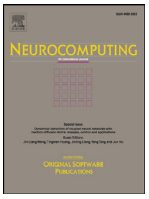
Neurocomputing 253 (2017) 201–214

Contents lists available at ScienceDirect

Neurocomputing

journal homepage: www.elsevier.com/locate/neucom



Web-ADARE: A web-aided data repairing system

Binbin Gu[a], Zhixu Li[a,*], Qiang Yang[a], Qing Xie[b], An Liu[a], Guanfeng Liu[a], Kai Zheng[a], Xiangliang Zhang[c]

[a] School of Computer Science and Technology, Soochow University, China
[b] School of Computer Science and Technology, Wuhan University of Technology, China
[c] King Abdullah University of Science and Technology, Jeddah, Saudi Arabia

A R T I C L E I N F O

Article history:
Received 8 June 2016
Revised 16 August 2016
Accepted 1 September 2016
Available online 8 March 2017

Keywords:
Data repairing
Rule-based method
Web-aided method
Multiple sources

A B S T R A C T

Data repairing aims at discovering and correcting erroneous data in databases. In this paper, we develop Web-ADARE, an end-to-end web-aided data repairing system, to provide a feasible way to involve the vast data sources on the Web in data repairing. Our main attention in developing Web-ADARE is paid on the interaction problem between web-aided repairing and rule-based repairing, in order to minimize the Web consultation cost while reaching predefined quality requirements. The same interaction problem also exists in crowd-based methods but this is not yet formally defined and addressed. We first prove in theory that the optimal interaction scheme is not feasible to be achieved, and then propose an algorithm to identify a scheme for efficient interaction by investigating the inconsistencies and the dependencies between values in the repairing process. Extensive experiments on three data collections demonstrate the high repairing precision and recall of Web-ADARE, and the efficiency of the generated interaction scheme over several baseline ones.

1. Introduction

As the data explode for decades, the quality of the data in various information systems decreases sharply. It has been estimated that erroneous data could cost US businesses 600 billion dollars each year [1]. With this come the need for data cleaning tools and systems, which aim at discovering and correcting erroneous data in databases. According to a Gartner's report [2], the market for data cleaning is growing at 17% annually.

Various data cleaning approaches have been developed so far. Most of the existing methods [3–5] rely on a variety of quality rules including ETL rules, FDs/CFDs, MDs, INCs and some customized rules to detect violations and conflicts in the data, and then use some arbitrary heuristics to select modifications that, for example, would introduce minimal changes to the data. However, the rules usually fall short to correctly identify the right fixes [6].

To have more reliable modifications, some recent efforts tend to involve external knowledge in the repairing process. For instance, Fan et. al. [7] assume an oracle reference database which can always provide reliable modification data by doing record matching between the target database and this reference database. However, this oracle reference data (absolutely correct, full coverage) is not likely to be available in general. Although there are some open structured knowledge bases like Freebase or DBPedia that contain plenty of general knowledge for reference, they might be not that helpful in repairing all kinds of incomplete datasets.

More recent efforts tend to involve crowdsourcing in data repairing. For instance, Yakout et. al. [6] use user's feedback to repair a database and to adaptively refine the training set for a repairing model. The NADEEF system [8] allows the users to specify data quality rules and how to repair erroneous data through writing code that implements predefined classes. In our published conference paper [9], we develop CrowdAidRepair, a novel Crowd-Aided Data Re- pairing approach, which performs crowd-based and rule-based repairing alternatively for achieving a high repairing quality at the minimum crowd cost. On the one hand, crowdsourcing can significantly improve the quality of the data, but on the other hand, although some efforts are made, it still requires high labor cost for data repairing. As reported in [6], every 20 records using 1 expert's feedback is the minimum requirement for reaching an acceptable quality. Thus, the cost would be unaffordable when the database is large.

Why not use a much cheaper and within reach knowledge depository, i.e., the Web, in data repairing? Compared to crowdsourcing, the Web has at least four advantages: first, it is almost free. Usually there is no need to pay for getting information from the

* Corresponding author.
E-mail addresses: gu.binbin@hotmail.com (B. Gu), zhixuli@suda.edu.cn (Z. Li), qiangyanghm@hotmail.com (Q. Yang), felixxq@whut.edu.cn (Q. Xie), anliu@suda.edu.cn (A. Liu), gfliu@suda.edu.cn (G. Liu), kaizheng@suda.edu.cn (K. Zheng), xiangliang.zhang@kaust.edu.sa (X. Zhang).

http://dx.doi.org/10.1016/j.neucom.2016.09.132
0925-2312/© 2017 Elsevier B.V. All rights reserved.

surface Web. Second, consulting the Web through web search engines has no restrictions on time since the Web does not sleep (but human experts do) and can be accessed anytime. Third, as the world's largest repository of human's knowledge, the Web naturally has comprehensive common knowledge with a huge quantity of raw data, either structured (such as Wikipedia) or unstructured, to support the repairing to a wide range of data in databases of different domains. Fourth, consulting the Web can be performed in parallel which can further improve working efficiency.

The four advantages above motivate us to develop a Web-Aided DAta REpairing (or *Web − ADARE* for short) system. We state that our work is orthogonal to the existing crowd-based repairing, and we consider it as a future work to merge the two by doing Web-aided repairing firstly and crowd-based repairing secondly. However, to develop an end-to-end Web-aided data repairing system, Web-ADARE faces at least the following challenges.

Fetching update values from the Web. When local repairing methods (such as rule-based repairing method) can not solve some detected conflict between errors with high confidence, Web-ADARE needs to consult the Web as long as the value involved in the conflict is "searchable", i.e., its update value is available somewhere that can be accessed by web search engines on the Web. Although fetching particular data automatically from the Web has been studied and applied successfully in set expansion [10–14], it is a new and challenging problem to fetch a particular attribute value at a certain position of a local database from the Web. Compared to set expansion which targets at getting a set of instances of the same kind from the Web, we need to accomplish many different web knowledge mining tasks in order to repair a database, each of which requires us to get a particular attribute value from the Web.

Controlling the quality. The Web is not clean at all, thus we may take the risk to bring web noises into the local database. Existing work on web data extraction estimates the quality of the extraction according to various factors [11,13], such as the employed patterns and the confidence of various web sources. But in our case, the quality of each formulated repairing query also needs to be taken into account, which is decided by all segments that consists the query such as the the employed relevant data quality rule as well as the leveraged attribute values from the local database.

Minimizing web consultation times. Although consulting the Web is cheap and fast compared with using crowdsourcing, repairing a large database with plenty of detected errors still requires issuing many web retrieving queries and processing times more retrieved web documents. Ideally, we hope to minimize the number of values for web-aided repairing while maintaining a high repairing quality. According to our observations, a possible way to achieve this is to alter between web-aided repairing and rule-based repairing. That is, each time we have some detected conflicts be resolved by the Web, then more left conflicts will become resolvable to FD/CFD rules under predefined quality constraints. Thus, the problem transforms into a scheduling problem for selecting values in detected conflicts for web-aided repairing and rule-based repairing alternatively under predefined quality constraints. This scheduling problem is nontrivial: primarily, as an optimal scheduling scheme, it only does web-aided repairing to those "persistent" conflicts that can never be resolvable to rule-based repairing. However, we do not know a priori which conflicts are not "persistent" ones until they are dismissed later without repairing, and we neither know which conflicts can never be resolved by rule-based repairing until all the other conflicts are resolved. Furthermore, the whole interaction issue is considered in a dynamic setting. As more and more conflicts are resolved, the rule-based repairing result to every unresolved conflict keeps on changing, and the set of unresolved conflicts is also changing as some new conflicts will be generated while some old ones will be solved/dismissed.

Our contributions in this paper are summarized below:

(1) We present Web-ADARE, the first end-to-end Web-aided data repairing system, which is supposed to provide high-quality repairing to searchable data in a wide range of databases.
(2) We propose novel ways relying on web search engines and information extraction tools to fetch update values for repairing detected erroneous values in databases.
(3) We provide a proper way to estimate the quality of the web-aided repairing results, and also unify the confidence estimation schemes for rule-based repairing and web-aided repairing.
(4) We identify the interaction problem that schedules the repeated alternation of web-aided and rule-based repairing for reaching a balance between the repairing quality and the web consultation cost. After proving in theory that the optimal interaction scheme is unlikely to be identified, we propose algorithms to generate the most efficient interaction schemes we could achieve.

We conduct extensive experiments to verify the scalability and effectiveness of our system on several data collections.

Roadmap. We give the system architecture followed with a running example in Section 2. We introduce how we fetch update values from the Web in Section 3, and how we measure the quality of the update value in Section 4. We address the interaction problem in Section 5. The experiments are reported in Section 6, and related work is covered in Section 7. We conclude in Section 8.

2. Architecture

We present the architecture of Web-ADARE, followed by an example to demonstrate the workflow.

2.1. Architecture overview

Fig. 1 depicts the architecture of Web-ADARE. It contains four components: (1) the *Conflict Detector* detects conflicts (or violations) between data according to the given quality rules; (2) the *Web Data Fetcher* fetches the update values from the Web; (3) the *Quality Supervisor* estimates the quality of the updates either from the Web or from the rule-based repairing; and (4) the *Core* component assigns different erroneous values in detected conflicts for either rule-based repairing or web-aided repairing, and controls the interaction between web-aided and rule-based repairing.

Conflict detector: Web-ADARE adopts the rule-based conflicts detection techniques [3–5,15] to use data quality rules in the form of database constraints, i.e., FDs and CFDS, to identify tuples with errors and inconsistencies. Note that when an error is detected for violating a quality rule, it can be fixed immediately according to the rule. For instance, given a quality rule "[Zip='4072'] → [City='Brisbane'] ", assume there is a tuple where "Zip='4072', City='Sydney' ", the "Sydney" in the tuple is apparently an error which will be updated to "Brisbane". However, the inconsistency between data, which we call as *Conflict*, is difficult to be fixed. More formally, a conflict can be defined as follows:

Definition 1. We say a *Conflict* happens between two sets of values $\{V_1\}$ and $\{V_2\}$, if the two sets of values cannot be both correct. Besides, a value involved in conflict is called a *Suspicious Value*.

For instance, given a quality rule `Inst`(Institution) → `City`, assume that two `City` values "Brisbane" and "Sydney" correspond to the same `Inst` "UQ" in separate tuples, a conflict between ("Brisbane", "Sydney") can be detected, where at least one of the two

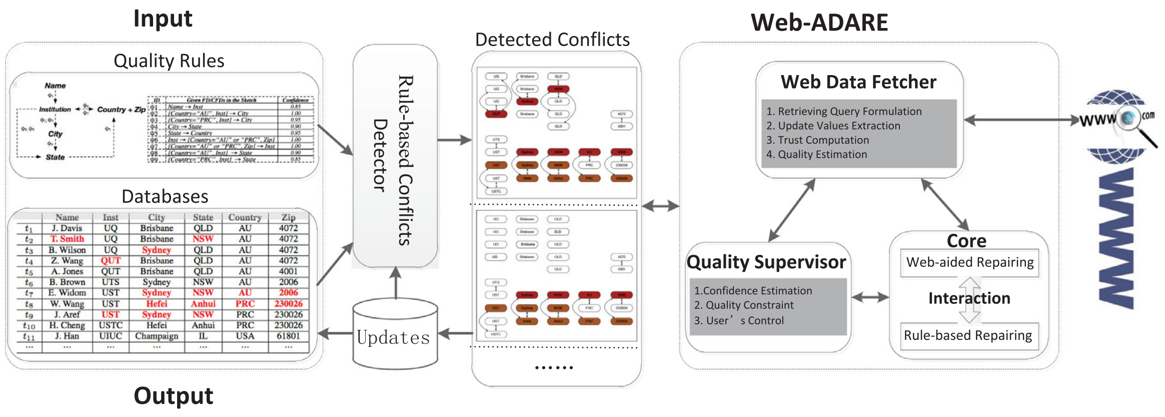
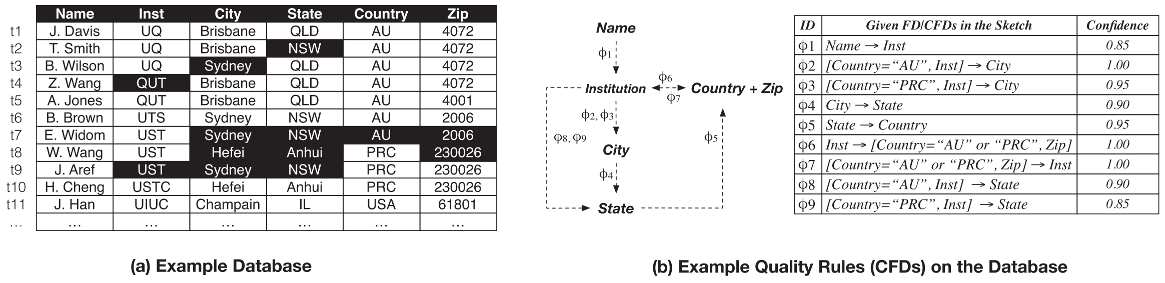


Fig. 1. Architecture of Web-ADARE.



	Name	Inst	City	State	Country	Zip
t1	J. Davis	UQ	Brisbane	QLD	AU	4072
t2	T. Smith	UQ	Brisbane	NSW	AU	4072
t3	B. Wilson	UQ	Sydney	QLD	AU	4072
t4	Z. Wang	QUT	Brisbane	QLD	AU	4072
t5	A. Jones	QUT	Brisbane	QLD	AU	4001
t6	B. Brown	UTS	Sydney	NSW	AU	2006
t7	E. Widom	UST	Sydney	NSW	AU	2006
t8	W. Wang	UST	Hefei	Anhui	PRC	230026
t9	J. Aref	UST	Sydney	NSW	PRC	230026
t10	H. Cheng	USTC	Hefei	Anhui	PRC	230026
t11	J. Han	UIUC	Champain	IL	USA	61801
...	...	...	...	...	...	...

(a) Example Database

ID	Given FD/CFDs in the Sketch	Confidence
φ1	Name → Inst	0.85
φ2	[Country="AU", Inst] → City	1.00
φ3	[Country="PRC", Inst] → City	0.95
φ4	City → State	0.90
φ5	State → Country	0.95
φ6	Inst → [Country="AU" or "PRC", Zip]	1.00
φ7	[Country="AU" or "PRC", Zip] → Inst	1.00
φ8	[Country="AU", Inst] → State	0.90
φ9	[Country="PRC", Inst] → State	0.85

(b) Example Quality Rules (CFDs) on the Database

Fig. 2. Example database and quality rules.

values must be incorrect. Thus, both the two values are suspicious values.

Web Data Fetcher: This component leverages traditional *Information Extraction* (IE) methods together with the capabilities of Web search engines towards the goal of getting the update values for detected suspicious values in certain tuples from the Web. The details on how this component works will be given in Section 3.

Quality Supervisor: This module relies on a quality measuring scheme to maintain the quality of the update values from both web-aided repairing and rule-based repairing to guarantee a high repairing accuracy. Integrating user's feedback to improve the scheme will be one of our future work. We will cover this quality measuring scheme in Section 4.

Core: The core module contains *rule-based repairing, web-aided repairing* and *interaction scheduler*.

(1) *Web-aided repairing:* This module does web-aided repairing to solve a conflict between attribute values w.r.t. a given quality rule. For an attribute value, the web-aided repairing module chooses proper attribute values and quality rules as keywords, and then forwards them to the Web Data Fetcher module to get update values from the Web. The details of this part will be given in Section 3.1.

(2) *Rule-based repairing:* This module adopts existing rule-based methods [3–5,15] to repair an assigned (possibly) erroneous value in the database. But to prevent from hurting the repairing accuracy, we constrain that a rule-based repairing operation will be performed to a deducible value, if and only if all the attribute values supporting this rule-based repairing operation are (verified or assumed as) correct values. In particular, we take values that are already repaired or verified, and those not involved in any conflicts as correct ones.

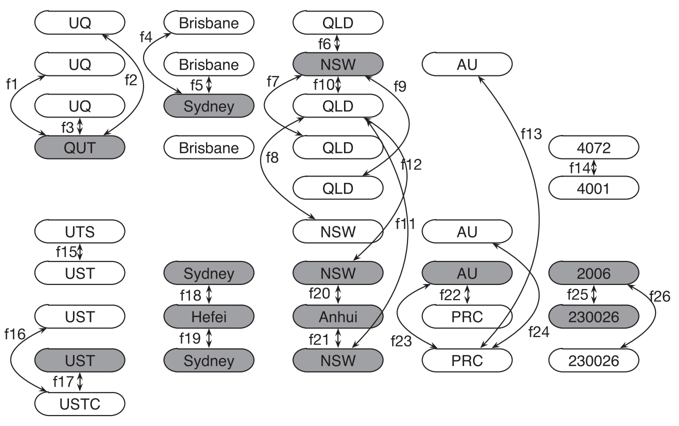
(3) *Interaction scheduler:* The scheduler module is responsible for assigning conflicts and their covered values for web-aided or rule-based repairing. The objective is to look for an optimal interaction scheme to minimize the number of web-aided repairing operations issued for resolving the detected conflicts. Note that there is no fixed number of values for repairing. An interaction scheme is a qualified one as long as it resolves all the detectable conflicts in the data set. This is the key component of Web-ADARE, which will be discussed in Section 5.

2.2. A demonstration example

Consider a contact database in Fig. 2(a), where each tuple contains the `Name` and `Inst` of a person, in addition to one's address information: `City`, `State`, `Country` and `Zip`. Whilst most values are correct, there are 13 erroneous ones in black boxes waiting to be unveiled and updated. Based on the natural dependencies between attributes, users may input a set of quality rules in the form of CFDs as listed in Fig. 2(b).

According to these rules, we will detect 26 pairs of conflicted values as depicted in Fig. 3(a), involving 34 values (taking up > 50% of the values) in the table. We list some example conflicts in Fig. 3(b) to illustrate how they are detected below:

(1) Rule ϕ_7 detects the conflict f_1 between "UQ" at position t_3[`Inst`] and "QUT" at position t_4[`Inst`], given that the two Institutions correspond to the same `Country + Zip` value "AU + 4072". Note that the rule only reports the conflict but does not repair it.
(2) Rule ϕ_2 detects the conflict f_5 between "Brisbane" at position t_2[`City`] and "Sydney" at position t_3[`City`], given that the two cities correspond to the same `Inst` value "UQ".
(3) Rule ϕ_4 detects the conflict f_9 between "NSW" at position t_2[`State`] and "QLD" at position t_5[`State`], given that the two states correspond to the same `City` value "Brisbane".
(4) Rule ϕ_6 detects the conflict f_{14} between "4072" at position t_4[`Zip`] and "4001" at position t_5[`State`], given that the two states correspond to the same `City` value "QUT".
(5) Rule ϕ_5 detects the conflict f_{23} between "AU" at position t_7[`Cou– ntry`] and "PRC" at position t_9[`Country`], given that



(a) Detected Conflicts from the Example in a Graph

ID	Conflict Pairs	Referred Values	Referred CFD
f1	(UQ, QUT)	AU + 4072	$\phi 7$
f5	(Brisbane, Sydney)	UQ	$\phi 2$
f9	(NSW, QLD)	Brisbane	$\phi 4$
f14	(4072, 4001)	QUT	$\phi 6$
f23	(AU, PRC)	NSW	$\phi 5$
…	…	…	…

(b) Some Example Conflicts for illustration

Fig. 3. Detected conflicts from the example.

the two countries correspond to the same `State` value "NSW".

With all the conflicts detected, we now come to the repairing step. To demonstrate the advantage of Web-ADARE, we compare the repairing results of pure rule-based repairing, and web-aided repairing in Fig. 4. As can be observed in Fig. 4(a), by doing pure rule-based repairing, we can only repair 6 erroneous values with the left 8 values un-repaired. For instance, we can not expect the value "Hefei" at t_7[`City`] be modified correctly into "Kowloon", given that there is no knowledge in the table telling that "UST" may locate at "Kowloon, HK".

In the following, we give a possible interactive way to do data repairing with both rule-based repairing and web-aided repairing, and the results are depicted in Fig. 4(b):

(1) We do web-aided repairing to:
- t_4[`Inst`] to modify "QUT" into "UQ";

(2) We do rule-based repairing to:
- t_3[`City`] to modify "Sydney" into "Brisbane";
- t_2[`State`] to modify "NSW" into "QLD";

(3) We do web-aided repairing to:
- t_7[`City`] to modify "Sydney" into "Kowloon";
- t_7[`State`] to modify "NSW" into "HK";
- t_7[`Country`] to modify "AU" into "PRC";
- t_7[`Zip`] to modify "2006" into "999077";

(4) We do rule-based repairing to:
- t_8[`City`] to modify "Hefei" into "Kowloon";
- t_8[`State`] to modify "Anhui" into "HK";
- t_8[`Zip`] to modify "230026" into "999077";

(5) We do web-aided repairing to:
- t_9[`Inst`] to modify "UST" into "USTC";

(6) We do rule-based repairing to:
- t_9[`City`] to modify "Sydney" into "Hefei";
- t_9[`State`] to modify "NSW" into "Anhui";

As can be observed in Fig. 4(b), we have all erroneous values correctly updated. In this way, we only have 6 values for web-aided repairing, which uses 50% less web-aided operations than the pure web-aided repairing method (at least 13 web-aided operations). But, can we further reduce the repairing cost (the number of web query operations) by scheduling the alternation of web-aided and rule-based repairing with consideration of the incompatible degree of values and the dependency between conflicts? We will discuss the question in Section 5.

3. Fetching updates from the Web

The web-aided repairing method leverages traditional *Information Extraction* (IE) methods together with the capabilities of Web search engines towards the goal of getting update values from the Web. Towards this, Web-ADARE introduces what we call a *web-aided data repairing query*, which is basically a web search query specially formulated for the purpose of data repairing. Such data repairing query is formally defined as follows:

Definition 2. For a relational tuple t, a *data repairing query* $q(X \to y)$ is one that is formulated to utilize the values of a certain set of attributes $X = \{x_1, x_2, \ldots\}$ to retrieve the update value of a certain attribute y from the web.

For the correctness of the update data, we do not use suspicious values in formulating data repairing queries. Basically, the web-aided repairing method needs to address the following critical questions:

(1) Given a pair $< X, y >$, how to *formulate* $q(X \to y)$ to effectively retrieve the update for a suspicious value y?
(2) How to *evaluate* the correctness of the extracted values, such that we can select the right answer from multiple candidates.

We will introduce how to deal with the two questions in the following two sections.

3.1. Queries formulation

Besides a set of attribute values X as the keywords, some *Auxiliary Information* is needed to express the relationship between X

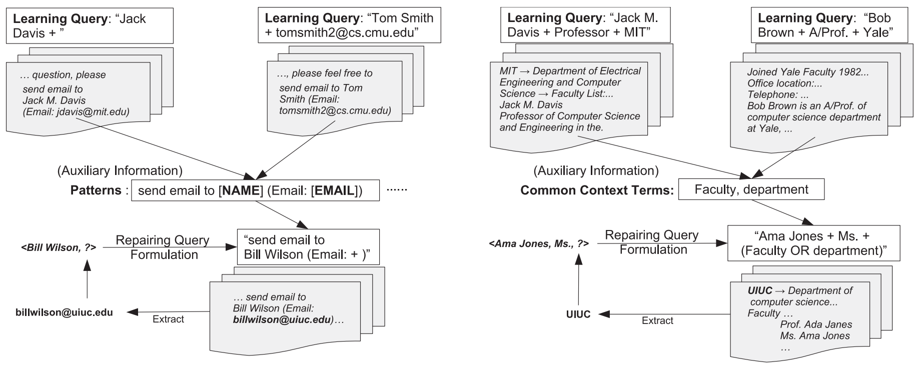
	Name	Inst	City	State	Country	Zip
t1	J. Davis	UQ	Brisbane	QLD	AU	4072
t2	T. Smith	UQ	Brisbane	QLD	AU	4072
t3	B. Wilson	UQ	Brisbane	QLD	AU	4072
t4	Z. Wang	UQ	Brisbane	QLD	AU	4072
t5	A. Jones	QUT	Brisbane	QLD	AU	4001
t6	B. Brown	UTS	Sydney	NSW	AU	2006
t7	E. Widom	UST	Hefei	Anhui	PRC	230026
t8	W. Wang	UST	Hefei	Anhui	PRC	230026
t9	J. Aref	UST	Hefei	Anhui	PRC	230026
t10	H. Cheng	UST	Hefei	Anhui	PRC	230026
t11	J. Han	UIUC	Champain	IL	USA	61801
…	…	…	…	…	…	…

(a) The Results of Rule-based Repairing

	Name	Inst	City	State	Country	Zip
t1	J. Davis	UQ	Brisbane	QLD	AU	4072
t2	T. Smith	UQ	Brisbane	QLD	AU	4072
t3	B. Wilson	UQ	Brisbane	QLD	AU	4072
t4	Z. Wang	UQ	Brisbane	QLD	AU	4072
t5	A. Jones	QUT	Brisbane	QLD	AU	4001
t6	B. Brown	UTS	Sydney	NSW	AU	2006
t7	E. Widom	UST	Kowloon	HK	PRC	999077
t8	W. Wang	UST	Kowloon	HK	PRC	999077
t9	J. Aref	USTC	Hefei	Anhui	PRC	230026
t10	H. Cheng	USTC	Hefei	Anhui	PRC	230026
t11	J. Han	UIUC	Champain	IL	USA	61801
…	…	…	…	…	…	…

(b) The Results of Web-aided Repairing

Fig. 4. The repairing results on the example database (values in Gray are those modified into correct values).


Learning Query: "Jack Davis + "
… question, please send email to Jack M. Davis (Email: jdavis@mit.edu)
Learning Query: "Tom Smith + tomsmith2@cs.cmu.edu"
…, please feel free to send email to Tom Smith (Email: tomsmith2@cs.cmu.edu)
(Auxiliary Information)
Patterns : send email to [NAME] (Email: [EMAIL]) ……
<Bill Wilson, ?>
Repairing Query Formulation
"send email to Bill Wilson (Email: +)"
… send email to Bill Wilson (Email: billwilson@uiuc.edu)…
Extract
billwilson@uiuc.edu
Learning Query: "Jack M. Davis + Professor + MIT"
MIT → Department of Electrical Engineering and Computer Science → Faculty List:... Jack M. Davis Professor of Computer Science and Engineering in the.
Learning Query: "Bob Brown + A/Prof. + Yale"
Joined Yale Faculty 1982... Office location:... Telephone: ... Bob Brown is an A/Prof. of computer science department at Yale, ...
(Auxiliary Information)
Common Context Terms: Faculty, department
<Ama Jones, Ms., ?>
Repairing Query Formulation
"Ama Jones + Ms. + (Faculty OR department)"
UIUC → Department of computer science... Faculty … Prof. Ada Janes Ms. Ama Jones …
Extract
UIUC

(a) Pattern-based Query Formulation and Extraction (b) Context Term-based Query Formulation and Extraction

Fig. 5. Example learning and retrieving process of the two web-aided repairing ways.

and y in the query, otherwise the query may not have those web documents containing the update value of y as top-ranked ones. With a better formulated query, we only need to process a set of top-ranked documents instead of the whole set of retrieved documents. In this way, on the one hand, a lot of processing time can be saved, and on the other hand, much fewer noises will be introduced into the results.

To learn the auxiliary information for a repairing query $q(X \rightarrow y)$, Web-ADARE retrieves Web documents that contain some of the data in those complete tuples and extracts the *auxiliary information* from those documents to use in future data repairing queries. Given that approach, our previous definition of data repairing query could be refined as: $q(X, A \rightarrow y)$ where A denotes the auxiliary information. The particular nature of this auxiliary information depends on the adopted IE method. Take two particular IE methods for instance: the Pattern-based IE method [13] and the co-occurrence based IE method [16].

(1) Patterns. The pattern based query formulation way extends the classical pattern based IE method [13,17], which relies on syntactic patterns to identify instances of a given entity type. Applying pattern-based query formulation in Web-ADARE involves learning and using auxiliary information in the form of patterns, which is accomplished via the following three tasks: (1) identifying all possible attribute pairs as $< X_i, y_j >$ based on the unsuspicious attribute values and suspicious values per tuple, (2) learning pattern $A_{i,j}$ for each possible pair $< X_i, y_j >$ in the set of tuples with no suspicious values, and (3) applying those learned pattern to formulate data repairing queries for tuples with suspicious values. For example in Fig. 5(a), to learn auxiliary information (i.e., patterns) based on instances without suspicious values such as:

("Jack M. Davis", "jdavis @mit.edu") and

("Tom Smith", "tomsmith2 @cs.cmu.edu") ,

we issue a *Learning Query* based on each one of those complete tuples. A learning query is a Web search query that returns a set of documents that are further utilized for pattern extraction. In particular, from the retrieved documents, we may learn patterns corresponding to $<$ {*Name*}, *Email* $>$ such as:

Pattern: "send email to `[NAME]` (Email: `[EMAIL]`)"

(as shown in Fig. 5(a)). Finally, we can easily formulate a data repairing query for each tuple with an update `Email` value using the values of `Name` and the extracted pattern. For example,

Query q_1: "send email to Bill wilson (Email:" + ")",

where the string in a quotation will be taken as an unseparated keyword, and "+" a blank space by the web search engine.

From a number of top-ranked documents retrieved by a formulated repairing query, we then work on extracting possible update values for the target suspicious value. For the pattern based query formulation way, we can easily extract the update value from the text with the used pattern. For instance in Fig. 5(a), since "billwilson@uiuc.edu" appears between the two keywords "send email to Bill wilson (Email:" and ")" in the documents retrieved by the query q_1, it will be naturally taken as the update value for the suspicious value under `Email` in the given tuple.

(2) Context terms. Given that patterns are sometimes too *strict* to capture many entities or relations on the web, we also adopt co-occurrence based IE method in the query formulation. In particular, a co-occurrence based IE method learns *common context terms* instead of patterns from seed instances of a given relation [16]. The formulated imputation query only requires the retrieved documents to contain the common context terms at any position. For example in Fig. 5(b), from instances such as (Jack M. Davis, Professor, MIT) and (Bob Brown, A/Prof., Yale), we could learn some common context terms for the relation ({Name, Title}, Institution) such as "Faculty", "department", which are mentioned closely and frequently with these instance pairs in some web pages. With these context terms as auxiliary information, we can formulate a query for the instance (Ama Jones, Ms., ?) as:

Query q_2: "Ama Jones + Ms. + (Faculty OR department)",

where the "OR" will be taken as an operation symbol by the web search engine.

It is nontrivial to extract update values for the co-occurrence based way, since the position of the update value can not be determined without a pattern. As an alternative, we rely on *Named Entity Recognition(NER)* [18] techniques to detect possible update values for the suspicious one. Specifically, for a given type of update values, the NER method identifies all phrases referring to organizations in the documents. However, the state-of-the-art NER methods can only identify limited types of entities such as "Organization", "Time" or "Location" etc. Hence, we use the list of all correct values under the same attribute in the database as a *dictionary* to aid the NER process, as we did in the dictionary-based entity extraction [19].

3.2. Value selection model

Note that there might be multiple possible update values extracted from the retrieved documents, but we suppose that only one of them is the correct one we look for while the others are all noisy ones. A basic way is to adopt a frequency-based model which takes the one with the highest extraction frequency as the update value. However, the frequency does not exactly reflect the correctness of values, since on the one hand, there can be copies between web pages, and on the other hand, every datum has its freshness. For example, if a professor used to work in a university for 10 years but has recently moved to a new one, then the frequency-based model might suggest that the correct value for his/her address is the old one.

Inspired by Dong et. al. [20], we consider both the precision of web sources and the dependence between sources in deciding the most likely correct value from multiple candidate values. In addition, we also take the freshness of data in various web sources [21] into account. Initially, we estimate the precision of different web sources, as well as the dependencies between them by giving all the values an uniform initial correct probability of values. Next, based on the learned web sources' accuracies and the dependencies between them, we compute a correct probability for every candidate value, based on which we can update the precision of web sources and the dependencies between them. We then keep on updating the correct probabilities of candidate values, the precision of web sources and dependencies between web sources alternatively until the correct probabilities become stable.

(1) Start-up step. Let $S(y_v)$ be the set of web sources that provide candidate update values to the target value y_v, $S(v)$ be the set of web sources supporting v as the correct update value to y_v. Initially, we assume that all sources are independent, then the probability that v is the correct update value to y_v can be calculated with the Bayes analysis as follows:

$$P(v) = Pr(v|\Psi(y_v)) = \frac{Pr(\Psi(y_v)|v)Pr(v)}{Pr(\Psi(y_v))} = \frac{\prod_{s\in S(v)} \frac{(n-1)A(s)}{1-A(s)}}{\sum_{v'\in V(y_v)} \prod_{s\in S(v')} \frac{(n-1)A(s)}{1-A(s)}} \tag{1}$$

where $\Psi(y_v)$ is the observation to which values that each source in $S(v_y)$ votes for, and $V(y_v)$ is the domain of y_v including one correct update value and the other $(n-1)$ incorrect update values to y_v. As the start-up step, we can set the value of all $A(s)$ to a constant value in (0, 1).

(2) Iteration steps. Given the start-up correct probability of every candidate value to every y_v in the target erroneous values set Y_v, we can calculate the precision of every web source s, denoted as $A(s)$, as follows:

$$A(s) = \sum_{y_v\in Y_v} \frac{\sum_{v\in V(s,y_v)} P(v)}{|V(s,y_v)|\cdot|Y_v|} \tag{2}$$

where $V(s, y_v)$ denotes all the candidate update values to y_v supported by the source s.

Based on the updated precision of all sources S, we will calculate/update a temporal correct probability for each value v which considers both the dependencies between sources as well as the freshness of the data, denoted as $Pt(v)$, as follows:

$$Pt(v) = \beta\cdot\sum_{s\in S(y_v)} (ln\frac{(n-1)A(s)}{1-A(s)} Dp(s, S(y_v))) + (1-\beta)\cdot\frac{F(v)-T}{T} \tag{3}$$

where β is a balanced factor, $F(v)$ is milliseconds from January 1, 1970 to the time value extracted from web for value v and T is a constant having a larger order of magnitude such as 10^6. $Dp(s, S(y_v))$ investigates the dependencies between s and all other sources in $S(v_y)$, which can be estimated as:

$$Dp(s, S(y_v)) = \prod_{s'\in S(y_v)} (1-d(s,s')) \tag{4}$$

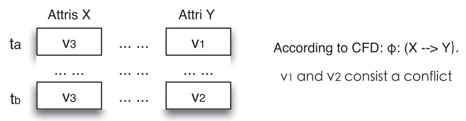


Fig. 6. Example conflict with refered CFD.

where $d(s, s')$ gets the dependency between two sources s and s'.

Based on temporal correct probability of each value v, we can update the correct probability as follows:

$$P(v) = \frac{e^{Pt(v)}}{\sum_{v'\in V(y_v)} e^{Pt(v')}} \tag{5}$$

We do the update iteratively until the correct probabilities of values become stable. Finally, for each v_y, the candidate update value with the highest $P(v)$ will be kept as the only update value from the Web.

4. Quality estimation

Web-ADARE employs a unified quality measuring scheme to estimate the quality of the update values from either rule-based or web-aided repairing. We adopt a rule-based quality measuring scheme to estimate the quality of rule-based update values [4], and then extend it to the web-aided repairing scenario.

(1) Rule based updates: Given that a rule-based update is deduced according to a given CFD and some existing relevant values in the database, the quality of the update is jointly decided by the confidence of the CFD and the quality of all the relevant values. Given that all these confidence/quality values play an equal role in this deduction operation, we just use the product of all these qualities and CFD in estimating the quality of the rule-based update, that is,

$$c(v_c) = c(\phi)\times\prod_{v_i\in V_R} c(v_i), \tag{6}$$

where V_R contains a set of referred values that are used to deduce v_c for the position, $c(v)$ denotes the confidence of a value v, and $c(\phi)$ denotes the confidence of the referred FD/CFD ϕ. For instance in Fig. 6, assume we update v_2 with v_1, the confidence of the update is: $c(\phi)\times c(t_a[X])\times c(t_b[X])\times c(t_a[Y])$.

(2) Web aided updates: Basically, there are three possible factors that may lead to erroneous update values from the Web: (1) *quality of the keywords*: the formulated repairing query may use incorrect values as keywords, (2) *effectiveness of the query:* every repairing query has a probability to get incorrect update values from the Web, and (3) *credibility of the web data sources:* possible noises might be gotten from various sources on the Web.

In the following, we discuss on estimating the influence of each factor to the quality of the web-aided update value, and finally provide a way to estimate the quality of a web-aided update value according to the three factors based on a set of training data:

- *Quality of keywords to query*: Assume a query $\mathbf{q}$ is formulated according to a CFD ϕ, and $KW(\mathbf{q})$ contains all values that are used in formulating the query, then the confidence of the query can be estimated as:

  $$c(KW(\mathbf{q})) = c(\phi)\times\prod_{v_i\in KW(\mathbf{q})} c(v_i), \tag{7}$$

  where $c(\phi)$ denotes the confidence of the used constraint ϕ, and $c(v)$ denotes the confidence of a value v.
- *Effectiveness of the query in web search*: We suppose that every particular query utilizing the same set of attributes X towards the same target attribute y have similar performance in getting relevant web pages containing the correct update values from

the Web. Thus, we can estimate a prior probability on the correctness of the web pages retrieved by queries in the form of $q(X \rightarrow y)$ as:

$$c(\mathbf{q}) = c(KW(\mathbf{q})) \times \frac{|D_+(\mathbf{q})|}{|D_+(\mathbf{q})| + |D_-(\mathbf{q})|}, \tag{8}$$

where $|\cdot|$ gets the number of elements in one set, $D_+(\mathbf{q})$ denotes the set of pages with positive results, while $D_-(\mathbf{q})$ denotes the set of pages with negative results.

- *Credibility of the web data sources*: Our issued queries get data from various data sources (i.e., websites) on the Web through the web search engine. Some of these websites provide us with correct update values, while the others do not. We tend to give websites that always provide positive results a higher credibility score. Thus we can calculate the credibility of the web sources as follows:

$$c(d) = \frac{\sum_{v_c \in V(s)} P(v_c)}{|V(s)|}, \tag{9}$$

where $P(v_c)$ is the probability that v_c is the correct update value, which was already introduced in the previous section.

Finally, we adopt the noisy-all model [13,22] to estimate the confidence of an update value v_c as follows:

$$c(v_c) = c(\mathbf{q}) \times \left(1 - \prod_{d \in D(\mathbf{q})} (1 - c(d))\right) \tag{10}$$

where $D(\mathbf{q})$ denotes the set of retrieved web pages having v_c as the answer, and $c(d)$ denotes the credibility of the web page d, which can be the credibility score of the website that d belongs to.

5. Interaction algorithms

We now study the interaction problem between rule-based repairing and web-aided repairing: At each web-aided repairing step, we select some values for web-aided repairing, and then in the succeeding rule-based repairing step, all deducible values will be repaired. This alternation will be repeated until no more erroneous data can be fixed. We name this as the interaction between rule-based repairing and web-aided repairing.

Ideally, we hope to identify an *optimal interaction scheme* that can resolve all the conflicts at the minimum web consultation cost. Meanwhile, all the update values should satisfy a predefined quality constraint. Formally, we define the *Quality-Constrained Interaction Problem* below:

Definition 3. (*Quality-Constrained Interaction Problem*). Given a relational table T for repairing, a set of predefined FD/CFDs Φ holding on T, a quality measuring scheme $c(\cdot)$ and a quality threshold τ $(0 \le \tau \le 1)$, the objective is to identify an optimal interaction scheme $\mathcal{S}_{op}$ for repairing values in T, which satisfies: (1) resolving all the conflicts in T w.r.t. Φ; (2) $\forall v_c$, $c(v_c) \ge \tau$, where v_c denotes an update value; (3) $\forall \mathcal{S}'$ satisfying the above two conditions, we have $cost(\mathcal{S}_{op}) \le cost(\mathcal{S}')$.

However, the optimal interaction scheme is not feasible to be constructed automatically. We analyze below that even the optimal interaction scheme to a simplified version of this problem is not feasible to be achieved. In this simplified version, we assume that all modifications we issue must be correct and they all satisfy the quality constraint in no doubt. Initially, assume we identify a set of possible erroneous values $\mathbb{V}$ in conflicts, waiting to be checked and repaired. The optimal interaction scheme should involve the least number of values in $\mathbb{V}$ for web-aided checking, that is, only those values that can never be repaired by rule-based repairing will be repaired by the Web. However, two things in dynamic do not allow us to achieve and verify the optimal interaction scheme: (1) The values in $\mathbb{V}$ are dynamic and unpredictable, since each modification may move some values out from $\mathbb{V}$, and add some new values into $\mathbb{V}$. (2) The set of values that can be repaired by rule-based repairing are also dynamic, since each modification may let some new values become repairable to rule-based repairing, and also possible to make some other values become un-repairable to rule-based repairing.

Given the above, the optimal scheduling scheme to the quality-constrained interaction problem is not feasible to be achieved. In the following, we provide ways to generate efficient interaction schemes approaching the optimal one.

5.1. A probabilistic-based heuristic algorithm

The key to generate an efficient interaction scheme lies on how to select values for web-aided repairing at each web-aided repairing step. In this algorithm, we tend to give a higher priority to a value for web-aided repairing if correcting this value can let more other values become deducible. Although this is impossible to know exactly a priori, we develop a heuristic method to estimate a probabilistic score of letting other values become deducible for each value, and then select values for web-aided repairing accordingly.

5.1.1. Incompatible degree

In particular, we estimate an “incompatible degree” between each value and all the other values in the data set, which can be roughly reflected by the number of conflicts it brings to the data set. For short, we call this “incompatible degree” as the *dScore* of the value. In particular, this probabilistic-based heuristic algorithm gives a value (at a position) with a high *dScore* a high priority to be checked with web-aided repairing, since repairing a value with the highest *dScore* will influence the more number of conflicts and their covered values, which may let the most of number of values become deducible.

We introduce how to calculate the *dScore* for each value in a simplified case. To begin with, we assume that the data set is consistent without the value at a position, that is, all the other values in the data set appear to be compatible. Then the value at this position comes, which may bring conflicts in two ways: (1) itself conflicts with some values; (2) it may let some values involved in a conflict. Usually, the more conflicts it brings, the higher probability it is an erroneous value. In other words, the *dScore* of a value can be manifested as the number of conflicts it caused in this simple setting. We now consider the situation in real case, where there are already erroneous values and conflicts in the data set. When a new value at a position comes, either an erroneous one or not, it brings some changes anyway, such as producing new conflicts, or voting for existing conflicts. In this case, the *dScore* of a value can be manifested by two things: (1) the new conflicts produced, and the “credibility”, or what we call the *cScore* of these conflicts, which will be discussed in Eq. (12); (2) the changes on the *cScore* of existing conflicts. Specifically, *dScore*(v) of a value v can be calculated by:

$$dScore(v) = \alpha \times \sum_{f \in F(v)} \Delta(cScore(f)), \tag{11}$$

where α is a normalization factor to scale *dScore*(v) between 0 and 1, $F(v)$ contains all conflicts that are influenced by putting v into the data set, and Δ(*cScore*(f)) is the change on the *cScore* of a conflict f.

In particular, the *cScore* of a conflict f is decided by four relevant values as given in Fig. 6. Previous work considers that a conflict is consisted of two values such as v_1 and v_2 in the Fig. 6, we say that a conflict is also closely related to another two values which are referenced to identify the conflict according to a certain CFD,

Table 1
The repairing interaction scheme generated by Algorithm 1.

$\mathcal{T}_0$	$\emptyset$
$\mathcal{W}_1$	t_4[Zip](“4072”), t_7[Inst](“UST”), t_8[Inst](“UST”), t_8[Country](“PRC”) are correct, not changed; t_2[State](“NSW”), t_3[City](“Sydney”), t_4[Inst](“QUT”), t_9[Inst](“UST”), t_9[State](“NSW”) and t_9[City](“Sydney”) are incorrect, modified;
$\mathcal{T}_1$	t_7[Country](“AU”) is incorrect, modified;
$\mathcal{W}_2$	t_8[Zip](“230026”) is incorrect, modified;
$\mathcal{T}_2$	t_7[Zip](“2006”) is incorrect, modified;
$\mathcal{W}_3$	t_7[State](“NSW”) is incorrect, modified;
$\mathcal{T}_3$	t_8[State](“Anhui”) is incorrect, modified;
$\mathcal{W}_4$	t_8[City](“Hefei”) is incorrect, modified;
$\mathcal{T}_4$	t_7[City](“Sydney”) is incorrect, modified.

such as the two v_3 in the figure. Thus, the correctness of the four values jointly decide the *cScore* of a conflict *f*. Furthermore, when a conflict is voted as a conflict by several groups of values w.r.t. different CFDs, we only pick the one with the highest *cScore* as the final *cScore* of the conflict. More specifically,

$$cScore(f) = arg \max_{\phi \subseteq \Phi(f)} [c(\phi) \times \prod_{v'_i \in V(f,\phi)} (1 - dScore(v'_i))], \tag{12}$$

where $\Phi(f)$ is the set of CFDs that voted *f* as a conflict, and $V(f, \phi)$ contains all values related to the conflict *f* w.r.t. ϕ.

5.1.2. The algorithm and example

According to Eqs. (11) and (12), the *dScore* of a value is decided by the *dScore* of the other values. As described in Algorithm 1, to calculate the *dScores* for all values, we give an initial *dScore* to every value, and then update all the *dScores* iteratively until they all become stable. After that, we rank all values according to their *dScores* in an descendent order, and issue web-aided operations for values one by one. Once a value is verified as incorrect, we update the *dScores* of all the remaining values. We repeat this until all conflicts are resolved.

Algorithm 1: Probabilistic-based heuristic interaction.

```
Input  : A table with a set of conflicts $\mathbb{F}$
Output: A repairing scheme$\mathcal{S} = \langle \mathcal{T}_0, \mathcal{W}_1, \mathcal{T}_1, \ldots, \mathcal{W}_n, \mathcal{T}_n \rangle$
Set $i = 0$;
while $\mathbb{F} \neq \emptyset$ do
    1. $\mathcal{T}_i \leftarrow$All deducible values at the moment;
    2. Deducing all values in $\mathcal{T}_i$;
    3. Updating $\mathbb{F}$;
    4. $i++$;
    5. Calculating dScores for all values in $\mathbb{F}$ with Eq. (11);
    6. while no new deducible values do
        $v \leftarrow$ Value with the highest dScore;
        $\mathcal{W}_i \leftarrow \mathcal{W}_i \cup \{v\}$;
        Checking/Repairing $v$ with the Web;
        if $v$ is updated with an update value then
            Updating $\mathbb{F}$ and dScores;
return $\langle \mathcal{T}_0, \mathcal{W}_1, \mathcal{T}_1, \ldots, \mathcal{W}_n, \mathcal{T}_n \rangle$;
```

Example 1. We apply the algorithm to the running example and the repairing interaction scheme is depicted in Table 1.

1) Initially, we have no value for rule-based repairing.
2) By calculating the dScores for all suspicious values, we take the one with the highest dScore for web-aided repairing and then update the dScores for the remaining suspicious values.
3) After we do web-aided repairing to 8 values (among which 6 values are true errors and updated), we get an inferable value. Thus, we have this value be updated with rule-based repairing as listed in $\mathcal{T}_1$.
4) We again update the dScores for the remaining suspicious values and get more values for web-aided repairing, until another inferable value comes out. We continue with this interaction way until no more values can be updated.

As can be counted, the 13 erroneous values are updated in the correct way. Overall, we do web-aided repairing to 13 values, among which 4 values are not erroneous values. There are also 4 values repaired by rule-based repairing.

5.2. Dependency-aware algorithm

Although all errors are updated correctly in Example 1, several correct values (which do not need to be updated) are assigned for web-aided repairing, and only four values are assigned for rule-based repairing. In this section, we hope to find a more efficient interaction scheme by taking the dependencies between conflicts into account.

Definition 4. We say a conflict f_a depending on another conflict f_b, if some values in f_b are the reasons (or part of the reasons) that aroused the conflict in f_a w.r.t. some FD/CFDs.

For instance in Fig. 3(a), f_5 depends on both f_1 and f_2, since the t_2[Inst](“UQ”) involved in f_1 and the t_3[Inst](“UQ”) involved in f_3 have aroused the conflict in f_5 w.r.t. ϕ_2.

When a conflict f_a depends on another conflict f_b, we normally should process f_b prior to processing f_a for three reasons below:

- Initially, it is possible that after the conflict in f_b is resolved, the conflict in f_a is dismissed automatically without any repairing operations. This happens when f_a is a “fake” (false-positive) conflict such as f_{13} and f_{14} in Fig. 7. As long as we resolve the conflicts they depend on, say f_1, f_2, f_3 for f_{14} and f_{25}, f_{26} for f_{15} in the correct way, then these fake conflicts will disappear immediately.
- To say the least, even if f_a is a true conflict and we need to do web-aided operations to check the values inside it, sometimes we have no other choices but to rely on those values in the conflicts that f_a depends on to formulate web-aided repairing queries.
- Lastly, after we process all conflicts it depends on, we can update the *dScores* for the values in f_a for better judging which value is more likely an error.

Given the intuition above, we present a dependency-aware interaction algorithm, which considers the dependencies between conflicts in scheduling conflicts for repairing.

5.2.1. Conflict dependency graph

To figure out the problems here, we need to get the dependency relations between each pair of conflicts, and then build a conflict

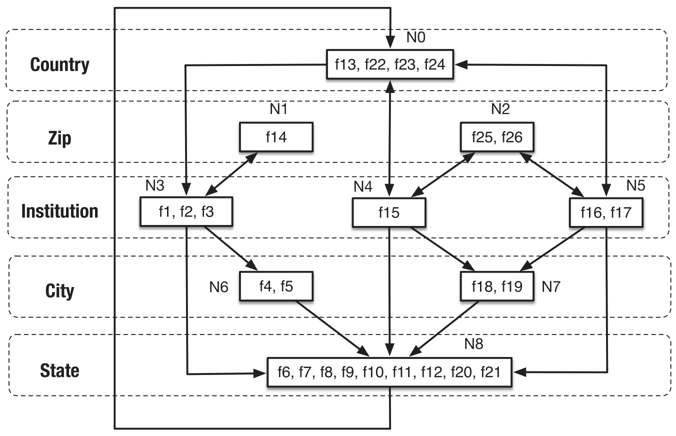


Fig. 7. The dependency graph of Table 2(a).

Table 2
Comparing the repairing quality with previous methods.

	PersonInfo			**DBLP**		
	Precision	Recall	F1	Precision	Recall	F1
CFD-ML	0.535	0.521	0.528	0.645	0.451	0.531
SCARE	0.655	0.512	0.574	0.743	0.453	0.563
GuidedRepair	0.891	0.762	0.821	0.931	0.772	0.844
Web-ADARE	**0.821**	**0.753**	**0.785**	**0.928**	**0.719**	**0.810**

dependency graph based on these relations. After that, we discuss on how the dependency-aware interaction algorithm works on the conflict dependency graph.

(1) *Relations between conflicts.* There are three kinds of relationships between each pair of conflicts. The first is the *Dependency Relation* as we introduced above. Note that the dependency relation is transitive, that is, if f_a depends on f_b, and f_b depends on f_c, then f_a also depends on f_c. Secondly, we say two conflicts are in *Overlapped Relation* if they share some positions, such as f_1 and f_2 sharing t_4[Inst](“QUT”) in Fig. 3(a). Finally, if two conflicts are in neither of the two relations above, they are *Independent* from each other.

(2) *Building conflict dependency graph.* With the relations between all conflicts, we can built a conflict dependency graph as in Fig. 7 (which is built on Table 2(a)) through the following steps: (1) Initially, we take each conflict as a node in the dependency graph. (2) We then put a directed edge pointing from every conflict f_a to every other conflict f_b if f_b depends on f_a. Note that we only need to put an edge between two conflicts if one directly depends on the other. (3) Finally, to make the graph easier to process, we merge nodes sharing at least one value into one node (i.e., we put overlapped conflicts into one node), and the directed edges of the same direction between two nodes are merged into one directed edge.

(3) *Dependency-aware interaction.* As introduced above, a conflict should be processed after all the conflicts it depends on are processed. But for those overlapped conflicts in the same node, we need to consider the priority of each value that involved in the conflicts for checking. Here we can still rely on the *dScores* of these values. A value with a highest *dScore* in a node can be checked firstly. Each time a value is modified, the graph needs to be updated accordingly.

5.2.2. Solving dependency loops

The main challenge here is how to schedule those conflicts in dependency loops for processing. We say a number of conflicts are in a *dependency loop* if they depend on each other such as f_1, f_2, f_3 and f_{14}. In this situation, the dependency-based interaction principle mentioned above does not work at all. Things become more intractable when there are several loops overlapped with each other at different nodes. As in Fig. 7, there are 19 loops in total and almost every loop is overlapped with some other loops at some nodes. Basically, we have to choose one (or more than one) node in a loop to process to “break up” the loop. In order to minimize the cost, we have to be very careful in selecting the *break-up node* for a loop as different break-up nodes will bring different costs.

(1) Breaking up a single loop: We basically consider two factors in selecting the break-up node for a loop: (1) factor 1: the number of values that must be verified in a node for breaking up the loop (for easier presentation, we call these values as break-up values); (2) factor 2: the *dScores* of these break-up values in a node. Usually, we tend to select the node with the least number of break-up values holding the highest *dScores* as the break-up node for the loop. More specifically, we calculate a break-up score, or *bScore* for short, for each node in a loop as given in Eq. (13) below. Among all nodes in a loop, the node with the highest *bScore* will be selected as the break-up node in priority.

$$bScore(\mathcal{N},\mathcal{L}) = \prod_{v\in V_b(node,loop)} dScore(v), \qquad (13)$$

where $V_b(\mathcal{N},\mathcal{L})$ is the set of break-up values in a node *node* for breaking up the loop *loop*.

(2) Breaking up multiple loops: For a number of loops overlapped with each other, we can not simply decide the break-up nodes for a single loop. Otherwise, we may not be able to reach the best performance in minimizing the number of web-aided operations. For each node, we consider a global *bScore*, or *gbScore* for short, to denote its break-up score for all loops in the graph, and the one with the highest *gbScore* will be selected as the break-up node in priority. The *gbScore* of a node is decided by two factors: (1) the local *bScore* of the node in each loop; and (2) the benefit of solving each loop, which is actually the number of values that can be moved out from the loops. More specifically,

$$gbScore(\mathcal{N}) = \sum_{\mathcal{L}\in L(\mathcal{N})} [bScore(\mathcal{N},\mathcal{L}) \times benefit(\mathcal{N},\mathcal{L})], \qquad (14)$$

where $L(\mathcal{N})$ is the set of loops having $\mathcal{N}$ as its node in the graph, and the $benefit(\mathcal{N},\mathcal{L})$ is the benefit of breaking up $\mathcal{L}$ by solving $\mathcal{N}$, which is mainly decided by the number of values in $\mathcal{L}$.

5.2.3. The algorithm

A formal description of this algorithm is given in Algorithm 2: Initially, we build the conflict dependency graph for a data set. For those nodes depending on nothing, we keep on choosing the value with the highest *dScore* within each node for web-aided repairing until all conflicts in the node are resolved. When there are no nodes of this kind but only loops, we calculate the *gbScores* for all nodes in these loops, and choose the one with the highest *gbScore* to process to break up the loops. Each time a value is modified, we need to update the graph and all *bScores* and *gbScores*. The algorithm stops when the graph is empty.

With this algorithm, we only do web-aided repairing to a much smaller selected subset of attribute values.

6. Experiments

This section presents our experimental results. The experimental environment is a four-core Intel Core i7, 8 GB memory machine, running Mac OS X. All the approaches are implemented using Java.

6.1. Data sets

We conduct experiments on 2 real and 1 synthetic data sets.

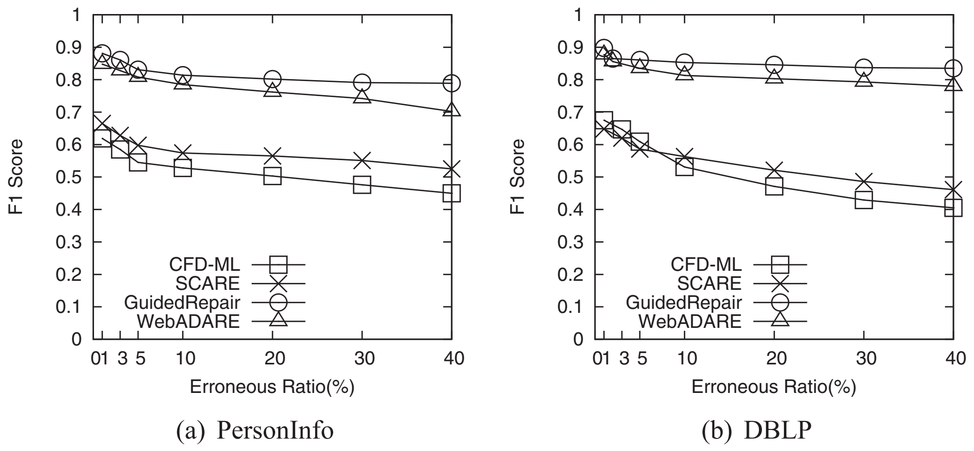


Fig. 8. Comparing the F1 scores of all methods.

Algorithm 2: Dependency-aware interaction.

```
Input  : A table with a set of conflicts $\mathbb{F}$
Output: A repairing scheme $\mathcal{S} = \langle \mathcal{T}_0, \mathcal{W}_1, \mathcal{T}_1, \ldots, \mathcal{W}_n, \mathcal{T}_n \rangle$

Set $i = 0$;
while $\mathbb{F} \neq \emptyset$ do
    1. $\mathcal{T}_i \leftarrow$All deducible values at the moment;
    2. Deducing all values in $\mathcal{T}_i$;
    3. Updating $\mathbb{F}$;
    4. $i++$;
    5. Calculating dScores for all values in $\mathbb{F}$ with Eq. (11);
    6. Building the Dependencies Graph on $\mathbb{F}$;
    7. while no new deducible values do
        $V \leftarrow$ Values in conflicts depending on nothing;
        if $V \neq \emptyset$ then $\mathcal{W}_i \leftarrow \mathcal{W}_i \cup V$;
        else
            Calculating gbScores for all conflicts in $\mathbb{F}$ with Eq. (14);
            $V \leftarrow$ Values with the highest dScore in conflicts
            with the highest gbScore;
            $\mathcal{W}_i \leftarrow \mathcal{W}_i \cup V$
        Checking/Repairing $V$ with the Web;
        if $V$ is updated with update value then
            Updating $\mathbb{F}$ and dScores;
return $\langle \mathcal{T}_0, \mathcal{W}_1, \mathcal{T}_1, \ldots, \mathcal{W}_n, \mathcal{T}_n \rangle$;
```

1) Personal information table (*PersonInfo*): This is a 50k-tuples, 9-attributes table, which contains contact information for academics including name, email, title, university, street, city, state, country and zip code. This information was collected from more than 1000 universities in the USA, UK, Canada and Australia.
2) DBLP publication table (*DBLP*): This is a 100k-tuples, 5-attributes table. Each tuple contains information about a published paper, including its title, first author and his/her affiliation, conference name, year and venue. All were randomly selected from DBLP.
3) Synthetic table (*Syn*): We also generate a 1million-tuples, 100-attributes table following a scheme containing 100 randomly generated approximate attribute dependencies with confidences near-uniformly distributed between 0.7 and 1, where the first attribute is the key.

To generate tables with errors for experiments from the three original data sets, we keep the key attribute value in each tuple and replace non-key attribute values at random positions with attribute values selected from random picked tuples. That is, either the person's name or his/her email will be kept in each tuple for PersonInfo dataset, and all paper titles will be kept for DBLP dataset, while the values of the first (key) attribute will be kept for the synthetic data set. The original tables are used as the ground truth.

Here we use a synthetic data set since it can be easily adjusted to test the effectiveness and scalability of our techniques in various situations. Note that the experiments on this synthetic data set do not really retrieve the values from the web but get them from the ground truth table.

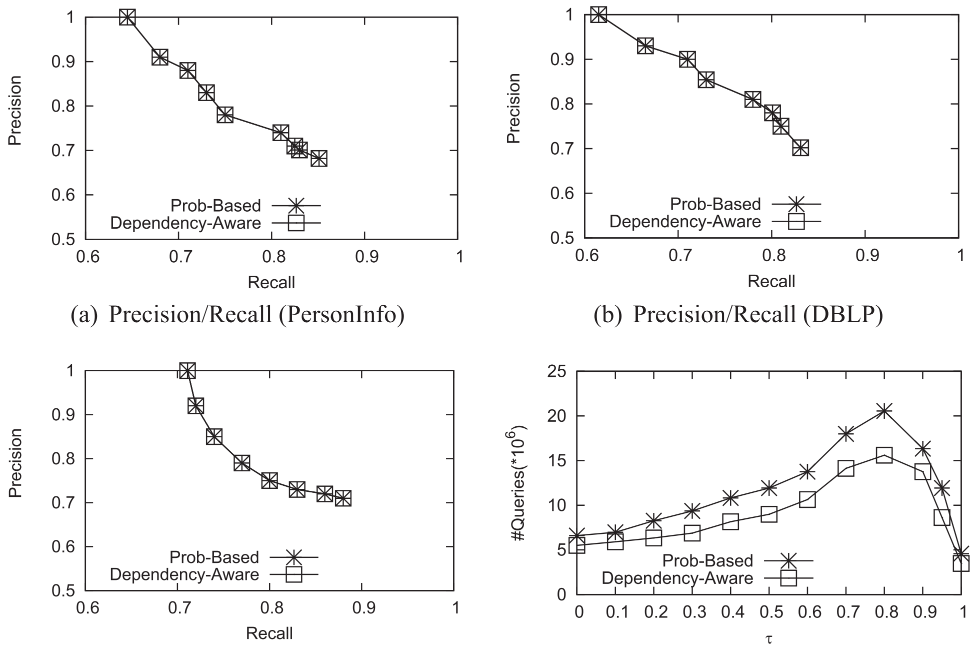
6.2. Repairing quality evaluation

We compare the repairing quality of PureWeb (Pure Web-Aided Repairing) and Web-ADARE with three state-of-the-art general textual data repairing approaches below.

1) Rule-based (*CFD-ML*): This method relies on FD/CFDs to detect and correct erroneous data [3], and follows the most-likely correct modification criterion.
2) Model-based (*SCARE*): This is a model-based repairing approach based on maximizing the correctness likelihood of replacement data given the data distribution, which is modeled using statistical machine learning techniques [23].
3) Crowd-based1 (*GuidedRepair*): We implement the Guided Data Repairing method proposed in [6] by using the experts' feedbacks (correct update or not) to adaptively refine the training set for the employed repairing model. The feedbacks are given by a group of students in our lab.

We use the standard precision, recall and F1 Score to measure the repairing quality: (1) *Precision*, the percentage of correctly modified values among all modified values; (2) *Recall*, the percentage of correctly modified values among all erroneous values; (3) *F1*, a combined measure calculated by 2*precision*recall/ (precision + recall). We first make a comprehensive comparison on the Precision, Recall and F1 of all methods at an erroneous ratio of 10% on the two real data sets in Table 2, and then compare the F1 scores of all methods at various erroneous ratios (1%, 3%, 5%, 10%, 20%, 30%, 40%) by setting $\tau = 0.7$ over the two real-world data sets in Fig. 8. The parameter setting for each method lets it reach the best repairing quality (w.r.t. F1).

(1) Rule/Model-based repair v.s. Web-aided repair: As shown in Table 2, the precision and recall of the the rule-based method (CFD-ML) is comparatively low, as it can only make correct modifications to about half of the erroneous values in the data sets, and in 40–60% chances they make wrong corrections. The reasons are two: (1) about 20% erroneous values in each data set can not be



(a) Precision/Recall (PersonInfo)

(b) Precision/Recall (DBLP)

(c) Precision/Recall (Syn)

(d) #Queries (Syn)

Fig. 9. Comparing the schemes: precision and recall in (a),(b),(c) (three data set), and #Queries in (d) (Syn).

detected at all by the employed conflict detector as these errors do not arouse any conflicts in the data set; (2) without external knowledge, following a simple modification criterion is very likely to make mistakes.

Comparatively, the precision of the model-based method (SCARE) is a bit (5–10%) higher than the rule-based methods since the models they build can understand the correlation between data and thus make better judgements. On the other hand, its recall is as low as that of the rule-based method, since there are some non-quantitive attributes like email, street, author, and venue which can not be handled well by the models. In contrast, the precision and recall of Web-ADARE is much higher (80–90% precision and 80–85% recall), which benefits from the external data on the Web. Fig. 8 also reflects that Web-ADARE always reaches much higher F1 scores than all the rule-based or model-based methods, which proves the advantage of Web-ADARE over the latter ones.

(2) Crowsourcing repair v.s. Web-aided repair: As depicted in Fig. 8 and Table 2, Web-ADARE reaches nearly as high precision and recall as the GuidedRepair. This is a significant achievement as our method Web-ADARE requires no human interventions, while GuidedRepair gets every answer from humans and requires 1 human intervention per 20 suspicious values on average. That would be a great cost for a large data set.

Specifically, as the erroneous rate increases from 1 to 40% in Fig. 8, the gap between the two methods increases from 0.3 to 5.6%. This is because the web-aided repairing needs to leverage the update values to formulate repairing queries for the other values. As a result, if the leveraged update data is incorrect, then more errors will be introduced. Therefore, as the erroneous rate increases, the overall repairing precision may decrease. The same problem also exists in other methods but is less obvious.

6.3. Comparison between interaction schemes

We first compare the cost of the hybrid repairing method (WebAidRepair for short) with pure web-based repairing method (PureWebRepair) and pure rule-based method (CFD-ML) on the number of issued Web queries (*#Queries*). We believe the number of issued queries is a much better indicator than time or the number of retrieved values to the web consultation cost of the method, since the time is greatly depend on the machines used for parallel computation, and the #Queries of a method is usually a bit larger than the number of retrieved values provided that a fired retrieving query may fail to retrieve the update value for the target erroneous value, and thus we need to fire alternative ones. As demonstrated in Fig. 10, both the cost of PureWebRepair and Web-ADARE increase linearly as the erroneous ratio increases from 1 to 40%, but Web-ADARE only retrieves about 20% of the values retrieved by PureWebRepair, which proves that our algorithm can greatly reduce the overhead of the web-aided repairing.

We then conduct our second group of experiments to evaluate the interaction schemes generated by the two proposed algorithms over the three data set. In particular, we set the erroneous ratio to 10%, and then compare the repairing quality (precision and recall) and the cost of the interactive repairing following each interaction scheme by changing the quality threshold τ from 0 to 1. As shown in Fig. 9(a)–(c), the two schemes can always reach the same precision and recall over the three data sets. Although the probabilistic-based scheme and the dependency-aware scheme can reach a higher recall than the quality-aware scheme, their precision is always 5% lower than that of the quality-aware scheme. Overall, the quality-aware scheme reaches a higher combination of precision and recall than the other two methods. On the other hand, the cost of the two schemes increases as τ increases from 0 to about 0.8, but decreases sharply as τ increases from 0.8 to 1.0 as shown in Fig. 9(d). This makes sense since when the quality constraint becomes too strict, much less values can be repaired to satisfy the constraint. Nonetheless, the cost of the dependency-aware interaction scheme is always about 20% less than the probabilistic-based scheme, which proves the advantage of the dependency-aware scheme over the other one.

6.4. Scalability

We finally evaluate the scalability of Web-ADARE. We run experiments on the Syn data set by first changing the number of tu-

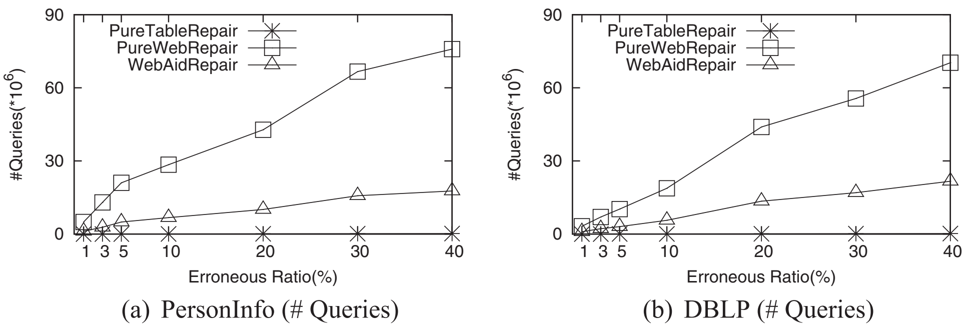


Fig. 10. #Queries on the real data sets ($\tau = 0.7$).

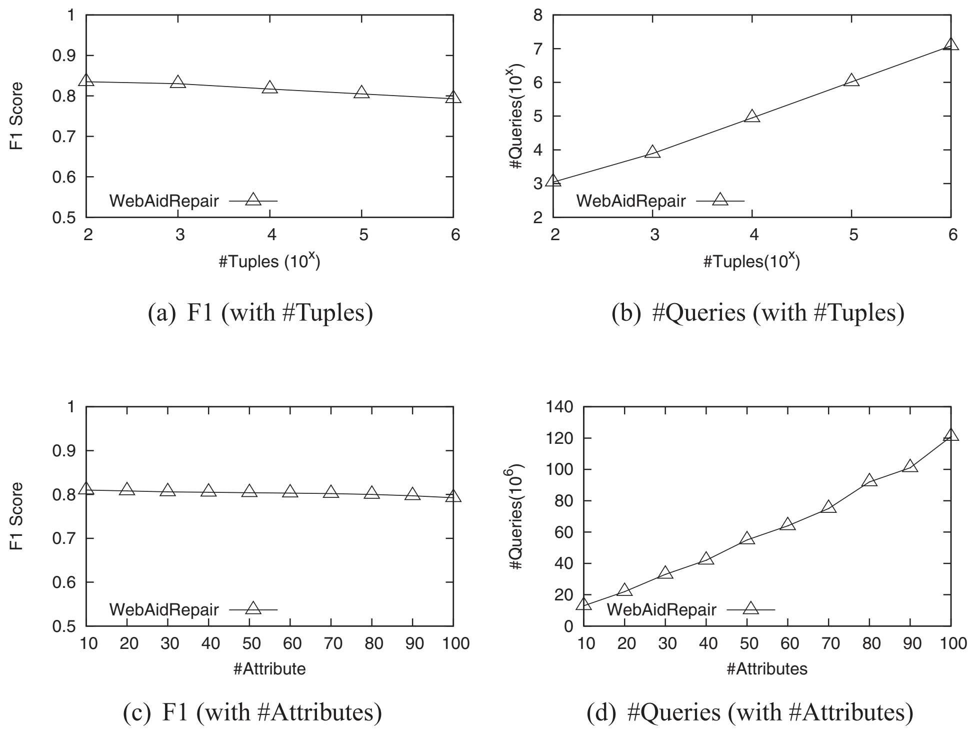


Fig. 11. Scalability: the F1 and cost (#Queries) with different number of tuples, or attributes (set $\tau = 0.7$, erroneous ratio = 10%).

ples from 100 to 1million (Fig. 11(a) and (b)), and then changing the number of attributes in the table from 10 to 100 (Fig. 11(c) and (d)). Fig. 11(a) and (c) demonstrate that the F1 does not change much as the table becomes large in either horizontal or vertical direction, while Fig. 11(b) and (d) show that the cost increases linearly as the table becomes larger in either of the two directions.

7. Related work

Textual data cleaning (or repairing) aims at detecting and repairing erroneous textual data in databases. So far, plenty of efforts have been put on this direction within data quality and databases communities, and the existing solutions can be roughly put into three categories below.

The first category of solutions, the mainstream ones, rely on a variety of constraints including Functional Dependencies (FDs) [24,25], Conditional Functional Dependencies (CFDs) [15], Integrity Constraints [5] and Inclusion Dependencies (INCs) [25] to detect inconsistencies (or conflicts) between data aroused by erroneous data, and then work on resolving all the conflicts with expecting to fix all erroneous data in this way [3–5]. For general textual databases, most work in this category use FD/CFDs for repairing as they are the constraints within a single relational table, while some other work uses INCs for repairing between multiple relational tables. Usually, this category of methods can effectively detect a large percent of erroneous data involved in the identified conflicts in a wide range of databases, but to repair these errors and resolve the conflicts, some work tends to make the least changes to the data set [3,5], while others prefer to make the most likely correct changes based on some simple prediction model [4,23]. However, neither criterion can have all errors modified correctly.

The second category of solutions are model-based repairing, which usually build some prediction models for detecting and correcting erroneous values in a data set [23,26–28]. The construction of the model employs statistical Machine Learning (ML) techniques for data cleaning, which can effectively capture the dependencies and correlations between data in the dataset based on various analytic, predictive or computational models [23,27]. However, not every erroneous data can be identified and corrected in the right way since there are always outliers that do not obey the captured constraints.

The third category of solutions are based on external sources such as the crowd [6] or some existing reference data [7]. However, the required external sources are not always available and thus the methods can not be applied in general scenarios. In this paper, we propose Web-ADARE, which is also an external source-based repairing approach. Compared to previous approaches of the kind, Web-ADARE is more general as we rely on the Web.

To minimize the cost of Web-ADARE, we work on the interaction problem between web-aided repairing and rule-based repairing. A similar interaction problem has been studied between record matching and data cleaning by Fan et. al.,[29], but the key settings of their problem is different from ours in the following two aspects: (1) The record matching in their problem is performed between the objective database and a reference master data which is assumed to contain absolute clean data for reference. However, this "oracle" master data is not always available, while the Web we rely on can be accessed everywhere. (2) They work towards using the minimum changes to make the objective data consistent with the reference master data w.r.t. a set of predefined matching rules, without considering whether these used changes and rules can repair all errors correctly. In our work, we consider the reliability (repairing quality) of the repairing operations as we lay more emphasis on the correctness of the repairing results.

8. Conclusions and future work

We propose Web-ADARE, an end-to-end web-aided data repairing system, which can greatly enhance the repairing quality of the existing rule-based repairing method by getting the update values from the Web. Compared to crowd-based repairing, the web-aided repairing does not need any human interventions, and thus can be much more efficient and cheaper. In Web-ADARE, we developed three modules: the Web Data Fetcher module fetches specified update values from the Web, the Quality Supervisor module controls the quality of the update values, while the Interaction Scheduler schedules the repeated alternation of web-aided and rule-based repairing. The experimental results based on several data sets demonstrate that Web-ADARE can reach as high repairing quality as crowd-based repairing, and our techniques can not only guarantee the high quality of the update values from the Web, but also greatly decrease the web consultation cost.

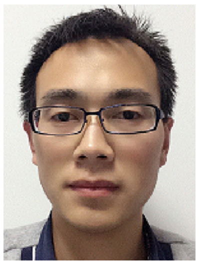
Several future work is in considerations: (1) the combination of web-aided repairing and crowd-based repairing to exploit the strengths of each; (2) to handle data in large volume, we will move our system from a single server to a cluster of servers with Hadoop platform; (3) develop a deep-web data fetcher that can get update data from the deep web; (4) develop a sensitive repairing task distributor that can repair different kind of errors with different repairing techniques.

Acknowledgments

This research is partially supported by Natural Science Foundation of China (Grant Nos. 61303019, 61402313, 61472263, 61572336), Postdoctoral scientific research funding of Jiangsu Province (No. 1501090B) National 58 batch of postdoctoral funding (No. 2015M581859), Collaborative Innovation Center of Novel Software Technology and Industrialization, Jiangsu, China, and the King Abdullah University of Science and Technology.

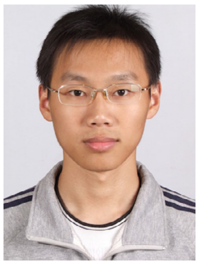
References

[1] W.W. Eckerson, Data Quality and The Bottom Line: Achieving Business Success Through a Commitment to High Quality Data, The Data Warehousing Institute, 2002, pp. 1–36.
[2] Gartner, in: Data Quality Tools, Worldwide, 2006–2011, Gartner, 2007.
[3] G. Cong, W. Fan, F. Geerts, X. Jia, S. Ma, Improving data quality: consistency and accuracy, in: Proceedings of the Thirty-Third International Conference on Very Large Data Bases, 2007, pp. 315–326.
[4] S. Kolahi, L.V. Lakshmanan, On approximating optimum repairs for functional dependency violations, in: Proceedings of the International Conference on Database Theory, 2009, pp. 53–62.
[5] A. Lopatenko, L. Bravo, Efficient approximation algorithms for repairing inconsistent databases, in: Proceedings of the International Conference on Data Engineering, 2007, pp. 216–225.
[6] M. Yakout, A.K. Elmagarmid, J. Neville, M. Ouzzani, I.F. Ilyas, Guided data repair 4 (5) (2011) 279–289.
[7] W. Fan, J. Li, S. Ma, N. Tang, W. Yu, Towards certain fixes with editing rules and master data, Proc. VLDB Endow. 3 (1–2) (2010) 173–184.
[8] M. Dallachiesa, A. Ebaid, A. Eldawy, A. Elmagarmid, I.F. Ilyas, M. Ouzzani, N. Tang, NADEEF: a commodity data cleaning system, in: Proceedings of the ACM SIGMOD International Conference on Management of Data, 2013, pp. 541–552.
[9] J. Zhou, Z. Li, B. Gu, Q. Xie, J. Zhu, X. Zhang, G. Li, CrowdAidRepair: a crowd-aided interactive data repairing method, in: Proceedings of the International Conference on Database Systems for Advanced Applications, 2016, pp. 51–66.
[10] R. Wang, W. Cohen, Iterative set expansion of named entities using the web, in: Proceedings of the International Conference on Data Mining, IEEE, 2008, pp. 1091–1096.
[11] R. Wang, W. Cohen, Automatic set instance extraction using the web, in: Proceedings of the Joint Conference of the Forty-Seventh Annual Meeting of the ACL and the Fourth International Joint Conference on Natural Language Processing of the AFNLP, Association for Computational Linguistics, 2009, pp. 441–449.
[12] S. Brin, Extracting Patterns and Relations From the World Wide Web, in: The World Wide Web and Databases, Springer, 1999, pp. 172–183.
[13] E. Agichtein, L. Gravano, Snowball: extracting relations from large plain-text collections, in: Proceedings of the Fifth ACM Conference on Digital Libraries, 2000, pp. 85–94.
[14] Z. Li, M.A. Sharaf, L. Sitbon, S. Sadiq, M. Indulska, X. Zhou, A web-based approach to data imputation, World Wide Web 17 (5) (2014) 873–897.
[15] W. Fan, F. Geerts, X. Jia, A. Kementsietsidis, Conditional functional dependencies for capturing data inconsistencies, ACM Trans. Database Syst. 33 (2) (2008) 6.
[16] Z. Li, L. Sitbon, X. Zhou, Learning-based relevance feedback for web-based relation completion, in: Proceedings of the International Conference on Information and Knowledge Management, 2011, pp. 1535–1540.
[17] M.A. Hearst, Automatic acquisition of hyponyms from large text corpora, in: Proceedings of the Fourteenth Conference on Computational Linguistics, 2, 1992, pp. 539–545.
[18] A. Mikheev, M. Moens, C. Grover, Named entity recognition without gazetteers, in: Proceedings of the Ninth Conference on European Chapter of the Association for Computational Linguistics, 1999, pp. 1–8.
[19] Z. Li, L. Sitbon, L. Wang, X. Zhou, X. Du, AML: efficient approximate membership localization within a web-based join framework, IEEE Trans. Knowl. Data Eng. 25 (2) (2013) 298–310.
[20] X.L. Dong, L. Berti-Equille, D. Srivastava, Integrating conflicting data: the role of source dependence., Proc. VLDB Endow. 2 (1) (2009) 550–561.
[21] T. Rekatsinas, X.L. Dong, D. Srivastava, Characterizing and selecting fresh data sources, in: Proceedings of the ACM SIGMOD International Conference on Management of Data, ACM, 2014, pp. 919–930.
[22] W. Lin, R. Yangarber, R. Grishman, Bootstrapped learning of semantic classes from positive and negative examples, in: Proceedings of the International Conference on Machine Learning Workshop on The Continuum from Labeled to Unlabeled Data, 2003.
[23] M. Yakout, L. Berti-Équille, A.K. Elmagarmid, Don't be scared: use scalable automatic repairing with maximal likelihood and bounded changes, in: Proceedings of the ACM SIGMOD International Conference on Management of Data, 2013, pp. 553–564.
[24] J. Wijsen, Database repairing using updates, ACM Trans. Database Syst. 30 (3) (2005) 722–768.
[25] P. Bohannon, W. Fan, M. Flaster, R. Rastogi, A cost-based model and effective heuristic for repairing constraints by value modification, in: Proceedings of the ACM SIGMOD International Conference on Management of Data, 2005, pp. 143–154.
[26] C. Mayfield, J. Neville, S. Prabhakar, ERACER: a database approach for statistical inference and data cleaning, in: Proceedings of the ACM SIGMOD International Conference on Management of Data, 2010, pp. 75–86.
[27] X. Zhu, X. Wu, Class noise vs. attribute noise: a quantitative study, Artif. Intell. Rev. 22 (3) (2004) 177–210.
[28] J.L. Koh, M.L. Lee, W. Hsu, K.T. Lam, Correlation-based detection of attribute outliers, in: Advances in Databases: Concepts, Systems and Applications, Springer, 2007, pp. 164–175.
[29] W. Fan, J. Li, S. Ma, N. Tang, W. Yu, Interaction between record matching and data repairing, in: Proceedings of the ACM SIGMOD International Conference on Management of Data, 2011, pp. 469–480.


Binbin Gu is a master student in the School of Computer Science and Technology at Soochow University, China. His research interests include Data Integration and Data Imputation. He has published several papers at some International Conferences including DASFAA, WAIM and APWEB. He is the external reviewer of several International Conferences such as ADC, WAIM and CCF BigData.

Zhixu Li is an Associate Professor in the School of Computer Science and Technology at Soochow University, China. He received his Ph.D. degree from the University of Queensland in 2013, and his B.S. and M.S. degree from Renmin University of China in 2006 and 2009, respectively. His research interests and experiences are in Data Cleaning, Big Data Applications, Information Extraction and Information Retrieval.

Qiang Yang is a master student in the School of Computer Science and Technology at Soochow University, China. His research interests include Data Quality, Information Extraction and NLP. He has published several papers at some International Conferences including DASFAA, WAIM, APWEB and NDBC. He is the external reviewer of several International Conferences such as ADC and WAIM.


Qing Xie is currently an Associate Professor in the School of Computer Science and Technology at Wuhan University of Technology. He received the B.E. degree in information science from University of Science and Technology of China in 2008, and the Ph.D. degree in computer science from the University of Queensland in 2013. After that, he worked as a postdoctoral research fellow at King Abdullah University of Science and Technology. His research interests include stream data mining, time series databases, continuous query optimization and e-Health.

An Liu is an associate professor in the Department of Computer Science & Technology at Soochow University. Prior to coming to Soochow University in 2014, he was a Senior Research Associate in the Joint Research Center of City University of Hong Kong (CityU) and University of Science & Technology of China (USTC). He received his Ph. D. from both CityU and USTC in 2009.

Guanfeng Liu is an Associate Professor in the School of Computer Science and Technology at Soochow University, China. He received his Ph.D. degree in Computer Science from Macquarie University, Australia in 2013. His research interests include social network mining and trust. He has published over 40 papers in the most prestigious journals and conferences such as ICDE, AAAI, ICWS and IEEE Transactions. He was the PC Chair of BDMS 2015.

Kai Zheng is Professor with the School of Computer Science at Soochow University. He received his Ph.D. degree in Computer Science from the University of Queensland in 2012. His research focus is to find effective and efficient solutions for managing, integrating and analyzing big data for business, scientific and personal applications. He has been working in the area of spatial-temporal databases, uncertain databases, trajectory computing, social- media analysis and bioinformatics. He has published over 60 papers in the highly referred journals and conferences such as SIGMOD, ICDE, EDBT, The VLDB Journal, ACM Transactions and IEEE Transactions.

Xiangliang Zhang is an Assistant Professor and directs the Machine Intelligence and kNowledge Engineering (MINE) Laboratory in King Abdullah's University of Science and Technology (KAUST). She earned her Ph.D. degree in computer science with great honors from INRIA-University Paris-Sud 11, France, in July 2010. Her main research interests and experiences are in diverse areas of machine intelligence and knowledge engineering.