2023 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)

FODGE - Fast Online Dynamic Graph Embedding

1st Shoval Frydman
Department of Mathematics, Bar Ilan University
Ramat Gan, Israel,123shovalf@gmail.com

2nd Yoram Louzoun
Department of Mathematics, Bar Ilan University
Ramat Gan, Israel,louzouy@math.biu.ac.il

***Abstract*—Graph embedding algorithms (GEA) project each vertex in a graph to a real-valued vector. Dynamic GEA (DGEA) are used to project dynamic graphs, where vertices and edges can appear and disappear. Such graphs are mostly divided into snapshots. Current DGEAs are often offline, and computationally expensive, and most do not ensure a slow change in vertices projection.**

We propose here FODGE, a novel DGEA algorithm to gradually shift the projection of vertices whose first and second neighbors changed. FODGE optimizes CPU and memory efficacy by initially projecting the graph's densest K-core using any existing global optimization and then projecting the periphery of the graph using a local approximation. FODGE then smoothly updates the projection of all vertices, through an iterative local update rule. As such it can be applied to extremely large dynamic graphs over long periods.

We show that FODGE is faster than current algorithms, more accurate in an auxiliary task of link prediction. and it ensures a limited difference in vertex positions between consecutive time points. FODGE is highly modular and can be combined with any static projection, including graph convolutional networks, and has a few hyperparameters to tune. The code is available at https://github.com/unknownuser13570/FODGE in GIT.

***Index Terms*—component, formatting, style, styling, insert**

I. Introduction

Static graphs are commonly used in real-world applications, such as social and communication networks. Recently, multiple graph-based machine-learning algorithms have been proposed, such as the prediction of new friendships in social media, the classification of people by their age or gender, or the prediction of future edges in graphs [1]. An important tool used to facilitate such tasks is graph embedding [2]. Graph Embedding Algorithms (GEA) project vertices to a representation in $\mathbb{R}^d$. Such projections are optimized to capture the graph topology, vertex-to-vertex relationship, or other relevant information about graphs, sub-graphs, or vertices (see [3] for a recent survey). Formally, GEA can be defined as follows: Given a graph: $G = (V, E, W)$, where V are vertices, E edges and W the weights of the edges. A GEA is a mapping $f : v_i \rightarrow y_i \in \mathbb{R}^d \ \forall i \in \{1, ..., |V|\}$ such that $d << |V|$ and the function f preserves some proximity measure on the graph G.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

ASONAM '23, November 6-9, 2023, Kusadasi, Turkey
© 2023 Association for Computing Machinery.
ACM ISBN 979-8-4007-0409-3/23/11…$15.00
http://dx.doi.org/10.1145/3625007.3627481

GEA are limited to static graphs. However, graphs often contain dynamics with edges appearing and disappearing. To address this issue, Dynamic GEA (DGEA) algorithms have been developed. DGEA are slightly different from GEA, and are projecting a graph $G = (V, E, W)$, with a sequence of edge addition, removal, and changes in weights. In this framework, the addition of a new vertex connected to another vertex can be treated as the edge containing it. The addition or removal of a vertex with no edges attached is irrelevant to the embedding. Weight changes can be defined as the removal of an existing edge and the insertion of a new one with a different weight. Following these definitions, we treat dynamic graphs as a set of edges' appearance and removal events. For the sake of simplicity, we group such events into snapshots g_t. We here propose a novel efficient DGEA, using the following formal problem definition.

Given a temporal graph $\Gamma = (g_1, g_2, .., g_T)$, where each snapshot $g_t = (v_t, e_t, w_t)$ contains the vertices and edges in a given time interval $[t, t + \tau]$, and the cumulative snapshot containing all the edges appearing up to time t. We are looking for an embedding $z_{i,t} = z(v_i, (G_t, g_t))$, such that:

A) The distance between $z_{i,t}$ and $z_{i,t+\tau}$ grows with τ and is almost zero on average for the following time steps. B) For neighboring vertices i and j at time t, $z_{i,t}$ and $z_{j,t}$ are much closer on average to each other than non-connected edges. C) The embedding is online, and one can add one snapshot or remove/add one edge, using only $\{z_{i,t}\}$. D) The embedding CPU cost is linear in the number of added edges. E) Edges in recent snapshots are of more importance than edges in distant snapshots.

II. Related work

Existing DGEA can be divided into three types of algorithms: Snapshots based [2], where each snapshot is independently projected [4]–[6], continuous where edges arrive continuously over time [7], [8], and events based where the graph is updated following every edge update [9], [10]. Initial solutions for DGEA were to apply GEA (such as HOPE [11], LLE [12], node2vec [13], and others (e.g., [2]) to each snapshot independently. However, this usually led to a limited performance in terms of: A) Stability - The embedding of consecutive snapshots can differ substantially even though the graphs did not change much, B) Flexibility - Such DGEA assume a predefined number of vertices, and C) CPU cost - All vertices are embedded again in each snapshot. These limitations are partially resolved when different snapshots were combined to produce a coherent embedding [4], [14], [15]. GNN and attention-based methods, such as CTGCN

[6] have also been proposed. Temporally Factorized Network Modeling (TFNM) and Temporal Matrix Factorization (TMF) [16] are factorization methods, while DynGEM [17] is a toolkit of dynamic graph embedding methods based on autoencoders.

Continuous graph embeddings (CGE) are online inductive embeddings, i.e., vertices and edges can be added/deleted at every time step and they are updated online. Those include among many others, TGAT [7] that efficiently aggregates temporal-topological neighborhood features, M^2DNE [18] that treats both the micro and macro dynamics, and ActiveEM [8] that produces vertex embeddings based on the activeness of the neighborhood of a vertex at a given time. An alternative is events-based methods for homogeneous/attributed graphs that include among many others TigeCMN [10], TGN [9] and NIS [19].

III. Novelty and Algorithm

The main novel aspects of FODGE compared with existing algorithms are:

Cumulative network. Given, a temporal network $\Gamma = (g_1, g_2, .., g_T); g_t = (v_t, e_t)$, the first snapshot G_1 is expanded to the time where the largest connectivity component has a 5-core, and at least 100 vertices (T_1):$G_1 = G_{T_1}$. Following that, FODGE projects every snapshot using the cumulative graph of all edges up to time t - G_t, even if an edge is currently absent from g_t. This ensures consistency in time and maintains connected components, even if they are not currently connected. The weight of edges is multiplied by a factor $\beta < 1$ in each snapshot they are absent. Assuming $\tilde{w_t}$ is the observed weight at time t of an edge: $w_t = max(\tilde{w_t}, \beta * w_{t-1})$ We define $CC_1 = G_1$.

Separation between core and periphery. For G_1 (or any snapshot following a significant change in the network), FODGE first projects its K-core ($k = 5$) - to produce Z_{Core}, using any state-of-the-art projection. We have here focused on four algorithms - Node2vec [13] Graph Factorization [20], HOPE [11] and (Variational) Graph Auto-encoder (GAE) [21], but they can be replaced by any other algorithm.

All vertices v_j that were either not projected or changed their neighbors in the current snapshot are put in a max-heap according to the number of previously embedded neighbors. All vertices in the heap are then projected, using a local update rule. The position of a vertex is a linear combination of the average positions of its first and second neighbors, where the relative weight of the second neighbors is a free parameter (ϵ). While there is no formal reason to stop at second neighbors, we limit the analysis to second neighbors to ensure a low computational cost. Formally, for the vertex v_j at the top of the heap, define the set of first and second neighbors of v_j:

- $N_{1j} = \{n : (n, j) | (j, n) \in E_t \ , \ n \in H\}$
- $N_{2j} = \{n : (n, a) | (a, n) \in E_t \ , \ (a, j) | (j, a) \in E_t, \ n, a \in H\}$

Fodge then defines the average embedding of each such sets: $\overline{y}_1 = \frac{1}{|N_{1j}|} \sum_{i \in N_{1j}} z_i$, $\overline{y}_2 = \frac{1}{|N_{2j}|} \sum_{i \in N_{2j}} z_i$. The projection is then:

$$\overline{z}_j = \overline{y}_1 + \epsilon \cdot \frac{|N_{2j}|}{|N_{1j}|} \cdot (\overline{y}_1 - \overline{y}_2). \quad (1)$$

Momentum to keep the old position. To ensure stability over time, the current time embedding is a linear combination of the previous one and the newly calculated one. For each vertex with a previous embedding:

$$z_j(t) = \alpha * \overline{z}_j(t) + (1 - \alpha) * z_j(t). \quad (2)$$

Otherwise, $z_j(t) = \overline{z}_j(t)$.

There are two exceptions to the rules above: A) **Limited overlap between two consecutive snapshots, as measured by the Jaccard Index [22] (less than 0.9).** In such a case, the densest K-core of the new snapshot is projected independently, and then this projection is rotated to fit the current projection. The rotation from the core is applied to all vertices in the new snapshot. B) **Connecting connectivity components.** When an edge connects two previously unconnected connectivity components, or when a new connected component emerges, all the vertices in the smaller connectivity components are re-projected using the local update rule in the larger connectivity component. This leads to two additional elements:

Rotation of core if needed. If two consecutive snapshots are too different (as defined by their edge Jacquard index), we project the core of the new snapshot, and a rotation is applied to the new core to fit the previous core. The same rotation is then applied to all remaining vertices.

Let $C_t = (V_{C_t}, E_{C_t})$ and $C_{t-1} = (V_{C_{t-1}}, E_{C_{t-1}})$ be the cores of graphs G_t and G_{t-1}, respectively. FODGE calculates the Jacquard Index between V_{C_t} and $V_{C_{t-1}}$: $\frac{|V_{C_t} \cap V_{C_{t-1}}|}{|V_{C_t} \cup V_{t-1}|}$.. If $J < 0.9$, Fodge calculates the rotation matrix R that most closely maps the projection of the common of C_t and C_{t-1} using an orthogonal projection [23]. It then applies the same rotation to all the vertices in C_t that are not in C_{t-1}. The remaining vertices in G_t are projected using the local update rule.

Connecting connected components. Suppose two previously unconnected components ($CC1$, $CC2$) are connected at time t. With no loss of generality, let $CC1$ be the largest CC between $CC1$ and $CC2$. Fodges defines $CC1 = CC1 \cup CC2$ and then repositions each $v \in CC2$ using the local update rule in $CC1$.

IV. Methods

Datasets: We evaluate the embedding approaches on the following: Math [1], DBLP [2], Facebook wall posts [3], Facebook [4], Facebook Friendships [5], Enron [6], Wiki-Talk [7].

Test of FODGE accuracy. We used a temporal link prediction task to test the accuracy of FODGE. Temporal link

[1] http://snap.stanford.edu/data/sx-mathoverflow.html
[2] http://dblp.uni-trier.de/
[3] http://konect.uni-koblenz.de/networks/facebook-wosn-wall
[4] http://networkrepository.com/fb-wosn-friends.php
[5] http://konect.uni-koblenz.de/networks/facebook-wosn-links
[6] http://networkrepository.com/ia-enron-email-dynamic.php
[7] https://snap.stanford.edu/data/wiki-talk-temporal.html

prediction tasks can be defined in different ways. To properly compare the performance of FODGE to other algorithms, we defined two different link prediction tasks, as defined by previous publications [6], [14], and compared each existing method using the test used by it.

Hyperparameter tuning. FODGE requires three parameters: α (the weight given to the recent embedding when calculating the current one), β (the rate of the exponential decay of the weights) and ϵ (relative weight given to first and second neighbors in the local update rule), as well as the static embedding method used. We set $\alpha = \beta = 0.1$. ϵ was selected out of [0.1,0.04,0.01,0.008], and the GEA tested were (node2vec [13], HOPE [11], GF [20] and GAE [21]). Both were selected on an internal validation set of 20 %, with a training set of 60 %, and a test set of 20 %.

Comparison to other methods. When code was available and functional, we reproduced the results of existing methods. Otherwise, the results of the compared methods were taken from their reported values. Algorithms that did not provide a working code and thus were not retested here are marked as / symbol in tables. Algorithms with code that failed in the large graphs tested here in a single run on a 64-bit Linux machine with 8 GPU cores of 32510Mib memory each during one day were marked as "x".

V. Results

Three main assumptions underlie FODGE:

1) The fraction of edges that change between the cores of following snapshots is limited.
2) Edges that disappeared have a high probability of reappearing.
3) The change in the position of a vertex can be computed using a local update rule.

To apply FODGE, we tested the validity of these assumptions.

Snapshot cannot be treated as slowly changing, but the cumulative graph can. We computed the overlap between the core of the snapshots. Formally, given a temporal network Γ, let $\Gamma_g = (g_1, ..., g_T)$ and $\Gamma_G = (G_1, ..., G_T)$ be the snapshots (all edges between t_{i-1} and t_i) and the cumulative graph (all edges until t_i), respectively. We computed the core of each snapshot as the 100 vertices with the highest K-core score (with an arbitrary choice among the vertices in the last shell). Then, we calculated the Jacquard Similarity Index between the edges in each two consecutive cores, as well as between each core and the core of the first snapshot, for either the snapshots or the cumulative graphs (Fig. 1). The overlap in the core of consecutive snapshots, as well as their overlap with the initial snapshot are low (red and blue markers in Figure 2). However, the overlap of the cumulative graph in consecutive snapshots is high (green markers in Figure 2). Similarly, the overlap with the first graph decreases slowly over 5-10 snapshots. Thus, the cumulative graph must be used to ensure a slow adaptation of the vertex positions.

Disappearing edges have a high probability of reappearing . We tested the reappearance probability of disappearing edges. Formally, for each time t, we define the set $E_d(t)$ as

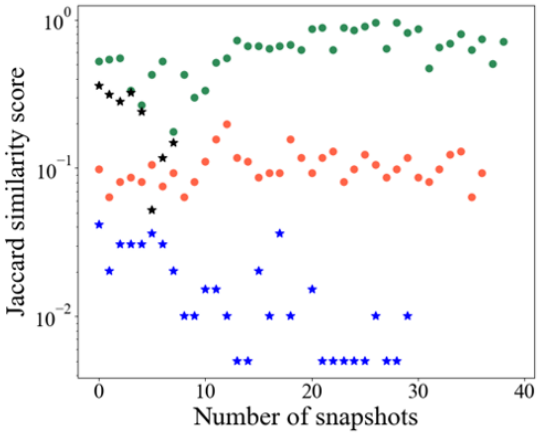

Fig. 1. Overlap of snapshots, measured by the Jaccard Similarity Index for the Facebook Wall Posts. Red stars are the comparison of consecutive snapshots. Blue stars are compared to the initial snapshot. Green and black are the same, but only for vertices included in the compared snapshot.

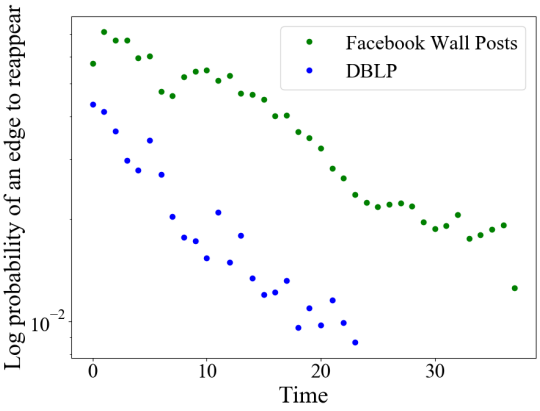

Fig. 2. Disappearing edges reappearance probability as a function of the time from disappearance. Blue dots are DBLP and green dots are Facebook Wall Posts. The graph is on a log scale on the y axis. The x axis is time from the last presence of an edge, and the y axis is the probability that an edge would reappear after such a time.

the set of edges existing at time t but not at time $t+1$. For each edge in $E_d(t)$, we measured the first time at which they reappear (if ever). We then computed the fraction of edges in $E_d(t)$ reappearing between time τ and $\tau + 1$ (Fig. 2). The probability decreases exponentially, but the probabilities are still higher by many orders than the random probability of a pair of vertices to have an edge between them. FODGE thus treats disappearing edges as unobserved existing edges with an exponentially decaying weight.

Local update rule FODGE uses any existing embedding methods on a small high-K-core score vertices core of the network. Then, it uses an online approximation, where we assume that the embedding of existing vertices is known, and project the remaining vertices. The local update rule is simply a local solution of a global optimization. While the same logic applies to many algorithms, the local approximation is best

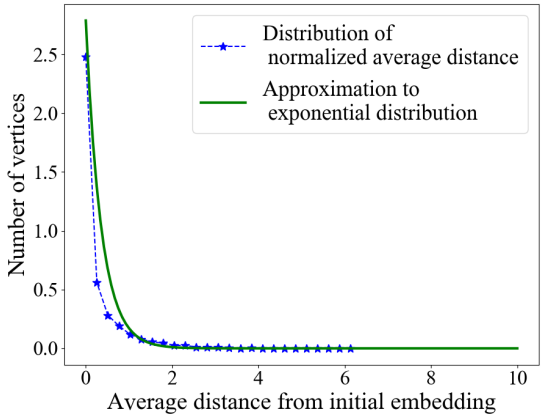

Fig. 3. Probability distribution function of normalized distance of the same vertex between following snapshots on Facebook Wall Posts dataset. The vast majority have a distance close to 0 (not changing), and even the ones changing are moving to much less than the average distance to other vertices.

TABLE I
FIRST TEMPORAL LINK PREDICTION PERFORMANCE MEASURED BY AUC SCORE. TEST RATIO IS 0.2.

Dataset	Math	DBLP	Facebook	
wall posts	Facebook			
Friendships				
HTNE	/	0.860	0.784	0.724
NIS	/	0.903	/	/
Dynamic				
Triad	/	0.618	0.643	0.535
TMFntv	/	0.900	0.925	/
TFNM	/	/	0.720	0.814
CTDNE	/	0.827	/	0.757
FODGE	0.813	0.867	0.928	0.863
	Facebook	Enron	Wiki-Talk	
ActiveEm	0.82	/	/	
FODGE	0.885	0.931	0.862	

explained using HOPE [11]. The loss function of HOPE can be written, as: $L = \sum_{i,j}(S_{i,j} - \overline{z}_i^T \overline{z}_j)^2$, where $\overline{x}_j$ is the projection of vertex j and S is some similarity matrix (e.g., the adjacency matrix). Assume that the projection of all vertices is known, and the projection of vertex j is now changing. The direction of the change in the projection of vertex j can be approximated to a first order by the gradient of the loss $\nabla_{z_j} L = \sum_i (S_{i,j} - z_i^T z_j) z_i^T$. Before the last update in the graph the loss was minimized, and as such: $\forall j; \nabla_{z_j} L_{Old} = 0$, where L_{Old} is the loss before the last update in the graph. Let $S_{i,j}(current) = S_{i,j}(old) + \Delta S_{i,j}$. One can compute: $\nabla_{z_j} L = \sum_i \Delta S_{i,j} z_i^T$ (since all other terms are 0). If S is the adjacency matrix, then $\triangle S_{i,j} = \begin{cases} 1 \text{ (i,j) edge added} \\ 0 \text{ Otherwise.} \end{cases}$ In such a case, the change in z_j is proportional to the change in the position of its first neighbors. The same happens when edges are removed.For more complex similarity matrices S, an expansion can be made to include the contribution of second neighbors. As such we propose a local update rule, which is simply a step in the direction of the linear combination of first and second neighbors.

A. FODGE link prediction accuracy

We have applied FODGE to multiple datasets and compared our results to existing algorithms on a link prediction task. This task has different versions and different results report different tasks. To ensure a proper comparison, we have tested two types of temporal link prediction tasks, used in the main existing algorithms (Tables 4 and 5). '/' represents an algorithm that did not provide a functioning code and had no published value. 'x' represents cases where code was available, but did not converge within one day of run.

To the best of our knowledge, FODGE is the first algorithm to provide results for Wiki-Talk. The ActiveEm algorithm reported results only on Facebook (among those tested here). We do not compare FODGE to TNodeEmbed [14], since their reported results are not consistent with the results of their own algorithm when tested on the same datasets. In the first temporal link prediction task (Table 2), FODGE outperforms all methods on Facebook Wall Posts and Facebook Friendships datasets. For DBLP, FODGE shows comparable results. The AUC on the other datasets cannot be compared, since no other algorithm managed to run on such large graphs, but they are high, and similar if not better than the other tested datasets. The same holds for the second temporal link prediction task (Table 1), except for the Math dataset.

TABLE II
SECOND TEMPORAL LINK PREDICTION AUC. VAL AND TEST RATIOS ARE 0.3 AND 0.2, RESPECTIVELY. FWP IS FACEBOOK WALLPOST. FB IS FACEBOOK. FFR IS FACEBOOK FRIENDSHIPS, AND WT IS WIKI-TALK

Dataset	Math	DBLP	FWP	FFr.	FB	Enron	WT
DynGEM	0.90	x	x	x	0.80	0.89	x
dynAE	0.92	x	x	x	x	x	x
GCRN	0.82	x	x	x	0.73	0.88	x
EvolveGCN	0.90	x	x	x	0.81	0.90	x
CTGCN-C	0.97	x	x	x	0.88	0.98	x
dynnode2V	/	/	/	/	/	0.90	/
FODGE	0.72	0.95	0.93	0.92	0.89	0.97	0.90

The three Facebook-based datasets have a small value of α (Supp. Mat. Table 2), suggesting that in such social networks, history is of limited importance. Similarly, in the Enron dataset $\alpha = 0$ and $\beta = 0.99$, i.e., there is no consideration of the history and the exponential decay rate is fast. Indeed, in these networks, the Jacquard Index values between consecutive snapshots are very high (even 1, i.e., 100% match). However, in the Math and Wiki-Talk datasets, α is relatively high and β is small, i.e., the history is taken into consideration, and edges that disappeared in previous snapshots have similar weights as the current time edges.

B. FODGE projections are changing slowly

One of the goals of FODGE is to ensure the continuity of the projection, such that the vertex position change is

limited between consecutive snapshots. We randomly chose 10k vertices for each dataset, and calculated the average distance between the embedding of vertices in consecutive snapshots, normalized by its average distance in the same snapshot to the other vertices. For all datasets, most of the vertices' initial embeddings are very close to the later embeddings. A movie for a typical evolution can be found in GIT https://github.com/unknownuser13570/FODGE .

C. FODGE is much faster than existing algorithms

Beyond the high accuracy and stability of FODGE, the only real computational limitation is the memory required to maintain the temporal network. For each snapshot, the computational cost of FODGE is composed of four elements: the construction of the neighbor dictionary, the rotation of cores if needed, the online projection of each neighbor, and the merging or creation of connected components. Of those, rotations and merging of components are rare, and the cost of every vertex update is constant. Indeed, the running time is approximately linear with the total number of vertices (data not shown).

VI. Discussion

We have presented here FODGE, a novel online cost-effective approach for dynamic graph embedding. The main assumption is that in any optimization-based projection, the contribution to the loss is low for low K-core vertices. As such, it is better to limit the initial projection only to vertices with a high K-core score, and then position all other vertices one by one, based on the positions of their already embedded neighbors. Moreover, to maintain the stability of the embedding, it is crucial to use the cumulative graph, instead of each snapshot by itself. Finally, we propose as do others ([14]) to rotate consecutive cores if they differ enough from each other. FODGE has many advantages, including stability, limited memory cost, and most importantly, a drastic gain in CPU cost and accuracy on the two temporal link prediction tasks. Another important advantage is its applicability to any kind of GEA.

The main limitation of FODGE is the local approximation, where the position of a vertex is a linear combination of the average positions of its first and second neighbors (controlled by a free hyperparameter). This local approximation when applied to too many consecutive snapshots can lead to a large deviation from the global optimization results. Still in all tested datasets, this did not seem to limit the accuracy of FODGE. When applied on very long time series, one may consider re-initialization at some stage.

FODGE was tested here on an auxiliary temporal link prediction task but can be expanded to other domains. One can use FODGE for the detection of highly variable vertices (in terms of neighbors and even connected components) or transition between communities, due to its stability. A vertex classification task can also be applied, using a GNN based graph embedding to initiate the core embeddings. This would produce a projection most adapted to a specific vertex classification task, even if it may lower the accuracy of link prediction. As such, FODGE can be considered as an additional approach to extend GNN to very large graphs.

References

[1] H. Cai, V. W. Zheng, and K. C.-C. Chang, "A comprehensive survey of graph embedding: Problems, techniques, and applications," *IEEE Transactions on Knowledge and Data Engineering*, vol. 30, no. 9, pp. 1616–1637, 2018.

[2] P. Goyal and E. Ferrara, "Graph embedding techniques, applications, and performance: A survey," *Knowledge-Based Systems*, pp. 78–94, 2018.

[3] P. Cui, X. Wang, J. Pei, and W. Zhu, "A survey on network embedding," *IEEE Transactions on Knowledge and Data Engineering*, vol. 31, no. 5, pp. 833–852, 2018.

[4] Y. Zuo, G. Liu, H. Lin, J. Guo, X. Hu, and J. Wu, "Embedding temporal network via neighborhood formation," *The 24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, August 19–23, 2018, London, United Kingdom*, 2018.

[5] A. Pareja, G. Domeniconi, J. Chen, T. Ma, T. Suzumura, H. Kanezashi, T. Kaler, T. B. Schardl, and C. E. Leiserson, "Evolvegcn: Evolving graph convolutional networks for dynamic graphs," 2019.

[6] J. Liu, C. Xu, C. Yin, W. Wu, and Y. Song, "K-core based temporal graph convolutional network for dynamic graphs," *IEEE Transactions on Knowledge and Data Engineering*, p. 1–1, 2020.

[7] D. Xu, C. Ruan, E. Korpeoglu, S. Kumar, and K. Achan, "Inductive representation learning on temporal graphs," *ICLR 2020*, 2020.

[8] L. Munasinghe and R. Ichise, "Activeem: A node embedding method for dynamic social networks," 10 2020.

[9] E. Rossi, B. Chamberlain, F. Frasca, D. Eynard, F. Monti, and M. Bronstein, "Temporal graph networks for deep learning on dynamic graphs," 2020.

[10] Z. Zhang, J. Bu, M. Ester, J. Zhang, C. Yao, Z. Li, and C. Wang, "Learning temporal interaction graph embedding via coupled memory networks," 2020.

[11] M. Ou, P. Cui, J. Pei, Z. Zhang, and W. Zhu, "Asymmetric transitivity preserving graph embedding," *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, vol. ACM, p. 1105–1114, 2016.

[12] S. T. Roweis and L. K. Saul, "Nonlinear dimensionality reduction by locally linear embedding," *Science*, pp. 2323–2326, 2000.

[13] A. Grover and J. Leskovec, "node2vec: Scalable feature learning for networks," 2016.

[14] U. Singer, I. Guy, and K. Radinsky, "Node embedding over temporal graphs," *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence (IJCAI-19)*, 2019.

[15] M. Haddad, C. Bothorel, P. Lenca, and D. Bedart, "Temporalnode2vec: Temporal node embedding in temporal networks," 2019.

[16] D. M. Dunlavy, T. G. Kolda, and E. Acar, "Temporal link prediction using matrix and tensor factorizations," *ACM Transactions on Knowledge Discovery from Data*, vol. 5, p. 1–27, Feb 2011.

[17] P. Goyal, N. Mehrabi, and E. Ferrara, "Dynamicgem: A library for dynamic graph embedding methods," *Journal of Machine Learning Research*, vol. 1, pp. 1–48, 2018.

[18] Y. Lu, X. Wang, C. Shi, P. S. Yu, and Y. Ye, "Temporal network embedding with micro- and macro-dynamics," 2019.

[19] M. Liu, Z. Quan, and Y. Liu, "Network representation learning algorithm based on neighborhood influence sequence," 2020.

[20] A. Ahmed, N. Shervashidze, S. Narayanamurthy, V. Josifovski, and A. J. Smola, "Distributed large-scale natural graph factorization," *Proceedings of the 22nd International Conference on World Wide Web, ACM*, pp. 37–48, 2013.

[21] T. N. Kipf and M. Welling, "Variational graph auto-encoders," *NIPS Workshop on Bayesian Deep Learning*, 2016.

[22] L. Hamers *et al.*, "Similarity measures in scientometric research: The jaccard index versus salton's cosine formula.," *Information Processing and Management*, vol. 25, no. 3, pp. 315–18, 1989.

[23] P. H. Schönemann, "A generalized solution of the orthogonal procrustes problem," *Psychometrika*, vol. 31, no. 1, pp. 1–10, 1966.