How to Repair Data – The HoloClean Framework

Sebastian Schmidl
Hasso Plattner Institute
University of Potsdam
sebastian.schmidl@student.hpi.de

ABSTRACT

In a world, where big data and machine learning approaches are used in a growing number of enterprise applications, maintaining data quality has become more important than ever. Data repairing is one way to deal with that. Most repairing tools limit themselves to only one input signal, such as quantitative statistics, Integrity Constraints, or external data, to compute repairs. Rekatsinas et al. therefore propose a new holistic framework, called HoloClean [21], that observes multiple input signals and generates a probabilistic model out of unclean datasets and those signals. It uses DeepDive to perform statistical learning and probabilistic inference to compute the marginal probabilities of repairs, while scaling to dataset sizes with millions of tuples. Rekatsinas et al. show that HoloClean achieves an average accuracy improvement of 2× against single-input state-of-the-art approaches across four different datasets.

KEYWORDS

HoloClean; DeepDive; data cleaning; data repairing; holistic data repairs; probabilistic inference; factor graphs

1 DATA REPAIRING

Data is a central aspect in most enterprises. It is the basis for operational and strategic business decision making. However, real world data is flawed and contains errors, which can impair reporting, customer service, and production facilitation. Studies show that poor data quality leads to costs of billions of dollars [12, 20]. Maintaining the quality and consistency of business data has therefore become a critical task.

1.1 Data cleaning

One way to improve data quality is data cleaning. It consists of two steps: Error detection and data repairing. Error detection describes the process of identifying incorrect values in a dataset. Subsequently data repairing transforms the erroneous dataset into a new one removing detected errors, so the new dataset adheres to data quality requirements.

1.2 Related work

Many researchers have made efforts to automate the task of error detection. Most of the approaches try to detect the violation of Integrity Constraints (ICs), such as conditional functional dependencies [2] or denial constraints [4]. Other approaches developed techniques to detect duplicates [16, 17] or outliers [7, 14] in dirty datasets. Those automatic detection algorithms are already advanced and were shown to achieve good results on real-world datasets [1].

Data repairing techniques can be classified according to whether and how humans are involved in the repairing process. On the one end there are fully automatic approaches, such as SCARE [23], that leverage machine learning and intelligent partition algorithms to repair a database without the need of user input. On the other end there are data wrangling tools that make extensive use of human knowledge to clean a dirty dataset. For example, Data Wrangler [15], the successor of Potter's Wheel [19], provides a visual interface for cleaning a dataset. Based on data statistics it creates visual suggestions for data transformations. The user selects and executes those transformations to create a cleaned dataset instance. The tool can export the transformation sequence, to repeat the process on another dataset. This makes it an appropriate tool for the manual definition of ETL processes. The final version of the tool, Trifacta Wrangler [13], is developed commercially.

All the mentioned repairing tools [13, 15, 19, 23] use quantitative statistics of the dataset itself to repair errors in it. Besides that, state-of-the-art methods also use ICs [5, 8, 11, 22, 24] and external knowledge [6], such as dictionaries, knowledge bases, or domain expert annotations, as input signals. Figure 1 shows those tools aligned on their level of user interaction needed to perform data repairing and the usage of the three different input signals.

1.3 Introducing HoloClean

Most of the tools limit themselves to only one signal to perform data repairing, ignoring other information sources. Each type of signal is associated with a different action on the dataset and has its own downsides. Data repairs that use ICs could introduce incorrect repairs, because they assume that most of the data values are clean (*minimality principle*). This is not necessarily the case and minimal repairs do not always correspond to correct repairs [21]. Repair algorithms relying on external information are dependent on the coverage of the external data source and can therefore perform poorly. Quantitative statistics heavily depend on the available information in the dataset itself. For small datasets and unfortunate situations this can drastically decrease repairing quality.

Rekatsinas et al. therefore propose a holistic data repairing approach that includes all aforementioned signals into one framework: HoloClean [21]. To tackle the problem that different signals could suggest conflicting repairs they convert all signals into features of a probabilistic model. Based on this combined view on the data and its inconsistencies they can then use statistical learning and probabilistic inference to suggest repairs for the dataset.

1.4 Overview

In the following work, we explain the framework HoloClean [21] developed by Rekatsinas et al. in more detail. Section 2 first introduces ICs, factor graphs and how they are used in DeepDive for the purpose of data repairing. The HoloClean framework is described in section 3. After explaining the general architecture, we

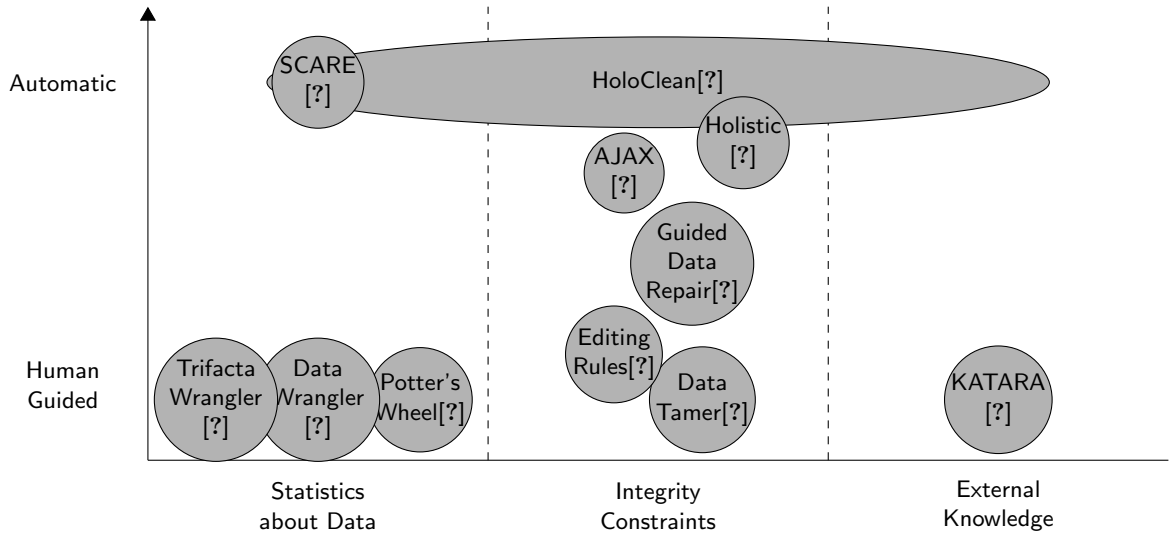

Figure 1: Data Repairing Tools in Context

go deeper into the three consecutive steps HoloClean executes to repair a dataset: Error detection in section 3.1, probabilistic model generation in section 3.2, and data repairing in section 3.3. Section 4 gives an overview about the accuracy of the suggested repairs and the runtime of HoloClean compared to other state-of-the-art approaches. The conclusion completes this work in section 5.

2 BACKGROUND

In the following paragraphs we review the terminology used in the next sections.

2.1 Integrity constraints

Users of the HoloClean framework can specify ICs in form of denial constraints, which combine different types of integrity definitions. Functional Dependencies as well as conditional functional dependencies can be expressed as denial constraints [3]. A denial constraint states that all predicates of it cannot be true at the same time, otherwise, there is a consistency conflict. It is notated as $\forall t_i, t_j \in D : \neg(P_1 \wedge \cdots \wedge P_K)$. Over all tuples $t \in D$ of a database instance D not all predicates P_k are allowed to be true. P_k are defined as $v_1 \circ v_2$ or $v_1 \circ c$ with $v_1, v_2 \in \{t_i.A, t_j.A\}$, where A denotes an attribute of the dataset D. c represents a constant value that is used to build the predicates. The operator $\circ$ is one of $\{=, <, >, \neq, \leq, \geq\}$ [4]. For readability reasons we use functional dependencies of the form $A_1 \rightarrow A_2$ to express integrity constraints in this report. They can easily be converted to denial constraints [3].

2.2 Factor graphs in DeepDive

HoloClean relies on DeepDive to perform statistical learning and inference. It is a data management system developed at Stanford University, which allows scalable statistical inference on big unclean datasets [18]. HoloClean uses DeepDive to create a factor graph [9] over all input signals and data cells. The factor graph can be defined via a declarative language, called DDlog. A collection of inference rules in DeepDive corresponds to a probabilistic model. Such inference rules are defined over a relation of random variables with weight annotations. We consider the following DDlog rule as a template for the inference rules generated by the HoloClean framework:

$$Value?(t, a, d) :- \\ Relation(t, a, d), [\text{conditions}], weight = \text{w} \quad (1)$$

$Value?(t, a, d)$ is the head of the rule and defines a random variable identified by t, a and d. HoloClean uses t to identify a tuple and a to identify an attribute. d corresponds to the value of cell $t.a$. The body of the rule consists of three parts: A number of other relations over the used variables (in the example, there is only one: $Relation(t, a, d)$), conditions using the same operators as denial constraints enclosed in brackets, and a weight annotation ($weight$ = w). Grounding of those rules will generate the nodes and factors of the factor graph in DeepDive.

3 THE HOLOCLEAN FRAMEWORK

Figure 2 shows an overview over the framework's architecture. It takes as input a dirty dataset and repairing constraints. Those are ICs in form of denial constraints and external clean data in form of matching dependencies and a dictionary. Matching dependencies map values of the unclean dataset cells to the corresponding entries in an external clean dictionary. As HoloClean treats error detection as a black box, the user has to specify error detection algorithms as an additional side-input.

HoloClean outputs repair suggestions for all cells which were not marked as clean. For each noisy cell it computes the marginal probabilities of all values of its domain. For example, if the cell $t_1.a$ is assigned value $\hat{v}$ with a probability of 0.73, this means that HoloClean is 73% confident about this repair. In a second step, HoloClean repairs the noisy cells by assigning it to the value suggestions with the highest probability.

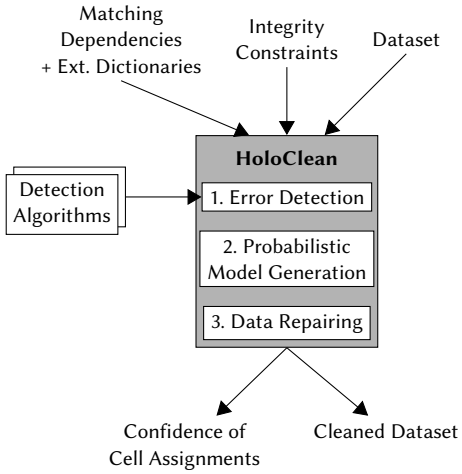

Figure 2: HoloClean Architecture Overview

To compute these repairs, HoloClean performs three consecutive steps: Error Detection, Probabilistic Model Generation, and Data Repairing. Those steps are described in the next sections in more detail.

3.1 Error detection

According to the data cleaning process, HoloClean separates clean cells from potentially dirty cells of the dataset D as a first step. As HoloClean treats this error detection step as a black box, the user can specify and use any method to separate dirty and clean cells from each other. The framework already provides exemplary algorithms that use denial constraints, outlier detection, or external labeled data for the separation of the cells. It is possible to use several detection algorithms at once. In this case the final set of dirty cells D_d is the union of all sets of dirty cells and the final set of clean cells D_c is the set $D_c = D \setminus D_d$.

3.2 Probabilistic model generation

HoloClean generates a probabilistic program written in DDlog, which is executed using DeepDive. This program describes a graphical probabilistic model over all cells of the input database and the various input signals.

Each cell in the dataset D is assigned to a random variable, which can take values of a finite domain. The random variables are linked via repairing constraints, which are derived from the input signals. They express the uncertainty over the values of noisy cells. HoloClean uses a factor graph to represent this probabilistic model and perform computations on it.

The compilation process consists of three steps which are executed consecutively:

(1) Generate DDlog relations.
(2) Use those relations to form inference rules.
(3) Use DeepDive to ground the model, yielding the factor graph.

First HoloClean generates DDlog relations, which encode information about the dataset D. They can be obtained by simple transformations of the original dataset and are shown in table 1.

Table 1: Generated DDlog Relations in HoloClean. Obtained from [21]

	Relation	Description
1	**Tuple(t)**	Relation of all tuple identifiers.
2	**InitValue(t,a,v)**	Maps a cell $t.a$ to its initial value v in the input dataset.
3	**Domain(t,a,d)**	Maps a cell $t.a$ to its finite domain $d \in Y_{t.a}$.
4	**HasFeature(t,a,f)**	Maps a cell $t.a$ to a feature vector f.
5	**ExtDict($\mathbf{t_k}$,$\mathbf{a_k}$,v,k)**	Maps a dictionary cell $t_k.a_k$ to its value v in the dictionary k.

Variable t identifies tuples in the dataset D and a identifies attributes in D. Relation 5 (**ExtDict**) is optional and only generated if there is an external dictionary and matching dependencies provided by the user.

HoloClean creates a categorical random variable for each cell $t.a$ in the dataset using relation 3 (**Domain**):

$$Value?(t, a, d) : -Domain(t, a, d) \tag{2}$$

Those random variables form the set of variable vertices that are connected with edges to factor vertices in the final factor graph. The factor vertices, short factors, encode repairing constraint on the random variables. During the generation of the factor graph, large domains of the random variables can lead to a combinatorial explosion [10]. Instead of assigning the whole domain of the corresponding attribute in the original dataset, HoloClean prunes the number of values a random variable can take in the relation **Domain**. It only considers those values of a cell for relation **Domain**, that exist in the domain of the cell's attribute and co-occur with other values of the cell's tuple with a probability greater than a threshold τ.

This means for each cell $t.A \in D$ HoloClean creates a new value-set $Y_{t.A}$ only containing those values $v \in dom(A)$ with a co-occurrence probability higher than the threshold $P(v|v') \geq \tau$. The co-occurrence probability for a value v of cell $t.A$ is computed based on v and the value of all other cells in $t.A$'s tuple $v' \in \{\forall a \in D \setminus \{A\} : v_{t.a}\}$:

$$P(v|v') = \frac{\#(v_i, v') \text{ appear together in } D}{\#v' \text{ appears in } D} \text{ [21]} \tag{3}$$

The threshold τ allows a trade-off between the quality of repairs and the runtime of HoloClean for big datasets. It controls the size of the random variable's domain and therefore the size of the grounded factor graph.

After definition of the random variables, the DDlog relations are used to form inference rules over the random variables:

Minimality principle. HoloClean uses the minimality principle as the prior that the dataset consists of more clean cells than erroneous ones. This is encoded in the following rule, which maps each cell's value to its initial value in the input dataset:

$$Value?(t, a, d) : -InitValue(t, a, d)\ weight = w \tag{4}$$

The weight w is set to a constant value during compilation of the DDlog rules and represents the strength of this prior.

Quantitative statistics. Relation **HasFeature** is used to capture quantitative statistics of the input dataset:

$$Value?(t, a, d) \ :- \ HasFeature(t, a, f) \ weight \ = \ w(d, f) \quad (5)$$

It stores the values of other cells in the same tuple of a given cell in a feature vector to encode the co-occurrences of attribute values. Users can customize HoloClean by adding more features to the **HasFeature** relation. This allows for example to track the provenance of cell values and prioritize data sources differently. HoloClean parametrizes the weight annotation for the feature inference rule by the value of the cell d and the feature vector f to allow different confidence levels for the combinations of the values of d and f. Those weights are not set during the generation of the rules itself and can be learned later on.

External information. Based on external dictionaries and matching dependencies HoloClean calculates all matches of the noisy dataset's cells and the values found in the dictionaries for the cells. These matches are stored in a new relation **Matched(t,a,v,k)**, where k denotes the dictionaries, which provided the corresponding values v. For each dictionary k and all cells (identified by $t.a$) the relation **Matched(t,a,v,k)** looks up the suggested dictionary value. This relation is used to create the following inference rule for the external information:

$$Value(t, a, d) : -Matched(t, a, d, k) \ weight = w(k) \quad (6)$$

Each dictionary can be trusted more or less and therefore the weights for these rules are parameterized by the dictionary identifier k. The weights can be learned later on.

Integrity constraints. The third input signal are ICs. They enforce constraints between the values of cells and therefore would connect multiple random variables with a factor in the factor graph.

In a later step HoloClean uses Gibbs sampling [25] to perform inference on the factor graph. This process iterates over all random variables of the factor graph and samples a value for a single variable from its conditional distribution, while keeping all other variables fixed. This approximate inference process is still computationally expensive. However, if there are only independent random variables in the factor graph, Gibbs sampling requires only $O(n \log n)$ iterations to converge.

Up until now, all introduced inference rules do not cause any dependencies between random variables, because they only contain one variable in their head expression. To avoid dependent random variables and utilize the guaranteed runtime of Gibbs sampling, HoloClean does not encode ICs with multiple random variables in the head of the inference rules as one would expect. For each denial constraint HoloClean generates multiple inference rules containing one of the involved cell's random variables in the head of the rule and the predicate constraints and the other cell's value mappings in the body. Under the following two assumptions this approximated approach can be used:

(1) There must exist more clean cells in the dataset than erroneous ones. This assumption represents the already introduced minimality prior.
(2) Each violation of an IC should be repairable by changing just one of the involved cells.

Lets consider the following denial constraint only containing two predicates that represents the functional dependency $t.A \rightarrow t.B$ as an example:

$$\forall t_1, t_2 \in D : \neg(t_1.A = t_2.A \wedge t_1.B \neq t_2.B) \quad (7)$$

Each instance of this constraint involves four cells ($t_1.A$, $t_1.B$, $t_2.A$ and $t_2.B$) of two tuples (t_1 and t_2). The quantifier of denial constraints ($\forall t_1, t_2$) is converted to a self-join of relation **Tuple**. This means, we only have to consider the cells of one tuple to be in the head of the rules:

$$\begin{aligned} !Value?(t_1, A, v_1) :- \\ & InitValue(t_2, A, v_2), InitValue(t_1, B, u_1), \\ & InitValue(t_2, B, u_2), Tuple(t_1), Tuple(t_2) \\ & [t_1 \neq t_2, v_1 = v_2, u_1 \neq u_2] \ weight = w \end{aligned} \quad (8)$$

$$\begin{aligned} !Value?(t_1, B, u_1) :- \\ & InitValue(t_2, B, u_2), InitValue(t_1, A, v_1), \\ & InitValue(t_2, A, v_2), Tuple(t_1), Tuple(t_2) \\ & [t_1 \neq t_2, v_1 = v_2, u_1 \neq u_2] \ weight = w \end{aligned} \quad (9)$$

The body of the rules then includes all other involved cells via relation **InitValue**. This relation maps the cells to their initial value and encodes the first assumption in DDlog. Additionally, relation **Tuple** is used to map the cells to their corresponding tuples and implement the self-join. The conditions of the predicates in the IC are applied on the cell's values (v_1, v_2, u_1, and u_2) and are enhanced with an additional condition preventing the join of a tuple with itself ($t_1 \neq t_2$). This template can then be applied to all the allowed combinations of tuples in D.

The weight w of each rule specifies how much emphasis is put on fulfilling it. Larger weights imply more emphasis on the rule. HoloClean does not set the weights of those rules to a fixed value, but rather estimates those weights during training of the model.

After all relations and inference rules were generated, the DDlog program is passed to DeepDive to create the factor graph. This step will generate all random variables for the cells and the factors between them. Some weights are already set by HoloClean, others are just represented by variables and will be learned in the next step.

3.3 Data repairing

To repair the dataset D HoloClean first has to learn all unset weights in the factor graph. This is done via statistical learning in DeepDive. Random variables in the factor graph that correspond to the clean cells in set D_c are used as labeled examples.

After fixing all weights HoloClean has to infer the correct values for the dirty cells in set D_d. Therefore, HoloClean computes the distribution of all values the random variables can be assigned to. For all random variables corresponding to the cells in D_d the marginal probabilities for all possible value assignments are obtained via Bayes inference over the joint distribution. As this is a very computationally expensive task an approximate approach is used: Gibbs sampling [25]. It is a sampling technique that estimate the full posterior probability distributions for all random variables corresponding to the noisy cells.

HoloClean's goal is to output a cleaned dataset. The system achieves this goal by assigning the maximum a posteriori estimates as the values of the noisy cells. Each repair is automatically annotated with its marginal probability which represents the confidence of the system that this repair is correct [21].

Example: If we consider a cell c, its domain $Y_c = \{v_1, v_2\}$ and the inferred posterior distribution

$$P(c) = \begin{bmatrix} P(c = v_1) \\ P(c = v_2) \end{bmatrix} = \begin{bmatrix} 0.81 \\ 0.19 \end{bmatrix} \tag{10}$$

then HoloClean proposes the value v_1 as a repair for cell c as it is the estimate with the highest marginal probability. This means that HoloClean is 81% confident about this repair. By implication, this also means that 19 out of 100 repairs with probability 0.81 will not be correct.

4 PERFORMANCE OF HOLOCLEAN

In this section we summarize the results of the experiments made by Rekatsinas et al. We will look at the accuracy of HoloClean's repair suggestions and the runtime of the framework to compute these. Both measures are compared to three other state-of-the-art approaches: Holistic [5] uses denial constraints to detect and repair errors, KATARA [6] generates repairs based on external knowledge bases, and SCARE [23] considers statistics about the dataset to perform machine learning based repairing. Before going deeper into the results, we review the datasets used in the evaluation.

4.1 Datasets

Rekatsinas et al. selected four datasets with different sizes and types of errors for their experiments:

Hospital. This is a benchmark dataset with about five percent errors and ground truth data available for all cells. It consists of 19,000 cells and nine denial constraints. The most frequent errors in this dataset are duplicates.

Flights. This dataset is also available with ground truth data and consists of 14,262 cells and four denial constraints. A large proportion of the cells are erroneous, thus we can use this dataset to evaluate the robustness of HoloClean.

Food. This food inspection dataset from the City of Chicago is created by manually filled out forms and contains a lot of non-systematic errors, such as misspelled entries or duplicates. At the date of usage the dataset contained 5.78 million cells and seven denial constraints that capture the mentioned errors. 2,000 cells were manually labeled based on detections of Holistic, KATARA, and SCARE.

Physicians. The last dataset is the Physician Compare National dataset, which contains a lot of systematic errors due to misspellings or inconsistencies. This dataset contains 37.29 million cells and is the biggest one used in the experiments. Nine denial constraints are used to capture the errors in the dataset. The same technique as for the *Food* dataset was used to obtain 2,500 labeled cells.

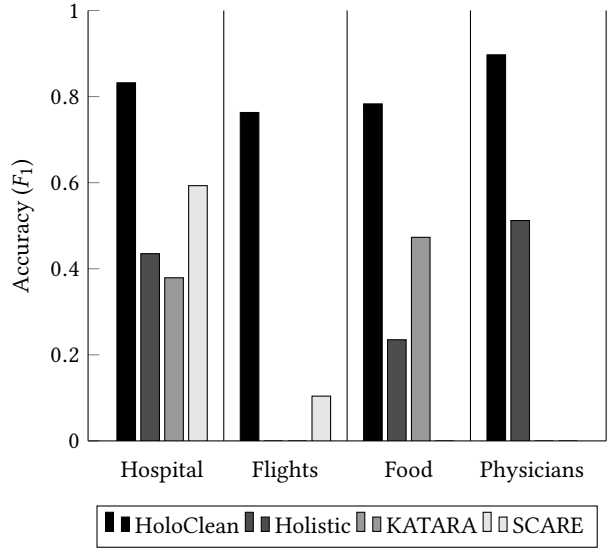

Figure 3: Accuracy (F_1-score) of HoloClean compared to other approaches. Obtained from [21]

4.2 Accuracy of repairs

We first look at the accuracy of the suggested repairs compared to the other three approaches. The accuracy is reported based on the F_1-score:

$$F_1 = 2 \cdot \frac{precision \cdot recall}{precision + recall} \tag{11}$$

The results for the different datasets are presented in Figure 3. For HoloClean the following thresholds τ for pruning the domain of random variables were used: Hospital $\tau = 0.5$, Flights $\tau = 0.3$, Food $\tau = 0.5$, and Physicians $\tau = 0.7$. Missing values (zero bars) in the diagram mean that no repairs were performed (Holistic and KATARA) or the process did not terminate after three days (SCARE).

HoloClean outperforms the other state-of-the-art approaches in all four datasets significantly. The F_1-score of HoloClean is more than 0.2 points higher than the one of the other approaches in all cases. Even on the *Flights* dataset, which has the highest error concentration, HoloClean reached an accuracy of 76%. This dataset was the hardest to fix, as we can deduce on the basis that Holistic and KATARA did not repair anything and SCARE only reached an accuracy of 10%.

This shows that the holistic approach used by HoloClean performs repairs with a significantly higher accuracy compared to other state-of-the-art approaches that only use one input signal.

4.3 Runtime of HoloClean

Considering the runtime to repair datasets, Rekatsinas et al. measured the end-to-end wall-clock time of each repairing method, including preprocessing and repairing. The results are shown in Table 2. As already mentioned, SCARE did not terminate after a three day runtime threshold for the datasets *Food* and *Physicians*. This is indicated with "n/t".

For a small dataset, such as *Hospital*, HoloClean does take significantly longer compared to all other three approaches. Besides

Table 2: Runtime of HoloClean compared to other approaches. Obtained from [21]

Dataset	HoloClean	Holistic	KATARA	SCARE
Hospital	148.0 sec	5.7 sec	2.0 sec	24.7 sec
Flights	70.6 sec	80.4 sec	n/a	14.0 sec
Food	32.8 min	7.6 min	1.7 min	n/t
Physicians	6.5 hours	2.0 hours	15.5 min	n/t

that, the runtime of HoloClean is in the same order of magnitude compared to Holistic. KATARA is faster than those approaches as it only matches cells with an external dictionary.

Overall HoloClean can scale to large datasets with several million cells. It takes longer to repair the dataset than other repairing approaches, but this overhead is justified by its accuracy improvements.

5 CONCLUSION

This paper introduced the HoloClean data cleaning framework developed by Rekatsinas et al. [21]. It is a holistic approach that considers multiple repair signals as inputs to generate a probabilistic model. Cells are separated into clean and potentially erroneous ones using user-supplied error detection algorithms. Those two datasets serve as labeled training data for the subsequent statistical learning of missing model weights. After this step, probabilistic inference is used to compute the marginal probabilities for the repair suggestions of erroneous cells. The maximum a posteriori estimates are used to repair the original dataset.

HoloClean can achieve 2× better accuracy than other state-of-the-art algorithms that only use a single input signal. HoloClean takes more time to compute the results, but the accuracy improvements justify the computational overhead. As HoloClean treats error detection as a black box, its performance greatly depends on the used error detection algorithms. It only performs repairs of those cells that were marked potentially erroneous by the error detection mechanism.

Rekatsinas et al. point out that future research directions could be the following: HoloClean outputs repairs with their marginal probabilities. Those could be used in a semi-supervised machine learning approach to get feedback from a user. This could improve HoloClean's suggested repairs. They also want to find out, when ICs can be relaxed to independent features on random variables and when not, as this part has a great impact on the scalability of the system and can be used in other approaches as well.

REFERENCES

[1] Ziawasch Abedjan, Xu Chu, Dong Deng, Raul Castro Fernandez, Ihab F Ilyas, Mourad Ouzzani, Paolo Papotti, Michael Stonebraker, and Nan Tang. 2016. Detecting data errors: Where are we and what needs to be done?. In *Proceedings of the VLDB Endowment*, Vol. 9. VLDB Endowment, 993–1004.

[2] Philip Bohannon, Wenfei Fan, Floris Geerts, Xibei Jia, and Anastasios Kementsietsidis. 2007. Conditional functional dependencies for data cleaning. In *Data Engineering, 2007. ICDE 2007. IEEE 23rd International Conference on*. IEEE, 746–755.

[3] Jan Chomicki and Jerzy Marcinkowski. 2005. Minimal-change integrity maintenance using tuple deletions. *Information and Computation* 197, 1-2 (2005), 90–121.

[4] Xu Chu, Ihab F Ilyas, and Paolo Papotti. 2013. Discovering denial constraints. In *Proceedings of the VLDB Endowment*, Vol. 6. VLDB Endowment, 1498–1509.

[5] Xu Chu, Ihab F Ilyas, and Paolo Papotti. 2013. Holistic data cleaning: Putting violations into context. In *2013 IEEE 29th International Conference on Data Engineering (ICDE)*. IEEE, 458–469.

[6] Xu Chu, John Morcos, Ihab F. Ilyas, Mourad Ouzzani, Paolo Papotti, Nan Tang, and Yin Ye. 2015. KATARA: A Data Cleaning System Powered by Knowledge Bases and Crowdsourcing. In *Proceedings of the International Conference on Management of Data SIGMOD*. ACM, 1247–1261.

[7] Kaustav Das and Jeff Schneider. 2007. Detecting anomalous records in categorical datasets. In *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Fiscovery and Data Mining*. ACM, 220–229.

[8] Wenfei Fan, Jianzhong Li, Shuai Ma, Nan Tang, and Wenyuan Yu. 2010. Towards certain fixes with editing rules and master data. *Proceedings of the VLDB Endowment* 3 (2010), 173–184.

[9] Nir Friedman and Daphne Koller. 2009. *Probabilistic Graphical Models: Principles and Techniques*. The MIT Press Cambridge, 123–124.

[10] Nir Friedman and Daphne Koller. 2009. *Probabilistic Graphical Models: Principles and Techniques*. The MIT Press Cambridge.

[11] Helena Galhardas, Daniela Florescu, Dennis Shasha, Eric Simon, and Cristian Saita. 2001. *Declarative data cleaning: Language, model, and algorithms*. Ph.D. Dissertation. INRIA.

[12] Anders Haug, Frederik Zachariassen, and Dennis van Liempd. 2011. The costs of poor data quality. *Journal of Industrial Engineering and Management* 4, 2 (2011), 168–193.

[13] Joe Hellerstein and Karthik Sethuraman. 2017. *Whitepaper: Making Data Wrangling Interactive at Any Scale*. Technical Report. Trifacta.

[14] Joseph M Hellerstein. 2008. Quantitative data cleaning for large databases. *United Nations Economic Commission for Europe (UNECE)* (2008).

[15] Sean Kandel, Andreas Paepcke, Joseph Hellerstein, and Jeffrey Heer. 2011. Wrangler: Interactive Visual Specification of Data Transformation Scripts. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 3363–3372.

[16] Nick Koudas, Sunita Sarawagi, and Divesh Srivastava. 2006. Record linkage: similarity measures and algorithms. In *Proceedings of the International Conference on Management of Data SIGMOD*. ACM, 802–803.

[17] Felix Naumann and Melanie Herschel. 2010. An introduction to duplicate detection. *Synthesis Lectures on Data Management* 2 (2010), 1–87.

[18] Feng Niu, Ce Zhang, Christopher Ré, and Jude W Shavlik. 2012. DeepDive: Web-scale Knowledge-base Construction using Statistical Learning and Inference. *VLDS* 12 (2012), 25–28.

[19] Vijayshankar Raman and Joseph M Hellerstein. 2001. Potter's wheel: An interactive data cleaning system. In *Proceedings of the VLDB Endowment*, Vol. 1. VLDB Endowment, 381–390.

[20] Thomas C. Redman. 2004. Data: An unfolding quality disaster. *Information Management Magazine* (aug 2004).

[21] Theodoros Rekatsinas, Xu Chu, Ihab F Ilyas, and Christopher Ré. 2017. Holoclean: Holistic data repairs with probabilistic inference. In *Proceedings of the VLDB Endowment*, Vol. 10. VLDB Endowment, 1190–1201.

[22] Michael Stonebraker, Daniel Bruckner, Ihab F Ilyas, George Beskales, Mitch Cherniack, Stanley B Zdonik, Alexander Pagan, and Shan Xu. 2013. Data Curation at Scale: The Data Tamer System.. In *CIDR*.

[23] Mohamed Yakout, Laure Berti-Équille, and Ahmed K Elmagarmid. 2013. Don't be SCAREd: use SCalable Automatic REpairing with maximal likelihood and bounded changes. In *Proceedings of the International Conference on Management of Data SIGMOD*. ACM, 553–564.

[24] Mohamed Yakout, Ahmed K Elmagarmid, Jennifer Neville, Mourad Ouzzani, and Ihab F Ilyas. 2011. Guided data repair. In *Proceedings of the VLDB Endowment*, Vol. 4. VLDB Endowment, 279–289.

[25] Ilker Yildirim. 2012. Bayesian inference: Gibbs sampling. *Technical Note, University of Rochester* (2012).