A Hybrid Data Cleaning Framework Using Markov Logic Networks

Congcong Ge, Yunjun Gao, *Member, IEEE*, Xiaoye Miao, Bin Yao, *Member, IEEE*, and Haobo Wang

Abstract—With the increase of dirty data, data cleaning turns into a crux of data analysis. The accuracy limitation of the existing integrity constraints-based cleaning approaches results from insufficient rules. In this paper, we present a novel hybrid data cleaning framework on top of Markov logic networks (MLNs), termed as MLNClean, which is capable of learning instantiated rules to supplement the insufficient integrity constraints. MLNClean consists of two steps, i.e., *pre-processing* and *two-stage data cleaning*. In the pre-processing step, MLNClean first infers a set of probable instantiated rules according to MLNs and then builds a two-layer MLN index structure to generate multiple data versions and facilitate the cleaning process. In the two-stage data cleaning step, MLNClean first presents a concept of *reliability score* to clean errors within each data version separately, and afterward eliminates the conflict values among different data version using a novel concept of *fusion score*. Considerable experimental results on both real and synthetic scenarios demonstrate the effectiveness of MLNClean in practice.

Index Terms—Data cleaning, integrity constraints, markov logic network

1 Introduction

Dirty data not only leads to erroneous decisions or unreliable analysis but probably causes a blow to the corporate economy [1]. As a consequence, there has been a surge of interest from both industry and academia in data cleaning [2]. Data cleaning includes two steps, i.e., error detecting and error repairing. The first step is to identify erroneous values, and the second step is to repair the detected errors. Existing data cleaning methods can be classified into two categories, i.e., (i) integrity constraints based (ICs-based for short) cleaning approaches [3], [4], [5], [6], [7], [8], [9], [10], which aims to clean errors violating ICs; and (ii) statistical-based cleaning methods, which construct appropriate models, and predict repair solutions on the basis of data distributions. Data experts in enterprises or academia, who are familiar with data in their respective fields, are the most probable target users of data cleaning. They can use the data cleaning approaches/prototypes to improve data quality in their specific areas for getting reliable analysis. Data experts are apt to favor the use of ICs-based data cleaning methods due to their interpretability. Specifically, ICs-based methods are easy to find the cause of errors, i.e., violation of ICs.

- *Congcong Ge, Yunjun Gao, and Haobo Wang are with the College of Computer Science, Zhejiang University, Hangzhou, Zhejiang 310027, China. E-mail: {gcc, gaoyj, wanghaobo}@zju.edu.cn.*
- *Xiaoye Miao is with the Center for Data Science, Zhejiang University, Hangzhou, Zhejiang 310058, China. E-mail: miaoxy@zju.edu.cn.*
- *Bin Yao is with the Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai 200000, China. E-mail: yaobin@cs.sjtu.edu.cn.*

Manuscript received 24 July 2019; revised 9 June 2020; accepted 13 July 2020. Date of publication 28 July 2020; date of current version 1 Apr. 2022. (Corresponding authors: Yunjun Gao and Xiaoye Miao.) Recommended for acceptance by B. Glavic. Digital Object Identifier no. 10.1109/TKDE.2020.3012472

Example 1. Table 1depicts a group of tuples sampled from a hospital information dataset T, which contains erroneous values highlighted in colored cells. It comprises four attributes, including hospital name (HN), city (CT), state (ST), and phone number (PN). The dataset needs to comply with three integrity constraints, i.e., (i) functional dependency (FD), which is a relationship that exists when one attribute value uniquely determines another attribute value; (ii) conditional functional dependency (CFD), which represents a FD under certain conditions; and (iii) denial constraint (DC), which is a declarative specification of rules which generalizes and enlarges the current class of ICs (including FDs and CFDs).

(r_1) FD: CT $\Rightarrow$ ST

(r_2) DC: $\forall t, t' \in T, \neg$(PN($t.v$) = PN($t'.v$) $\wedge$ ST($t.v$) $\neq$ ST ($t'.v$))

(r_3) CFD: HN("ELIZA"), CT("BOAZ")$\Rightarrow$PN ("2567688400")

where t and t' represent any two different tuples in the dataset, $t.v$ denotes the value in the specific attribute w.r.t the tuple t. As an example, PN($t.v$) means the value of tuple t on attribute PN. Rule r_1 indicates that a city value uniquely determines a state value. Rule r_2 indicates that two hospitals located in different states have different phone numbers. Rule r_3 means that a hospital called "ELIZA" and located in the city "BOAZ" has a specific phone number "2567688400". Errors appeared in the tuples relate to different error types, including *known erroneous values* and *unknown erroneous values*, in this example. A known erroneous value, also called a replacement error, means that a value is incorrectly recorded as another value within the same attribute domain. For instance, given the attribute domain related to the attribute CT as D ={"DOTHAN", "BOAZ"}, t_3.[CT] being "DOTHAN" is a replacement error, and the correct value of t_3.[CT] should be "BOAZ" in this cell. An unknown erroneous value refers to a value that is not in its

TABLE 1
Example of a Dirty Hospital Information Dataset

TID	HN	CT	ST	PN
t_1	ALABAMA	DOTHAN	AL	3347938701
t_2	ALABAMA	DOTH	AL	3347938701
t_3	ELIZA	DOTHAN	AL	2567638410
t_4	ELIZA	BOAZ	AK	2567688400
t_5	ELIZA	BOAZ	AL	2567688400
t_6	ELIZA	BOAZ	AL	2567688400

corresponding attribute domain. In this example, the typo caused by the typing process is a kind of unknown erroneous values. For example, t_2.[CT] being "DOTH" is a typo, and the correct value should be "DOTHAN" in this cell. These errors may lead to violating against ICs within the dataset. For instance, tuples t_4, t_5, and t_6 are violated on the attribute ST w.r.t. the rule r_1.

For data cleaning experts, it is easy to know the integrity constraints that a dataset should follow. Given integrity constraints, ICs-based methods first instantiate these rules by the corresponding values on the dataset and then detect errors that violate the constraints by judging the correctness of each instantiated rule. Taking values of tuples t_4 and t_5 in Table 1 as an example, r_1 can be instantiated to two rules, i.e., CT("BOAZ") $\Rightarrow$ ST("AK") and CT("BOAZ") $\Rightarrow$ ST ("AL"). It is obvious that the violation exists in t_4 and t_5 w.r. t. r_1, but the ICs-based methods cannot tell which instantiated rule is wrong. For small datasets, experts can judge the correctness of each instantiated rule manually. But it is impractical when encountering large datasets. Currently, many ICs-based approaches are proposed to solve this problem. First, methods relying on external matching dictionaries [11], [12], [13] consider the values in a dataset that mismatch any value in the external dictionary as errors. For example, if a dictionary only contains the information that the city "BOAZ" located in the state "AL", we can tell that the instantiated rule CT("BOAZ") $\Rightarrow$ ST("AK") is wrong. In other words, the value "AK" of t_4 is erroneous based on the external dictionary. However, when there exist unknown but correct instantiated rules that cannot find any relevant information from the dictionary, these methods may wrongly treat them as errors because of insufficient matching information. Second, some methods repair errors violating ICs with *minimality principle* [9], [11], which aims to minimize the impact on the dataset by trying to preserve as many values as possible. We know tuples t_4 and t_5 are violated on the attribute ST w.r.t. r_1. Thus, according to the principle of minimality, these methods replace the value "AK" with "AL" on the attribute ST of t_4, whereas it fails to repair the attributes CT of t_3. In addition, the attribute CT of t_2 cannot be repaired since it does not violate any IC. To be more specific, the instantiated rule of t_2 w.r.t. r_1 is CT ("DOTH") $\Rightarrow$ ST("AL"), and it does not conflict with other relevant instantiated rules. Third, some ICs-based techniques employ heuristic approaches [14], [15], which train a statistical model and guess erroneous values of each tuple based on the inferred probabilities. High quality of cleaning result needs a well-trained model that relies on sufficient data from a labeled pool. That is, those methods can get the appropriate possibility of each *known* value that has related information in the labeled pool, but are not able to handle any *unknown* value, which lacks corresponding information in the labeled pool.

Challenges. We summarize the challenges as follows.

1. *Instantiated rules insufficiency.* Errors may be missed due to the insufficient instantiated rules.
2. *Human involvement.* Humans involved in the data cleaning process can undoubtedly improve the quality of cleaning results. Crowdsourcing is an effective way to include humans in data cleaning tasks. Since data is increasing, effectively engaging the crowd into cleaning tasks requires designing easy-to-answer questions, reducing the number of questions, and minimizing monetary cost.
3. *Unknown values judgment.* Existing methods are difficult to clean the unknown erroneous values because of inadequate external information.

To address these challenges, we propose a novel hybrid data cleaning framework, termed as MLNClean, that leverages Markov logic network (MLN) to supplement the instantiated rules and make it possible to clean unknown errors. MLNClean first uses MLN [16] to generate a set of probable instantiated rules with their corresponding probabilities. Then, an MLN index is formed according to the given ICs. MLN index is built as a two-layer hash table. The first layer consists of a set of blocks, each of which has a set of groups in the second layer. One block corresponds to one integrity constraint. Each group consists of a set of data pieces and each piece with regard to an instantiated rule. We call the collection of data pieces in a block a *data version*. Thus, there are multiple data versions, each of which comes from a unique block. Next, MLNClean executes two cleaning stages, i.e., *cleaning multiple data versions* and *deriving the unified clean data*, which seamlessly perform detecting and error repairing for solving both *unknown* and *known* values.

Contributions. We make the key contributions below.

- *Instantiated rules supplement.* We present a variant of MLN for instantiated rules generations, which can learn the possibilities of instantiated rules by leveraging the Markov logic network, so as to make instantiated rules adequate for resulting the high quality of data cleaning (Section 3).
- *Human involved MLN index structure.* We develop an effective *MLN index* to generate multiple data versions and shrink the search space of errors (Section 4). In the first layer, blocks are generated according to the integrity constraints. In the second layer, we present a crowd-based algorithm (CBG for short) to generate groups for each block (Section 5). It is noteworthy that, instead of deciding whether each value is clean or not per time in traditional methods, MLN index chooses to decide whether one γ is clean or not per time. Hence, the efficiency of MLNClean is further gained.
- *Two-stage cleaning process.* MLNClean presents the two-stage cleaning process, i.e., *cleaning multiple data versions* and *deriving the unified clean data* to detect and repair both *unknown erroneous values* and *known erroneous values*. In the first stage, MLNClean executes

a reliability score based cleaning strategy (RSC) that detects and repairs errors within each data version according to the possibilities of instantiated rules (Section 6.1). In the second stage, MLNClean eliminates conflicting values among different data versions, and derives the final unified clean data (Section 6.2).

The rest of this paper is organized as follows. We review related work in Section 2. Then, Section 3 first introduces data cleaning semantics and then presents the concepts related to instantiated rules and Markov logic networks. In Section 4, we overview the framework of MLNClean. In Section 5, we describe how the groups of the MLN index are generated. Section 6 elaborates the two-stage data cleaning process. In Section 7, we report the experimental results and our findings, and then, we conclude our work in Section 8.

2 Related Work

*ICs-Based Cleaning Techniques.*ICs-based techniques have hitherto been in the majority. They detect errors violating integrity constraints, and leverage effective measures for repairing errors.

Many attempts [5], [9], [11], [17], [18] are made to repair errors using the minimality principle, which means the repair solution needs to retain as many original values as possible to achieve the minimum repair cost. However, they are not able to correctly repair costly errors. Nevertheless, ICs-based methods can still effectively deal with a part of errors, which inspire us to combine the principle of minimality and the probability derived from MLNs as a hybrid measure in MLNClean to cope with complex situations.

Some research efforts leverage external matching dictionaries for repairing errors [11], [12], [13]. The mismatched values between the original dataset and the external dictionaries are treated as errors. KATARA [11] and DRs [13] use KBs as external dictionary to identify and repair erroneous values. Some approaches use matching dependencies [11], [19], [20] to describe the matching process between the original dataset and the external dictionaries. They are limited by the insufficient external matching dictionaries. Different from the methods that only use matching dictionaries as the basis for data cleaning, MLNClean treats values in the external dictionaries as evidence and can infer new information to supplement the matching dictionaries.

Others employ heuristic models [14], [15], which first train statistical models based on the known labeled values and then clean values based on their possibilities inferred by the models. [14] only combines FDs and statistical methods without considering other types of integrity constraints. Closer to our work is HoloClean [15], which integrates an existing ICs-based error detection method to separate correct values from erroneous values, trains a probabilistic graphical model using the detected correct, and repairs every detected single erroneous value based on the inferred marginal probabilities. Different from HoloClean, our proposed MLNClean presents a variant MLN model for the support of inferring the possibilities of the co-occurrence of the values in each instantiated rule. Besides, HoloClean cannot clean values that do not exist in the known labeled data, and MLNClean makes up for the deficiency. The experiments demonstrate that MLNClean is superior to HoloClean in Section 7.2.

TABLE 2
Symbols and Description

Notation	Description
T	a dataset with dirty values
t_i	a tuple belonging to a dataset T
$t_i.[A]$	the value of t_i on attribute A
r_i	an integrity constraint/rule
γ	a data piece that represents the collections of attribute values in an instantiated rule
B_i	a block (corresponding to a rule r_i) in the MLN index over a dataset T
G_{ij}	a group containing a set of γs that share the same values on the reason part of the rule w.r. t. the block B_i

Statistical-Based Cleaning Techniques. Statistical-based techniques identify errors with abnormal behaviors and predict repair solutions on the basis of data distributions. ERACER [21], SCARE [22], ActiveClean [23], and HoloDetect [24] are among this category. ERACER is an iterative statistical framework based on belief propagation and relationship-dependent networks. SCARE cleans data by combining machine learning and probability models. It repairs errors based on the maximum likelihood estimation. ActiveClean is a stepwise cleaning method in which models are updated incrementally rather than retrained, and thus, the cleaning accuracy could gradually increase. HoloDetect employs an expressive model to represent values as vectors, and classifies values into the correct part and erroneous part. They are orthogonal to MLNClean because our presented framework focuses on clean errors by leveraging the integrity constraints, while statistical-based approaches clean errors that are treated as outliers.

Crowdsourcing. Crowdsourcing techniques are certainly valid and useful for data cleaning. Some approaches have been proposed in order to use crowd power to guide cleaning. For example, CrowdER [25] and Power [26] utilize crowdsourcing for entity resolution, which can be used for data deduplication in data cleaning tasks. KATARA [11] annotates data as either correct or incorrect by interleaving crowdsourcing and knowledge bases (KBs). We will show how crowdsourcing can help to clean errors in MLNClean (see Section 5).

3 Preliminaries

In this section, we first formalize the problem of data cleaning in Section 3.1 and then describe background materials and techniques used in sections later. Table 2 summarizes the symbols used frequently throughout this paper.

3.1 Problem Statement

We assume that errors in a dirty dataset T occur due to inaccurate value assignments, which is a common assumption made by many data cleaning systems [6], [11], [15]. For each structured dataset, we denote $A_1, A_2, \ldots, A_d$ the attributes that characterize T. Each attribute A_i has its specific correct

value domain, denoted as D_i. Thus, we denote $\mathcal{V} = D_1 \cup D_2 \cdots \cup D_d$ the whole value set that is collected from all correct value domains of the dataset. In MLNClean, given the correct attribute domain $\mathcal{V}$ of a dirty dataset, the inaccurate value assignments lead to two types of error problems: (i) *unknown* erroneous values, denoted as $\mathcal{V}_f = \{v_f^1, \ldots, v_f^i, \ldots, v_f^{|\mathcal{V}_f|} | v_f^i \notin \mathcal{V}\}$, and (ii) *known* erroneous values, denoted as $\mathcal{V}_t = \{v_t^1, \ldots, v_t^i, \ldots, v_t^{|\mathcal{V}_t|} | v_t^i \in \mathcal{V}\}$. The goal of MLNClean is to detect and repair both unknown and known erroneous values.

3.2 Instantiated Rules

Instantiated rules derive from the integrity constraints (ICs) by replacing their variables with the corresponding constants (i.e., attribute values). We desire to infer probable instantiated rules on the basis of a part of *evidence*. In MLNClean, an evidence contains two parts. The first part consists of a set of known *literals*. A literal is an expression that contains a predicate symbol applied to an attribute value, e.g., CT("DOTHAN"). The second part can be treated as an external dictionary, which is composed of sets of instantiated rules. This external dictionary is used to infer other possible instantiated rules exclusive of which exist in the evidence and learn appropriate probabilities for them. Also, if an instantiated rule can be found in the evidence, it is correct. As mentioned in Section 1, there are usually different types of integrity constraints that should hold on a dataset T, such as FDs, CFDs, and DCs. Since both FDs and CFDs can be easily transformed into DCs' format $\forall\, t_1, \ldots, t_k\ (\rho_1 \wedge \rho_2 \wedge \cdots \wedge \rho_n)$ where ρ_i represents a predicate (i.e., attribute name), existing data cleaning methods [15], [27] tend to use DCs to express different kinds of integrity constraints. However, it is challenging to infer possible instantiated rules using DCs' format due to the reason that DCs focus on whether the values of each tuple satisfy the constraints, while lacking the ability to express the causal relationship between the values.

In view of this, our current implementation choose *implication formulas* to express both FDs and CFDs. An implication formula has the following form: $\rho_1 \wedge \rho_2 \wedge \ldots \wedge \rho_m \Rightarrow q_1 \vee q_2 \vee \cdots \vee q_n$. The antecedent predicates belong to the *reason* part $P = \{\rho_1, \rho_2, \ldots, \rho_m\}$. The consequent predicates pertain to the *result* part $Q = \{q_1, q_2, \ldots, q_m\}$. According to the reason, the corresponding result can be derived. Take the sample dataset in Table 1 as an example, the FD r_1 and CFD r_3 are inherently expressed in implication formulas, and the DC r_2 can be transformed into an implication formula, i.e., PN $\Rightarrow$ ST. Assuming that the evidence contains the following information, i.e., (i) a hospital called "ALABAMA" locates in the state "AL", and (ii) a city "DOTHAN" is in the state "AL", we can infer that the hospital "ALABAMA" may locate in the city "DOTHAN". Formally, according to the evidence that (i) HN("ALABAMA") $\Rightarrow$ ST("AL") and (ii) CT ("DOTHAN") $\Rightarrow$ ST("AL"), there may exist a possible instantiated rule HN("ALABAMA") $\Rightarrow$ CT("DOTHAN").

3.3 Markov Logic Networks

MLNClean integrates Markov logic networks (MLNs) for generating possible instantiated rules exclusive of which exist in the evidence and assigning appropriate probabilities for them. In a traditional viewpoint of data cleaning study, if there are conflicting values between two instantiated rules, one of them has zero probability to be correct. The attraction of MLNs lies that, it is able to *soften* those constraints. We first briefly describe the concept of Markov logic networks, and then, we introduce a variant of MLNs for the purpose of inferring instantiated rules along with their probabilities. The formal definition of the Markov logic network is stated below.

Definition 1 (Markov logic network) [16]. *A Markov logic network L is defined as a set of rule-weight pairs (r_i, w_i), where r_i is an integrity constraint, and $w_i \in [0, 1]$ is a real-number weight of r_i.*

Each integrity constraint has an associated weight that reflects how strong the constraint is. In MLNClean, we assume that all ICs given by experts are correct and thus set the weight of each rule as $w_i = 1$ initially. Together with a finite set of attribute values $\mathcal{V}$ for each predicate within every r_i and an evidence containing a set of literals, it defines a Markov network $M_{L,\mathcal{V}}$. Thereafter, the probability of other possible literals exclusive of which exist in the evidence specified by $M_{L,\mathcal{V}}$ is given by

$$\Pr(x) = \frac{1}{Z} \exp\left(\sum_{i=1}^{N} w_i n_i(x)\right), \tag{1}$$

where x represents a literal, Z is the normalized function, which can be treated as a constant, $n_i(x)$ is the number of true groundings (evidence) of r_i in x, w_i is the weight of r_i, and N represents the number of ICs.

As defined in Equation (1), the evidence contains only a set of literals, and only the marginal probabilities of possible literals can be inferred. Nevertheless, it is not able to derive the joint distribution probability of the co-occurrence of values existing in each instantiated rule. We do a series of equivalent transformations on MLN for enabling it to infer the possibilities of instantiated rules, according to the following definitions.

Theorem 1. *Implication formulas satisfy the properties:*

(P.1) *(Inverse property)* $(P \Rightarrow Q) \Leftrightarrow (\neg Q \Rightarrow \neg P)$

(P.2) *(Idempotent law)* $P \Leftrightarrow (P \wedge P)$ *and* $Q \Leftrightarrow (Q \wedge Q)$

(P.3) *(Equation property)* $(P \Rightarrow Q) \Rightarrow (P \wedge S \Rightarrow Q \wedge S)$

Lemma 1. *For each r_i in the MLN, we have*

$$\begin{aligned} &\rho_1 \wedge \rho_2 \wedge \ldots \wedge \rho_m \Rightarrow q_1 \vee q_2 \vee \cdots \vee q_n \\ =&\neg(q_1 \wedge q_2 \wedge \cdots \wedge q_n) \Rightarrow \neg(\rho_1 \wedge \cdots \wedge \rho_m \wedge q_1 \wedge \cdots \wedge q_n) \end{aligned}$$

Proof. We give the proof according to the properties of Theorem 1 as follows.

$$\rho_1 \wedge \rho_2 \wedge \ldots \wedge \rho_m \Rightarrow q_1 \vee q_2 \vee \cdots \vee q_n \tag{2}$$

$$= (\rho_1 \wedge \rho_2 \wedge \ldots \wedge \rho_m) \wedge (q_1 \wedge \cdots \wedge q_n) \Rightarrow (q_1 \wedge \cdots \wedge q_n) \tag{3}$$

TABLE 3
Example of Instantiated Rules w.r.t. r_1

IC	Instantiated rules
CT $\Rightarrow$ ST	$\neg$ST("AL") $\Rightarrow \neg$(CT("DOTHAN") $\wedge$ ST("AL"))
	$\neg$ST("AK") $\Rightarrow \neg$(CT("DOTHAN") $\wedge$ ST("AK"))
	$\neg$ST("AL") $\Rightarrow \neg$(CT("BOAZ") $\wedge$ ST("AL"))
	$\neg$ST("AK") $\Rightarrow \neg$(CT("BOAZ") $\wedge$ ST("AK"))

$$= \neg(q_1 \wedge q_2 \wedge \cdots \wedge q_n) \Rightarrow \neg(\rho_1 \wedge \cdots \wedge \rho_m \wedge q_1 \wedge \cdots \wedge q_n), \tag{4}$$

where Equation (3) is derived from Equation (2) based on (P.2) and (P.3). Equation (4) is derived from Equation (3) based on (P.3). □

Thus, an instantiated rule can be generated based on Equation (4) by using any corresponding value from $\mathcal{V}$ for each predicate to form a literal. Table 3 shows an example of instantiated rules by selecting values from both attribute domains of CT and ST. For simplicity, we call the values co-occur in an instantiated rule a data piece, denoted as γ. Thus, the probability of each possible instantiated rule derived from Equation (1) is given by

$$\Pr(\gamma) = \frac{1}{Z}\exp\left(\sum_{i=1}^{N} w_i n_i(\gamma)\right). \tag{5}$$

4 MLNClean Framework

In this section, we briefly introduce the procedure of MLNClean. It receives a dirty dataset together with some evidence and a set of integrity constraints (ICs) with the format of implication formulas. It outputs clean data through *two* steps, including *pre-processing* and *two-stage data cleaning*. Fig. 1 illustrates the framework.

Pre-Processing. First, MLNClean infers the possible instantiated rules based on the given evidence by using a variant of MLN. Then, MLNClean builds a two-layer MLN index with a set of *blocks* in the first layer and a set of *groups* in the second layer. The MLN index is a *vital* structure, which helps to narrow the search space of repair candidates for the subsequent data cleaning phase. In the first layer, each block corresponds to one IC. In other words, the data pieces (γs) that belong to one same IC are put together to form one block. Thus, the number of blocks is equivalent to the number of IC. In the second layer, we present two approaches so-called AGP and CBG respectively, to generate groups for each block.

Two-Stage Data Cleaning. In the first cleaning stage, MLNClean cleans each group using an *r-score* based cleaning method (i.e., RSC), which will be elaborated in Section 6. After cleaning multiple version data independently, there naturally exist conflicts among different data versions. Thus, in the second cleaning stage, MLNClean strives to eliminate those conflicts using a newly defined concept of *fusion score* (i.e., f-score), in order to get the final clean data. It is with respect to an f-score based conflict resolution strategy (i.e., FSCR) (to be detailed in Section 6).

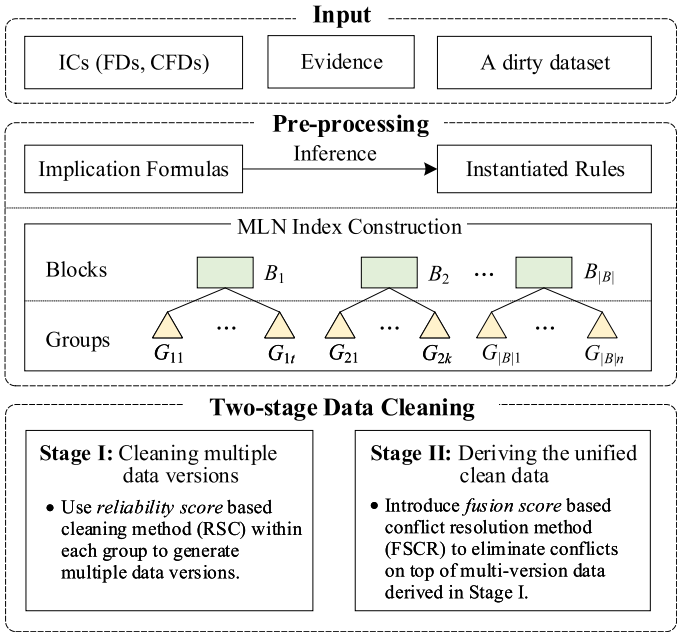


Fig. 1. MLNClean architecture.

5 Groups of MLN Index

According to the definition of implication formula, the attribute values in the antecedent/reason part uniquely determine the attribute values in the consequent/result part. Therefore, we could collect a set of data pieces (w.r.t the values co-occur in each instantiated rule) as a group, where the data pieces should be precisely the same according to the concept of the implication formula but the existence of errors makes them behave differently. In this section, we introduce two strategies for generating groups in the MLN index. First, we present a simple grouping strategy called AGP, and use it to illustrate the difficulty of putting each data piece into a correct group. Then, we focus on our proposed crowd-based grouping method, i.e., CBG, and show how we use crowdsourcing for solving the grouping problem. We will verify that the newly proposed CBG is better than AGP in our experiments (see Section 7.3.1).

Abnormal Group Process Strategy. Since the same reason in the implication formulas is not able to produce different results, we collect a set of data pieces γs with the same reason part into one group. As a result, in the second layer of MLN index, each block is divided into several groups and γs within a group share the same reason values. Take the sample dataset in Table 1 as an example. We depict the MLN index structure in Fig. 2. There are three blocks B_1, B_2, and B_3 corresponding to three rules r_1, r_2, and r_3 respectively shown in Example 1. They have 3, 3, and 2 groups, respectively. Let $|B|$ and $|T|$ be the number of blocks (or rules) and tuples in the dataset, respectively. In addition, it is easy to realize that, there might be multiple data pieces pertaining to each tuple in the dataset, and each of them comes from different blocks. In other words, for each tuple, there are $|B|$ data pieces derived from it. Therefore, we can say that, there are *multiple data versions*, each of which comes from different blocks.

Based on an MLN index, for a tuple with error(s) in the reason part of a rule, the corresponding data piece (i.e., γ)

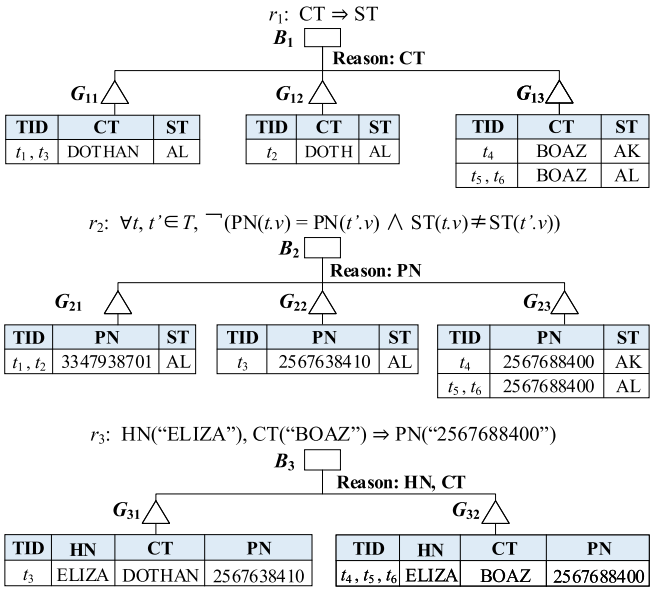


Fig. 2. Illustration of the MLN index over the sample dataset.

might erroneously form or belong to a group, and thereby, we call the corresponding data piece *an abnormality*. For example, there is a typo t_2.[CT] being "DOTH" in the sample dataset shown in Table 1. It relates to a data piece γ_{t_2} = {CT: DOTH, ST: AL} $\in G_{12}$ in the MLN index, as depicted in Fig. 2. Actually, the γ_{t_2} should be in the group G_{11} which contains γ ={CT: DOTHAN, ST: AL}. To this end, we propose an *abnormal group process* strategy, termed as AGP, to put the abnormalities into the correct groups.

AGP utilizes a threshold to distinguish abnormal groups and regular groups. Specifically, if the number of tuples related to γs contained in a group is not larger than a threshold τ_{AGP}, AGP regards this group as an abnormal group; otherwise as a normal group. The optimal value of τ_{AGP} is empirically chosen. Then, for each abnormal group in a block B_i, AGP merges it with its nearest normal group within B_i. Specifically, let γ^* be the data piece related to the most tuples in a group. The distance of two groups is defined as the distance of their respective γ^*s. For instance, in terms of the MLN index shown in Fig. 2, when setting τ as 1, groups G_{12}, G_{22}, and G_{31} are identified as abnormal groups. Then, for G_{12}, its nearest group is G_{11}. Thus, G_{12} is merged with G_{11}. Similarly, G_{22} is merged with G_{23}, and G_{31} is merged with G_{32}.

Crowd-Based Grouping Strategy. Since grouping correctly data pieces helps directly to guarantee the subsequent cleaning process performance, it never overstates the importance of this step. However, the number of tuples related to γs contained in a regular group may be smaller than τ_{AGP}. It is not very flexible to get a proper threshold value and hence restricts the advantage of AGP method. Crowdsourcing methods are designed to solve the problems that are computer hostile but human friendly. Thus, we employ crowdsourcing for identifying a set of data pieces that belong to the same group within each block via consulting crowd workers through a series of questions.

Algorithm 1 shows the pseudo-code of the proposed crowd-based grouping method (CBG for short). Initially, CBG generates data piece pairs according to γs in each block B_i (line 2). Given two data pieces γ_{t_a} and γ_{t_b} (t_a and t_b represent two different tuples) belonging to B_i, the pair $(\gamma_{t_a}, \gamma_{t_b})$ can be denoted as p_{ab}. Intuitively, we do not need to generate the data piece pair if they are extremely different from each other, meaning that they have low probabilities referring to the same group. We use $S(\gamma_{t_a}^{A_k}, \gamma_{t_b}^{A_k})$ to denote the similarity between γ_{t_a} and γ_{t_b} on attribute A_k and prune pairs whose similarities are less than a threshold τ_{CBG} on every attribute. Theoretically, we can employ any similarity metric to measure the similarity of each pair. In our current implementation, we utilize the edit similarity.

Edit similarity can be derived from edit distance. Edit distance (ED) quantifies the dissimilarity between γ_{t_a} and γ_{t_b} on attribute A_k by counting the minimum number of operations required to transform one string into the other. The equation of edit similarity is as follows

$$S_{edit}(\gamma_{t_a}^{A_k}, \gamma_{t_b}^{A_k}) = 1 - \frac{ED(\gamma_{t_a}^{A_k}, \gamma_{t_b}^{A_k})}{max(|\gamma_{t_a}^{A_k}|, |\gamma_{t_b}^{A_k}|)}, \tag{6}$$

where $\gamma_{t_a}^{A_k}$ and $\gamma_{t_b}^{A_k}$ represent the value of γ_{t_a} and γ_{t_b} on attribute A_k separately.

Algorithm 1. Crowd-Based Grouping Method (CBG)

Input: a set of blocks $\mathcal{B} = \{B_1, B_2, \ldots, B_{|R|}\}$
Output: a set of blocks $\mathcal{B}^*$
1: **foreach** $B_i \in \mathcal{B}$ **do**
2: $P_i \longleftarrow$ generate pairs from B_i
3: get the answer of each pair $p \in P_i$ from the crowd
4: $P_i^* \leftarrow \varnothing$
5: **foreach** $p \in P_i$ **do**
6: **if** *the answer of p is "Yes"* **then**
7: add p into P_i^*
8: **while** P_i^* *is not empty* **do**
9: $p_0^* \longleftarrow$ get the first pair in P_i^*
10: $G^* \longleftarrow$ find all γs related to $p_0^*.\gamma_1$ and $p_0^*.\gamma_2$
11: remove the γs belonging to G^* from P_i
12: add G^* into B_i^*
13: **return** $\mathcal{B}^*$

Then, CBG algorithm asks questions to the crowd for identifying whether the data pieces refer to the same group (line 3). We generate questions in pairwise comparisons, where each question is composed of a pair of two data pieces. For each pair, crowd workers are required to answer whether the two data pieces belong to the same group. Fig. 3 shows an example of the pairwise comparison-based question. Two data pieces γ_{t_1} ={CT: DOTHAN, ST: AL} (w.r.t. t_1) and γ_{t_2} ={CT: DOTH, ST: AL} (w.r.t. t_2) are asked whether they belong to the same group or not. In addition to the data pieces themselves, the attribute values of relevant tuples are provided for instructing the crowd to choose the answers in higher accuracy.

Note that it is impractical to use a brute-force method that enumerates all pairs from a dataset. Take the hospital dataset as an example, which has 231,265 tuples. It will generate $\frac{231265 \times 231264}{2}$ (more than 26 billion) pairs/questions in the worst case where the data piece for each tuple is different from each other. It may consume a huge amount of monetary and time costs.

Pair #1 w.r.t Rule r_1: CT ⇒ ST

TID	Piece of data		Relevant Tuple
t_1	CT: DOTHAN	ST: AL	HN: ALABAMA, CT: DOTHAN, ST: AL, PN: 3347938701
t_2	CT: DOTH	ST: AL	HN: ALABAMA, CT: DOTH, ST: AL, PN: 3347938701

Your Choice:
☒ They belong to the *same* group. (Yes)
☐ They belong to *different* groups. (No)

Fig. 3. Illustration of a pairwise comparison-based question.

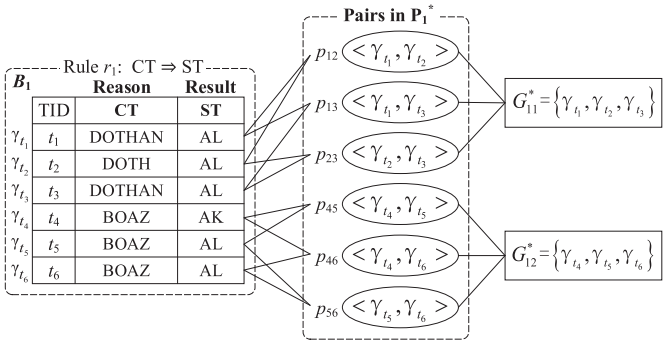


Fig. 4. Example of generating groups.

MLNClean uses a *partial-order* based algorithm [26], which greatly minimizes the number of asked pairs, to reduce the crowdsourcing cost without at the expense of reducing the quality. We overview the algorithm procedure as follows. To begin with, the partial-order based algorithm constructs a directed acyclic graph, where each vertex in the graph is a data piece pair and each edge describes the similarity relationship between two pairs. Next, it selects some optimal pairs (vertexs) in the graph as questions, and asks the crowd to give answers. After getting the answer of a pair, it infers the answer of other relevant pairs based on the partial order. Finally, the algorithm repeats the previous step to find the remaining optimal pairs until all pairs on the graph have their answers. Therefore, we only need to ask some optimal pairs to get all the corresponding answers of pairs. Besides, since different pairs may contain the same value, we can assign the answer of a pair to all pairs containing the same value to further reduce the crowdsourcing cost. For example, γ_{t_4} ={CT: BOAZ, ST: AK}, $\gamma_{t_5}, \gamma_{t_6}$ ={CT: BOAZ, ST: AL}. Thus, the pair $p_{45} = \{\gamma_{t_4}, \gamma_{t_5}\}$ and $p_{46} = \{\gamma_{t_4}, \gamma_{t_6}\}$ contain the same value. We can fill in the same answer for p_{45} and p_{46}.

Third, CBG records all the pairs with the answer "Yes" in each $\mathcal{P}_i^*$ (line 4-7). Finally, CBG generates groups based on the identified pairs in each $\mathcal{P}_i^*$ (line 8-12). Take the block B_1 w.r.t rule r_1 as an example, Fig. 4 shows the process of grouping. The middle part of the diagram displays pairs (whose answers are "Yes") in $\mathcal{P}_1^*$. The right of the diagram depicts the result of groups G_{11}^* and G_{12}^*. We first take out the top pair p_{12} from $\mathcal{P}_1^*$ (line 9). Since p_{12} corresponds to tuples t_1 and t_2, we enumerates every pairs from $\mathcal{P}_1^*$ to find other pairs which are related to t_1 and t_2. Because p_{13} correlates with t_1, we put the γs in p_{12} and p_{13} into the same group. However, p_{13} also correlates with t_3, and thus, we need to find other pairs related to t_3. We repeat the above process until all relevant γs are included in the same group (line 10). As a result, $\gamma_{t_1}, \gamma_{t_2}$, and γ_{t_3} are included into G_{11}^*, $\gamma_{t_4}, \gamma_{t_5}$, and γ_{t_6} are included into G_{12}^*.

6 Two-Stage Data Cleaning

In this section, we present the two-stage data cleaning process in MLNClean, including cleaning multiple data versions and deriving the unified clean data.

6.1 Cleaning Multiple Data Versions

Ideally, if data are clean, one group contains one and only one data piece, indicating that the same values on the reason part cannot derive different values on the result part. However, when one group contains different data pieces, there definitely exist dirty values. In light of this, we present a cleaning strategy using the concept of *reliability score*, called r-score based cleaning (RSC for short), to clean erroneous values within each group.

The reliability score is introduced to evaluate the probability of each data piece γ being correct. It considers two factors: (i) *Minimality cost*. According to the principle of minimality, we should retain the γ associated with the most tuples. Therefore, the greater the number of tuples related to a γ, the more likely the γ is correct. (ii) *Possibility*. It indicates the possibility of the γ being correct. As mentioned in Section 3.3, the higher the possibility, the more likely the γ is correct.

Definition 2 (Reliability score). *For a data piece γ in a group G, its reliability score, denoted as r-score(γ), is defined in.*

$$\gamma^*) \times \Pr(\gamma)\text{r-score}(\gamma) = N(\gamma) \times \Pr(\gamma), \tag{7}$$

where $N(\gamma)$ represents the number of tuples related to the γ and $\Pr(\gamma)$ is computed by Equation (5).

RSC judges which data piece included in the group is correct using the reliability score. Note that, a data piece containing unknown erroneous values cannot find its corresponding $Pr(\gamma)$. In view of this, in each group, we uniformly replace all unknown values with the corresponding values in the next data piece that does not contain unknown values. Then, RSC corrects the other erroneous data pieces with the detected correct one, so that each group has one and only one data piece eventually. As a result, all the other data pieces within the same group are replaced with the data piece having the highest reliability score. It is worth noting that, RSC cleans pieces of data within every *block* independently, which does not need the information outside the block. Actually, we can even know that, RSC is executed to clean data within each *group* separately, if regardless of Markov weight learning. In other words, what we would like to highlight here is that, the MLN index structure indeed helps to minimize the search space for RSC.

Example 2. Take the group G_{13} belonging to block B_1 (as depicted in Fig. 2) as an example. G_{13} includes two data pieces, denoted as γ_1 and γ_2, namely, γ_1 is {CT: BOAZ, ST: AK} and γ_2 is {CT: BOAZ, ST: AL}. They have the same value on the reason part but different values on the result part. Obviously, there is at least one error at the value of attribute ST within the group, according to the

TID	CT	ST	N(γ)	Pr(γ)	r-score
t_4	BOAZ	AK	1	0.73	0.73
t_5, t_6	BOAZ	AL	2	0.88	1.76

Fig. 5. Illustration of reliability score computation.

data rule r_1. The reliability score of each γ is derived, as shown in Fig. 5. The data piece γ_2 being {CT: BOAZ, ST: AL} has higher reliability score than γ_1 being {CT: BOAZ, ST: AK}. Thus, γ_2 is regarded as the correct one in this group, and γ_1 is finally replaced with γ_2.

In addition, for ease of understanding, Fig. 6 illustrates three clean data versions after executing this cleaning step. In particular, the group G_{21} from B_2 is skipped to calculate the reliability score, because this group has reached the ideal state with only one γ in it. Finally, the first clean data version contains {CT: DOTHAN, ST: AL} in G_{11} (w.r.t. t_1, t_2, and t_3) and {CT: BOAZ, ST: AL} in G_{13} (w.r.t. t_4, t_5, and t_6). The second clean data version incorporates {PN: 3347938701, ST: AL} in G_{21} (w.r.t. t_1 and t_2) and {PN: 2567688400, ST: AL} in G_{23} (w.r.t. t_3, t_4, t_5, and t_6). The third clean data version consists of {HN: ELIZA, CT: BOAZ, PN: 2567688400} in G_{32} (w.r.t. t_3, t_4, t_5, and t_6).

6.2 Deriving the Unified Clean Data

The first cleaning stage of MLNClean has obtained the ruled-based multiple clean data versions. Now, we are ready to enter the second cleaning stage. It aims to execute the conflicts on top of multi-version data, in order to get the final clean data. Note that, this stage provides the second opportunity to clean erroneous data that are not (or incorrectly) repaired in the first cleaning stage as many as possible.

Take the tuple t_3 depicted in Table 1 as an example. After finishing the first cleaning stage, the data piece w.r.t. t_3 in B_1 (w.r.t. the first data version) is {CT: DOTHAN, ST: AL}, while in B_3 (w.r.t. the third data version), the data piece related to t_3 is {HN: ELIZA, CT: BOAZ, PN: 2567688400}. It is obvious that, t_3 has two different values on the attribute named CT (i.e., "DOTHAN" and "BOAZ") from the two data versions. That is to say, there exist conflicts on attributes CT of t_3, and that should be eliminated to get the final clean t_3. Besides, although {CT: DOTHAN, ST: AL} conforms to the rule r_1, there might still exist errors w.r.t t_3 in the case that it is erroneously classified into a group and thereby cannot be correctly repaired in the first cleaning stage.

Consequently, during the process of executing value conflicts on top of multiple data versions, we identify a set of *candidates* to solve conflicts, which refers to all the possible fusion versions for a tuple. Motivated by our previous proposed concept of r-score, in each group, a data piece with the highest r-score is most likely to be clean. According to the definition of r-score (see Equation (7)), r-score is composed of two parts, which are minimality part and possibility part. Note that, in order to comply with the minimality principle, we make a variant on it. Specifically, the minimality in this cleaning stage represents the distance of a candidate data piece from the original data piece. We employ the edit distance in our current implement. They are derived from the edit similarity (Equation 6). The equation of edit distance between two data pieces is given by

$$_{edit}(\gamma, \gamma^*) = \frac{1}{Z}\sum_{k=1}^{M}\frac{ED(\gamma^{A_k}, \gamma^{*A_k})}{max(|\gamma^{A_k}|, |\gamma^{*A_k}|)}, \quad (8)$$

where we denote γ the original data piece and γ^* a candidate data piece for γ. Z is a normalized function, and M represents the number of the attributes in the data piece.

Algorithm 2. F-Score Based Conflict Resolution (FSCR)

Input: a set of tuples containing conflicts $\mathbb{T}$, a set of blocks $B = \{B_1, \ldots, B_{|B|}\}$
Output: the set of tuples that are eliminated conflicts $\mathbb{T}^c$

$T^c \longleftarrow \varnothing$;
foreach tuple $t \in \mathbb{T}$ **do**
  $\mathbb{V}(t) \longleftarrow \{\gamma_t^1, \ldots, \gamma_t^{|B|}\}$;
  $f_{max} \longleftarrow 0$; $t_{f_{max}} \longleftarrow t$;
  foreach γ_t^j from $\mathbb{V}(t)$ **do**
    $f \longleftarrow Pr(\gamma_t^j)$; $\mathbb{V}(t) \longleftarrow \mathbb{V}(t) - \{\gamma_t^j\}$;
    $\langle t_{f_{max}}, f_{max}\rangle \longleftarrow$ GetFusionT$(B, f, f_{max}, \gamma_t^j)$;
  replace t with $t_{f_{max}}$;
  add $t_{f_{max}}$ to $\mathbb{T}^c$;
return T^c
Function GetFusionT$(B, f, f_{max}, \gamma_t^j)$;
foreach $B_j \subseteq B$ **do**
  foreach $\gamma^j \in B_j$ **do**
    if there exists $\gamma^j \in B_j$ with the highest r-score(γ^j) such that there is no conflict between γ_t^j and γ^j **then**
      $\gamma_t^j \longleftarrow \gamma^j$;
    else
      $f \longleftarrow 0$; **return** $\langle t_{f_{max}}, f_{max}\rangle$;
    $\gamma_t^j \longleftarrow \gamma_t^j \cup \gamma_t^j$; $f \longleftarrow f \times Pr(\gamma_t^j)$;
    GetFusionT$(B, f, f_{max}, \gamma_t^j)$;
return $\langle t_{f_{max}}, f_{max}\rangle$

Hence, we introduce a novel concept of *fusion score* (i.e., f-score) to get the most likely correct values for the tuple containing conflicts. Specifically, the *fusion score* of a tuple t, denoted by f-score(t), is defined as the product of r-score of data pieces $\gamma_t^1, \ldots, \gamma_t^{|B|}$ (related to t) from different data versions, as written in Equation 9.

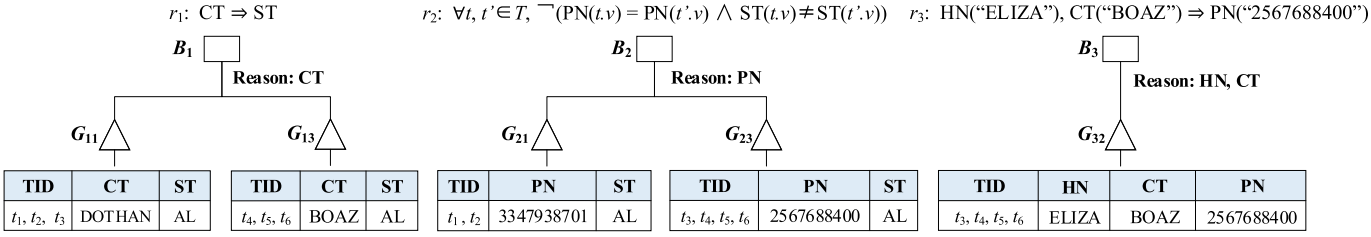


Fig. 6. The three clean data versions over the sample dataset.

$$\text{f-score}(t) = \text{r-score}(\gamma_t^1) \times \cdots \times \text{r-score}(\gamma_t^{|B|}). \quad (9)$$

It is easy to know that as long as we ensure that each data piece γ_t^i is optimal within its corresponding data version, then the global f-score obtained by multiplying each data piece is also the optimal one. Thus, the larger the f-score, the more likely clean the fusion version of t.

We develop an f-score based conflict resolution (FSCR for short) method. Algorithm 2 shows its pseudo-code. It receives a set of tuples containing conflicts, denoted as $\mathbb{T}$, and a set of blocks $B = \{B_1, \ldots, B_{|B|}\}$. First of all, FSCR initializes the tuples that are eliminated conflicts (denoted by T^c) as an empty set (line 1). Then, for each tuple $t \in \mathbb{T}$, it attempts to eliminate the conflicts (lines 3-9). FSCR first puts all the data pieces $\gamma_t^1, \ldots, \gamma_t^{|B|}$ into a set $\mathbb{V}(t)$ (line 3), i.e., $\mathbb{V}(t)$ collects all the data versions of t. Next, a temporal variable f_{max} is initialized as zero, which is used to store the maximal f-score, and the corresponding fusion version of t, i.e., $t_{f_{max}}$, is initialized as t (line 4). Then, for each data piece γ_t^i of t, FSCR merges it with other data pieces from every block $B_j \subseteq B$, where $B_j \neq B_i$, using the function GetFusionT, in order to find the optimal fusion version with the highest f-score (lines 5-7). Thereafter, t is updated using the derived optimal fusion version $t_{f_{max}}$, and is added to T^c (lines 8-9). FSCR proceeds to process the remaining tuples in $\mathbb{T}$ one by one. Finally, it returns the clean dataset T^c (line 10). Hence, GetFusionT is a recursive function (lines 11-20). Given a γ_t^i, for each data piece γ from $B_j \subseteq B$ ($B_j \neq B_i$), GetFusionT needs to find the one that does not conflict with γ_t^i accompanied by the highest $Pr(\gamma)$ (lines 12-15). It terminates if f-score is zero, or one fusion version of tuple t has been obtained (i.e., $\mathbb{V}(t)$ becomes empty) (lines 16-20).

Example 3. We illustrate how FSCR works in terms of tuple t_3 in the sample dataset, based on three data versions (denoted by $\gamma_{t_3}^1, \gamma_{t_3}^2$, and $\gamma_{t_3}^3$) of tuple t_3 (obtained from the first cleaning stage). In particular, $\gamma_{t_3}^1$ denotes {CT: DOTHAN, ST: AL} from block B_1, $\gamma_{t_3}^2$ is with respect to {PN: 2567688400, ST: AL} from B_2, and $\gamma_{t_3}^3$ represents {HN: ELIZA, CT: BOAZ, PN: 2567688400} from block B_3. Hence, following Algorithm 2, for the tuple t_3, FSCR gets $\mathbb{V}(t_3) = \{\gamma_{t_3}^1, \gamma_{t_3}^2, \gamma_{t_3}^3\}$. For simplicity, we show two fusion attempts: (i) merging data pieces based on $\gamma_{t_3}^1$, and (ii) merging data pieces based on $\gamma_{t_3}^3$. For the first attempt, $\gamma_{t_3}^1$ are merged with a γ^2 ={PN: 2567688400, ST: AL} $\in B_2$, which has the highest r-score. Then, it proceeds to merge with a γ from block B_3. Here, GetFusionT tries to find a γ^3 that has the same value on the common attribute CT for merging it with $\gamma_{t_3}^1$ and γ^2. Unfortunately, there does not exist such a γ^3 in block B_3. Thus, GetFusionT terminates the current chance with a zero f-score. For the second attempt, $\gamma_{t_3}^3$ and γ^1 ={CT: BOAZ, ST: AL} are first merged. Then, GetFusionT merges a data piece γ^2 ={PN: 2567688400, ST: AL} $\in B_2$ with $\gamma_{t_3}^3$ and γ^1. Finally, the attribute values of t_3 are {HN: ELIZA, CT: BOAZ, ST: AL, PN: 2567688400}.

Let $|\mathbb{T}|$ be the number of tuples containing conflicts in the dirty dataset, $\overline{b}$ be the average number of data pieces within each block, and $|B|$ be the number of blocks/rules, there are at most $|B|$ possible fusion versions for a tuple. In each block B_i, we need $O(\overline{b})$ time to find the most probable γ^i. Therefore, FSCR takes $O(|\mathbb{T}| \times |B| \times \overline{b})$ time.

7 Experiments

In this section, we present a comprehensive experimental evaluation. In what follows, we evaluate our proposed data cleaning framework MLNClean in the following scenarios: (i) the experimental comparisons between MLNClean and the state-of-the-art method HoloClean [15] in both real-world and synthetic cleaning scenarios, and (ii) the effect of various parameters on the performance of MLNClean in the synthetic cleaning scenarios.

7.1 Experimental Setup

Datasets. In the experiments, we use three real-world datasets: (i) *Hospital*[1] provides a ground truth information about healthcare associated infections occurred in hospitals and it contains 231,265 tuples. It also provides a real-world dataset[2] which contains erroneous values (i.e., typos) and it contains 1,000 tuples. The tuples have dense relationships in this dataset. In other words, different tuples share the same attribute values. We use it to evaluate how effective MLNClean is at cleaning tuples with dense relationships. (ii) *Car*[3] contains the used vehicle information, including *model*, *make*, *type*, *year*, *condition*, *wheelDrive*, *doors*, and *engine* attributes. It consists of 30,760 tuples. In this dataset, the attribute values in the same attribute domain have high similarity. We utilize it to evaluate the performance of MLNClean is at selecting the correct one from a bunch of similar attribute values during cleaning. (iii) *Flight*[4] contains data on the departure and arrival time of flights from different data sources. Both ground truth information and a real-world dataset which includes erroneous values (i.e., missing values) are available, and they all have 2,377 tuples. In this dataset, tuples have sparse relationships. Only the same flight from multiple sources shares the same values, and the values are almost different between different flights. We employ it to evaluate MLNClean against datasets where values have sparse relationships. Table 4 summarizes data quality rules of each dataset used in our experiments, and they are given by domain experts.

Competing Method. To verify the performance of our proposed MLNClean, we compare it with the state-of-the-art method HoloClean.[5] It first separates erroneous values that violate integrity constraints from other clean values, and then, it inputs the clean values into a probabilistic model for learning the parameters of the model. Thereafter, HoloClean uses the learned model to infer the probability of the candidate repairs for each erroneous value.

Experimental Scenarios. In real-world cleaning scenarios, we use two datasets (i.e., Hospital and Flight), both of which have full ground truth information and dirty datasets with real-world errors. In addition, since the Car dataset only provides ground truth information, it is not suitable for real-world scenarios. In synthetic cleaning scenarios, we

1. The ground truth information of Hospital dataset is available at https://data.medicare.gov/data/hosp-italcompare
2. The real-world Hospital dataset is available at https://github.com/HoloClean/holo-clean/tree/master/testdata/hospital.csv
3. Car dataset is available at https://www.cars.com/
4. Flight dataset is available at https://github.com/HoloClean/holo-clean/tree/master/testdata/flight.csv
5. We download publicly available source codes to reproduce results on all datasets.

TABLE 4
Rules Used in Each Dataset

Dataset	Rules
Hospital	ZIPCode ⇒ CountyName PhoneNumber ⇒ ZIPCode PhoneNumber ⇒ State ZIPCode ⇒ City City ⇒ State ZIPCode ⇒ City ProviderID ⇒ City ProviderID ⇒ State
Car	Make, Type ⇒ Doors Model, Type ⇒ Make
Flight	Flight ⇒ ScheduledDept Flight ⇒ ActualDept Flight, ScheduledDept ⇒ DeptGate Flight ⇒ ScheduledArrival Flight, ActualArrival ⇒ ArrivalGate

add erroneous values randomly, including *unknown erroneous values* and *known erroneous values*, on attributes related to the given integrity constraints shown in Table 4 for each dataset (including Hospital, Car, and Flight). The total number of errors injected is controlled by the parameter *error percentage*, which is defined as the ratio of the total number of erroneous values to the total number of values. For generating unknown erroneous values, we construct typos by randomly delete any letter of an attribute value. For creating each known erroneous value, we replace a value with another value from the same attribute domain randomly. Without loss of generality, we use the terms of "unknown erroneous values/typos" and "known erroneous values/ replacement errors" interchangeably in the following experimental evaluation. The number of typos and replacement errors are controlled by the parameter *error type ratio* (denoted as R_{ret}), which is defined as the ratio of the number of known erroneous values to the total number of errors (including known and unknown erroneous values). In particular, $R_{ret} = 0$ means that all errors are unknown erroneous values. $R_{ret} = 100\%$ indicates that all errors are known erroneous values.

Default Settings. Unless explicitly specified MLNClean uses the following parameters by default. First, we set the error percentage to 10 percent. This is because the erroneous values usually represent a small fraction of the entire dataset in real-world datasets. Second, we set the error type ratio to 0 since we focus on cleaning typos. Third, we use the corresponding data in the ground truth to simulate the crowd for answering questions in CBG.

Evidence. In MLNClean, an evidence file is needed, and it contains two parts. The first part consists of all known values of each dataset, and MLNClean allows a repair candidate to choose any value from all known values in this part. We collect all values in the ground truth information as the known values to form the first part. The second part can be treated as an external dictionary for each data piece. The dictionary can be simulated by using the data pieces in each dataset's corresponding ground truth information. Unless explicitly specified, MLNClean does not make use of the external dictionary.

TABLE 5
Experimental Results in Real-World Cleaning Scenario

Hospital				
Cleaning method	Precision	Recall	F1-score	Runtime
MLNClean	**1**	**0.833**	**0.909**	**44.08 sec**
HoloClean	1	0.713	0.832	148.53 sec
Flight				
Cleaning method	Precision	Recall	F1-score	Runtime
MLNClean	**0.995**	**0.993**	**0.994**	31996.57 sec
HoloClean	0.887	0.669	0.763	**72.15 sec**

Evaluation Methodology. We utilize F1-score to evaluate the accuracy of data cleaning methods. It is defined as F1-score=$\frac{2\times precision\times recall}{precision+recall}$, where *precision* is equal to the ratio of correctly repaired attribute values to the total number of updated attribute values, and *recall* equals to the ratio of correctly repaired attribute values to the total number of erroneous values.

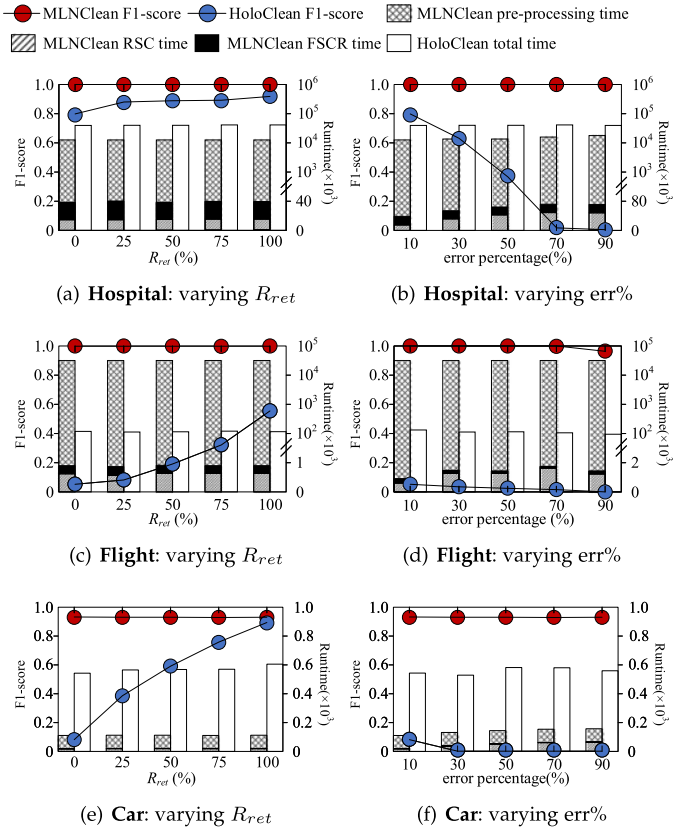
Implementation Details. All the experiments were conducted on a Dell PowerEdge R430 with one Intel(R) Xeon (R) E5-2620 v3 2.40GHz processors (2 physical cores and 24 CPU threads) and 125GB RAM.

7.2 Comparisons With HoloClean

In this section, we verify the performance of MLNClean and HoloClean in real-world and synthetic cleaning scenarios.

7.2.1 Real-World Cleaning Scenarios

We report the F1-score and the end-to-end runtime obtained by MLNClean and its competitor HoloClean. The results are shown in Table 5. The first observation is that MLNClean outperforms HoloClean with relative F1-score improvements of around 10 percent in the Hospital dataset, and relative F1-score improvements of more than 20 percent in the Flight dataset. This verifies that our proposed framework can provide more accurate cleaning results for real-world datasets. For HoloClean, its cleaning accuracy is limited by the error detection algorithm. It can only fix errors caught by the error detection method while ignoring false-positive values that are errors but wrongly regarded as correct values. Since HoloClean uses the correct values detected by the error detection method to train the parameters of the probabilistic model, the existence of false-positive values leads to Simpson's paradox [23] and makes the probabilistic model unreliable for the subsequent repairing phase. On the contrary, MLNClean actually employs correct values from evidence to generate the possible instantiated rules, and it can support reliable error detection and repairing results. The second observation is that MLNClean runs more than three times faster than HoloClean in the Hospital dataset, but runs much slower than HoloClean in the Flight dataset. It is due to the characteristics of the datasets (see Section 7.1). In the Hospital dataset, multiple tuples contain the same values, and hence, the attribute domain is not large. It is not time-consuming for MLNClean to generate all possible instantiated rules from their corresponding attribute domains and infer appropriate probabilities for them. In contrast, in the Flight dataset, different flight information



(a) **Hospital**: varying R_{ret} (b) **Hospital**: varying err%

(c) **Flight**: varying R_{ret} (d) **Flight**: varying err%

(e) **Car**: varying R_{ret} (f) **Car**: varying err%

Fig. 7. Performance of MLNClean and HoloClean under synthetic cleaning scenarios.

does not share the same values. Although the total amount of data in this dataset is small, the attribute domain is vast. Therefore, MLNClean needs much time to generate and assign appropriate probabilities for all possible data pieces in the Flight dataset.

7.2.2 Synthetic Cleaning Scenarios

Effect of Error Type Ratio. We investigate the impact of different error types on the performance of MLNClean and HoloClean by varying the error type ratio R_{ret} from 0 to 100 percent, and report the F1-score and runtime. The corresponding experimental results are plotted in Figs. 7a, 7c, and 7e.

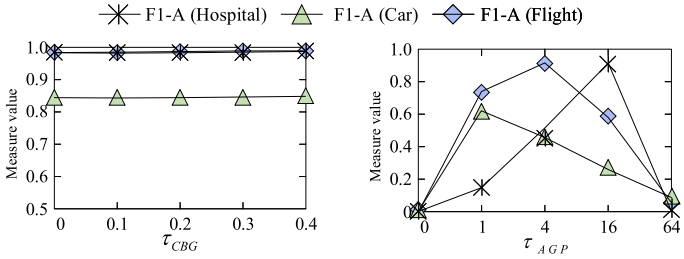
We first focus on evaluating results of the accuracy for both MLNClean and HoloClean. The first observation is that, the F1-score of HoloClean decreases when R_{ret} increases, while the F1-score of MLNClean performs stably on all datasets. As for HoloClean, we use the following two cases to explain the reasons of the poor cleaning results for unknown erroneous values. First, we assume that HoloClean's error detection approach can completely distinguish erroneous values and correct values. In other words, there is no false-positive value. In this case, no typo is treated as evidence for learning the parameters of the probability model. Hence, typos are unknown for HoloClean, and the inference for typos is unreliable without enough evidence. Furthermore, HoloClean's data repairing algorithm allows a cell to obtain any value (including the typo) from its corresponding attribute domain. It is likely to choose a typo as a correct value for repairing an erroneous cell. Second, error detection usually cannot reach 100 percent accuracy, which means that some typos are regarded as the correct values (i.e., there exist false positive values), the mixture of erroneous and correct values leads to Simpson's paradox (as described earlier in Section 7.2.1). Therefore, HoloClean cannot infer reliable repair results for typos. As for MLNClean, two reasons contribute to its resilience to the error percentage. The first reason is that, our RSC strategy provides the ability to correctly identify both replacement errors and typos, and assigns the most appropriate correct values for each error. The second reason is that our FSCR method can re-clean the missed errors after the RSC strategy executes. It further confirms the superiority of MLNClean. We will analyse the performance of RSC strategy and FSCR method in detail in Section 7.3.

We now turn our attention to the runtime evaluation. We depict the runtime of each stage of MLNClean in Figs. 7a, 7c, and 7e, including the execution of pre-processing time, RSC strategy time, and FSCR method time. We can observe that the entire runtime of MLNClean is shorter than that of HoloClean. The reason is that we use data pieces which contain multiple values of cell as the minimum unit for inference, whereas HoloClean needs to infer the probability for each cell. One exception is Flight dataset, the reason is explained in Section 7.2.1. Besides, it is observed that the pre-processing stage takes up most of the time in MLNClean while the other stages execute quickly. This is because, the pre-processing needs time to learn the appropriate possibility of each data piece.

Effect of Error Percentage. We study the performance of MLNClean and HoloClean by varying the error percentage (denoted as err%) from 10 to 90 percent, and report the F1-score and runtime results in Figs. 7b, 7d, and 7f.

We first focus on analyzing results of the accuracy for both MLNClean and HoloClean. Our proposed MLNClean is observed to maintain a stable performance when the error percentage grows. It attributes to the well-trained Markov logic network (MLN), which provides the ability to distinguish erroneous values from the correct values. Specifically, MLN assigns a higher probability for the correct data piece and a lower probability for the erroneous data piece. It is only related to the provided evidence, not affected by the amount of errors. However, the F1-score of HoloClean drops in all datasets as the error percentage grows. This is because, when there are more errors, there is relatively fewer correct values. It means that there is less evidence available for training. Insufficient evidence leads to inaccurate inference for the probability of each error, resulting in a decline in the accuracy of data cleaning results.

We now turn our attention to the runtime evaluation. First, it is observed that the execution time of RSC strategy slightly increases. The reason is that, the execution time of RSC is positively correlated with the number of errors when the error percentage grows, and RSC needs to perform cleaning for more errors. Second, we find that there is no obvious correlation between the runtime of FSCR method and the error percentage. This is because only conflicting tuples generated after executing the RSC strategy need to be re-cleaned by FSCR method, and it is irrelated to the error percentage. Third, as expected, training time, which only related to the number of evidence, takes up most of the pre-processing runtime. Thus, the change of



(a) *CBG* (b) *AGP*

Fig. 8. The performance of CBG and AGP versus thresholds.

error percentage has no apparent effect on the pre-processing runtime.

7.3 In-Depth Investigations of MLNClean

In this section, for in-depth investigations of the performance of MLNClean, we explore the effect of different parameters on the three phases of MLNClean, including AGP and CBG strategies for generating groups of MLN index, RSC method for cleaning within each group, and FSCR strategy for deriving the unified clean data, each of which has an impact on the data cleaning accuracy of MLNClean. In particular, in order to appropriately measure the accuracy of each component, we introduce a series of metrics for them. For AGP and CBG, we define *F1-A* as $\frac{2\times Precision-A\times Recall-A}{Precision-A+Recall-A}$, where *Precision-A* as the fraction of correctly generated groups over the total generated groups, and *Recall-A* as the fraction of correctly generated groups on the total number of real groups. For RSC strategy, we define *F1-RSC* as $\frac{2\times Precision-R\times Recall-R}{Precision-R+Recall-R}$, where *Precision-R* represents the ratio of correctly repaired γs to the total number of repaired γs, and *Recall-R* is equal to the ratio of correctly repaired γs to the number of γs which contain errors. For FSCR method, we define *accuracy* that corresponds to the fraction of correctly repaired tuples by FSCR over the number of tuples that include the detected conflicts.

7.3.1 Comparison of CBG and AGP

We first compare CBG and AGP strategies for generating groups in the MLN index. (1) CBG uses a threshold τ_{CBG} to prune data pieces that do not need to be executed so as to boost efficiency; and (2) AGP uses a threshold τ_{AGP} to identify abnormalities. Fig. 8a shows the effect of τ_{CBG} on the performance of CBG. Fig. 8b depicts the effect of τ_{AGP} on the performance of AGP.

In Fig. 8a, we observe that the accuracy of CBG is not sensitive when varying τ_{CBG}. This is because τ_{CBG} only affects the number of questions generated in the CBG strategy. The smaller the τ_{CBG}, the greater the number of questions. However, the accuracy of CBG depends on the accuracy of answers, and the answers of the generated questions are given by the crowd, regardless of the threshold.

In Fig. 8b, the observation is that, the accuracy of AGP first ascends and then drops as the value of τ_{AGP} grows over datasets. When getting the optimal value of τ_{AGP}, the accuracy is the highest. In contrast, when the value of τ_{AGP} exceeds the optimal value, the accuracy deteriorates. This is because, more normal groups are detected as abnormal groups. In short, it is not flexible to get a proper threshold τ_{AGP}, and thus restricts the advantage of the AGP strategy. Instead, CBG is insensitive to the threshold τ_{CBG} and is therefore more versatile for different datasets. Also, the results show that CBG is superior to AGP in accuracy.

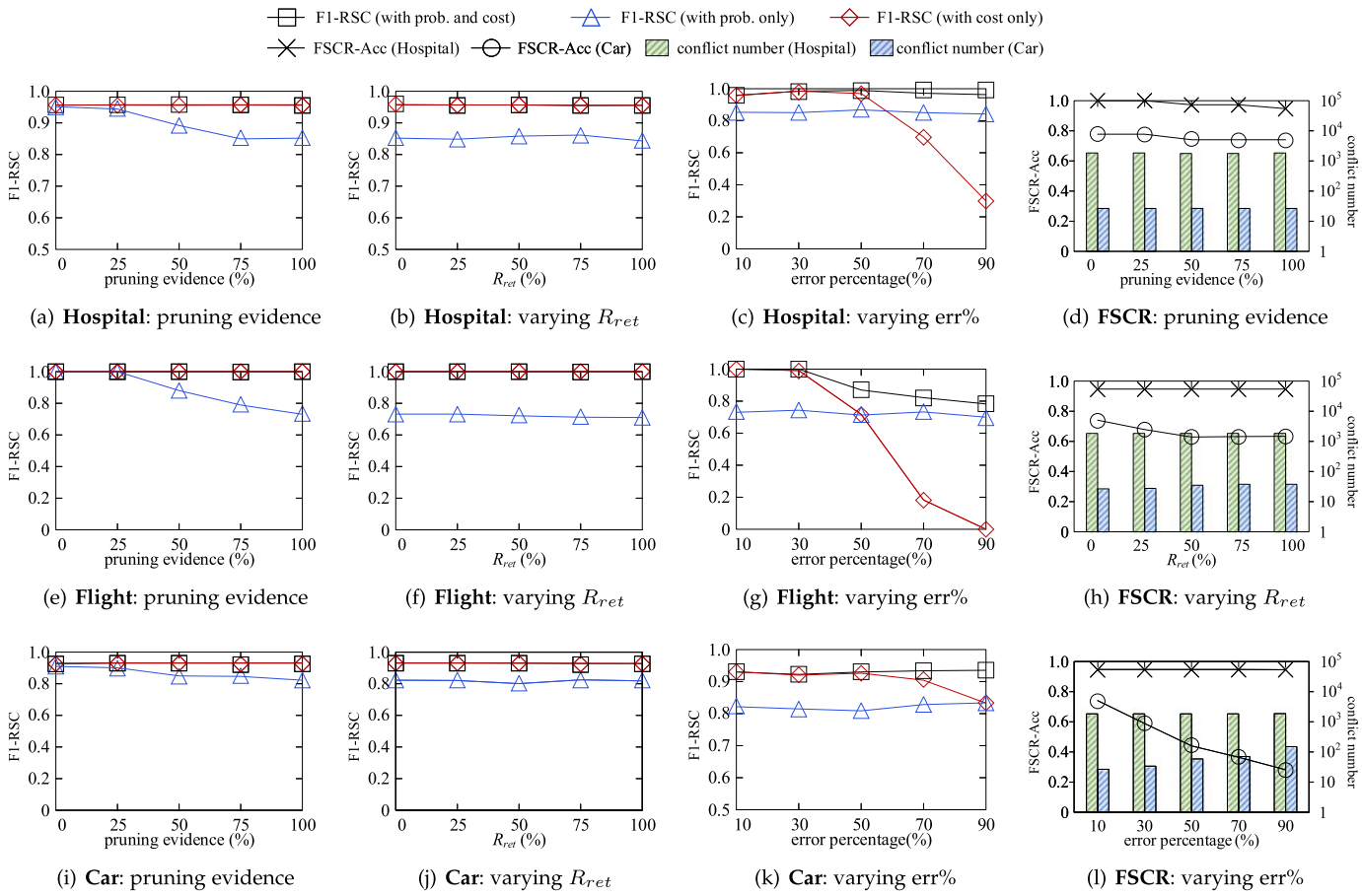
7.3.2 Results on RSC Strategy

Second, we investigate the performance of RSC strategy, which detects and repairs errors violated rules within each data version based on the concept of *r-score*. We also perform an ablation study to evaluate the effect of the factors in the r-score, which are the possibility of a data piece being correct (*prob.* for short) and the minimality cost of repairing a γ (*cost* for short).

Effect of Pruning Evidence. We study the impact of evidence on the accuracy of RSC. We define a new variable *pruning rate* as the proportion of external data pieces contained in the evidence. To be more specific, the pruning rate being 100 percent means no external data piece is referred for cleaning. The pruning rate being 0 indicates that the evidence contains sufficient external information, and the mismatched data piece (between the original dirty dataset and the evidence) contains erroneous values in this case. We vary the rate of pruning evidence from 0 to 100 percent, and report the F1-RSC in Figs. 9a, 9e, and 9i. First, as expected, RSC has high-quality cleaning results when all correct data pieces can be correlated with evidence. Second, it is observed that the accuracy of the RSC strategy (only using *prob.* for r-score) slightly decreases when more evidence is pruned. This is because the loss of evidence may cause some data pieces to fail in obtaining reliable inference results. We want to highlight that even if there is no external data piece in the evidence, the cleaning accuracy of RSC is still above 70 percent. This proves the robustness of the MLN inference for data pieces. Third, we find that executing RSC strategy only with *cost* for r-score performs stably when the pruning rate changes. This is because, the evaluations are carried out on the premise that the total error percentage is 10 percent. There are more correct data pieces than erroneous data pieces, so it is easy to distinguish the correctness of the data pieces based on the number.

Effect of Error Type Ratio. We evaluate the accuracy of the RSC strategy by varying the error type ratio R_{ret} from 0 to 100 percent, and report the F1-RSC in Figs. 9b, 9f, and 9j. First, it is observed that only using *prob.* for r-score maintains a stable performance when facing different types of errors. The reason is that, the well-trained MLN is able to precisely predict the probability of each data piece. In other words, it can assign a high probability to the correct data piece and a low probability to the erroneous data piece, and the erroneous data piece can be correctly treated as errors. Second, as expected, the performance of RSC strategy only with *cost* for r-score is not related to the changes in error type ratio. Therefore, using both *prob.* and *cost* for r-score enables the RSC strategy to obtain stable and high-quality results at different error type ratios.

Effect of Error Percentage. We study the accuracy of the RSC strategy by varying the error percentage (*err%* for short) from 0 to 100 percent, and report the F1-RSC in Figs. 9c, 9g, and 9k. First, only using the *prob.* based r-score for RSC strategy is observed to maintain a stable performance when the error percentage changes. Its resilience to the error percentage is mainly contributed by the well-trained MLN, which



(a) **Hospital**: pruning evidence (b) **Hospital**: varying R_{ret} (c) **Hospital**: varying err% (d) **FSCR**: pruning evidence

(e) **Flight**: pruning evidence (f) **Flight**: varying R_{ret} (g) **Flight**: varying err% (h) **FSCR**: varying R_{ret}

(i) **Car**: pruning evidence (j) **Car**: varying R_{ret} (k) **Car**: varying err% (l) **FSCR**: varying err%

Fig. 9. Performance of MLNClean versus different parameters.

provides the ability to distinguish the erroneous data pieces from the correct ones via the inferred probabilities. Second, we find that the accuracy of RSC strategy only using the cost-based r-score decreases when the error percentage enlarges. Especially when the number of errors exceeds the correct ones, the decline becomes obvious. The reason is that the cost-based r-score may wrongly regard the erroneous data pieces as correct. Despite this, using both *prob.* and *cost* for r-score still enables the RSC strategy to obtain high-quality results at different error percentages.

7.3.3 Results on FSCR Method

Last but not the least, we evaluate the performance of the FSCR method by varying the pruning rate, error percentage, and error type ratio, respectively. We report the accuracy of the FSCR method and the number of tuples containing conflicts in every case. Note that, we do not evaluate the performance of the FSCR method on the Flight dataset. This is because no conflict is detected on the Flight dataset after executing the RSC strategy.

Effect of Pruning Evidence. First, we explore the impact of error percentage on the performance of the FSCR method by varying the pruning rate from 0 to 100 percent. The corresponding results are depicted in Fig. 9d. We can observe that the accuracy has no significant fluctuation with the change of pruning rate. It attributes to the robustness of the MLN inference, as described in the experiment of varying the pruning rate on RSC strategy.

Effect of Error Type Ratio. Then, we study the performance of the FSCR method by varying the error type ratio from 0 to 100 percent. The results are shown in Fig. 9h. We find that the accuracy is not sensitive to the error type ratio on the hospital dataset. Nevertheless, the accuracy drops from 75 to 40 percent on the Car dataset. This is because, the dense relationships among values in the Hospital dataset ensure that the candidate with the highest f-score is the correct repairing result for most tuples containing conflicts. On the contrary, on the Car dataset, values in the same attribute domain have high similarity, and we empirically find that the high similarities among values make the f-score between different candidates close. When the error type ratio increases, the similarity between the values within the same attribute domain is higher as well. Therefore, given a tuple containing conflicts, it is likely to happen that the repair candidate with the highest f-score has the same values with another tuple in the data set, but it is not the correct repairing result for this tuple.

Effect of Error Percentage. Finally, we change the error percentage to evaluate the performance of the FSCR method. The results are shown in Fig. 9l. As expected, the accuracy of FSCR performs stably on both datasets. It also attributes to the ability that the MLN inference is able to identify errors and correct data, and the ability is not related to the error percentage. Again, the high accuracy of FSCR reflects that, FSCR is capable of cleaning out the errors which have not been correctly cleaned by the RSC strategy.

8 Conclusion

In this paper, we propose MLNClean, a novel hybrid data cleaning framework. It overcomes the drawback of cleaning accuracy caused by the lack of sufficient integrity constraints in the existing ICs-based approaches. MLNClean is capable of (i) learning instantiated rules to supplement the insufficient integrity constraints on top of Markov logic networks and (ii) dealing with both unknown and known erroneous values. It receives a dirty dataset together with a set of ICs and some evidence. It outputs clean data through two steps, including *pre-processing* and *two-stage data cleaning*. In the pre-processing step, MLNClean first infers a set of probable instantiated rules according to MLNs and then builds a two-layer MLN index structure to generate multiple data versions. Since a well-generated MLN index helps to guarantee the cleaning accuracy, we propose an effective crowd-based grouping strategy (CBG) to ensure the correctness of the MLN index. In the two-stage data cleaning step, MLNClean first presents a concept of *reliability score* to clean errors within each data version separately, and afterward eliminates the conflict values among different data version using a novel concept of *fusion score*. Considerable experimental results on both real and synthetic scenarios demonstrate the effectiveness of MLNClean in practice.

Acknowledgments

This work was supported in part by the National Key R&D Program of China under Grant No. 2018YFB1004003, the NSFC under Grants No. 61972338 and 61902343, and the NSFC Zhejiang Joint Fund under Grant No. U1609217.

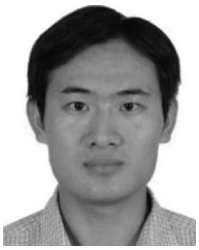
References

[1] W. W. Eckerson, "Data warehousing special report: Data quality and the bottom line," *Appl. Develop. Trends*, vol. 1, no. 1, pp. 1–9, 2002.
[2] X. Chu, I. F. Ilyas, S. Krishnan, and J. Wang, "Data cleaning: Overview and emerging challenges," in *Proc. Int. Conf. Manage. Data*, 2016, pp. 2201–2206.
[3] L. Bertossi, L. Bravo, E. Franconi, and A. Lopatenko, "Complexity and approximation of fixing numerical attributes in databases under integrity constraints," in *Proc. Int. Workshop Database Program. Languages*, 2005, pp. 262–278.
[4] P. Bohannon, W. Fan, F. Geerts, X. Jia, and A. Kementsietsidis, "Conditional functional dependencies for data cleaning," in *Proc. IEEE 23rd Int. Conf. Data Eng.*, 2007, pp. 746–755.
[5] P. Bohannon, M. Flaster, W. Fan, and R. Rastogi, "A cost-based model and effective heuristic for repairing constraints by value modification," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2005, pp. 143–154.
[6] X. Chu, I. F. Ilyas, and P. Papotti, "Holistic data cleaning: Putting violations into context," in *Proc. IEEE 29th Int. Conf. Data Eng.*, 2013, pp. 458–469.
[7] M. Dallachiesa *et al.*, "NADEEF: A commodity data cleaning system," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2013, pp. 541–552.
[8] W. Fan, F. Geerts, X. Jia, and A. Kementsietsidis, "Conditional functional dependencies for capturing data inconsistencies," *ACM Trans. Database Syst.*, vol. 33, no. 2, pp. 6:1–6:48, 2008.
[9] F. Geerts, G. Mecca, P. Papotti, and D. Santoro, "The LLUNATIC data-cleaning framework," *Proc. VLDB Endowment*, vol. 6, no. 9, pp. 625–636, 2013.
[10] Z. Khayyat *et al.*, "BigDansing: A system for big data cleansing," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2015, pp. 1215–1230.
[11] X. Chu *et al.*, "KATARA: A data cleaning system powered by knowledge bases and crowdsourcing," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2015, pp. 1247–1261.
[12] W. Fan, J. Li, S. Ma, N. Tang, and W. Yu, "Towards certain fixes with editing rules and master data," *VLDB J.*, vol. 21, no. 2, pp. 213–238, 2012.
[13] S. Hao, N. Tang, G. Li, J. Li, and J. Feng, "Distilling relations using knowledge bases," *VLDB J.*, vol. 27, no. 4, pp. 497–519, 2018.
[14] N. Prokoshyna, J. Szlichta, F. Chiang, R. J. Miller, and D. Srivastava, "Combining quantitative and logical data cleaning," *Proc. VLDB Endowment*, vol. 9, no. 4, pp. 300–311, 2015.
[15] T. Rekatsinas, X. Chu, I. F. Ilyas, and C. Ré, "HoloClean: Holistic data repairs with probabilistic inference," *Proc. VLDB Endowment*, vol. 10, no. 11, pp. 1190–1201, 2017.
[16] P. M. Domingos and D. Lowd, *Markov Logic: An Interface Layer for Artificial Intelligence*, San Rafael, CA, USA: Morgan & Claypool, 2009.
[17] G. Beskales, I. F. Ilyas, and L. Golab, "Sampling the repairs of functional dependency violations under hard constraints," *Proc. VLDB Endowment*, vol. 3, no. 1, pp. 197–207, 2010.
[18] G. Cong, W. Fan, F. Geerts, X. Jia, and S. Ma, "Improving data quality: Consistency and accuracy," in *Proc. 33rd Int. Conf. Very Large Data Bases*, 2007, pp. 315–326.
[19] W. Fan, X. Jia, J. Li, and S. Ma, "Reasoning about record matching rules," *Proc. VLDB Endowment*, vol. 2, no. 1, pp. 407–418, 2009.
[20] L. E. Bertossi, S. Kolahi, and L. V. S. Lakshmanan, "Data cleaning and query answering with matching dependencies and matching functions," in *Proc. 14th Int. Conf. Database Theory*, 2011, pp. 268–279.
[21] C. Mayfield, J. Neville, and S. Prabhakar, "ERACER: A database approach for statistical inference and data cleaning," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2010, pp. 75–86.
[22] M. Yakout, L. Berti-Équille, and A. K. Elmagarmid, "Don't be scared: Use scalable automatic repairing with maximal likelihood and bounded changes," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2013, pp. 553–564.
[23] S. Krishnan, J. Wang, E. Wu, M. J. Franklin, and K. Goldberg, "Activeclean: Interactive data cleaning for statistical modeling," *Proc. VLDB Endowment*, vol. 9, no. 12, pp. 948–959, 2016.
[24] A. Heidari, J. McGrath, I. F. Ilyas, and T. Rekatsinas, "Holodetect: Few-shot learning for error detection," in *Proc. Int. Conf. Manage. Data*, 2019, pp. 829–846.
[25] J. Wang, T. Kraska, M. J. Franklin, and J. Feng, "Crowder: Crowdsourcing entity resolution," *Proc. VLDB Endowment*, vol. 5, no. 11, pp. 1483–1494, 2012.
[26] C. Chai, G. Li, J. Li, D. Deng, and J. Feng, "A partial-order-based framework for cost-effective crowdsourced entity resolution," *VLDB J.*, vol. 27, no. 6, pp. 745–770, 2018.
[27] X. Chu, I. F. Ilyas, and P. Papotti, "Discovering denial constraints," *Proc. VLDB Endowment*, vol. 6, no. 13, pp. 1498–1509, 2013.

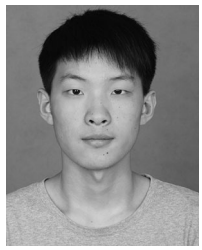
Congcong Ge received the BS degree in computer science from the Zhejiang University of Technology, China, in 2017. She is currently working toward the PhD degree in the College of Computer Science, Zhejiang University, China. Her research interests include data cleaning and data integration.

Yunjun Gao (Member, IEEE) received the PhD degree in computer science from Zhejiang University, China, in 2008. He is currently a professor with the College of Computer Science, Zhejiang University, China. His research interests include spatial and spatio-temporal databases, metric and incomplete/uncertain data management, graph databases, spatio-textual data processing, and database usability. He is a Member of the ACM.

Xiaoye Miao received the PhD degree in computer science from Zhejiang University, China, in 2017. She is currently a ZJU young assistant professor with the Center for Data Science, Zhejiang University, China. Her research interests include uncertain/incomplete databases, graph management, data cleaning, and data pricing.



Bin Yao (Member, IEEE) received the PhD degree in computer science from the Department of Computer Science, Florida State University, in 2011. He is an associate professor with the Department of Computer Science and Engineering, Shanghai Jiao Tong University. His research interests include management and indexing of large databases, query processing in spatial and multimedia databases, string and keyword search, and scalable data analytics.



Haobo Wang received the BS degree in computer science and technology from Zhejiang University, China, in 2018. He is currently working toward the PhD degree in the College of Computer Science and Technology, Zhejiang University. His research interests include machine learning and data mining, especially on multi-label learning and weakly-supervised learning.