Discovering Conditional Functional Dependencies to Detect Data Inconsistencies

Peter Z. Yeh
Accenture Technology Labs
San Jose, CA 95113
peter.z.yeh@accenture.com

Colin A. Puri
Accenture Technology Labs
San Jose, CA 95113
colin.puri@accenture.com

ABSTRACT

Poor quality data is a growing and costly problem that affects many enterprises across all aspects of their business ranging from operational efficiency to revenue protection. In this paper, we present an approach that efficiently and robustly discovers conditional functional dependencies for detecting inconsistencies in data and hence improves data quality. We evaluate our approach empirically on three real-world data sets, and show that our approach performs well on these sets across several dimensions such as precision, recall, and runtime. We also compare our approach to an established solution and show that our approach outperforms this solution across the same dimensions. Finally, we describe efforts to deploy our approach as part of an enterprise tool being developed at Accenture to accelerate data quality efforts such as data profiling and cleansing.

1. INTRODUCTION

Many organizations suffer from poor quality data – a problem that is getting worse because data is growing at astonishing rates and few organizations have an effective data governance process. A 2002 study estimated that data quality problems cost U.S. businesses more than $600 billion annually [5]. These problems impact all aspects of an organization ranging from operational efficiency to revenue protection.

Poor data quality can occur along several dimensions (e.g. conformity, duplication, consistency, etc.), and consistency (i.e. ensuring that values across interdependent attributes are correct) is one dimension that many organizations struggle with. The process of detecting inconsistencies in data is labor-intensive. For example, Table 1 shows a sample of records (and attributes) for U.S. federal grants given to the state of Michigan as part of the economic recovery program. This sample contains several inconsistencies, which are not obvious. In row 1, the *Rcpt City* attribute (i.e. the recipient's city) has the value of *Lansing*, but the *Rcpt District* attribute (i.e. the recipient's congressional district) has the value of *6*, which is incorrect. The correct value is 8. Similarly, in row 4 the *Rcpt Category* and *Agency* attributes have values of *For Profit* and *HUD* respectively, but the *Program* attribute has the value of *Public Housing*, which is also incorrect. The correct value is *Section 8 Housing* because the recipient is a "for profit".

Hence, many organizations often ignore this critical dimension, which leads to various problems such as inaccurate reporting of key metrics (e.g. who received grants, what types of grants, etc.) used to inform critical decisions.

Recently Conditional Functional Dependencies (CFDs) were introduced for detecting inconsistencies in data [2], and were shown to be more effective than standard Functional Dependencies (FDs) [2] and association rules [4]. We can use CFDs to formulate the following rules which can detect the inconsistencies in Table 1.

(Rcpt City $\rightarrow$ Rcpt District, (Lansing $\|$ 8))

*(Rcpt Category, Agency $\rightarrow$ Program,
(For Profit, HUD $\|$ Section 8 Housing))*

Each CFD is a rule of the form $(X \rightarrow Y, T_p)$ where X and Y are attributes from a relation of interest (e.g. Table 1), $X \rightarrow Y$ is a FD, and T_p is a pattern tuple. This tuple consists of values from attributes in X and Y along with a wildcard (i.e. '_') that can match any arbitrary value.

Approaches have also been proposed for automatically discovering CFDs from data [4, 6, 7]. These approaches, however, have various limitations. Approaches such as [7] require FDs as inputs which is not feasible in practice, as the FDs are not always available. Approaches such as [4, 6] do not have this limitation, but they have difficulty scaling to relations with a large number of attributes (it is not uncommon for enterprises to have relations with 100 attributes) and are not robust to dirty data (these approaches will overlook many CFDs, and clean data sets are often not available for discovering CFDs in practice).

In this paper, we present a solution that addresses these limitations. Our solution can:

- Effectively prune the search space and hence can handle relations with a large number of attributes (e.g. up to 100).
- Robustly handle dirty data during discovery and hence can discover useful CFDs even when the data has a large percentage of inconsistencies (e.g. up to 50%).
- Determine when a rule becomes stable and hence can avoid examining the entire data set and overfitting.

We evaluate our approach empirically on three real-world data sets. We show that our approach performs well on these sets across several dimensions such as precision, recall, and runtime. We also compare our approach to an established solution and show that our approach outperforms this solution across the same dimensions. We conclude by describing efforts to deploy our approach as part of

Table 1: A sample of records and attributes from U.S. Federal grants received by Michigan.

#	Rcpt Category	Rcpt City	Rcpt District	Agency	Agency Code	Program	CFDA No.
1	Government	Lansing	6	ED	9131:DOED	Pell	84.063
2	Government	Lansing	8	FHA	6925:DOT	Highway Planning	20.205
3	Government	Lansing	8	FHA	6925:DOT	Highway Planning	20.205
4	For Profit	Lansing	8	HUD	8630:HUD	Public Housing	14.885
5	Higher ED	Ann Arbor	15	ED	9131:DOED	Pell	84.063
6	Higher ED	Ann Arbor	15	ED	9131:DOED	Work Study	84.033
7	For Profit	Detroit	13	HUD	8630:HUD	Section 8 Housing	14.317
8	For Profit	Detroit	13	HUD	8630:HUD	Section 8 Housing	14.317

an enterprise tool being developed at Accenture to accelerate data quality efforts such as data profiling and cleansing.

2. APPROACH

Our approach – we call CFinder – discovers Conditional Functional Dependencies (CFDs) from a relation of interest through the following steps. CFinder first generates an initial set of candidate CFDs. CFinder then refines each CFD by removing extraneous (or invalid) conditions, and stops refining a CFD when it becomes stable. Finally, CFinder filters weak (and subsumed) CFDs, and generalizes the remaining ones to increase their applicability.

2.1 Generate Candidate CFD

Given a relation R, CFinder generates candidate CFDs – i.e. $(X \rightarrow Y, T_p)$ – by first generating all attribute combinations of size $N + 1$ from R where N is the maximum number of attributes (and hence conditions) allowed in the antecedent (i.e. X) of a CFD. CFinder imposes this restriction because CFDs with a large number of conditions in the antecedent have limited applicability in practice.

CFinder then generates candidate CFDs from each combination. For each attribute in a combination, CFinder turns that attribute into the consequent (i.e. Y) of a CFD and turns the remaining attributes into the antecedent (i.e. X).[1] CFinder then instantiates the pattern tuple with respective values from these attributes whose frequency exceeds the minimum support threshold.

For example, given a minimum support of 20% and the following attribute combination from Table 1:

(Agency, Agency Code, Program, CFDA No.)

some of the CFDs that CFinder will generate include:

(Agency Code, Program, CFDA No. $\rightarrow$ Agency,
(9131:DOED, Pell, 84.063 $\|$ ED))

(Agency, Agency Code, Program $\rightarrow$ CFDA No.,
(HUD, 8630:HUD, Pell $\|$ 14.317))

However, the number of combinations (and hence candidate CFDs) can be extremely large, so CFinder prunes combinations that are unlikely to produce useful CFDs based on two heuristics.

The first heuristic is useful CFDs are more likely to be generated from attributes that are strongly related (e.g. *Agency* and *Agency Code*). CFinder implements this heuristic by treating each combination c as a fully connected graph with attributes as nodes and by computing the average *strength* across all edges (and hence how strongly are the attributes related to each other) using the following equation:

$$\frac{\sum_{(A,B)\epsilon E(c)} Strength(A,B)}{|E(c)|}$$

where $E(c)$ are all edges in c, (A, B) is an edge between attributes A and B, and $Strength(A, B)$ measures how strongly A is related to B. A good measure for $Strength(A, B)$ is the mutual dependence between these attributes, as high mutual dependence indicates a strong relationship between A and B. Hence, CFinder defines $Strength(A, B)$ as the mutual information shared between A and B:

$$\sum_{a\epsilon U(A)} \sum_{b\epsilon U(B)} P(a,b) log \frac{P(a,b)}{P(a)P(b)}$$

where $U(A)$ and $U(B)$ are the unique values in A and B respectively; and P is the relative frequency of a value (or value pair) in an attribute (or attribute pair).

CFinder prunes combinations with low strength, and sets the default strength threshold to 0.5. For example, Figure 1 (top) shows the fully connected graph for the following attribute combination from Table 1.

c_1: *(Rcpt Category, Rcpt City, Agency, Agency Code)*

The edge labels indicate the strengths between these attributes. Since the average strength (i.e. 1.13) is greater than 0.5, CFinder will keep this combination.

The second heuristic is many combinations are variants of one another and can be pruned. These variants often result in the discovery of the same CFDs because CFinder refines CFDs by removing extraneous (or invalid) conditions from the antecedent (see Section 2.2).

CFinder implements this heuristic by first sorting – in descending order based on strength – combinations that remain after applying the first heuristic. CFinder then traverse this list in descending order, and for each combination c it finds all preceding combinations C' that have minimal difference with c. CFinder defines this difference as the number of attributes in c that are not in c' where $c' \epsilon C'$, and sets the default difference to 1 – i.e. C' will contain all combinations that differ from c by one attribute.

Since C' contains more promising combinations (and hence CFDs) than c, CFinder should prune c if it has significant overlap with C'. Because each combination can be treated as a fully connected graph, the overlap between c and any combination in C' is their maximum common subgraph. If the non-overlapping edges in c (i.e. edges not found in C') are weak, then it is unlikely that this combination will produce any new, useful CFDs. CFinder captures

[1]Generating candidate CFDs with only one attribute in the consequent (i.e. *minimal* CFDs) does not limit the generality of our approach because CFDs with multiple attributes in the consequent can be decomposed into *minimal* CFDs, which can be considered individually [2].

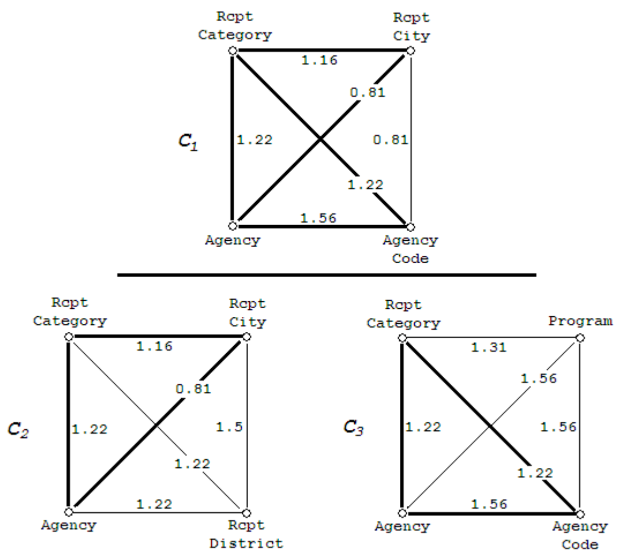


Figure 1: Top: The fully connected graph for c_1. Bottom: Two combinations that rank higher than c_1, and have high overlap with c_1. Bold edges indicate overlaps.

this notion formally as:

$$\frac{\sum_{(A',B')\epsilon E'(c)} Strength(A',B')}{\sum_{(A,B)\epsilon E(c)} Strength(A,B)}$$

where $E(c)$ are all edges in c and $E'(c)$ are edges in c that overlap with combinations in C'. If this value exceeds the prune threshold H_P, then the combination is pruned.

For example, Figure 1 (bottom) shows two additional combinations from Table 1 whose strengths rank higher than c_1. If H_P is 0.85, then CFinder will prune c_1 because it has high overlap (shown in bold) with c_2 and c_3, and the non-overlapping edge in c_1 is weak.

CFinder generates candidate CFDs from the remaining combinations – starting with the strongest one – and refines these CFDs in the order they are generated (see Section 2.2).

2.2 Refine CFD

CFinder refines each candidate CFD by comparing it with records from the relation of interest. For each record, CFinder determines whether the record is consistent, inconsistent, or irrelevant to the CFD.

1. A record is consistent with a CFD (and hence supports it) if all values in the pattern tuple T_p of the CFD match the respective values in the record. If so, then CFinder increments the consistent record count R_C by 1.

2. A record is inconsistent with a CFD if all values in T_p – that correspond to the antecedent of the CFD – match the respective values in the record, but values that correspond to the consequent do not. If so, then CFinder increments the inconsistent record count R_I by 1.

3. Otherwise, the record is irrelevant to the CFD, and CFinder increments the irrelevant record count R_V by 1.

CFinder uses these counts to check whether the CFD is too specific (and hence needs to be refined) and whether inconsistencies encountered for the CFD are real errors in the data or anomalies, which can be ignored. CFinder performs these checks – once every M records – using the minimum support threshold H_S – i.e. $\frac{R_C}{R_C+R_V} \geq H_S$ – and the maximum inconsistency threshold H_I – i.e. $\frac{R_I}{R_I+R_C} \leq H_I$.

If a CFD fails to meet the minimum support threshold H_S, then CFinder refines the CFD by removing extraneous (or invalid) conditions from its antecedent. However, the difference between the observed support (i.e. $\frac{R_C}{R_C+R_V}$) and the expected support (i.e. H_S) may be due to a "sampling" effect with the M records examined. This effect can cause the CFD to be over-refined and become too promiscuous. Hence, CFinder needs to determine the significance of this difference, and it uses the $\mathcal{X}^2$ test, which is instantiated as:

$$\frac{(R_C - H_S(R_C+R_V))^2}{H_S(R_C+R_V)} + \frac{(R_V - (1-H_S)(R_C+R_V))^2}{(1-H_S)(R_C+R_V)}$$

CFinder will refine a CFD only if the difference is significant – i.e. the resulting $\mathcal{X}^2$ value exceeds the critical $\mathcal{X}^2$ value at the specified confidence level, which CFinder defaults to 99%.

CFinder selects the top K most promising conditions to remove from the antecedent of the CFD. Since the goal is to improve support, CFinder should remove conditions whose value occurs infrequently with the consequent's value and whose attribute has high uncertainty (i.e. many different unique value pairs) with the consequent's attribute because these conditions cause many records to be irrelevant. CFinder implements this notion formally as:

$$(1 - P(T_p(A), T_p(B)))Entropy(A,B)$$

where A and B are attributes of the condition and consequent respectively; $T_p(*)$ is the value of an attribute in the pattern tuple; P is the relative frequency of the value pair $T_p(A)$ and $T_p(B)$ across all records examined so far; and $Entropy(A,B)$ is the joint entropy between A and B across all records examined so far.

CFinder selects K conditions with the highest scores based on the equation above, and for each condition CFinder removes the condition from the antecedent of the original CFD to generate a new CFD. For example, let's assume CFinder needs to refine the following CFD by selecting the top two conditions, and the records in Table 1 are the ones examined so far.

(Agency Code, Program, CFDA No. $\longrightarrow$ Agency,
(9131:DOED, Pell, 84.033 || ED))

CFinder will select *CFDA No.* and *Program* – whose scores are 1.97 and 1.69 respectively (*Agency Code* has the lowest score of 0.98) – and remove them from the original CFD to generate the following new CFDs.

(Agency Code, Program $\longrightarrow$ Agency,
(9131:DOED, Pell || ED))

(Agency Code, CFDA No. $\longrightarrow$ Agency,
(9131:DOED, 84.033 || ED))

For each new CFD, CFinder records the CFD to prevent it from being generated again; and recomputes R_C, R_I, and R_V for the CFD. If no conditions remain in the antecedent, then the CFD is discarded.

Similarly, if a CFD exceeds the maximum inconsistency threshold H_I, then CFinder determines whether the difference between the observed inconsistency (i.e. $\frac{R_I}{R_I+R_C}$) and the expected inconsistency (i.e. H_I) is significant using the $\mathcal{X}^2$ test, which is instantiated as:

$$\frac{(R_I - H_I(R_C+R_I))^2}{H_I(R_C+R_I)} + \frac{(R_C - (1-H_I)(R_C+R_I))^2}{(1-H_I)(R_C+R_I)}$$

If the difference is significant, then CFinder penalizes the CFD by adding R_I to R_V and then resetting R_I to 0. This penalty increases the likelihood that the CFD will fail to meet the minimum support threshold, which will cause the CFD to be refined and eventually discarded (if the inconsistencies persist).

CFinder repeats the above process until all records have been examined or the CFD becomes stable (see Section 2.3).

2.3 Determine Stable CFD

Examining all records to discover CFDs is computationally expensive and can result in CFDs that overfit the data. CFinder addresses these two issues by determining whether a CFD is stable and hence does not need to be refined further. A CFD is stable if both the support for the CFD and the certainty of the values that make up the attributes referenced in the CFD are constant over a given period of time.

CFinder captures this notion by first computing a stability score St for the CFD using the following equation:

$$\frac{R_C}{R_C + R_V} \sum_{A \epsilon X \cup Y} Entropy(A)$$

where R_C and R_V are consistent and irrelevant record counts for the CFD respectively (see Section 2.2); $X \cup Y$ are all attributes referenced in the CFD; and $Entropy(A)$ is the entropy of A across all records examined so far. CFinder computes this score once every M records – when it checks the minimum support and maximum inconsistency thresholds (see Section 2.2 also).

CFinder then computes the standard deviation SD_{St} for the past L stability scores; and marks the CFD as stable if SD_{St} is constant according to the equation $\frac{SD_{St}}{Avg_{St}} \leq H_{St}$ where Avg_{St} is the average of the past L stability scores and H_{St} is the stability threshold.

For example, if the certainty of the values for the attributes in a CFD fluctuates or a condition is removed from a CFD, then the entropy component of the stability score St will change significantly, which will prevent the CFD from becoming stable. Similarly, if the support for a CFD fluctuates, then St will fluctuate as well, which will prevent the CFD from becoming stable.

2.4 Filter and Generalize CFD

CFinder uses the measures of support [1] and conviction [3] to filter weak CFDs – i.e. CFDs that do not meet (or exceed) the thresholds specified for these measures. Support measures how much evidence there is for a CFD, and can be defined using the consistent and irrelevant record counts (see Section 2.2). Conviction measures how much the antecedent and consequent of a CFD deviate from independence while considering directionality. This measure has been shown to be effective for filtering weak CFDs [4]. Given space limitations, we refer the reader to [1, 3, 4] for additional details on these measures.

In addition to these measures, CFinder applies an additional filter to remove subsumed CFDs. A CFD – i.e. $F_1 : (X_1 \rightarrow Y_1, T_{p1})$ – subsumes another CFD – i.e. $F_2 : (X_2 \rightarrow Y_2, T_{p2})$ – if Y_1 equals Y_2, $X_1 \subset X_2$, and $T_{p1} \subset T_{p2}$. If these conditions are met, then CFinder removes the subsumed CFD (i.e. F_2) because it has less applicability.

CFinder then generalizes the remaining CFDs to further increase their applicability. A CFD F_1 can be generalized if there exists another CFD F_2 such that 1) F_1 and F_2 have the same antecedents and consequents (i.e. X_1 equals X_2 and Y_1 equals Y_2) and 2) the pattern tuples of F_1 and F_2 differ by a single value. If these conditions are met, then CFinder generalizes F_1 and F_2 into a single CFD by replacing the differing value in their pattern tuples with a wildcard (i.e. '_'). For example, given the following CFDs:

(Rcpt Category, Agency → Program,
(Government, ED || Pell))

(Rcpt Category, Agency → Program,
(Higher ED, ED || Pell))

CFinder can generalize them into:

(Rcpt Category, Agency → Program, (_, ED || Pell))

CFinder repeats this final step until there are no more CFDs that can be generalized.

3. EVALUATION

We evaluated the following claims to show that our approach (i.e. CFinder) can efficiently and robustly discover CFDs that can effectively detect inconsistencies in data.

- **Claim 1**: CFinder can robustly discover useful CFDs even when the data has a large percentage of inconsistencies.
- **Claim 2**: CFinder can efficiently discover CFDs by effectively pruning the space of candidate CFDs.
- **Claim 3**: CFinder can further improve efficiency and prevent overfitting by determining when a CFD becomes stable.

3.1 Data Sets

We used three real-world data sets for our evaluation. The first data set – we call *Recovery MI* – contains U.S. federal grants given to the state of Michigan as part of the economic recovery program. Each record has information about the recipient, the grant type, etc. This data set has 41 attributes and 2,916 records.

The second data set – we call *Manifest* – contains manifest information from a large U.S. shipping and logistics organization. Each record has information about the item being shipped, the sender, the recipient, etc. This data set has 102 attributes and 21,182 records.

The last data set – we call *Ops* – contains operational information from the same shipping and logistics organization. Each record has information about which facility processed an item for shipping, when an item was processed, etc. This data set has 12 attributes and 51,067 records.

In each case, ensuring data consistency is important in preventing inaccurate reporting (and hence poor decisions), revenue loss, and operational inefficiency.

3.2 Experiment Setup and Results

To evaluate Claim 1, we measured the precision and recall of inconsistencies detected using the CFDs discovered by CFinder for all three data sets. To obtain these measures, we first randomly introduced inconsistencies into each data set at rates of 10%, 20%, 30%, 40%, and 50% – e.g. if the inconsistency rate is 30%, then there is a 30% chance that a value in a data set will be randomly replaced with a different value from the same attribute in that set. We then performed a 10-fold cross-validation for each data set at each inconsistency rate. We defined precision as the number of true inconsistencies (i.e. inconsistencies that were randomly introduced) detected by CFinder over all inconsistencies detected; and recall as the number of true inconsistencies detected by CFinder over all inconsistencies introduced.

To evaluate Claim 2, we measured the runtime (in seconds) of CFinder in discovering CFDs and the number of attribute combinations (and hence candidate CFDs) pruned by CFinder. We obtained both measures from all 10-fold cross-validation runs performed in evaluating Claim 1.

Table 2: The average precision, recall, and runtime from 10-fold cross-validations performed for all evaluated approaches and data sets at inconsistency rates from 10% to 50%. * and $^+$ indicate cases where CFinder performed significantly better than CFD-TANE and CFinder-NoStable respectively ($p < 0.01$ for the 2-tail pairwise t-test, df = 9).

	CFinder			**CFD-TANE**			**CFinder-NoStable**		
	Precision (%)	Recall (%)	Time (s)	Precision (%)	Recall (%)	Time (s)	Precision (%)	Recall (%)	Time (s)
Recovery MI									
0.1	0.7712^*	0.5060^*	2428.1^{*+}	0.2840	0.3494	17273.5	0.7744	0.5056	2781.0
0.2	0.8062^*	0.5047^*	2186.3^{*+}	0.5771	0.2227	17098.0	0.8147	0.5013	2503.8
0.3	0.7878^*	0.5599^*	1966.8^*	0.6866	0.1497	17291.7	0.7911	0.5560	1988.4
0.4	0.8447^*	0.5089^*	1374.0^{*+}	0.6423	0.0625	17411.1	0.8423	0.5075	1362.0
0.5	0.8691	0.4007^*	781.3^{*+}	0.8469	0.0145	17240.1	0.8704	0.3980	883.0
Manifest									
0.1	0.8406	0.6548^+	4791.4^+	N/A	N/A	N/A	0.9391	0.4625	5487.5
0.2	0.8536	0.6587^+	6170.3^+	N/A	N/A	N/A	0.8957	0.4347	6338.3
0.3	0.8778	0.6612^+	6105.8^+	N/A	N/A	N/A	0.9136	0.5646	6733.3
0.4	0.9077	0.6861^+	7106.4^+	N/A	N/A	N/A	0.9305	0.4905	7529.9
0.5	0.9209	0.6323^+	8312.5^+	N/A	N/A	N/A	0.9543	0.3241	8883.5
Ops									
0.1	0.9202^*	0.4019^{*+}	4226.0^{*+}	0.0764	0.1771	5581.2	0.9268	0.3283	5299.2
0.2	0.7378^*	0.3585^{*+}	3727.0^{*+}	0.0833	0.0662	5424.6	0.9156	0.2285	4628.9
0.3	0.7014^*	0.3586^{*+}	3386.7^{*+}	0.0773	0.0482	5379.5	0.7338	0.2125	4316.0
0.4	0.7322^*	0.3676^{*+}	3457.5^{*+}	0.2230	0.0812	5685.8	0.7062	0.1546	4070.2
0.5	0.7928^*	0.3747^{*+}	3021.8^{*+}	0.2698	0.0769	5234.3	0.9107	0.1658	3520.6

For both evaluations, we compared CFinder with an established solution for discovering CFDs [4], which we'll call CFD-TANE. CFD-TANE is a TANE-based [8] solution that performs a breadth-first search of an attribute lattice for CFDs – i.e. CFDs with N+1 attributes are derived from sets of N attributes. CFD-TANE also produces approximate CFDs to handle inconsistencies encountered during discovery.

To evaluate Claim 3, we created a variant of CFinder – we call CFinder-NoStable – which does not determine whether a CFD is stable and hence examines the entire data set. We used the same methodology described above to obtain the measures of precision, recall, and runtime for this variant.

We ran all evaluations using a dual-core 2.4 gigahertz AMD Opteron processor with 4GB of memory on a Linux Ubuntu operating system. We set the minimum support H_S for both CFinder and CFD-TANE to 0.02, 0.03, and 0.05 for the Manifest, Ops, and Recovery MI data sets respectively. We set the minimum conviction to 5.0. We set the size of attribute combinations generated by CFinder to 4; and to ensure a fair comparison with CFD-TANE, we bounded its search depth at level 4 inclusive. Hence, both approaches discovered CFDs over the same search space.

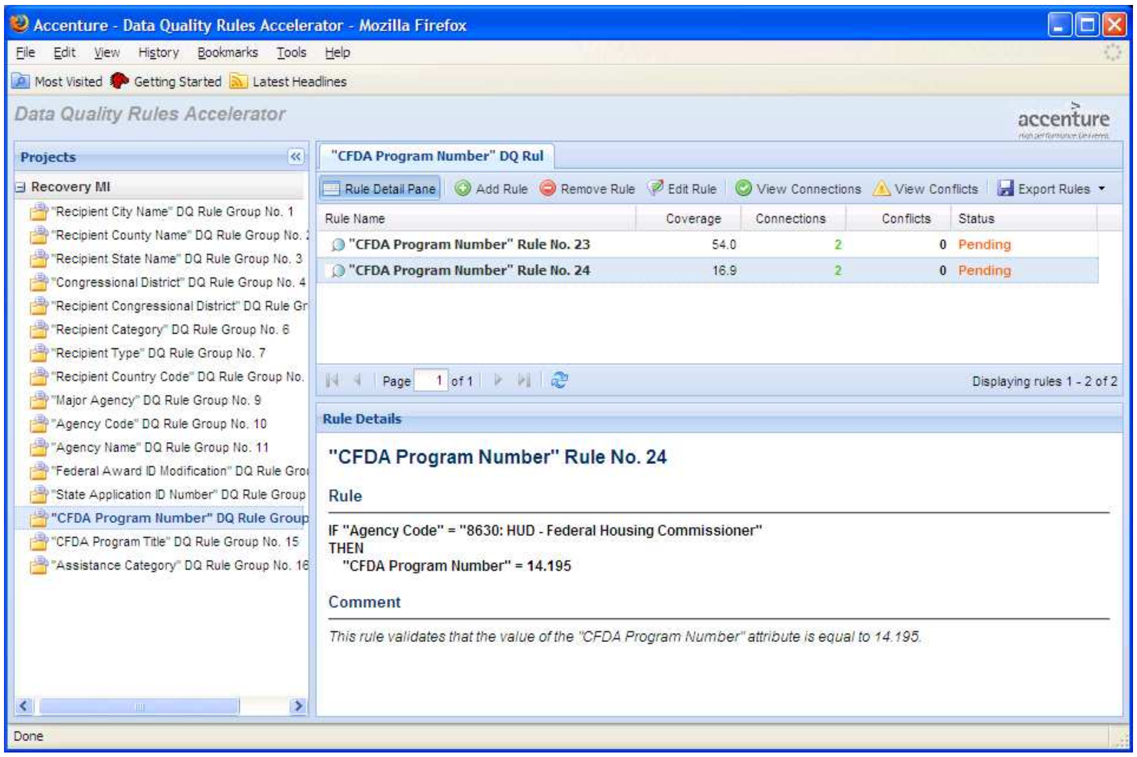
We set parameters specific to CFinder as follows. The prune candidate threshold H_P was set to 0.85; and the maximum inconsistency threshold H_I was set to 0.025, 0.05, 0.1, 0.15, and 0.2 for inconsistency rates of 10%, 20%, 30%, 40%, and 50% respectively. Minimum support and maximum inconsistency checks were performed once every 200, 1,000, and 2,000 records for Recovery MI, Manifest, and Ops respectively. We had CFinder select the top 2 conditions when a CFD needs to be refined. Finally, the stability threshold H_{St} was set to 0.025 over the last 10 stability scores St. Applicable parameters for CFinder-NoStable were the same as CFinder.

Tables 2 and 3 show the results for all three evaluations. CFinder performed significantly better than CFD-TANE on recall across all data sets and inconsistency rates; and significantly better on precision in most cases. Moreover, the recall of CFinder remained relatively stable, even as the inconsistency rate increased. CFinder performed well because it can robustly handle inconsistencies during discovery, which CFD-TANE could not. CFD-TANE either overlooked many useful CFDs or discovered ones that were too promiscuous. These results support our first claim that CFinder can robustly discover useful CFDs even when the data has a large percentage of inconsistencies.

Table 3: The Recovery MI, Manifest, and Ops data sets have a total of 101,270; 4,249,575; and 495 attribute combinations of size 4 respectively. This table reports the average number of attribute combinations pruned by CFinder for these data sets across all inconsistency rates.

	Recovery MI	**Manifest**	**Ops**
0.1	99,906	4,249,486	490
0.2	100,154	4,249,489	490
0.3	100,336	4,249,490	490
0.4	100,430	4,249,493	490
0.5	100,480	4,249,478	490

CFinder performed significantly better than CFD-TANE on runtime for the Recovery MI and Ops data sets at all inconsistency rates. We did not report results for CFD-TANE on the Manifest data set because it could not handle the large number of attributes (i.e. 102 attributes). CFinder performed well because it can effectively prune the search space – e.g. in the best case CFinder pruned over 99.99% of all possible attribute combinations and hence candidate CFDs (see Table 3). These results support our second claim that CFinder can efficiently discover CFDs by effectively pruning the space of candidate CFDs.


Accenture - Data Quality Rules Accelerator - Mozilla Firefox
File Edit View History Bookmarks Tools Help
Most Visited Getting Started Latest Headlines
Data Quality Rules Accelerator
accenture
Projects
Recovery MI
"Recipient City Name" DQ Rule Group No. 1
"Recipient County Name" DQ Rule Group No. 2
"Recipient State Name" DQ Rule Group No. 3
"Congressional District" DQ Rule Group No. 4
"Recipient Congressional District" DQ Rule Gr
"Recipient Category" DQ Rule Group No. 6
"Recipient Type" DQ Rule Group No. 7
"Recipient Country Code" DQ Rule Group No.
"Major Agency" DQ Rule Group No. 9
"Agency Code" DQ Rule Group No. 10
"Agency Name" DQ Rule Group No. 11
"Federal Award ID Modification" DQ Rule Grou
"State Application ID Number" DQ Rule Group
"CFDA Program Number" DQ Rule Group
"CFDA Program Title" DQ Rule Group No. 15
"Assistance Category" DQ Rule Group No. 16
"CFDA Program Number" DQ Rul
Rule Detail Pane
Add Rule
Remove Rule
Edit Rule
View Connections
View Conflicts
Export Rules
Page 1 of 1
Displaying rules 1 - 2 of 2
Rule Details
Done

Rule Name	Coverage	Connections	Conflicts	Status
"CFDA Program Number" Rule No. 23	54.0	2	0	Pending
"CFDA Program Number" Rule No. 24	16.9	2	0	Pending

"CFDA Program Number" Rule No. 24

Rule

IF "Agency Code" = "8630: HUD - Federal Housing Commissioner"
THEN
"CFDA Program Number" = 14.195

Comment

This rule validates that the value of the "CFDA Program Number" attribute is equal to 14.195.

Figure 2: Browser displaying data quality rules discovered from a user-specified dataset. The left panel groups related rules into folders. The top right panel shows all rules within a selected folder. The bottom right panel shows the details for a selected rule.

The precision of CFinder and CFinder-NoStable was comparable. CFinder-NoStable did have significantly better precision in some cases, but its recall was significantly lower in most. These results show that CFinder-NoStable overfit the data to produce CFDs with higher precision but lower recall. Moreover, the runtime of CFinder was significantly lower than CFinder-NoStable in general because it does not need to examine the entire data set. We attributed these results to the only difference between these two approaches – CFinder determines when a CFD is stable, CFinder-NoStable does not. These results support our last claim that CFinder can further improve efficiency and prevent overfitting by determining when a CFD becomes stable.

4. DEPLOYMENT EFFORTS

Accenture – a global technology consulting and outsourcing company – performs a wide range of large-scale enterprise projects for its client from master data management to business intelligence. An important factor in the success of these projects is ensuring good quality data through efforts such as data profiling and cleansing. These efforts, however, are expensive and time consuming. In particular, Accenture client teams currently spend a significant amount of time manually identifying industry (and client) relevant rules that are required to profile and cleanse the data. The process of identifying these rules typically involves interviewing relevant subject matter experts on the client side.

To address this problem, Accenture is developing a tool – called the Data Quality Rules Accelerator – that can automatically discover relevant data quality rules; and our approach has been implemented as part of this tool to discover CFDs, which can detect and correct data inconsistencies.

The intended users of this tool are Accenture client teams performing data profiling and cleansing; and they interact with this tool through a web-based interface. The typical sequence of interactions is described below:

1. The user selects a data file – in CSV format – to discover data quality rules from (in particular CFDs). The user also has the option of connecting directly to a database through an ODBC connection to select a relation for discovery.

2. The user sets parameters such as the minimum support, the maximum inconsistency, etc. These parameters along with the selected data are then sent to a backend server on which our approach is implemented. Discovered rules are sent back to the user and displayed in a rules browser (see Figure 2).

3. The user then reviews the discovered rules with subject matter experts on the client side to accept those that should be deployed and to reject those that are extraneous. The user can also edit these rules or add additional ones through a rules editor.

4. The user deploys accepted rules by exporting them – through the tool's export feature – to either SQL statements or a commercial solution for data profiling and cleansing such as Informatica Data Quality.

The Data Quality Rules Accelerator (and hence our approach) is currently being piloted with select Accenture client teams performing data profiling and cleansing – several of the data sets used in our evaluations are from these pilots. The goal of these pilots is to evaluate the technical feasibility and business value of this tool in real-world situations. Initial results (and feedbacks) from these pilots indicate that this tool can reduce the overall effort required to profile and cleanse the data. Upon successful completion of these pilots, this tool will be made available to all Accenture client teams, and will be incorporated as part of Accenture's delivery methodology for data quality.

5. CONCLUSION

In this paper, we presented an approach that can efficiently and robustly discover effective Conditional Functional Dependencies (CFDs) for detecting inconsistencies in data, and hence can address the growing problem of poor quality data faced by many organizations. We evaluated our approach on three real-world data sets and showed that our approach performed well on these sets across several dimensions such as precision, recall, and runtime. We also compared our approach to an established solution and showed that our approach outperformed this solution across the same dimensions. Finally, we described efforts to deploy our approach as part of an enterprise tool being developed at Accenture to accelerate data quality efforts such as data profiling and cleansing.

6. ACKNOWLEDGMENT

The authors would like to thank the reviewers for their helpful feedback and suggestions for improving the paper. The authors would also like to thank Scott Kurth, Sanjay Mathur, Mark Wagman, and Ajay Easo from Accenture Technology Labs for their contributions to this work.

7. REFERENCES

[1] R. Agrawal and R. Srikant. Fast algorithms for mining association rules in large databases. In *VLDB*, 1994.

[2] P. Bohannon, W. Fan, F. Geerts, X. Jia, and A. Kementsietsidis. Conditional functional dependencies for data cleaning. In *ICDE*, 2007.

[3] S. Brin, R. Motwani, J. Ullman, and S. Tsur. Dynamic itemset counting and implication rules for market basket data. In *SIGMOD*, 1997.

[4] F. Chiang and R. Miller. Discovering data quality rules. In *VLDB*, 2008.

[5] W. Eckerson. Data quality and the bottom line. Technical report, TDWI Report Series, 2002.

[6] W. Fan, F. Geerts, L. Lakshmanan, and M. Xiong. Discovering conditional functional dependencies. In *ICDE*, 2009.

[7] L. Golab, H. Karloff, F. Korn, D. Srivastava, and B. Yu. On generating near-optimal tableaux for conditional functional dependencies. In *VLDB*, 2008.

[8] Y. Huhtala, J. Kinen, P. Porkka, and H. Toivonen. Efficient discovery of functional and approximate dependencies using partitions. In *ICDE*, 1998.