electronics

Article

A Swell Neural Network Algorithm for Solving Time-Varying Path Query Problems with Privacy Protection

Man Zhao

School of Electrical and Electronic Engineering, North China Electric Power University, Beijing 102206, China; 50301363@ncepu.edu.cn

Abstract: In this paper, a swell neural network (SNN) algorithm was proposed for solving time-varying path query (TVPQ) problems with privacy protection with the following goals: (i) querying the K-nearest paths with time limitations in a time-varying scenario, and (ii) protecting private information from neighborhood attacks. The proposed SNN is a network in which the optimal paths can be calculated at the same time with no need for training. For TVPQ, a node is considered a neuron, and time-varying means that an edge has different costs in different time windows. For SNN, the query paths are swell sets from the start to the target within an upper limit. An encrypted index is designed for privacy protection. The evaluation of the efficiency and accuracy of the SNN was carried out based on New York road instances.

Keywords: time-varying path query (TVPQ); swell neural network (SNN); privacy protection; encrypted index

Citation: Zhao, M. A Swell Neural Network Algorithm for Solving Time-Varying Path Query Problems with Privacy Protection. *Electronics* **2024**, *13*, 1248. https://doi.org/10.3390/electronics13071248

Academic Editors: Domenico Rosaci

Received: 14 March 2024
Revised: 24 March 2024
Accepted: 26 March 2024
Published: 27 March 2024

1. Introduction

Path query problems with privacy protection have attracted attention in many fields, such as industry [1,2], management science [3], computer science [4], and transportation [5]. Path query problems with privacy protection were first proposed in 2011 by Cao [4], who proposed utilizing the principle of filtering and verification to keep cloud data secure. Shang H [6], Gouda K [7], and Lin W [8] carried out further research based on this idea. Meng X studied the problem of graph encryption and proposed an approximate shortest distance query method (GRECS) for encrypted graphs [9]. A CryptGraph scheme has been designed, with which graph analysis can be performed on encrypted graphs, and the privacy of users' graph data and analysis results can be protected [10]. However, these studies lack consideration of time variation.

There is a suite of studies addressing the time-varying path query problem. The shortest path algorithm through a time-varying network was first proposed in 1966 by Cooke and Halsey [11]. Such path query methods for dynamic graphs are valuable references. Frigioni D proposed the global dynamic algorithm FMN to solve the single-source shortest path problem for dynamic graphs [12]. Additionally, SWSF-FP was proposed by Ramalingam G [13]. A global dynamic algorithm for computing the shortest paths between all pairs of vertices in a dynamic graph was also proposed [14]. An improved algorithm was proposed to maintain the shortest paths between all pairs of vertices in a dynamic graph [15]. In addition, Liu C [16], Ghosh E [17], Sun F [18], and Wu B [19] described their research on this topic. However, these works neglect privacy protection.

To our knowledge, current algorithmic models either ignore the effect of the time factor or the need for privacy protection. There are still unresolved TVPQ problems regarding privacy protection, despite the urgent need for solutions in applications such as mapping services [20]. Different departure times and expected arrival times lead to changes in planned routes. Users expect to obtain the top recommended routes under the current time constraints. Pioneering works by Huang W [3,21,22], Zhang C [23], and Shen M [24] provided schemes for reference. Huang W's work lacks a design for multiple paths, and accuracy cannot be fully guaranteed in Shen M's work. Current algorithms have limitations in terms of multipath requirements and accuracy guarantees.

In this paper, an SNN algorithm was proposed to solve TVPQ problems with privacy protection. An SNN is a network in which the optimal paths can be calculated at the same time and there is no need for training. For TVPQ, nodes are considered neurons, and time-varying means that an edge has different costs in different time windows. The core of the SNN is that the query paths are swell sets from the start node to the target node, and their arrival times are less than the upper limit.

The main contributions of this paper are as follows:

1. The SNN algorithm can help to find multiple paths at once, including the shortest paths. This is difficult to achieve with other algorithms under time-varying conditions.
2. For privacy protection, a scheme was designed with an encrypted index that effectively prevents the leakage of user information.
3. Theoretical analyses and contrasting experimental results prove the efficiency, security, and accuracy of the algorithm.

The remainder of this paper is organized as follows. Section 2 introduces prerequisites, Section 3 presents the SNN algorithm, Section 4 reports the experimental results, and concluding statements are presented in Section 5.

2. Preliminaries

This section introduces several prerequisites that will be used throughout this paper.

2.1. Definition of Time-Varying Path Query

Definition 1 (Time-Varying Network)**.** *A time-varying network graph is defined as* $G=(N,E)$*, where* N *represents the set of nodes and* E *denotes the set of network edges.* $E_{ij} \in E$ *stands for the edge from node i to node j, and it is associated with different time weights* c_{ij}^{r} *for different time windows* D_{ij}^{r}*.*

Definition 2 (Path of a Time-Varying Network)**.** *A path* P_{1k} *on a time-varying network is defined as a sequence of nodes* $<n_1,\ldots,n_{i,}\ldots,n_k>$*, where n1 is the start node,* n_k *is the target node,* n_i *is the ith node in the path, and* T_e *is the arrival time at* n_k*. A route based on* P_{1k} *is a sequence of nodes* $<n_1,\ldots,n_i,\ldots,n_k>$ *[25].*

According to the definition of the path of a time-varying network, the arrival time is $Te = Ts + \sum_{i=1}^{k} c_{n_i,n_{i+1}}$ for a path $P_{1k} =< n_1,\ldots,\mathrm{n_i},\ldots,n_k >$.

2.2. Model of a Time-Varying Network Query

Given a directed graph $G=(N,E)$, a start node $n_s \in N$, a target node $n_t \in N$, a start time T_s, an upper limit of the arrival time T_u, and the number f of paths to query for, a time-varying network query finds the set of paths $(p1_{st}, p2_{st}, \ldots\ldots)$ such that the arrival time is less than T_u.

For node *i*, the successor set can be defined as $S_i = \{sd_j \mid n_j \in N\}$, where the length of d_i represents the number of swells from node *i* to other nodes.

$$sd_i = \begin{cases} [], d_i = 0 \\ \left[[t_1, t_2]\right], d_i = 1 \\ \left[[t_1, t_2], \ldots, [t_k, t_{k+1}]\right], d_i = k \end{cases}$$

For node *j*, the predecessor set can be defined as $F_j = \{fd_i \mid n_i \in N\}$, where the length of e_j represents the number of swells from other nodes to *j*.

$$fd_i = \begin{cases} [], e_j = 0 \\ \left[[t_1, t_2]\right], e_j = 1 \\ \left[[t_1, t_2], \ldots, [t_k, t_{k+1}]\right], ej = k \end{cases}$$

Clearly, if there is one swell from n_i to n_j, at t_1 to t_2 both sd_j and fd_i above contain $[t_1, t_2]$. The predecessor set of n_t collects the swells leading to n_t with the arrival times. Based on the above two features, paths can be obtained by tracing the predecessor sets of n_t iteratively.

For privacy protection, three algorithms were used: RSA, AES, and ORE.

In RSA, the encryption keys are public, while the decryption keys are not, so only the person with the correct decryption key can decipher an encrypted message. This avoids the need for a "courier" to deliver keys to recipients through another secure channel before transmitting the originally intended message [26,27].

AES is a substitution-permutation network block cipher based on the design principles of Ron Rivest, Adi Shamir, and Leonard Adleman's earlier Data Encryption Standard (DES). It uses a variable-length key from 128 bits to 256 bits and operates on fixed-size blocks of 128 bits. Both parties must agree on the key in advance to ensure that the key information cannot be obtained by a third party [27,28].

An order-revealing encryption (ORE) scheme is a tuple of three algorithms Π = (ORE.Setup, ORE.Encrypt, ORE.Compare) defined over a well-ordered domain *D* with the following properties:

ORE.Setup(1^λ) $\rightarrow sk$: On inputting a security parameter λ, the setup algorithm outputs a secret key sk.

ORE.Encrypt(sk, m) $\rightarrow ct$: On inputting a secret key sk and a message $m \in D$, the encryption algorithm outputs a ciphertext ct.

ORE.Compare($ct1$, $ct2$) $\rightarrow b$: On inputting two ciphertexts $ct1$, $ct2$, the compare algorithm outputs a bit $b \in \{0, 1\}$.

An ORE scheme over a well-ordered domain D is correct if for $sk \leftarrow ORE.Setup(1^\lambda)$ and all messages $m1, m2 \in D$ [29–31]:

$$Pr\left[ORE.Compare(ct1, ct2) = 1(m1 < m2)\right] = 1 - negl(\lambda)$$

Definition 3 (Encrypted Index)**.** *An encrypted index consists of two parts, as follows:*

- T_e *encrypted with the ORE key*
- p_{st} *encrypted with the public secret key of RSA-1.*

Definition 4 (Encrypted Query)**.** *An encrypted message for a query consists of three parts, as follows:*

- $(s,\ t,\ T_s)$ *encrypted with the AES key*
- T_u *encrypted with the ORE key and the AES key*
- f *is the plaintext representing the number of paths queried.*

3. Construction of the SNN

This section presents the SNN algorithm and its complexity and security analyses.

3.1. Design of the SNN Algorithm

The SNN is a swell neural network without any training. To formalize the SNN, each node is viewed as a neuron and the graph composed of nodes and the edges connecting them is viewed as a neural network.

Figure 1 illustrates a general neuron structure for an SNN. As shown in Figure 1, there are four parts in each loop: the input, the neuron state, the neuron feedback, and the output. The main functions of these four parts are as follows: the input swell coming from its predecessor nodes over time, the neuron state that determines whether to spread, the feedback that updates the neuron state, and the output swells that spread to the successor nodes.

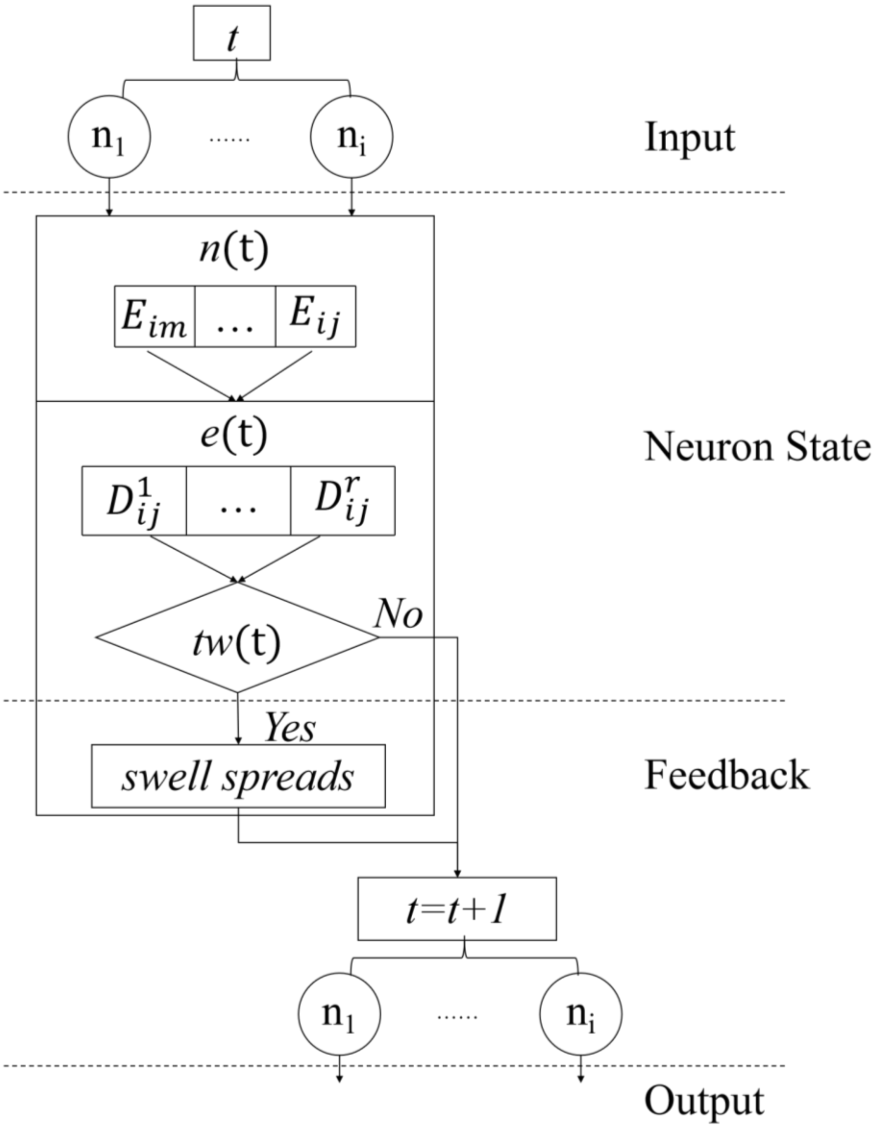

Figure 1. A general neuron structure of SNN.

1. Input: The input swells come from the predecessor nodes.
2. Neuron state: The neuron state consists of three parts: nodes, edges, and time windows. The functions *n(t)*, *e(t)* and *tw(t)* represent the processing of the node, edge, and time window, respectively, and *t* represents the current time.
   For n_i, the successor set can be defined as $S_i = \{sdj \mid nj \in N\}$, where the length of d_i represents the number of swells from n_i to n_j.

$$sd_i = \begin{cases} [], d_i = 0 \\ \left[[t_1, t_2]\right], d_i = 1 \\ \left[[t_1, t_2], \ldots, [t_k, t_{k+1}]\right], d_i = k \end{cases}$$

   The predecessor set can be defined similarly.
   For edge E_{ij}, the time set collects the arrival times of n_i for which no swell is currently formed.

$$P_{ij} = \begin{cases} [], no \\ [t_1, t_2 ...], yes \end{cases}$$

For time window D_{ij}^r, v_{ij}^r represents the set of key-value pairs for arrival and available times when no swell is formed.

$$v_{ij}^r = \begin{cases} [], no \\ [t_1, t_2 ...], yes \end{cases}$$

3. Feedback: The swells spread along the edge in the time window for which the available time c_k satisfies the cost c_{ij}^r, and c_k is computed as follows:

$$c_k = \begin{cases} t - t_k, if\ l_{ij}^r <= t_k < u_{ij}^r \\ t - l_{ij}^r, if\ t_k < l_{ij}^r \end{cases}$$

4. Output: The output swells continue to spread to the successor nodes.

The overall algorithm, referred to as Algorithm 1, serves as the foundation for the SNN. The partitioning algorithms of the SNN algorithm are introduced as follows. To find the shortest paths, the SNN algorithm works as follows:

- Initialize the graph as Algorithm 2.
- Activate the start node directly as Algorithm 3.
- Iterate over the successor edges of the start node as Algorithm 4 to spread ripples.
- Iterate over all nodes over time to activate the nodes as Algorithm 4 until the target node is activated. The path can be obtained if the current time is within the maximum time range. Otherwise, no path meets the requirements.

Algorithm 1 Encrypted Index Construction (EIC)

```
Input: G, s, t, T_s, T_u, f
Output: P_st, T_e
Initialize G as Algorithm 2.
T_e = T_s
Initialize n_s as Algorithm 3.
while l_st < f and T_e <= T_u do
    T_e = T_e + 1
    for n_i ∈ G do
        Try to activate n_i as Algorithm 4.
    if F_t has been changed do
        p_st >> P_st as Algorithm 5.
    end if
end while
Encrypt each path with K_ORE and Pk_RSA2.
```

First, initialize the graph as shown in Algorithm 2. A directed graph is constructed. Each node is initialized with an empty father set and son set for collecting predecessors and successors. For the edges, a time set is initialized for the arrival times of the predecessors, and time windows with boundaries, costs, and the set of key-value pairs for arrival and available times are initialized.

Algorithm 2 Graph Initialization (GI)

Input: G

Output: G

for $E_{ij} \in G$ do

    $F_i = None$

    $S_i = None$

    $F_j = None$

    $S_j = None$

    $c_{ij} = None$

    for $D_{ij}^r \in D_{ij}$ do

        $v_{ij}^r = None$

    end for

end for

return G

The start node is initialized according to Algorithm 3. The algorithm iterates over the predecessor edges of the start node n_s and adds the start time T_s to its time set and time window set.

Algorithm 3 Start Node Initialization (SI)

Input: G, n_s, T_s

Output: G

for $E_{si} \in E_s^S$ do

    $T_s >> E_{sj}$

    for $D_{si}^r \in D_{si}$ do

        $(T_s,\ 0) >> v_{ij}^r$.

    end for

end for

To activate the nodes over time, if the input node is a start node or its father set is not empty, then the algorithm iterates over the successor edges of the input node to spread swells, as Algorithm 4.

Algorithm 4 Node Activation (NA)

Input: G, n_i, n_s

Output: G

if $n_i = n_s$ or $F_i \neq None$:

    for $E_{ij} \in E_i^S$ do

        Spread as Algorithm 6

    end for

end if

New paths are added to paths according to Algorithm 5. The function AP is executed iteratively until f paths are found that meet the requirements.

Algorithm 5 Add Paths (AP)

Input: G, n_i, p_{jt}, n_s, f

Output: P_{st}

if $l_{st} < f$ do

  $p_{it} = [i] + p_{jt}$

  if $n_i = n_s$

    $p_{st} >> P_{st}$

    if $l_{st} = f$ do

      break

    end if

  end if

  else

  for $j \in F_i$ do

    AP(Algorithm 5)

  end for

end if

The swells spread along the edge as shown in Algorithm 6. Whether the swells spread is dependent on the determination of the time window over time. The swells are retained, process records are deleted, and the network prepares for the next iteration, as shown in Algorithm 3, after the swells spread.

Algorithm 6 Swell Spreading (SS)

Input: $G,\ E_{ij},\ T\ \left(current\ time\right)$

Output: G

$de_{ij} = None$ //Initialize the set of arrival times to be deleted from E_{ij}

for $D_{ij}^r \in D_{ij}$

  key = Perform time window determination as Algorithm 7

  if $key \neq None$ do

    $key >> de_{ij}$

  end if

end for

for $de \in de_{ij}$:

  Delete de from c_{ij}

  Delete (de ,*) from v_{ij}^r

  $\left(i\ ,\left[time, T\right]\right) >> F_j$, $\left(j\ ,\left[time, T\right]\right) >> S_i$ and $\left(T,\ 0\right) >> v_{ij}^r$ as Algorithm 3

end for

Determine each time window of the edges over time, as shown in Algorithm 7. The swells spread along the edge in the time window for which the available time satisfies the cost.

Algorithm 7 Time Window Determination (TD)

Input: $G, D_{ij}^r, T(currenttime)$.
Output: G
$de = None$ // Initialize the arrival time to be deleted from E_{ij}
for $key \in v_{ij}^r$ (the key-value pair set of D_{ij}^r)
  if $l_{ij}^r <= key < u_{ij}^r$ do
    $v_{ij}^r[key] = T - key$
  else if $key < l_{ij}^r$ do
    $v_{ij}^r[key] = T - l_{ij}^r$
  end if
  if $v_{ij}^r[key] = c_{ij}^r$ do
    $de = key$
  end if
end for
return de

3.2. An Example of the SNN Algorithm

To illustrate the SNN algorithm, let us consider the example shown in Figure 2 and Table 1. There are three nodes and three edges in the time-varying network, where A is the start node, C is the target node, the start time is 0, the upper limit of the arrival time is 5, and the number f of paths to query is 2. For each edge, there are two time windows.

Table 2 illustrates the detailed steps of the running algorithm on the SNN. It is evident that the paths are <A, C> and <A, B, C>. At moment 0, only n_A is reached and the swell has not spread to the next node. At moment 1, a swell spread to n_B, which is a child of n_A. At moment 2, a swell reaches C and a path from A to C is found as [A, C] in time 2. At moment 3, the swell that spread to n_B at moment 1 arrived at n_C, and the second path from A to C is found as [A, B, C], in time 3.

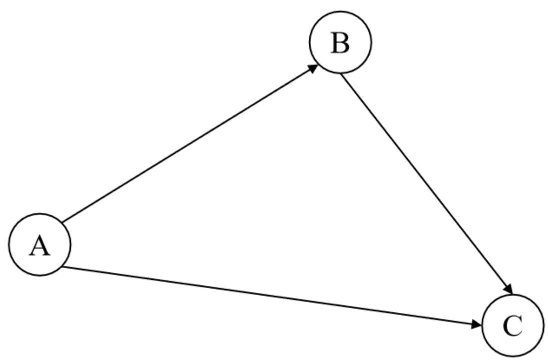

Figure 2. An example of a time-varying neural network.

Table 1. Description of the example.

Node	Edge	Time Window	Cost
A	AB	[0, 3)	1
		[3, +∞)	2
	AC	[0, 2)	2
		[3, +∞)	4
B	BC	[0, 3)	2
		[3, +∞)	3

Table 2. Steps of SNN as an example.

node	n_A				n_B		n_C
edge	E_{AB}		E_{AC}		E_{BC}		-
time window	D_{AB}^1	D_{AB}^2	D_{AC}^1	D_{AC}^2	D_{BC}^1	D_{BC}^2	-
(a) Running SI ($t=0$)							
S_i	$None$				$None$		$None$
F_i	$None$				$None$		$None$
P_{ij}	$[0]$		$[0]$		$None$		-
v_{ij}^r	$\{0:0\}$	$\{0:0\}$	$\{0:0\}$	$\{0:0\}$	$None$	$None$	-
(b) Running NA ($t=1$)							
S_i	$\{B:[[0,1]]\}$				$None$		$None$
F_i	$None$				$\{A:[[0,1]]\}$		$None$
P_{ij}	$None$		$[0]$		[1]		-
v_{ij}^r	$None$	$None$	$\{0:1\}$	$\{0:-2\}$	$\{1:0\}$	$\{1:0\}$	-
(c) Running NA ($t=2$)							
S_i	$\{B:[[0,1]],C:[[0,2]]\}$				$None$		$None$
F_i	$None$				$\{A:[[0,1]]\}$		$\{A:[[0,2]]\}$
P_{ij}	$None$		$None$		[1]		-
v_{ij}^r	$None$	$None$	$None$	$None$	$\{1:1\}$	$\{1:-1\}$	-
(d) Running AP							
paths	$[[A,C,2]]$						
(e) Running NA ($t=3$)							
S_i	$\{B:[[0,1]],C:[[0,2]]\}$				$\{C:[[1,3]]\}$		$None$
F_i	$None$				$\{A:[[0,1]]\}$		$\left\{\begin{array}{l}A:[[0,2]]\\B:[[1,3]]\end{array}\right\}$
P_{ij}	$[0]$		[0]		$None$		-
v_{ij}^r	$\{0:0\}$	$\{0:0\}$	$\{0:0\}$	$\{0:0\}$	$None$	$None$	-
(f) Running AP							

paths	$[[A, C, 2], [A, B, C, 3]]$

Furthermore, to ensure the data security of the cloud and users, four sets of keys for three encryption algorithms were configured. Initially, the cloud server holds the private key of RSA1. A user is authorized if he or she has obtained the private secret key of RSA1 and the public secret key of RSA2. The user constructs the encrypted index, as demonstrated in definition 3 for the graph, and outsources indexes to the cloud server.

The system interaction is as follows:

1. The user, say Bob, generates a unique secret key for AES K_{AES}, distinguished from those of other users to prevent leakage from others.
2. Bob→Cloud: Send K_{AES}, which is encrypted by Pk_{RSA2}
3. Bob→Cloud: Encrypt $(A,\ C,\ 0,\ 5,\ 2)$ as an encrypted query according to Definition 4, encrypt $(A,\ C,\ 0)$ with the AES key K_{AES} and encrypt 5 with the ORE key K_{ORE} and the AES key K_{AES}.
4. Cloud: Decrypt the query with K_{AES}, use $(A,\ C,\ 0)$ to find f matching encrypted indices, and compare the second part of the encrypted index with 5. If the former is not greater, the item meets the query.
5. Cloud→Bob: Query the result encrypted with K_{AES}.
6. Bob: Decrypt the information with K_{AES} and then with Sk_{RSA1} to obtain the query result $< A,\ C >: 1,\ < A,\ B,\ C >: 3$.

3.3. Complexity

Theorem 1 (Time Complexity of the SNN)**.** *A directed graph G with n nodes and m edges is given, where each edge has an average of k time windows. The SNN algorithm finds the result in* $O(n^2 \times k \times m \times f)$ *time for f paths.*

Proof of Theorem 1. The complexity of Algorithm 1 needs to be combined with the complexity of graph initialization, start node initialization, node activation, and path addition, i.e., Algorithms 2–7.

(1) Algorithm 2 focuses on traversing the edges to initialize the time windows on each edge. The complexity of graph initialization is $O(m \times k)$.

(2) Algorithm 3 involves traversing the time windows of the successor edge set of n_i. The time complexity is approximately $O(m \times k)$.

(3) Algorithm 4 involves judgement. When the judgement condition is true, the time complexity is $O(m \times k \times n)$, after considering Algorithm 6.

(4) Algorithm 5 is an iteration related to the number of nodes and edges. The time complexity is $O(m \times n)$.

(5) Algorithm 6 includes two side-by-side cyclic operations, one related to the number of time windows of the edges, and the other is the removal operation in the process processing which makes a call to Algorithm 3. The time complexity of Algorithm 6 is approximately $O(m \times k)$.

(6) The complexity of Algorithm 7 is $O(1)$ obviously.

Combining the calls of Algorithm 1 to other algorithms and the call relationships between the algorithms, the time complexity of the SNN can be calculated using the following expression:

$$O(n^2 \times k \times m \times f)$$

Theorem 1 is proven. □

3.4. Security

The design of the encrypted index incorporates three different cryptography techniques: RSA, AES, and ORE. Each of these three algorithms plays an important role in the field of cryptography, and together they provide a multi-layered security.

Assuming that the cloud provider is secure and trustworthy, i.e., it does not tamper with the data and returns the query results truthfully, it is assumed that the key distribution process is secure.

The security of RSA depends on the key length and the complexity of the factorization algorithm. As the computational power increases, the key length of RSA increases to maintain its security. Although quantum computing may pose a threat to the security of RSA, RSA keys of a sufficient length are still secure for the time being. RSA is used to make real data transparent to the cloud provider and to protect the user's independent secret key [26,27].

The design of AES consists of a multi-round encryption process, with each round including operations such as byte substitution, row shifting, column mixing, and round key addition. The security of AES has been extensively researched and empirically verified, and to date, no effective attack method has been found that can break AES-256 in a practicable amount of time. AES is used to prevent information leakage between users. Even if a user has access to another user's information, the data cannot be deciphered without the user's secret key [27,28].

ORE is designed to prevent the inference of sensitive information through sequential relationships, thus providing greater security than traditional OPE. ORE is secure, in that it hides the sequential information of the data, making it impossible to infer anything about the original data even when the encrypted data is sorted or a range query is performed. ORE is for path filtering in the cloud [29–31].

4. Experiments

This section presents the evaluation of the SNN algorithm through experiments on New York road instances.

To illustrate the efficiency and accuracy of the SNN algorithm, four random network topologies with 50 nodes, 100 nodes, 500 nodes, and 1000 nodes were considered. The time limits for the above topologies are 50, 200, 500, and 1000, respectively. All the algorithms in our experiment are implemented in Python. The experiments are conducted using a desktop PC sourced from Dell, headquartered in Round Rock, Texas, USA. The PC is equipped with an 11th Gen Intel(R) Core (TM) i7-11390H processor at 3.40 GHz and 16 GB of RAM.

The datasets used in our experiments are listed in Table 3.

Table 3. Datasets for Experiments.

Dataset	Nodes	Edges	Time Windows	T_u	Storage
Dataset 1	50	115	230	50	4 KB
Dataset 2	100	230	460	200	7 KB
Dataset 3	500	1244	2488	500	44 KB
Dataset 4	1000	2493	4986	1000	92 KB

Table 4 shows the query time statistics of existing algorithms before as Dijkstra [32], PCNN [33], and TDNN without the time-varying [25]. These algorithms have the same accuracy, i.e., they all find the shortest paths. The optimal algorithm TDNN was selected for comparison experiments with SNN.

Table 4. Comparison of existing algorithms.

Dataset	Dijkstra	PCNN	TDNN
Dataset 1	0.001	0.001	0.001
Dataset 2	0.004	0.003	0.003
Dataset 3	0.10	0.08	0.08
Dataset 4	0.26	0.31	0.24

Table 5 shows the efficiency comparison of TDNN and SNN with time-varying factors. As the size of datasets grows, both algorithms show a substantial increase in the time required. However, the TDNN algorithm takes significantly longer than SNN, and the time difference between the two algorithms is becoming greater. The SNN algorithm has better efficiency than TDNN with time-varying factors, and it can find multiple paths at once and contains the shortest path, which indicates that the accuracy of SNN is significantly better than that of others (e.g., TDNN).

Table 5. Efficiency Comparison of SNN, TDNN, Connor.

Dataset	SNN	TDNN	Connor
Dataset 1	0.03	0.08	0.05
Dataset 2	0.12	0.53	0.1
Dataset 3	3.13	64.10	0.8
Dataset 4	18.72	225.75	8.9

Accuracy comparison of SNN, TDNN, and Connor is presented in the form of line chart in Figure 3. In addition, the SNN is significantly less efficient compared to the Connor. However, the Connor framework has the obvious limitation that it sacrifices accuracy [24]. This is less applicable in multiple scenarios such as map services. This leads to its low applicability in scenarios where accuracy is required.

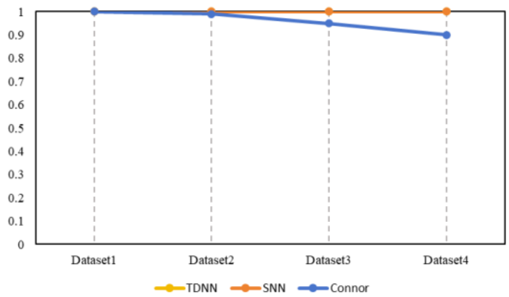

Figure 3. Accuracy Comparison of SNN, TDNN, and Connor.

Tables 6 and 7 show the time statistics for different numbers of edges, nodes, time windows and T_u, which are presented in the form of a line chart in Figure 4. For practical reasons, the number of time windows will not change exponentially. It can be easily seen that the time consumption of SNN is positively correlated with the number of nodes, edges, time windows, and the upper limit of time.

Table 6. SNN experimental results (k = 2).

Dataset	T_u=50s	T_u=200s	T_u=500s	T_u=1000s
Dataset 1	0.04	0.04	0.03	0.03
Dataset 2	0.04	0.17	0.16	0.16
Dataset 3	0.06	0.60	4.36	4.10
Dataset 4	0.08	0.88	5.34	26.41

Table 7. SNN experimental results (k = 4).

Dataset	T_u=50s	T_u=200s	T_u=200s	T_u=200s
Dataset 1	0.04	0.04	0.04	0.04
Dataset 2	0.05	0.18	0.15	0.15
Dataset 3	0.07	0.4	3.16	3.32
Dataset 4	0.10	0.62	4.86	29.18

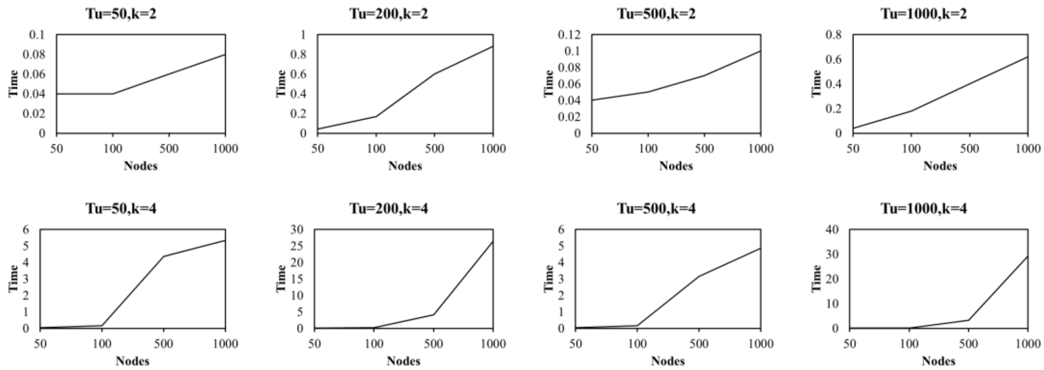

Figure 4. Comprehensive comparison of SNN experimental results.

In summary, the SNN algorithm shows more comprehensive advantages in terms of efficiency, accuracy, etc. The SNN algorithm is of great significance for solving time-varying path query problems with privacy protection.

However, the efficiency decreases significantly as the scale of the graph increases according to the experimental results. To adapt to more demand scenarios, such as large-scale graph applications, the hardware configuration can be improved or more research on optimization and iteration needs to be performed.

5. Conclusions

In this paper, the SNN algorithm was proposed for solving TVPQ problems with privacy protection. This approach is highly important for road planning, network routing, project scheduling, and other issues in data outsourcing scenarios. The most prominent advantage is that it can find multiple paths at once, including the shortest paths, which other algorithms cannot find. Additionally, an encrypted index for privacy protection has been designed. Experiments with New York road instances demonstrated the efficiency and accuracy of the SNN algorithm.

In future work, further studies of path planning algorithms with privacy protection can be conducted, such as optimization algorithms for large-scale encrypted graphs.

Funding: This research received no external funding.

Data Availability Statement: Publicly available datasets were analyzed in this study. This data can be found here: https://www.diag.uniroma1.it/challenge9/download.shtml (accessed on 15 September 2022).

Conflicts of Interest: The author declares no conflicts of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript; or in the decision to publish the results.

Abbreviations

The following abbreviations are used in this manuscript:

Symbols	Explanation
G	The graph
n_s	The start node
n_t	The target node
T_s	The departure time
T_e	The arrival time
Sk_{RSA1}	The private key of RSA-1
Pk_{RSA1}	The public key of RSA-1
Sk_{RSA2}	The private key of RSA-2
Pk_{RSA2}	The public key of RSA-2
K_{AES}	The key of AES
K_{ORE}	The key of ORE
T_u	The upper limit of the arrival time
p_{st}	One path from n_s to n_t
P_{st}	The paths set from n_s to n_t
l_{st}	The length of P_{st}
E_{ij}	The edge from n_i to n_j
D_{ij}	The time window of the edge from n_i to n_j
D_{ij}^r	The rth time window of the edge from n_i to n_j
l_{ij}^r	The lower boundary of D_{ij}^r
u_{ij}^r	The upper boundary of D_{ij}^r
c_{ij}	The cost of E_{ij}
c_{ij}^r	The cost of D_{ij}^r
v_{ij}^r	The cost of D_{ij}^r in real time
N_{ij}^P	The predecessor n_i of n_j
N_{ij}^S	The successor n_j of node i
E_i^P	The predecessor edge set of n_i
E_i^S	The successor edge set of n_i
N_s	The number of nodes
f	The number of paths queried
k	The number of time windows of each edge
F_i	The father set of n_i
P_{ij}	The arrival time set of n_j with that swell has not spread to next node currently
S_i	The son set of n_i

References

1. References Ge, X.; Yu, J.; Zhang, H.; Bai, J.; Fan, J.; Xiong, N.N. SPPS: A search pattern privacy system for approximate shortest distance query of encrypted graphs in iiot. *IEEE Trans. Syst. Man Cybern. Syst.* **2022**, *52*, 136–150.
2. Zhou, Y.; Lu, Y.; Lv, L. Grid-based non-uniform probabilistic roadmap-based agv path planning in narrow passages and complex environments. *Electronics* **2024**, *13*, 225–240.
3. Huang, W.; Gao, L. A time wave neural network framework for solving time-dependent project scheduling problems. *IEEE Trans. Neural Netw. Learn. Syst.* **2020**, *31*, 274–283.
4. Cao, N.; Yang, Z.; Wang, C.; Ren, K.; Lou, W. Privacy-preserving query over encrypted graph-structured data in cloud computing. In Proceedings of the 2011 31st International Conference on Distributed Computing Systems, Minneapolis, MN, USA, 20–24 June 2011; pp. 105–117.
5. Memon, I.; Arain, Q.A. Dynamic path privacy protection framework for continuous query service over road networks. *World Wide Web* **2017**, *20*, 639–672.
6. Shang, H.; Zhang, Y.; Lin, X.; Yu, J.X. Taming verification hardness: An efficient algorithm for testing subgraph isomorphism. *Proc. Vldb Endow.* **2008**, *1*, 364–375.
7. Gouda, K.; Hassaan, M. Compressed feature-based filtering and verification approach for subgraph search. In Proceedings of the EDBT'13: Proceedings of the 16th International Conference on Extending Database Technology, Genoa, Italy, 18–22 March 2013; pp. 201–213.
8. Lin, W.; Xiao, X.; Cheng, J.; Bhowmick, S.S. Efficient algorithms for generalized subgraph query processing. In Proceedings of the CIKM'12: Proceedings of the 21st ACM international conference on Information and knowledge management, Maui, HI, USA, 29 October–2 November 2012; Association for Computing Machinery: New York, NY, USA, 2012; pp. 325–352.
9. Meng, X.; Kamara, S.; Nissim, K.; Kollios, G.N. GRECS: Graph encryption for approximate shortest distance queries. In Proceedings of the CCS'15: Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, USA, 12–16 October 2015; Association for Computing Machinery: New York, NY, USA, 2015; pp. 25–42.
10. Xie, P.; Xing, E. CryptGraph: Privacy Preserving Graph Analytics on Encrypted Graph. *arXiv* **2014**, arXiv:1409.5021.
11. Cooke, K.L.; Halsey, E. The shortest route through a network with time-dependent internodal transit times. *J. Math. Anal. Appl.* **1966**, *14*, 493–498.
12. Frigioni, D.; Marchetti-Spaccamela, A.; Nanni, U. Fully dynamic output bounded single source shortest path problem. In Proceedings of the SODA'96: Proceedings of the Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, Atlanta, GA, USA, 28–30 January *1996*; pp. 212–221.
13. Ramalingam, G.; Reps, T. An incremental algorithm for a generalization of the shortest-path problem. *J. Algorithms* **1996**, *21*, 267–305.
14. King, V. Fully dynamic algorithms for maintaining all-pairs shortest paths and transitive closure in digraphs. In Proceedings of the 40th Annual Symposium on Foundations of Computer Science, New York, NY, USA, 17–19 October 1999; pp. 81–89.
15. Demetrescu, C.; Italiano, G. A new approach to dynamic all pairs shortest paths. *J. ACM* **2004**, *51*, 968–992.
16. Liu, C.; Zhu, L.; He, X.; Chen, J. Enabling privacy-preserving shortest distance queries on encrypted graph data. *IEEE Trans. Dependable Secur. Comput.* **2021**, *18*, 192–204.
17. Ghosh, E.; Kamara, S.; Tamassia, R. Efficient graph encryption scheme for shortest path queries. In Proceedings of the ASIA CCS'21: ACM Asia Conference on Computer and Communications Security, Hong Kong, China, 7–11 June 2021; pp. 31–43.
18. Sun, F.; Yu, J.; Ge, X.; Yang, M.; Kong, F. Constrained top-k nearest fuzzy keyword queries on encrypted graph in road network. *Comput. Secur.* **2021**, *111*, 430–442.
19. Wu, B.; Chen, X.; Wu, Z.; Zhao, Z.; Mei, Z.; Zhang, C. Privacy-guarding optimal route finding with support for semantic search on encrypted graph in cloud computing scenario. *Wirel. Commun. Mob. Comput.* **2021**, *doi 10.1155/2021/6617959*.
20. Zhang, D.; Liu, Y.; Liu, A.; Mao, X.; Li, Q. Efficient path query processing through cloud-based mapping services. *IEEE Access* **2017**, *5*, 12963–12973.
21. Huang, W.; Sun, M.; Zhu, L.; Oh, S.; Pedrycz, W. Deep fuzzy min-max neural network: Analysis and design. *IEEE Trans. Neural Netw. Learn. Syst.* **2022**, doi: 10.1109/TNNLS.2022.3226040.
22. Huang, W.; Wang, Y.; Zhu, L. A time impulse neural network framework for solving the minimum path pair problems of the time-varying network. *IEEE Trans. Knowl. Data Eng.* **2023**, *35*, 7681–7692.
23. Zhang, C.; Luo, X.; Liang, J.; Liu, X.; Zhu, L.; Guo, S. POTA: Privacy-preserving online multi-task assignment with path planning. *IEEE Trans. Mob. Comput.* **2023**, *4*, 1–13.
24. Shen, M.; Ma, B.; Zhu, L.; Mijumbi, R.; Du, X.; Hu, J. Cloud-based approximate constrained shortest distance queries over encrypted graphs with privacy protection. *IEEE Trans. Inf. Forensics Secur.* **2018**, *13*, 940–953.
25. Huang, W.; Wang, J.; Wang, W. A time-delay neural network for solving time-dependent shortest path problem. *Neural Netw.* **2017**, *90*, 21–28.
26. Tahat, N.; Tahat, A.A.; Abu-Dalu, M. A new RSA public key encryption scheme with chaotic maps. *Int. J. Electr. Comput. Eng.* **2020**, *10*, 1430–1437.
27. Huang, X.; Wang, W. A novel and efficient design for an rsa cryptosystem with a very large key size. *IEEE Trans. Circuits Syst.* **2015**, *62*, 972–976.

28. Masoumi, M. Novel hybrid cmos/memristor implementation of the aes algorithm robust against differential power analysis attack. *IEEE Trans. Circuits Syst.* **2020**, *67*, 1314–1318.
29. Peyrin, T. *Practical Order-Revealing Encryption with Limited Leakage*; Springer: Berlin/Heidelberg, Germany, 2016; Volume 9783, pp. 474–493.
30. Zhang, C.; Zhao, M.; Liang, J.; Fan, Q.; Zhu, L.; Guo, S. NANO: Cryptographic enforcement of readability and editability governance in blockchain databases. *IEEE Trans. Dependable Secur. Comput.* **2023**, *4*, 1–14.
31. Hu, C.; Zhang, C.; Lei, D.; Wu, T.; Liu, X.; Zhu, L. Achieving privacy-preserving and verifiable support vector machine training in the cloud. *IEEE Trans. Inf. Forensics Secur.* **2023**, *18*, 3476–3491.
32. Zhu, D.; Sun, J. A new algorithm based on dijkstra for vehicle path planning considering intersection attribute. *IEEE Access* **2021**, *9*, 19761–19775.
33. Sang, Y.; Lv, J.; Qu, H.; Yi, Z. Shortest path computation using pulse-coupled neural networks with restricted autowave. *Knowl.-Based Syst.* **2016**, *114*, 1–11.