

Approximate Functional Dependencies Discovery Using Markov Blanket

Jinqi Liu, Anzhen Zhang, Jiajia Li, Na Guo, Jing Zhang

Shenyang Aerospace University

Shenyang, China

liujinqi@stu.sau.edu.cn, {azzhang, lijiajia, guona, zhangjing2349}@sau.edu.cn

ABSTRACT

In this paper, we study the problem of discovering approximate functional dependencies (AFDs) from a noisy data set. A vast body of work have been proposed to recover the true AFDs in the dataset including constrained-based and probabilistic-based approaches. However, constrained-based approaches suffered from low accuracy due to overfitting to spurious AFDs, while probabilistic-based approaches are limited in recall since they cannot discover all of the determinants of an attribute. Motivate by this, we propose a hybrid approach MAFD that combines probabilistic-based structure learning techniques with constrained-based candidate search strategy to discover all determinants that are both syntactically valid and statistically significant. In particular, MAFD first use the probabilistic graphical model to learn the dependency structure of the probability distribution governing the input dataset. Then MAFD leverages the independent property of the Markov blanket to generate the search space for each attribute. Finally, MAFD traverses the search space to discover all determinants whose error is smaller than a given threshold. Extensive experiments on synthetic datasets demonstrate that MAFD can effectively address the overfitting problem without sacrificing too much efficiency, and outperform existing approaches significantly in terms of F_1 score.

VLDB Workshop Reference Format:

Jinqi Liu, Anzhen Zhang, Jiajia Li, Na Guo, Jing Zhang. Approximate Functional Dependencies Discovery Using Markov Blanket. VLDB 2024 Workshop: The 1st International Workshop on Data-driven AI (DATAI).

VLDB Workshop Artifact Availability:

The source code, data, and/or other artifacts have been made available at <https://github.com/jinqiliu/MAFD>.

1 INTRODUCTION

Functional Dependencies (FDs) are the most common constraints in relational databases, describing the relationships between attributes in a database relation. In few words, a FD $X \rightarrow A$ states that the value of the *right-hand side* (RHS) attribute A is uniquely determined by the value(s) of the *left-hand side* (LHS) attribute set X . For example, in a table with address data, the zip code is determined by the city and the street address. FD has wide applications in areas

such as data integration [1], schema normalization [2, 3], query optimization [4] and data cleaning [5].

Despite their importance, FDs of a specific dataset are usually unknown and almost impossible to discover manually. Therefore, various algorithms have been proposed to automatically discover *exact* dependencies, which are completely satisfied by the data, without even a single violation [6–11]. However, real-world dependencies are all too often not exact due to data error, exception and ambiguities. To cope with this problem, *approximate* functional dependencies (AFDs), which relax the definition of FDs to allow for a certain degree of violation, have been proposed [12].

Recent years have witnessed a surge of interest in AFDs discovery, which can be broadly classified into two categories according to their interpretation of AFDs: *constrained-based* and *probabilistic-based*. The former takes a closed-world assumption and aims to find all minimal non-trivial AFDs with error less than a threshold [12]. However, the constrained-based approaches tend to discover *overfitting* AFDs with large number of attributes in the determinant (i.e., LHS attribute set); these AFDs are syntactically valid but semantically incorrect. For example, consider a finite dataset I over a relation schema R and a candidate dependency $X \rightarrow A$ with $X \subseteq R$ and $A \in R$. Intuitively, as the number of attributes in X increases, it is more likely that $X \rightarrow A$ approximately holds in I (i.e., the error is less than a given threshold) since most X values are distinct and thus cannot introduce any inconsistencies. Moreover, constrained-based approaches suffer from the huge search space that grows exponentially with the number of attributes in the dataset [13].

While the probabilistic interpretation of AFDs [14] assumes that the instance of a relation schema is generated randomly according to a probability distribution and an AFD $X \rightarrow A$ indicates that X and A are strongly correlated in that distribution. Various statistical learning methods have been proposed from both the database and data mining areas to recover AFDs according to the correlations between the LHS and RHS. However, they all suffer from a common fundamental limitation: they cannot recover all of the determinants of a RHS attribute. As a consequence, the recall of these methods is far from satisfactory. For example, FDX [14], the state-of-the-art approach in database, can only recover one determinant for each RHS attribute, and RFI[15], the state-of-the-art approach in data mining, detects only top k determinants for each RHS attribute.

Motivated by this, we propose to take the best of both approaches to recover as many true AFDs as possible. In particular, a hybrid AFDs discovery approach MAFD is proposed, which combines probabilistic-based structure learning techniques with constrained-based candidate search strategy in two phases. In the first phase,

This work is licensed under the Creative Commons BY-NC-ND 4.0 International License. Visit <https://creativecommons.org/licenses/by-nc-nd/4.0/> to view a copy of this license. For any use beyond those covered by this license, obtain permission by emailing info@vldb.org. Copyright is held by the owner/author(s). Publication rights licensed to the VLDB Endowment.

Proceedings of the VLDB Endowment. ISSN 2150-8097.

MAFD employs a probabilistic graphical model, e.g., Bayesian network, to learn the dependency structure of the underlying distribution. Then MAFD leverages the independent property of the Markov blanket to generate the search space for each RHS attribute. The key idea of this step is that, for each RHS attribute A , the Markov blanket of A should contain at least one determinant of A , which has been proved in lemma 1. In the second phase, MAFD traverses the search space to discover all determinants whose error is smaller than a given threshold.

While the independent property of the Markov blanket simplifies the search of determinants, involving Markov blanket for AFDs discovery is not straightforward as it raises the following challenges: the quality of the learned structure of the probabilistic graphical model is heavily influenced by the nature of the data from which it is learned, including data size, distribution, noise level and so on. For example, if the data is biased or does not cover the entire range of possible values, the learned structure might miss important dependencies or introduce false ones, leading to inaccurate models. Therefore, how to effectively preprocess the training data to improve the quality of the learned structure is a critical challenge.

To address this challenge, MAFD first creates a set of random sample over the input instance and transforms the sample data before training. The reason for transformation is that for correlated attributes with large domains, the co-occurrences on a limited sample may not be frequent enough for the learning methods to capture the correlation. Therefore, MAFD decomposes each large domain into a relatively small number of disjoint categories and then bucketize the sample data accordingly. It’s worth mentioning that such transformation will not effect the accuracy of MAFD, as will be discussed in Section 4.1.

The remainder of this paper is structured as follows. In Section 2, we survey related work. In Section 3, we formalize the problem of AFDs discovery and present the overall framework of MAFD. Then we describe each stage of MAFD in detail in Section 4 and Section 5, respectively. Finally we conclude in Section 6.

2 RELATED WORK

constrained-based approaches. This line of work can be classified as follows. (i) *Row-efficient approaches*, which model the search space of AFDs as a power lattice and employ different pruning and lattice traversal strategies, e.g., TANE [6], Fun [7], FDMine [8], DFD [9] and Pyro [12]. They have been shown to perform well on datasets with many tuples, but scale poorly with the number of columns in the input dataset. (ii) *Column-efficient approaches*, which first find non-FDs rules by comparing all pairs of tuples and then deduce candidate FDs rules from these non-FDs rules. Representative methods include Fdep [10], Dep-Miner [11] and FastFDs [16]. Compared with the lattice traversal strategy, this type of algorithm exhibits good scalability with respect to the number of attributes, but its complexity is quadratic with respect to the number of tuples, making it performing poorly on datasets with a large number of tuples. (iii) *Hybrid approaches*, which manage to scale equally well with growing numbers of tuples and attributes by combining column-efficient FD induction techniques with row-efficient FD search techniques. For example, HyFD [17] calculates

non-FDs by tuple comparisons and validate them by position list indices (PLI) intersections.

Note that constrained-based approaches aim to find AFDs that satisfy the syntactical requirements, without considering the statistical characteristics of the dependencies. As a result, hundreds of spurious AFDs will be discovered by the constrained-based approaches for datasets with only tens of attributes [12]. Moreover, constrained-based approaches suffer from the huge search space that grows exponentially with the number of attributes, since they need to verify all candidate dependencies that might be syntactically valid. Although various pruning strategies have been employed in constrained-based approaches to reduce the size of the search space, it is still very hard for them to scale up to datasets with real-world size as show in [13].

Probabilistic-based approaches. There are three representative approaches in this line and all of them are limited in recall. (1) FDX [14] is the first to use structure learning for AFDs discovery. It models the distribution that AFDs impose over pairs of tuples and leverages linear dependencies to recover AFDs. However, as discussed before, FDX is limited in terms of recall since it can only recover one determinant for each RHS attribute. In addition, FDX cannot discover cyclic dependencies since it assumes a global ordering over the attributes and only allows all attributes in the LHS precede the RHS attribute in that ordering. (2) RFI[15], the state-of-the-art AFDs discovery approach in data mining, detects only top k determinants for each RHS attribute according to an information theoretic score. As a result, the performance of RFI heavily depends on the choice of k . Unfortunately, there usually does not exist a global optimal k which can be appropriate for all RHS attributes since the number of determinants for each RHS attribute varies a lot. (3) CORDS [18], leverages correlation-related statistics to obtain AFDs. However, it can only discover unary AFDs whose determinant contains only one attribute and therefore it fails to identify binary and ternary AFDs, which are quite common in real-word datasets.

3 PROBLEM STATEMENT

In this section, we first review some background material and then give the formal definition of the AFDs discovery problem.

3.1 Approximate Functional Dependency

Given a relation schema \mathcal{R} with \mathcal{A} denoting its set of attributes and D be an instance over \mathcal{R} . An approximate functional dependency $X \rightarrow A$ is a statement over a set of attributes $X \subseteq \mathcal{A}$ and an attribute $A \in \mathcal{A}$, denoting that the value(s) of X approximately determines the value of A . Under the constrained-based interpretation of AFDs, there are many possible ways of defining the approximateness of an AFD. In this paper, we use a slight adaptation of the well-established g_1 error as in [12], which has a natural interpretation as the fraction of tuple pairs that violate the dependency; the dependency error is defined as:

$$e(X \rightarrow A, D) = \frac{|\{(t_1, t_2) \in D^2 \mid t_1[X] = t_2[X] \wedge t_1[A] \neq t_2[A]\}|}{|D|^2 - |D|} \quad (1)$$

Note that $e(X \rightarrow A, D)$ ranges over $[0, 1]$, having the value close to 0 when $X \rightarrow A$ almost holds in D . The constrained-based interpretation of AFDs is defined based on the dependency error.

Definition 3.1. (AFD under constrained interpretation). Given a relation D and an error threshold ϵ , $0 \leq \epsilon \leq 1$, we say $X \rightarrow A$ is an approximate functional dependency (AFD) if and only if $e(X \rightarrow A, D)$ is at most ϵ .

An AFD is *minimal* if the errors of all its generalizations are greater than ϵ . Accordingly, the determinant of a minimal AFD is called a *minimal* determinant. Now we introduce the probabilistic interpretation of AFDs. Let each attribute $A \in \mathcal{A}$ has a domain $dom(A)$ and the domain of a set of attributes $X = \{A_1, A_2, \dots, A_k\} \subseteq \mathcal{A}$ is defined as $dom(X) = dom(A_1) \times dom(A_2) \times \dots \times dom(A_k)$. Then we assume that the instance of \mathcal{R} is generated randomly according to a probability distribution P_R and the probabilistic interpretation of AFD is defined as below.

Definition 3.2. (AFD under probabilistic interpretation). Given the distribution P_R , we say that $X \rightarrow A$ is an AFD if there is a function $f : dom(X) \rightarrow dom(A)$ such that :

$$\forall x \in dom(X) : P_R(A = a | X = x) = \begin{cases} 1 - \epsilon, & \text{when } a = f(x) \\ \epsilon, & \text{otherwise} \end{cases} \quad (2)$$

with ϵ being a small constant, allowing some violations to the functional relation.

3.2 Problem Definition

As analysed before, in order to discover all AFDs that are both syntactically valid and statistically significant, we combine the advantages of the constrained-bases interpretation and the probabilistic interpretation of AFDs. The AFDs discovery problem under this setting is defined as below.

Definition 3.3. (AFDs discovery problem). Given a relation D and an error threshold ϵ , we want to identify all AFDs that characterize the distribution behind D and the dependency errors are at most ϵ .

Note that the AFDs discovered in this way can solve the overfitting problem of the constrained-bases approaches since the uncorrelated attributes cannot appear in the determinant of a RHS attribute. Besides, the low recall problem of the probabilistic-based approaches can be alleviated to a large extent by verifying all possible determinants (i.e., comparing the dependency error with the threshold) among the correlated attributes of a RHS attribute.

3.3 Solution Overview

An overview of our approach MAFD is shown in Figure1. The input is a noisy dataset D with an error threshold ϵ , and the output is a set of discovered AFDs. MAFD mainly contain two stages: dependency structure learning and power lattice traversal.

Dependency Structure Learning. In this stage, MAFD first creates a random sample over the input instance and transform the sample data according to our bucket strategy in order to enhance the correlations among attributes. Then a Bayesian network is trained to learn the dependency structure of the transformed sample. Finally, MAFD computes the Markov blankets of each attribute in the Bayesian network.

Power Lattice Traversal. In this stage, MAFD first generates one search space per RHS attribute with the Markov blanket of that attribute to be the peak in the power lattice. Then a depth first search strategy is employed to discover all minimal determinants with dependency errors less than ϵ . The dependency errors are computed efficiently based on position list index, a widely used structure in constrained-based approaches.

4 DEPENDENCY STRUCTURE LEARNING

In this section, we describe the workflow in the first stage of MAFD including data transformation, Bayesian network learning and Markov blanket computation.

4.1 Data Transformation

First, we create a random sample over the input instance and then employ the hash bucketization to distribute attribute values across a set of buckets using a hash function. As discussed before, in order to capture all dependencies among attributes, especially those strong dependencies introduced by AFDs, we need to compress the large domain space to a small space. In this way, the co-occurrences among correlated attributes will be increased in the sample and it will be much more easier for the Bayesian network learning approach to discover such correlation.

Specifically, for each attribute A with large domain, a hash function h is applied to the attribute value x to compute its hash value $h(x)$. Then we determine the bucket in which x will be placed by using a modulo operation: $bucket = h(a) \bmod m$, where m is the total number of buckets. For example, consider an attribute value apple, a hash function $djb2$ (a simple yet effective hash function) and a bucket number 10, then we can determine the bucket is 5 since $djb2(\text{apple}) \bmod 10 = 5$.

Note that this transformation strategy will not effect the effectiveness of MAFD since the true dependencies among correlated attributes are enhanced. However, the efficiency of MAFD might downgrade to some extent since false correlations among independent attributes could be introduced. For example, the Markov blanket of a RHS attribute could possibly contain some attributes that are independent with that attribute after data transformation. Therefore, the search space based on the Markov blanket will grow and the search cost will increase accordingly. Nevertheless, these false correlations will not be discovered as AFDs since MAFD will valid them on the original dataset, i.e., compute their dependency errors.

4.2 Bayesian Network Learning

Learning the structure of a Bayesian network involves discovering the dependencies among variables and determining the best graphical representation of these dependencies. There are three main categories of methods used for learning Bayesian network structures: constrained-based methods, score-based methods, and hybrid methods. In this paper, we adopt a score-based method, BANJO [19], to learn the dependency structure of the transformed sample. BANJO adopts a heuristic search strategy simulating annealing algorithm to avoid falling into local optimal solutions. Note that the other Bayesian network learning approaches can also be adopted in this step.

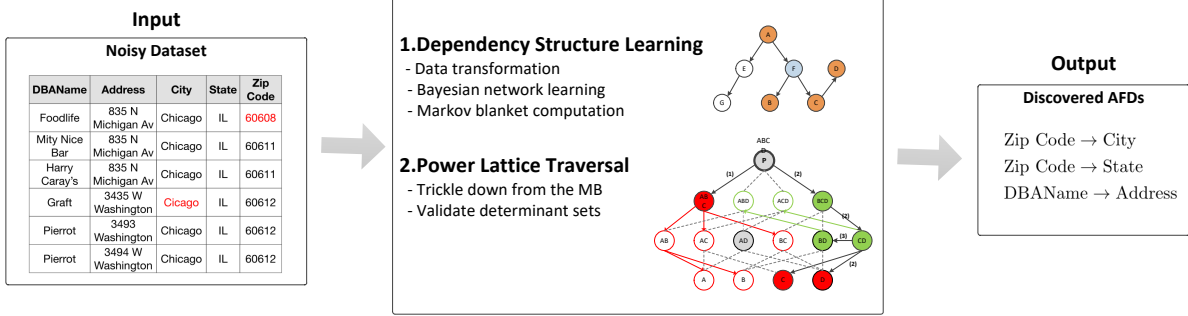


Figure 1: An overview of MAFD framework

4.3 Markov Blanket Computation

After obtaining the Bayesian network, we now compute the Markov blanket for each node in the network. The Markov blanket of a node X in a Bayesian network is defined as the minimal set of nodes in the network such that X is conditionally independent of all other nodes given this set. In particular, the Markov blanket of X includes the parents of X , children of X and parents of the children of X in the Bayesian network.

Definition 4.1. (Markov blanket). In a Bayesian network, let G be a directed acyclic graph representing the network structure, and let $Pa(X)$, $Ch(X)$, and $Pa(Ch(X))$ denote the parents of X , the children of X , and the parents of the children of X (excluding X), respectively. Then, the Markov blanket $MB(X)$ of node X is defined as:

$$MB(X) = Pa(X) \cup Ch(X) \cup Pa(Ch(X)) \setminus X \quad (3)$$

Example 4.2. Figure 2 is an example of a Bayesian network. In this Bayesian network, the Markov blanket of node T includes $Pa(X) = \{X_1, X_2\}$, $Ch(T) = \{X_6, X_7\}$ and $Pa(Ch(T)) = \{X_8\}$, i.e., $MB(T) = \{X_1, X_2, X_6, X_7, X_8\}$.

Let A be an attribute in the Bayesian network learned from the sample. According to the independent property of the Markov blanket, the attributes affecting the distribution of A are all in the Markov blanket of A . Suppose there exists an AFD $X \rightarrow A$, i.e., A is approximately determined by X . Hopefully, X should be involved

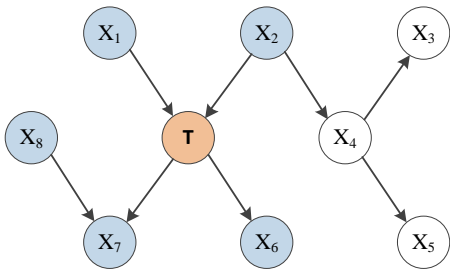


Figure 2: Example of Bayesian Network

in the Markov blanket of A . In fact, according to the following lemma, we know that the Markov blankets of A contains at least one determinant of A as long as the learned Bayesian network can accurately reflect the underlying dependencies among attributes.

LEMMA 1. *Given a relation schema R and an accurate Bayesian network G describing the dependencies in R . Let $D(A)$ denote the set of A 's determinants. Then the Markov blanket of A contains at least one determinant in $D(A)$.*

PROOF. Note that $\forall X \in D(A)$, we have $X \rightarrow A$. If $X \subseteq MB(A)$, then the lemma is proved. Suppose $X \not\subseteq MB(A)$. According to $X \rightarrow A$, we know that the value of A is approximately determined by the value(s) of X . Suppose when $X = x$, the corresponding $A = a$. Then we have $P_R(A = a | MB(A), X = x) = 1 - \epsilon$, for a small ϵ . According to the conditional independence of the Markov blanket, we have $P_R(A = a | MB(A), X = x) = P_R(A = a | MB(A))$. In other words, $P_R(A = a | MB(A)) = 1 - \epsilon$ must be true, which requires at least one determinant of A in $MB(A)$. Hence, the lemma is proven. \square

The above lemma demonstrates that we can recover at least one determinant of a RHS attribute based on the Markov blanket. In fact, the experiments on various datasets demonstrate that MAFD can discover almost all of the determinants.

5 POWER LATTICE TRAVERSAL

In this section, we introduce how to traverse the search space and validate the candidate determinants efficiently.

5.1 Trickle Down From Markov Blanket

According to our previous analysis, one can search the determinants of a RHS attribute within its Markov blanket. Therefore, MAFD creates one search space per RHS attribute with the Markov blanket of that attribute to be the *peak* in the power lattice. Then MAFD employs a depth first search strategy similar to [12] to find all minimal determinants of the RHS attribute. Algorithm 1 outlines the traversal process.

First, we calculate the dependency error of the peak and if it is less than ϵ (Line 3), then the peak is a determinant, but potentially not minimal. Therefore, we perform a downward process to validate its generations (i.e., its subsets). Specifically, we first initialize the

Algorithm 1: Discover minimal determinants

```

1 Input: dataset  $D$ , peak  $P$ , maximum error  $\epsilon$ ;
2 Output: a set of minimal determinants  $\mathcal{M}$ ;
3 if  $e(P, D) \leq \epsilon$  then
4    $\mathcal{M} \leftarrow \phi$ ;
5    $\mathcal{P} \leftarrow$  immediate generalizations of  $P$ ;
6   while  $\mathcal{P}$  is not empty do
7      $P' \leftarrow$  a candidate peek from  $\mathcal{P}$ ;
8      $\mathcal{M}' \leftarrow$  look up subsets of  $P'$  in  $\mathcal{M}$ ;
9     if  $\mathcal{M}' \neq \emptyset$  then
10      remove  $P'$  from  $\mathcal{P}$ ;
11      for each subset  $p$  of  $P'$  but not superset of any
12        element in  $\mathcal{M}'$  do
13          if  $p$  is not a non-FD then
14            add  $p$  to  $\mathcal{P}$ ;
15      else
16         $M \leftarrow$  trickle-down-from( $P'$ ,  $\epsilon$ );
17        if  $M \neq \perp$  then
18          add  $M$  to  $\mathcal{M}$ ;
19        else
20          remove  $P'$  from  $\mathcal{P}$ ;
21 Function trickle-down-from( $P'$ ,  $\epsilon$ ):
22   if  $|P'| > 1$  then
23      $\mathcal{G} \leftarrow$  immediate generalizations of  $P'$ ;
24     while  $\mathcal{G}$  is not empty do
25        $G \leftarrow$  a candidate determinant from  $\mathcal{G}$ ;
26       if  $e(G, D) > \epsilon$  then break;
27        $C \leftarrow$  trickle-down-from( $G$ ,  $\epsilon$ );
28       if  $C \neq \perp$  then return  $C$ ;
29   if  $e(P', D) \leq \epsilon$  then return  $P'$ ;
30   return  $\perp$ 

```

minimal determinants \mathcal{M} to be an empty set and set the candidate peak set \mathcal{P} to contain the immediate generalizations of P (i.e., removing one attribute from P) (Line 5). Then MAFD randomly picks one candidate peak P' from \mathcal{P} at one time (line 6-7) and checks whether it is pruned by a minimal determinant in \mathcal{M} (Line 8). If there exists a set of minimal determinants \mathcal{M}' such that P' is their superset (line 9), then we can prune P' from the candidate peak set \mathcal{P} (line 10) and add all subsets of P' but not supersets of \mathcal{M}' in \mathcal{P} (line 12); otherwise, we perform a recursive downward search until we find a minimal determinant in P' or no determinant can be found (line 14-28).

Example 5.1. Consider a RHS attribute F . Suppose the Markov Blanket of F is $ABCD$ and the dependency error of $ABCD$ is less than ϵ , then the search space of F is shown in Figure 3 with peak $ABCD$. In the first loop, the peak candidate set includes the immediate generalizations of $ABCD$, i.e., $\mathcal{P} = \{ABC, ABD, ACD, BCD\}$, and the minimal determinants set $\mathcal{M} = \phi$. Then we randomly select a candidate peak from \mathcal{P} , say ABC , and check whether ABC is a superset of any determinant in \mathcal{M} . If so, ABC is not a minimal determinant and we can remove it from the candidate set. Since $\mathcal{M} = \phi$ at this time, we continue to trickle down from ABC . Suppose

the dependency error of ABC is greater than ϵ , then the *trickle-down-from* function will return \perp and we remove ABC from \mathcal{P} , i.e., $\mathcal{P} = \{ABD, ACD, BCD\}$. We call $ABC \rightarrow F$ a non-AFD and we can prune all generalizations of ABC from the search space (the pruned nodes are marked in red in Figure 3).

In the second loop, we randomly select a peak from \mathcal{P} , say BCD . Suppose its error is less than ϵ , then we recursively trickle down from BCD in a depth-first manner until we find a minimal determinant. Assume the error of CD is less than ϵ but the errors of all its generations, i.e., C and D , are larger than ϵ . Then CD will be discovered as a minimal determinant and we put CD in \mathcal{M} , i.e., $\mathcal{M} = \{CD\}$.

In the third loop, suppose we also select BCD from \mathcal{P} . Since BCD is a super set of CD , which is a minimal determinant, we need to search the other minimal determinants in BCD . To achieve this goal, we first remove BCD from the peak candidate set to avoid infinitely traverse the CD path and add all unpruned subsets of BCD but not superset of CD in the peak candidate set, i.e., $\mathcal{P} = \{ABD, ACD, BD\}$. In this way, we are guaranteed to traverse all paths in BCD except the one including CD .

In the fourth loop, suppose we select BD from \mathcal{P} and its error is less than ϵ while its generations errors are larger than ϵ . Then BD is another minimal determinant and we add it in \mathcal{M} , i.e., $\mathcal{M} = \{CD, BD\}$. The above process will continue until there is no candidate peak in \mathcal{P} .

It is not difficult to verify that our search strategy can identify all minimal determinants that are subsets of the Markov blanket. In addition, note that the Markov blanket of a RHS attribute A is usually very small since it only involves attributes correlated with A . Therefore MAFD is not sensitive to the number of attributes in the dataset. Experimental results show that MAFD scales equally well with growing numbers of tuples and attributes. Besides, MAFD effectively addresses the issue of overfitting since it only consider determinants within the Markov blanket rather than the entire attribute set.

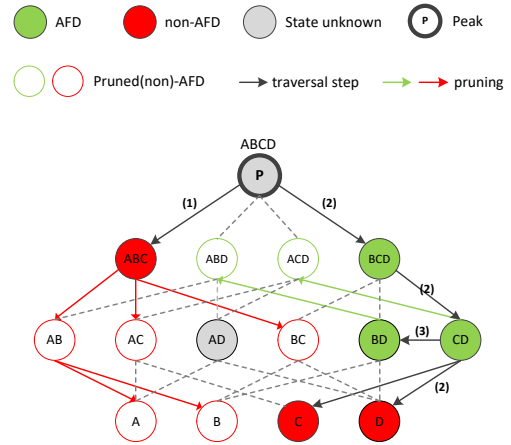


Figure 3: Example of a search space

6 EXPERIMENTS

In this section, we compare our algorithm with other approximate functional dependency discovery algorithms. We attempt to validate: (1) whether the use of Markov blankets to determine the left decision set can accurately discover AFDs, (2) the impact of datasets with different features on various FD discovery methods, (3) does the MAFD exhibit robustness to different parameters in the dataset.

6.1 Experimental Setup

We employ the BANJO framework for Bayesian network training on data, which focuses on score-based structural inference. It adopts a heuristic search strategy simulating annealing algorithm to avoid falling into local optimal solutions.

Datasets and Evaluation Goals. For a more comprehensive assessment of FD detection methods with consideration for FD type integrity, we conduct experiments on synthetic datasets. In real-world datasets, the true FDs are often unknown or inaccurate, which may lead to imprecise results when evaluating FD discovery methods. In the synthetic datasets, we can accurately identify the true FDs and control the quantity of FDs, thus enabling evaluate various FD discovery methods more precisely.

In our experiments, we adopt a data generation method similar to FDX. Given a relational schema with r attributes, we design two common functional dependency patterns: Categorical and numerical type. The Categorical type include FDs with one attribute on the RHS, FDs with two attributes on the RHS, FDs with three attributes on the RHS, and FDs that are set with cycles. For categorical FDs, our generator initially determines the number of FD to be generated for each type, and allocates a corresponding number of attributes for each FD. Let (X, Y) be a set of attributes distributed in the aforementioned manner. Our generator initially assigns a domain $\text{dom}(X)$ for each attribute in X with the setting for Domain Cardinality, while simultaneously assigning a domain $\text{dom}(Y)$ for Y . For each attribute x in X , we randomly select a value from $\text{dom}(x)$ to assign to x . For each determined x , a value y from $\text{dom}(y)$ is randomly selected as its corresponding RHS. If there are multiple attributes in the LHS, they are considered as a whole and the same process as above is followed. The number of samples generated is determined by the value of the parameter assigned to the number of tuples. For numeric FDs, first assign a domain to each of the LHS attributes $\text{dom}(X_1)$ and $\text{dom}(X_2)$, respectively select random values from $\text{dom}(X_1)$ and $\text{dom}(X_2)$ to be assigned to x_1 and x_2 . Then, based on the function: $x_1 + x_2 = y$, we determine the value of the corresponding RHS.

Next, in order to test the robustness of the AFD discovering algorithm to noise, we randomly flip the cells corresponding to the attributes involved in the real functional dependency to different

values in their domain, thereby changing the noise environment of the data. The percentage of flipped cells is adjusted by controlling the settings of the noise ratio.

We evaluate the above methods as we vary four key factors in the data: (1) Noise Rate, in order to test the robustness of FD discovery methods; (2) Number of Tuples, in order to the scalability of the FD discovery methods with respect to data sample size; (3) Number of Attributes, in order to test the scalability of the FD discovery methods with respect to the number of data attributes. (4) Domain Cardinality, in order to evaluate the sample complexity of FD methods. The values of each factor are summarized in the table 1.

Methods. We considered seven comparative methods: (1) TANE, which is a classical algorithm to discovery of functional and approximate dependencies from relations. The algorithm derives effective dependencies from the partition of relations and adopts a breadth-first search strategy. The code is released by the authors.¹ (2) Reliable Fraction of Information (RFI), which relies on an information theoretic score to find FDs and uses an approximation scheme to optimize performance. The approximation ratio of RFI is controlled by the hyperparameter α to balance the accuracy and efficiency of the algorithm [15]. The code is also released by the authors.² (3) PYRO, as an advanced AFD mining method in the field of databases, its main goal is to identify all syntactically valid AFDs in the dataset. This method adopts a separate-and-conquer strategy, quickly determines the minimal dependencies through agree set samples, achieves efficient verification, and effectively uses the discovered dependencies to prune the search space. The code is released by the authors.³ (4) FDX, adopts a statistical perspective to transforms AFD mining into a structural learning problem on linear structural equation models. It constructs samples by sampling value differences of tuple pairs from the original data and performs structural learning on the samples. The code is released by the authors.⁴ (5) FEDP, addresses the problem of discovering functional dependencies from a machine learning perspective. It constructs negative coverage and negative coverage inversion by pairwise comparison of all tuples in a given relation, and uses top-down and bottom-up algorithms to hypothesize and test dependencies. (6) DFD, which Models the search space into multiple lattices. It discovers all functional dependencies following a depth first traversal strategy of the attribute lattice that combines aggressive pruning and efficient result verification. (7)CORDS, by considering correlations to discover correlations between columns and soft FDs. It proposes an example-based method that uses system cataloging to retrieve the number of different values in columns. However, it only studies FDs with a single attribute on the LHS [18].

Evaluation Methodology. We measure the effectiveness of the competing methods by using precision (P), recall (R), and F_1 -score. F_1 -score is defined as :

$$F_1 = \frac{2 * Precision * Recall}{Precision + Recall} \quad (4)$$

Table 1: The different settings for synthetic data sets.

Factors	Settings
Noise Rate	1% - 30%
Tuples	$10^3 - 10^5$
Attributes	5 - 30
Domain Cardinality	[25,100] - [625,900]

¹<https://www.cs.helsinki.fi/research/fdk/datamining/tane/>

²<http://eda.mmci.uni-saarland.de/prj/dora/>

³<https://github.com/HPI-Information-Systems/pyro/releases>

⁴<https://github.com/sis-ethz/Profiler-Public>

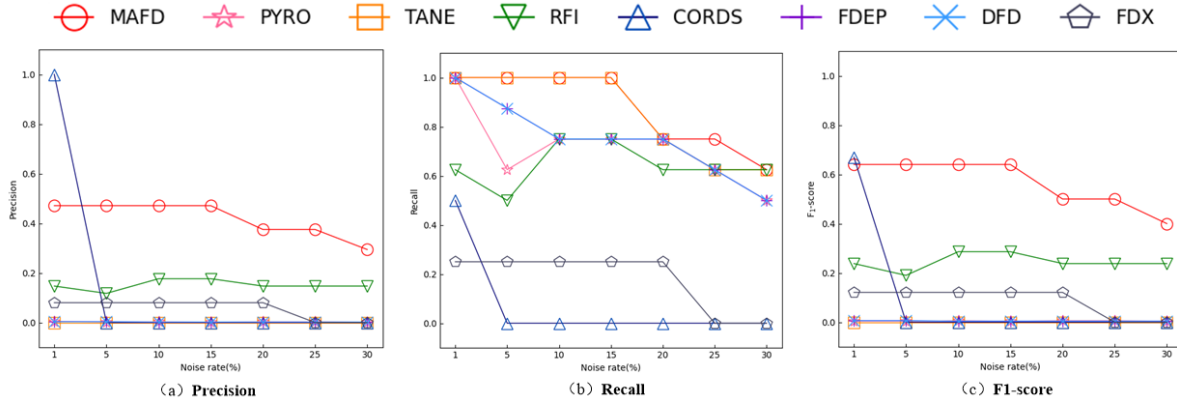


Figure 4: Experimental comparison of varying the rate of noise on synthetic datasets

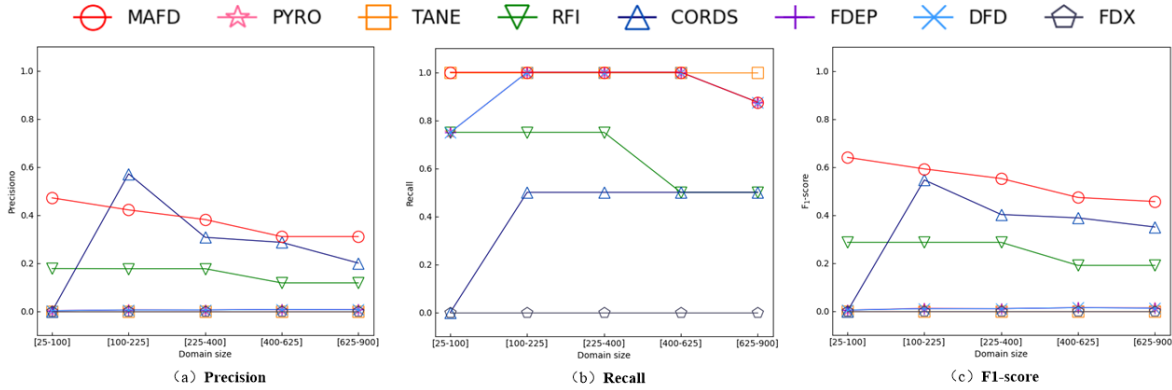


Figure 5: Experimental comparison of varying the size of domain on synthetic datasets

where precision refers to the fraction of correctly discovered edges that participate in true FDs by the total number of edges in discovered FDs, recall refers to the fraction of correctly discovered edges that participate in the true FDs by the total number of true edges in the FDs of a data set.

Implementation Details. All experiments were executed on a machine with Intel(R) Core(TM) i7-7700HQ CPU @ 2.80GHz processor, 8GB RAM, Windows 10 operating system.

6.2 Validity Evaluation

6.2.1 Robustness of noise.

We evaluate the performance of our approach and other methods on synthetic datasets by altering various key factors of the data. In our first experiment, we compared the performance of various methods in different noise levels. The experiment uses the synthetic dataset with 19 columns and 5000 rows, its domain size is set to 216. We measured the performance under varying noise rates ranging from 1% to 30%. The results are shown in Figure 4.

The results indicate that with the increase in noise ratio, the performance of all methods shows a decreasing trend. It is worth noting that MAFD demonstrates better precision, recall and F_1 score overall compared to other methods, and maintains relative stability.

This suggests that MAFD has excellent robustness to noisy data. We turn our attention to constrain-based approaches, Pyro, Tane, Fedp and Dfd, although their recall rates are considerable, their precision and F_1 -scores are both vary low. This is because their goal is to discover grammatically valid AFD, thereby identifying a large number of spurious AFDs.

For probabilistic-based approaches, RFI identifies AFDs by optimizing information theory scores, but this score also tends to overfit the input samples, resulting in low precision. Therefore, its F_1 -score is also lower than that of MAFD. Cords is extremely sensitive to noise, and due to inherent limitations of the method, it can only discover AFDs whose determinant contains only one attribute. When the dataset has few or no unary AFDs, it struggles to discover other types of AFDs, resulting in a low F_1 -score. FDX exhibits relative stability in response to changes in the noise environment, but its accuracy is low. FDX employs structural learning to globally rank attributes and requires that the priority of left-hand attributes be higher than right-hand attributes. Consequently, attribute prioritization affects performance, and it is unable to recognize AFDs with cyclic dependencies, resulting in lower accuracy and recall.

6.2.2 Complexity evaluation.

The domain cardinality is one of the important indicators to

Table 2: Experimental comparison of varying the number of tuples on synthetic dataset.

Tuple Sizes	MAFD	TANE	PYRO	RFI	CORDS	FDEP	DFD	FDX
1000	10.832	19.485	9.694	10.676	0.106	16.814	2.797	0.321
2000	12.444	20.68	7.69	10.546	0.127	15.766	3.09	0.305
5000	12.339	49.691	10.405	13.769	0.156	15.399	3.261	0.721
10000	13.167	124.308	9.174	8.958	0.195	25.031	4.817	0.351
20000	13.871	295.258	8.779	19.169	0.267	69.019	6.1	0.339
50000	15.875	-	12.183	-	0.35	385.501	11.847	0.73
100000	18.242	-	17.407	-	0.495	1489.647	14.202	1.41

'-' method exceeds runtime limit (8 hours).

Table 3: Experimental comparison of varying the number of attributes on synthetic dataset.

Attributes	MAFD	TANE	PYRO	RFI	CORDS	FDEP	DFD	FDX
5	5.285	0.137	1.11	0.265	0.1	1.751	0.113	0.162
10	10.755	4.748	5.787	1.09	0.111	8.741	0.442	0.198
15	21.113	570.468	118.49	8.205	0.131	113.824	2.156	0.365
20	41.216	-	227.16	17.96	0.142	297.536	11.643	0.552
25	64.595	-	1268.542	30.75	0.175	1421.856	53.246	0.791
30	66.041	-	2455.25	41.1	0.225	-	-	1.195

'-' method exceeds runtime limit (2 hours).

measure the sample complexity of AFD methods, reflecting the number of different types of attribute values in the dataset. To comprehensively evaluate the performance of different AFD discovering methods under different sample complexities, we generated a series of datasets with 17 columns, 5000 tuples, and a noise rate of 5%. The domain cardinality is set from 25 to 900. By adjusting the domain cardinality of the dataset, we can compare the performance of various methods under different complexity conditions. The experimental results are shown in Figure 5.

From the experimental results, it can be observed that as the domain cardinality increases, the performance of all methods shows a downward trend. Notably, we preprocess the data used for training our Bayesian network to mitigate the impact of domain cardinality on network training. The Markov blanket of a determined attribute can still be guaranteed, therefore, the F_1 -score is higher than other methods. For constrain-based approaches, Pyro, Tane, Fdep and Dfd still perform poorly due to overfitting. At the same time, we found that the performance of FDX has been consistently poor because the synthetic dataset includes cyclic FDs, and FDX can only find one determinant on the left for each attribute on the right, which leads to a low recall rate.

6.3 Scalability of Attributes and Tuples

6.3.1 Varying the number of tuples.

We compare the scalability of various AFD discovery methods by changing the number of tuples. To comprehensively study the impact of different sample sizes on algorithm performance, we set up a series of synthetic datasets with 17 columns, a domain cardinality of 216, and a noise rate of 5% and the number of tuples ranges from 1,000 to 100,000. To ensure the efficiency and controllability of the experiment, it automatically stops when the memory consumption

exceeds 8GB or the running time exceeds 8 hours. The results are shown in table 2.

Firstly, MAFD maintains good efficiency across datasets of different scales. The runtime of MAFD can be divided into data preprocessing time, data training time, and FD mining time. Based on the Bayesian network framework we have chosen, through experiments, we have found that setting the training time to 10 seconds results in a stable network structure while maintaining a high F_1 -score. New we turn our attention to constrain-based approaches, Pyro and Dfd all column-efficient approach. Therefore, they can still perform quickly on datasets with a large number of tuples. However, Tane cannot terminate on larger datasets due to severe overfitting. In our dataset with 17 columns, it can even find over ten thousand AFDs. It is worth noting that Fdep is a row-inefficient algorithm. Its complexity is $O(n^2)$ (n is the number of tuples), as the number of tuples increases, its runtime increases dramatically.

For probabilistic-based approaches, the runtime of FDX and Cords is consistently low, as they only mine a single left-hand determinant for each AFD, resulting in a correspondingly low recall rate. RFI also cannot terminate on larger datasets, we attribute this performance to RFI's score that tend to overfit the input sample.

6.3.2 Varying the number of attributes.

This experiment analyzes the algorithms' scalability of columns, we set up a series of synthetic datasets with 5000 tuples, domain cardinality of 216, noise rate of 5% and the number of attributes ranges from 5 to 30. To ensure the efficiency and controllability of the experiment, it automatically stops when the memory consumption exceeds 8GB or the running time exceeds 2 hours. The results are shown in table 3.

In our attribute scalability experiments, MAFD can maintain efficient discovery efficiency as the number of attributes increases. This is attributed to our utilization of the Markov blanket pruning

rule, which significantly reduces the search space. It is worth noting that the majority of the runtime of MAFD is dedicated to learning the network structure. As the number of attributes increases, we will augment the time spent on learning the network structure to ensure the acquisition of higher-scoring network architectures. We observe that traditional power lattice-based methods, TANE and PYRO show exponential increases in runtime as the number of attributes grows, with TANE even failing to terminate when there are many attributes. FDX requires the transformation of the dataset into a set of observations for a linear model, hence the runtime of FDX significantly increases on datasets with a larger number of attributes.

7 CONCLUSIONS

In this paper, We introduced MAFD, an approximate functional dependency discovering algorithm based on Markov blankets. The key of the algorithm is to propose the use of Bayesian networks to learn the relationships between attributes, determine the Markov blanket of the right-hand side attributes, and thereby pinpoint the peak of the maximum search space for the left-hand side determinant. This greatly reduces the search space while avoiding overfitting. Our experimental results indicate that we can improve the precision of discovering while ensuring a high recall rate, thereby achieving a better F_1 -score.

REFERENCES

- [1] V. Matos and B. Grasser. Sql-based discovery of exact and approximate functional dependencies. *ACM SIGCSE Bulletin*, 36(4):58–63, 2004.
- [2] U. S. Chakravarthy, J. Grant, and J. Minker. Logic-based approach to semantic query optimization. *ACM Transactions on Database Systems (TODS)*, 15(2):162–207, 1990.
- [3] J-M Petit, F Toumani, J-F Boulicaut, and J Kouloumdjian. Towards the reverse engineering of denormalized relational databases. In *Proceedings of the International Conference on Data Engineering (ICDE)*, pages 218–227, 1996.
- [4] J Kossmann, T Papenbrock, and F Naumann. Data dependencies for query optimization: a survey. *The VLDB Journal*, 31(1):1–22, 2022.
- [5] T Rekatsinas, X Chu, I F Ilyas, et al. Holoclean: Holistic data repairs with probabilistic inference. *arXiv preprint arXiv:1702.00820*, 2017.
- [6] Y Huhtala, J Kärkkäinen, P Porkka, et al. Tane: An efficient algorithm for discovering functional and approximate dependencies. *The computer journal*, 42(2):100–111, 1999.
- [7] N Novelli and R Cicchetti. Functional and embedded dependency inference: a data mining point of view. *Information Systems*, 26(7):477–506, 2001.
- [8] H Yao, H J Hamilton, and C J Butz. *fd_mine*: Discovering functional dependencies in a database using equivalences. In *2002 IEEE International Conference on Data Mining, 2002. Proceedings*, pages 729–732. IEEE, 2002.
- [9] Z Abedjan, P Schulze, and F Naumann. Dfd: Efficient functional dependency discovery. In *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management*, pages 949–958, 2014.
- [10] P A Flach and I Savnik. Database dependency discovery: a machine learning approach. *AI communications*, 12(3):139–160, 1999.
- [11] S Lopes, J M Petit, and L Lakhal. Efficient discovery of functional dependencies and armstrong relations. In *International Conference on Extending Database Technology*, pages 350–364. Springer, Berlin, Heidelberg, 2000.
- [12] S Kruse and F Naumann. Efficient discovery of approximate dependencies. *Proceedings of the VLDB Endowment*, 11(7):759–772, 2018.
- [13] Thorsten Papenbrock, Jens Ehrlich, Jannik Marten, Tommy Neubert, Jan-Peer Rudolph, Martin Schönberg, Jakob Zwiener, and Felix Naumann. Functional dependency discovery: An experimental evaluation of seven algorithms. *Proc. VLDB Endow.*, 8(10):1082–1093, 2015.
- [14] Yunjia Zhang, Zhihan Guo, and T Rekatsinas. A statistical perspective on discovering functional dependencies in noisy data. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, pages 861–876. New York: ACM Press, 2020.
- [15] P Mandros, M Boley, and J Vreeken. Discovering reliable approximate functional dependencies. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 355–363, 2017.
- [16] C Wyss, C Giannella, and E Robertson. Fastfids: A heuristic-driven, depth-first algorithm for mining functional dependencies from relation instances extended abstract. In *International Conference on Data Warehousing and Knowledge Discovery*, pages 101–110. Springer, Berlin, Heidelberg, 2001.
- [17] T Papenbrock and F Naumann. A hybrid approach to functional dependency discovery. In *Proceedings of the 2016 International Conference on Management of Data*, pages 821–833, 2016.
- [18] I F Ilyas, V Markl, P Haas, et al. Cords: Automatic discovery of correlations and soft functional dependencies. In *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, pages 647–658. New York: ACM Press, 2004.
- [19] Banjo: Bayesian network inference with java objects. <https://user.cs.duke.edu/~amink/software/banjo/>. Accessed: 2023.9.7.