Available online at https://icsejournal.com/index.php/JCSE
Journal of Computer Science an Engineering (JCSE)
e-ISSN 2721-0251
Vol. 4, No. 2, August 2023, pp. 78-87

Enhancing Privacy in Graph Algorithms: Data-Oblivious Approaches to DFS and Dijkstra's Algorithm

Koteswara Rao Ch[1*], Kunwar Singh[2], Anoop Kumar[3]

[1] VIT-AP University, Amaravati, Andhra Pradesh 522237, India
[2,3] National Institute of Technology - Trichy, 620 015, Tamil Nadu, India
[1] koteswararao.ch@vitap.ac.in*; [2] kunwar@nitt.edu; [3] 406116006@nitt.edu
* corresponding author

ARTICLE INFO

Article History:
Received October 6, 2022
Revised September 30, 2023
Accepted October 1, 2023

Keywords:
Oblivious execution
Secure computation
Graph algorithms

Correspondence:
E-mail:
koteswararao.ch@vitap.ac.in

ABSTRACT

Data obliviousness is characterized by a consistent sequence of operations irrespective of input data and data-independent memory accesses, making it a suitable solution for users utilizing outsourced storage data services who aim to conceal their data access patterns. In ACM SIGSAC 2013, data-oblivious algorithms were introduced for breadth-first search, single-source single-destination (SSSD) algorithms, maximum flow, and minimum spanning tree. In this study, we present novel data-oblivious algorithms designed for depth-first search and single-source shortest path (Dijkstra's algorithm). Our proposed data-oblivious algorithms demonstrate efficiency comparable to non-data-oblivious counterparts, particularly for graphs containing fewer than 1000 nodes. This research contributes to advancing data privacy in outsourced storage services by providing effective data-oblivious solutions for common graph algorithms.

1. Introduction

In the contemporary landscape of cloud computing, the trend of outsourcing data to remote entities or servers is gaining substantial traction, proving advantageous for both individual users and business enterprises. The development of secure outsourcing techniques for diverse functionalities to untrusted servers has become a focal point in contemporary research. Although encryption methods are widely employed by businesses to protect the confidentiality of outsourced data, they fall short in safeguarding access patterns to private data. Access patterns often inadvertently disclose information about the input data and the underlying graph structure. The emergence of data-oblivious algorithms addresses these privacy concerns by ensuring the same sequence of operations irrespective of input data and promoting data-independent memory accesses.

Data obliviousness is characterized by the consistent sequence of operations regardless of input data, making it particularly suitable for use in outsourced tasks. A significant proportion of algorithms and data structures commonly utilized in practice lack data-obliviousness, thereby inadvertently revealing information about private or protected data. Naive approaches to achieving data obliviousness can lead to increased computation time, especially in operations such as searching, sorting, traversals, and binary trees. Hence, a meticulous design of algorithms and data structures is imperative to minimize the complexity of oblivious execution, approaching that of non-oblivious algorithms and data structures.

Assuming computations are conducted on secure data using secure multi-party techniques, the only information leakage in this context is through memory accesses and the set of intermediate results of the computation. However, guaranteed data privacy can be achieved by ensuring that memory accesses are oblivious or data-independent.

Marina Blanton et al. [1] proposed data-oblivious algorithms for breadth-first search, single source single destination (SSSD) shortest path, maximum flow, and minimum spanning tree at ACM SIGSAC 2013. Building upon this foundation, our work introduces data-oblivious depth-first search

http://dx.doi.org/10.36596/jcse.v4i2.277

and single-source shortest path algorithms using Dijkstra’s algorithm as the base. Considering a graph G = (V,E), we assume solutions based on the adjacent matrix representation of the graph. Notably, the execution time of data-oblivious algorithms remains comparable to that of non-data-oblivious algorithms when the number of vertices is fewer than 1000 nodes.

The concepts of secure computation and oblivious computation were initially proposed in [2, 3, 4]. In untrusted environments, Oblivious RAM (ORAM) was introduced for secure execution, with a focus on preventing information leakage through memory accesses. With the advent of cloud computing and related storage services, there has been a heightened interest in ORAM, as evidenced by [5, 6, 2, 7, 8, 9]. Private Information Retrieval (PIR) [10, 11, 12] addresses scenarios where clients interact with a server to retrieve specific records without disclosing the type of records being accessed.

Another significant privacy concept is Location-Based Services (LBS), commonly used in mobile services and cloud computing environments. Motivated by the LBS concept, our work introduces data-oblivious algorithms through multi-party computation. Additional solutions focusing on privacy-preserving public or joint graph algorithms have been proposed in [14] for all pairs shortest path and single-source shortest distance, and [15] for a privacy-preserving minimum spanning tree, both in a two-party environment.

Within the external memory model, data-oblivious algorithms have been proposed for sorting, selection, and compaction [16]. Subsequently, data-oblivious graph drawing algorithms such as Euler tours and Treemap drawings were proposed in [17]. Furthermore, based on external memory cache-oblivious modeling, algorithms were proposed in [3, 18, 4].

This research aims to contribute to the ongoing efforts in enhancing data privacy in cloud computing by introducing novel data-oblivious algorithms for DFS and Dijkstra’s algorithm, ensuring robust protection against unauthorized access patterns disclosure.

2. Method

We assume graph problems, that take the graph as input, G = (V, E). The graph is represented in adjacency matrix representation M. The assumed M is adjacency matrix with $|V| \times |V|$, that contains the values are in secure protected form. If the adjacency matrix is unweighted graph representation then it contains Boolean values of either 1 or 0. If there is an edge from u to v i.e. if $(u, v) \in E$, then the cell values is 1, otherwise 0. For the weighted graphs the weight of the edge is represented in protected form. We use the notations as follows.

$[\![x]\!]$	The protected value of x.
$[\![x]\!] \overset{?}{=} [\![y]\!]$	Comparison of equality of two protected vales, if condition is true 1 will return, otherwise 0 will be return.
$[\![x]\!] \overset{?}{<} [\![y]\!]$	Comparison of two protected vales, if true will produce 1, else 0.

if statement:
if (statement) then x = y else x = z
Oblivious execution of the **if**
statement: x = statement . y + (1 - statement). z

The code above is an if-else statement that is used to determine the value of the variable x. If the statement is True, then the variable x will be assigned the value of y. If the statement is False, then the variable x will be assigned the value of z. The code can also be written in the form of oblivious execution. Oblivious execution is a way of writing if-else code that is more concise and efficient. The code works by multiplying the value of the statement by y and (1 - statement) by z, and then adding the two results together. If the statement is True, then the result of the first multiplication will be greater and the variable x will be assigned the value of y. If the statement is False, then the result of the second multiplication will be greater and the variable x will be assigned the value of z.

```
Loop statement:
for i in 1 to $|V|$ do
if($M_{v,i}$ == 1)&&$C_i.col$ == $white$)
then $C_i.col$ == $gray$
oblivious execution of the above
is: for i in 1 to $|V|$ do
$[\![condition]\!] = ([\![M_{v,i}]\!] \overset{?}{=} 1) . ([\![C_i.col]\!] \overset{?}{=} \text{white})$
$[\![C_i.col]\!] = [\![condition]\!] . \text{gray} + (1 - [\![condition]\!]) . [\![C_i.col]\!]$
```

Meaning of the above loop is, if there exists an adjacent vertex of v and color of vertex is white, then the condition is true, it will give value 1. Correspondingly second statement is executed, i.e, color will change to gray if true or remains same.

2.1 Security Model of Data Oblivious Algorithm

Security model shows the indistinguishability of access patterns of the algorithm for different inputs of equal length. We introduce the security model as below.

Definition 1**,** (Security Model of Data Oblivious Algorithm): Adversary chooses his choice of input d_0 to a graph of the algorithm. Now let A(d_0) represents the sequence of memory accesses. Let the algorithm makes for the same sequence of instructions for all possible inputs. The algorithm is treated as data oblivious if access patterns of A(d_0) is independent of the input d_0.

2.2 Our Oblivious Algorithm for DFS

We propose an oblivious DFS algorithm with adjacent matrix representation to protect the graph structure. The idea of our proposed algorithm is, we denote a protected vector C of size |V |, initially all the nodes are colored with white except the starting node s and is colored with gray. After identify the adjacent nodes to s, those nodes are colored with gray and s with black i.e., visited node. Distance of these gray color nodes is updated in a protected vector C, from the source. The next visited node will be the node with maximum distance from the source.

There may be more than one node with maximum distance from source. All the gray color nodes are stored in another protected vector C′, one of the node in C′ randomly selected with minimum indexed value as next node to be processed.

Our proposed oblivious DFS algorithm is shown in Algorithm 1. Generally if we reveal any row in the matrix, it will leak the information about, how the graph is structured. To overcome this problem, we used random order of the nodes that are stored in the matrix. The second thing is among the gray color nodes, we choose the maximum indexed node at the maximum distance from the source node. Before entering into the algorithm first we have to create a vector named with C of size |V |. The elements of the vector C having two different fields: fist is color, second is distance from the source vertex. Next initialize the vector C, set all the color fields with white and the color s with gray. The initial distance is set to 0 from s. In the execution of our algorithm, the vales of the vector are protected vales.

Assume that given graph is connected. In the initial step starting node s is colored with gray. Step 1 of the algorithm describes about retrieving the next row to be selected from the matrix M. In step 2 of the proposed algorithm obliviously we are checking the condition that whether there exists the adjacent node or not and at the same time it’s color also. If both conditions are true respective node is colored with gray and the distance is incrementing with a value 1. In this step all the adjacent nodes are converted into gray color with respective distance. In step 3, we are computing the maximum distance of gray color nodes. We are storing gray nodes with distance = max into another vector C′, nodes of C′ are permuted. In step 5 of the proposed algorithm we are choosing the next node to be processed with minimum indexed value. OPEN(.) is revealing the index value of the node. Even though it is revealing the node value there is no information leakage, because we permuted all the nodes with π. Where π : [1,|V |]→[1,|V |]. Our proposed algorithm always choose a gray color node with the maximum distance from the source with minimum key value for the all iterations, it will correctly execute the DFS stack. If the given graph is not a connected, we have to assume the fake nodes in between the node of adjacency matrix representation.

Algorithm 1: Data-oblivious Depth First Search Algorithm

```
1 Assign node v to s and access row $M_V$ of M. After that update C using $M_V$ as follows
2 for i in 1 to $|V|$ do
    $[condition] = (\llbracket M_{v,i}] \overset{?}{=} 1) . ([[C.col] \overset{?}{=} \text{white})$
    $[C_i.col] = [condition] . \text{gray} + (1 - [condition\rrbracket) . [C_i.col]$
    $[C_i.dist] = [cond\rrbracket(\llbracket C_v.dist] + 1) + (1 - [cond\rrbracket)\llbracket C_i.dist]$
3 $[max] = 0$
    for i in 1 to / V / do
    $[condition_i] = [C_i.col] \overset{?}{=} \text{gray}$
    $[condition'_i] = [C_i.dist] >^? [max]$
    $[max] = [cond_i\rrbracket.\llbracket condition'_i\rrbracket.\llbracket C_i.dist] + (1 - [condition_i\rrbracket.\llbracket condition'_i\rrbracket)\llbracket max]$
4 for i in 1 to | V | do
    $[condition_i] = [C_i.dist] = [max]$
    $[C'_i.value] = [condition_i] [condition'_i\rrbracket.i$
    $[C'_i.key] = [condition_i] [condition'_i\rrbracket.\llbracket\pi(i)\rrbracket$
5 $[min] = 0$
    $[i_{min}] = 0$
    for i in 1 to / V / do
    $[condition] = [min] <^? [C'_i.key]$
    $[min] = [cond\rrbracket. [C'_i.key] + (1 - [cond\rrbracket).\llbracket min]$
    $[i_{min}] = [condition\rrbracket. [C'_i.value] + (1 - [condition\rrbracket).\llbracket i_{min}]$
    $[i_{min}] = \text{OPEN}(\llbracket i_{min}\rrbracket)$
    $v = \text{OPEN}(\llbracket C'_{i_{min}}.value\rrbracket)$
6 Set the color of $C_V$ to black
7 Repeat the algorithm $|V|$ - 1 times
```

3. Results and Discussion

3.1 Complexity Analysis

Here we are giving the complexity analysis of the proposed algorithms. The com- plexity of the proposed oblivious DFS algorithm in step 1 is O(1), it is like general algorithm description. Step 2 of the proposed DFS algorithm will be evaluated $|V|$ number of times, therefore its complexity is O($|V|$). Step 3 is part of step 2, for the calculation of the maximum distance from the source node in the given graph. Therefore the overall run time of these steps is O($|V|^2$). This is optimal for representation of the graph in adjacency matrix format, with $|E| = \theta(|V|\ )$.

3.2 Security Analysis

The proposed algorithms are data oblivious according to the Definition 1.

Theorem 1 The proposed DFS algorithm is data oblivious.

Proof (Sketch) The proof of this theorem as follows. According to Definition 1, let d is the input and A(d) is the memory access pattern of the input by the algorithm. We will show that the access pattern of A(d) is independent of d. We analyze each and every operation in the proposed algorithm. In the proposed algorithm, the sequence of instructions is same for all the possible given connected graphs

G=(V,E) with |V | number of nodes. At the same time the accessing of the memory is indistinguishable for given graphs as well as randomly permuted graphs with |V | number of nodes.

The initial steps of the proposed algorithm are independent of graph G with their inputs. Means it is common for all the possible inputs of the graph. In step 1 of the proposed algorithm, the position of the node is indistinguishable for input node values as well as permuted node values. In step 2 of the proposed algorithm, every time it is checking the adjacent node and color, therefore it is common for all the possible inputs. That means the instructions and accessing of the memory locations of protected vector C are same for all inputs.

The only chance of information leakage is OPEN(.). However the revealed information here is randomly permuted order. The next node to be selected is also a randomly permuted node. Therefore there is no chance of information leakage of the graph structure. We conclude that the revealed positions are random at the same time accessing of memory locations are indistinguishable from the randomly generated graphs.

3.3 Our Oblivious Algorithm for Dijkstra's

In Algorithm 1 we already computed the distances from the source node to all the other nodes that are existed in the given graph. In this work, we are obliviously proposing the shorted distances from source node to other nodes using Dijkstra's algorithm as base. The oblivious DFS is basic requirement for this. In a given undirected weighted graph G=(V,E) from the source node s to all the other nodes we can find the shortest distances. Our proposed oblivious Dijkstra's algorithm is shown in Algorithm 2. Before entering into the algorithm first we have to create a vector named with C of size |V |. The elements of the vector C having four different fields: fist is color, second is distance from the source vertex information in DFS tree, third is cost of the edge (u, v) and fourth is calculated distance i.e, d[v]. Next initialize the vector C, set all the color fields with white and the color s with gray. The initial distance is set to 0 from s. In the execution of our algorithm, the vales of the vector are protected vales.

Assume that given graph is undirected weighted graph. In the initial step starting node s is colored with gray. Step 1 of the algorithm describes about retrieving the next row to be selected from the matrix M. In step 2 of the proposed algorithm obliviously we are checking the condition that whether there exists the adjacent node or not and at the same time we are checking it's color. If both conditions are true [cond1] will return 1 other wise 0. At the same time we are checking [cond2] also, to identify the minimum distance value of the indexed node. If [cond1] and [cond2] both are true respective node is colored with gray next we are calculating the distance by incrementing with a value 1. In this step all the adjacent nodes are converted into gray color with respective distance.

In step 3 of the proposed algorithm we are choosing obliviously all the gray color nodes from the protected vector C with minimum distance. To achieve the data-obliviousness, another vector C′ is created, which contains all the gray nodes with minimum distance from the source node. In step 4 of the algorithm we are choosing random permutation π of nodes and assigning π(i) to all the nodes in C′. In step 5 of the proposed algorithm we are choosing the next node to be processed with minimum indexed value. OPEN(.) is revealing the index value of the node. Even though it is revealing the node value there is no information leakage, because we permuted all the nodes with π. Where π : [1,|V |]→[1,|V |]. Our proposed algorithm choose a gray color node always with minimum distance and minimum indexed node from the source with minimum key value.

Algorithm 2: Single Source Shortest Distance (Dijkstra's) Algorithm

```
1 Assign node v to s and access row $M_V$ of M. After that update C using $M_V$ as follows
2 for i in 1 to / V / do
    $[cond_1] = ([\![M_{v,i}]\!] \overset{?}{=} 1)$ && $([C.col] \overset{?}{=}$ white$)$
    $[cond_2] = ([\![C_i.d[u] + C_i.c(u,v)]\!]) < [C_i.d[v]]\!]$
    $[C_i.d[v]]\!] = [cond_1]\!][\![cond_2]\!]([\![C_i.d[u] + C_i.c(u,v)]\!]) + (1- [cond_1] . [$
    $cond_2]\!]) . [C_i.d[v]]\!] \; [C_i.dist] = [cond_1] \; [cond_2]\!]([\![C_v.dist] + 1) + (1 - [$
    $cond_1]\!][\![cond_2]\!])[\![C_i.dist]$
3 $[min] = \infty$
    for i in 1 to | V | do
    $[condition_i] = [C_i.col] \overset{?}{=}$ gray
    $[condition'_i] = [C_i.dist] <^? [min]$
    $[min] = [condition_i]\!].[\![condition'_i]\!].[\![C_i.dist] + (1 - [condition_i]\!].[\![condition'_i]\!])[\![min]$
4 for i in 1 to | V | do
    $[condition_i] = [C_i.col] \overset{?}{=}$ gray
    $[condition''_i] = [C_i.dist] \overset{?}{=} [min]$
    $[C'_i.value] = [condition_i] \; [condition'_i{}']\!].i$
    $[C'_i.key] = [condition_i] \; [condition'_i{}']\!].[\![\pi(i)]\!]$
5 $[max] = 0$
    $[imax] = 0$
    for i in 1 to | V | do
    $[condition] = [C'_i.key] >^? [max]$
    $[max] = [condition]\!]. [C'_i.key] + (1 - [condition]\!]).[\![max]$
    $[imax] = [condition]\!]. [C'_i.value] + (1 - [condition]\!]).[\![imax] \; [imax] =$ OPEN$([[imax]])$
    v = OPEN$([\![C'_{i_{max}}.value]\!])$
6 Set the color of $C_V$ to black
7 Repeat the algorithm | V | - 1 times
```

3.4 Experimental Results

The practical efficiency of the proposed algorithms is demonstrated in this work. We have implemented data-oblivious DFS and data-oblivious Dijkstra algorithms. Also compared the time complexity with traditional DFS and Dijkstra algorithms for sparse and dense graphs with different number of nodes.

All the experiments are implemented on machines configured with an Intel(R) i7-6700 3.41 GHz processor, 16 GB RAM and 64-bit Operating system. The implementation is held in a visual studio code C++ and TDM g++ with 1.1309.0 for compiling and running the programs.

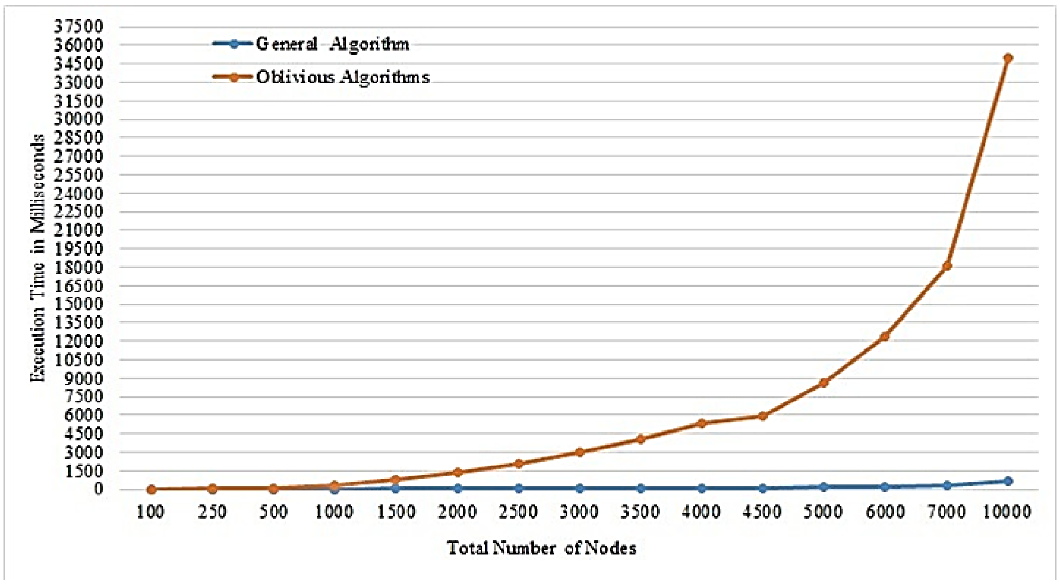

Figure 1. Performance comparison of the proposed oblivious DFS with the general DFS, when sparse graph is given as input. x-axes shows number of nodes and y-axes shows time taken

Figure 1, exhibits the comparison of the proposed oblivious DFS algorithm with the general DFS algorithm. The input graph for this experiment is sparse graph.

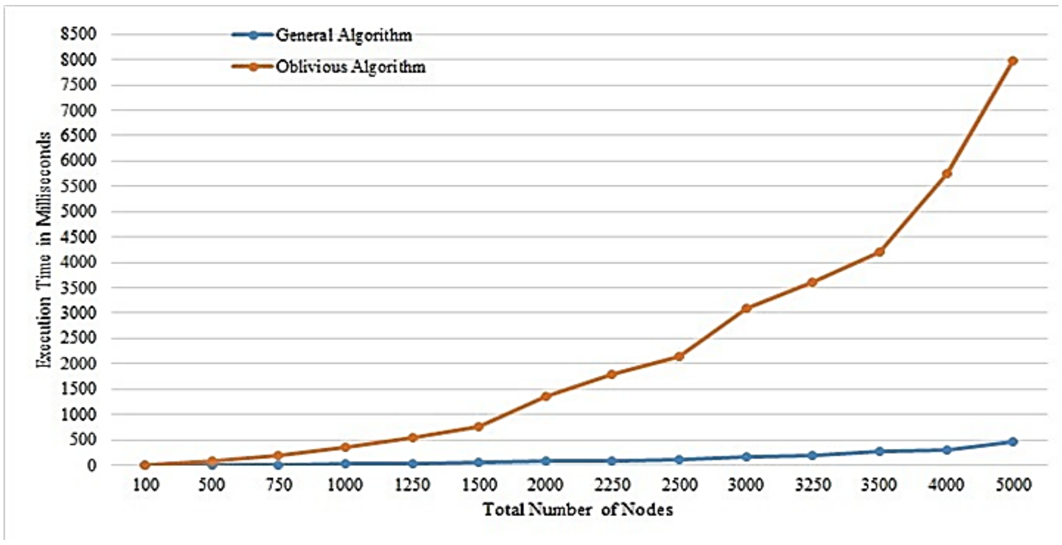

Figure 2. Performance comparison of the proposed oblivious DFS with the general DFS, when dense graph is given as input. x-axes shows number of nodes and y-axes shows time taken.

Figure 2, exhibits the comparison of the proposed oblivious DFS algorithm with the general DFS algorithm. The input is dense graph. Figure 3 exhibits the comparison of the proposed oblivious Dijkstra algorithm with the general Dijkstra algorithm. The input is sparse graph with weight.

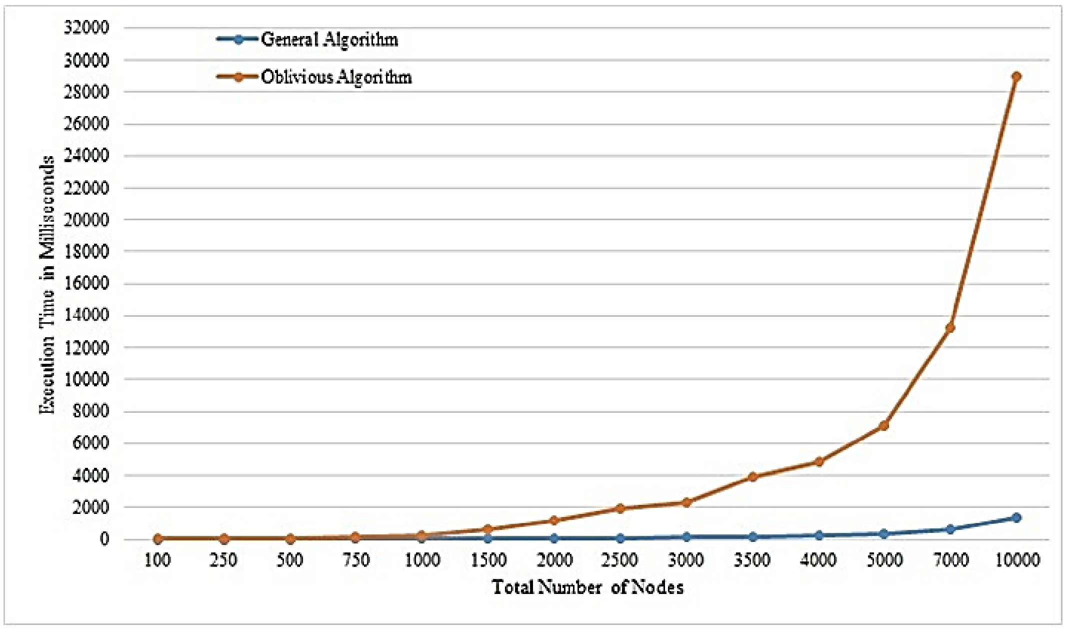

Fig. 3 Performance comparison of the proposed oblivious Dijkstra algorithm with the general Dijkstra algorithm, when weighted sparse graph is given as input. x-axes shows number of nodes and y-axes shows time taken.

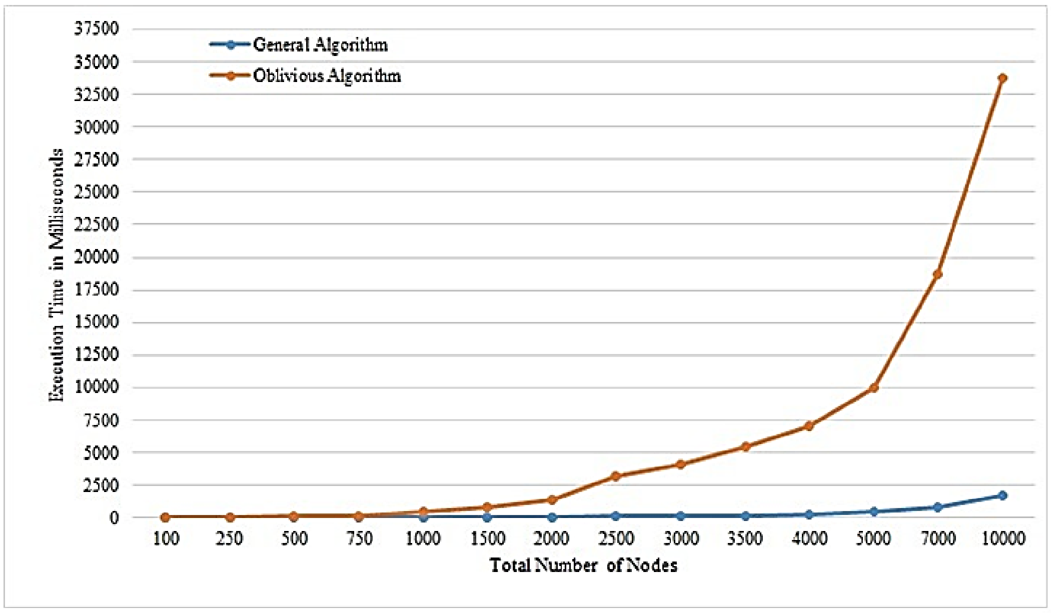

Figure 4. Performance comparison of the proposed oblivious Dijkstra algorithm with the general Dijkstra algorithm, when weighted dense graph is given as input. x-axes shows number of nodes and y-axes shows time taken.

Figure 4, exhibits the comparison of the proposed oblivious Dijkstra algorithm with the general Dijkstra algorithm. The input is dense graph with weight.

4. Conclusion

Overall, this research successfully proposes a provably secure data-oblivious algorithm for DFS and Dijkstra. Although the time complexity of the data-oblivious algorithm is the same as the general algorithm, experimental results indicate that the time difference becomes more significant when the number of nodes reaches 5000. The main findings affirm that the data-oblivious algorithm effectively operates on both sparse and dense graphs. Its primary advantage lies in its ability to conceal data access patterns without significantly sacrificing performance. Nevertheless, it must be acknowledged that this research has certain limitations, such as the Production Environment Context and Performance on Very Large Graphs. The implications of these findings can be felt in the development of data security systems and graph processing at a higher level. It is hoped that this research can serve as a foundation for further exploration in securing graph processing algorithms in the context of outsourced data storage. For future research, it is recommended to explore Optimization for Performance on a Larger Scale, Research Integration with Cloud Platforms, and the Development of Additional Algorithms, which can lay the groundwork for the development of more advanced data security technologies.

References

[1] M. Blanton, A. Steele, and M. Alisagari, "Data-oblivious graph algorithms for secure computation and outsourcing," Proceedings of the 8th ACM SIGSAC symposium on Information, computer and communications security. ACM, May 08, 2013. doi: 10.1145/2484313.2484341.

[2] M. Ajtai, "Oblivious RAMs without cryptogrpahic assumptions," Proceedings of the forty-second ACM symposium on Theory of computing. ACM, Jun. 05, 2010. doi: 10.1145/1806689.1806716.

[3] L. Arge, M. A. Bender, E. D. Demaine, B. Holland-Minkley, and J. I. Munro, "Cache-oblivious priority queue and graph algorithm applications," Proceedings of the thiry-fourth annual ACM symposium on Theory of computing. ACM, May 19, 2002. doi: 10.1145/509907.509950.

[4] L. Arge, M. A. Bender, E. D. Demaine, B. Holland-Minkley, and J. Ian Munro, "An Optimal Cache-Oblivious Priority Queue and Its Application to Graph Algorithms," SIAM Journal on Computing, vol. 36, no. 6. Society for Industrial & Applied Mathematics (SIAM), pp. 1672–1695, Jan. 2007. doi: 10.1137/s0097539703428324.

[5] P. Williams, R. Sion, and B. Carbunar, "Building castles out of mud," Proceedings of the 15th ACM conference on Computer and communications security. ACM, Oct. 27, 2008. doi: 10.1145/1455770.1455790.

[6] B. Pinkas and T. Reinman, "Oblivious RAM Revisited," Advances in Cryptology – CRYPTO 2010. Springer Berlin Heidelberg, pp. 502–519, 2010. doi: 10.1007/978-3-642-14623-7_27.

[7] I. Damg˚ard, S. Meldgaard, J. B. Nielsen, Perfectly secure oblivious ram without random oracles, in: Theory of Cryptography Conference, Springer, 2011, pp. 144–163.

[8] M. T. Goodrich and M. Mitzenmacher, "Privacy-Preserving Access of Outsourced Data via Oblivious RAM Simulation," Automata, Languages and Programming. Springer Berlin Heidelberg, pp. 576–587, 2011. doi: 10.1007/978-3-642-22012-8_46.

[9] E. Kushilevitz, S. Lu, R. Ostrovsky, On the (in) security of hash-based oblivious ram and a new balancing scheme, in: Proceedings of the twenty-third annual ACM-SIAM symposium on Discrete Algorithms, SIAM, 2012, pp. 143–156. doi: 10.1137/1.9781611973099.13

[10] C. Gentry, Z. Ramzan, Single-database private information retrieval with constant communication rate, in: International Colloquium on Automata, Languages, and Programming, Springer, 2005, pp. 803–815. doi: 10.1007/11523468_65

[11] H. Lipmaa, "An Oblivious Transfer Protocol with Log-Squared Communication," Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 314–328, 2005. doi: 10.1007/11556992_23.

[12] S. Wang, X. Ding, R. H. Deng, and F. Bao, "Private Information Retrieval Using Trusted Hardware," Computer Security – ESORICS 2006. Springer Berlin Heidelberg, pp. 49–64, 2006. doi: 10.1007/11863908_4.

[13] D. Eppstein, M. T. Goodrich, and R. Tamassia, "Privacy-preserving data-oblivious geometric algorithms for geographic data," Proceedings of the 18th SIGSPATIAL International Conference on Advances in Geographic Information Systems. ACM, Nov. 02, 2010. doi: 10.1145/1869790.1869796.

[14] J. Brickell, V. Shmatikov, Privacy-preserving graph algorithms in the semi-honest model, in: International Conference on the Theory and Application of Cryptology and Information Security, Springer, 2005, pp. 236–252. doi: 10.1007/11593447_13

[15] C. H. K. Rao and K. Singh, "Securely solving privacy preserving minimum spanning tree algorithms in semi-honest model," International Journal of Ad Hoc and Ubiquitous Computing, vol. 34, no. 1. Inderscience Publishers, p. 1, 2020. doi: 10.1504/ijahuc.2020.107501.

[16] M. T. Goodrich, "Data-oblivious external-memory algorithms for the compaction, selection, and sorting of outsourced data," Proceedings of the twenty-third annual ACM symposium on Parallelism in algorithms and architectures. ACM, Jun. 04, 2011. doi: 10.1145/1989493.1989555.

[17] M. T. Goodrich, O. Ohrimenko, R. Tamassia, Data-oblivious graph drawing model and algorithms, arXiv preprint arXiv:1209.0756 (2012).

[18] G. S. Brodal, R. Fagerberg, U. Meyer, and N. Zeh, "Cache-Oblivious Data Structures and Algorithms for Undirected Breadth-First Search and Shortest Paths," Algorithm Theory - SWAT 2004. Springer Berlin Heidelberg, pp. 480–492, 2004. doi: 10.1007/978-3-540-27810-8_41.