

Received April 23, 2021, accepted May 3, 2021, date of publication May 6, 2021, date of current version May 17, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3077888

# AORM: Fast Incremental Arbitrary-Order Reachability Matrix Computation for Massive Graphs

SUNG-SOO KIM<sup>1,2</sup>, YOUNG-KUK KIM<sup>2</sup>, AND YOUNG-MIN KANG<sup>3</sup>, (Member, IEEE)

<sup>1</sup>Artificial Intelligence Research Laboratory, Electronics and Telecommunications Research Institute, Daejeon 34129, South Korea

<sup>2</sup>Department of Computer Science and Engineering, Chungnam National University, Daejeon 34314, South Korea

<sup>3</sup>Department of Game Engineering, Tongmyong University, Busan 48520, South Korea

Corresponding authors: Young-Min Kang (ymkang@tu.ac.kr) and Young-Kuk Kim (ykim@cnu.ac.kr)

This work was supported by the Institute of Information and Communication Technology Planning and Evaluation (IITP) grant funded by the Korea Government (MSIT), Development of Cloud-Edge based City-Traffic Brain Technology, under Grant 2020-0-00073.

**ABSTRACT** Processing a reachability query in large-scale networks using existing methods remains one of the most challenging problems in graph mining. In this paper, we propose a novel incremental algorithmic framework for arbitrary-order reachability computation in massive graphs. The proposed method is intuitive and significantly outperforms the currently known methods in terms of computation time. We focus on the arbitrary-order reachability matrix framework called AORM, which can handle directed and disconnected networks such as citation networks. The AORM can handle diverse types of real-world datasets. We conduct extensive experimental studies with twenty synthetic networks generated from five random graph generation models and twenty massive real-world networks. The experimental results show the advantages of the method in terms of both computational efficiency and approximation controllability. In particular, the proposed method outperforms up to 10 times compared to NetworkX for incremental all-pairs shortest paths computation. Moreover, the computational results of the method rapidly converge to the ground truths. Thus, we can get the correct solution in the early stage of the incremental approximation. We can employ the method as a versatile feature extraction framework for network embedding. Overall, the experimental results present a remarkable improvement in speed-up for reachability computation.


**INDEX TERMS** Reachability query, approximate all-pairs shortest paths, graph girth, graph embedding, higher-order structural proximity.

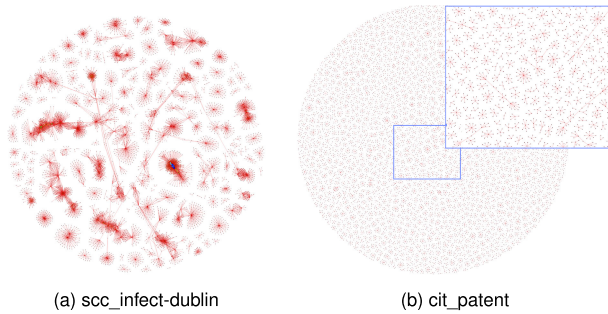
## I. INTRODUCTION

The World Health Organization (WHO) declared COVID-19 a *pandemic* caused by the new SARS-CoV-2 virus in March 2020. Accordingly, COVID-19 has formed a new scientific community to address the pandemic, unlike other recent crises. Many data analysts and researchers in modern graph analytics utilize new pandemic analysis methods and recent artificial intelligence approaches to tackle this problem. In particular, *massive graph* analytics is essential to predict virus diffusion and discover super-spreaders on a global scale [1]. In terms of graph theory, the data analysts employ *directed*, *disconnected* graphs to investigate these viral epidemics. Most researchers often employ *adjacency matrices* as representations of these graphs to perform fundamental

graph analytics operations, such as reachability queries, all-pairs shortest paths (APSP), and so on. Specifically, in most practical applications, adjacency matrices of massive graphs are usually *sparse* [2]–[4]. Moreover, these applications appropriate *unweighted edges* rather than weighted edges to describe relationships between vertices in a graph.

*Motivations:* Prior researchers in graph mining found that existing networks manifest surprisingly *small diameters* even though graphs are massive. Specifically, the average edge degree of hyperlink networks for the web is nineteen [3]. Furthermore, most social networks also present the *six-degrees* or even *four-degrees of separation* characteristics [4], [5]. Also, most citation networks are *disconnected* graphs with *directionality* due to their low reciprocity [6]. Figure 1 shows examples of real networks in the form of disconnected graphs. A *temporal reachability network* is a network whose edges are active only at certain points in time. This network is

The associate editor coordinating the review of this manuscript and approving it for publication was Walter Didimo .



**FIGURE 1.** Two examples of real networks in the form of disconnected graphs. (a) The *scc\_infect-dublin* is the contact network whose edges are face-to-face contacts. A contact network is one of the temporal reachability networks. (b) The *cit\_patent* is the US patent citation network (1975-1999). This network is an unweighted and directed graph.

formed by putting an edge in the temporal reachability graph if there exists a strong *temporal path* between two vertices. Hence, a temporal path describes a sequence of contacts (i.e., face-to-face contact in Figure 1-(a)) that follows time. Therefore an edge in this kind of graphs represents the fact that an entity might have transmitted a piece of information (or disease, etc) to another (and vice-versa). *Citation networks* are disconnected graphs that represent collections of academic publications as shown in Figure 1-(b). In these networks, vertices denote authors or papers. Edges represent relationships of author-coauthor or paper-paper citation. Furthermore, *directed* graphs are indispensable in applications utilizing edge directionality of graphs: road networks, infectious diseases networks, and citation networks.

**Motivating Example (Traffic Congestion Prediction)** All-pairs shortest paths (APSP) and reachability query computations are important graph operations in modern graph analytics. In urban road networks, arbitrary congested road segments connect to nearby other congested ones. And as time progresses, these congested roads have the characteristic of *spreading congestion* to nearby road segments. Thus, predicting how congestion will propagate over the road network in the near future is one of the crucial things to alleviate congestion in advance [7]. From a graph theory point of view, these congested road links are *subdigraphs* in a graph  $\mathcal{G}$  for the road network. To find and trace these subdigraphs, we utilize reachability queries and APSP to find neighbor roads from the current congested road segment.

The majority of prior studies have focused on unweighted, undirected graphs [8], [9]. Thus, these methods cannot capture the directions of edges: required in several applications, such as tracing virus diffusion, analysis of traffic congestion propagation [7], [10], [11]. To address these problems, we exploit a directed graph to represent not only symmetrical but also *asymmetrical* relationships between nodes in a network. A few researchers proposed some approaches for graph-related operations for directed graphs, such as APSP computation, network embedding, and so on [12]–[18]. However, most of the existing methods concentrated on undirected graphs with limited attention to the directed, disconnected networks [19], [20]. The APSP

algorithm working on weighted directed graphs via *circuit complexity* efficiently reduced a matrix product over the min-plus algebra [15], [21]. Unfortunately, this algorithm can work with a relatively small number of rectangular matrix products. Consequently, this method is hard to perform the APSP computation for the massive graphs.

Inspired by the characteristics of small diameter, low reciprocity (directionality), and sparsity of common real-world networks, we present a novel arbitrary-order reachability matrix computation framework for directed, disconnected graphs. To the best of our knowledge, no previous research has studied the arbitrary-order reachability computation problem via *incremental* fashion. Moreover, the proposed algorithm works on directed graphs and disconnected networks; these graphs frequently arise in citation networks.

**Our Goal:** We aim to develop a new algorithmic framework for reachability query processing. The proposed framework can rapidly and incrementally compute arbitrary-order reachability in directed graphs (i.e., road networks) and disconnected graphs (i.e., citation networks and collaboration networks). We call this new framework, AORM. Besides, we focus on designing and implementing the APSP computation algorithms based on AORM to show the validity and the efficiency of the method.

We can efficiently employ our proposed framework to diverse graph representation learning approaches for unweighted and directed graphs. Those approaches include embeddings or encodings of the original network nodes onto a low-dimensional vector space while maintaining the *proximity* and *structural equivalence* properties. The aim of graph embedding for given graph  $\mathcal{G}$  is to learn a mapping,  $\mathcal{E}(\mathcal{G}) : V \mapsto Z$ , that maps each vertices  $v_i$  of  $\mathcal{G}$  to a vector  $z_i$  in a space with lower-dimension known as embedding space [22]–[24]. The *proximity* between vertices in graphs is one of the essential properties to preserve for network representation learning. Existing graph embedding strategies, such as combinatorial approaches and stochastic graph traversal methods, depended on adjacency matrix multiplications for measuring the proximity between nodes. Hence, these methods are hard to parallelize the computations for capturing the proximity in graphs. Furthermore, these techniques are arduous to utilize for machine learning-based methods; since the powers of any adjacency matrix rapidly become dense so that the matrix multiplications require a massive amount of computations for massive graphs.

**Challenges:** The first challenge is processing *arbitrary-order reachability queries* for massive graphs. The *computation of the powers* of the adjacency matrix is one of the core operations for node embeddings. Network embedding with *higher-order proximity* considers the ‘roles’ of nodes, such as a hub, bridge, and near-clique in a graph. However, we cannot capture these *roles* by stochastic neighbor search methods such as random walk sequences [25]. So, we should define higher-order proximity to capture the higher-order structural natures of vertices in a graph. For example, we can apprehend the probability of the *context* node  $v_j$  being reached by

node  $v_i$  by examining the *second-order* proximity [26]. Here, the second-order proximity of two nodes means the similarity of the neighbors of two nodes. We can define any  $k$ -order proximities as the similarity of reachable nodes with *path length*  $k$ . Also, we obtain these latent properties by computing the powers of adjacency matrix [27]. However, for massive graphs, this operation requires a lot of computations due to the many matrix multiplications. Thus, the *improvement* in computing the powers of adjacency matrix is one of the important challenges.

The second challenge is improving an *all-pairs shortest paths computation* (APSP) algorithm supporting directed graphs. The APSP in a graph is a reasonable measure for pairwise distance for node embedding. However, the complete computation of APSP is notoriously expensive. Traditional APSP solution, FLOYD–WARSHALL algorithm, requires  $\mathcal{O}(n^3)$  time for  $n$ -node weighted graphs with no negative cycle. This time complexity makes it impossible to employ this algorithm as a generic method for massive graphs.

Last challenge is to develop a framework of reachability and APSP computation working on even *disconnected graphs*. As previously mentioned, most real networks include directed and disconnected graphs. Nevertheless, existing studies have focused on unweighted and connected graphs. These algorithms cannot support practical applications related to collaboration networks, citation networks, and temporal networks.

*Our Approach:* First, to compute the powers of the adjacency matrix without matrix multiplications, we propose a new incremental reachability computation framework, called AORM. Although existing methods are effective to make reachability queries and all-pairs shortest paths computation, most of them assume that the input graphs are unweighted, undirected [8], [9]. Unfortunately, many graph analytics applications involve directed, disconnected networks, such as road networks, collaboration networks, and citation networks. The key idea is that we exploit the *neighborhood* information in the first-order proximity of a graph and *hierarchical reachability* matrices to compute an arbitrary-order reachability. In particular, the characteristic of incremental matrix computation provides benefits for massive graphs. These benefits include fundamental operations in graph embedding, such as capturing arbitrary-order proximity, computing all-pairs shortest paths, and processing reachability queries in a graph. Besides, we propose a novel feature representation of node embedding by applying high-order proximity via APSP and shortest cycle length computations based on AORM. Also, our APSP method based on AORM approximates the correct solution with small errors.

*Difference from Previous Work:* Existing reachability computation methods typically depend on matrix multiplications for the powers of the adjacency matrix. In contrast, AORM exploits neighborhood information and boolean reachability matrix for arbitrary-order reachability computation instead of expensive matrix multiplications. AORM's contributions include incremental arbitrary-order reachability computation

and fast APSP approximations. In terms of parallel computing, prior techniques have focused on the parallelization for *matrix multiplications* under multi-core or distributed computing environments. In contrast, we present a novel *incremental* reachability matrix computation approach without any matrix multiplications. Also, the results of AORM are asymptotically correct in  $\mathcal{O}(\delta_v md)$  time;  $n$  denotes the number of vertices,  $\delta_v$  refers to  $v$ -dimensional vectorized operation time;  $\delta_v \ll n$ , and  $d$  refers to a diameter of a graph. To summarize, the main advantages of AORM over existing techniques are the *incremental* property and *fast convergence* property in the computational aspect for the approximated APSP and reachability queries.

*Main Contributions:* Our major contributions can be summarized as follows:

- *Efficient framework:* We formulate AORM framework as an *arbitrary-order reachability matrix* computation with the objective of approximating multi-hop similarity among vertices in a graph. Also, we propose several techniques based on AORM to efficiently solve graph connectivity-related problems, including the efficient approximation of the reachability matrices, fast APSP, and shortest cycle length computation.
- *Directed and disconnected graphs:* We further consider edge direction to support directed graphs. Besides, AORM works on disconnected graphs that are common in real-world networks (i.e., citation networks).
- *Incremental computation:* One of the key features in AORM framework is incremental computation for efficiently approximated reachability and APSP solution. This property enables data analysts to perform fast graph analytics (i.e., node classification and clustering) on *massive* graphs.
- *Hierarchical representation:* AORM framework can store the weighted accumulation of the incremental  $k$ -order reachability hierarchically. We can efficiently appropriate this AORM's feature for flexible  $k$ -order reachability queries and diverse graph analytics, such as computation of minimum cycle and feature extraction for each node.

*Reproducibility:* Our source codes and data<sup>1</sup> are publicly available on <https://github.com/sungsoo/AORM>.

*Organization:* The rest of the paper is organized as follows. We first detail the preliminaries and notations used in the paper in Section II. In Section III, we introduce the algorithmic framework of the proposed reachability computation approach. Then we present the detailed algorithms of incremental and constrained  $k$ -order reachability computation for massive graphs in Section IV. Section V describes the experimental results. Related works are reviewed in Section VI. Finally, Section VII concludes the paper.

<sup>1</sup>We utilize the datasets from SNAP [28] and the network repository [29] for our experiments.

II. PRELIMINARIES

This section describes some background about graphs, notations, and definitions. We discuss a few reachability-related matrices and sets in Section III.

First, we describe the notations related to the superscript of several matrices in this paper. The superscript used as  $x^k$  refers to the  $k$ -th power of  $x$ . Whereas the superscript with parenthesis such as  $x^{(k)}$  denotes the instance  $x$  at the  $k$ -th iteration. Second, we use the notations for representing an element in a matrix as the followings. We specify with two subscripts  $i, j$  to describe the element at  $i$ -th row and  $j$ -th column in a matrix. The subscript  $:$  means all rows or all columns in accordance with the location of the subscript.

Let  $V$  and  $E$  be a finite set of  $n$  vertices  $\{v_1, \dots, v_n\}$  and a finite set of  $m$  edges  $\{e_{i,j}|v_i, v_j \in V\}$  respectively. Let  $\mathcal{G} = (V, E)$  be an unweighted graph (directed or undirected) with  $|V| = n$  and  $|E| = m$ . We call vertices and edges as nodes and links respectively and interchangeably. Also, we define a *path* in a graph  $\mathcal{G}$  to be a sequence  $P$  of nodes  $\langle v_1, v_2, \dots, v_{k-1}, v_k \rangle$  with the property that each consecutive pair  $v_i, v_{i+1}$  is joined by an edge in  $\mathcal{G}$  [30].  $P$  is often called a *path* from  $v_1$  to  $v_k$ , or a  $v_1 \rightsquigarrow v_k$  path. Then, the *cycle* with  $n \geq 3$  vertices, denoted by  $C$ , is a subgraph of the graph  $\mathcal{G}$ , isomorphic to  $v_s \rightsquigarrow v_s$  path with an identical start and end node  $v_s$ .

*Directed Graphs and Disconnected Graphs:* Directed graphs play a crucial role in various user applications, such as navigation systems. Unfortunately, the majority of prior studies have focused on *undirected* graphs [26]. A graph is said to be disconnected if it is not connected, i.e., if there exist two vertices such that no path has those vertices as endpoints. This graph refers to a *disconnected* graph that frequently appears in real-world such as infectious disease networks. For these reasons, most of the data analysts exploit not only directed networks but disconnected graphs.

A. NOTATIONS

Table 1 summaries symbols and notations used throughout this paper.

*Definition 1 (First-Order Proximity):* The first-order proximity is the local pairwise similarity between vertices linked by edges. Each pair of vertices linked by an edge  $(u, v)$  means the first-order proximity between  $u$  and  $v$  [26]. First-order proximity is 0 if and only if no edge exists between  $u$  and  $v$ .

Two vertices in the graph are similar if they are connected with the edge. It characterizes the local network structure. If there exists an edge  $e_{i,j}$  between node  $v_i$  and  $v_j$ , we assign non-zero value of element  $A_{i,j}$  according to the edge weight in adjacency matrix  $\mathbf{A}$  to represent the first-order proximity. In addition, we set zeros to the diagonal elements of  $\mathbf{A}$ .

*Definition 2 (Arbitrary-Order Proximity):* Given adjacency matrix  $\mathbf{A}$  of an undirected or directed graph, we define an arbitrary-order proximity as a polynomial function  $\mathcal{P}(\cdot)$  of  $\mathbf{A}$  similar to the proximity defined in [9]:

$$\mathcal{P}(\mathbf{A}) = w_1\mathbf{A} + w_2\mathbf{A}^2 + \dots + w_k\mathbf{A}^k, \quad (1)$$

where  $k$  is the order and  $w_1, w_2, \dots, w_k$  are the weights.

TABLE 1. Summary of symbols and notations.

Notations	Description
$V$	finite set of $n$ vertices $\{v_1, \dots, v_n\}$
$E$	finite set of $m$ edges $\{e_{i,j} v_i, v_j \in V\}$
$e_{i,j}$	directed edge from $v_i$ to $v_j$
$n$	number of nodes in a graph, or $ V $
$m$	number of edges in a graph, or $ E $
$\mathcal{G} = (V, E)$	graph with $n$ vertices in $V$ and $m$ edges in $E$
$\mathbf{A}$	adjacency matrix of graph $\mathcal{G}$
$\mathbf{A}^k$	$k$ -th power of adjacency matrix of matrix $\mathbf{A}$
$A_{i,j}^k$	element at $i$ -th row and $j$ -th column in matrix $\mathbf{A}^k$
$\mathbf{A}_{i,:}^k$	$i$ -th row vector in matrix $\mathbf{A}^k$
$\mathbf{A}_{:,i}^k$	$i$ -th column vector in matrix $\mathbf{A}^k$
$\mathcal{H}(\cdot)$	heaviside function
$d$	diameter of the graph
$\mathbf{R}^{(d)}$	reachability matrix
$\mathbf{R}^{(k)}$	$k$ -order reachability matrix
$\mathbf{R}^{(k)*}$	$k$ -order optimal reachability matrix
$\mathcal{N}_{\mathbf{A}}$	neighborhood information of matrix $\mathbf{A}$
$N_i$	set of vertices directly linked with vertex $v_i$ in $\mathcal{N}_{\mathbf{A}}$
$\mathbf{I}$	identity matrix
$F_i^{(k)}$	footprint set set of vertices found by vertex $v_i$ within $k$ hops
$\mathbf{F}^{(k)}$	footprint matrix boolean matrix of which entry $F_{i,j}^{(k)}$ is 1 iff $v_j \in F_i^{(k)}$
$\mathbf{D}$	all-pairs path distance matrix of matrix $\mathbf{A}$

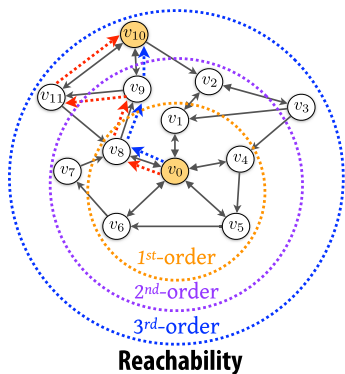
Our algorithm aims to provide arbitrary-order reachability computation for undirected, directed, and disconnected graphs. Thus, the adjacency matrix  $\mathbf{A}$  of  $\mathcal{G}$  is *asymmetric*. To estimate arbitrary-order proximity, we need to compute the  $k$ -th power of the adjacency matrix. Let  $\mathbf{A}^k$  be the  $k$ -th power of the adjacency matrix. Then the element at the  $i$ -th row and  $j$ -th column of the  $k$ -th power matrix is  $A_{i,j}^k$ . Let  $\mathbf{A}_{i,*}^k$  and  $\mathbf{A}_{*,i}^k$  denote the  $i$ -th row vector and the  $j$ -th column vector respectively. The element of  $A_{i,j}^k$  means the number of paths from  $v_i$  to  $v_j$  with exactly  $k$  hops. The Heaviside function  $\mathcal{H}(\cdot)$  is a non-continuous function whose value is zero for a negative input and one for a positive input [31]. The diameter of the graph  $d$  is the *longest* distance between any pair of vertices. The Heaviside function  $\mathcal{H}(\cdot)$  and the diameter are beneficial for efficient computation of reachability-related matrices.

*Reachability* refers to the ability to move from one vertex to another within a graph. In directed graphs, a vertex  $u$  can reach a vertex  $v$  if there exists a  $u \rightsquigarrow v$  path. The boolean reachability matrix  $\mathbf{R}^{(d)}$  is a matrix including all the reachability information of a graph whose adjacency matrix is  $\mathbf{A}$ . A nonzero element  $R_{i,j}^{(d)}$  indicates that there exists at least one path  $v_i \rightsquigarrow v_j$ . In other words,  $\mathbf{R}^{(d)} = \mathcal{H}(\sum_{s=1}^d \mathbf{A}^s)$ .

*Definition 3 (Arbitrary-Order Reachability):* Given adjacency matrix  $\mathbf{A}$  of an undirected or directed graph, we define an arbitrary-order reachability as follows:

$$\mathbf{R}^{(k)} = \mathcal{H}\left(\sum_{s=1}^k \mathbf{A}^s\right), \quad (2)$$

$\mathbf{R}^{(k)}$  is the  $k$ -order reachability of which entries  $R_{i,j}^{(k)}$  is nonzero when we can find a path  $v_i \rightsquigarrow v_j$  with  $k$  or less hops. Figure 2 illustrates the notion of arbitrary-order reachability.



**FIGURE 2.** Arbitrary-order reachability. The  $v_0 \rightsquigarrow v_{10}$  path has two different paths. One is  $(v_0, v_8, v_9, v_{10})$  path (blue dashed lines). This path provides third-order reachability. The other is  $(v_0, v_8, v_9, v_{11}, v_{10})$  path (red dashed lines). This is fourth-order reachability.

*Definition 4 (Arbitrary-Order Optimal Reachability):* Given adjacency matrix  $\mathbf{A}$  of an undirected or directed graph, we define a  $k$ -order optimal reachability; which is accessible by the *shortest path* as the following. The  $k$ -order optimal reachability matrix  $\mathbf{R}^{(k)*}$  of which nonzero element  $R_{i,j}^{(k)*}$  means that the shortest  $v_i \rightsquigarrow v_j$  path is exactly  $k$ .

The main benefit of *incremental computation* is to provide significant performance gain than computing new outputs naively. We exploit two sets of vertices  $N_i$  and  $F_i^{(k)}$  to utilize the incremental computation of reachability-related matrices. We employ the set  $N_i$  composed of the vertices linked with vertex  $v_i$ . Also, we define a footprint set  $F_i^{(k)}$  with the vertices found by vertex  $v_i$  within  $k$  steps including  $v_i$  itself. In incremental reachability computation,  $N_i$  is static and  $F_i^{(k)}$  is dynamically updated.  $\mathbf{F}^{(k)}$  is a boolean matrix of which  $i$ -th row has the nonzero element at  $j$ -th location if the  $v_i \rightsquigarrow v_j$  path has been already found during the incremental computation of reachability. Therefore,  $F_{i,j}^{(k)}$  is 1 if and only if  $v_j \in F_i^{(k)}$ . With above notations, we formally define our problem related to reachability queries as follows:

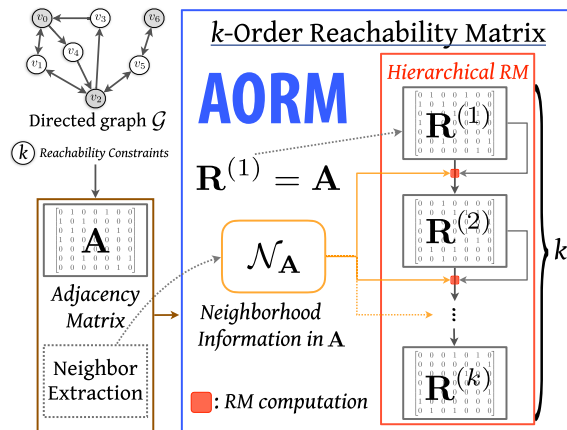
**Problem (Arbitrary-Order Reachability Queries)** Let  $\mathcal{G} = (V, E)$  be a large directed or undirected graph that has  $n$  nodes and  $m$  edges. The  $k$ -order reachability queries is denoted as  $u \rightsquigarrow v$ , where  $u$  and  $v$  are two vertices in  $\mathcal{G}$ . Here,  $u \rightsquigarrow v$  returns true if and only if there exists a path via  $k$ -hops in the graph  $\mathcal{G}$ . The problem is to compute the reachability via  $k$ -hops among every vertex in  $\mathcal{G}$  as  $k$ -order reachability matrix.

- **Input:** Adjacency matrix  $\mathbf{A}$  of  $\mathcal{G}$ , reachability constrains  $k$
- **Output:**  $k$ -order reachability matrix  $\mathbf{R}^{(k)}$

Next section, we describe the overview of the proposed algorithmic framework (AORM).

### III. AORM FRAMEWORK OVERVIEW

In this section, we introduce our proposed algorithmic framework as shown in Figure 3. Then we describe the proposed method for  $k$ -order reachability computation problem.



**FIGURE 3.** The overall architecture of arbitrary-order reachability matrix (AORM): First, we extract the neighborhood information of  $\mathbf{A}$  as  $\mathcal{N}_A$ . We assign input adjacency matrix to first-order reachability matrix  $\mathbf{R}^{(1)}$ . Then, we use  $\mathcal{N}_A$  and  $\mathbf{R}^{(1)}$  to compute the second-order reachability  $\mathbf{R}^{(2)}$  (red boxes). Likewise, we can compute the third-order reachability  $\mathbf{R}^{(3)}$  using  $\mathcal{N}_A$  and  $\mathbf{R}^{(2)}$ .

The advantage of our framework is providing *performance* and *efficiency* for arbitrary-order reachability queries. First, we extract the neighborhood information of  $\mathbf{A}$  as  $\mathcal{N}_A$ . The  $\mathcal{N}_A$  is essential information to compute the reachability in an iterative fashion. We assign the input adjacency matrix to the first-order reachability matrix  $\mathbf{R}^{(1)}$ . Then, we use  $\mathcal{N}_A$  and  $\mathbf{R}^{(1)}$  to compute the second-order reachability  $\mathbf{R}^{(2)}$ . Likewise, we can compute the third-order reachability  $\mathbf{R}^{(3)}$  using  $\mathcal{N}_A$  and  $\mathbf{R}^{(2)}$ . Based on the AORM framework, we can efficiently find the all-pairs shortest paths in  $\mathcal{G}$ . Besides, we utilize the AORM framework to extract and represent features for graph representation learning. We will describe the AORM computation algorithm in detail in Section IV.

Next, we introduce the arbitrary-order reachability matrix that answers both first- and higher-order reachability queries. One of the essential properties for performing network or graph embedding is the *proximity* between vertices in graphs. The adjacency matrix most intuitively represents the proximity of a graph. However, the adjacency matrices for massive real networks are very *sparse* [3], [4], [24]. Thus, an adjacency matrix is not sufficient for graph analytics or the feature extraction for machine learning. Consequently, we exploit the higher-order proximity gathering from the powers of an adjacency matrix, such as  $\mathbf{A}^2, \mathbf{A}^3$ , and so on.

#### A. REACHABILITY MATRIX

The ultimate goal of our research is to develop an efficient graph embedding method. In particular, we aim to capture the proximity as well as structural properties of a graph. We propose a novel incremental reachability matrix computation algorithm for unweighted, directed, and disconnected graphs. The core idea of AORM is to construct a  $k$ -order reachability matrix (RM) in an incremental fashion. We exploit the neighborhood information in first-order proximity and  $(k - 1)$ -th reachability matrix to compute  $k$ -order RM.

The  $k$ -order reachability is useful in many practical applications in graph analytics, such as product recommendations, viral marketing, and community detection. We now briefly describe why  $k$ -order reachability is essential for these applications in terms of graph analytics.

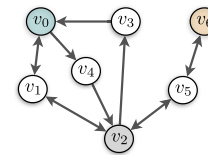
**Higher-Order Proximity:** The higher-order proximity refers to  $k$ -hop reachable relations between indirectly connected vertices, where  $k \geq 3$ . This proximity property can capture the more *global structure* of a graph rather than the structural role proximity [32], [33]. We can get the higher-order proximity between two vertices by computing the powers of the adjacent matrix  $\mathbf{A}$ . The  $\mathbf{A}^k$ , the  $k$ -th power of  $\mathbf{A}$ , has non-zero elements  $A_{i,j}^k$  if there exists any path length  $k$  from  $v_i$  to  $v_j$ . In the circumstances of real-world networks,  $\mathbf{A}^k$  is always sparse; this is because most real-world networks follow small world phenomenon. Thus, we can compute the  $\mathbf{A}^k$  in an efficient manner by the boolean matrix computation approach in a similar fashion proposed in [34]. We perform the  $\mathbf{A}^k$  computation separately by each row  $i$  using the first-order proximity information in  $\mathbf{A}$ . The  $N_i$  refers to the set of the first-order proximal neighbors of node  $i$ . Then, we can update the  $i$ -th row of  $\mathbf{A}^k$  as follows:

$$\mathbf{A}_{i,:}^k = \sum_{j \in N_i} \mathbf{A}_{j,:}^{k-1} \quad (3)$$

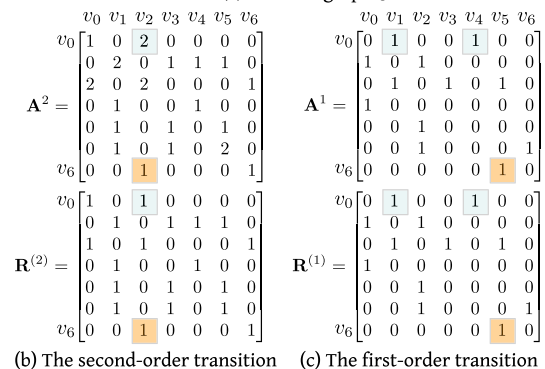
Here, the time complexity of  $\mathbf{A}^k$  computation is  $\mathcal{O}(n^3 p)$ ;  $p$  denotes the probability that an arbitrary entry of the matrix is not zero. Note that the computation  $\mathbf{A}^k$  from  $\mathbf{A}$  requires  $\mathcal{O}(kn^3 p)$  without any multiplications [34]. Besides,  $p$  is usually a small value in real-world networks, such as citation networks, social networks, and so on.

There are several existing methods for high-order proximity based on an adjacency matrix, such as, skip-gram [35] and GraRep [27]. The skip-gram model, a technique for word representation learning, employed a normalized adjacency matrix as the transition probability from node  $v_i$  to  $v_j$ . Also, the GraRep utilized normalized  $\mathbf{A}$  to produce the transition probabilities with  $k$  hops. However, this method does not consider the linear relationship among the powers of adjacency. Moreover, these two methods have the critical limitation that the powers of the adjacency matrix rapidly decays as  $k$  increases. So, these methods often failed to capture the high-order connectivity in a graph. To avoid this problem, one can employ an original adjacency matrix rather than normalized adjacency to capture the high-order proximity. However, this naive approach has some difficulties. For example, the powers of the adjacency matrix can diverge or produce a large number. To overcome this problem, we exploit the boolean reachability matrix  $\mathbf{R}^{(1)}$  instead of adjacency matrix  $\mathbf{A}$ . Here,  $R_{i,j}^{(1)}$  is TRUE if and only if there exist any path from node  $v_i$  to  $v_j$  (i.e.,  $A_{i,j} > 0$ ), otherwise FALSE. Then we compute the boolean powers of the reachability matrix  $\mathbf{R}$  as follows:

$$\mathbf{R}_{i,:}^{(k)} = \bigvee_{j \in N_i} \mathbf{R}_{j,:}^{(k-1)}, \quad (4)$$



(a) Directed graph  $\mathcal{G}$



(b) The second-order transition (c) The first-order transition

**FIGURE 4.** An example of directed graph  $\mathcal{G}$  with 7 vertices and 12 directed edges. (a) Both  $v_0$  and  $v_6$  are reachable to  $v_2$  by second-order (or 2-hop). (b) We can find these connectivity information in  $\mathbf{A}^2$  and  $\mathbf{R}^{(2)}$ . Here, the value of  $\mathbf{A}_{0,2}^2$  entry is 2. (See the green box in  $\mathbf{A}^2$ .) This is because we have two possible two-hop paths from  $v_0$  and  $v_2$ ;  $\langle v_0, v_1, v_2 \rangle$  path and  $\langle v_0, v_4, v_2 \rangle$  path. In contrast, we have only one 2-hop path from  $v_6$  to  $v_2$ ,  $\langle v_6, v_5, v_2 \rangle$  path. Therefore, the value of  $\mathbf{A}_{6,2}^2$  is 1. (See the yellow box in  $\mathbf{A}^2$ .) (c) AORM contains the lower-order connectivity information in  $\mathbf{A}^1$  and  $\mathbf{R}^{(1)}$ . These matrices include information about the different paths to the same vertex.

where  $\bigvee$  operator performs the element-wise *logical OR* operation with all the  $\mathbf{R}_{j,:}^{(k-1)}$  such that  $j$  is the first-order proximal nodes of  $v_i$ . For the computational simplicity, we use 0 for FALSE, and 1 for TRUE.

### B. HIERARCHICAL REACHABILITY REPRESENTATION

We describe the *hierarchical reachability matrix* (HRM) representation for the high-order proximity of a graph. HRM prevents the divergence problem in the powers of the adjacency matrix by boolean reachability representation. The main benefit of this representation is *incrementally* providing arbitrary-hop connectivity information. Besides, this supports the hierarchical management and the reuse of the previously obtained intermediate reachability results.

We start with an analysis that how HRM can provide an arbitrary-order reachability matrix. To do this, we explain the structure of HRM through an example of a directed graph  $\mathcal{G}$  in Figure 4-(a). Both  $v_0$  and  $v_6$  are reachable to  $v_2$  by second-order (or 2-hop). In Figure 4-(b), we can find these connectivity information in  $\mathbf{A}^2$  and  $\mathbf{R}^{(2)}$ . Here, the value of  $\mathbf{A}_{0,2}^2$  entry is 2. This is because we have two possible two-hop paths from  $v_0$  and  $v_2$ ;  $\langle v_0, v_1, v_2 \rangle$  path and  $\langle v_0, v_4, v_2 \rangle$  path. In contrast, we have only one two-hop path from  $v_6$  to  $v_2$ ,  $\langle v_6, v_5, v_2 \rangle$  path. Therefore, the value of  $\mathbf{A}_{6,2}^2$  is 1. However, the second-order reachability matrix  $\mathbf{R}^{(2)}$  in AORM describes the *existence of the paths* of  $k$ -hop reachability (Here,  $k = 2$ ). On the other hand, one might worry that it is difficult to capture accurate connectivity and proximity of a graph using a boolean reachability matrix. Fortunately, AORM contains

the lower-order connectivity information in  $\mathbf{A}^1$  and  $\mathbf{R}^{(1)}$  as shown in Figure 4-(c). This hierarchical structure enables us to capture the connectivity information of a graph based on the *arbitrary-order* reachability.

#### IV. THE AORM ALGORITHM

In this section, we introduce our proposed AORM-based three algorithms for computing reachability and all-pairs shortest paths (APSP) on directed, disconnected graphs. The first one is the  $k$ -order optimal reachability matrix algorithm, called AORM. This algorithm utilizes the parallel matrix multiplications for reachability computations. The second one is the algorithm using the hierarchical reachability matrix based on neighborhood information. This algorithm appropriates HRM to accelerate the reachability computation. The last one is the computation algorithm for APSP employing AORM algorithm. We elaborate on each of the algorithms in subsequent sections.

Before going into details, we briefly describe some notions related to reachability on a graph, such as walks, cycles, trails, and paths. A *walk* can visit the same vertices and edges more than once. A *cycle* is a closed walk where the first and last vertices are the same. On the other hand, a *trail* means a walk where all edges are unique. In particular, a *path* is a trail in which all edges and vertices are unique. One can exploit adjacency matrices to represent graphs without any loss of information. However, the vital matters to consider are information *redundancy* and *noise*, not information loss. The powers of the adjacency matrix produce all the possible walks, including trails with redundant node visits. Usually, these walks, cycles, and trails are not simple paths in a graph.

Another popular idea for capturing the connectivity is to exploit the *shortest path* between nodes, which is the measure for nodes to examine their neighbors in a graph. In contrast, in the context of random walk approaches, the accumulation of adjacency matrix powers includes noise information about the excessive random walks of little significance.

##### A. $k$ -ORDER OPTIMAL REACHABILITY

Our goal is to compute an arbitrary-order reachability matrix including *only* the all-pairs shortest paths in directed, disconnected graphs. To achieve this, we introduce a novel algorithm, called AORM, which performs the fast computation of the optimal reachability matrix. As previous mentioned in Definition 4, the element  $R_{i,j}^{(k)*}$  in  $k$ -order optimal reachability matrix  $\mathbf{R}^{(k)*}$  is nonzero if and only if the shortest path from  $v_i$  to  $v_j$  is  $k$ .

*Reachability Pruning:* Now, we describe the algorithm details for AORM and all-pairs shortest path computation using AORM. First, we perform the pruning for the following three types of insignificant walks such as cycles, trails, and longer paths to obtain an optimal reachability matrix:

- *Cycle pruning:* The powers of the reachability matrix produce the cycles that revisit the starting vertices.

- *Trail pruning:* Some trails also occur in the powers of the reachability matrix as the number of powers (i.e.,  $k$ -order) increases.
- *Paths pruning:* During the iterative reachability matrix computation, two or more paths from  $v_i$  to  $v_j$  with different hops may happen. We remove these longer paths leaving only the *shortest* paths to obtain the optimal reachability matrix.

*Cycle Pruning:* To perform cycle pruning, we apply the following equation for subtracting the diagonal matrix for the reachability matrix from the initial matrix. Here,  $\tilde{\mathbf{R}}$  refers to an intermediate result of a reachability matrix.

$$\tilde{\mathbf{R}}^{(1)} = \mathbf{R}^{(1)} - \text{diag}(\mathbf{R}^{(1)}), \quad (5)$$

where  $\text{diag}(\mathbf{X})$  is  $n \times n$  diagonal matrix whose diagonal entries are identical to the those of the matrix  $\mathbf{X} \in \mathbb{R}^{n \times n}$ . Then, we eliminate the generated cycles, which occurred in previous iterative reachability computation as follows:

$$\tilde{\mathbf{R}}_{i,:}^{(k)} = \bigvee_{j \in N_i} \tilde{\mathbf{R}}_{j,:}^{(k-1)} - \text{diag}(\bigvee_{j \in N_i} \tilde{\mathbf{R}}_{j,:}^{(k-1)}) \quad (6)$$

Thus, we can get a reachability matrix with every cycle returning to the starting vertices eliminated. However, we still have the trails with multiple visits to any other vertices different from the starting nodes.

*Trail Pruning:* To proceed trail pruning, our algorithm exploits the footprint set,  $F$ . This set refers to the set of vertices found by vertex  $v_i$  within  $k$  steps including  $v_i$  itself. The first-order optimal reachability matrix can be initialized as follows:

$$\mathbf{R}^{(1)*} = \mathbf{R}^{(1)} \quad (7)$$

$$F_i^{(1)} = \{v_i\} \cup \{v_j | R_{i,j}^{(1)} = 1\} \quad (8)$$

Initially, the first-order optimal reachability matrix  $\mathbf{R}^{(1)*}$  is initialized to be an initial reachability matrix  $\mathbf{R}^{(1)}$ . The set of found nodes with one-hop from node  $v_i$  (i.e.,  $F_i^{(1)}$ ) is initially filled with the node  $v_i$  itself, and linked nodes  $v_j$  such that  $R_{i,j}^{(1)}$  is 1.

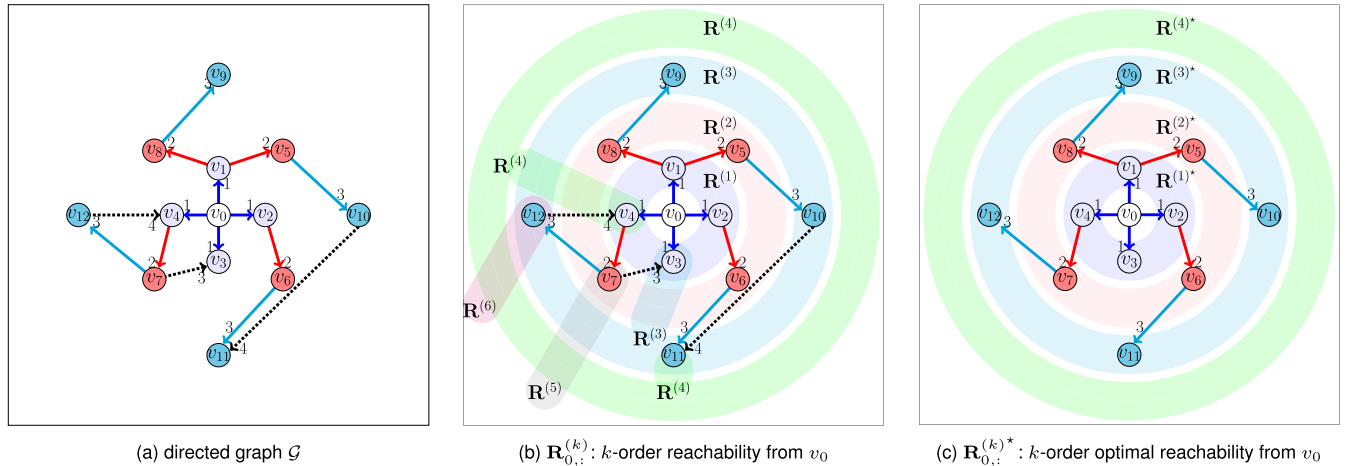
We can compute the optimal reachability matrix  $\mathbf{R}^{(k)*}$  removing any trails with redundant nodes in an incremental fashion described in Eq. (4) as follows:

$$R_{i,j}^{(k)*} = \begin{cases} 0, & \text{if } j \in F_i^{(k-1)} \\ \bigvee_{l \in N_i} R_{l,j}^{(k-1)*}, & \text{otherwise} \end{cases} \quad (9)$$

$$F_i^{(k)} = F_i^{(k-1)} \cup \{v_j | R_{i,j}^{(k)*} = 1\} \quad (10)$$

*Path Pruning:* Note that we restrict the element  $R_{i,j}^{(k)*}$  not to produce 1 (True) if any shorter path from  $v_i$  to  $v_j$  was already in the footprint set  $F_i^{(k-1)}$  in order to remove paths longer than the shortest path between two nodes. Recall that the iterative computation yields  $k$ -order optimal reachability matrix  $\mathbf{R}^{(k)*}$  by Definition 4.

The computation of each  $i$ -th row of  $\mathbf{R}^{(k+1)*}$  requires the  $c_i$  iterations for summation ( $\sum$  or  $\bigvee$ ) of  $n$ -element vectors while  $c_i$  is the number of edges from node



**FIGURE 5.** Comparison between  $k$ -order reachability matrix and  $k$ -order optimal reachability matrix: (a) a given directed graph with unit edge costs, (b) nodes are included in different color bands in accordance with the order of the reachability from the node  $v_0$  which is encoded in the row  $\mathbf{R}_{0,:}^{(k)}$ , and (c) the optimal reachability information encoded in  $\mathbf{R}_{0,:}^{(k)*}$ .

$v_i$  and  $\sum_{i=1}^n c_i = m$ . Here, the time complexity is  $\mathcal{O}(nc_i)$ . Therefore, the computation of  $\mathbf{R}^{(k)*}$  from  $\mathbf{R}^{(k-1)*}$  takes  $\mathcal{O}(nm)$  time. Besides, the shortest path from node  $v_i$  to node  $v_j$  has the length of  $k$  iff  $R_{i,j}^{(k)*}$  is nonzero. The following two lemmas support the fact that the optimal reachability matrix is the result of the path pruning:

*Lemma 1:* If  $R_{i,j}^{(k)*}$  is nonzero, no elements  $R_{i,j}^{(k-s)*}$  is nonzero for any positive integers  $s$  less than  $k$ .

*Proof:* If there exists any nonzero element  $R_{i,j}^{(k-s)*}$ , the node  $v_j$  must have been inserted in  $F_i^{(k-s)}$  which is a subset of  $F_i^{(k)}$ . If the node is in  $F_i^{(k)}$ ,  $R_{i,j}^{(k)}$  is enforced to be 0. This is contradictory to the assumption. Therefore, the lemma is true.  $\square$

*Lemma 2:* If  $R_{i,j}^{(k)*}$  is nonzero, no elements  $R_{i,j}^{(k+s)*}$  is nonzero for any positive integers  $s$ .

*Proof:* Since  $N_i^{(k)}$  is the subset of  $N_i^{(k+s)}$  and node  $v_j$  is the element of  $N_i^{(k)}$ , it is trivial that  $R_{i,j}^{(k+s)*}$  will be enforced to be 0.  $\square$

First simple idea is to accelerate the computation for summation of  $n$ -dimensional vectors using *vector parallelism*. Vector parallelism exploits data level parallelism [36]. By Flynn’s categorization in parallel processors, single instruction, multiple data (SIMD) processing exploits the same instruction broadcasting to all ALUs and execution in parallel on ALUs. This same operation can handle eight 32-bit floating-point operations or four 64-bit floating-point operations. The number of elements in a SIMD operation varies from a small number to thousands. Besides, modern standard array processing libraries such as NumPy employs SIMD vectorized operations based on SSE2 [37]. The vectorization enables the processing of  $v$ -element array to be done in  $\mathcal{O}(\delta_v)$ , and  $\delta_v \ll n$ . Thus, the improved SIMD-based computation for  $\mathbf{R}^{(k)*}$  from  $\mathbf{R}^{(k-1)*}$  takes  $\mathcal{O}(\delta_v m)$  time.

Our arbitrary-order reachability computation algorithm works by employing the *reachability pruning*. **Algorithm 1**

**Algorithm 1**  $k$ -Order Reachability Matrix (AORM)

```

Input: Adjacency matrix  $\mathbf{A} \in \mathbb{R}^{n \times n}$ ,
          Reachability constraints  $k_c$ 
Output: Sequence of reachability matrix  $\mathbf{R}^{(k)*}$ ,
          where  $\mathbf{R}^{(k)*} \in \mathbb{R}^{n \times n}$  for  $k \in \{2, \dots, d\}$ 

    • Phase #1. Initialization
    1:  $k \leftarrow 1$ 
    2:  $\mathbf{R}^{(k)*} \leftarrow \mathbf{A}$   $\triangleright$  initialize reachability matrix
    3:  $\mathbf{F} \leftarrow \mathbf{I} + \mathbf{A}$   $\triangleright$  initialize footprint matrix

    • Phase #2. Incremental  $\mathbf{R}^{(k)*}$  via matrix multiplication
    4: while  $\mathbf{R}^{(k)*}$  is not converged or  $k < k_c$  do
    5:    $k \leftarrow k + 1$ 
    6:    $\triangleright$  perform vectorized reachability pruning using @ op.
    7:    $\triangleright$  @: SIMD-based matrix multiplication
    8:    $\mathbf{R}^{(k)*} \leftarrow \mathcal{H}(\mathcal{H}(\mathbf{A} @ \mathbf{R}^{(k-1)*}) - \mathbf{F})$ 
    9:    $\mathbf{F} \leftarrow \mathbf{R}^{(k)*} + \mathbf{F}$   $\triangleright$  update a footprint matrix
    10:  yield  $\mathbf{R}^k$ 
    11: end while
    
```

presents the  $k$ -order reachability matrix computation algorithm. This algorithm receives the adjacency matrix  $\mathbf{A}$  and the reachability constraints  $k_c$  as inputs. Then the algorithm yields the sequence of  $\mathbf{R}^{(k)*}$  for a given  $k = 2, 3, \dots, d$ , where  $d$  is a diameter of a graph. This algorithm consists of two phases: the initialization phase and the incremental *optimal reachability matrix*  $\mathbf{R}^{(k)*}$  computation phase via matrix multiplication. In the initialization phase, first, the algorithm initializes a hop count variable, reachability matrix, and footprint matrix (Line 1–3). In the incremental  $\mathbf{R}^{(k)*}$  computation phase, it performs  $k$ -order optimal reachability matrix computation via the vectorized reachability pruning (Line 6). The  $\mathcal{H}(x)$  function is heaviside step function (or unit step function). This function returns 1 if  $x$  is larger than 0, and 0 if otherwise. Then it updates a footprint matrix (Line 7). Then it incrementally produces the optimal reachability matrix



according to the hop count (Line 8). It performs iteratively this procedure until the  $k$ -order reachability matrix is converged or reachability constraints are reached (Line 4–9).

To present a more detailed AORM procedure, we describe how we can obtain the optimal reachability with the example in Figure 5. We mark nodes with different color bands according to the order of reachability from  $v_0$ . Also, we express the directed edges with different colors by the path length from  $v_0$ . Let  $S \subset E$  be a subset of edges in an arbitrary-order shortest path  $S$  and  $\bar{S} = E \setminus S$  be a subset of edges included in longer paths than  $S$ . For instance, given the directed graph  $\mathcal{G}$ , we have  $\bar{S} = \{e_{7,3}, e_{10,11}, e_{12,4}\}$  in Figure 5 (dashed black edges). Besides, the color bands represent the information encoded in  $\mathbf{R}_{0,:}^{(k)}$ . More specifically, the node  $v_{11}$  appears in  $\mathbf{R}_{0,:}^{(3)}$  and  $\mathbf{R}_{0,:}^{(4)}$  because of the edge  $e_{6,11}$  and  $e_{10,11}$  respectively. In particular, the edge  $e_{12,4}$  generate a cycle. Thus,  $v_7$  and  $v_{12}$  appear again in  $\mathbf{R}_{0,:}^{(5)}$  and  $\mathbf{R}_{0,:}^{(6)}$  respectively. Even though the diameter of the graph is only 3, these nodes will appear in  $\mathbf{R}_{0,:}^{(5+3s)}$  and  $\mathbf{R}_{0,:}^{(6+3s)}$  for every integer  $s \geq 0$  due to the cycle.

*Convergence Property:* We can get the  $k$ -order optimal reachability matrix via reachability pruning as shown in Figure 5-(c). The color bands in the figure shows the information encoded in  $\mathbf{R}_{0,:}^{(k)*}$ . Note that  $\mathbf{R}_{0,:}^{(k)*}$  is  $\mathbf{0}$  when  $k$  is greater than the diameter of the graph. So, each node appears in *only one* of the color bands representing the  $k$ -order optimal reachability. The  $\mathbf{R}^{(k)*}$  converges to  $\mathbf{0}$  as  $k$  increases according to the reachability convergence characteristic. Hence, the maximum  $k$  with non-zero  $\mathbf{R}^{(k)*}$  is the diameter of the graph. Recall that we can not provide this property via  $\mathbf{A}^k$  and  $\mathbf{R}^{(k)}$ . One can utilize a normalized adjacency to compute the proximity for graph embedding. However, the powers of the adjacency matrix only decay but never converge to  $\mathbf{0}$ . The convergence of  $\mathbf{R}^{(k)}$  makes it possible for the proximity measure based on the powers to determine the truncation order. We can exploit a weighted sum of a finite number of the  $k$ -order optimal reachability matrix as a proximity measure for the higher-order representation learning of graphs as follows:

$$\mathbf{P} = \sum_{k=1}^d w_k \mathbf{R}^{(k)*}, \quad (11)$$

where  $w_k$  is the weight for the  $k$ -th reachability matrix. Here, we can define weights for the significance of each order of reachability simply  $w_k = 1/k$ . Another approach to finding the appropriate weights is to employ machine learning techniques using deep learning models such as Deepwalk [38], node2vec [39], and LINE [26].

## B. INCREMENTAL $k$ -ORDER REACHABILITY

Although SIMD-based AORM, called M-AORM, in **Algorithm 1** accelerates the matrix multiplication operations on relatively small graphs, it still requires substantial time to handle massive sparse graphs in practice. To overcome this limitation, we present incremental AORM to utilize the summation of non-zero elements in the hierarchical reachability matrix based on neighborhood information. Our algorithm

---

### Algorithm 2 Incremental $k$ -Order Reachability Matrix (I-AORM)

---

**Input:** Adjacency matrix  $\mathbf{A} \in \mathbb{R}^{n \times n}$ ,  
Reachability constraints  $k_c$

**Output:** Sequence of reachability matrix  $\mathbf{R}^{(k)*}$ ,  
where  $\mathbf{R}^{(k)*} \in \mathbb{R}^{n \times n}$  for  $k \in \{2, \dots, d\}$

• **Phase #1.** Initialization

1:  $k \leftarrow 1$   
2:  $\mathbf{R}^{(1)*} \leftarrow \mathcal{H}(\mathbf{A})$   $\triangleright$  initialize reachability matrix  
3:  $\mathbf{F} \leftarrow \mathbf{I} + \mathbf{R}^{(1)*}$   $\triangleright$  initialize footprint matrix  
   $\triangleright$  construct neighbors information  $\mathcal{N}_{\mathbf{A}}$

4: **for**  $i \in \{0, \dots, n-1\}$  **do**

5:  $N_i \leftarrow \text{extract\_neighbors}(\mathbf{A}_{i,*})$

6: **end for**

• **Phase #2.** Incremental  $\mathbf{R}^{(k)*}$  computation

7: **while**  $\mathbf{R}^{(k)*}$  is not converged **or**  $k < k_c$  **do**

8:  $k \leftarrow k + 1$

$\triangleright$  compute an optimal reachability matrix  $\mathbf{R}^{(k)*}$

9: **for**  $i \in \{1, \dots, n\}$  **do**

10:  $\mathbf{R}_{i,:}^{(k)*} \leftarrow \mathcal{H}(\sum_{l \in L_i} \mathbf{R}_{l,:}^{(k-1)*})$

11: **end for**

$\triangleright$  perform reachability pruning

12:  $\mathbf{R}^{(k)*} \leftarrow \mathcal{H}(\mathcal{H}(\mathbf{R}^{(k)*}) - \mathbf{F})$

13:  $\mathbf{F} \leftarrow \mathbf{R}^{(k)*} + \mathbf{F}$   $\triangleright$  update a footprint matrix

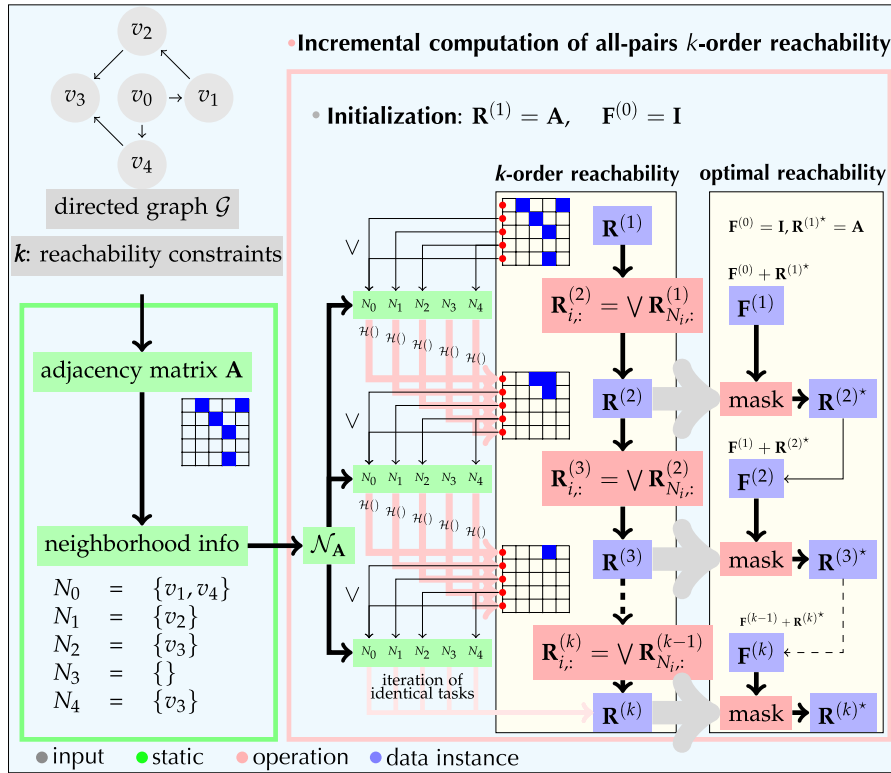
14: **yield**  $\mathbf{R}^{(k)*}$

15: **end while**

---

so-called I-AORM consists of two phases: the initialization phase and the incremental  $\mathbf{R}^{(k)*}$  computation phase as shown in **Algorithm 2**. In the initialization phase, first, the algorithm initializes a hop count variable, reachability matrix, and footprint matrix (Line 1–3). Then it constructs neighbors information  $\mathcal{N}_{\mathbf{A}}$  (Line 4–6). Based on  $\mathcal{N}_{\mathbf{A}}$ , the algorithm performs the computation of an optimal reachability matrix  $\mathbf{R}^{(k)*}$  (Line 9–11) with reachability pruning (Line 12–13) in the second phase. For each node, it updates the optimal reachability according to the reachability constraint (Line 9). Then it performs the reachability pruning (Line 12) and updates a footprint matrix (Line 13). It performs iteratively this procedure until the  $k$ -order reachability matrix is converged or reachability constraints are reached (Line 7).

*Architecture Details:* Figure 6 illustrates the I-AORM framework. First, we extract the neighborhood information  $\mathcal{N}_{\mathbf{A}}$  from  $\mathbf{A}$ . Then, we exploit this information to perform incremental computation for reachability matrices from the order of 1 to  $k$ . This information is static throughout the computation. Thus, computational cost for each iteration for incremental computation is also constant. Each iteration, we utilize the previous order reachability  $\mathbf{R}^{(k-1)}$  to compute the  $k$ -order reachability  $\mathbf{R}^{(k)}$ . Our algorithm constructs the  $i$ -th row of  $\mathbf{R}^{(k)}$  in accordance with the neighbor information  $N_i$  by aggregating all  $\mathbf{R}_j^{(k-1)}$  such that  $j \in N_i$ . We easily obtain the  $k$ -order optimal reachability matrix  $\mathbf{R}^{(k)*}$  by applying the footprint matrix  $\mathbf{F}$  as mask to the reachability matrix. Then, the framework keeps the nonzero elements in the reachability



**FIGURE 6. I-AORM framework: First, I-AORM extracts the neighborhood information  $\mathcal{N}_A$ . Then, we exploit this information to perform incremental computation for reachability matrices from the order of 1 to  $k$ . Our algorithm constructs the  $i$ -th row of  $\mathbf{R}^{(k)}$  using  $\mathcal{N}_A$  and  $\mathbf{R}_{i,:}^{(k-1)}$ . For reachability pruning we use the footprint matrix  $\mathbf{F}$  as mask to the reachability matrix. This procedure continues until reachability converged or the reachability constraints are satisfied.**

matrix to  $\mathbf{F}$  to compute the next order optimal reachability matrix. This procedure continues until reachability converged or the reachability constraints are satisfied. Recall that the shortest path from node  $v_i$  to node  $v_j$  has the length of  $k$  if and only if  $\mathbf{R}_{i,j}^{(k)*}$  is nonzero. In this context, we can efficiently exploit the proposed algorithm for incremental APSP computation with  $k$ -order optimal reachability matrix. We can easily determine the weight  $w_s$  in Eq. (11) to yield all-pair shortest paths matrix  $\mathbf{D}$  based on the Lemma 1 and Lemma 2. We now introduce the following Theorem called the Incremental APSP theorem adapting the AORM convergence property to the APSP problem.

*Theorem 1 (Incremental APSP):* Let  $\mathbf{R}^{(k)*}$  be the  $k$ -order optimal reachability matrix for a unweighted, directed graph, diameter  $d$  and order  $k$ . Then the distance matrix for APSP is

$$\mathbf{D} = \sum_{k=1}^d k\mathbf{R}^{(k)*}, \quad (12)$$

where  $k = 1, 2, \dots, d$ . Here, the element  $D_{i,j} \in \mathbf{D}$  means the length of the shortest path from  $v_i$  to  $v_j$ .

*Proof:* If the shortest path from node  $v_i$  to  $v_j$  has the length of  $k$ ,  $\mathbf{R}_{i,j}^{(k+s)*}$  is 1 iff  $s = 0$  by the Lemma 1 and Lemma 2. Thus,  $D_{i,j} = k$  means  $\sum_{k=1}^d k\mathbf{R}_{i,j}^{(k)*}$ .  $\square$

Note that we figure out the diameter  $d$  when all the entries in  $\mathbf{R}^{(k)*}$  is zero; this means no longer path with a length of  $k + s$  with a positive integer  $s$  exists. Thus, we can terminate the iterative computation by checking whether the current optimal reachability matrix is  $\mathbf{0}$  or not. The pseudo code in Algorithm 3 shows how we compute all-pairs shortest paths based on the AORM. This algorithm receives the adjacency matrix  $\mathbf{A}$  and the reachability constraints  $k_c$  as inputs. Then the algorithm returns the all-pairs shortest path distance matrix  $\mathbf{D}$  according to the reachability constraints  $k_c$ . First, the algorithm sets the reachability constraints (Line 1). Then, it assigns an initial distance matrix as an input adjacency matrix (Line 2).

And it creates an AORM instance as an iterator (Line 3). Based on the Theorem 1, it incrementally computes the all-pairs shortest path distance matrix according to the reachability constraints (Line 4–6). This computation terminates if the reachability matrix is converged or the reachability constraint is satisfied. Note that the algorithm performs the reachability matrix computation in a *whole graph-wise* fashion via matrix operations.

### C. COMPLEXITY ANALYSIS

The complexity analysis of our algorithms includes the following two aspects: reachability computation for exact APSP

**Algorithm 3** All-Pairs Shortest Paths Computation

---

**Input:** Adjacency matrix  $\mathbf{A} \in \mathbb{R}^{n \times n}$   
 Reachability constraints  $k_c$  [default:  $k_c = \infty$ ]

**Output:** All-pairs shortest path distance matrix  $\mathbf{D} \in \mathbb{R}^{n \times n}$

- 1:  $k \leftarrow k_c$  ▷ set the reachability constraints
- 2:  $\mathbf{D} \leftarrow \mathbf{A}$
- 3: AORMITERATOR  $\leftarrow$  AORM( $\mathbf{A}, k$ )  
▷ perform incremental APSP computation
- 4: **for** each  $\mathbf{R}^{(k)*}$  from AORMITERATOR **do**
- 5:    $\mathbf{D} \leftarrow \mathbf{D} + k\mathbf{R}^{(k)*}$
- 6: **end for**
- 7: **return**  $\mathbf{D}$

---

and arbitrary-hop constrained APSP with reachability constraints for graph embedding applications.

*Reachability Computation for Exact APSP:* The time complexity of AORM optimal reachability computation for an unweighted, directed graph  $\mathcal{G}$  is  $\mathcal{O}(nm)$ . Hence, time complexity of exact APSP is  $\mathcal{O}(nmd)$ , where  $d = \text{diam}(\mathcal{G})$  or the diameter of  $\mathcal{G}$ . Furthermore, by exploiting the vectorized processing for matrix column-wise sum operations, we improve the time complexity to  $\mathcal{O}(\delta_v md)$ , where  $\delta_v = \frac{n}{\gamma}$  and  $\gamma$  denotes the number of elements in a SIMD operation.<sup>2</sup> The nonzero elements of  $\mathbf{R}^k$  are the  $k$ -th level nodes visited by the breadth-first search (BFS) from all the nodes.

It is well known that the time complexity of BFS for finding the single-source shortest paths is  $\mathcal{O}(n + m)$ . Then, we can extend this algorithm for all  $n$  nodes to compute the all-pair shortest paths. Therefore, the time complexity of BFS-based all-pair shortest paths computation is  $\mathcal{O}(n^2 + nm)$ . The complexity of one of the best previous algorithms is  $\mathcal{O}(n^\omega \log n)$ , where  $n^\omega$  denotes the time required for the multiplication of two  $n \times n$  matrices [8]. The best known algorithm for solving APSP problem is proposed in [40]. However, both methods are only applicable to *undirected* graphs. It is also proposed to solve the APSP problem with directed graphs [41]. Nevertheless, this method demands more time than the other APSP algorithms for undirected graphs.

In contrast, our AORM-based APSP algorithm via the power iteration of the reachability matrix works on directed graphs without any difficulties. Furthermore, our algorithm can handle even *disconnected* graphs. The time complexity of the method is  $\mathcal{O}(\delta_v md)$ . Most real networks such as temporal reachability networks, citation networks, and collaboration networks are sparse. So,  $\delta_v$  is small enough compared to  $n$ . Consequently, AORM performance only depends on  $|E|$  and  $\text{diam}(\mathcal{G})$ . Therefore, the proposed algorithm is suitable for massive graphs with sparse edges.

*Arbitrary-Hop Constrained APSP:* One of the vital problems in massive graph mining is to capture the local neighbor structure of a node as fast as possible. To solve this problem efficiently, we need to exploit arbitrary-hop constrained

reachability computation for massive graphs [42], [43]. In the context of traffic congestion prediction (our motivating example in Section I), all-pairs shortest paths are less meaningful for predicting the diffusion of the future congestion in the near future [7]. In particular, congestion of a road segment does not affect the congestion from road segments reachable via many hops in a road network. Motivated by this example, AORM-based APSP algorithm in **Algorithm 3** can handle the arbitrary-hop constrained APSP for practical proximity analysis. Moreover, in the context of graph embedding, the relationship between two nodes far away from each other does not show a significant similarity in massive networks. Specifically, our arbitrary-hop constrained reachability algorithm is beneficial for node embedding approaches based on the shortest path distance [44]. Therefore, once we regard the distance greater than  $k$ -hop constraints  $d_k$  as an unimportant factor, we can cut off the sequence of the power iteration of the reachability matrix with  $d_k$ . The time complexity for our arbitrary-hop constrained APSP algorithm is  $\mathcal{O}(\delta_v md_k)$ . Lastly, the space complexity of Algorithm 1, 2, and 3 is  $\Theta(n^2)$  since these algorithms require a constant number of matrices.

**D. INCREMENTAL GIRTH**

We now present another attractive characteristic of AORM for graph analytics. The *girth* of a graph is an essential graph quantity to analyze the characteristics of a directed graph in graph mining [45]–[47]. A  $q$ -cycle is a cycle of length  $q$ . If a directed graph  $\mathcal{G}$  has a  $q$ -cycle, where  $q$  is the minimum integer, then  $q$  means the girth of  $\mathcal{G}$ ; denoted by  $g(\mathcal{G})$  [46].

In our work, we can compute the girth of a unweighted directed graph via AORM with a particular operator, SEMIN operator. In addition, our algorithm provides the lengths of the shortest cycle and the longest one for each node. Note that the shortest cycle has the smallest accessibility distance since our algorithm focuses on handling directed unweighted graphs which can be even disconnected.

*Definition 5 (Selection of Element-wise Minimum):* Let a matrix  $\mathbf{X} = [a_{ij}]_{0 \leq i, j \leq n}$  be given as the following:

$$\mathbf{X} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} \quad (13)$$

For simplicity, we use this matrix notation  $\mathbf{X}$  as  $[a_{ij}]$ , where  $i, j \in [n]$ . We define the selection of element-wise minimum (SEMIN) operator,  $\mathcal{S}_j^u$ . The SEMIN operator selects the element-wise minimum positive value among given matrices  $\mathbf{X}_s$  and returns a matrix  $\mathbf{X}_m$  as follows.

$$\mathcal{S}_{k=1}^s(\mathbf{X}_k) = \min(\mathbf{X}_1, \mathbf{X}_2, \dots, \mathbf{X}_s) \quad (14)$$

$$= \min_{i, j \in [n]} ([a_{ij}]_1, [a_{ij}]_2, \dots, [a_{ij}]_s) = \mathbf{X}_m, \quad (15)$$

where  $k = 1, 2, \dots, s$ .

For  $\mathcal{G}$ , let  $\mathcal{G}_k$  be the  $k$ -hop constrained reachable configuration, and let  $k$  be a positive integer. In other words, every node in  $\mathcal{G}$  has at most  $k$ -order reachability to other nodes. Also,

<sup>2</sup>The bit widths of SIMD in modern commodity CPUs are 128, 256, and 512. Hence,  $16 \leq \gamma \leq 64$  in case of the byte operations.

if  $k = d$ , every node can traverse a graph without any hop constraints, where  $d$  denotes the diameter of  $\mathcal{G}$ .

*Theorem 2 (Incremental Girth:)* For any unweighted directed graph  $\mathcal{G}$ , and unknown girth  $g(\mathcal{G})$ , we can compute the incremental girth  $g(\mathcal{G}_k)$  via AORM and SEMIN operator under  $k$ -hop constrained reachable configuration  $\mathcal{G}_k$  as follows.

$$\mathbf{C}^{(k)} = \prod_{i=2}^k (i+1) \mathcal{H}(\text{diag}(\mathbf{A}\tilde{\mathbf{R}}^{(i)})) \quad (16)$$

$$g(\mathcal{G}_k) = \min_{i \in [n]} \{C_{i,i}\} \quad (17)$$

where  $\mathbf{C}^{(k)}$  denotes the diagonal matrix for  $\mathcal{G}_k$ , its diagonal entry  $C_{i,i}$  refers to the incremental girth under  $k$ -hop constrained reachable configuration, and  $d$  is a diameter of  $\mathcal{G}$ .

*Proof:* Assume we have distinct vertices  $v_1, v_2, \dots, v_s$ , where  $s = k - 1$ . A path  $\mathcal{C} = \langle v_1, v_2, \dots, v_s, v_k \rangle$  is a  $s$ -cycle if and only if  $v_1 = v_k$ . Here,  $\tilde{\mathbf{R}}^{(k)}$  is the  $k$ -order reachability matrix before performing the reachability pruning. Thus, the element  $\tilde{r}_{i,s}^{(k)}$  has 1. Since  $v_s \in N_i$ ,  $\mathbf{A}\tilde{\mathbf{R}}^{(i)}$  will produce a matrix with  $i$ -th diagonal element being nonzero. At arbitrary-order  $k$  for reachability, the cycle length at  $k$  is  $k + 1$ . Thus, the minimum of  $C_{i,i}$  is the girth of  $\mathcal{G}_k$ . Furthermore, the maximum of  $C_{i,i}$  means the longest cycle length in  $\mathcal{G}_k$ . So, the result of  $\min\{C_{i,i}\}$  for  $\mathbf{C}^{(k)}$  at  $k = d$  is the *girth* of the given graph  $\mathcal{G}$ . According to the reachability order  $k$ , cycle length  $l$  is 3 and  $d + 1$  from  $\tilde{\mathbf{R}}^{(2)}$  and  $\tilde{\mathbf{R}}^{(d)}$ , respectively;  $3 \leq l \leq d + 1$ . Hence the computation result via AORM and SEMIN operator on  $\mathcal{G}_k$  according to  $k$  is incremental girth  $g(\mathcal{G}_k)$ .  $\square$

*Application to Versatile Feature Generation:* The rows of all-pairs distance matrix  $\mathbf{D}$  has diagonal of 0's, and the shortest cycle length (girth) matrix  $\mathbf{C}$  is a diagonal matrix. Therefore, we can generate a feature matrix by performing the addition of two matrices as  $\mathbf{D} + \mathbf{C}$ .

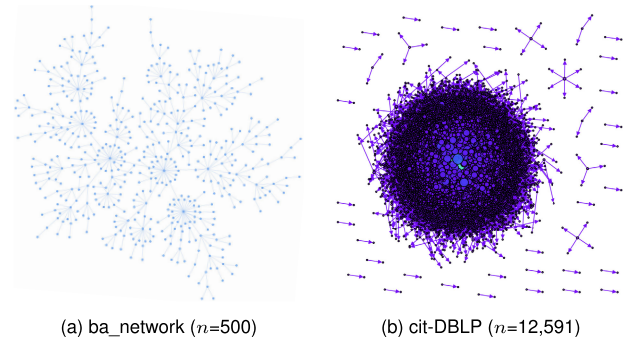
Besides, we utilize the matrix  $\mathbf{D}$  to produce the weighted sum of  $\mathbf{R}^{(k)*}$  with several weights for  $k$ . For example, if we want to obtain  $\sum_{k=1}^d w_k \mathbf{R}^k$  with  $w_k = 1/k$ , then we can compute the weighted sum as follows:

$$\sum_{k=1}^d \frac{1}{k} \mathbf{R}^{(k)*} = \mathbf{1} \oslash^{\tilde{0}} \mathbf{D}, \quad (18)$$

where  $\oslash^{\tilde{0}}$  is the same as the Hadamard division  $\oslash$  except that the result is 0 when the divider is 0. Moreover, we can obtain the weighted sum with  $w_k = 1/k^2$  as  $\sum_{k=1}^d \frac{1}{k^2} \mathbf{R}^{(k)*} = (\mathbf{1} \oslash^{\tilde{0}} \mathbf{D})^2$ . To provide general form, we can generate the weighted sum with  $w_k = w(k)$  by employing  $\mathcal{W}(\mathbf{X})$ ; this performs the element-wise application of the function  $w(\cdot)$  to each element of the matrix  $\mathbf{X}$  as follows:

$$\sum_{k=1}^d w(k) \mathbf{R}^{(k)*} = \mathcal{W}(\mathbf{D}) \quad (19)$$

In graph embedding, one can create the node features in diverse ways. The proposed method for versatile feature generation enables us to embed different features for graph representation learning. Furthermore, we can perform the



**FIGURE 7.** Datasets of a random graph and a real world network. (a) The **ba\_network** is a random graph based on Barabási-Albert model. (b) The **cit-DBLP** is one of the citation networks that is a disconnected directed real-world network.

comparison between the embedding results. Thus, our versatile feature generation method is useful for obtaining several weighted sums in a flexible manner applicable to graph embedding.

## V. EXPERIMENTS

In this section, we conduct extensive experiments on several types of synthetic and real-world networks for verifying the effectiveness of the AORM framework.

### A. EXPERIMENTS SETUP

We perform all our experiments on the single machine with Intel Xeon E5 3GHz, 64 GB memory. We implement the AORM framework using Python. Also, we utilize the NetworkX library [48] to create random graphs and compare the APSP performance comparison.

*Datasets:* We summarize data statistics of synthetic networks and real-world networks in Table 2 and Table 3, respectively. Figure 7 shows examples of synthetic datasets and real-world datasets; a Barabási-Albert model-based random graph and a citation network, respectively.

For synthetic network experiments, we exploit twenty datasets in Table 2. These datasets are random networks generated from several random graph generation models, such as Barabási-Albert, Erdős-Rényi, random geometric graph, power-law cluster graph, and random  $k$ -out graph models. For the objective evaluation and reproducibility, we utilize the seventeen public benchmark datasets from the network repository [29]. We also employ the other three datasets generated by our implemented random graph generator exploiting the NetworkX API. Here, the random  $k$ -out graph is a connected, *directed* network.

For real-world network experiments, we employ twenty real-world networks for APSP performance evaluation. We obtain these datasets from the SNAP [28] and the network repository [29]. These datasets have various network types such as directed networks, citation networks, collaboration networks, temporal reachability networks, social networks, and road networks in Table 3.

*Baselines:* We perform a comparison of the APSP computation performance of the AORM against the following

**TABLE 2.** Data statistics of synthetic networks. We conduct the experiments for twenty synthetic networks. These networks include seventeen datasets from the network repository [29] and three datasets (marked with \*) generated from our implemented random graph generator exploiting the NetworkX API. Most datasets are connected and undirected networks. On the other hand, we exploit three disconnected networks (marked with †) and one directed network (marked with ‡). (Model - BA: Barabási–Albert, ER: Erdős–Rényi, GEO: random geometric graph, PLC: powerlaw cluster graph, KOG: random  $k$ -out graph).

Dataset	# Nodes	# Edges	Avg. degree	Model
ba_1k_20k	1,000	19,600	39.2	BA
ba_1k_30k	1,000	30,039	60.1	BA
ba_1k_40k	1,000	40,236	80.5	BA
ba_1k_100k	1,000	100,000	12	BA
ba_1k_150k	1,000	150,144	300.3	BA
er_graph_1k_8k	1,000	8,000	16	ER
er_graph_1k_30k	1,000	30,000	60	ER
er_graph_1k_60k	1,000	60,000	120	ER
er_graph_1k_100k	1,000	100,000	200	ER
er_graph_1k_200k	1,000	37,330	74.7	ER
geo1k_50k	1,000	50,000	100	GEO
geo1k_100k	1,000	100,000	200	GEO
geo1k_150k	1,000	150,000	300	GEO
geo1k_200k	1,000	200,000	400	GEO
CL-10k-1d8-trial1†	9,221	44,256	9.6	PLC
CL-10k-1d9-trial2†	9,236	36,356	7.9	PLC
CL-10k-1d8-trial3†	9,251	43,811	9.5	PLC
BA-graph*	30,000	359,928	24	BA
PLC-graph*	5,000	118,977	47.59	PLC
k-out-graph*‡	10,000	298,979	59.8	KOG

three baseline methods. The baseline methods used in our work are currently known to be superior in terms of computational complexity (Seidel’s method [8]) and the most widely employed (NetworkX [48] and vectorized BFS-based approaches) in the network analytics community. Besides, these methods are publicly available for performance comparison.

- **P-SM** [8]: This method is Seidel’s method using vectorized matrix multiplications.
- **NetworkX** [48]: NetworkX is an open-source network analysis library in Python. This library supports many standard graph algorithms including APSP computation. We exploit APSP computation API in NetworkX to perform performance comparison.
- **V-BFS** [49]: Breadth First Search (BFS) is a means of traversing the portion of the graph that is reachable from a particular vertex. We can get the APSP via computation of Single-Source Shortest Paths (SSSP) for each node in a directed unweighted graph. We implement the vectorized BFS-based algorithm to accelerate APSP computations via SIMD utilization.

We perform the APSP computation as described **Algorithm 3** which may utilize either **Algorithm 1** or **2**. Based on the AORM algorithm, the method can be distinguished as follows:

- **M-AORM**: We implement the APSP algorithm using the vectorized matrix multiplication-based AORM.
- **I-AORM**: We implement the APSP algorithm via incremental AORM with vectorized column-wise summation based on the first-order reachability.

The P-SM method *only* supports *connected unweighted undirected* graphs due to its recursive algorithm nature.

**TABLE 3.** Data statistics of real-world networks. All directed networks and citation networks are directed graphs. (Type - DN: directed networks, CIT: citation networks, CA: collaboration networks, TRN: temporal reachability networks, SOC: social networks, RN: road networks).

Type	Dataset	# Nodes	# Edges	Avg. degree
DN	wiki-Vote	7,115	100,762	28.32
	p2p-Gnutella04	10,876	39,994	7.35
	p2p-Gnutella24	26,518	65,369	4.93
	email-Eu-core	986	16,064	32.58
CIT	soc-Slashdot	82,168	948,464	23.09
	cit-DBLP	12,591	49,635	7.88
	cit-HepPh	34,546	421,578	24.37
	cit-HepTh	27,770	352,807	25.27
CA	ca-GrQc	4,159	13,422	6.46
	ca-Erdos992	5,094	7,515	2.95
	ca-HepPh	11,204	117,619	20.99
TRN	CollegeMsg	1,899	12,838	14.57
	scc_infect-dublin	10,972	175,573	32.00
	scc_twitter-copen	2,623	473,614	361.12
SOC	ego-facebook	4,039	88,234	43.69
	soc-epinions	26,588	100,120	7.53
	socfb-Maryland58	20,871	744,862	71.37
RN	road-chesapeake	39	170	8.72
	road-minnesota	2,642	3,303	2.50
	road-euroroad	1,174	1,417	2.41

To compare with P-SM, we perform the APSP computation performance on connected undirected random graphs based on several generation models; Erdős–Rényi, Barabási–Albert, random geometric graph, and powerlaw clustered graph models [50].

## B. APSP PERFORMANCE

In this section, we evaluate the performance of APSP computation compared with baseline methods on synthetic graphs and diverse real-world networks. Tables 4 and 5 show the performance comparison results of APSP computation for random graphs and real-world networks, respectively. According to Table 4 and 5, our proposed methods, M-AORM and I-AORM, outperform three baselines. We perform the comparison between our methods and three baselines for ten different random networks based on Barabási–Albert growth model in Figure 8. The result shows that our method outperforms NetworkX up to 11 times.

*Incremental APSP:* We conduct the experiments for incremental APSP computation on the directed network (email-Eu-core) and the collaboration network (ca-HepPh) in Table 3. For our experiments, we set reachability constraints according to the diameters of these networks. Note that P-SM does not support the incremental APSP computation due to its recursive algorithm nature. Our methods (I-AORM, M-AORM) are up to 10X and 11.1X faster than the NetworkX in Figure 9 (a) and (b), respectively.

## C. ASYMPTOTIC CORRECTNESS

In this section, we evaluate the asymptotic correctness of our approach. Our AORM approach provides fast convergence property for asymptotic correctness. We perform the experiment on a random graph with 5,000 nodes based on Erdős–Rényi model for the proposed AORM-based incremental APSP computation. The elements of approximate

**TABLE 4.** Performance comparison of APSP computation for synthetic networks. We mark three disconnected graphs and one directed network as † and ‡, respectively. The P-SM cannot handle these networks due to its recursive algorithm nature. Here, bold font denotes the fastest running time. (Methods - P: P-SM, NX: NetworkX, V-B: V-BFS, M-A: M-AORM, I-A: I-AORM, unit: seconds).

Dataset	P-SM	NX	V-B	M-A	I-A
ba_1k_20k	25.1	0.84	1.09	<b>0.71</b>	1.05
ba_1k_30k	26.63	1.28	1.19	<b>0.73</b>	1.25
ba_1k_40k	26.18	1.33	1.29	<b>0.69</b>	1.41
ba_1k_100k	2.39	11.71	0.77	<b>0.36</b>	2.13
ba_1k_150k	5.63	3.49	1.08	<b>0.44</b>	2.54
er_graph_1k_8k	10.24	2.66	0.66	0.56	<b>0.41</b>
er_graph_1k_30k	27.12	1.41	1.29	<b>0.64</b>	0.98
er_graph_1k_60k	5.79	2.07	1.18	<b>0.44</b>	1.01
er_graph_1k_100k	5.89	3.1	1.18	<b>0.49</b>	1.80
er_graph_1k_200k	27.03	1.74	1.28	<b>0.58</b>	1.14
geo1k_50k	44.49	1.97	1.43	<b>0.65</b>	1.36
geo1k_100k	26.98	2.97	1.29	<b>0.52</b>	1.94
geo1k_150k	26.45	3.88	1.24	<b>0.47</b>	2.48
geo1k_200k	26.3	5.01	1.12	<b>0.42</b>	3.01
CL-10k-1d8-trial1†	N/A	134.82	728.99	209.44	<b>70.37</b>
CL-10k-1d9-trial2†	N/A	123.83	679.09	165.05	<b>53.28</b>
CL-10k-1d8-trial3†	N/A	140.2	721.89	176.01	<b>57.49</b>
BA-graph	52303	37006	37942	12127	<b>350</b>
PLC-graph	1670.09	131.59	155.47	29.51	<b>11.20</b>
k-out-graph‡	N/A	399.89	1141.85	476.91	<b>71.39</b>

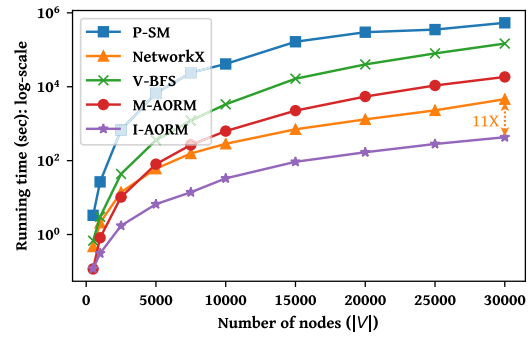
**TABLE 5.** Running time comparison of APSP computation for real-world networks. Note that P-SM is not applicable for directed networks and disconnected networks. Here, bold font denotes the best performance. (Methods - P: P-SM, NX: NetworkX, V-B: V-BFS, M-A: M-AORM, I-A: I-AORM, unit: seconds).

Dataset	P-SM	NX	V-B	M-A	I-A
wiki-Vote	N/A	117.67	285.48	105.03	<b>24.65</b>
p2p-Gnutella04	N/A	341.95	666.51	529.00	<b>146.26</b>
p2p-Gnutella24	N/A	2703.1	5599.5	2857.1	<b>863.7</b>
email-Eu-core	N/A	3.103	1.484	<b>0.456</b>	0.758
soc-Slashdot	N/A	97312	23404	31863	<b>12318</b>
cit-DBLP	N/A	428.37	134.83	297.03	<b>104.70</b>
cit-HepPh	N/A	539.30	189.57	193.22	<b>170.90</b>
cit-HepTh	N/A	553.16	181.97	193.20	<b>169.06</b>
ca-GrQc	3474.13	40.43	2.72	11.32	<b>6.93</b>
ca-Erdos992	N/A	51.03	3.15	10.04	<b>5.39</b>
ca-HepPh	N/A	539.30	189.57	193.22	<b>158.83</b>
CollegeMsg	N/A	9.240	7.886	1.525	<b>1.448</b>
scc_infect-dublin	N/A	553.2	170.1	<b>107.0</b>	152.6
scc_twitter-copen	N/A	195.89	7.86	<b>1.60</b>	22.13
ego-facebook	N/A	68.0	16.4	<b>10.1</b>	25.1
soc-epinions	N/A	4160.2	1826.5	2424.0	<b>687.0</b>
socfb-Maryland58	N/A	5023.4	5487.7	<b>819.4</b>	903.3
road-chesapeake	0.093	0.003	0.013	0.009	<b>0.001</b>
road-minnesota	N/A	21.30	16.92	32.50	<b>16.07</b>
road-euroroad	N/A	2.864	1.612	1.351	<b>1.037</b>

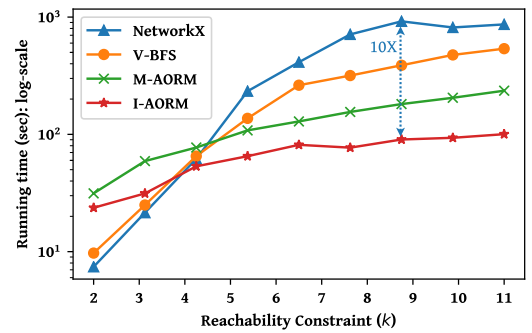
APSP matrix  $\mathbf{D}^{(k)}$  monotonically increase as the number of iteration  $k$  increases. Therefore, the correctness of the approximate solution can be easily measured by comparing the total sum of elements of the matrix with that of the exact solution matrix. We measure the asymptotic correctness  $\mathcal{C}$  of our method by dividing the element sum of the approximate matrix  $\mathbf{D}^{(k)}$  by that of the correct solution  $\mathbf{D}$  as follows:

$$\mathcal{C} = \frac{\sum_{i=1}^n \sum_{j=1}^n D_{ij}^k}{\sum_{i=1}^n \sum_{j=1}^n D_{ij}} \quad (20)$$

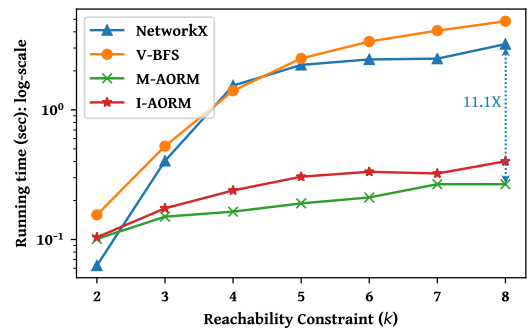
Table 6 shows the measured correctness of approximate APSP computation for a 5,000-node random graph based



**FIGURE 8.** Comparison of APSP computation performance for synthetic networks. We conduct the comparison between our methods (M-AORM, I-AORM) and three baselines (P-SM, NetworkX, V-BFS) and for ten random graphs based on Barabási–Albert growth model ( $n = \{500, 1,000, 2,500, 5,000, 7,500, 10,000, 15,000, 20,000, 25,000, 30,000\}$ ). Our method outperforms up to 11X compared to NetworkX.



(a) ca-HepPh dataset ( $2 \leq k \leq 11$ )



(b) email-Eu-core dataset ( $2 \leq k \leq 8$ )

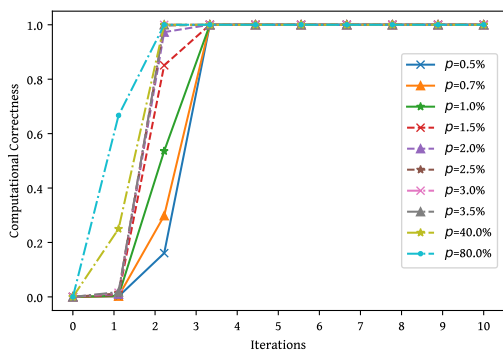
**FIGURE 9.** Performance comparison of incremental APSP computation for real-world networks. We compare the performance of incremental APSP computation between our methods (M-AORM, I-AORM) and baselines (NetworkX, V-BFS). (a) ca-HepPh dataset ( $n = 11, 204$ ): I-AORM outperforms up to 10X compared to NetworkX. The average speedup compared to NetworkX is 5.3X. (b) email-Eu-core dataset ( $n = 986$ ): M-AORM outperforms up to 11.1X compared to NetworkX. The average speedup compared to NetworkX is 8.2X.

on Erdős–Rényi model with different edge probabilities  $p$ . Our method provides *fast convergence property* finding the exact solution within 4 iterations in this experimental setting.

In Figure 10, the line plots represent the correctness of the approximate solutions for graphs with 10,000 nodes and different edge probabilities. As the number of iterations increases, the lines approach to one. The approximate solution more rapidly approaches to the correct result when the

**TABLE 6.** The asymptotic correctness. The correctness  $C$  of approximate computation of APSP for the 5,000-node random graph based on Erdős-Rényi model with different edge probabilities  $p$ . The proposed method provides fast convergence property for asymptotic correctness finding the exact solution within 4 iterations in this experimental setting.

Edge probability	$k=0$	$k=1$	$k=2$	$k=3$	$k=4$
$p = 0.5\%$	0.0000	0.0017	0.0815	0.9243	<b>1.0000</b>
$p = 0.7\%$	0.0000	0.0025	0.1582	0.9989	<b>1.0000</b>
$p = 1.0\%$	0.0000	0.0038	0.3046	<b>1.0000</b>	<b>1.0000</b>
$p = 1.5\%$	0.0000	0.0065	0.5837	<b>1.0000</b>	<b>1.0000</b>
$p = 2.0\%$	0.0000	0.0095	0.8117	<b>1.0000</b>	<b>1.0000</b>
$p = 2.5\%$	0.0000	0.0124	0.9363	<b>1.0000</b>	<b>1.0000</b>
$p = 3.0\%$	0.0000	0.0151	0.9837	<b>1.0000</b>	<b>1.0000</b>
$p = 3.5\%$	0.0000	0.0178	0.9968	<b>1.0000</b>	<b>1.0000</b>
$p = 40.0\%$	0.0000	0.2501	<b>1.0000</b>	<b>1.0000</b>	<b>1.0000</b>
$p = 80.0\%$	0.0000	0.6669	<b>1.0000</b>	<b>1.0000</b>	<b>1.0000</b>

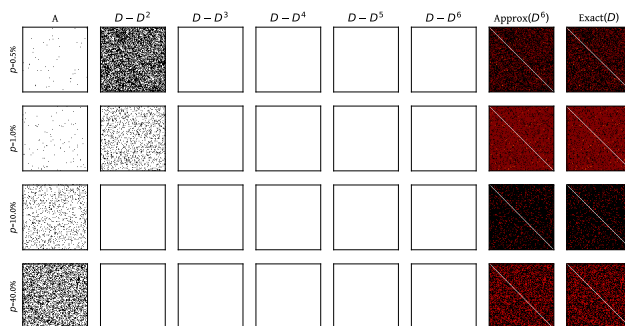


**FIGURE 10.** The asymptotic correctness. We conduct the experiment of incremental APSP computation for the random graph based on Erdős-Rényi model with 10,000 nodes and the edge probabilities  $p$ . The proposed method provides fast convergence property for asymptotic correctness. The line plots represent the correctness of the approximate solutions for graphs with different edge probabilities.

edge probabilities increase. Note that the convergence speed is not proportional to the edge probability.

Besides, we experiment with a graph with 20,000 nodes to measure the correctness of approximate APSP with massive number of nodes. Figure 11 presents the difference between the approximate and correct computations of all-pair shortest paths. The rows illustrate the progressive convergence of the approximate computation with different edge probabilities  $p$ . The first column shows the adjacency matrices. The matrices from the 2nd column to the 6th column visualize the difference between the correct APSP matrix  $D$  and the approximate solutions with  $k$  iterations, i.e.,  $D^{(k)} = \sum_{s=1}^k R^{(k)*}$ . The 7th column is the approximate solution obtained only with six iterations. The last column shows the exact APSP matrix. Note that 0-value entries are colored as white. As shown in the figure, the approximate APSP with incremental computation rapidly converges to the correct solution as the number of edges increases.

*Analysis and Limitations:* Based on the sparsity on real-world networks, the results of our extensive experiments show that AORM significantly improves the performance of reachability computations. The main advantage of our method is its properties of *incremental computation* and *fast convergence*. One limitation of our current implementation



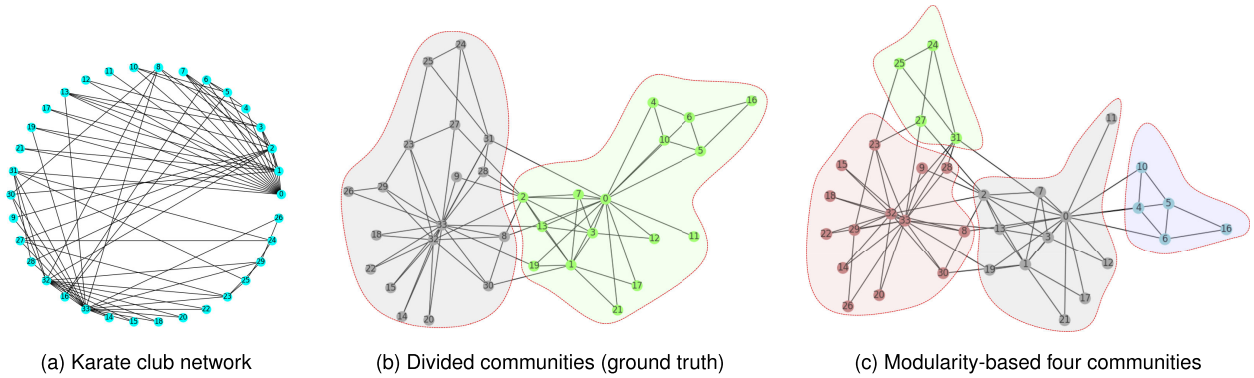
**FIGURE 11.** The difference between the approximate and exact computations of all-pair shortest paths. Each row shows the convergence of the approximate computation with different edge probabilities  $p$ . Note that the differences between the exact results and the approximate estimations is visualized in black rectangles, and 0-value entries are colored as white.

is that we employ data structures based on matrix representation requiring substantial memory spaces compared to sparse matrices. However, it is easy to re-implement our method with *sparse matrix* representation for further optimization in the aspect of memory usage and computational cost.

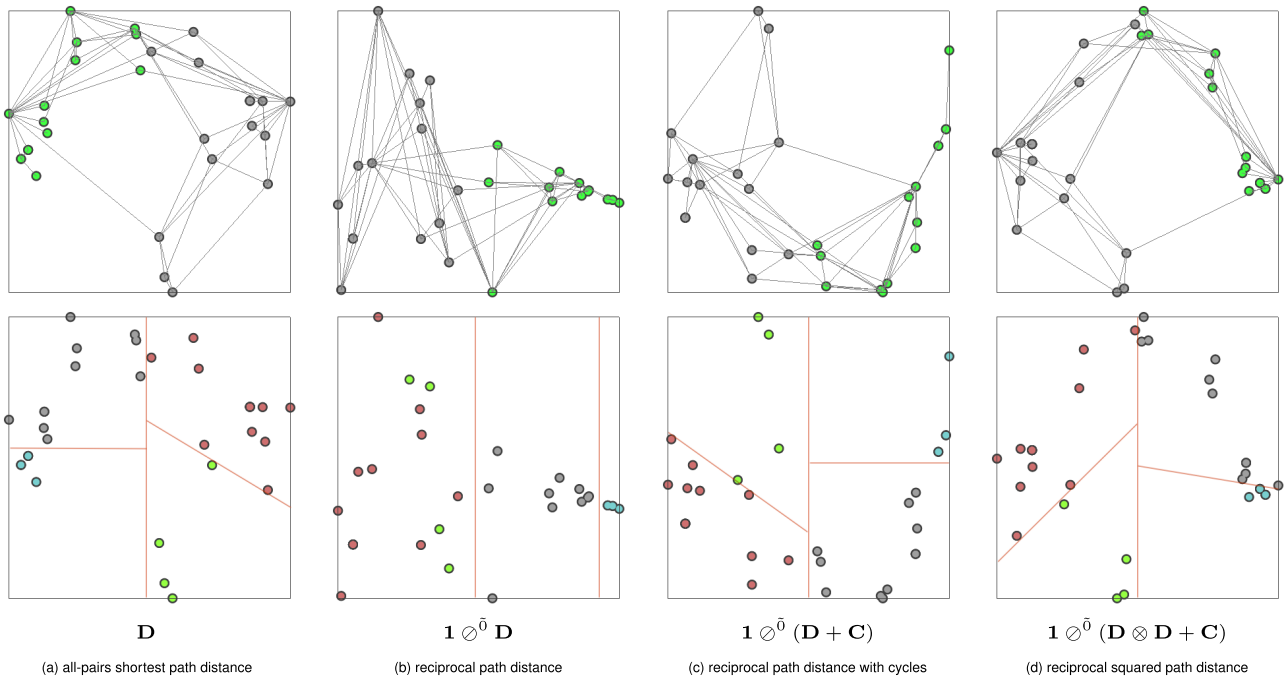
#### D. APPLICATION: NODE EMBEDDING

We perform the experiment for node embedding as one of the AORM-based applications. Figure 12 shows the famous social network first introduced in [51]. The club was divided into two clubs as shown in Figure 12 (b). Figure 12 (c) shows the four communities are detected based on modularity. Modularity measures how many more edges are there within groups compared to the expected number in an equivalent network with random edges [52].

We exploit the AORM framework to compute the various feature matrices in node embedding tasks. Based on these feature matrices, we perform the network embedding for Zachary’s karate network using the matrix factorization shown in Figure 13. The node communities represent different colors to see how the method embedded the nodes in the same cluster in the latent space. In our experiments, the representation space is two-dimensional. The upper row illustrates the embedding results with colors indicating two communities after the actual division of the club (the ground truth). The lower row shows the same embedding results, but the communities are determined by the modularity measure. The embedded nodes in the upper row connect each other based on the original connectivity. Here, we can observe that the nodes in different communities are separated along the vertical axis. In the lower row, we represent only nodes but distinguish the nodes by the auxiliary lines. In particular, the embedding result in the 2nd column cannot be divided by linear line segments. Each column presents the different feature matrix. We exploit the all-pairs shortest path distance  $D$  to represent node features in Figure 13-(a). Figure 13 (b), (c), and (d) show the results when we employ  $\mathbf{1} \odot D$ ,  $\mathbf{1} \odot D^0 (D + C)$ , and  $\mathbf{1} \odot D^0 (D \otimes D + C)$  for features



**FIGURE 12.** Zachary's karate network. (a) the connectivity of the club members, (b) two communities after the actual division, and (c) the four communities detected by modularity analysis.



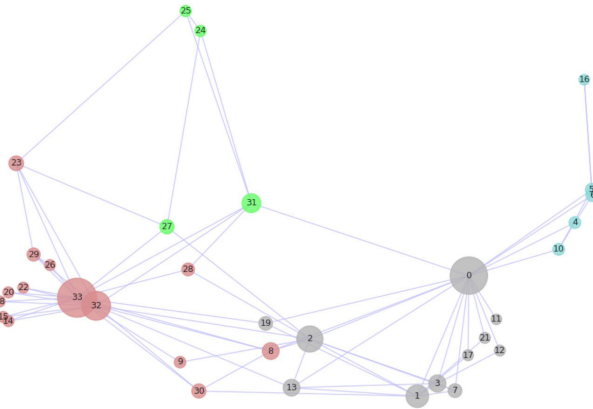
**FIGURE 13.** Node embedding results. We perform the matrix factorization of the various feature matrices for Zachary's karate network. Communities of nodes are distinguished with colors. (upper row: two communities observed, lower row: four communities detected with modularity measure) The node features are represented with (a) all-pairs shortest path distance  $\mathbf{D}$ , (b) reciprocal path distance  $\mathbf{1} \varnothing^{\bar{0}} \mathbf{D}$ , (c) reciprocal path distance with cycles  $\mathbf{1} \varnothing^{\bar{0}} (\mathbf{D} + \mathbf{C})$ , and (d) reciprocal squared path distance  $\mathbf{1} \varnothing^{\bar{0}} (\mathbf{D} \otimes \mathbf{D} + \mathbf{C})$ .

respectively. We normalize each row of matrices before the matrix factorization. We obtain the feature matrices by the accumulation of the  $k$ -order reachability matrices. We can see that the node embedding result successfully preserves the structural proximity between nodes, as shown in Figure 13. We successfully employ the AORM for the node significance analysis of networks in node classification. Figure 14 illustrates the node significance analysis based on the reachability information. Also, we use Zachary's karate club network. We measure the significance of a node based on row of the accumulated sum of the reachability matrices, i.e.,  $\mathbf{D}^{(k)} = \sum_{s=1}^k \mathbf{R}^{(s)*}$ . Each row of the matrix represents the reachability or closeness between the corresponding node and all other nodes. Therefore, we can estimate the node importance

by the frequency of the short paths to other nodes. For the analysis shown in Figure 14, we compute the significance of a node  $v_i$  as  $\sum_{j=1}^n 1/\bar{0} D_{i,j}^3$ , where  $\bar{0}$  is identical to division operation except that it returns 0 when the divider is 0. Hence, we compute the significance by obtaining  $\mathbf{1} \varnothing^{\bar{0}} \mathbf{D}$ .

*Graph Classification:* One of the popular downstream tasks in network embedding is graph classification: to predict the class labels of graphs [22]. Various graph kernels and graph neural network approaches have been studied for graph classification [23], [44]. For graph classification, we expect that AORM can be applied to graph kernels and deep learning on graphs as fundamental reachability query operators for analyzing the network structure. For instance, we can detect similar community structures in graphs by computing





**FIGURE 14. Node significance.** We measure density or significance by several operations on all-pairs path distance matrix,  $\mathbf{1} \oslash^0 \mathbf{D} \otimes \mathbf{D} \otimes \mathbf{D}$ .

arbitrary-order reachability using AORM. Our framework will be helpful to classify similar subgraphs by leveraging both network context and node features in semi-supervised learning.

## VI. RELATED WORK

This section briefly presents recent literature on all pairs shortest path algorithms, reachability computation, and graph embedding for massive graphs.

*All-Pairs Shortest Path Computation:* Providing efficient routes on road networks [53] is one of the popular commercial applications of *shortest path* algorithm, such as Google Maps, Yahoo Maps, or MapQuest. The *all-pairs shortest path problem* (APSP) [54] is to discover the distance and shortest path between every pair of vertices in a directed [41], [55] or undirected graph  $\mathcal{G}$  [56], [57]. A road network or communication network is one of sparse graphs [10]. This kind of graph has *directional* and *sparse* properties [49]. In terms of graph sparsity, an average node degree in the graph is seven or less. Thorup proposed a deterministic linear time and space algorithm for the *single-source shortest paths* (SSSP) problem [58]. His SSSP algorithm works in undirected, positive integer-weighted graphs.

Seidel's algorithm works by recursively shrinking the given graph  $G$  to a graph  $H$  such that  $H$  has 2 useful properties:  $H$ 's shortest pairs are about half as long as  $G$ 's, and the APSP of  $H$  can be exploited to quickly find the APSP of  $G$ . The time complexity of this algorithm is  $\tilde{O}(n^w)$ , where  $w = 2.373$  [8]. However, it only works on *undirected* graphs where each edge has weight one. Moreover, this algorithm cannot compute arbitrary-order reachability since it performs in a *recursive* fashion. The directionality of the edge is one of the essential information for analyzing the asymmetric relationship in graphs. However, the majority of prior researches focused only on undirected graphs. Pettie's algorithm solved the APSP problem to accommodate *directed*, real-weighted graphs, which can perform in  $\mathcal{O}(mn + n^2 \log \log n)$  time [59]. This algorithm adapted the *hierarchy approach* based

on Thorup's algorithm [58]. Note that our algorithm works in unweighted *directed* graphs, even for disconnected graphs.

In the high-performance computing research community [60], many researchers focused on studying the acceleration algorithms for solving the APSP problem based on the *equivalence* between finding the shortest path and solving a linear system [34], [61], [62]. Recent work showed a parallel APSP algorithm via sparse matrix computation and shared-memory parallelism for sparse graphs [63]. This study proposed a method to employ *graph sparsity* in the FLOYD-WARSHALL algorithm [64] to utilize efficient shared-memory parallelism. Similarly, many previous works have focused on studying parallel APSP algorithms based on sparse linear algebra [15], [65]–[67]. Aaron *et al.* introduced a randomized algorithm via random filtered broadcast for distributed all-pairs shortest paths problems [68]. Their algorithm solved the distributed APSP problem on the CONGEST model of distributed network in  $\tilde{O}(n)$  expected time [69].

*Reachability Computation:* Reachability queries refers to examining whether there exists a path between a node  $u$  and node  $v$  in a graph [70]. In particular, processing reachability queries for *massive graphs* is one of the challenging problems in diverse domains, such as graph databases, graph mining, and social network analytics [71]–[73]. The graph summarization approach for label-constrained reachability queries has been tackled in recent work [74]. This study concentrated on regular path queries to recognize graph paths and approximate evaluation of counting label-constrained reachability queries. However, this method has serious problems to apply in application areas requiring exact answers for reachability queries, such as path-finding in road networks. In contrast, we can give the *exact* optimal paths in a directed graph, as well as arbitrary-order reachable paths through AORM.

*Graph Embedding:* Graphs contain *discrete* information such as nodes and edges. However, most modern machine learning (ML) techniques operate on *continuous* inputs. To exploit ML for graph analytics, researchers utilize graph embeddings for the *continuous* representation of graphs [26], [38], [75]. These methods map graph nodes, edges, or sub-graphs to low-dimensional vectors while preserving *positional* properties and *structural* characteristics in the original graph [27], [39], [76], [77].

The major three common approaches for graph embedding include matrix factorization, random walk and graph neural networks. Wang *et al.* introduced one of the matrix factorization approaches, so-call modularized nonnegative matrix factorization (M-NMF). This method preserved both the local structure and the global structure of graphs [78]. DeepWalk is one of the random walk-based methods [38]. This technique simulates random walks, and then encodes them with a neural network model to learn the graphs.

In real-world graphs, random walks will cross the community boundaries very often. Each node in graphs has many *roles* and belongs to many *communities* that the random walk will explore partially. A more comprehensive survey of network embedding can be found in [77], [79]. Similar

to graph embedding, *graph summarization* is to find a compact representation of the input graph; that preserves specific structural, positional, and other properties in original graphs [44], [80], [81]. The representation of graph summarization consists of a summary graph and edge corrections [82].

## VII. CONCLUSION

We proposed a novel incremental algorithmic framework (AORM) to compute arbitrary-order reachability in massive graphs. The proposed method is simple and fast. Moreover, the method outperforms the previous methods in terms of computation time for the all-pairs shortest paths. Another meaningful advantage of the method is that we can compute the reachability and APSP even with *directed* and *disconnected graphs*. This advantage supports practical applications related to collaboration networks, citation networks, and temporal networks.

Extensive experimental studies demonstrate the advantages of our novel method in the aspects of both computational efficiency and approximation controllability. The main advantage of our method is its properties of *incremental computation* and *fast convergence*. The AORM showed better computational performance against the currently known algorithms for the all-pair shortest path problem. Our method outperforms up to 10 times compared to NetworkX for synthetic and real-world networks. Moreover, the computational result of the AORM is asymptotically correct and rapidly converges to the exact solution. Therefore, we can approximate the APSP result with a small negligible error in the early computation stage. Furthermore, we can exploit the AORM for versatile feature extraction framework for node embedding.

*Future Work:* AORM is mainly focusing on the connectivity relation between vertices. It is worth extending our approaches to analyzing community structures in graphs. Further work should address computing structural equivalence and accelerating graph neural networks cooperating with our framework for effective network analytics. Furthermore, learning on temporal reachability graphs is related to incremental learning theory since we should continuously extend a model over time. We hope that future studies will apply our proposed framework to a more general case of incremental learning on temporal reachability graphs. Future work also includes parallelization of AORM for better performance on massive graphs. The algorithm of AORM is embarrassingly parallelizable because of the independent sub-matrices operations.

## REFERENCES

- [1] A. Kapoor, X. Ben, L. Liu, B. Perozzi, M. Barnes, M. Blais, and S. O'Banion, "Examining COVID-19 forecasting using spatio-temporal graph neural networks," 2020, *arXiv:2007.03113*. [Online]. Available: <https://arxiv.org/abs/2007.03113>
- [2] D. Chakrabarti and C. Faloutsos, *Graph Mining: Laws, Tools, and Case Studies* (Synthesis Lectures on Data Mining and Knowledge Discovery). San Rafael, CA, USA: Morgan & Claypool, 2012.
- [3] A. Reka, H. Jeong, and A. Barabasi, "Diameter of the world-wide Web," *Nature*, vol. 401, no. 9, pp. 130–131, 1999.
- [4] D. Chakrabarti and C. Faloutsos, "Graph mining: Laws, generators, and algorithms," *ACM Comput. Surv.*, vol. 38, no. 1, p. 2, 2006.
- [5] P. Boldi and S. Vigna, "Four degrees of separation, really," in *Proc. IEEE/ACM Int. Conf. Adv. Social Netw. Anal. Mining*, Istanbul, Turkey, Aug. 2012, pp. 1222–1227, doi: [10.1109/ASONAM.2012.211](https://doi.org/10.1109/ASONAM.2012.211).
- [6] J. Kunegis, "Handbook of network analysis [KONECT—the Koblenz network collection]," 2014, *arXiv:1402.5500*. [Online]. Available: <https://arxiv.org/abs/1402.5500>
- [7] S.-S. Kim, M. Chung, and Y.-K. Kim, "Urban traffic prediction using congestion diffusion model," in *Proc. IEEE Int. Conf. Consum. Electron.-Asia (ICCE-Asia)*, Nov. 2020, pp. 360–363.
- [8] R. Seidel, "On the all-pairs-shortest-path problem in unweighted undirected graphs," *J. Comput. Syst. Sci.*, vol. 51, no. 3, pp. 400–403, Dec. 1995.
- [9] Z. Zhang, P. Cui, X. Wang, J. Pei, X. Yao, and W. Zhu, "Arbitrary-order proximity preserved network embedding," in *Proc. 24th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, London, U.K., Jul. 2018, pp. 2778–2786.
- [10] S.-S. Kim and Y.-B. Kang, "Congestion avoidance algorithm using extended Kalman filter," in *Proc. Int. Conf. Conver. Inf. Technol. (ICCIT)*, Nov. 2007, pp. 913–918.
- [11] S.-S. Kim, O. Min, and Y.-K. Kim, "Improved spatial modeling using path distance metric for urban traffic prediction," in *Proc. 7th Int. Conf. Big Data Appl. Services (BIGDAS)*, 2019, pp. 521–522.
- [12] T. Funke, T. Guo, A. Lancic, and N. Antulov-Fantulin, "Low-dimensional statistical manifold embedding of directed graphs," in *Proc. 8th Int. Conf. Learn. Represent. (ICLR)*. Addis Ababa, Ethiopia: OpenReview.net, Apr. 2020, pp. 1–18. [Online]. Available: <https://dblp.org/rec/conf/iclr/Funke0LA20.bib>
- [13] S. Zhu, H. Peng, J. Li, S. Wang, and L. He, "Adversarial directed graph embedding," 2020, *arXiv:2008.03667*. [Online]. Available: <http://arxiv.org/abs/2008.03667>
- [14] M. Khosla, J. Leonhardt, W. Nejdl, and A. Anand, "Node representation learning for directed graphs," in *Proc. Mach. Learn. Knowl. Discovery Databases-Eur. Conf. (ECML)* (Lecture Notes in Computer Science), vol. 11906. Würzburg, Germany: Springer, Sep. 2019, pp. 395–411, doi: [10.1007/978-3-030-46150-8\\_24](https://doi.org/10.1007/978-3-030-46150-8_24).
- [15] R. R. Williams, "Faster all-pairs shortest paths via circuit complexity," *SIAM J. Comput.*, vol. 47, no. 5, pp. 1965–1985, Jan. 2018.
- [16] C. Zhou, Y. Liu, X. Liu, Z. Liu, and J. Gao, "Scalable graph embedding for asymmetric proximity," in *Proc. 31st AAAI Conf. Artif. Intell.* San Francisco, CA, USA: AAA Press, Feb. 2017, pp. 2942–2948. [Online]. Available: <http://aaai.org/ocs/index.php/AAAI/AAAI17/paper/view/14696>
- [17] M. Ou, P. Cui, J. Pei, Z. Zhang, and W. Zhu, "Asymmetric transitivity preserving graph embedding," in *Proc. 22nd ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, San Francisco, CA, USA, Aug. 2016, pp. 1105–1114, doi: [10.1145/2939672.2939751](https://doi.org/10.1145/2939672.2939751).
- [18] M. Chen, Q. Yang, and X. Tang, "Directed graph embedding," in *Proc. 20th Int. Joint Conf. Artif. Intell. (IJCAI)*, Hyderabad, India, Jan. 2007, pp. 2707–2712.
- [19] C. Yang, M. Sun, Z. Liu, and C. Tu, "Fast network embedding enhancement via high order proximity approximation," in *Proc. 26th Int. Joint Conf. Artif. Intell.*, Aug. 2017, pp. 1–7.
- [20] J. Kim, S. Lim, J. Lee, and B. S. Lee, "LinkBlackHole\*\*: Robust overlapping community detection using link embedding," *IEEE Trans. Knowl. Data Eng.*, vol. 31, no. 11, pp. 2138–2150, Nov. 2019.
- [21] Y. Han and T. Takaoka, "An  $O(n^3 \log n \log^2 n)$  time algorithm for all pairs shortest paths," *J. Discrete Algorithms*, vols. 38–41, pp. 9–19, May/Nov. 2016, doi: [10.1016/j.jda.2016.09.001](https://doi.org/10.1016/j.jda.2016.09.001).
- [22] K. Riesen and H. Bunke, *Graph Classification and Clustering Based on Vector Space Embedding* (Series in Machine Perception and Artificial Intelligence), vol. 77. Singapore: World Scientific, 2010.
- [23] W. L. Hamilton, "Graph representation learning," *Synth. Lectures Artif. Intell. Mach. Learn.*, vol. 14, no. 3, pp. 1–159, 2020.
- [24] M. Aggarwal and M. N. Murty, *Machine Learning in Social Networks—Embedding Nodes, Edges, Communities, and Graphs*. Singapore: Springer, 2021. [Online]. Available: <https://dblp.org/rec/books/sp/AggarwalM21.bib>, doi: [10.1007/978-981-33-4022-0](https://doi.org/10.1007/978-981-33-4022-0).
- [25] R. A. Rossi, N. K. Ahmed, and E. Koh, "Higher-order network representation learning," in *Proc. Companion The Web Conf. Web Conf. (WWW)*, Lyon, France, 2018, pp. 3–4.
- [26] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei, "LINE: Large-scale information network embedding," in *Proc. 24th Int. Conf. World Wide Web*, Florence, Italy, May 2015, pp. 1067–1077, doi: [10.1145/2736277.2741093](https://doi.org/10.1145/2736277.2741093).

- [27] S. Cao, W. Lu, and Q. Xu, "GraRep: Learning graph representations with global structural information," in *Proc. 24th ACM Int. Conf. Inf. Knowl. Manage. (CIKM)*, Melbourne, VIC, Australia, Oct. 2015, pp. 891–900.
- [28] J. Leskovec and A. Krevl. (Jun. 2014). *SNAP Datasets: Stanford Large Network Dataset Collection*. [Online]. Available: <http://snap.stanford.edu/data>
- [29] R. A. Rossi and N. K. Ahmed, "The network data repository with interactive graph analytics and visualization," in *Proc. 29th AAAI Conf. Artif. Intell.* Austin, TX, USA: AAAI Press, Jan. 2015, pp. 4292–4293.
- [30] J. Kleinberg and É. Tardos, *Algorithm Design*. Reading, MA, USA: Addison-Wesley, 2006.
- [31] A. García Floriano, "Classification model supervised using the heaviside function," *Computación y Sistemas*, vol. 23, no. 4, Dec. 2019.
- [32] S. Cao, W. Lu, and Q. Xu, "Grarep: Learning graph representations with global structural information," in *Proc. CIKM*, 2015, pp. 891–900.
- [33] D. Zhang, J. Yin, X. Zhu, and C. Zhang, "Network representation learning: A survey," 2017, *arXiv:1801.05852*. [Online]. Available: <http://arxiv.org/abs/1801.05852>
- [34] M. A. Razzaque, C. S. Hong, M. Abdullah-Al-Wadud, and O. Chae, "A fast algorithm to calculate powers of a Boolean matrix for diameter computation of random graphs," in *Proc. Algorithms Comput., 2nd Int. Workshop (WALCOM)* (Lecture Notes in Computer Science), vol. 4921. Dhaka, Bangladesh: Springer, Feb. 2008, pp. 58–69.
- [35] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Proc. Adv. Neural Inf. Process. Syst.: 27th Annu. Conf. Neural Inf. Process. Syst.*, Lake Tahoe, NV, USA, Dec. 2013, pp. 3111–3119.
- [36] M. McCool, A. D. Robison, and J. Reinders, *Structured Parallel Programming: Patterns for Efficient Computation*. San Mateo, CA, USA: Morgan Kaufmann, 2012. [Online]. Available: <http://parallelbook.com/>
- [37] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, and R. Kern, "Array programming with numpy," *Nature*, vol. 585, no. 7825, pp. 357–362, 2020.
- [38] B. Perozzi, R. Al-Rfou, and S. Skiena, "DeepWalk: Online learning of social representations," in *Proc. 20th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, New York, NY, USA, Aug. 2014, pp. 701–710, doi: [10.1145/26223330.2622732](https://doi.org/10.1145/26223330.2622732).
- [39] A. Grover and J. Leskovec, "node2vec: Scalable feature learning for networks," in *Proc. 22nd ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, San Francisco, CA, USA, Aug. 2016, pp. 855–864.
- [40] A. Shoshan and U. Zwick, "All pairs shortest paths in undirected graphs with integer weights," in *Proc. 40th Annu. Symp. Found. Comput. Sci.*, New York, NY, USA, 1999, pp. 605–615.
- [41] U. Zwick, "All pairs shortest paths in weighted directed graphs—exact and almost exact algorithms," in *Proc. 39th Annu. Symp. Found. Comput. Sci. (FOCS)*, Palo Alto, CA, USA, Nov. 1998, pp. 310–319, doi: [10.1109/SFCS.1998.743464](https://doi.org/10.1109/SFCS.1998.743464).
- [42] Y. Peng, Y. Zhang, X. Lin, W. Zhang, L. Qin, and J. Zhou, "Hop-constrained ST simple path enumeration: Towards bridging theory and practice," *Proc. VLDB Endow.*, vol. 13, no. 4, pp. 463–476, 2019.
- [43] Z. Lai, Y. Peng, S. Yang, X. Lin, and W. Zhang, "PEFP: Efficient K-hop Constrained s-t Simple Path Enumeration on FPGA," 2020, *arXiv:2012.11128*. [Online]. Available: <https://arxiv.org/abs/2012.11128>
- [44] J. You, R. Ying, and J. Leskovec, "Position-aware Graph Neural Networks," in *Proc. 36th Int. Conf. Mach. Learn., (ICML)* (Proceedings of Machine Learning Research), vol. 97. Long Beach, CA, USA: PMLR, Jun. 2019, pp. 7134–7143.
- [45] V. Chvátal and E. Szemerédi, "Short cycles in directed graphs," *J. Combinat. Theory, Ser. B*, vol. 35, no. 3, pp. 323–327, Dec. 1983.
- [46] J. Bang-Jensen and G. Z. Gutin, *Digraphs—Theory, Algorithms and Applications*, (Springer Monographs in Mathematics), 2nd ed. Springer, 2009.
- [47] S. Chechik, Y. P. Liu, O. Rotem, and A. Sidford, "Constant girth approximation for directed graphs in subquadratic time," in *Proc. 52nd Annu. ACM SIGACT Symp. Theory Comput.*, Chicago, IL, USA, Jun. 2020, pp. 1010–1023.
- [48] *NetworkX: Network Analysis in Python*. Accessed: Mar. 15, 2021. [Online]. Available: [https://networkx.org/documentation/stable/reference/algorithms/generat\\_ed/networkx.algorithms.shortest\\_paths.unweighted.all\\_pairs\\_shortest\\_path.html#networkx.algorithms.shortest\\_paths.unweighted.all\\_pairs\\_shortest\\_path](https://networkx.org/documentation/stable/reference/algorithms/generat_ed/networkx.algorithms.shortest_paths.unweighted.all_pairs_shortest_path.html#networkx.algorithms.shortest_paths.unweighted.all_pairs_shortest_path)
- [49] S. Pettie, "All pairs shortest paths in sparse graphs," in *Encyclopedia of Algorithms*. New York, NY, USA: Springer, 2016, pp. 52–55. [Online]. Available: <https://dblp.org/rec/reference/algo/Pettie16.bib>, doi: [10.1007/978-1-4939-2864-4\\_11](https://doi.org/10.1007/978-1-4939-2864-4_11).
- [50] P. Holme and B. J. Kim, "Growing scale-free networks with tunable clustering," *Phys. Rev. E, Stat. Phys. Plasmas Fluids Relat. Interdiscip. Top.*, vol. 65, no. 2, Jan. 2002, Art. no. 026107.
- [51] W. W. Zachary, "An information flow model for conflict and fission in small groups," *J. Anthropological Res.*, vol. 33, no. 4, pp. 452–473, Dec. 1977.
- [52] M. E. J. Newman, "Modularity and community structure in networks," *Proc. Nat. Acad. Sci. USA*, vol. 103, no. 23, pp. 8577–8582, Jun. 2006.
- [53] Y.-B. Kang and S.-S. Kim, "A fastest route planning for LBS based on traffic prediction," in *Proc. 9th WSEAS Int. Conf. Syst.* Stevens Point, WI, USA: World Scientific and Engineering Academy and Society (WSEAS), 2005.
- [54] A. Madkour, W. G. Aref, F. U. Rehman, M. A. Rahman, and S. M. Basalamah, "A survey of shortest-path algorithms," 2017, *arXiv:1705.02044*. [Online]. Available: <https://arxiv.org/abs/1705.02044>
- [55] S. Kang, S.-S. Kim, J. Won, and Y.-M. Kang, "GPU-based parallel genetic approach to large-scale travelling salesman problem," *J. Supercomput.*, vol. 72, no. 11, pp. 4399–4414, Nov. 2016.
- [56] T. M. Chan, "All-pairs shortest paths for unweighted undirected graphs in  $o(mn)$  time," *ACM Trans. Algorithms*, vol. 8, no. 4, p. 34, 2012.
- [57] T. Takaoka, "All pairs shortest paths via matrix multiplication," in *Encyclopedia of Algorithms*. New York, NY, USA: Springer, 2016, pp. 55–59, doi: [10.1007/978-1-4939-2864-4\\_12](https://doi.org/10.1007/978-1-4939-2864-4_12).
- [58] M. Thorup, "Undirected single-source shortest paths with positive integer weights in linear time," *J. ACM*, vol. 46, no. 3, pp. 362–394, May 1999, doi: [10.1145/316542.316548](https://doi.org/10.1145/316542.316548).
- [59] S. Pettie, "A new approach to all-pairs shortest paths on real-weighted graphs," *Theor. Comput. Sci.*, vol. 312, no. 1, pp. 47–74, Jan. 2004.
- [60] V. Kumar and V. Singh, "Scalability of parallel algorithms for the all-pairs shortest-path problem," *J. Parallel Distrib. Comput.*, vol. 13, no. 2, pp. 124–138, Oct. 1991.
- [61] R. C. Backhouse and B. A. Carré, "Regular algebra applied to path-finding problems," *IMA J. Appl. Math.*, vol. 15, no. 2, pp. 161–186, 1975.
- [62] V. V. Williams, "Multiplying matrices faster than coppersmith-winograd," in *Proc. 44th Symp. Theory Comput. - STOC*, New York, NY, USA, 2012, pp. 887–898.
- [63] P. Sao, R. Kannan, P. Gera, and R. Vuduc, "A supernodal all-pairs shortest path algorithm," in *Proc. 25th ACM SIGPLAN Symp. Princ. Pract. Parallel Program.*, San Diego, CA, USA, Feb. 2020, pp. 250–261.
- [64] R. W. Floyd, "Algorithm 97: Shortest path," *Commun. ACM*, vol. 5, no. 6, p. 345, Jun. 1962.
- [65] D. B. Johnson, "Efficient algorithms for shortest paths in sparse networks," *J. ACM*, vol. 24, no. 1, pp. 1–13, Jan. 1977.
- [66] J. Alman and V. V. Williams, "Limits on all known (and some Unknown) approaches to matrix multiplication," in *Proc. IEEE 59th Annu. Symp. Found. Comput. Sci. (FOCS)*, Beijing, China, Oct. 2018, p. 10.
- [67] U. Agarwal and V. Ramachandran, "Faster deterministic all pairs shortest paths in congest model," in *Proc. 32nd ACM Symp. Parallelism Algorithms Archit.*, Jul. 2020, pp. 11–21.
- [68] A. Bernstein, M. P. Gutenberg, and C. Wulff-Nilsen, "Near-optimal decremental sssp in dense weighted digraphs," 2020, *arXiv:2004.04496*. [Online]. Available: <https://arxiv.org/abs/2004.04496>
- [69] A. Bernstein and D. Nanongkai, "Distributed exact weighted all-pairs shortest paths in near-linear time," in *Proc. 51st Annu. ACM SIGACT Symp. Theory Comput.*, Phoenix, AZ, USA, Jun. 2019, pp. 334–342.
- [70] J. X. Yu and J. Cheng, "Graph reachability queries: A survey," in *Managing and Mining Graph Data* (Advances in Database Systems), vol. 40. Springer, 2010, pp. 181–215.
- [71] R. Angles, M. Arenas, P. Barceló, A. Hogan, J. L. Reutter, and D. Vrgoc, "Foundations of modern query languages for graph databases," *ACM Comput. Surv.*, vol. 50, no. 5, p. 68, 2017.
- [72] R. Angles, M. Arenas, P. Barceló, P. A. Boncz, G. H. L. Fletcher, C. Gutiérrez, T. Lindaaker, M. Paradies, S. Plantikow, J. F. Sequeda, O. van Rest, and H. Voigt, "G-CORE: A core for future graph query languages," in *Proc. Int. Conf. Manage. Data (SIGMOD)*, Houston, TX, USA, Jun. 2018, pp. 1421–1432.
- [73] A. Bonifati, G. Fletcher, H. Voigt, and N. Yakovets, "Querying graphs," in *Synthesis Lectures on Data Management*. San Rafael, CA, USA: Morgan & Claypool, 2018.
- [74] A. P. Iyer, A. Panda, S. Venkataraman, M. Chowdhury, A. Akella, S. Shenker, and I. Stoica, "Bridging the GAP: Towards approximate graph analytics," in *Proc. 1st ACM SIGMOD Joint Int. Workshop Graph Data Manage. Experiences Syst. (GRADES)*, 2018, p. 10.

- [75] A. Epasto and B. Perozzi, "Is a single embedding enough? Learning node representations that capture multiple social contexts," in *Proc. World Wide Web Conf. (WWW)*, San Francisco, CA, USA, 2019, pp. 394–404.
- [76] D. Wang, P. Cui, and W. Zhu, "Structural deep network embedding," in *Proc. 22nd ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, San Francisco, CA, USA, Aug. 2016, pp. 1225–1234.
- [77] P. Cui, X. Wang, J. Pei, and W. Zhu, "A survey on network embedding," *IEEE Trans. Knowl. Data Eng.*, vol. 31, no. 5, pp. 833–852, May 2019.
- [78] X. Wang, P. Cui, J. Wang, J. Pei, W. Zhu, and S. Yang, "Community preserving network embedding," in *Proc. 31st AAAI Conf. Artif. Intell.* San Francisco, CA, USA: AAAI Press, Feb. 2017, pp. 203–209.
- [79] H. Chen, B. Perozzi, R. Al-Rfou, and S. Skiena, "A tutorial on network embeddings," 2018, *arXiv:1808.02590*. [Online]. Available: <https://arxiv.org/abs/1808.02590>
- [80] Y. Liu, T. Safavi, A. Dighe, and D. Koutra, "Graph summarization methods and applications: A survey," *ACM Comput. Surv.*, vol. 51, no. 3, p. 62, 2018.
- [81] D. Jin, R. A. Rossi, E. Koh, S. Kim, A. Rao, and D. Koutra, "Latent network summarization: Bridging network embedding and summarization," in *Proc. 25th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining (KDD)*, Anchorage, AK, USA, Aug. 2019, pp. 987–997.
- [82] S. Navlakha, R. Rastogi, and N. Shrivastava, "Graph summarization with bounded error," in *Proc. ACM SIGMOD Int. Conf. Manage. Data (SIGMOD)*, J. T. Wang, Ed. Vancouver, BC, Canada: ACM Press, Jun. 2008, pp. 419–432.



**SUNG-SOO KIM** received the M.S. degree in computer science from Pusan National University, in 1999. He is currently pursuing the Ph.D. degree with Chungnam National University. He worked for the Korea Advanced Institute of Science and Technology (KAIST). He researched reuse-based software engineering for large-scale information systems. In 2000, he joined Electronics and Telecommunications Research Institute (ETRI), where he is currently a Principal Researcher with

the Artificial Intelligence Research Laboratory. His research interests include machine learning, graph mining, graph signal processing, parallel/distributed computing, and graph neural networks.



**YOUNG-KUK KIM** received the M.S. degree in computer science and statistics from Seoul National University, South Korea, in 1987, and the Ph.D. degree in computer science from the University of Virginia, Charlottesville, VA, USA, in 1995. He is currently a Professor of computer engineering with Chungnam National University, South Korea. His research interests include mobile and ubiquitous information systems, real-time, mobile, and main-memory databases.



**YOUNG-MIN KANG** (Member, IEEE) was born in Busan, South Korea. He received the B.S., M.S., and Ph.D. degrees in computer science from Pusan National University, in 1996, 1999, and 2003, respectively. From January 2002 to July 2002, he was a Temporary Researcher with the MIRALab, University of Geneva, Switzerland, for virtual clothing project. In 2003, he joined Electronics and Telecommunications Research Institute (ETRI), for development of next generation game engine. In 2005, he moved to Tongmyong University. He was the Director of the Industry-Academy Cooperation Foundation, the Head of Information and Computing Center, and the Director of the Advancement of College Education (ACE) project, Tongmyong University. He has been a Full Professor with the Game Engineering Department, since 2013. He is currently a Dispatched Researcher with the AI Research Laboratory, ETRI, from September 2020 to August 2021.

• • •