AUDIO: An Integrity *Audi*ting Framework of *O*utlier-Mining-as-a-Service Systems

Ruilin Liu[1], Hui (Wendy) Wang[1], Anna Monreale[2], Dino Pedreschi[2], Fosca Giannotti[3], Wenge Guo[4]

Stevens Institute of Technology, NJ, USA[1], University of Pisa, Pisa, Italy[2], ISTI-CNR, Pisa, Italy[3], New Jersey Institute of Technology, NJ, USA[4]

{rliu3,Hui.Wang}@stevens.edu,{annam, pedre}@di.unipi.it, fosca.giannotti@isti.cnr.it,wenge.guo@njit.edu

Abstract. Spurred by developments such as cloud computing, there has been considerable recent interest in the data-mining-as-a-service paradigm. Users lacking in expertise or computational resources can outsource their data and mining needs to a third-party service provider (server). Outsourcing, however, raises issues about *result integrity*: how can the data owner verify that the mining results returned by the server are correct? In this paper, we present *AUDIO*, an integrity auditing framework for the specific task of distance-based outlier mining outsourcing. It provides efficient and practical verification approaches to check both completeness and correctness of the mining results. The key idea of our approach is to insert a small amount of artificial tuples into the outsourced data; the artificial tuples will produce artificial outliers and non-outliers that do not exist in the original dataset. The server's answer is verified by analyzing the presence of artificial outliers/non-outliers, obtaining a probabilistic guarantee of correctness and completeness of the mining result. Our empirical results show the effectiveness and efficiency of our method.

1 Introduction

Advances in networking technologies have triggered a new computing paradigm called cloud computing. It allows data owners, especially the ones who have large volume of data but limited budget for data analysis, to outsource their data and data mining needs to a third-party service provider. This is referred as the *data-mining-as-a-service* ($DMAS$) model [26, 30]. The model allows data owners to leverage hardware and software solutions provided by $DMAS$ providers, without developing their own. There are a few active cloud-based $DMAS$ projects in industry. For example, Google provides cloud-based Google Prediction APIs [2].

Among various types of data mining applications, outlier mining, a classic data mining problem, has seen the possibility to be married with the $DMAS$ paradigm. Outlier mining has been in critical need in many real-world applications such as credit card fraud detection, discovery of criminal activities, weather prediction, marketing and customer segmentation. The task of outlier mining is to find data objects that do not comply with the general patterns of the majority. Sometimes outliers are data errors whose existence may affect the data analysis [8, 9]. The problem of outlier detection has been widely studied in the

data mining community [3, 6, 17, 27]. It has been shown that detecting outliers is of high computational complexity [17], becoming prohibitive for data of high dimensionality [3]. Although researchers have identified several important optimization techniques [4, 27] to improve the efficiency of outlier detection, it is difficult for the data owner who lacks of professional expertise to implement these techniques. $DMAS$ provides a natural solution for such data owner who desires to find outliers from her datasets for analysis purpose.

Although outsourcing is advantageous for the data owner of limited capabilities to achieve sophisticated analysis on their large volume of data, it triggers serious security concerns. One of the major security issues is *result integrity*; the question to be answered is how the data owner (client) of weak computational power can be assured that a service provider (server) returns faithful mining results [1, 7]. There are many reasons that the server may return wrong answer. For example, a server may return wrong mining result accidentally due to software bugs, or keep part of the mining result to itself intentionally so that it can sell the retained result to the competitors of the client for profit. There also exists a strong financial incentive for the server to return incorrect answers that require less work and are unlikely to be detected by the client.

We consider two types of service providers, the *semi-honest* server and the *malicious* server, that may return wrong result. The semi-honest server executes the mining algorithm honestly; however, it may modify the outlier mining result by accident. The malicious server executes the mining algorithm unfaithfully (e.g., runs the algorithm on a portion of the dataset) and returns the incorrect result on purpose. Our goal of integrity verification is to enable the client, who is of weak computational power, to verify whether the server that is potentially semi-honest or malicious returns *correct* and *complete* outlier mining result. By *correctness*, we mean that each returned tuple is a true outlier. By *completeness*, we mean that all true outliers are returned by the server.

We design and implement $AUDIO$, a lightweight integrity auditing framework for outlier mining-as-a-service. $AUDIO$ includes two entities, the client and the remote, untrusted third-party server. To catch the semi-honest server, before outsourcing, the client constructs a set of artificial tuples that consist of *artificial outliers* (AOs) and *artificial non-outliers* (ANOs). Both AOs and ANOs are inserted into the original dataset and sent together to the server. Meanwhile, the client maintains a small piece of auxiliary information locally. After receiving the result from the server, the client verifies the correctness and completeness of the result by analyzing the returned outliers against AOs and ANOs, and quantifies the probabilistic guarantees of completeness and correctness. There are a few nice properties of our verification techniques. First, incorrect and incomplete answers from the server can be caught with high confidence by a small number of AOs and ANOs. Second, the complexity of our solution is linear to the number of AOs and $ANOs$, which are independent from the database size. This makes it feasible to efficiently verify the outlier mining results of large databases.

Although it is not new to accomplish verification by inserting counterfeit tuples in other contexts [28, 32] and other data mining problems (e.g., association

rule mining [31]), it has not been explored of how to verify the result of outsourced outlier mining via counterfeit tuples. Indeed, inserting counterfeit tuples leads to unique challenges to outlier mining. For example, since some outliers in the original dataset may not be outliers anymore in the new dataset after insertion, how to construct counterfeit tuples and design verification techniques to ensure all original outliers are still discoverable in the dataset with newly inserted counterfeit tuples? None of the existing techniques based on insertion of counterfeit tuples can address such challenge.

To our best knowledge, we are the first to address the problem of providing integrity assurance for outsourcing of outlier mining. Our contributions include the following:

(1) To catch the semi-honest server, we propose an artificial-tuple (AT) based approach providing both correctness and completeness guarantee by inserting artificial outliers (AOs) and non-outliers (ANOs) into the original data. We formally quantify both correctness and completeness guarantees, and discuss how to design AOs and ANOs so that the outlierness of AOs and non-outlierness of ANOs do not need to be verified by mining of the dataset.

(2) Inserting AOs and $ANOs$ may change the (non)outlierness of the real tuples. We propose a verification mechanism that will not eliminate any true outlier. We also discuss how to remove the false positive outliers (i.e., the non-outliers returned as outliers) introduced by AOs and $ANOs$ and recover all true outliers efficiently.

(3) We define the possible misbehaviors by the malicious server, and show how the malicious server can defeat the AT-based approach. We also discuss the challenges of designing efficient verification approaches to catch the malicious server.

(4) We complement our analytical results with an extensive set of experiments showing the efficiency and the effectiveness of our AT-based approach.

The paper is organized as following. Sec.2 discusses related work. Sec.3 introduces the preliminaries. Sec.4 presents our AT-based approach to catch the semi-honest server. Sec.5 discusses the limits of our AT-based approach to catch the malicious server, as well as the design of deterministic approaches. Sec.6 presents our experimental results. Sec.7 concludes the paper.

2 Related Work

The issue of integrity assurance for database management was initially raised in the database-as-a-service (DAS) paradigm [14]. The studied problem is how to assure the integrity of SQL query evaluation over the hosted relational databases. The proposed solutions include Merkle hash trees [20, 23], signatures on a chain of paired tuples [25], challenge tokens [28], and counterfeit records [32]. [33, 34] extend the security concerns to the data in a metric space and spatial domain. These work share the same integrity concerns in the outsourcing paradigm. However, their focus is different from ours.

The problem of how to protect sensitive data and data mining results in the data-mining-as-a-service (DMAS) paradigm has caught much attention recently. Wong et al. [30] consider utilizing a one-to-n item mapping together with

non-deterministic addition of cipher items to protect the identification of individual items in the scenario that frequent pattern mining task is outsourced. Unfortunately, this work has potential privacy flaws; Molloy et al. [22] show how privacy can be breached in the framework of [30]. Tai et al. [29] consider the same scenario and proposed a database transformation scheme that is based on a notion of k-support anonymity. To achieve k-support anonymity, they introduced a pseudo taxonomy tree; the third party server will discover the generalized frequent itemsets instead. Giannotti et al. [12] define a similar privacy model as k-support that requires each item must be indistinguishable from the other $k-1$ items regarding their frequencies. They provide formal privacy analysis of their k-privacy model on both items and frequent patterns. Although these works focus on frequent pattern mining, their encryption techniques can be applied to our work to provide further protection on data and mining results.

Our problem falls into the category of integrity assurance of data-mining-as-a-service (DMAS) paradigm. However, there is very little work in this category. Wong et al. [31] propose auditing techniques for outsourcing of frequent itemset mining. They generate a (small) artificial database such that all itemsets in the database are guaranteed to be frequent and their exact support counts are known. By hosting the artificial database with the original one and checking whether the server has returned all artificial itemsets, the data owner can verify whether the server has returned correct and complete frequent itemsets. To our best knowledge, Wong et al. [31] are the first (and the only) work that addresses the integrity auditing issue of the DMAS paradigm. Their techniques on frequent itemset mining cannot be directly applied to our problem of outlier mining.

3 Preliminaries

Distance-based Outlier Mining. In this paper, we focus on distance-based outliers, which is well-defined for datasets of any dimension and more suitable for real-world applications where the data distribution does not fit any standard distribution [17]. In particular, an object O in a dataset D is a (p, d)-*outlier* if at least $p\%$ of the objects in D lie greater than distance d from O [17]. Otherwise, O is a non-outlier with regard to the (p, d) setup. For simplicity, we say O is a non-outlier if it is not a (p, d)-*outlier*. We assume $p\% * |D|$ always returns an integer, as it indicates the number of tuples. We use Euclidean distance to measure the distance between two tuples. In particular, given two tuples $t(a_1, \ldots, a_k)$ and $t'(a'_1, \ldots, a'_k)$, $dist(t, t') = \sqrt{\sum_{i=1}^{k}(a_i - a'_i)^2}$.

Outsourcing Setting. In the outlier-mining-as-a-service framework, the data owner (client) outsources her data D, with the configuration of p and d values for (p, d)-outlierness, to the server. The server discovers (p, d)-outliers from D and returns them to the client. We assume the server will return exact outliers instead of approximate ones [16, 18].

Types of Dishonest Servers and Assurance Goal. We consider two types of servers that may return invalid answer.

- The *semi-honest* server that runs the outlier mining algorithm on the outsourced dataset faithfully. However, it may return wrong outliers accidentally due to software bugs or human mistakes when collecting the answer.

– The *malicious* server that returns wrong answer intentionally. For example, it may only examine a portion of the outsourced dataset and returns cheaper (and incorrect) answer. Furthermore, the malicious server tries to escape the verification if it knows the details of the verification mechanism.

Let O be the real outliers in the outsourced data D, and O^S be the outliers returned by the server. We define the *precision* of O as $P = \frac{|O \cap O^S|}{|O^S|}$ (i.e., the percentage of returned outliers that are correct), and the *recall* of O as $R = \frac{|O \cap O^S|}{|O|}$ (i.e., the percentage of correct outliers that are returned). Our aim is to catch incorrect answer (i.e., $P < 1$) and incomplete answer (i.e., $R < 1$) with high probability. To this end, we define (α, a)-*completeness* and (β, b)-*correctness*.

Definition 1. *Given a dataset D and a verification method M, let pr_p and pr_r be the probabilities that M catches the server that returns the result of precision $P \leq a$ and recall $R \leq b$ respectively, where $a, b \in [0, 1]$ are given thresholds. We say M can verify* (α, a)-correctness *if $pr_p \geq \alpha$, and can verify* (β, b)-completeness *if $pr_r \geq \beta$, where $\alpha, \beta \in [0, 1]$ are given thresholds.*

4 Artificial Tuple (AT) based Verification

We develop a verification mechanism using artificial tuples (ATs) to catch the semi-honest server. In particular, before outsourcing the dataset D, the client generates a set of artificial outliers AO and a set of artificial non-outliers ANO respectively. Then the client inserts AO and ANO into the original dataset D, and sends the new dataset $D^+ = D \cup AO \cup ANO$ to the server. Since the semi-honest server cannot distinguish AOs and $ANOs$ from the real tuples in D, it should return all AOs but no $ANOs$, if it is honest. Thus by checking against AOs and ANOs, the client will be able to obtain probabilistic guarantees of completeness and correctness. How to measure the probabilistic guarantees will be discussed in Section 4.2. Next, we first discuss how to construct AOs and ANOs efficiently for preparation of verification (Section 4.1). Second, we discuss how to use AOs and $ANOs$ to accomplish verification (Section 4.2).

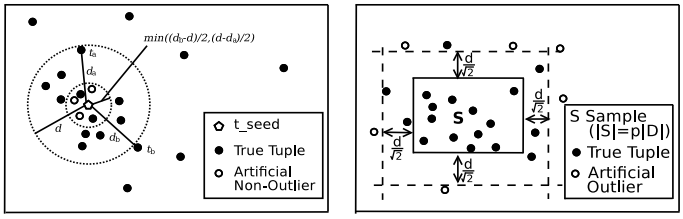
4.1 Verification Preparation

Construction of Artificial Non-outliers (ANOs). Our ANO construction procedure is based on the concept of *close* tuples that we will define soon. Given a tuple t and a distance d, we define two sets, $T_L(t, d)$ that stores all tuples whose distance to t is less than d, and $T_U(t, d)$ that stores all tuples whose distance to t is greater than d. Formally, $T_L(t, d) = \{t' | t' \in D, dist(t, t') < d\}$, and $T_U(t, d) = \{t' | t' \in D, dist(t, t') > d\}$. We say a tuple $t' \in T_L(t, d)$ is the *farthest close neighbor* of tuple t with regard to distance d, if the distance between t and t' is the largest among all tuples in $T_L(t, d)$, and a tuple $t' \in T_U(t, d)$ is the *closest distant neighbor* of tuple t with regard to distance d, if the distance between t and t' is the smallest among all tuples in $T_U(t, d)$. Then we have:

Definition 2. *Given the dataset D of r dimensions and a tuple $t \in D$, let $t_a \in D$ be the farthest close neighbor of t, and $t_b \in D$ be the closest distant neighbor of t. Let P be an r-sphere with t as the centroid, and $min(\frac{d-d_a}{2}, \frac{d_b-d}{2})$ as the radius,*

where d is the distance parameter of (p,d)-outlier mining, $d_a = dist(t,t_a)$, and $d_b = dist(t,t_b)$. Then we call any tuple $t \in P$ a close *tuple to t.*
Next, we show that the close tuples of t have the same distance property as t.

Lemma 1. *Given a tuple t and a close tuple t_c of t, for each tuple $t' \neq t, t_c$, it must be true that: (1) if $dist(t,t') < d$, then $dist(t_c,t') < d$; (2) if $dist(t,t') > d$, then $dist(t_c,t') > d$.*
Proof: *Since t_a is the farthest close neighbor of t, for any t' s.t. $dist(t,t') < d$, we have $dist(t,t') \leq d_a < d$, and $dist(t,t_c) < \frac{d-d_a}{2}$, leading to $dist(t',t_c) \leq dist(t,t')+dist(t',t_c) < d_a+\frac{d-d_a}{2} = \frac{d+d_a}{2}$. Since $d_a < d$, then it must be true that $dist(t_c,t') < d$. Similarly, we have $dist(t,t') \geq d_b > d$, and $dist(t,t_c) < \frac{d_b-d}{2}$. Thus, $dist(t_c,t') \geq dist(t,t') - dist(t,t_c) > d_b - \frac{d_b-d}{2} = \frac{d_b+d}{2}$. Since $d_b > d$, then it must be true that $dist(t_c,t') > d$.* □



(a) ANO construction (b) AO construction

Fig. 1. Construction of *ANO*s and *AO*s

Based on Lemma 1, next, we prove that any close tuple of tuple t always has the same non-outlierness as t.
Theorem 1. *Given a dataset D and any tuple $t \in D$ that is a non-(p,d)-outlier, any close tuple of t must be a non-(p,d)-outlier in D.*

The correctness of Theorem 1 is straightforward as t_c and t have the same number of tuples whose distances is greater than the given distance threshold d.

We make use of Theorem 1 to construct *ANO*s. In particular, we pick a *seed* tuple t_{seed} that is a non-outlier tuple in the original dataset D, and construct its close tuples as *ANO*s. Fig. 1(a) illustrates the construction procedure of *ANO*s. To pick t_{seed}, we repeatedly pick a tuple from the dataset randomly, and verify its non-outlierness, until a non-outlier tuple is reached. The probability that t_{seed} will be picked at the x-th trial is $g(x) = (1 - \frac{f_{to}}{n})(\frac{f_{to}}{n})^{x-1}$, where f_{to} and n are the number of outliers and the number of tuples in D. It is straightforward that $1 \leq x \leq n - f_{to}$. Define $\phi = \frac{f_{to}}{n}$. Then $g(x) = (1-\phi)\phi^{x-1}$, where $1 \leq x \leq n - n\phi$. The expected value of x equals to $E(x) = \frac{(n-\phi n)\phi^{n-\phi n+1} - (n-\phi n+1)\phi^{n-\phi n}+1}{(\phi-1)^2}$. As the outliers always take a small portion of the database, ϕ is a small number (e.g., $\phi = 0.05\%$ [24]). Therefore, $E(x) \approx 1$. In other words, it is highly likely that t_{seed} can be picked by the first random trial.
Construction of Artificial Outlier (*AO*s). Our construction procedure is based on the definition of *distant* tuples. To be more specific, given a r-dimension dataset D and a set of tuples $S \subseteq D$, we say a tuple $t \notin S$ is a *distant* tuple of S if for each tuple t' in S, $dist(t,t') > d$, where d is the parameter setting of (p,d)-outlierness. We have the following lemma to show what kind of tuples can be distant tuples.

Lemma 2. *Given a r-dimension dataset D and a set of tuples $S \subseteq D$, let min_i and max_i be the minimum and maximum value of the i-th ($1 \leq i \leq r$) attribute of S. Then any tuple $t(a_1, \ldots, a_r) \notin S$ is a* distant *tuple of S if there are k ($1 \leq k \leq r$) attributes such that on each attribute A_i, $t[A_i] < (min_i - \frac{d}{\sqrt{k}})$ or $a_i > (max_i + \frac{d}{\sqrt{k}})$.*

The correctness of Lemma 2 follows naturally from the distance functions and the properties of the distant tuples. Next, we show that (p, d)-outliers can be generated from the distant tuples.

Theorem 2. *Given the dataset D and a set of tuples $S \subseteq D$, any distant tuple t of S must be a (p, d)-outlier in D if $|S| \geq p|D|$.*

The correctness of Theorem 2 is straightforward as the tuple t of S is of distance of d to p percentage of tuples in D. Based on Theorem 2, we design the *AO* construction procedure as follows. First, the client picks a sample S of size $\lceil p|D| \rceil$ tuples randomly from D. Second, the client treats S as an r-dimension hypercube $\mathcal{R}$. In $\mathcal{R}$, the edge at the i-th dimension represents the range $[min_i, max_i]$ of the data values in S. Then the client randomly picks k dimensions (possibly $k = 1$) of $\mathcal{R}$. Last, the client expands the k dimensions of $\mathcal{R}$ by $\frac{d}{\sqrt{k}}$ (i.e., change the minimum and maximum value of the i-th attribute to be $min_i - \frac{d}{\sqrt{k}}$ and $max_i + \frac{d}{\sqrt{k}}$). Let the expanded hypercube be $\mathcal{R}'$. Then any tuple t_{ao} that is created outside of $\mathcal{R}'$ must be a (p, d)-outlier of D. Fig. 1(b) illustrates the construction procedure in a 2-dimensional dataset.

Complexity analysis. *ANO*s can be constructed with at most two passes of the original dataset. The complexity of constructing *AOs* is $O(n)$, where n is the size of D. Therefore, the complexity of the verification preparation is $O(n)$.

4.2 Verification

We realized that (p, d)-outliers in the original dataset D may not be (p, d)-outliers in $D^+ = D \cup ANO \cup AO$, as inserting tuples into the original dataset D may change the (non)outlierness of some true tuples in the original dataset. This may ruin the verification method as the semi-honest server may be wrongly caught as returning incorrect answer. Therefore, the client should ensure that all (p, d)-outliers in D appear in mining of D^+. In this section, we discuss: (1) how the client configures the p parameter so that the true (p, d)-outliers in D are still present in D^+, and (2) how to eliminate the *false positives* (i.e., the tuples returned according to the new p parameter but not (p, d)-outliers in D).

First, we show how to guarantee that the (p, d)-outliers in D are still outliers in D^+ by changing the parameter p of (p, d)-outlier mining.

Theorem 3. *Given a dataset D and a set of AOs and ANOs, any (p, d)-outlier in D must be a (p_1, d)-outlier in $D^+ = D \cup AO \cup ANO$, where $p_1 = \frac{p|D|}{|D^+|}$.*

Proof*: For a true tuple $t \in D$, let m and f be the number of true and artificial tuples (including both AOs and ANOs) whose distance to t is at least d in D^+. Now we prove that it must be true that $\frac{m+f}{|D^+|} \geq p_1 = \frac{p|D|}{|D^+|}$. This can be proven by the following. Since tuple t is a (p, d)-outlier in D, it must be true that $m \geq p|D|$. This naturally leads to that $\frac{m+f}{|D^+|} \geq \frac{p|D|}{|D^+|}$, with $f \geq 0$.* □

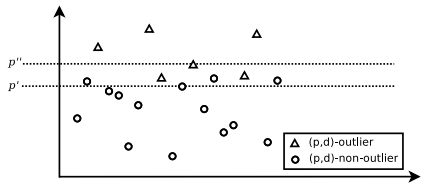


Fig. 2. (p_1, d)-outliers and (p_2, d)-outliers in D^+ VS. (p, d)-outlier in D; O_1: (p_1, d)-outliers in D^+, O_2: (p_2, d)-outliers in D^+.

Following Theorem 3, the client asks for (p_1, d)-outliers in the outsourced dataset D^+; all outliers in D must appear in the answer if the server is honest. Note that all AOs must be appear in (p_1, d)-outliers of D^+ too. However, it is not true that all (p_1, d)-outliers in D^+ must be (p, d)-outliers in D. To eliminate those (p_1, d)-outliers in D^+ that are not (p, d)-outliers in D, we have:

Theorem 4. *Given a dataset D, let $|AO|$ and $|ANO|$ be the numbers of inserted AOs and $ANOs$ respectively. Then any (p_2, d)-outlier in $D^+ = D \cup AO \cup ANO$ must be a (p, d)-outlier in D, where $p_2 = \frac{p|D|+|AO|+|ANO|}{|D^+|}$.*

Proof*: For a tuple $t \in D^+$, let m and f be the number of true and artificial tuples (including both AOs and $ANOs$) whose distance to t is at least d in D^+. Since the m true tuples must exist in D, we aim to prove that $\frac{m}{|D|} \geq p$. This can be proven as follows. Since t is a (p_2, d)-outlier in D^+, it must hold that $\frac{m+f}{|D^+|} \geq p_2$. This leads to that $m + f \geq p|D| + |AO| + |ANO|$. Since $f \leq |AO| + |ANO|$, it must be true that $m \geq p|D|$. Then the theorem follows.* □

Following Theorem 4, all constructed ANOs must be (p_2, d)-non-outliers in D^+.

Fig. 2 illustrates the relationship among (p_1, d), (p_2, d) outliers in D^+ and (p, d)-outliers in D. For a given tuple t, let per_t be the percentage of tuples in $D^+ = D \cup AO \cup ANO$ whose distance to t is at least d (d: the d parameter for (p, d)-outlierness). Then t will fall into one of the following three categories:

- t is a (p, d)-outlier in D, if $per_t \geq p_2 = \frac{p|D|+|AO|+|ANO|}{|D^+|}$;
- t is a (p, d)-non-outlier in D, if $per_t < p_1 = \frac{p|D|}{|D^+|}$;
- t is either a (p, d)-outlier or a (p, d)-non-outlier in D, otherwise.

Based on both Theorem 3 and Theorem 4, the outsourcing and the verification procedures are designed the following. When outsourcing $D^+ = D \cup AO \cup ANO$ to the server, the client asks for (p_1, d)-outliers and (p_2, d)-outliers, where $p_1 = \frac{p|D|}{|D^+|}$, and $p_2 = \frac{p|D|+|AO|+|ANO|}{|D^+|}$. Note that $O_2 \subseteq O_1$; therefore the client can get both sets of outliers by outsourcing the task once.

After receiving the (p_1, d)-outliers O_1 and (p_2, d)-outliers O_2 from the server, the client verifies the completeness and correctness as the following.

Completeness Verification. To verify whether the server has returned all true outliers, the client checks whether $AO \subseteq O_1$. If $AO \not\subseteq O_1$, the client catches the incomplete outlier answer with 100%; otherwise, the client computes the completeness probability $pr_r = 1 - \alpha^{|AO|}$. To satisfy (α, a)-completeness (i.e., $pr_r \geq a$), the client has to construct AOs of size

$$|AO| = \lceil log_{\alpha}(1-a) \rceil. \quad (1)$$

Correctness Verification. For the correctness verification, the client checks whether $ANO \cap O_2$ is empty. If it is not, the client catches the incorrect answer with 100%; otherwise, the client returns the correctness probability $pr_p = 1 - \beta^{|ANO|}$. To meet the (β, b)-correctness (i.e., $pr_p \geq b$) requirement, $|ANO|$ must satisfy that

$$|ANO| = \lceil log_{\beta}(1-b) \rceil. \quad (2)$$

Equation 1 and 2 show that $|AO|$ and $ANOs$ are independent of the size of the original dataset; thus our mechanism is especially useful for large datasets. Furthermore, we observe that it does not need large number of $|ANO|$ and $|AO|$ to catch with high correctness probability the server that changes a small fraction of result. For instance, when $\alpha = 0.99$ (i.e., the server changes 1% of the outliers) and $a = 0.99$ (i.e., the probability to catch such answer is at least 0.99), we only need to add 459 AOs.

Overhead Analysis. The complexity of distinguishing $ANOs$ and AOs from real tuples in the returned result is $O(|ANO| + |AO|)$. Both correctness and completeness verification take $O(1)$ complexity. Therefore, the complexity of verification is $O(|ANO|+|AO|)$. Our empirical study shows that $|ANO|$ and $|AO|$ are relatively small even for large datasets (Section 6), this enables the client to accomplish verification on resource-constrained devices (e.g., smart phones).

4.3 Recovery of True (p, d)-Outliers at Client Side

Since the returned (p_1, d)-outliers may contain some false positive tuples that are not (p, d)-outliers in D, the client will recover the real (p, d)-outliers in D from the returned (p_1, d)-outliers O_1 and (p_2, d)-outliers O_2. To achieve this, first, the client eliminates all AOs (if there is any) from O_2 and O_2 (how to distinguish real tuples, AOs, and $ANOs$ will be discussed in Section 4.4). Let the remaining (p_2, d)-outliers be O_2'. Second, the client examines each tuple in $O_1 - O_2$, and keeps those that are (p, d)-outliers in D. Let these tuples be O_{12}. Then $O_{12} \cup O_2'$ includes all true (p, d)-outliers in D. As will shown in Section 6, the tuples in $O_1 - O_2$ takes a negligible portion of D (less than 0.2%). Therefore, the complexity of outlierness examination of tuples in O_{12} should be $O(|D|)$.

4.4 Auxiliary Data for Verification

Auxiliary data sent to the server. Before the data owner (client) sends out her dataset, she signs each tuple with a cryptographic signature. The signature consists of two sub-signatures: Sig_a and Sig_t. Sig_a provides authenticity guarantee, so that any modification on the original tuples can be caught, while Sig_t is used to distinguish the true tuples from the artificial ones that will be inserted for verification of completeness and correctness. In particular, given a tuple $t(a_1, \ldots, a_n)$, $Sig_a = H(a_1 \oplus \ldots a_n)$, and

$$Sig_t = \begin{cases} H(Sig_a \oplus c_1), \text{If } t \text{ is a true tuple in } D; \\ H(Sig_a \oplus c_2), \text{If } t \text{ is an } AO; \\ H(Sig_a \oplus c_3), \text{If } t \text{ is an } ANO. \end{cases}$$

The client predefines three constants c_1, c_2, and c_3, for the true tuples, the AOs, and the ANOs respectively. The client stores the three constants and the

hash function H locally. We require that the hash function H is an efficiently computable collision-resistance hash function. It takes a variable-length input and returns a fixed length binary sequence. Furthermore, it should be difficult to reverse the hash value, i.e., given $H(x)$, it is computational infeasible to compute x. Therefore, the server cannot easily modify the signature.

After completes the computation of signatures, the client sends the dataset to the server. Each tuple in the dataset is associated with its two signatures Sig_a and Sig_t. When the server returns the outlier tuples to the client, he is required to return the two signatures of these tuples too.

Auxiliary data at the client side. The client maintains the hash function H, the number of AOs and $ANOs$, and the three constants c_1, c_2, and c_3 that are used to construct the signatures. It is straightforward that the space overhead of these auxiliary information is negligible. For each outlier tuple t returned by the server, the client computes its signature Sig_a, and the three signatures $Sig_t^1 = H(Sig_a \oplus c_1)$, $Sig_t^2 = H(Sig_a \oplus c_2)$, and $Sig_t^3 = H(Sig_a \oplus c_3)$, by using the three constants c_1, c_2, and c_3 that it has stored locally. Then by comparing the signatures Sig_t^1, Sig_t^2, Sig_t^3 against the Sig_t that is attached to t, the client can distinguish whether t is a true tuples, an AO or an ANO.

5 Discussion

5.1 Dealing with Malicious Server

We consider the following two types of misbehaviors by the malicious server:
(1) **Cheap computation**: the server picks a portion of the dataset $D' \subseteq D$ and return outliers of D' as the outliers of D;
(2) **Verification-aware cheating**: the server is aware of the details of the verification mechanism, and escapes from verification by returning incorrect mining result that fits what the verification mechanism expects.

Unfortunately, our AT-based approach cannot catch the two types of misbehaviors by the malicious server. First, our AT-based approach cannot catch cheap computation. If the dataset portion D' that the server picks satisfies that $D' \subseteq S$, where S is the sample that is used to construct AOs (Section 4.1), the outliers of D' should contain all AOs, i.e., the server can succeed to escape from the completeness verification. Since the probability that $D' \subseteq S$ is $prob = |S|/|D| = p$, where p is the parameter used for (p, d)-outlier mining that is close to 1 in practice [17], the server can escape the completeness verification with very high probability even by mining a small portion of dataset. On the other hand, if the portion D' contains t_{seed} (i.e., the seed tuple that is used to construct ANOs), none of the ANOs will be returned, and thus the server can escape the correctness verification. The probability that the server picks D' that contains t_{seed} is $prob = \frac{|D'|}{|D|}$. The server has to pick a large portion of D if it tries to escape the correctness verification with high probability.

Second, our AT-based approach cannot catch verification-aware cheating. When the server knows the verification mechanism, especially how $AOs/ANOs$ are constructed, it is able to identify $AOs/ANOs$. First, it can find all AOs easily by two passes of the outsourced dataset. In particular, due to the fact that the

p value of the (p, d)-outlierness setup is always very close to 1 in practice, the sample S used to construct AOs (Section 4.1) is close to the entire dataset D. Therefore, the constructed AOs will be the skyline points of D. Based on this, the malicious server can easily identify the tuples that have the maximum/minimum values of each attribute as AOs. This can be done by traversing the dataset twice (one pass to find maximum/minimum values of each attribute and another one to decide AOs). On the other hand, due to the fact that all $ANOs$ must be the *close* tuples (Definition 2) to a non-outlier tuple, the malicious server can identify all $ANOs$ by searching for non-outlier tuples that are *close* to at least one non-outlier. This requires the server to find all non-outliers, whose complexity is at least as high as the cost of mining all outliers. Though expensive, this enables the server to identify $ANOs$ and escape from the correctness verification.

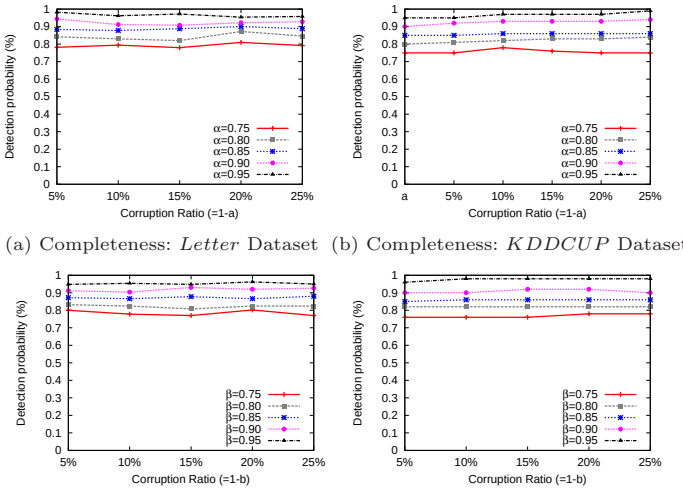
Intuitively, any verification based on transformation of the original dataset (e.g., inserting tuples) may not be able to catch the malicious server as it may launch verification-aware cheating. On the other hand, randomly picking tuples from the original dataset as samples and verifying the outlierness of the samples may resist the verification-based cheating. However, to return high correctness/completeness guarantee, it needs to find a large portion of real outliers, which may not be affordable by the client with limited computational power. We conjecture that to catch a malicious server, especially to catch the verification-aware cheating, the complexity of verification is as expensive as outlier mining.

5.2 From Probabilistic Approach to Deterministic Approach

It is possible that the client may require verification guarantee of 100% certainty. For this case, our AT-based approach cannot meet the requirement, even though it can provide a high probabilistic guarantee. Intuitively, let O^S be the outliers returned by the server, the client can verify the correctness of O^S with 100% certainty by checking whether each tuple in O^S is an outlier in D. The complexity is $O(kn)$, where k is the size of O^S, and n is the size of D. To verify completeness, the client checks whether there exist any tuple in $D - O^S$ that is an outlier. The complexity of completeness verification is $O((n-k)n)$. The total complexity of the deterministic approach is $O(n^2)$, which is as high as outlier mining itself. Although we can optimize the outlier detection procedure [4, 27], we have to pay for additional space overhead. We conjecture that the complexity of deterministic approach (in terms of both time and space) is as expensive as outlier mining itself.

6 Experimental Evaluation

We conducted an extensive set of experiments to evaluate both the assurance guarantee and performance of our approach. In particular, we measured: (1) the completeness and correctness guarantee; (2) the verification overhead at the client side, including a) the construction time for AOs and ANOs, b) the verification time to check AOs and ANOs, and c) the time of examining the outlierness of tuples to eliminate false positives; (3) the mining overhead at the server side.



(a) Completeness: *Letter* Dataset (b) Completeness: $KDDCUP$ Dataset

(c) Correctness: *Letter* dataset (d) Correctness: $KDDCUP$ dataset

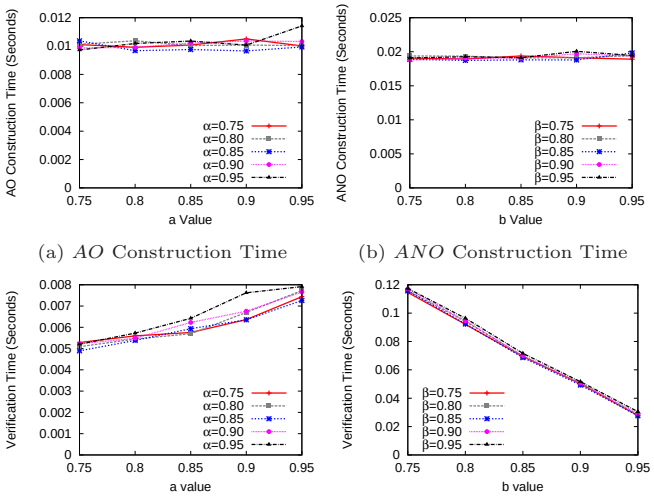
Fig. 3. Robustness of AT-based Approach

Setup. In our experiment, we used two datasets, the *Letter* dataset[1] from UCI MLC++ Library that contains $20k$ tuples, and the $KDDCUP$ dataset[2] that contains $100k$ tuples. The *Letter* dataset has 16 numerical (integer) attributes. The $KDDCUP$ dataset contains 41 (numerical or categorical) attributes. In our experiments, we used five numerical attributes, including `duration`, `dst_bytes`, `flag`, `count`, and `serror_rate` attributes, of the $KDDCUP$ dataset. For *Letter* dataset, we used $p = 0.8$ and $d = 15$ that return 1% of the tuples as outliers, while for $KDDCUP$ dataset, we used $p = 0.99$ and $d = 5000$ that return 2% of the tuples as outliers. All of our experiments are evaluated on a PC with a 2.4GHz Intel Core 2 Duo CPU and $4GB$ RAM running Windows 7. We implemented the algorithm in Java.

Robustness of the AT-based Approach. We simulate the incomplete result by the semi-honest server as removing 5%, 10%, 15%, 20%, and 25% outliers randomly (i.e., $a = 95\%$, 90%, 85%, 80%, and 75%), and the possible incorrect result as picking 5%, 10%, 15%, 20%, and 25% non-outliers randomly as outliers (i.e., $b = 95\%$, 90%, 85%, 80%, and 75%). Then, we measure the probability of catching these incomplete and incorrect results by our AT-based approach by the following: we repeat 500 trials on the *Letter* dataset and 100 trials on the $KDDCUP$ dataset. For each trial, first, we insert AOs and $ANOs$ that are constructed by using our AT-based approach. Second, we randomly pick a set of outliers to remove (for producing incomplete answer) and non-outliers to insert (for producing incorrect answer).

For completeness verification, we examine if all the AOs are returns. If at least one AO is missing, we count the trial as a *detected* one. Similarly, for correctness verification, we check if the result contains any ANO, and count the

[1] http://www.sgi.com/tech/mlc/db/letter.all
[2] http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html



(a) *AO* Construction Time (b) *ANO* Construction Time

(c) Verification Time: Completeness (d) Verification Time: Correctness

Fig. 4. *AO&ANO* Construction Time and Verification Time, *Letter* dataset

trial as detected if it does. After we finish all trials, we calculate the detection probability as the $pr_d = \frac{n_{det}}{n_{tr}}$, where n_{det} is the number of detected trials and n_{tr} is the total number of trials.

First, we measured the detection probability of incomplete answer. Figure 3 (a) shows the result on the *Letter* dataset. We observe that the detection probability is always higher than the required α value (i.e., the probability threshold). The same observation also holds for the $KDDCUP$ dataset (Figure 3 (b)). We also measured the detection probability of incorrect result. Figure 3 (c) & (d) show the detection probability for *Letter* dataset and $KDDCUP$ dataset respectively. It can be easily observed that the detection probability of incorrect result is always better than the required β value, i.e., the correctness probability threshold. This proves the robustness of our *AT*-based approach for both completeness and correctness verification.

Cost Analysis at Client Side. First, we measured the time performance of constructing *AOs* and *ANOs*. Figure 4 (a) shows the *AO* construction time on *Letter* dataset. We observe that the *AO* construction is extremely fast, which needs 0.012 seconds at most even when $\alpha = 0.95$ and $a = 0.95$. Furthermore, when α and a values increase, *AO* construction time increases too, but only slightly. Figure 4 (b) shows the *ANO* construction time on *Letter* dataset. It takes more time than *AO* construction since it needs to find the t_{seed} and verifies whether it is a non-outlier. Nevertheless, it is still fast; it only needs 0.022 seconds at most even when $\beta = 0.95$ and $b = 0.95$. Compared with the mining time (around 200 seconds for the *Letter* dataset), the construction time of *AOs* and *ANOs* is negligible. We also measured the construction time on $KDDCUP$ dataset. The construction time of the $KDDCUP$ dataset is longer than that of the *Letter* dataset, as the size of the $KDDCUP$ dataset is four times larger than that of the *Letter* dataset. However, the *AO*/*ANO* construction is still

fast; AO construction only needs 0.16 seconds at most, while ANO construction only needs 0.035 seconds at most. We omit the result due to limited space.

Second, we measured the verification time at the client side. Figure 4 (c) shows the time of completeness verification on *Letter* dataset. We observed that the time grows when α and a increase. This is straightforward as higher α and a require larger number of AOs. Figure 4 (d) shows the time of correctness verification on *Letter* dataset. Contrast to completeness verification, the time of correctness verification decreases with the growth of β and b. This is because with increasing b value, there are fewer real non-outliers inserted as incorrect answer (for simulation). Even though meanwhile larger b value requires more $ANOs$, the number of inserted real non-outliers decreases faster than that of $ANOs$. This leads to the decreasing number of outliers (including real non-outliers and $ANOs$) that the client receives, and consequently less verification time.

a, b	α, β 0.75	α, β 0.8	α, β 0.85	α, β 0.9	α, β 0.95
0.75	1	1	1	2	2
0.80	1	1	2	2	4
0.85	2	2	4	4	5
0.90	4	4	5	5	6
0.95	5	6	6	8	8

(a) *Letter* Dataset (20K tuples)

a, b	α, β 0.75	α, β 0.8	α, β 0.85	α, β 0.9	α, β 0.95
0.75	2	4	4	6	13
0.80	4	4	6	13	14
0.85	6	8	14	17	23
0.90	14	19	23	23	48
0.95	47	70	77	101	167

(b) $KDDCUP$ Dataset (100K tuples)

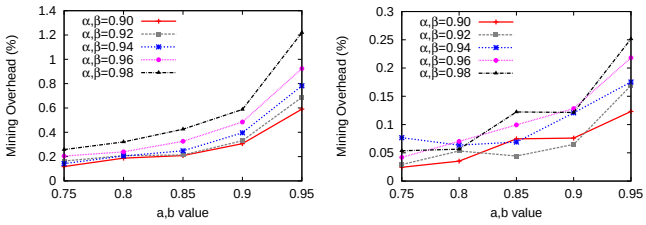
Table 1. Number of Tuples Whose Outlierness Are Examined during Post-Processing

Third, we measured the performance of outlier recovery at the client side. We first measured the number of tuples whose outlierness needs to be examined in the dataset. Table 1 (a) & (b) show the result of *Letter* dataset and $KDDCUP$ dataset respectively. Both tables show that the number of tuples whose outlierness needs to be examined only takes a small portion of the dataset. For example, at most 8 tuples in *Letter* dataset (0.04% of the dataset) and at most 167 tuples in $KDDCUP$ dataset (0.16% of dataset) that were examined. Furthermore, we noticed that though it is possible that the tuples that need to be examined can contain true and false positive outliers, in our experiments, all examined tuples are false positive outliers (real non-outliers).

Overhead at Server Side. We measured the mining overhead at the server side as $|T_{D^+} - T_D|/T_D$, where T_D and T_{D^+} are the time of mining outliers from the original database D and the dataset $D^+ = D \cup AO \cup ANO$. Figure 5 (a) and (b) show the result of the *Letter* dataset and $KDDCUP$ dataset respectively. We observed that adding AOs and $ANOs$ does not introduce much mining overhead. For example, it leads to at most additional 1.2% of the original mining time on the *Letter* dataset, and at most additional 0.25% of the original mining time on the $KDDCUP$ dataset. The overhead on $KDDCUP$ dataset is much smaller because we insert the same number of artificial tuples for the same α, β, a, b values into a larger dataset. This proves that our AT-based approach is more suitable for datasets of large size.

7 Conclusion

Outsourcing mining tasks to a third-party data-mining-as-a-service (DMAS) provider which is potentially untrusted arises serious concern on the integrity



(a) *Letter* Dataset (b) *KDDCUP* Dataset

Fig. 5. Mining overhead

of the mining results. In this paper, we focused on verifying the integrity of outlier mining result. We consider two types of server, namely the semi-honest server and the malicious server, that may return incorrect/incomplete outliers. We proposed *AUDIO*, a practical and efficient integrity auditing framework that can provide high correctness and completeness guarantees for the semi-honest server in the DMAS paradigm. We designed efficient approaches of constructing artificial tuples for verification purpose, and demonstrated the efficiency and effectiveness of our approach via an extensive set of experiments.

In the future, we will continue to explore the verification methods for the malicious server. We will investigate whether it is feasible to design efficient verification mechanisms if the server only knows partial details of the verification mechanism, e.g., the server knows there are artificial tuples in the dataset but does not know how these tuples are constructed. We will also examine how to design verification techniques to deal with datasets with updates.

References

1. Cloud Security Alliance. Security Guidance for Critical Areas of Focus in Cloud Computing. http://www.cloudsecurityalliance.org/guidance/csaguide.pdf. 2009.
2. Google Prediction APIs. http://code.google.com/apis/predict/.
3. C. C. Aggarwal and P. S. Yu. Outlier detection for high dimensional data. In *SIGMOD*, 2001.
4. F. Angiulli and F. Fassetti. Dolphin: An efficient algorithm for mining distance-based outliers in very large datasets. In *TKDD*, 2009.
5. S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy. Proof verification and the hardness of approximation problems. In *Journal of ACM*, 1998, Vol. 45.
6. V. Barnett and T. Lewis. *Outliers in Statistical Data*. John Wiley and Sons, 1994.
7. R. Chow, P. Golle, M. Jakobsson, E. Shi, J. Staddon, R. Masuoka, J. Molina. Controlling Data in the Cloud: Outsourcing Computation without Outsourcing Control. In *CCSW*, 2009.
8. A. Cuzzocrea and W. Wang. Approximate range-Sum query answering on data cubes with probabilistic guarantees. In *JIIS* Vol. 27, 2007.
9. A. Cuzzocrea, W. Wang, and U. Matrangolo. Answering approximate range aggregate queries on OLAP data cubes with Probabilistic Guarantees. In *DaWaK*, 2004.
10. J. Du, W. Wei, X. Gu, and T. Yu. Runtest: assuring integrity of dataflow processing in cloud computing infrastructures. In *ASIACCS*, 2010.
11. W. Du, J. Jia, M. Mangal, and M. Murugesan. Uncheatable grid computing. In *ICDCS*, 2004.

12. F. Giannotti, L. V. Lakshmanan, A. Monreale, D. Pedreschi, and H. Wang. Privacy-preserving mining of association rules from outsourced transaction databases. In *SPCC*, 2010.
13. S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof systems. In *SIAM Journal of Computing*, Vol. 18, 1989.
14. H. Hacigümüş, B. Iyer, C. Li, and S. Mehrotra. Executing SQL over encrypted data in the database-service-provider model. In *SIGMOD*, 2002.
15. E. S. Jeevanand and N. U. Nair. On determining the number of outliers in exponential and pareto samples. On determining the number of outliers in exponential and Pareto samples In *Stat. Pap.* Vol. 39, 1998.
16. S. Papadimitriou, H. Kitawaga, P. B. Gibbons, and C. Faloutsos, LOCI: Fast Outlier Detection Using the Local Correlation Integral. In *ICDE*, 2002.
17. E. M. Knorr and R. T. Ng. Algorithms for mining distance-based outliers in large datasets. In *VLDB*, 1998.
18. G. Kollios, D. Gunopulos, N. Koudas and S. Berchtold. An Efficient Approximation Scheme for Data Mining Tasks. In *ICDE*, 2001.
19. C. Kreibich and J. Crowcroft. Honeycomb: creating intrusion detection signatures using honeypots. *SIGCOMM Computer Communication Review*, Vol. 34, 2004.
20. F. Li, M. Hadjieleftheriou, G. Kollios, and L. Reyzin. Dynamic authenticated index structures for outsourced databases. In *SIGMOD*, 2006.
21. K. Liu, C. Giannella, H. Kargupta An attacker's view of distance preserving maps for privacy preserving data mining. In *PKDD*, 2006.
22. I. Molloy, N. Li, and T. Li. On the (in)security and (im)practicality of outsourcing precise association rule mining. In *ICDM*, 2009.
23. E. Mykletun, M. Narasimha, and G. Tsudik. Authentication and integrity in outsourced databases. In *Trans. Storage*, 2, May 2006.
24. H. V. Nguyen, V. Gopalkrishnan, and I. Assent, An unbiased distance-based outlier detection approach for high-dimensional data. *DASFAA*, 2011.
25. H. Pang, A. Jain, K. Ramamritham, and K.-L. Tan. Verifying completeness of relational query results in data publishing. In *SIGMOD*, 2005.
26. L. Qiu, Y. Li, and X. Wu. Protecting business intelligence and customer privacy while outsourcing data mining tasks. *Knowledge Information System*, Vol. 17(1), 2008.
27. S. Ramaswamy, R. Rastogi, K. Shim, and Aitrc. Efficient algorithms for mining outliers from large data sets. In *SIGMOD*, 2000.
28. R. Sion. Query execution assurance for outsourced databases. In *VLDB*, 2005.
29. C.-H. Tai, P. S. Yu, and M.-S. Chen. k-support anonymity based on pseudo taxonomy for outsourcing of frequent itemset mining. In *SIGKDD*, 2010.
30. W. K. Wong, D. W. Cheung, E. Hung, B. Kao, and N. Mamoulis. Security in outsourcing of association rule mining. In *VLDB*, 2007.
31. W. K. Wong, D. W. Cheung, B. Kao, E. Hung, and N. Mamoulis. An audit environment for outsourcing of frequent itemset mining. In *PVLDB*, Vol. 2, 2009.
32. M. Xie, H. Wang, J. Yin, and X. Meng. Integrity auditing of outsourced data. In *VLDB*, 2007.
33. M. L. Yiu, I. Assent, C. S. Jensen, and P. Kalnis. Outsourced Similarity Search on Metric Data Assets. In *TKDE*, Vol. 24, 2012.
34. M. L. Yiu, G. Ghinita, C. S. Jensen, and P. Kalnis. Enabling search services on outsourced private spatial data. In *VLDB J.*, Vol.19, 2010.