Explanations for Data Repair Through Shapley Values

Daniel Deutch
Tel Aviv University
danielde@tauex.tau.ac.il

Nave Frost
Tel Aviv University
navefrost@mail.tau.ac.il

Amir Gilad
Duke University
agilad@cs.duke.edu

Oren Sheffer
Tel Aviv University
orensheffer@mail.tau.ac.il

ABSTRACT

Data repair, i.e., the identification and fix of errors in the data, is a central component of the Data Science cycle. As such, significant research effort has been devoted to automate the repair process. Yet it still requires significant manual labor by the Data Scientists, tweaking and optimizing repair modules (up to 80% of their time, according to surveys).

To this end, we propose in this paper a novel framework for explaining the results of any data repair module. Explanations involve identifying the table cells and database constraints having the strongest influence on the process. Influence, in turn, is quantified through the game-theoretic notion of Shapley values, commonly used for explaining Machine Learning classifier results. The main technical challenge is that exact computation of Shapley values incurs exponential time. We consequently devise and optimize novel approximation algorithms, and analyze them both theoretically and empirically. Our results show the efficiency of our approach when compared to the alternative of adapting existing Shapley value computation techniques to the data repair settings.

CCS CONCEPTS

• **Information systems** → *Integrity checking*; *Relational database model*; **Inconsistent data**; **Data provenance**; *Incomplete data.*

KEYWORDS

Data repair, Explainability, Shapley value, Denial constraints

ACM Reference Format:
Daniel Deutch, Nave Frost, Amir Gilad, and Oren Sheffer. 2021. Explanations for Data Repair Through Shapley Values. In *Proceedings of the 30th ACM International Conference on Information and Knowledge Management (CIKM '21), November 1–5, 2021, Virtual Event, QLD, Australia.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3459637.3482341

1 INTRODUCTION

There is a wide variety of data repair algorithms [3, 8, 13, 17, 20, 21, 29, 36]. For example, Holoclean [36] uses a graphical Machine Learning (ML) model to infer repairs, NADEEF [13] focuses on repairs based on a conflict hypergraph, Llunetic [20] performs a repair process that is based on the chase algorithm, and Livshits et. al. [29] computes the repair by repeatedly splitting the database table and eliminating attributes in the constraints.

CIKM '21, November 1–5, 2021, Virtual Event, QLD, Australia

ACM ISBN 978-1-4503-8446-9/21/11…$15.00
https://doi.org/10.1145/3459637.3482341

Users can, therefore, set the constraints and choose a repair algorithm based on their domain knowledge. These choices are highly non-trivial and even after executing the algorithm, the users may wonder whether they have chosen correctly and whether the data was repaired correctly. *To this end, in this paper we focus on generating explanations for black box repair algorithms, using both the constraints and the cells that have the highest influence on the repair.* Such explanations give a holistic view that allows users to understand the repair process. Our solution focuses on Denial Constraints (DCs), a formalism that was shown to be expressive enough in many realistic cases [12, 36].

Repair algorithms often use combinations of DCs and cells in intricate ways, and a single DC/cell may interact with different subsets to affect the repair process. To nevertheless obtain an interpretable view of this complex process, we follow an approach originating in cooperative game theory and gaining recent attention in the context of Machine Learning [23, 28, 30, 41]: we quantify the contribution of each individual cell/DC to the repair process using the notion of Shapley values [38], and return their ranked list. As we shall demonstrate, this leads to explanations that are both simple and informative.

C_1. $\forall t_1, t_2. \neg(t_1[Team] = t_2[Team] \wedge t_1[City] \neq t_2[City])$
C_2. $\forall t_1, t_2. \neg(t_1[City] = t_2[City] \wedge t_1[Country] \neq t_2[Country])$
C_3. $\forall t_1, t_2. \neg(t_1[League] = t_2[League] \wedge t_1[Country] \neq t_2[Country])$
C_4. $\forall t_1, t_2. \neg(t_1[Place] = t_2[Place] \wedge t_1[Year] \neq t_2[Year])$

Figure 1: Denial constraints

Dirty table

	Team	City	Country	Year	League	Place
t_1	F.C. Barcelona	Barcelona	Spain	2019	Spanish League	1
t_2	Atletico Madrid	Madrid	Spain	2019	La Liga	2
t_3	Real Madrid	Madrid	Spain	2019	La Liga	3
t_4	F.C. Barcelona	Barcelona	Catalonia	2018	La Liga	1
t_5	Real Madrid	Capital	España	2018	La Liga	2
t_6	Atletico Madrid	Madrid	Spain	2018	Spanish League	3

Repaired table

	Team	City	Country	Year	League	Place
t_1	F.C. Barcelona	Barcelona	Spain	2019	La Liga	1
t_2	Atletico Madrid	Madrid	Spain	2019	La Liga	2
t_3	Real Madrid	Madrid	Spain	2019	La Liga	3
t_4	F.C. Barcelona	Barcelona	Spain	2019	La Liga	1
t_5	Real Madrid	Madrid	Spain	2019	La Liga	2
t_6	Atletico Madrid	Madrid	Spain	2019	La Liga	3

Figure 2: Pre-repair and post-repair database tables for La Liga standings

Example 1. *Consider the DCs depicted in Figure 1 (for brevity, only a subset of the DCs is shown). Also consider the input database table and the table after it was repaired by the algorithm in Figure 3, shown in Figure 2 (dirty cells are colored red and clean cells are colored green). The value in the cell $t_4[Year]$, colored red in the repaired table in Figure 2, was changed from 2018 to 2019. When viewing this result, it is clear that it is an incorrect repair since, the same team may*

Input: Set of constraints $\mathcal{C}$, a dirty database table T^d
Output: A repaired table T^c
for tuple $t_i \in T^d$: **do**
  for constraint $C_j \in \mathcal{C}$ s.t. $C_j := \forall t_1, t_2. \neg(t_1[A] \neq t_2[A] \wedge t_1[B_1] = t_2[B_1] \wedge t_1[B_2] = t_2[B_2] \ldots$ **do**
    if tuple t_i has a contradiction with any other tuple according to C_j **then**
      The A attribute will be modified to the most common one given $t_i[B_1], t_i[B_2]$..., i.e., $\arg\max_c \mathbb{P}[t[A] = c \mid t_i[B_1], t_i[B_2]...]$
    end if
  end for
end for

Figure 3: Majority repair algorithm for the example

finish in a different place in different years. The DC with maximum contribution to this repair is C_4, stating that there cannot be two teams who finish in the same place in different years. Indeed, our system presents this faulty DC as an explanation for the incorrect repair (that also affects $t_5[Year]$ and $t_6[Year]$), and the user may choose to omit it from the set of DCs.

Further consider the cell $t_5[City]$ whose value was changed from "Capital" to "Madrid". The cells with the maximum contribution to this change are $t_3[City], t_3[Team], t_5[Team], t_2[City]$, and $t_6[City]$. The first three cells form a violation of C_1 along with $t_5[City]$, and the two other cells are from the same column and contain the value "Madrid". We can conclude that they are used for choosing value of $t_5[City]$ (since the value "Madrid" is more common than "Capital"). Thus, we can understand that the algorithm in Figure 3 uses a majority vote to repair the value. Such explanations are useful for gaining insight into the repair processes.

In addition to the model, our main contributions are as follows.

Algorithms and Optimizations. Direct computation of Shapley values in our settings is intractable for large-scale data: it requires iterating over all subsets of constraints and data cells, and for each subset executing the black-box repair algorithm (whose execution may be costly). Therefore, we present a two-phase algorithm, tailored to the specific setting of data repair:

(1) **Constraints and Table Pruning:** We develop a pruning algorithm based on an analysis of the connections between the attributes through the DCs, thereby removing irrelevant DCs, attributes and tuples from the computation.
(2) **Optimized Monte Carlo Sampling:** We devise an algorithm that is based on the Monte Carlo sampling approach for approximating Shapley values [40], and combines two optimizations: (1) sample re-use, i.e., the ability to use a single sample for various "players" (cells). This significantly reduces runtime by running the repair algorithm fewer times, and (2) an informed choice of the sample size – we experimentally show that some sample sizes are more informative than others. These sizes are unknown a-priori; we find them using a multi-armed bandit technique [24, 39], showing that our algorithm converges to the true Shapley value.

Experiments We have performed a comprehensive experimental evaluation of our approach. We have compared our approach to the Shapley value formula [38] (used as the ground truth when possible), Monte Carlo sampling [40], and SHAP [30] in terms of accuracy and scalability. We use multiple datasets including Hospital [13], Adult [15], and Microsoft Academic Search Database (MAS) [1]. Our solution converges to the accurate Shapley values faster than the baselines (~ 70 times faster on average), and scales to significantly larger database sizes (millions of tuples as opposed to ~ 10^4 cells for the baselines). We have conducted a user study that further validates the quality and usefulness of our computed explanations.

Related Work. Several works have proposed explanations for *database errors* (instead of repairs) in specific scenarios. Specifically, Wisteria [22] supports the design of development and optimization of data repair workflows by allowing users to interact with the workflow iteratively. Chalamalla et. al. [10] focuses on explaining errors in the initial database. ExplainER [16] provides an approach for explaining ML based entity-resolution algorithms by adapting existing ML explanation approaches such as LIME [37]. Data X-Ray [43] focuses on explaining data errors rather than the repair process using Bayesian analysis. QFix [44] focuses on tracing when errors were inserted to the data by update queries. As for *non black box explanations for repairs*, CLeanEX [6] provides explanations for a pipeline generated by an automated data repair system, assuming this system can provide its pipeline search space. XPlode [35] allows users to first manually clean small subsets of the data and then automatically generates Conditional Functional Dependencies (CFDs) that are consistent with the repairs provided by the user. These CFDs are then considered the explanations for the subsequent overall repair that is performed by an automatic system. In contrast to these works, in our setting, we do not assume any knowledge of the repair algorithm and do not require any user input except the cell whose repair needs to be explained.

2 PRELIMINARIES

We start by reviewing the relevant notions regarding databases, data repair, and Shapley values.

Database Tables. T is a database table with schema $\mathcal{S}(T) = (A_1, \ldots, A_m)$ where A_i is the i-th attribute of T. For a tuple $t \in T$, we denote by $t[A]$ the cell in attribute A of the tuple t. Note that $t[A]$ denotes the cell rather than its current value (values may change due to the repair process). We denote by T^d and T^c the database table before and after the repair, respectively. Similarly, $t^d[A]$ and $t^c[A]$ are used to denote the original and repaired values in the cell $t[A]$, respectively. We assume each tuple and cell in T has a unique identifier. We further abuse the syntax of set operations to describe value changes in T^d to *null*. For instance, if S is a subset of cells in T^d, then the meaning of $T^d \setminus S$ is that $\forall t[A] \in S.\ t[A] = null$.

Example 2. *Consider the dirty and clean tables shown in Figure 2 (T^d and T^c, respectively). Observe that the content of the cell $t_5[Country]$ in T^d has changed in T^c from "España" to "Spain".*

Denial Constraints. DCs [11–13] are a general form of integrity constraint that contain functional dependencies, metric functional dependencies [25] and conditional functional dependencies [9]. They can be expressed as a negation of a conjunctive statement.

Definition 1. *Given a table with schema $T(A_1, \ldots, A_m)$, a Denial Constraint (DC) is logical statement of the form*

$$\forall t_1, \ldots, t_k. \neg(\phi_1 \wedge \ldots \wedge \phi_l)$$

where ϕ_i is either $t_i[A_k] \circ t_j[A_l]$ or $t_i[A_k] \circ c$, c is a constant, $1 \leq k, l \leq m$, and $\circ \in \{<, >, \leq, \geq, =, \neq\}$.

Example 3. *The DC C_1 shown in Figure 1 states that no two tuples can have the same value in the Team attribute value ($t_i[Team] = t_j[Team]$) and a different City attribute value ($t_i[City] \neq t_j[City]$).*

Shapley Values. In Cooperative Game Theory, Shapley value [38] is a way to distribute the contribution of each player, assuming they cooperate. Let N be a finite set of players and $v : 2^N \to \mathbb{R}$, $v(\varnothing) = 0$ be a characteristic function. v maps sets of players to the joint worth they generate according to the game. The contribution of player a to a coalition S (such that $a \notin S$) is defined by the change in v due to a's addition, i.e., $v(S \cup \{a\}) - v(S)$. The Shapley value of player a is the average of this contribution over the possible different permutations in which the coalition can be formed. Intuitively, for any player a, this value is the sum of a's contribution to a change in v over any possible coalition, factored by the size of the coalition such that mid-sized coalitions (of such there exists more of) are factored with a lower contribution.

$$Shapley(N, v, a) = \sum_{S \subseteq N \setminus \{a\}} \frac{|S|!(|N| - |S| - 1)!}{|N|!} \cdot (v(S \cup \{a\}) - v(S))$$

Shapley value is the only distribution function that satisfies several natural properties. Some of the properties are (1) *Efficiency*, i.e., that the sum of Shapley values of all players is equal to the gain of the grand coalition, (2) *Symmetry*, i.e., the worth of two players should be the same if they contribute equally to all possible coalitions, and (3) *Linearity*, i.e., if two separate games are combined together, then the worth distribution of the combined game should be equal to the sum of distributions of both games separately.

3 FRAMEWORK FOR DATA REPAIR EXPLANATIONS

We next define our framework, including a very general notion of data repair, DC and cell contributions through Shapley values.

3.1 Black Box Data Repair

The repair algorithm is considered a black box, and thus we define it simply as a function. Its input is a dirty table and a set of constraints and its output is a clean table.

Definition 2 (Repair Algorithm). *Given a dirty table T^d with schema $\mathcal{S}(T)$ and a set of constraints $\mathcal{C}$, a repair algorithm $Alg_{\mathcal{C},\mathcal{S}(T),T^d}$ is a mapping from a dirty table T^d to a clean table T^c, denoted by $Alg_{\mathcal{C},\mathcal{S}(T),T^d}(t[A]) = t[A]^c$, where $t[A]$ is a cell in T^d and $t[A]^c$ is the value of $t[A]$ in T^c after running Alg with $\mathcal{C}$, $\mathcal{S}(T)$, and T^d.*

In the sequel of the paper, we do not mention the schema in the notation and simply write $Alg_{\mathcal{C},T^d}(t[A]) = t[A]^c$ for simplicity.

Example 4. *Reconsider the algorithm in Figure 3, the denial constraints in Figure 1, and the dirty and clean tables in Figure 2. The cells $t_1[League]$ and $t_5[City]$ that have the values "Spanish League" and "Capital" are mapped by the repair algorithm to $t_5[League]^c$ = "La Liga" and $t_5[City]^c$ = "Madrid", respectively.*

We next define an indicator function based on the repair algorithm. Intuitively, we are interested in examining whether the repair of a cell remains consistent for different subsets of DCs and cells.

Definition 3 (Fixed Function). *Given a dirty table T^d, a set of constraints $\mathcal{C}$, a repair algorithm Alg, and a cell $t[A]$, $Fixed_{Alg} : T^d \times 2^{\mathcal{C}} \times 2^{T^d} \to \{0, 1\}$ is a predicate that returns 1 iff the value of $t[A]$, $t^d[A]$, was mapped to $t[A]^c$ by the repair algorithm Alg when given a subset of DCs and a subset of the cells of T^d. Formally:*

$$Fixed_{Alg}(t[A], S_{\mathcal{C}}, S_{T^d}) = \begin{cases} 1, & Alg_{S_{\mathcal{C}},S_{T^d}}(t[A]) = t[A]^c \\ 0, & otherwise \end{cases}$$

where $S_{\mathcal{C}} \subseteq \mathcal{C}$ and $S_{T^d} \subseteq T^d$ $(\forall t[A'] \in T^d \setminus S_{T^d}.\ t[A'] = null)$.

Observe that $Fixed_{Alg}(t[A], \mathcal{C}, T^d) = 1$ for all $t[A]$.

The *Fixed* function will be used as the value function in the Shapley formula in order to compute the contribution of cells/constraints to the repair of a specific cell. We denote the repaired cell inputted to the *Fixed* function by δ.

Notice that the repair algorithm must support *null* values, as we replace cells in $T^d \setminus S_{T^d}$ with *null*. We find this treatment of missing players as the most natural, but one may consider different strategies (e.g., replacing the value with a random value).

Example 5. *Continuing Example 4, denote $\delta = t_5[City]$, then $Fixed_{Alg}(\delta, \{C_1, C_2, C_3\}, T^d) = 1$ using the algorithm in Figure 3, the dirty table T^d in Figure 2, and the DCs $\{C_1, C_2, C_3\}$ in Figure 1. However, $Fixed_{Alg}(\delta, \{C_3\}, T^d) = 0$ since this cell will not change when the algorithm is only given C_3.*

3.2 Black Box Data Repair Explanations

We now detail the definitions of Shapley values for DCs and data cells as a measure for their contribution to the repair of a cell.

Shapley Values for Denial Constraints. We adapt the concept of Shapley values for explaining data repair algorithms. Intuitively, we think of each DC as a player whose removal changes the repair, i.e., the result of the game.

Definition 4 (DC Contribution). *Let T^d be database table, let $\mathcal{C}$ be a set of DCs, and let Alg be a repair algorithm. Given a DC $C \in \mathcal{C}$ and a repaired cell $\delta \in T^d$, the contribution of C to the repair of δ is the Shapley value of the constraint C:*

$$Shapley(\mathcal{C}, Fixed_{Alg}, \delta, C) = \sum_{S \subseteq \mathcal{C} \setminus \{C\}} \frac{|S|!(|\mathcal{C}| - |S| - 1)!}{|\mathcal{C}|!} \cdot (Fixed_{Alg}(\delta, S \cup \{C\}, T^d) - Fixed_{Alg}(\delta, S, T^d))$$

Recall that Shapley values measure the contribution of each player on the outcome. Here, $\mathcal{C}$ is the set of players, the function $Fixed_{Alg}$ is the value function and T^d remains constant. This paper focuses on DCs. However, Definition 4 is not specific to DCs and can be easily extended to other forms of constraints such as external mappings [19], equality generating dependencies, and tuple generating dependencies [5, 18].

Example 6. *Reconsider the tables in Figure 2. Let us compute the contribution of each DC C_1, C_2, C_3 (shown in Figure 1) to the change of $\delta = t_5[Country]$ when using the repair algorithm in Figure 3. Note that the algorithm will change "España" to "Spain" if we have the DCs $\{C_1, C_2\}$, or $\{C_3\}$. According to the definition, we can compute the contribution of C_1 as follows: there are 4 subsets of $\{C_2, C_3\}$, and only for $S = \{C_2\}$ we have $Fixed_{Alg}(\delta, S \cup \{C_1\}, T^d) = 1$ and $Fixed_{Alg}(\delta, S, T^d) = 0$, so $Shapley(\mathcal{C}, T^d, C_1) = \frac{1}{6}$, that is due to the coefficient in Definition 4. A similar computation applies to C_2. For C_3 we have 3 out of 4 subsets S of $\{C_1, C_2\}$ that result in $Fixed_{Alg}(\delta, S \cup \{C_3\}, T^d) = 1$ and $Fixed_{Alg}(\delta, S, T^d) = 0$, including $S = \varnothing$. Thus,*

$Shapley(\mathcal{C}, T^d, C_3) = \frac{1}{6} + \frac{1}{6} + \frac{2}{6} = \frac{2}{3}$. The intuition for the value of C3 being double that of the sum of the pair $\{C_1, C_2\}$ is that there are 5 subsets of the DCs $\{C_1, C_2, C_3\}$ for which we repair δ. These are $\{C_3\}$, $\{C_1, C_2\}$, $\{C_1, C_3\}$, $\{C_2, C_3\}$, and $\{C_1, C_2, C_3\}$. Four of these contain C3 while only two contain the pair $\{C_1, C_2\}$ (for subsets where one of these is present without its partner, the repair is due to C_3), thus, the contribution of C_1, C_2, as a pair, is half that of C_3.

Shapley Values for Data Cells. We define the contribution of each data cell to the repair process through Shapley values. Given a repaired cell $\delta \in T^d$, we define the Shapley value for a cell $t[A]$.

Definition 5 (Cell Contribution). *Let T^d be a table, let $\mathcal{C}$ be a set of DCs, and let Alg be a repair algorithm. Given a cell $t[A] \in T^d$ and a repaired cell $\delta \in T^d$, the contribution of $t[A]$ to the repair of δ is the Shapley value of the cell $t[A]$:*

$$Shapley(T^d, Fixed_{Alg}, t[A]) = \sum_{S \subseteq T^d \setminus \{t[A]\}} \frac{|S|!(|T^d| - |S| - 1)!}{|T^d|!} \cdot (Fixed_{Alg}(\delta, \mathcal{C}, S \cup \{t[A]\}) - Fixed_{Alg}(\delta, \mathcal{C}, S))$$

where $S \subseteq T^d$ is the table T^d and $\forall t[A'] \in T^d \setminus S.\ t[A'] = null$.

In this definition, the set of cells in T^d are the players, while $\mathcal{C}$ remains constant (as opposed to Definition 4).

Example 7. *Reconsider the DCs in Figure 1, the algorithm in Figure 3, and the tables in Figure 2. Consider the cell $\delta = t_5[Country]$. The value of this cell is changed from "España" to "Spain". The Shapley value of the cell $t_5[League]$ will be significant as it is required for the repair through C_3 (established earlier as the most influential constraint). If $t_5[League]$ is not present in S, $t_5[Team]$ and $t_5[City]$ are both needed for the repair of $t_5[Country]$ through C_1 and C_2, respectively, so their Shapley value will also be high. These cells are necessary, but they are not sufficient, according to the algorithm in Figure 3. To analyze when the repair happens, observe that any set of cells $S \subseteq T^d$ containing $t_5[League]$ and a majority of cells of the form $t_i[Country], t_i[League]$ with the values "Spain", "La Liga" will be enough to repair the cell $t_5[Country]$. Conversely, without $t_5[League]$, we will need a set $S \subseteq T^d$ containing both $t_5[Team]$ and $t_5[City]$ and enough cells of the form $t_i[Team]$ with the value "Real Madrid" and enough cells of the form $t_i[City]$ with the value "Madrid", and enough cells of the form $t_i[Country]$ with the value "Spain" to repair $t_5[Country]$. This intuitively shows why the Shapley value of $t_5[League]$ is higher than of that of $t_5[Team]$ and $t_5[City]$ as it affects the repair of more coalitions. For simplicity we overlooked the coalitions sizes, though they play an important role.*

Problem Definition. We now define the *data repair explanation problem*. Intuitively, given a cell δ whose repair we want to explain, we are interested in returning a ranked list of all cells that support this repair, as these are the cells that can explain the repair of δ in the best manner. Cells that contributed negatively to the repair will be ignored.

Definition 6 (Data Repair Explanation Problem). *Given a dirty table T^d, a set of DCs $\mathcal{C}$, a data repair algorithm Alg, and a cell δ that has been repaired by Alg, the solution to the data repair explanation problem is a ranked list of the DCs that positively contributed to the repair of δ and a ranked list of the cells that positively contributed to the repair of δ.*

Example 8. *Given the dirty table in Figure 2, the DCs in Figure 1, the algorithm in Figure 3, the cell $\delta = t_5[Country]$, the solution to the data repair explanation problem is a ranked list of DCs contributing to the repair of $t_5[Country]$ from "España" to "Spain": (1) C_3, (2) C_1, (3) C_2, and a list of cells contributing to the same repair. The first three cells in this list are (1) $t_5[League]$, (2) $t_3[Country]$, (3) $t_3[League]$ (equivalently, $t_2[League]$).*

Although the model and the problem are defined for a specific repaired cell, δ, we wish to explain, it could be easily extended to a collection of cells (or even all the cells that were modified by the repair algorithm). Thanks to Shapley linearity property, the influence of a cell (or a constraint) on the repair of $\delta_1, \ldots, \delta_m$ is the sum of its influences on each of the δ_i separately.

4 OPTIMIZED COMPUTATION OF CONTRIBUTIONS

Direct computation of Shapley values would require exponentially many invocations of the cleaning algorithm, one for each subset of cells and constraints. While the number of constraints is typically small (See Section 5.2, paragraph "Number of relevant DCs"), exponential data complexity is prohibitively costly [7, 34]. A commonly used approximation is via a standard Monte Carlo sampling algorithm [31] that repeatedly samples subsets of players (table cells and constraints in our case) and computes the difference in the game outcome ($Fixed_{Alg}(\delta, S_{T^d})$ in our case) with and without the player in question. In our settings, m samples would entail $2m \cdot (|T^d| - 1)$ invocations of the repair algorithm *Alg*; each such invocation is typically costly. Therefore, we present an optimized sampling algorithm based on three optimizations that we describe later in this section: pruning, sample reuse and an intelligent sampling strategy. We first describe our optimized sampling algorithm and then detail the optimizations embedded in it.

4.1 Optimized Sampling Algorithm

The input to Algorithm 1 is a black-box repair algorithm *Alg*, a dirty table T^d, a repaired cell to explain δ, and a bound m over the number of iterations. We start by initializing the returned value φ which is a vector that holds the contribution of each cell to the repair of δ by *Alg* (Line 1). Next, in Line 3 we nullify all table cells and DCs that do not contribute to the repair; this is done through an analysis of the constraints and database schema (see details in Section 4.2, paragraph "Constraints and table pruning"). Then, in each iteration of the algorithm, we first choose a value for p according to a pre-decided parameter search strategy (Line 7, see Section 4.2 paragraph "Implementing the Search Strategy" for details). We then sample cells from the table (importantly, during the execution of Algorithm 1 the m samples are drawn i.i.d.). Based on the sample we generate a table, S, such that every cell $t_i[B] = t_i^d[B]$ with probability p and $t_i[B] = null$ with probability $1 - p$ (Line 8).

Next, we utilize the sampled table S to assess the contribution of table cells (see paragraph "Sample Reuse Optimization" for more details). We check, using $Fixed_{Alg}(\delta, S)$, whether the sampled subtable S is sufficient for repairing δ (Line 10). If so, we check whether removing any of the cells (i.e. setting them to null) would prevent the repair (Lines 12–13). After completing the test, we restore the

Algorithm 1: Approximate the Contribution of All Cells

input : Repair algorithm Alg, dirty table T^d, repaired cell to explain δ, number of repeats m
output: Approximation of the contribution for every cell φ

1 $\varphi \leftarrow \{0\}^{|T^d|}$;
2 /* pruning optimization (Algorithm 2) */
3 $T^d, \mathcal{C} \leftarrow$ `table_pruning`$(Alg, T^d, \delta, \mathcal{C})$;
4 $reward \leftarrow$ `dict()` initialized with zeros;
5 **for** 1 *to* m **do**
6 /* search strategy optimization */
7 $p \leftarrow$ `search_prob_strategy`$(reward,$ hyper-parameters$)$;
8 $S \leftarrow$ `sample_subset`$(T^d \setminus \{\delta\}, p)$;
9 /* sample reuse optimization */
10 **if** $Fixed_{Alg}(\delta, S)$ **then**
11 **foreach** $c \in S$ **do**
12 $S[c] \leftarrow null$;
13 **if** *not* $Fixed_{Alg}(\delta, S)$ **then**
14 $\varphi_c = \varphi_c + \frac{1}{p^{|S|-1}(1-p)^{|T^d|-|S|+1}\binom{|T^d|-1}{|S|-1}\cdot|T^d|}$;
15 $reward[p] \leftarrow reward[p] + 1$;
16 $S[c] \leftarrow T^d[c]$;
17 **else**
18 **foreach** $c \in T^d$ *s.t.* $S[c] = null$ **do**
19 $S[c] \leftarrow T^d[c]$;
20 **if** $Fixed_{Alg}(\delta, S)$ **then**
21 $\varphi_c = \varphi_c + \frac{1}{p^{|S|+1}(1-p)^{|T^d|-|S|-1}\binom{|T^d|-1}{|S|+1}\cdot|T^d|}$;
22 $reward[p] \leftarrow reward[p] + 1$;
23 $S[c] \leftarrow null$;
24 **return** $\frac{\varphi}{m}$;
25 `sample_subset`(T, p):
26 S is a copy of T where each cell in S is null with prob $1-p$;
27 **return** S;

cell to its original value (Lines 16). If the cell positively affects the repair, we add the weighted coefficient to our Shapley approximation (Line 14). For intuition on the used coefficient, let S be a random subset, and recall that according to Shapley value definition, the weight of S is equal to $\left(|S|!(|T^d| - |S| - 1)!\right)/|T^d|! = 1/\left(\binom{|T^d|-1}{|S|}|T^d|\right)$, whereas in every iteration of Algorithm 1 the probability of sampling S is equal to $p^{|S|}(1-p)^{|T^d|-|S|-1}$ (selecting S items with probability p, and not selecting any other item).

When the sub-table S is insufficient for fixing δ, we check whether adding a single cell (i.e. restoring its original value) to S would cause Alg to repair δ (Lines 18-20). After completing the test, we restore the cell value to *null* (Line 23). Finally, we return the average contribution of each cell across iterations (Line 24).

4.2 Optimizations based on repair properties

We next provide details on the optimizations embedded in Algorithm 1. In particular, we draw connections to certain properties that the black-box cleaning algorithm is expected to satisfy (indeed, our experiments show that popular algorithms satisfy them).

Constraints and Table Pruning (Line 3). The first optimization discards DCs, attributes, and tuples irrelevant to the repair of δ. For this optimization, we assume that there is a way to detect the subset of cells and the subset of constraints that are relevant to the repair of a cell. Formally, we look at the *attribute hypergraph of the constraints*. This graph contains the set of attributes in the schema $\mathcal{S}(T)$, where there is an undirected edge that includes a set of attributes $A_1, \ldots, A_k$ if $A_1, \ldots, A_k$ are all involved in the same DC. We then assume:

(1) The only DCs that can influence the repair of the cell δ are the DCs that have a path from the edge they define to an attribute of δ
(2) The only attributes that can influence the repair of δ are the ones involved in the constraints mentioned in the first bullet
(3) The only tuples that can influence the repair of δ are the ones violating the constraints mentioned in the first bullet

Intuitively, all DCs that can influence the repair should be connected in some manner to the attribute of δ. Similarly, the attributes (tuples) that can influence the repair are likely to be connected to the attribute (tuple) of δ through some path of DCs.

The pruning is then detailed in Algorithm 2. The algorithm starts by generating the attribute (hyper)graph. Next, it performs a BFS on the graph from the attribute A of the cell δ, and gets a list of edges, where each edge in the list is a part of a path to A (Line 2). The set $\mathcal{C}_{pruned}$ is then defined to be the labels of the edges in the list (Line 3). In Lines 4–5 the algorithm initializes an empty set for the collection of relevant attributes and an empty list for the relevant tuples. The algorithm iterates over all relevant DCs in $\mathcal{C}_{pruned}$, updates the attributes that they include (Line 7), and collects the tuples that violate them (Lines 8–9). Finally, the pruned table T^d_{pruned} is defined as the list of tuples that violate some DC in $\mathcal{C}_{pruned}$ with only the attributes that participate in a relevant DC violation included in the table (Line 10). Both $\mathcal{C}_{pruned}$ and T^d_{pruned} are returned in Line 11.

Algorithm 2: DCs and Table Pruning

input : Dirty table T^d, a set of constraints $\mathcal{C}$, and a repaired cell $\delta = t[A]$
output: pruned DC set $\mathcal{C}_{pruned}$ and pruned table T^d_{pruned}

1 $G \leftarrow constructAttrGraph(\mathcal{C})$;
2 $L_{edges} \leftarrow BFS(G, source = A)$;
3 $\mathcal{C}_{pruned} \leftarrow \{L(e) \mid e \in L_{edges}\}$;
4 $attrs \leftarrow \varnothing$;
5 $tuples \leftarrow []$;
6 **foreach** $C \in \mathcal{C}_{pruned}$ **do**
7 $attrs \leftarrow attrs \cup Attr(C)$;
8 $L_t \leftarrow Violations(C, T^d)$;
9 $tuples.add(L_t)$;
10 $T^d_{pruned} \leftarrow tuples[attrs]$;
11 **return** $\mathcal{C}_{pruned}, T^d_{pruned}$

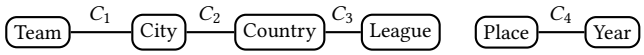


Figure 4: Attribute graph for the tables in Figure 2 with the DCs in Figure 1. Two attributes are connected if there is a constraint that includes both (denoted by the edge label)

	Team	City	Country	League
t_1	F.C. Barcelona	Barcelona	Spain	Spanish League
t_2	Atletico Madrid	Madrid	Spain	La Liga
t_3	Real Madrid	Madrid	Spain	La Liga
t_4	F.C. Barcelona	Barcelona	Catalonia	La Liga
t_5	Real Madrid	Capital	España	La Liga

Figure 5: Pruned table for explaining $t_5[Country]$. t_6 and the attributes Year and Place were pruned

Example 9. *Consider the (dirty) table in Figure 2, DCs $\{C_1, C_2, C_3, C_4\}$ in Figure 1, and the repaired cell $\delta = t_5[Country]$. Algorithm 2 first generates the attribute graph in Figure 4, and performs BFS from the node Country; the set of connected constraints are $\mathcal{C}_{pruned} = \{C_1, C_2, C_3\}$. Then, the algorithm continues to collect the attributes included in each of the chosen DCs, i.e., Team, City, Country, and League. It also collects the tuples that participate in a violation of the three DCs. These are t_1, t_2, t_3, t_4, t_5. Thus, the pruned table returned in Line 11 is depicted in Figure 5.*

Under the context assumption, all cells in $T \setminus T_{pruned}$ and DCs in $\mathcal{C} \setminus \mathcal{C}_{pruned}$ do not influence the outcome of *Alg*. As a result, for every cell $c \in T \setminus T_{pruned}$ it holds that $Shapley(T^d, Fixed_{Alg}, c) = 0$. Since deleting a cell with Shapley value 0 does not impact the value of the other cells, cells in $T \setminus T_{pruned}$ can be removed without modifying the Shapley values of the remaining cells. Thus, under the context assumption, the Shapley values of cells remain intact after the pruning process. The same argument applies to DCs.

Sample Reuse Optimization (Lines 10–16 and 17–23). As opposed to standard the Monte Carlo sampling algorithm, our solution computes the Shapley values of all remaining table cells simultaneously. The theoretical justification is based on a *monotonicity* assumption over the data repair algorithm, that we next formalize. The assumption is that for any cell $t[A]$, if there exists some $S \subseteq T^d$, such that $Fixed_{Alg}(\delta, \mathcal{C}, S \cup \{t[A]\}) - Fixed_{Alg}(\delta, \mathcal{C}, S) = 1$, then for every $S' \subseteq T^d$, it holds that $Fixed_{Alg}(\delta, \mathcal{C}, S' \cup \{t[A]\}) - Fixed_{Alg}(\delta, \mathcal{C}, S') \geq 0$. Conversely, if there exists some $S \subseteq T^d$, such that $Fixed_{Alg}(\delta, \mathcal{C}, S \cup \{t[A]\}) - Fixed_{Alg}(\delta, \mathcal{C}, S) = -1$, then for every $S' \subseteq T^d$, the following holds $Fixed_{Alg}(\delta, \mathcal{C}, S' \cup \{t[A]\}) - Fixed_{Alg}(\delta, \mathcal{C}, S') \leq 0$. Intuitively, we expect that adding a cell that is consistent with the change of δ would not hinder the repair; conversely, adding a cell that "contradicts" the repair should not cause the algorithm to perform the repair.

Given a sample S, Algorithm 1 then tests if $Fixed_{Alg}(\delta, S)$, i.e., whether *Alg* fixed the value of δ given the set S, if the condition is true (Lines 10–16) then due to the monotonicity requirement, the only 'single cell' change to S that might affect the outcome of $Fixed$ is a removal of cells from S. If $Fixed_{Alg}(\delta, S)$ is false (Lines 17–23), then adding a cell from $T^d \setminus S$ to S might change the outcome of the repair process. Thus, given the outcome of the repair process over S we may test multiple cells, and limit the reuse only to potentially relevant cells.

Recall that the Monte Carlo sampling algorithm calls the repair algorithm, *Alg*, $2m \cdot |T^d|$ times. The sample reuse optimization reduces the number of calls to *Alg*. Algorithm 1 calls *Alg* at most $m + m \cdot |T^d_{pruned}|$ times, which implies ~ 2 factor reduction, though usually the number of calls will be even smaller (see Section 5), since each sample will be tested only for a subset of the table cells; based on the condition in Line 10.

Implementing the Search Strategy (Line 7). Since samples are reused to evaluate the Shapley value of multiple cells simultaneously, we aim for samples that will be informative for many cells, i.e., sample a set S such that $\sum_{c \in T^d \setminus \{\delta\}} |Fixed_{Alg}(\delta, S \cup \{c\}) - Fixed_{Alg}(\delta, S)|$ is large. Thus, we employ a new sampling strategy.

Recall that in the Monte Carlo sampling algorithm, for every size $k \in \{0, \ldots, |T^d| - 2\}$, it holds that $\mathbb{P}[|S| = k] = \frac{1}{|T^d|-1}$, i.e., all set sizes are equally likely. We identified that some sample sizes are more likely to yield information for multiple cells. Consequently, sampling all subset sizes with the same probability would not be the ideal strategy. Intuitively, every repair algorithm has a different level of confidence required for the repair, and every dataset has a ratio of violating/non-violating tuples (in the context of a specific examined repaired cell δ). These parameters will affect the sample sizes that are more likely to result in a repair.

The benefit of the sampling strategy optimization is a better utilization of the samples, which enables convergence with smaller number of iterations. This is shown empirically in Section 5.2 (see paragraph "Effect of optimizations on the Runtime").

Since the impact of the sample size is unknown a priori, we propose a Multi-Armed Bandits (MAB) ε-decreasing strategy to best utilize the samples. The MAB setting is a commonly used online learning method with multiple variants [26]. In a nutshell, given a set of discrete points and a reward function, MAB finds which points yield the most reward. To sample sets that are likely to provide information about many cells in $T^d \setminus \{\delta\}$, we propose a variant of the ε-Greedy-Decreasing strategy; we transform $[0, 1]$ to a discrete space with d and refer to each point as 'arm'. In each iteration, we either sample a random arm with probability ε or (with probability $1 - \varepsilon$) pick a random arm out of the $k < d$ top performing arms so far, based on the reward function. We use a decreasing ε value, i.e., as the number of iterations grows, we increase the exploitation and decrease the exploration by a factor of η (the step size). We also define a minimum value of ε. The reward of each arm is the sum of cells that were identified as impactful in previous samples; it is updated by Algorithm 1 in Lines 15 and 22.

5 EXPERIMENTS

We next present our experimental study. Our main findings are:

(1) Our approach converges at a significantly higher rate than the other baselines and existing methods examined (see Section 5.1), even for small database sizes (up to $\sim$ 200 times faster, and $\sim$ 70 times faster on average).
(2) Our approach is the only scalable one compared to the examined baselines. In particular, for 100K–500K cells, our approach is the only feasible one among those tested.
(3) The MAB strategy outperforms the commonly used uniform random strategy.

(4) Algorithm 2 (pruning) plays the most significant role in reducing the runtime of our solution.
(5) Our solution assists users in understanding the workings of a repair algorithm and in detecting errors in the DCs.
(6) Our solution proved to be helpful for explaining errors in the repair of a commonly used dataset using the state-of-the-art Holoclean algorithm. It was found useful in identifying errors in the DC set as well as relevant errors in the dataset and properties of the repair algorithm.

Our system, named T-Rex [14], is implemented in Python 3.6 with an underlying database engine in PostgreSQL 10.6. The experiments were performed on Windows 10 Pro, 64-bit, with 16GB of RAM and Intel Core Duo i7-8665U @ 2.11 GHz processor. The code was parallelized and run on 10 workers.

5.1 Experimental Setup

We next detail the datasets and baselines used in our experimental evaluation. All experiments were performed using the algorithm in Figure 3 as the repair algorithm and focused on explaining repairs through table cells, as computing the Shapley values of constraints using Definition 4 is usually negligible in terms of runtime, as their number is typically small. We have used the algorithm depicted in Figure 3 and the Holoclean system [36] to examine our approach.

Datasets. We next describe the datasets used in our evaluation.

(1) **Spanish Soccer** This is the dataset in the running example shown in Figure 2 with the DCs described in Figure 1. It was created by scraped data off several sources and includes details about Spanish soccer teams.
(2) **Hospital** This dataset has been used in the data repair literature [13, 36]. It contains information about different hospitals and is characterized by a large number of repetitions.
(3) **Adult** This dataset [15, 36] was extracted from the Census Bureau database. It contains information about individuals, e.g., income and education.
(4) **MAS** The MAS dataset [1, 27] is an academic database with data about academic institutions, conferences, papers, etc.

Baselines for Shapley Values Approximation. Our approach is the first, to our knowledge, that uses Shapley values in the context of repair algorithms. Therefore, the baselines we have used are based on prominent approaches devised in previous work for computing Shapley values in other domains.

(1) **Shapley Formula [38]:** As it is only feasible for very small datasets, we compare results to the direct Shapley formula calculation results for small tables (up to 30 cells).
(2) **Monte-Carlo Sampling [31]:** Outlined at the start of Section 4. It is usually used for explaining ML models [33, 42].
(3) **SHAP [30]:** This Python library is widely used as an ML model feature importance approximation system, based on Shapley values. To adapt SHAP to our setting, we have replaced the ML model with the repair algorithm and the data features are replaced by a vector of binary features that describe whether a table cell exists in a sample. Essentially, a permutation of the table cells is simulated by 'model samples'. We input this vector into the SHAP's KernelExplainer and extract the given explanations.

Table 1: Summary of experimental settings

Table/Figure	Algorithm	Repair Algo	Dataset
Search Strategy Experiments			
Fig. 6a	MAB vs random	Algorithm in Fig. 3	Hospital
Fig. 6b	MAB vs random	Algorithm in Fig. 3	Soccer
Accuracy Experiments			
Table 2	Baselines vs ours	Algorithm in Fig. 3	Soccer
Table 2	Baselines vs ours	Algorithm in Fig. 3	Hospital
Scalability Experiments			
Fig. 7a	Baselines vs ours	Algorithm in Fig. 3	Adult
Fig. 7b	Baselines vs ours	Algorithm in Fig. 3	Hospital
Fig. 7c	Ours	Holoclean	MAS
Fig. 8 (**Optimizations**)	Baselines vs ours	Algorithm in Fig. 3	Hospital
Fig. 9 (**User Study**)	Ours	Holoclean	MAS

Accuracy Metrics. We use two metrics for measuring accuracy: $L1$ to measure convergence to the ground truth, and Top-k accuracy to measure the stability of the top-k result.

(1) $L1$ **Distance:** Given a vector of the true Shapley scores calculated for every cell in the table by Shapley formula in Definition 5 and a vector of approximated values, we compute the $L1$ distance between them.
(2) **Top-k Accuracy at Convergence:** We say that a top-k ranked list of cell contributions is converged at the point in which the top-k set of cells has not changed for the last ≥ 5 samples of each cell in that set. Although this metric is not referring to the correct order within the top-k, our method did accurately estimate the rankings of the contributing cells within the examined top-k in the experiments we conducted. This metric hails from the precision at k metric which is widely used in information retrieval [4, 32] and is sensible for our case, as a user or analyst would most probably be interested in the most influential cells.

Recall that we defined two desired properties of black box repair algorithms in Section 4.2. In all the examined settings (Table 1), we have manually checked the validity of both the monotonicity and the context assumptions and found that they hold.

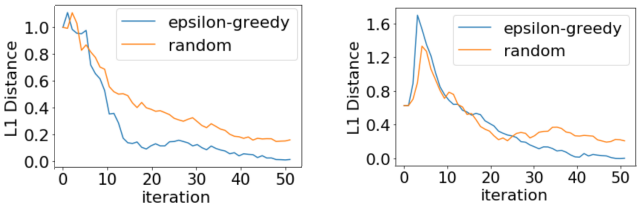
5.2 Accuracy and Scalability Results

In this set of experiments, we examine the performance of T-Rex for computing the contributions of cells to the repair, since we assume that the number of relevant DCs is small, and the cells outnumber the DCs many times over (this was also the case in practice. See paragraph "Number of Relevant DCs"). The settings and associated tables and figures are summarized in Table 1.

Sampling Parameter Search Strategy Comparison. In Line 7 of Algorithm 1 we have employed a search strategy to find the optimal set sizes to be used in the sampling. We compare two strategies:

(1) Baseline: a random strategy where the value of p is chosen randomly and uniformly out of $[0, 1]$
(2) The MAB strategy, described in the paragraph titled 'Implementing the Search Strategy' in Section 4.2.

In Figure 6 we compare the performance of the two and measure the $L1$ distance from the true Shapley values using the Soccer and Hospital datasets. We run MAB for multiple values of the input parameters d, k and η. We found that the parameters $d = 11$, $k = 4$, and $\eta = 0.01$ yielded the best performance in terms of accuracy-runtime balance. The random policy starts close to ours during the exploration phase which is essentially random. In subsequent



(a) Hospital dataset (b) Soccer dataset

Figure 6: $L1$ convergence rates of the epsilon-greedy (MAB) and random search strategies to the ground truth, as a function of the number of iterations

Table 2: Convergence to ground truth on small datasets

Dataset	Algorithm	Top3 (%)	Top10 (%)	L1
Hospital (100x21)	By formula	100	100	0
	Monte Carlo	100	100	0.344
	SHAP	100	100	0.337
	T-Rex	100	100	**0.298**
Soccer (6x6)	By formula	100	100	0
	Monte Carlo	100	100	0.88
	SHAP	100	100	**0.051**
	T-Rex	100	100	0.398

iterations, the MAB policy converges faster than the random policy. In Figure 6b there is a peak in the first few iterations due to the Shapley values non-uniform distribution in the soccer database. In the initial iterations, the algorithm finds a large number of cells that contribute together and spreads the Shapley values uniformly between them. This creates a large L1 gap between the computed Shapley values and the true ones, unlike the Hospital dataset where the true values are more uniformly distributed, and no peak appears. Over time this is corrected and converges to the true values.

Convergence to Ground Truth. Table 2 show a comparison of the accuracy of Algorithm 1 with the baselines w.r.t. our soccer dataset from Figure 2 and the Hospital dataset, respectively. The benchmarks (SHAP, Monte Carlo) and our algorithm all converge to the true top-k Shapley valued cells, both for $k = 3$ and $k = 10$. All the positive valued Shapley cells were ranked correctly. These results imply that our algorithm indeed converges to the true Shapley values. Although SHAP was able to outperform our algorithm in the $L1$ accuracy metric on the soccer dataset, as opposed to T-Rex, SHAP suffers from false positives, i.e., cells with non-positive Shapley value were wrongly approximated as positively affecting the repair. This does not dramatically affect the L1 or top-k metrics, it mostly causes cells that have true Shapley value of 0, get an ϵ value from SHAP. Furthermore, in the next paragraph, we show that SHAP is much slower than T-Rex and does not scale.

Runtime as a Function of Database Size. Figures 7a and 7b depict the runtime on the Adult and Hospital datasets, respectively. Both show the runtimes of T-Rex, the Monte-Carlo algorithm and our adaption of SHAP, for various data sizes. This experiment was run until convergence (as defined in Section 5.1) with the algorithm in Figure 3 and the set of constraints that was given with the datasets (2 and 15 DCs respectively). The cell δ was chosen at random from the repaired cells in the full table. In Figure 7a, for Monte-Carlo and SHAP, sizes larger than 15K cells were already infeasible, while our solution was able to scale to 537K cells in reasonable time. For the Hospital dataset (Figure 7b), for the Monte-Carlo and SHAP algorithms, sizes larger than 11K cells were infeasible, while T-Rex scaled up to 150K in reasonable runtime. In Figure 7c we examined the runtime of T-Rex with the MAS dataset, using Holoclean repair algorithm [36]. T-Rex is the only feasible approach for Shapley value based explanations of data repair that can run with Holoclean and such data scales. We have increased the size of the database from ~ $2M$ up to ~14.5M cells and the runtime was 45 and 130 minutes, respectively.

Effect of Optimizations on the Runtime. Figure 8 shows the contribution the three optimizations embedded in Algorithm 1, presented Section 4: (1) constraints and table pruning (2) sample re-use (3) sampling strategy (MAB). These are presented w.r.t. the Hospital dataset with three different sizes. It ran with 15 DCs with the algorithm in Figure 3. We run five different combinations of (1), (2), and (3). The pruning pre-processing offered the most dramatic contribution to scalability. Optimizations (2) and (3) both contribute approximately equally when used individually, but the combination of all three (as used in Algorithms 2 and 1) is the fastest to converge in a rate of up to ×3 faster than using the pre-processing method alone, and up to ×200 faster than the Monte Carlo baseline. The factors written on the bars represent the multiplicative improvement in runtime in comparison to the Monte Carlo sampling with our pruning method (orange bar).

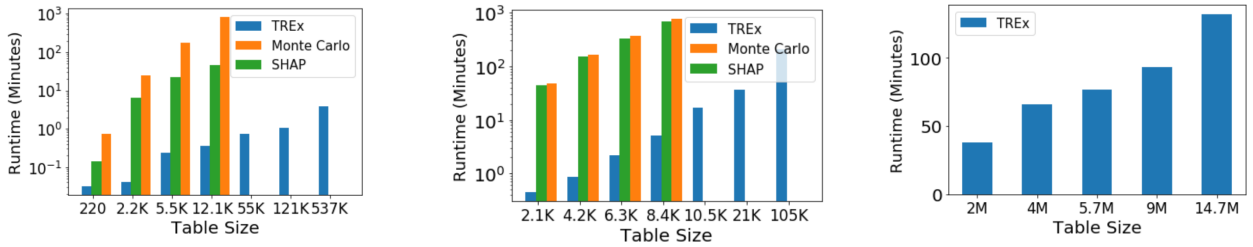
Number of Relevant DCs. In Section 4 we assumed that the number of constraints is typically small and thus we can run the original Shapley formula when computing their contribution. For all the datasets in our experiments, after pruning the DCs according to Algorithm 2, the number of constraints was always ≤ 7, This supports our assumption that computing the contributions of the constraints using the Shapley formula is feasible.

5.3 T-Rex Usability

We next describe our user study and examine a specific use case to showcase the utility of the T-Rex system.

User Study. In order to show that T-Rex explanations are useful for understanding of the repair process, we devised a user study composed of five questions. Each question presents a problem; some synthetic and some from a realistic scenario (see the use case paragraph in this section). For the study, we used two different datasets: Hospital of size 21K cells and 19 DCs, and the MAS dataset of size 2M cells and 11 DCs (the databases and DCs can be found in [2]). The study included twenty users. All users were given files with the databases and the constraints used in the repair process for every question. They were also briefed about the study, relevant notations for it, and the datasets used in it. The users were divided into 2 groups: one group had access to T-Rex output and the second did not. All the users that have participated in our study are from a background of data analysis and have either official education or practical experience in the field of data analysis or data science.

Table 3 shows the questions and Figure 9 depicts a summary of the results. The results depict the advantage of having the T-Rex explanations. For example, in question Q3, all 10 users with access to T-Rex have answered correctly while only 2 users without access



(a) Adult dataset (b) Hospital dataset (c) MAS dataset for T-Rex using Holoclean

Figure 7: Runtime comparison as a function of database size. The MAS dataset is organically much larger and thus diverse. Hence, Algorithm 2 (pruning) has a much higher impact for this dataset. Therefore, for this dataset (Figure 7c), our solution scales much better for larger table sizes. Running the baselines for this dataset with Holoclean is infeasible

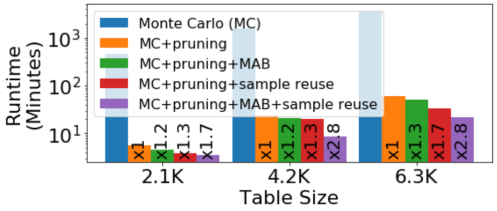


Figure 8: Comparison of different settings (subsets) of the optimizations described in Section 4.2

Table 3: User study questions as shown to the users. Databases and accompanying DCs can be found in [2]

Num.	Dataset	Question
1	Hospital	After running a commercial black-box repair algorithm, an analyst noticed that the cell in row 1, in the attribute "EmergencyService" was changed from 'Yes' to 'No'. She knows that the hospital in this line, "callahan eye foundation hospital" does support Emergency Services, yet the cleaning algorithm has decided to repair this cell. She noticed one constraint was logically flawed and caused this wrong 'fix'. Which constraint was that?
2	Hospital	Review the cell in row 42 in the attribute CountyName, after we run the repair algorithm on the hospital table it is changed from"jxffxrson" to "jefferson". The system architect would like to minimize the number of constraints used as they are computationally expensive. In your opinion, and assuming there are no other relevant data, what minimal set of constraints (plural) would you keep in order to ensure the repair still happens?
3	Hospital	Repeat question 2 with the cell in row 20 in the CountyName attribute, after we run the repair algorithm on the hospital table it is changed from "xe kalb" to "de kalb".
4	MAS	After adding a 2021 VLDB publication and running a state-of-the-art cleaning algorithm, Holoclean, with the constraints attached to the data, the new 2021 VLDB record is 'repaired' - the conference homepage submitted was "https://vldb.org/2021" yet all the values were changed to "http://www.vldb.org/2011". Can you find the problem?
5	MAS	Give a suggestion as to how to fix the problem (remove a constraint? edit? suggest a new setting/constraint?)

were correct. Furthermore, no user without access to T-Rex was able to answer Q1, Q2, Q4, Q5 while at least 7 users with access to the T-Rex results were able to answer these questions. Furthermore, all of the participating users with access to the T-Rex system were able to correctly answer the Q4 while all the users without access to T-Rex were unsuccessful. Furthermore, 6 out of 10 T-Rex users were able to suggest the correct fix, and 2 had offered a partial fix (Q5). This study shows how users can harness T-Rex to easily identify errors in the repair process and achieving better repairs.

Use Case. We focus on questions 4,5 in the user study to showcase a use case for T-Rex in which it helps to quickly pinpoint issues in the MAS dataset. We have considered an instance of the MAS database with ~2M cells, and a set of 11 functional dependencies. We have

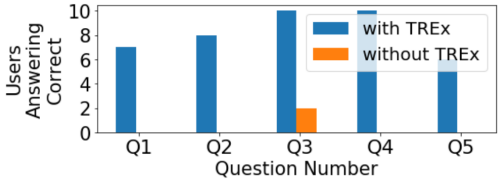


Figure 9: T-Rex user study results

artificially augmented the database with a tuple of a 2021 VLDB publication and run the Holoclean system with the constraints on a large subset, sampled of the DB. The added tuple of 2021 VLDB paper was 'repaired' so that its "conference homepage" attribute changed from "vldb.org/2021" to "vldb.org/2011". We then examined the repair of this cell by running T-Rex. The system identifies that the top influencing constraint "conference name → conference homepage" and top influencing cells, all from tuples with a VLDB publication, but from different years. This reveals 2 issues:

(1) The above constraint should be changed to, e.g., "conference name and publication year → conference homepage".
(2) The database instance contains an inherent mistake in the "conference homepage" attribute, where all VLDB publications are linked to a single wrong url. This, in turn, throws off the repair algorithm and causes it to change a correct value into a wrong one.

6 CONCLUSIONS AND FUTURE WORK

We have defined and studied, for the first time to our knowledge, the problem of explaining data repair using Shapley values for black box repair algorithms and DCs. We have devised an optimized algorithm for computing Shapley values of constraints and cells to explain the repair of a specific cell. We showed that our approach converges to the Shapley values faster and scales better than previous work, and that our solution is effective in allowing users to detect errors and get explanations for the repair. Future work includes the development of further optimizations for both black-box and white-box repair algorithms, as well as interactive User Interfaces allowing further exploration of the repair process.

Acknowledgements. This research was funded by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (Grant agreement No. 804302), the Israeli Science Foundation (ISF) Grant No. 978/17, and the NSF awards IIS-2008107 and IIS-1703431.

REFERENCES

[1] Microsoft academic search. http://academic.research.microsoft.com.
[2] T-rex code repository and datasets. https://bitbucket.org/JanedoeforCIKM/systemxcikm.
[3] F. N. Afrati and P. G. Kolaitis. Repair checking in inconsistent databases: Algorithms and complexity. In *ICDT*, pages 31–41, 2009.
[4] E. Agichtein, E. Brill, and S. T. Dumais. Improving web search ranking by incorporating user behavior information. *SIGIR*, 52(2):11–18, 2018.
[5] M. Arenas, J. Pérez, and C. Riveros. The recovery of a schema mapping: Bringing exchanged data back. *ACM Trans. Database Syst.*, 34(4):22:1–22:48, 2009.
[6] L. Berti-Équille and U. Comignani. Explaining automated data cleaning with cleanex. In *IJCAI-PRICAI 2020 Workshop on Explainable Artificial Intelligence (XAI)*, 2021.
[7] L. Bertossi, L. Bravo, E. Franconi, and A. Lopatenko. The complexity and approximation of fixing numerical attributes in databases under integrity constraints. *Information Systems*, 33(4-5):407–434, 2008.
[8] L. E. Bertossi, S. Kolahi, and L. V. S. Lakshmanan. Data cleaning and query answering with matching dependencies and matching functions. *Theory Comput. Syst.*, 52(3):441–482, 2013.
[9] P. Bohannon, W. Fan, F. Geerts, X. Jia, and A. Kementsietsidis. Conditional functional dependencies for data cleaning. In *ICDE*, 2007.
[10] A. Chalamalla, I. F. Ilyas, M. Ouzzani, and P. Papotti. Descriptive and prescriptive data cleaning. In *SIGMOD*, pages 445–456. ACM, 2014.
[11] J. Chomicki and J. Marcinkowski. Minimal-change integrity maintenance using tuple deletions. *Inf. Comput.*, 197(1-2):90–121, 2005.
[12] X. Chu, I. F. Ilyas, and P. Papotti. Discovering denial constraints. *PVLDB*, 6(13):1498–1509, 2013.
[13] X. Chu, I. F. Ilyas, and P. Papotti. Holistic data cleaning: Putting violations into context. In *ICDE*, pages 458–469, 2013.
[14] D. Deutch, N. Frost, A. Gilad, and O. Sheffer. T-rex: Table repair explanations. In *SIGMOD*, pages 2765–2768, 2020.
[15] D. Dua and C. Graff. UCI machine learning repository, 2017.
[16] A. Ebaid, S. Thirumuruganathan, W. G. Aref, A. Elmagarmid, and M. Ouzzani. Explainer: Entity resolution explanations. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*, pages 2000–2003. IEEE, 2019.
[17] R. Fagin, B. Kimelfeld, and P. G. Kolaitis. Dichotomies in the complexity of preferred repairs. In *PODS*, pages 3–15, 2015.
[18] 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.
[19] W. Fan, X. Jia, J. Li, and S. Ma. Reasoning about record matching rules. *PVLDB*, 2(1):407–418, 2009.
[20] F. Geerts, G. Mecca, P. Papotti, and D. Santoro. The LLUNATIC data-cleaning framework. *PVLDB*, 6(9):625–636, 2013.
[21] A. Gilad, D. Deutch, and S. Roy. On multiple semantics for declarative database repairs. In *SIGMOD*, pages 817–831, 2020.
[22] D. Haas, S. Krishnan, J. Wang, M. J. Franklin, and E. Wu. Wisteria: Nurturing scalable data cleaning infrastructure. *PVLDB*, 8(12):2004–2007, 2015.
[23] D. Janzing, L. Minorics, and P. Blöbaum. Feature relevance quantification in explainable ai: A causal problem. In *AISTATS*, pages 2907–2916, 2020.
[24] M. N. Katehakis and A. F. Veinott Jr. The multi-armed bandit problem: decomposition and computation. *Mathematics of Operations Research*, 12(2):262–268, 1987.
[25] N. Koudas, A. Saha, D. Srivastava, and S. Venkatasubramanian. Metric functional dependencies. In *ICDE*, 2009.
[26] V. Kuleshov and D. Precup. Algorithms for multi-armed bandit problems. *arXiv preprint arXiv:1402.6028*, 2014.
[27] F. Li and H. V. Jagadish. Constructing an interactive natural language interface for relational databases. *PVLDB*, pages 73–84, 2014.
[28] S. Lipovetsky and M. Conklin. Analysis of regression in game theory approach. *Applied Stochastic Models in Business and Industry*, 17(4):319–330, 2001.
[29] E. Livshits, B. Kimelfeld, and S. Roy. Computing optimal repairs for functional dependencies. In *SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, pages 225–237, 2018.
[30] S. M. Lundberg and S. Lee. A unified approach to interpreting model predictions. In *NIPS*, pages 4765–4774, 2017.
[31] I. Mann and L. Shapley. Values for large games, iv: Evaluating the electoral college by monte carlo techniques. the rand corporation. *Research Memorandum*, 2651, 1960.
[32] D. Metzler and W. B. Croft. Linear feature-based models for information retrieval. *Inf. Retr.*, 10(3):257–274, 2007.
[33] C. Molnar. *Interpretable machine learning*. Lulu.com, 2020.
[34] C. H. Papadimitriou and M. Yannakakis. On the complexity of database queries. *J. Comput. Syst. Sci.*, 58(3):407–427, 1999.
[35] J. Rammelaere and F. Geerts. Explaining repaired data with cfds. *PVLDB*, 11(11):1387–1399, 2018.
[36] T. Rekatsinas, X. Chu, I. F. Ilyas, and C. Ré. Holoclean: Holistic data repairs with probabilistic inference. *PVLDB*, 10(11):1190–1201, 2017.
[37] M. T. Ribeiro, S. Singh, and C. Guestrin. "why should I trust you?": Explaining the predictions of any classifier. In *SIGKDD*, pages 1135–1144, 2016.
[38] L. SHAPLEY. A value for n-person games. *Contributions to the Theory of Games*, (28):307–317, 1953.
[39] A. Slivkins. Introduction to multi-armed bandits. *arXiv preprint arXiv:1904.07272*, 2019.
[40] E. Strumbelj and I. Kononenko. Explaining prediction models and individual predictions with feature contributions. *Knowl. Inf. Syst.*, 41(3):647–665, 2014.
[41] M. Sundararajan and A. Najmi. The many shapley values for model explanation. *arXiv preprint arXiv:1908.08474*, 2019.
[42] M. Sundararajan and A. Najmi. The many shapley values for model explanation. In *International Conference on Machine Learning*, pages 9269–9278, 2020.
[43] X. Wang, X. L. Dong, and A. Meliou. Data x-ray: A diagnostic tool for data errors. In *SIGMOD*, pages 1231–1245. ACM, 2015.
[44] X. Wang, A. Meliou, and E. Wu. Qfix: Diagnosing errors through query histories. In *SIGMOD*, pages 1369–1384. ACM, 2017.