Gao JU, Chen W, Xu JJ *et al.* An efficient framework for multiple subgraph pattern matching models. JOURNAL OF COMPUTER SCIENCE AND TECHNOLOGY 34(6): 1185–1202 Nov. 2019. DOI 10.1007/s11390-019-1969-x

An Efficient Framework for Multiple Subgraph Pattern Matching Models

Jiu-Ru Gao[1], Wei Chen[1], Jia-Jie Xu[1], *Member, CCF, ACM*, An Liu[1], *Member, CCF, ACM*
Zhi-Xu Li[1], *Member, CCF, ACM*, Hongzhi Yin[2], *Member, CCF, ACM*, and Lei Zhao[1,*], *Member, CCF, ACM*

[1]*School of Computer Science and Technology, Soochow University, Suzhou 215006, China*
[2]*School of Information Technology and Electrical Engineering, The University of Queensland, Brisbane 4072, Australia*

E-mail: jrgao@stu.suda.edu.cn; wchzhg@gmail.com; {xujj, anliu, zhixuli}@suda.edu.cn; db.hongzhi@gmail.com
zhaol@suda.edu.cn

Received December 14, 2018; revised September 12, 2019.

Abstract With the popularity of storing large data graph in cloud, the emergence of subgraph pattern matching on a remote cloud has been inspired. Typically, subgraph pattern matching is defined in terms of subgraph isomorphism, which is an NP-complete problem and sometimes too strict to find useful matches in certain applications. And how to protect the privacy of data graphs in subgraph pattern matching without undermining matching results is an important concern. Thus, we propose a novel framework to achieve the privacy-preserving subgraph pattern matching in cloud. In order to protect the structural privacy in data graphs, we firstly develop a k-automorphism model based method. Additionally, we use a cost-model based label generalization method to protect label privacy in both data graphs and pattern graphs. During the generation of the k-automorphic graph, a large number of noise edges or vertices might be introduced to the original data graph. Thus, we use the outsourced graph, which is only a subset of a k-automorphic graph, to answer the subgraph pattern matching. The efficiency of the pattern matching process can be greatly improved in this way. Extensive experiments on real-world datasets demonstrate the high efficiency of our framework.

Keywords subgraph pattern matching, k-automorphism, label generalization

1 Introduction

Usually we can use a graph to represent objects and their relationships. The increasing number of applications making use of graph data in recent years, such as disease transmission[1,2], communication patterns[3], and social networks[4−7], has promoted the development of graph data management, especially subgraph pattern matching. Subgraph pattern matching is traditionally defined in terms of subgraph isomorphism[8,9], which is an NP-complete problem[10]. It is often too strict to catch sensitive matches, as it requires matches to have the same topology with data graphs. The problem will hinder its applicability in some certain applications like social networks and crime detection. The family of graph simulation models provide a practical alternative to subgraph isomorphism by relaxing its matching conditions[11,12]. Our work focuses on multiple subgraph pattern matching models including strong simulation[13], strict simulation[14], and tight simulation[15]. These models are revisions of graph simulation, which impose more flexible constraints on topology in data graphs and retain cubic-time complexity.

Example 1. Consider a real-life social network shown in Fig.1. Each vertex in graph G represents an entity, such as a human resources person (HR_i), a development manager (DM_i), and a project manager (PM_i). Each directed edge in G indicates one recommendation relationship, e.g., edge $HR_1 \rightarrow PM_1$ repre-

Regular Paper

A preliminary version of the paper was published in the Proceedings of DASFAA 2018.

This work is supported by the National Natural Science Foundation of China under Grant No. 61572335, and the Natural Science Foundation of Jiangsu Province of China under Grant No. BK20151223.

*Corresponding Author

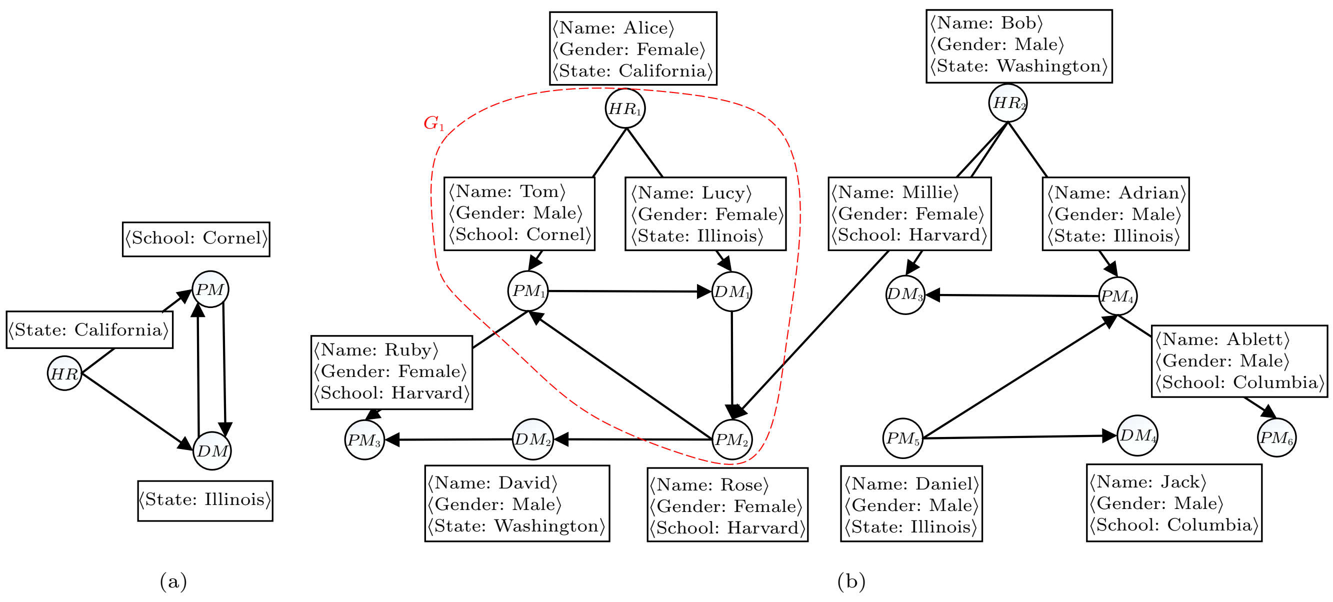

⟨School: Cornel⟩
⟨State: California⟩
PM
HR
DM
⟨State: Illinois⟩
(a)
⟨Name: Alice⟩ ⟨Gender: Female⟩ ⟨State: California⟩
G1
HR1
⟨Name: Tom⟩ ⟨Gender: Male⟩ ⟨School: Cornel⟩
⟨Name: Lucy⟩ ⟨Gender: Female⟩ ⟨State: Illinois⟩
PM1
DM1
⟨Name: Ruby⟩ ⟨Gender: Female⟩ ⟨School: Harvard⟩
PM3
DM2
PM2
⟨Name: David⟩ ⟨Gender: Male⟩ ⟨State: Washington⟩
⟨Name: Rose⟩ ⟨Gender: Female⟩ ⟨School: Harvard⟩
⟨Name: Bob⟩ ⟨Gender: Male⟩ ⟨State: Washington⟩
HR2
⟨Name: Millie⟩ ⟨Gender: Female⟩ ⟨School: Harvard⟩
⟨Name: Adrian⟩ ⟨Gender: Male⟩ ⟨State: Illinois⟩
DM3
PM4
⟨Name: Ablett⟩ ⟨Gender: Male⟩ ⟨School: Columbia⟩
PM5
DM4
PM6
⟨Name: Daniel⟩ ⟨Gender: Male⟩ ⟨State: Illinois⟩
⟨Name: Jack⟩ ⟨Gender: Male⟩ ⟨School: Columbia⟩
(b)

Fig.1. (a) Pattern graph Q and (b) original data graph G.

sents HR_1 recommends PM_1. Each entity has some attributes like "name", "gender", "state", and "school".

As shown in Fig.1, a headhunter wants to employ development manger DM to help project manger PM. A qualified candidate must live in Illinois and at the same time, he/she must recommend the PM and be recommended by the HR and the PM. The headhunter issues a subgraph pattern matching of Q over G, as shown in Fig.1. When subgraph isomorphism is taken, no match can be found. When it comes to strong simulation, we can find the subgraph G_1 is an appropriate match to pattern Q, since there exists a path (DM_1, PM_2, PM_1) from DM_1 to PM_1. Obviously, compared with subgraph isomorphism which imposes a very strict constraint on the topology of the matched graphs, strong simulation provides a more flexible constraint.

We can clearly find from example 1 that subgraph isomorphism returns the strictest matches for subgraph pattern matching in terms of topology and the problem is of high computation cost. In order to lower its time complexity, variants of the subgraph pattern matching models have been proposed[10,11,13−16].

Although storing large-data graphs in cloud can greatly save storage cost, it brings another inevitable challenge, i.e., how to process users' queries without compromising sensitive information in cloud[17]. The sensitive information in our work focuses on personal information like names, occupations and social security number[18]. We cannot make sure that the cloud platform is completely credible in many real scenarios; therefore the sensitive information in graphs may be disclosed. The main privacy leakage problem is the "identity disclosure" problem[19,20]. Assuming a graph G containing sensitive information is uploaded to the cloud platform, when an adversary can locate target t as a vertex v in G with a high probability, we say the entity t's identity is disclosed[19]. A naive anonymous approach is to remove all identifiable personal information from a data graph before uploading it to cloud. However, even though the data graph is uploaded without any sensitive information, it is still possible for an adversary to locate the target through structural attacks[20−23]. For example, if an attacker knows the structure (such as degree) around the target t, it is of a high probability that he/she can locate a vertex v according to the target t. All sensitive information associated with v will be compromised as a result. This is called structural attack[18,24,25]. Many methods have been proposed to protect the privacy of data graphs from multiple structural attacks. One typical approach is k-automorphism, which uses the symmetry of the published data graph[18]. For each vertex v in a k-automorphic graph, there are at least $k-1$ structurally equivalent counterparts. An adversary cannot distinguish v from other $k-1$ symmetric vertices, because there is no structural difference between them.

Uploading the original graph G to cloud directly will cause privacy leakage. To address the problem, we propose a basic solution which can protect the privacy of both data graphs and pattern graphs during the matching process. In this solution, we propose a k-automorphism model based method to protect structural privacy in data graphs. In order to protect the label privacy in both data graphs and pattern graphs, we apply a label generalization technique[17], where each vertex in the k-automorphic graph and pattern graph is replaced by a label group.

However, the basic solution suffers from the following limitation. During the generation of a k-automorphic graph, it may generate a large number of noise edges, which will result in more expensive storage cost and much larger communication overhead. Thus, we add some optimizations into the basic framework to achieve a more efficient framework. Firstly, we only upload the outsourced graph (the definition of outsourced graph is given in Subsection 4.2), which is only a subset of the k-automorphic graph, to cloud. Secondly, we re-design a cost-model based label generalization method which has been proved to greatly improve the efficiency of the pattern matching process in our experiments.

Contributions. Our work provides an efficient framework for multiple subgraph pattern matching models in cloud. In this paper, the cloud always offers correct computations without cheating, but it is curious about the information of the data graph. The main contributions of our work are summarized as follows.

• We propose an efficient framework to provide multiple subgraph pattern matching services while preserving private information in both data graphs and pattern graphs in public cloud.

• In order to reduce the search space in pattern matching process, we re-design a cost-model based label generalization method to select effective vertex label combinations for anonymizing labels in both data graphs and pattern graphs.

• We conduct extensive experiments on several real-world datasets to study the efficiency of our framework.

The rest of the paper is organized as follows. Section 2 narrates the related work. Section 3 gives the problem formulation. Section 4 describes the main solution. Section 5 reports the experimental analysis. Section 6 concludes the paper.

2 Related Work

Privacy Preserving. The question of how to publish information on graphs in a privacy-preserving way has been of interest for a number of years[21,26–28]. Most previous work focused on protecting data privacy from structural attacks[21,26,27]. Some of them assume that the adversary only launches one type of structural attack[21,26,27]. For example, Liu and Terzi[25] studied how to protect privacy in published data from degree attack only. However, an attacker can launch multiple types of structural attacks to identify the target in practice. Some privacy preserving techniques may cause the data graph losing of structure information in the original, which will lead to the infeasibility of subgraph pattern matching on published graphs or uploaded graphs[24]. Thus, Zou *et al.*[18] proposed the k-automorphism based framework. Each vertex in a k-automorphic graph has at least $k-1$ counterparts so that it is hard for an adversary to identify the vertex from others. The k-automorphism model can protect privacy of data graph from multiple structural attacks. It can also significantly preserve the integrity of the data since the model does not need to delete any vertices or edges from the data graph. Chang *et al.* proposed a framework which can well protect the privacy in both data graph and pattern graph[17]. However, the framework only adapts to subgraph isomorphism and it is significantly needed to propose a privacy-preserving framework for more subgraph pattern matching models.

Differential privacy[29,30] is also an essential and prevalent model that has been widely explored in recent decades. However, due to the perturbation introduced to the data graph, these techniques[28,31,32] only adapt to finding statistics information of a graph. They are not feasible in answering subgraph pattern matching queries exactly. Zhang *et al.*[33] defined an isomorphic graph possessing similar statistical properties with the original graph. However, their work only returns subgraph counts instead of matching subgraphs; therefore we cannot determine the correctness of the query answers.

Graph Simulation. Subgraph pattern matching is typically defined in terms of subgraph isomorphism[8,9]. Subgraph isomorphism is an NP-complete problem[34] since it returns the strictest matches for subgraph pattern matching in terms of topology. Some previous work focuses on subgraph similarity matching on large graphs[35,36]. Given a query graph Q and a data graph G, subgraph similarity matching is to retrieve all matches of Q in G with the number of missing edges bounded by a given threshold ε. Also, the family of graph simulation algorithms has been considered[10,11,13–16] to lower the complexity of graph isomorphism. Fan *et al.*[11] extended simulation by allowing bounds on the number of hops in pattern graphs and further and proposed bounded simulation. Fan *et al.* extended it by incorporating regular expressions as edge constraints on pattern graphs[16]. Both the two extensions of simulation are in cubic-time.

Nevertheless, the lower complexity comes with the price that two extentions of simulation do not preserve the topology of data graphs and yield false matches. Thus, Ma *et al.*[13] proposed the notation of strong simulation by enforcing two additional conditions: the duality to preserve the parent relationships and the locality to eliminate excessive matches. Strong simulation is capable of capturing the topological structures of pattern and data graphs, and it retains the same cubic-time complexity of former extensions of graph simulation[10]. However, it is still computationally expensive for very large data graphs. Thus, Fard *et al.* introduced a new model named strict simulation[14], which is more scalable and preserves the important properties of strong simulation. Strict simulation reduces the computation time of strong simulation by decreasing the size of balls, but the number of the balls remains the same. Moreover, it is still desirable to shrink the size of the balls in terms of both computation time and the quality of the matching results. Therefore Fard *et al.* proposed tight simulation[15]. Tight simulation not only decreases the size of balls in comparison to strict simulation, but also reduces the number of balls.

Compared with our previous work[37], this paper provides a privacy-preserving framework for multiple simulation models. The framework can be applied to strong simulation, strict simulation and tight simulation and can greatly decrease the running time of the subgraph pattern matching in cloud.

3 Problem Formulation

In this section, we first present the basic notations and definitions frequently used in this paper. Then we give a definition of our problem.

We model a social network as an attributed graph[17], $G = \{V(G), E(G), L_G(V(G))\}$, where $V(G)$ is the set of vertices, $E(G)$ is the set of edges, and $L_G(V(G))$ is the set of vertex labels. The notational conventions of this paper are summarized in Table 1.

Definition 1 (Path). *A directed path p is a sequence of nodes $(v_1, v_2, ..., v_n)$, where $i \in [1, n-1]$ and (v_i, v_{i+1}) is an edge in graph G. The number of edges in a path p is the length of p, denoted by $len(p)$.*

Definition 2 (Distance and Diameter). *Consider two nodes u, v in graph G, the distance from u to v is the length of the shortest undirected path from u to v, denoted by $dis(u, v)$. The diameter of the connected graph G is defined as the longest distance of all pairs of nodes in G, denoted by d_G. More specifically, $d_G = \max\{dis(u, v)\}$ for all nodes u, v in graph G.*

Table 1. Notations

Notation	Description
d_Q	Diameter of pattern Q
G	Original data graph
G^*	"Undirected" data graph
G^k	Data graph released by the k-automorphism model
G^o	Outsourced data graph
$\widehat{G}[v, d_Q]$	Ball with center v and radius d_Q
Q	Original pattern graph
Q^o	Anonymous pattern graph of Q
$r(Q, G)$	Set of graph pattern matches of Q over G
$dis(u, v)$	Distance between u and v

For example, the distance between HR and PM in pattern graph Q in Fig.1 is 1 since HR can directly arrive PM. The diameter of Q is 1 because the longest distance of all pairs of nodes in Q is 1.

Definition 3 (Ball). *For a node v in graph G, a ball is a subgraph of G, where v is the center node and r is the radius, denoted by $\widehat{G}[v, r]$. For all nodes u in $\widehat{G}[v, r]$, the shortest distance between u and v should satisfy $dis(u, v) \leqslant r$ and edges must exactly appear in graph G over the same node set.*

Considering the pattern Q and the data graph G in Fig.1, we can figure out that $d_Q = 1$ according to Definition 2. If we take the vertex DM_1 as the center node and d_Q as the radius, then we can obtain the ball $\widehat{G}[DM_1, d_Q]$ (i.e., G_1), which is a subgraph of G based on Definition 3.

Definition 4 (Subgraph Isomorphism). *Subgraph isomorphism is the most traditional model for subgraph pattern matching. It preserves most restrictive topological features of the query graph. Given a pattern graph Q and a subgraph G_s of data graph G, if there exists a bijective function f from vertices of Q to the vertices in G_s such that*:

1) *for each pattern vertex u in Q, u and $f(u)$ have same labels and*

2) *there exists an edge (u, u') in Q if and only if $(f(u), f(u'))$ is an edge in G_s,*

then G_s is a match of via Q subgraph isomorphism.

For example, a subgraph of data graph containing vertices $\{A_1, B_1, C_1\}$ and edges $A_1 \rightarrow B_1$, $B_1 \rightarrow A_1$ and $B_1 \rightarrow C_1$ in Fig.2 is a subgraph isomorphic match of the pattern graph in Fig.2.

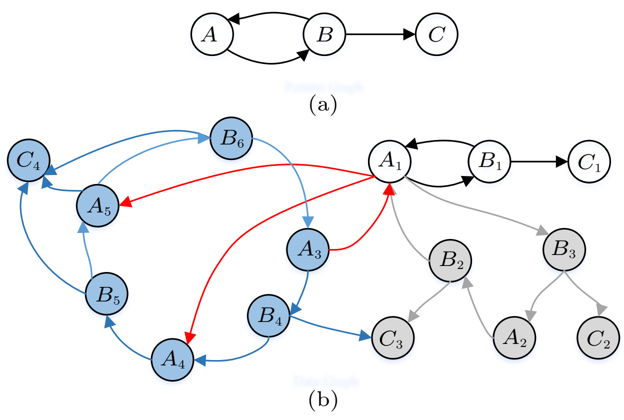


Fig.2. Example for different models. (a) Pattern graph. (b) Data graph.

Definition 5 (Graph Simulation). *Given a pattern graph $Q\{V(Q), E(Q), L_Q(V(Q))\}$ and a data graph $G\{V(G), E(G), L_G(V(G))\}$, a binary relation $R \subseteq V(Q) \times V(G)$ is a match if*

1) *for each $(u, v) \in R$, u and v have same labels and*

2) *for each edge $(u, u') \in E(Q)$, there exists an edge (v, v') in $E(G)$ such that $(u', v') \in R$.*

The matching result is a maximum match set of vertices. Graph G matches pattern Q via graph simulation, if there exists a total match relation M that for each $u \in V(Q)$, there exists $v \in V(G)$ and $(u, v) \in M$.

Definition 6 (Dual Simulation). *Graph simulation only preserves child relationships of each vertex in pattern graph Q. Dual simulation improves the matching results of graph simulation by taking the parent relationships into consideration.*

Pattern graph $Q\{V(Q), E(Q), L_Q(V(Q))\}$ matches $G\{V(G), E(G), L_G(V(G))\}$ via dual simulation, if

1) *Q matches G via graph simulation with a binary match relation $R \subseteq V(Q) \times V(G)$, and*

2) *for each edge $(u', u) \in E(Q)$, there exists an edge (v', v) in $E(G)$ such that $(u', v') \in R$.*

Definition 7 (Strong Simulation[13]). *Given a data graph $G = \{V(G),\ E(G),\ L_G(V(G))\}$ and a pattern graph $Q = \{V(Q),\ E(Q),\ L_Q(V(Q))\}$, if there exists a node v in G_s of G such that:*

1) *Q is a match result of G_s via dual simulation with maximum match relation S and G_s is the match graph towards match relation S;*

2) *G_s is contained in $\widehat{G}[v, d_Q]$, where d_Q is the diameter of pattern graph Q,*

Q is a subgraph match to G via strong simulation.

Definition 8 (Strict Simulation[14]). *Pattern graph $Q = \{V(Q),\ E(Q),\ L_Q(V(Q))\}$ matches data graph $G = \{V(G),\ E(G),\ L_G(V(G))\}$ via strict simulation if there exists a vertex $v \in V(G)$ such that:*

1) *$G_d = \{V(G_d),\ E(G_d),\ L_{G_d}(V(G_d))\}$ is the match result of Q over G via dual simulation and $v \in V(G_d)$;*

2) *Q is the dual match result of $\widehat{G_d}[v, d_Q]$, where $\widehat{G_d}[v, d_Q]$ is extracted from G_d and d_Q is the diameter of pattern Q;*

3) *v is contained in the matching result.*

Definition 9 (Tight Simulation[15]). *Pattern graph $Q = \{V(Q),\ E(Q),\ L_Q(V(Q))\}$ matches data graph $G = \{V(G),\ E(G),\ L_G(V(G))\}$ via tight simulation if there exist vertices $u \in Q$ and $u' \in G$ such that:*

1) *u' is the matching vertex of u in dual match relation R_d;*

2) *u is the center of Q with the highest defined selectivity;*

3) *$G_d = \{V(G_d),\ E(G_d),\ L_{G_d}(V(G_d))\}$ is the match result of Q over G via dual simulation and Q is the dual match result of $\widehat{G_d}[u', r_Q]$, where $\widehat{G_d}[u', r_Q]$ is extracted from $G_d = \{V(G_d),\ E(G_d),\ L_{G_d}(V(G_d))\}$ and r_Q is the diameter of pattern Q;*

4) *u' is contained in the matching result.*

There is an example to show the difference in the results of different pattern matching models in Fig.2. In this example, all vertices in the data graph will remain in the dual match graph. The subgraph containing vertices $\{A_1,\ B_1,\ C_1\}$ and edges $A_1 \rightarrow B_1$, $B_1 \rightarrow A_1$ and $B_1 \rightarrow C_1$, will be the matching result when subgraph isomorphism is taken, since the subgraph has the same topology with the pattern in Fig.2. In strong and strict simulation, a ball with radius 2 will be created for any vertex in the data graph. For the ball centered at vertex A_1, strong simulation results in a subgraph containing all vertices in the data graph, while strict simulation results in a subgraph containing vertices $\{A_1, A_2, B_1, B_2, B_3, C_1, C_2, C_3\}$ and edges between them.

In tight simulation, the center of the pattern is vertex B and it will be picked as the candidate vertex. Therefore, the ball centered with vertices $\{B_1,\ B_2,\ B_3,\ B_4,\ B_5,\ B_6\}$ and radius 1 will be the matching candidates. Only the ball centered at vertex B_1 which contains vertices $\{A_1, B_1, C_1\}$ can be a matching result in tight simulation. Compared with strict simulation, the matching results of tight simulation are the closest to subgraph isomorphism, since the matching results of tight simulation are subgraphs of the corresponding results of strict simulation and they always contain all the subgraph isomorphic matches.

Problem Definition. Given a data graph G and a pattern graph Q, our work is to find all subgraph pattern matches of Q over G without compromising the privacy of graphs through the cloud server. We propose a framework which can protect both label privacy and structure privacy in graphs. The framework can apply to multiple subgraph matching models including strong simulation, strict simulation, and tight simulation.

Our work aims to protect the privacy of data graph and pattern graph against cloud. When the cloud server returns the matching results to the client, the client has the ability to de-anonymize, filter, and verify the results.

4 Privacy-Preserving Framework

Our framework mainly consists of three parts, and they are privacy preserving, subgraph pattern matching, and result processing. In the privacy preserving process, we consider both structural privacy and label privacy in data graphs and pattern graphs. Given a data graph G, we firstly transform the original graph G to an "undirected" graph G^*. During the process, if an edge $u \rightarrow v$ is unidirectional, we will add an edge $v \rightarrow u$. For example, we add an edge $PM_2 \rightarrow DM_1$ for the edge $DM_1 \rightarrow PM_2$ in Fig.1. Then, we can use the k-automorphism model to generate graph G^k, where k = 2 in Fig.3. On the other hand, to protect label privacy, we apply a cost-model based label generalization technique[17], where each vertex label in G^k and Q is replaced by a label group. The mapping between label groups and vertex labels is given in the label correspondence table (LCT), presented in Fig.3(a).

Since G^k will be very large, we only upload part of it, i.e., G^0 to cloud. Next, the cloud executes subgraph pattern matching of Q^0 over G^0 to obtain $r(Q^0, G^0)$ and transmits it to the client side. On the basis of k-automorphic functions F_{k_i} $(i = 1, 2, ..., k-1)$, the client can firstly compute $r(Q^0, G^k)$ according to $r(Q^0, G^0)$. Then, it filters out false positives based on the original data graph G and pattern Q to derive $r(Q, G)$. Note that we assume the client is the data owner who has access to the original graph G for the filtering step. The entire process of our framework is shown in Fig.4.

4.1 Privacy Preserving

In order to provide a privacy-preserving matching process, we need to consider two aspects: one is the structural privacy, and the other is the label privacy.

4.1.1 *Structural Privacy*

We firstly develop a novel approach based on k-automorphism model to protect structural privacy in data graphs. When a directed data graph G is given, we firstly transform it to an "undirected" graph G^* by introducing noise edges. Then we convert G^* into graph G^k, where G^k satisfies the k-automorphic graph model.

Definition 10 (k-Automorphic Graph). *A k-automorphic graph G^k is defined as G^k = $\{V(G^k), E(G^k)\}$, where $|V(G^k)|$ is the number of*

Label Group	Labels
A	HR, PM, DM
B	Female, Male
C	California, Illinois, Washington
D	Harvard, Conell, Columbia

(a)

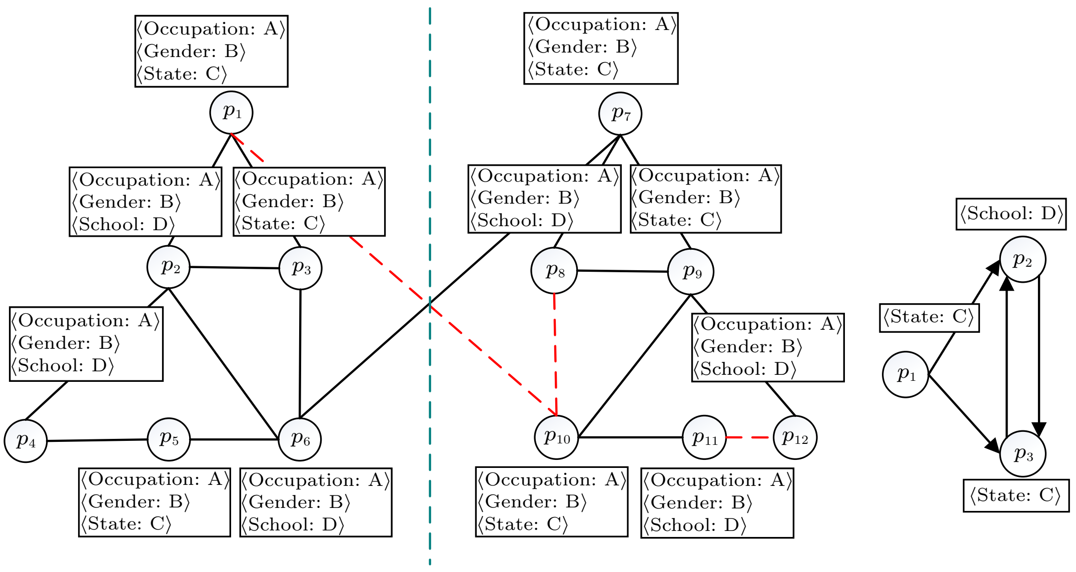
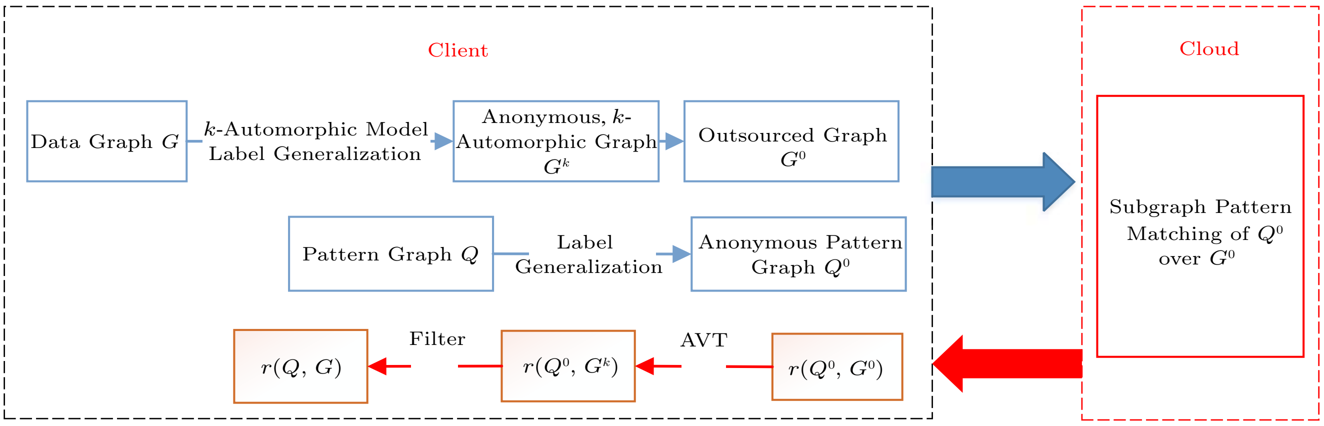


Fig.3. k-automorphic graph G^k and anonymous pattern Q^0. (a) Label correspondence table (LCT). (b) Graph G^k. (c) Anonymous pattern Q^0.



Client
Cloud
Data Graph G
k-Automorphic Model Label Generalization
Anonymous, k-Automorphic Graph G^k
Outsourced Graph G^0
Pattern Graph Q
Label Generalization
Anonymous Pattern Graph Q^0
Subgraph Pattern Matching of Q^0 over G^0
$r(Q, G)$
Filter
$r(Q^0, G^k)$
AVT
$r(Q^0, G^0)$

Fig.4. Privacy-preserving framework.

$V(G^k)$. $|V(G^k)|$ *can be divided into* k *blocks and each block has* $\lceil\frac{|V(G^k)|}{k}\rceil$ vertices.

A k-automorphic graph G^k has k blocks and each block is isomorphic to other blocks. Intuitively, for any vertex v in a k-automorphic graph G^k, there are $k-1$ symmetric vertices. An adversary can hardly distinguish v from its structurally equivalent counterparts. Thus, the structural privacy in data graphs can be well preserved. According to Definition 10, we can transform G^* to a k-automorphic graph G^k as follows.

Firstly, we adopt the METIS algorithm[17,38] to partition the graph G^* into k blocks. We choose the multilevel k-way partitioning schemes[39]. There are three main phases in the multilevel k-way partitioning algorithm. They are coarsening phase, initial partition phase, and uncoarsening phase. As shown in Fig.5, during the coarsening phase, the size of the graph is decreased. A k-way partitioning of the smaller graph is computed during the initial partitioning phase. During the uncoarsening phase, the partitioning is refined as it is objected to the larger graphs. In order to guarantee that each block has exactly $\lceil\frac{|V(G^k)|}{k}\rceil$ vertices, some noise vertices will be introduced if $|V(G^*)|$ cannot be divided into k blocks. Finding the optimal block alignment is an NP-hard problem (even in the case of no attributes), and it is more complicated while considering the attributes of vertices. Thus, Zou *et al.* proposed an efficient method to build the alignment vertex table (AVT) after the partition[18]. The method will firstly choose the largest same degree with the same label in different blocks. Then it will breadth-first traverse different blocks and pair up vertices with the same label and/or the same (or similar) degrees. For example, we divide the graph G^k in Fig.3(b) into two blocks and build the corresponding AVT, which is presented in Table 2. Each row in AVT denotes they are symmetric vertices, such as p_1 and p_7 in Fig.3(b). Each column in AVT contains the vertices in one block, such as (p_1, p_2, p_3, p_4, p_5, p_6) in the first block of G^k. According to the AVT, we define the k-automorphic function F_{k_1}, as shown in Fig.6.

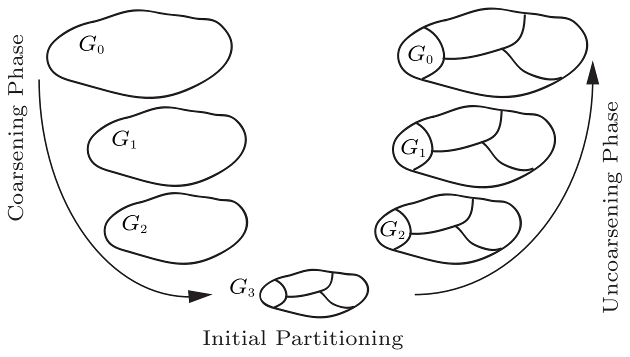


Fig.5. Various phases of the multilevel k-way partitioning algorithm.

Table 2. Alignment Vertex Table (AVT)

Vertices in B_0	Vertices in B_1
p_1	p_7
p_2	p_9
p_3	p_8
p_4	p_{12}
p_5	p_{11}
p_6	p_{10}

Secondly, we perform block alignment and edge copy[18] to obtain the k-automorphic graph G^k. For example, we can obtain two isomorphic blocks: B_0(p_1, p_2, p_3, p_4, p_5, p_6) and B_1(p_7, p_8, p_9, p_{10}, p_{11}, p_{12}) by adding noise edges (p_8, p_{10}) and (p_{11}, p_{12}) via block alignment in Fig.3. According to the crossing edge (p_6, p_7) between two blocks, we add an edge (p_1, p_{10}) based on the edge copy techniques.

$$\begin{array}{ll}
F_{k_1}(p_1) = p_7 & F_{k_1}(p_7) = p_1 \\
F_{k_1}(p_2) = p_9 & F_{k_1}(p_8) = p_3 \\
F_{k_1}(p_3) = p_8 & F_{k_1}(p_9) = p_2 \\
F_{k_1}(p_4) = p_{12} & F_{k_1}(p_{12}) = p_4 \\
F_{k_1}(p_5) = p_{11} & F_{k_1}(p_{11}) = p_5 \\
F_{k_1}(p_6) = p_{10} & F_{k_1}(p_{10}) = p_6
\end{array}$$

Fig.6. Automorphic function.

4.1.2 *Label Privacy*

Since the k-automorphism model based method can only protect the structural privacy of the original graph G, we define a cost-model based label generalization method to protect label privacy in both data graph G and pattern graph Q. The method considers two factors: label matching and searching space, while estimating the number of candidates of a vertex u in Q^0, denoted as $sim(u)$.

According to the definition of strong simulation[13], strict simulation[14], and tight simulation[15], when a vertex v in graph G^k matches vertex u in pattern Q^0, it must firstly contain u's label groups. We let $|V_g(G^k, i)|$ and $|V_g(Q^0, i)|$ denote the set of vertices with the label group i in G^k and Q^0 that are obtained after the label generalization respectively. Then, we define:

$$P^g_{G^k}(i) = \frac{|V_g(G^k, i)|}{|V(G^k)|}, \quad P^g_{Q^0}(i) = \frac{|V_g(Q^0, i)|}{|V(Q^0)|}.$$

$P^g_{G^k}(i)$ and $P^g_{Q^0}(i)$ estimate the probability of a vertex in G^k and Q^0 having an i-th label group after the label generalization, respectively. Then, the estimating number of vertices that can match vertex u in Q^0 while considering label matching can be defined as follows:

$$L = |V(G^k)| \sum_{i=1}^{\alpha} P^g_{G^k}(i) \times P^g_{Q^0}(i).$$

Next we need to consider the searching space of checking whether each of u's parent vertices and child vertices can find matching vertices. We define the average in-degree $D_i(G^k)$ and the average out-degree $D_o(G^k)$ to represent the in-degree and the out-degree of vertex v (u's matching vertex) respectively. Similarly, $D_i(Q)$ and $D_o(Q)$ represent the in-degree and the out-degree of vertex u in Q^0 separately. Note that $D_o(Q^0) = D_o(Q)$ and $D_i(Q^0) = D_i(Q)$. Therefore, the maximum potential searching space of u's first child vertex is $D_o(G^k)^{2d_Q}$, and that of the second child vertex is $(D_o(G^k) - 1)D_o(G^k)^{2d_Q-1}$. Thus, the total searching space of u's child vertices can be estimated as $D_o(G^k)^{2d_Q} \times (D_o(G^k) - 1)D_o(G^k)^{2d_Q-1} \cdots (D_o(G^k) - D_o(Q) + 1)D_o(G^k)^{2d_Q-1}$. We estimate it as $D_o(G^k)^{D_o(Q)} \times D_o(G^k)^{(2d_Q-1)^{D_o(Q)}}$ for simplicity, i.e., $D_o(G^k)^{D_o(Q)+(2d_Q-1)^{D_o(Q)}}$. In the same way, we can define the searching space of u's parent vertices as $D_i(G^k)^{D_i(Q)+(2d_Q-1)^{D_i(Q)}}$. We define sum_o and sum_i separately to represent $D_o(G^k)^{D_o(Q)+(2d_Q-1)^{D_o(Q)}}$ and $D_i(G^k)^{D_i(Q)+(2d_Q-1)^{D_i(Q)}}$. Thus, the estimation of searching space S can be defined as follows:

$$S = \left(D_o(G^k)\left(\sum_{i=1}^{\alpha} P^g_{G^k}(i)P^g_{Q^0}(i)\right)\right)^{sum_o} \times \left((D_i(G^k)\left(\sum_{i=1}^{\alpha} P^g_{G^k}(i)P^g_{Q^0}(i)\right)\right)^{sum_i}.$$

The searching space of tight simulation is smaller than S since the radius of a potential ball is possibly equal to the radius of Q. Thus in the computation of searching space in different pattern matching models, we consider the maximum amount of calculation.

We assume the total labels in original graph G can be divided into α groups, and each group contains θ different labels without loss of generality. We define $\langle p_1, p_2, p_3, ..., p_{\alpha\theta}\rangle$ to form a permutation of $\langle 1, 2, 3, ..., \alpha\theta\rangle$. According to [17], we can obtain that $P^g_{G^k}(i) \leqslant \sum_{j=1}^{\theta} P_G(p_{\theta(i-1)+j})$.

Thus, we can define the cost model:

$$\begin{aligned}
&|sim(u)| \\
&= L \cdot S \\
&= |V(G^k)|(\sum_{i=1}^{\alpha} P^g_{G^k}(i)P^g_{Q^0}(i)) \times \\
&\quad (D_o(G^k)(\sum_{i=1}^{\alpha} P^g_{G^k}(i)P^g_{Q^0}(i)))^{sum_o} \times \\
&\quad (D_i(G^k)(\sum_{i=1}^{\alpha} P^g_{G^k}(i)P^g_{Q^0}(i)))^{sum_i} \\
&= |V(G^k)|D_o(G^k)^{sum_o} \times D_i(G^k)^{sum_i} \times \\
&\quad (\sum_{i=1}^{\alpha} P^g_{G^k}(i)P^g_{Q^0}(i))^{sum_o+sum_i+1} \\
&\leqslant |V(G^k)|D_o(G^k)^{sum_o} \times D_i(G^k)^{sum_i} \times \\
&\quad (\sum_{i=1}^{\alpha}(\sum_{j=1}^{\theta} P_G(p_{\theta(i-1)+j})) \\
&\quad (\sum_{j=1}^{\theta} P_Q(p_{\theta(i-1)+j})))^{sum_o+sum_i+1}.
\end{aligned} \qquad (1)$$

According to the cost model in (1), an effective permutation of $\langle 1, 2, 3, ..., \alpha\theta\rangle$, i.e., $\langle p_1, p_2, p_3, ..., p_{\alpha\theta}\rangle$, can decrease the cost of the searching space of pattern graph Q over G. We choose the component that concerns the label combination to define label combination cost.

$$cost(L) = \sum_{i=1}^{\alpha}(\sum_{j=1}^{\theta} P_G(p_{\theta(i-1)+j})) \times (\sum_{j=1}^{\theta} P_Q(p_{\theta(i-1)+j})). \quad (2)$$

There is an iterative solution that can explore the optimal permutation to decrease $cost(L)$ according to (2). At first, a random label combination is generated. For example, we define $\theta = 2$ and firstly we combine *California* and *Harvard* in Fig.1 randomly as a label group A, and *Male*, *Illinois* and *Cornel* as a label group B. Then, we try to swap two labels in two different label groups for each iteration randomly. For example, we can swap *California* and *Male* and compute the $cost(L)$ according to (2). If $cost(L)$ becomes smaller, we will keep the swap; otherwise, we will ignore that. We consider all possible swap sequentially and once there is no swap that can lead to a smaller cost, the iteration stops and we obtain the effective combination.

4.2 Subgraph Pattern Matching

Given a data graph $G = \{V(G), E(G), L_G(V(G))\}$ and a pattern graph $Q = \{V(Q), E(Q), L_Q(V(Q))\}$, Q is a subgraph match to G via strong simulation, if there exists a node u in Q and a connected subgraph G_s of G such that:

1) there exists a match relation R, and for each pair (u, v) in R:

a) $L_Q(u) \subseteq L_{G_s}(v)$;

b) $\forall\ (u', u) \in E(Q)$, there exists a path $(v', ..., v)$ in $E(G_s)$;

c) $\forall\ (u, u') \in E(Q)$, there exists a path $(v, ..., v')$ in $E(G_s)$;

2) G_s is contained in the ball $\widehat{G}[v, d_Q]$, where d_Q is the diameter of pattern Q.

Strict simulation is a novel modification of strong simulation which not only improves its performance but also maintains a better quality of results because of its revised definition of locality[14]. Compared with strong simulation, strict simulation creates balls from the dual result match graph rather than from the original graph. For strict simulation, the match relation of dual simulation is computed first, and then a ball $\widehat{G}[v, d_Q]$ is created for each vertex contained in the dual match set. The members of the ball are selected regardless of their relationship in pairs of dual match set.

Strict simulation decreases the size of the balls to reduce the computation time of strong simulation, while tight simulation[15] not only decreases the size of balls, but also reduces the number of balls. In the reprocessing of tight simulation, a single vertex $u \in Q$, is chosen as candidate match to the center of a potential ball in the data graph G. The radius of a potential ball is defined as the longest distance between u and any other vertex in Q. In the phase of ball creation, only those vertices in the data graph which are contained in the dual match set of u would be selected as the center of balls.

After obtaining an anonymous k-automorphic graph G^k, a basic solution is to upload G^k to cloud directly. However, G^k is larger than the original graph G since G^k contains a large number of noise edges. Therefore, we only upload the outsourced graph, which is only a subset of G^k, denoted as G^0, to the cloud platform. The definition of G^0 is given below.

Definition 11 (Outsourced Graph). *An outsourced graph is defined as* $G^0 = \{V(G^0), E(G^0), L_{G^0}(V(G^0))\}$ *where* 1) $V(G^0)$ *is the set of vertices in the first block of* G^k *(i.e., block* B_0*), denoted as* $V(B_0)$*, together with their neighbors within* $2d_Q$*-hops, denoted as* $V(N_{2d_Q})$; 2) $E(G^0)$ *is the set of edges that connect vertices within* $V(B_0)$ *and vertices between* $V(B_0)$ *and* $V(N_{2d_Q})$; 3) $L_{G^0}(V(G^0))$ *is the set of vertex labels in graph* G^0.

According to Definition 11, we can generate an outsourced graph G^0 based on the graph G^k and upload it to cloud. For example, an outsourced graph G^0 (as shown in Fig.7) can be generated based on graph G^k in Fig.3. Although G^0 is a part of G^k, we can easily recover G^k based on G^0 together with k-automorphic functions F_{k_i} $(i = 1, 2, ..., k-1)$.

When a pattern Q is given at a client side, we first generalize its vertex labels by the label generalization technique introduced in Subsection 4.1.2 to form Q^0 and submit Q^0 to cloud. According to the matching process of strong simulation, strict simulation, and tight simulation, we define $|V(G^0)|$ contains the vertices in the first block B_0 of the k-automorphic graph G^k and the vertices within their $2d_Q$-hops neighbors. Since each match iteration aims to find the maximum perfect subgraph match and each subgraph match is involved in a ball $\widehat{G}[w, d_Q]$ (w is an arbitrary vertex in G^0), the operation can guarantee that all subgraph matches could be found.

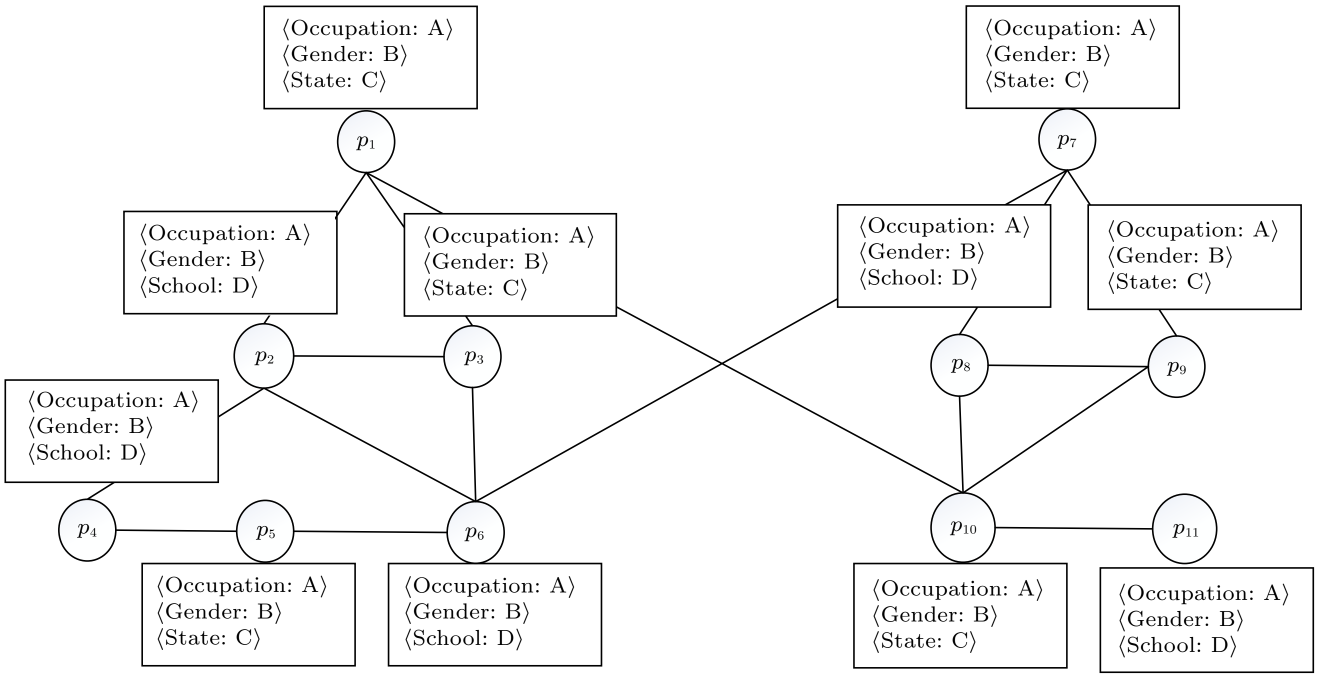


Fig.7. Outsourced graph G^0 for G^k.

An algorithm for the subgraph pattern matching is designed as Algorithm 1. At first, a *MatchSet* would contains all dual match sets (line 2 in Algorithm 1). For each *match* in *MatchSet*, if it satisfies the simulation conditions (strong simulation, strict simulation, or tight simulation), the graph G_s constructed for *match* would be added to the results $r(Q^0, G^0)$. Else, *match* would be removed from *MatchSet* (lines 3–8 in Algorithm 1).

Algorithm 1. Subgraph Pattern Match

```
  Input: $Q^0$ and $G^0$
  Output: $r(Q^0, G^0)$
1 $r(Q^0, G^0) := \emptyset$;
2 Make a MatchSet of the dual simulation results;
3 for match $\in$ MatchSet do
4     if match satisfies the simulation conditions then
5         $G_s$ := construct match graph for match;
6         $r(Q^0, G^0) := r(Q^0, G^0) \cup G_s$;
7     else
8         Remove match from MatchSet;
9 return $r(Q^0, G^0)$
```

4.3 Result Processing

When G^0 and Q^0 are uploaded, the cloud executes subgraph pattern matching via strong simulation, strict simulation, or tight simulation to obtain the matching result $r(Q^0, G^0)$ and transmits it to client. There are two steps for the client side to process the result.

Firstly, the client computes $r(Q^0, G^k)$ based on $r(Q^0, G^0)$ together with k-automorphic functions F_{k_i} ($i = 1, 2, ..., k-1$) (lines 1–3 in Algorithm 2). For each subgraph G_s in $r(Q^0, G^0)$, we can compute $F_{k_i}(G_s)$ ($i = 1, 2, ..., k-1$) and add them to $r(Q^0, G^k)$. Then, we obtain the final $r(Q^0, G^k)$ by adding $r(Q^0, G^0)$ (line 4 in Algorithm 2).

Secondly, the client needs to filter out the false matches in $r(Q^0, G^k)$ according to the original data graph G and pattern Q. For each matching subgraph G_s in $r(Q^0, G^k)$, if there exist vertices that are not contained in graph G or whose labels cannot match those of the corresponding vertices in the original pattern Q (we have anonymized the vertex labels in pattern Q via a label generalization method), then we remove them from G_s (lines 7–11 in Algorithm 2). Note that we have introduced noise edges when generating "undirected" graph G^* and k-automorphic graph G^k, if G_s contains edges that do not exist in the original graph G, then we remove them from G_s (lines 12–14 in Algorithm 2). When all the noise vertices or edges and unmatch vertices are filtered out from G_s, we need to consider whether G_s is a candidate. We define that if there exists a subgraph which is a *match* (meeting the requirements of strong simulation, strict simulation or tight simulation) to pattern Q in G_s, it is a right positive and we need to add it to $r(Q,G)$ (lines 15 and 16 in Algorithm 2).

Algorithm 2. Result Processing Algorithm

```
   Input: $r(Q^0,G^0)$ and AVT
   Output: $r(Q,G)$
1  $r(Q^0,G^k) := \emptyset$;
2  for $i := 1$ to $k-1$ do
3      $r(Q^0,G^k) := r(Q^0,G^k) \cup F_{k_i}(r(Q^0,G^0))$;
4  $r(Q^0,G^k) := r(Q^0,G^k) \cup r(Q^0,G^0)$;
5  $r(Q,G) := \emptyset$ ;
6  for each subgraph $G_s \in r(Q^0,G^k)$ do
7      for each vertex $v \in V(G_s)$ do
8          if $v \notin V(G)$ then
9              Remove node $v$ from $G_s$;
10         else if $L_G(v)$ does not match the corresponding
           vertex on pattern $Q$ then
11             Remove node $v$ from $G_s$;
12     for each edge $e \in E(G_s)$ do
13         if $e \notin E(G)$ then
14             Remove edge $e$ from $G_s$;
15
16     if $G_s$ contains the connected component that
       matches to pattern $Q$ then
17         $r(Q,G) := r(Q,G) \cup G_s$;
18
19 return $r(Q,G)$
```

Complexity Analysis. It takes $O(|E|\log k)$ time to partition the graph into k blocks and $O(\sum_{i=1}^{k}|E(P_j)|)$ time to align blocks where P_j $(j=1,2,...,k)$ indicates different blocks. The time complexity of edge copy and constructing AVT is $O(|E|)$ and $O(\text{Max}(|E(P_j)|+|V(P_j)|))$ respectively. The time complexity of subgraph pattern matching and result processing in the client is both $O(|V|(|V|+(|V(Q)|+|E(Q)|)(|V|+|E|)))$ since the result processing phase needs to check whether the subgraph is a candidate. We use the adjacency list to store the graph and the space complexity is $O(|V|+|E|)$.

5 Experimental Study

The main concern in subgraph pattern matching is its time efficiency since the subgraph isomorphism is an NP-complete problem and researchers have been making effort to improve the efficiency of subgraph matching queries. It is particularly important when considering a privacy-preserving framework for subgraph pattern matching since the pretreatment of original data graph will cause many noise edges or vertices. In this paper, we propose an efficient framework to achieve privacy-preserving subgraph pattern matching in cloud; thus the efficiency is the focus of our experimental study.

In our framework, both the label privacy and the structure privacy can be well protected; thus it can significantly preserve the sensitive information in the graph from multiple structural attacks[18,24,25]. Even if an attacker accesses the graph in cloud illegally at worst, it is still difficult for him/her to obtain the information he/she wants to get.

5.1 Datasets and Setup

We evaluate our method in three real-world datasets in our experiments. The statistics of these datasets are given in Table 3.

Table 3. Real-World Data Graphs

Dataset	$\|V\|$	$\|E\|$	Number of Labels
p2p-Gnutella08	6 301	20 777	62
Brightkite_edges	58 228	428 156	134
Web-NotreDame	325 729	1 090 108	208

*p2p-Gnutella08*①. p2p-Gnutella08 is a sequence of snapshots of the Gnutella peer-to-peer file sharing network collected in August 8, 2002. Nodes represent hosts in the Gnutella network topology and edges represent connections between the Gnutella hosts.

*Brightkite_edges*②. Brightkite_edges is the friendship network collected using Brightkite's public API. Nodes correspond to users having checked-in Brightkite and directed edges correspond to relationships among them.

*Web-NotreDame*③. Web-NotreDame is a web graph collected in 1999. Nodes represent pages from University of Notre Dame and directed edges represent hyperlinks between them.

SETUP. To the best of our knowledge, we are the first to provide a privacy-preserving framework for multiple simulation models. The previous work focused on exact subgraph pattern matching, i.e., subgraph isomorphism[17], and it is not adapted to other subgraph pattern matching models. As a result, we only compare our improved framework with a basic solution in our experiments. We compare four methods All_Ran,

①http://snap.stanford.edu/data/p2p-Gnutella08.html, Sept. 2019.
②http://snap.stanford.edu/data/loc-Brightkite.html, Sept. 2019.
③http://snap.stanford.edu/data/web-NotreDame.html, Sept. 2019.

All_Eff, Part_Ran, and Part_Eff, where All_Ran applies the random label generalization method and uploads G^k to cloud; All_Eff applies the cost-model based label generalization method introduced in Subsection 4.1.2 and uploads G^k to cloud; Part_Ran applies the same label generalization approach with All_Ran but only uploads G^0 to cloud; Part_Eff applies the same label generalization method with All_Eff but uploads G^0 to cloud.

All methods are implemented in C++. We use a Windows 10 PC with 2.30 GHz Intel Core i5 CPU and 8 GB of memory as the client side. The cloud server is on a virtualized Linux machine within Microsoft Azure Cloud with 4 CPU cores and 200 GB main memory.

5.2 Experiments Analysis

We evaluate the cost of our experiments from three aspects: time cost of generating G^k, time cost of pattern matching, and time cost of result processing in the client. The time costs of generating G^k and result processing have nothing to do with the pattern matching models. Therefore we only compare time cost of pattern matching via different subgraph pattern matching models in our experiments.

Time Cost of Generating G^k. We first evaluate the performance of the proposed framework while generating graph G^k. We conduct a set of experiments to observe the effect of the parameter θ and we finally find that our framework will have a better efficiency when $\theta = 2$. In these experiments, we set $k = 3$ and $|E(Q)| = 6$. Fig.8 shows the performance of Part_Eff considering strong simulation in p2p-Gnutella08. The overall running time increases with θ going from 2 to 4 since the time of label generalization phase will be larger when θ increases (we find similar results in strict simulation and tight simulation). Thus in these experiments, we define that each label group contains two labels, i.e., the default value of θ is 2.

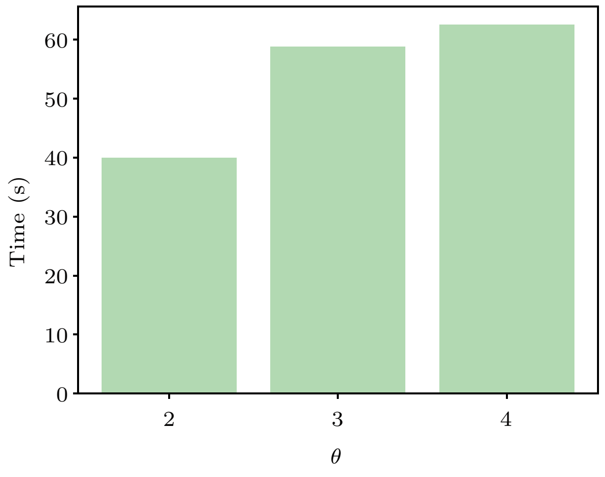


Fig.8. Overall running time of strong simulation vs θ.

According to Fig.9, the cost-model based label generalization method and the random label generalization method have similar performance while generating graph G^k, i.e., the four proposed methods have similar time cost. The reason is that all of them need to generate graph G^k firstly despite the ultimately uploaded graph is either G^k or G^0. We note that the time cost on the three datasets increases when k varies from 2 to 5. The reason is that when k increases, more and more noise edges are added to G^k, as shown in Table 4. Note that each "undirected" edge in G^k represents two directed edges. We can intuitively see that the number of noise edges has slight difference when using different label generalization methods and increases with k.

Time Cost of Pattern Matching. Then we pay attention to the time cost of subgraph pattern matching in cloud. We compare the running time of strong simulation, strict simulation and tight simulation. Firstly,

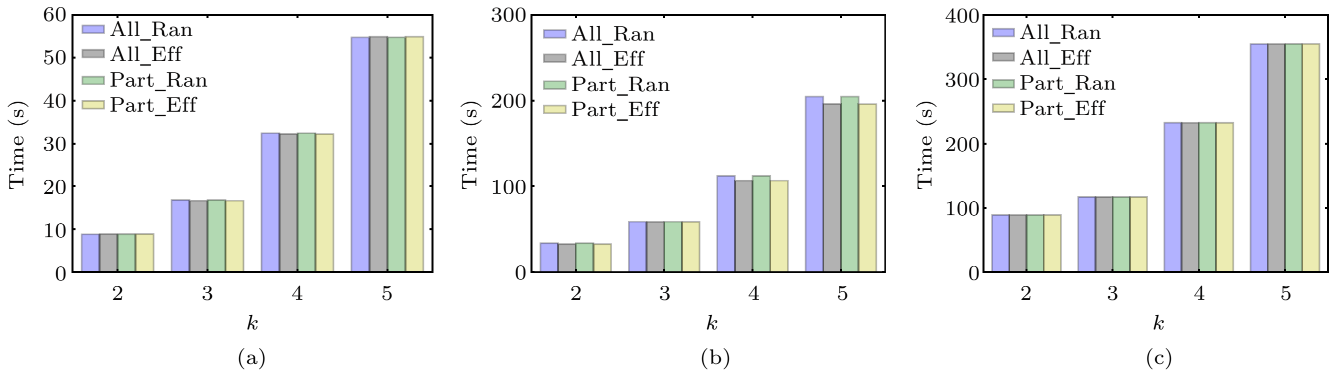


Fig.9. Time cost in generating G^k. (a) p2p-Gnutella08. (b) Brightkite_edges. (c) Web-NotreDame.

Table 4. Number of Noise Edges in Generating G^k

Dataset	Method	$k = 2$	$k = 3$	$k = 4$	$k = 5$
p2p-Gnutella08	All_Ran	16 417×2	34 267×2	53 195×2	71 388×2
	Part_Ran				
	All_Eff	16 309×2	34 309×2	53 195×2	71 443×2
	Part_Eff				
Brightkite_edges	All_Ran	178 278×2	367 859×2	553 810×2	753 265×2
	Part_Ran				
	All_Eff	178 674×2	368 399×2	554 794×2	752 677×2
	Part_Eff				
Web-NotreDame	All_Ran	923 266×2	1 829 324×2	2 749 760×2	3 747 812×2
	Part_Ran				
	All_Eff	923 382×2	1 846 433×2	2 745 792×2	3 767 437×2
	Part_Eff				

we evaluate the time cost of the proposed methods while varying the number of edges in pattern Q, i.e., $|E(Q)|$. Pattern graphs are generated by randomly extracting subgraphs from the original data graph G. We use $|E(Q)|$ to control the size of pattern graphs. In these experiments, the value of k is set to 3. According to Fig.10, we can clearly find out that Part_Eff performs better than the other three approaches on the three datasets. The one reason is that Part_Eff only uploads G^0 to cloud. Note that Part_Ran and Part_Eff are only different in label generalization. Thus, the results demonstrate the effectiveness of our cost-based label generalization method. We find similar results in strict simulation and tight simulation, as shown in Fig.11 and Fig.12 respectively. The matching time increases with $|E(Q)|$ varying from 4 to 10, since the searching space will become larger for subgraph pattern matching when $|E(Q)|$ increases.

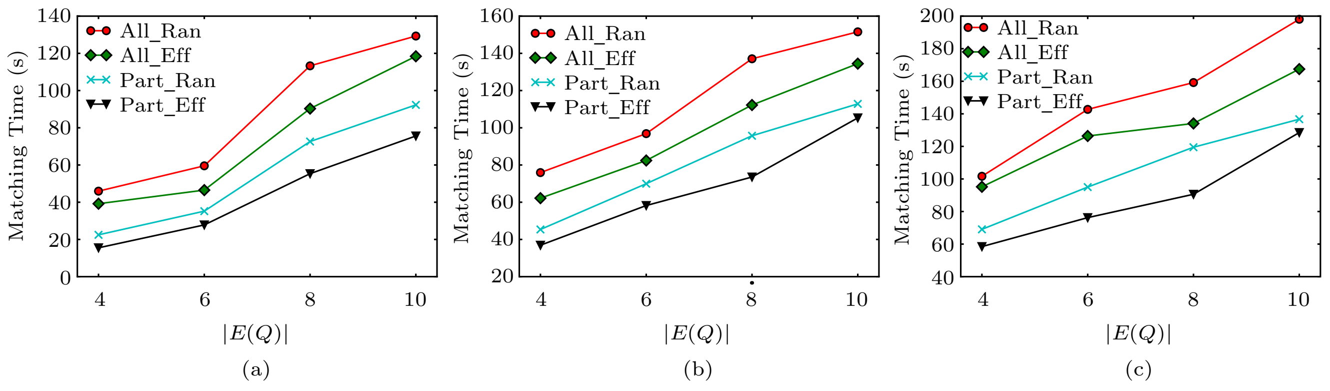


Fig.10. Matching time vs $|E(Q)|$ of strong simulation ($k = 3$). (a) p2p-Gnutella08. (b) Brightkite_edges. (c) Web-NotreDame.

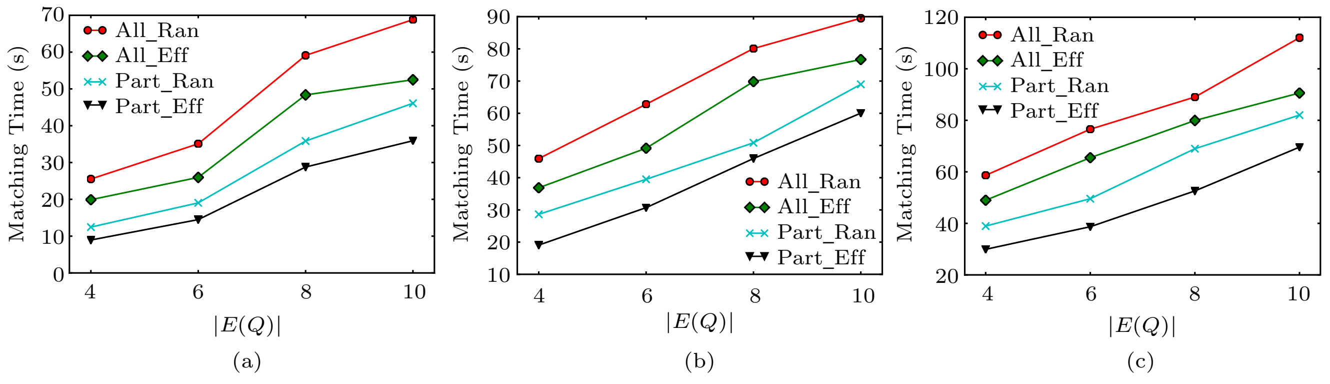


Fig.11. Matching time vs $|E(Q)|$ of strict simulation ($k = 3$). (a) p2p-Gnutella08. (b) Brightkite_edges. (c) Web-NotreDame.

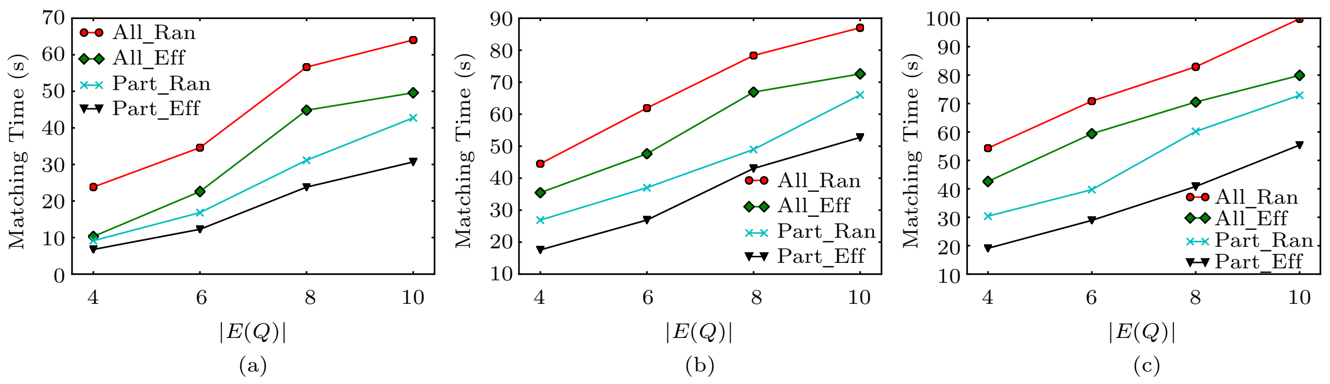


Fig.12. Matching time vs $|E(Q)|$ of tight simulation ($k = 3$). (a) p2p-Gnutella08. (b) Brightkite_edges. (c) Web-NotreDame.

Next, we evaluate the running time of subgraph pattern matching while varying the parameter k. The time cost increases with k varying from 2 to 5 in the three subgraph pattern matching models, as shown in Fig.13–Fig.15. The method Part_Eff, which uses the cost-model based label generalization method and uploads G^0 to cloud, has better performance than other methods in all three datasets. It demonstrates the superiority of our cost-model based generalization method as well.

Fig.16(a) and Fig.16(b) show the comparison of running time of different subgraph pattern matching models while varying the parameter $|E(G^0)|$ and k separately. We perform the experiments on Brightkite_edges. The time increases as $|E(Q)|$ varies from 4 to 10 in Fig.16(a) and Fig.16(b), and the time increases with k varying from 2 to 5. Because of the high cost of ball creation in strong simulation, strong simulation is slower than strict simulation and tight simulation in both Fig.16(a) and Fig.16(b). And we

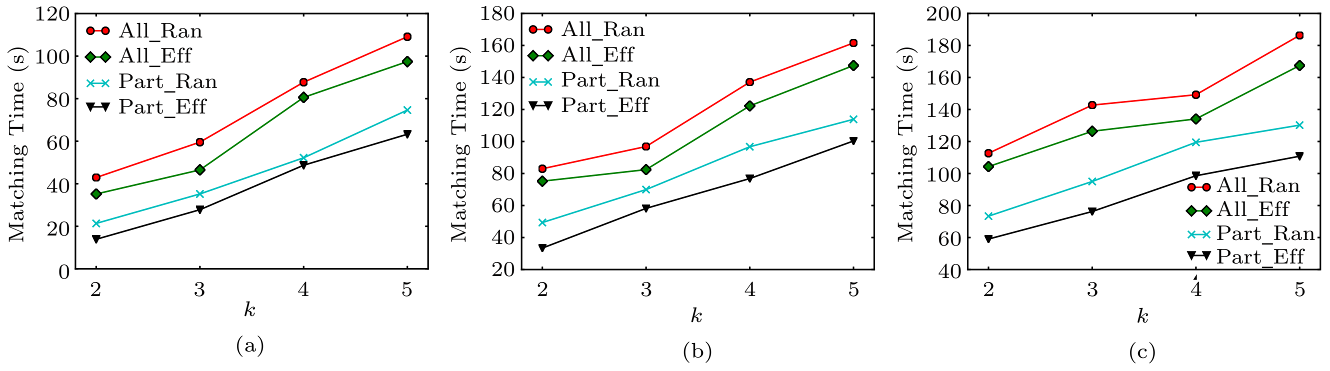


Fig.13. Matching time vs k of strong simulation ($|E(Q)| = 6$). (a) p2p-Gnutella08. (b) Brightkite_edges. (c) Web-NotreDame.

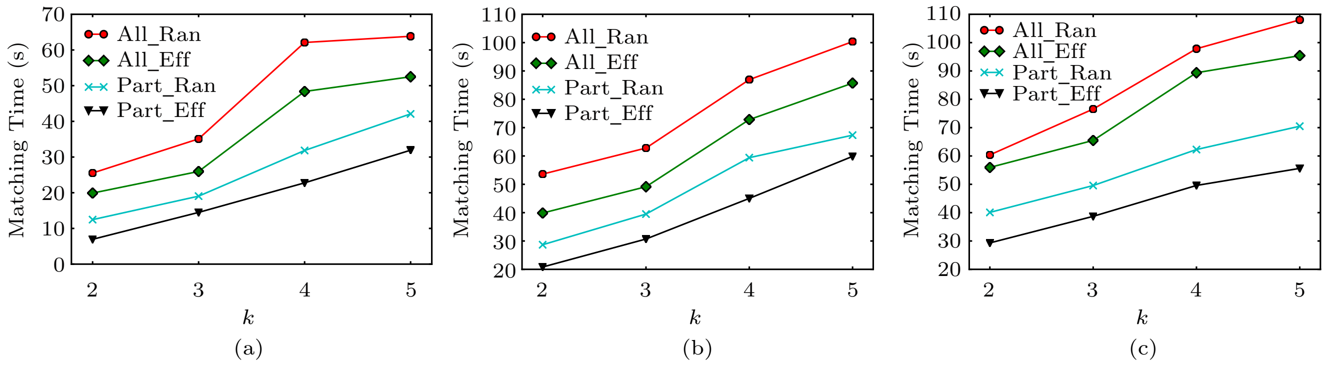


Fig.14. Matching time vs k of strict simulation ($|E(Q)| = 6$). (a) p2p-Gnutella08. (b) Brightkite_edges. (c) Web-NotreDame.

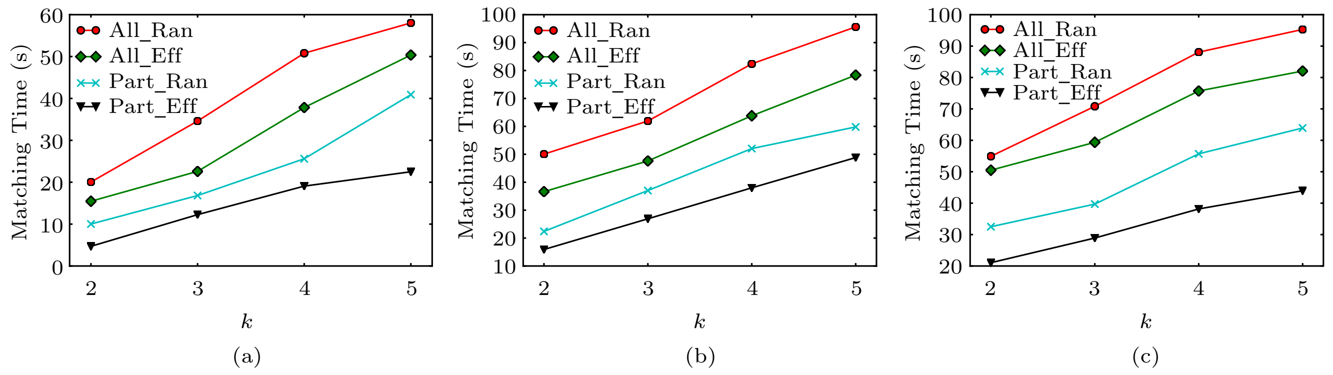


Fig.15. Matching time vs k of tight simulation ($|E(Q)| = 6$). (a) p2p-Gnutella08. (b) Brightkite_edges. (c) Web-NotreDame.

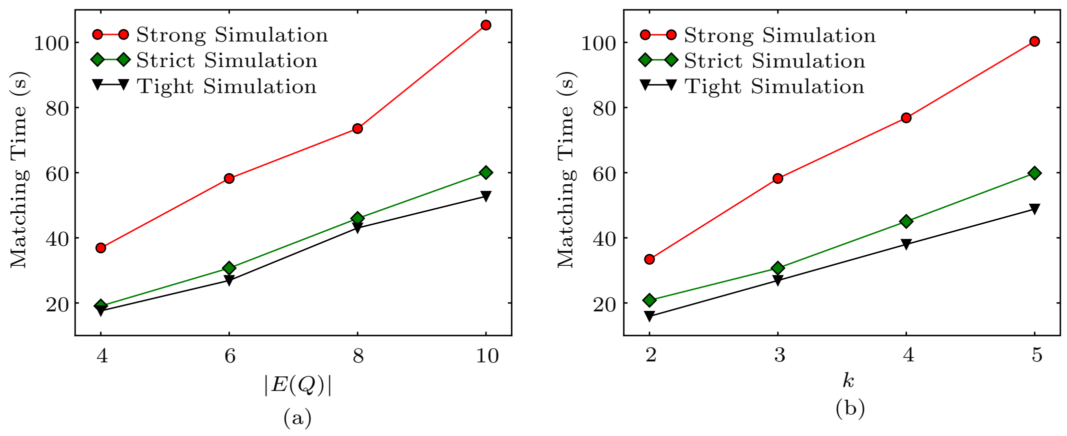


Fig.16. Comparison of matching time with different models.

can clearly find out that tight simulation spends less running time compared with other pattern matching models.

Time Cost of Result Processing in the Client. At last, we evaluate the performance of four methods involving result processing in the client side while varying parameters k and $|E(Q)|$ respectively. The difference between different models is small since the result processing time with different models is small compared with matching time. Thus we only present the result processing time via strong simulation.

According to Fig.17, the result processing time increases with k varying from 2 to 5 for all four methods, since the client needs to filter out more noise edges when k becomes larger. Note that both All_Ran and All_Eff upload G^k to cloud, the step to obtain

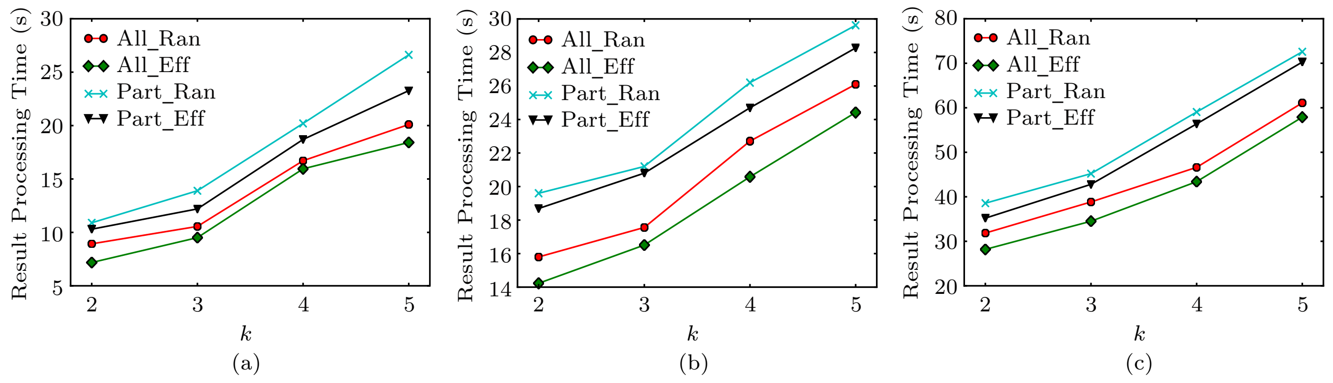


Fig.17. Result processing time vs k ($|E(Q)| = 6$). (a) p2p-Gnutella08. (b) Brightkite_edges. (c) Web-NotreDame.

$r(Q^0, G^k)$ on the basis of $r(Q^0, G^0)$ together with F_{k_i} $(i = 1, 2, ..., k-1)$ can be omitted. Thus, the time costs of result processing with All_Ran and All_Eff are smaller than those of the other two methods. However, the method Part_Eff that uses our cost-model based label generalization method still performs better than Part_Ran.

The result processing time increases with $|E(Q)|$ varying from 4 to 10, as shown in Fig.18. The reason lies in that the search space will become larger for the filtering process when $|E(Q)|$ increases. Similarly, the time costs of All_Ran and All_Eff are smaller than those of other two methods. However, the gap is smaller compared with the time cost of pattern matching. As shown in Table 5, Part_Eff runs faster than the other three methods in terms of the overall running time either in all subgraph pattern matching models. The overall running time of tight simulation is always slightly better than that of strict simulation. Note that the overall running time consists of the subgraph pattern matching time in cloud and the result processing time in the client side.

6 Conclusions

In this paper, we provided a privacy-preserving framework for multiple subgraph pattern matching models including strong simulation, strict simulation, and tight simulation in public cloud. Without losing utility, the framework protects structural and label privacy of both data graphs and pattern graphs. In this framework, a directed graph is transformed to an "undirected" graph by adding noise edges, so that the k-automorphism model based method can be applied to protect structural privacy in data graphs. We also applied a cost-model based label generalization method to protect label privacy in both data graphs and pattern graphs. Due to the k-symmetry in a k-automorphic graph, we only uploaded the outsourced graph to cloud to decrease the time cost of subgraph pattern matching in cloud. We conducted our experiments on three real-world datasets and the results showed that the proposed framework can well preserve the privacy of data graph and has greater performance in time cost.

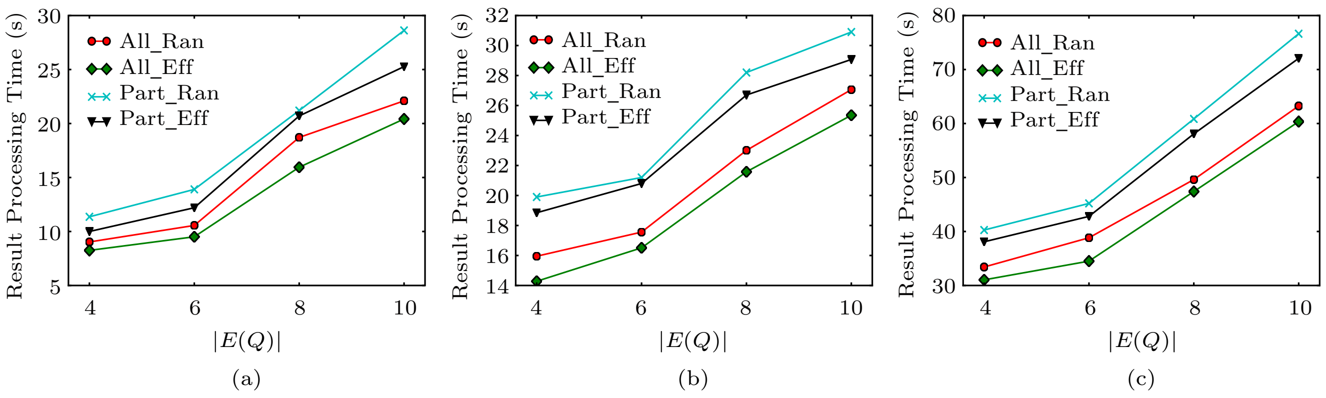


Fig.18. Result processing time vs $|E(Q)|$ ($k = 3$). (a) p2p-Gnutella08. (b) Brightkite_edges. (c) Web-NotreDame.

Table 5. Overall Running Time (s) ($|E(Q)| = 6$ and $k = 3$)

Method	Model	p2p-Gnutella08	Brightkite_edges	Web-NotreDame
Part_Eff	Strong	40.00	79.02	119.05
	Strict	26.69	51.50	81.50
	Tight	24.46	47.70	71.70
Part_Ran	Strong	49.10	91.15	140.22
	Strict	32.94	60.70	94.77
	Tight	30.72	58.22	84.90
All_Eff	Strong	56.02	98.93	160.89
	Strict	35.47	65.66	100.00
	Tight	32.10	64.13	93.88
All_Ran	Strong	70.12	114.42	181.56
	Strict	45.65	80.36	115.39
	Tight	45.16	79.48	109.67

References

[1] Lu H, Chang Y. Mining disease transmission networks from health insurance claims. In *Proc. the 2017 International Conference on Smart Health*, June 2017, pp.268-273.

[2] Ray B, Ghedin E, Chunara R. Network inference from multimodal data: A review of approaches from infectious disease transmission. *Journal of Biomedical Informatics*, 2016, 64: 44-54.

[3] Balsa E, Pérez-Solà C, Díaz C. Towards inferring communication patterns in online social networks. *ACM Trans. Internet Techn.*, 2017, 17(3): Article No. 32.

[4] Yin H, Zhou X, Cui B, Wang H, Zheng K, Hung N Q V. Adapting to user interest drift for POI recommendation. *IEEE Trans. Knowl. Data Eng.*, 2016, 28(10): 2566-2581.

[5] Yin H, Hu Z, Zhou X, Wang H, Zheng K, Hung N Q V, Sadiq S W. Discovering interpretable geo-social communities for user behavior prediction. In *Proc. the 32nd IEEE International Conference on Data Engineering*, May 2016, pp.942-953.

[6] Xie M, Yin H, Wang H, Xu F, Chen W, Wang S. Learning graph-based POI embedding for location-based recommendation. In *Proc. the 25th ACM International Conference on Information and Knowledge Management*, October 2016, pp.15-24.

[7] Yin H, Wang W, Wang H, Chen L, Zhou X. Spatial-aware hierarchical collaborative deep learning for POI recommendation. *IEEE Trans. Knowl. Data Eng.*, 2017, 29(11): 2537-2551.

[8] Aggarwal C C, Wang H. Managing and Mining Graph Data. Spring, 2010.

[9] Gallagher B. Matching structure and semantics: A survey on graph-based pattern matching. In *Proc. the 2006 AAAI Fall Symposium on Capturing and Using Patterns for Evidence Detection*, October 2006, pp.45-53.

[10] Henzinger M R, Henzinger T A, Kopke P W. Computing simulations on finite and infinite graphs. In *Proc. the 36th Annual Symposium on Foundations of Computer Science*, October 1995, pp.453-462.

[11] Fan W, Li J, Ma S, Tang N, Wu Y, Wu Y. Graph pattern matching: From intractable to polynomial time. *Proceedings of the VLDB Endowment*, 2010, 3(1): 264-275.

[12] Brynielsson J, Högberg J, Kaati L, Mårtenson C, Svenson P. Detecting social positions using simulation. In *Proc. the 2010 International Conference on Advances in Social Networks Analysis and Mining*, August 2010, pp.48-55.

[13] Ma S, Cao Y, Fan W, Huai J, Wo T. Strong simulation: Capturing topology in graph pattern matching. *ACM Trans. Database Syst.*, 2014, 39(1): Article No. 4.

[14] Fard A, Nisar M U, Ramaswamy L, Miller J A, Saltz M. A distributed vertex-centric approach for pattern matching in massive graphs. In *Proc. the 2013 IEEE International Conference on Big Data*, October 2013, pp.403-411.

[15] Fard A, Nisar M U, Miller J A, Ramaswamy L. Distributed and scalable graph pattern matching: Models and algorithms. *International Journal of Big Data*, 2014, 1(1): 1-14.

[16] Fan W, Li J, Ma S, Tang N, Wu Y. Adding regular expressions to graph reachability and pattern queries. In *Proc. the 27th International Conference on Data Engineering*, April 2011, pp.39-50.

[17] Chang Z, Zou L, Li F. Privacy preserving subgraph matching on large graphs in cloud. In *Proc. the 2016 International Conference on Management of Data*, June 2016, pp.199-213.

[18] Zou L, Chen L, Özsu M T. K-automorphism: A general framework for privacy preserving network publication. *Proceedings of the VLDB Endowment*, 2009, 2(1): 946-957.

[19] Tai C, Tseng P, Yu P S, Chen M. Identity protection in sequential releases of dynamic networks. *IEEE Trans. Knowl. Data Eng.*, 2014, 26(3): 635-651.

[20] Liu K, Terzi E. Towards identity anonymization on graphs. In *Proc. the ACM SIGMOD International Conference on Management of Data*, June 2008, pp.93-106.

[21] Zhou B, Pei J. Preserving privacy in social networks against neighborhood attacks. In *Proc. the 24th International Conference on Data Engineering*, April 2008, pp.506-515.

[22] Li J, Xiong J, Wang X. The structure and evolution of large cascades in online social networks. In *Proc. the 4th International Conference on Computational Social Networks*, August 2015, pp. 273-284.

[23] Hay M, Miklau G, Jensen D *et al.* Resisting structural re-identification in anonymized social networks. *VLDB*, 2010, 19(6): 797-823.

[24] Cheng J, Fu A W, Liu J. K-isomorphism: Privacy preserving network publication against structural attacks. In *Proc. the ACM SIGMOD International Conference on Management of Data*, June 2010, pp. 459-470.

[25] Wu W, Xiao Y, Wang W *et al.* K-symmetry model for identity anonymization in social networks. In *Proc. the 13th International Conference on Extending Database Technology*, March 2010, pp.111-122.

[26] Liu K, Terzi E. Towards identity anonymization on graphs. In *Proc. the ACM SIGMOD International Conference on Management of Data*, June 2008, pp.93-106.

[27] Tai C H, Yu P S, Yang D H, Chen M S. Privacy preserving social network publication against friendship attacks. In *Proc. the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, August 2011, pp.1262-1270.

[28] Chen S, Zhou S. Recursive mechanism: Towards node differential privacy and unrestricted joins. In *Proc. the ACM SIGMOD International Conference on Management of Data*, June 2013, pp.653-664.

[29] Zhu T, Li G, Zhou W, Yu P S. Differentially private data publishing and analysis: A survey. *IEEE Trans. Knowl. Data Eng.*, 2017, 29(8): 1619-1638.

[30] Qian Q, Li Z, Zhao P *et al.* Publishing graph node strength histogram with edge differential privacy. In *Proc. the 23rd International Conference Database Systems for Advanced Applications*, May 2018, pp.75-91.

[31] Sala A, Zhao X, Wilson C *et al.* Sharing graphs using differentially private graph models. In *Proc. the 11th ACM SIGCOMM Internet Measurement Conference*, November 2011, pp.81-98.

[32] Chen R, Fung B C, Yu P S *et al.* Correlated network data publication via differential privacy. *VLDB*, 2014, 23(4): 653-676.

[33] Zhang J, Cormode G, Procopiuc C M *et al.* Private release of graph statistics using ladder functions. In *Proc. the 2015 ACM SIGMOD International Conference on Management of Data*, May 2015, pp.731-745.

[34] Ullmann J R. An algorithm for subgraph isomorphism. *J. ACM*, 1976, 23(1): 31-42.

[35] Yuan Y, Wang G, Chen L *et al.* Efficient subgraph similarity search on large probabilistic graph databases. *VLDB*, 2012, 5(9): 800-811.

[36] Yuan Y, Wang G, Xu J Y *et al.* Efficient distributed subgraph similarity matching. *VLDB*, 2015, 24(3): 369-394.

[37] Gao J, Xu J, Liu G *et al.* A privacy-preserving framework for subgraph pattern matching in cloud. In *Proc. the 23rd International Conference on Database Systems for Advanced Applications*, May 2018, pp.307-322.

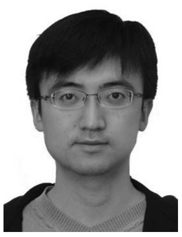
[38] Karypis G, Kumar V. Analysis of multilevel graph partitioning. In *Proc. the 1995 ACM/IEEE Conference on Supercomputing*, December 1995.

[39] Karypis G, Kumar V. Multilevel k-way partitioning scheme for irregular graphs. *Journal of Parallel and Distributed computing*, 1998, 48(1): 96-129.

Jiu-Ru Gao received her B.S. degree in computer science and technology from Changshu Institute of Technology, Changshu, in 2016. She is currently a M.S. candidate in Soochow University, Suzhou. Her research interests include data mining, data analysis and graph computing.

Wei Chen is a lecturer at the School of Computer Science and Technology, Soochow University, Suzhou. He got his Ph.D. and M.S. degrees in computer science and technology from the Soochow University, Suzhou. His research interests include data mining and spatial-temporal database.

Jia-Jie Xu is an associate professor at the School of Computer Science and Technology, Soochow University, Suzhou. He got his Ph.D. and M.S. degrees in computer science from the Swinburne University of Technology, Victoria, and the University of Queensland, Brisbane, in 2011 and 2006 respectively. Before joining Soochow University in 2013, he worked as an assistant professor in the Institute of Software, Chinese Academy of Sciences, Beijing. His research interests mainly include spatio-temporal database systems, big data analytics and workflow systems.



An Liu is a professor in the Department of Computer Science and Technology at Soochow University, Suzhou. He received his Ph.D. degree in computer science from both City University of Hong Kong (CityU), Hong Kong, and University of Science and Technology of China (USTC), Hefei, in 2009. His research interests include spatial databases, crowdsourcing, recommender systems, data security and privacy, and cloud/service computing.

Zhi-Xu Li is an associate professor in the School of Computer Science and Technology at Soochow University, Suzhou. He worked as a research fellow at King Abdullah University of Science and Technology, Thuwal. He received his Ph.D. degree in computer science from the University of Queensland, Brisbane, in 2013, and his B.S. and M.S. degrees in computer science from Renmin University of China, Beijing, in 2006 and 2009 respectively. His research interests include data cleaning, big data applications, information extraction and retrieval, machine learning, deep learning, knowledge graph and crowdsourcing.

Hongzhi Yin received his Ph.D. degree in computer science from Peking University, Beijing, in 2014. His research interests include recommender system, user profiling, topic models, deep learning, social media mining, and location-based services. He has published more than 30 papers in the most prestigious journals and conferences such as SIGMOD, KDD, VLDB, ICDE, the IEEE Transactions on Knowledge and Data Engineering, the ACM Transactions on Information Systems, the ACM Transactions on Intelligent Systems and Technology, and the ACM Transactions on Knowledge Discovery from Data. In addition, he has one monograph published by Springer. He is an ARC DECRA Fellow with the University of Queensland, Brisbane.

Lei Zhao is a professor in the School of Computer Science and Technology at Soochow University, Suzhou. He received his Ph.D. degree in computer science from Soochow University, Suzhou, in 2006. His research focuses on graph databases, social media analysis, query outsourcing, parallel and distributed computing. His recent research is to analyze large graph database in an effective, efficient, and secure way.