2024 IEEE 48th Annual Computers, Software, and Applications Conference (COMPSAC) | 979-8-3503-7696-8/24/$31.00 ©2024 IEEE | DOI: 10.1109/COMPSAC61105.2024.00079

Node Embedding Accelerates Randoms Walk on a Graph

Han Nay Aung, Hiroyuki Ohsaki
Graduate School of Science and Technology
Kwansei Gakuin University
email: {hannayaung,ohsaki}@kwansei.ac.jp

***Abstract*—Graphs serve as powerful representations for various real-world systems such as social networks, biological networks, and communication networks. Random walk algorithms have gained popularity for graph-based data analysis and processing, finding applications across various domains. Understanding and enhancing these algorithms is crucial for ensuring high-quality protocols, controls, and services in large-scale communication networks. While conventional random walks typically rely on local information, there is potential to improve node search efficiency by incorporating information beyond the local context. Concurrently, there is growing interest in machine learning techniques that represent data as graphs rather than vectors, known as graph and node embedding algorithms. This paper investigates whether leveraging node embedding vectors generated by such techniques can enhance the efficiency and effectiveness of random walks on a graph. To address this, we propose EmbedRW (Embedded Random Walk), which integrates node embedding techniques with random walk design. Through simulation experiments, we demonstrate that utilizing node embeddings can significantly reduce the search time for the target node across a wide range of graphs.**

***Index Terms*—EmbedRW (Embedded Random Walk), random walk, node embedding, feature learning, hitting time, cover time**

I. Introduction

Graphs are a powerful tool for representing various real-world systems such as social networks, biological networks, traffic networks, citation networks, and communication networks [1, 2]. They enable us to visually describes and analyze complex relationships and structures.

Algorithms based on random walks have become popular for graph-based data analysis and processing [3, 4]. A random walk of an agent is a random process that describes a path by taking a series of random steps in a mathematical space. In the mathematical context, a simple random walk model is a random walk on a regular lattice, where at each step, the agent at a point can jump to another point based on a specific probability distribution. When we apply this model to a specific graph, like a network, the likelihood of transition between nodes is directly related to how strongly they are connected. The stronger the connection between nodes, the higher are the chances of traveling from one to the other. After enough steps, a random path that effectively represents the network structure is generated [5].

Random walks are widely applied to a large number of problems in various domains. For example, they play an important role in computer science [6-13]. Random walks are used in target node search, graph exploration, node embedding [7, 8], clustering [9], and sampling [10]. For instance, random walks have been applied for load balancing in peer-to-peer networks [11], routing protocols in wireless sensor networks [12], and in transfer machine learning [13].

Understanding and enhancing random walks on a graph is crucial for achieving high-quality and reliable protocols, controls, applications, and services in large-scale communication networks [14, 15]. In small-scale and static communication networks, traditional deterministic algorithms may suffice in many cases. However, in large-scale and dynamic communication networks, an entity in the network cannot know global knowledge on the entire network. Such an entity can only know partial knowledge (i.e., local information) in their vicinity. For instance, in dynamic large-scale networks, algorithms for static graphs such as BFS (breadth first search) and DFS (depth first search) are virtually useless due to the dynamics of the network. In dynamic large-scale networks, a class of algorithms based on random walks, relying solely on local information of an agent (i.e., random walker), is often the only viable solution [16, 17].

Random walks on a graph serve as the foundation in various essential technologies to enhance the Quality of Service (QoS) and Quality of Experience (QoE) in large-scale and high-performance communication networks. Examples of the use of random walks in the network layer of challenging large-scale networks include: graph entropy [18], network topology discovery, network sampling [19], message routing [20], network resource discovery, and node clustering [21]. Furthermore, in the application layer, examples of their use include applications and services, for instance, in large-scale content network (e.g., Web) and social networks: network sampling [19], node ranking [22], page ranking [23], node classification [24], node matching [8], and node embedding [7]. Note that simple random walks on a graph are simple and tractable but inefficient for most practical applications; instead, a variety of random walk-based algorithms such as random walks with history and random walks with intelligence is utilized for significantly better efficiency and reliability.

Various types of random walks have been proposed, and each is characterized by its unique attributes. These include simple random walk (SRW), biased random walk (BiasedRW) [25], non-backtracking random walk (NBRW) [26],

2836-3795/24/$31.00 ©2024 IEEE
DOI 10.1109/COMPSAC61105.2024.00079

self-avoiding random walk (SARW) [27], and vicinity-avoiding random walk (VARW) [28].

On the other hand, In machine learning, there is growing interest in techniques that allow data to be represented not as vectors but as graphs. Graph and node embedding are major techniques used to represent a graph as a vector [7, 8]. Graph embedding techniques enable the modeling of graphs as vectors, which is beneficial for machine-learning tasks such as node classification, clustering, and missing link prediction on incomplete graphs.

Traditional random walks, such as SRW, NBRW, and SARW, typically involve simple transitions from one node to a neighboring node, and they are based on local information that mobile agents can observe. We consider a scenario where an mobile agent exploring an unknown graph not only has access to the neighboring nodes of the node they are currently visiting but can also retrieve the embedding vectors of neighbor nodes, which are generated by a node embedding technique. For example, in social network analysis, where individual nodes represent people or entities and the edges between nodes represent relationships, when mobile agents explore social networks, using node embeddings allows them to access information about each node's profile as well as interests (represented as an embedding vector). This enables mobile agents to infer more suitable people or entities to look for in specific context.

In this paper, we aim to answer the following research questions.

- In the context of random walks on a graph, how can node embedding vectors be effectively utilized to enhance target node search and graph exploration on a unknown graph?
- To what extent does the utilization of embedding vectors of the target node and neighboring nodes by a mobile agent contribute to the improvement of search time and exploration time?
- From a perspective of target node search and graph exploration with random walks on a graph, what type embedding vectors are preferable?

In this paper, we consider a scenario where a mobile agent exploring on an unknown graph not only has access to the neighboring nodes of the node they are currently visiting but can also retrieve the embedding vectors of each of these neighboring nodes. We aim to achieve efficient target node search and graph exploration by incorporating the embedding vectors of nodes into the decision-making process. Toward this end, we propose Embedded Random Walk (EmbedRW), a random walk algorithm that utilizes node embedding vectors. Through the integration of node embedding vectors into the decision-making process, we aim to optimize the selection of the next node to visit during the walk.

The main contributions of this paper are summarized as follows.

- We propose a class of random walk algorithms called EmbedRW on a graph utilizing node embedding for accelerating target node search and graph exploration.
- Through simulation experiments, we quantitatively reveal that EmbedRW can significantly reduce the search time with random walk algorithms in wide range of graphs.
- We reveal that the efficiency of EmbedRW is not significantly affected by the choice of the node embedding technique; i.e., major node embedding techniques such as Node2vec and DeepWalk works equally well.

The rest of this paper is organized as follows. In Section II, we provide an overview of related works on random walks on a graph and node embedding techniques. In Sections III and IV, we initially outline the problem and then introduce our random walk algorithm called EmbedRW. In Section V, we conduct experiments to investigate the performance of EmbedRW. Finally, in Section VI, we summarize this paper and discuss future research directions.

II. Related Works

This section offers a brief introduction to existing random walks on unknown graphs. The complexity of each random walk is summarized in Table I. Additionally, it provides a brief introduction to major node embedding techniques.

A. Random walks

Simple Random Walk (SRW) is a fundamental random walk where a mobile agent traverses randomly from one node to another without memory or restrictions on revisiting nodes. At each step, the mobile agent selects a neighboring node with equal probability, potentially revisiting nodes and crossing its own path.

Biased Random Walk (BiasedRW) is another random walk variant, where the probability of a mobile agent transitioning from a node to one of its neighbors is determined by the neighbor's degree k raised to the power of α. This parameter α controls the bias [25].

Non-Backtracking Random Walk (NBRW) [26] closely resembles SRW but with a crucial difference: it aims to avoid returning to the previously visited node whenever possible. By incorporating short-term memory into its traversal, NBRW demonstrates advantageous properties compared to SRW.

Self-Avoiding Random Walk (SARW) [27] represents a more stringent variant of random walk, where the mobile agent abstains from revisiting any previously visited node throughout the entire walk, thereby ensuring a unique, non-intersecting path. SARW demands significant memory resources as it necessitates maintaining a record of all visited nodes to prevent revisits. This characteristic makes SARW particularly advantageous in applications where maintaining a non-intersecting path is imperative.

Vicinity-Avoiding Random Walk (VARW) [28] shares similarities with both NBRW and SARW, as the mobile agent retains memory of previously visited nodes. However, unlike SARW, VARW does not record all visited nodes. Instead, it aims to prevent the mobile agent from revisiting nodes it encountered in the past t time steps, as well as their neighboring nodes, whenever feasible.

These random walk variants offer different properties and complexities, catering to various applications and scenarios.

TABLE I
RANDOM WALK ALGORITHMS

algorithm		reference	agent capability	description
Simple Random Walk	SRW		low	Basic random walk with no memory or restrictions on revisiting nodes
Biased random walk	BiasedRW	[25]	medium	Does not require memory but require some computation for calculating biased probabilities
Non-backtracking random walk	NBRW	[26]	medium	Requires minimal memory to avoid revisiting recent nodes
Self-avoiding random walk	SARW	[27]	high	Avoids revisiting any previously visited node by recording visited nodes, requiring extensive memory
Vicinity-avoiding random walk	VARW	[28]	high	Avoids visited nodes and their neighbors within a time window, needing memory management

B. Node embeddings

In this subsection, we provide a concise overview of node embedding techniques, which encompass both node embedding and graph embedding methods. Graph embedding involves mapping the intricate structure of a graph, comprising nodes and edges, into a vector space.

Node embedding, a subset of graph embedding, focuses on transforming individual nodes within a graph into high-dimensional vector representations. These vectors encapsulate crucial structural and semantic information about the nodes and their relationships within the graph, rendering them valuable for various machine learning tasks such as link prediction, node classification, and community detection [29]. Node embedding can be generated using various methods, including matrix factorization, random walks, and deep learning [8]. This area has witnessed considerable research interest, resulting in a plethora of methodologies, including renowned techniques like Word2Vec [30], DeepWalk [31], and Node2vec [32].

Among these methods, DeepWalk and Node2vec stand out as sophisticated graph embedding techniques that have garnered extensive attention [33-35]. DeepWalk employs simple random walks, where the probability of transitioning from one node to another is uniform, to capture structural information within a graph. On the other hand, Node2vec is a semi-supervised algorithm that utilizes random walks to represent each node in a graph as a high-dimensional vector. This approach yields a mapping of nodes into a low-dimensional space while preserving essential network properties.

Consider a graph denoted as $G = (V, E)$, where V represents the set of nodes and E is the set of edges. Node2vec constructs vector representations for each node in graph G through the following process. Similarly, DeepWalk employs random walks based on a uniform probability distribution, implying that the probability of transitioning from one node to another is equal.

The Node2vec algorithm initiates by selecting a specific start node u from V and defining a predetermined walk length (step) l. It then conducts a random walk of length l starting from node u and records all encountered nodes during the walk. Let the current node at the n-th step be denoted as v, its neighboring node as x, and the node at the $(n-1)$-th step as t. The transition probability to node x, denoted as π_{vx}, is calculated using the equation:

$$\pi_{vx} = \alpha_{pq}(t, x) w_{vx} \tag{1}$$

$$\alpha_{pq}(t, x) = \begin{cases} \frac{1}{p} & \text{if } d_{tx} = 0 \\ 1 & \text{if } d_{tx} = 1 \\ \frac{1}{q} & \text{if } d_{tx} = 2 \end{cases} \tag{2}$$

where d_{tx} represents the shortest distance between node t and node x, and p and q are parameters governing the trade-off between exploration and exploitation of the graph. Specifically, p and q influence the speed of exploration during the random walk and the likelihood of leaving the neighborhood of the starting node u. w_{vx} denotes the edge weights between node v and x, with a value of 1 in the case of unweighted graphs. Finally, by generating a set of random walks for each node in the graph, both Node2vec and DeepWalk utilize stochastic gradient descent (SGD) to learn node embeddings for every node in the graph.

III. Random Walk with Node Embeddings

Let $G = (V, E)$ represent an undirected and weighted graph, where V denotes the finite vertex set and E is a collection of links. The currently visiting node is denoted as u. For a node u, $N(u)$ denotes its neighborhood, defined as the set $\{v|(u, v) \in E\}$. The target node is denoted as t. The embedding vector of node v, generated using a given node embedding technique E, is represented as e_v.

We assume that the mobile agent can access the local information on the currently visiting node: the list of neighbor nodes, the degrees (i.e., the number links connected) of all neighbor nodes, node embedding vectors of the currently visiting node as well as those of all neighbor nodes. In the context of the target node search, the mobile agent also knows the identifier of the target node and its embedding vector.

Our objective is to minimize the time required to search for the target node and/or to visit all nodes on an unknown graph at least once while utilizing the node embeddings of each node.

Conventional random walk algorithms such as SRW, Biased-RW, NBRW, and SARW typically operate based solely on local information when conducting searches for target nodes or exploring graphs. In the case of SRW and Biased-RW, mobile agents can navigate the graph without the need for memory. Conversely, SARW and NBRW employ memory to record visited nodes and prevent revisiting them. Despite

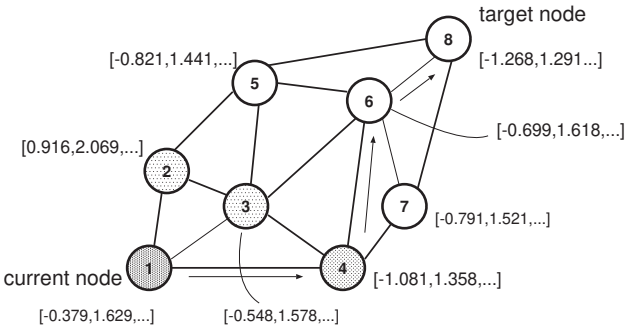


Fig. 1. An example operation of EmbedRW

these differences, all these algorithms primarily rely on local information, particularly information about neighboring nodes, as they conduct searches for target nodes or explore the graph. However, in large and complex unknown graphs, relying solely on local information may lead to decreased efficiency.

In this paper, we investigate how a random walk on an unknown graph can be improved with the node embedding vector e_v.

IV. Embedded Random Walk (EmbedRW)

The EmbedRW algorithm is an advanced technique for searching target nodes and exploring graphs, which leverages node embedding algorithm to expedite the process.

A. Example operation of EmbedRW

We first start explanation on EmbedRW with its example operation on a rather simple graph (Fig. 1).

Let us consider a scenario where the mobile agent aims to traverse efficiently from node 1 to node 7 (target node search) in a graph with eight nodes. For simplicity, we assume an equal transition probability among all nodes, akin to a Simple Random Walk (SRW), although other transition probabilities are possible. Each node is associated with an embedded vector, like $e_1 = [-0.379, 1.629, \ldots]$, generated by a node embedding algorithm like Node2vec and DeepWalk. These vectors encapsulate the information on the node in a graph, aiding in guiding the mobile agent toward the target node.

EmbedRW initially explores the neighboring nodes of node 1, such as nodes 2, 3, and 4 in this example. It calculates transition probabilities using Eq. (4) based on several factors such as node embedding vectors, transition probabilities, and proximity to the target node 7. In this case, node 4, identified with the highest probability, becomes the next node to visit. Transitioning to node 4, EmbedRW reassesses transition probabilities for neighboring nodes 1, 3, and 7. Node 6 emerges with the highest probability, leading the mobile agent to move to node 6. Throughout this process, EmbedRW utilizes intricate calculations and insights from node embeddings to navigate the graph's connections efficiently and accurately.

The concept behind EmbedRW is for the mobile agent currently visiting node u to attempt to move to a (good) neighboring node v, ensuring that i) for efficient traversal of the entire graph, nodes u and v appear to be sufficiently distant from each other, and ii) in the context of target node seach, to quickly approach the target node, node v appears to be closer to the target node than the current node u. In other words, EmbedRW leverages the node embedding vectors of the currently visited node and neighboring nodes, along with other local information accessible to the mobile agent (e.g., the degrees of neighboring nodes and the weights of outgoing links).

Algorithm 1 EmbedRW algorithm

Input: start node v_s, target node v_t (only in node seach), node embeddings e_v, transition weight $w(u \to v)$, weight parameters α, β, γ
$u \leftarrow v_s$ ▷ currently visiting node
for each step **do**
  $\mathcal{N} \leftarrow \{v | (u, v) \in E\}$ ▷ neighbor nodes
  for all $v \in \mathcal{N}$ **do**
    $w'_{u,v} \leftarrow w_{u,v} \left(\alpha \left\| e_{v_t} - e_v \right\|^{\beta} + (1-\alpha) \| e_u - e_v \|^{\gamma} \right)$
  $u \leftarrow$ randomly chosen node from $\mathcal{N}$ with weight $w'_{u,v}$
  terminate the algorithm if $u = v_t$ (in node seach)

B. Algorithm

In our study, we consider a scenario where a mobile agent exploring an unknown graph can access to the neighboring nodes of the currently visited node v and the target node v_t (only in target node search), and then retrieve the embedding vectors of each of these neighboring nodes and the target node.

Assume that the transition weight of random walk algorithms (e.g., SRW, NBRW, SARW, BiasedRW, VARW) from node u to node v is denoted as $w_(u, v)$ and that of the proposed EmbedWalk algorithm between nodes u and v is denoted as $w'(u, v)$. For searching the target node in an unknown graph, EmbedRW establishes the transition probability from node u to node v based on the distances between the embedding vectors of neighboring nodes and the target node as well as the distance between the currently visited node and its neighboring nodes.

The EmbedRW algorithm (Algorithm 1) begins by taking as input the starting node v_s , the target node v_t (this is only used in node search scenarios), the node embeddings e_v for each node, the transition weight $w_{u,v}$ for moving from one node to another, and weight parameters α, β, and γ.

1) The algorithm initializes by setting the currently visiting node u to the start node v_s.
2) It then enters a loop for each step of the algorithm, where it performs the following actions:
3) For each neighbor $v \in \mathcal{N}$, the algorithm calculates a new transition weight $w_{u,v'}$ using the following equation:

$$w'_{u,v} = f(w_{u,v}, e_u, e_v, e_{v_t}) \tag{3}$$

$$= w_{u,v} \left(\alpha \left\| e_{v_t} - e_v \right\|^{\beta} + (1-\alpha) \| e_u - e_v \|^{\gamma} \right) \tag{4}$$

This calculation adjusts the original transition weight $w_{u,v}$ by considering the distance between the embedding of the target node v_t and the embedding of the current neighbor v, weighted by parameters α and β. Additionally, it factors in the distance between the embeddings of the current node u and its neighbor v, weighted by γ. The idea is to blend these distances to guide the walk: closer nodes to the target and nodes that maintain the walk's diversity are preferred.

4) After calculating the new weights $w_{u,v'}$ for all neighbors, the algorithm selects the next node u to visit. This selection is random but weighted by the newly calculated transition probabilities $w_{u,v'}$, favoring nodes with higher weights.
5) The algorithm terminates if the currently visiting node u equals the target node v_t in a node search scenario.

EmbedRW supports both target node search and graph exploration. In the context of graph exploration, the target node v_t is not given, and the weight parameter is set to $\alpha = 0$.

This process iteratively guides the mobile agent through the graph, using both the structure of the graph (via the transition weights) and the semantic information encoded in the node embeddings (via the distances between embeddings) to efficiently search for the target node or explore the graph.

V. Experiments

A. Experiment design

This section presents an evaluation of the EmbedRW algorithm through experiments conducted on five distinct types of graphs to assess its performance across diverse scenarios. Evaluation metrics such as hitting time and cover time are utilized for this assessment. The hitting time H_{v_s,v_t} from node v_s to node v_t represents the expected number of steps before node v_t is visited, starting from node v_s. The cover time C_{v_s} denotes the expected number of steps needed to visit all nodes in the graph at least once starting from node v_s.

In this study, we adopt a network generation model to construct five different types of graphs, each comprising 100 nodes. These include Erdös-Réényi (ER), Barabási-Albert (BA), Voronoi, lattice, and tree graphs. These graph models are frequently used in simulations to replicate the diverse structures observed in real-world networks. We randomly select both the starting and target nodes for our simulations.

Characteristics of these five types of graphs are as follows.

- Erös-Rényi (ER) [36] model generates a random graph. For a given size N (the number of nodes) and the connection probability p, all node pairs are randomly connected with the probability p.
- Barabási-Albert (BA) graph [37] is a scale-free network model known for its power-law degree distribution. This distribution suggests that a few nodes have substantially more connections than others. For instance, scale-free networks effectively capture aspects of social networks, including the presence of hubs and the inclination of connected nodes to share connections [38], .
- Voronoi graph partitions a space into regions determined by the proximity to specific seed points. Each point within the space is assigned to the region associated with its nearest seed point. It finds applications in spatial analysis and pattern recognition [39].
- Lattice graphs represent regular, grid-like structures where nodes are arranged in rows and columns.
- Tree graphs represent a hierarchical structure without any cycles. It begins with a single root node and branches out into subtrees, with each node having exactly one parent except for the root.

As baseline random walk algorithms, we used SRW, NBRW, SARW, Biased RW, and VARW. Each simulation is conducted 3,000 times, generating different network structures for each trial based on the specified graph type. The control parameters (α, β, γ) of EmbedRW were set to (1, -0.5, 0.5) for target node search and (0, -0.5, 0.5) for graph exploration. We used the Node2vec algorithm for generating the node embedding vectors. The dimension of the node embedding vectors was set to 128.

B. Results

We examine the influence of node embeddings on the average hitting time and the cover time of random walks, including SRW, NBRW, SARW, BiasedRW, and VARW. Figures 2 and 3 depict the performance of these random walks with and without our proposed EmbedRW framework across different graph types.

In the figures, for instance, the label SRW represents pure SRW algorithm while the label EmbedSRW represents the combination of SRW algorithm with our EmbedRW algorithm. In SRW, the transition weight between any two nodes is always 1. Conversely, in EmbedSRW, this weight is adjusted using Eq. (4). Similarly, the label NBRW stands for pure NBRW algorithm, and the label EmbedNBRW represents the combination of NBRW and EmbedRW algorithms.

Figure 2 indicates that employing the proposed EmbedRW can significantly enhance the search time of conventional random walk algorithms across various graphs, except for the tree graph. For instance, our EmbedRW algorithm dramatically reduce the average hitting time of SRW. The reduction in the average hitting time is not significant in the tree graph, but in all other graphs, the reduction ratio reaches 40–90%.

The EmbedRW algorithm proves to be effective not only against the simplest random walk algorithm, SRW, but also against more sophisticated random walk algorithms. It has been observed that the introduction of EmbedRW leads to a reduction in the average hitting time across all graph types, with the exception of tree graphs. However, it is also noted that in the case of the VARW algorithm, integrating the EmbedRW algorithm can, in some instances, result in an increase in the average hitting time. Specifically, in tree graphs, the effectiveness of EmbedRW is either limited or, conversely, can have a negative impact on the search for the target node. From this, it can be inferred that EmbedRW is particularly effective

in relatively dense graphs (that is, graphs where multiple paths to the target node exist).

On the contrary, Fig. 3 indicates that the effectiveness of EmbedRW on graph exploration is marginal. In almost all cases, integrating the EmbedRW algorithm with existing random walk algorithms does not significantly reduce the cover time. On the contrary, in some graphs (such as BA and tree graphs), the introduction of EmbedRW has resulted in a substantial increase in cover time. This suggests that EmbedRW is particularly effective when using node embedding vectors for the purpose of searching for the target node.

We also investigated the effect of node embedding algorithms on search target node and exploration in an unknown graph. We utilized two major node embedding algorithms, Node2vec and DeepWalk. Figures 4 and 5 show the average hitting time and cover time of EmbedRW when using both Node2vec and DeepWalk node embedding algorithms. For example, in the figure, the label EmbedSRW refers to EmbedSRW utilizing the Node2vec algorithms, while the label EmbedSRW-DW refers to EmbedSRW employing the DeepWalk algorithms. Our results suggest that the effectiveness of our EmbedRW is not significantly affected by the choice of the node embedding algorithm. EmbedRW with Node2vec, which employs both width-first and depth-first sampling, exhibits performance similar to that with DeepWalk, which employs a simple random walk sampling. Overall, the choice of algorithm used to compute node embedding vectors minimally affects the performance of EmbedRW.

Acknowledgements

This work was supported by JSPS KAKENHI Grant Number 24K02936.

VI. Conclusion

In this paper, we have proposed EmbedRW, a novel random walk algorithm that utilizes node embedding techniques to improve the efficiency of target node search and exploration in unknown graphs. Experimental evaluations have been conducted to compare the performance of EmbedRW with traditional random walk algorithms. The results have demonstrated significant improvements with EmbedRW, achieving reductions of approximately 40% to 90% in node exploration across various graphs, except for tree graphs. Moving forward, we plan to explore the effectiveness of EmbedRW in dynamic graph scenarios and investigate how the control parameters of node embedding algorithms impact the performance of EmbedRW.

References

[1] S. H. Fard, "Machine Learning on Dynamic Graphs: A Survey on Applications," in *Proceedings of the IEEE Ninth Multimedia Big Data (BigMM 2023)*, pp. 32–39, Dec. 2023.

[2] F. Xia, K. Sun, S. Yu, A. Aziz, L. Wan, *et al.*, "Graph Learning: A Survey," *IEEE Transactions on Artificial Intelligence*, vol. 2, pp. 109–127, Apr. 2021.

[3] F. Xia, J. Liu, H. Nie, Y. Fu, *et al.*, "Random Walks: A Review of Algorithms and Applications," *IEEE Transactions on Emerging Topics in Computational Intelligence*, vol. 4, pp. 95–107, Apr. 2020.

[4] F. Niu, J. Yue, J. Shen, X. Liao, *et al.*, "FlashWalker: An In-Storage Accelerator for Graph Random Walks," in *Proceedings of the IEEE International Parallel and Distributed Processing Symposium (IPDPS 2022)*, pp. 1063–1073, May 2022.

[5] L. Lszl, L. Lov, and O. P. Erdos, "Random Walks on Graphs: A Survey," *Combinatorics, Paul Erdos is Eighty*, pp. 1–46, Jan. 1996.

[6] A. das Sarma, D. Nanongkai, and G. C. Pandurangan, "Fast distributed random walks," in *Proceedings of the ACM symposium on Principles of distributed computing (PODC 2009)*, pp. 161–170, Aug. 2009.

[7] J. R. Barr, P. Shaw, F. N. Abu-Khzam, T. Thatcher, and T. D. Hocking, "Graph Embedding: A Methodological Survey," in *Proceedings of the International Conference on Transdisciplinary AI (TransAI 2022)*, pp. 142–148, Sept. 2022.

[8] M. Grohe, "Word2vec, Node2vec, Graph2vec, X2vec: Towards a Theory of Vector Embeddings of Structured Data," in *Proceedings of the ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems (SIGMOD/PODS 2020)*, pp. 1–6, June 2020.

[9] A. E. Barinov and A. A. Zakharov, "Clustering Using a Random Walk on Graph for Head Pose Estimation," in *Proceedings of the International Conference on Mechanical Engineering, Automation and Control Systems (MEACS 2015)*, pp. 1–5, Dec. 2015.

[10] S. Basirian and A. Jung, "Random walk sampling for big data over networks," in *Proceedings of the International Conference on Sampling Theory and Applicatqions (SampTA 2017)*, pp. 427–431, July 2017.

[11] D. R. Karger and M. Ruhl, "Simple Efficient Load Balancing Algorithms for Peer-to-Peer Systems," in *Proceedings of the Annual ACM symposium on Parallelism in algorithms and architectures (SPAA 2004)*, pp. 36–43, June 2004.

[12] A. N. Mian, M. Fatima, R. Khan, and R. Prakash, "An Empirical Evaluation of Lightweight Random Walk Based Routing Protocol in Duty Cycle Aware Wireless Sensor Networks," *The Scientific World Journal*, pp. 1–9, Feb. 2014.

[13] Q. Xiao, Y. Zhang, and Q. Yang, "Selective Random Walk for Transfer Learning in Heterogeneous Label Spaces," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pp. 1–12, Jan. 2024.

[14] S. Jack and R. Contents, "A SURVEY OF RANDOM WALKS ON NETWORKS." https://mathweb.ucsd.edu/~s5robert/documents/random-walks.pdf, Jan. 2020.

[15] J. Gong, L. Cai, and Y. Shen, "Evaluating accuracy and performance of GPU-accelerated random walk computation on heterogeneous networks," in *Proceedings of the IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD 2016)*, pp. 541–545, June 2016.

[16] M. Bonaventura, V. Nicosia, and V. Latora, "Characteristic times of biased random walks on complex networks," *Physical review. E, Statistical, nonlinear, and soft matter physics*, vol. 89, July 2013.

[17] A. D. Sarma, A. R. Molla, and G. Pandurangan, "Fast Distributed Computation in Dynamic Networks via Random Walks," in *Proceedings of the International Symposium on Distributed Computing (DISC 2012)*, pp. 136–150, Oct. 2012.

[18] C. Zhang, C. Deng, L. Fu, X. Wang, *et al.*, "RWE: A Random Walk Based Graph Entropy for the Structural Complexity of Directed Networks," *IEEE Transactions on Network Science and Engineering*, vol. 11, no. 2, pp. 2264–2278, 2024.

[19] C. Cooper, T. Radzik, and Y. Siantos, "Estimating network parameters using random walks," in *Proceedings of the Fourth International Conference on Computational Aspects of Social Networks (CASoN 2012)*, pp. 33–40, Nov. 2012.

[20] Y. Gao, T. Fan, and S. Cai, "Navigation in Complex Networks Using Random Walk Theory and Principal Component Analysis," in *Proceedings of the 16th International Computer Conference on Wavelet Active Media Technology and Information Processing (ICWAMTIP 2019)*, pp. 349–354, Dec. 2019.

[21] B. Cai, H. Wang, H. Zheng, and H. Wang, "An improved random walk based clustering algorithm for community detection in complex networks," in *Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics (IEEE SMC 2011)*, pp. 2162–2167, Oct. 2011.

[22] R. Sun, L. Zhang, and Z. Chen, "A Sample-Based Approximate Ranking Method for Large Graphs," in *Proceedings of the International Conference on Advanced Cloud and Big Data (CBD 2018)*, pp. 112–117, Aug. 2018.

[23] S. Brin and L. Page, "The anatomy of a large-scale hypertextual Web search engine," in *Proceedings of the International Conference on World Wide Web (WWW 2017)*, pp. 107–117, Apr. 1998.

[24] H. Rahmani and G. Weiss, "Node classification in graph data using augmented random walk," in *Proceedings of the International Conference on Web Research (ICWR 2015)*, Apr. 2015.

[25] M. Bonaventura, V. Nicosia, and V. Latora, "Characteristic times of biased random walks on complex networks," *Phys. Rev. E*, vol. 89, Jan. 2014.

[26] D. Fasino, A. Tonetto, and F. Tudisco, "Hitting times for non-backtracking random walks," *ArXiv*, vol. abs/2105.14438, 2021.

[27] N. Madras and G. Slade, *The self-avoiding walk*. Springer Science & Business Media, 2013.

[28] K. Kitaura, R. Matsuo, and H. Ohsaki, "Random Walk on a Graph with Vicinity Avoidance," in *proceedings of the International Conference on Information Networking (ICOIN 2022)*, pp. 232–237, Jan. 2022.

[29] 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.

[30] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient Estimation of Word Representations in Vector Space," in *Proceedings of the International Conference on Learning Representations (ICLR 2013)*, pp. 1–12, Jan. 2013.

[31] B. Perozzi, R. Al-Rfou, and S. Skiena, "DeepWalk: Online Learning of Social Representations," in *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD 2014)*, pp. 701–710, Aug. 2014.

[32] A. Grover and J. Leskovec, "Node2vec: Scalable Feature Learning for Networks," in *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (SIGKDD 2016*, pp. 855–864, Aug. 2016.

[33] K. M. Hamakarim, A. F. Mohammed, R. A. F. Mohammed, and B. N. Hamatahir, "Using DeepWalk to Link Prediction in Subgraph," in *Proceedings of the International Workshop on Semantic and Social Media Adaptation Personalization (SMAP 2023)*, pp. 1–6, Sept. 2023.

[34] J. Ha and S. Park, "NCMD: Node2vec-Based Neural Collaborative Filtering for Predicting MiRNA-Disease Association," *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, vol. 20, pp. 1257–1268, Mar–Apr. 2023.

[35] N. Yadav and D. Gopinathan, "Node Embedding Approach For Topic Search From Academic Papers," in *Proceedings of the International Conference on Signal Processing and Communication (ICSC 2022)*, pp. 445–450, Dec. 2022.

[36] P. L. Erdos and A. Rényi, "On random graphs. I.," *Publicationes Mathematicae Debrecen*, vol. 6, pp. 290–297, Nov. 1959.

[37] A. L. Barabasi and R. Albert, "Emergence of Scaling in Random Networks," *Science*, vol. 286, pp. 509–512, Oct. 1999.

[38] C. Sirocchi and A. Bogliolo, "Topological network features determine convergence rate of distributed average algorithms," *Scientific Reports*, vol. 12, p. 21831, Dec. 2022.

[39] H. Choset, K. M. Lynch, S. Hutchinson, G. A. Kantor, *et al.*, *Principles of Robot Motion: Theory, Algorithms, and Implementations*. Bradford Books, May 2005.

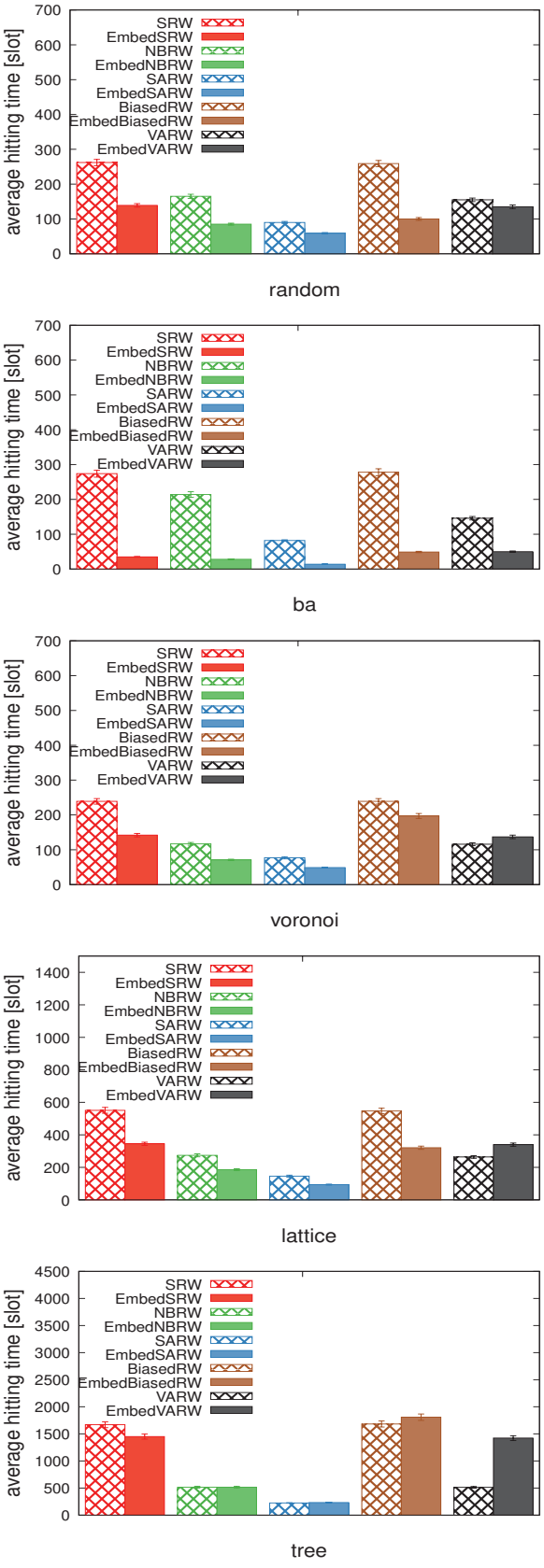
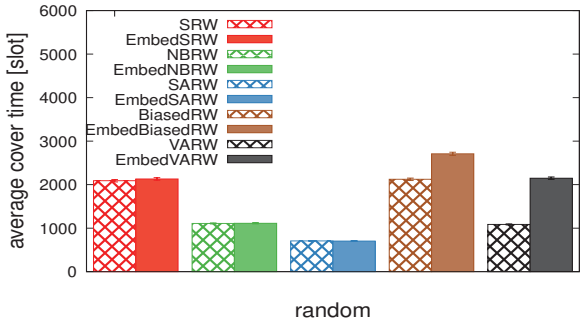
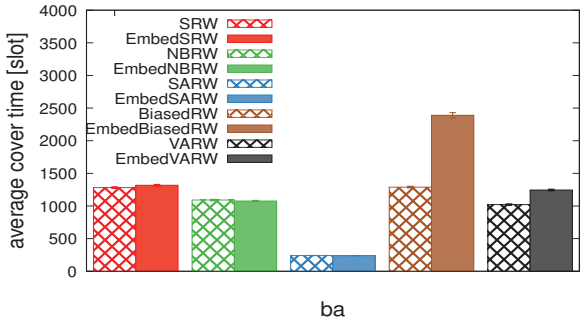
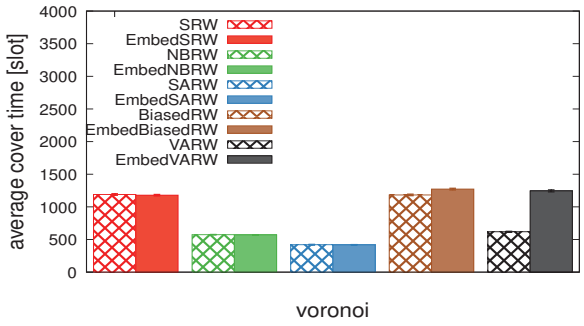
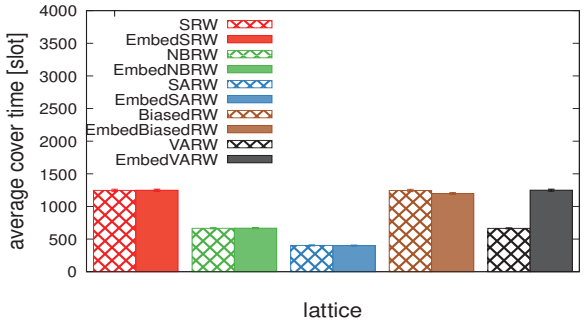


Fig. 2. Hitting time









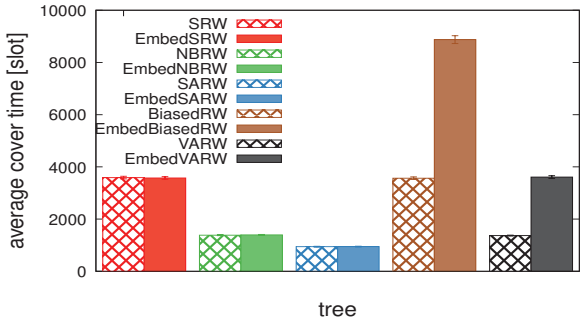
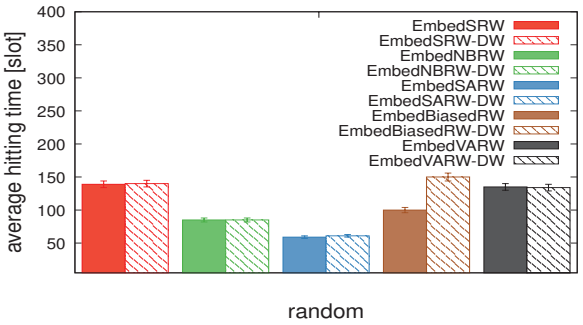
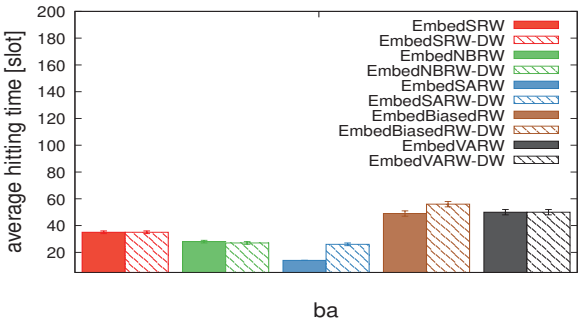
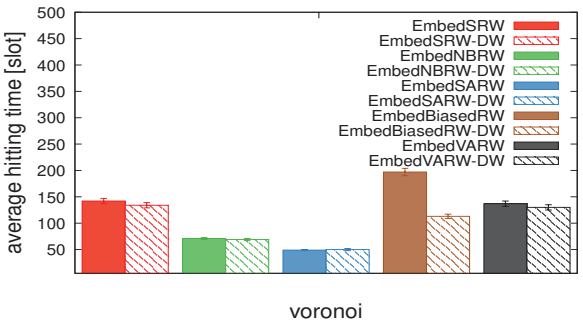
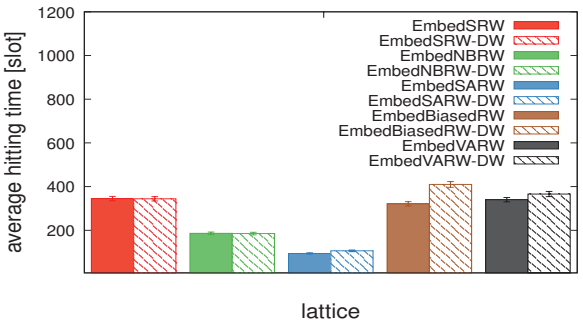


Fig. 3. Cover time









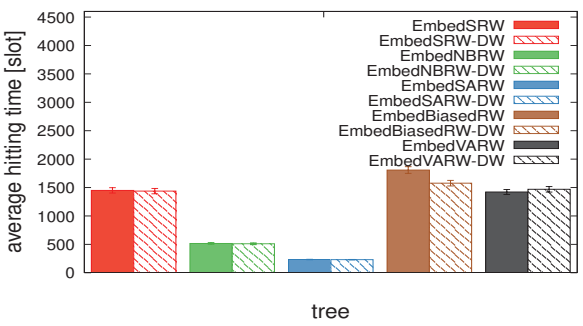


Fig. 4. Effect of node embedding algorithm on hitting time

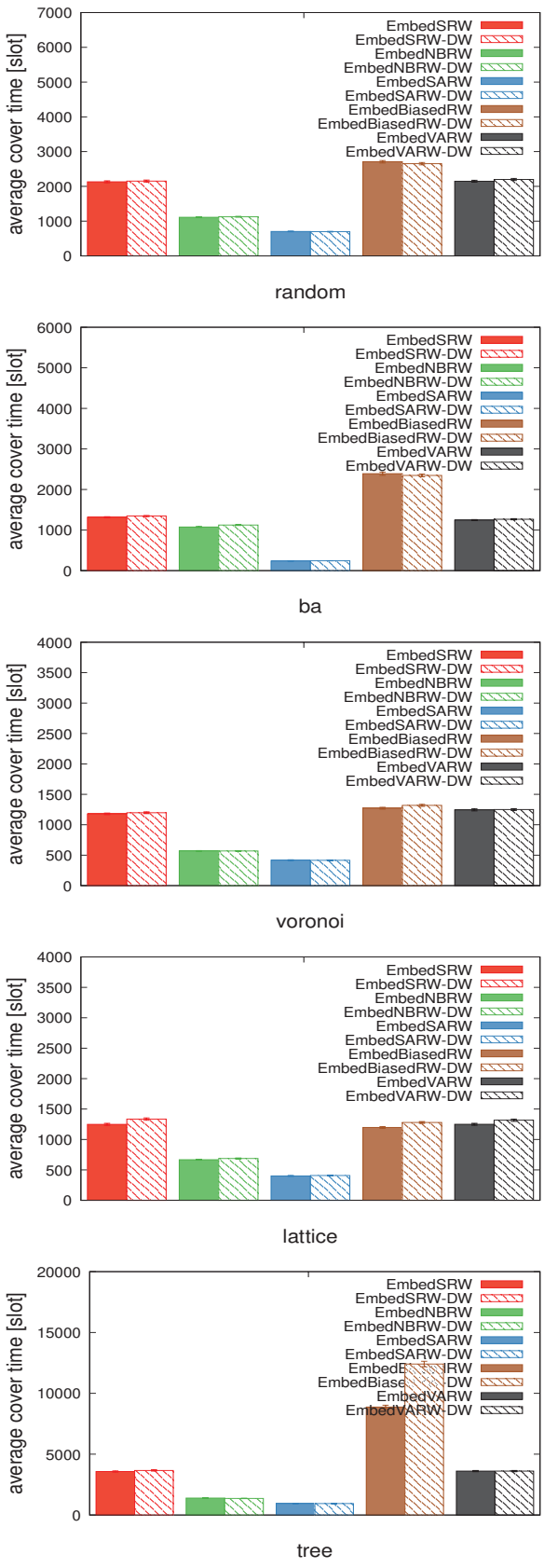


Fig. 5. Effect of node embedding algorithm on cover time