Result Integrity Verification of Outsourced Bayesian Network Structure Learning

Ruilin Liu[*] Hui (Wendy) Wang[†] Changhe Yuan[‡]

Abstract

There has been considerable recent interest in the data-mining-as-a-service paradigm: the client that lacks computational resources outsources his/her data and data mining needs to a third-party service provider. One of the security issues of this outsourcing paradigm is how the client can verify that the service provider indeed has returned correct data mining results. In this paper, we focus on the problem of result verification of outsourced Bayesian network (BN) structure learning. We consider the untrusted service provider that intends to return wrong BN structures. We develop three efficient probabilistic verification approaches to catch the incorrect BN structure with high probability and cheap overhead. Our experimental results demonstrate that our verification methods can capture wrong BN structure effectively and efficiently.

Keywords: Data-mining-as-a-service (*DMaS*); Cloud computing; result integrity; Bayesian network structure learning.

1 Introduction

The advancement of technologies in the Internet and distributed computing has enabled a data owner who possesses large volume of data but lacks computational resources to outsource his/her data mining needs to a third-party service provider (e.g., the Cloud). This outsourcing model, named the *data-mining-as-a-service* (*DMaS*) paradigm, provides a cost-effective solution for the data owner of limited budget (e.g., small and medium-sized enterprises). Big corporations like Amazon and Google are providing cloud-based data mining services in various forms. Amazon Web Services (AWS) provides computation capacity and data storages via Amazon Elastic Compute Cloud (EC2) [1]. Google provides APIs [2] for various data mining applications, including customer purchase prediction and spam detection, to name a few.

Though efficient, outsourcing data and computations to third-party service providers (e.g., the Cloud) raises a few security concerns. One of the concerns is that an untrusted service provider may return incorrect data mining result intentionally (e.g., by returning an intermediate result of cheaper computations). However, it is challenging for the data owner (client) of weak computational power to verify whether the result of intensive data mining computations (in many cases over large-scale datasets) that is returned by the computationally powerful service provider (server) is correct.

In general, outsourced computations can be verified by using techniques such as interactive proofs [4] and probabilistically checkable proofs [3]. Informally, a client can check an answer's correctness with a suitably encoded proof. However, it is not clear how to apply these verification techniques to data mining problems, due to the complexity of these techniques and the difficulty of using general-purpose cryptographic techniques to solve specific data mining problems. Some recent research work [18, 11, 12] designed result integrity assurance techniques for various data mining problems, including association rule mining [18], outlier mining [11], and k-means clustering [12]. These work show that the design of verification methods can vary significantly for different data mining problems and applications.

In this paper, we focus on a different data mining problem related to Bayesian network structure learning. A Bayesian network (BN) represents the uncertain relations between a set of random variables with a compact and intuitive graphical structure. We consider the untrusted service provider (server) that may intentionally return wrong BN results to the client. We define two types of cheating behaviors on BN learning, namely the *cheating on computations* and the *cheating on learning results*. By cheating on computations, the server executes the BN construction computation only on a subset of variables, and assigns random structure to the others. By cheating on learning results, the server constructs the correct BN structure but intentionally changes the structure of a subset of variables.

We develop three probabilistic verification methods, namely the *random-check* method (*RC*), the *insertion-*

[*]Stevens Institute of Technology, NJ, USA. rliu3@stevens.edu.
[†]Contact author. Stevens Institute of Technology, NJ, USA. Hui.Wang@stevens.edu.
[‡]Queens College, City University of New York, Flushing, NY. changhe.yuan@qc.cuny.edu.

based (IB) method, and the *hybrid* (HY) method. By the RC method, the client randomly picks a parent candidate set for each variable, and verifies whether the picked candidate set is better than the server's result. By the IB method, the client inserts a set of artificial variables into the dataset, and verifies whether the structure of those artificial variables in the returned BN is correct. We ensure that the BN structure of those artificial variables follows certain patterns, so that the client can easily verify the correctness of their structure without executing any learning. We also ensure that the artificial variable will not change the original BN structure significantly, and the original BN structure can be easily recovered by removing all artificial variables. The verification complexity of IB method is independent of the original database size. It is linear in the number of artificial variables, which is decided by the correctness probabilistic guarantee. This is extremely useful for verification of large datasets. Our HY method integrates both RC and IB methods to reduce the number of required artificial variables while providing high probabilistic guarantee. Our experimental results show that the verification time of the three verification approaches is no more than 14% of BN structure learning time. To our best knowledge, we are the first to investigate the problem of verifying the correctness of outsourced BN structure learning.

2 Preliminaries

In this section, we introduce the preliminaries.

BN Structure Learning Algorithm Many BN structure learning algorithms rely on scoring functions in measuring the quality of a Bayesian network structure conditioned on given data. The best network structure is the one that maximizes the scoring functions. In this paper, we focus on a classic scoring function named $K2$ scoring scheme [7]. Formally, let X be a set of n discrete variables, where each variable x_i in X has r possible value assignments. Let D be a database of m instances, where each instance contains a value assignment for each variable in X. Let B denote a BN structure constructed from the variables in X. Assume there exists an ordering on all n variables such that if x_i precedes x_j in the ordering, then it is not allowed that there is an arc from x_j to x_i in B. We use a function $pred(x_i)$ to denote the set of nodes that precede x_i in the node ordering. Each variable x_i in B has a set of parents, represented with a list of variables π_i, which is picked from $pred(x_i)$. Let w_{ij} denote the j^{th} unique instantiation of π_i. Suppose there are q_i such unique instantiations of π_i. Define N_{ijk} to be the number of instances in D in which variable x_i is instantiated with its k^{th} value, and π_i is instantiated as w_{ij}. Let $N_{ij} = \sum_{k=1}^{r} N_{ijk}$. K2 score function is defined as [7]:

$$(2.1)\quad K2(x_i, \pi_i) \quad = \quad \prod_{j=1}^{q_i} \frac{(r-1)!}{(N_{ij}+r-1)!} \prod_{k=1}^{r} N_{ijk}!$$

The function measures the $K2$ score of the candidate π_i being the parent set of the node x_i in B, where q_i is the number of possible instantiations of the parents of x_i. The candidate π_i of the highest K2 score $K2(x_i, \pi_i)$ will be picked as the parent set of the node x_i. We call those candidates of the highest K2 score the *candidates of the optimal parent set* of x_i. As there may exist several candidates of optimal parent for each variable, the learned structure may not be unique. When there is a tie, we usually prefer a smaller parent set. If two parent sets have the same score and size, we prefer parents that appear early in the variable ordering. It is well known that finding the best BN structure is NP-hard [6]. We note that what we used is a variation of the K2 algorithm that finds an optimal structure for a given ordering of variables, in contrast to the original version which uses a non-optimal greedy method to select a parent set for each variable. For a dataset of m instances, n variables, u ($u \leq n$) maximum number of parents that any variable can have, and r states of each variable, the complexity of our $K2$ algorithm is $O(umT(n, u))$, where $T(n, u) = \sum_{j=0}^{u} \binom{n}{j} r^j$. In this paper, we only consider binary variables, i.e., $r = 2$. We note that variables in the dataset correspond to the nodes in BN. The terms of variables and nodes are exchangeable in our paper. We will discuss how to extend to other BN learning algorithms in Section 6.

Outsourcing Scenario We consider the *infrastructure-as-a-service* ($IaaS$) paradigm where the client outsources the dataset D and the code for BN structure learning to an $IaaS$ service provider (server). The client also sends in the configuration of structure learning, including the ordering of variables in D and the upper bound u of number of parents that any variable can have. A (faithful) server executes the received code on the outsourced D according to the configurations. A typical $IaaS$ example is Amazon EC2 Web Service [1].

Cheating Behaviors of Dishonest Server We consider two types of cheating behaviors:

(1) *Cheating on computations*: An economically-motivated server may intend to reduce the computational cost by executing the learning algorithm only on a subset of variables. For the remaining variables, the server picks variables arbitrarily (in a uniform fashion) as their optimal parents in the returned BN.

(2) *Cheating on learning results*: A compromised service provider may try to steal the correct mining result and sell it later to the competitors of the client for profit.

Approach	Verification at Client Side			Mining at Server Side	Correctness Guarantee
	Verification Preparation	Verification	Post-processing		
RC	N/A	$O(urmn)$	N/A	$O(umT(n,u))$	No guarantee
IB	$O(mn)$	$O(w)$	$O(\|B^S\|)$	$O(umT(n+w,u))$	(α,β)-correctness

Table 1: *RC* vs *IB* ($T(n,u)=\sum_{j=0}^{u}\binom{n}{j}r^j$; m: # of instances of D; n: # of real variables of D; w: # of artificial variables; u: max. # of parents in BN that any variable can have; r: # of possible instances of each variable; B^S: BN returned by the server.)

It also may modify the mining result intentionally and return the wrong result to the client. For such cheating, the server may deliberately pick a subset of variables whose BN structure information is believed to be important. Then the server executes learning on these variables faithfully but picks the candidates of the (wrong) k^{th} ($k>1$) highest $K2$ score as the optimal parent in the returned BN.

We assume that the server does not possess any prior knowledge of the original dataset or the details of the verification schemes. This assumption holds in many real-world *DMaS* applications.

Verification Goal Given n variables $X=\{x_1,\dots,x_n\}$, let B be the correct BN of X, and B^S be the BN returned by the server. We use *precision* to measure the correctness of B^S. The *precision* of B^S is defined as $R=\frac{n'}{n}$, where n' is the number of variables in B^S whose parent has the optimal $K2$ score. We aim at designing efficient verification techniques that can provide *(α,β)-correctness*. Formally, given a BN structure B^S, and the user-specified thresholds $\alpha,\beta\in[0,1]$, let P be the probability to catch B^S whose precision $R\le\beta$. We say a verification method M can verify *(α,β)-correctness* if $P\ge\alpha$.

Solution Overview. We develop three probabilistic verification methods, namely the *random-check method (RC)*, the *insertion-based (IB) method*, and the *hybrid (HY)* method. The key idea of the *RC* method is that the client randomly picks a candidate parent set for each variable, and verifies whether the picked candidate set is better than the server's result. By the *IB* method, the client inserts a set of artificial variables into the dataset, and verifies whether the BN structure of those artificial variables is correct. The artificial variables and their instances are constructed in a deliberate manner so that the client can easily verify the structure of any artificial variable without executing any learning. Furthermore, the original BN structure can be easily recovered by removing all artificial variables. The *HY* method integrates both *RC* and *IB* to reduce the number of required artificial variables while providing high probabilistic guarantee. Comparing these three approaches, first, by the *RC* approach, the client does not need to prepare for verification before outsourcing the dataset, while by the *IB* and *HY* approaches, it has to construct artificial variables to serve the verification purpose. Second, *RC* does not introduce any additional mining overhead at the server side, while the *IB* and *HY* methods may introduce extensive mining overhead at the server side due to the insertion of artificial variables. However, *RC* has two disadvantages. First, for verification at the client side, *RC* needs to compute $K2$ score of a number of parent candidates (the same as *HY*), while *IB* does not need to compute any $K2$ score. Second, *RC* may not be able to provide high probabilistic guarantee, while *IB* and *HY* can guarantee (α,β)-correctness. Table 1 summarizes the complexity of *RC* and *IB* approaches.

3 Random-check (*RC*) Approach

The basic idea of the RC method is that the client randomly picks a set of variables to verify their structure. In particular, for each variable $x\in X$, let π^S be the parent of x in B^S. Then the client randomly picks a parent candidate π from *pred*(x), and compares $K2(x,\pi)$ with $K2(x,\pi^S)$. If $K2(x,\pi)>K2(x,\pi^S)$, the client catches the server's incorrect result with 100% certainty. The complexity of the random-check verification approach is $O(mnur)$, given n variables, m tuples, r instances of each variable, and u the maximum number of parents of any variable.

Assume the server changes the structure of a set of variables $X'\subseteq X$. The probability P_R that the server is caught by the random-check verification is

$$P_R = 1-\prod_{x_i\in X'}(1-p_i), \tag{3.2}$$

where p_i is the probability that the server is caught by the verification of the structure of x_i. Next, we discuss how to measure p_i for the cheating on computations and on the learning results respectively.

Cheating on computations. Assume that by random pick, the server picks the parent of the k^{th} ($k>1$) highest $K2$ score. The probability of catching the server's cheating on computations is:

$$p_i = \frac{2\sum_{j=1}^{k-1}o_j}{m_i-1}, \tag{3.3}$$

where o_j is the number of parent candidates of the j^{th} $K2$ score, and m_i is the total number of parent candidates of x_i. In general,

$$m_i = \sum_{j=0}^{u} \binom{i-1}{j}, \tag{3.4}$$

where i is the order of variable x_i in X. More details of how p_i and m_i are computed can be found in the supplementary file [13].

Cheating on learning results. As the server uses the parent candidate of the k^{th} $(k > 1)$ highest $K2$ score, the probability of catching the server's cheating is:

$$p_i = \frac{\sum_{j=1}^{k-1} o_j}{m_i - 1}, \tag{3.5}$$

where o_j is the number of parent candidates of the j^{th} $(j < k)$ $K2$ score, and m_i is the total number of parent candidates of x_i (Eqn. 3.4). Since the probability of catching the cheating on results (Eqn. 3.5) is smaller than that of catching the cheating on computations (Eqn. 3.3), it is more difficult to catch the cheating on the learning results than on computations.

In general, it is difficult to determine whether the server cheated on computations or results. Therefore, we compute the lower bound of P_R. We have:

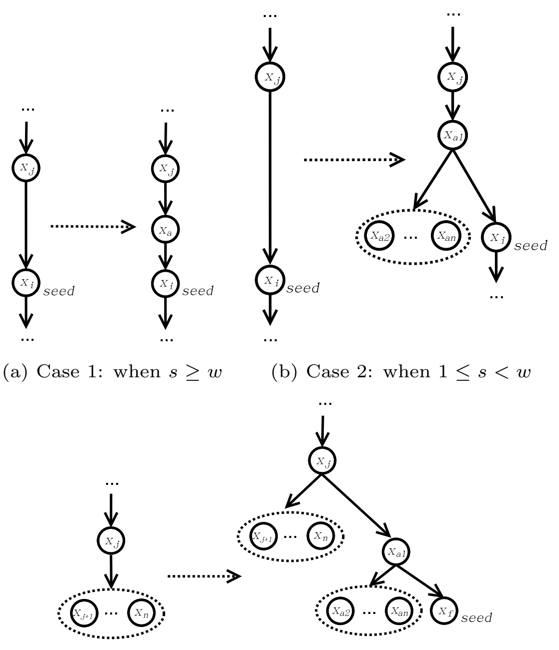
$$Min_{P_R} = 1 - \prod_{i=[\beta n]}^{n} \left(1 - \frac{1}{\sum_{j=0}^{u} \binom{i-1}{j} - 1}\right). \tag{3.6}$$

More details of how to compute Min_{P_R} are in the supplementary file [13]. When $Min_{P_R} \geq \alpha$, the random-check verification approach can provide (α, β)-correctness guarantee to catch both cheating on computations and results. However, when $Min_{P_R} < \alpha$, the random-check verification cannot guarantee (α, β)-correctness. In the next section, we present our insertion-based verification approach that can always guarantee (α, β)-correctness with cheap verification overhead.

4 Insertion-based (IB) Approach

The key idea of the *IB* approach is that the client inserts a set of artificial variables into the original dataset, being aware of where these artificial variables will locate in the returned BN. The correctness of the returned BN is verified by checking whether the structure of the artificial variables is correct.

4.1 Mirror Nodes and Their Properties In this section, we define *mirror nodes*, which will be used for the construction of artificial variables. First, we define the *identical nodes* and *opposite* nodes. For any binary node $x_i \in X$, a node $x_j \in X$ is an *identical node* (*opposite node*, resp.) of x_i if for all $k \in [1, m]$, $x_i[k] = x_j[k]$ ($x_i[k] = 1 - x_j[k]$, resp.). Then for any binary node $x_i \in X$, a node $x_j \in X$ is a *mirror node* of x_i if x_j is either an identical node or an opposite node of x_i. We say node x_i is the *copyee* node of x_j. Mirror



(a) Case 1: when $s \geq w$ (b) Case 2: when $1 \leq s < w$

(c) Case 3: when $s = 0$

Figure 1: Illustration of Three Cases of Artificial Variables

nodes have an important property. The next lemma will show that the mirror nodes are *identical* to their copyee nodes in terms of $K2$ score.

LEMMA 4.1. *Given n ordered nodes $X = \{x_1, \cdots, x_n\}$, for any node $x_i \in X$, let x_c be one of its mirror nodes. Then for any parent instantiation π_i of x_i, $K2(x_i, \pi_i) = K2(x_c, \pi_i)$.*

The proof of Lemma 4.1 can be found in the supplementary file [13].

4.2 Verification Preparation at Client Side The key idea of the insertion-based verification approach is that the client inserts a number of artificial variables into the dataset. We require that the artificial variables must satisfy the following two requirements: (1) *known-structure* requirement: the client knows the structure of artificial variables in the constructed BN without execution of the learning algorithm; and (2) *structure-preservation* requirement: inserting artificial variables will only change the structure of a small portion of real variables. Furthermore, the changed structure should be able to be easily restored.

Next, we first describe how to construct the artificial variables. Then we will discuss how the artificial variables satisfy the two aforementioned requirements. We use A_{i_0} (A_{i_1}, resp.) to denote the number of instances that $x_i = 0$ ($x_i = 1$, resp.). We define

$$sc(A_i) = (A_{i_0} + 1) \times (A_{i_1} + 1).$$

We say a variable x_i is a *seed* if for all $x_j \in pred(x_i)$, $sc(A_j) > sc(A_i)$. Given a dataset D with n variables, our goal is to construct w artificial variables that satisfy (α, β)-correctness, where w is decided by α and β (more details of how to value w is in Eqn. 4.8 of Section 4.3). To achieve this goal, first, the client finds all seed variables by scanning D once. Let s be the number of found seed variables. There are three possibilities: (1) $s \geq w$, (2) $1 \leq s < w$, and (3) $s = 0$. Next, we discuss how to construct the artificial variables for each case. We use X to denote the real variables of D.

Case 1: when $s \geq w$. For this case, the client randomly picks w variables from s seeds. For each picked seed variable x_i, the client generates a mirror variable x_a of x_i in a probabilistic manner: it constructs an identical node x_a of x_i of probability θ, and an opposite node x_a of x_i of probability $1 - \theta$, where θ is a user-defined threshold. After constructing x_a, the client inserts x_a into X right before x_i. The following theorem shows that x_a must be the optimal parent of x_i in the constructed BN.

THEOREM 4.1. *Given n ordered nodes $X = \{x_1, \cdots, x_n\}$, for each node $x_i \in X$, if for each $x_j \in pred(x_i)$, $sc(A_j) > sc(A_i)$, then any mirror node x_a of x_i that is inserted into X before x_i must be the optimal parent of x_i.*

The proof of Theorem 4.1 is in the supplementary file [13]. Figure 1 (a) illustrates the BN after inserting artificial variables for Case 1.

Case 2: when $1 \leq s < w$. When there does not exist sufficient number of seed variables, the client constructs multiple copies of mirror variables, and inserts all the copies into X right before the seed variables. All the copies are constructed in the same probabilistic manner as Case 1 and are constructed independently. The structure of the constructed artificial variables will follow Theorem 4.2.

THEOREM 4.2. *Given n variables X with the ordering of $x_1 < \cdots < x_n$, let $x_i \in X$ be a seed variable and $x_j \in X$ be the optimal parent of x_i in BN constructed from X. Let $F = \{x_{a_1}, \ldots, x_{a_k}\}$ be a set of artificial variables that are copies of mirror nodes of x_i, with the ordering $x_{a_1} < \cdots < x_{a_k}$. let $X' = X \cup F$, with F inserted into X right before x_i, and B' be the BN constructed from X'. Then B' must have the following properties: (1) x_{a_1} must be the optimal parent of x_i; (2) the parent of x_{a_1} must be x_j; and (3) for any artificial variable $x_{a_i} \neq x_{a_1}$, its optimal parent must be x_{a_1}.*

The proof of Theorem 4.2 is in the supplementary file [13]. Figure 1 (b) illustrates the BN change for Case 2.

Case 3: when $s = 0$. In this case, the client constructs an artificial seed variable x_f such that for each real variable $x_i \in X$, $sc(A_i) > sc(A_f)$. Then the client constructs w copies of mirror variables of x_f, and inserts x_f and all copies of mirror variables of x_f into X after all real variables. We have:

THEOREM 4.3. *Given n variables X in the ordering of $x_1 < \cdots < x_n$, let x_f be an artificial seed variable that is constructed as above, and $F = \{x_{a_1}, \ldots, x_{a_k}\}$ be a set of artificial variables that are copies of mirror nodes of x_f, with the ordering $x_{a_1} < \cdots < x_{a_k}$. Let $X' = X \cup F \cup \{x_f\}$ be the set of variables by inserting F and x_f into X after all real variables, with x_f after all variables of F. Let B' be the BN constructed from X'. Then B' must have the following properties: (1) for any artificial variable $x_{a_i} \neq x_{a_1}$ (including x_f), its optimal parent must be x_{a_1}; and (2) no artificial variable in $F \cup \{x_f\}$ can be the optimal parent of any real variable in X.*

The proof of Theorem 4.3 is in the supplementary file [13]. The BN constructed after inserting artificial variables for this case is illustrated in Figure 1 (c).

The complexity of the preparation procedure is dominated by finding the seed variables via one scan of the dataset. Thus the complexity of the preparation procedure (including all three cases) is $O(mn)$, where m is the number of instances of D, and n is the number of variables of D.

Now we are ready to discuss how the constructed artificial variables satisfy the known-structure and the structure-preservation requirements. The known-structure requirement is naturally followed by Theorem 4.1, 4.2, and 4.3, as the client expects that the artificial variables are optimal parents of the seed variables. Regarding the structure-preservation requirement, by following Theorem 4.1, 4.2, and 4.3, it can be easily shown that: (1) for each real variable in B that is not used as a seed, its parent in B^S is exactly the same as its parent in B, and (2) for each real variable x_i in B that is used as a seed, its parent x_j in B must be its ancestor in B^S, with only one artificial variable between x_i and x_j.

4.3 Verification After inserting artificial variables, the client outsources the updated dataset to the server, while maintains the artificial variables and their expected structure locally. We assume that the server cannot distinguish the artificial variables from real ones, so that it has equal probability to cheat on the structure of both types of variables. We will discuss how to deal with the server that can tell the artificial variables apart from real ones in Section 6. The correctness verification of the returned BN is performed by checking whether the structure of all artificial variables is correct. In particular, for each artificial variable x_a, the client finds the node x_t next to x_a in the ordered set X. We note that x_t can be either real or artificial. For both cases, the client checks whether the parent of x_t is x_a itself or a mirror node of x_a. If neither case is true, the client concludes that the returned BN result is incorrect with

100% certainty. Otherwise, the client concludes that x_a passes the verification with a probabilistic guarantee. In particular, let p be the probability that the parent of a variable is correct. Given the precision threshold β, the probability p that the parent of any variable is correct is $p = \beta$. Therefore, the probability P_I of catching a server that returns a wrong BN but in which all w artificial variables are of correct structure is

$$P_I = 1 - \beta^w. \tag{4.7}$$

To satisfy (α, β)-correctness requirement, the number of artificial variables w should satisfy

$$w = \lceil log_\beta(1-\alpha) \rceil. \tag{4.8}$$

Since the verification only involves the structure checking of the artificial variables, the complexity of the verification procedure is $O(w)$, where w is the number of artificial variables. A nice property is that w is independent of the number of real variables in the original dataset. Further analysis of Equation 4.8 shows that it does not need large number of artificial variables to catch a server that changes the structure of a small fraction of variables with high correctness probability. For instance, when $\alpha = 0.9$ and $\beta = 0.9$ (i.e., 10% of variables are of wrong structure), it only needs $w = 22$ artificial variables. Therefore, the insertion-based approach is especially suitable for datasets of large number of variables. However, a disadvantage of the insertion-based approach is that w can be extremely large for $\alpha, \beta \geq 0.95$. For example, when $\alpha = 0.99$ and $\beta = 0.99$, $w = 458$. We will show how to fix this shortcoming in Section 5.

4.4 Recovery of Original BN Structure Given the returned BN B^S, for each artificial variable $x_a \in B^S$, if its parent x_j is a real variable, the client finds the child x_i of x_a in B^S, and assigns x_j as the parent of x_i by adding an edge from x_j to x_i. At the end, the client removes all artificial variables and their associated edges. Since the post-processing requires one scan of B^S, its complexity is $O(|B^S|)$. We have the following theorem to show that after running our post-processing procedure on B^S, the output is the same as the original BN B. We use $Recovery(B^S)$ to denote the result of running our post-processing procedure on B^S.

THEOREM 4.4. *Given a dataset D, let D' be the dataset after inserting artificial variables by our preparation approach. Let B and B^S be the BN constructed from D and D' respectively. Then $Recovery(B^S) = B$.*

The proof of Theorem 4.4 is in the supplementary file [13].

5 Hybrid (HY) Verification Approach

As discussed so far, the random-check verification approach may not be able to satisfy (α, β)-correctness requirement, while the insertion-based approach, which always can achieve (α, β)-correctness, may lead to a large number of artificial variables. Therefore, we propose a *hybrid* approach that integrates the random-check and the insertion-based approaches to reduce the number of artificial variables while guarantees (α, β)-correctness. In general, the client first checks whether the minimal catch probability Min_{P_R} (Equation 3.6) by the random-check verification approach satisfies that $Min_{P_R} \geq \alpha$. If it does, the client runs the random-check verification approach alone. Otherwise, it runs the insertion-based verification then the random-check verification (i.e., calculating $K2$ scores of a subset of nodes) for further verification. Let P_R and P_I be the probability of catching the server by the random-check verification and the insertion-based verification respectively. Then the probability P of catching the untrusted (either lazy or the malicious) server by running the hybrid approach is $P = 1 - (1 - P_R)(1 - P_I)$, where P_R and P_I are calculated by Eqn. 3.2 and 4.7 respectively. It can be easily inferred that:

$$P \geq 1-(1-Min_{P_R})\beta^w = 1-\prod_{i=[\beta n]}^{n}(1-\frac{1}{\sum_{j=0}^{u}\binom{i-1}{j}-1})\beta^w.$$

To ensure that $P \geq \alpha$ (i.e., it satisfies (α, β)-correctness), we have:

$$w = log_\beta(\frac{1-\alpha}{\prod_{i=[\beta n]}^{n}(1-\frac{1}{\sum_{j=0}^{u}\binom{i-1}{j}-1})}).$$

Compared with the number of artificial variables by the insertion-based approach (Eqn. 4.8), the number of artificial variables by the hybrid approach can be much smaller than that of the insertion-based approach for small u and n values. This shows that the hybrid approach is suitable for the datasets with small number of attributes as well as for BNs whose nodes have small number of parents.

6 Discussion

Catch Server with More Cheating Power If the server is aware of the design details of the verification procedure, it may try to escape the verification by identifying the artificial variables. Indeed the artificial variables can be easily identified since their instances are either identical of or opposite to the instances of some real variables. To fix this, we propose to construct the artificial variables as *partial copies* of the real variables. Informally, only a part of the instances of artificial variables are copied from the instances of real variables,

dataset	# of instances	# of variables
Adult	30162	14
Chipseq	72228	30

Table 2: Summary of Datasets

while the remaining instances are constructed randomly. Introducing randomness will increase the difficulty that the attacker can identify the artificial variables. More details of how to construct partial copies can be found in the supplementary file [13].

Extend to Other BN Structure Learning Algorithms Our random-check approach can be easily adapted to any BN structure learning algorithm by following the strategy of randomly picking a candidate parent set and validating its optimality. Our insertion-based verification method can be adapted to those BN structure learning algorithms that are usually formalized to be a maximization problem based on scores that are measured for all parent candidates by using specific scoring functions. The proposed insertion-based approach in this paper is customized for the $K2$ score function. Changing the scoring function may lead to the redesign of construction procedure of artificial variables. However, given the similarity of the Bayesian scoring functions, the proposed insertion-based approach can be easily adapted to other Bayesian scoring functions, e.g., $BDeu$ [5] and MDL [17]. The key idea is the same: the client constructs artificial variables that are guaranteed to be optimal parents for a set of real variables regarding their scores, and attest the structure of those artificial variables in the returned BN for verification.

7 Experiments

In this section, we present our experimental results of our random-check (RC), insertion-based (IB), and hybrid (HY) verification approaches.

Experiment Setup. We implemented our three approaches in Java. We use the K2 implementation from the open source Bayesian network learning software *URLearning* developed by Yuan's group[1]. We used a PC (Intel Core i5 CPU@2.4GHz, 8GB RAM) for our experiments. All experiments about time performance is measured as the average of 500 trials.

Datasets. We use two real datasets: *Chipseq* and the *Adult*.*Chipseq* dataset is constructed from the genome-wide distribution dataset[2] The *Adult* dataset is downloaded from UCI repository[3]. Table 2 summarizes the two datasets. More details of these two datasets are in the supplementary file [13].

[1]URLearning ("You Are Learning") is downloadable at http://url.cs.qc.cuny.edu/software/URLearning.html

[2]The genome-wide distribution dataset: `http://dir.nhlbi.nih.gov/papers/lmi/epigenomes/hgtcell.aspx`.

[3]`http://archive.ics.uci.edu/ml/dataset`.

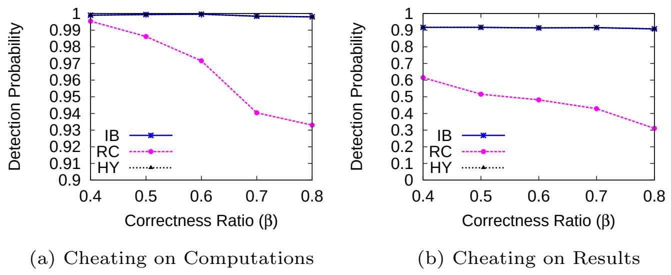
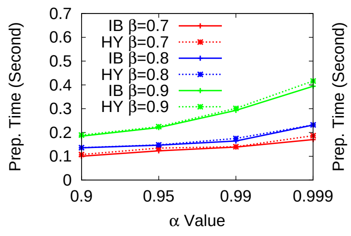


Figure 2: Detection Prob. ($\alpha = 0.9$, $u = 3$, *Adult* dataset)

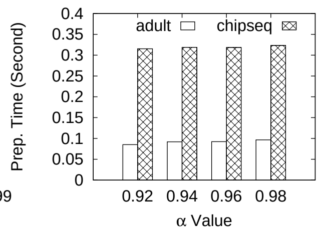
Robustness of Our Verification Approaches. We evaluate the robustness of our three verification approaches by simulating two types of cheating behaviors on BN structure and measuring how likely the incorrect BN structure can be caught by our verification methods. In particular, we pick $1 - \beta$ percent of variables randomly. Then we change the structure of these picked variables by simulating the cheating on computations and on results respectively. After that, we run our three verification approaches and record whether the incorrect BN structure can be caught. We repeat this experiment 5000 times and record the percentage of trials (as *detection probability*) that the server is caught.

We measure the detection probability of our three approaches to catch cheating on the learning results, as it is more difficult to be caught than the cheating on the computations. We simulate the cheating on results by changing the parent of $(1 - \beta)$ percent of variables to a parent candidate of the *second highest* $K2$ *score* with various β values. We pick this case as it is the most difficult cheating scenario that can be caught. The measurement result is illustrated in Figure 2. We observe that first, when β increases, IB and HY yield a stable detection probability but RC returns a decreasing detection probability. This shows that it is more difficult for RC to catch smaller errors compared with IB and HY. Second, both IB and HY approaches always yield better detection probability than the required threshold α, which proves the robustness of these two approaches (i.e., both of them always satisfy (α, β)-correctness). However, in some cases, IB approach cannot deliver the detection probability higher than the required α. This is consistent with our expectation that RC is not robust as IB and HY in terms of the correctness guarantee. The observation of detection probability on *Chipseq* dataset is similar; we omit it due to the limited space.

Verification Preparation. We measure the time overhead of verification preparation by IB and HY approaches on both *Adult* and *Chipseq* datasets. Note that RC does not need any verification preparation at the client side. Figure 3 (a) shows the preparation time of both approaches for various α and β values on *Adult* dataset. In this setting, both HY and IB approaches

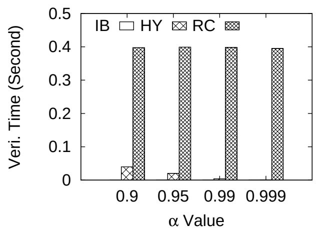


(a) IB vs. HY ($\beta = 0.8$, *Adult*)

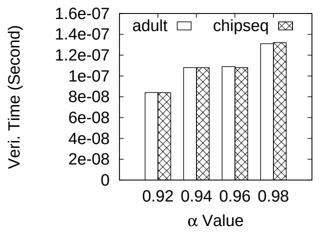


(b) IB approach ($\beta = 0.5$)

Figure 3: Preparation Time ($u = 3$)



(a) Comparison of three approaches ($\beta = 0.5$, *Adult* dataset)



(b) IB approach ($\beta = 0.5$, *Adult* vs. *Chipseq*)

Figure 4: Verification Time ($u = 3$)

construct the same number of artificial variables. First, we observe that the preparation time of both approaches is extremely short (at most 0.42 seconds). Second, HY needs more time for verification preparation compared with IB, but the time difference is negligible as it is the time to calculate Min_{P_R}, which can be computed very fast. Third, for both approaches, the preparation time increases when α and β grow, as larger α and β values require more artificial variables to be constructed.

To measure the scalability of the IB approach, we also compare its verification preparation time for both *Adult* and *Chipseq* datasets. The result (Figure 3 (b)) shows that the number of artificial variables to be constructed is the same for both *Adult* and *Chipseq* datasets. This is not surprising as the number of artificial variables is decided by α and β only. Given the same number of artificial variables to be constructed, the main factor of the preparation time is to find seed variables, whose time cost is linearly in the data size and the number of attributes. This explains the difference of the preparation time on *Adult* and *Chipseq* datasets.

Verification Time. We measure the verification time at the client side for both *Adult* and *Chipseq* datasets. First, we measure the impact of various α values to the time performance. Figure 4 (a) shows the result of the *Adult* dataset. We observe that the verification of the three approaches is very fast (no more than 0.4 seconds). In all cases, IB always has the cheapest verification overhead (around $3 * 10^{-6}$ seconds), since it only deals with the structure of the artificial variables. RC is always the slowest compared with the other two approaches, as it has to compute $K2$ scores. The performance of the HY approach is much faster than the RC approach, but a bit slower than the IB approach. Furthermore, its performance speeds up for larger α values. This is because that the growth of α value leads to more artificial nodes, and thus higher probability of catching the cheating answer by using the HY approach. We have the similar observation on *Chipseq* dataset. We omit the result due to the page limit.

We also measure the impact of data size to verification performance of IB approach by comparing its verification time on both *Adult* and *Chipseq* datasets. The result (Figure 4 (b)) shows that the verification time of IB is very similar for the two datasets. This is because the number of artificial variables, which is only decided by α and β, is the same for both datasets. Therefore, the verification time does not vary with data size. This is especially useful for outsourcing of large datasets.

Post-Processing Time. We measure the post-processing time of both IB and HY approaches for different α, β, and u values. We observe that in all of our testings, the post-processing time is extremely short (at most $2.04 * 10^{-5}$ seconds on *Adult* dataset and at most $6.95 * 10^{-5}$ seconds on *Chipseq* dataset). We omit the result due to limited space.

Mining Overhead of IB Approach. We measure the mining overhead at the server side due to the insertion of artificial variables. We define the mining overhead as $MO = \frac{T_a - T_o}{T_o}$, where T_o and T_a are the mining time of the original dataset and the dataset with artificial variables respectively. We observe that the mining overhead of *Adult* dataset is between 12% and 24% (with α varied from 0.9 to 0.999, and fixed β=0.8). However, the mining overhead is much higher for the *Chipseq* dataset; it is between 42% and 175.31%. The high mining overhead of the *Chipseq* dataset comes from its large number of attributes; adding more artificial variables make the search space exponentially larger. Higher α and β values always result in larger mining overhead, since higher α and β requires more artificial variables. That is the price for higher integrity guarantee.

Verification vs. Mining Locally. We compare the verification time of the three approaches with that of executing $K2$ algorithm locally by the client. We use *Adult* dataset with various α values, and show the result in Figure 4 (a). We define the ratio $r = \frac{T_v}{T_o}$, where T_v is the verification time and T_o is the time of executing $K2$ algorithm. The result shows that our three approaches are fast compared with mining locally (RC & HY: at most 14.08% of mining time; IB: at most $1.07{*}10^{-6}$% of mining time). This demonstrates that our verification approaches is suitable for the $DMaS$ paradigm. The result on *Chipseq* datasets is similar. We omit the details due to the page limit.

8 Related Work

The issue of providing verification for database management was initially raised in the database-as-a-service (*DaS*) paradigm [10]. The problem being studied is to ensure the correctness and completeness of the result of SQL queries over outsourced databases (e.g., [15]). As our focus is the result integrity of data mining computations instead of SQL queries, our principles and techniques are fundamentally different from these work.

In theory, unconditional verification techniques are designed using interactive proofs [4] and probabilistically checkable proofs (PCPs) [3]. The interactive proofs and PCPs prove the integrity by answering questions interactively with the client. The proof might be very long, potentially too long for the client to process [8]. Most of the argument systems [9] can only be used for a restricted class of functions.

Only a few work [18, 11, 12] have studied the issue of integrity verification of data mining computations outsourced to third-party *DMaS* providers. [18] studies the problem of verifying the correctness and completeness of outsourced association rule mining. Its basic idea is to insert artificial items into the dataset; the mining result can be verified via the artificial (in)frequent itemsets constructed from artificial items. [11] and [12] apply the same strategy of using artificial data mining objects to the problems of verifying outlier mining and k-means clustering respectively. These work show that the design of artificial mining objects vary dramatically for different data mining problems. In this paper, we focus on Bayesian network structure learning problem. One of the main issues that we address is how to construct artificial structure to serve the verification purpose.

9 Conclusion

In this paper, we investigated how to provide result integrity guarantee for BN structure construction computation that is outsourced to an untrusted third-party service provider (e.g., the Cloud). We developed three probabilistic integrity verification methods that can provide high correctness guarantee with computation effort much cheaper than that of local BN structure construction. An interesting research direction for the future is to extend our methods to handle with the datasets in which each variable has more than two values (i.e., $r > 2$). It will also be interesting to explore how to adapt our verification techniques to BN learning algorithms that can find a globally optimal structure for all possible variable orderings [19, 14]. One possible solution to adapt our insertion-based method to these algorithms is to design artificial variables that can construct unique BN structures [16].

References

[1] Amazon elastic compute cloud (Amazon EC2). http://aws.amazon.com/ec2/.

[2] Google Prediction APIs. https://developers.google.com/prediction/.

[3] S. Arora and S. Safra. Probabilistic checking of proofs: A new characterization of NP. *Journal of ACM*, 45(1), 1998.

[4] L Babai. Trading group theory for randomness. In *STOC*, 1985.

[5] W. Buntine. Theory refinement on bayesian networks. In *UAI*, 1991.

[6] D. M. Chickering, D. Heckerman, and C. Meek. Large-sample learning of bayesian networks is NP-hard. *The Journal of Machine Learning Research*, 5:1287–1330, 2004.

[7] G. F. Cooper and E. Herskovits. A bayesian method for the induction of probabilistic networks from data. *Machine Learning*, 9(4):309–347, 1992.

[8] R. Gennaro, C. Gentry, and B. Parno. Non-interactive verifiable computing: outsourcing computation to untrusted workers. In *CRYPTO*, 2010.

[9] S. Goldwasser, Y. T. Kalai, and G. N. Rothblum. Delegating computation: interactive proofs for muggles. In *STOC*, 2008.

[10] H. Hacigümüş, B. Iyer, C. Li, and S. Mehrotra. Executing sql over encrypted data in the database-service-provider model. In *SIGMOD*, 2002.

[11] R. Liu, W. Wang, A. Monreale, D. Pedreschi, F. Giannotti, and W. Guo. Audio: An integrity *audit*ing framework of *o*utlier-mining-as-a-service systems. In *ECML/PKDD*, 2012.

[12] R. Liu, W. Wang, P. Mordohai, and H. Xiong. Integrity verification of k-means clustering outsourced to infrastructure as a service (iaas) providers. In *SDM*, 2013.

[13] R. Liu, W. Wang, C. Yuan. Result Integrity Verification of Outsourced Bayesian Network Structure Learning. Supplementary file. `http://www.cs.stevens.edu/~hwang4/papers/BN-integrity-sdm14-supplementary.pdf`.

[14] B. Malone, C. Yuan, E. Hansen, and S. Bridges. Improving the scalability of optimal Bayesian network learning with external-memory frontier breadth-first branch and bound search. In *UAI*, 2011.

[15] H. Pang, A. Jain, K. Ramamritham, and K.-L. Tan. Verifying completeness of relational query results in data publishing. In *SIGMOD*, 2005.

[16] J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann Publishers, Inc., San Mateo, CA, 1988.

[17] J. Rissanen. Stochastic complexity and modeling. *The Annals of Statistics*, pages 1080–1100, 1986.

[18] W. K. Wong, D. W. Cheung, E. Hung, B. Kao, and N. Mamoulis. An audit environment for outsourcing of frequent itemset mining. *Proceeding of VLDB Endowsement*, 2(1):1162–1173, 2009.

[19] C. Yuan, B. Malone, and X. Wu. Learning optimal Bayesian networks using A* search. In *IJCAI*, 2011.