Leveraging Currency for Repairing Inconsistent and Incomplete Data

Xiaoou Ding, Hongzhi Wang, Jiaxuan Su, Muxian Wang, Jianzhong Li, and Hong Gao

Abstract—Data quality plays a key role in big data management today. With the explosive growth of data from a variety of sources, the quality of data is faced with multiple problems. Motivated by this, we study the multiple data cleaning on incompleteness and inconsistency with currency reasoning and determination in this paper. We introduce a 4-step framework, named Imp3C, for errors detection and quality improvement in incomplete and inconsistent data without timestamps. We achieve an integrated currency determining method to compute the currency orders among tuples, according to currency constraints. Thus, the inconsistent data and missing values are repaired effectively considering the temporal impact. For both effectiveness and efficiency consideration, we carry out inconsistency repair ahead of incompleteness repair. A currency-related consistency distance metric is defined to measure the similarity between dirty tuples and clean ones more accurately. In addition, currency orders are treated as an important feature in the missing imputation training process. The solution algorithms are introduced in detail with case studies. A thorough experiment on three real-life datasets verifies our method Imp3C improves the performance of data repairing with multiple quality problems. Imp3C outperforms the existing advanced methods, especially in the datasets with complex currency orders.

Index Terms—Data cleaning, data quality management, currency determining, temporal data repairing

1 Introduction

Data quality plays a key role in data-centric applications [1]. The data quality problems are quite serious and trouble data transaction steps (e.g., acquisition, copy, query). Specifically, currency, consistency and completeness are three important issues in data cleaning [2]. Various information systems store data with different formats or semantic. It may lead to costly consistency problems in multi-source data integration. In addition, with imperfect integrity standard of information systems, the tuples in database may have missing values [3]. Worse still, the low frequency in data update makes it out-of-date to some degree when timestamps are missing under loose or imprecise data copy functions among sources [4]. These three problems result in the low reliability of data, which add to the confusion in data applications. Thus, the low-quality data may result in negative impact on many fields.

Researchers have gone a long way in data repairing, particularly in consistency and completeness. It is acknowledged that consistency and completeness issues may affect each other during repairing, rather than completely isolated [2], [5], and sometimes the both approaches are complementary. We have found that currency issues also seriously impact the repair of inconsistent and incomplete values. With precise timestamps in database, data cleaning tasks such as entity resolution and constraint-based data repairing can be solved by the state-of-art data cleaning methods [6] and systems [7]. However, incomplete timestamps in temporal data make this problem challenged to be both detected and repaired. As it is of great importance to use up-to-date data and understand timeliness of records in information systems such as Customer Relationship Management (CRM) [8] and Human Resource management (HRM) [5], [9], incomplete timestamps in low-quality datasets with both inconsistent and incomplete data make this problem challenging to be both detected and repaired. We present a motivation example for an industry talents database to illustrate shortcomings of existing data repairing methods without leveraging currency issues.

Example 1. Table 1 shows several tuples recorded in an industry talents database, describing two individuals (entities), Mike and Helen, with 8 attributes. Level is an industry-recognized career rank, while Title is the post of the employee. City records the city that the Company locates in. Abbreviations are used in the current professional Email. As the data came from multi-sources, the timestamps are missing. Both inconsistent and incomplete problems exist in attribute values.

Table 1 marks wrong values in 4 tuples in red. For Mike, r_5 describes Mike works in Comp2 (Hangzhou). Meanwhile, the city is Beijing. It leads to a confusion and we can conclude that inconsistent values exist in City, or even in Company and Group of r_5. For Helen, r_9, r_{10} and r_{14} contain missing values. It is difficult to deduce when she began working in Shanghai and how much her current salary is.

Table 2 shows optional repair solutions with existing data repairing methods. For Mike, we can give a relative clean value $15k$ to r_9's missing salary referring to its most

• *The authors are with the Harbin Institute of Technology, P.O.Box 750, Harbin, Heilongjiang 150001, China. E-mail: dingxiaoou@stu.hit.edu.cn, {wangzh, wangmuxian, lijzh, honggao}@hit.edu.cn, itx351@126.com.*

Manuscript received 18 Mar. 2019; revised 3 Apr. 2020; accepted 17 Apr. 2020. Date of publication 4 May 2020; date of current version 3 Feb. 2022. (Corresponding author: Hongzhi Wang.)
Recommended for acceptance by S. Sadiq.
Digital Object Identifier no. 10.1109/TKDE.2020.2992456

TABLE 1
Personal Career Information for Mike and Helen

		Name	Level	Title	Comp	City	Salary	Email	Group
E_1:	r_1:	Mike	L2	E (Engineer)	Baidu	Beijing	13k	M@Bai	Java
	r_2:	Mike	L2	E	Baidu	Beijing	13k	M@Bai	Map
	r_3:	Mike	L2	ME (Major Engineer)	Baidu	Beijing	15k	M@Bai	Map
	r_4:	Mike	L3	ME	Baidu	Beijing	20k	M@Bai	Map
	r_5:	Mike	L4	E	Alibaba	Beijing	22k	M@ali	Tmall
	r_6:	Mike	L4	E	Alibaba	Hangzhou	22k	M@ali	Financial
	r_7:	Mike	L4	RE (Research Engineer)	Alibaba	Hangzhou	23k	M@ali	Financial
E_2:	r_8:	Helen	L2	RA (Research Assistant)	Tencent	Shenzhen	15k	H@QQ	Game
	r_9:	Helen	L3	R (Researcher)	Tencent	Shenzhen		H@QQ	Game
	r_{10}:	Helen	L3	R	Tencent		18k	H@QQ	Financial
	r_{11}:	Helen	L3	R	Tencent	Shanghai	20k	H@QQ	Social Network
	r_{12}:	Helen	L4	R	Microsoft	Beijing	22k	H@outlook	Social Computing
	r_{13}:	Helen	L4	MR (Major Researcher)	Microsoft	Beijing	22k	H@outlook	Social Computing
	r_{14}:	Helen	L5		Microsoft	Beijing		H@outlook	Social Computing

similar record r_8, but things are not simple when repairing other errors. Both company and group that Mike works in do not coincide with the city in r_5. It is possible to repair r_5 with r_4's attribute values, according to the minimum repair principle proposed in [10]. However, Mike has actually begun working in Comp2 at the time of r_5, which implies r_5 is more current than r_4. Thus, this repair is poor without considering the temporal issues.

For Helen, the edit distance $dist(r_9, r_{10}) = dist(r_{10}, r_{11})$ according to [11] makes it difficult to distinguish which is closer to r_{10}, and it also presents no currency difference among r_9, r_{10}, and r_{11}. Similarly, it seems no difference to repair r_{14} with either *R* or *MR* because of the equal $dist(r_{12}, r_{14})$ and $dist(r_{13}, r_{14})$.

From the above, it is difficult to clean inconsistent and incomplete values without the guidance of available timestamps. If repairing them simply with values from their most similar tuples, it is likely to obtain wrongly repair result, and even run into serious mistakes in data applications. Thus, the repairing of data quality problems in currency, consistency and completeness (3C) together is in demand. However, mixed quality cleaning issues are faced with challenges. First, with attribute values change and evolve with time, the temporal features in tuples influence the repairing accuracy. Moreover, as some overall fundamental problems are known as computationally hard [4], [10], multi-errors data repairing makes it even more challenged. Worse still, repairing one error may cause any other kind of errors. Without a sophisticated method, it is costly to repair dirty data due to the iteratively error repairing steps caused by data cleaning.

As yet, works on cleaning multiple errors with data currency are still inadequate. Currency orders are difficult to be determined when timestamps are unavailable. Existing currency repairing methods mostly depend on definite timestamps (e.g., [9]), and few works provide feasible approaches for the data with the absence of valid timestamps. Motivated by this, we improve the quality of data repairing on consistency and completeness with data currency orders. We continue to illustrate the benefit of currency determining in data repairing from Example 1.

Table 2 presents better repairs of tuples in Table 1 in green. We deduce a currency order that the title of an employee in a company is increasing in the real world. Thus, Mike's title can only change from *E* to *ME* when he works in the same company. Similarly, the salary is always monotonically increasing. Thus, r_5 is expected to be more current than r_4. r_5[City] should be repaired. The error occurrence is possibly because of the delay between database update and changes in the real world. In addition, if emails fail to be well-repaired, both employees and companies will suffer losses.

For Helen, it can be deduced that $r_{10}[\mathsf{City}] = Shanghai$ according to a *CFD*: $(([\mathsf{Comp}] = Comp3 \wedge [\mathsf{Group}] = Financial) \rightarrow [\mathsf{City}] = Shanghai)$. It reveals that Helen has changed her work to the financial group in Shanghai at L3. This repair improves the accuracy of her career information. With a currency order: $R \prec_{\mathsf{Title}} MR$, we know Helen has become a MR at L4, and r_{13} is more current than r_{12}. According to another currency order: $L4 \prec_{\mathsf{Level}} L5$, r_{14} is the most current and fresh tuple now. Its missing title and salary should be filled with the latest values, i.e., *L5* and $22k$, respectively. It indicates that Helen's salary is no less than $22k$ at L5 as a MR in her group. These cases indicate the complex conditions in dirty data and the necessary of the interaction method in data cleaning on 3C. From Table 2, the combination of these three issues makes contributions to the accuracy improvement of data cleaning.

Contributions. We summarize our contributions as follows:

1) We propose a comprehensive data repairing approach for consistency, completeness and currency, named Imp3C. To the best of our knowledge, it is the first study to address data cleaning on both inconsistent repair and missing imputation with data currency reasoning without reliable timestamps.
2) We propose an integrated method of currency graph construction and currency order computation with

TABLE 2
Possible Repair Solutions Summary

	Dirty attributes	After repair		Explanation
r_5:	[City]	(*a*).*"Hangzhou"*	✓	An effective repair from currency-related consistency method.
	[Comp], [Group]	(*b*).*"Baidu", "Map"*	×	A poor repair without taking account currency issues.
r_9:	missing [Salary]	(*a*).*"15k"*	✓	A proper clean value.
r_{10}:	missing [City]	(*a*). *"Shanghai"*	✓	An effective repair from currency-related completeness methods.
		(*b*).*"Shenzhen"*	×	A poor repair fails to capture the closet current values.
r_{14}:	missing [Title]	(*a*).*"MR"*	✓	An accurate and current repair
		(*b*).*"R"*	⊗	The repair is less accurate and current.

currency constraints. It achieves a reliable time-related replacement when timestamps is not valid, in order to make sufficient usage of currency information in database. Moreover, we quantitatively compute the reliability metric of the currency order pairs according to the currency graph. In this way, the internal temporal knowledge from tuples can be discovered and awakened to maximize the repairing effectiveness.

3) We define a currency-consistency distance metric between the dirty data and the standard one to repair the inconsistent attributes together with *CFD*s and currency orders. In addition, we propose the solution for missing values imputation based on Naïve Bayes, where the currency order is considered as a key feature for the training process.
4) We conduct a thorough experiment on three real-life datasets. The experimental results verify that the repair strategy in Imp3C is effective. Imp3C outperforms existing methods in repairing multi-errors, specifically in data with frequent currency evolution and changes.

Organization. The rest of the paper is organized as follows: Section 2 discusses the basic definitions and the overview of our approach. Section 3 introduces currency graph construction method, and Section 4 discusses currency order determination algorithms. Section 5 (resp. Section 6) presents inconsistency repairing (resp. missing values imputation) approach. Experimental study is reported in Section 7. Section 8 reviews the related work, and Section 9 draws the conclusion.

2 Overview

In this section, we first introduce background and fundamental definitions in Section 2.1, and then propose the method framework in Section 2.2.

2.1 Preliminaries

In the following, we denote $\mathcal{D}$ as a dataset with tuples satisfying a relational schema $\mathcal{R} = \{\mathsf{rID}, \mathsf{eID}, A_1, \ldots, A_n\}$. rID is the ID number of tuples, and eID is the ID number of entities. $attr(\mathcal{R})$ is the attribute set of $\mathcal{R}$, and $r_i[A_j]$ is the attribute value of r_i on $A_j (A_j \in attr(\mathcal{R}))$.

We assume a partial order on the values of an attribute A_j, to which we refer with "$\prec$". A currency order $\prec_{A_k}$ for each attribute in $\mathcal{D}$ is a partial order on tuples of $\mathcal{R}$ with the property that it only relates tuples with the same eID, i.e., such that $r_i \prec_{A_k} r_j$ implies $r_i[\mathsf{eID}] = r_j[\mathsf{eID}]$. Next, we introduce currency constraints (*CC*s for short), which connect the values of the attribute A_k and their order to the currency order $\prec_{A_k}$ in Definition 1. The semantic of currency constraints adopted in this paper refers to [10].

Definition 1. *A* currency constraint ψ *for* $\mathcal{R}$ *is denoted as* $\forall s,t : \mathcal{R}\big((s[\mathsf{eID}] = t[\mathsf{eID}] \wedge \phi) \longrightarrow s \prec_{A_k} t\big)$. *Let* r_i, r_j *be tuple variables denoting a tuple of* $\mathcal{R}$ *and* ϕ *is a conjunction of predicates of the one from: (1)* $r_i[A_h] = c$ (resp. $r_i[A_h] \neq c$), *where* c *is a constant; (2)* $r_i[A_h]$ *op* $r_j[A_h]$, *op* $\in \{>, <, \geq, \leq, =, \neq\}$; *and (3)* $r_i \prec_{A_h} r_j$.

Accordingly, we can draw the currency constraints for tuples of one entity in Table 1 as follows:

$$\psi_1 : \forall s,t : (s[\mathsf{Salary}] < t[\mathsf{Salary}]) \longrightarrow (s \prec_{\mathsf{Salary}} t).$$
$$\psi_2 : \forall s,t : (s[\mathsf{Level}] < t[\mathsf{Level}]) \longrightarrow (s \prec_{\mathsf{Level}} t).$$
$$\psi_3 : \forall s,t : (s[\mathsf{Comp}] = t[\mathsf{Comp}] \wedge s[\mathsf{Title}] = E \wedge t[\mathsf{Title}] = ME) \longrightarrow (s \prec_{\mathsf{Title}} t).$$
$$\psi_4 : \forall s,t : (s[\mathsf{Comp}] = t[\mathsf{Comp}] \wedge s[\mathsf{Title}] = RA \wedge t[\mathsf{Title}] = R) \longrightarrow (s \prec_{\mathsf{Title}} t).$$

Generally, currency constraints are defined on *attributes*. This contributes to initially determining the currency order between *tuples*. We briefly introduce how to derive tuples' currency order on tuples in Definition 2.

Definition 2 (Currency Order on tuples). *Let* r_i, r_j *be two tuples with* $r_i[\mathsf{eID}] = r_j[\mathsf{eID}]$ *in* $\mathcal{D}$. r_j *is more current than* r_i, *denoted by* $r_i \prec r_j$, *if it satisfies:* i) *there exists* $\mathcal{A}' \subseteq attr(\mathcal{R})$, $\forall A_k \in \mathcal{A}'$, *that* $r_i \prec_{A_k} r_j$, *and* ii) $\nexists A_h \in attr(\mathcal{R}) \backslash \mathcal{A}', r_j \prec_{A_h} r_i$.

Tuples' currency order deduced from *CC*s satisfies irreflexivity, antisymmetry and transitivity. It is the foundation of determining currency on all tuples in a dataset. Accordingly, we propose *currency graph* $\mathcal{G}$ (see Definition 5) for the tuples, and construct *currency order graph* $\mathcal{G}_c$ (see Definition 6) by transitive closure discovery in Section 3.

Conditional function dependencies (*CFD*s) have been developed to detect and resolve inconsistency in datasets [10]. We adopt *CFD*s in our method to improve data consistency as discussed in Definition 3.

Definition 3. *A* CFD *on* $\mathcal{R}$ *is defined as* $\mathcal{R}(X \rightarrow Y, T_p)$, *where* i) $X \rightarrow Y$ *is a standard FD, and* ii) T_p *is a tableau with attributes in* X *and* Y.

Here, $X \subseteq attr(\mathcal{R}), Y \subseteq attr(\mathcal{R})$, *and for each attribute* $A \in X \cup Y$ *and for each pattern tuple* $t_p \in T_p$, $t_p[A]$ *is either a constant value in* $dom(A)$, *or an unnamed variable "_" which draws values from* $dom(A)$.

Note that each pattern tuple t_p in one T_p indicates a constraint. A *CFD* $(X \rightarrow Y, t_p)$ with single-tuple tableau is considered as a constant *CFD* if t_p only consists of constants, i.e., for all attributes $A \in X \cup Y$, $t_p[A]$ is a constant value. For the purpose of this paper, we focus on such constant *CFD*s to detect inconsistency in attributes of the same tuple. We denote φ as one constraint with a *pattern tuple* t_p. For example, given *CFD*: ([Comp], [Group] → [City]) for Table 1, we have three instances of the semantic of *CFD*s below.

$$\varphi_1 : ([\mathsf{Comp}] = Comp1 \wedge [\mathsf{Group}] = Map) \longrightarrow ([\mathsf{City}] = Beijing).$$
$$\varphi_2 : ([\mathsf{Comp}] = Comp2 \wedge [\mathsf{Group}] = E-commerce) \longrightarrow ([\mathsf{City}] = Hangzhou).$$
$$\varphi_3 : ([\mathsf{Comp}] = Comp3 \wedge [\mathsf{Group}] = Games) \longrightarrow ([\mathsf{City}] = Shenzhen).$$

Our work in this paper focuses on cleaning databases with multiple kinds of deficiencies: 1) tuples lacking timestamps, 2) tuples lacking values for some attributes, and 3) tuples that are inconsistent in so far as violating the *CFD*s. To repair such deficiencies, we apply the above-mentioned *CC*s and *CFD*s. In addition, when imputing missing values, we assume that for each attribute A, there is a confidence threshold $\sigma(A)$ that an imputed value has to reach in order to be accepted. Since that imputation results may not be quite accurate with incomplete information, $\sigma(A)$ can be set artificially to avoid meaningless imputation results. We

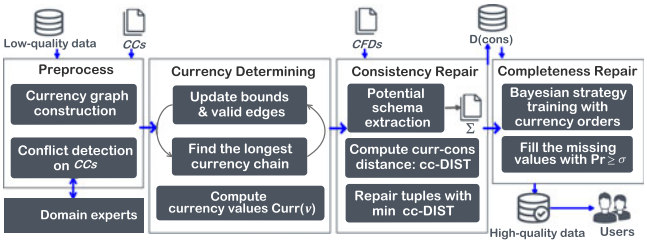


Fig. 1. Framework overview of Imp3C.

consider dataset $\mathcal{D}$ is low quality with these three deficiencies. Definition 4 outlines our problem definition.

Definition 4 (Problem Definition). *Given a low-quality dataset $\mathcal{D}$, including massive instances of a relational schema $\mathcal{R}$, data quality rules including a set Φ of* CCs *and a set Σ of* CFDs. *Data cleaning on $\mathcal{D}$ including three tasks:*

1) *To extend the partial currency order to all tuples in $\mathcal{D}$ and compute a comparable currency value for each tuple. The currency value is treated as a kind of relative timestamps which must be in accord with the original orders of* CCs*;*
2) *To detect the satisfaction of CFD set Φ[1] by dataset $\mathcal{D}$, and repair the tuples containing inconsistent attribute values with pattern tuple t_p to make $\mathcal{D}$ satisfy Φ; and*
3) *To impute missing attribute values in tuples with solutions which guarantee the confidence of the imputed attribute value is no smaller than a given threshold $\sigma(A)$.*

2.2 Framework Overview

It is acknowledged that completeness and consistency are metrics focusing on the quality measurement in the *values* of tuples, while currency always focuses on the temporal order of *tuples* in the whole dataset [9]. Thus, we process consistency and completeness repairing along the currency order. We develop Imp3C to serves two purposes: According to the known data quality rules, i.e., *CC*s and *CFD*s, i) each repair operation in Imp3C will not cause any new errors violating 3C issues, and ii) few errors exist on 3C after processing Imp3C. Imp3C achieves an overall data repairing on currency, consistency and completeness, with four main steps, as shown in Fig. 1.

(1) We first construct currency order graphs for tuples with currency constraints, and detect conflicts in the currency graphs. (Section 3).

(2) We determine the currency order in tuples extracted from *CC*s, which is obtained as a direct and unambiguous metric among tuples on currency. We determine the currency value of tuples by iteratively update valid edges and discover the longest currency order chain in the currency graph. We also measure the reliability of the currency order of tuple pairs, in order to present the currency order in tuples more reasonably. This step is discussed in Section 4.

(3) We do inconsistent repair ahead of incompleteness. For consistency repair issues, we input consistency constraints (e.g., *CFD*s) first, and extract potential consistency schema from the original dataset to capture undiscovered consistent tableau. After the consistency schema is determined, we define a metric *cc*-DIST to measure the distance between dirty data and clean ones. We repair errors not only according to the consistency schema, but also take into account the global currency order, i.e., repair the dirty data with proper values with the closest current time point. The process is reported in Section 5.

(4) When detecting the satisfaction of CFD by a tuple, we treat the missing values when matching a tuple with CFD pattern tuples as a kind of violation in consistency. Some missing values can be repaired by the given *CFD*s. It is beneficial to the training accuracy for the later imputation process. In addition, the two-step imputation saves time in the whole data cleaning process. We repair the remaining incomplete values with Bayesian strategy in the final step. We treat the currency order as a weighted feature and train the complete records to fill in the missing values if the filling probability no less than the confidence measure $\sigma(A)$ on A. Detailed steps are presented in Section 6.

The repaired part will cause few new violations on 3C. On one hand, the integrated currency order method ensures that Imp3C provides clean values with the closest currency metrics. On the other hand, the missing values repaired by Σ will not lead to new violations in consistency according to the known consistency constraints. In addition, the computing time cost is decreased in Imp3C. It is because Imp3C prevents some unnecessary repairs with the reasonable repair order between consistency and completeness issues.

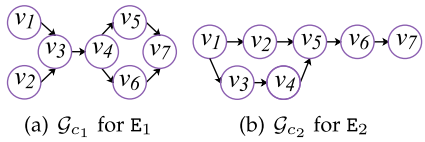


Fig. 2. Currency order graphs for Example 1.

3 Currency Order Graph Construction

We aim to determine the currency order of tuples with *graph*. The currency graph construction is discussed in this section.

We propose the currency graph $\mathcal{G}$ in Definition 5, according to which there is an edge between any two nodes that are temporally related. To make it concise, we use transitive reduction to obtain currency order graph $\mathcal{G}_c$ from $\mathcal{G}$ according to Definition 6.

Definition 5 (Currency Graph). *$r(\mathtt{E}) = \{r_1, \ldots, r_n\}$ denotes n tuples of $\mathcal{R}$. A directed acyclic graph $\mathcal{G} = (V, E)$ is the currency graph of $\mathtt{E}$ in $\mathcal{D}$ if it satisfies:* i) *$\forall v_i \in V, r_i \in r(\mathtt{E}), v_i = r_i$, and* ii) *$\exists e(m,k) \in E$ iff $r_m \prec r_k$ is deduced from Definition 2.*

Definition 6 (Currency Order Graph). *For a given currency graph $\mathcal{G} = (V, E)$, its subgraph $\mathcal{G}_c = (V, E^{\mathsf{TR}})$ is the* transitive reduction *graph of $\mathcal{G}$ iff i) $\mathcal{G}$ and $\mathcal{G}_c$ have the same transitive closure, and ii) $\forall \mathcal{G}_c^{'} \subseteq \mathcal{G}_c$, the transitive closure of $\mathcal{G}_c^{'}$ is different from the one of $\mathcal{G}$.*

Such $\mathcal{G}_c$ is called the currency order graph.

With both Definitions 5 and 6, we let Mike and Helen be two entities $\mathtt{E}_1$ and $\mathtt{E}_2$ in Table 1, and construct the currency order graph for them in Example 2.

Example 2. We deduce from *CC*s in Section 2.1 that $r_2 \prec_{\mathsf{Salary}} r_3 \prec_{\mathsf{Salary}} r_4 \prec_{\mathsf{Salary}} r_5$, $r_6 \prec_{\mathsf{Title}} r_7$ in Fig. 2a. Similarly, with

1. See Definition 2.4 in [10] for satisfaction of *CFD*s by a database.

$r_8 \prec_{\text{Level}} r_9 \prec_{\text{Level}} r_{12}$, $r_8 \prec_{\text{Salary}} r_{10} \prec_{\text{Salary}} r_{11} \prec_{\text{Salary}} r_{12}$, $r_{12} \prec_{\text{Title}} r_{13}$, and $r_{13} \prec_{\text{Level}} r_{14}$, $\mathcal{G}_{c_2}$ is constructed in Fig. 2b.

As currency graphs are constructed, we need to resolve conflicts in currency constraints to achieving accurate and unambiguous currency order. Accordingly, conflicts in *CC*s can be identified by discovering loops in $\mathcal{G}_c$. Conflicts may result from either ambiguous currency constraints or definite currency problems in some tuples. Without credible external knowledge, these conflicts cannot be resolved. As conflicts only happen in a small part of data, they will be returned for artificial process (e.g., repairing by domain experts or assigning crowdsourcing tasks [12]).

4 Currency Order Determination

In this section, we outline the proposed currency value computing method in Section 4.1, and discuss the reliability evaluation about the currency order result in Section 4.2.

4.1 Currency Value Computation Algorithm

Since that *CC*s describe *partial* orders among values on several target attributes, the currency order among some tuples still cannot be deduced. Data without any currency order reasoning from *CC*s is hard to be evaluated on currency. This motives us to determine currency orders of all nodes in $\mathcal{G}_c$ and achieve a total currency order for all tuples.

With the given *CC*s, we can refine the original partial order expressed by the currency order graph $\mathcal{G}_c$ to a liner order. To some extent, the currency order is a kind of replacement of timestamps when the real timestamps are not available. Accordingly, the temporal orders of data is uncovered and the metrics assist data quality resolutions on both consistency and completeness.

The general principle of our solution is to equidistantly assign distinct numbers from the interval (0,1) to all nodes in the currency order graph $\mathcal{G}_c$ in a way that is compatible with the graph structure. These assigned numbers are considered as *currency values* defined on tuples, denoted by *Curr*(v). An intuitive currency value assignment method is to perform topological sorting on the currency order graph $\mathcal{G}_c$ and obtain such values. However, the topological sorting result [13] is not always stable and accurate enough for computing the currency order of all vertices on $\mathcal{G}_c$. On this occasion, we propose a currency order determination method in order to achieve a uniform and accurate currency order determination.

In a currency order graph, currency-comparable tuples of the same entity make up paths. This assists to determine currency values on the graph. We consider any a path in graph $\mathcal{G}_c$ as a *currency order chain*, denoted by $\mathcal{S} = \{v_1, \ldots, v_m\}$, where $\forall\ v_k \in \mathcal{S}, k \in [1, m)$, $v_k \prec v_{k+1}$. Accordingly, an overview of the approach is formulated as follows: i) We first add a global start node *s* that is connected to all graph nodes without incoming edge and a global terminal node *t* to all nodes without outgoing edge linked. Then *s* gets the timestamp 0, and *t* the timestamp 1; ii) Repeatedly, a maximal *currency order chain* without timestamps is selected. Then new timestamps are assigned to these nodes in the way that time differences between adjacent nodes in a chain are equal.

There are one crucial thing should be considered in the method solution. For the equidistant assignment to be feasible, all nodes in the chain must have inherited the same constraints on the range of possible timestamps from their predecessors and successors. Thus, we need to find out the maximal currency order chain $\mathcal{S}_{max}$ with the longest depth on $\mathcal{G}_c$ between two nodes, and then assign currency values $Curr(v)$ to each node in $\mathcal{S}_{max}$.

To compute possible value range of both predecessor and successor of node v, we introduce the *currency order bounds* of v in Definition 7 to describe v's potential $\min$ and $\max$ values with $inf(v)$ and $sup(v)$, respectively. The bounds as well as the position of the node in a currency order chain are important to guarantee the accuracy of the value range of each node. Thus, we define *valid edges* on the graph to determine whether an edge should be selected to form the maximal currency order chain. This phase is discussed in Section 4.1.1.

Definition 7 (The Currency Order Bounds). *When determining currency values, the upper and lower bound of a node v_i in $\mathcal{S} = \{v_1, \ldots, v_m\}$ is defined as:*

- *(a) The* upper currency order bound *of v_i is $sup(v_i) = \min$ Curr($v_{i.}$), where $v_{i.}$ represents any a descendant node connecting from v_i,*
- *(b) The* lower currency order bound *of v_i is $inf(v_i) = \max$ Curr($v_{.i}$), where $v_{.i}$ represents any an ancestor node connecting to v_i.*

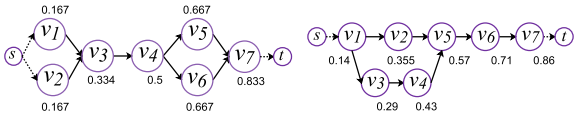
With the bound values, we need to discover the maximal currency order chains between two nodes. It is worth noting that such discovery method is different from simply finding out a longest path on graph. Since that new currency values are assigned according to the values determined previously, we need to discover the maximal currency order chains between two nodes whose currency values are known now. Moreover, we need to guarantee that $\mathcal{S}_{max}$ has correct currency order bounds. We propose the definition about *valid currency order chain*, and then introduce the algorithm of discovering such chains. This part is discussed in Section 4.1.2.

Algorithm 1. CurrValue

Input: the currency graph $\mathcal{G}_c = (V, E)$ of the entity E
Output: $\mathcal{G}_c$ with each $v \in V$ valued with *Curr*(v)

```
1  add s and t to G_c, let s point to all 0 in-degree edges and t
   pointed from all 0 out-degree edges;
2  initialize Curr(s) = sup(s) = inf(s) ← 0, Curr(t) = sup(t) =
   inf(t) ← 1;
3  while ∃ Curr(v_i) has not been determined do
4  copy G_c and obtain its transpose graph G_c^T;
5    Updatebound(G_c^T, sup, <);
6    Updatebound(G_c, inf, >);
7    for e(i, j) ∈ E do
8      label the validation of e(i, j) according to Definition 8;
9    restore the involved nodes;
10   S ← getMaxS(G_c), k ← |S|;
11   Value ← inf(S[1]), Inc ← (sup(S[k]) − inf(S[1])) / (k − 1);
12   foreach h from 2 to k − 1 do
13     Curr(v_h) ← Value + Inc;
14     Value ← Curr(v_h);
15 return G_c;
```

Algorithm 1 shows the whole currency value computing process. We first add the above-mentioned global start and



(a) $\mathcal{G}_{c_1}$ (b) $\mathcal{G}_{c_2}$

Fig. 3. Determine currency values for $\mathcal{G}_{c_1}$ and $\mathcal{G}_{c_2}$.

terminal nodes s and t to $\mathcal{G}_c$ to ensure currency values are located in (0,1). We update currency order bounds of nodes on the graph and discover valid edges (Lines 5-9), in which we record a copy of $\mathcal{G}_c$ and update *sup* of nodes in $\mathcal{G}_c$ with the transpose graph of $\mathcal{G}_c$ first, and then update *inf* on $\mathcal{G}_c$. (see Section 4.1.1 for detail). After that, we find the present longest candidate chain $\mathcal{S}$ in line 10 (see Algorithm 3 for detail), where $k = |\mathcal{S}|$ is the number of elements in $\mathcal{S}$. We assign normalized currency values to each v in $\mathcal{S}$ (Lines 8-11). As the bounds are determined, we use the lower (resp. upper) bound of $inf(\mathcal{S}[1])$ (resp. $sup(\mathcal{S}[k])$) in $\mathcal{S}$ to compute currency values of all elements in $\mathcal{S}$. We iteratively run the above steps until the currency orders on all nodes have been computed.

Example 3. We now compute the currency values in $\mathcal{G}_{c_1}$ and $\mathcal{G}_{c_2}$. We first find out $\mathcal{S}_{max} = \{s, v_1, v_3, v_4, v_5, v_7, t\}$ in Fig. 3a. For each node in $\mathcal{S}_{max}$, $Curr(v_{i+1}) = Curr(v_i) + \frac{sup(v_k) - inf(v_1)}{k-1}$ with $k = 7$ (including s and t). Then, we can easily determine $Curr(v_2) = 0.167$ and $Curr(v_6) = 0.667$. For $\mathcal{G}_{c_2}$ in Fig. 3b, we find $\mathcal{S}_{max} = \{s, v_1, v_3, v_4, v_5, v_6, v_7, t\}$ first and compute $Curr(v_i)$ to be $\{0, 0.14, 0.29, 0.43, 0.57, 0.71, 0.86, 1\}$. After that, only v_2's currency value has not been determined. We use $sup(v_5)$ and $inf(v_1)$ to obtain $Curr(v_2) = 0.355$.

4.1.1 Updating Bounds and Valid Edges

Each chain on the graph reveals the length of a transitive currency relation deduced according to CCs. Not all edges contribute to $\mathcal{S}_{max}$ discovery in $\mathcal{G}_c$ during each iteration. We should determine whether a node v can make up $\mathcal{S}_{max}$ by computing v's currency order bounds during the currency value computing process.

In order to distinguish the present maximal currency order chain $\mathcal{S}_{max}$ during each iteration in Algorithm 1, we need to determine the edges which can form $\mathcal{S}_{max}$. We consider such edges as *valid edges*. We define the validation of edges in Definition 8. The candidate $\mathcal{S}_{max}$ exists in such currency order chains that are formed with valid edges. Thus, $\mathcal{S}_{max}$ can be effectively found according to these valid edges.

Definition 8 (Validation of Edges). *An edge $e(i,j) \in E(\mathcal{G}_c)$ is a valid edge under three cases:* Case 1. *If both $Curr(v_i)$ and $Curr(v_j)$ are not determined, $e(i,j)$ is a valid edge* iff $sup(v_i) = sup(v_j)$ *and* $inf(v_i) = inf(v_j)$. Case 2. *If $Curr(v_i)$ is determined and $Curr(v_j)$ is not, $e(i,j)$ is a valid edge* iff $inf(v_i) = inf(v_j)$. Case 3. *If $Curr(v_j)$ is determined and $Curr(v_i)$ is not, $e(i,j)$ is a valid edge* iff $sup(v_i) = sup(v_j)$.

Note that if both $Curr(v_i)$ and $Curr(v_j)$ are determined, $e(i,j)$ is not a valid edge because v_i and v_j have been already visited in previous iterations. As their currency values have been obtained, $e(i,j)$ is not valid in the present step.

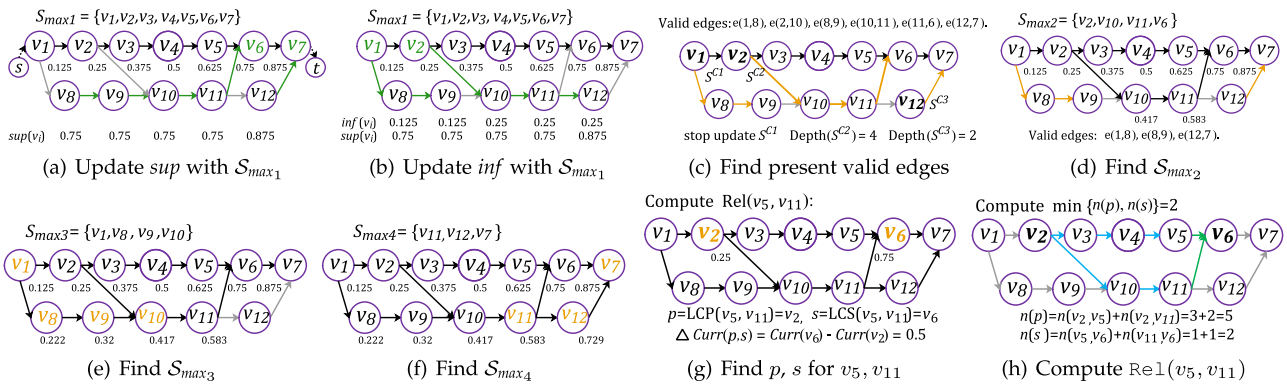
When updating valid edges, we use v_i with its determined $Curr(v_i)$ to update $inf(v_{i^*})$ and $sup(v_{i^*})$ of the nodes reachable from v_i. Both bounds are updated via a function Updatebound. We take *sup* for example in Algorithm 2. We recursively choose v_i with 0 in-degree, enumerate all v_{i^*} and compare *sup* values between v_i and v_{i^*}. If $sup(v_{i^*}) > sup(v_i)$, then $sup(v_{i^*})$ will be updated with the value of $sup(v_i)$. After all $e(i,^*)$ are processed, we temporarily remove v_i from V.

Come back to Algorithm 1 lines 4-6. We execute function Updatebound($\mathcal{G}$, *bound*, *op*) twice to update *sup* and *inf* separately. We restore all nodes on $\mathcal{G}_c$ in line 9 after valid edges have been labelled, and continue the following steps.

Example 4. We discuss another currency graph $\mathcal{G}_{c_3}$ to clearly show how our method works on tuples with a complex currency relation. Fig. 4a shows $\mathcal{G}_{c_3}$'s first longest chain $\mathcal{S}_{max_1}$. With the determined $Curr(v_i)$ $(i \in [1,7])$, we update the bounds of the rest nodes $v_8, \ldots, v_{12}$, and find the next $\mathcal{S}_{max}$. As v_8, v_9, v_{10}, v_{11} all reach v_6 which has the min currency value among descendant nodes of them, $sup(v_8, v_9, v_{10}, v_{11}) = Curr(v_6) = 0.75$. v_{12} only reaches v_7, so $sup(v_{12}) = Curr(v_7) = 0.875$. Similarly, v_1 is reachable from v_8 and v_9, while v_{10}, v_{11}, v_{12} reach v_2 in Fig. 4b. Thus, $inf(v_8, v_9) = Curr(v_1) = 0.125$, $inf(v_{10}, v_{11}, v_{12}) = Curr(v_2) = 0.25$. Since currency values of v_8, v_9, v_{10}, v_{11} are not determined, and v_8, v_9 (resp. v_{10}, v_{11}) have the same bounds. $e(1,8), e(2,10)$, $e(8,9)$ and $e(10,11)$ are valid according to Case 1 in Definition 8. All valid edges are marked in orange in Fig. 4c.

4.1.2 Discovering the Maximal Chain

After the bounds and valid edges are updated (in each iteration), we begin to find out the maximal currency order chain. To guarantee that the output currency value difference between adjacent nodes in the chain are equal, we



(a) Update *sup* with $\mathcal{S}_{max_1}$ (b) Update *inf* with $\mathcal{S}_{max_1}$ (c) Find present valid edges (d) Find $\mathcal{S}_{max_2}$

(e) Find $\mathcal{S}_{max_3}$ (f) Find $\mathcal{S}_{max_4}$ (g) Find p, s for v_5, v_{11} (h) Compute $\mathtt{Rel}(v_5, v_{11})$

Fig. 4. Examples of updating valid edges.

need to discover $\mathcal{S}_{max}$ among the nodes connected by valid edges. We consider such chains as valid currency order chains as defined in Definition 9. And then our task is to find out a valid currency order chain with the longest depth.

Algorithm 2. Updatebound($\mathcal{G}_c$, *bound*, *op*)

Input: $\mathcal{G}_c$, $bound \in \{sup, inf\}$, $op \in \{<, >\}$
Output: the updated $\mathcal{G}_c$ with *bound*
1 **while** $V \neq \emptyset$ **do**
2 **foreach** $e(i,j) \in E$ with 0 in-degree **do**
3 **if** *bound*$[v_i]$ *op bound*$[v_j]$ **then**
4 *bound*$[v_j] \leftarrow$ *bound*$[v_i]$;
5 $V \leftarrow V \backslash v_i$;

Definition 9. *A chain $\mathcal{S} = \{v_1, \ldots, v_m\}$ on $\mathcal{G}_c$ is* a valid currency order chain*, denoted by $\mathcal{S}^c$,* iff *i) Both Curr(v_1) and Curr(v_m) are determined, where v_1 and v_m is the start and terminal element in $\mathcal{S}$, respectively; and ii) $\forall k \in [1, m)$, $e(k, k+1)$ is a valid edge.*

Algorithm 3 finds the maximal valid currency order chain among valid edges, based on the breadth-first search. Here, we perform a topological sorting until all nodes in V have been visited (Lines 3-11). According to Definition 9, $\mathcal{S}^c$ begins with such v_i that *Curr*(v_i) is determined. We begin to update the current chains with the sorting process from line 3. We enumerate all edges beginning from v_i, and update each $\mathcal{S}^c$'s depth with valid edges (Lines 8-10). If we reach any invalid edge, we quit the present chain because it does not satisfy Definition 9 any longer. We finally obtain $\mathcal{S}^c_{max}$ and restore the edges deleted in pervious computing steps.

Example 5. We continue to find the second $\mathcal{S}^c_{max}$ in $\mathcal{G}_{c_3}$ beginning with v_1, v_2, and v_{12} and let them be $\mathcal{S}^c_1$, $\mathcal{S}^c_2$, and $\mathcal{S}^c_3$, respectively in Fig. 4c. We update $\mathcal{S}^c_1$ with $e(1,8)$, $e(8,9)$. It is no longer candidate when it reaches v_{10}, for $e(9,10)$ is not valid. We quit $\mathcal{S}^c_1$, and obtain Depth($\mathcal{S}^c_2$) $= 4$ as it finally reaches v_6 and Depth($\mathcal{S}^c_3$) $= 2$. We obtain the present $\mathcal{S}^c_{max_2} = \mathcal{S}^c_2$ in Fig. 4d, and compute $Curr(v_{10}) = 0.417$, $Curr(v_{11}) = 0.583$ according to the bounds of v_2 and v_6 with Algorithm 1. We iteratively run the above steps, and find $\mathcal{S}^c_{max_3} = \{v_1, v_8, v_9, v_{10}\}$ in Fig. 4e, thus, $Curr(v_8)$ and $Curr(v_9)$ and determined. Finally, we find $\mathcal{S}^c_{max_4} = \{v_{11}, v_{12}, v_7\}$ and obtain $Curr(v_{12}) = 0.729$. The currency value determining on $\mathcal{G}_{c_3}$ is finished. Thus, $Curr(v_8) = 0.222$ and $Curr(v_9) = 0.32$. Finally, we get $Curr(v_{12}) = 0.729$ and the currency value determining of $\mathcal{G}_{c_3}$ is finished.

Complexity. It takes $O(|V| + |E|)$ times to update bounds of nodes and valid edges with Updatebound in lines 4-8 within the inner loop in Algorithm 1. Algorithm 3 takes $O(|V| + |E|)$ in total to find out $\mathcal{S}_{max}$. The outer loop (lines 3-14) in Algorithm 1 costs $O(|V|)$ for the worst time. Thus, Algorithm 1 takes $O\big(|V| \cdot (|V| + |E|)\big)$ in total.

We finish computing currency value of each tuple in $\mathcal{D}$ after the above process. Theorem 1 shows that the proposed method can always correctly refine the original partial order to a liner order on the whole graph for tuples. Now, the original $\mathcal{D}$ is kept in right currency order in accordance with the given *CC*s, and the currency between each record pair is easy to be obtained.

Theorem 1. *Given* E*'s tuples $r(\mathtt{E}) = \{r_1, \ldots, r_n\}$ in $\mathcal{D}$ and a set of CCs for $r(\mathtt{E})$. $\forall\ r_i \prec r_j$ $(i, j \in [1, n])$, then we always have $Curr(r_i) < Curr(r_j)$ after Algorithm 1.*

Algorithm 3. getMaxS

Input: $\mathcal{G}_c$
Output:the maximal chain $\mathcal{S}^c_{max}$ in $\mathcal{G}_c$
1 initialize Depth[] $\leftarrow 0$, *endDepth* $\leftarrow 0$;
2 initialize *pre*[] $\leftarrow$ *Null*, *endPoint* $\leftarrow$ *Null*;
3 **while** $\exists v_i (v_i \in V)$ *with 0 in-degree* **do**
4 **if** *Curr(v_i) is determined* **then**
5 **if** Depth$[v_i] >$ *endDepth* **then**
6 *endDepth* $\leftarrow$ Depth$[v_i]$, *endPoint* $\leftarrow v_i$;
7 Depth$[v_i] \leftarrow 0$;
8 **foreach** $e(i,k) \in E$ **do**
9 **if** $e(i,k)$ *is a valid edge and* Depth$[v_i]$+1 $>$ Depth$[v_k]$ **then**
10 Depth$[v_k] \leftarrow$ Depth$[v_i]$+1, *pre*$[v_k] \leftarrow v_i$;
11 $V \longleftarrow V \backslash v_i$;
12 $\mathcal{S}_{max} \leftarrow$ select the maximal $\mathcal{S}^c$ with *endPoint* and *pre*[];
13 restore all v_i in V;
14 **return** $\mathcal{S}_{max}$;

Proof of Theorem 1 For a tuple pair r_i, r_j from $r(\mathtt{E}) = \{r_1, \ldots, r_n\}$, *w.l.o.g.*, assume that r_j is more current than r_i deduced from *CC*s. We now only need to prove that our method can always compute $Curr(r_i) < Curr(r_j)$.

Since that r_i is less current than r_j, we have $r_i \prec r_j$. According to Definition 5, the two tuples belong to different node in the currency order graph $\mathcal{G}_c = (V, E)$. Might as well let r_i and r_j belong to v_i and v_j in $V(\mathcal{G}_c)$, respectively. There exists edges between v_i and v_j, where v_j is reachable from v_i. Now, we only need to prove *Curr*(v_i) $<$ *Curr*(v_j). There exists two possible cases as discussed below.

Case 1. When v_i and v_j are in the same currency order chain $\mathcal{S}$, (e.g., v_{10} and v_{11} in $\mathcal{S}^c_{max2}$ in Fig. 4d), then *Curr* (v_j) = *Curr*(v_i) + Value, where Value is the increment value discussed in Algorithm 1. Since Value > 0, *Curr* (v_i) $<$ *Curr*(v_j) is true.

Case 2. When v_i and v_j are not in the same currency order chain, 1) if the currency value of v_i is determined ahead of v_j's, (e.g., v_{10} and v_{12} in Figs. 4d, 4e, and 4f), then *Curr*(v_i) $< inf(v_j)$. According to Definition 7, *Curr* (v_i) $< inf(v_j) \leq$ *Curr*(v_j). Thus, *Curr*(v_i) $<$ *Curr*(v_j) is true; 2) If the currency value of v_i is determined after v_j's, (e.g., v_8 and v_{10} in Figs. 4d and 4e), then *Curr*(v_j) $>$ *sup* (v_i) $\geq$ *Curr*(v_i). Thus, *Curr*(v_i) $<$ *Curr*(v_j) is true.

Above all, the conclusion in Theorem 1 is proved. □

4.2 The Reliability of Currency Order Pairs

Up to now, any nodes pair (v_i, v_j)'s currency order can be achieved by computing the difference between *Curr*(v_i) and *Curr*(v_j). If $v_i \prec v_j$ is deduced by *CC*s, there exists an edge $e(i,j)$. It means that the currency order between them is definite and reliable. However, things are not the same for the disconnected nodes, e.g., v_2 and v_3 in $\mathcal{G}_{c_2}$. The currency order between v_2 and v_3 is computed by Algorithm 1, rather than directly derived by *CC*s. Though their currency order is comparable by computing the currency value difference $Curr(v_i, v_j) = |Curr(v_i) - Curr(v_j)|$ now, this currency order is not as believable as the pair (v_1, v_2) be. Thus, we address a

metric to measure different *reliability* degree of the currency order of any two nodes in a currency order graph.

When comparing $Curr(v_i)$ and $Curr(v_j)$ in $\mathcal{G}_c$, the reliability of the currency order pair (v_i, v_j) reaches the top degree when v_j is reachable from v_i in $\mathcal{G}_c$. We focus on the reliability degree measurement in the case that v_j is not reachable from v_i. Since v_i and v_j are disconnected, we address the lowest common ancestor (LCA) in Defintion 10 in our reliability metric according to [14]. Accordingly, we can easily discover the lowest common ancestor of nodes v_i and v_j in $\mathcal{G}_c$ with algorithms in [14] as well as the lowest common successors (LCS) by discovering LCA on the transposed graph of $\mathcal{G}_c$.

Definition 10. *Given two nodes u and v in a DAG (i.e., $\mathcal{G}_c$), p is* the lowest common ancestor *of nodes u and v iff p is an ancestor of both u and v where p has no descendants that are also ancestors of both u and v.*

$\text{SLCA}(u, v, \mathcal{G}) = \{p_1, \ldots, p_n\}$ is the set of all the lowest common ancestors of u and v on $\mathcal{G}$.

Inspired by the Hop-distance in wireless sensor networks (WSN) [15] and graph reliability evaluation with topology-based methodology [16], [17], the reliability degree is influenced by two factors: 1) The *range of currency values* from (v_i, v_j)'s LCA p to LCS s, i.e., $Curr(p, s)$. It measures the tightness degree of the possible currency order of (v_i, v_j). Considering some worse cases that $Curr(p, s)$ is too large, the currency values assigned to the nodes between p and s become not so reliable; and 2) the total distance of the path from a LCA p to v_i concatenated with the path from p to v_j. Referring to topology structure evaluation in WSN, if v_i is closer to it LCA with v_j, then the currency value on v_i have higher reliability compared with nodes far from LCA in the same path with v_i. Since that the general definition of LCA does not consider distances bewteen p and nodes v_i and v_j, we need to find out such a LCA that has the nearest distance with the two nodes. We first define $\text{Dist}(p)$ in Definition 11 to find the required LCA (called d-LCA).

Definition 11. *Given $\text{SLCA}(u, v, \mathcal{G})$ of nodes u and v, $\text{Dist}(p)$ measures the path from a LCA p to u concatenated with the path from p to v, i.e., $\text{Dist}(p) = depth(p, u) + depth(p, v)$. p is a d-LCA of nodes u and v on $\mathcal{G}$ iff $p = \arg\min_{p_i \in \text{SLCA}} \text{Dist}(p_i)$.*

Considering the both factors, the reliability degree is inversely proportional to $Curr(p, s)$ and $\text{Dist}(p)$ (or $\text{Dist}(s)$), respectively. We compute the reliability of currency order pairs according to Definition 12.[2] We select $\min\{\text{Dist}(p), \text{Dist}(s)\}$ in $\texttt{Rel}(v_i, v_j)$ computation in accordance with the foresaid nearest distance principle. The value domain of $\texttt{Rel}(v_i, v_j)$ is (0,1], with $Curr(p, s) \in (0, 1)$ and $\min\{\text{Dist}(p), \text{Dist}(s)\}$ no less than 2.

Definition 12. *The* reliability *of the currency order between u and v on $\mathcal{G}_c$ is defined as*

$$\texttt{Rel}(u, v) = \begin{cases} 1, & v \text{ is reachable from } u. \\ \frac{2^{1-Curr(p,s)}}{\min\{\text{Dist}(p), \text{Dist}(s)\}}, & \text{otherwise.} \end{cases}$$

where p is a d-LCA of (u, v), and s is a d-LCS of (u, v).

2. The currency order graph is not as complicated as WSN. Definition 12 is a heuristic reliability function which presents effectiveness in the experimental results.

Algorithm 4 shows the reliability computing process. For the disconnected nodes u and v, we find out all d-LCAs on $\mathcal{G}_c$. We then select p^* with the max currency value according to the tightness range principle. Similarly, we execute SLCA function on $\mathcal{G}_c^T$ and select s^* with the min currency value (Lines 7-9). Algorithm 4 finishes after computing $\texttt{Rel}(u, v)$ according to Definition 12.

Algorithm 4. $\texttt{Rel}\ (u, v)$

Input: u, v on $\mathcal{G}_c$
Output: $\texttt{Rel}(u, v)$
1 **if** *v is reachable from u on $\mathcal{G}_c$ or vice versa* **then**
2 $\texttt{Rel}(u, v) = 1$;
3 **else**
4 $\text{SLCA}_d \leftarrow$ find all d-LCAs from $\text{SLCA}(u, v, \mathcal{G}_c)$;
5 **foreach** $p_i \in \text{SLCA}_d$ **do**
6 find $p^* = \arg\max Curr(p_i)$;
7 $\text{SLCS}_d \leftarrow$ find all d-LCAs from $\text{SLCA}(u, v, \mathcal{G}_c^T)$;
8 **foreach** $s_j \in \text{SLCS}_d$ **do**
9 find $s^* = \arg\min Curr(s_j)$;
10 compute $\texttt{Rel}(u, v)$ with p^* and s^*;

Example 6. The reliability of (v_5, v_{11}) on $\mathcal{G}_{c_3}$ is computed in Fig. 4g. We get $Curr(p, s) = 0.5$ with $\text{LCA}(v_5, v_{11}, \mathcal{G}_{c_3}) = v_2$ and $\text{LCS}(v_5, v_{11}, \mathcal{G}_{c_3}) = v_6$. We obtain $\min\{\text{Dist}(p), \text{Dist}(s)\} = 2$ with $\text{Dist}(p)$ marked in blue and $\text{Dist}(s)$ in green in Fig. 4h. Thus, $\texttt{Rel}(v_5, v_{11}) = \frac{2^{1-0.5}}{2} = 0.707$.

After currency orders are determined, we further repair the inconsistent and incomplete data. To realize less violation on both dimensions after the whole repairing, we address the inconsistency issues ahead of the incompleteness.

5 Inconsistency Repair With Currency

Due to the increasing importance of currency issues in temporal data, currency orders contribute to improving inconsistent data repair quality. We aim to provide high-quality inconsistent repair solutions leveraging currency-consistency combined edit distance measurement.

High-quality *CFD*s are not easily to be either manually designed or automatically discovered. In the third step of Imp3C, we could use the methods proposed in [18], [19] to automatically discover *CFD*s to find out reasonable pattern tableaux not described by given *CFD*s. Below, we propose a function about currency-consistency combined distance in Section 5.1, and introduce the inconsistent repair algorithm ImpCCons in Section 5.2.

5.1 Combined Edit Distance

One common solution to repairing an inconsistent dirty tuple r is to find out *CFD*s violated by r, compute the edit distance between dirty attribute values and cleaned tuple patterns, and then choose a possible repair pattern according to data cleaning principle (e.g., cost minimality principles [20]). For inconsistent data repairing tasks on temporal datasets, simply using minimal edit distance with *CFD*s is not enough. In this section, we propose a combined distance function in order to obtain an effective repair pattern considering the distance of currency orders.

Consistency Distance. We first present the edit distance function on consistency between a dirty tuple r and a CFD pattern tuple φ in Equation (1). Here, $dist_b$ is a binary distance that $dist_b(i,j) = 1$ if $i = j$, and $dist_b(i,j) = 0$, otherwise. It needs to be mentioned that we set $dist_b(i,j) = 1$ when i is a null value on an attribute. That is, null values do not equal to any other values in consistency distance measurement. $|\text{LHS}(\varphi)|$ (resp. $|\text{RHS}(\varphi)|$) is the number of attributes in LHS(φ) (resp. RHS(φ)). We measure the consistency distance as the ratio of the number of attributes violating a given φ. This distance principle is generally adopted in records distance measurement [10]

$$co\text{-Dist}(r,\varphi) = \frac{\sum_{A_i \in \text{-HS}} dist_b(r[A_i], \varphi[A_i])}{|\text{LHS}(\varphi)| + |\text{RHS}(\varphi)|}. \quad (1)$$

Currency Distance. We introduce the currency distance between two tuples x and y in Equation (2), with the currency value computation method discussed in Section 4

$$cu\text{-Dist}(x,y) = \frac{Curr(x,y)}{\texttt{Rel}(x,y)+1}, \quad (2)$$

where $Curr(x,y) = |Curr(x) - Curr(y)|$. In reality, there may exist several possible value modification solutions for a dirty tuple with inconsistent attributes. Our repairing principle is the minimality of a consistency-currency combined edit distance. We generally present the computation of the consistency-currency distance *cc*-Dist in Definition 13, according to which we repair the inconsistent attribute values.

Algorithm 5. ImpCCons

Input: tuple r_i in $\mathcal{D}$, Σ, $dist$, μ
Output:tuple r_i after repair

```
1  foreach Σ_{T_p} onto attributes X, Y in Σ do
2    if r_i ⊨ Σ_{T_p} then
3      continue;
4    Vio[]← copy all φs that r_i violates;
5    initialize Vio[].dist ← dist;
6    find neighbor tuples {r_L, r_{L+1}, ..., r_R} from r_i's G_c with
     Curr(r_i) ± dist;
7    for j from L to R do
8      extract pattern p used in r_j w.r.t. X and Y;
9      if p ∉ Vio then
10       insert p into Vio;
11       initialize Vio[p].dist ← dist;
12     Vio[p].dist ← min{Vio[p].dist, cu-Dist(r_i, r_j)};
13   minDist ← +∞, φ* ← null;
14   foreach φ ∈ Vio do
15     cc-Dist(r_i, φ) ← μ · co-Dist(r_i, φ) + (1 − μ) · Vio[φ].dist;
16     if cc-Dist < minDist then
17       minDist ← cc-Dist, φ* ← φ;
18   repair r_i with φ*;
19 return r_i;
```

Definition 13 (cc-Dist). *The currency-consistency distance between tuples x and y is denoted by*

$$cc\text{-Dist}(x,y) = \mu \cdot co\text{-Dist}(x,y) + \nu \cdot cu\text{-Dist}(x,y), \quad (3)$$

where μ, ν are weight coefficients in (0,1), and $\mu + \nu = 1$.

Further, we aim to select a repair solution not only consistent to *CFD*s, but also be close to the dirty tuple r in currency. We will introduce our method ImpCCons in Section 5.2 below to present how to *imp*rove repair in*con*sistent attribute values leveraging *c*urrency orders.

5.2 Algorithm ImpCCons

The general idea in ImpCCons is to find candidate repair patterns from clean tuples first and then determine repair solution according to distance minimality principle. In reality, it is hard to discuss the currency distance between a pattern tuple φ and a tuple r with Equation (3). In this case, we find out r's neighbor tuples according to the currency distance with a distance threshold $dist$, and extract candidate repair pattern φs from these neighbor tuples. The currency distance between φ and r is updated with *cu*-Dist of r and r's neighbor tuple.

Algorithm 5 presents the repair process of one tuple. We denote Σ_{T_p} as the set of *CFD*s: $(X \rightarrow Y, T_p)$. Generally, we enumerate the set Σ_{T_p} with the same tableau T_p in attribute X and Y, and detect the inconsistencies in tuples *w.r.t.* Σ_{T_p}. Note that we follow the rule that, given two set $\Sigma_{T_{p1}}$ and $\Sigma_{T_{p2}}$, we have $A_1 \cap A_2 = \emptyset$, where $A_i \in X_i \cup Y_i, (i \in \{1,2\})$.

In the outer loop lines 1-18, we detect whether tuple r_i satisfies one kind of *CFD*s, i.e., Σ_{T_p}. When violation of Σ_{T_p} happens in r_i, we begin the repair process in line 4. We record all pattern tuples, i.e., φs, violated by r_i in *Vio*[]. Here, *Vio*[] serves as a dictionary, and records possible repair patterns for r_i *w.r.t* Σ_{T_p}. We set an initial currency distance value of each element in *Vio*, and then find out r_i's neighbor tuples with a given currency distance threshold $dist$. We enumerate each neighbor tuple r_j and obtain the pattern p on attribute X and Y in r_j. It is possible that p does not exists in *Vio*. In this case, we insert p to the dictionary *Vio* and set initial value for $Vio[p].dist$ (Lines 9-11). In line 12, we update $Vio[p].dist$ if the current $Vio[p].dist$ is smaller than $dist$, in order to record the *nearest* appearance of each candidate repair pattern with r_i. After updating *cu*-Dist of all elements in *Vio*, we compute the proposed currency-consistency distance between φ and r_i with an input threshold μ. We finally decide φ^* to be the repair pattern for r_i with the minimum *cc*-Dist.

Example 7. We now repair tuple r_5 in Table 1 with Algorithm 5. We detect that r_5 violates φ_2 and φ_3 as mentioned in Section 2.1. With a given $dist = 0.2$, we initialize $Vio[\varphi_2].dist = Vio[\varphi_3].dist = 0.2$. r_4 and r_6 are selected as r_5's neighbor tuples. We record the pattern tuples of r_4 and r_6 as $p(r_4)$ and $p(r_6)$, respectively. We get $cu\text{-Dist}(r_5, p(r_4)) = 0.167$, $cu\text{-Dist}(r_5, p(r_6)) = 0$. We compute $cc\text{-Dist}(r_5, p(r_4)) = 0.4 \cdot \frac{2}{3} + (1-0.4) \cdot \frac{0.167}{2} = 0.32$, and $cc\text{-Dist}(r_5, p(r_6)) = 0.27$. Thus, we repair r_5 according to $p(r_6)$: $r_5[\text{City}] = Hangzhou$. Similarly, errors in r_{10} can be also captured by Algorithm 5 and repaired with $r_{10}[\text{City}] = Shanghai$.

Complexity. Given dataset $\mathcal{D}$ and a set Σ of *CFD*s, it takes $O(|\Sigma| \cdot N)$ time to detect the violation of Σ, where $|\Sigma|$ is the total number of φ, and N is the total number of tuples in $\mathcal{D}$. For the whole repairing process, it spends $O(K \cdot (N_d + N_{vio}))$ time on average, where K is the total number of inconsistent pattern tuples detected by Σ, N_d is the number of neighbor tuples selected by $dist$, and N_{vio} is the number of *CFD* pattern

TABLE 3
Analogy Between the Incompleteness Repair and Naïve Bayes

An incomplete tuple, $r_{\overline{\mathbf{C}}} = \{a_1, \ldots, a_{m-1}, Curr(r_{\overline{\mathbf{C}}})\}$	$\rightarrow X = \{a_1, \ldots, a_m\}$
The domain of the missing values, $dom(A^{\bar{c}}) = \{z_1, \ldots, z_n\}$	$\rightarrow Y = \{y_1, \ldots, y_n\}$
The prior probability, $\mathsf{Pr}(A^{\bar{c}})$	$\rightarrow \mathsf{Pr}(Y)$
The class-conditional probability, $\mathsf{Pr}(r_{\overline{\mathbf{C}}}\|A^{\bar{c}})$	$\rightarrow \mathsf{Pr}(X\|Y)$
The filling posterior probability, $\mathsf{Pr}(z_i\|r_{\overline{\mathbf{C}}})$	$\rightarrow \mathsf{Pr}(y_i\|X)$

TABLE 4
Summary of the Datasets

Data	#Tuples	#Entities	#Tables	#Attributes
NBA	25,050	1,050	5	10
HOSP	22,500	3,200	3	16
EReg	50,720	870	3	22

tuples violated by each tuple. To put it together, inconsistency repairing process on dataset $\mathcal{D}$ costs $O(|\Sigma| \cdot\ N + K \cdot (N_d + N_{vio}))$ in time.

Specially, Algorithm 5 performs on the assumption that there is no conflict between the given *CFD*s and *CC*s. Works have been done (like [21]) on conflict resolution in *CFD*s and *CC*s, which are applied in the preprocess of our method.

6 Incompleteness Repair

Repairing missing values is one classical key problem in data completeness solutions [22]. We highlight that our method processes two steps for missing values imputation. In the former step, the missing attribute values in tuples detected by Σ are treated as a kind of violation in data consistency. That is, we consider $r[A] \not\approx t_p[A]$ when $r[A]$ has null values ($A \in attr(\mathcal{R})$).[3] In this case, we are able to repair such missing values by Algorithm 5 (e.g., r_{10}'s repair with CFD : ([Comp], [Group] $\rightarrow$ [City])), ahead of the incompleteness repair step.

In the latter step, we adopt Naïve Bayes method [23] to repair the missing values which cannot be captured by CFD pattern tuples. Bayes method is acknowledged to perform well in missing values imputation issues. In general, we treat $Curr(r_i)$ as an important feature and insert it to the training process, and improve missing values imputation by filling the missing part with time-related clean values.

Table 3 draws the analogy between our repair approach and the general elements in Naïve Bayes. For an incomplete record $r_{\overline{\mathbf{C}}} = \{a_1, \ldots, a_{m-1}, Curr(r_{\overline{\mathbf{C}}})\}$, a_j is the value on A_j and $r_{\overline{\mathbf{C}}}$'s missing value is on $A^{\bar{c}}$, (*w.l.o.g.*, assuming that $r_{\overline{\mathbf{C}}}$ only contains one missing value whose value domain is $\{z_1, \ldots, z_n\}$). We train the complete tuples to compute the possible value for the test data like $r_{\overline{\mathbf{C}}}$. $\mathsf{Pr}(A^{\bar{c}})$ and $\mathsf{Pr}(r_{\overline{\mathbf{C}}}|A^{\bar{c}})$ are adopted to address missing values imputation. Accordingly, to classify and repair an incomplete tuple, the Naïve Bayes computes the posterior probability for each complete tuple according to Equation (4)

$$\mathsf{Pr}(z_i|r_{\overline{\mathbf{C}}}) = \frac{\mathsf{Pr}(z_i) \cdot \left(\prod_{j=1}^{m-1} \mathsf{Pr}(a_j|z_i) \cdot \mathsf{Pr}(Curr(r_{\overline{\mathbf{C}}})|z_i)\right)}{\mathsf{Pr}(r_{\overline{\mathbf{C}}})}. \quad (4)$$

Accordingly, the completeness repairing issue named ImpCCom, is solved by the following steps.

Step 1: Input data $\mathcal{D}_{cons}$ and the confidence threshold $\sigma(A)$ for each attribute in $attr(\mathcal{R})$. Treat the currency values $Curr(r)$ as a new attribute, and insert $Curr(r)$ to each tuple.

Step 2: Detect the tuples which contain missing values i.e., $r_{\overline{\mathbf{C}}}$s. We treat the set of $r_{\overline{\mathbf{C}}}$s as the test data. Construct the training set with the complete tuples in $\mathcal{D}_{cons}$.

Step 3: Preprocess training data and discretize continuous variables in $attr(\mathcal{R})$ and currency values into intervals with (dummy) categorical variables. Concretely, create a contingency table for a continuous variable A and dummy variables B and do chi-squared test on A and B. Divide A into intervals represented by B according to the max chi-squared statistic.

Step 4: Compute $\mathsf{Pr}(z_1|r_{\overline{\mathbf{C}}}), \mathsf{Pr}(z_2|r_{\overline{\mathbf{C}}}), \ldots, \mathsf{Pr}(z_n|r_{\overline{\mathbf{C}}})$ for the missing value on attribute $A^{\bar{c}}$ in the test data.

Step 5: Find $\mathsf{Pr}(z|r_{\overline{\mathbf{C}}}) = \max_{k\in[1,n]} \mathsf{Pr}(z_k|r_{\overline{\mathbf{C}}})$, and fill the missing value of $r_{\overline{\mathbf{C}}}$ on $A^{\bar{c}}$ with z if $\mathsf{Pr}(z|r_{\overline{\mathbf{C}}}) \geq \sigma(A^{\bar{c}})$.

Step 6: Recursively repeat Step 4 and Step 5 until the missing values on all attributes have been solved.

The currency value of tuples helps train the model to repair missing part with the data shares the same (or close) current order with $r_{\overline{\mathbf{C}}}$. Bayes model is a proper instance in our method which performs well in experiments. The currency values can be used in other alternative imputation strategies such as Regression models [7] and Decisions trees [24] in Imp3C in view of the characteristic of data.

7 Experimental Study

We now evaluate the experimental study of the proposed methods. All experiments run on a computer with 3.40 GHz Core i7 CPU and 32 GB RAM.

7.1 Experimental Settings

Experimental Data. To report the generality of the proposed method, three real-life datasets are used in experiments, where attribute values change temporally in different degrees. Table 4 summaries the datesets.

NBA. This statistics data[4] reports NBA players' career information. We integrate five tables and create a dataset collecting tuples for over 1,000 players with more than 4 regular seasons with attributes: (Pid, Name, Age, Nationality, Team, Arena, City, Season, PPG, Scores). Pid is used to identify different players, and the data describes which Team players belongs to at the corresponding Season. It records the home arena of each team in Arena and City. PPG is the averaged points the player achieves pre game in each season, and Score records players' total score. Values on attributes such as Scores and PPG evolve frequently over regular seasons. The samples of *CC*s and *CFD*s are as follows.

$$\psi_1: \quad \forall r_i, r_j, (r_i[\mathsf{Pid}] = r_j[\mathsf{Pid}] \text{ and } r_i[\mathsf{Age}] < r_j[\mathsf{Age}]) \rightarrow (r_i \prec_{\mathsf{Age}} r_j).$$

$$\psi_2: \quad \forall r_i, r_j, (r_i[\mathsf{Pid}] = r_j[\mathsf{Pid}] \text{ and } r_i[\mathsf{Scores}] < r_j[\mathsf{Scores}]) \rightarrow (r_i \prec_{\mathsf{Scores}} r_j).$$

$$\Sigma_\varphi: \quad \forall r_i, ([\mathsf{Season}], [\mathsf{Team}]) \rightarrow ([\mathsf{Arena}], [\mathsf{City}]).$$

3. The operator $\approx$ is defined on constants and extends to tuples. $\not\approx$ is the negative form of $\approx$.

4. http://www.basketball-reference.com.

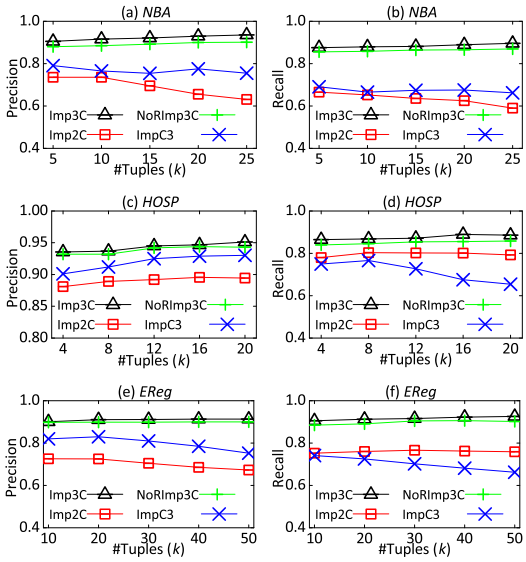


Fig. 5. Effectiveness on error repairing in three datasets.

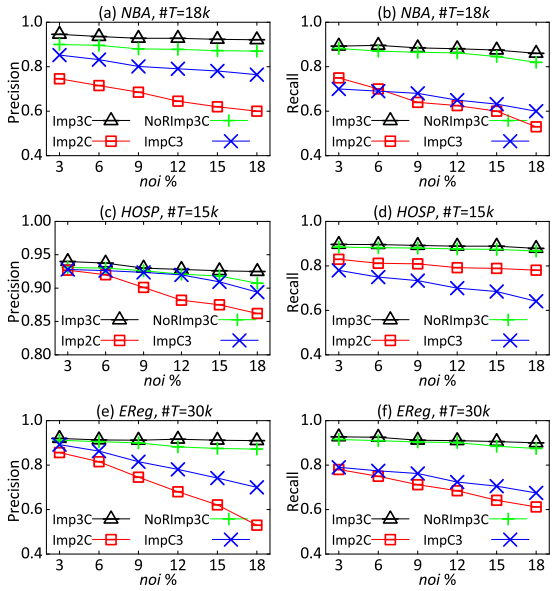


Fig. 6. Noise tolerance on three datasets.

HOSP is taken from from American Hospital Compare Database,[5] which is a general public dataset in data cleaning literatures [6], [25]. We create experiment data from *HOSP* with 20k tuples with 16 attributes (e.g., Name, Addr, City, ZIPcode, MeasureID, MeasureName, Score, etc).

EReg. The Enterprise Registration Data[6] for a province in China records companies information for over 10 years. We use 50k tuples with 22 attributes describing Addr, Province, ProductType, Registertime, Taxinfo, etc. We manually adopt 30 *CFD*s and 12 *CC*s.

Implementation. We set $\sigma(A) = 0.9$ and $\mu = 0.6$ for the proposed Imp3C. And the default value for $dist$ is 0.2, which is a proper parameter values obtained from lots of experiments. Besides Imp3C, we implement another three algorithms for comparative evaluation:

- NoRImp3C, measures currency distance without computing REL in Section 4.2, e.g., let REL=1 always;
- ImpC3, executes ImpCCom first, and then ImpCCons;
- Imp2C, executes consistency repair and missing imputation with the methods discussed in Section 5 without computing currency orders.

In preprocessing step, we discover both *CC*s and *CFD*s according to methods proposed in [19], [26]. We find *ground truth*[7, 8] for datasets and prepare high-quality datasets with clean values. In addition, both *CC*s and *CFD*s applied in each dataset cover over 70 percent tuples of the whole data.

We generate dirty data in dataset $\mathcal{D}$ to make it become low-quality as described in Section 2.1. Error values i.e., noises are added randomly to tuples of entities. *noi* is used to describe the noise ratio of the erroneous values to the total number of values. The error rate of the inconsistent and missing values in each attribute are equal.

Measure. We apply precision (P) and recall (R) to measure algorithms effectiveness. P is the ratio between the number of values correctly repaired and the total number of repaired values. R is the ratio between the number of values correctly repaired and the total number of dirty values.

5. https://data.medicare.gov/data/hospital-compare
6. For privacy issues, the dataset is not publicly accessible.
7. http://en.wikipedia.org/wiki/List_of_National_Basketball_Association_arenas
8. https://tools.usps.com/zip-code-lookup.htm

7.2 Effectiveness

We discuss algorithms performance on two parameters, namely total number of tuples #Tuples (N) and the noise rate *noi*. We implement the aforesaid algorithms under the same condition correspondingly on the three datasets.

7.2.1 Effectiveness on Error Repairing

We evaluate algorithm effectiveness of repairing errors within the three datasets with P and R, as shown in Fig. 5. As the total number of tuples i.e., #Tuples increases, the performance of both Imp3C and NoRImp3C are stable and show an upward trend, while Imp2C and ImpC3 report a poor performance. Generally, ImpC3 outperforms Imp2C in P, for it considers the currency order of tuples. However, there is an apparent decline in R of ImpC3, because ImpC3 fills missing values first and then repairs inconsistent ones. It trains the data with many uncovered inconsistent values in its missing values imputation step, and this leads to the fail in R with the growth of #Tuples. Figs. 5a and 5e show that Imp3C and NoRImp3C perform quite better than Imp2C and ImpC3, for the reason that attributes' value changes frequently in *NBA* and *EReg*. It verifies that the currency order determination in our proposed method does improve the data repairing quality. In addition, Imp3C always performs a little better than NoRImp3C on both P and R. It indicates that determining the reliability of currency orders is effective for improving the repair accuracy.

7.2.2 Effectiveness on Tolerance With Error Rate

Fig. 6 reports algorithms performance with *noi* varies from 3 to 18 percent. It shows that both P and R of Imp3C stand stable when *noi* increases. Imp3C outperforms other three algorithms. The repair quality of NoRImp3C is tightly lower than Imp3C, but obviously better than Imp2C and ImpC3.

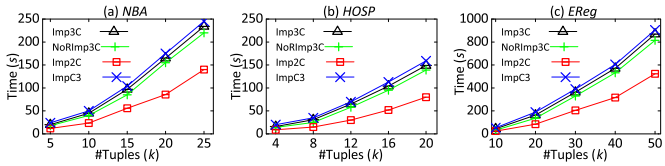


Fig. 7. Efficiency results with $noi = 10\%$.

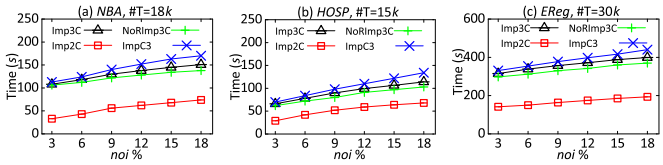


Fig. 8. Efficiency: varying noi.

Fig. 6c shows all algorithms' P reach 0.9 with $noi \leq 9\%$. It is because the attribute values in *HOSP* do not change frequently. However, Imp3C and Imp2C fail to repair the dirty data well when the data contains more errors. For *NBA* and *EReg*, tuples of one entity change frequently, which makes it necessary to determine currency order among them. For NoRImp2C, without considering the reliability of currency, it drops in both P and R when $noi \geq 12\%$. The repair quality of Imp3C is quite stable, for it continues to capture the errors exactly and repair them accurately with currency measures when noi increases. The proposed Imp3C achieves high data repair performance. Not only the currency determination improves the repair quality, but also the processing order between inconsistent and incomplete values has a distinct influence in the repairing effectiveness.

7.3 Efficiency

Fig. 7 shows the time costs with varying #Tuples. Imp3C costs less time than ImpC3, because Imp3C prevents more unnecessary repairs with the proper repair orders than ImpC3 does. Imp3C trends to reduce more repair times than ImpC3 with N grows. Imp3C just costs a little more time than NoRImp3C, which shows the efficiency in the reliability computing for the involved currency order pairs. Imp2C has the least time costs without computing tuples' currency order, and it costs about 60 percent of the time in Imp3C. It takes less time in *HOSP* than in *NBA* and *EReg*, because the structure of currency graphs is simple in *HOSP*, so it takes less time in currency order computing (Algorithm 1). Fig. 7c shows Imp3C repairs 50k tuples in 15 minutes in *EReg*. It verifies the scalability of our method on the data with many attributes and potential complex currency orders.

Fig. 8 shows the time costs with the varying error rate (#Tuples=18k for *NBA*, #Tuples=15k for *HOSP* and #Tuples=30k for *EReg*). Imp3C runs faster than ImpC3 in all the datasets, and the time cost gap between them becomes larger when *noi* increases, as expected. It reveals that Imp3C is more efficient than ImpC3, and Imp3C performs better when repairing data with a higher error rate. Imp3C needs a little more time than NoRImp3C, for Imp3C computes the reliability of currency order pairs in *cc*-DIST to improve the repair quality. The growing speed of Imp2C is as much as Imp3C's, showing the error rate does not affect the efficiency of computing currency graphs and determining currency orders. Fig. 8c shows that Imp3C repairs 18 percent noises in 30k tuples of *EReg* in 6.67 minutes, which shows a potential in scalability for large amount data quality repair issues.

7.4 Comparison With Existing Methods

In order to compare with the existing advanced methods, we implement two repairing algorithms: CRep in [27], which repairs the errors with *CFD*s with the min edit cost, without the temporal detection. And URM in [28], an advanced repair method which makes the trade-off between consistency repair by FDs and data repair based on core patterns, and minimize the modification cost. Since that both methods take less consideration of missing values, we apply CMI which fills in missing values according to an effective clustering on the most similar tuples [3]. We choose one optional repair method and one missing imputation method, and combine the two steps as an integrated approach. Specifically, we compared our Imp3C with four methods as follows.

- ImpCCons+CMI makes inconsistent repairing with the algorithm ImpCCons (in Section 5.2) first, and then make missing values imputation with CMI.
- URM+ImpCCom makes inconsistent repairing with URM first, and then make missing values imputation with the algorithm ImpCCons (in Section 6).
- URM+CMI executes URM first, and then executes CMI.
- CRep+CMI executes CRep first, and then executes CMI.

Effectiveness. Fig. 9 reports P and R on the three datasets with $noi = 10\%$. Imp3C outperforms other methods and has a rising trend with N increases. Imp3C well determines the currency in tuples, captures errors exactly and repairs them according to the proposed metric *cc*-DIST. ImpCCons+CMI and URM+ImpCCom have a closer performance on *NBA* and *HOSP*, and they both outperform URM+CMI and CRep+CMI. It from a side indicates that both ImpCCons and ImpCCom steps in our method provide reliable solutions in inconsistency and incompleteness repair when combined with existing methods.

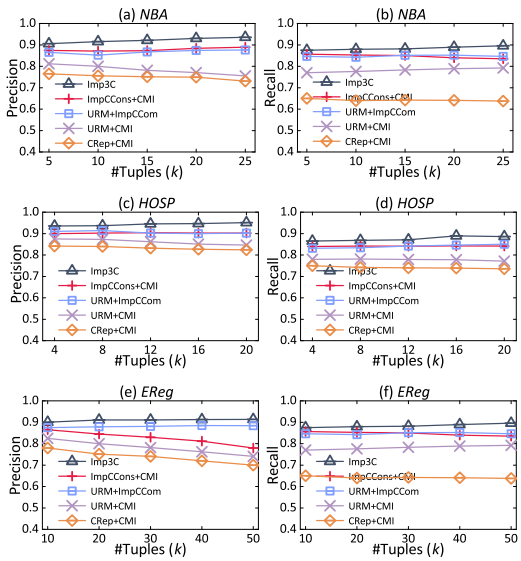


Fig. 9. Effectiveness comparison with $noi = 10\%$.

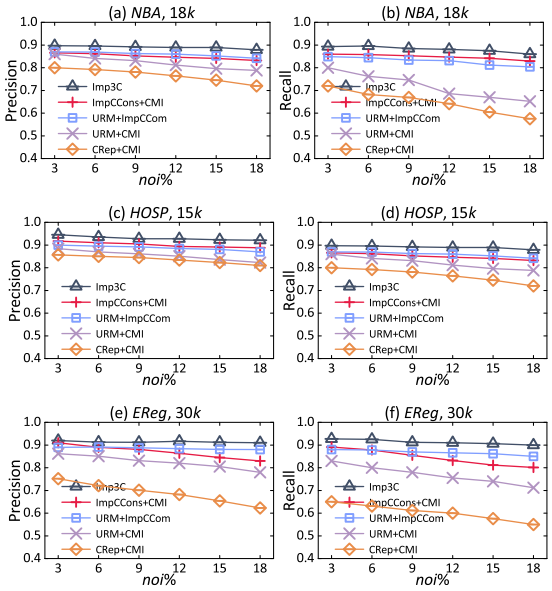


Fig. 10. Noise tolerance comparison with the existing methods.

Since CMI fills missing values with the similarity over all attributes, it performs poor in the data with many attributes (e.g., *EReg* with 22 attributes). The similarities are too close to distinguish the nearest tuple pairs. Thus, the three approaches consisting CMI fail fast in P while R holds a low level in Figs. 9e and 9f. CRep+CMI has the worse performance. Besides the weakness of CMI, CRep only modifies values to minimize the edit cost. This principle is not always accurate enough for data cleaning [29], [30]. In addition, CRep also overlooks the interaction between consistent and currency constraints within tuples. For URM, it beats CRep because URM considers both constraint-based and core-pattern-based repair. The strategy is similar to our method, which is confirmed to be effective in the experiment. However, frequency threshold is not always reliable enough when the currency evolution discipline is not clear. Thus, Imp3C performs better than URM, especially in *NBA* where players have complex career changes.

Fig. 10 shows the noise tolerance comparison with $N = 18k$ in *NBA*, $N = 15k$ in *HOSP*, and $N = 30k$ in *EReg*. Imp3C performs the best and only has a tight drop when *noi* increases. It verifies Imp3C repairs multi-errors more accurately with the proposed steps. ImpCCons+CMI and URM+ImpCCom take the second place, but they both have a obvious drop when *noi* over 12 percent. Both URM and CRep fail to consider the currency orders in datasets. That lets some error patterns difficult to be captured and to be modified accurately with the increasing *noi*. It highlights the importance of currency evaluation with the temporal feature in tuples in our proposed Imp3C. Also, ImpCCons is better than CRep. It verifies the proposed *cc*-DIST contributes to repairing errors more effectively. Moreover, ImpCCom beats CMI in missing imputation. It emphasizes the currency order is really an important factor for filling missing values more accurately.

Efficiency. Table 5 reports the algorithms comparison on both F1-measure ($\frac{2PR}{P+R}$) and time costs, with #Tuple=25k in *NBA*, 20k in *HOSP* and 50k in *EReg*. Imp3C has the highest F1-measure score and the least time cost in all datasets. It

TABLE 5
Algorithms Comparison With $noi = 10\%$

	NBA 25k		*HOSP* 20k		*EReg* 50k	
	F1	Time(s)	F1	Time(s)	F1	Time(s)
Imp3C	**0.92**	234	**0.91**	149	**0.91**	875
ImpCCons+CMI	0.87	255	0.87	160	0.8	1123
URM+ImpCCom	0.85	259	0.86	172	0.86	1001
URM+CMI	0.72	271	0.8	176	0.76	1356
CRep+CMI	0.68	245	0.77	158	0.66	1245

shows that our repair strategy and currency computing approach are both effective and efficient in multi-errors data cleaning. ImpCCons+CMI and URM+ImpCCom reach 0.8 in F1. However, their time costs are higher and grow faster than Imp3C. On one hand, the currency-consistency distance *cc*-DIST in Imp3C is direct and efficient for repairing errors. On the other hand, the missing imputation in Imp3C is efficient to fill missing values with the currency orders. However, URM costs a lot of computing among core and deviant patterns based on frequency to decide repair schema. In addition, CMI is costly when the dataset has a large number of tuples with many attributes. From the above, the proposed Imp3C improves data cleaning performance on consistency and completeness with currency determining. Imp3C not only saves repair times with both currency values and the proper repair order, but also exactly detects the errors and repairs them to be cleaned ones.

8 Related Work

Data quality dimension is a characteristic for data requirements, measuring to which degree the data quality meets the demand. Consistency, completeness, and currency are three important dimensions besides accuracy [9]. *Consistency* describes the violation of integrity constraints. Semantic constraints such as FD, CFD, and CIND have been defined to guide data cleaning under specific circumstance [31]. Missing values imputation is a key point in data *completeness* issues, measuring to which degree a dataset has complete attribute values to describe the real-world information [9]. Similarity-based method [3] has been developed to fill the missing values. In general, inconsistency and incompleteness problems have been studied independently. Most of the existing methods fail to well consider the complementary between the two issues in data cleaning.

Data currency describes to which extent a dataset is up-to-date [9]. As temporal changes and evolution of values have drawn researchers' attention, currency determination methods have been developed in data quality evaluation [5]. However, most of the methods determine currency according to the timestamps. As timestamps are often incomplete in data integration, it promotes the study on currency determination independent of timestamps. [4] first proposes a constraint-based model for data currency reasoning and fundamental theoretical problems. [20] proposes timestamps cleaning methods based on temporal constraints.

Data Cleaning. As a necessary step in data preprocess, data cleaning and repairing problems have been studied for long, including constraint-based [28], [32], statistical [33], and machine learning strategies [34]. Moreover, crowdsourcing

solutions have been proposed to combine the automatic data cleaning and human cognitive [12]. The state-of-art data cleaning systems include NADEEF, a data cleaning platform that applies multiple types of data quality rules [7], and Unified Repair Model, allows both data and constraints repairs [28].

As the temporal data plays an important role in big data analysis, the survey paper [35] comprehensively reviews the state-of-the-art methods in time series data cleaning, and summarizes data cleaning tools and systems from research and industry, such as Piclean [36], Cleanits [37], PACAS [38], and MLClean [39]. The principle of cost minimality and modification based on similarity commonly guides the repairing course [6], however, more should be done in effective cleaning applications of real-world data [29].

As the dimensions are not independent issues, cleaning approaches have been developed with integrating data quality dimensions. Though the model of multiple data quality improvement has been proposed [5], technological breakthroughs are still in demand in developing a comprehensive method of data cleaning. Our work develops an integrated data cleaning approach considering three dimensions. Specifically, we enrich the inconsistency and incompleteness repairing with currency determining. The propose method can also complement existing data cleaning techniques considering currency issues.

9 Conclusion

We propose a 4-step approach to solve the repairing problem of low-quality data with inconsistent and missing values, which lacks for timestamps. We compute currency order values, according to which, we propose a data repair method based on currency-consistency distance, and introduce a missing value imputation approach considering currency orders. Experimental results on real-life data verify that our method validly improves data consistency and completeness with currency computing. Future work includes the iteration strategy design between data repair and missing imputation with little time costs, and an efficient application of comprehensive data cleaning method on big data.

Acknowledgments

This paper was partially supported by NSFC grant U1866602 and 61602129. Hongzhi Wang and Xiaoou Ding contributed to the work equally and should be regarded as co-first authors. We thank our anonymous reviewers whose comments contributed to the quality improvement of this paper.

References

[1] F. Sidi, P. H. S. Panahy, L. S. Affendey, M. A. Jabar, H. Ibrahim, and A. Mustapha, "Data quality: A survey of data quality dimensions," in *Proc. Int. Conf. Inf. Retrieval Knowl. Manage.*, 2012, pp. 300–304.

[2] W. Fan, F. Geerts, S. Ma, N. Tang, and W. Yu, "Data quality problems beyond consistency and deduplication," in *In Search of Elegance in the Theory and Practice of Computation - Essays Dedicated to Peter Buneman*. Berlin, Germany: Springer, 2013, pp. 237–249.

[3] S. Zhang, J. Zhang, X. Zhu, Y. Qin, and C. Zhang, "Missing value imputation based on data clustering," *Trans. Comput. Sci.*, vol. 1, pp. 128–138, 2008.

[4] W. Fan, F. Geerts, and J. Wijsen, "Determining the currency of data," *ACM Trans. Database Syst.*, vol. 37, no. 4, pp. 71–82, 2012.

[5] C. Cappiello, C. Francalanci, and B. Pernici, "Time related factors of data accuracy, completeness, and currency in multi-channel information systems," in *Proc. Conf. Adv. Inf. Syst. Eng.*, 2008, pp. 145–153.

[6] S. Hao, N. Tang, G. Li, J. He, N. Ta, and J. Feng, "A novel cost-based model for data repairing," *IEEE Trans. Knowl. Data Eng.*, vol. 29, no. 4, pp. 727–742, Apr. 2017.

[7] A. Ebaid *et al.*, "NADEEF: A generalized data cleaning system," *Proc. VLDB Endowment*, vol. 6, no. 12, pp. 1218–1221, 2013.

[8] B. Heinrich, M. Klier, and M. Kaiser, "A procedure to develop metrics for currency and its application in CRM," *J. Data Inf. Qual.*, vol. 1, no. 1, pp. 1–28, 2009.

[9] C. Batini, C. Cappiello, C. Francalanci, and A. Maurino, "Methodologies for data quality assessment and improvement," *ACM Comput. Surv.*, vol. 41, no. 3, pp. 16:1–16:52, 2009.

[10] W. Fan and F. Geerts, *Foundations of Data Quality Management*. San Rafael, CA, USA: Morgan & Claypool, 2012.

[11] A. K. Elmagarmid, P. G. Ipeirotis, and V. S. Verykios, "Duplicate record detection: A survey," *IEEE Trans. Knowl. Data Eng.*, vol. 19, no. 1, pp. 1–16, Jan. 2007.

[12] G. Li, J. Fan, J. Fan, J. Wang, and R. Cheng, "Crowdsourced data management: Overview and challenges," in *Proc. ACM Int. Conf. Manage. Data*, 2017, pp. 1711–1716.

[13] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 3rd ed. Cambridge, MA, USA: MIT Press, 2009.

[14] M. A. Bender, M. Farach-Colton, G. Pemmasani, S. Skiena, and P. Sumazin, "Lowest common ancestors in trees and directed acyclic graphs," *J. Algorithms*, vol. 57, no. 2, pp. 75–94, 2005.

[15] C. Bettstetter and J. Eberspcher, "Hop distances in homogeneous ad hoc networks," in *Proc. IEEE Veh. Technol. Conf.*, 2003, pp. 2286–2290.

[16] H. M. F. AboElFotoh, E. S. Elmallah, and H. S. Hassanein, "On the reliability of wireless sensor networks," in *Proc. IEEE Int. Conf. Commun.*, 2006, pp. 3455–3460.

[17] I. M. D. Silva, L. A. Guedes, P. Portugal, and F. Vasques, "Reliability and availability evaluation of wireless sensor networks for industrial applications," *Sensors*, vol. 12, no. 1, pp. 806–838, 2012.

[18] W. Fan, F. Geerts, J. Li, and M. Xiong, "Discovering conditional functional dependencies," *IEEE Trans. Knowl. Data Eng.*, vol. 23, no. 5, pp. 683–698, May 2011.

[19] T. Papenbrock *et al.*, "Functional dependency discovery: An experimental evaluation of seven algorithms," *Proc. VLDB Endowment*, vol. 8, no. 10, pp. 1082–1093, 2015.

[20] S. Song, Y. Cao, and J. Wang, "Cleaning timestamps with temporal constraints," *Proc. VLDB Endowment*, vol. 9, no. 10, pp. 708–719, 2016.

[21] W. Fan, F. Geerts, N. Tang, and W. Yu, "Conflict resolution with data currency and consistency," *J. Data Inf. Qual.*, vol. 5, no. 1/2, pp. 1–37, 2014.

[22] T. Deng, W. Fan, and F. Geerts, "Capturing missing tuples and missing values," in *Proc. 29th ACM SIGMOD-SIGACT-SIGART Symp. Princ. Database Syst.*, 2010, pp. 169–178.

[23] T. M. Mitchell, *Machine Learning*. New York, NY, USA: McGraw-Hill, 1997.

[24] J. R. Quinlan, "Induction of decision trees," *Mach. Learn.*, vol. 1, no. 1, pp. 81–106, 1986.

[25] W. Fan, S. Ma, N. Tang, and W. Yu, "Interaction between record matching and data repairing," *J. Data Inf. Qual.*, vol. 4, no. 4, pp. 16:1–16:38, 2014.

[26] X. Chu, I. F. Ilyas, P. Papotti, and Y. Ye, "RuleMiner: Data quality rules discovery," in *Proc. IEEE Int. Conf. Data Eng.*, 2014, pp. 1222–1225.

[27] 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.

[28] C. Fei and R. J. Miller, "A unified model for data and constraint repair," in *Proc. IEEE Int. Conf. Data Eng.*, 2011, pp. 446–457.

[29] I. F. Ilyas, "Effective data cleaning with continuous evaluation," *IEEE Data Eng. Bull.*, vol. 39, no. 2, pp. 38–46, Jun. 2016

[30] J. Szlichta, J. Szlichta, C. Fei, and D. Srivastava,, "Combining quantitative and logical data cleaning," *Proc. VLDB Endowment*, vol. 9, no. 4, pp. 300–311, 2015.

[31] 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.

[32] X. Chu, I. F. Ilyas, and P. Papotti, "Holistic data cleaning: Putting violations into context," in *Proc. IEEE Int. Conf. Data Eng.*, 2013, pp. 458–469.

[33] L. Berti-Equille, T. Dasu, and D. Srivastava, "Discovery of complex glitch patterns: A novel approach to quantitative data cleaning," in *Proc. IEEE Int. Conf. Data Eng.*, 2011, pp. 733–744.
[34] 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.
[35] X. Wang and C. Wang, "Time series data cleaning: A survey," *IEEE Access*, vol. 8, pp. 1866–1881, 2020. [Online]. Available: https://doi.org/10.1109/ACCESS.2019.2962152
[36] Z. Yu and X. Chu, "PIClean: A probabilistic and interactive data cleaning system," in *Proc. Int. Conf. Manage. Data SIGMOD Conf.*, 2019, pp. 2021–2024.
[37] X. Ding, H. Wang, J. Su, Z. Li, J. Li, and H. Gao, "Cleanits: A data cleaning system for industrial time series," *Proc. VLDB Endowment*, vol. 12, no. 12, pp. 1786–1789, 2019.
[38] Y. Huang, M. Milani, and F. Chiang, "PACAS: Privacy-aware, data cleaning-as-a-service," in *Proc. IEEE Int. Conf. Big Data*, 2018, pp. 1023–1030.
[39] K. H. Tae, Y. Roh, Y. H. Oh, H. Kim, and S. E. Whang, "Data cleaning for accurate, fair, and robust models: A big data - AI integration approach," in *Proc. 3rd Int. Workshop Data Manage. End-to-End Mach. Learn.*, 2019, pp. 5:1–5:4.

Xiaoou Ding received the bachelor's degree from the Harbin Institute of Technology, Harbin, China, in 2015. She is currently working toward the PhD degree in the School of Computer Science and Technology, Harbin Institute of Technology, Harbin, China. Her research interests include data cleaning, temporal data quality management, and temporal data mining and analysis. She is also interested in IoT data cleaning and anomaly detection.

Hongzhi Wang received the PhD degree in computer science from the Harbin Institute of Technology, Harbin, China, in 2008. From 2008 to 2010, he was an assistant professor with the Harbin Institute of Technology, China. From 2010 to 2015, he was an associate professor. Since 2015, he has been a professor and doctoral supervisor of the Department of Computer Science and Technology. His research interests include big data management, data quality, graph data management, and Web data management. He has published more than 100 papers in refereed journals and conferences. He is a recipient of the Outstanding Dissertation Award of CCF, Microsoft fellow, Chinese excellent database engineer, and IBM PhD fellowship.

Jiaxuan Su received the bachelor's degree from the Harbin Institute of Technology, Harbin, China, in 2017. He is currently working toward the PhD degree in the School of Computer Science and Technology, Harbin Institute of Technology, Harbin, China. His research interests include data cleaning, data integration, and data quality management system development.

Muxian Wang is currently working toward the graduate degree in the School of Computer Science and Technology, Harbin Institute of Technology, Harbin, China. Interested in time series data management, he has joined in an Apache open-source program cooperated with Tsinghua University for two years. He is also interested in probabilistic querying problem of industrial uncertain big data and did a research on vertically distributed uncertain data by probabilistic skyline computation. He takes part in industrial big data cleaning system program, making a research of anomaly detection on multi-dimensional time series with fellow students. Except research programs above, he has also done a research on Chinese knowledge graph by text simplification in NLP.

Jianzhong Li received the BS degree from Heilongjiang University, Harbin, China, in 1975. He worked with the University of California at Berkeley as a visiting scholar in 1985. He has also been a visiting professor with the University of Minnesota at Minneapolis, from 1991 to 1992 and from 1998 to 1999. Since 1998, he has been a professor and doctoral supervisor with the Department of Computer Science and Technology, Harbin Institute of Technology, China. His current research interests include database management systems, data warehousing and data mining, wireless sensor network, and data intensive super computing. He was a recipient of awards and honors, including the chairman of the ACM SIGMOD China and the director of the China Computer Federation.

Hong Gao received the PhD degree from the Harbin Institute of Technology, Harbin, China. She is a professor and doctoral supervisor with the Institute of Technology. She has long engaged in research work of massive data computation and quality management, wireless sensor networks and graphic data management and computation. She was a recipient of awards and honors, including the assistant director of the China Computer Federation Technical Committee on Databases, a member of the China Computer Federation Technical Committee on Sensor Network, and the deputy director of the Massive Data Computing Lab. She is a senior member of the CCF. Her research interests include database, parallel computing, wireless sensor networks, etc.