

FD_Mine: Discovering Functional Dependencies in a Database Using Equivalences

Hong Yao, Howard J.Hamilton, and Cory J.Butz

Department of Computer Science, University of Regina
 Regina, SK, Canada, S4S 0A2
 {yao2hong, hamilton, butz}@cs.uregina.ca

Abstract

The discovery of FDs from databases has recently become a significant research problem. In this paper, we propose a new algorithm, called *FD_Mine*. *FD_Mine* takes advantage of the rich theory of FDs to reduce both the size of the dataset and the number of FDs to be checked by using discovered equivalences. We show that the pruning does not lead to loss of information. Experiments on 15 UCI datasets show that *FD_Mine* can prune more candidates than previous methods.

1 Introduction

This paper proposes a new method for finding functional dependencies in data. Formally, let r be a relation on schema R , with X and Y subsets of R . Relation r satisfies the *functional dependency* (FD) $X \rightarrow Y$, if, for any two tuples t_1 and t_2 in r , whenever $t_1[X] = t_2[X]$ then $t_1[Y] = t_2[Y]$.

FDs play an important role in relational theory and relational database design [3]. Current research is based on the fact that FDs may exist in a dataset that are independent of the relational model of the dataset. It is useful to discover these FDs. For example, from a database of chemical compounds, it is valuable to discover compounds that are functionally dependent on a certain structure attribute [2]. As a result, the discovery of FDs from database has recently become a popular research problem [1, 2, 4, 6, 7].

The remainder of this paper organized as follows. A statement of the problem is given in section 2. In section 3, the relationship among FDs is analyzed. The *FD_Mine* algorithm is presented in section 4. Next, the experimental results are shown in section 5. Finally, conclusions are drawn in section 6.

2 Problem Statement

Early methods for discovering of FDs were based on repeatedly sorting and comparing tuples to determine

whether or not these tuples meet the FD definition. For example, in Table 2.1, the tuples are first sorted on attribute A , then each pair of tuples that have the same value on attribute A is compared on attribute B , C , D , and E , in turn, to decide whether or not $A \rightarrow B$, $A \rightarrow C$, $A \rightarrow D$, or $A \rightarrow E$ holds. Then the tuples are sorted on attribute B and so on, until $BCDE$ has been checked. In this paper, we say a FD is *checked* if data is used to examine whether or not a FD holds. Otherwise, we say a FD is *not checked*. A *candidate* is a combination of attributes over a dataset. All candidates of five attributes are represented in Figure 2.1 This approach is inefficient because of the extra sorting and because it needs to examine every value of the candidate attributes to decide whether or not a FD holds. As a result, this approach is highly sensitive to the number of tuples and attributes. It is impractical for a large dataset.

	A	B	C	D	E
t_1	0	0	0	1	0
t_2	0	1	0	1	0
t_3	0	2	0	1	2
t_4	0	3	1	1	0
t_5	4	1	1	2	4
t_6	4	3	1	2	2
t_7	0	0	1	1	0

Table 2.1 An example dataset.

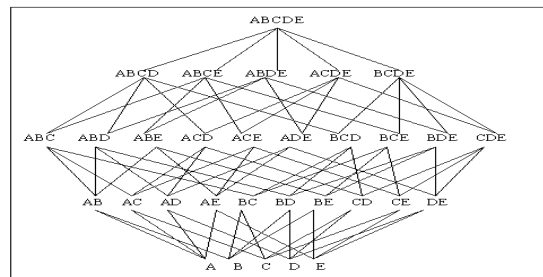


Figure 2.1 Lattice for 5 attributes.

Recent papers have proposed algorithms that do not sort on any attribute or compare any values. Mannila et al. [4, 5] introduced the concept of a *partition*, which places tuples that have the same values for an attribute into the

same group. The problem of determining whether or not a FD holds on a given dataset can be addressed by comparing the number of groups among the partitions for various attributes. For *dataset r*, the data over the relational schema R, shown in Table 2.1, the partition for attribute A can be denoted as $\Pi_A(r) = \{\{t_1, t_2, t_3, t_4, t_7\}, \{t_5, t_6\}\}$. The partition for the attribute combination AD for Table 2.1 is $\Pi_{AD}(r) = \{\{t_1, t_2, t_3, t_4, t_7\}, \{t_5, t_6\}\}$. The *cardinality of the partition* $|\Pi_A(r)|$, which is the number of groups in partition Π_A , is 2, and $|\Pi_{AD}(r)|$ is 2 as well. Because $|\Pi_A(r)|$ is equal to $|\Pi_{AD}(r)|$, $A \rightarrow D$ can be obtained [2].

Our research addresses two related questions. First, can other information from discovered FDs be used to prune more candidates than previous approaches? Secondly, can this pruning be done so that the overall efficiency of the algorithm is improved? We address both these problems by further considering the theoretical properties of FDs, formulating algorithm FD_Mine, and testing it on a variety of datasets.

3 Properties of Functional Dependencies

In this section, the relationships among FDs are analyzed with particular attention to equivalent candidates and nontrivial closure.

Definition 3.1 Let X and Y be candidates over a dataset D, if $X \rightarrow Y$ and $Y \rightarrow X$ hold, then X and Y are said to be *equivalent candidates*, denoted as $X \leftrightarrow Y$.

Using Definition 3.1 and the Armstrong axioms [3], Lemma 3.1 and Lemma 3.2 can be obtained.

Lemma 3.1. Let X, Y and Z be candidates over D. If $X \leftrightarrow Y$ and $XW \rightarrow Z$ hold, then $YW \rightarrow Z$ holds.

Lemma 3.2. Let X, Y and Z be candidates over D. If $X \leftrightarrow Y$ and $WZ \rightarrow X$ hold, then $WZ \rightarrow Y$ holds.

Using Lemmas 3.1 and 3.2, the number of candidates and possibly the size of the dataset that needs to be checked can be reduced.

Example 3.1. In Table 2.1 the FDs $A \rightarrow D$ and $D \rightarrow A$ were discovered first. According to Definition 3.1, $A \leftrightarrow D$ holds. By examining all entries for attribute A, B, and C, we can determine that $AB \rightarrow C$ and $BC \rightarrow A$ hold. Without Lemma 3.1 and 3.2, we would need to examine all entries in Table 2.1 again to determine whether or not $BD \rightarrow C$ and $BC \rightarrow D$ hold. But with Lemma 3.1, $BD \rightarrow C$ can be inferred, and with Lemma 3.2, $BC \rightarrow D$ can also be inferred. Once the equivalence between A and D has been discovered, attribute D and all its values are redundant to further searching for FDs. FDs involving D can be inferred instead of being determined by checking the data. The result after removing D is shown in Table 3.1.

Definition 3.2 Let F be a set of FDs over a dataset D and X be a candidate over D. The closure of candidate X with respect to F, denoted $\text{Closure}(X)$, is defined as $\{Y \mid X \rightarrow Y \text{ can be derived from F by the Armstrong axioms}\}$.

The *nontrivial closure* of candidate X with respect to F, denoted $\text{Closure}'(X)$, is defined as $\text{Closure}'(X) = \text{Closure}(X) - X$.

Lemma 3.3. Let X and Y be two candidates of dataset D, $Z = X \cap Y$. If $\text{Closure}'(X) \supseteq Y - Z$, and $\text{Closure}'(Y) \supseteq X - Z$, then $X \leftrightarrow Y$.

According to the definition of an FD and the Armstrong axioms, the following properties can be inferred.

Property 3.1. Let X and Y be candidates over a dataset. Then $\text{Closure}'(X) \cup \text{Closure}'(Y) \subseteq \text{Closure}'(XY)$ holds.

Property 3.2. Let R be a relational schema and X be a candidate of R over a dataset D. If $X \cup \text{Closure}'(X) = R$, then X is a key.

The downward closure property for itemsets can be described as follow: if any subset of size (k-1) of a k-itemset is not frequent then the k-itemset is also not frequent. A similar property exists concerning the FDs that need to be checked.

Definition 3.3. Let $X_1, X_2, \dots, X_k, X_{k+1}$ be (k+1) attributes over a dataset D. If a $X_1X_2\dots X_k \rightarrow X_{k+1}$ is a FD with k attributes on its left hand side, then it is called a *k-level FD*.

Property 3.3. [2,3] If $X_1X_2\dots X_{k-1}X_k \rightarrow X_{k+1}$ is a k-level FD, then it needs to be checked when none of its (k-1)-level subsets $X_{i(1)}X_{i(2)}\dots X_{i(k-1)} \subset X_1X_2\dots X_{k-1}X_k$ satisfies $X_{i(1)}X_{i(2)}\dots X_{i(k-1)} \rightarrow X_{i(k)}$.

	A	B	C	E
t_1	0	0	0	0
t_2	0	1	0	0
t_3	0	2	0	2
t_4	0	3	1	0
t_5	4	1	1	4
t_6	4	2	1	2
t_7	0	0	1	0

Table 3.1 Redundant Attribute D is removed from Table 2.1

4 The FD_Mine Algorithm

The FD_Mine algorithm uses the above properties to prune the dataset and the candidates.

The FD_Mine uses a level-wise search, where results from level k are used to explore level k+1. First, at level 1, all FDs $X \rightarrow Y$, where X and Y are single attributes, are found and stored in FD_SET F_1 . The set of candidates that are considered at this level is denoted L_1 . F_1 and L_1 are used to generate candidates $X_i X_j$ of L_2 . At level 2, all FDs of the form $X_i X_j \rightarrow Y$ are found and stored in FD_SET F_2 . F_1, F_2, L_1 , and L_2 are used to generate the candidates of L_3 , and so on, until no candidates remain, i.e., $L_k = \emptyset$ ($k \leq n-1$).

Before introducing the FD_Mine algorithm, the following identifiers are introduced.

- *CANDIDATE_SET*: a set of candidates.
- *FD_SET*: the set of discovered functional dependencies, each in the form $X \rightarrow Y$.
- *EQ_SET*: the set of discovered equivalences, each in the form $X \leftrightarrow Y$.
- *KEY_SET*: the set of discovered keys.

Algorithm FD_Mine

To discover all functional dependencies in a dataset.

Input: Dataset D and its attributes X_1, X_2, \dots, X_m

Output: *FD_SET*, *EQ_SET* and *KEY_SET*

1. Initialization Step

set $R = \{X_1, X_2, \dots, X_m\}$, set *FD_SET* = ϕ ,
 set *EQ_SET* = ϕ , set *KEY_SET* = ϕ
 set *CANDIDATE_SET* = $\{X_1, X_2, \dots, X_m\}$
 for all $X_i \in \text{CANDIDATE_SET}$ do
 set $\text{Closure}'[X_i] = \phi$

2. Iteration Step

while *CANDIDATE_SET* $\neq \phi$ do
 for all $X_i \in \text{CANDIDATE_SET}$ do
 ComputeNonTrivialClosure(X_i)
 ObtainFDandKey(X_i)
 ObtainEQSet(*CANDIDATE_SET*)
 PruneCandidates(*CANDIDATE_SET*)
 GenerateCandidates(*CANDIDATE_SET*)
 3. Display(*FD_SET*, *EQ_SET*, *KEY_SET*)

Procedure **ComputeNonTrivialClosure**(X_i)

for each $Y \subset R - X_i - \text{Closure}'[X_i]$ do
 if $(|\Pi_{X_i}| = |\Pi_{X_i Y}|)$ add Y to $\text{Closure}'[X_i]$

Procedure **ObtainFDandKey** (X_i)

add $X_i \rightarrow \text{Closure}'[X_i]$ to *FD_SET*
 if $(R = X_i \cup \text{Closure}'[X_i])$ add X_i to *KEY_SET*

Procedure **ObtainEQSet**(*CANDIDATE_SET*)

for each $X_i \in \text{CANDIDATE_SET}$ do
 for all $X \rightarrow \text{Closure}'(X) \in \text{FD_SET}$ do
 set $Z = X \cap X_i$
 if $(\text{Closure}'(X) \supseteq X_i - Z \text{ and } \text{Closure}'[X_i] \supseteq X - Z)$
 add $X \leftrightarrow X_i$ to *EQ_SET*

Procedure **PruneCandidates**(*CANDIDATE_SET*)

for each $X_i \in \text{CANDIDATE_SET}$ do
 if $\exists X_j \in \text{CANDIDATE_SET}$ and $X_j \leftrightarrow X_i \in \text{EQ_SET}$
 delete X_i from *CANDIDATE_SET*
 if $\exists X_i \in \text{KEY_SET}$ then
 delete X_i from *CANDIDATE_SET*

Procedure **GenerateCandidates**(*CANDIDATE_SET*)

for each $X_i \in \text{CANDIDATE_SET}$ do
 for each $X_j \in \text{CANDIDATE_SET}$ and $i < j$ do
 if $(X_i[1]=X_j[1], \dots, X_i[k-2] = X_j[k-2], X_i[k-1] < X_j[k-1])$
 set $X_{ij} = X_i \text{ join } X_j$
 if $\exists X_i \rightarrow X_j[k-1] \notin \text{FD_SET}$
 compute the partition $\Pi_{X_{ij}}$ of X_{ij}
 set $\text{Closure}'(X_{ij}) = \text{Closure}'(X_i) \cup \text{Closure}'(X_j)$
 if $(R = X_{ij} \cup \text{Closure}'[X_{ij}])$ add X_{ij} to *KEY_SET*
 else add X_{ij} to *CANDIDATE_SET*
 delete X_i from *CANDIDATE_SET*

Example 4.1: Suppose that FD_Mine is applied to dataset D, as shown in Table 2.1, with $R = \{A, B, C, D, E\}$.

Level 1				Level 2			
Candidate X	\Pi _X	Closure'(X)	FD	Candidate X	\Pi _X	Closure'(X)	FD
A	2	D	A → D	AB	6	E	AB → E
B	4	φ		AC	3	φ	
C	2	φ		AE	5	φ	
D	2	A	D → A	BC	6	φ	
E	4	φ		BE	6	A	BE → A
				CE	6	A	CE → A
FD_SET = {A → D, D → A}				FD_SET = {AB → E, BE → A, CE → A}			
EQ_SET = {(A, D)}				EQ_SET = {(AB, BE)}			
PrunedSet = {A, B, C, E}				PrunedSet = {AB, AC, AE, BC, CE}			
Next Level Candidates: {AB, AC, AE, BC, BE, CE}				KEY_SET = {ABC}			
				Next Level Candidates: ()			
FD_SET = {A → D, D → A, AB → E, DB → E, BE → A, BE → D, CE → A, CE → D}							
SAME_SET = {(A, D), (AB, BE)}							
KEY_SET = {ABC, BCE}							

Table 4.1 Trace of FD_Mine Applied to the Table 2.1 Dataset

Table 4.1 summarizes the actions of FD_Mine. In iteration 1, since $|\Pi_A| = |\Pi_{AD}| = 2$, $\text{Closure}'(A)$ is set to D, and $A \rightarrow D$ is deduced. In same way, $D \rightarrow A$ is discovered, so the equivalence $A \leftrightarrow D$ is obtained. As a result, we only need to combine A, B, C, and E to generate the next level candidates $\{AB, AC, AE, BC, BE, CE\}$. At the same time, the nontrivial closure of each generated candidate is computed. For example, the closure $(AB) = \text{closure}'(A) \cup \text{closure}'(B) = \{D\} \cup \phi = \{D\}$. In iteration 2, for candidate AB, only $AB \rightarrow C$ and $AB \rightarrow E$ need to be checked, because $R - \{A, B\} - \text{Closure}'(AB) = \{A, B, C, D, E\} - \{A, B\} - \{D\} = \{C, E\}$. Since $|\Pi_{AB}| = |\Pi_{ABE}| = 6$, then $AB \rightarrow E$ is obtained. In the same way, at this level, $BE \rightarrow A$ and $CE \rightarrow A$ are also discovered, so the equivalence $AB \leftrightarrow BE$ is obtained. As a result, we only need to combine AB, AC, AE, BC, and CE to form the level 3 candidates, which are $\{ABC, ACE\}$. Since $CE \rightarrow A$, ACE is pruned by property 3.3. Since $AB \rightarrow E$, then $ABC \rightarrow E$. Since $A \leftrightarrow D$, then $ABC \rightarrow D$, so ABC is a key, and ABC is also pruned by property 3.2. No other candidate remains, so the algorithm halts.

5 Experimental Results

FD_Mine was applied to fifteen datasets, obtained from the UCI Machine Learning Repository [8] and the results were compared to TANE [2]. TANE was selected for comparison because it establishes the theoretical framework for the problem. For the dataset given in Table 2.1, Figure 5.1(a) shows the semi-lattice for FD_Mine, and Figure 5.1(b) shows that for TANE. Each node represents a combination of attributes. If an edge is shown between nodes X and XY, then $X \rightarrow Y$ needs to be checked. Hence, the number of edges is the number of FDs that need to be checked. Both semi-lattices shown in Figure 5.1 have fewer edges than the lattice shown in Figure 2.1. In addition, the semi-lattice for FD_Mine has fewer edges than that for TANE.

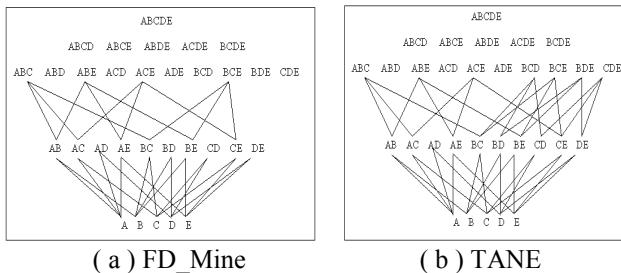


Figure 5.1 Semi-lattices for the Data in Table 2.1

Table 5.1 compares the number of FDs that are checked on data by FD_Mine and TANE for 15 UCI datasets. Figure 5.2 shows more detailed results for the Imports-85 dataset. At levels 1 through 5, both algorithms check approximately the same number of FDs, but at levels 6 through 11, FD_Mine checks fewer FDs than TANE, because it prunes more unnecessary candidates than TANE by using the equivalences and FDs discovered at previous levels. For more results, see [9].

Dataset Name	# of Attr.	# of Rows	Total # of FDs Checked	
			FD_MINE	TANE
Abalone	8	4177	594	594
Balance-scale	5	625	70	70
Breast-cancer	10	191	5,095	5,095
Bridge	13	108	15,397	15,626
Cancer-Wisconsin	10	699	4,562	4,562
Chess	7	28,066	434	434
Crx	16	690	79,418	130,605
Echocardiogram	13	132	2,676	2,766
Glass	10	142	405	455
Hepatitis	20	155	1,161,108	1,272,789
Imports-85	26	205	2,996,737	3,564,176
Iris	5	150	70	70
Led	8	50	477	477
Nursery	9	12,960	2,286	2,286
Pendigits	17	7,494	223,143	227,714

Table 5.1 Comparison on UCI Datasets

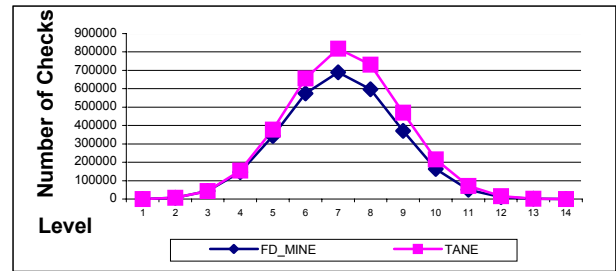


Figure 5.2 Graphical Comparison on Imports-85 Dataset

6 Conclusion

Based on our analysis of the theoretical properties of functional dependencies, equivalences among attributes were identified. Next, the FD_Mine algorithm, which finds functional dependencies in a dataset, is introduced here. Finally, FD_Mine was run on 15 UCI datasets. The results show that the FD_Mine is valuable because it reduces the size of the dataset or the number of checks required, but it does not lead to loss of information or eliminate any valid candidates.

References

- Flach, P. A., and Savnik, I., Database Dependency Discovery: A Machine Learning Approach. *AI Communications*, 12(3):139-160 1999.
- Huhtala, Y., Kärkkäinen, J., Porkka, P., and Toivonen, H., TANE: An Efficient Algorithm for Discovering Functional and Approximate Dependencies. *The Computing Journal*, 42(2):100-111 1999.
- Maier, D., *The Theory of Relational Databases*, Computer Science Press, 1983.
- Mannila, H., and Räihä, K.J., Algorithms for Inferring Functional Dependencies from Relations. *Data and Knowledge Engineering*, 12(1):83-99 1994.
- Mannila, H., and Toivonen, H., Levelwise Search and Borders of Theories in Knowledge Discovery. *Data Mining and Knowledge Discovery*, 1(3):241-258 1997.
- Novelli, N., and Cicchetti, R., Functional and Embedded Dependency Inference: A Data Mining Point of View. *Information Systems*, 26(7):477-506 2001.
- Roddick, J.F., Craske, N.G., Richards, T. J., Handling Discovered Structure in Database Systems. *IEEE Transactions on Knowledge and Data Engineering*, 8(2): 227-240 1996.
- UCI Machine Learning Repository, <http://www1.ics.uci.edu/~mllearn/MLRepository.html>
- Yao, H., Hamilton, H. J. and Butz, C. J., FD_MINE: Discovering Functional Dependencies in a Database Using Equivalences, University of Regina, Computer Science Department, Technical Report CS-02-04, August, 2002, ISBN 0-7731-0441-0.