STATISTICS, OPTIMIZATION AND INFORMATION COMPUTING
Stat., Optim. Inf. Comput., Vol. **16**, December 2026, pp 6494–6514.
Published online in International Academic Press (www.IAPress.org)

DynβRW: Dynamic β-Random Walk for link prediction

Asmaa Moussaddar*, Badr Hirchoua, Ibrahim Guelzim

Artificial Intelligence, Modeling and Computational Engineering (AIMCE), National Higher School of Arts and Crafts (ENSAM), Hassan II University of Casablanca (UH2), Casablanca, Morocco

Abstract
Graph embedding represents nodes in a continuous, low-dimensional latent space while preserving the graph's underlying structure. In practice, however, most real-world graphs are far from static — nodes and edges appear, vanish, and shift in weight as time passes. Despite this, a large portion of the embedding literature is built around the static setting, leaving the temporal dimension as an afterthought or ignoring it entirely.
We propose DynβRW-GNN, a dynamic graph embedding framework that couples β-Random Walks with a GraphSAGE-based encoder. Rather than folding time into a single monolithic pipeline, the architecture keeps two concerns cleanly separate: the β-Random Walk module handles temporal exploration, producing temporally-aware node embeddings that reflect how each node's neighborhood evolves over time; a subsequent multi-layer GraphSAGE encoder then refines these into richer structural representations. A lightweight MLP decoder operates on candidate node pairs to produce a link prediction score, completing the pipeline.
We separate these tasks on purpose. Mixing temporal exploration with representation learning often leads to poor results for both. By giving each task its own stage, the model can better capture detailed structural patterns and long-term temporal changes without the two goals interfering with each other. We also prove the convergence of the β-RW to a stable state, and we prove that the dimension of the embeddings is much lower than n, the number of nodes in the graph. Experiments on dynamic graph benchmarks show competitive improvements over baselines on the link prediction task, confirming that explicit temporal exploration is a worthwhile inductive bias when the graph itself is moving.

Keywords Node weighting, Link prediction, β-Random walks, Dynamic graph embedding, Balanced Sampling

AMS 2010 subject classifications 68T09, 60J10, 05C82

DOI: 10.19139/soic-2310-5070-4135

1. Introduction

The growing amount of relational data from social networks [1], protein-protein interactions [2, 40], research collaborations [3], and cybersecurity applications [44] highlights the importance of analyzing graph-structured data to capture diverse types of relationships and structural dependencies. Most real-world networks are dynamic by nature, as interactions evolve over time, making modeling and analysis more challenging. On the other hand, graph data is highly irregular and non-Euclidean, which makes it difficult to process directly with traditional machine learning algorithms. For this reason, node embedding techniques have been the subject of many studies [4]. The node embedding objective is to represent nodes as low-dimensional continuous vectors while preserving the local and global graph information. Beyond looking at algorithms, we can also see dynamic graph embedding from geometric and dynamical perspectives. From another view, node embeddings correspond to projections into a latent space, where the geometry encodes structural information between nodes. Temporal evolution of the graph generates a diffusion process over this space, controlling the elaboration of representations over time. Classical

*Correspondence to: Asmaa Moussaddar (Email: asmaa.moussaddar13@gmail.com). Artificial Intelligence, Modeling and Computational Engineering (AIMCE), National Higher School of Arts and Crafts (ENSAM), Hassan II University of Casablanca (UH2), Morocco

ISSN 2310-5070 (online) ISSN 2311-004X (print)
Copyright © 2026 International Academic Press

embedding techniques such as DeepWalk [5] and node2vec [6] tend to perform better in static contexts [11]. However, these approaches struggle to capture both structural and temporal dynamicity in evolving graphs, leading to information loss (including the appearance or disappearance of links or nodes). Modeling dynamic graphs is still very challenging, particularly for tasks such as link prediction that require capturing temporal dependencies. All these challenges lead to an important question: **How to design dynamic node embeddings that capture both temporal evolution and structural-geometric consistency?**

To address this issue, dynamic graph integration methods are typically categorized based on their temporal modeling approach: snapshot-based or continuous-time [7]. Snapshot-based methods treat graph evolution as a sequence of discrete states, whereas continuous-time methods model interactions as a continuous sequence of events. Within snapshot-based methods, some approaches utilize random walks [8], in which exploration is determined by transition probabilities between nodes. These trajectories actually capture both local and global graph structures. However, adapting these methods to changing environments requires tracking how things shift over time. This type of transformation poses challenges; it involves maintaining consistency over time, ensuring representation stability, and managing long-term modeling. Traditional approaches rely on static representations that can't really capture how graph structure evolves. Because of this, they often miss important patterns in temporal interactions. Graph Neural Networks (GNNs) [9] present a practical approach for representing nodes by diffusing information through the local graph neighborhoods, unlike random walk-based methods. The thing is, GNNs often depend on fixed graph structures and don't really give you clear control over exploration, and that can actually limit their ability to pick up on complex structural and temporal patterns. In this work, we introduce a novel methodology, Dynamic β-Random (DynβRW-GNN), which extends the static β-random walk method [10, 11] to dynamic graphs and integrates it with Graph Neural Networks (GNNs) to enhance performance. Our approach separates graph exploration from representation learning. Initially, we conduct a dynamic exploration of the graph using a stochastic random walk to extract both structural and temporal information. Every node is represented by a concise vector across the viewing span that captures its time-linked shifts. The resulting embeddings are then refined by a GNN, allowing the model to learn more subtle representations. The proposed method is governed by a single parameter, β, which affects the random sampling process, node weight calculation, and convergence behavior. In our approach, we employ a Full Rotation Convergence strategy, where the method sequentially processes each snapshot before advancing to the next. This procedure captures the dynamic influence of nodes, while the GNN component enhances representation learning, resulting in embeddings that are both resilient and temporally coherent for link prediction tasks.

We evaluate our methodology on three real-world empirical datasets and demonstrate its effectiveness in link prediction tasks using standard metrics, including AUC and Average Precision (AP). The results demonstrate that the β-RW component provides stable and interpretable node representations over time, which effectively guide the GNN component and contribute to competitive future link prediction performance compared with baseline methods. [12]. The rest of the paper is arranged as follows: Section 2 provides an overview of related work, separating static from dynamic methods. Section 3 presents the preliminary concepts needed to properly understand the article. Section 4 details our methodology and includes an illustrative example. Section 5 presents and discusses experimental results, including comparisons for the link prediction task. Finally, Section 6 concludes the study and highlights potential improvements and future research directions.

2. Related Work

Graph embedding techniques seek to generate a vector representation of nodes or the entire graph within a low-dimensional continuous space, while preserving essential structural and semantic properties. These embeddings are used for multiple tasks, such as link prediction, community detection, and node classification. Embedding methods are generally categorized as static or dynamic, according to whether the graph structure remains fixed or evolves over time. The following sections review significant advancements both in static and dynamic graph embedding methodologies.

2.1. Static graph embedding

Static graph embedding strategies break down into four main classes: matrix factorization-based, neural network, encoder methods, and random walk-based ones [13]. DeepWalk [5] is among the earliest approaches out there. It borrows concepts from word embeddings. Sequences of nodes are generated via random walks, and then a skip-gram model is trained to create node representations.

Node2vec[6] extends DeepWalk by introducing unstable random walks, controlled by two parameters, p and q, to more effectively explore both local and global graph structures.

Alternative methods, including LINE[14] and GraRep[15], utilize matrix factorization to capture relationships among nodes at different distances. SDNE[16] applies deep autoencoders to preserve both local and global graph structures. Hirchoua et al.[10] propose the β-random-walk method, which relies on a single parameter, β, to enable controlled random sampling and an adaptive weighting mechanism.

2.2. Dynamic graph embedding

Networks from Real-world evolve over time, making static graph embedding methods insufficient for capturing their dynamic nature [17]. To overcome this limitation, dynamic graph embedding methods have emerged, incorporating temporal information into the learning process. These methods can be classified into two main categories: snapshot-based approaches, which represent the network as a sequence of static graphs, each corresponding to a different time interval. On the other hand, continuous-time approaches model network changes as they occur [17]. While snapshot-based methods are simpler and easier to implement, continuous-time models offer a more precise view of network evolution, especially in applications where the exact timing of interactions matters[17]. Early dynamic graph embedding methodologies, such as DynGEM [18] and DynamicTriad [19], may be seen as natural extensions of static graph embedding techniques. These models build upon the foundations of static methods by incorporating temporal information across graph snapshots, allowing them to capture the evolving structure of real-world networks. These two methods focus on preserving structural stability and modeling triadic evolution across temporal snapshots. Building on these foundations, several other snapshot-based methods have been proposed, including dynNode2Vec [20], DyG2vec [21], DySAT [22], dynGraph2vec [23], K-Core CTGCN [24], TemporalNode2vec[25], and Cluster-GCN [26]. These approaches enhance dynamic embedding by leveraging various techniques such as random walks, attention mechanisms, core decomposition, and scalable graph convolution, improving the modeling of temporal and structural changes in evolving networks. Additionally, recurrent and graph neural network-based approaches such as DynAERNN [39], VGRNN [27], and EvolveGCN [28] further enhance temporal modeling by learning evolving node representations across snapshots.

Method	**Type**	**Exploration strategy**	**Temporal modeling**	**Embedding scheme**	**Relation to DynβRW**
DeepWalk	Static	Uniform random walks	None	Skip-gram node embeddings	Baseline static random walk approach
Node2vec	Static	Biased walks (p, q)	None	Skip-gram node embeddings	Adds bias but still static setting
Temporal Node2vec	Discrete temporal	Biased temporal walks	Snapshot-based evolution	One embedding per snapshot	Captures time via snapshots, but coarse-grained
CTDNE	Continuous temporal	Time-respecting walks	Continuous timestamps	Static embedding updated over time	Preserves temporal order but limited structural exploration
DynamicTriad	Discrete temporal	Triad-based modeling	Snapshot evolution	Node embeddings via triadic closure	Focus on local temporal motifs
DynβRW (ours)	Dynamic	β-guided adaptive walks	Continuous weighting mechanism	History-aware node representation	Balances exploration and temporal influence via β

Table 1. Comparison of static and dynamic graph embedding methods.

In contrast to snapshot-based approaches, continuous-time dynamic graph embedding methods model network evolution as a sequence of events occurring at irregular time intervals. Instead of discretizing time into fixed

snapshots, these methods capture fine-grained temporal dynamics by updating node representations at each interaction [29]. Notable continuous-time models include Temporal Graph Networks (TGN) [30], Euler [31], TemporalGAT [32], Temporal Backtracking [33], and CTDNE [34]. Furthermore, models such as JODIE [35], DyRep [36], and TGAT [37] rely on recurrent architectures and temporal attention mechanisms to capture complex interaction dynamics. By directly incorporating timestamps, continuous-time methods offer improved accuracy in scenarios where the precise timing of interactions significantly influences network behavior [38].

3. Preliminaries

This section elucidates the essential concepts of dynamic graph embedding. Additionally, we emphasize the temporal random walk and its connection to Markov chain theory.

3.1. Graphs

Definition 1 (Graph)
A graph is defined as $G = (V, E)$, where V is a set of nodes and E is a set of edges representing interactions between nodes.

Definition 2 (Random Walk)
A random walk on a graph $G = (V, E)$ is a stochastic process where moving from a node to one of its neighbors follows a transition probability distribution.

For a uniform random walk, the transition probability from a node v to a node u is given by:

$$P(v \to u) = \frac{1}{\deg(v)}$$

where $deg(v)$ represent the number of adjacent nodes to v.

Property 1 (Transition Matrix)
A random walk can be represented by a transition matrix P, where each entry P_{vu} denotes the probability of moving from node v to node u:

$$P_{vu} = P(v \to u)$$

The transition matrix is defined as:

$$P \in \mathbb{R}^{|V| \times |V|}$$

and satisfies the stochastic property:

$$\sum_{u \in V} P_{vu} = 1, \quad \forall v \in V$$

Definition 3 (Markov Chain)
A Markov chain is a stochastic process $\{X_t\}_{t \geq 0}$ satisfying the Markov property:

$$\mathbb{P}(X_{t+1} = v \mid X_t = u, X_{t-1}, \dots) = \mathbb{P}(X_{t+1} = v \mid X_t = u)$$

A random walk on a given graph is equivalent to a Markov chain whose state space is the set of nodes V.

3.2. Dynamic Graphs

Definition 4 (Dynamic Graph)
A dynamic graph can be presented as a sequence of snapshots $G = (G_1, G_2, \dots, G_T)$, where each snapshot $G_t = (V_t, E_t)$ corresponds to the graph state at a specific time t, where V_t and E_t denote the set of G nodes and edges at that time.

Definition 5 (Temporal Random Walk)
We define temporal random walk as an extension of the classical random walk that incorporates temporal information in the transition mechanism, so that the choice of the next node depends also on the evolution of interactions over time not only on the graph structure.

Let $G_t = (V_t, E_t)$ be a snapshot of the dynamic graph G at timestamp t. At each step, the walk moves from a node v to one of its neighbors u according to a time-dependent transition probability.

Remark 1
Unlike classical random walks, temporal random walks take into account the temporal evolution of edges, allowing more recent interactions to influence the traversal process.

In dynamic graphs, the structure evolves over time, inducing a sequence of transition matrices:

$$\{P^{(t)}\}_{t=1}^{T}$$

Unlike classical Markov chains, the transition matrix depends on time:

$$P^{(t)} \neq P^{(t+1)}$$

This leads to a **time-inhomogeneous Markov process** on the dynamic graph.

3.3. Node Embeddings

Definition 6 (Node Embedding)
Given a graph $G = (V, E)$, a node embedding is a mapping

$$f : V \to \mathbb{R}^D$$

that assigns to each node a low-dimensional vector representation that preserves the graph's structural relationships.

Remark 2
A random walk on a graph can be interpreted as a Markov chain with state space V and transition probabilities determined by the graph structure. This connection enables the analysis of random walks using Markov chain theory.

Definition 7 (Temporal Node Embedding)
Given a dynamic graph, a temporal node embedding is a mapping:

$$f(v) \in \mathbb{R}^D$$

that encodes the temporal evolution of node v across all timestamps into a single fixed-size vector representation.

4. Methods

4.1. The β-random walk method

We first briefly recall the static β-random walk model [10]. This methodology depends on a single parameter β allocated to each node v of the static graph G, initially set to 1 and subsequently decreased with each visitation to the node by $\eta_v = \frac{1}{|V|+\deg(v)}$, where $|V|$ denotes the total number of nodes in the graph G and $\deg(v)$ represents the number of adjacent nodes to v. The process includes two essential phases, which are iterated in each episode until convergence: *trajectory sampling* and *weighting cycles*. The sampling process is governed by the parameter β. For each new trajectory sampling, a depth is randomly chosen, and all nodes in the graph are simultaneously highlighted. Exploration commences from this node, prioritizing neighbors that have the highest β value. In instances where multiple nodes share this β value, selection is random among them. This exploration method

ensures that even low-degree nodes are visited, and the β values of high-degree nodes diminish more slowly, thereby prolonging the significance of these crucial nodes. β_v of each node v decreases after each sampling episode following the equation

$$\beta_v(t+1) = \beta_v(t) - l_v(t) \times \eta_v \tag{1}$$

where $l_v(t)$ denotes the number of visits to the node v in episode t; when the node is in focus, it is regarded as a single visit.η_v is v's decreasing rate. Following each sampling episode comes a weighting episode during which the algorithm computes a new weight for all nodes with a non-zero β value. The initial weight at $t = 0$, is $\psi_v(0) = \frac{1}{\deg(v)}$, for every node v in the graph. Then, the weight is calculated using the following equation:

$$\psi_v(t+1) = \psi_v(t) + \beta_v(t) \sum_{w \in N_v} |\psi_w(t_l) - \psi_v(t)| \tag{2}$$

where N_v is the set of v neighbors and $\psi_w(t_l)$ is the last known weight of the node w. The two steps are repeated until convergence, at which point the weights stabilize, or the parameter β for all nodes vanishes. The dimension D of the output vectors depends on the convergence speed of the method on the given graph G.

4.2. Convergence Proof

In this section, we will prove the convergence of the method mathematically. Let's take a connected graph G with $|V| = n$ nodes. v_i is a node from G with a maximum degree $deg(v_i) = |N_{v_i}| = k$, where N_{v_i} is the set of v_i's neighbors. We want to prove that the weight of the node v_i attains a stable state. Formally

$$\exists t \in \mathbb{T}, \forall \varepsilon > 0, |\psi_{v_i}(t+1) - \psi_{v_i}(t)| < \varepsilon \tag{3}$$

Since for $t \in \mathbb{T}$

$$|\Psi_{v_i}(t+1) - \Psi_{v_i}(t)| = |\beta_{v_i}(t) \sum_{v_j \in N_{v_i}} |\Psi_{v_j}(t_l) - \Psi_{v_i}(t)|| \underset{\forall v_i \in V, \beta_{v_i} > 0}{=} \beta_{v_i}(t) \sum_{v_j \in N_{v_i}} |\Psi_{v_j}(t_l) - \Psi_{v_i}(t)|, \tag{4}$$

For every $t \in \mathbb{T}$ we have

$$\forall v_j \in N_{v_i} \exists M_{v_j} > 0 : |\Psi_{v_j}(t_l) - \Psi_{v_i}(t)| < M_{v_j}, \tag{5}$$

let's take $M = max(M_{v_j})$, for $v_j \in N_{v_i}$.
Then

$$\beta_{v_i}(t) \sum_{j=0}^{N_{v_i}} |\Psi_{v_j}(t_l) - \Psi_{v_i}(t)| < \beta_{v_i}(t).M.k \tag{6}$$

It is established that β_{v_i} diminishes to zero, yet the reduction occurs after a finite number of iterations denoted t_{v_i}, and $t_{v_i} \leq k + n$, since $\beta_{v_i}(t+1) = \beta_{v_i}(t) - \frac{1}{k+n}$. So $\beta_{v_i}(t_{v_i}) = 0$.

4.3. Dimension Proof

In this section, we will prove that the dimension D of the embeddings produced by the β-Random Walk method applied to a static dense and large graph $G = (V, E)$, where $|V| = n$, satisfies $D \ll n$.

Theorem 1
Let $G = (V, E)$ be a static graph with $n = |V|$, and let v be a node of maximum degree. Let D denote the dimension of the node embeddings produced by the β-Random Walk algorithm. Then,

$$D \leq \frac{n + \deg(v)}{1 + \deg(v)}.$$

Proof

The embedding dimension is equal to e, the number of episodes required for the algorithm to converge. Hence, $D = e$. The value of β_v is initialized to 1 and decreases by: $\frac{1}{n+\deg(v)}$ each time node v is processed as the initial node of a trajectory.

Therefore, the total number of visits required for β_v to reach zero is $n_v = n + \deg(v)$. According to the implementation of the algorithm, each episode processes every node once as an initial node. Moreover, for every neighbor $u \in N_v$, one trajectory starts with the transition $u \to v$.

Consequently, node v is visited at least $(1 + \deg(v))$ times during each episode: once when it is selected as the initial node and once as the first visited neighbor of each of its neighbors.

Therefore,

$$(1 + \deg(v))e \leq n_v.$$

Substituting $n_v = n + \deg(v)$ gives

$$e \leq \frac{n + \deg(v)}{1 + \deg(v)}.$$

Finally, since $D = e$,

$$\boxed{D \leq \frac{n + \deg(v)}{1 + \deg(v)}.}$$

This completes the proof. □

Corollary 1

Let $G = (V, E)$ be a large dense graph whose maximum degree satisfies $\Delta = \Theta(n)$. Then, the embedding dimension generated by the β-Random Walk algorithm satisfies:

$$D \ll n.$$

Proof

From Theorem 1,

$$D \leq \frac{n + \Delta}{1 + \Delta}.$$

Since $\Delta = \Theta(n)$,

$$\frac{n + \Delta}{1 + \Delta} = \Theta(1).$$

Hence,

$$D = O(1) \ll n.$$

This completes the proof.

□

4.4. Dynamic β-Random Walk (DynβRW)

The dynamic β-random walk is an adaptation of the β-random walk model to dynamic graphs. Algorithm 1 is the training algorithm of the DynβRW model. This algorithm highlights the steps of training the model and adapting its parameters. A negligible convergence threshold, $\varepsilon = 10^{-20}$, is employed to guarantee that the iterative process persists until the weight changes are insignificant. This rigorous criterion enables the algorithm to execute adequate iterations to thoroughly utilize structural information present in the graph snapshots and to prevent premature stopping.

Algorithm 1 DynβRandom Walk for Dynamic Node Embedding

Require: Sequence of graph snapshots $G = (G_1, G_2, \ldots, G_T)$
Ensure: Node embeddings $\mathcal{Z} = \{\mathbf{z}_v \in \mathbb{R}^D \mid v \in V\}$

Initialization:
for each node $v \in V$ **do**
    $\beta_v(0) \leftarrow 1, \quad \psi_v(0) \leftarrow 0$
end for
$e \leftarrow 0$
converged $\leftarrow$ **False**
while *converged* = **False do**
    $e \leftarrow e + 1$
    for $t \leftarrow 1$ to T **do**
        for each node $v \in V_t$ **do**
            $\eta_v(t) \leftarrow \frac{1}{\deg_{G_t}(v)+c}$
            if $\psi_v(t-1) = 0$ **then**
                $\psi_v(t) \leftarrow \frac{1}{\deg_{G_t}(v)}$
            end if
        end for
        Sample β-biased random-walk trajectories on G_t and update $\beta_v(t)$ for all $v \in V_t$ following equation 1
        Learn $Z^{(t)}$ from sampled walks by calculating $\psi(t+1)$ following Equation 2.
        if $e > 1$ **and** all $\beta_v = 0$ **then**
            converged $\leftarrow$ **True**
            break
        end if
    end for
    if *converged* = **False and** $\|\psi_v(t+1) - \psi_v(t)\| < \varepsilon$ **then**
        converged $\leftarrow$ **True**
    end if
end while
return $\mathcal{Z}$

4.4.1. Notations A dynamic graph $G = (G_1, G_2, \ldots, G_T)$ is considered, where each snapshot $G_t = (V_t, E_t)$ and $V = \bigcup_{t=1}^{T} V_t$ set of all nodes over time. $\eta_v(t)$ is the decreasing rate of the node v in snapshot G_t, and $\deg_{G_t}(v)$ is the degree of v in snapshot G_t .

We use a *Full Rotation Convergence*, which iteratively rotates through snapshots while maintaining β continuity. Furthermore, we show that a convergence-per-snapshot strategy is ineffective, leading to weight explosion and significantly increasing computational time. Fig. 1 and Fig. 2 illustrate the complete framework of the proposed algorithm.

Complexity analysis: The complexity of the DynβRW is fundamentally dependent on two primary processes that are executed for each snapshot, multiplied by the number of episodes: sampling episodes and weighting cycles. The computation of a node's weight can commence once all its trajectories have been sampled. The complexity of these two procedures for every node is $O(k \times d \times n)$, as explained in [10], where k, d, and n denote the maximum node degree in the dynamic graph G, the depth variable, and the total number of nodes in the dynamic graph G. Observe that the depth variable d is selected randomly in each episode, independent of the graph size n, and its upper limit is generally established to be significantly less than n, thus $d << n$. Conversely, it can be asserted that $k << n$ since it is uncommon for a node to be connected to every other node in the graph, and in such instances, the number of such nodes is limited. In the Full Rotation Convergence parameters for each episode, there are

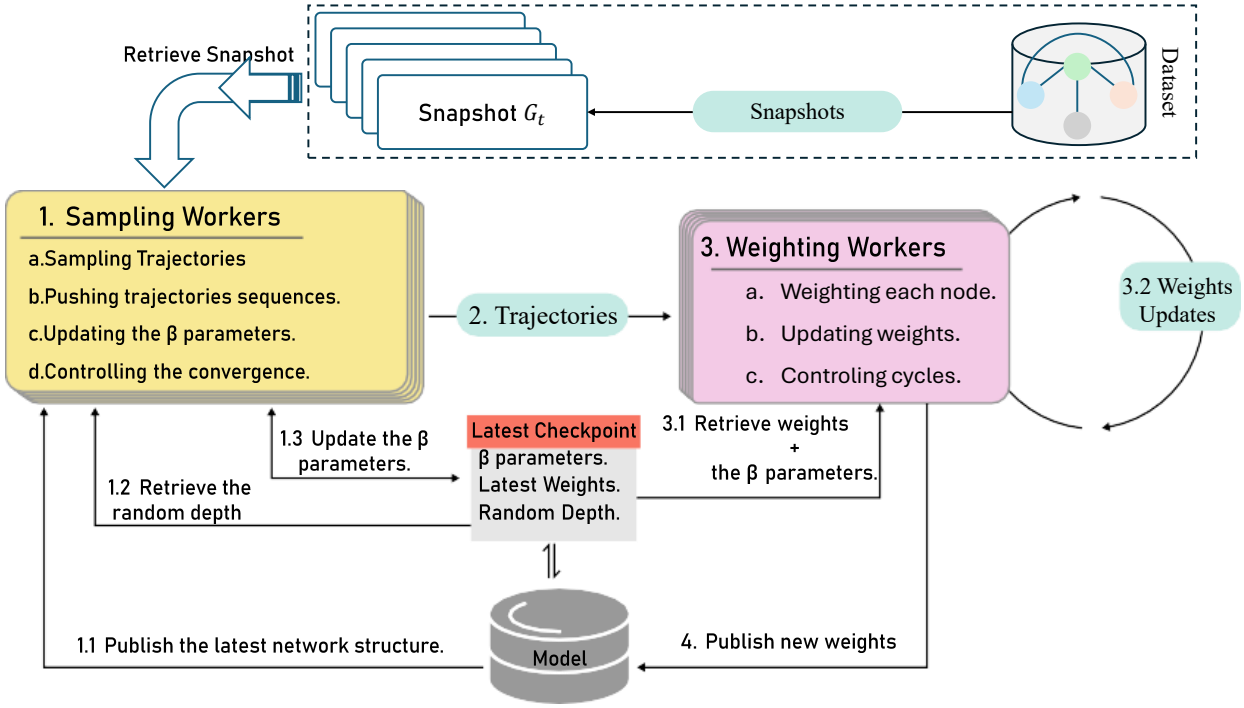


Figure 1. DynβRW framework.

At each time step, sampling workers generate trajectories showing the graph's structure. Meanwhile, weighting workers use adaptive β parameters to update node representations. A shared storage module keeps weights and checkpoints, creating a feedback loop that keeps snapshots consistent over time.

T snapshots, and let e denote the total number of episodes; thus, the complexity of the DynβRW approach is $O(k \times d \times n \times e \times T)$, where k,d, and T are always very small compared to n.

4.5. Illustrative Example

This section presents a comprehensive theoretical analysis, illustrated with an example, to demonstrate the efficiency of our framework's parts. We set the constant of the decreasing rate to $c = 30$ for this example. Fig. 3 shows two snapshots of the utilized graph, featuring varying neighbor sizes, with a maximum of four neighbors in snapshot 3a. We employ diverse neighbor counts to monitor changes in the β parameter alongside the weight progression of both significant and less significant nodes. One node is newly added, M, and one is deleted, E, in snapshot 3b, to observe how the algorithm handles the dynamic nature of nodes and links. Tables 3 and 4 present the starting configuration for the first and second episodes.

As an example, the initial weight for node A is $\psi(t = 0) = \frac{1}{2} = 0.5$, and its decreasing rate for the associated β parameter is $\frac{1}{30+deg(A)} = \frac{1}{30+2} = 0,03125$, where $deg(A)$ designates the number of adjacent nodes to the node A,in index 1 of table 2 β of A become $\beta_A = 1 - \frac{1}{30+2} = 0.96875$. For each new episode, it is necessary to recalculate the decreasing rate for each node, since node degrees vary from one snapshot to another. We only compute the initial weight for newly added nodes, while others retain the previously known weight as their starting configuration.

Significantly, critical nodes with a greater number of neighbors should exhibit a reduced β decay rate. This behavior tends to make the sampling procedure traverse more significant nodes than those of lesser importance. It offers the anticipated functionality and the requisite flexibility to assess all potential trajectories. This refers to node B in this example, which serves as the sampling point for most trajectories. The rate of decrease fluctuates across snapshots as the significance of nodes evolves over time.

Given the first snapshot 3a of the graph G as input, the DynβRW involves two major steps. In the initial step,

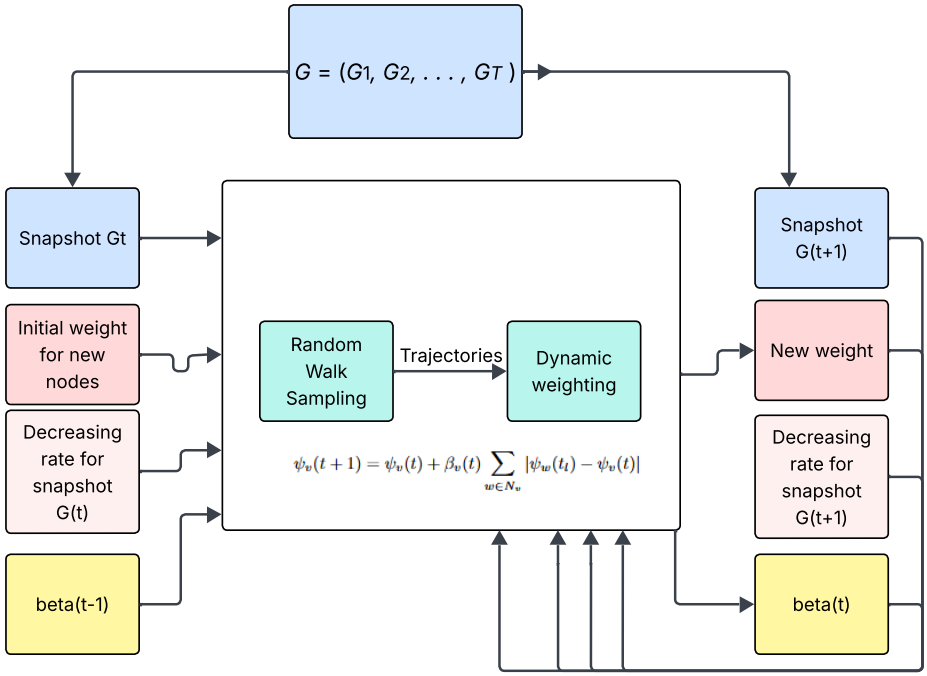
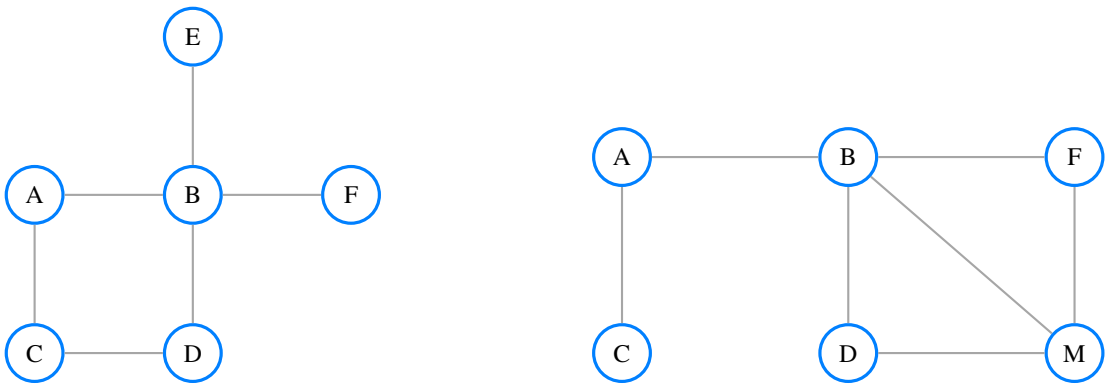


Figure 2. Overview of the DynβRW Framework for Dynamic Node Weight Updating.

At every time step t, the current snapshot G_t, the initial weights assigned to newly appearing nodes, a list of nodes with decreasing rates for the specific snapshot G_t, and the previous values $\beta(t-1)$ are provided to the Random Walk Sampling module to generate node trajectories. These trajectories are subsequently processed by the Dynamic Weighting module, which updates the weight of each node according to Equation 2. More specifically, the weight of a node v is adjusted based on the discrepancy between its current weight and those of its neighboring nodes, scaled by an adaptive learning rate $\beta_v(t)$. The updated node weights, together with the new decay rates and a list of $\beta(t)$ values, are then propagated to the next snapshot G_{t+1}, allowing continuous and adaptive representation learning over the evolving graph structure.



(a) Snapshot 1 (b) Snapshot 2

Figure 3. Evolution of the graph through two snapshots

sampling trajectories are governed by the β parameter corresponding to each node and a randomly chosen depth. In this example, the decreasing rate of the parameter β is displayed in Table 3, and we define a depth of 3 for the initial episode in snapshot 3a. The provided graph G allows two consecutive sampling episodes to be displayed simultaneously in tables 2 and 5, with the first occurring in snapshot 3a and the second in snapshot 3b. The model initiates a new sampling episode after examining all nodes and after sampling their trajectories. Each node possesses a similar probability of being in focus, irrespective of its significance within the graph. It initiates with node A in focus (index 1 in Table 2). The anticipated number of trajectories is 2, as node A has 2 neighbors, B and C. Typically, the neighbor with the highest β value is selected; however, at this initial stage, all nodes are assigned an equal β value of 1. Consequently, node E is chosen randomly from the neighbors of node B, and node D is selected from the neighbors of node C. Node B is in focus, with four neighboring nodes, resulting in four anticipated trajectories; the first two have depth 2, as F and E lack any remaining neighbors. The other two trajectories from B indicate that C is the sole connection from A and D. For instance, (index 4 in table 2), the second trajectory from D to B has three additional neighbors, and the algorithm selects F due to its superior β value of 0.96774 in comparison to A and E. E and F each have only one neighbor B then the algorithm selects the neighbor with the largest β, D, for the trajectory from E and A for the trajectory from F. In the second episode depicted in Table 5, the selected depth is 4. The method exhibits consistent behavior: each node with a non-null β is put in focus exactly once. Subsequently, from that node, the algorithm selects the neighbor with the highest β parameter. In instances where two nodes possess the highest β, the selection occurs randomly, continuing until the depth reaches 4 or no neighbors remain.

All sampling episodes conclude with the start of a new weighting episode. Table 6 presents two weighting episodes based on the sampled trajectories from the previous two sampling episodes. The column named "β value" contains the recent β for each node listed in the "Node in Focus" column, while the third column, "new weight," displays the weight that has been recalculated for the node in focus using the parameter β, as illustrated in Eq. 2.

Table 2. First sampling episode based on the β parameter on snapshot 3a. The $\emptyset$ denotes terminal states with no adjacent nodes.

Index	Focus	Trajectories	β values					
			A	B	C	D	E	F
1	A	$A \to B \to E$ $A \to C \to D$	0.96875	0.970588	0.96875	0.96875	0.96774	1
2	B	$B \to F \to \emptyset$ $B \to E \to \emptyset$ $B \to A \to C$ $B \to D \to C$	0.9375	0.94117	0.90625	0.96875	0.93548	0.96774
3	C	$C \to D \to B$ $C \to A \to B$	0.90625	0.88235	0.875	0.9375	0.93548	0.96774
4	D	$D \to B \to F$ $D \to C \to A$	0.875	0.85294	0.84375	0.90625	0.93548	0.93548
5	E	$E \to B \to D$	0.875	0.8235	0.84375	0.84375	0.90322	0.93548
6	F	$F \to B \to A$	0.84375	0.79411	0.84375	0.84375	0.90322	0.90322

Table 3. Initial weight configurations and decrement rates for each node in the first snapshot 3a.

Node	Decreasing rate	Initial Weight
A	0.03125	0.5
B	0.02941	0.25
C-D	0.03125	0.5
E-F	0.03225	1

Table 4. Initial weight configurations and decrement rates for each node in the snapshot 3b.

Node	New Decreasing rate	Initial weight
A	0.03125	last Known weight
C	0.03225	last Known weight
B-D-F	0.03030	last Known weight
E	0	last Known weight
M	0.03125	0.5

Table 5. Second sampling episode based on the β parameter for snapshot 3b. The $\emptyset$ denotes terminal states with no adjacent nodes.

Index	Focus	Trajectories	β values					
			A	B	C	D	F	M
1	A	$A \to B \to D \to M$ $A \to C \to \emptyset$	0.8125	0.7638	0.81149	0.6884	0.9032	0.96875
2	B	$B \to A \to C \to \emptyset$ $B \to D \to M \to F$ $B \to F \to M \to D$	0.78125	0.73349	0.77923	0.62779	0.87289	0.96774
3	C	$C \to A \to B \to F$	0.75	0.70318	0.74697	0.62779	0.84258	0.90625
4	D	$D \to B \to F \to M$ $D \to F \to M \to \emptyset$ $D \to M \to F \to B$	0.75	0.64257	0.74697	0.59748	0.75167	0.84375
5	F	$F \to B \to A \to C$ $F \to D \to B \to A$ $F \to M \to D \to B$	0.6875	0.55166	0.71471	0.56717	0.72136	0.8125
6	M	$M \to F \to D \to B$ $M \to D \to F \to B$	0.6875	0.49105	0.71471	0.50656	0.66075	0.78125

Table 6. First weighting episode on snapshot 3a and second weighting episode on snapshot 3b

Episode	Node in Focus	β value	New Weight
1	A	0.96875	0.74218
	B	0.94117	2.3382
	C	0.875	0.7119
	D	0.90625	2.3579
	E	0.90322	2.20869
	F	0.90322	2.20869
2	A	0.8125	2.06354
	B	0.73349	2.64910
	C	0.74697	1.72153
	D	0.59748	3.73109
	E	0.90322	2.20869
	F	0.72136	4.85716
	M	0.78125	6.42832

4.6. Dynamic-β-RW with GNN(DynβRW-GNN)

While the Dynamic β-Random Walk (DynβRW) effectively captures temporal consistency across graph snapshots, its performance may remain limited on real-world networks characterized by high structural heterogeneity and sparse local connectivity, such as email communication graphs. In such settings, temporal dynamics alone are often insufficient to fully exploit local topological dependencies. To overcome this limitation, an extension of the

Dynamic β-Random Walk is proposed by incorporating a Graph Neural Network (GNN) as a post-processing refinement module. The key idea is to preserve the temporal stabilization achieved by the DynβRW while enhancing the node representations through localized message passing.

4.6.1. GNN-based Refinement of Dynamic Embeddings: Let $Z = \{\mathbf{z}_1, \mathbf{z}_2, \ldots, \mathbf{z}_n\}$ denote the node embeddings obtained after convergence of the Dynamic β Random Walk using the Full Rotation Convergence strategy. These embeddings encode temporal information accumulated across multiple snapshots.
We use Z as input features to a lightweight GNN, which propagates information along graph edges to capture local structural patterns that are not explicitly modeled by the random walk process. The GNN updates node representations according to

$$\mathbf{h}_v^{(l+1)} = \sigma\left(\mathrm{AGG}^{(l)}\left(\{\mathbf{h}_v^{(l)}\} \cup \{\mathbf{h}_u^{(l)} : u \in \mathcal{N}_v\}\right)\right), \tag{7}$$

where $\mathbf{h}_v^{(0)} = \mathbf{z}_v$, $\mathcal{N}_v$ denotes the set of adjacent nodes to v, $\sigma(\cdot)$ is a non-linear activation function, and $\mathrm{AGG}^{(l)}(\cdot)$ represents a neighborhood aggregation operator.

In our implementation, we adopt a GraphSAGE-based architecture due to its scalability and effectiveness in inductive settings. Only two GraphSAGE layers are used to avoid over-smoothing and to preserve the temporal signal learned by the Dynamic β-Random Walk (DynβRW).

4.6.2. Learning Objective The GNN-refined node embeddings are optimized for the link prediction task, as illustrated in Figure 4. Starting from the temporal embeddings $Z \in \mathbb{R}^{N\times F}$, where N denotes the number of dynamic graph nodes and F the embedding dimension produced by the DynβRW process, we apply a GraphSAGE encoder to obtain refined representations $Y \in \mathbb{R}^{N\times H}$, where H is the hidden dimensionality. Each GraphSage Convolution layer is followed by Batch Normalization, a Leaky ReLU activation function, and a Dropout with rate=0.1. The hidden representation dimension H is set to 512 for both GraphSage layers. The hidden dimension was established via manual hyperparameter optimization on the explored datasets. According to the acquired empirical findings, we opted for a hidden dimension of 512 and maintained this configuration uniformly across all datasets. For a pair (u, v) of nodes, with embeddings (y_u, y_v), we construct an edge-level representation using element-wise operations, the Hadamard product, the absolute difference, and the scalar product.

$$\mathbf{h}_{uv} = \left[y_u \odot y_v \parallel |y_u - y_v| \parallel y_u^\top y_v\right]. \tag{8}$$

With $y_u, y_v \in \mathbb{R}^{512}$, where $\odot$ is the Hadamard product (element-wise product) of dimension 512, $||$ is the absolute difference also of dimension 512, the scalar product of dimension 1, and $\parallel\ \parallel$ is the concatenation of vectors. The resulting vector $h_{uv} \in \mathbb{R}^{1025}$. This representation is then passed to a multi-layer perceptron (MLP) decoder, which outputs a compatibility score s_{uv} indicating the probability of a link between the two nodes. The MLP decoder consists of two dense layers: the first one followed by a Leaky ReLU activation function and Dropout(0.1); the second layer outputs the compatibility score s_{uv}. The model is trained using the BCEWithLogitsLoss function with negative sampling, which encourages higher scores for observed edges and lower scores for non-existent edges. Importantly, the β-Random Walk parameters are fixed in this stage, ensuring a clear distinction between temporal embedding generation and neural refinement.

4.6.3. Discussion The proposed DynβRW-GNN framework does not replace the underlying DynβRW approach. Instead, the GNN acts as a spatial regularizer that complements the temporally stabilized embeddings by incorporating local structural information. This modular design enables a clear analysis of the respective contributions of temporal dynamics and neural message passing, while yielding improved performance on complex dynamic networks.

4.7. Ablation Study

We perform an ablation study to evaluate the contribution of each part of our model, DynβRW. In particular, we examine the effect of the decreasing rate, the initialization strategy, and the transition probability in the random

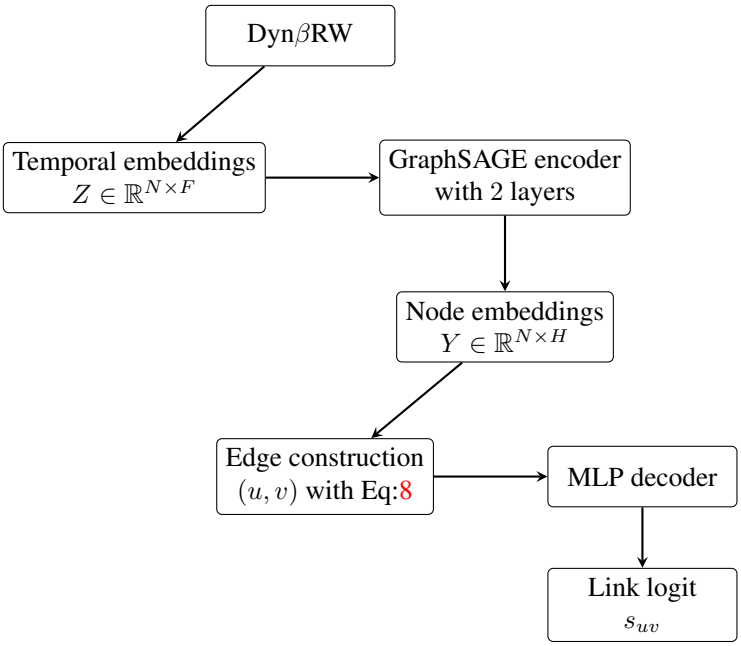


Figure 4. DynβRW-GNN Architecture

walk cite moussaddar2025single. In this setting, each node $v \in \mathcal{N}(u)$ is associated with a temporal importance weight $\beta_v(t)$ that reflects its relevance at time t.

The probability of transition from a node u to one of its neighbors v is then defined as:

$$P_t(u \to v) = \frac{\beta_v(t)}{\sum_{k\in\mathcal{N}(u)} \beta_k(t)}$$

This formulation biases the random walk toward nodes with higher temporal importance, allowing the process to capture the dynamics of evolving graphs more effectively than uniform random walks.

The decreasing rate associated with node v in our dynamic formulation is defined as: $\frac{1}{c+\deg(v)}, c \in \mathbb{R}$ which differs from the static β-Random-Walk formulation: $\frac{1}{n+\deg(v)}$, where n denotes the total number of nodes in the graph G. In the static setting, the term $n + \deg(v)$ approximates the maximum number of visits to node v before convergence. However, for large-scale graphs, this formulation substantially increases convergence time and computational cost. By replacing n with a predetermined constant $c \in \mathbb{R}$ that we choose based on each dataset, our dynamic formulation accelerates convergence while preserving the influence of node degree on the transition dynamics. For each dataset, we adjust c from 0 to 200 and execute the DynβRw, selecting the constant c that yields the optimal AP for both logistic regression and cosine similarity. Figure **??** illustrates the outcomes for the UCI dataset, revealing that $c = 60$ yields the optimal AP. We perform the same procedure for the three datasets, resulting in the appropriate c value as illustrated in Table 11.

5. Experimental Results

5.1. Training Strategy and Convergence Scheme

In dynamic graph embedding, the choice of convergence strategy is crucial for the stability and efficiency of the optimization process. In this work, we investigated two alternative training schemes for the dynamic β-Random Walk model. The first strategy enforces convergence at each snapshot before proceeding to the next one. The

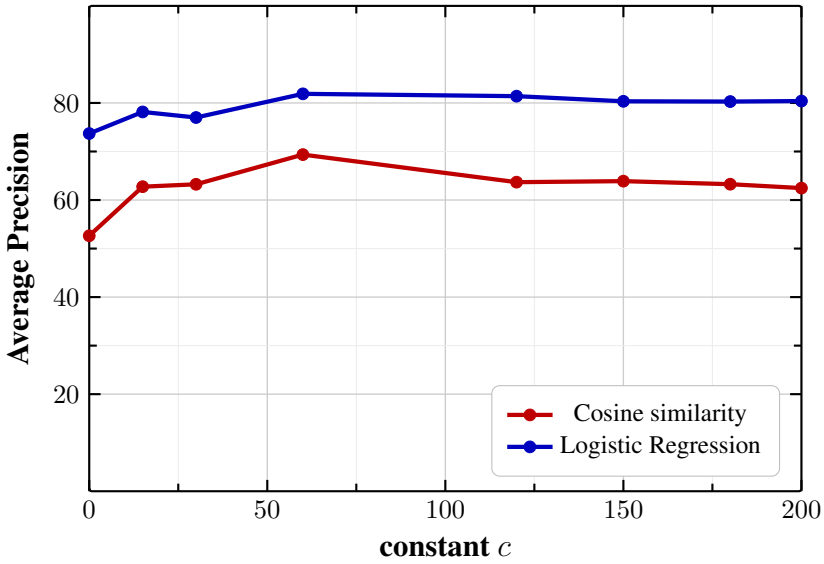


Figure 5. Evolution of Average Precision on the UCI dataset for different values of the constant c.

second strategy performs cyclic updates across all snapshots before checking global convergence. This section reports the outcomes of using the β-Random Walk model on dynamic graphs extracted from the High-School data 2013 dataset. The analysis focuses on embedding stability, convergence behavior, and node weight dynamics. Snapshots were extracted from the temporal graph data to serve as the basis for experimentation. The generated dynamic graph contains 246 snapshots and 327 nodes.
The β-Random Walk model was applied in two convergence settings: "Full Snapshot Convergence and Full Rotation Convergence".

Full Snapshot Convergence

In the Full Snapshot Convergence setting, the β-RW model was applied to each graph snapshot in a single training cycle. The embeddings generated in this setting exhibited instability, with the appearance of NaN values—indicating an explosion in weight values during the learning process.

Node	β	**LastWeight**
454	1.0000	NaN
640	1.0000	NaN
1	1.0000	5.806×10^{280}
939	1.0000	NaN
185	1.0000	NaN
258	1.0000	NaN
55	1.0000	NaN
170	1.0000	NaN
9	1.0000	NaN
453	1.0000	NaN

Table 7. Full Snapshot convergence weight explosion

Full rotation Convergence A rotation convergence strategy was employed as an alternative. This approach significantly mitigated the issue of weight explosion. The model converged at the 98th episode, and the embeddings remained stable throughout the training cycle. These results showed the effectiveness of rotation-based execution in controlling the dynamic behavior of node weights.

Result: We observe that the *Full Snapshot Convergence* model, which processes every snapshot until convergence, is ineffective due to its tendency toward weight explosion and excessive time and resource consumption. That's why we choose the *Full rotation convergence* model, which iteratively passes on snapshots until convergence, explained in detail in Algorithm 1. In the conducted experiment, the *Full rotation convergence* is shown in snapshot 62 of episode 98.

Justification of the Selected Strategy: The problem of weight explosion in *Full Snapshot Convergence* comes from the inheritance of weights from snapshots combined with a strong disparity between old and newly added nodes. In a particular snapshot G_t, a highly active node u gathers a considerable weight $\psi_v(t)$ following convergence. In the subsequent snapshot G_{t+1}, a novel node u connected to v (which is absent from prior snapshots) possesses an initial weight $\psi_u(0) = \frac{1}{deg_{G_{t+1}}(u)} \leq 1$, resulting in a substantial disparity in weights between u and v. The weights of both u and v experience a significant increase since β is reset to 1 in this context, and the weight equation 2 relies on the disparity between node weights, resulting in a weight explosion after T snapshots, which recurs for every pair of connected nodes appearing in two separate snapshots. The convergence for each snapshot results in disproportionate weights, whereas *Full rotation convergence* mitigates this issue by facilitating a gradual rebalancing of weights across all snapshots; we transition from one snapshot to the next without pausing for convergence, ensuring that the weights stay balanced and convergence is achieved in a single step. The complexity study of *Full rotation convergence* reveals that it is $O(k \times d \times n \times e \times T)$, while the complexity of *Full Snapshot Convergence*is $O((k \times d \times n \times e)^T)$. The second reason for choosing the *Full rotation convergence* lies in the extensive running time and high dimensionality of the resultant embeddings in the initial method, as the frequency of visits to each node is amplified by T.

5.2. Experimental Setup

5.2.1. Datasets To evaluate the effectiveness of our approach, we conduct experiments on three real-world datasets. The main statistics of the datasets are summarized in Table 8.
Wikipedia[†] is a dataset of edits made to wiki pages over the course of a month, with nodes symbolizing human editors and wiki pages, and links indicating timestamped edits.
Enron [‡] is a communication network comprised of email exchanges among core employees of a corporation over several years.
UCI [§] is a dataset documenting online contributions by university students on a forum, yet remains unattributed

Datasets	# Nodes	# Temporal links	# of snapshots	# exact split
Wikipedia	9,227	157,474	21	16/5
Enron	184	125,235	11	9/2
UCI	1,899	59,835	88	68/20

Table 8. Summary of dataset statistics.

5.2.2. Implementation Details: This section describes the experimental setup and implementation choices necessary to reproduce our results. We detail the dataset pre-processing, snapshot construction, temporal weighting procedure, embedding configuration, training protocol, and evaluation methodology. We use three datasets, as seen in 8. Each dynamic graph is portioned into **T** discrete snapshots according to chronological order. Every snapshot $G_t = (V_t, E_t)$ contains edges whose timestamps fall within the interval t. We use the first 80 % of snapshots for training and the remainder for the test set; this division is chronological. Details are presented in Table 8. Then the DynβRW method is applied to the training and test sets. The dimension of output embeddings is adjusted to

[†]http://snap.stanford.edu/jodie/wikipedia.csv
[‡]https://www.cs.cmu.edu/ ./enron/
[§]http://konect.cc/networks/opsahl-ucforum/

match using a zero-padding strategy. $\mathbf{E}$ denotes the set of all edges in all snapshots. The training positive edges are obtained from the union of all edges appearing in the training snapshots.

$$\mathbf{E^+_{train}} = \bigcup_{t=1}^{T_{train}} \mathbf{E_t} \tag{9}$$

Each distinct edge is included only once, regardless of the number of snapshots in which it appears. The training negative edges are randomly generated from the training node set that are not connected by any edge in $\mathbf{E^+_{train}}$. The number of negative edges is controlled by a predefined negative-to-positive ratio (1:1 in our experiments). Consequently,

$$\mathbf{E^-_{train}} \subseteq (\mathbf{V_{train}} \times \mathbf{V_{train}}) \backslash \mathbf{E^+_{train}} \tag{10}$$

The positive and negative edges are then merged and randomly shuffled to form the final training dataset.

Test set construction: The test positive edges are constructed from the union of all edges appearing in the test snapshots:

$$\mathbf{E^+_{test}} = \bigcup_{t=T_{train}+1}^{T} \mathbf{E_t} \tag{11}$$

To generate the test negative edges, node pairs are randomly sampled from the training node set while excluding all edges that appear either in the training positives or the test positives. Therefore, the negative test set satisfies:

$$\mathbf{E^-_{test}} \subseteq (\mathbf{V_{train}} \times \mathbf{V_{train}}) \backslash (\mathbf{E^+_{train}} \cup \mathbf{E^+_{test}}) \tag{12}$$

As in the training phase, the positive and negative edges are combined according to the specified negative sampling ratio and randomly shuffled before evaluation. This protocol follows the *transductive* setting because all candidate node pairs are generated exclusively from nodes observed during the training period. No unseen nodes are introduced during testing; only the presence or absence of future interactions between known nodes is predicted.

For the first method version, without a GNN, link prediction is evaluated using two different scoring approaches: cosine similarity and logistic regression with Hadamard features. Cosine similarity and logistic regression are performed between node embeddings of the test set and used as a link prediction scoring function. Given two nodes u and v with embeddings z_u and z_v, the cosine similarity score is defined as:

$$s(u, v) = \frac{z_u z_v^T}{||z_u||_2 ||z_v||_2} \tag{13}$$

This score is used directly to rank node pairs for link prediction. A pair with a similarity score surpassing a predetermined threshold — calculated as the 25th percentile of similarity scores from the training dataset — is predicted as an existing edge. For a pair of nodes (u, v), we construct a pair descriptor using the Hadamard product (element-by-element product) of the two embedding vectors: $h_{uv} = z_u \odot z_v$. Then a logistic regression classifier is trained over this h_{uv}, after standardization of features (mean zero, variance one), to estimate the probability of a relationship existing:

$$P(y_{uv} = 1 \mid h_{uv}) = \sigma(w^\top h_{uv} + b) \tag{14}$$

where σ denotes the sigmoid function, w denotes the weight vector, and b denotes the bias, both of which are learned by maximizing the likelihood (minimizing the log-loss). Then this probability is used to estimate whether an edge exists between u and v. For the second method, DynβRW-GNN, we use a graph neural network decoder as explained in Section 4.6.1.

5.2.3. Evaluation Metrics: The efficacy of our approach is assessed using two widely adopted evaluation metrics for link prediction in dynamic graphs: Area Under the Receiver Operating Characteristic Curve (AUC) and Average Precision (AP). The **AUC** measures the likelihood that a randomly chosen positive edge is assigned a higher prediction score than a randomly chosen negative edge. It evaluates the model's ranking capability without reliance

on a predetermined classification threshold. An **AUC** value of 50 corresponds to random predictions, while a value close to 100 indicates perfect discrimination between positive and negative edges. The **AP** summarizes the precision-recall curve and evaluates the ranking of predicted links. It accounts for the order of predicted edges and gives greater weight to correctly ranking positive edges at the top of the list. Higher **AP** values indicate better link prediction performance.

5.3. Results without GNN

Initially, we evaluate the proposed framework without the GNN component by using two different link prediction strategies based on the generated node embeddings: cosine similarity and logistic regression. Tables 9 and 10 report the results obtained with DynβRW using cosine similarity and logistic regression, respectively. The relatively high AP and AUC scores obtained with logistic regression indicate that the generated embeddings contain discriminative information that can be effectively exploited for link prediction. In contrast, the relatively low AP and AUC scores obtained with cosine similarity suggest that direct similarity between node embeddings is not sufficient to reliably distinguish positive from negative links. This difference indicates that the information encoded in the embeddings cannot be fully captured through a simple proximity measure and may require a learned decision function. These limitations motivate the integration of a GNN component to improve link prediction performance and more effectively leverage the graph's structural dependencies. Table 11 illustrates the selected c value for each dataset using the same strategy as explained in the ablation study section4.7 and the execution time for each dataset employing the DynβRW approach without GNN.

Dataset	AP	AUC
Enron	55.53	57.14
Wikipedia	61.08	63.19
UCI	69.36	67.29

Table 9. AUC and AP with cosine similarity

Dataset	AP	AUC
Enron	81.04	79.77
Wikipedia	84.87	86.14
UCI	79.36	78.60

Table 10. AUC and AP with Logistic regression

Dataset	c value	Runtime
Enron	40	2.83s
Wikipedia	120	10.67s
UCI	60	7.11s

Table 11. The selected c value and runtime of the DynβRW technique for each dataset

"'latex

Figure 6 presents the temporal evolution of a node's weights in the Enron dataset, demonstrating the stability and convergence of the learned embedding vector.

5.4. Results with GNN

Tables 12 and 13 present the future link prediction performance on transductive settings measured by AUC and AP metrics on Enron, Wikipedia, and UCI datasets. All experiments were conducted 10 times, and the results

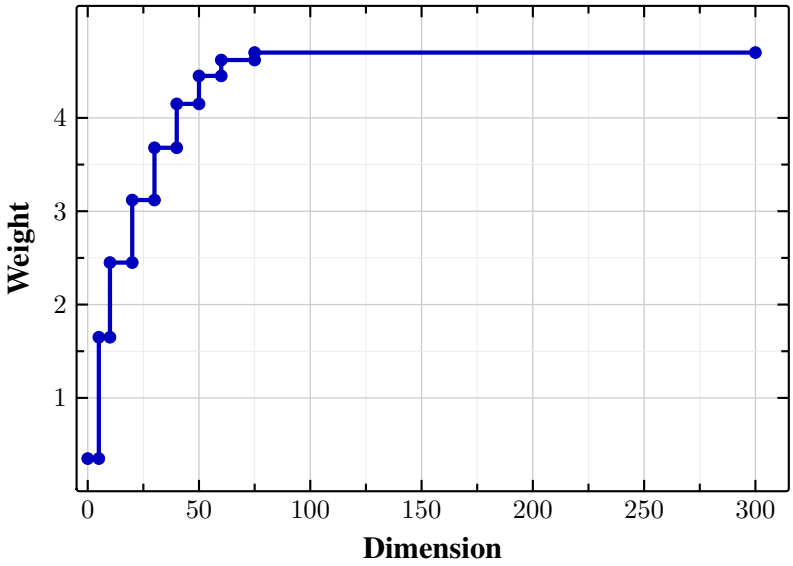


Figure 6. Evolution of embedding weights across dimensions on the ENRON dataset. The curve saturates rapidly after dimension 75, indicating that most representational capacity is captured in the first few dimensions.

Model	Enron	Wikipedia	UCI
DynAERNN	60.90±2.70	76.58±1.41	84.95±2.13
JODIE	91.94±0.01	91.39±0.04	80.27±0.08
dyREp	97.14±0.07	77.36±0.25	95.29±0.13
VGRNN	94.02±0.52	67.66±0.89	82.23±0.56
EvolveGCN	69.72±0.49	55.84±0.37	81.63±0.23
TGAT	57.37±0.18	96.65±0.06	60.25±0.31
DynβRW-GNN	95.38±0.14	95.88± 0.04	87.22±0.5

Table 12. Future Link Prediction Performance in AP

Model	Enron	Wikipedia	UCI
DynnAERNN	63.11±1.13	71.00±1.10	83.72±1.79
JODIE	89.36∓0.06	88.43±0.02	74.63±0.11
dyREp	96.71±0.04	77.40±0.13	95.23±0.25
VGRNN	79.08±0.02	71.20±0.65	89.43±0.27
EvolveGCN	74.02±0.31	60.48±0.47	78.30±0.22
TGAT	60.88±0.37	96.36±0.05	77.67±0.27
DynβRW-GNN	93.43±0.18	96.48± 0.07	88.03±1.05

Table 13. Future Link Prediction Performance in AUC

provided are expressed as the average and standard deviation across the 10 iterations. The proposed DynβRW-GNN model achieves competitive results on all datasets. Our method gets an AP of 95% and an AUC of 93% on the Enron dataset. This is better than most baseline methods and almost as good as the best model (dyRep) in terms of AP and AUC. DynβRW-GNN gets an AP of 95% and an AUC of 96% on the Wikipedia dataset. This is similar to, or better than, many other methods already available. The model also achieves an AP of 87% and an AUC of 88% on the UCI dataset, demonstrating that it performs well across diverse dynamic graph reality-based datasets. Comparing those results to Table 10 indicates that adding the GNN component makes the model much

better at finding temporal and structural dependencies in time-evolving graphs, which leads to more accurate link predictions than the method that only uses embedding similarity.

6. Conclusion

In this paper, we propose DynβRW-GNN, a dynamic graph embedding framework combining β-random walk-based temporal exploration with a GNN-based refinement mechanism. Unlike snapshot-based or continuous GNN architectures, our approach decouples temporal exploration from learning, enabling more flexible and interpretable representations of evolving graphs. We first evaluated the DynβRW method without the GNN component, and the results showed that it can provide promising node representations for link prediction. We then incorporated the GNN-based refinement mechanism. The experimental findings indicate that the proposed framework enhances representation quality and link prediction performance. However, a key limitation lies in the use of a fixed decay constant in the β decrease rate: $\frac{1}{c+deg(v)}$, where c is a dataset-dependent constant. The decreasing rate controls the exploration depth per node. This constant requires manual tuning for each graph to balance sufficient exploration and avoid overfitting, making the framework less generalizable out of the box. Future work will focus on making this constant adaptive or learnable based on local graph properties such as node degree or snapshot density. Additional directions include enhanced integration with graph neural networks and applications to dynamic systems such as predictive maintenance and disease progression modeling.

Acknowledgements

The authors thank the Laboratory of Artificial Intelligence, Modeling and Computational Engineering (AIMCE) at ENSAM, Hassan II University, for providing computational resources and administrative support.
A heartfelt appreciation to the anonymous reviewers for their pertinent observations and recommendations that significantly enhanced our work.

REFERENCES

1. S. Kumar, A. Mallik, A. Khetarpal and B. S. Panda, *Influence maximization in social networks using graph embedding and graph neural network*, Information Sciences, vol. 607, pp. 1617–1636, 2022.
2. M. Xu, P. Qian, Z. Zhao, Z. Zeng, J. Chen, W. Liu and X. Yang, *Graph Neural Networks for Protein-Protein Interactions – A Short Survey*, arXiv preprint arXiv:2404.10450, 2024.
3. G. Madaan and S. Jolad, *Evolution of scientific collaboration networks*, IEEE International Conference on Big Data, pp. 7–13, 2014.
4. G. Xue, M. Zhong, J. Li, J. Chen, C. Zhai and R. Kong, *Dynamic network embedding survey*, Neurocomputing, vol. 472, pp. 212–223, 2022.
5. B. Perozzi, R. Al-Rfou and S. Skiena, *DeepWalk: Online Learning of Social Representations*, Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 701–710, 2014.
6. A. Grover and J. Leskovec, *node2vec: Scalable Feature Learning for Networks*, Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 855–864, 2016.
7. C. D. T. Barros, M. R. F. Mendonça, A. B. Vieira and A. Ziviani, *A Survey on Embedding Dynamic Graphs*, ACM Computing Surveys, vol. 55, no. 1, pp. 1–37, 2021.
8. P. Goyal and E. Ferrara, *Graph embedding techniques, applications, and performance: A survey*, Knowledge-Based Systems, vol. 151, pp. 78–94, 2018.
9. X. Liu, J. Chen and Q. Wen, *A survey on graph classification and link prediction based on GNN*, arXiv preprint arXiv:2307.00865, 2023.
10. B. Hirchoua and S. El Motaki, *β-random walk: Collaborative sampling and weighting mechanisms based on a single parameter for node embeddings*, Pattern Recognition, vol. 142, 109730, 2023.
11. A. Moussaddar, B. Hirchoua and I. Guelzim, *Single-Parameter Link Prediction: Balancing Sampling and Node Weighting*, International Conference on Artificial Intelligence and Cognitive Science for Emerging Technologies, pp. 411–421, 2025.
12. F. Bérès, D. M. Kelen, R. Pálovics and A. A. Benczúr, *Node embeddings in dynamic graphs*, Applied Network Science, vol. 4, no. 1, 64, 2019.
13. X. Yue et al., *Graph embedding on biomedical networks: methods, applications and evaluations*, Bioinformatics, vol. 36, no. 4, pp. 1241–1251, 2020.
14. J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan and Q. Mei, *LINE: Large-scale Information Network Embedding*, Proceedings of the 24th International Conference on World Wide Web, pp. 1067–1077, 2015.

15. S. Cao, W. Lu and Q. Xu, *GraRep: Learning Graph Representations with Global Structural Information*, Proceedings of the 24th ACM International Conference on Information and Knowledge Management, pp. 891–900, 2015.
16. D. Wang, P. Cui and W. Zhu, *Structural Deep Network Embedding*, Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 1225–1234, 2016.
17. K. B. Enes, M. Nunes, F. Murai and G. L. Pappa, *Temporally-aware node embeddings for evolving network topologies*, AI Communications, vol. 37, no. 3, pp. 411–428, 2024.
18. P. Goyal, N. Kamra, X. He and Y. Liu, *DynGEM: Deep Embedding Method for Dynamic Graphs*, arXiv preprint arXiv:1805.11273, 2018.
19. L. Zhou, Y. Yang, X. Ren, F. Wu and Y. Zhuang, *Dynamic network embedding by modeling triadic closure process*, Proceedings of the AAAI Conference on Artificial Intelligence, vol. 32, no. 1, 2018.
20. S. Mahdavi, S. Khoshraftar and A. An, *dynnode2vec: Scalable Dynamic Network Embedding*, arXiv preprint arXiv:1812.02356, 2019.
21. M. A. Alomrani, M. Biparva, Y. Zhang and M. Coates, *DyG2Vec: Efficient Representation Learning for Dynamic Graphs*, arXiv preprint arXiv:2210.16906, 2024.
22. A. Sankar, Y. Wu, L. Gou, W. Zhang and H. Yang, *DySAT: Deep Neural Representation Learning on Dynamic Graphs via Self-Attention Networks*, Proceedings of the 13th International Conference on Web Search and Data Mining, pp. 519–527, 2020.
23. P. Goyal, S. R. Chhetri and A. Canedo, *dyngraph2vec: Capturing Network Dynamics using Dynamic Graph Representation Learning*, Knowledge-Based Systems, vol. 187, 104816, 2020.
24. 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, vol. 34, no. 8, pp. 3841–3853, 2022.
25. M. Haddad, C. Bothorel, P. Lenca and D. Bedart, *Temporalnode2vec: Temporal node embedding in temporal networks*, International Conference on Complex Networks and Their Applications, pp. 891–902, 2019.
26. W.-L. Chiang, X. Liu, S. Si, Y. Li, S. Bengio and C.-J. Hsieh, *Cluster-GCN: An Efficient Algorithm for Training Deep and Large Graph Convolutional Networks*, Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, pp. 257–266, 2019.
27. E. Hajiramezanali, A. Hasanzadeh, K. Narayanan, N. Duffield, M. Zhou and X. Qian, *Variational Graph Recurrent Neural Networks*, Advances in Neural Information Processing Systems, vol. 32, 2019.
28. A. Pareja, G. Domeniconi, J. Chen, T. Ma, T. Suzumura, H. Kanezashi, T. Kaler, T. Schardl and C. Leiserson, *EvolveGCN: Evolving Graph Convolutional Networks for Dynamic Graphs*, Proceedings of the AAAI Conference on Artificial Intelligence, vol. 34, no. 04, pp. 5363–5370, 2020.
29. K. Cheng, J. Ye, X. Lu, L. Sun and B. Du, *Temporal Graph Network for continuous-time dynamic event sequence*, Knowledge-Based Systems, vol. 304, 112452, 2024.
30. E. Rossi, B. Chamberlain, F. Frasca, D. Eynard, F. Monti and M. Bronstein, *Temporal Graph Networks for Deep Learning on Dynamic Graphs*, arXiv preprint arXiv:2006.10637, 2020.
31. I. J. King and H. Huang, *EULER: Detecting Network Lateral Movement via Scalable Temporal Link Prediction*.
32. A. Fathy and K. Li, *TemporalGAT: Attention-Based Dynamic Graph Representation Learning*, Advances in Knowledge Discovery and Data Mining, vol. 12084, pp. 413–423, 2020.
33. C. Huang, L. Wang, X. Cao, W. Ma and S. Vosoughi, *Learning dynamic graph embeddings using random walk with temporal backtracking*, NeurIPS 2022 Temporal Graph Learning Workshop, 2022.
34. G. H. Nguyen, J. B. Lee, R. A. Rossi, N. K. Ahmed, E. Koh and S. Kim, *Continuous-Time Dynamic Network Embeddings*, Companion of The Web Conference 2018, pp. 969–976, 2018.
35. S. Kumar, X. Zhang and J. Leskovec, *Predicting Dynamic Embedding Trajectory in Temporal Interaction Networks*, Proceedings of the 25th ACM SIGKDD International Conference, pp. 1269–1278, 2019.
36. R. Trivedi, M. Farajtabar, P. Biswal and H. Zha, *DyRep: Learning Representations Over Dynamic Graphs*, International Conference on Learning Representations, 2019.
37. D. Xu, C. Ruan, E. Korpeoglu, S. Kumar and K. Achan, *Inductive Representation Learning on Temporal Graphs*, arXiv preprint arXiv:2002.07962, 2020.
38. Z. Wang, Z. Zhang, T. Ma, N. V. Chawla, C. Zhang and Y. Ye, *Beyond message passing: Neural graph pattern machine*, arXiv preprint arXiv:2501.18739, 2025.
39. Goyal, Palash and Chhetri, Sujit Rokka and Canedo, Arquimedes, *dyngraph2vec: Capturing network dynamics using dynamic graph representation learning*, Knowledge-Based Systems, volume=187, pages=104816, year=2020, publisher=Elsevier
40. Yassine, Senhaji and Abdelfattah, Oulaika and Boujemaa, EL Marnissi and Hafidi, Youssef and others, *Relation type-aware knowledge graph embeddings for biomedical data: a semantically adaptive framework*, Statistics, Optimization & Information Computing, volume=15, number=6, pages=5459–5471, year=2026.
41. Hirchoua, B., Ouhbi, B., and Frikh, B., *Topic hierarchies for knowledge capitalization using hierarchical Dirichlet processes in big data context*, International Conference on Advanced Intelligent Systems for Sustainable Development, pages=592–608, year=2018.
42. El Motaki, S., and Hirchoua, B., *A Novel Deep Learning Architecture Based IoT Time-Series for Energy Consumption Forecasting in Smart Households*, AI and IoT for Sustainable Development in Emerging Countries, Lecture Notes on Data Engineering and Communications Technologies, volume=105, year=2022.
43. Hirchoua, B., Ouhbi, B., and Frikh, B., *A new knowledge capitalization framework in big data context*, Proceedings of the 19th International Conference on Information Integration and Web-based Applications & Services (iiWAS '17), pages=40–48, year=2017.
44. Hafid, M., Abdelalim, S., Karim, D., and Elmouki, I., *Graph Optimization and Programming through Interval Reduction: Case for Secure Control Flow Graph Analysis*, Statistics, Optimization & Information Computing, volume=16, number=1, pages=749, year=2026, publisher=International Academic Press (Hong Kong).