Cleaning Data with Constraints and Experts

Ahmad Assadi
Tel Aviv University
ahmadassadi@mail.tau.ac.il

Tova Milo
Tel Aviv University
milo@post.tau.ac.il

Slava Novgorodov
Tel Aviv University
slavanov@post.tau.ac.il

ABSTRACT

Popular techniques for data cleaning use integrity constraints to identify errors in the data and to automatically resolve them, e.g. by using predefined priorities among possible updates and finding a minimal repair that will resolve violations. Such automatic solutions however cannot ensure precision of the repairs since they do not have enough evidence about the actual errors and may in fact lead to wrong results with respect to the ground truth. It has thus been suggested to use *domain experts* to examine the potential updates and choose which should be applied to the database.

However, the sheer volume of the databases and the large number of possible updates that may resolve a given constraint violation, may make such a manual examination prohibitory expensive. The goal of the DANCE system presented here is to help to optimize the experts work and reduce as much as possible the number of questions (updates verification) they need to address. Given a constraint violation, our algorithm identifies the *suspicious tuples* whose update may contribute (directly or indirectly) to the constraint resolution, as well as the possible dependencies among them. Using this information it builds a graph whose nodes are the suspicious tuples and whose weighted edges capture the likelihood of an error in one tuple to occur and affect the other. PageRank-style algorithm then allows us to identify the most beneficial tuples to ask about first. Incremental graph maintenance is used to assure interactive response time. We implemented our solution in the DANCE system and show its effectiveness and efficiency through a comprehensive suite of experiments.

1 INTRODUCTION

Data cleaning is a long-standing problem that has attracted much research interest in the past years in the databases community. Many key business decisions are made based on underlying databases. Yet, real-life databases sometimes contain incomplete, wrong or inconsistent data, that may lead to incorrect output and bad decision making. Consequently, much effort has been targeted to the development of techniques to clean the underlying data.

Popular techniques for data cleaning use data-integrity and consistency rules to identify errors in the data and to automatically resolve them, e.g. by finding a *minimal repair* that will resolve the constraints violation [23], or by using predefined *priorities* among possible resolutions [15]. Such automatic solutions, however, cannot ensure the precision of the repairs since they do not have enough evidence about the actual errors and thus may, in fact, lead to wrong results with respect to the ground truth. In order to overcome the limitations of such automatic techniques it has been suggested to use *domain experts* that have extensive knowledge about the ground truth, to examine the potential updates and choose which should be applied to the database [9, 15, 20]. However, the sheer volume of the databases and the large number of possible updates that may resolve a given constraint violation, may make such a manual examination prohibitory expensive. The goal of the DANCE system presented here is to help to optimize the experts work and reduce as much as possible the number of questions (updates verification) they need to address. As we will describe, our algorithms effectively prune the search space to minimize the amount of interaction with the experts while, at the same time, try to maximize the potential "cleaning benefit" derived from the expert answers. DANCE can be used to optimize the initial cleaning of a database as well as to assist in its ongoing maintenance - whenever a constraint violation is reported, DANCE can take over to efficiently clean the underlying database by interacting with the experts.

Given a constraint violation, our algorithm first identifies the tuples in the database whose update may contribute (directly or indirectly) to the constraint resolution. We call those *suspicious tuples*. Database constraints may be inter-related and thus when analyzing a constraint violation, these relationships must be taken into consideration. To determine which tuples should be considered first, we examine for each suspicious tuple t (1) the potential effects of updates to t, namely what tuples may potentially become non-suspicious if t is found to be incorrect and correspondingly updated/removed, (2) the number of potential updates (attribute errors) to t that may lead to such an effect, and (3) the uncertainty, if known, for the values in the database relation to which t belongs. Using this information we build a graph whose nodes are the suspicious tuples and whose weighted edges capture the likelihood of an error in one tuple to occur and affect the other. PageRank-style algorithm is then used to identify the most beneficial tuples to ask about first.

Example 1.1. To illustrate let us consider the following simple example. The database in Figure 1 shows a portion of UEFA Champions League 2016/17 statistics database. The dark gray rows represent wrong tuples and lightgray rows represent missing tuples. The *Games* relation describes the results of a match between two teams, it stores the team's name, goals score and the stage. The *Teams* relation describes a football team, it stores the team name and country. The *Countries* relation describes the name of the country and number of teams that advanced to the group stage. We consider in our work integrity constraints described by standard tuple-generating and condition-generating dependencies [14]. The following two integrity constraints are relevant to this database: (i) two teams from the same country cannot play against each other on a group stage, and (ii) if a country has at least one representative, its team must appear in the teams table. These are captured by the following constraints.

- $Games(x_1, x_2, x_3, x_4, x_5) \wedge x_5 = \text{“}GroupStage\text{”} \wedge Teams(x_1, y_1) \wedge Teams(x_2, y_2) \rightarrow y_1 \neq y_2$
- $Countries(x_1, x_2) \wedge x_2 > 0 \rightarrow Teams(y_1, x_1)$

We assume that all the given constraints are correct and reflect the ground truth. In our running example, the constraints are derived

WebDB'18, June 10, 2018, Houston, TX, USA

ACM ISBN 978-1-4503-5648-0/18/06…$15.00
https://doi.org/10.1145/3201463.3201464

Games

team1	team2	t1_goals	t2_goals	stage
Celtic	Manchester City	3	3	Group Stage
Celtic	Hapoel Beer Sheva	5	2	Qualification

Teams

name	country
Celtic	UK
Manchester City	UK
Hapoel Beer Sheva	Israel
CSKA Moscow	Russia

Countries

name	num_of_teams
Israel	0
UK	5
England	4
Scotland	1

Figure 1: Sample of UEFA Champions League DB

from UEFA official regulation. Since the database is aggregated from multiple sources it contains mistakes and violates some of the constraints. One can notice for instance that the database mistakenly associates both the Celtic and the Manchester City football clubs to the United Kingdom. However, despite the fact that Celtic and Manchester City are actually located in the United Kingdom they belong to distinct federations (that represent Scotland and England separately), hence can play against each other.

When applying the integrity constraints to the database, we discover several inconsistencies. Each such inconsistency involves several tuples that when assigned together to the atoms in the body of the constraint yielded a constraint violation. For example, a violation of the first constraint involves a set of three tuples: *Games(Celtic, Manchester City, 3, 3, Group Stage)*, *Teams(Manchester City, UK)*, *Teams(Celtic, UK)*, whose existence in the database lead to the violation. Intuitively, each of the tuples is *suspicious* and at least one is wrong and needs to be updated/deleted (otherwise the constraint is incorrect which we assume is not the case). Also note that since the two constraints are inter-related, when a given tuple is suspicious other tuples become suspicious as well. Consider for example the second constraint, that requires that for each country in the *Countries* relation with a positive number of teams, there must be at least one team in *Teams* relation from this country. Relation *Countries* contains the tuple *(UK, 5)*, which enforces the existence of teams from United Kingdom. Since the *Teams(Celtic, UK)* and *Teams(Manchester City, UK)* tuples are *suspicious* (and may generally both be wrong), we may suspect also the tuple *Countries(UK, 5)*.

Which of these four suspicious tuples is more beneficial to consult about first with the expert? To determine this we build a directed graph whose nodes are the suspicious tuples and whose (weighted) edges capture the dependency between the suspicious tuples. Let β be the uncertainty of the values in the relation R to which a tuple t belongs to where β is between 0 (all the values are valid) and 1 (all the values are wrong). Intuitively, there is an edge from tuple s to t with a weight $n \times \beta$ if there are n attributes in t that one can change in order to eliminate at least one violating assignment that involves s. For example, data from official UEFA website will get β close to 0 while user-generated content in the other relations should get much more. We use 0.5 as a default value. The graph for the four tuples that we obtain is depicted in Figure 2 (ignore for now the number labels on the nodes).

Intuitively, to minimize the number of questions, we would like to catch early errors whose correction may have the largest effect. To decide which tuple to verify first, we process the graph using a PageRank-style [10] algorithm, to rank the nodes, and ask the experts about the nodes with the highest rank. When answers are gathered, the database is updated accordingly, and incremental computation is applied to update the graph and identify the next candidates. The resulting ranks for our running examples are depicted on the nodes, and so we will ask about C (which is indeed incorrect and will be removed, instead *(England, 4)* and *(Scotland, 1)* will be inserted by the expert), $T1$ (incorrect, updated to *(Celtic, Scotland)*) and $T2$ (incorrect, updated to *(Manchester City, England)*). G is then no longer suspicious and no constraint is violated.

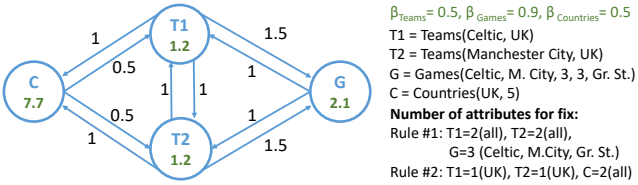


Figure 2: Suspicious tuples graph

Our contributions can be summarized as follows.

(1) We formulate and present a constraints based framework for data cleaning with experts. Under this framework, the database is updated by (minimally) interacting with domain experts in order to fix the violations of the constraints.
(2) To address this problem we focus on a *suspicious tuples* group of tuples, that are the potential cause of the constraints violation, which we infer by analyzing the available data and constraints.
(3) We present an effective algorithm that, using the suspicious tuples and the inferred dependencies among them, builds and incrementally maintains a weighted graph that captures the potential "cleaning benefit" that a correction/verification to one suspicious tuple may yield to its neighbors. PageRank-style ranking, applied to the graph, determines the order of questions issued to the experts.
(4) We have implemented our solution in the DANCE prototype system and applied it to real use cases, demonstrating the efficiency of our constraints-based approach using domain experts, showing how our algorithms consistently outperform alternative baseline algorithms, and effectively clean the data while asking fewer questions.

A first prototype of DANCE was demonstrated in [8]. The short paper accompanying the demonstration gave only a high level overview of the system's capabilities and user interface whereas the present paper details the model and algorithms underlying our solution as well their experimental evaluation.

Outline of paper. Section 2 provides the basic definition and formalisms. The graph construction and maintenance is explained in Section 3. The experimental results are described in Section 4. Related work is in Section 5, and we conclude in Section 6.

2 PRELIMINARIES

We will briefly present our preliminaries for the underlying model. For space constraints the presentation is rather intuitive and informal. Full definitions can be found in the technical report [7].

Database Let D be a relational database instance. To model real-world data, we adopt the truly open world assumption where a fact that is in D can also be true or false, in addition to the assumption that a fact that is not in D can be true or false. In other words, we assume that a given database can contain mistakes, in addition to being incomplete. The truth of a tuple is given by the ground truth database D_G that contains all true tuples and only them. Hence, a database D is dirty w.r.t. D_G if $D \neq D_G$.

Questions to the Expert For simplicity of presentation, we assume that there is an expert that has an extensive knowledge about the domain. Otherwise standard techniques [17] may be applied to aggregate multiple answers. There are two types of questions:

- *Update question:* the expert is asked to examine a database tuple t. The answer can be: (1) t is correct as is, i.e. $t \in D_G$, (2) t is wrong and should be deleted, i.e. $t \notin D_G$ or (3) Update the tuple t to tuple t', i.e. $t \notin D_G \wedge t' \in D_G$.

- *Fill question:* the system decides (e.g. based on a constraint) to add new tuple to the database. Some of the fields are filled automatically and the expert is asked to complete the rest.

Constraints The integrity constraints are database assertions that are similar to the standard tuple-generating and equality-generating dependencies [5].

Tuple-Generating Constraints (tgcs) are in the spirit of the *tuple-generating dependencies with arithmetic comparisons* from [5]. The tgcs are a first order logic formulas of the form:

$$\forall x_1, ..., x_n \varphi(x_1, ..., x_n) \rightarrow \exists z_1, ..., z_m R(x_1, ..., x_n, z_1, ..., z_m)$$

The left hand side (LHS) of the implication, φ, is a conjunction of relational atoms and conditions (boolean expression of the form $v\ op\ w$ where v, w are variables or constants and *op* is a boolean operation defined on the variables' domain).
The right hand side (RHS) contains only one relational atom R. Intuitively, given tuples satisfying the constraint of the LHS, tgcs asserts existence of a tuple in the RHS.
Condition-Generating Constraints (cgcs) have the same form as tgcs, but the RHS is a conjunction of conditions, and are defined as the *arithmetic-comparison-generating dependencies* in [5].

The constraints from Example 1.1 are cgc and tgc respectively.

Assignments and constraints satisfaction An assignment v for a constraint φ is a mapping from the constraint variables to constants. An assignment v satisfies a relational atom R iff $R(v) \in D$, denoted by $v \vDash_D R$. In a similar way an assignment v satisfies a constraint $\varphi \rightarrow \psi$ iff $v \nvDash_D \varphi$ or $v \vDash_D \varphi$ and $v \vDash_D \psi$, denoted by $v \vDash_D \varphi \rightarrow \psi$. Database D satisfies a constraint φ iff for any assignment v it holds that $v \vDash_D \varphi$.

Violations, proofs and suspicious tuples

The cleaning process is triggered when the given set of constraints is not satisfied. To clean D, we identify the *suspicious tuples* that may be (directly or indirectly) the cause of the problem. We first define the *violation set* - a set of tuples that is a direct cause of a constraint violation. Next, we define the different types of *proof tuples* - the tuples that (through the same or other constraints) assert the existence of some violating sets members. The suspicious tuples are then the union of the violation and proof tuples. We give here intuitive description and the formal definition appear in the technical report [7].

Violation sets A *violation set* of a constraint φ in a database D is a minimal set of tuples in D that implies existence of an assignment v (from the tuples values) that is not satisfying the given constraint φ. Intuitively, each violation set is a set of tuples that caused the database D to violate the constraint φ.

For a set of constraints, the violation set is the union of the violation sets of each individual constraints.

Example 2.1. Consider the database and the first constraint from Example 1.1: two teams from the same country cannot play against each other on a group stage. Therefore, the set of tuples *{Games(Celtic, M.City, 3, 3, Group Stage), Teams(Celtic, UK), Teams(M.City, UK)}* is a violation.

Tuple Values Proof Let $t = (v_1, ..., v_n)$ be a tuple in D and let $\hat{t}$ be a subset of elements of t. Let φ be a tgc. Intuitively, a set of tuples $\{t_1, ..., t_k\}$ "proves" the validity of the values of $\hat{t}$ in t if, by using only φ and the assumption that the tuples $\{t_1, ..., t_k\}$ are valid, we can conclude that the values of $\hat{t}$ in t are also valid.

Recall Example 1.1, let $\hat{t} = (UK)$ and t= Teams(Celtic, UK). The set of tuples {Countries(UK, 5)} is part of tuple value proof of $\hat{t}$ since the tuple Countries(UK, 5) implies the existence of the value UK in t.

Relevant proofs Note that not all attribute values (and their proofs) are suspicious. As we interact with the expert, some attribute values may be validated, either by verifying tuple as correct, or during the fill-up question. We will not exclude them (and their proofs) from the suspicious set.

Moreover, not all the attributes of the violation tuples contribute to the violation. For a constraint φ, we call the variables that appear in a conditional atom as conditional variables. The values assigned to the conditional variables are *conditional values*.

As a result, in order to resolve the violation T, the expert must perform one of the three following actions: (i) remove at least one of the tuples in T that is responsible for the satisfaction of the constraint's body, (ii) insert a tuple which completes the incomplete tuple of T (if φ is a tgc) or (iii) update the conditional values of T.

To summarize, for a tuple t and a constraint φ, we are interested only in tuple value proofs of its conditional attributes whose value has not been verified yet. Note that when all the conditional values of a tuple t are validated means that t is no longer be responsible for the violation of φ.

Suspicious tuples For a database D and a set f of constraints, the set of suspicious tuples includes all the tuples in the proofs, excluding those that have already been validated by the expert (through update questions). Note that the *violation sets* are included in the *suspicious tuples* as they are part of the *proofs*.

3 BUILDING THE TUPLES GRAPH

As mentioned in the Introduction, to determine which tuples should be considered first (the next question that will be posed to the expert), we build a directed graph with nodes that are the suspicious tuples and weighted edges that capture the likelihood of an error in one tuple to affect the other. We call this graph the *tuples graph*. PageRank style algorithm is then applied to the graph to identify the most beneficial tuples to ask about first.

Vertices and edges The graph vertices V are the set of suspicious tuples. The graph edges capture the potential effect of updates to t, namely what tuples may potentially become non-suspicious if t is found to be incorrect and correspondingly updated/removed.

Consider two suspicious tuples t_{src}, t_{dst} and their corresponding vertices $v_{t_{src}}$, $v_{t_{dst}}$. We include in E an edge $e = (v_{t_{src}}, v_{t_{dst}})$ if and only if the tuple t_{dst} could cancel at least one proof/violation set T that contains t_{src}. Intuitively, it happens when both tuples participate in T.

In particular all edges are bidirectional, and the tuples graph is a union of a collection of cliques where each clique is defined by some violation/proof set.

Recall Example 2.1, the set of tuples *{Games(Celtic, M. City, 3, 3, Group Stage), Teams(Celtic, UK), Teams(M. City, UK)}* is a violation and the tuple *Countries(UK, 5)* is a proof of the tuples *Teams(Celtic, UK)* and *Teams(M. City, UK)* by the second constraint. Therefore, the four tuples *Teams(M. City, UK)*, *Teams(Celtic, UK)*, *Countries(UK, 5)* and *Games(Celtic, M. City, 3, 3, Group Stage)* are suspicious. Figure 2 depicts the Tuples Graph of these four suspicious tuples. The tuples *T1*, *T2* and *G* are connected to each other because of the violation {*G, T1, T2*}. Since *T1* and *T2* may have been generated from *C* by second constraint, *C* is a proof of *T1* and *T2*, implying the connection between *T1*, *T2* and *C*. The graph weights explained below.

Edge weights The edge weights capture the likelihood of an error in one tuple to occur and to affect the other. Intuitively, the weight w_e assigned to an edge $e = (v_{t_{src}}, v_{t_{dst}})$ is $n \times \beta$ where β is the uncertainty measure of the relation of the tuple, and n is overall number of attributes in t_{dst} that one can update in order to resolve the violation. n is calculated using function that checks for each *conditional attribute* $attr$, whether there is an update to the tuple t, yielding a tuple t' that differs from t at the attribute $attr$, s.t. the violation set without $\{t\} \cup \{t'\}$ is no longer a violation/proof set. The full details are omitted here for space constraints and provided in [7].

Node Weights Finally, to decide which tuple to verify first, we process the graph using a PageRank-style algorithm [10], to rank the nodes, and ask the experts about the nodes with the highest rank. Intuitively, the higher rank for a tuple captures the potential for higher influence in terms errors (violations) elimination.

To complete our running example, Figure 2 depicts the tuples graph. The edge weights are calculated with $\beta = 0.5$ for all relations. For instance, the weight of the edge from G to $T1$ is 1.5 because there are 3 values in G that can be updated in order to cancel the violation (the values are "ManCity", "Celtic", "Group Stage") and $\beta = 0.5$. The node weights are their ranks after running our PageRank algorithm on the graph. The node C has the highest rank (7.7). Therefore we ask the experts about tuple C.

Incremental maintenance and optimizations As more answers obtained, the graph is updated to reflect the current database state and the (remaining) violations. Inferring validated values, parallel computation and other technique are used. Details in [7]

4 EXPERIMENTS

We have implemented the DANCE prototype system, using Java and SQLite as the DBMS. All experiments were executed on an Intel i7 2.4Ghz with 16GB RAM. We run experiments over real-life data sets and examined the system performance both in terms of the number of questions posed to the experts and the running time, measuring contribution of each component of our solution.

Algorithms: The main algorithm in DANCE builds the tuples graphs and ranks the nodes so that their rank reflects their potential importance for the database cleaning. This is achieved by assigning to the edges weights that not only reflect the potential influence tuple updates but also the uncertainty of the values in the corresponding relations. To assess the importance of this ranking, we compare DANCE to three alternatives alternative algorithms.

- Random: a naïve algorithm that randomly picks tuples in the graph to ask about
- DANCE v1: a simplified version of DANCE where all edges are assigned an equal weight (equals to 1)
- DANCE v2: a simplified version of DANCE where the uncertainty of the values of all the relations are the same (by setting $\beta = 0.5$ for all relation).

We will see that the full fledged algorithm yields fewer questions than its restricted variants.

We have also compared DANCE to a related previous work, described next. We note that data cleaning with the help of experts has been previously considered in [9] by a subset of the authors, where the goal was to update the database for eliminating incorrect query answers. The problem studied there was simpler because it does not consider transitive dependencies as the ones entailed by constraints. It is easy to show that an assertion that a given tuple should not be included in the query result can be expressed as simple constraint violation in our formalism (we omit the translation algorithm for space constraints). We are thus able to compare the performance of DANCE to that of QOCO for solving the same problem. Since QOCO allows users only to add and delete tuples, whereas DANCE allows also tuple update, we also examine here a restricted variant of DANCE :

- DANCE v3: a simplified version of DANCE that does not include tuple updates

Datasets, Constraints and Queries: We consider three datasets. The first dataset is a soccer-related. It contains information about World Cup games, goals, players, teams, etc. and consists of around 5000 tuples. The teams and players relations are derived from the FIFA official data [1] and are thus assigned $\beta = 0$. The games relation is derived using automatic website scraping tools from sites such as [4] and other similar sources. We first cleaned the database by comparing the games data with reference data from FIFA official data and used the cleaned database as our ground truth database, with the expert answers following this ground truth. Sampling the games data and comparing to the ground truth we derived an uncertainty measure β of 30%. We have experimented with various integrity constraints based on FIFA competition rules and show here the results for the following representative constraints, informally described below.

- φ_1^{WC}: If teams scores are not equal, then penalties are 0.
- φ_2^{WC}: If penalties are not equal, then the scores are equal.
- φ_3^{WC}: If the winning and losing teams penalties are equal, then the winning team score is bigger than the losing team.

For the comparison with QOCO we examine the following two queries and what it takes to remove wrong tuples from the result.

- Q_1^{WC}: All games between a Asian and any other country.
- Q_2^{WC}: All games of 'Round of 16' without penalties.

Since QOCO does not exploit constraints, to make the comparison as "fair" as possible, we assume in this experiment no constraints (other than the assertions on the erroneous query answers).

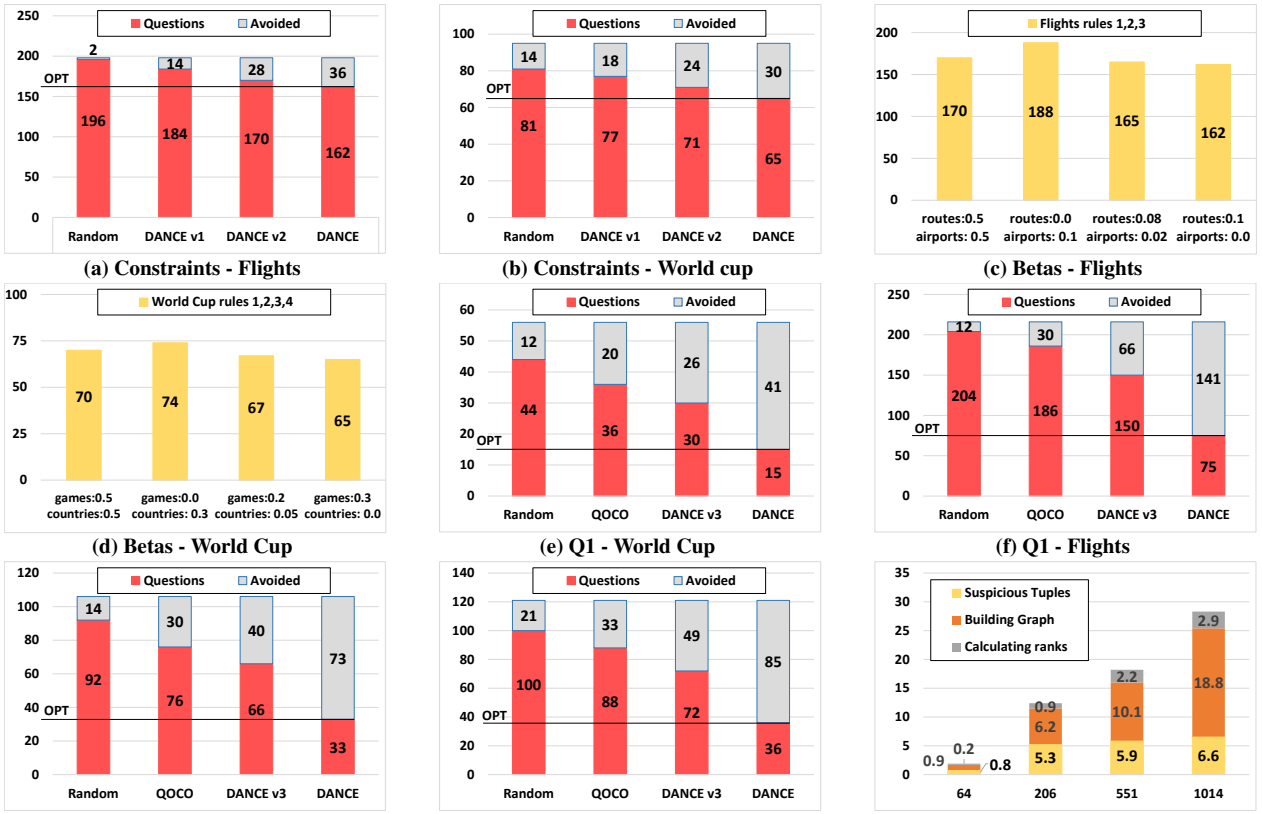
Our second dataset is a flights database from [3]. This database records information about flights all around the world, and was last updated on 2012. It contains data about flights (68K tuples is a routes relation), airports (8.2K tuples in an airports relation) and airlines (6.1K tuples in an airlines relation). We first cleaned the flights database by comparing the data with current reference data from Google flights website [2] and used it as our ground truth database, again, with the expert answers corresponding to this ground truth. By sampling the data and comparing to the ground truth we derived the uncertainty measures β for routes, airlines and airports to be 10%, 5% and 0% respectively. For our experiments we used the following three real-life constraints that follow from the fact that after 2012 there was a political conflict between Russia and each of Ukraine, Egypt and Turkey that caused the cancellation of the direct flights between that countries.

- φ_1^{Fl}: There are no direct flights between Russia and Ukraine.
- φ_2^{Fl}: There are no direct flights between Russia and Egypt.
- φ_3^{Fl}: There are no direct flights between Russia and Turkey.

For the comparison with QOCO we examine here the following two representative queries (again, assuming no constraints).

- Q_1^{Fl}: All direct flights to China or to Greenland.
- Q_2^{Fl}: All direct flights from Russia or from USA.

To test scalability of performance, we used a third dataset which was synthetically constructed by taking the flights databse mentioned above and replicating data (with variations) to achieve a 400K tuples dataset. Each tuple was replicated between 3 to 6



(a) Constraints - Flights (b) Constraints - World cup (c) Betas - Flights

(d) Betas - World Cup (e) Q1 - World Cup (f) Q1 - Flights

(g) Q2 - World Cup (h) Q2 - Flights (i) runtime(sec)/#violations on extended data

Figure 3: Experimental results

times, while generating new unique primary keys (airport ID for airports, pair of source and destination airport IDs for the flight, etc), by padding a number between 1-6 to the original key. The constraints used for the third dataset experiments are similar to the second dataset, with the addition of an extra constraint:

- φ_4^{Fl}: There are no 2 different airlines with same code.

This constraint is added especially to stress-test the system, since it has many violations in the dataset.

Results: Our first experiment compares the performance of DANCE, in terms on the number of questions posed to the experts, to that of Random and the restricted variants DANCE v1 and v2. The results for the two datasets are depicted in Figures 3a and 3b. In both figures, each vertical bar corresponds to one of the algorithms. The height of each bar shows the maximal number of possible questions (the number of suspicious tuples at the beginning of the experiment). The lower part of the bar (in red) denotes the number of questions asked by the algorithm to fix all violations. The horizontal (black) line indicates the number of questions the would have been asked by an optimal algorithm that knows the underlying ground truth and asks only about the actual erroneous tuples. In both cases Random shows the worst performance, then come DANCE v1, v2 and finally DANCE. This demonstrates the importance of the ingredients in our solution.

To better understand the results we examined the effect of choosing appropriate β values on the performance of our algorithms. The results of varying β values are depicted in Figures 3c and 3d. We can see that while the use of β values that reflect the uncertainty measure is useful, rough estimation suffices for obtaining good results.

Our second experiment compares the performance of DANCE, with and without tuples updates, to that of QOCO, for the queries listed above. To get a clearer perspective on the performance of the algorithm, we also add the results of Random for the same problems. The results are depicted in Figures 3e-3h. The bars for each of the algorithm have the same structure as in the first experiment and the horizontal black line indicates again the optimum. We can see that both variants of DANCE perform better than QOCO. This is interesting since the improved performance is achieved even without this use of additional constraints. (In the presence of constraints the gap grows. We omit the results here). We can also see that allowing users to update tuples, rather than only add and delete, results in fewer needed updates.

To conclude this section we examine running time of DANCE as a function of the number of suspicious tuples. We consider here the extended flights database which contains around 400K tuples. The results are shown in Figure 3i. To vary the number of suspicious tuple have run four experiments each with a different set of constraints (φ_1^{Fl} in the 1st bar of figure, φ_1^{Fl} and φ_2^{Fl} in the 2nd bar, $\varphi_1^{Fl} - \varphi_3^{Fl}$ in the 3rd, and all the four constraints in the last bar). For each experiment we have measured the run time of the main algorithm in seconds from its start until finding the first question that will be posed to the expert. The number of suspicious tuples in each experiment (column) are depicted on the X axis. In each case we also detail the time spent on each part of the algorithm. As expected the time grows with the number of suspicious tuples, but in all our experiments was below 30 seconds. The iterations took just 1 to 4 seconds in all cases, due to our incremental graph maintenance, thus sufficiently fast to maintain an interactive experience and work as a real time cleaning system.

5 RELATED WORK

Data cleaning has attracted much attention in recent years. A large set of work focuses on fully-automated cleaning, using dedicated object similarity measures, probabilistic and statistical methods, and machine learning techniques [11, 16, 19, 23]. The problem with automatic solutions is that they cannot ensure precision of the repairs since they do not have enough evidence about the ground

truth and may lead to wrong results [9]. Hence it suggested to use experts to examine the data and choose which updates to apply.

Multiple data cleaning tools leverage the crowd to assist in data cleaning (e.g. [6, 9, 13, 15, 18, 20]), typically using the crowd to identify problematic spots in the data, e.g. by running queries and validating the results or by iteratively generating cleaning task for the crowd. [20] introduces the idea of cleaning only a sample of data to obtain unbiased query results with confidence intervals. [9] uses experts to identify errors in query answers and attempts to minimize the number of posed questions. However, as mentioned in Section 4, ignores the databases constraints and its performance is inferior than ours even in the absence of constraints. Our work complements these previous efforts by using the set of integrity constraints to identify data errors and to effectively use the experts.

Several data cleaning tools employ integrity constraints in the cleaning process (e.g. [6, 12, 15, 18, 19]). Some of the papers (e.g. [21]) rely on high quality reference data. Others are fully automatics (hence suffer from the problems mentioned above) and use predefined preferences among updates and/or minimal-repair strategy. When no unique update may be inferred from the available preferences, systems like [15] turn to experts to assist in the constraint resolution. But they not optimize the experts exploration of the possible updates space. Our work may be integrated into such systems to optimize the experts work in such scenarios. The authors of recent related research [18] propose a framework for detecting functional dependencies (FDs) violations. Their main focus is finding the (subset of) FDs that can detect the errors and studying different types of questions that can be asked from the experts under a limited budget (e.g. verifying if the proposed FD is correct) in order to detect the errors in the data. Our efforts are complimentary, since we are not focusing on identifying FDs and only detecting the data errors, but given a set of FDs we are trying to find and also fix underlying violations.

The very recent [6] is the most related to our work. They study user-guided cleaning of Knowledge Bases w.r.t violations of tgds and a subset of denial constraints, called contradiction detecting dependencies. While they too handle tgds these include only equalities and they do not support cgds. This, together with the different data model, make the works incomparable but complementary.

Crowdsourcing, using ordinary users and domain experts, has been an active field of research in recent years, being employed for a variety of cleaning-related tasks such as entity resolution [22] and schema matching [24]. Our system may be used to resolve violations generated by these methods.

6 CONCLUSIONS

We presented DANCE, a system that assists in the efficient resolution of integrity constraints violation. DANCE identifies the suspicious tuples whose update may contribute to the violation resolution, and builds a graph that captures the likelihood of an error in one tuple to occur and affect the other. PageRank-style algorithm identifies the most beneficial tuples to ask about first. Incremental graph maintenance is used to assure interactive response time. Our experimental results on several different real-world datasets demonstrate the promise that DANCE is an effective and efficient tool for data cleaning.

There are several directions for future research. Supporting a richer constraint language, and in particular constraints on aggregations (i.e. a team cannot have more then 23 players in the World Cup) is challenging. Violations of such constraints may be corrected in multiple ways, hence it is interesting to find the most efficient way. Also, we plan to integrate our approach with mechanisms that infer additional constraints or corrections to existing constraints, possibly with the help of the experts. Parallel processing for speeding up the computation is another intriguing future direction.

Acknowledgements Partially funded by the European Research Council under the FP7, ERC grant MoDaS, agreement 291071, and by grants from Intel and the Blavatnik Cyber Security center.

REFERENCES

[1] Fifa official site. http://www.fifa.com/.
[2] Google flights. https://www.google.com/flights/.
[3] Open flights. http://openflights.org/data.html.
[4] World cup history. http://www.worldcup-history.com/.
[5] F. N. Afrati, C. Li, and V. Pavlaki. Data exchange in the presence of arithmetic comparisons. In *EDBT*, 2008.
[6] A. Arioua and A. Bonifati. User-guided repairing of inconsistent knowledge bases. In *EDBT*, 2018.
[7] A. Assadi, T. Milo, and S. Novgorodov. DANCE - Technical Report. http://slavanov.com/research/dance-tr.pdf.
[8] A. Assadi, T. Milo, and S. Novgorodov. DANCE: Data Cleaning with Constraints and Experts. In *ICDE*, 2017.
[9] M. Bergman, T. Milo, S. Novgorodov, and W. C. Tan. Query-oriented data cleaning with oracles. In *SIGMOD*, 2015.
[10] S. Brin and L. Page. The anatomy of a large-scale hypertextual web search engine. *Computer Networks*, 30, 1998.
[11] F. Chiang and R. J. Miller. A unified model for data and constraint repair. In *ICDE*, pages 446–457, 2011.
[12] X. Chu, I. F. Ilyas, and P. Papotti. Holistic data cleaning: Putting violations into context. In *ICDE*, 2013.
[13] X. Chu, J. Morcos, I. F. Ilyas, M. Ouzzani, P. Papotti, N. Tang, and Y. Ye. KATARA: A data cleaning system powered by knowledge bases and crowdsourcing. In *SIGMOD*, 2015.
[14] R. Fagin, P. G. Kolaitis, R. J. Miller, and L. Popa. Data exchange: semantics and query answering. *Theor. Comput. Sci.*, 336(1):89–124, 2005.
[15] F. Geerts, G. Mecca, P. Papotti, and D. Santoro. The LLUNATIC data-cleaning framework. *PVLDB*, 6(9), 2013.
[16] S. Krishnan, J. Wang, E. Wu, M. J. Franklin, and K. Goldberg. Activeclean: interactive data cleaning for statistical modeling. *PVLDB*, 9(12):948–959, 2016.
[17] V. C. Raykar, S. Yu, L. H. Zhao, A. K. Jerebko, C. Florin, G. H. Valadez, L. Bogoni, and L. Moy. Supervised learning from multiple experts: whom to trust when everyone lies a bit. In *ICML*, pages 889–896, 2009.
[18] S. Thirumuruganathan, L. Berti-Equille, M. Ouzzani, J.-A. Quiane-Ruiz, and N. Tang. Uguide: User-guided discovery of fd-detectable errors. In *SIGMOD*, pages 1385–1397, 2017.
[19] M. Volkovs, F. Chiang, J. Szlichta, and R. J. Miller. Continuous data cleaning. In *ICDE*, pages 244–255, 2014.
[20] J. Wang, S. Krishnan, M. J. Franklin, K. Goldberg, T. Kraska, and T. Milo. A sample-and-clean framework for fast and accurate query processing on dirty data. In *SIGMOD*, 2014.
[21] J. Wang and N. Tang. Towards dependable data repairing with fixing rules. In *SIGMOD*, pages 457–468. ACM, 2014.
[22] S. E. Whang, P. Lofgren, and H. Garcia-Molina. Question selection for crowd entity resolution. *PVLDB*, 6(6), 2013.
[23] M. Yakout, L. Berti-Equille, and A. K. Elmagarmid. Don't be scared: use scalable automatic repairing with maximal likelihood and bounded changes. In *SIGMOD*, 2013.
[24] C. J. Zhang, Z. Zhao, L. Chen, H. V. Jagadish, and C. C. Cao. Crowdmatcher: crowd-assisted schema matching. In *SIGMOD*, pages 721–724, 2014.