IADIS International Journal on Computer Science and Information Systems
Vol. 1, No. 2, pp. 117-131
ISSN: 1646-3692

DATA CLEANING USING FD FROM DATA MINING PROCESS

Kollayut Kaewbuadee *Department of Computer Science.Thammasat University, Thailand*

Yaowadee Temtanapat *Department of Computer and Information Science. King Mongkut's Institute of Technology North Bangkok, Thailand*

Ratchata Peachavanish *Department of Computer Science. Thammasat University, Thailand*

ABSTRACT

Functional Dependency (FD) is an important feature for referencing to the relationship between attributes and candidate keys in tuples. It also shows the relationship between entities in a data model (Calvanese et al. 2001). In research areas of data cleaning (Arenas et al. 1999; Bohannon et al. 2005), the FD is used for improving the data quality. In a data mining research, an FD discovery technique has been studied (Savnik and Flach 1993; Huhtala et al. 1999). However, an FD discovery could find too many FDs and, if use directly in a cleaning process, could cause it to NP time (Bohannon et al. 2005). In this research, we have developed a cleaning engine by combining an FD discovery technique with data cleaning technique and use the feature in query optimization called "Selectivity Value" to decrease the number of discovered FDs.
Testing results showed that this work can identify duplicates and anomalies with high recall and low false positive.

KEYWORDS

Functional Dependency, Data Cleaning, Functional Dependency Discovery

1. INTRODUCTION

Clean data is crucial for a wide variety of applications in many industries (Erhard and Do 2000). When data has kept increasing in an explosive rate, a task to keep data correct and consistent can be overwhelming. Worse than that main causes of dirty data come from many basic mistakes such as mistaken data entry, missing fields, typos, etc. Although, data in general has some dependency semantics and they usually help to avoid such mistakes, several

times, they are ignored or unaware during database designs or may be dropped for performance improvement.

Researches (Arenas et al. 1999; Bohannon et al. 2005) presented that a functional dependency (FD) is a property in data that has the ability for cleaning dirty data. In general, FDs depend directly on the semantic of a system. However, FDs can be retrieved from data by using a data mining technique (Savnik and Flach 1993; Huhtala et al. 1999; Ilyas et al. 2004). To make automatic cleaning using FDs, we developed a cleaning engine by combining the FD discovery technique to a data cleaning technique.

However, the combining solution is sensitive to data size. When the data increases, it decreases the speed of the discovery algorithm. Moreover, when a number of attributes increases, the discovery creates more candidates of FDs and generates too many FDs including noise ones. The large amount of FDs can degrade the performance of the data cleaning. To decrease the number of generated FDs, we use a query optimization technique, "Selectivity Value" to prune an unlikely FD.

1.1 Basic Background

We revised some basic terms in a relational concept.

Functional Dependency: Formally, let r be a relation of relation schema R, with X and Y are 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 $t_1[Y] = t_2[Y]$ (Garcia-Molina et al. 2001). The set of attributes X is called the *left-hand side* of the FD and Y is called the *right-hand side*.

Partition: For *dataset r*, the data over the relational schema R, a *partition* for attribute A, denoted as $\Pi_A(r)$, is groups of disjoint sets of tuples that are a projection of attribute A. In table 1, for example, $\Pi_A(r) = \{\{t_1, t_2, t_3, t_4, t_7\}, \{t_5, t_6\}\}$ and a partition for the attribute AD is $\Pi_{AD}(r) = \{\{t_1, t_2, t_3, t_4, t_7\}, \{t_5, t_6\}\}$. The cardinality of the partition $|\Pi_A(r)|$ is the number of classes in the partition Π_A. For this example, $|\Pi_A(r)|$ is 2, and $|\Pi_{AD}(r)|$ is 2 also. Because $|\Pi_A(r)|$ is equal to $|\Pi_{AD}(r)|$, $A \rightarrow D$ can be obtained (Huhtala et al. 1999).

Table 1. A sample dataset

	A	B	C	D	E
t_1	a0	b0	c0	d1	e0
t_2	a0	b1	c0	d1	e0
t_3	a0	b2	c0	d1	e1
t_4	a0	b3	c1	d1	e0
t_5	a2	b1	c1	d2	e2
t_6	a2	b3	c1	d2	e3
t_7	a0	b0	c1	d1	e0

Approximate FD: $X \rightarrow Y$ or $e(X \rightarrow Y)$ is a set of tuples containing proportionally less members which if taken this set out will accept FD $X \rightarrow Y$

$e(X \rightarrow Y)$ can be calculated as following: set Equivalence Class c of $\prod_X$ as a union of Equivalence Class c'1, c'2, etc. of $\prod_{X \cup \{Y\}}$. To accept (FD) $X \rightarrow Y$, we have to remove all sets of tuple except c'_is. The less set of tuple that have to be cut off to accept (FD) $X \rightarrow Y$ is

equal to size of |c| - size of the set of the largest tuple c'$_i$s. Thus, the equation is $e(X \rightarrow Y) = 1 - \sum_{c \in \Pi x} \max \{|c'| \mid c' \in \prod_{X \cup \{Y\}} and \quad c' \subseteq c\} / |r|$.

Approximate Threshold ε, if e(X $\rightarrow$ Y) ≤ ε then (FD) X $\rightarrow$ Y, ε equal to the largest set that has to be removed to accepted (FD) X $\rightarrow$ Y.

1.2 Related Researches

(Maletic and Marcus 1999) introduced an automated data cleaning framework. Their work separated into 2 parts: identifying error and cleaning data. The underlying theoretical aspects of the data quality of their research is a combination of existing problem-solving methods in software testing, data mining, knowledge based systems, and machine learning to address the framework. According to their research, to design automated data cleaning, one has to identify errors and then clean such dirty data. Thus, our design use the FD discovery algorithm for identifying errors and cleaning algorithm together to produce FD cleaning tool.

Several researchers in this field have mentioned that too many FDs has been generated (Andritsos et al. 2004; Ilyas et al. 2004). (Huhtala et al. 1999) showed a pruning technique for generating a candidate set and computing each candidate member to determine FDs. The ranking technique has been proposed in (Ilyas et al. 2004) and (Andritsos et al. 2004). (Ilyas et al. 2004) applied a selectivity value for ranking FDs from generated FDs (called "SoftFD"). Their work proposed that if p1 and p2 are predicates on respective columns C_1 and C_2, then the selectivity of the conjunctive predicate p1 $\wedge$ p2 is estimated by simply multiplying together the individual selectivity of $|C_1||C_2|/|C_1,C_2|$. (Andritsos et al. 2004) proposed that the FD ranking should be concerned on the first merge of the attribute that has the most amount of duplicate attribute value. These 2 ranking techniques give us the idea of ranking by looking at the data distribution. However, the merging technique will consume more times than the selectivity value because it generates the clustered matrix but the selectivity value can be found by counting attribute value directly. Therefore, our work will choose the selectivity value technique to ranking the generated FDs.

There are 2 parts for cleaning algorithms: FD repairing technique and Duplicate Elimination. FD repairing has been proposed by (Bohannon et al. 2005). Their research used a cost based technique which used a low cost data to repair a high cost data. (Hernandez and Stolfo 1995) proposed Sorted Neighborhood methods for Data Duplicate elimination by finding keys to determine duplicate tuples, then sorting the duplicate tuples and finally, matching tuples in the window to identify its duplication.

1.3 Contributions

To combine the FD discovery technique to the cleaning tool, we found and solved the following problems:

- The result of FD discovery can produce too many FDs. To reduce its number, we merge the ranking technique using selectivity value to prevent a wrong chosen FD that can cause data inconsistency and errors in the FD discovery. During the discovery step, we also identify suspicious tuples for cleaning.
- The duplicate elimination algorithm will sort all attributes to group the similar tuples, this algorithm increases work load. Therefore, this research will repair suspicious

error data first and then do the duplicate elimination. It helps to reduce the number of sorting attributes and, as a result, decrease a work load.

2. SYSTEM ARCHITETURE

The system architecture consists of Data Collector, FD Engine, Cleaning Engine, and Data in Relational Database (as shown in Figure 1). The methods for data cleaning start at the Data Collector retrieving the dirty data from relational database and the FD Engine will identify duplicate data and inconsistency error, after that the Cleaning Engine will bring the FD generated from the FD engine to repair dirty data. Next, the cleaning engine will store data in the relational database and make it ready to import to a data warehouse.

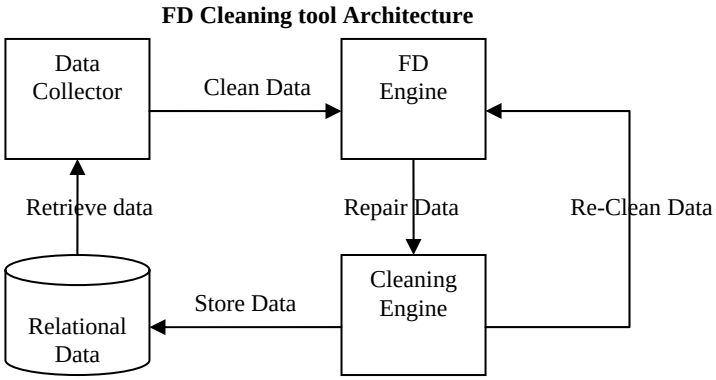


Figure 1. FD cleaning tool architecture

2.1 Data collector

The Data collector improves some quality of data and prepares it for the next module. The module corrects data from basic typos, invalid domains and invalid formats. These problems can cause algorithm in the FD engine to run incorrectly because the FD engine use exactly matching. The output data from this module will be in a relational format.

2.2 FD engine

The FD engine is an FD finding module. Since the dirty data usually has some errors, so we use the Approximate FD technique (Huhtala et al. 1999) to remove errors and find FD. But to select only useful FDs, we apply the selectivity value technique to rank the candidates in its Pruning step and select the candidates only with the high and low rank from the computing FD step. At the same time, any errors detected from this modified FD engine are suspicious tuples for cleaning. The errors can be separated into 2 types: errors from finding a candidate key FDs and errors from finding non-candidate key FDs. The non-candidate key FDs' errors are

inconsistent data. The candidate key FDs are potentially duplicated data. Together with the discovered FDs, all suspicious error tuples will be sent to the next step, the cleaning engine.

2.3 Cleaning engine

The cleaning engine will receive the suspicious error tuples with FD selected from the FD engine and then will assign weight to the data. A high error produces a high weight. Tuples with low weights will repair the high weight tuples. After updating the weight, the engine brings the FD to clean the data by using the Cost-based algorithm (Bohannon et al. 2005). The last step is to find the duplicate data by improving the sorted neighbor-hood method algorithm (Hernandez and Stolfo 1995) through using the candidate key FD from the FD engine to assign key and sorting data from the attribute on the left-hand side of FDs.

```
PROCEDURE FIND_FD(LowRankingThreshold, HighRankingThreshold, ApproximateThreshold)
OUTPUT set_of_fds
BEGIN
        Initialize Level 0, Level 1
        set_of_fds = empty
        level = 1
        WHILE (GetNoOfCandidates(level)>0)
        BEGIN
                set_of_fds = set_of_fds U  ComputeFD(ApproximateThreshold,level)
                set_of_fds = set_of_fds U ComputeFDPrimaryKey(ApproximateThreshold,level)
                PruneNextLevel(LowRankingThreshold, HighRankingThreshold, level+1)
                level = level +1
          END
END
```

Figure 2. Procedure FIND_FD

3. PROCEDURE TO SYNTHESIZE FD

The FD synthesizes starts on the procedure FIND_FD as shown in Figure 2 which have to specify three threshold values: Low Ranking Threshold, High Ranking Threshold, and Approximate Threshold. The procedure returns the set of FDs that can be used for cleaning the data. In the first step, this procedure assigns empty candidates to level 0 and set all attributes of input data to new candidates in level 1. Then, set set_of_fds variable to empty and set start level as 1. Next it finds FDs from candidates in the current level, synthesize FD and Key FD from candidates in the current level, and store the result FDs in set_of_fds variable. Then, PruneNextLevel procedure using candidates in the current level has been called to store the results at next level (level +1). The procedure ends after there is no more candidate member.

4. SELECTING THE FD

We apply selectivity value for ranking the candidate in order to find the appropriate FD.

4.1 Selectivity value

As mention in (Ilyas et al. 2004), the selectivity value, $|C_1||C_2|/|C_1, C_2|$, determined its distribution. If the selectivity value of any attribute is high, the attribute value is highly distributed. But if the attribute value is low then the attribute value is more likely to be united. Thus, the highly distributed attribute is potentially a candidate key and can be used to eliminate duplicates. While the lowest distributed attribute can be applied to improve the error of distortion of attribute values in the cleaning engine.

The above selectivity value, according to (Huhtala et al. 1999), can be calculated from $|\Pi_X||\Pi_Y|/|\Pi_{X, Y}|$ where the $|\Pi_X|$ represents a number of classes in a partition X, the $|\Pi_Y|$ represents a number of classes in a partition Y and $|\Pi_{X, Y}|$ represents a number of classes in a partition $X \cap Y$. For example, as in table 1, selectivity value of $A \wedge B$ is 2 x 4 / 6 = 1.33.

4.2 Ranking the candidate

After calculating the selectivity value for determining the ranks of candidates, we sort these ranks in ascending order as shown in Figure 3.

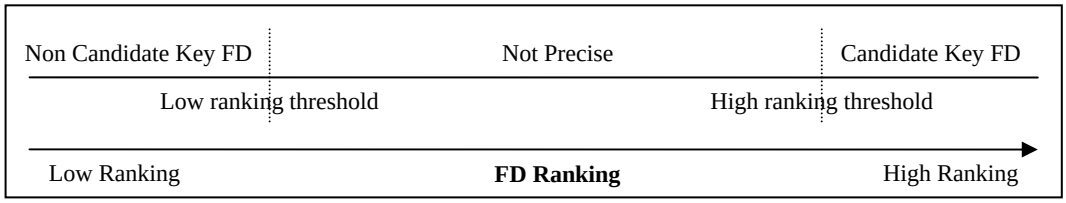


Figure 3. Ranking FD example

To choose potentially good candidates, we first define the low ranking threshold and high ranking threshold as a pruning point. The selected candidates are chosen from the candidates with either high ranking or low ranking values. The high ranking candidate has high selectivity (i.e., its cardinality is closed to the table's cardinality). Thus, it is potentially a candidate key. The low ranking candidates is potentially an invariant valued which can be functionally determined by some attribute in a trivial manner. Thus, it can be computed to be a non-candidate key on the right-hand side. The middle ranking is not precise so we drop it.

4.3 Improve the pruning step

The example of candidate generation is shown in Figure 4. At level 0, the starting level, the amount of candidate has been set to 0. At level 1, the member of candidate is set to {A, B, C, D, E, F, G} then we calculate the ranking and cut off some members in the middle rank. The remaining candidate is {A, C, D, E}. At level 2, we generate the candidate set {DA, DC, E} while candidate E in the high ranking will not be combined with the low ranking. At level 3

the last level, {DAC, E} has been generated after the algorithm described above cut off some members.

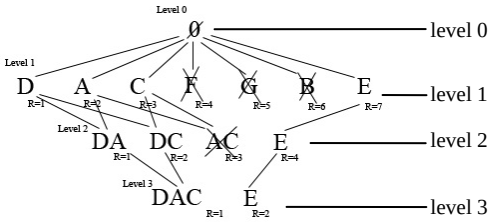


Figure 4. Pruning lattice example

Algorithm in Figure 5 shows the improved pruning technique applying with the low ranking and high ranking threshold. Thus, the PruneNextLevel procedure has LowRankingThreshold, HighRankingThreshold and a pruning level as its arguments. The algorithm works as follow: first, it begins the pruning by getting the set of candidates in level - 1 and then, checks the candidates. If they are not the FD and in either high or low accepted ranking, then we use StoreCandidate function to store new candidate from candidate_x and candidate_y in the current level. Other candidates that are in a neither low nor high ranking will be ignored.

```
PROCEDURE PruneNextLevel(LowRankingThreshold, HighRankingThreshold, level)
BEGIN
        set_of_candidates = GetCandidateSet(level - 1)
        superkey_threshold = (1 – HighRankingThreshold) x no_of_tuples
        FOR i = 0 TO |set_of_candidates| - 1
                FOR j = 1 TO |set_of_candidates| - 1
                BEGIN
                        candidate_x = GetCandidate(i, set_of_candidates)
                        candidate_y = GetCandidate(j, set_of_candidates)
                        IF (NOT IsFDAccept(candidate_x)) AND
                        ((GetRanking(candidate_x, candidate_y) <= LowRankingThreshold) OR
                        (GetRanking(candidate_x, candidate_y) >= HighRankingThreshold)  OR
                        (GetNoOfClasses(candidate_x, candidate_y) >= superkey_threshold))
                        BEGIN
                                StoreCandidate(candidate_x, candidate_y, level)
                        END
                END
END
```

Figure 5. Improved pruning method

5. TESTING

To demonstrate performance of the tool in cleaning data, we test the cleaning tool with 2 different groups of data. The first group is the actual customer data containing 50,000 records.

The testing with customer data shows the ability of tool to clean data by automatic finding proper FD comparing to specify FD manually. The second group is Part-Supplier data from (TPC.ORG 2006) containing 239,000 records, generated from a program. This data is used in order to test any affects of its parameters to the capability and limitation of the cleaning tool.

5.1 Measurement

In our experimental, we already known the error and duplicate of data, so we can compare the input and output of our algorithm by using the following measurement;

1. Error Corrected = (Number of error tuples that has been repaired correctly in the output / Number of error tuples in the input) * 100%
2. False Positive = (Number of error tuples that has been repaired and the result is still error in the output / Number of correct tuples in the input) * 100%
3. Recall = (Number of error tuples that has been repaired in the output / Number of error tuples in the input) * 100%

5.2 The Dataset Generator

This dataset generator provides duplicate tuples and inconsistency to be introduced in the tuples in any of the attributes. The inconsistency introduced in tuples is performed by given FDs. We distort one tuple per FD randomly. The dataset generator accepts %duplicates and %inconsistency as its arguments. For example, to generate dataset with 10% duplicates, the program randomly chooses 47,500 tuples from 50,000 real customer tuples, insert to a new dataset. Next, it randomly chooses 2,500 tuples from 47,500, create duplicates and append to the new dataset. For the dataset with 10% inconsistency, it inserts all tuples of 50,000 real customer tuples into a new dataset. To make inconsistency, it chooses 5,000 tuples in the new dataset and randomly selects 5,000 tuples and randomly picks an FD from the given FDs to distort the data. Last example, for dataset with 10% duplicates and inconsistency, it creates a new dataset with 10% duplicates and then makes 10% inconsistency in the same way as previously mentioned.

5.3 Real Dataset Testing

50,000 real customer tuples are used as a data source. Each customer tuple consists of the following attributes: CustID, Title, Thai_Name, Thai_Surname, Eng_Name, Eng_Surname, Occupation, Address, Alley, Road, Sub_District, District, Province, Postcode and Phone. To allow us to perform controlled studies and to evaluate the accuracy of our method, all test dataset for our cleaning algorithm was distorted automatically by making inconsistency and duplication via a dataset generator.

Table 2. Ten Given FDs for the customer data

Thai_Name, Eng_Surname → Occupation
Postcode → Province
Road, District → Province
Sub_District, District → Province
Thai_Name, Thai_Surname → Eng_Name, Eng_Surname
CustID, Thai_Name → Thai_Surname
CustID → Eng_Name, Eng_Surname
Thai_Name, Thai_Surname → Address, Alley, Road, Sub_District, District, Province, Postcode
Thai_Name, Thai_Surname → Phone
District, Phone → Province

Results

In our experiment, we separate the dataset into 3 sets, as follows: first dataset has 10% duplicates, second dataset has 10% inconsistency and last dataset has 10% duplicates and inconsistency. We assign the ApproximateThreshold 0.05 for all cases except 10% inconsistency which uses 0.03 for this threshold, Low Ranking Threshold 0.1, and High Ranking Threshold 0.005 to FD discovery algorithm.

Each dataset has been tested and compared between the cleaning result from the FD discovery in our algorithm (aka., Discovery FD) to the cleaning result from manually given FDs (aka., Manual FD). The cleaning result of the Discovery FD method is expected to be as good as the Manual FD.

Result of 10% duplicates

As shown in Figure 6, the Discovery FD has improved 100% of Error Corrected but the Manual FD has improved 100%. Figure 7 showed that the Discovery FD and Manual FD have 0% False Positive. Figure 8 showed that the Discovery FD has 100% of recall similar to the Manual FD.

The result of this dataset, Manual FD has the ability to detect 100% of duplication. The result in Discovery FD has ability to detect 100% of duplication. The results showed that the Discovery FD has the ability to detect the duplication of tuples in the dataset similar to the Manual FD.

Result of 10% inconsistency

As shown in Figure 6, the Discovery FD has improved 10% of Error Corrected but the manual one has improved 34%. Figure 7 showed that the Discovery FD has 3.38% False Positive while the Manual FD has 0.78% False Positive. Figure 8 showed that the Discovery FD has 32% of recall and the Manual FD has 30% of recall.

The cleaning result for this dataset in both cases can improve not much. The reason is that the algorithm is not able to find the conflict tuples to help in the cleaning process. Although, the Manual FD gives a better result than the Discovery FD, the Discovery FD is able to detect inconsistency better than the Manual FD.

Result of 10% duplicates and inconsistency

As shown in Figure 6, the Discovery FD can improve 70% of Error Corrected and Manual FD can improve 66% of Error Corrected. Figure 7 showed that the Discovery FD has 0.0049% of False Positive but the Manual FD can improve 0.83% of False Positive. So, the Discovery FD

has given a lower False Positive than the manual one. Figure 8 showed that the Discovery FD has 70% of recall but the Manual FD has 66% of recall. In this case, the Discovery FD also gives a better recall than the Manual FD.

For this dataset, both methods are able to correct some inconsistency but in the Discovery FD gave a higher percentage of error corrected than the Manual FD. Overall, the Discovery FD seems to have the ability to find FD almost equal to Manual FD but it can detect suspicious tuples better than the Manual FD.

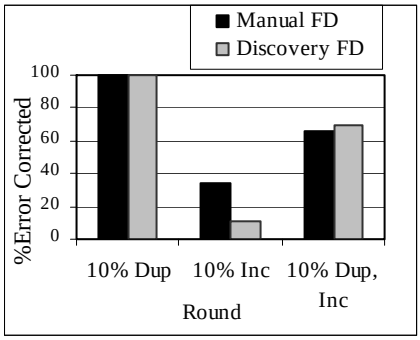


Figure 6. %Error Corrected

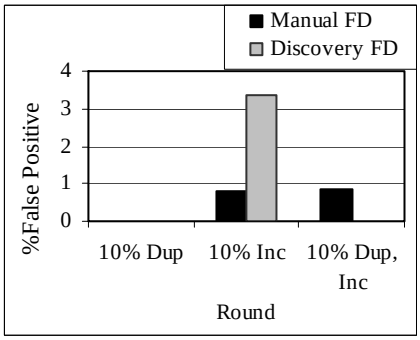


Figure 7. %False Positive

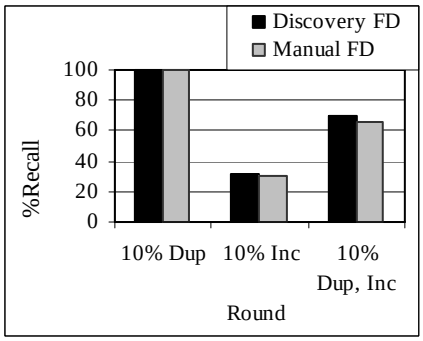


Figure 8. %Recall

Synthesize Dataset Testing

Data for testing is the information of products and their vendors which generated from dbgen of TCP-H (TPC.ORG 2006) In data generation, we use the Scale of data at 300K which will output total 239,200 records and 19 attributes. The characteristic of this generated data is using 2 attributes as a key attribute.

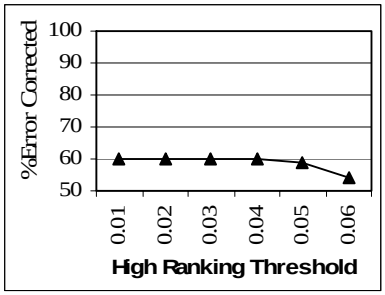
Table 3. FDs found in PART-SUPPLIER data

P_BRAND → P_MFGR
P_NAME → P_TYPE
P_NAME → P_SIZE
P_NAME → P_MFGR
P_PARTKEY → P_CONTAINER
P_PARTKEY → P_NAME
S_PHONE → S_ADDRESS
S_NAME → S_PHONE
S_NAME → S_ACCTBAL
S_SUPPKEY → S_NATIONKEY

The testing of varies threshold value

To find the effect of threshold change on the data cleaning, we divided the test into 3 parts: testing High Ranking Threshold, testing Low Ranking Threshold, and testing Approximate Threshold. We use 2 groups of data: 10% Dup, Inc and 10% Inc, to compare the effect causing by different anomaly data.

High Ranking Threshold: Set Approximate Threshold to 0.05 and Low Ranking Threshold to 0.3 According to the graph from Figure 9 and 10, it shows, from 0.01 to 0.04, %Error Corrected is ~59% and %False Positive is 0%. After increasing the High Ranking Threshold to more than 0.4, the %Error Corrected is decreased and the %False Positive is increased.



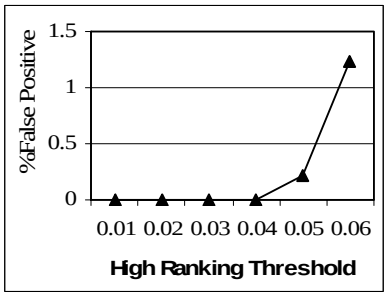


Figure 9: %Error Corrected in High Ranking Threshold change on 10% Dup, Inc data

Figure 10: %False Positive in High Ranking Threshold change on 10% Dup, Inc data

According to the graph from Figure 11 and 12, %Error Corrected starts at ~20% and decreases gradually. At High Ranking Threshold equal to 0.0001, %False Positive is 0%.

After increasing High Ranking Threshold, %False Positive trend is increasing. The best High Ranking Threshold of 10% Dup, Inc is 0.01. The High Ranking Threshold is so different because 10% Inc data contains no duplicate and dispersing data due to some attributes are free content causing High Ranking Threshold to high and giving wrong candidate keys. In the next experiment, 10% Inc data will be assigned High Ranking Threshold to 0.0001 and 10% Dup, Inc will be assigned High Ranking Threshold to 0.01, resulting from this test.

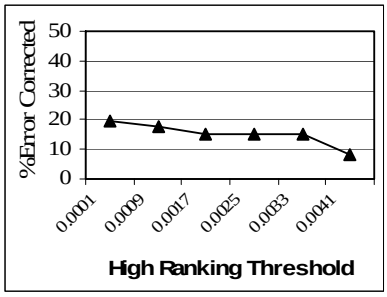


Figure 11: %Error Corrected in High Ranking Threshold change on 10% Inc data

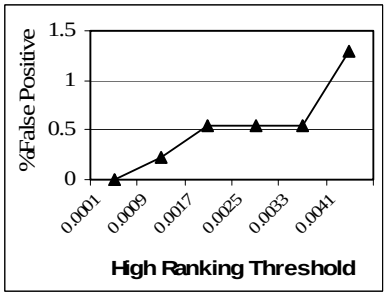


Figure 12: %False Positive in High Ranking Threshold change on 10% Inc data

Low Ranking Threshold: Set Approximate Threshold to 0.05 and High Ranking Threshold to 0.01 for 10% Dup, Inc and 0.0001 for 10% Inc. According to the graph from Figure 13 and 14, in Low Ranking Threshold between 0.3 and 0.4 for 10% Dup, Inc and 10% Inc data, %Error Corrected is ~59% and 20% respectively whereas %False Positive is ~0%. After increasing Low Ranking Threshold to more than 0.4, %Error Corrected of 10% Inc data dramatically decreased and %Error Corrected of 10% Dup, Inc increased slightly while %False Positive of both data increased. As considered, the less Low Ranking Threshold is found at 0.3.

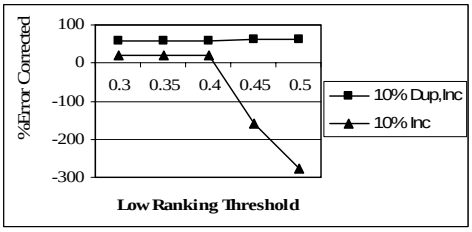


Figure 13: %Error Corrected in Low Ranking Threshold

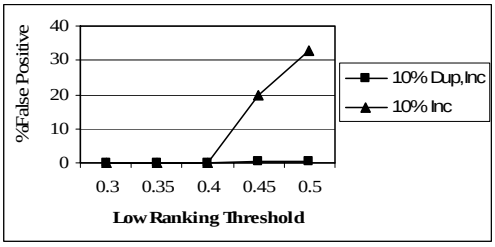


Figure 14: %False Positive in Low Ranking Threshold change

Approximate Threshold: Set Low Ranking Threshold to 0.3 and High Ranking Threshold to 0.01 for 10% Dup, Inc and 0.0001 for 10% Inc. According to the graph from Figure 15 and 16, for the data of 10% Dup, Inc, %Error Corrected is low about 10% at threshold equal to 0.03 while at threshold equal to 0.05, %Error Corrected is ~59% and steady after that. The %False Positive is ~0% on all thresholds. For the data of 10% Inc, %Error Corrected is ~20% and %False Positive ~0% on all thresholds. In the Figure 15 on 10% Dup, Inc, %Error

Corrected increased at Approximate Threshold equal to 0.05 because 0.05 (5%) is the minimum (1/2 of %Dup) that Discovery FD can synthesize Candidate Key FD.

In summary, the experiment of Low Ranking Threshold and High Ranking Threshold change found that increasing both threshold values up to some certain point will not improve and worsen the cleaning process.

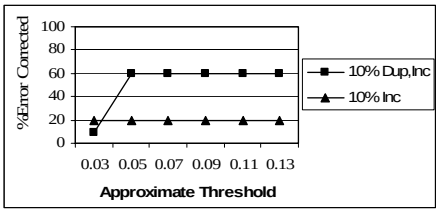


Figure 15: %Error Corrected in Approximate Threshold change

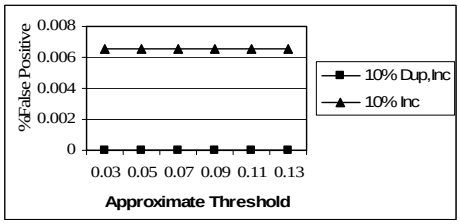


Figure 16: %False Positive in Approximate Threshold change

The cleaning test on three data sets

The cleaning test divided the data into 3 sets: the first is 10% Dup, the second is 10% Inc and the third is 10% Dup, Inc. As from the testing of varies threshold value, all sets will be assigned threshold value as Approximate Threshold = 0.05, Low Ranking Threshold = 0.3, and High Ranking Threshold = 0.1 except for the second set that High Ranking Threshold is assigned to 0.0001. Each data set will be compared between cleaning data by Discovery FD and Manual FD. We expected the result of Discovery FD should be close to that of Manual FD.

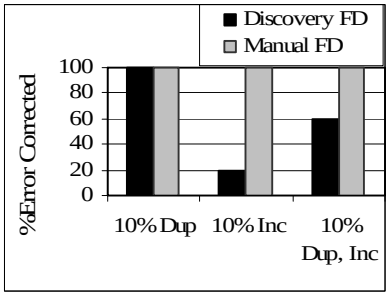


Figure 17: %Error Corrected

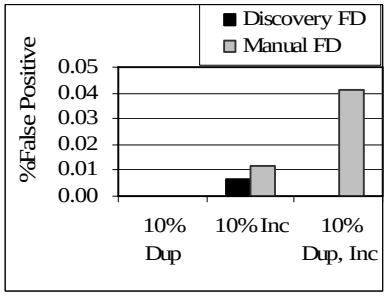


Figure 18: %False Positive

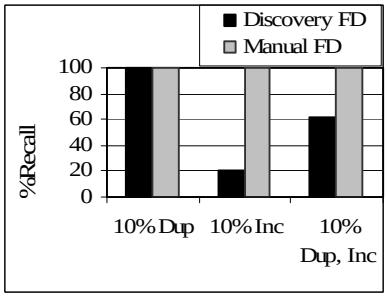


Figure 19: %False Positive

The results of cleaning test on three data sets as shown in Figure 17, 18, and 19 shows %Error Corrected, %False Positive and %Recall respectively. For 10% Dup, both Discovery FD and Manual FD correct duplicated data at 100%. For 10% Inc, Manual FD produced ~99% of %Error Corrected and %False Positive at ~0.011% while Discovery FD gave %Error Corrected at ~ 19% and %False Positive at ~0.006%.

For the testing on 10% Dup, Inc, Manual FD is able to clean data at %Error Corrected as ~99% and %False Positive as 0.04% while Discovery FD is able to clean data at %Error Corrected as ~60% and %False Positive as 0%.

In conclusion, Manual FD is able to clean data efficiently. Although Discovery FD detects duplicated and correct data less efficient than Manual FD, it did not damage (low %False Positive) the data.

6. CONCLUSION AND FUTURE WORK

We have developed a cleaning tool using FD discovery. Our tool uses an FD discovery with a ranking technique to reduce the FD discovery's number. Also, the discovery step can help to identify suspicious tuples for cleaning. The algorithm passes these errors to the cleaning step for repairing to reduce the number of sorting attributes and, as a result, decrease a work load.

From our result, the first experiment showed that our algorithm can clean, especially duplicate data, efficiently. The FD discovery algorithm can find the useful FDs that can be used to clean data effectively almost equal to the manually setting ones. In the second experiment, we discovered that the tool might not be able to synthesize all correct FDs and somehow it depends on the input data. Nevertheless, the tool is able to clean data especially in the duplicated data without any given FD. For contaminated data, the tool does not damage any data even though it cannot clean all of them. The experiment also shows that the chosen threshold value should not be too high (for example, less than 0.05%). Any high value threshold may damage the data and generate incorrect FDs. The approximate threshold varies to the percentage of the contamination in data. So, to utilize it effectively, one may have to estimate its contamination to provide some proper scope of the parameters of the tool.

In the future, the tool parameters must be explored to justify the proper values without human intervention. Since the tool generates FDs from data if some good sampling technique is applied, we may find and generate FDs from a smaller set of data rather than the whole data and it will help to reduce the time for generating, consequently it will help to improve the cleaning time.

ACKNOWLEDGEMENT

The authors would like to thank Chuleerat Rattanaprateep, Eakkapol Wattanatittan and Chakkrit Kheawsa-ad for their helpful discussions and comments on this paper.

REFERENCES

Andritsos, P. et al, 2004. Information-Theoretic Tools for Mining Database Structure from Large Data Sets. Proceedings of the 2004 ACM SIGMOD international conference on Management of data. Paris, France, pp. 731-742.

Arenas, M. et al, 1999. Consistent Query Answers in Inconsistent Databases. Proceedings of the 18th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems. Philadelphia, USA, pp. 68-79.

Bohannon, P. et al, 2005. A Cost-Based Model and Effective Heuristic for Repairing Constraints by Value Modification. Proceedings of the 2005 ACM SIGMOD international conference on Management of data. Maryland, USA, pp. 143-154.

Calvanese, D. et al, 2001. Identification Constraints and Functional Dependencies in Description Logics. Proceedings of the 17th International Joint Conference on Artificial Intelligence. Washington, USA, pp. 155-160.

Erhard, R. and Do, H. H., 2000. Data Cleaning: Problems and Current Approaches. IEEE Data Engineering Bulletin, Vol. 23, No. 4, pp. 3-13.

Garcia-Molina, H. et al, 2001. Database Systems The Complete Book. Prentice Hall, New Jersey, USA.

Hernandez, M. A. and Stolfo, S. J., 1995. The Merge/Purge Problem for Large Databases. Proceedings of the 1995 ACM SIGMOD international conference on Management of data. San Jose, California, USA, pp. 127-138.

Huhtala, Y. et al, 1999. TANE: An Efficient Algorithm for Discovering Functional and Approximate Dependencies. The Computer Journal, Vol. 42, No. 2, pp. 100-111.

Ilyas, I. F. et al, 2004. CORDS: Automatic Discovery of Correlations and Soft Functional Dependencies. Proceedings of the 2004 ACM SIGMOD international conference on Management of data. Paris, France, pp. 647-658.

Maletic, J. I. and Marcus, A. 1999. Progress Report on Automated Data Cleansing. from http://www.cs.kent.edu/~jmaletic/papers/TR-CS-99-02.pdf.

Savnik, I. and Flach, P. A., 1993. Bottom-up Induction of Functional Dependencies from Relations. Proceedings of the AAAI93 Workshop on Knowledge Discovery in Databases. California, USA, pp. 174-185.

TPC.ORG 2006. TPC-H Decision Support for Ad Hoc Queries. from http://www.tpc.org.