NOTES ON PRIVACY-PRESERVING DISTRIBUTED MINING AND HAMILTONIAN CYCLES

Renren Dong and Ray Kresman
Department of Computer Science, Bowling Green State University, Bowling Green, OH, U.S.A.

Keywords: Data mining, Privacy-preserving mining, Hamiltonian cycle, Edge disjoint Hamiltonian cycle.

Abstract: Distributed storage and retrieval of data is both the norm and a necessity in today's computing environment. However, sharing and dissemination of this data is subject to privacy concerns. This paper addresses the role of graph theory, especially Hamiltonian cycles, on privacy preserving algorithms for mining distributed data. We propose a new heuristic algorithm for discovering disjoint Hamiltonian cycles in the underlying network. Disjoint Hamiltonian cycles are useful in a number of applications; for example, to ensure that someone's private data remains private even when others collude to discover the data.

1 INTRODUCTION

In our everyday lives, we see the use and collection of various kinds of data by a number of entities. Data mining is the process of extracting hidden patterns from the underlying data repository. It provides techniques for new knowledge discovery, finding hidden patterns in the data set such as classifiers and association rules (Han and Kamber, 2006). With the explosion of the internet, distributed storage and retrieval of data is both the norm and a necessity. Distributed data mining (DDM) helps mine this data by distributing the mining computation too.

The globally networked society places great demand on the dissemination and sharing of private data. For example, imagine a situation where competing schools trying to discover a set of core classes associated with high grades. While knowing that calculus, English 10, and Spanish 101 are associated with high scholastic achievement on an aggregate level, it would be unfavorable, in particular, for Underdog Academy to admit its rampant failure at teaching any of these subjects with excellence. Hence, Underdog Academy would like to know what classes to put its small budget into without giving its competition a leg up on its weaknesses (Shepard, 2007).

Privacy issues are also relevant to a host of other domains, including banking (Bottcher and Obermeier, 2008), patient medical records (Karr et al., 2007), electronic voting (Baiardi et al., 2005), and others (Chaum, 1988; Verykios et al., 2004) . Privacy preserving distributed data mining (PPDM) algorithms mine distributed data while addressing these privacy concerns.

Secure multiparty computation (SMC) is the basic operation in PPDM. SMC was first introduced in (Andrew, 1986). It is predicated on the notion that a computation is secure if at the end of the computation, no party knows anything except its own input and the results. Hamiltonian cycles, and its derivative edge-disjoint Hamiltonian cycles, play a crucial role in SMC/PPDM.

This paper is about privacy preserving distributed data mining algorithms and the role of these cycles, specifically edge disjoint ones. The rest of this paper is organized as follows. In Section 2 we formally define these cycles. We also provide the motivation for our work and describe a popular approach that uses these cycles to mine distributed data. In Section 3, we take a closer look at the cycles and state two new theorems. Then, we propose a heuristic method for finding the edge disjoint cycles. Concluding remarks are given in Section 4.

2 HAMILTONIAN CYCLES AND MINING

2.1 Mathematical Preliminaries

Hamiltonian cycles (HC) play an important role in graph theory. A Hamiltonian cycle, H, is a cycle

Dong R. and kresman R. (2010).
NOTES ON PRIVACY-PRESERVING DISTRIBUTED MINING AND HAMILTONIAN CYCLES.
In *Proceedings of the 5th International Conference on Software and Data Technologies*, pages 103-107
DOI: 10.5220/0002922001030107

that visits every node in the graph exactly once before returning to the source node. Alternatively, a Hamiltonian cycle can be described as a sequence of edges where any two adjacent edges share exactly one unique node in the graph, where the first and last edge in the cycle share exactly one unique node in the graph, and where the number of edges in H equals the number of nodes in the graph. In graph $G = (V,E)$, if we define Hamiltonian cycle as $H = \{h_1, h_2, \ldots, h_N\}$, $h_i \in E$, $1 \leq i \leq N$, we will have $N = |V|$, and $V = (\cup_{i=1}^{N-1} h_i \cap h_{i+1}) \cup (h_N \cap h_i)$.

Now we define an edge-disjoint Hamiltonian cycle (EDHC). A set of Hamiltonian cycles is called edge-disjoint if every edge in one Hamiltonian cycle is disjoint with every edge in the rest of the Hamiltonian cycles of the set. Formally, suppose we have a set $H = \{H^{(i)} | H^{(i)}$ is a Hamilton cycle in graph $G\}$, $|H|$ is the size of H. H is edge-disjoint if $\forall e \in H^{(i)}$, $e \neq d$, $\forall d \in H^{(j)}$, $1 \leq i \neq j \leq |H|$.

EDHC is useful in computer networks. It can be used to improve the capacity of the network or to provide fault-tolerance and computer security (Urabe et al., 2007b). For example, if a route in a network is faulty or compromised, traffic can be routed through another - edge disjoint cycle or - route that does not share any edges with the previous route. So, an interesting question is how to find the various EDHCs.

Other researchers have investigated this problem for specialized network topologies. Alspach et. al., (Alspach et al., 1990) provide an algorithm to generate the EDHCs for fully connected networks. Bae and Bose (Bae and Bose, 2003) give a method for generating EDHCs in Torus and k-ary n-cubes by using Gray Codes. A polynomial time distributed algorithm for k-ary n-cubes and hypercubes is given in (Stewart, 2007). Our research, in this paper, appeals to the generation of EDHCs in more general networks, and to their application in privacy preserving data mining

2.2 An Example

A number of useful primitives for privacy preserving distributed data mining are given in (Clifton et al., 2002), including secure set union (Bottcher and Obermeier, 2008), secure size of intersection, and secure sum, the last of which we use in this paper. Many more algorithms exist, see (Verykios et al., 2004) for a review. Secure sum (Schneier, 2007) can be defined formally as follows.

- *Secure Sum* (SS): The goal of SS is that the value of each site's individual input be masked while the global sum of all inputs is universally known. This means that at the end of the computation, no party knows anything except its own input and the global sum. Assume we have N sites. For each site we have a value V_i, $1 \leq i \leq N$ and we want to calculate global sum $V = \sum_{i=1}^{N} V_i$.

Figure 1 shows an example of HC (Dong and Kresman, 2009). Assume the presence of a HC as shown in Figure 1. The algorithm initiates a SS computation starting at the source station p_1 whose private value is $x_1 = 8$. p_1 uses a random r_x, $r_x = 5$, to mask its private value. Then p_1 sends $x_1 + r_x = 13$ to the next station. p_2 receives $x_1 + r_x = 13$ and adds its local value, x_2, to the sum before sending it to the next station. At the end, start node, p_1, receives $r_x + \sum_{i=1}^{5} x_i$. Since p_1 selected $r_x = 5$, the global sum can be easily calculated. A similar algorithm can also be used to compute secure set union and secure set intersection (Clifton et al., 2002).

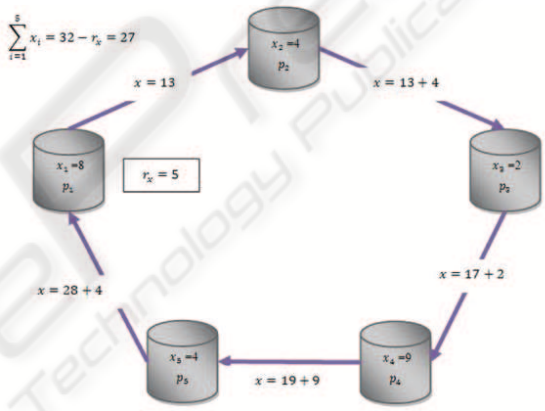

Figure 1: Secure Sum Computation - One Cycle.

Clifton and Vaidya (2002) lay the foundation for a cycle-partitioned secure sum (CPSS) algorithm by noting how any node in a secure sum can divide its value into random shares such that each share is passed along a different secure sum circuit or cycle. Collusion protection is then achieved when no node has the same neighboring nodes twice for each cycle. (See (Urabe et al., 2007a) for additional discussions.) An example of CPSS is given in Figure 2 (Dong and Kresman, 2009). Here, p_3 is connected to 4 other participants which means that at least 4 other participants have to collude - or join together - in order to discover p_3's contribution to the global sum.

Note that the HCs used in Figure 2 are edge-disjoint. However, the number of EDHCs in a given graph is usually very limited. The limited number of EDHCs in the graph constrains the application of CPSS algorithms. So an interesting problem is how to enumerate the EDHCs in a graph: in the rest of this paper, we refer to it as the EDHC problem or just EDHC.

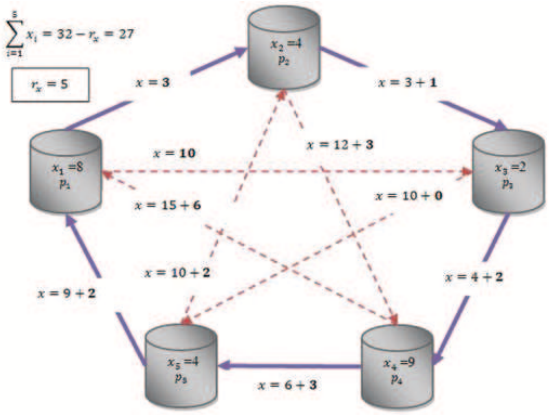

Figure 2: Secure Sum Computation - Two Cycles.

3 OUR CONTRIBUTION

Recall that edge-disjoint HC is a special type of Hamiltonian cycles. It is well-known that HC is NP-complete (Garey et al., 1979). We have proved the following two theorems. However, because of the application nature of this conference, we state them without the underlying mathematical proof (please see (Dong and Kresman, 2010) for the proof).

- *Theorem 1*: The EDHC problem is NP Complete.
- *Theorem 2*: The order in which the various edge-disjoint HC is discovered may limit the number of HC that can be found.

In Section 2 we noted the importance of EDHC for privacy preserving data mining algorithms. In this section we focus on how we can find these EDHCs. One obvious way to find EDHC is to use a greedy algorithm that invokes the single HC algorithm repeatedly. However, due to Theorem 2 above, a simple greedy algorithm is not guaranteed to find many edge-disjoint HCs in a given graph.

Theorem 1 notes the difficulty of solving the EDHC problem. In fact, even the decision problem of determining whether a given graph has a given number of edge disjoint Hamiltonian cycles can also shown to be NP complete; again, we omit the proof here, but it can be found in (Dong and Kresman, 2010). Just like many other NP-complete and NP-hard problems, we do not have any efficient (polynomial time) algorithm for the EDHC problem. Heuristic algorithms, which work by using some heuristic rules to guide their search of the solution space, are powerful tools to solve NP-hard problems. However, they may converge to an approximate or suboptimal solution.

In this section, we will first present a connection between the EDHC problem and another graph theoretic problem, maximum clique (MC): for MC, given an undirected graph G and a positive integer k, we ask if graph G has a fully-connected subgraph $SG(V,E)$ with the constraint $|SG(V)| \geq k$. In the later part of this section, we will provide a heuristic algorithm for EDHC problem.

MC is one of the fundamental NP-complete problems (Garey et al., 1979) and is well researched. The purpose of finding a connection between EDHC and MC is to apply some of the results from MC to the EDHC. We present a procedure to transform the EDHC problem to MC. Note, we assume that we have an oracle machine that lists all the HCs in a given graph. The oracle machine we assume here is only used to explain the transform and it will be replaced by another algorithm later (see Section 3.1). The transform works as follows: take a graph $G(v,e)$ as an input to the EDHC problem; then use the oracle machine to obtain a set *HCPool* which contains all the HC in the given graph; finally, construct another graph $G'(v,e)$ in which each vertex in G' has a corresponding HC in the set *HCPool*, and if two elements in *HCPool* is edge-disjoint then there is an edge between the two corresponding vertices in G' and vice versa. Then, solving the MC problem in G' gives us the results for the original EDHC problem.

Now, the question is what is a good MC algorithm that can be used as a basis for our EDHC problem. Pullan and Hoos (2006) present a stochastic MC algorithm – that does a local search – called Dynamic Local Search (DLS-MC). According to them, the DLS-MC outperforms other state-of- the-art - MC search algorithms in many instances. See (Pullan and Hoos, 2006) for a discussion of other algorithms. In our work, we will take DLS-MC as a basis to construct a dynamic local search algorithm to solve EDHC.

We explain DLS-MC in the next paragraph. Then, we note the process for constructing the oracle that can enumerate the HCs. Finally, we formally describe our dynamic local search algorithm in Section 3.1.

DLS-MC algorithm works as follows: start with a random initial vertex from the given graph as a current clique C. Then the search alternates between an iterative improvement phase and a plateau search phase. In the iterative improvement (or expand) phase, suitable vertices (connect to each element in current clique) are repeatedly added to current clique. The expand phase terminates when there is no more vertex can be added into C. In the plateau search phase, one vertex of current clique C is swapped with a vertex currently not in C. The plateau search terminates when all the possible swap operation have been tried. Following the plateau search, a numerical value is added to each vertex as vertex penalties. The al-

gorithm repeats the above steps - expand phase and plateau phase - until large enough clique is found or the step limitation is reached. Algorithm details and empirical results can be found in (Pullan and Hoos, 2006).

As noted earlier, we can transform the EDHC problem to MC, and there are many algorithms for solving MC. The only barrier left is the construction of the oracle machine to enumerate the HCs. Our solution is to replace the oracle machine with some heuristic algorithms (or backtrack algorithm) for HC problem. We will not try to generate all the HCs - an impossible task, due to Theorem 1 above. Instead, we will dynamically construct the graph for the DLS-MC program. In our algorithm, we will consider a HC algorithm for finding HC as a component since it is very hard to determine what the best heuristic HC algorithm is. In practice, one can choose a heuristic HC algorithm based on his/her needs because algorithms that need more accuracy may take up more computing time. In our work, all HC algorithms take two parameters as input, a graph and a random number. We know that most of the HC algorithms start with picking an edge (or vertex) in the graph. The purpose of the random number is to give the algorithm an opportunity to return a different HC each time. All algorithms return a HC if it finds a new HC else return 'NULL.' For notational convenience, we use the term $singleHCAlg(G,r)$ to mean any HC algorithm with parameters G and r.

3.1 A Heuristic Algorithm for EDHC

Based on the above considerations, we introduce a new heuristic algorithm Algorithm 1 for finding ED-HCs: edge-disjoint HC using Heuristic Dynamic Local Search-EDHC. An outline of the algorithm is given in Algorithm 1. For the given graph $G(v,e)$, we start with finding a set of EDHC as current solution *curEDHC* by *expandSearch*. Then, we execute *expandSearch* and *plateauSearch* alternatively. The process repeats until enough EDHC are found or or step limit is reached. For brevity, the code for *expandSearch* and *plateauSearch* are omitted here, but we explain their functionality in the next two paragraphs.

In *expandSearch*, our goal is to expand the *curEDHC* set as much as possible. To achieve this goal, we construct a graph G' by removing all the HCs in *curEDHC* from G, and feed *singleHCAlg* with G'. It is obvious that any new HC found by *singleHCAlg* can be used to expand the size of *curEDHC*. Repeat this process until *singleHCAlg* returns NULL which means no more HC can be found in the remaining graph G'. *expandSearch* returns the updated version of *curEDHC*.

Algorithm 1. Heuristic Dynamic Local Search-EDHC.

```
Procedure HDLS-EDHC(G,targetSize,maxStep)
Iuput:
  G: Graph G(v,e)
  targetSize: Number of HC we want to find
  maxStep: Steps limitation
Output:
  a set of EDHC with size at least target-
  Size or the best EDHC set found.
Begin
  numStep:=0;
  bestEDHC:={};
  curEDHC:={};
  (HCPool,HCPenalties):={};
  Do
    curEDHC:=expandSearch(G,curEDHC);
    If |curEDHC| is at least targetSize
      return curEDHC as result and exit;
    If curEDHC is better then bestEDHC
      replace the bestEDHC with curEDHC;
    curEDHC:= plateauSearch(G,curEDHC);
    update Penalties for HCs in curEDHC;
  While numStep < maxStep;
  Return bestEDHC;
End
```

In *plateauSearch*, we try to find a different EDHC set based on the *curEDHC*. The EDHC set we find in *plateauSearch* should also has $|curEDHC|$ HCs. The main idea behind *plateauSearch* is to replace some of the HCs in *curEDHC* by other new HCs found by *singleHCAlg*. We first construct a new graph, graph G'', by combining a HC in *curEDHC* with the remaining graph G' (after removing all the HC in *curEDHC* from G). So we have $|curEDHC|$ possible G''. We start with the HC with the least penalty vaule. The penalty value here is used to indicate how frequent a HC is used in the previous search process. By selecting the HC with least penalty vaule, we are trying to avoid searching the same group of HCs. Feed *singleHCAlg* with each G''; if any *singleHCAlg* returns a new HC, then replace the new HC with the one used to construct G''. If all *singleHCAlg* return nothing, set *curEDHC* to nothing and then exit. So the program can start over again.

At the end of *plateauSearch*, the cycle penalties are updated by incrementing the penalty values of all cycles in *curEDHC* by 1. So in HDLS-EDHC, if a HC has less penalty, this means the HC is less frequently used in the searching process and vice versa.

In sum, following the computation of the algo-

rithm (Algorithm 1), we return a set of EDHCs with a size at least *targetSize* or we will return the best solution found within the search process as the result, given by *bestEDHC* - the set of EDHCs in *G*.

4 CONCLUDING REMARKS

This paper focused on distributed mining and the role of Hamiltonian cycles in keeping information private. We stated a couple of theorems on edge disjoint Hamiltonian cycles, but without proof. Then, we proposed a heuristic algorithm to enumerate these cycles. These cycles have applications in data mining, network routing and fault tolerant computing. In an extended version of this paper, we will formally prove these theorems and compare the performance of the heuristic algorithm with that of the greedy algorithm.

REFERENCES

Alspach, B., Bermond, J., and Sotteau, D. (1990). Decomposition into cycles 1: Hamilton decompositions. *Cycles and Rays*, page 9.

Andrew, C. (1986). How to generate and exchange secrets. In *Proc. of the 27th IEEE Symposium on Foundations of Computer Science*, pages 162–167.

Bae, M. and Bose, B. (2003). Edge disjoint Hamiltonian cycles in k-ary n-cubes and hypercubes. *IEEE Transactions on Computers*, 52(10):1271–1284.

Baiardi, F., Falleni, A., Granchi, R., Martinelli, F., Petrocchi, M., and Vaccarelli, A. (2005). SEAS, a secure e-voting protocol: design and implementation. *Computers & Security*, 24(8):642–652.

Bottcher, S. and Obermeier, S. (2008). Secure set union and bag union computation for guaranteeing anonymity of distrustful participants. *Journal of Software*, 3(1):9.

Chaum, D. (1988). The dining cryptographers problem: Unconditional sender and recipient untraceability. *Journal of Cryptology*, 1(1):65–75.

Clifton, C., Vaidya, J., Kantarcioglu, M., Lin, X., and Zhu, M. Y. (2002). Tools for privacy preserving distributed data mining. *SIGKDD Explor. Newsl.*, 4(2):28–34.

Dong, R. and Kresman, R. (2009). Indirect disclosures in data mining. In *Frontier of Computer Science and Technology, Japan-China Joint Workshop*, pages 346–350, Shanghai, China. IEEE Computer Society.

Dong, R. and Kresman, R. (2010). Proof of certain properties of edge disjoint Hamiltonian cycles. *Under preparation*.

Garey, M., Johnson, D., et al. (1979). *Computers and Intractability: A Guide to the Theory of NP-completeness*. W.H. freeman San Francisco.

Han, J. and Kamber, M. (2006). *Data mining: concepts and techniques*. Morgan Kaufmann.

Pullan, W. and Hoos, H. (2006). Dynamic local search for the maximum clique problem. *Journal of Artificial Intelligence Research*, 25:159–185.

Schneier, B. (2007). *Applied cryptography: protocols, algorithms, and source code in C*. A1bazaar.

Shepard, S. (2007). Anonymous Opt-Out and Secure Computation in Data Mining. Master's thesis, Bowling Green State University, Bowling Green, OH, USA.

Stewart, I. (2007). Distributed algorithms for building Hamiltonian cycles in k-ary n-cubes and hypercubes with faulty links. *Journal of Interconnection Networks*, 8(3):253.

Urabe, S., Wang, J., Kodama, E., and Takata, T. (2007a). A high collusion-resistant approach to distributed privacy-preserving data mining. *IPSJ Digital Courier*, 3(0):442–455.

Urabe, S., Wong, J., Kodama, E., and Takata, T. (2007b). A high collusion-resistant approach to distributed privacy-preserving data mining. In *Proceedings of the 25th conference on Proceedings of the 25th IASTED International Multi-Conference: parallel and distributed computing and networks*, page 331. ACTA Press.

Verykios, V. S., Bertino, E., Fovino, I. N., Provenza, L. P., Saygin, Y., and Theodoridis, Y. (2004). State-of-the-art in privacy preserving data mining. *SIGMOD Rec.*, 33(1):50–57.