Cloud based Privacy Preserving Top-k Subgraph Querying on Large Graphs

Jianwen Zhao†, Ada Wai-chee Fu†
†*Department of Computer Science and Engineering, The Chinese University of Hong Kong*
jwzhao@cse.cuhk.edu.hk, adafu@cse.cuhk.edu.hk

***Abstract*—Subgraph isomorphism search is an important problem in graph data management. Due to its computational hardness, recent studies consider cloud computing while incorporating data anonymization for privacy protection. The state-of-the-art solution provides a framework but targets at the enumeration of all the querying results, which can be prohibitively expensive. In this work, we study the problem of privacy-preserving cloud-based diversified top-k subgraph querying, that is, given a query graph Q and a number k, we aim to retrieve k isomorphic subgraphs from the given data graph G such that together they cover as many distinct vertices as possible. We show that the state-of-the-art solution cannot address top-k query and we propose 1) a new graph anonymization technique equipped with a novel densest block based vertex mapping method and a simple and effective label generalization method; 2) an iterative querying method that involves low communication overhead. Our extensive experiments on real-life datasets verify the efficiency and the effectiveness of the proposed methods, which significantly outperform the baselines.**

***Index Terms*—privacy preservation, subgraph isomorphism, top-k, results diversity**

I. Introduction

Subgraph querying is one of the most well studied operations for graph data processing in numerous applications [1], [2]. As noted in [3], it is a key component for many explorative mining tasks, which is to help identify user specific patterns from the networks, and has become an integral part of some visual graph analytic platforms [4]–[6]. Some applications include team formation in social networks [7] and suspicious transaction pattern identification in finance informatics [6]. Given a data graph G and a query graph Q, the problem is to retrieve from G all the subgraphs that are isomorphic to Q. Many practical algorithms have been proposed to address this problem [1], [2], [8]–[14]. However, due to its NP-hardness, the computation cost is typically very high.

Cloud computing is a promising way of handling complex computing tasks. Under the common *honest-but-curious*[1] assumption [15], [16], the potential of sensitive information leakage is a serious concern when adopting cloud computing services. To our knowledge, [15] is the pioneer and the state-of-the-art work that considers *Privacy-preserving Cloud-based Subgraph Querying* (PCSQ). In their setting, a *client* who is the data owner, first adopts *K-automorphism* [17] and *label generalization* techniques to anonymize G for the purpose of protecting sensitive structures and labels and then publish the anonymized version to a remote *cloud*, when a query Q is launched, the client sends the anonymized Q to the cloud and asks for all querying results R, finally, the client filters out all the false positives from R and obtains the desired solution set S. The following example illustrates the basic ideas of the state-of-the-art solution for PCSQ in [15].

Example 1 (Algorithm for PCSQ)**.** *Fig. 1(a) shows a collaboration network G, in which each vertex represents a person, and the labels a, b, c and d stand for project manager, programmer, database developer and software tester, respectively. Suppose that a company wants to find a team consisting of a project manager, a programmer and a database developer and everyone in this team has collaboration relations with the others, then this request can be answered by issuing a query Q as depicted in Fig. 1(d). [15] provides the following cloud-based solution: (1) Anonymizing. Partition G into K blocks (or K parts, here $K = 2$) and convert it into a K-automorphic graph G^K (whose formal definition will be presented shortly in Section II, and intuitively, G^K is a symmetric graph) by introducing some dummy vertices and edges (denoted as dashed lines in Fig. 1), generalize the real labels into the corresponding label groups, see Fig. 1(b); (2) Publishing. Due to symmetry, only one block of G^K together with its one-hop neighbors, referred to as G^o (see Fig. 1(c)), are published to the cloud; the anonymized query graph Q^o (see Fig. 1(d)) is also published; (3) Querying. The cloud then performs querying over G^o and finds all subgraphs that are isomorphic to Q^o, in this example, $R_1 = \{(v_1, v_3, v_4), (v_1, v_3, v_8), (v_3, v_7, v_4), (v_9, v_{10}, v_8)\}$ will be found and returned to the client (note that the query decomposition method is adopted in [15] to find the cross-block matchings); (4) De-anonymizing. Due to symmetry, the client is able to recover all the isomorphic subgraphs of Q^o in G^K. In this example, $R_2 = \{(v_2, v_6, v_5), (v_2, v_6, v_{11}), (v_6, v_{14}, v_5), (v_{13}, v_{12}, v_{11})\}$ is found in Block 2 based on R_1 and $R = R_1 \cup R_2$. Finally, the client filters out the false positives (v_1, v_3, v_8), (v_9, v_{10}, v_8), (v_2, v_6, v_{11}), (v_6, v_{14}, v_5) and (v_{13}, v_{12}, v_{11}) from R based on Q and G and gets $S = \{(v_1, v_3, v_4), (v_7, v_3, v_4), (v_6, v_2, v_5)\}$ as the real solution.*

The *Anonymizing-Publishing-Querying-De-anonymizing* in the above is an useful framework for our work. However, the algorithm for PCSQ in [15] suffers from the following limitations: (1) Enumerating all the isomorphic subgraphs for a given query Q can be prohibitive in a large graph,

[1]A cloud is honest in performing correct computations but curious about gaining any sensitive information from user's data.

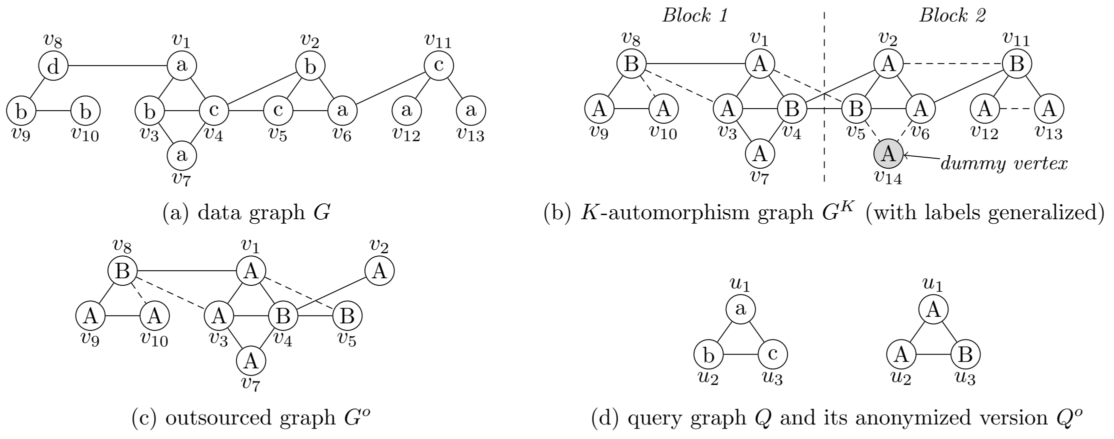

Block 1
Block 2
dummy vertex
(a) data graph G
(b) K-automorphism graph G^K (with labels generalized)
(c) outsourced graph G^o
(d) query graph Q and its anonymized version Q^o

Fig. 1. An illustration example of solving PCSQ. The meaning of the labels are: a-project manager, b-programmer, c-database developer, d-software tester. The labels are generalized to two label groups A={a, b} and B={c, d}.

especially when the data graph becomes much larger due to anonymization. Even if the enumeration can be done by the cloud, the original data graph is severely distorted because of the generalized labels and dummy edges, the amount of false positives will be excessively large and the filtering will be a daunting task for the client. (2) It is noted that the top-k diversified results are preferable in practice [18]–[22]. However, due to large number of false positives, it is hard to obtain top-k real answers from the anonymized graph efficiently, as has been confirmed by our experiments.

Contributions. To handle the above issues, (1) we propose to study the *Privacy-preserving Cloud-based Diversified top-k Subgraph Querying* (PCDSQ) problem, where we are interested in the top-k diversified results instead of enumerating all the querying results; (2) we propose a densest block based vertex mapping method and a simple yet effective label generalization algorithm for anonymizing the data graph, which allow us to focus on an area of the anonymized G with high utilities, and hence answer a query efficiently and effectively; (3) we propose an iterative querying method based on the DSQL algorithm in [19] to solve PCDSQ, which allows the client and the cloud to communicate in few iterations with low overhead to answer queries; (4) we conduct extensive experiments on large real-life graphs to verify the efficiency and effectiveness of our methods, which show that our method significantly outperforms the baselines.

The rest of this paper is organized as follows. Section II gives the related concepts and problem definition. Section III describes a baseline solution for PCDSQ. The optimization mechanisms are introduced in section IV. The experimental results are given in section V. We summarize the related works in section VI and conclude in section VII.

II. Preliminaries and Problem Definition

We first give some preliminary definitions related to our problem and then formally define PCDSQ. Two basic techniques for solving PCDSQ are also presented in this section.

Data Graph. A data graph $G = (V, E, \Sigma, L)$ is a vertex-labeled, undirected graph, where V is the set of vertices, $E \subseteq V \times V$ is the set of edges, Σ is the set of vertex labels and L is a function that assigns to each vertex $v \in V$ a label $\ell \in \Sigma$, if $L(v) = \ell$, then ℓ is the label of v.

Query Graph. A query graph $Q = (V_Q, E_Q, \Sigma_Q, L_Q)$ is defined in a similar way as a data graph, where V_Q is the set of query nodes, $E_Q \subseteq V_Q \times V_Q$ is the set of undirected query edges, Σ_Q is the set of labels in a query graph and L_Q gives each query node $u \in V_Q$ a label $L_Q(u) \in \Sigma_Q$.

Subgraph. A graph $G_s = (V_s, E_s, \Sigma_s, L_s)$ is a subgraph of G if (1) $V_s \subseteq V$, $E_s \subseteq E$ and $\Sigma_s \subseteq \Sigma$, and (2) For each edge $(v, v') \in E_s$, $v \in V_s, v' \in V_s$ and for each $v \in V_s$, $L_s(v) = L(v)$.

Subgraph Isomorphism. A subgraph G_s of G is said to be isomorphic to Q (and vice versa) if and only if there is an injective function $f : V_Q \to V_s$, such that (1) $\forall u \in V_Q$, $L_Q(u) = L(f(u))$, and (2) $\forall (u, u') \in E_Q$, $(f(u), f(u')) \in E$.

Analogous to existing works, we also say that G_s is a *matching* or an *embedding* of Q in G if the above holds. Henceforth, we will use these two terms interchangeably.

Subgraph Querying. Given a data graph G and a query graph Q, the subgraph querying problem asks to find all subgraphs of G which are isomorphic to Q.

As mentioned earlier, people prefer to find the top-k representative embeddings, this leads to the study of the diversified top-k subgraph querying (DSQ) which is defined as follows.

Definition 1 (DSQ [19]). *Given a data graph G, a query graph Q and an integer k, DSQ is to find a set S of up to k subgraphs of G which are isomorphic to Q, such that S has the largest coverage among all such selections of k or less subgraphs. Where the coverage $C(S)$ of S is defined as the number of distinct vertices it covers, i.e, suppose $S = \{s_1, \cdots, s_t\}$ and $s_i = (V_i, E_i)$ is a subgraph of G for $i =$*

$1, \cdots, t$, *then* $C(S) = \left|\bigcup_{i=1}^{t} V_i\right|$ *and* $t \leq k$.

Since we attempt to ask a remote cloud for the help of answering queries, for privacy concern, we have to protect the sensitive information in data and query graphs. Similar with existing works [23], [24] that identify the *NodeInfo* (e.g., identities, sensitive labels) and the *LinkInfo* (e.g., relationships) as two major types of information of graph data that require protection, we define the sensitive information as follows.

Definition 2 (Sensitive Information). *For a data graph* $G = (V, E, \Sigma, L)$*, the real label* $L(v)$ *of each* $v \in V$ *and the existence of an edge* $e = (u, v)$ *between two vertices* $u \in V$ *and* $v \in V$ *are regarded as its sensitive information. For a query graph* $Q = (V_Q, E_Q, \Sigma_Q, L_Q)$*, the real label* $L(u)$ *of each* $u \in V_Q$ *is the sensitive information.*

With all the above definitions, we are now ready to define PCDSQ formally.

Definition 3 (PCDSQ). *Given a data graph G, a query graph Q and an integer k, PCDSQ is to slove DSQ in a remote cloud without compromising the sensitive information of G and Q to the cloud.*

Theorem 1. *The problem of PCDSQ is NP-hard.*

Proof. It has been proved that DSQ is NP-hard [19], so PCDSQ is also NP-hard. □

Despite the NP-hardness of PCDSQ, [19] provides an approximation algorithm DSQL that can solve DSQ elegantly. And [15] solves the PCSQ problem with the privacy of both data and query graphs well protected. Our first attempt is to combine these two nice works together to tackle PCDSQ. In fact, that forms our baseline solution. The privacy model of [15] is based on K-automorphism, which we also adopt in this work. That said, we will briefly review the concept of K-automorphism and the main idea of DSQL in the following subsections.

A. K-automorphism

There are many existing works for privacy preserving graph data publication [17], [23], [25], [26]. A general framework known as K-automorphism that resists structural attacks is proposed in [17]. A formal definition of K-automorphic graph is given below.

Definition 4 (K-automorphic Graph [17]). *A graph* $G^K = (V^K, E^K)$ *is K-automorphic if there exist* $K-1$ *different automorphic functions* F_i $(i = 1, 2, ..., K-1)$ *such that* $\forall v \in V^K$*,* $F_i(v) \neq F_j(v)$ $(1 \leq i \neq j \leq K-1)$*, and* $\forall e = (u, v) \in E^K$*,* $(F_i(u), F_i(v)) \in E^K$*.*

Intuitively, in a K-automorphic graph, each vertex has $K-1$ symmetric vertices so that *an adversary cannot distinguish among them structurally with a probability higher than* $1/K$. Such *symmetric relations* are captured by a *Vertex Mapping* table (VM) with K columns and $\left|V^K\right|/K$ rows. Each entry VM$[r, c]$ at row r and column c of VM is a vertex $v \in G^K$, and each tuple of VM corresponds to K vertices that form a mapping with each other. More formally, the mapping relations are defined by the K-automorphic functions based on VM.

v_1	v_2
v_3	v_6
v_4	v_5
v_7	v_{14}
v_8	v_{11}
v_9	v_{13}
v_{10}	v_{12}

(a) VM

$F_1(v_1) = v_2$,	$F_1(v_2) = v_1$
$F_1(v_3) = v_6$,	$F_1(v_6) = v_3$
$F_1(v_4) = v_5$,	$F_1(v_5) = v_4$
$F_1(v_7) = v_{14}$,	$F_1(v_{14}) = v_7$
$F_1(v_8) = v_{11}$,	$F_1(v_{11}) = v_8$
$F_1(v_9) = v_{13}$,	$F_1(v_{13}) = v_9$
$F_1(v_{10}) = v_{12}$,	$F_1(v_{12}) = v_{10}$

(b) K-automorphic functions

Fig. 2. The vertex mapping table VM and the K-automorphic function F_1 with respect to G^K in Fig. 1(b)

Definition 5 (K-automorphic Function [17]). *For a vertex* $v =$ *VM*$[r, c]$*, its automorphic function* F_i *is defined as follows:*

$$F_i(v) = \begin{cases} VM[r, c+i], & (c+i \leq K) \\ VM[r, c+i-K], & (otherwise) \end{cases}$$

where $i = 1, \cdots, K-1$*,* $c = 1, \cdots, K$*.*

Example 2. *The VM and K-automorphic function corresponding to* G^K *in Fig. 1(b) are shown in Fig. 2.*

Given a data graph G, one can first partition it into K blocks, such that each block contains exactly $\lceil |V|/K \rceil$ vertices (otherwise, just add some dummy vertices), and then build the vertex mapping table VM based on some existing methods [15], [17]. After that, with the aid of K-automorphic functions, for each edge $(v, v') \in E$, if $(F_i(v), F_i(v')) \notin E$, add an edge $(F_i(v), F_i(v'))$ in G. This will convert G into a K-automorphic graph G^K, which is able to defend structural attacks.

B. DSQL

To our knowledge, [19] is the only existing work that considers DSQ. In [19], a novel level-based approximation algorithm DSQL is proposed. The main idea is to find the top-k embeddings of a query graph Q progressively level by level, it consists of two phases.

Phase 1. Maintain a solution set S, initially set $S = \emptyset$ and set level to be 0. At level 0, select from the data graph G a maximal set of disjoint embeddings (no overlapping) and add them into S, then move on to level 1. At the end of each level i, suppose the solution set is S_i, then at level $i+1$, find a maximal set of embeddings such that each new embedding s satisfies $|V(s) \cap V(S_i)| = i+1$, and for any two new generated embeddings s and s', $V(s) \cap V(s') \subseteq V(S_i)$, where $V(s)$, $V(s')$ and $V(S_i)$ denote the vertices set of s, s' and S_i respectively. Whenever $|S| = k$ or the last level $|V_Q| - 1$ is finished, phase 1 terminates.

Phase 2. If the solution set S generated by phase 1 does not have a good coverage ratio[2], then continuing at the level at

[2]The coverage ratio is defined as $C(S)/(k|V_Q|)$, and is considered not good if it is less than 0.5.

which phase 1 terminates, phase 2 will be triggered to generate new embeddings in the same way as phase 1 does, but each new embedding may swap with an embedding in S if a certain swapping condition is satisfied.

Example 3. *Consider the query graph Q in Fig. 1(d) and the data graph G in Fig. 1(a) again. Suppose $k = 2$, instead of returning $\{(v_1, v_3, v_4), (v_7, v_3, v_4)\}$, DSQL is able to collect (v_1, v_3, v_4) and (v_6, v_2, v_5) at level 0 and directly return them as the answer, since this set covers 6 vertices while the former set only covers 4. If $k = 3$, then DSQL will select (v_1, v_3, v_4) and (v_6, v_2, v_5) at level 0, find nothing at level 1, and finally generate (v_7, v_3, v_4) at level 2 and return the top-3 embeddings.*

DSQL has both non-trivial theoretical guarantee[3] and excellent practical performance. We adopt it as an important component in our methods. Interested readers can refer to [19] for more details.

III. A Baseline Solution

As previously mentioned, our first attempt is combining the two nice works [15] and [19] together to tackle PCDSQ, which gives a baseline solution, BAS, as described in Algorithm 1.

Algorithm 1: BAS (A BASELINE SOLUTION)

```
Input: data graph $G$, query graph $Q$, an integr $k$
Output: the solution set $S$ of PCDSQ
1  Initialize $S \leftarrow \emptyset, S' \leftarrow \emptyset, k' \leftarrow k$;
2  METIS + $K$-automorphism + Label generalization:
     $G^K \leftarrow G$;
3  Label generalization: $Q^o \leftarrow Q$;
4  Publish $Q^o$ and one block $G^o$ of $G^K$ to the cloud;
5  while $|S| < k$ do
6      increase the value of $k'$ and submit it to the cloud;
       /* Lines 7--8 are executed by the cloud   */
7      $S' \leftarrow DSQL(G^o, Q^o, k')$;
8      pass $S'$ to the client; // the client is the data
          owner who knows the real $G$ and $Q$
9      $S \leftarrow$ recover all embeddings and prune false
        positives;
10 return $S$
```

BAS follows the framework and the technical details of [15] except for the querying part, we replace the *subgraph enumeration query* with *diversified top-k subgraph query*. The data graph G is first partitioned and anonymized by using the techniques in [15], [17], [27], the query Q is also anonymized to Q^o by the generalized labels (lines 2–3). Except for the K-automorphism and label generalization techniques that have been described before, here METIS[4] is a tool for partitioning the data graph G into K blocks, it is quite efficient and is widely adopted in dealing with large graph partitioning,

[3]DSQL has the approximation ratio $\max\{\frac{1}{4}(1+\frac{1}{k}), \frac{1}{4}(1+\frac{1}{|V_Q|})\}$.

[4]http://glaros.dtc.umn.edu/gkhome/views/metis

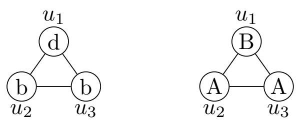


Fig. 3. Another query graph Q and Q^o

more details can be found in [27]. Afterwards, we can safely publish Q^o and one block G^o of G^K to the cloud (line 4). $DSQL(G^o, Q^o, k')$ returns the top k' diversified subgraphs in G^o matching query Q^o. Note that due to anonymization, there are likely some false embeddings in the solution set S', hence it is a sound strategy to submit a larger k' value to the cloud and hopefully we can get *enough* real embeddings within a few rounds of communications. Once sufficient[5] real results are obtained, the algorithm terminates and returns S (lines 5–10). It is easy to see that BAS can protect the sensitive information of G and Q.

From our experiments, BAS is not a good solution to PCDSQ, since the results returned by the cloud contain too many false positives, the client has to either submit a very large k' or communicate with the cloud many rounds to get the desired results, both are highly costly. This is due to:
(1) *Many fake edges in each block of G^K*. METIS aims to divide the data graph G into K blocks with similar number of vertices while minimizing the block-crossing edges. However, our requirement is that the K blocks should have similar graph structures in order to reduce the number of fake edges to be added when enforcing K-automorphism. This conflict in objectives leads to very different structures in the K blocks, and hence the anonymization process has to introduce many fake edges in each block. This phenomenon becomes more pronounced in large power-law graphs.
(2) *Small probability for an embedding to pass the label filter*. Due to label generalization, each label group can represent different real labels. Suppose each group contains θ distinct labels, and also assume a uniform distribution of the labels in G, then the probability that an embedding returned by the cloud passes the *label filter* (i.e., matches all the real labels of Q) is $(1/\theta)^{|V_Q|}$. This is very small especially when θ and Q are large. Thus, a large number of false positives are returned.

Example 4. *Consider a new query Q and Q^o in Fig. 3 and the data graph G and G^o in Fig. 1, set $k = 1$. Obviously, there is no embedding of Q in G. However, due to structure and label anonymization, there do exist several embeddings of Q^o in G^o. Therefore, the cloud has to explore all the embeddings of Q^o before the client makes the conclusion that there is no matching for Q. This introduces much unnecessary querying cost, filtering cost and communication cost.*

The above analysis shows that the anonymization techniques used in [15] is not well-suited for solving PCDSQ. More

[5]For simplicity we assume the number of embeddings for a given query is at least k, note that DSQL can detect the cases when this does not hold.

specifically, the METIS partition method is not a good way of providing K blocks for vertex mapping, and the label generalization hides too much information and makes the querying difficult. These are two main challenges for solving PCDSQ. Nevertheless, BAS is still a solution to PCDSQ albeit its huge cost.

IV. The Optimization Strategies

To address the two aforementioned challenges, we derive a *densest block based vertex mapping* method and a *sliding window based label generalization* algorithm, as well as an *iterative querying* framework, which will be discussed in detail in the following subsections.

A. Densest Block based Vertex Mapping

Most real-life large graphs exhibit power-law degree distributions, which creates challenges for graph partition. The difficulties root at the fact that there are always some blocks that retain much more edges than the others after partitioning, although they have similar number of vertices. There are some representative works such as [28], [29] that try to remedy the issues, but they are found not applicable to our problem after our trial, since simply balancing the number of edges in each block does not give similar block structures in most cases. However, this *biased block* phenomenon inspires us to ask the following question:

Can we intentionally construct a block of G^K such that it contains as many real edges as possible after anonymization and we answer queries only over this block?

The rationality behind this question is that: (1) we only ask for diversified top-k results in PCDSQ, and the number of embeddings for a given query in a large data graph is usually explosive, thus it is not necessary to look into all the blocks, as long as the diversity of the results is acceptable; (2) forcing as many real edges as possible into one block will likely decrease the ratio of fake edges in that block, this means after anonymization, there is an area of G^K which preserves higher utilities compared with the other parts, and hence querying over that block will likely return less false positives.

The above question can be formulated as the following Densest Block (DB) problem.

Definition 6 (DB). *Given a data graph G and an integer K, DB (Dense Block partitioning) is to partition G into K blocks, such that each block has the same number of vertices and the number of edges retained by one of the blocks is maximized.*

DB is related to the Densest k Subgraph problem (DkS), which is defined as follows:

Definition 7 (DkS). *Given a graph G and an integer k, find an induced subgraph of G on k vertices with maximum density (average degree).*

Theorem 2. *The problem of DB is NP-hard.*

Proof. DB is equivalent to DkS when $k = \lceil |V_G|/K \rceil$. The theorem follows since DkS is NP-hard [30]. □

Algorithm 2: DBVM

```
  Input: data graph $G$, an integer $K$
  Output: the vertex mapping table VM
1 Initialization: VM is empty;
2 $V_s \leftarrow \mathcal{A}(G, K)$;
3 Put $V_s$ into the first block $B_1$;
4 $V_{remain} \leftarrow V_G - V_s$, $G_{remain} \leftarrow G[V_{remain}]$;
5 Partition $G_{remain}$ into $K-1$ blocks $(B_2, \cdots, B_K)$ by
    METIS;
6 Construct VM in a BFS manner; // refer to [15],
    [17] for more details
7 return VM
```

Let us first assume that there is a black-box algorithm $\mathcal{A}$ that can solve DkS given $k = \lceil |V_G|/K \rceil$, that is, $\mathcal{A}$ takes G and K as its input, and will output a subset $V_s \subseteq V_G$, such that $|V_s| = \lceil |V_G|/K \rceil$ and V_s induced subgraph $G[V_s]$ has maximum density. With this, we are now ready to present our *Densest Block based Vertex Mapping* (DBVM) method, whose details are summarized in Algorithm 2.

Given a data graph G and an integer K, we first run the black-box algorithm $\mathcal{A}$ to get the vertices set V_s and put it into block B_1, which is the dense block that we want (lines 2–3); Then the subgraph $G[V_{remain}]$ induced by the remaining vertices V_{remain} will be partitioned by METIS into $K-1$ blocks (lines 4–5), here we use METIS again because the subgraph querying will be answered over the anonymized B_1, which should be the densest block before anonymization, hence using METIS to form the other blocks will not affect the querying procedure too much, and also METIS is quite efficient in partitioning large graphs. Afterwards, we follow the method in [15], [17] to construct the vertex mapping table VM in a BFS manner (lines 6–7), which is to preserve the original structure of G. With VM, we can simply follow Definition 4 and 5 and convert G into a K-automorphic graph G^K.

Since METIS is very efficient in practice and line 6 only takes $O(|V| + |E|)$ time, so the complexity of Algorithm 2 is mainly dominated by the black-box algorithm $\mathcal{A}$ of solving DkS. Due to the NP-hardness of DkS, many approximation algorithms [31]–[33] have been proposed to solve it. [31] describes a simple greedy algorithm: *start from G and repeatedly delete the min-degree vertex from the graph induced by the remaining vertices until there are exactly $\lceil |V_G|/K \rceil$ vertices left.* We adopt this greedy algorithm as an instance of $\mathcal{A}$ in our empirical study since it can be implemented in $O(|V| + |E|)$ time, and it has a good worst-case density guarantee, more details can be found in [31].

It is worth mentioning that the greedy algorithm described above is not the only choice of $\mathcal{A}$, any algorithm that can efficiently and effectively solve DkS can be plugged into our method DBVM. We have also tried some other algorithms (e.g., [34]) and finally choose the simple greedy since it strikes a good balance between efficiency and density. The following example demonstrates the effects of DBVM.

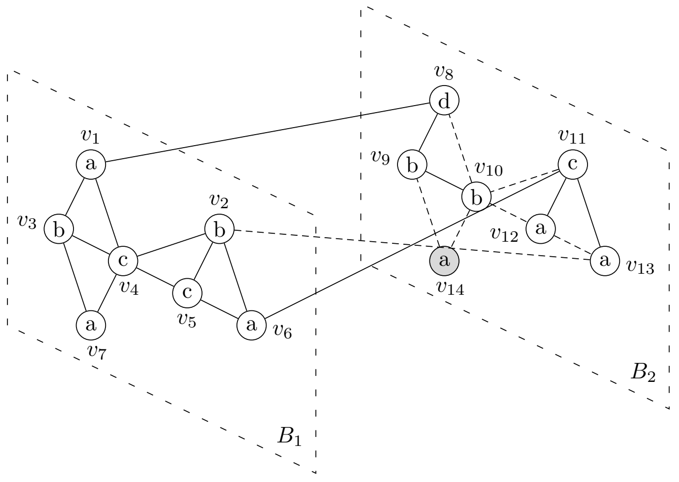


Fig. 4. G^K corresponding to G in Fig. 1(a), generated by DBVM and K-automorphism ($K = 2$)

Example 5. *Let us revisit the data graph G depicted in Fig. 1(a). If DBVM takes the greedy as an instance of $\mathcal{A}$, if will greedily remove the min-degree vertices at each step, and obtain the dense block B_1, the remaining vertices are put into B_2. Then by BFS and K-automorphism, G is converted to G^K, as shown in Fig. 4. Compared with the baseline method (refer to Fig. 1(b)), the DBVM generated B_1 contains no fake edges in this example, and thus preserves high utility. Furthermore, the privacy guarantee is not affected, since there is another block B_2 that is structurally identical to B_1, which fits the requirement of K-automorphism.*

It is worth mentioning that the anonymization cost defined by graph edit distance $|E_{G^K} - E_G|$ is also an important factor that should be considered, since if the cost was high, G^K would be large, and publish it to the cloud would cause high communication overhead. Note that our DBVM method has a similar anonymization cost when compared to the method in [15], which will be shown later in the experimental study.

B. Label Generalization by Sliding Window Grouping

We next introduce our new label generalization method. As discussed previously, the drawback of the existing method in [15] is that it maps θ distinct labels to a same label group, then in the process of de-anonymization, each label group only has one out of θ possibilities to match the real query label, which dramatically decreases the probability of passing the label filter for an embedding. Therefore, a new label generalization method that is able to hide the real label of a vertex but can distinguish different labels based on different label groups is desirable in our case. To this end, we propose a new simple and effective label generalization method, which forms label groups by a sliding window and is presented in Algorithm 3.

LGSW (label generalization by sliding window) takes a sequence $\mathcal{L}$ of all the distinct labels of a data graph and a parameter θ which denotes the size of a label group as the input, and will output a table LM in which each row records the one-to-one mapping relation between a label and its corresponding label group. Specifically, it first initializes an empty LM (line 1) and perform random permutation on $\mathcal{L}$ to get an array $\mathcal{L}'$ (line 2). Note that $\mathcal{L}'$ can be viewed as a circular array. A random integer x in the range $[1, \theta]$ is also generated (line 3). Subsequently, a sliding window $\mathcal{W}$ with fixed size θ is constructed (line 4) to slide on $\mathcal{L}'$ and wrap up labels into label groups. Denote by $\mathcal{W}.start$ the starting position of $\mathcal{W}$ on the track of $\mathcal{L}'$. Initially, $\mathcal{W}.start = 1$, and the labels in $\mathcal{L}'$ covered by $\mathcal{W}$ are $\{\mathcal{L}'_1, \cdots, \mathcal{L}'_\theta\}$, they form the label group $\mathcal{G}_1$ that maps to the x-th real label in $\mathcal{G}_1$. After this, $\mathcal{W}$ will move forward along $\mathcal{L}'$ by one position at each step, and wrap up a different label group. This process will be repeated $|\mathcal{L}'|$ times until each of the real labels finds its corresponding label group (lines 5–9). All the mapping relations between real labels and label groups are captured by a *label–label group* mapping table LM and will be returned (line 10).

It is easy to see that LGSW runs in $O(|\mathcal{L}|)$ time. The following shows an example of obtaining the *label–label group* mapping table.

Example 6. *For a sequence of distinct labels $\mathcal{L} = \{a, b, c, d, e\}$, one possible random permutation of $\mathcal{L}$ is $\mathcal{L}' = \{a, c, e, d, b\}$. Let $\theta = 3$ and suppose $x = 2$, LGSW will output a label–label group mapping table LM as shown in Table I. In the data publication process, the real label of each vertex in both G and Q with be replaced by its label groups and then uploaded to the cloud.*

Denote by ℓ_i the x-th real label in a label group $\mathcal{G}_i$, then the following theorem guarantees the main advantage of LGSW.

Algorithm 3: LGSW

```
Input: a sequence $\mathcal{L}$ of distinct labels, an integer $\theta$
Output: a label – label group mapping table LM
1 Initialize LM as empty;
2 $\mathcal{L}' \leftarrow$ randomly permute $\mathcal{L}$ and store its contents into a circular array;
3 $x \leftarrow$ a random integer in the range $[1, \theta]$;
4 $\mathcal{W}.size = \theta$;
5 for $i = 1$ to $|\mathcal{L}'|$ do
6     $\mathcal{G}_i \leftarrow \emptyset$;
7     $\mathcal{W}.start = i$;
8     $\mathcal{G}_i.append$(all the labels in $\mathcal{L}'$ covered by $\mathcal{W}$);
9     LM.$append$($< \mathcal{G}_i[x], \mathcal{G}_i >$);
10 return LM
```

TABLE I
LABEL–LABEL GROUP MAPPING TABLE, LM

Real labels	Label groups
c	{a, c, e}
e	{c, e, d}
d	{e, d, b}
b	{d, b, a}
a	{b, a, c}

Theorem 3. *If $\theta < |\mathcal{L}|$, LGSW can always produce a label group $\mathcal{G}_i$ for each real label $\ell_i \in \mathcal{L}$, such that $\ell_i \in \mathcal{G}_i$, and for any $i, j = 1, 2, \cdots, |\mathcal{L}|$ and $i \neq j$, if $\mathcal{G}_i \neq \mathcal{G}_j$, then $\ell_i \neq \ell_j$.*

Proof. According to the algorithm, the sliding window $\mathcal{W}$ will move along the track of the circular array $\mathcal{L}'$ by $|\mathcal{L}|$ steps, with one label group produced at each step. Then if $\theta < |\mathcal{L}|$, there will be no two different steps that $\mathcal{W}$ is at the same position, thus in total $|\mathcal{L}|$ groups will be generated. For two different label groups $\mathcal{G}_i \neq \mathcal{G}_j$ ($i \neq j$), the common label they share must be in different positions in each of the two groups, thus $\ell_i \neq \ell_j$. And $\ell_i \in \mathcal{G}_i$ directly follows from the mechanism of the algorithm. □

Note that $\theta \ll |\mathcal{L}|$ in practice, then since there are θ labels in each label group, the above theorem implies that for each label group, the cloud cannot infer the real label with a probability higher than $1/\theta$. Even if the adversary knows the LGSW algorithm, and thereby deduces the circular list $\mathcal{L}'$, the value of x is not known and there are still θ possible values for each label group. More importantly, as a direct result of Theorem 3, we have the following corollary.

Corollary 1. *LGSW guarantees a one-to-one mapping relation between a real label and its label group, and hence if we launch a query over the anonymized graph, the false positives in the querying results will only be produced due to the fake edges introduced by K-automorphism.*

The above results highlight the soundness of LGSW, which is a simple method but can drastically decrease the probability of returning a false positive when a query is issued. Moreover, the sensitive label information is well protected.

C. Enhanced Iterative Querying

As mentioned earlier, there must be some false positives contained in the solution set of PCDSQ due to the fake edges introduced for data anonymization. Hence, when dealing with top-k query, the client must submit a larger k (denoted by k') to the could for the hope of getting k real embeddings.

There are two possible ways of doing this. On one hand, if a sufficiently large k' ($k' \gg k$) is submitted, the client may get enough real embeddings in only one iteration, however, in such case, the cloud has to do more redundant computations. On the other hand, if k' is just slightly larger than k, then the client may need to communicate with the cloud for many times to finally obtain sufficient results, which causes high communication overheads, and a more serious problem is that in such case, the cloud needs to restart DSQL from scratch at each iteration for answering the top-k' query with a different k' value, this is obviously time-consuming.

To remedy the above issues, we propose an iterative querying process by observing the special structure of DSQL, that is: DSQL runs in a level-wise manner, where a level number i indicates the overlapping size between a newly collected embedding and the solution set that has been generated at the end of last level. In a large graph, the number of embeddings for a given query is typically excessively large, and k is small in practice. Thus, DSQL rarely reaches its last level and usually returns results at a low level. As a result, for a given query Q and k, the client can first submit an appropriate k' to the cloud to see if s/he can get enough real embeddings in one iteration, and if not, change the value of k' and ask the cloud to continue generating more results from the level at which it terminates the last time instead of restarting DSQL from scratch, which will save some computations and accelerate the querying process. This can repeat for multiple iterations until sufficient top-k results are obtained.

Algorithm 4: ENHANCED ITERATIVE QUERYING (THE FINAL SOLUTION TO PCDSQ)

```
Input: data graph $G$, query graph $Q$, an integr $k$, a
       user specified parameter $\alpha \in (0, 1)$
Output: the solution set $S$ of PCDSQ
1  Initialize $S \leftarrow \emptyset, S' \leftarrow \emptyset, T \leftarrow \emptyset, k' \leftarrow 0, i \leftarrow 0$;
   /* Anonymizing */
2  DBVM + $K$-automorphism + LGSW: $G^K \leftarrow G$
3  LGSW: $Q^o \leftarrow Q$;
   /* Publishing */
4  Publish $Q^o$ and one block $G^o$ of $G^K$ to the cloud;
5  while $|S| < k$ do
6      $k' \leftarrow k' + (k - |S|)/\alpha$;
7      submit $k'$ to the cloud;
       /* Querying, executed by the cloud */
8      $(T, j) \leftarrow mDSQL(G^o, Q^o, S', k', i)$;
9      $i \leftarrow j$;
10     $S' = S' \cup T$;
11     pass the newly generated embeddings $T$ to the
         client;
       /* De-anonymizing, executed by the client */
12     $S \leftarrow$ filter out false positives from $T$ and add the
         real embeddings into $S$;
13 return $S$
```

Algorithm 4 illustrates the iterative querying method and also summarizes the major contributions of our work, it assembles all the optimization techniques that we have introduced in the previous subsections. As discussed before, we follow the framework of the existing work [15] but replace some important components with our methods. Specifically, the anonymizing procedure is enhanced with DBVM and LGSW (lines 2-3), the querying part is strengthened by the iterative querying (lines 5-12). Line 6 describes how we update k' by using a user-specified parameter α. From our empirical studies, setting α to be the ratio of real edges in G^o is a good choice. Line 8 is the key step of the iterative querying method, we modify DSQL so that it takes two extra parameters S' and i as input, where S' contains all the embeddings that the cloud has generated so far, i denotes the level that it terminates in the last iteration (Note that DSQL runs from level 0 up to the level j when k' embeddings are found, and no further embeddings can be generated by DSQL at levels 0 up to $j - 1$). This modified version mDSQL is executed at the cloud to continue computing from level i instead of restarting from scratch, it returns the newly generated embeddings T once $|S' \cup T| = k'$. Note that S' and i have very small sizes and will be cached in the main memory of the cloud for the execution of the next iteration of the algorithm.

V. Empirical Study

We empirically evaluate our methods on large real life datasets. The data graphs, query graphs and experimental setup are described in the following.

Datasets. We use five datasets in our experiments. They are DBpedia, IMDB, Youtube, USpatents and WikiTalk. DBpedia is an RDF graph crawled from Wikipedia and used in [15]; IMDB models rich information of movies and TV series, Youtube captures relationships between videos, USpatents is a citation graph that includes all citations made by patents, these three datasets are used in [19]; WikiTalk is a network in which each vertex represents a Wikipedia user and an edge means two users have discussion and communication in between, this dataset is obtained from KONECT network collection[6]. Each of the datasets is a data graph. Note that DBpedia, Youtube and WikiTalk have no given labels originally, we have assigned a label for each vertex of each graph uniformly at random from a label set with size 100. The statistics of the five data graphs are assembled in Table II.

Query set. The query set for each data graph is generated by randomly extracting connected subgraphs of the data graph G using the generator provided by [13], this is to ensure that there is always at least one embedding in G for a given query. Similar to [19], there are 1000 query graphs in one query set with the same query size (the number of edges) for each data graph. Let the query size be q, the query generator begins with an empty Q, and randomly picks a vertex u from G, puts it into Q, and continues to randomly choose an edge $e = (u, v)$ incident to a vertex u in Q from E_G, and adds v and e to Q, until there are q edges in Q. We vary q from 1 to 10 in our experiments and set the default value of q to be 5.

Measurements. We first present some experimental results about subgraph enumeration cost, which serves as the motivation of studying top-k query.

We then compare the following four methods in the experiments to show the effectiveness of the proposed optimization strategies.

- BAS: the baseline solution.
- BAS + DBVM: improves BAS by only adopting DBVM.
- BAS + LGSW: improves BAS by only adopting LGSW.
- DBSW: the method equipped with both DBVM and LGSW.

Afterwards, we measure the querying performance of our enhanced iterative querying method by showing the processing time at both the cloud and the client sides, the quality (coverage) of the querying results, and the communication cost. We also present the sensitivity analysis of the parameter α used in Algorithm 4. Note that the query related performance such as querying time, coverage ratio, etc. is measured by taking average over 1000 query graphs unless otherwise stated. Also note that we set α to be the ratio of real edges in G^o in all related experiments except for the sensitivity analysis.

[6]http://konect.uni-koblenz.de/

TABLE II
Statistics of datasets (* Indicates Synthetic Labels)

Datasets	$\|V\|$	$\|E\|$	$\|\Sigma\|$	**Avg. degree**
DBpedia	1121413	3722556	100*	6.64
Youtube	1157828	2987624	100*	5.16
WikiTalk	2987535	8146544	100*	5.45
USpatents	3774768	16518947	418	8.75
IMDB	4487324	7489209	123	3.34

TABLE III
Subgraph Enumeration Cost

Datasets	**Avg. number**	**Max. number**	**time (sec)**
DBpedia	11.1 million	509 million	113.536
Youtube	53 million	1258.2 million	527.714
Others	–	–	–

Finally, we show the anonymization cost of coverting G to G^K. For anonymization, we partition each data graph into K blocks, where K varies from 2 to 6, we always set $\theta = K$ for LGSW in all the experiments.

Setup. We use a PC running Ubuntu 18.04 LTS and equipped with 3.70GHz Intel Xeon(R) E5-1630 CPU, 16 GB RAM as the client side. The cloud sever is a linux virtual machine with 8 CPU cores and 64 GB main memory, provided by Microsoft Azure cloud. All algorithms are implemented in C++.

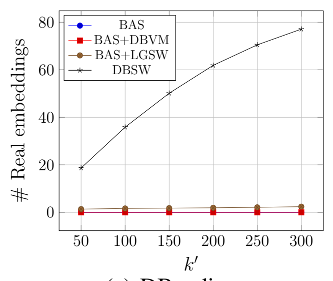
A. Subgraph Enumeration Cost

We first study the enumeration of all the embeddings for a given query in a large data graph to show that the result size is typically too large and the computation cost too high. We randomly generate 50 queries for DBpedia and show the results by taking the average, and for the other datasets, this value is set to be 25, as 50 and 25 queries already take several hours. The algorithm we use is *BoostIso* [13] and the code is provided by their authors. Note that the more recent work of [14] is more for handling large query graphs and thus does not suit our settings.

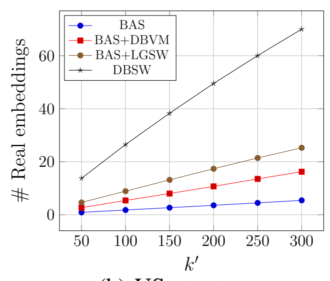
As shown in Table III, we only have results for DBpedia and Youtube, and the number of embeddings for a given query is in millions. For other datasets, we could not get the results in a reasonable time (taking more than 24 hours), and the result sizes should be much bigger. If the data graph becomes much larger after anonymization, it would be prohibitively expensive to explore all the embeddings, even the cloud cannot afford the computation workload. This study reflects the necessity of studying top-k query.

B. Effectiveness of DBVM and LGSW

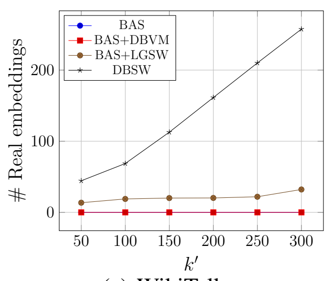
We show in this subsection the effectiveness of the two main optimization strategies DBVM and LGSW proposed in



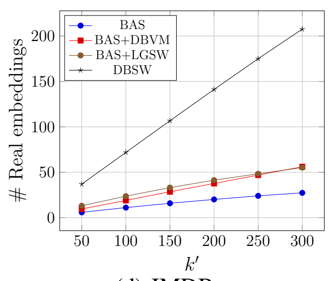
(a) DBpedia



(b) USpatents

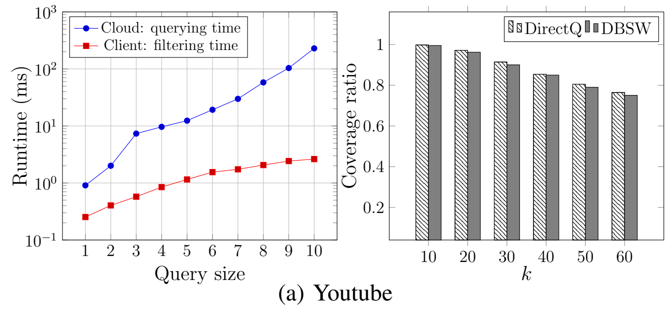


(c) WikiTalk

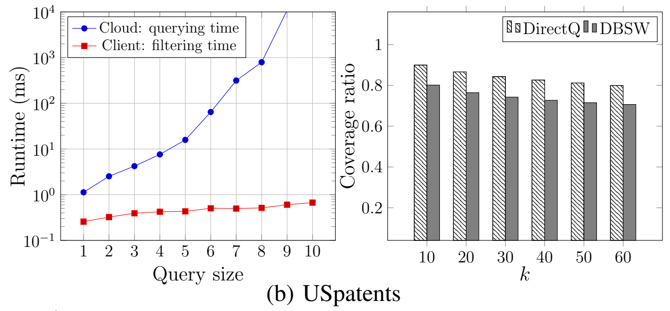


(d) IMDB

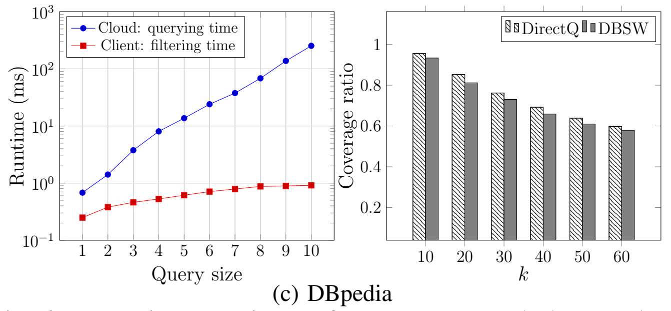
Fig. 5: Measuring # real embeddings obtained by different methods. X-axis: the submitted k', varies from 50 to 300; Y-axis: k, the # real embeddings obtained after remove false positives; set $K = 6$ and $q = 5$ as their default values.



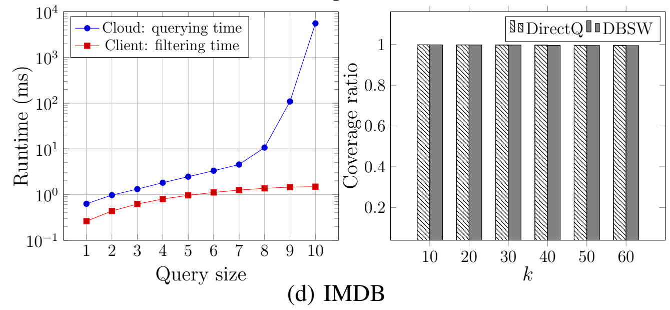
(a) Youtube



(b) USpatents

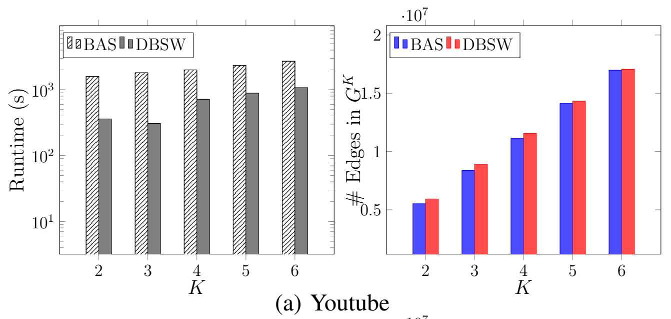


(c) DBpedia

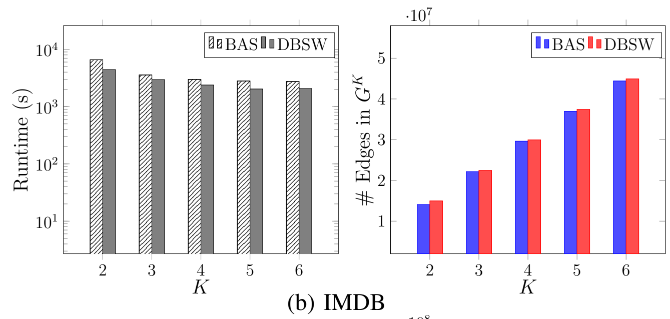


(d) IMDB

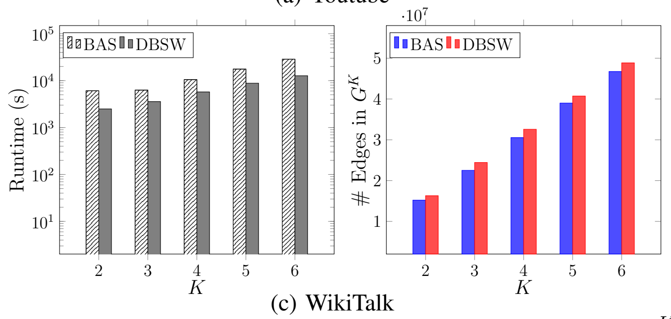
Fig. 6: Measuring querying performance. For each dataset, the left figure shows the querying time at the cloud and the filtering time at the client side (q varies from 1 to 10, $K = 6$ and $k = 50$); the right figure shows the coverage ratio of the PCDSQ results compared with directly querying over the original data graph G ($q = 5$, $K = 6$ and k varies from 10 to 60).



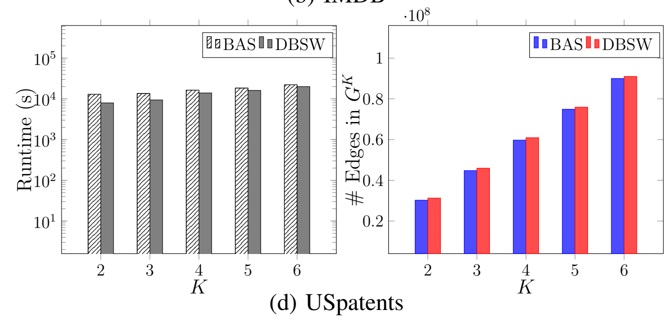
(a) Youtube



(b) IMDB



(c) WikiTalk



(d) USpatents

Fig. 7: Measuring anonymization cost of converting G to G^K. For each dataset, the left figure shows the running time, the right figure shows the number of edges in G^K. K varies from 2 to 6 and we always set $\theta = K$.

this work. Specifically, we anonymize G into G^K by using four different methods: BAS, BAS + DBVM, BAS + LGSW and DBSW that are mentioned before. For each dataset, We set $\theta = K = 6$ as this is the worst case test (recall that K ranges from 2 to 6 in out setting), since the larger K, the more fake edges (and hence false positives) introduced, which can be observed from Fig. 7. We set the query size q as its default value 5, and vary the value of k' from 50 to 300 and ask the cloud to compute top-k' results, after pruning false positives, we count the number of real embeddings that remain in the solution set. Fig. 5 shows the results over different datasets.

Clearly, DBSW, which is the combination of DBVM and LGSW, significantly outperforms the other three methods and BAS is the worst in all cases. For DBpedia and WikiTalk, BAS, BAS + DBVM and BAS + LGSW have nearly the same poor performance, the plots of BAS and BAS + DBVM completely coincide since they almost return 0 real embeddings. Interestingly, for IMDB and USpatents, the three baselines can return more real results than that on other datasets, this is due to the large skewness of their label distributions, which is an important assumption made by the state-of-the-art [15] in order to effectively solve PCSQ, we do not have such assumption in our work. Furthermore, it can be observed that the two optimization methods DBVM and LGSW are equivalently important, adopting one of them cannot give too much improvements. Only by combining them together, can we hope to reach our goals of obtaining more real embeddings.

C. Querying Performance

This subsection demonstrates the overall querying performance of our iterative querying method (Algorithm 4), including the processing cost at both the cloud and the client sides, the quality of the querying results, the communication cost and also the sensitivity analysis of the parameter α. Note that in this part, we only report the results of our final solution instead of comparing with the other baselines, since we have already showed in the above subsection that the other methods cannot even return sufficient embeddings.

Querying and filtering cost. We set $\theta = K = 6$ (the worst-case), $k = 50$ (a relatively large k) and varies the query size from 1 to 10 to test the querying performance, the running time is taken average over 1000 queries for each query set. Fig. 6 reports the results. It can be observed that both the processing in the cloud and the client side are quite efficient over different datasets, and can be finished in just seconds on average for different query size. Moreover, the querying time at the cloud side grows exponentially as q goes large, which is due to the NP-hardness of subgraph isomorphism search, while the filtering time at the client side grows slowly since filtering out false positives can be done in linear time.

Quality of querying results. We set $\theta = K = 6$ (the worst-case), $q = 5$ (a medium query size) and varies k from 10 to 60 to test the quality (i.e., the coverage ratio) of the querying results. We compare two methods: DBSW and DirectQ, where DBSW is our optimized method for solving PCDSQ and it queries over one block of the anonymized G^K (i.e., G^o), whereas DirectQ directly runs DSQL over the original, entire data graph G. We compare their results to show that our method do not lose two much interesting results. Fig. 6 shows the results. Clearly for the datasets Youtube, DBpedia and IMDB, the two compared methods return top-k embeddings with almost the same coverage. For USpatents, DBSW is slightly worse than DirectQ, which is reasonable due to the existence of false positives, nevertheless, it still returns top-k embeddings with the coverage ratio larger than 0.6, which is considered to be good in practice, since DSQL has approximation ratio guarantee less than 0.5 [19].

TABLE IV
ITERATIVE QUERYING PERFORMANCE: # ITERATIONS ($K = 6, q = 5$)

Datasets	α	$k = 10$	$k = 20$	$k = 30$	$k = 40$	$k = 50$	$k = 60$
DBpedia	0.67	3.5	3.9	3.1	3.8	3.7	3.6
Youtube	0.70	1	1.1	1	1.1	1.1	1.1
WikiTalk	0.71	1	1	1	1.4	1.1	1.7
IMDB	0.80	2.3	2.5	2.2	2.4	3	2.8
USpatents	0.47	4.8	14.4	9	7	6.8	6.7

TABLE V
COMMUNICATION COST ($K = 6, q = 5, k = 60$)

Datasets	$size(G^K)$	$size(G^o)$	$T(G^o)$	$size(S')$	$T(S')$
DBpedia	404.2 MB	87.9 MB	31.3 s	6.6 kB	0.89 s
Youtube	320.0 MB	75.1 MB	30.4 s	5.0 kB	0.84 s
WikiTalk	920.0 MB	228.7 MB	45.8 s	4.9 kB	0.84 s
USpatents	1.8 GB	407.3 MB	55.6 s	23.8 kB	1.21 s
IMDB	911.2 MB	209.8 MB	45.5 s	5.8 kB	0.85 s

Communication cost. In this part, we first report the number of iterations needed for the iterative querying method to get top-k real embeddings for a typical setting of α (i.e., the ratio of real edges in G^o), and then based on that to discuss the communication cost. Specifically, we set $\theta = K = 6$ and $q = 5$ to be their default values and vary k from 10 to 60 to show the number of iterations, and we then take the result of $k = 60$ (the largest k, which means the result size is the largest) to measure the communication cost. Table IV and Table V summarize the results.

The results in Table IV show that the querying can be finished in very few iterations for different datasets and different k values. The dataset USpatents needs more iterations because of the smaller percentage of real edges, this can be improved by changing α to a smaller value, as will be shown later in the part of sensitivity analysis.

We use the size of the data that needs to be transferred between the cloud and the client to evaluate the communication cost. The data includes the data graph and the set S' of embeddings generated by the cloud (the query graph is very small and hence omitted). Note that for the data graph, the client only needs to send one block G^o of G^K to the cloud once, which is quite efficient due to the smaller data size of G^o compared with that of G^K, the $2nd$ to $4th$ column of Table V report the size of G^K, the size of G^o and the transfer time of G^o respectively. Observe that in less than 1 minute, the data graph can be uploaded to the cloud, and this only needs to be

TABLE VI
SENSITIVITY ANALYSIS: # ITERATIONS ($K = 6, q = 5, k = 50$)

Datasets	$\alpha = 0.3$	$\alpha = 0.45$	$\alpha = 0.6$	$\alpha = 0.75$	$\alpha = 0.9$
DBpedia	1	1.6	2.7	4.3	5.8
Youtube	1	1	1	1.1	2.5
WikiTalk	1	1	1	1.3	2.1
IMDB	1	1.6	1.4	2.6	3.5
USpatents	3.4	5.5	8.5	7.7	9

done once. For the size of the embeddings set S' that needs to be transferred from the cloud to the client, we use the result with the setup $q = 5$ and $k = 60$, and report the size of S' and the transfer time in the last two columns of Table V. Clearly, the communication between the cloud and the client is very efficient and the overhead is pretty low, which benefits from our enhanced iterative querying algorithm.

Sensitivity analysis of α. Note that we use a user specified parameter α in Algorithm 4 to balance the computation workload of the cloud and the rounds of communication between the cloud and the client. In the previous experiments, we set α to be the ratio of real edges in G^o, which is a straightforward choice since essentially α should be an estimation of the ratio of real embeddings in G^o for a given query, and we believe that setting is reasonable, and the experiments aforementioned have validated it. Nevertheless, to see how α affects the querying process, we conduct sensitivity analysis in this part. We set $\theta = K = 6$, $q = 5$, $k = 50$ and count the number of iterations needed for iterative querying by setting different α values. Table VI gives the results. Note that for each dataset, as α goes large, indeed, more iterations of communication between the cloud and the client will be needed to obtain the real top-50 embeddings. However, the number of iterations is not that sensitive to the value of α, the statistics in both Table IV and Table VI show that the querying can be done in less than 10 iterations in most cases for different values of α, which gives more freedom to the client in the choice of α.

To summarize, our iterative querying method equipped with the two optimization techniques DBVM and LGSW can effectively solve PCDSQ with very few iterations, very low processing cost in both the cloud and the client side, as well as their communications.

D. Anonymization Cost

Finally, we present the experimental results of anonymizing data graphs. Specifically, we measure the time cost of converting G to G^K and also the number of edges in G^K with different K values. Since BAS and BAS+LGSW produce G^K with the same graph structures, and so do BAS+DBVM and DBSW, and the time of label generalization of LGSW and the baseline solution are nearly the same (both can be done in linear time), it suffices to compare BAS and DBSW, and we only report the results of these two methods.

Fig .7 shows that DBSW performs slightly better in terms of running time while it introduces a few more noise edges. For running time, although DBSW spends some time in finding the dense block, but it saves a lot of time in doing vertex mapping. This is because DBSW typically chooses the max-degree vertex that has not been mapped yet at each step of BFS in each of the other blocks to map with a vertex in the dense block, while BAS has to spend more time in finding vertices with similar degrees. For the number of edges in G^K, BAS performs a little better since it uses METIS to partition the entire data graph and can get more balanced blocks, while DBSW intentionally makes the K blocks unbalanced to get an area with high utilities. Overall, our optimized solution DBSW has as good time and size performance as the existing methods in graph anonymization, while the quality of the anonymized graph is much better as verified by the querying results.

VI. RELATED WORK

Subgraph Isomorphism Search. Much effort has been devoted to subgraph isomorphism search in spite of its NP-hardness. Ullmann [35] first proposes the recursive back-tracking framework, the crux is to search and verify partial solutions incrementally. Based on this, many representative variants have been developed [1], [2], [8]–[14]. Among them, VF2 [8], QuickSI [9] and GraphQL [1] introduce different strategies of selecting appropriate querying orders for avoiding fruitless search eailier; STwig [10], SPath [2] and CFL-Match [14] adopt query decomposition methods to decompose a query graph into small pieces for easy matching and finally join the results together; RBSub [11], TurboIso [12] and BoostIso [13] solve the problem by using graph compression techniques, which merge similar vertices together to boost the matching process. CFL-Match [14] mainly focuses on large queries. For diversified top-k subgraph isomorphism search, [19] is the only work to our knowledge, some other related works are diversified top-k graph pattern matching under graph simulation [18], and diversified top-k clique search [36].

Privacy Preserving Graph Data Processing. Many existing works focus on privacy preserving graph data publication [17], [23]–[26], [37]–[41]. Most of them are proposed for defending structural attacks, some of them [37], [39], [40] only consider one type of structural attack. The techniques [17], [23], [26] based on the k-symmetry concept can resist any structural attacks and thus are useful in our work. K-automorphism [17] is preferred in our case since we can convert a data graph into a K-automorphic graph without deleting any vertices and edges. There are also some work on protecting sensitive labels in a graph, [41] proposes a k-degree-l-diversity model for label protection, but it needs to modify the labels of some vertices in order to satisfy l-diversity, which may affect the querying results, thus we use the label generalization idea. For privacy preserving graph querying in the cloud, [15] is the only existing work, some other related works are [42] over encrypted graph data and [43] for shortest path queries.

VII. CONCLUSION

Our problem is to find k subgraphs in a data graph that are isomorphic to a given query graph with the maximum coverage with the help of a remote cloud, so that the sensitive information is well protected. We propose an iterative querying method equipped with a densest block based vertex mapping (DBVM) and a sliding window based label generalization (LGSW) technique. We analyse the privacy preservation of the algorithms and empirically show the efficiency and the effectiveness of our method, which outperforms the baseline solutions significantly.

VIII. Acknowledgment

We would like to thank the authors of [13], [15] and [19] for kindly sharing their major source code with us, and the anonymous reviewers for their helpful comments.

References

[1] H. He and A. K. Singh, "Query language and access methods for graph databases," in *Managing and mining graph data*. Springer, 2010, pp. 125–160.

[2] P. Zhao and J. Han, "On graph query optimization in large networks," *Proceedings of the VLDB Endowment*, vol. 3, no. 1-2, pp. 340–351, 2010.

[3] B. Du, S. Zhang, N. Cao, and H. Tong, "First: Fast interactive attributed subgraph matching," in *SIGKDD*. ACM, 2017, pp. 1447–1456.

[4] R. Pienta, A. Tamersoy, H. Tong, and D. H. Chau, "Mage: Matching approximate patterns in richly-attributed graphs," in *Big Data (Big Data), 2014 IEEE International Conference on*. IEEE, 2014, pp. 585–590.

[5] H. Tong and C. Faloutsos, "Center-piece subgraphs: problem definition and fast solutions," in *SIGKDD*. ACM, 2006, pp. 404–413.

[6] H. Tong, C. Faloutsos, B. Gallagher, and T. Eliassi-Rad, "Fast best-effort pattern matching in large attributed graphs," in *SIGKDD*. ACM, 2007, pp. 737–746.

[7] L. Li, N. Cao, K. Ehrlich, Y.-R. Lin, and N. Buchler, "Replacing the irreplaceable: Fast algorithms for team member recommendation," in *WWW*. ACM, 2015, pp. 636–646.

[8] L. P. Cordella, P. Foggia, C. Sansone, and M. Vento, "A (sub) graph isomorphism algorithm for matching large graphs," *IEEE transactions on pattern analysis and machine intelligence*, vol. 26, no. 10, pp. 1367–1372, 2004.

[9] H. Shang, Y. Zhang, X. Lin, and J. X. Yu, "Taming verification hardness: an efficient algorithm for testing subgraph isomorphism," *PVLDB*, vol. 1, no. 1, pp. 364–375, 2008.

[10] Z. Sun, H. Wang, H. Wang, B. Shao, and J. Li, "Efficient subgraph matching on billion node graphs," *Proceedings of the VLDB Endowment*, vol. 5, no. 9, pp. 788–799, 2012.

[11] W. Fan, X. Wang, and Y. Wu, "Querying big graphs within bounded resources," in *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*. ACM, 2014, pp. 301–312.

[12] W.-S. Han, J. Lee, and J.-H. Lee, "Turbo iso: towards ultrafast and robust subgraph isomorphism search in large graph databases," in *SIGMOD*. ACM, 2013, pp. 337–348.

[13] X. Ren and J. Wang, "Exploiting vertex relationships in speeding up subgraph isomorphism over large graphs," *Proceedings of the VLDB Endowment*, vol. 8, no. 5, pp. 617–628, 2015.

[14] F. Bi, L. Chang, X. Lin, L. Qin, and W. Zhang, "Efficient subgraph matching by postponing cartesian products," in *SIGMOD*. ACM, 2016, pp. 1199–1214.

[15] Z. Chang, L. Zou, and F. Li, "Privacy preserving subgraph matching on large graphs in cloud," in *SIGMOD*. ACM, 2016, pp. 199–213.

[16] J. Wang and Z. A. Kissel, *Introduction to network security: theory and practice*. John Wiley & Sons, 2015.

[17] L. Zou, L. Chen, and M. T. Özsu, "K-automorphism: A general framework for privacy preserving network publication," *Proceedings of the VLDB Endowment*, vol. 2, no. 1, pp. 946–957, 2009.

[18] W. Fan, X. Wang, and Y. Wu, "Diversified top-k graph pattern matching," *Proceedings of the VLDB Endowment*, vol. 6, no. 13, pp. 1510–1521, 2013.

[19] Z. Yang, A. W.-C. Fu, and R. Liu, "Diversified top-k subgraph querying in a large graph," in *SIGMOD*. ACM, 2016, pp. 1167–1182.

[20] J. Cheng, X. Zeng, and J. X. Yu, "Top-k graph pattern matching over large graphs," in *ICDE*. IEEE, 2013, pp. 1033–1044.

[21] M. Gupta, J. Gao, X. Yan, H. Cam, and J. Han, "Top-k interesting subgraph discovery in information networks," in *ICDE*. IEEE, 2014, pp. 820–831.

[22] L. Zou, L. Chen, and Y. Lu, "Top-k subgraph matching query in a large graph," in *Proceedings of the ACM first Ph. D. workshop in CIKM*. ACM, 2007, pp. 139–146.

[23] J. Cheng, A. W.-c. Fu, and J. Liu, "K-isomorphism: privacy preserving network publication against structural attacks," in *SIGMOD*. ACM, 2010, pp. 459–470.

[24] E. Zheleva and L. Getoor, "Preserving the privacy of sensitive relationships in graph data," in *Privacy, security, and trust in KDD*. Springer, 2008, pp. 153–171.

[25] M. Hay, G. Miklau, D. Jensen, D. Towsley, and P. Weis, "Resisting structural re-identification in anonymized social networks," *Proceedings of the VLDB Endowment*, vol. 1, no. 1, pp. 102–114, 2008.

[26] W. Wu, Y. Xiao, W. Wang, Z. He, and Z. Wang, "K-symmetry model for identity anonymization in social networks," in *EDBT*. ACM, 2010, pp. 111–122.

[27] G. Karypis and V. Kumar, "Analysis of multilevel graph partitioning," in *Supercomputing, 1995. Proceedings of the IEEE/ACM SC95 Conference*. IEEE, 1995, pp. 29–29.

[28] R. Chen, J. Shi, Y. Chen, and H. Chen, "Powerlyra: Differentiated graph computation and partitioning on skewed graphs," in *Proceedings of the Tenth European Conference on Computer Systems*. ACM, 2015, p. 1.

[29] J. E. Gonzalez, Y. Low, H. Gu, D. Bickson, and C. Guestrin, "Powergraph: distributed graph-parallel computation on natural graphs." in *OSDI*, vol. 12, no. 1, 2012, p. 2.

[30] Y. Asahiro, R. Hassin, and K. Iwama, "Complexity of finding dense subgraphs," *Discrete Applied Mathematics*, vol. 121, no. 1-3, pp. 15–26, 2002.

[31] Y. Asahiro, K. Iwama, H. Tamaki, and T. Tokuyama, "Greedily finding a dense subgraph," *Journal of Algorithms*, vol. 34, no. 2, pp. 203–221, 2000.

[32] A. Bhaskara, M. Charikar, E. Chlamtac, U. Feige, and A. Vijayaraghavan, "Detecting high log-densities: an $O(n^{1/4})$ approximation for densest k-subgraph," in *STOC*. ACM, 2010, pp. 201–210.

[33] U. Feige, D. Peleg, and G. Kortsarz, "The dense k-subgraph problem," *Algorithmica*, vol. 29, no. 3, pp. 410–421, 2001.

[34] D. Gibson, R. Kumar, and A. Tomkins, "Discovering large dense subgraphs in massive graphs," in *Proceedings of the 31st international conference on Very large data bases*. VLDB Endowment, 2005, pp. 721–732.

[35] J. R. Ullmann, "An algorithm for subgraph isomorphism," *JACM*, vol. 23, no. 1, pp. 31–42, 1976.

[36] L. Yuan, L. Qin, X. Lin, L. Chang, and W. Zhang, "Diversified top-k clique search," *The VLDB Journal*, vol. 25, no. 2, pp. 171–196, 2016.

[37] K. Liu and E. Terzi, "Towards identity anonymization on graphs," in *SIGMOD*. ACM, 2008, pp. 93–106.

[38] C.-H. Tai, P.-J. Tseng, S. Y. Philip, and M.-S. Chen, "Identity protection in sequential releases of dynamic networks," *IEEE transactions on Knowledge and Data Engineering*, vol. 26, no. 3, pp. 635–651, 2014.

[39] B. Zhou and J. Pei, "Preserving privacy in social networks against neighborhood attacks," in *ICDE*. IEEE, 2008, pp. 506–515.

[40] C.-H. Tai, P. S. Yu, D.-N. Yang, and M.-S. Chen, "Privacy-preserving social network publication against friendship attacks," in *SIGKDD*. ACM, 2011, pp. 1262–1270.

[41] M. Yuan, L. Chen, S. Y. Philip, and T. Yu, "Protecting sensitive labels in social network data anonymization," *IEEE Transactions on Knowledge and Data Engineering*, vol. 25, no. 3, pp. 633–647, 2013.

[42] N. Cao, Z. Yang, C. Wang, K. Ren, and W. Lou, "Privacy-preserving query over encrypted graph-structured data in cloud computing," in *ICDCS*. IEEE, 2011, pp. 393–402.

[43] J. Gao, J. X. Yu, R. Jin, J. Zhou, T. Wang, and D. Yang, "Neighborhood-privacy protected shortest distance computing in cloud," in *SIGMOD*. ACM, 2011, pp. 409–420.