Journal of Web Engineering, Vol. 15, No.1&2 (2016) 029-044
© Rinton Press

(K, P)-SHORTEST PATH ALGORITHM IN THE CLOUD MAINTAINING NEIGHBORHOOD PRIVACY

SHYUE-LIANG WANG JIA-WEI CHEN I-HSIEN TING TZUNG-PEI HONG

National University of Kaohsiung, Taiwan

slwang@nuk.edu.tw sk413025@gmail.com iting@nuk.edu.tw tphong@nuk.edu.tw

Received January 27, 2015
Revised October 4, 2015

Privacy-preserving computation has recently attracted much attention in areas of transaction, social networking, location-based, and mobile services. The inexpensive storage and efficient computation of cloud computing technology is expected to further escalate these services to a higher and wider level, without compromising the breaches of sensitive information. In this work, we study the shortest path distance computing in the cloud while preserving two types of privacy in the same time: k-neighborhood privacy and sensitive path privacy. We propose a new privacy model called (k, p)-shortest path neighborhood privacy, which is an extension of [19] and more flexible than 1-neighborhood-d-radius model [6]. We also develop an efficient four-step shortest distance computation scheme to achieve (k, p)-shortest path neighborhood privacy on p outsourced servers in the cloud, which combines the construction of k-skip shortest path sub-graphs, sensitive vertex adjustment, vertex hierarchy labeling and bottom-up partitioning techniques. Numerical experiments show that the proposed approach is more efficient than prior model of constructing the 1-neighborhood privacy graph and also requires less querying time.

Keywords: privacy preservation, k-neighborhood privacy, sensitive path privacy, shortest path, k-skip.

Communicated by: M. Gaedke & Q. Li

1 Introduction

Preserving privacy explicitly or implicitly from published data has been studied extensively for more than a decade. Many anonymizaton techniques have been proposed to protect user identity, relationships, locations, etc., on relational, set, text, graph, and location data, with increasing interests in cloud and mobile data.

Privacy preserving network publishing (PPNP), which deals with graph data, typically faces four common types of privacy breaches: identity disclosure, link disclosure, attribute disclosure, and edge weight disclosure [4, 8, 11, 12, 23, 24]. Identity disclosure refers to the threat of re-identification of nodes in a graph [12, 24]. For example, user profiles (such as photos, birth date, residence, interests and friend links) can be used to estimate personal identity information such as social security number. Link disclosure refers to the threat of identifying the relationship between nodes [8], for example, identifying the friendship between Ada and Bob. Attribute disclosure refers to the threat of identifying the attributes of a node or a link [23]. Edge weight disclosure refers to the threat of identifying the weights of a link or a path [11, 13, 20, 21].

Many anonymization techniques have been proposed for PPNP. Most of existing works focus on certain structural anonymization, such as k-degree, k-neighborhood, k-isomorphism, k-security, etc., by using least amount of modification on the original graph. However, the anonymized graphs produced by these privacy protections generally do not maintain the statistical and theoretical characteristics of the original graph. As such, there is no guarantee of the degree of similarity or preservation of shortest distances between anonymized and original graphs. In addition, due to the complexity of graph structure and real graph datasets are growing rapidly in size, computation is time-consuming and demanding.

Cloud computing creates a fluid pool of resources across servers that enable users (client) to access stored data (outsourced server) and applications on an as-needed basis. It is therefore desirable to employ cloud computing to manage large graphs and process complex operations efficiently. To speed up the running time of shortest path discovery, there are some works to exploit pre-computed indices. Unfortunately, we cannot simply outsource these indices as they also disclose sensitive information of the graph. On the other hand, there are some work proposing to protect sensitive data and verification of query results in the outsourced servers [7, 14, 22]. However, they do not consider protecting the sensitive information on the original graph. As such, our goal is to search for a computational scheme which can compute shortest path distances while protecting sensitive information in the original graph.

To preserve neighborhood privacy for computing shortest path distance in the cloud, (Gao et al., 2011) [6] proposed the one-neighborhood-d-radius privacy in the cloud server model. A given graph is transformed to a link graph on client side and a number of outsourced graphs on the cloud server side. The shortest path distance can be answered using both types of graphs together, with a greedy based graph generation scheme. However, their approach requires calculation of all-pairs of shortest paths for all vertices first and then builds the outsourced graphs randomly. In addition, sensitive shortest paths might appear on the same server entirely.

In this work, we introduce a novel privacy model called (k, p)-shortest path neighborhood privacy, which is an extension of [19] and more flexible than 1-neighborhood-d-radius model [6], and propose an efficient shortest distance computation scheme for p outsourced servers in the cloud. Adopting k-skip shortest path sub-graphs, sensitive vertex adjustment, vertex hierarchy labeling and bottom-up partitioning techniques, the proposed approach not only subsumes one-neighborhood privacy but also provides efficient query processing for sensitive shortest paths. Numerical experiments demonstrating the characteristics of proposed approach and comparison with prior approaches are presented.

The rest of the paper is organized as follows. Section 2 reviews related works. Section 3 gives the problem description. Section 4 describes the proposed algorithms. Section 5 reports the numerical experiments. Section 6 presents the conclusion and future works.

2 Related Work

Computing fastest routes in road networks from a given source to a given target location is one of the real-world applications, as could be found in Google Map or Bing Map services, not to mention it also is a classical problem of computer science. Numerous applications, e.g., social networks, biological and chemical pathways, transportation networks can be modeled as graph structure data and need to solve a huge number of shortest path queries. In principle, we could use Dijkstra's algorithm for calculating all-pairs shortest paths. However, it becomes in-efficient and time-consuming when the complexity of structural connectivity and graph size grow. Moreover, real graph datasets are growing rapidly in size and making the attainment of high efficiency even harder.

Basically there are three categories of approaches to accelerate the calculation of shortest paths on large graphs: preprocessing, efficient search, and distributed processing. The preprocessing approach usually generates auxiliary data that can be used to speed up all subsequent queries, with a moderate amount of space required. For example, the Pre-computed Cluster Distances technique partitions a graph into clusters and the shortest connection between any pair of clusters is pre-computed. Then during a query, upper and lower bounds can be derived that can be used to prune the search. The efficient search approach includes bidirectional search, goal directed search, separators, reach-based routing, and heuristics [16]. For example, the bidirectional search allows simultaneously searches forward from the source and backward from the target until the search frontiers meet. For the distributed processing approach, there are abundant studies not only for general structure but also for classes of network topologies [3, 10].

With the advent of cloud computing, it becomes desirable to utilize this distributed facility to efficiently calculate the shortest paths and shortest distances in large graphs without compromising their sensitive information [1, 16, 17]. Some studies have been investigating secured outsourced service in the cloud environment [6, 7, 14, 22]. In particular, Yiu studies the verification issue in outsourced graphs for the shortest path discovery. The graph data are outsourced along with verification objects, and the client side will validate the correctness of query results using the objects. However, the work does not consider the protection of the sensitive information of the original graph.

In order to protect the privacy of these sensitive information (sensitive edges and shortest paths), four types of work have been proposed. The first type of work tries to preserve the shortest path characteristic between pairs of source and destination vertices. In other words, the shortest path remains to be the shortest path after all edge weights are minimally modified [4, 13]. To preserve the shortest paths between pairs of vertices, Gaussian randomization perturbation and greedy perturbation techniques that minimally modify the edge weights without adding or deleting any vertices and edges have been proposed in [13]. The anonymized graph thus preserves the shortest path but all edge weights are perturbed. A linear programming abstract model that can preserve linear properties of edge weights (including shortest paths) after anonymization is presented in [4].

The second type of work tries to preserve the privacy of edges emitting from a given vertex such as k-anonymous weight privacy [11]. In another word, the difference between the weights of the edges emitting from a given vertex are within a predefined parameter. The k-anonymous weight privacy is defined as: the edge $(i \rightarrow j)$ is k-anonymous if and only if there exist at least k edges in $\Phi(i)$ whose weights $w_{i,l}$, $l = 1, ..., c$, and $c \geq k$, satisfy $\| w_{i,j} - w_{i,l} \| \leqq \mu$. Here, μ is a predefined positive parameter to control the degree of privacy and $\Phi(i)$ is the adjacent edge set in which all edges come from the i-th vertex.

The third type of work tries to anonymize the shortest path between pairs of source and destination vertices. In another word, there will be at least k shortest paths after edge weights are minimally modified such as k-shortest path privacy. Several heuristic algorithms were proposed to minimally perturb edge weights so that there become k shortest paths in a graph. In addition, to deal with degree attacks on the anonymized k shortest paths, a new concept called (k1, k2)-shortest path privacy was proposed in [21].

The fourth type of work studies the shortest distance computing in the cloud which aims at preventing outsourced graphs from one neighborhood attacks [6, 25]. It basically transforms an original graph into a link graph that is kept locally in the client and a set of outsourced graphs in the servers in the cloud. In another word, when an adversary compromises an outsourced server, he/she still cannot calculate the shortest path or shortest distance between neighboring nodes. However, their approach requires calculation of all-pairs of shortest paths for all vertices first and then builds the outsourced graphs randomly.

Our work differs from the first type of work [4, 13] in which their shortest path distance has been changed due to perturbation of all edge weights. Our work is also different from second type of work [11], in which they tried to keep the edge weights emitting from designated sensitive nodes indistinguishable. Our work differs from the third type of work in which it tries to obfuscate k paths so that they all have same path length as the shortest path. Our work basically extends the fourth type of work for stronger and more flexible privacy in calculating the shortest path distance in the cloud.

3 Problem Description

In this section, we first review the one-neighborhood-d-radius privacy model proposed by (Gao et al., 2011) [6] and point out two issues of the model. We then propose the (k, p)-shortest path neighborhood privacy model which alleviates the problems. Basically, the one-neighborhood-d-radius privacy model is to protect the sensitive local neighborhood information. The information of how an individual vertex links to its neighbors and the edge weights for these links is sensitive and need to be protected. In particular, it focuses on 1-neighborhood attacks and vertices that are very closed to each other, within d threshold radius.

Given a fixed number of p outsourced servers, the objective of one-neighborhood-d-radius privacy model is to transform a large weighted un-directed graph into p sub-graphs that are to be stored in outsourced servers, so that (1) shortest distance between any two vertices can be efficiently computed in the cloud, (2) one-neighborhood-d-radius neighborhood attacks can be prevented on each server, and (3) overhead on the client side can be minimized.

For example, given an original graph shown at the bottom of Figure 1, and assuming the number of server in the cloud is p = 2. It shows two transformed sub-graphs on the top and the link graph in the middle, which is expressed as dashed lines here. The link graph will be saved in the client side (data owner) and the two sub graphs will be stored in two servers in the cloud respectively. Note that not all links are shown here to keep the figure clear and ease to read.

For a query to calculate the shortest path distance on neighboring vertices such as A and B, vertex A is on G_1 and vertex B is on G_2. The search then has to start from original graph, e.g. vertex A. Since there are two vertices {B,L} on graph G_2 that are linked to vertex A, there are two ways to calculate the shortest distance: the distance $d(A,B) = |\{A \rightarrow B\}| = 1$ and the distance $d'(A,B) = |\{A \rightarrow L \rightarrow B\}| = 1+2 = 3$. The shortest path distance is the minimum of the two, which is one.

For a query on non-neighboring vertices such as E and J, both vertices are not on graph G_1 or graph G_2 (they are on the original graph only). However, vertex E is linked to vertices C and F on graph G_1 and vertex J is linked to vertices H and K on graph G_1. The shortest path distance between E and J can be calculated as $d(E,J) = |\{E \rightarrow F \rightarrow H \rightarrow J\}| = 1+4+4 = 9$.

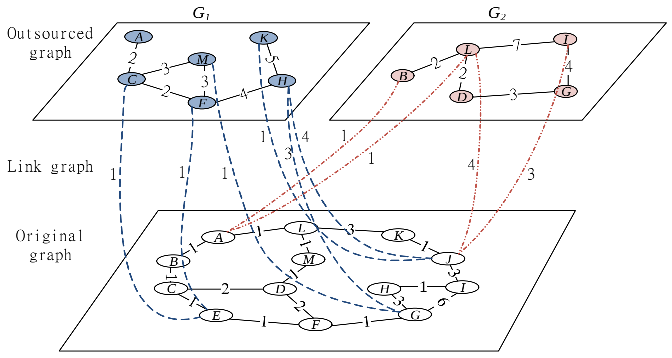


Figure 1. Link graph and outsourced graphs

However, there are two basic short comings in the (Gao et al., 2011) approach. The first problem is that to obtain the transformed sub graphs, the scheme requires to calculate all-pairs shortest paths first and then randomly select a set of shortest paths to form a sub-graph, which is to be stored in one server. Random selection of shortest path may cause overlaps between sub graphs. Calculating all shortest path may consume a lot of computation effort. In addition, if all shortest paths have been obtained, their shortest paths and distances could be saved directly (either on servers or on client), instead of saving sub-graphs which required more calculation to obtain results already known. Another problem is that certain sensitive paths appear on the same sub-graph, in which, the privacy of the shortest path might be breached by the cloud server. For example, for path from vertex C to vertex H, d(C, H) = 6, the whole path appears on graph G_1, and the privacy is not preserved (assuming outsourced server G_1 is compromised by adversary). To preserve the privacy of sensitive paths, our approach is to separate and save the source and destination vertices in different sub-graphs. As such, we propose a new flexible (k, p)-shortest path neighborhood privacy model in the cloud environment, in which, given a fixed number of p outsourced servers, the objective is to transform a large weighted un-directed graph into p sub-graphs that are to be stored in outsourced servers, so that (1) shortest distance between any two vertices can be efficiently computed in the cloud, (2) k neighborhood attack can be prevented, (3) privacy (shortest path distance) of user specified sensitive paths can be preserved.

To avoid calculating all-pairs shortest path, we adopt k-skip shortest paths [17] and vertex labeling hierarchy techniques [5]. For example, Figure 2 shows an original graph at the bottom, a 2-skip graph in the middle and two sub-graphs on the top to be stored in the cloud.

The middle figure in Figure 2 is a 2-skip shortest path graph generated from the original graph at the bottom. All-pairs shortest path calculation is not required here. The two sub-graphs on the top can be obtained by the vertex labeling and bottom-up partitioning techniques efficiently. Notice that the proposed k-skip graph is more flexible and general than one-neighborhood-d-radius privacy graph. Given two vertices on a sub graph in the cloud server, there may be one hidden neighboring node for one-neighborhood privacy graph. However, for proposed k-skip approach, depending on k, there may be zero to (k-1) hidden neighbors between two vertices on a sub-graph, which is more flexible than one-neighborhood privacy.

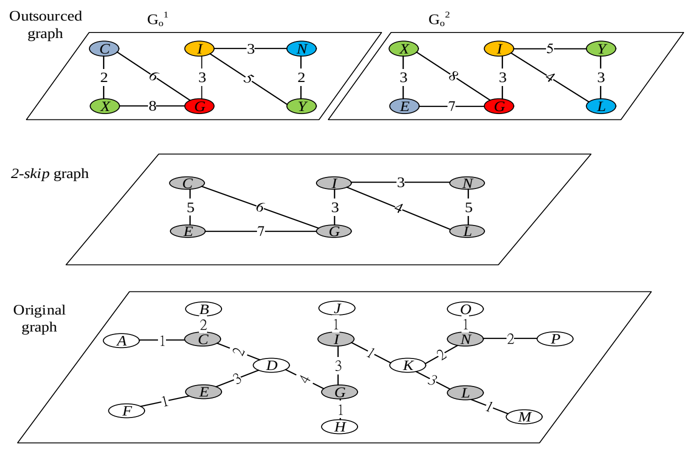
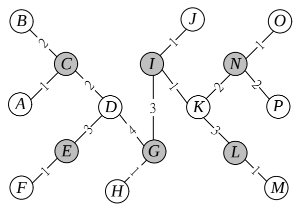


Figure 2. 2-skip graph and outsourced graphs

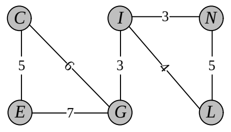
4 Proposed Algorithm

In this section, we present an efficient four-step scheme to achieve k neighborhood privacy and sensitive path privacy in computing shortest path distance on p outsourced servers in the cloud. The four steps are: (1) building k-skip sub graph, (2) adjusting vertices on sensitive paths, (3) building vertex labeling hierarchy, and (4) bottom-up partitioning of sub graphs, as shown in Figures 3, 4, and 5. The sub graphs built will be stored in the p outsourced graphs respectively. The shortest path distance query can be answered using outsourced graphs and link graph.

4.1 Construction of k-skip graph



(a) Original graph and 2-skip cover



(b) 2-skip graph

Figure 3. Construction of 2-skip graph

For the first step, building k-skip sub graph, we adopt the k-skip graph which was introduced in the context of spatial network databases [17]. The k-skip graph was designed to answer k-skip queries significantly faster than finding the original shortest paths in their entirety. The objective here is to avoid calculating all-pairs shortest paths to find 1-neighborhood shortest path candidates, which was used in prior model [6]. In addition, we can extend 1-neighborhood privacy to [0, k-1] neighborhood privacy. To obtain a k-skip graph, a k-skip cover must be constructed first. All vertices are sorted in descending order according to the node degree. The vertex with the highest node degree is placed in the k-skip cover. Subsequent vertices are added to the cover if it cannot be reached within k steps in the cover. For example, vertices {C, E, G, I, L, N} in Figure 3(a) form the 2-skip cover for the graph and Figure 3(b) shows the final 2-skip graph. For vertex C, there are two vertices E and G that can be reached in two steps, and the final distances are five and six respectively.

4.2 Adjusting Vertices on Sensitive Paths

For the second step, adjusting vertices on sensitive paths, the objective is to adjust the source and destination vertices so that they will not appear on the same outsourced graph. For example, assuming that there are two sensitive paths {C→E} and {N→L} to be preserved, where the source vertices are C, N and the destination vertices are E and L respectively as shown in Figure 3(a). However, the source and destination vertices become neighboring vertices in the 2-skip graph. As such, we need to add extra nodes, X and Y, as shown in Figure 4(a), where node X is added between node C and node E, node Y is added between node N and node L. To achieve this, we propose to apply the concept of maximum independent set (MIS) in graph theory. An independent set of a graph is a subset of vertices such that no two of which are adjacent. A maximum independent set of a graph is an independent set of largest possible size for a give graph. As such, for any two nodes in a maximum independent set, we can always add a new node between them. The distance between the two nodes will be the sum of the two new edges connecting to the new node. This serves the purpose of separating the source and destination vertices into two outsourced servers. On the other hand, we can always add a new edge between any two nodes in the maximum independent set. Setting the new edge weight as the shortest path distance, it would speed up the calculation of longer shortest path distance if these two vertices are on the path of the queried shortest path. This feature will be used in the second algorithm in the construction of vertex hierarchy. The algorithm for adjusting the vertices on sensitive paths is shown as follows.

Algorithm 1 Adjusting Vertices on Sensitive Paths

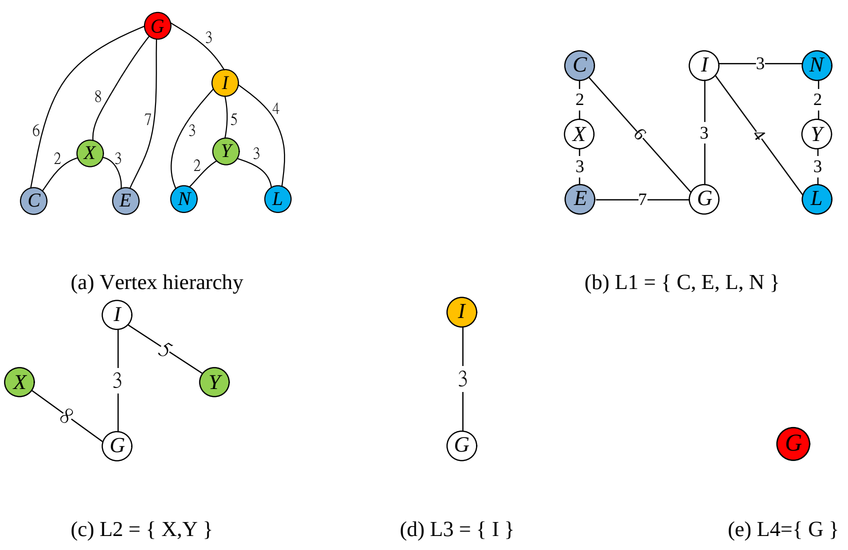
Input Gk : k-skip anonymous graph, SNP: Sensitive Node Pairs

Output MIS: Maximum Independent Set, Bottom: nodes in layer one, Gk: Adjusted k-skip anonymous graph

```
1. initialize Queue = {All vertices in Gk}, Bottom = { start = ∅, dest = ∅, other = ∅ }, MIS = SNP
2. for node s, t ∈ SNP
3.       insert s into start , insert t into dest
4. for node u ∈ MIS
5.       for node v that adjacent to u
```

```
6.            mark v as don't select
7.              if  v $\in$ MIS  then
8.                  insert  new node w  into e(u, v) on graph Gk
9.                  add the appropriate weight to e(u, w) and e(v,w)
10.  for node m $\in$ Queue
11.       if m has not been marked as don't select  then
12.             insert  m  into MIS
13.             for node n that is adjacent to m
14.                  mark n as don't select
15.  for node u $\in$ MIS
16.       if u $\notin$ SNP then
17.             add  u  to  other
28.  return  MIS, Bottom, Gk
```

4.3 Construction of Vertex Hierarchy



(a) Vertex hierarchy

(b) L1 = { C, E, L, N }

(c) L2 = { X,Y }

(d) L3 = { I }

(e) L4={ G }

Figure 4. Construction of the vertex hierarchy

For the third step, building vertex labeling hierarchy is to add shortest distances between levels of vertices in a hierarchical manner. The idea is to control the sensitive node pairs in the lowest hierarchy level, so that we can spread the source and destination nodes to different outsourced graphs. Utilizing the concept of independent set, our technique starts with selecting a vertex with lowest node degree in the graph, deleting the vertex and its direct edges, connecting its direct neighboring vertices and repeats the process. The selected vertices form an independent set. For example, it starts with selecting independent set {C, E, N, L} from graph Gk, as shown in Figure 4(b). Figure 4(c) shows a reduced graph after deleting the independent set of vertices. For the given example, the graph is reconstructed into a 4-level hierarchy.

The algorithm for constructing such vertex hierarchy is given as following.

Algorithm 2 Constructing Vertex Hierarchy

Input Gk :k-skip anonymous graph, MISadj : maximum independent set from algorithm one

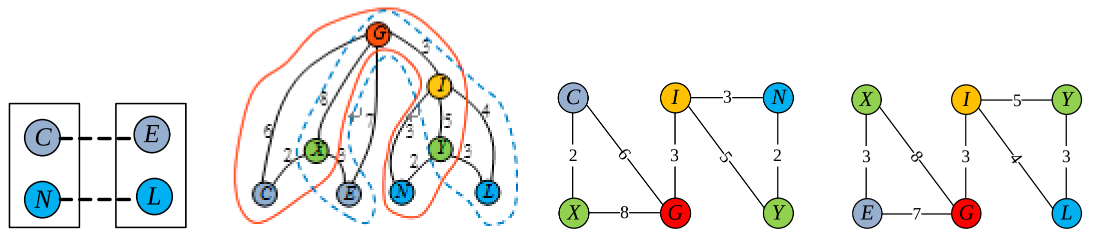
Output VH : vertex hierarchy

```
1. initialize i = 1, Li = VH = Gk
2. while  Li  not empty
3.      if  i = 1  then
4.           MIS = MISadj
5.      else
6.           MIS = calculate the independent set from Li
7.      for v ∈ MIS
8.           remove u, w  that are adjacent to v on Li
9.           for e(u, v) and e(u, w) in Li
10.               if  e(v, w) exist in Li  then
11.                    if  w(u, v) + w(u, w) < d(v, w) then
12.                         d(v, w) = w(u, v) + w(u, w)
13.                         add e(v, w) with weight d(v, w) to  VH
14.               else
15.                    d(v, w) = w(u, v) + w(u, w)
16.                    add   e(v, w) with weight d(v, w) to Li , VH
17.      Li+1 = Li  ,  i=i+1
18. return  VH
```

4.4 Bottom-up partitioning

For the fourth step (bottom-up partitioning of sub graphs), based on layers of vertex hierarchy, the k-skip graph is partitioned in a bottom-up manner into p sub graphs (outsourced graphs) to be stored in p cloud servers. The process starts with partitioning the level one independent set into p subsets. In fact, the partition starts with partitioning source and destination vertices of sensitive paths into p cloud servers. For example, Figure 5(a) shows the starting subgroups for case of two cloud servers (p=2).

Figures 5(b), 5(c) and 5(d) demonstrate vertex hierarchy partition and the final two partitions, as current example has only two vertices in the level one independent set. The algorithm starts with each vertex in an initial partition. It then traverses upward through the vertex hierarchy graph to reach its root node. Combining all nodes and paths they passed to form an outsourced graph.



(a) p=2 partition (b) Hierarchy partition (c) Outsourced graph G01 (d) Outsourced graph G02

Figure 5. Bottom-up partition

Algorithm 3 Bottom-up partitioning

Input VH: vertex hierarchy, p: number of outsourced graphs

Output OSG: outsource graphs

```
1. initialize BOX with p empty box (partition);
2. assign all sensitive source vertices evenly to the first [p/2] box,
3. assign all sensitive destination vertices evenly to the rest of box,
4. assign all layer one vertices evenly to all p box,
5. for box in BOX
6.      initialize G' = empty graph, LayerNodes = box
7.      while | LayerNodes | > 0
8.           UpperNodes = ∅
9.           for vl in LayerNodes
10.               level of vl in G' = level of vl in VH
11.               for each neighbor node vn from node vl
12.                    if level of vn > level of vl
13.                         add edge e(vl, vn) to graph G'
14.                         if vn ∉ UpperNodes then add vn to UpperNodes
15.           LayerNodes = UpperNodes
16.     add G' to OSG
```

5 Numerical Experiments

In this section, we present our experimental set up, the characteristics of data sets we perform experiments on, the outsourced graph construction times, the shortest path distance query times, and comparison with other techniques.

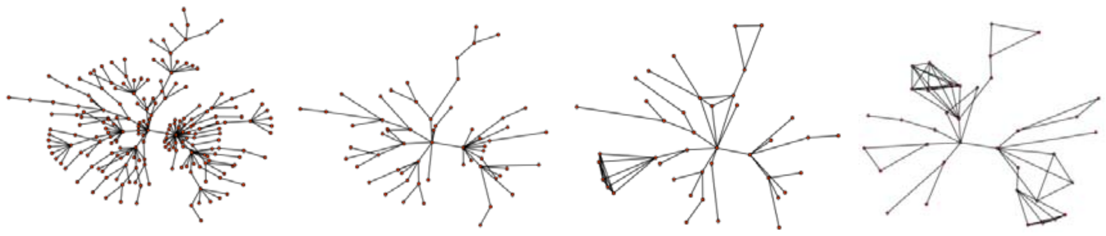
To simulate the cloud environment, we build a one-master, 5-slaves experimental distributed system. The master machine works as local client and the five slaves work as outsourced servers in the cloud. The six virtual machines are built on three physical machines which include two 32-bit Pentium(R) dual-core CPU 3.20 GHz machine with 4 GB memory running windows 7 and one 32-bit Celeron(R) CPU 1.80 GHz machine with 2 GB machine running windows 7. This is due to available physical machines are limited. The machines are built by using Virtual Box [18] under Linux Ubuntu operating system. The proposed algorithms are implemented in Python and IPython.parallel package [9].

We run simulations on synthetic datasets generated by Networkx library [15] graph generator. Three Power_law_cluster_graphs were generated with 3,000 (dp3k), 6,000 (dp3k), and 9,000 (dp3k) nodes respectively. The number of random edges added to each node is set as one and the probability of adding a triangle after adding a random edge is set as 0.01. The numbers of edges are 2,999, 5,999, 8,999, and the average numbers of edges of all shortest paths are 7.39, 8.32, and 9.16 respectively. The graphs produced here follow the small-world phenomena. In social networking, a graph is considered small-world, if (1) its average clustering coefficient is significantly higher than a random graph constructed on the same vertex set, and (2) its average shortest path length is approximately the same as its corresponding random graph. Figure 6(a) shows a scale_free Power_law synthetic data set.

Table 1. k-skip contraction effect and hierarchy shortcut addition

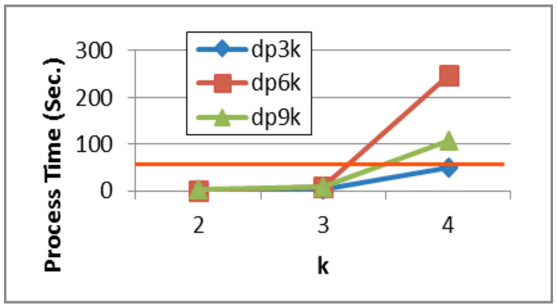
	Original		K=2			K=3			K=4		
Dataset	|V|	|E|	|V'|	|E'|	|E*|	|V'|	|E'|	|E*|	|V'|	|E'|	|E*|
dp3k	3000	2999	919	919	992	616	762	808	472	799	835
dp6k	6000	5999	1821	1830	1990	1230	1563	1676	1040	1881	1957
dp9k	9000	8999	2751	2762	3009	1835	2222	2428	1432	2322	2438

To examine the characteristic of k-skip graphs and vertex hierarchy graphs (algorithm 2), Table 1 shows the k-skip construction effect and hierarchy shortcut addition effect, where |V'| and |E'| are number of vertices and edges respectively in k-skip graph and |E*| is the number of edges in vertex hierarchy graph. It can be seen that as k increases, the number of vertex |V'| and the number of edge |E'| decreases accordingly. In addition, when k increases, the number of shortcut addition for |E*| also decreases. However, the average node degree increases (|E'|/|V'|) when k increases. This implies that when k increases, the density of the contracted graph also increases. In another word, the complexity of the contracted graph is higher than original graph, as k increases. Figure 6 shows graphically the effects of k on the structure of k-skip graph.

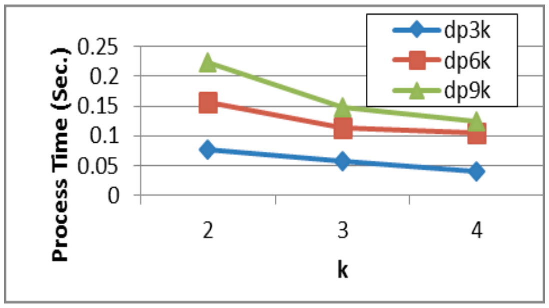


(a) Original graph (b) K=2 (c) K=3 (d) K=4

Figure 6. k-skip graphs for |V|=200, |E|=199, ASPL=7.11



(a) k-skip graph construction time



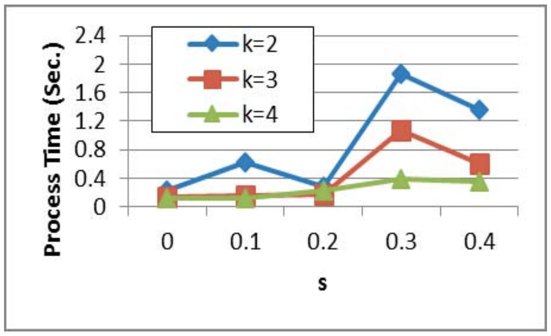
(b) label hierarchy construction time

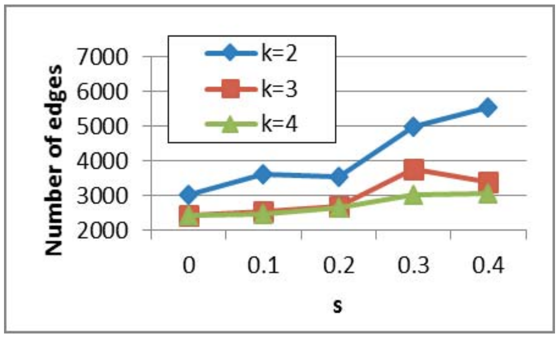
Figure 7. Effects of k on preprocessing construction time

To investigate effects of the outsourced graph construction time, we examine effects of k on preprocessing time, which include k-skip graph construction time (Figure 7(a)) and label hierarchy construction time (Figure 7(b)). We also examine effects of percentage of sensitive paths on label hierarchy construction time (Figure 8).

From Figure 7(a), it can be observed that the k-skip graph construction time increases as k value increases. In particular, it shows sharp increase at k=4. One reason is due to the graph is a scale-free network, in which most nodes have low degrees and few nodes have very high degrees. When k=2 or 3, the shortest path tree covers few low degree nodes on the edges of the graph. When k=4, the shortest path tree needs to check entire graph and thus requires more computation time. The time shown here is average of 5 runs. However, if we compared with the time to calculate all-pairs shortest paths using Dijkstra's algorithm on dp3k data set, which requires 59.7 seconds (shown as horizontal line on Figure 7(a)), our approach requires less computation times for 2<= k <= 4. Notice that all-pairs shortest path calculation is required in the 1-neighborhood-d-radius model. Figure 7(b) shows the construction time required by generating vertex hierarchy. It basically decreases as k increases. However, it is nominal compared to the k-skip graph construction time.

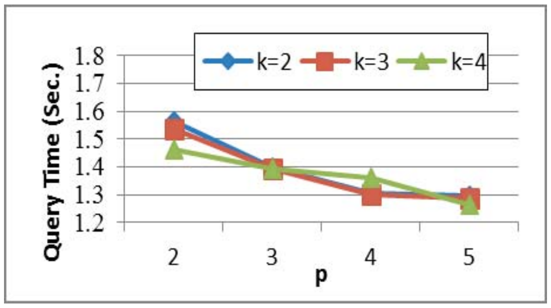
Figure 8 shows effects of percentage of sensitive paths on label hierarchy construction time. The value s is defined as the ratio of the number of source and destination vertices on sensitive paths over all vertices on the graph. For example, for s = 0.1 in data set dp9k, there are 900 nodes selected randomly, and forms 450 sensitive shortest paths. Figure 8(a) shows that when the number of sensitive shortest path increases, vertex hierarchy construction time tends to increase. The construction time shown here is average of 5 runs. The trend is not monotonic for k = 2 and 3. One explanation is that it could be due to random selection of sensitive shortest paths (source and destination nodes), and more nodes with higher node degrees are selected. As such, these nodes must lie at the bottom level of the hierarchy and therefore require more link edges to nodes on the higher level of the hierarchy. Figure 8(b) shows that when s increases, the number of link edge between hierarchies increases. The more link edges means that it takes more process time to construct the vertex hierarchy.

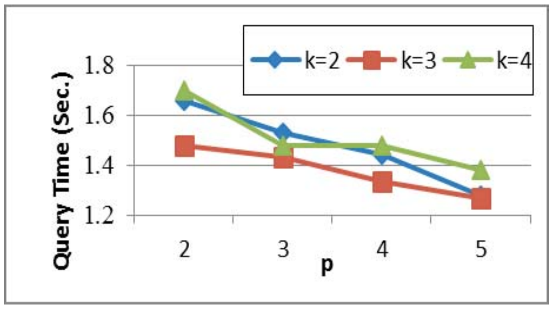




(a) label hierarchy construction time (b) Number of link edges

Figure 8. Effects of s on label hierarchy

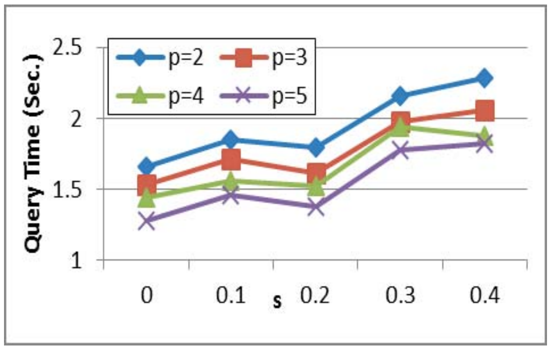




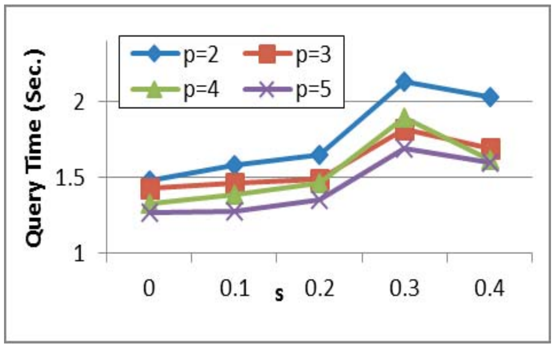
(a) dp6k (b) dp9k

Figure 9. Effects of p on shortest distance query time

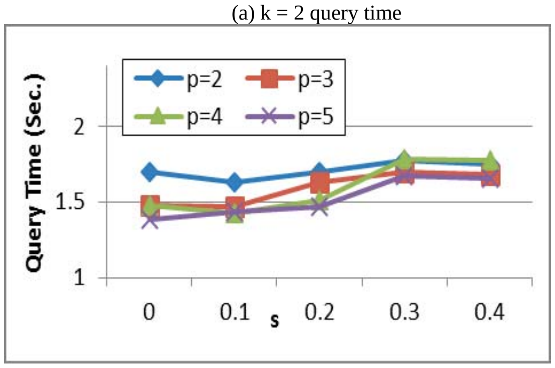
To investigate the effects of shortest path distance query times, we examine the effects of k (neighborhood), s (sensitive path), and p (outsourced server) on query processing time. Figure 9 shows the query proceeding times for 2<=k<=4, 2<=p<=5. The query time shown here is the average of five sets of queries. Each set of queries contain twenty randomly selected shortest path queries, which are selected from a pool of 20 source and 20 destination vertices. The time represents the average of twenty queries for each set. It can be seen that when p (number of outsourced server) increases, the query process time decreases for all k values. In addition, when compared to the time required by using Dijkstra's algorithm directly to calculate the shortest path distance, which is 8.28 seconds on dp6k data set, our approach appears to be more efficient.



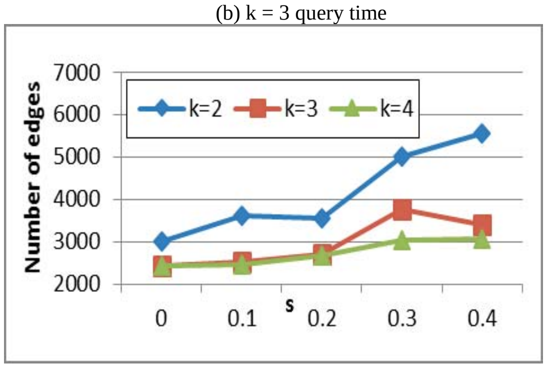
(a) k = 2 query time



(b) k = 3 query time



(c) k=4 query time



(d) Number of link edges

Figure 10. Effects of s on query time

Figure 10 shows that the query processing time increases as s (sensitive path) value increases, for 2<=k<=4, 2<=p<=4. This is due to when the number of sensitive paths increase, there will be more nodes (source and destination) on the bottom level of vertex hierarchy and it requires more fill-in artificial nodes. In turns, this will create more link edges and require more query processing time. The number of link edges is shown in Figure 10(d).

6 Conclusions

In this work, we study the problem of efficient calculation of shortest path distance in the cloud while preserving the k-neighborhood privacy and sensitive path privacy. We propose a stronger privacy model than existing 1-neighborhood-d-radius model. Our model is also more flexible that it can randomly protect zero to k-1 direct neighbors in each outsourced graph. In addition, for specified sensitive paths, their privacy is protected in each outsourced graph. We also devise an efficient 4-step scheme to calculate the shortest path distances. Numerical experiments show that the proposed approach is more efficient than prior model of constructing the 1-neighborhood privacy graph and also requires less querying time.

However more works need to be done. We plan to investigate on various real-world data sets for evaluating the overall performance characteristics. Trade-off between privacy and utility is another issue that needs to be studied and quantified. For example, when the level of privacy (in terms of k) increases, how will it affect the utility of query or be affected by the number of outsourced servers. We also need to deal with dynamic data sets that change in time, where adversary may infer sensitive information from the difference between times on the same outsource server.

Acknowledgement

This work was supported in part by the National Science Council, Taiwan, under grant NSC 101-2221-E-390 -028 -MY3.

References

1. I. Abraham, A. Fiat, A.V. Goldberg, and R.F.F. Werneck, “Highway dimension, shortest paths, and provably efficient algorithms,” In Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), 782–793, 2010.

2. D. Agrawal, A. El Abbadi, S. Antony, and S. Das, “Data management challenges in cloud computing infrastructures,” In Proceedings of the 6th international conference on Databases in Networked Information Systems, Berlin, Heidelberg, 1–10, 2010.

3. K.M. Chandy ad J. Misra, “Distributed computation on graphs: shortest path algorithm,” In Communications of ACM, Vol. 25, No. 11, 833-857, 1982.

4. S. Das, O. Egecioglu, and A. El Abbadi, “Anonymizing weighted social network graphs,” In 2010 IEEE 26th International Conference on Data Engineering (ICDE), 904–907, 2010.

5. A. W.-C. Fu, H. Wu, J. Cheng, S. Chu, and R. C.-W. Wong, “IS-LABEL: an independent-set based labeling scheme for point-to-point distance querying on large graphs,” arXiv:1211.2367, Nov. 2012.

6. J. Gao, J. X. Yu, R. Jin, J. Zhou, T. Wang, and D. Yang, “Neighborhood-privacy protected shortest distance computing in cloud,” In Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data, New York, NY, USA, 409–420, 2011.

7. H. Hacigümüs, B.R. Iyer, and S. Mehrotra. “Providing database as a service,” In ICDE, pages 29–40, 2002.

8. M. Hay, G. Miklau, D. Jensen, D. Towsley, and P. Weis, “Resisting structural re-identification in anonymized social networks,” In Proc Vldb Endow, vol. 1, no. 1, 102–114, 2008.

9. IPython.parallel, http://ipython.org/ipython-doc/dev/parallel

10. S. Kanchi, D. Vineyard, “An optimal distributed algorithm for all pairs shortest path,” In International Journal of Information Theories and Applications, Vol. 11, 141-146, 2004.

11. L. Liu, J. Liu, J. Zhang, “Privacy preservation of affinities in social networks”, In Proceedings of the International conference on Information Systems, 372-376, 2010.

12. K. Liu and E. Terzi, “Towards identity anonymization on graphs,” In Proceedings of the 2008 ACM SIGMOD international conference on Management of data, New York, USA, 93–106, 2008.

13. L. Liu, J. Wang, J. Liu, J. Zhang, “Privacy preservation in social networks with sensitive edge weights,” In SDM, 954-965, 2009.

14. S. Nath, H. Yu, and H. Chan. “Secure outsourced aggregation via one-way chain,” In SIGMOD, pages 31–44, 2009.

15. Networkx, http://networkx.github.io/

16. P. Sanders and D. Schultes. “Engineering highway hierarchies,” In Proceedings of European Symposium on Algorithms (ESA), 804–816, 2006.

17. Y. Tao, C. Sheng, and J. Pei, “On k-skip shortest paths,” In Proceedings of the 2011 ACM SIGMOD International Conference on Management of data, New York, USA, 421-432, 2011.

18. VirtualBox, https://www.virtualbox.org/

19. S.L. Wang, J.W. Chen, H.H. Ting, and T.P. Hong, “Sensitive and neighborhood privacy on shortest paths in the cloud”, In Proceedings of the 15th International Conference on Information Integration and Web-based Applications & Services, Vienna, Austria, 555-559, 2013.

20. S.L. Wang, C.C. Shih, H.H. Ting, and T.P. Hong, “Degree anonymization for K-shortest-path privacy”, In 2013 IEEE International Conference on SMC, Manchester, UK, October, 2013.

21. S.L. Wang, Z.Z. Tsai, T.P. Hong, and H.H. Ting, “Anonymizing shortest paths on social network graphs”, In Proceedings of the The Third Asian Conference on Intelligent Information and Database Systems (ACIIDS), Daegu, Korea April, 2011.

22. M.L. Yiu, Y.Lin, and K.Mouratidis. “Efficient verification of shortest path search via authenticated hints,” In ICDE, 237–248, 2010.

23. M. Yuan, L. Chen, and P. S. Yu, “Personalized privacy protection in social networks,” In Proc VLDB Endow, vol. 4, no. 2, 141–150, 2010.

24. B. Zhou and J. Pei, “Preserving privacy in social networks against neighborhood attacks,” In Proceedings of the 2008 IEEE 24th International Conference on Data Engineering, Washington, DC, USA, 506–515, 2008.

25. L. Zou, L. Chen, and M. T. Özsu, “Distance-join: pattern match query in a large graph database,” In Proc Vldb Endow, vol. 2, no. 1, pp. 886–897, Aug. 2009.