Sparsity-Aware Reachability Computation for Massive Graphs

Sung-Soo Kim
Artificial Intelligence Research Laboratory
ETRI
Daejeon, South Korea
sungsoo@etri.re.kr

Young-Min Kang
Dept. of Game Engineering
Tongmyong University
Busan, South Korea
ymkang@tu.ac.kr

Young-Kuk Kim†
Dept. of Computer Science & Engineering
Chungnam National University
Daejeon, South Korea
ykim@cnu.ac.kr

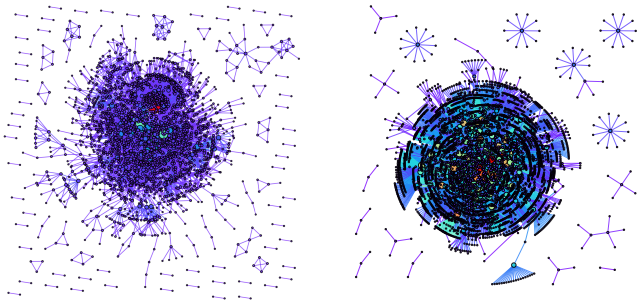
***Abstract*—We propose a novel sparsity-aware reachability computation framework so-called S-AORM, which provides fast performance for massive graphs via incremental fashion using sparse matrices. S-AORM is straightforward to comprehend and outperforms existing methods in terms of computational performance. Five synthetic networks generated from Barabási–Albert model and five real-world networks are used in comprehensive experiments. In terms of all-pairs shortest paths computation performance on the citation network, which is a directed and disconnected network, the proposed approach surpasses SNAP by up to 12.1 times and NetworkX by up to 80.2 times. The overall experimental results demonstrate that our approach provides significant performance improvement in the graph reachability computation.**

***Keywords*-Graph reachability, all-pairs shortest paths approximation, higher-order proximity, graph embedding.**

I. Introduction

With the persistence of COVID-19, reachability and diffusion analysis are more significant than ever for efficient contact tracing improves in the disease's control.
In particular, reachability computation on massive graphs is essential for various graph analytics utilization: not only epidemic networks but versatile applications such as social networks, recommender systems, and traffic prediction systems [1], [2]. For real-world networks, modern data analysts utilize disconnected, directed graphs with time-varying to obtain insights via diffusion analysis on networks.
Most researchers employ *adjacency matrices* as representations of these networks to perform fundamental graph analytics operations like reachability queries, all-pairs shortest paths, and so on. These adjacency matrices of large networks are often *sparse* in most practical applications. Furthermore, instead of using weighted edges to express relationships among vertices in a network, most applications utilize unweighted edges along with node attributes.

Motivations. Even though graphs are gigantic, previous graph mining scientists discovered that existing networks had remarkably *small diameters* [1]. Due to their low reciprocity, most citation networks are directed and *disconnected* graphs. Figure 1 illustrates two real-world networks in the type of disconnected graphs. Unfortunately, most prior researches have focused on handling unweighted and *undirected* graphs [3], [4]. As a result, these techniques are unable to reflect edge directions, which are required in a variety of applications such as tracing viral dispersion, fake news spread, and traffic congestion monitoring [5].
To overcome the limitation, we propose a novel sparsity-aware reachability computation framework, called S-AORM, for directed and unweighted graphs.



(a) Cora (b) ca-Erdos992

Figure 1. Two examples of disconnected graphs. (a) Cora dataset is the citation network. (b) ca-Erdos992 dataset is the collaboration network.

Key Contributions. The proposed approach contributes the following three major key benefits:

- *Incremental reachability*: S-AORM provides fast performance for computing reachability and all-pairs shortest paths on massive graphs (APSP) in an incremental fashion.
- *Space and computational efficiency*: Our approach supports space and computational efficiency by using a sparse matrix for processing massive networks.
- *Directed and disconnected graphs*: The proposed algorithm handles directed and disconnected networks, which are prevalent in real-world networks such as citation networks and road networks.

II. Sparsity-Aware Graph Reachability

In this section, we describe the proposed sparsity-aware graph reachability framework for large-scale networks. We explicitly define the problem in terms of arbitrary-order reachability queries based on the notations [1].

† Corresponding author: Young-Kuk Kim (ykim@cnu.ac.kr)

2375-9356/22/$31.00 ©2022 IEEE
DOI 10.1109/BigComp54360.2022.00038

Notations. Table I lists the symbols and notations used in the paper. Note that the superscript with parenthesis such as $x^{(k)}$ denotes the instance x at the k-th iteration.

Table I
NOTATIONS AND SYMBOLS.

Notations	Description
$\mathcal{G} = (V, E)$	a graph consisting a vertex-set V and a edge-set E
$\mathbf{A}$	an adjacency matrix for $\mathcal{G}$
$\mathbf{A}_s$	sparse matrix for $\mathbf{A}$
$\mathcal{H}(\cdot)$	heaviside function
$\mathbf{R}_s^{(d)}$	sparse matrix-based reachability matrix
$\mathbf{R}_s^{(k)}$	sparse matrix-based k-order reachability matrix
$\mathbf{R}_s^{(k)^\star}$	sparse matrix-based k-order optimal reachability matrix
$\mathcal{N}_\mathbf{A}$	non-zero elements in $\mathbf{A}$
$\mathbf{I}$	identity matrix
$F_i^{(k)}$	footprint set vertex-set that appeared in vertex v_i within k hops
$\mathbf{F}_s^{(k)}$	sparse matrix-based footprint matrix boolean matrix of which entry $F_{i,j}^{(k)}$ is 1 iff $v_j \in F_i^{(k)}$
$\mathbf{D}_s$	sparse matrix-based all-pairs path distance matrix

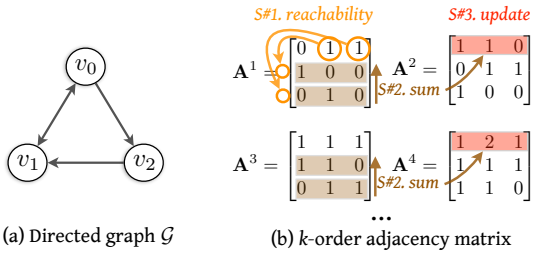
PROBLEM. **Arbitrary-Order Reachability Queries**
Let $\mathcal{G} = (V, E)$ be an n-node, m-edge directed or undirected graph. $u \rightsquigarrow v$ denotes k-order reachability queries, where u and v are two vertices in $\mathcal{G}$. In this case, if and only if a path via k-hops appears in the graph $\mathcal{G}$, $u \rightsquigarrow v$ yields true. The goal of arbitrary-order reachability queries is to compute the reachability of each vertex in $\mathcal{G}$ as a k-order reachability matrix using k-hops [1].

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

For undirected, directed, and disconnected graphs, the proposed approach affords arbitrary-order reachability computation. To calculate arbitrary-order reachability, we should compute the adjacency matrix's k-th power. However, one of the major difficulties is to enhance the computation of the powers of the adjacency matrix for massive graphs.
To overcome this limitation, the incremental arbitrary-reachability framework called I-AORM has been proposed to utilize the summation of non-zero elements in the hierarchical reachability matrix based on neighborhood information [1]. The Heaviside function is useful for computing reachability-related matrices efficiently. $\mathcal{H}(\cdot)$ is a non-continuous Heaviside function with a value of zero for a negative input and one for a positive input.

A. Sparsity-Aware Arbitrary-Order Reachability

Figure 2 illustrates the core concept of computation procedure in I-AORM. Note that this procedure presents the part of matrix computation procedure for an arbitrary reachability. I-AORM constructs the i-th row of $\mathbf{A}^{(k)}$ in accordance with the neighbor information $\mathcal{N}_i$ by summing all non-zero elements $\mathbf{A}_j^{(k-1)}$ such that $j \in \mathcal{N}_i$ [1]. The footprint matrix keeps the set of vertices that appeared in vertex v_i within



(a) Directed graph $\mathcal{G}$ (b) k-order adjacency matrix

Figure 2. I-AORM algorithm [1] for computing k-order adjacency matrix before performing reachability pruning.

k hops. Then, the method exploits the footprint matrix $\mathbf{F}$ to perform pruning for cycles, trails, and paths. Although SIMD-based arbitrary-order reachability computation framework (M-AORM [1]) speeds up reachability computation on small graphs, it still takes a long time to handle large sparse graphs in practice.

Algorithm 1 SPARSITY-AWARE AORM (S-AORM)

Input: $\mathbf{A} \in \mathbb{R}^{n \times n}$: Adjacency matrix,
k_c: Constraint
Output: $\mathbf{R}^{(k)^\star}$: Reachability matrix sequence,
where $\mathbf{R}^{(k)^\star} \in \mathbb{R}^{n \times n}$ for $k \in \{2, \ldots, d\}$
• **Phase #1.** Sparse matrix conversion
1: $k \leftarrow 1$
2: $\mathbf{A}_s \leftarrow$ convertSparseMatrix($\mathbf{A}$)
3: $\mathbf{R}_s^{(k)^\star} \leftarrow \mathbf{A}_s$ ▷ assign initial reachability sparse matrix
4: $\mathbf{F}_s \leftarrow \mathbf{I} + \mathbf{A}_s$ ▷ footprint sparse matrix
• **Phase #2.** Incremental $\mathbf{R}_s^{(k)^\star}$ via sparse matrices
5: **while** $\mathbf{R}_s^{(k)^\star}$ is not converged **or** $k < k_c$ **do**
6: $k \leftarrow k + 1$
▷ $@_c$: sparse matrix-based matrix multiplication
7: $\mathbf{R}_s^{(k)^\star} \leftarrow \mathcal{H}(\mathcal{H}(\mathbf{A}_s @_c \mathbf{R}_s^{(k-1)^\star}) - \mathbf{F}_s)$
8: $\mathbf{F}_s \leftarrow \mathbf{R}_s^{(k)^\star} + \mathbf{F}_s$
9: **yield** $\tilde{\mathbf{R}}_s^k$ ▷ $\tilde{\mathbf{R}}_s^k$ refers to an intermediate result
10: **end while**

To overcome the limitation of M-AORM, we introduce *sparse matrix*-based approach called S-AORM for computing graph reachability. The proposed algorithm of sparse matrix-based k-order reachability computation is shown in **Algorithm 1**. First, to provide space and computational efficiency, it converts from a dense matrix to a sparse matrix in the initialization step. The incremental procedure (Line 5–9) is repeated until the matrix is converged or the constraint is met. Note that the sparse matrix can be used as the compressed storage by columns (csr) or the compressed storage by rows (csc) format.

B. All-Pairs Shortest Paths Algorithm

The algorithm for computing all-pairs shortest paths is shown in **Algorithm 2**. To provide efficiencies, sparse

matrix conversion for distance matrix is performed similar to **Algorithm 1**. The algorithm returns the distance matrix $\mathbf{D}$ for APSP according to the reachability constraints k_c.

Algorithm 2 S-AORM-BASED APSP COMPUTATION

Input: $\mathbf{A} \in \mathbb{R}^{n \times n}$: Adjacency matrix
k_c: Constraint [default: $k_c = \infty$]
Output: $\mathbf{D}_s \in \mathbb{R}^{n \times n}$: Sparse distance matrix

```
k ← k_c                                   ▷ set the constraint
D_s ← convertSparseMatrix(A)
S-AORMITERATOR ← S-AORM(A, k)
    ▷ Incremental computation using S-AORM
for each R_s^(k)* from S-AORMITERATOR do
    D_s ← D_s + kR_s^(k)*
end for
return D_s
```

The complexity of the best Seidel's APSP for *undirected* graphs is $\mathcal{O}(n^w \log n)$, where w=2.373 [3]. In contrast, the complexity for **Algorithm 2** for *directed* and *disconnected* graphs is $\mathcal{O}(nmd)$, where n is the number of nodes, e denotes the number of edges, and d is a diameter of a graph.

III. EXPERIMENTS

In our experiments, we conduct comprehensive experiments on synthetic networks and real-world networks.

A. Experiments Setup

All of the experiments are conducted on a single computer with an Intel Xeon E5 3GHz processor and 64 GB of RAM. The proposed framework is implemented in Python. Additionally, we use the NetworkX package [6] to generate synthetic networks based on the Barabási–Albert model with five different node sizes and evaluate APSP performance compared with five baselines.

Datasets. The data statistics of real-world networks are described in Table II. We exploit these datasets downloaded from the SNAP site [7] and the network repository [8]. These networks include various types of networks, such as directed networks (DT), citation networks (CIT), collaboration networks (CA), temporal reachability networks (TRN), and social networks (SOC).

Table II
DATA STATISTICS FOR VARIOUS TYPES OF REAL-WORLD NETWORKS.

Type	Dataset	# Nodes	# Edges	Avg. degree
DN	p2p-Gnutella04	10,876	39,994	7.35
CIT	cit-DBLP	12,591	49,635	7.88
CA	ca-Erdos992	5,094	7,515	2.95
TRN	CollegeMsg	1,899	12,838	14.57
SOC	ego-facebook	4,039	88,234	43.69

Baselines. The APSP computation performance of the S-AORM is compared to the performance of the following five baseline methods.

- **SNAP** [7]: The Stanford Network Analysis Platform (SNAP) is a network analysis and graph mining toolkit, which core is implemented in C++. We implement the APSP computation module using the SNAP for python.
- **NetworkX** (NX) [6]: This method is all-pairs shortest paths computation program for networks which is implemented in NetworkX API.
- **V-BFS** (VB) [9]: This method is the vectorized BFS-based method to use SIMD for improving the APSP computation.
- **M-AORM** (M-A) [1]: This technique is incremental reachability computation similar to the vectorized matrix multiplication method.
- **I-AORM** (I-A) [1]: This method is the APSP algorithm via incremental AORM based on the first-order reachability.

To compute the APSP for given networks, we perform the procedure in **Algorithm 2**.

- **S-AORM** (S-A): The proposed method is based on the APSP algorithm using the sparse matrix-based AORM.

B. Performance Comparison for APSP

On synthetic graphs and a various types of real-world networks, we perform the computational performance comparison against APSP to baseline techniques in this section. In experiments for synthetic graphs, the APSP computation is performed by four different methods including the S-AORM. We generate random graphs based on the Barabási–Albert model with different node sizes ($|V| = 1000, 2500, 5000, 7500, 10000$). Our approach beats SNAP by 3.7 times and NetworkX by 13 times, as shown the results in Figure 3.

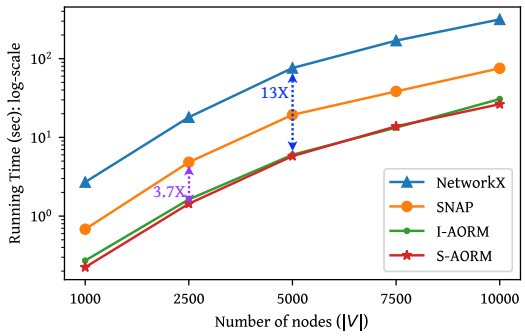


Figure 3. All-pairs shortest paths (APSP) performance results for five synthetic networks generated from the the Barabási–Albert model.

The comparison results for all-pairs shortest paths performance for real-world networks are shown in Table III. The best performance is indicated by the use of the bold font. The results show that the proposed method also significantly outperforms five baseline methods. Specifically, the S-AORM outperforms up to 12.1 times compared to SNAP and 80.2 times compared to NetworkX as shown in Figure 4.

Table III
ALL-PAIRS SHORTEST PATHS (APSP) PERFORMANCE RESULTS FOR REAL-WORLD NETWORKS. (UNIT: SECONDS)

Dataset	SNAP	NX	V-B	M-A	I-A	S-A
p2p-Gnutella04	60.1	341.9	666.5	529.0	146.3	**34.5**
cit-DBLP	60.5	401.4	134.8	297.3	104.3	**5.0**
ca-Erdos992	6.1	51.0	3.2	10.0	5.4	**0.28**
CollegeMsg	2.6	9.240	22.6	1.525	1.448	**0.7**
ego-facebook	29.2	68.0	16.4	19.6	18.7	**1.3**

As such, it can be seen that the proposed method provides better performance in a citation graph, which is a *directed* and *disconnected* graph among real-world graphs.

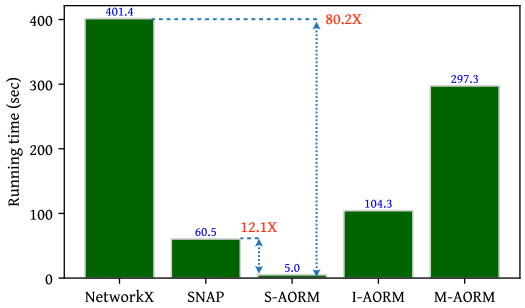


Figure 4. All-pairs shortest paths (APSP) performance results for the citation network (cit-DBLP dataset).

On the temporal reachability network, we perform experiments for incremental APSP computation. For these tests, we gradually set reachability constraints based on the network diameter. As a result, the proposed method is up to 19.2X faster than the incremental computation of NetworkX as shown in Figure 5. Note that SNAP does not provide a function for APSP computation via incremental fashion.

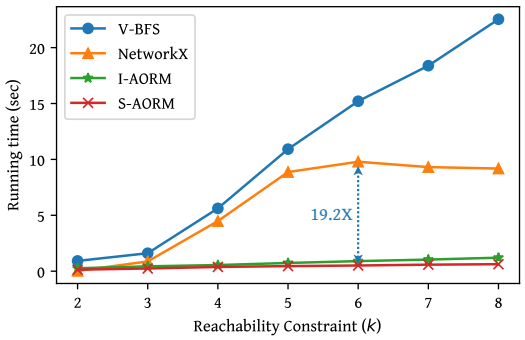


Figure 5. Incremental all-pairs shortest paths (APSP) performance results according to the reachability constraint (CollegeMsg dataset).

Analysis. The S-AORM provides good performance for computing of all-pairs shortest paths through the sparsity-aware graph reachability algorithm. Specifically, especially compared to dense matrix representations for real-world networks, the S-AORM delivers both computational and space efficiencies by exploiting sparse matrices.

IV. CONCLUSION

Computing *reachability* on massive graphs is important for modern real-world graph analytics. We have proposed the *sparsity-aware* reachability algorithm, which can provide fast performance for massive graphs via incremental fashion. Our framework can improve the performance for massive graph reachability by using *incremental* sparse matrix-based computation. In particular, experimental results show that our framework provides significant speed-up for synthetic graphs and real-world networks compared to existing methods. The proposed approach also has the benefit of being able to compute graph reachability for both *directed* and *disconnected* graphs.

Acknowledgment This work was supported by Institute of Information & Communication Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No. 2020-0-00073, *Development of Cloud-Edge based City-Traffic Brain Technology*).

REFERENCES

[1] S.-S. Kim, Y.-K. Kim, and Y.-M. Kang, "AORM: Fast Incremental Arbitrary-Order Reachability Matrix Computation for Massive Graphs," *IEEE Access*, vol. 9, pp. 69 539–69 558, 2021.

[2] J. X. Yu and J. Cheng, "Graph Reachability Queries: A Survey," in *Managing and Mining Graph Data*, ser. Advances in Database Systems. Springer, 2010, vol. 40, pp. 181–215.

[3] R. Seidel, "On the All-Pairs-Shortest-Path Problem in Unweighted Undirected Graphs," *Journal of Computer and System Sciences*, vol. 51, no. 3, pp. 400–403, 1995.

[4] Z. Zhang, P. Cui, X. Wang, J. Pei, X. Yao, and W. Zhu, "Arbitrary-Order Proximity Preserved Network Embedding," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD 2018, London, UK, August 19-23, 2018*. ACM, 2018, pp. 2778–2786.

[5] S.-S. Kim, M. Chung, and Y.-K. Kim, "Urban Traffic Prediction using Congestion Diffusion Model," in *International Conference on Consumer Electronics (ICCE) Asia 2020 (IEEE ICCE-Asia)*, 2020, pp. 360–363.

[6] "NetworkX: Network Analysis in Python," https://networkx.org/documentation/stable//reference/algorithms/generated/networkx.algorithms.shortest_paths.unweighted.all_pairs_shortest_path.html#networkx.algorithms.shortest_paths.unweighted.all_pairs_shortest_path, accessed: 2021-03-15.

[7] "Snap.py - SNAP for Python," https://snap.stanford.edu/snappy/index.html, accessed: 2021-10-09.

[8] R. A. Rossi and N. K. Ahmed, "The Network Data Repository with Interactive Graph Analytics and Visualization," in *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA*. AAAI Press, 2015, pp. 4292–4293.

[9] S. Pettie, "All Pairs Shortest Paths in Sparse Graphs," in *Encyclopedia of Algorithms*, 2016, pp. 52–55.