Privacy-Preserving Subgraph Discovery

Danish Mehmood, Basit Shafiq, Jaideep Vaidya, Yuan Hong, Nabil Adam, Vijayalakshmi Atluri

Privacy-Preserving Subgraph Discovery

Danish Mehmood[1], Basit Shafiq[1,2], Jaideep Vaidya[2], Yuan Hong[2], Nabil Adam[2], and Vijayalakshmi Atluri[2]

[1]Lahore University of Management Sciences, Pakistan
[2]CIMIC, Rutgers University, USA
{danish.mehmood,basit}@lums.edu.pk,
{jsvaidya,yhong,adam,atluri}@cimic.rutgers.edu

Abstract. Graph structured data can be found in many domains and applications. Analysis of such data can give valuable insights. Frequent subgraph discovery, the problem of finding the set of subgraphs that is frequent among the underlying database of graphs, has attracted a lot of recent attention. Many algorithms have been proposed to solve this problem. However, all assume that the entire set of graphs is centralized at a single site, which is not true in a lot of cases. Furthermore, in a lot of interesting applications, the data is sensitive (for example, drug discovery, clique detection, etc). In this paper, we address the problem of privacy-preserving subgraph discovery. We propose a flexible approach that can utilize any underlying frequent subgraph discovery algorithm and uses cryptographic primitives to preserve privacy. The comprehensive experimental evaluation validates the feasibility of our approach.

1 Introduction

Today, graph structured data is ubiquitous. All types of relationships, including, spatial, topological, and other characteristics, can be modeled through the graph abstraction, thus capturing different data semantics. Graph-based analysis gives valuable insights into the data and has been successfully applied to various application domains including protein analysis [17],chemical compound analysis [3], link analysis[12] and web searching[15]. One of the most useful forms of analysis is to find frequent subgraphs from a large database of graphs. This has application in many different domains including cheminformatics (drug disovery), transportation networks, etc. Unlike standard frequent itemset discovery, used for association rule mining in transactional databases, frequent subgraph discovery is a much tougher problem due to underlying fundamental problems, such as canonical labeling of graphs and subgraph isomorphism.

In recent years, this has attracted a lot of attention with many efficient algorithms being developed to solve this problem. However, all of these algorithms assume that the set of graphs is either public or available at a single site. In reality, in many valuable cases, the set of graphs may be distributed between multiple parties, with each party owning a subset of the graphs. Chemical compound databases are one such example. Many pharmaceutical companies have local databases of pharmaceutical drugs which can be represented as graphs. Furthermore, in a lot of interesting applications, the data is sensitive (for example, drug discovery, clique detection, etc). Therefore, due to privacy

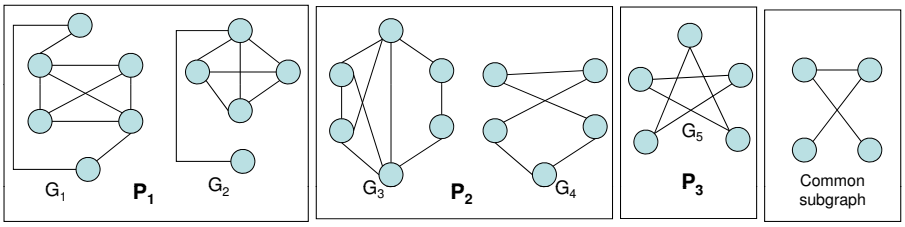

Fig. 1. Graphs owned by three parties (P_1, P_2, and P_3) and the subgraph present in all the graphs of each party.

and security concerns, the parties may not wish to reveal their individual graphs to each other or to a central site. In this case, a distributed privacy-preserving algorithm must be developed to enable mining in such cases. In this paper, we develop such an algorithm using cryptographic primitives to preserve privacy. Our algorithm uses any known subgraph discovery method as a subroutine, and therefore can be enhanced in the future as well. We have implemented our approach and the comprehensive experimental evaluation validates the feasibility of our approach.

Illustrative Example: Consider the case depicted in Figure 1, where 3 parties together own 5 graphs (Party 1 owns 2, Party 2 owns 2, and Party 3 owns 1). Figure 1 also depicts the common subgraph that is common to all of the five graphs. Therefore, even with a support threshold of 5, this graph will be detected when subgraph mining is done on the global set of graphs. Note that in this case, we assume that the graph is unlabeled (i.e., neither nodes nor edges are labeled). However, our approach is agnostic to the labeling - either the nodes, the edges, both, or neither could be labeled, based on the data semantics. As long as the underlying subgraph discovery algorithm can handle these cases, our approach will be able to take all of these requirements correctly into account.

1.1 Problem Statement

The basic problem is to discover frequent subgraphs in a privacy-preserving way from a set of graphs owned by different parties. This can be formalized as follows:

Definition 1. *Given k parties $P_1, \ldots, P_k$, each of which own a set of graphs S_i (let $S = \bigcup S_i$), and a global threshold δ ($0 \leq \delta \leq 1$), find the set of frequent subgraphs FS in a privacy-preserving fashion, wherein the global support of each subgraph in FS is over δ. Thus, for each subgraph $fs_j \in FS$, $\sum_i support(S_i, fs_j) \geq \delta|S|$, where $support(S_i, fs_j) =$ # graphs in S_i that include fs_j as a subgraph.*

Note that in the definition above, we simply require that the set of frequent subgraphs is found in a privacy-preserving fashion. Under the framework of secure multi-party computation[24, 5], this equates to not leaking any additional information to any

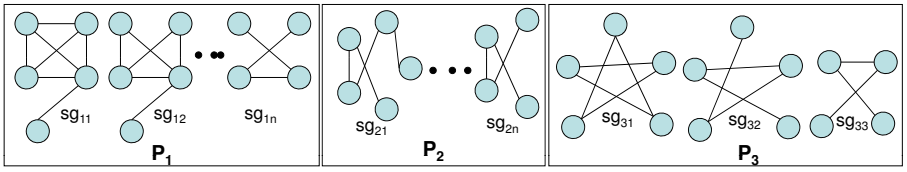

Fig. 2. Local candidate sets with respect to the example in Fig. 1

party beyond what can be inferred (in polynomial time) through the local input and output.

Instead of strictly following Definition 1, our protocol satisfies a relaxed form of this definition that allows efficient computation at the expense of leaking additional information. Below is the revised formulation with relaxed privacy requirement that the protocol satisfies.

Definition 2. *Given k parties $P_1, \ldots, P_k$, each of which own a set of graphs S_i (let $S = \bigcup S_i$), and a global threshold δ ($0 \leq \delta \leq 1$), find the set of frequent subgraphs FS (wherein the global support of each subgraph in FS is over δ) without leaking any additional information beyond the set of all local candidates, their support counts, and the number of parties owning them.*

2 Proposed Approach

In this section, we present our proposed approach for privacy-preserving discovery of frequent subgraphs in the set of graphs distributed among multiple parties. The proposed approach essentially involves three key steps:

1. Generation of local candidates – each party computes a candidate set of frequent subgraphs from it local graph set.
2. Generation of a global candidate set of subgraphs – secure union of local candidates to form a global candidate set – the global candidate set is generated by performing secure union of the local candidate sets.
3. Removal of non-frequent subgraphs from the global candidate set – the frequency count of each subgraph in the global candidate set is compared against a global threshold to check if this candidate is a real result.

The overall algorithm encompassing the above steps for subgraph discovery is given in Algorithm 1.This algorithm implements the distributed protocol involving k parties and a coordinator site. Each party owns a local set of graphs, and N is the total number of graphs, which is the sum of the number of graphs in the local set of all parties. The algorithms also uses a commutative cryptography system [14] for computing secure union and a homomorphic cryptography system [13] for computing secure sum. Correspondingly, each party and coordinator site has a pair of commutative encryption and decryption keys; a public homomorphic encryption key which is shared among all the

Algorithm 1 SubgraphDiscovery

Require: k parties $(P_1, \ldots, P_k)$ each owning a set of graphs and a coordinator site $Coord$
Require: P_i owns the graph set, $GS^{(i)} = G_1^{(i)}, \ldots, G_m^{(i)}$
Require: $N = |GS^{(1)}| + \ldots + |GS^{(k)}|$
Require: $E_C^{(i)}, D_C^{(i)}$ Commutative encryption and decryption keys of party P_i
Require: E_H A public Homomorphic encryption key
Require: $D_H^{(i)}$ Homomorphic decryption key of party P_i
Require: s_T, support threshold, percentage of total graphs in which the resulting subgraph(s) is/are present
Require: $Node_{min}$, minimum number of nodes in each of the subgraphs
Require: $Edge_{min}$, minimum number of edges in each of the subgraphs
Ensure: FSG, frequent subgraph set

{**STEP 1:** Generation of Local Candidates}
At each party P_i:
{select an appropriate support threshold, $s_T^{(i)}$. This threshold value is used to find a set of candidate frequent subgraphs from the set of local graphs $GS^{(i)}$}
{select an appropriate support threshold, $s_T^{(i)}$. This threshold value is used to find a set of candidate frequent subgraphs from the set of local graphs $GS^{(i)}$}
$LocalCand^{(i)} \leftarrow filter(gSpan(GS^{(i)}, s_T^{(i)}, Node_{min}, Edge_{min}))$ {run the gSpan algorithm locally to find the candidate frequent subgraphs from the local graph set. The $filter$ function ensures that only those subgraphs that a party is comfortable with are included in the local candidate set.}
{**STEP 2:** Generation of Global Candidate Set}
$m^{(i)} \leftarrow Encrypt(LocalCand^{(i)}, E_C^{(i)})$. {The encryption method treats $LocalCand^{(i)}$ as a string so that the resulting cipher text $m^{(i)}$ does not reveal the structure of any of the local graphs.}
send $m^{(i)}$ to $Coord$ for secure union
At $Coord$:
$FSG \leftarrow \{\}$
receive $m^{(i)}$ from each party P_i
$M \leftarrow \bigcup_{i=1}^{k} m^{(i)}$
$GlobalCand \leftarrow SecureUnion(M)$
{**STEP 3:** Removal of Non-frequent Subgraphs}
send $GlobalCand$ to each party P_i
At each party P_i:
create an array $Ecount^{(i)}$ with length equal to the length of $GlobalCand$ and initialize all the elements of $Ecount^{(i)}$ to random encryption of 0 using the homomorphic encryption key E_H
for each subgraph $sg_j \in GlobalCand$ **do**
  for each local graph $G_x \in GS^{(i)}$ **do**
    if sg_j is present in G_x **then**
      $h_{enc}^1 \leftarrow HomomorphicEncrypt(1, r, E_H)$ {Random encryption of 1 using homomorphic encryption key E_H)}
      $Ecount^{(i)}[j] \leftarrow Ecount^{(i)}[j] * h_{enc}^1$
    end if
  end for
end for
send $Ecount^{(i)}$ to $Coord$ for secure sum
At $Coord$:
receive $Ecount^{(i)}$ from each party P_i
$Count \leftarrow SecureSum(Ecount^{(1)}, \ldots, Ecount^{(k)})$
for each subgraph $sg_j \in GlobalCand$ **do**
  if $Count[j] \geq s_T * N$ **then**
    $FSG \leftarrow FSG \bigcup sg_j$
  end if
end for
return FSG

Algorithm 2 SecureUnion(M)

Require: k parties ($P_1, \ldots, P_k$) and a coordinator site $Coord$
Require: $E_C^{(i)}, D_C^{(i)}$ Commutative encryption and decryption keys of party P_i
At $Coord$:
$EM \leftarrow \{\}$
$GlobalCand \leftarrow \{\}$
{Commutaive encryption of M by all parties}
for each $m^{(i)} \in M$ **do**
  $q \leftarrow m^{(i)}$ {$m^{(i)}$ is received from P_i}
  for $j = 1 \ldots k$ **do**
    if $j \neq i$ **then**
      shuffle and send q to party P_j for commutative encryption with its key $E_C^{(j)}$
      receive eq from P_j
      $q \leftarrow eq$
    end if
  end for
  $EM = EM \bigcup \{q\}$
end for
send commutative encryption complete signal
At each party P_i:
while commutative encryption complete signal is not received from $Coord$ **do**
  receive q from $Coord$
  send $eq \leftarrow Encrypt(q, E_C^{(i)})$ to $Coord$ {eq is encryption of q with the ommutative encryption key $E_C^{(i)}$}
end while
{Decryption of EM by all parties}
for each $em \in EM$ **do**
  $q \leftarrow em$
  for $j = 1 \ldots k$ **do**
    send dq to party P_j for decryption with its key $D_C^{(j)}$
    receive dq from P_j
    $q \leftarrow dq$
  end for
  $GlobalCand = GlobalCand \bigcup \{q\}$
end for
send decryption complete signal
remove duplicate elements (subgraphs) from $GlobalCand$
return $GlobalCand$
At each party P_i:
while decryption complete signal is not received from $Coord$ **do**
  receive q from $Coord$
  send $dq \leftarrow Decrypt(q, D_C^{(i)})$ to $Coord$ {dq is decryption of q with the Commutative decryption key $D_C^{(i)}$}
end while

Algorithm 3 SecureSUM($Ecount^{(1)}, \ldots, Ecount^{(k)}$)

Require: Threshold-based homomorphic crypto system
Require: E_H A public Homomorphic encryption key
Require: $D_H^{(i)}$ Homomorphic decryption key of party P_i and D_H^{Coord} Homomorphic decryption key of Coord
Require: T, Threshold for homomorphic decryption (No. of parties needed for decryption)
1: **At** $Coord$:
2: Create an array $Ecnt$ with length equal to the length of $GlobalCand$ and initialize all the elements of $Ecnt$ to random encryption of 0 using the homomorphic encryption key E_H
3: **for** $i = 1 \ldots length(Ecnt)$ **do**
4: **for** $j = 1 \ldots k$ **do**
5: $Ecnt[i] \leftarrow Ecnt[i] * Ecount^{(j)}[i]$
6: **end for**
7: **end for**
8: Collaboratively decrypt $Ecnt$ with T parties to get actual frequency count of each subgraph
9: **return** Decrypted $Ecnt$

parties and coordinator; a private homomorphic decryption key derived from the public key. In addition, the algorithm requires the user to specify the support threshold (s_T), which is the percentage of total graphs in which the computed subgraphs are present. The user also specifies the minimum size of these subgraphs that need to be computed. This minimum size is specified in terms of the number of nodes ($Node_{min}$) and number of edges ($Edge_{min}$).

Below we discuss each of the above three steps and how these steps are implemented in the Subgraph discovery algorithm.

2.1 Generation of Local Candidates

This step of generation of local candidates is implemented in lines 1 - 4 of Algorithm 1. For generation of local candidates, each party runs the frequent subgraph mining algorithm. For frequent subgraph mining, we use the gSpan algorithm [23]. Our approach is agnostic to the underlying frequent subgraph mining algorithm. We chose gSpan since it was easily available and reasonably efficient. gSpan takes a collection of graphs and a minimum support threshold as input and computes all the subgraphs whose frequency is greater than or equal to the given threshold. In addition, we constrain the minimum size of subgraphs to avoid retrieving trivial subgraphs. For this, we use the ($Node_{min}$) and ($edge_{min}$) parameters which are defined globally by the user.

Note that a $filter$ function is applied to the output of $gSpan$ to ensure that only those subgraphs that a party is comfortable with are included in its local candidate set. This improves the privacy protection. For computing the local candidate set, the support threshold needs to be closer to the global support threshold (s_T) or smaller to reduce the number of frequent sub-graphs that are missed from the final solution. Clearly, if a local support threshold corresponding to one graph only (i.e., a subgraph is present in only one of the local graph) is used, there will not be any miss. However, this significantly increases the computational overhead as there will be too many subgraphs

in the local candidate set. We analyze this trade-off between performance and accuracy in our experiments discussed in Section 4.2.

Fig. 2 depicts the set of local candidates computed locally by each of the three parties P_1, P_2, and P_3 using their local set of graphs discussed in the Introduction and illustrated in Figure 1. In this Figure, the minimum size of the subgraphs are restricted to 4 nodes and 3 edges.

2.2 Generation of a Global Candidate Set

This step of generation of a global candidate set of frequent subgraphs is implemented in lines 5 - 12 of Algorithm 1. Essentially, the global candidate set is the union of the local candidate sets computed by each party in Step 1. However, due to privacy requirements this union needs to be computed in a secure manner without revealing which candidate subgraph comes from which party. We employ a commutative encryption-based approach for computing the secure union of the local candidate sets.

An encryption algorithm is commutative if plain text data item enciphered with multiple encryption keys in any arbitrary order will have the same enciphered text.Formally, an encryption algorithm is commutative if the following two equations hold for any given encryption keys $K_1, \ldots, K_n \in K$, any data item to be encrypted $m \in M$, and any permutations of $i, j : \forall m_1, m_2 \in M$ such that $m_1 \neq m_2$:

$$E_{K_{i_1}}(...E_{K_{i_n}}(m)...) = E_{K_{j_1}}(...E_{K_{j_n}}(m)...) \tag{1}$$

and for any given $k, \epsilon < 1/2^k$

$$Pr(E_{K_{i_1}}(...E_{K_{i_n}}(m_1)...) = E_{K_{j_1}}(...E_{K_{j_n}}(m_2)...)) < \epsilon \tag{2}$$

Pohlig-Hellman [14] is one example of a commutative encryption scheme (based on the discrete logarithm problem). This or any other commutative encryption scheme would work well for our purposes.

The basic idea of computing the secure union using the commutative encryption protocol is that each subgraph in every local candidate set is encrypted by all the parties using their commutative encryption keys. Then all these encrypted subgraphs are put into a global candidate set with their order shuffled. However, the elements in the encrypted global candidate set would have duplicates that need to be removed for the global candidate set to the union of all the local candidate sets. The encryption method used in the proposed approach treats each element in the local candidate set as a string so that the resulting cipher text mask the structural information of each local subgraph. Without knowing the structural information, duplicate subgraphs in the encrypted global candidate set cannot be removed. The following substeps elaborate on the proposed commutative encryption-based strategy for computing the secure union of local candidate set to form the global candidate set.

Substep 1. Each party P_i represents its local candidate set into a string and applies its commutative encryption key $E_C^{(i)}$ on the resulting string. This encrypted local candidate set is then sent to $Coord$ for secure union. (Lines 5 and 6 of Algorithm 1)

Substep 2. The $Coord$ receives the encrypted local candidate set from each party and routes each candidate set to all other parties for commutative encryption. When all

the local candidate sets are encrypted by all parties, the $Coord$ combines them into one global encrypted set (Lines 7 - 11 of Algorithm 1 and lines 1 - 16 of Algorithm 2).

Substep 3. The encrypted global candidate set is sent by the coordinator to each party for decryption. Each party upon receiving the encrypted global candidate set shuffles its order and then applies its commutative decryption key. After all parties have applied their decryption keys, the structural information in the global candidate set is restored. After this the duplicate subgraphs in the global candidate set are removed (lines 17 - 39 of Algorithm 2)). For duplicate removal, we perform a pairwise comparison between the subgraphs in the global candidate set using gSpan.

This strategy computes the global candidate set without revealing private information about the local subgraphs of any party to other parties – specifically, which subgraphs belong to which party. This is because during the commutative encryption substeps (substeps 1 and 2), the local candidate set of each party is encrypted by the encryption key of the party owning the graph. During the decryption phase all the local subgraphs are merged into one set with their order shuffled. Therefore, inferring which subgraph belongs to which party based on its position in the global candidate set is also not possible. The drawback of removing duplicates after decryption is that the coordinator would know how many parties have a given subgraph present in their graphs. We discuss this issue further in Section 3.

2.3 Removal of Non-frequent Subgraphs

The global candidate set is the union of the local candidate set. As discussed in Section 2.1 the support threshold for generation of subgraphs in the local candidate set may be smaller than the global support threshold. Therefore, all those subgraphs that do not satisfy the global threshold need to be removed from the global candidate set. This requires computing the frequency count (number of graphs in which a given subgraph is present) for each subgraph. This frequency count also needs to be computed in a secure and distributed manner as there is no global set of graphs and all the graphs are with their owner parties.

The step of removal of non-frequent subgraphs from the global candidate set is implemented in lines 13 - 33 of Algorithm 1. intuitively in this step, each party P_i computes the frequency count of each subgraph in the global set with respect to its local graph set $GS_i^{(i)}$ (lines 17 - 24 of Algorithm 1). This frequency count is stored in a vector which is sent to the coordinator. The coordinator after receiving the frequency count vector from all parties computes the sum for each subgraph in the global candidate set. If this sum is less than the given global support threshold, the corresponding subgraph is removed from the global candidate set (lines 22 - 33 of Algorithm 1).

For secure computation and summation of the frequency counts, we employ an additive homomorphic cryptosystem such as the Paillier cryptosystem [13]. An additive homomorphic encryption is semantically-secure public-key encryption which has the additional property that given any two encryptions $E(A)$ and E(B), there exists an encryption $E(A+B)$ such that $E(A)*E(B) = E(A+B)$, where * denotes multiplication operator.

Following this homomorphic encryption property, each party initializes its frequency count vector by putting a random homomorphic encryption of '0' in each element of

its frequency count vector (line 16 of Algorithm 1). When computing the frequency count of each subgraph in the nested for loop of lines 17 - 24 of Algorithm 1, if a match is found the party increments the value of the corresponding element of the frequency count vector by '1'. This is done by multiplying the value of such element with a random homomorphic encryption of '1' (line 21 of Algorithm 1). Similarly, the coordinator employs a secure sum protocol to compute the sum of the frequency count vectors received from each party and employs threshold based decryption to decrypt the values of the global count for each subgraph. The reason for using threshold-based decryption strategy is to prevent a single party (coordinator) to decrypt the values in the frequency count vector received from each party. Threshold-based decryption with threshold of T requires T parties to collaboratively decrypt the encrypted values.

3 Complexity and Security Analysis

We now analyze the computation and communication complexity of our algorithms as well as the security provided through our approach.

3.1 Computation Cost

The computation cost of the distributed algorithm is actually comparable to the cost incurred in the centralized case. The main cost is incurred due to three steps, the local calls to the gSpan algorithm to find the local candidates, the commutative encryption based protocol to find the global candidate set, and the second round of local calls to the gSpan algorithm to find the frequency count of each candidate subgraph. Compared to these steps, the cost of the secure sum to find the global frequencies is negligible and can be ignored. Let us now consider the cost of each step.

Essentially, in the first step, even though each party invokes gSpan independently, it does so, only over the local set of graphs. Therefore the total computation cost is comparable to the cost of running gSpan over the entire set of graphs in a centralized case. In the second step, the secure union protocol is used to create a global candidate set. Essentially, each candidate subgraph is represented by a string, which is encrypted by all the parties, and then decrypted after merging into a combined set. Thus, assuming a total of l candidate graphs in the global set, the total cost is that of $O(kl)$ encryptions and decryptions. In the third step, gSpan is run over each pair of candidate subgraph, and local graph. Thus, gSpan is invoked $l|S|$ times, where $|S|$ is the total number of graphs. Note that each invocation of gSpan in step 3 takes much less time than in step 1 as only two graphs are being compared.

3.2 Communication Cost

Communication between the parties only occurs when the local candidates are merged into a global set and sent to all of the parties, and in the final frequency determination phase. For the secure union, there are a total of $O(kl)$ messages, for encryption and decryption. For the secure sum, it is the same with $O(kl)$ messages being exchanged.

3.3 Security Analysis

Consider Algorithm 1. Step 1 is completely local, so no private information is disclosed. In step 2, the global candidate set is generated from the set of local candidates. Encryption is used to obscure the link between the candidates and the party generating them. The Secure Union protocol is used to securely combine the candidate sets. Assuming that this protocol is secure, nothing is leaked through the combination process, though all parties will learn the global candidate set (and can therefore identify candidates that do not belong to them, though they cannot identify which party the candidates come from). Note that, in reality, the secure union does leak some additional information. In the secure union (Algorithm 2), after merging local sub-graphs, duplicate sub-graphs are removed. However, the duplicates are removed after decryption. Therefore, the coordinator would know the number of parties that support each candidate sub-graph. In the extreme case where all parties support a particular subgraph, the coordinator would now know that the particular subgraph is frequent for all parties (though it still would not know which local subgraphs does it belong to for each party, as long as there are at least 3 parties).

In step 3, the non-frequent subgraphs are removed. For this, the support count of each of the candidate subgraphs is computed using the secure sum algorithm. Assuming this is secure, nothing is leaked, except for the support count (though, again, it is not clear which graphs contributed to this support count). In total, the overall process simply leaks the set of global candidates to all parties (along with the number of parties supporting each candidate, though only to the coordinator), as well as the frequency count (again, only to the coordinator).

Assuming that this additional information is given to the simulator, we can prove that the algorithm does not leak anything further. The question is whether this constitutes too much information. Let us consider each independently. Our algorithm leaks the set of global candidates, from which the final set of frequent subgraphs is picked. However, since each party locally mines its frequent subgraphs in step 1, it can easily refrain from including any of the subgraphs that it is uncomfortable with. This makes it difficult for any party to learn the entire graph or any unique / identifying subgraph of other parties. In the case of the frequency counts, if these are considered sensitive, it can be easily handled through the use of a simple add and compare[18] protocol that can check whether the global count of the candidate is above the threshold or not. Such a protocol can easily be implemented through the VIFF system[1].

Given a large set of graphs, this extra information can be considered to be acceptable and worth allowing, given the gain in efficiency that is obtained, as compared to generic secure multiparty computation techniques which would leak no information but be extremely slow.

[1] http://viff.dk/

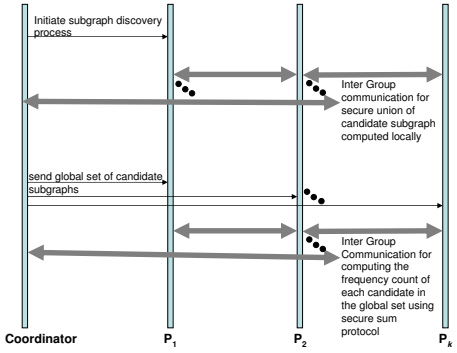

Fig. 3. Basic Interaction Diagram

4 Experimental Evaluation

4.1 Implementation Details

In this section we cover the details of our implementation of the privacy-preserving subgraph discovery algorithm on modern hardware and present experimental results demonstrating the usability of this algorithm.

The general model of privacy preserving subgraph is as follows. The coordinator initiates a request for subgraph discovery. The coordinator could be a separate entity or one of the graph owner parties. There is one global coordinating class/interface that provides access to the subgraph discovery functionality. We call this the `RmtMain` interface. However, since this interface is present at a user site, the class implementing it should have access to no private information about the graphs of other parties. The class implementing this is the `SiteCoordinator` class which is initialized with the appropriate site information. There is another interface called, `RmtSlave` interfaces. All other parties that are involved in the sub-graph discovery process are treated as slave sites. The coordinator site coordinates with the slave sites to perform the required action.

Figure 3 demonstrates the basic interaction diagram. Java RMI is used to implement the distributed protocol between all of the sites.

We used Pohlig-Hellman encryption scheme [14] for implementing commutative encryption and Paillier Crypto system [13] for implementing Homomorphic encryption.

4.2 Experimental Evaluation

We now present experimental results demonstrating the usability of the proposed algorithms. We ran our experiments on two real graph datasets [23]: i) Chemical 340; ii) Compound 422. The Chemical 340 dataset 340 chemical compounds, 24 different atoms, 66 atom types, and 4 types of bonds. The average graph size in this dataset is 27 nodes and 28 edges. The largest graph in this dataset has 214 nodes and 214 edges.

Chemical 340							
Dataset		Total graphs = 340; Average graph size = (27 nodes, 28 edges); Largest graph size = (214 nodes, 214 edges)					
Frequent Subgraphs		Size($Node_{min}$=5, $Edge_{min}$=5); Global Threshold = 12%; Number of frequent subgraphs = 550					
Sites		Randomly distributed among 3 sites with each site having equal number of graphs					
No.	**Local Threshold (%age)**	**Average subgraphs per site**	**Step 1 Time (sec)**	**Step 2 Time (sec)**	**Step 3 Time (sec)**	**Total Time (sec)**	**Accuracy (%age)**
1	9	1308	23	8,871	1,537	10,423	100
2	13	513	14	2,761	458	3,233	99.82
3	15	393	12	2,087	359	2,458	95.82
4	18	264	11	1336	249	1,596	71.09
5	19	203	10	1008	205	1,223	58.55
6	22	120	11	578	151	740	33.82

Compound 422							
Dataset		Total graphs = 422; Average graph size = (40 nodes, 42 edges); Largest graph size = (189 nodes, 196 edges)					
Frequent Subgraphs		Size ($Node_{min}$=5, $Edge_{min}$ edges=5; $Node_{max}$=7, $Edge_{max}$ edges=7); Global Threshold = 12%; Number of frequent subgraphs = 562					
Sites		Randomly distributed among 3 sites with each site having equal number of graphs					
No.	**Local Threshold (%age)**	**Average subgraphs per site**	**Step 1 Time (sec)**	**Step 2 Time (sec)**	**Step 3 Time (sec)**	**Total Time (sec)**	**Accuracy (%age)**
1	9	728	14	3768	661	4433	100
2	12	571	13	2739	513	3265	100
3	15	442	12	2094	437	2543	95.20
4	18	351	14	1631	341	1986	78.29

Fig. 4. Computation time and accuracy vs. local threshold for Chemical 340 and Compound 422 datasets

The Compound 422 dataset has 422 graphs with average graph size of 40 nodes and 42 edges. The largest graph in this dataset has 189 nodes and 196 edges.

Figure 4 shows the computation time and accuracy results of the $SubgraphDiscovery$ algorithm for the Chemical 340 and Compound 422 datasets. The global threshold was set to 12% for both data sets. For the Chemical 340 dataset, the minimum size of the frequent subgraph was set to 5 nodes and 5 edges. The total number of frequent subgraphs in the Chemical 340 dataset satisfying the global threshold and minimum graph size requirements was 550. For the Compound 422 dataset, the minimum size of the frequent subgraph was also set to 5 nodes and 5 edges. In addition, we also constrain the maximum size to be 7 nodes and 7 edges. The total number of frequent subgraphs in the Compound 422 dataset satisfying the global threshold and minimum graph size requirements was 562. The graphs in both datasets were randomly distributed among three sites with each site having almost equal number of graphs.

Figure 4 shows the computation time and accuracy results against different local threshold values. Note that the running time depends much more on the local threshold level rather than the global threshold level, since the local threshold determines the

Chemical 340 - multiple sites						
Dataset		Total graphs = 340; Average graph size = (27 nodes, 28 edges); Largest graph size = (214 nodes, 214 edges); Randomly distributed among sites.				
Frequent Subgraphs		Size($Node_{min}$=5, $Edge_{min}$=5); Global Threshold = 20%; Local Threshold = 20%; Number of frequent subgraphs = 134				
Sites	**Average subgraphs per site**	**Step 1 Time (sec)**	**Step 2 Time (sec)**	**Step 3 Time (sec)**	**Total Time (sec)**	**Accuracy (%age)**
3	171	10	830	188	1028	100
4	263	13	2455	336	2804	100
5	284	16	4188	605	4809	100

Fig. 5. Computation time and accuracy vs. number of sites

number of candidates which in turn determines the time taken by steps 2 and 3. For both datasets, the computation time decreases as the local threshold value increases. This is because increasing the local threshold results in smaller number of local candidate subgraphs and consequently the size of the global candidate set decreases. Also, it is obvious from the results that the computation overhead of step 2 (Generation of global candidate set) dominates all other steps. This step involves encryption of the local candidate set, computing secure union, and removing duplicates.

As expected the accuracy is much higher for local threshold values that are closer to the global threshold or smaller. For example in both datasets, local threshold value of 9% yields 100% accuracy.

The appropriate local threshold is set by the parties in order to generate a reasonable set of candidates. From the security perspective, higher thresholds are better than lower. Therefore one possibility is to start from high threshold and progressively lower it to get an interesting set of results. This incremental computation does not incur any additional privacy loss since the results obtained at a higher threshold level are a subset of the results obtained at a lower threshold.

Figure 5 compares the computation time results for the Chemical 340 dataset distributed among 3, 4, and 5 sites. For this experiment both local and global threshold was set to 20%. The computation time increases with the number of sites. This is mainly due to the increase in number of messages for commutative encryption and decryption in step 2. Moreover, as the number of sites increases the coordinator has to interact with more sites for receiving the frequency count vector and summing them up for removal of non-frequent subgraphs (step 3).

5 Related Work

Privacy-preserving Data Mining (PPDM). The Work in PPDM has followed two major directions: i) randomization/perturbation; and ii) secure multiparty computation.

In the perturbation approach data is locally perturbed by adding "noise" before mining is done. For example, if we add a random number chosen from a Gaussian distribution to the real data value, the data miner no longer knows the exact value. However, important statistics on the collection (e.g., average) will be preserved. Agrawal and Srikant [2] introduced this notion as PPDM to the data mining community. Zhu and Lei [25] study the problem of optimal randomization for privacy-preserving data mining and demonstrate the construction of optimal randomization schemes for density estimation.

The alternative approach of using cryptographic techniques to protect privacy was first utilized for the construction of decision trees by Lindell and Pinkas[11]. Later, these techniques were utilized in many subfields of data mining, e.g. association rule mining [21], clustering[8], classification [4, 19, 22], outlier detection [20] and regression [9, 16]. Our work presents a secure method for frequent subgraph mining, which also follows the same line of research.

All of the cryptographic work falls under the theoretical framework of Secure Multiparty Computation [24, 5].

Frequent Subgraph Discovery. The graph mining techniques, in general, can be categorized into two categories:i) apriori-based approaches and pattern-growth based approaches.

In the first category, the apriori-based approaches follow the idea of apriori algorithm in frequent pattern mining for itemsets [1] – all the subgraphs of a discovered frequent subgraph are also frequent. AGM (apriori-based graph mining) [7], FSG (frequent subgraph discovery) [10] and PM (path mining) [6] enumerate candidate subgraphs using vertices, edges and edge-disjoint paths respectively. Specifically, AGM [7] discovers frequent subgraphs that occur above the percentage threshold of all graphs and uses a canonical representation of subgraphs for improving the efficiency of checking the subgraph isomorphism. FSG [10] generates candidates with the edges which is shown in the adjacency matrix. The class of subgraphs discovered to connected subgraphs has been limited, and several heuristics have been proposed in [10] to improve the efficiency of computing the subgraph support. Meanwhile, the efficiency of generating pattern candidates is also guaranteed. Similar to AGM and FSG, PM [6] also generates candidate subgraph patterns using breadth-first enumeration. Nevertheless, this approach utilizes edge-disjoint paths to generate candidate patterns which reduces the number of iterations while still maintaining the completeness of the search space.

In the second category, the algorithm of gSpan (graph-based Substructure pattern mining) [23] discovers frequent subgraphs without candidate generation. It encodes the tree representation of graphs rather than the adjacency matrix using depth-first search code which provides a lexicographical order for searching the candidate patterns (subgraphs). gSpan algorithm performs efficiently not only on reducing the runtime cost but also saving memory space.

6 Conclusions and Future Work

In this paper, we have looked at the problem of finding frequent sub-graphs from a large distributed set of graphs in a privacy-preserving fashion. Our algorithm is flexi-

ble and can use any underlying subgraph discovery approach as a subroutine. We have implemented our approach and the experimental evaluation shows that our approach is effective and allows a trade-off between utility and computation time. While we conducted the experimental evaluation with pharmaceutical data that have relatively small graph size, we plan to follow on with experiments on social network data. In the future, we also plan to consider the case of a single global graph, which is distributed between multiple parties (this happens in many cases such as transactions shared between financial organizations, call graphs, etc.) Here, you can find local frequent substructures as described in our paper, however, the inter-edges cause a problem. This could perhaps be solved using the graph duality restructuring approach (by building the dual of the graph, with nodes becoming edges, and vice versa). We plan to explore this in the future.

Acknowledgements

The work of Mehmood and Shafiq is supported in part by the LUMS Departmental Research Grant. The work of Vaidya is supported in part by the National Science Foundation under Grant No. CNS-0746943. The work of Atluri is supported through the IR/D by the National Science Foundation.

References

1. Agrawal, R., Srikant, R.: Fast algorithms for mining association rules. In: Proceedings of the 20th International Conference on Very Large Data Bases. pp. 487–499. VLDB, Santiago, Chile (Sep 12-15 1994), http://www.vldb.org/dblp/db/conf/vldb/vldb94-487.html
2. Agrawal, R., Srikant, R.: Privacy-preserving data mining. In: Proceedings of the 2000 ACM SIGMOD Conference on Management of Data. pp. 439–450. ACM, ACM, Dallas, TX (May 14-19 2000), http://doi.acm.org/10.1145/342009.335438
3. Chittimoori, R.N., Holder, L.B., Cook, D.J.: Applying the subdue substructure discovery system to the chemical toxicity domain. In: Proceedings of the Twelfth International Florida Artificial Intelligence Research Society Conference. pp. 90–94. AAAI Press (1999), http://dl.acm.org/citation.cfm?id=646812.707494
4. Du, W., Zhan, Z.: Building decision tree classifier on private data. In: Clifton, C., Estivill-Castro, V. (eds.) IEEE International Conference on Data Mining Workshop on Privacy, Security, and Data Mining. vol. 14, pp. 1–8. Australian Computer Society, Maebashi City, Japan (Dec 9 2002), http://crpit.com/Vol14.html
5. Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game - a completeness theorem for protocols with honest majority. In: Proceedings of the 19th ACM Symposium on the Theory of Computing. pp. 218–229. ACM, New York, NY (1987), http://doi.acm.org/10.1145/28395.28420
6. Gudes, E., Shimony, S.E., Vanetik, N.: Discovering frequent graph patterns using disjoint paths. IEEE Trans. on Knowl. and Data Eng. 18, 1441–1456 (November 2006), http://dx.doi.org/10.1109/TKDE.2006.173
7. Inokuchi, A., Washio, T., Motoda, H.: Complete mining of frequent patterns from graphs: Mining graph data. Mach. Learn. 50, 321–354 (March 2003), http://dl.acm.org/citation.cfm?id=608108.608123

8. Jagannathan, G., Wright, R.N.: Privacy-preserving distributed k-means clustering over arbitrarily partitioned data. In: Proceedings of the 2005 ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. pp. 593–599. ACM, Chicago, IL (Aug 21-24 2005)
9. Karr, A.F., Lin, X., Sanil, A.P., Reiter, J.P.: Secure regressions on distributed databases. Journal of Computational and Graphical Statistics 14, 263 – 279 (2005)
10. Kuramochi, M., Karypis, G.: Frequent subgraph discovery. In: Cercone, N., Lin, T.Y., Wu, X. (eds.) ICDM. pp. 313–320. IEEE Computer Society (2001)
11. Lindell, Y., Pinkas, B.: Privacy preserving data mining. Journal of Cryptology 15(3), 177–206 (2002)
12. Mukherjee, M.: Graph-based data mining for social network analysis. In: In Proceedings of the ACM KDD Workshop on Link Analysis and Group Detection (2004)
13. Paillier, P.: Public key cryptosystems based on composite degree residuosity classes. In: Advances in Cryptology - Eurocrypt '99 Proceedings, LNCS 1592. pp. 223–238. Springer-Verlag (1999)
14. Pohlig, S.C., Hellman, M.E.: An improved algorithm for computing logarithms over GF(p) and its cryptographic significance. IEEE Transactions on Information Theory IT-24, 106–110 (1978)
15. Rakhshan, A., Holder, L.B., Cook, D.J.: Structural web search engine. International Journal on Artificial Intelligence Tools 13(1), 27–44 (2004)
16. Sanil, A.P., Karr, A.F., Lin, X., Reiter, J.P.: Privacy preserving regression modelling via distributed computation. In: KDD '04: Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining. pp. 677–682. ACM Press, New York, NY, USA (2004)
17. Su, S., Cook, D.J., Holder, L.B.: Application of knowledge discovery to molecular biology: Identifying structural regularities in proteins. In: Pacific Symposium on Biocomputing. pp. 190–201 (1999)
18. Vaidya, J., Clifton, C.: Privacy-preserving k-means clustering over vertically partitioned data. In: The Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. pp. 206–215. ACM, Washington, DC (Aug 24-27 2003), `http://doi.acm.org/10.1145/956750.956776`
19. Vaidya, J., Clifton, C.: Privacy preserving naïve bayes classifier for vertically partitioned data. In: 2004 SIAM International Conference on Data Mining. pp. 522–526. SIAM, Philadelphia, PA (Apr 22-24 2004)
20. Vaidya, J., Clifton, C.: Privacy-preserving outlier detection. In: Proceedings of the Fourth IEEE International Conference on Data Mining (ICDM'04). pp. 233–240. IEEE Computer Society Press, Los Alamitos, CA (Nov1 - 4 2004)
21. Vaidya, J., Clifton, C.: Secure set intersection cardinality with application to association rule mining. Journal of Computer Security 13(4), 593–622 (Nov 2005)
22. Vaidya, J., Clifton, C., Kantarcioglu, M., Patterson, A.S.: Privacy-preserving decision trees over vertically partitioned data. ACM Trans. Knowl. Discov. Data 2(3), 1–27 (2008)
23. Yan, X., Han, J.: gspan: Graph-based substructure pattern mining. In: ICDM. pp. 721–724 (2002)
24. Yao, A.C.: How to generate and exchange secrets. In: Proceedings of the 27th IEEE Symposium on Foundations of Computer Science. pp. 162–167. IEEE, IEEE Computer Society, Los Alamitos, CA, USA (1986)
25. Zhu, Y., Liu, L.: Optimal randomization for privacy preserving data mining. In: KDD '04: Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining. pp. 761–766. ACM Press, New York, NY, USA (2004)