than the results of small interval time ($\leq$ 30 days). For competing methods, these values are sometimes very poor that it is meaningless for practical use. For instance, for Epinion, each of the competing methods have an MAE around 20 or more for large interval, whereas *GraNiTE* has an MAE value of 5.36 only. Likewise, for Digg-friends, each of the competing methods have an MAE more than 32, but *GraNiTE*'s average MAE is merely 5.95. Overall, for both intervals over all the datasets, *GraNiTE*shows significantly (t-test with p-value $\ll 0.01$) lower MAE than the second best method. The main reason for poor performance of competing methods is that, those methods can capture the local and/or global structural information of nodes/edges but fail to capture temporal information. While for *GraNiTE*, the graphlet embedding method is able to translate the patterns of local neighborhood into time-ordering edge vector; at the same time, time preserving node embedding method is able to capture the interval time information into node embedding vector. Both of the features help to enhance the performance of *GraNiTE*.

6 Conclusion

In this paper, we propose a novel problem of triangle completion time prediction (*TCTP*) and provide an effective and robust framework *GraNiTE* to solve this

problem by using graphlet based time-ordering embedding and time-preserving node embedding methods. Through experiments on five real-world datasets, we show the superiority of our proposed method compared to baseline methods which use known graph topological features, graphlet frequency features or popular and state-of-art network embedding approaches. To the best of our knowledge, we are the first to formulate the *TCTP* problem and to propose a novel framework for solving this problem.

References

1. Antal, T., Krapivsky, P., Redner, S.: Social balance on networks: The dynamics of friendship and enmity. Physica D: Nonlinear Phenomena **224**, 130 – 136 (2006)
2. Bianconi, G., Darst, R.K., Iacovacci, J., Fortunato, S.: Triadic closure as a basic generating mechanism of communities in complex networks. Phys. Rev. E **90**(4) (Oct 2014)
3. Bonner, S., Brennan, J., Kureshi, I., Theodoropoulos, G., McGough, A.S., Obara, B.: Temporal graph offset reconstruction: Towards temporally robust graph representation learning. In: IEEE Big Data. pp. 3737–3746 (2018)
4. Dave, V.S., Ahmed, N.K., Hasan, M.A.: E-clog: Counting edge-centric local graphlets. In: IEEE Intl. Conf. on Big Data. pp. 586–595 (Dec 2017)
5. Dave, V.S., Al Hasan, M., Reddy, C.K.: How fast will you get a response? predicting interval time for reciprocal link creation. In: Eleventh International AAAI Conference on Web and Social Media. ICWSM '17 (2017)
6. Dave, V.S., Hasan, M.A., Zhang, B., Reddy, C.K.: Predicting interval time for reciprocal link creation using survival analysis. Social Network Analysis and Mining **8** (Mar 2018)
7. Dave, V.S., Zhang, B., Al Hasan, M., AlJadda, K., Korayem, M.: A combined representation learning approach for better job and skill recommendation. In: ACM International Conference on Information and Knowledge Management. pp. 1997–2005. CIKM '18 (2018)
8. Dave, V.S., Zhang, B., Chen, P.Y., Hasan, M.A.: Neural-brane: Neural bayesian personalized ranking for attributed network embedding. Data Science and Engineering (Jun 2019)
9. Dong, Y., Tang, J., Wu, S., Tian, J., Chawla, N.V., Rao, J., Cao, H.: Link prediction and recommendation across heterogeneous social networks. In: IEEE Intl. Conf. on Data Mining. pp. 181–190 (2012)
10. Duchi, J., Hazan, E., Singer, Y.: Adaptive subgradient methods for online learning and stochastic optimization. J. of Machine Learning Research **12**(Jul), 2121–2159 (2011)
11. Durak, N., Pinar, A., Kolda, T.G., Seshadhri, C.: Degree relations of triangles in real-world networks and graph models. In: ACM Intl. Conf. on Information and Knowledge Management. pp. 1712–1716 (2012)
12. Grover, A., Leskovec, J.: Node2vec: Scalable feature learning for networks. pp. 855–864. KDD '16 (2016)
13. Hamilton, W., Ying, Z., Leskovec, J.: Inductive representation learning on large graphs. In: Advances in Neural Information Processing Systems (NIPS) 30, pp. 1024–1034 (2017)
14. Hasan, M.A., Dave, V.: Triangle counting in large networks: a review. Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery **8**(2) (2018)

15. Hasan, M.A., Zaki, M.J.: A Survey of Link Prediction in Social Networks, pp. 243–275. Springer US, Boston, MA (2011)
16. Huang, H., Tang, J., Liu, L., Luo, J., Fu, X.: Triadic closure pattern analysis and prediction in social networks. IEEE Transactions on Knowledge and Data Engineering **27**(12), 3374–3389 (Dec 2015)
17. Leskovec, J., Backstrom, L., Kumar, R., Tomkins, A.: Microscopic evolution of social networks. pp. 462–470. KDD '08 (2008)
18. Nguyen, G.H., Lee, J.B., Rossi, R.A., Ahmed, N.K., Koh, E., Kim, S.: Continuous-time dynamic network embeddings. In: Companion of the The Web Conference 2018. pp. 969–976 (2018)
19. Perozzi, B., Al-Rfou, R., Skiena, S.: Deepwalk: Online learning of social representations. pp. 701–710. KDD '14 (2014)
20. Sala, A., Cao, L., Wilson, C., Zablit, R., Zheng, H., Zhao, B.Y.: Measurement-calibrated graph models for social network experiments. In: ACM Intl. Conf. on World Wide Web. pp. 861–870 (2010)
21. Tang, J., Qu, M., Wang, M., Zhang, M., Yan, J., Mei, Q.: Line: Large-scale information network embedding. In: International Conference on World Wide Web. pp. 1067–1077 (2015)
22. Tsitsulin, A., Mottin, D., Karras, P., Müller, E.: Verse: Versatile graph embeddings from similarity measures. In: The World Wide Web Conference. pp. 539–548 (2018)
23. Watts, D.J., Strogatz, S.H.: Collective dynamics of small-world networks. nature **393**(6684), 440 (1998)
24. Zhang, Z., Cui, P., Wang, X., Pei, J., Yao, X., Zhu, W.: Arbitrary-order proximity preserved network embedding. pp. 2778–2786. KDD '18 (2018)
25. Zhou, L., Yang, Y., Ren, X., Wu, F., Zhuang, Y.: Dynamic network embedding by modeling triadic closure process. In: Conference on Artificial Intelligence (2018)
26. Zuo, Y., Liu, G., Lin, H., Guo, J., Hu, X., Wu, J.: Embedding temporal network via neighborhood formation. In: ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. pp. 2857–2866 (2018)