

Accelerating the Discovery of Data Quality Rules: A Case Study

Peter Z. Yeh, Colin A. Puri, Mark Wagman, and Ajay K. Easo

Accenture Technology Labs

San Jose, CA 95113

{peter.z.yeh,colin.puri,mark.wagman,ajay.k.easo}@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 application – Data Quality Rules Accelerator (DQRA) – that accelerates Data Quality (DQ) efforts (e.g. data profiling and cleansing) by automatically discovering DQ rules for detecting inconsistencies in data. We then present two evaluations. The first evaluation compares DQRA to existing solutions; and shows that DQRA either outperformed or achieved performance comparable with these solutions on metrics such as precision, recall, and runtime. The second evaluation is a case study where DQRA was piloted at a large utilities company to improve data quality as part of a legacy migration effort. DQRA was able to discover rules that detected data inconsistencies directly impacting revenue and operational efficiency. Moreover, DQRA was able to significantly reduce the amount of effort required to develop these rules compared to the state of the practice. Finally, we describe ongoing efforts to deploy DQRA.

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 (Eckerson 2002). These problems impact all aspects of an organization from operational efficiency to revenue protection.

Poor quality data can occur along several dimensions such as conformity, duplication, consistency, etc. However, existing commercial solutions (Informatica ; Trillium) address only a subset of these dimensions, and no cost-effective commercial solution exists for addressing consistency – i.e. ensuring values across interdependent attributes are correct. The state of the practice still involves working closely with Subject Matter Experts (SMEs) – who know the data and domain – to manually identify relevant rules that can then be applied by commercial solutions to detect data inconsistencies like those in Table 1. For example, the guideline used by one division at Accenture – a global technology consulting

and outsourcing company – for estimating the effort required to identify relevant rules for a data quality effort is two hours per attribute per SME. Hence, many organizations overlook consistency from efforts such as data profiling and cleansing. This oversight can lead to numerous problems such as inaccurate reporting of key metrics (e.g. who received grants, what types of grants, etc.) used to inform critical decisions or derive business insights.

Recently Conditional Functional Dependencies (CFDs) were introduced for detecting inconsistencies in data (Bohannon et al. 2007), and were shown to be more effective than Functional Dependencies (FDs) (Bohannon et al. 2007) and association rules (Chiang and Miller 2008). We can use CFDs to formulate the following data quality rules which can detect the inconsistencies in Table 1.

(Rcpt City → Rcpt District, (Lansing || 8))
(Rcpt Category, Agency → Program,
(For Profit, HUD || Section 8 Housing))

Intuitively, a CFD is an *if-then* rule that captures how the attribute values on the *if*-side of the rule constrain the attribute values on the *then*-side. For example, the first CFD above says if the recipient’s city (i.e. *Rcpt City*) is Lansing then the recipient’s congressional district (i.e. *Rcpt District*) must be constrained to 8. Formally, a 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.

Previous research has proposed solutions for automatically discovering data quality rules (in particular CFDs) from data. These approaches, however, have various limitations. Approaches such as (Golab et al. 2008) require FDs as inputs which is not feasible in practice, as the FDs are not always available. Approaches such as (Chiang and Miller 2008; Fan et al. 2009) do not have this limitation, but they 1) have difficulty scaling to relations with a large number of attributes (it is not uncommon for enterprises to have relations with 100 attributes) and 2) 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 our previous research, we proposed a solution that addresses many of these limitations (Yeh and Puri 2010), but efficiency still remains an issue.

#	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

Table 1: A sample of records and attributes for U.S. federal grants given to the state of Michigan as part of the economic recovery program. In row 1, the *Rcpt City* attribute (i.e. recipient’s city) has the value of *Lansing*, but the *Rcpt District* attribute (i.e. 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”.

In this paper, we present an application – Data Quality Rules Accelerator (DQRA) – to accelerate data quality efforts (e.g. data profiling and cleansing) by automatically discovering data quality rules (in particular CFDs) for detecting data inconsistencies. We give an overview of the delivery process our application supports and its main features, followed by a description of the AI algorithm for discovering the rules, which improves upon the efficiency of our previous solution. We then present two evaluations. The first evaluation compares DQRA to existing solutions; and shows that DQRA either outperformed or achieved performance comparable with these solutions on metrics such as precision, recall, and runtime. The second evaluation is a case study where DQRA was piloted at a large utilities company to improve data quality as part of a legacy migration effort. DQRA was able to discover rules that detected data inconsistencies directly impacting revenue and operational efficiency. Moreover, DQRA was able to significantly reduce the amount of effort required to develop these rules compared to the state of the practice. We conclude by describing ongoing efforts to deploy DQRA.

Application Overview

Accenture performs a wide range of large-scale enterprise projects for clients from legacy migration 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. To perform these efforts, Accenture Client Teams (ACTs) follow a six step process (see Figure 1). However, the *Investigate* through *Define* steps of this process are expensive because ACTs currently spend a significant amount of time manually identifying domain (and client) relevant rules to profile and cleanse the data.

To accelerate these steps, Accenture has developed an application – Data Quality Rules Accelerator (DQRA) – that can automatically discover rules, which ACTs can use to detect and correct data inconsistencies. ACTs interact with DQRA through a web-based interface, and the typical sequence of interactions is:

1. The user selects a data file – in CSV format – to discover data quality rules from (in particular CFDs). The user also

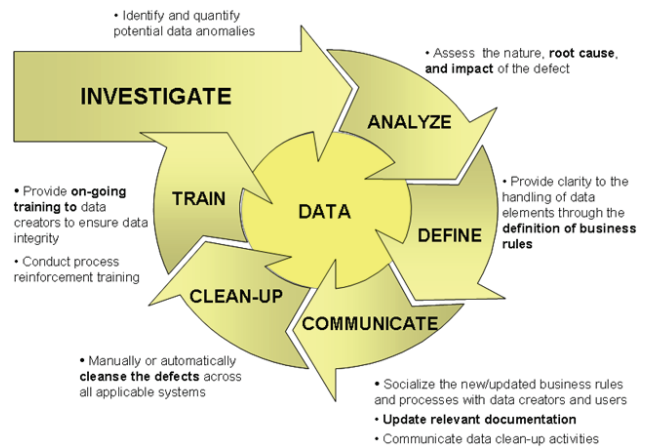


Figure 1: High-level schematic of Accenture’s DQ process.

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 maximum number of rules, the minimum support for a rule, etc. These parameters along with the selected data are then sent to a backend server which performs the discovery. Discovered rules are sent back to the user and displayed in a rules browser (see Figure 2).
3. The user examines the discovered rules 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 DQRA’s automated export feature – to vendor solutions for data profiling and cleansing such as Informatica Data Quality.

Algorithm Overview

The Data Quality Rules Accelerator (DQRA) discovers Conditional Functional Dependencies (CFDs) from a rela-

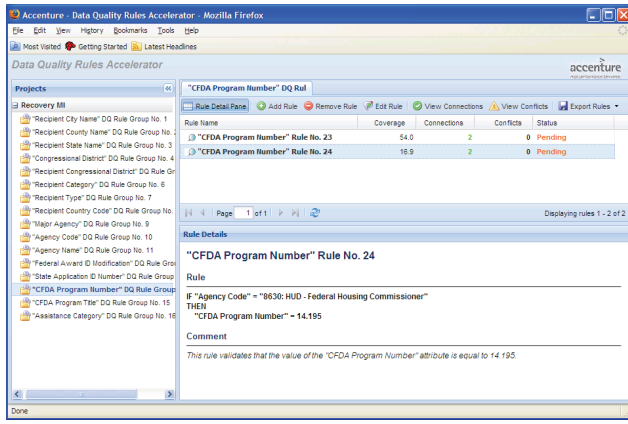


Figure 2: Browser displaying discovered DQ rules.

tion of interest through the following steps. DQRA first generates an initial set of candidate CFDs. DQRA then revises each CFD to improve its precision. Finally, DQRA filters weak (and subsumed) CFDs, and generalizes the remaining ones to increase their applicability.

Generate Candidate CFD

Given a relation R ,¹ DQRA generates candidate CFDs – i.e. rules of the form $(X \rightarrow Y, T_p)$ where X and Y are attributes from R , and T_p is a pattern tuple which consists of values from these attributes.

DQRA first generates all attribute pair combinations. However, the number of attribute pairs (and hence candidate CFDs) can be extremely large, so DQRA prunes pairs that are unlikely to produce useful CFDs based on the heuristic that useful CFDs are more likely to be generated from attributes that are strongly related (e.g. *Agency* and *Agency Code*). A good measure of this strength is the mutual dependence between two attributes A and B , so DQRA defines $Strength(A, B)$ as the mutual information shared between A and B :

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

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

DQRA prunes pairs with low strength, and defaults the strength threshold H_S to 1.0. For example, DQRA will keep the attribute pair $(Agency, Agency Code)$ from Table 1 because its strength (i.e. 1.56) is greater than 1.0, but will prune $(Agency, Rcpt City)$ because its strength (i.e. 0.81) is below the threshold.

DQRA then generates candidate CFDs from the remaining pairs. For each pair, DQRA turns the first attribute into

¹For a large relation (i.e. $> 1MM$ records), DQRA will split it into multiple blocks for scalability purposes and perform discovery on each block. The resulting CFDs are then merged. Due to page limit, we do not cover this aspect of the algorithm in this paper.

the antecedent (i.e. X) of a CFD and turns the second attribute into the consequent (i.e. Y)² – and vice versa. DQRA then instantiates the pattern tuple with unique value pairs from the attribute pair whose relative frequency exceeds the minimum support threshold (Agrawal and Srikant 1994), which is specified by the user. For example, given a minimum support of 20% and the attribute pair $(Agency, Agency Code)$, DQRA will generate CFDs such as:

$$(Agency Code \rightarrow Agency, (9131:DOED \parallel ED))$$

$$(Agency \rightarrow Agency Code, (HUD \parallel 8630:HUD))$$

Revise CFD

The initial candidate CFDs may be too promiscuous, and hence may detect many inconsistencies that are false positives. To reduce false positives (and hence improve precision), DQRA must determine whether a CFD should be revised.

For each CFD, DQRA determines the number of records that are inconsistent with the CFD. 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.

DQRA uses this information to check whether the inconsistencies are real errors to expect from the data or are the result of the CFD being too promiscuous. CFinder performs this check using the expected inconsistency threshold H_I – i.e. $\frac{R_I}{R_I + R_S} \leq H_I$ where R_I and R_S are the number of inconsistent and supporting records respectively. This threshold is specified by the user, and reflects his/her expectation of the level of inconsistency in the data.

If the observed inconsistency (i.e. $\frac{R_I}{R_I + R_S}$) for a CFD exceeds H_I , then DQRA revises the CFD by constraining its antecedent with additional conditions. However, the difference between the observed inconsistency and the expected inconsistency (i.e. H_I) may be due to a “sampling” effect with the records examined, which can cause the CFD to be over-constrained. Hence, DQRA needs to determine the significance of this difference. DQRA uses the χ^2 test because it is analyzing counts over mutually exclusive categories (i.e. inconsistent and supporting records), and it instantiates this test as:

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

DQRA will revise a CFD if the difference is significant – i.e. the resulting χ^2 value exceeds the critical χ^2 value at the specified confidence level, which DQRA defaults to 99%.

DQRA selects K new attributes to revise the CFD with, using our previous heuristic (i.e. useful CFDs are likely to be generated from strongly related attributes). For each new attribute, DQRA treats it and the existing attributes of the

²Generating 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 (Bohannon et al. 2007).

CFD as a fully connected graph G (with attributes as nodes), and computes the average *strength* across all edges using:

$$\frac{\sum_{(A,B) \in E(G)} \text{Strength}(A,B)}{|E(G)|}$$

where $E(G)$ are all edges in G and $\text{Strength}(A,B)$ measures how strongly attribute A is related to B (see previous subsection). DQRA selects the top K attributes where the average strength exceeds the strength threshold H_S .

For each selected attribute A , DQRA finds all unique values v_i of A such that the relative frequency of the tuple – resulting from adding v_i to T_p of the original CFD – exceeds the minimum support threshold. For each v_i , DQRA then generates a new CFD by adding A and v_i to X and T_p of the original CFD respectively. DQRA records each new CFD to prevent it from being generated again, and discards the original CFD.

For example, let's assume DQRA must revise the following CFD by selecting the top attribute from Table 1.

$$(Agency \rightarrow CFDA\ No., (ED \parallel 84.063))$$

DQRA will select *Program* because it has the highest score (i.e. 1.79) and this score exceeds the default strength threshold H_S . Assuming a minimum support of 20%, DQRA will generate the following new CFD from the original.

$$(Agency, Program \rightarrow CFDA\ No., (ED, Pell \parallel 84.063))$$

For each new CFD, DQRA determines whether the CFD needs to be revised further by applying the expected inconsistency threshold H_I from above. If so, then DQRA will repeat the above steps until the CFD does not violate H_I or the maximum number of revisions is reached, in which case the CFD is discarded.

To ensure progress on each subsequent revision, DQRA measures whether there is a significant difference w.r.t the number of supporting and inconsistent records between the original and new CFD. DQRA uses the χ^2 test for the same reason as before, and instantiates this test as:

$$\frac{(R_S - p_S(R_S + R_I))^2}{p_S(R_S + R_I)} + \frac{(R_I - p_I(R_S + R_I))^2}{p_I(R_S + R_I)} + \frac{(R'_S - p_S(R'_S + R'_I))^2}{p_S(R'_S + R'_I)} + \frac{(R'_I - p_I(R'_S + R'_I))^2}{p_I(R'_S + R'_I)}$$

where R_S and R'_S are the number of supporting records for the original and new CFD resp.; R_I and R'_I are the number of inconsistent records for the original and new CFD resp.; and p_S (i.e. $\frac{R_S + R'_S}{R_S + R'_S + R_I + R'_I}$) and p_I (i.e. $\frac{R_I + R'_I}{R_S + R'_S + R_I + R'_I}$) are *category percentages* for computing the expected number of supporting and inconsistent records resp.

If there is not a significant difference – i.e. the resulting χ^2 value does not exceed the critical χ^2 value at the 99% confidence level – then DQRA stops revising the new CFD and discards it.

Filter and Generalize CFD

DQRA uses the measure of conviction (Brin et al. 1997) to filter weak CFDs – i.e. CFDs that do not meet (or exceed)

the threshold specified by the user for this measure. 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, and we refer the reader to (Chiang and Miller 2008) for additional details.

In addition to conviction, DQRA 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 DQRA removes the subsumed CFD (i.e. F_2) because it has less applicability.

DQRA 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

- F_1 and F_2 have the same antecedents and consequents – i.e. X_1 equals X_2 and Y_1 equals Y_2
- The pattern tuples of F_1 and F_2 differ by a single value

If these conditions are met, then DQRA generalizes F_1 and F_2 into a single CFD by replacing the differing value in their pattern tuples with a wildcard (i.e. $'_'$) which can match any arbitrary value. For example, given the CFDs:

$$\begin{aligned} &(Rcpt\ Category, Agency \rightarrow Program, \\ &(Government, ED \parallel Pell)) \\ &(Rcpt\ Category, Agency \rightarrow Program, \\ &(Higher\ ED, ED \parallel Pell)) \end{aligned}$$

DQRA can generalize them into:

$$(Rcpt\ Category, Agency \rightarrow Program, (_ ED \parallel Pell))$$

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

Evaluation: Comparative Study

We present a comparative study to evaluate the performance of DQRA against existing solutions for discovering CFDs.

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, the granting agency, 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.

Experiment Setup and Results

We evaluated the performance of DQRA using the metrics of precision, recall, and runtime. We measured the precision and recall of inconsistencies detected using the CFDs

	DQRA			CFinder			CFD-TANE		
	Prec. (%)	Recall (%)	Time (s)	Prec. (%)	Recall (%)	Time (s)	Prec. (%)	Recall (%)	Time (s)
Recovery MI									
0.1	0.7595 ⁺	0.6347 ⁺⁺	166.2 ⁺⁺	0.7712	0.5060	2428.1	0.2840	0.3494	17273.5
0.2	0.8103 ⁺	0.5791 ⁺⁺	104.2 ⁺⁺	0.8062	0.5047	2186.3	0.5771	0.2227	17098.0
0.3	0.8338 ⁺⁺	0.5709 ⁺⁺	69.3 ⁺⁺	0.7878	0.5599	1966.8	0.6866	0.1497	17291.7
0.4	0.8739 ⁺⁺	0.5297 ⁺⁺	49.0 ⁺⁺	0.8447	0.5089	1374.0	0.6423	0.0625	17411.1
0.5	0.8889 ⁺⁺	0.4499 ⁺⁺	42.5 ⁺⁺	0.8691	0.4007	781.3	0.8469	0.0145	17240.1
Manifest									
0.1	0.9048 [*]	0.6454	246.9 [*]	0.8406	0.6548	4791.4	N/A	N/A	N/A
0.2	0.9798 [*]	0.6618	252.0 [*]	0.8536	0.6587	6170.3	N/A	N/A	N/A
0.3	0.9830 [*]	0.6147	298.9 [*]	0.8778	0.6612 ^x	6105.8	N/A	N/A	N/A
0.4	0.8907	0.6352	402.8 [*]	0.9077	0.6861 ^x	7106.4	N/A	N/A	N/A
0.5	0.8878	0.6448 [*]	415.9 [*]	0.9209 ^x	0.6323	8312.5	N/A	N/A	N/A
Ops									
0.1	0.9429 ⁺	0.4158 ⁺	201.4 ⁺⁺	0.9202	0.4019	4226.0	0.0764	0.1771	5581.2
0.2	0.7737 ⁺⁺	0.4007 ⁺⁺	159.3 ⁺⁺	0.7378	0.3585	3727.0	0.0833	0.0662	5424.6
0.3	0.7403 ⁺⁺	0.3756 ⁺⁺	131.1 ⁺⁺	0.7014	0.3586	3386.7	0.0773	0.0482	5379.5
0.4	0.5653 ⁺	0.3654 ⁺	105.0 ⁺⁺	0.7322 ^x	0.3676	3457.5	0.2230	0.0812	5685.8
0.5	0.4897 ⁺	0.2748 ⁺	91.8 ⁺⁺	0.7928 ^x	0.3747 ^x	3021.8	0.2698	0.0769	5234.3

Table 2: The average precision (Prec.), 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 DQRA performed significantly better than CFinder and CFD-TANE respectively. x indicates cases where CFinder performed significantly better than DQRA. In all cases, $p < 0.05$ for the 2-tail pairwise t-test and $df = 9$.

discovered by DQRA for all three data sets. We defined precision as the number of true inconsistencies detected by DQRA over all inconsistencies that it detected; and recall as the number of true inconsistencies detected by DQRA over all true inconsistencies.

However, the “ground truth” did not exist for these data sets, and constructing a gold standard was too expensive and did not allow us to assess the robustness of DQRA as inconsistencies increased. Hence, we 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 using a dual-core 2.4 gigahertz AMD Opteron processor with 4GB of memory on a Linux Ubuntu operating system. Finally, we measured the runtime (in seconds) of DQRA for each run.

We also evaluated the performance of two existing solutions using the above methodology. The first solution – we’ll call CFD-TANE – is an established approach for discovering CFDs (Chiang and Miller 2008). CFD-TANE is a TANE-based (Huhtala and others 1998) approach 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.

The second solution – i.e. CFinder (Yeh and Puri 2010) – differs from our approach in one important way. CFinder generates overly specific candidate CFDs (i.e. CFDs with multiple conditions in the antecedent), and then generalizes

them as needed by removing extraneous conditions. DQRA generates overly general candidate CFDs (i.e. CFDs with only one condition in the antecedent), and then specializes them as needed by adding additional conditions.

We set parameters common to all three approaches as follows. We set the minimum support 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 parameters specific to CFD-TANE and CFinder to values given in (Yeh and Puri 2010), which achieved the best performance for these two approaches.

We set parameters specific to DQRA as follows. The strength threshold H_S was set to 0.5, and the expected inconsistency threshold H_I was set to the inconsistency rate (e.g. we set H_I to 0.2 if the inconsistency rate is 20%). We had DQRA select the top 8 attributes when a CFD needs to be revised.

Table 2 shows the results for this evaluation. DQRA performed significantly better than CFD-TANE on precision, recall, and runtime across all data sets and inconsistency rates. DQRA had better precision and recall 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.

DQRA had better runtime because it 1) effectively pruned the initial space of candidate CFDs, and 2) discarded unpromising CFDs early (and hence further bounded the discovery space) by ensuring progress is made every time a CFD is revised (see *Revise CFD* section). In contrast, the bread-first search strategy used by CFD-TANE was not sufficient when the discovery space was large. For example, we

could not report runtime results for CFD-TANE on the Manifest data set because it could not handle the large number of attributes (i.e. 102 attributes).

DQRA performed significantly better than CFinder on runtime across all data sets and inconsistency rates. DQRA performed well because it generates overly general candidate CFDs, which are fewer than the number of overly specific candidate CFDs generated by CFinder. Moreover, DQRA discarded unpromising CFDs early, which further reduced runtime. Interestingly, we observed that discarding unpromising CFDs early occasionally prevented DQRA from discovering useful CFDs, which resulted in lower precision and recall for a few isolated cases (e.g. the 40% and 50% inconsistency rates for the Ops data set).

Besides these isolated cases, the precision and recall of DQRA were comparable to CFinder. In several cases, DQRA even had significantly better precision and recall.

These results show that DQRA in general either outperformed or achieved performance comparable with existing solutions on the metrics of precision, recall, and runtime.

Evaluation: Case Study

We present a case study where Data Quality Rules Accelerator (DQRA) was piloted at a large U.S. utilities company to improve data quality as part of a legacy migration effort.

Background and Evaluation Goals

Working with Accenture, a Large Utilities Company (LUC) embarked on a large scale migration of their legacy customer information system to a new platform consisting of Oracle CC&B and SAP Business Objects. Two years into the effort, LUC began migrating 30 years of data – previously managed by a third party contractor – into the new platform. Issues with the quality of the data surfaced immediately, and could be linked directly to operational inefficiency and revenue loss.

Both LUC and Accenture determined that improving the quality of the data is critical to the overall success of the migration effort. However, tight project timeline and budget made it infeasible to apply Accenture’s Data Quality (DQ) process (see *Application Overview* section). This would require augmenting the existing Accenture team with data quality experts who would work with LUC Subject Matter Experts (SMEs) to manually identify relevant rules to profile and cleanse the data.

Given this challenge, the client team and LUC engaged Accenture Technology Labs for a four week pilot to use DQRA to accelerate the discovery of DQ rules. This pilot enabled us to evaluate DQRA on 1) time savings compared to manual discovery and 2) the efficacy of the discovered rules in detecting DQ issues impacting key business functions such as billing and inventory management.

Pilot Overview and Result

Week 1: We worked with the Accenture client team and LUC to define the pilot scope. LUC has 31 Business Units (BUs) – each operating over a different U.S. geographic region. These BUs share a common data model, which consists of 36 tables grouped into 13 table families. However,

BU	# Records	# Attrs.	Time (sec)	# Rules
U004	314,269	363	9,564.3	188
U050	1,370,810	363	16,953.8	368
U051	514,293	363	13,640.7	216
U060	325,639	363	8,802.5	209
U127	154,969	363	8,372.3	211
U303	320,191	363	8,231.3	218

Table 3: The data size (i.e. no. of records and attributes), discovery time, and no. of discovered rules for 6 LUC BUs.

Rule Group	# Rules (%)
Bill & Read Cycle/Route	406 (28.8%)
Meter Dial & Size	130 (9.2%)
Account & Service Type	119 (8.4%)
Meter Disposition	98 (6.9%)
Service Point Location	97 (6.9%)
Rate & Revenue Code	90 (6.4%)
Service Contract Billing	80 (5.7%)
Manufacturer Code	75 (5.3%)
Meter Set/Unset Date	60 (4.3%)
Misc.	255 (18.1%)

Table 4: Rule groups and distribution across groups.

each BU is operated independently, and is subject to different regulations. Hence, rules discovered for one BU cannot be applied to another.

Given these considerations, LUC selected 4 table families – covering meter, service point, service agreement, and account type information – to focus on for 6 of their BUs.

Weeks 2-3: We applied DQRA to the selected data sets. We hosted DQRA on a Linux Ubuntu operating system with a dual-core 2.4 gigahertz AMD Opteron processor and 4GB of memory. For each BU, we joined the tables within a family to enable discovery of cross table rules, and then applied DQRA to the resulting join. The support and inconsistency threshold H_I were set to 1% and 5% respectively. We used default values for all other parameters. Table 3 shows the data size and discovery time for each BU.

The total discovery time across all 6 BUs was 18.21 hours. We estimated a total of 726 man hours would be required to manually identify relevant rules by working with LUC SMEs. We based this estimate on Accenture’s DQ estimation guidelines of 2 hours per attribute per SME applied to 363 attributes and 6 independent BUs (and hence 6 SMEs). We then adjusted the resulting estimate by $\frac{1}{6}$ because DQRA discovers rules for one of six target DQ dimensions (i.e. consistency). We assumed equal effort for each dimension due to the absence of additional estimation guidelines. This difference represents a 97.5% potential reduction in effort.

We then reviewed the discovered rules with six LUC SMEs – one from each respective BU – over two 1.5 hour sessions to determine the utility of these rules. LUC SMEs determined that a rule is useful if it captures a piece of business logic which can detect DQ issues impacting key busi-

Rule Group	Rule	DQ Issue
Meter Dial & Size	MFG_Code, Base_Size → Num_Dials, (NEP, 0058 4)	Detects wrong meter dial number which results in over or under billing
Bill & Read Cycle/Route	Bill_Cycle → Read_Route, (108 11)	Detects bill cycle & read route code mismatch which results in estimated meter reads
Meter Disposition	Inventory_Loc → Disposition, (INVN AVAIL)	Detects wrong meter disposition which results in inaccurate inventory

Table 5: Examples of representative rules from select rule groups and the DQ issues they detect.

BU	# Valid	# Invalid	# N/A
U004	33,852	47	2,085
U050	312,836	273	15,566
U051	100,669	12	16,905
U060	138,571	159	317
U127	56,586	329	673
U303	41,015	272	38,785

Table 6: No. of valid vs. invalid meter dial numbers. N/A denotes records not validated b/c no rules were discovered.

ness functions such as billing and inventory management. However, the SMEs could not review every rule due to limited availability and the large number of rules. Hence, we grouped together related rules (see Table 4), and selected representative examples from each group (and BU) for review (see Table 5).

The SMEs determined the *Meter Dial & Size*, *Bill & Read Cycle/Route*, and *Meter Disposition* groups were useful because they could detect problems with over and under billing, estimated meter reads (and hence revenue loss), and inaccurate meter inventory (and hence operational inefficiency) respectively. These three groups account for 44.9% of all discovered rules.

Week 4: We applied rules from the three groups above – using a commercial data quality solution – to their respective data set (and BU) to determine the efficacy of these rules in detecting DQ issues.³ We then reported (and validated) the resulting DQ issues with the same group of LUC SMEs. We present a selection of these issues below.

Table 6 shows the result of applying *Meter Dial & Size* rules to meter data to detect incorrectly recorded meter dial numbers. This number is used in billing the customer associated with the meter. If the recorded number is more (or less) than the actual number, then the customer is over (or under) billed. The discovered rules validated that the majority of dial numbers were recorded correctly. They also detected 1,092 incorrect instances (and hence 1,092 incorrect billings). Moreover, LUC has invested significant resources to eliminate this problem, but it still exists – as shown by this analysis. LUC has incorporated these rules into the migration process to validate meter dial numbers.

Table 7 shows the result of applying *Bill & Read Cycle/*

³Ideally, we would discover rules from a training set and apply the discovered rules to a hold out set. However, LUC wanted to perform both discovery and detection on the entire data set.

BU	# Actual	# Estimated
U004	66,499	26,636
U050	252,039	124
U051	90,094	13
U060	105,733	3,056
U127	40,990	933
U303	76,005	245

Table 7: No. of actual vs. estimated meter reads.

BU	# In Service	# In Inventory
U004	19,035	16,928
U050	192,661	122,393
U051	72,997	44,589
U060	84,070	54,977
U127	36,812	20,776
U303	36,248	41,945

Table 8: No. of in-service vs. in-inventory meters.

Route rules to service point data to detect estimated meter reads. If the bill cycle code does not align with the read route code, then LUC will estimate the meter usage instead of performing an actual read, which results in inaccurate customer billing. The discovered rules validated that the majority of readings are actual. They also detected a large number of estimated reads.

An estimated read is required when a meter cannot be accessed physically, but these cases are infrequent. The more common causes are either incorrect recording of bill cycle and read route codes or issuing a temporary estimated read (due to severe weather) but failing to realign the codes afterwards. LUC's goal is to systematically identify (and investigate) all estimated reads, and the discovered rules were able to accelerate this goal.

Table 8 shows the result of applying *Meter Disposition* rules to meter inventory data to detect inaccurate inventories. These rules detected a large number of meters were recorded as being in inventory, which exceeded LUC SMEs' expectation. The SMEs investigated this discrepancy, and confirmed that many meters have incorrect dispositions. For example, some decommissioned meters were recorded as being in inventory (and available for installation), which can lead to operational inefficiencies when a crew requests a non-existent meter from inventory to install in the field. These kinds of problems occurred in all 6 BUs. Hence, the discovered rules

helped LUC SMEs uncover a systemic problem with the accuracy of their meter disposition data (and hence meter inventory), which directly impacts operational efficiency.

Deployment Efforts

We describe three ongoing, parallel efforts to deploy DQRA. First, over 40% of DQ rules discovered during the case study are currently being used by LUC as part of their legacy migration effort. For example, LUC is currently using *Meter Dial & Size* rules discovered by DQRA to flag (and prevent) records with inconsistent meter dial numbers from entering their new platform. Moreover, the case study demonstrated to LUC the value of DQRA and the needed for a comprehensive DQ assessment despite tight project timelines. LUC is currently in discussions with the Accenture client team to expand the scope of the legacy migration effort to apply DQRA to all table families across all 31 business units.

Our evaluations also demonstrated the value of DQRA for Accenture and its client teams. DQRA provides a technology differentiation that can reduce the effort required to develop domain (and client) specific DQ rules (and hence accelerate DQ efforts). In an effort to deploy DQRA to the rest of Accenture, Accenture Technology Labs (ATL) is transferring it to Accenture's Product and Offering Development (P&OD) group – an organization focused on building and maintaining Accenture offerings and their associated assets such as software, processes, and delivery capabilities. Specifically, ATL is transitioning DQRA to an India-based P&OD team that is developing a new DQ offering through the integration of various DQ assets such as DQRA, DQ processes, and offshore delivery capabilities. The resulting offering will provide an end-to-end service that Accenture (and its client teams) can provide to clients for DQ assessment, cleansing, and governance. The transition process includes various activities such as:

- Creation of technical documents and training materials
- Functional training of offshore resources on the DQRA
- Provisioning of hardware and application installation
- Code transfer and walkthrough
- Creation of feature extension roadmap

At time of writing, the ATL and P&OD teams completed the first three items above, and have started the fourth item.

Finally, we are exploring how DQRA can provide a value-added capability on top of commercial DQ solutions such as Informatica and Trillium. These solutions provide a platform for authoring and applying DQ rules, but they do not provide the rules themselves. Although these solutions can profile data out of the box, profiling alone cannot detect the domain (and client) specific inconsistencies uncovered by DQRA in our evaluations. For example, profiling cannot determine whether the meter dial number for a particular record is consistent. It can only reveal that the value for a meter dial must be a number within a particular range. Hence, DQRA can complement existing commercial DQ solutions by automatically discovering domain (and client) specific DQ rules that can feed into these solutions.

Conclusion

In this paper, we presented an application – Data Quality Rules Accelerator (DQRA) – that accelerates Data Quality (DQ) efforts (e.g. data profiling and cleansing) by automatically discovering DQ rules for detecting inconsistencies in data. We then presented two evaluations of our application. The first evaluation compared DQRA to existing solutions; and showed that DQRA either outperformed or achieved performance comparable with these solutions on metrics such as precision, recall, and runtime. The second evaluation was a case study where DQRA was piloted at a large utilities company to improve data quality as part of a legacy migration effort. The DQRA significantly reduced the effort required to develop DQ rules compared to Accenture's existing DQ process. The DQRA also discovered rules that detected DQ issues related to inaccurate customer billing and meter inventory, which directly impacted revenue and operational efficiency respectively. We concluded by describing ongoing efforts to deploy DQRA.

Acknowledgments

We want to thank the reviewers for their helpful comments. We especially want to thank R. Uthurusamy for his help and suggestions for improving this paper. We also want to thank Ryan Brook, Brian Cone, and Geoffrey Plese from the Accenture client team along with Steven Friedman and the other LUC SMEs for their assistance during the case study. Finally, we want to thank Scott Kurth and Sanjay Mathur from Accenture Technology Labs for their contributions.

References

- Agrawal, R., and Srikant, R. 1994. Fast algorithms for mining association rules in large databases. In *VLDB*.
- Bohannon, P.; Fan, W.; Geerts, F.; Jia, X.; and Kementsietidis, A. 2007. Conditional functional dependencies for data cleaning. In *ICDE*.
- Brin, S.; Motwani, R.; Ullman, J.; and Tsur, S. 1997. Dynamic itemset counting and implication rules for market basket data. In *SIGMOD*.
- Chiang, F., and Miller, R. 2008. Discovering data quality rules. In *VLDB*.
- Eckerson, W. 2002. Data quality and the bottom line. Technical report, TDWI Report Series.
- Fan, W.; Geerts, F.; Lakshmanan, L.; and Xiong, M. 2009. Discovering conditional functional dependencies. In *ICDE*.
- Golab, L.; Karloff, H.; Korn, F.; Srivastava, D.; and Yu, B. 2008. On generating near-optimal tableaux for conditional functional dependencies. In *VLDB*.
- Huhtala, Y., et al. 1998. Efficient discovery of functional and approximate dependencies using partitions. In *ICDE*.
- Informatica. www.informatica.com.
- Trillium. www.trilliumsoftware.com.
- Yeh, P., and Puri, C. 2010. Discovering conditional functional dependencies to detect data inconsistencies. In *QDB*.