Algorithm for Discovering Multivalued Dependencies

Men Hin Yan
Chinese University of Hong Kong
mhyan@cse.cuhk.edu.hk

Ada Wai-chee Fu
Chinese University of Hong Kong
adafu@cse.cuhk.edu.hk

1. INTRODUCTION

While extracting functional dependencies has received considerable attention [6, 3, 4, 5], relatively less research effort has been put in finding multivalued dependencies. Based on the previously proposed techniques of discovering functional dependencies [3], we propose a new algorithm for finding multivalued dependencies from a large database.

We assume the usual interpretation of a relation (or table) in the relational database model where no duplicate tuples are allowed. The definition of multivalued dependency is given as below:

Let R be a relation schema and let $X = X_1, X_2, ..., X_n$ be a subset of R , let $Y = Y_1, Y_2, ..., Y_n$ be a subset of R and let $Z = R - Y - X$. The **multivalued dependency** $X \rightarrow\rightarrow Y$ holds in R if, in any legal relation $r(R)$, for all pairs of tuples t_1 and t_2 in r such that $t_1[X] = t_2[X]$, there exists tuples t_3 and t_4 in r such that: $t_1[X] = t_2[X] = t_3[X] = t_4[X]; t_3[Y] = t_1[Y]; t_3[Z] = t_2[Z]; t_4[Y] = t_2[Y]; t_4[Z] = t_1[Z]$.

Given a relation schema R, a multivalued dependency $X \rightarrow\rightarrow Y$ is said to be ***non-minimal*** if there exists a multivalued dependency $X' \rightarrow\rightarrow Y$ where X' is a proper subset of X; $X \rightarrow\rightarrow Y$ is said to be ***trivial*** if $Y \subset X$ or $X \cup Y = R$. It is obvious that in order to have a *non-trivial* multivalued dependency $X \rightarrow\rightarrow Y$, all X, Y and $Z = R - X - Y$ cannot be $\emptyset$. We aim to discover minimal non-trivial multivalued dependencies from the data.

1.1 Partitions and Multivalued Dependencies

A new method for determining if a functional dependency holds or not was proposed by [3]. The method is based on representing attribute sets by equivalent class partitions of the set of tuples. We find that with suitable modification, the representation is also useful in discovering multivalued dependencies.

For a relation schema R, given a relation (or table) r, two rows (or tuples) t and u are **equivalent** with respect to a given set $X \subset R$ if attributes $t[A] = u[A]$ for all $A \in X$. Any attribute set X partitions the tuples of the relation into **equivalence classes**. We denote the equivalence class of a tuple $t \in r$ with respect to a given set $X \subseteq R$ by $[t]_X$, i.e. $[t]_X = \{u \in r \mid t[A] = u[A] \text{ for all } A \in X\}$. The set $\pi_X = \{[t]_X \mid t \in r\}$ of equivalence classes is a **partition** of r under X. That means π_X is a collection of disjoint sets (equivalence classes) of tuples, such that each set has a unique value for the attribute set X, and the union of the sets equals the relation r.

We summarize below some of the notations that we use:

R	A given relation schema
r	A given relation over R
$t[A]$	The value of attribute A in tuple t
$[t]_X$	Equivalence class of tuple t with respect to $X \subseteq R$
π_X	Partition of r under X under X
X_j	jth equivalence class in π_X
$\theta(Y, X_j)$	Number of Y-values in equivalence class X_j

Tuple ID	A	B	C	D
1	1	1	2	3
2	1	2	1	4
3	1	2	2	2
4	1	1	1	4
5	1	1	2	2
6	1	2	2	3
7	2	3	1	3
8	2	3	2	4

Table 1: An example relation

For example, consider the above relation. Attribute A has value 1 for tuples 1 to tuples 6, so they form an equivalence class $\{1, 2, 3, 4, 5, 6\}$ (here we use tuple identifiers to denote tuples). Attribute A has value 2 in tuple 7 and tuple 8, so they form another equivalence class $\{7, 8\}$. The whole partition with respect to A is $\pi_{\{A\}} = \{\{1, 2, 3, 4, 5, 6\}, \{7, 8\}\}$. The partitions for other attributes are $\pi_{\{B\}} = \{\{1, 4, 5\}, \{2, 3, 6\}, \{7, 8\}\}$, $\pi_{\{C\}} = \{\{1, 3, 5, 6, 8\}, \{2, 4, 7\}\}$ and $\pi_{\{D\}} = \{\{1, 6, 7\}, \{2, 4, 8\}, \{3, 5\}\}$ respectively. The partition with respect to $\{CD\}$ is $\pi_{\{CD\}} = \{\{1, 6\}, \{2, 4\}, \{3, 5\}, \{7\}, \{8\}\}$.

1.1.1 *Partition Refinement*

A partition π is a **refinement** of another partition π' if every equivalence class in π is a subset of some equivalence class of π'. For example, consider the relation in Table 1. Let t_i be the tuple with Tuple ID $= i$. $\pi_{\{CD\}}$ refines $\pi_{\{A\}}$ since each equivalence class in $\pi_{\{CD\}}$ is totally contained by

CIKM '01 Atlanta, Georgia USA

some equivalence class in $\pi_{\{A\}}$. On the other hand, $\pi_{\{CD\}}$ dose not refine $\pi_{\{B\}}$ since some equivalence classes in $\pi_{\{CD\}}$ are not contained in any equivalence class in $\pi_{\{B\}}$. For instance, $[t_1]_{\{CD\}} = \{1, 6\}$ in $\pi_{\{CD\}}$ is not contained in any equivalence class in $\pi_{\{B\}}$.

It is easy to see that the partitions can be computed as a product of two previously computed partitions. As shown in [3], the **product** of two previously computed partitions π' and π'', denoted by $\pi' \cdot \pi''$, is the least refined partition π that refines both π' and π'': For all $X, Y \subseteq R$, $\pi_X \cdot \pi_Y = \pi_{\{X \cup Y\}}$.

2. VALIDATION OF MULTIVALUED DEPENDENCIES

According to [3], a functional dependency $X \rightarrow Y$ holds if and only if π_X refines π_Y. Thus the concept of partition refinement gives almost direct functional dependencies. We find that we can also make use of the idea of partition refinement to find multivalued dependencies. With the concept of partition, each X-value actually forms an equivalence class. Thus we can see the validation of multivalued dependency in the way below:

Assume there are no duplicated tuples in the relation over the schema R with attribute sets X, Y, and Z (Z represents attributes in R other than X and Y, and $Z \neq \emptyset$). Arrange the equivalence classes in a partition by the smallest tuple ID in each equivalence class in ascending order. E.g. $\pi_{\{AB\}}$ is sorted as $\{\{1, 4, 5\}, \{2, 3, 6\}, \{7, 8\}\}$. We use X_j to represent the jth equivalence class in π_X. E.g. for $\pi_{\{AB\}}$, $\{AB\}_1 = [t_1]_{\{AB\}} = \{1, 4, 5\}$, $\{AB\}_2 = [t_2]_{\{AB\}} = \{2, 3, 6\}$, while $\{AB\}_3 = [t_7]_{\{AB\}} = \{7, 8\}$.

Lemma 1 *Given a relation schema R with attributes X, Y and $Z = R - X - Y$. The multivalued dependency $X \rightarrow\rightarrow Y$ holds in R iff for any valid relation r, for every equivalence class X_j in π_X, the number of different Y-values, $\theta(Y, X_j)$, times the number of different Z-values, $\theta(Z, X_j)$, equals to the size of X_j, $\mid X_j \mid$, i.e.*

$$\theta(Y, X_j) * \theta(Z, X_j) = \mid X_j \mid$$

$\theta(Y, X_j)$ for each equivalence class X_j in π_X can be computed by comparing π_X and π_{XY}. *By checking how many equivalence classes in π_{XY} refine (are the subsets of) X_j*, we can obtain the $\theta(Y, X_j)$ for X_j. $\theta(Z, X_j)$ can be computed in the same way.

3. SEARCH STRATEGY AND PRUNING

The general search strategy for our algorithm is as follows. The search starts from singleton sets of right-hand side candidates for a multivalued dependency, and works its way to larger attribute sets of right-hand side candidates. For each right-hand side candidate, it is first validated with singleton sets of left-hand side candidates, and works its way to larger attribute sets of left-hand side candidates.

Similar to previous work in discovering functional dependencies, our focus is also to find only the **minimal non-trivial** multivalued dependencies by pruning the search space as much as possible. We find that there is some similarity between the properties of left-hand sides of functional dependency and those of multivalued dependency. Due to this similarity, we found that the small-to-large searching concept in previous functional dependency discovery algorithm can be applied in discovering multivalued dependencies as well.

However, there is a crucial difference between the properties of left-hand sides of functional dependency and multivalued dependency which results in the increase of the search space for finding multivalued dependencies. We try to reduce the search space based on a unique property of multivalued dependency. The search strategies for both left-hand and right-hand sides candidates are discussed in the following sections.

3.1 Search Strategy for Left-hand Sides Candidates

For functional dependency, if $X \rightarrow Y$ holds then $XZ \rightarrow Y$ is also valid where Z represents some attributes in R other than X and Y. The dependency like $XZ \rightarrow Y$ above is not the minimal dependency found in the relation. Existing algorithms for finding functional dependencies try to avoid considering those non-minimal left-hand side candidates, and we considered the possibility to do that in our algorithm for finding multivalued dependencies as well.

For multiple dependency if $X \rightarrow\rightarrow Y$ holds, $XZ \rightarrow\rightarrow Y$ would also be valid where Z represents some attributes in R other than X and Y. This minimality rule is a special case for *multivalued augmentation rule* [1]. The rule states that if $\alpha \rightarrow\rightarrow \beta$ holds and $\gamma \subseteq R$ and $\delta \subseteq \gamma$, then $\gamma\alpha \rightarrow\rightarrow \delta\beta$ holds. When substituting δ with $\emptyset$, we can get the rule we just stated. With this property, we can prune the search space of left-hand side candidate as follows.

Rule 1 *Once we find a valid multivalued dependency $X \rightarrow\rightarrow Y$, we do not have to further validate any left-hand side which is a superset of X for a candidate multivalued dependency with Y as the right-hand side.*

3.2 Search Strategy for Right-hand Sides Candidates

For functional dependency, given $X \rightarrow A$ and $X \rightarrow B$ are valid dependencies, we can be sure that $X \rightarrow AB$ holds. Given $X \rightarrow AB$, we can be sure that $X \rightarrow A$ and $X \rightarrow B$ hold as well. The dependency like $X \rightarrow AB$ above is also called a non-minimal dependency, since obviously it can be easily deduced from $X \rightarrow A$ and $X \rightarrow B$. Existing algorithms for finding functional dependencies tried to avoid those non-minimal candidates by just considering the singleton sets as the possible right-hand side candidates.

Unfortunately, the above property for functional dependency is not always true for multivalued dependency, i.e. if $X \rightarrow\rightarrow AB$ holds, it may not be true that $X \rightarrow\rightarrow A$ and $X \rightarrow\rightarrow B$ hold as well. For instance, in the relation of Table 1, we can see that $A \rightarrow\rightarrow CD$ holds but neither $A \rightarrow\rightarrow C$ nor $A \rightarrow\rightarrow D$ holds. In this case, $A \rightarrow\rightarrow CD$ is actually a minimal non-trivial multivalued dependency. As a result, other than singleton sets, we have to also consider larger attribute sets as the possible right-hand side candidates.

Let us go back to the idea about validating multivalued dependencies introduced in Section *1.1.1*. When we consider a possible dependency $X \rightarrow\rightarrow Y$, our validation must involve Z (Z equals $R - Y - X$). It is obvious that in order to have a valid *non-trivial* multivalued dependency $X \rightarrow\rightarrow Y$, all X, Y and Z cannot be $\emptyset$. For R with k attributes, the right-hand side candidates Y would be the largest when X and Z are singleton sets. Thus the largest Y would be the $(k-2)$-attribute sets, and the search space is expected to contain from singleton sets to all sets with $k - 2$ attributes.

Using Figure 1 as an example, where *level* refers to the number of attributes in an attribute set, we have to consider all the sets in level 1 and 2 (indicated by larger fonts) as right-hand side candidates only.

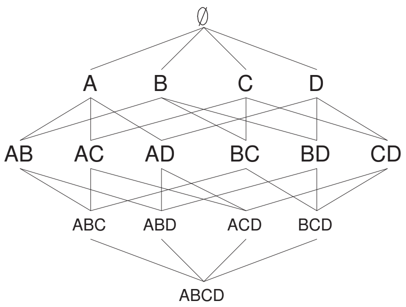

Figure 1: A set containment lattice for $R = \{A, B, C, D\}$

However, by applying the ***complementation rule*** [1] of multivalued dependency, we can actually reduce the search space to less than half. The complementation rule states that if $X \rightarrow\rightarrow Y$ holds, then $X \rightarrow\rightarrow Z$ (Z equals $R - Y - X$) also holds. From this rule, it is obvious that $X \rightarrow\rightarrow Z$ holds iff $X \rightarrow\rightarrow Y$ holds. Thus while we are validating $X \rightarrow\rightarrow Y$, we are actually validating $X \rightarrow\rightarrow Z$ at the same time. The size of Z is the largest when that of X is the smallest, i.e. when X is a singleton set. As a result, we do not have to consider any attribute set as a possible Y (right-hand side candidate) if that attribute set has already acted as a Z before. We apply the above idea to our search strategy as follows.

Rule 2 *A multivalued dependency candidate $X \rightarrow\rightarrow Y$ will not be validated if the level (size) of Y is greater than that of $Z = R - X - Y$.*

Using Figure 1 as an example where the number of attributes $k = 4$. First we consider a singleton set $\{A\}$, which is at level 1, as the Y-candidate. We consider the $X = \{B\}$. As a result, the corresponding Z is $\{CD\}$, which is in level 2. Since the level of the Y-candidate, $\{A\}$, is smaller than that of the corresponding Z, $\{CD\}$, the validation of the dependency $B \rightarrow\rightarrow A$ is performed. By the complementation rule, it is clear that we have also validated the dependency $B \rightarrow\rightarrow CD$ at the same time. Later the validation will come to take $\{CD\}$ as the Y-candidate, $\{B\}$ as X and $\{A\}$ as the corresponding Z. Since the level of the Y-candidate, $\{CD\}$, is larger than that of the corresponding Z, $\{B\}$, the validation of the dependency $B \rightarrow\rightarrow CD$ will not be performed.

For the case of having candidate Y and the corresponding Z in the same level, we apply a similar strategy to avoid redundant validation.

Rule 3 *Assume a total order on the attributes (e.g. alphabetical order). Consider a multivalued dependency candidate $X \rightarrow\rightarrow Y$ where Y at the same level (has the same size) as $Z = R - X - Y$. The dependency candidate will be validated iff the smallest attribute (in alphabetical order) of Y is smaller than that of Z.*

We also have other pruning strategies, which are summarized into the following rules. The details are omitted here for the interest of space.

Rule 4 *During validation, if the left-hand side candidate is identified as a key or superset of a key, we do not have to further verify it with our multivalued dependency validation.*

Rule 5 *Given a relation of schema R and a multivalued dependency candidate $X \rightarrow\rightarrow Y$ and $Z = R - Y - X$. if either $X \rightarrow Y$ or $X \rightarrow Z$ is identified as a functional dependency, we do not have to further verify candidate $X \rightarrow\rightarrow Y$ with our multivalued dependency validation.*

Rule 6 *Given a multivalued dependency candidate $X \rightarrow\rightarrow Y$. When Y is a key, if the number of equivalence classes of π_X is not equal to that of π_{XZ}, i.e. $\mid \pi_X \mid \neq \mid \pi_{XZ} \mid$, then we do not have to perform dependency validation.*

4. CONCLUSION

A new method for discovering multivalued dependency is introduced in this paper. The idea of tuples partition has been used for discovering functional dependencies. We have further extended the idea of partitioning in finding multivalued dependencies. In our algorithm, we make use of the properties of multivalued dependency as well as the relation between functional and multivalued dependencies and propose possible search strategies and pruning methods. Experimental results show that our algorithm, can have nearly linear scalability over the number of tuples and it can be applicable to real life databases with a large amount of tuples. More details can be found in [7].

Acknowledgements

We thank the authors of [3] who have kindly provided us the source code of TANE for discovering functional dependencies. All the datasets we used in our experiments are obtained from the UCI Machine Learning Repository [2]. We thank the anonymous referees for their helpful comments. This research is supported by the RGC Earmarked Research Grant CUHK 4436/99E.

5. REFERENCES

[1] C. Beeri, R. Fagin, and J. Howard. A complete axiomatization for functional and multivalued dependencies. In *Proc. of the ACM SIGMOD Conf.*, pages 47–61, 1977.

[2] C. Blake and C. Merz. UCI repository of machine learning databases, 1998. University of California, Irvine, Dept. of Information and Computer Sciences.

[3] Y. Huhtala, J. Käkkäinen, P. Porkka, and H. Toivonen. Efficient discovery of functional and approximate dependencies using partitions. In *Proceedings of 14th International Conference on Data Engineering (ICDE'98)*, pages 392–401, 1998.

[4] H. Mannila and K. Räihä. *The Design of Relational Database*. Addison-Wesley, Menlo Park, CA, 1992.

[5] H. Mannila and K. Räihä. On the complexity of inferring functional dependencies. In *Discrete Applied Mathematics*, volume 40, pages 237–243, 1992.

[6] H. Mannila and K. Räihä. Algorithms for inferring functional dependencies. In *Data Knowledge Engineering*, 1994.

[7] M. H. Yan. *From XML to Relational Database*. Mphil Thesis, Chinese University of Hong Kong, Dept. of Computer Science, 2001.