Protecting Privacy in Incremental Maintenance for Distributed Association Rule Mining

W. K. Wong[1] and David W. Cheung[1] and Edward Hung[2] and Huan Liu[3]

[1] Department of Computer Science, The University of Hong Kong, Pokfulam Road, Hong Kong
{wkwong2, dcheung}@cs.hku.hk
[2] Department of Computing, The Hong Kong Polytechnic University, Hung Hom, Kowloon
csehung@comp.polyu.edu.hk
[3] Department of Computer Science and Engineering, Arizona State University Tempe, Arizona, USA
huan.liu@asu.edu

Abstract. Distributed association rule mining algorithms are used to discover important knowledge from databases. Privacy concerns can prevent parties from sharing the data. New algorithms are required to solve traditional mining problems without disclosing (original or derived) information of their own data to other parties. Research results have been developed on (i) incrementally maintaining the discovered association rules, and (ii) computing the distributed association rules while preserving privacy. However, no study has been conducted on the problem of the maintenance of the discovered rules with privacy protection when new sites join the old sites. We propose an algorithm SIMDAR for this problem. Some techniques we developed can even further reduce the cost in a normal association rule mining algorithm with privacy protection. Experimental results showed that SIMDAR can significantly reduce the workload at the old sites by up to 80%.

1 Introduction

Protecting privacy is an important element in many database applications. Many countries set up privacy laws to clearly protect privacy, e.g. Australia, United States, United Kingdom. For example, medical records and personal information of patients in a hospital should not be disclosed. Direct public access to private information stored in databases is not allowed. Some traditional algorithms may hence be rendered infeasible in practice.

In a distributed association rule mining process, branches of the same companies or even different companies cooperate together to find out the global association rules. Apart from the privacy concerns about individual records, each party may not be willing to share its own data or even let other parties know any derived information. Most traditional algorithms cannot work without disclosing sensitive information like the counts of itemsets in a particular party.

There are new approaches to handle the privacy-preserving data mining problems. One approach is to modify the database randomly so that other parties can fully access the modified data. However, data mining algorithms can then produce only approximate results using the modified data. Another approach is to develop new algorithm applying cryptographic techniques so that accurate results can be obtained without direct access to the source data. This kind of approach is more expensive but security can be usually proved (with limited information disclosure). [10] gives a solution to the problem of association rule mining with privacy protection.

When new parties join, the old association rules may become out-dated and need updates. The naive method of recomputing the association rules from scratch is expensive. In fact, we can reduce the cost by using the old results to incrementally update the rules. The maintenance problem of association rules in a centralized database is studied in [5, 4]. They greatly reduce the number of candidate sets required to scan the database and hence reduce the total process time. Our idea is to apply a similar property used in [5, 4] *securely*, and aim to reduce the candidate set size and so the running time.

There is a tradeoff between privacy and efficiency in privacy-preserving problems. The solutions may be even impractical when complete privacy protection is required. In real world applications, controlled and limited information disclosure is usually acceptable. By lowering the restriction on privacy protection, we can achieve a much better performance. We have developed an efficient algorithm with acceptable privacy protection to maintain the association rules. Besides, some techniques we developed can reduce the cost in the recomputation algorithm in [10].

2 Related Work

The problem of association rule mining is to find interesting patterns among large set of data items [1]. The main focus of the problem is on mining large itemsets. An iterative approach is usually used. The k-th iteration finds all the large itemsets with size k. In [6], the problem is extended into a distributed environment. Different sites hold different individual databases. The problem is to find the global association rules. [6] points out that a globally large itemset must be locally large in some of the sites and gives an efficient algorithm to solve the problem.

[5] and [4] studied the maintenance problem of association rules and large itemsets when one needs to update the database. Old large itemsets can be used to save some effort in the new computation. [4] focuses on the maintenance when there are new transactions. [5] is a more general solution which also considers deletions of transactions. The computational cost in these maintenance algorithms is greatly reduced compared to a recomputation.

To solve the problem of association rule mining with privacy protection, some researchers take the data perturbutation approaches [2]. On the other hand, [10] and [11] both proposed secure association rule mining algorithms with

cryptographic techniques. [10] studied the problem with horizontal partitioned databases, i.e., the databases have the same schema. [11] focuses on vertically partitioned databases. The parties share the same set of records with the same primary key but they have different schema. [11] can only handle the two-party case. The multiparty case is solved in [12] using secure set intersections.

We now study the problem of maintenance of association rule mining in horizontally partitioned databases using cryptographic technique. Note that none of the above work handles the maintenance problem in distributed environment with privacy protection. Although [10] can be also used in our problem by total recomputation but it wastes the effort that we have put in before. A more efficient algorithm which protects privacy as well is required.

3 Problem definition

Let I be the set of items. Each transaction K is a subset of items, i.e., $K \subseteq I$. A transaction K contains an itemset X if and only if $X \subseteq K$. Given a support threshold $s\%$, an itemset X is said to be *large* in the database DB if and only if at least $|DB| \times s\%$ transactions contain X, where $|DB|$ is the number of transactions in DB. Given a confidence threshold $c\%$, we find association rules in the form of $X \Rightarrow Y$ where X, $X \bigcup Y$ are large itemsets and $c\%$ of the transactions that contain X also contain Y.

Suppose there are n sites, $S_1, S_2, ..., S_n$. Each site S_i has a private transaction database DB_i , where $i = 1$ to n, all having the same schema. Each site holds a number of transactions, which is DB_i, for $i = 1$ to n. We have found the large itemsets (and the association rules) in $\bigcup_{k=1}^{n} DB_k$. There are r new sites, $S_{n+1}, S_{n+2}, ..., S_{n+r}$ to join the n existing sites. Each of the new sites owns a private database DB_i, for $i = n + 1$ to $n + r$. The goal is to find the new set of association rules more efficiently than simple recomputation.

Definition 1. *Let $X.count_i$ be the support count of X in S_i. An itemset X is said to be globally large if $\sum_{k=1}^{n+r} X.count_k \geq \sum_{k=1}^{n+r} |DB_k| \times s\%$. X is said to be group large in new sites if $\sum_{k=n+1}^{n+r} X.count_k \geq \sum_{k=n+1}^{n+r} |DB_k| \times s\%$. X is said to be group large in old sites if $\sum_{k=1}^{n} X.count_k \geq \sum_{k=1}^{n} |DB_k| \times s\%$.*

Privacy preserving in necessary in our data mining process. Assume all the parties are semi-honest, i.e. each party follows the protocol with the exception that it keeps a record of all its intermediate messages during the execution of the protocol. The formal definition of private multiparty computation in the semi-honest model can be found in [8]. A computation is secure if at the end of the computation, no party (site) knows anything except its own input and the results. Some limited information disclosure in allowed practice as tradeoff between privacy and efficiency.

The input of our problem is the private databases in the sites and the old results (e.g., old large itemsets) in the old sites. Note that the old results are only known to each old site but not the new sites. The support and confidence thresholds are known to all sites. The result of our solution is the new set of large

itemsets and association rules. We should not disclose any other information to any other parties in the mining process apart from these inputs and results (and the limited information disclosure).

4 Secure Protocol Utilities

There are several developed secure protocols which help us solve some of our sub-problems. More details of these protocols can be found in [7, 13, 9].

Secure Sum. Suppose there are n sites, $\{S_1, S_2, ..., S_n\}$, where $n \geq 3$. Each site S_i holds a value v_i. Our goal is to securely find out the sum of these values $s = \sum_{i=1}^{n} v_i$, which has a known upper limit m, i.e., $s \leq m$. Assume S_1 is designated as the master site. First, S_1 generates a random number R which is in the range $[1, m]$. S_1 adds its value v_1 with R and sends the value $(v_1 + R)$ mod m to S_2. Then, for the remaining sites S_j, $j = 2$ to n, S_j receives a value from S_{j-1}, which is equal to $R + \sum_{i=1}^{j-1} v_i$. S_j then adds its own value v_j to it and sends the new value $(R + \sum_{i=1}^{j} v_i)$ mod m to S_{j+1}. When the process finally goes to S_n, S_n will send the final sum $R + \sum_{i=1}^{n} v_i$ to S_1. S_1 then subtracts the received value by R and gets the actual sum of all the values.

Secure Union. Each site S_i holds a set of items $I_i \in I$. The goal is to find the union of the set of items $\bigcup_{i=1}^{n} I_i$ for n sites without revealing the private items to any parties except the owners of the items. A commutative encryption is applied in the solution, i.e., for any permutation of order p, q, $E_{K_{p_1}}(...E_{K_{p_n}}(X)...) = E_{K_{q_1}}(...E_{K_{q_n}}(X)...)$. First, each site encrypts its own items. Next, the site sends the encrypted items to another site. When a site receives an encrypted item, it would then encrypt the item as well and send it to another site which has not encrypted the item yet. The process keeps going until all the sites have encrypted all items. Due to the property of commutative encryption, if the encrypted value is the same, it means the same item, so we can remove the duplicated items. The encrypted items will then be decrypted by the sites one by one, and we get the result we want.

Secure Comparison. Suppose there are two parties, Alice and Bob. Each of them holds a number, a and b respectively. The problem is to find out the larger number without revealing the numbers to each other. Assuming the number is bounded by n, Yao [13] suggested a generic protocol which takes a linear time complexity $O(n)$ to solve the problem. There is a more efficient protocol for solving this problem [9]. The protocol takes $O((\lg n)^2)$ time and can securely find the answer without a trusted third party using one-out-of-two oblivious transfer.The details of this protocol can be found in [9].

5 Incremental Maintenance of Association Rule Mining with Privacy Protection

We propose our solution SIMDAR (Secure Incremental Maintenance of Distributed Association Rules) to perform an incremental update to the found association

rules while protecting privacy. The group large itemsets (in old results) in the old sites, denoted L, is known to all old sites but not the new sites. Let L_k be the set with all itemsets in L with size k. No individual site knows the exact counts of these large itemsets (we will discuss it more in Sec 5.2). We take the Apriori algorithm as the framework and construct our algorithm using an iterative approach. The outline of SIMDAR is shown as follow:

1. Generating the candidate sets
2. Gathering information of candidates in the new data
3. Pruning itemsets and finding large itemset
4. Repeating steps 1-3 until no more large candidates can be found
5. Checking association rules

5.1 Candidate set generation

The aim of candidate set generation is to get a minimized list of itemsets C_k which may be large globally in the k-th iteration.

For the first iteration $k = 1$, we do not have enough information to conclude if an itemset must be small. So we simply include all the itemsets, $C_k = I$ where I is the entire set of items. If $k > 1$, a globally large itemset must be locally large in some new sites or it is group large in the old sites [5]. The Apriori property says that if some of the subsets with size $k-1$ of an itemset X are small, X cannot be large (proved in [1]). So, we first generate local candidates in the new sites, $C_k^i = Apriori\_gen(L'_{k-1} \bigcap LL_{k-1}^i)$ at S_i where L'_{k-1} is the new globally large itemsets with size $k-1$ and LL_{k-1}^i is locally large itemset at site S_i for the $(k-1)$-th iteration. One of the old sites prepares the group large itemsets from the old large results, $C_k^{old} = Apriori\_gen(L'_{k-1}) \bigcap L_k$. Then we perform a Secure Union to find the candidate sets $C_k = \bigcup_{k=1}^{n} C_k^i \bigcup C_k^{old}$.

5.2 Information collection and storage

We can determine if an itemset is large without knowing the counts of itemsets by combining Secure Comparison and Secure Sum [10]. Suppose there are n sites, S_1 to S_n, involved in the Secure Sum process of finding count of X. S_1 is the master site holding the generated random protecting key R_X. The last site S_n gets the sum with random key added, $\sum X.count_i + R_X$, in the last step. Then, if we want to check if $\sum X.count_i \geq c$ for some c, we can perform a Secure Comparison between $\sum X.count_i + R_X$ and $R_X + c$ at S_1 and S_n respectively. Hence we can know if X is large while $\sum X.count_i$ is protected. Instead of summing the support counts, excess count of each item is summed in [10].

Definition 2. *Let $|DB_i|$ be the number of total transactions in DB_i. The excess count of an itemset X at site S_i corresponding to a support threshold $s\%$, denoted as $X.excess_i$, equals $X.count_i - s\% * |DB_i|$.*

Each site, instead of giving $X.count_i$, supplies $X.excess_i$ as the input to Secure Sum. We can check for large itemsets by checking $\sum X.excess_i \geq 0$ using

Secure Comparisons. However, since we have not calculated the exact value, problems arise if we want to reuse this value. We need to store the information in a secure and efficient way.

A simple approach in [10] is that each site stores its local counts of globally large itemsets on its own and we can perform a Secure Sum whenever we need the (excess) count of an itemset. It takes both more time and space compared to normal storage without concerning privacy. Actually, we can store the counts securely in a more efficient way. In our case, the excess counts are all generated by Secure Sum. The master site S_1 keeps the value of random number key R and the last site S_n stores the protected count $X.excess+R$. These information are some intermediate messages which the sites may store it on its own. Thus, storing the protected excesses and the keys does not introduce any further privacy problem. This requires less space and less access time. Sections 5.3 and 5.4 will discuss how we can use such protected values in the future securely.

5.3 Pruning mechanism and checking large itemsets

A globally large itemset must be locally large in some new sites or group large in the old sites. After we have got the candidate set C_k, each new site scans database to get the counts of candidates. We can first prune away itemsets which are locally small in all new sites and not large in the old sites. One site from old sites and all the new sites take part in a Secure Union process. The inputs to the Secure Union process are the locally large itemsets of the new sites and the originally large itemset L_k. After the Secure Union process, we have a possibly smaller candidate itemsets, C'_k.

The handling of the old large itemsets and the new potential large itemsets is different. This eventually requires us to partition the candidate sets into two groups of itemsets. Define $P'_k = C'_k \bigcap L_k$ and $Q'_k = C'_k - P'_k$. For P'_k, we just add the excess counts in the new sites to the stored excess counts. For Q'_k, we first sum the excess in the new sites. If an itemset is group large in new sites, we scan the databases in old sites. However, as new sites do not know L_k, new sites cannot distinguish the groups P'_k and Q'_k and we should not reveal this piece of information to the new sites. Our idea is to make the new sites have the same view in our algorithm for all itemsets. We perform a merged process consisting of four phases to find large itemsets and prune unnecessary candidates.

Phase 1: Pick up participants All the new sites will join and we will pick two old sites to join. For itemset $X \in P'_k$, the two old sites are the sites holding the protected excess count and the protecting key. They can supply stored excess count of X. If $X \in Q'_k$, the two old sites are just randomly picked among all the old sites. These two sites are picked just to make the process looks like the same to the new sites.

Phase 2: Collect information We perform a Secure Sum with all the selected participants in phase 1. One of the participants from old sites is assigned as the master site. The other participant from old sites cannot be the second site or the last site of Secure Sum. So, we will have a new site holding the protected sum and an old site holding the protecting key. The new sites use

their $X.excess_i$ as the inputs to Secure Sum. If $X \in P'_k$, the two old sites use their stored protected excess count and the protecting key as the inputs. For $X \in Q'_k$, the two old sites do not have the count for X in the old sites, and will add 0 to the sum which does not affect the sum. Let Sum_X be the sum from the Secure Sum. The last site gets $Sum_X + R_X$ where R_X is the protecting key hold in the master site. Note that, Sum_X means global excess count if $X \in P'_k$, or excess count in new sites otherwise.

Phase 3: Pruning We check whether $Sum_X \geq 0$ by a Secure Comparison. All itemsets that cannot pass this condition are pruned. If $X \in P'_k$, we are checking whether X is globally large. If $X \in Q'_k$, we are checking if X is group large in new sites.

Phase 4: Final check The itemsets which passed the pruning with the corresponding protected summed values are passed to old sites apart from the two old sites participated in previous phases. The remaining part will be done by the old sites. Let C''_k contain the candidate sets after the second pruning. For each itemset $X \in C''_k$, if $X \in P'_k$, X is large already. If $X \in Q'_k$, the itemset is broadcasted among the old sites requesting a scan for its count. Another Secure Sum is used to get the total excess count. Suppose S_1 is the master site holding the key R_X of last Secure Sum. An old site S_k receives the protected excess of X, $Sum_X + R_X$ from the new sites. Secure Sum continues, starts from S_k until the last site S_l. S_k also adds another random number R'_X to prevent the two old sites that have joined the previous Secure Sum process from discovering a partial excess of a group of sites. S_k sends R'_k to S_l so that S_l can find the protected actual excess count $\sum X.excess_i + R_X$. Finally, we can then check whether X is a globally large itemset by comparing the protected excess and the protecting key at S_l and S_1.

Lemma 1. *SIMDAR privately computes the large itemsets L'_k from a list of candidate itemsets C_k and revealing at most:*

1. *the old sites know some globally small candidates which are locally small in all new sites and group small in old sites.*
2. *the old sites know some globally small candidates which are group small in the new sites.*
3. *the old sites know some globally small candidates which are group large in new sites but group small in old sites.*
4. *the new sites know some globally small candidates which are locally small in all new sites and group small in old sites.*
5. *the new sites know some globally small candidates which are group large in new sites but group small in old sites.*

Proof. According to definition of secure computation in [8], a computation is secure if the view of each party during the execution of the protocol can be effectively simulated given the results, the listed leaked information (which is acceptable), and the input of that party. So, we only need to show the existence of such a simulator for each party in our proof. Secure protocols in Section 4 are not discussed here. They are assumed to be secure in our proof.

Before the pruning, the sites find C'_k by a Secure Union. The old sites can find C'_k by using a set difference on C_k and the set of itemsets in point 1. The new sites can find C'_k by using a set difference on C_k and the set of itemsets in point 4. We will prove the security of pruning phase by phase.

Phase 1. The only communication in this phase is to tell all the sites which two old sites will join the later operations. Each old site, for an itemset $X \in L_k$, knows that the process will pick the sites which hold the protected excess information of X. Otherwise, the old site can simulate as we pick any two old sites randomly. This random picking simulation can also be applied in the view of each new site.

Phase 2. This phase consists of a Secure Sum only.

Phase 3. C''_k is generated by pruning some itemsets in C'_k. The old sites can generate C''_k by using a set difference on C'_k and the set of itemsets in point 2. One of the old sites also receives the summed information from the new sites. Suppose the arithmetic in Secure Sum is mod m. The site randomly chooses a real number in $[0, m)$. As the summed value is protected by a random number and these two numbers also fall in the range $[0, m)$ (after mod m), the view of the party and the output of the simulator are computationally indistinguishable. The probability of seeing a specific value in both is equal.

The new sites can construct a simulated C''_k, denoted C''_{k-sim}. the simulator first add all itemsets in L'_k into C''_{k-sim}. Next, it adds the itemsets in point 5 to C''_{k-sim}. This gives us a simulated C''_k for the new sites. Note that, the simulator in the new sites ends here.

Phase 4. After the new sites gives C''_k to the old sites and the corresponding protected sums, the old sites can divide C''_k into two groups. For an itemset $X \in L_k$, X is automatically add to the large itemsets.

For an itemset $X \notin L_k$, a Secure Sum is performed followed by a Secure Comparison. We can create a simulator for these two protocols in a similar way as proofs in these two protocols. For $X \in L'_k$, the simulation gives a positive result in Secure Comparison. For X in itemsets in point 3, the result in Secure Comparison in the simulation is negative. Note that the two old sites in phase 1 also join the Secure Sum. However, as S_k will add another random number to the sum, each party in Secure Sum gets a value which has two or more variables in $[0, m)$. So, the simulator (which randomly picks a number in $[0, m)$) gives an indistinguishable view of a party. □

5.4 Checking association rules

First, we sum up the database size in each site by a Secure Sum process. All the new sites join in this Secure Sum. As we may have already stored the total database size in the old sites, the two old sites storing the protected total database size can be representatives and give the protected total database size and the protected key as input to Secure Sum. The total database size is used to find out the confidence of an association rule from excess counts of itemsets. Let $T = \sum_{i=1}^{n+r} |DB_i|$. Let R_T be the generated random key in the Secure Sum process to protect the total database size. At the final stage, a site S_u holds the

value of $T + R_T$ and another site S_v holds R_T. S_u and S_v will not store the protected excess count and the protecting key of an itemset.

When we check an association rule $X \Rightarrow Y$, we need to check whether $\frac{Z.count}{X.count} \geq c\%$ where Z is $X \bigcup Y$. However, what we have got for X and Z are $X.count - s\% * T + R_X$, R_X, $Z.count - s\% * T + R_Z$, and R_Z. These values are available in two to four sites. Besides, we also get $T + R_T$ and R_T from S_u and S_v. We may rephrase $\frac{Z.count}{X.count} \geq c\%$ as follows:

$$\begin{aligned}\frac{Z.count}{X.count} \geq c\% &\Longleftrightarrow Z.count - c\% * X.count \geq 0 \\ &\Longleftrightarrow (Z.excess + R_Z) - c\% * (X.excess + R_X) - R_Z + (c\% * R_X) \\ &\quad +s\%(1 - c\%) * (T + R_T) + (c\% - 1) * s\% * R_T \geq 0\end{aligned}$$

The six terms in the final inequality can be derived from the stored values we have. Thus, we can perform a Secure Sum process to add all these six terms together and check if the sum is greater than zero.

6 Experiments

We carried out a set of experiments to analyze (i) the efficiency of the algorithm, and (ii) the overhead introduced with privacy protection. We take CPU time as the measurement of cost and do not take idle time into account. We implemented two programs for comparisons. One program, which we call it SEC, is a simple privacy preserving mining algorithm without considering incremental maintenance. SEC (re)computes the new set of large itemsets and association rules securely. Efficiency of our algorithm is measured by comparing the CPU time used by SEC and our algorithm. We present the efficiency as the reduction ratio of SIMDAR over SEC. Another program we implemented, which we call it MAN, is a maintenance algorithm without privacy concerns. MAN uses simple messages for communication instead of secure protocols. Overhead of privacy protection of our algorithm is measured as the difference between CPU time consumed by MAN and our algorithm.

Definition 3. *Let t_{SEC} (resp. t_{SIM}, t_{MAN}) be the average CPU time consumed by sites when running SEC (resp. SIMDAR , MAN).*

Let Eff_{PP} denote the efficiency of maintenance algorithm with privacy protection, represented as a ratio. $Eff_{PP} = \frac{t_{SEC} - t_{SIM}}{t_{SEC}}$.

Let OH_{PP} be the overhead of privacy protection. $OH_{PP} = t_{SIM} - t_{MAN}$.

In the experiments, each site was simulated using a stand-alone computer (Dell Optiplex Gx240SD Pentium 4 1.7 GHz computers running Linux). We generated a large number of transactions using IBM synthetic data generator [3]. We supplied three parameters to the data generator: (i) number of transactions, (ii) number of items, and (iii) the length of maximal potentially large itemsets. We used the default values for other parameters. In order to introduce a larger difference between the large itemsets in the old sites and that in the new sites,

we set the length of maximal large itemsets of databases in the old sites to be 6 and that in the new sites to be 8. We set the number of items to 1000.

We performed three sets of experiments with the following varying factors:

Database sizes. The database size varies from 200K to 1M. We have 5 old and 5 new sites. The support threshold is set to 2%.

Support thresholds. The support threshold varies from 0.75% to 2%. We have 5 old and 5 new sites. The database size is set to 500K.

Ratios of the number of old sites to that of the new sites. We have in total fifteen sites. The number of old sites increases from 3 to 12 linearly while the number of new sites decreases from 12 to 3 respectively. The support threshold is 2%. The database size is 500K.

6.1 Database sizes

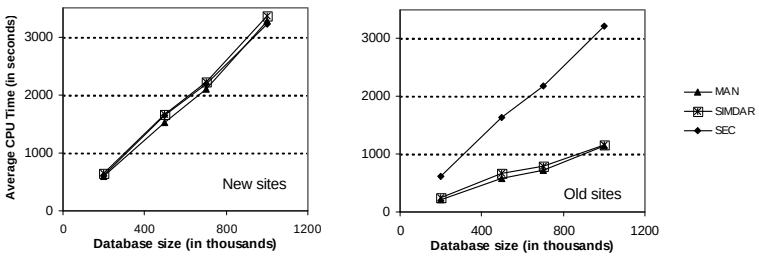

Fig. 1. Average CPU time with varying database size in each site, 5 old sites and 5 new sites, 2% support threshold

Figure 1 shows the average CPU time in the new sites and in the old sites in this experiment. It shows that the CPU time is approximately linear to the database size for both new and old sites. All programs have a similar CPU time in new sites but SIMDAR and MAN have a lower CPU time in old sites. It shows that the incremental maintenance technique can efficiently reduce the CPU time for old parties. Eff_{PP} varies from 59.6% to 63.8% in the old sites. MAN has the lowest CPU time in all cases because both SEC and SIMDAR have implemented secure protocols like Secure Comparison which induce additional cost. However, as the major workload actually goes to the scanning of databases, the additional cost of secure protocols is relatively low. OH_{PP} takes 2.5% to 8.8% of the CPU time in the new sites and 2% to 13% of the CPU time in old sites.

6.2 Support threshold

Figure 2 shows the average CPU time in the new sites and in the old sites in this experiment. MAN and SIMDAR perform better than SEC in the old sites. As the support threshold decreases, the gap between SEC and SIMDAR increases significantly. Eff_{PP} increases from about 63.4% (3% support threshold) to about

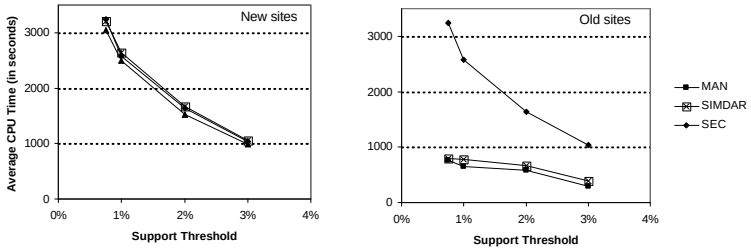

Fig. 2. Average CPU time with varying support threshold, 5 old sites and 5 new sites, $500K$ transactions in each site

75.4% (0.75% support threshold). When the number of large itemset increases, the number of candidate sets generated is exponentially increased. However, a large portion of candidate sets is pruned. OH_{PP} takes 5.4% to 8.9% in the new sites and 4.1% to 25.0% of the CPU time in the old sites.

6.3 Ratio of old sites to new sites

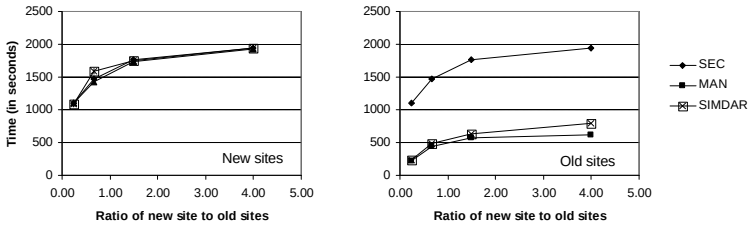

Fig. 3. Average CPU time in the old sites with varying ratio of number of old sites to new sites, 15 sites in total, 2% support threshold, $500K$ transactions in each site

Figure 3 shows the average CPU time in the new sites and in the old sites in this experiment. The average CPU time decreases when the number of old sites increases for all three programs in all sites. It is because the total number of large itemsets in the old sites is fewer than that in the new sites. Thus, when the majority is the old sites with fewer large itemsets, the total number of large itemsets and candidates decreases.

Eff_{PP} increases when the proportion of old sites increases. Eff_{PP} at the ratio of 3 old sites to 12 new sites is 58.8%. When the ratio increases to 12 old sites to 3 new sites, Eff_{PP} significantly increases to 79.1%. It is because we have already known the large itemsets of the old sites which become the majority. When the number of old sites increases, it becomes more difficult for new sites to add new changes to the old results.

7 Conclusions

We studied an efficient algorithm to solve the maintenance problem of adding new sites. The developed method SIMDAR can successfully reduce the number of candidate sets required to be scanned in the old sites. Experimental results showed that our algorithm SIMDAR can effectively reduce the workload of the old sites while the cost in the new sites is almost the same as in a recomputation. The entrance cost for a new party is not reduced much but the maintenance cost for an old party is much lower.

After working on the case of addition of new sites, we are now studying other cases: (i) removing sites, and (ii) updates of databases in one or more old sites. It is also challenging to consider the combination of all these cases, which is more likely to happen in practice.

References

1. Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules. In *VLDB*, Santiago, Chile, 1994.
2. Rakesh Agrawal and Ramakrishnan Srikant. Privacy-preserving data mining. In *SIGMOD*, Dallas, Texas, 2000.
3. IBM Almaden Research Center. Synthetic data generation code for association and sequential patterns.
4. David W. Cheung, Jiawei Han, Vincent T. Ng, and C.Y. Wong. Maintenance of discovered association rules in large databases: An incremental updating technique. In *ICDE*, Washington, DC, USA, 1996.
5. David W. Cheung, S.D. Lee, and Benjamin Kao. A general incremental technique for maintaining discovered association rules. In *Database Systems for advanced Applications*, Melbourne, Australia, 1997.
6. David W. Cheung, Vincent Ng, Ada W. Fu, and Yongjian Fu. Efficient mining of association rules in distributed databases. In *Special Issue in Data Mining, IEEE Transaction on Knowledge and Data Engineering, IEEE Computer Society, V8, N6*, Dec 1996.
7. Chris Clifton, Murat Kantarcioglu, Jaideep Vaidya, Xiaodong Lin, and Michael Zhu. Tools for privacy preserving distributed data mining. In *ACM SIGKDD Explorations Newsletter*, 2002.
8. Oded Goldreich. *Foundations of Cryptography*, volume 2. Weizmann Institute of Science, Israel, May 2004.
9. Ioannis Ioannidis and Ananth Grama. An efficient protocol for yao's millionaires' problem. In *HICSS*, Waikoloa Village, Hawaii, 2003.
10. Murat Kantarcioğlu and Chris Clifton. Privacy-preserving distributed mining of association rules on horizontally partitioned data. In *IEEE Trans. Knowledge Data Eng., 16(4)*, July 2004.
11. Jaideep Vaidya and Chris Clifton. Privacy preserving association rule mining in vertically partitioned data. In *KDD*, Edmonton, Alberta, Canada, 2002.
12. Jaideep Vaidya and Chris Clifton. Secure set intersection cardinality with application to association rule mining. In *Journal of Computer Security 13(4), IOS Press*, November 2005.
13. A. C. Yao. How to generate and exchange secrets. In *Proceedings of the 27th IEEE Symposium on Foundations of Computer Sciences.*, 1986.