TSINGHUA SCIENCE AND TECHNOLOGY
ISSN 1007-0214 01/09 pp227–242
Volume 22, Number 3, June 2017

Efficient Currency Determination Algorithms for Dynamic Data

Xiaoou Ding, Hongzhi Wang*, Yitong Gao, Jianzhong Li, and Hong Gao

Abstract: Data quality is an important aspect in data application and management, and currency is one of the major dimensions influencing its quality. In real applications, datasets timestamps are often incomplete and unavailable, or even absent. With the increasing requirements to update real-time data, existing methods can fail to adequately determine the currency of entities. In consideration of the velocity of big data, we propose a series of efficient algorithms for determining the currency of dynamic datasets, which we divide into two steps. In the preprocessing step, to better determine data currency and accelerate dataset updating, we propose the use of a topological graph of the processing order of the entity attributes. Then, we construct an Entity Query B-Tree (EQB-Tree) structure and an Entity Storage Dynamic Linked List (ES-DLL) to improve the querying and updating processes of both the data currency graph and currency scores. In the currency determination step, we propose definitions of the currency score and currency information for tuples referring to the same entity and use examples to discuss methods and algorithms for their computation. Based on our experimental results with both real and synthetic data, we verify that our methods can efficiently update data in the correct order of currency.

Key words: data quality management; data currency; dynamic determining

1 Introduction

With todays rapid growth in the volumes of data, data quality is becoming a crucial problem in data management. As one of the most important dimensions of data quality, data currency problems are becoming more troublesome in practical databases and information systems. Statistically speaking, about 2% of all customer business data information will be obsolete within a month[1]. In other words, about 50% of all data is rendered unavailable because it is stale. Furthermore, out-of-date datasets may lead to incorrect decisions by decision-makers, which can result in economic losses in organizations[2]. In the United States alone, businesses are reported to suffer annual financial losses of 600 billion dollars due to data quality problems (Refs. [1, 3]). In one 2005 example, out-of-date customer information in a bank database led to thousands of completed tax forms being sent to obsolete addresses, making it possible for identity thieves to effortlessly obtain the names and bank accounts of many individuals. In another case, the Internal Revenue Service (IRS) accused some people for overdue tax caused by errors in the IRS database system[4]. With the increasing seriousness of data currency problems in this big data era, research is both necessary and urgent.

For practical applications in which timestamps are invalid or unavailable (Refs. [5, 6]), one major area of data currency research involves currency determination by the analysis of the attribute currency order of entities with reliable currency constraints. However, during the updating process of massive volumes of dynamic data, traditional static determination methods cannot adapt to the need for prompt updates in the big data and are therefore ineffective (such as the price problem in Ref. [7]). Existing currency determination methods can

• Xiaoou Ding, Hongzhi Wang, Yitong Gao, Jianzhong Li, and Hong Gao are with School of Computer Science and Technology, Harbin Institute of Technology, Harbin 150001, China. E-mail: dingxiaoou@stu.hit.edu.cn; wangzh@hit.edu.cn; gaoyitong@163.com; lijzh@hit.edu.cn; honggao@hit.edu.cn.

∗To whom correspondence should be addressed.

Manuscript received: 2017-03-26; revised: 2017-04-06; accepted: 2017-04-11

fail to provide sufficient guidance for making timely information updates in entities.

As yet, little research has been done to determine the currency of dynamic data that lack available timestamps. The main challenges in determining data currency are as follows:

• **Performance bottlenecks related to large volumes of data.** With the rapid growth in the volume and diversity of information, the performance of algorithms in determining the currency of data faces serious challenges with respect to efficiency, effectiveness, and accuracy.

• **Complex conditions in dynamic data updating.** To minimize confusion in temporal dataset management, when data arrives to be updated, the corresponding entity must be queried and its tuples be maintained in their order of currency. Accordingly, different dynamic data types and sizes add to the complexity of updating the original dataset.

• **High demand for real-time updating.** With our growing ability to generate and obtain data, demand for real-time information and knowledge is increasing both in business and between customers. The efficiency and quality of the data updating process is influenced by numerous factors such as the computer resources and capabilities, data volume, and algorithm performances.

• **Difficulties in identifying dependable principles for currency determination.** It is not easy to identify reliable and practicable principles for analyzing the currency of dynamic datasets that lack timestamps.

In this paper, we propose a series of efficient algorithms for the dynamic and real-time determination of data currency based on the volume and velocity of large-scale dynamic data. We present examples that illustrate the motivation for this paper in Section 1.1 below.

1.1 Motivating examples

Example 1 As shown in Table 1, the personal information relational dataset named ***Info*** has two entities that record student information when they are enrolled in college and after they graduate. The table presents recent study and work information of two students, Alice and Tom. Each tuple provides personal details including tID, eID, Name, Sex, Degree, Position, College, Address, Salary, and Status. Here, tID and eID represent tuple ID number and entity ID number, respectively.

To rapidly update the dataset ***Info***, while maintaining the correct order of currency of the tuples of each entity, we first preprocess the dataset ***Info*** (initial dataset) by recording the storage addresses of all entities and maintaining the currency order of the tuples in ***Info***. If we assume that there are no valid or complete timestamps in the entities in ***Info***, we can then identify the following currency constraints (rules) and use them to determine the currency order in each entity.

r_1: The degree in a given entity is only from *Bachelor* to *Master*, and from *Master* to *PhD*.

r_2: The status of a given entity is only from *Single* to *Married*.

r_3: The salary of a given entity only increases with time.

r_4: If a given Salary value in the same entity is the most current, the corresponding tuple of the Address value is also most current.

We use $t_i \prec t_j$ to represent that tuple t_j is more current than t_i, and $t_i = t_j$ to indicate that t_i has the same currency order with t_j[5]. Accordingly, the above currency rules, as shown in Definition 3, can be represented as follows:

r_1: $\forall t_1, t_2 \in \textbf{\textit{Info}}, \big(t_1[\mathsf{eID}] = t_2[\mathsf{eID}] \wedge (t_1[\mathsf{Degree}] = \textit{Bachelor} \wedge t_2[\mathsf{Degree}] = \textit{Master})\big) \longrightarrow t_1 \prec_{\mathsf{Degree}} t_2$;
and $\forall t_1, t_2 \in \textbf{\textit{Info}}, \big(t_1[\mathsf{eID}] = t_2[\mathsf{eID}] \wedge (t_1[\mathsf{Degree}] = \textit{Master} \wedge t_2[\mathsf{Degree}] = \textit{PhD})\big) \longrightarrow t_1 \prec_{\mathsf{Degree}} t_2$.

r_2: $\forall t_1, t_2 \in \textbf{\textit{Info}}, \big(t_1[\mathsf{eID}] = t_2[\mathsf{eID}] \wedge (t_1[\mathsf{Status}] = \textit{Single} \wedge t_2[\mathsf{Status}] = \textit{Married})\big) \rightarrow t_1 \prec_{\mathsf{Status}} t_2$.

r_3: $\forall t_1, t_2 \in \textbf{\textit{Info}}, \big(t_1[\mathsf{eID}] = t_2[\mathsf{eID}] \wedge$

Table 1 Entity relations dataset *Info*.

tID	eID	Name	Sex	Degree	Position	College	Address	Salary	Status
t_1	e_1	Alice	F	Bachelor	Student	HIT	3-DP	40	Single
t_2	e_1	Alice	F	Master	Student	HIT	15-DP	500	Single
t_3	e_1	Alice	F	Master	Programmer	HIT	Beijing	12 000	Single
t_1	e_2	Tom	M	Bachelor	Student	HIT	1-DP	40	Single
t_2	e_2	Tom	M	Master	Student	HIT	16-DP	500	Single
t_3	e_2	Tom	M	PhD	Student	HIT	10-DP	1000	Married

$(t_1[\mathsf{Salary}] < t_2[\mathsf{Salary}])) \longrightarrow t_1 \prec_{\mathsf{Status}} t_2$.

r_4: $\forall t_1, t_2 \in \boldsymbol{Info}, (t_1[\mathsf{eID}] = t_2[\mathsf{eID}] \wedge (t_1 \prec_{\mathsf{Status}} t_2)) \longrightarrow t_1 \prec_{\mathsf{Address}} t_2$.

To take the entity *Alice* as instance, according to the above currency rules, we can determine the currency order of each attribute of the entity *Alice* to be follows: (1) Degree: *Bachelor* $\prec$ *Master*; (2) Address: *3-DP* $\prec$*15-DP* $\prec$ *Beijing*; (3) Salary: 40 $\prec$ 500 $\prec$ 12 000. To describe the currency order of each attribute's value in each of the entities in the dataset, we introduce currency scores (numerical values) to compute the currency of the values of some particular attributes that have a major impact on determining the currency of the tuple. In general, the greater is this score, the more current is the attribute's value. We discuss the relative definitions and computation process in Section 4.2.

Next, we suppose we have the need to insert a tuple t_{new} (as shown in Table 2) into ***Info*** as described in the following example.

Firstly, we must rapidly identify the storage address of the entity in the dataset ***Info*** corresponding to the tuple t_{new}. Then, using the currency rules (r_1 to r_4), we determine the currency of the attributes based on the t_{new} currency scores. Then, we efficiently insert this t_{new} currency information in the correct location to ensure that the dataset maintains the correct data currency.

1.2 Contributions

In this paper, we introduce a model for dynamically determining data currency and propose several efficient algorithms related mainly to preprocessing and real-time dynamic determination. We can summarize our contributions as follows:

• To the best of our knowledge, we are the first to propose algorithms for determining the currency of dynamic data, and we also introduce an integrated process for the currency determining algorithms.

• To accelerate the entity updating process, we introduce an efficient structure for indexing and querying dataset entities, known as the Entity Query B-Tree index (EQB-Tree) and the Entity Storage Dynamic Linked List (ES-DLL).

• We propose methods for directly determining and computing the currency of the attributes in an entity. This updating approach, which uses both a currency graph and currency score, accurately determines the currency of information in dynamic data.

• In a series of experiments, we verify the efficiency of our methods and algorithms on both real-life and synthetic data.

Organization. The organization of the rest of this paper is as follows: In Section 2, we discuss related work in data currency determination and management. In Section 3, we introduce our model for dynamic data currency determination and we discuss the algorithms used for currency determination in Section 4. We report our experimental study results in Section 5, and in Section 6, we draw our conclusions.

2 Related Work

In the data quality management literature, there is no standard definition of currency[8]. In recent years, research on data currency has mainly involved two approaches to data repair, namely currency determination methods based on available timestamps and those based on constraints and rules.

Currency determining with timestamps. Datasets that include timestamps provide clear time points for each transaction, and out-of-date records in the dataset can be easily identified via querying and computing operations. The main research focus in Refs. [9, 10] is querying the most current records based on timestamps and temporal constraints within the databases. In some papers, (e.g., Refs. [11–16]), the freshness degree of the dataset is indicated by the attribute parameter *age*, which is defined as the time gap between the assessment of currency and the acquisition of the attribute's values. *Shelf life* is also an indicator of value volatility and currency is calculated as a function of the *age* of an attribute's value and its *shelf life* in the dataset. The authors in Refs. [14, 15] also proposed a probability-based metric for determining currency that estimates the decline rate based on historical data, in which a quick decline rate inevitably leads to outdated currency.

With valid, accurate, and complete timestamps, determining currency becomes much simpler and easier as the algorithms can be easily and clearly designed, which makes them highly efficient, scalable, and

Table 2 New tuple for *Alice*.

Name	Sex	Degree	Position	College	Address	Salary	Status
Alice	F	Master	Programmer	HIT	Shanghai	12 000	Married

accurate in their data currency analysis. Moreover, less expertise is required to develop these algorithms, which means lower costs related to determining currency and repairing data. However, dependence on timestamps also makes the metrics inapplicable to real information systems that have no valid timestamps due to the extensiveness of the data sources, changes in data storage, or other circumstances. Furthermore, data currency determination is not always reliable when it is based merely on the age of the data or other similar parameters in the data records. Under certain conditions, old data may not necessarily mean that it is no longer current. For example, historical meteorological information for a given location continues to be important for climate change research.

Currency determining based on constraints and rules. In view of the fact that timestamps are often incomplete or nonexistent in real applications, the authors in Ref. [5] were the first to propose a rule-based model for determining data currency. In the paper, the authors discussed theoretical issues related to several fundamental problems. To investigate data currency, they associated the partial currency orders, denial constraints[17, 18], and copy functions[13, 19] of data sources, but proposed no practical algorithms. The authors in Refs. [2,20] conducted further research based on the theory presented in Ref. [5]. In Ref. [2], the authors presented currency evaluation methods using currency constraints and redundant records on which they based the construction of a currency graph for a given entity. Then, they used a topological algorithm to determine a time series and new values of the entity for different attributes, as well as the time complexity of the algorithm in polynomial time. The authors in Ref. [20] also developed effective algorithms and top-k heuristic algorithms underlying a model for determining relative accuracy and currency. Based on the work in Ref. [2] and with a focus on improving data quality, the authors in Ref. [6] were the first to combine data quality rules and statistical techniques to improve data currency. They also proposed currency repairing rules and discussed relative problems in theory.

In other work, currency problems have been associated with other impact factors (e.g., Refs. [21–23]) to solve data cleaning and repair problems. The authors in Refs. [21, 23] studied both data currency and consistency to achieve conflict resolution in datasets and introduced a framework and efficient algorithms for conflict resolution that combines partial currency orders, currency constraints, and conditional functional dependencies[5].

Rule-based methods for determining data currency have wide applications in real information systems. There is no denying that the process of determining currency is generally complex and costly. Faced with large-scale data, currency determination research involves challenges in efficiency and data updating. In Ref. [24], we proposed currency determination methods for dynamic datasets, based on the use of currency graphs and scores to determine the currency of the different tuples of entities.

Based on our work in Ref. [24], here we present further research to improve the accuracy and effectiveness of the determination of data currency. We propose the use of a topological graph of the processing order of entity attributes and discuss the preprocessing algorithms used to create an EQB-Tree and ES-DLL. We also define the term *Currency Information* and present a corresponding computation and updating method.

3 Overview

In this section, we present an overview and discuss the dynamic data currency problem. In Section 3.1, we provide background knowledge and some fundamental definitions of data currency, and we propose our method framework in Section 3.2.

3.1 Definitions

The dataset and currency rules we employ in this paper for determining currency are the same as those used by the authors in Ref. [5], and we present them below as Definition 1 and Definition 2, respectively.

Definition 1 ***Initial Relations Dataset D*** Suppose the data schema $\mathcal{R} = (\mathsf{tID}, \mathsf{eID}, A_1, ..., A_n)$, where tID is the tuple ID number, eID is the entity ID number, and $\mathcal{A} = \{A_1, ..., A_n\}$ is the set of attributes. $\mathcal{E} = \{e_1, e_2, ..., e_m\}$ is the set of all the entities involved in the dataset. If $t_i[\mathsf{eID}] = t_j[\mathsf{eID}]$, then the tuples t_i and t_j represent the same entity, and $t_i[A_k]$ represents the value of the attribute A_k in tuple t_i. Dataset D is the set including massive instances like those in data schema $\mathcal{R}$.

Definition 2 ***Currency Rules*** The currency constraints are used to determine the currency of data for which timestamps are incomplete, unreliable, or do not exist. In the set of currency

rules: $\Phi_{\mathrm{CR}} = \{t_i[\mathsf{eID}] = t_j[\mathsf{eID}] \wedge \psi \mid i, j \in [1, N_e]\}$, N_e is the number of entities in the dataset D, and ψ represents the predicate in an instance of rules. There are mainly three main kinds of rules regarding ψ:

φ_1: $\forall t_1, t_2, \big(t_1[\mathsf{eID}] = t_2[\mathsf{eID}] \wedge (t_1[A_k] = v_1 \wedge t_2[A_k] = v_2)\big) \longrightarrow t_1 \prec_{A_k} t_2$;

φ_2: $\forall t_1, t_2, \big(t_1[\mathsf{eID}] = t_2[\mathsf{eID}] \wedge (t_1[A_k]\ op\ t_2[A_k])\big) \longrightarrow t_1 \prec_{A_k} t_2, op = \{>, <, \geqslant, \leqslant, =, \neq\}$;

φ_3: $\forall t_1, t_2, \big(t_1[\mathsf{eID}] = t_2[\mathsf{eID}] \wedge (t_1 \prec_{A_k} t_2)\big) \longrightarrow t_1 \prec_{A_m} t_2$.

We refer to the Left-Hand Side of $\longrightarrow$ as the LHS of φ_i and the Right-Hand Side as RHS. Accordingly, we use Φ_1 to describe a set of instances, such as φ_1, Φ_2 to describe φ_2, and Φ_3 to describe φ_3 (Φ_1, Φ_2, and $\Phi_3 \in \Phi_{\mathrm{CR}}$). Considering the *Currency Rules* in the above motivating example, r_1 and r_2 are the instances similar to φ_1, r_3 is an instance similar to φ_2, and r_4 is similar to φ_3.

To analyze the currency of the tuples of each entity, we build a currency graph for entity e_i with respect to attribute A_k according to the method used in Ref. [2], as expressed as Definition 3 below.

Definition 3 ***Entity's Currency Graph*** The Directed Graph $\mathcal{G}(e_i) = \{\mathcal{G}(e_i, A_k) \mid e_i \in \mathcal{E}, A_k \in \mathcal{A}\}$ is the currency graph of e_i, in which the vertex set $V = \{t \mid t \in \mathcal{T}_{e_i}\}$, and the edge set $E = \big\{(t_i, t_j) \mid t_i \prec_{A_k} t_j \in \text{RHS of } (\Phi)\big\}$.

3.2 Framework

In our **Dynamic Data Currency Problem (DDC)**, our goal is to establish dynamic real-time updating of the data in the dataset, while maintaining the correct currency order of all the entities in the dataset. The dataset we use in this DDC is similar to the traditional currency problems described in Ref. [5], which occur immediately after the entity recognition process[25]. Due to the fact that the massive entities in large-scale datasets can have multiple records (tuples) for the same entity, we propose approaches for determining the data currency of dynamic data. Figure 1 shows the framework of our method, which consists of two main phases: Data Preprocessing and Dynamic Currency Determination.

As shown in Fig. 1, we propose to conduct data preprocessing offline and data currency determination online. We preprocess the initial dataset offline to obtain concise currency information and avoid duplicate computation in the following step. During the dynamic determination step, we can efficiently update online the current tuples to be inserted into the dataset in the correct currency order based on the analysis results from the preprocessing step.

(a) **Preprocessing** Since the original data in many applications is often disordered and unsystematic, preprocessing is necessary to achieve better DDC processing. First, we create a B-Tree index structure for querying entities (EQB-Tree) and initialize all the entities in dataset D that record the head address of each entity. Then, we create and initialize a dynamic linked list for the storage of entities (ES-DLL), which helps to reduce the time required to update data in the dynamic determination phase. Next, we create currency scores and currency graphs for the entities in D are created. Suppose that the currency graph of entity e_i: $\mathcal{G}(e_i) = \big\{\mathcal{G}(e_i, A_k) \mid i \in [1, N_e], j \in [1, n]\big\}$, where $\mathcal{G}(e_i, A_k)$ represents the attribute currency graph of A_k with respect to entity e_i, shows the currency scores of the attribute's values in A_k that were generated by the currency rules. We propose a definition of currency score in Definition 4 below:

Definition 4 ***Currency Score*** Suppose the entity

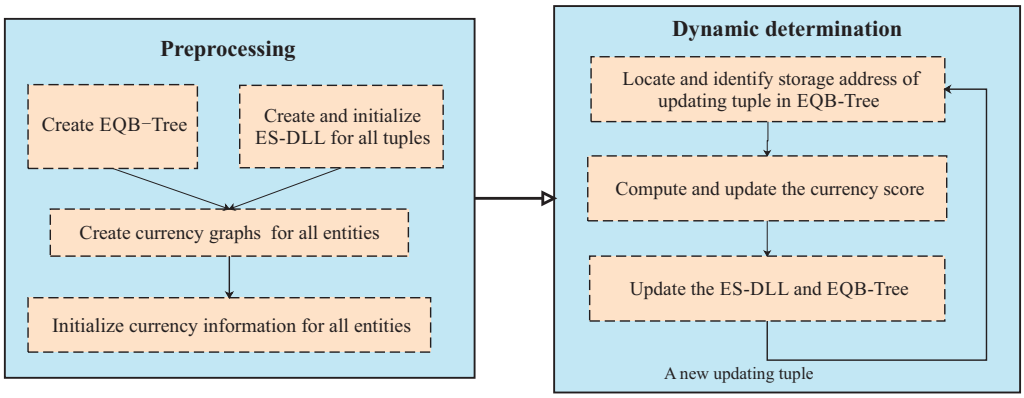


Fig. 1 Framework for determining currency of dynamic data.

e is one of the entities in the relational dataset D, we express the score of the tuple t_i of $(t_{i,e})$ as shown in Eq. (1):

$$\text{score}(t_{i,e}) = \sum_{A_k \in A, j=1}^{n} \text{score}\big(t_i[A_k]\big) \quad (1)$$

And the currency score of e is shown in Eq. (2):

$$\text{score}(e_i) = \sum \text{score}(t_{j,e}),\ j \in [1, N_{e_j}] \quad (2)$$

In Eq. (1), the score of the tuple $t_{i,e}$ is the sum of the currency scores $\text{score}\big(t_i[A_k]\big)$ of all attribute values of the entity e, and all of the scores for the tuples or attribute values are in positive integer forms. The score of the least current attribute value is 1, and the more current is the attribute value $[A_k]$, the larger is the currency score of $[A_k]$.

After we compute both the currency graph and currency score, we can establish the currency information of entity e_i, as defined in Definition 5 below:

Definition 5 ***Currency Information*** The Currency Score and Currency Graph both together comprise the Currency Information: $\text{curInfo}_{e_i} = \big(\mathcal{G}(e_i), \text{score}(e_i)\big)$.

(b) **Dynamic Determination** Considering a tuple t_{new} to be updated, we firstly recognize the corresponding entity of t_{new} by appropriate entity recognition methods proposed in Ref. [26], and then find the storage address of entity e_i and create its currency graph $\big(\mathcal{G}(e_i).addr\big)$ in EQB-Tree and put the records and the corresponding currency graph $\mathcal{G}(e_i)$ into the memory. After that, both the currency graph of e_i and the currency scores of attributes' values of t_{new} are updated. Finally, the ES-DLL with t_{new} will be updated and the most fresh currency information of e_i will be written to the external storages.

4 Algorithms for Currency Determining of Dynamic Data

In this section, we describe in detail our proposed methods and algorithms for determining data currency. We introduce the algorithms used in offline processing in Section 4.1 and present the currency rule processes in Section 4.1.1, the creation of the EQB-Tree and ES-DLL in Section 4.1.2, and the creation of currency information for entities in Section 4.1.3. In Section 4.2, we discuss the algorithms for online currency determination, including updating the EQB-Tree and ES-DLL (Section 4.2.1), updating the currency graphs (Section 4.2.2), and updating the currency scores (Section 4.2.3). In addition, we analyze examples for each algorithm. Table 3 lists some of the notations frequently used in this section.

4.1 Algorithms for preprocessing

4.1.1 Processing currency rules

Determining whether the currency constraint (namely CSP in Refs. [5, 27]) is satisfied as well as the currency orders of attributes are essential to effectively address currency problems in data quality management, and represent the first step in the determination process. When determining the currency of information, numerous attributes may need to be considered, which can result in high time costs. To improve the efficiency of calculating the currency of entity information, we propose the use of a topological graph showing the processing order of the entity attributes, based on Definition 6 below.

When processing currency rules Φ_{CR}, such as φ_1 and φ_2, that help to maintain tuples in the correct temporal order, only one identical attribute is contained in the LHS and RHS of φ. However, the involvement of two distinct attributes in φ_3 does change things. For $\varphi_3 = \{\forall t_1, t_2, \big(t_1[\mathsf{eID}] = t_2[\mathsf{eID}] \wedge (t_1 \prec_{A_k} t_2)\big) \longrightarrow t_1 \prec_{A_m} t_2\}$, the rules must follow a certain processing partial order between attributes A_k and A_m.

Definition 6 ***Attributes Processing Order in Determining Currency*** For all attributes involved in the set of currency rules Φ_{CR}, suppose $A_i,\ A_j \in \mathcal{A}$, for a certain currency rule $\varphi \in \Phi$ form as $\varphi_3 = \{\forall t_1, t_2, \big(t_1[\mathsf{eID}] = t_2[\mathsf{eID}] \wedge (t_1 \prec_{A_i} t_2)\big) \longrightarrow t_1 \prec_{A_j} t_2\}$. As A_i is in the LHS of φ_3 and A_j in the RHS, A_i must be determined before A_j, and their order is represented as $A_i \prec^{curr} A_j$. Another two attributes, A_m involved in φ_m and A_n in φ_n $(\varphi_m \neq \varphi_n)$,

Table 3 Frequent notations.

Notation	Description
$\mathcal{E}$	The set of entities in dataset D
$\mathcal{T}(e)$	The set of tuples referring to a given entity e
ψ	The predicate in a currency rule
Φ_{CR}	The set of currency rules
$\mathcal{G}(e_i, A_k)$	The currency graph of the entity e_i on A_k
$\text{score}(t_{i,e})$	The currency score of the tuple t_i referring to e
score_e	The total currency score of the entity e
curInfo_e	The currency information of e
$\mathcal{Q}_{\Phi}$	The currency processing order of attributes
$\asymp^{curr}$	Independent processing order
$\mathcal{O}(e)^T$	The structure used in EQB-Tree for e
$\mathcal{O}(e)^L$	The structure used in ES-DLL for e

are independent of each other when processing rules, so $A_m \asymp^{curr} A_n$.

According to Definition 6, we can obtain a topological graph of the processing order of the entity attributes. On one hand, with this processing order, the attributes not in Φ_{CR} can be filtered out to save time and avoid having to determine attributes that cannot help in the determination of currency in corresponding entities. In addition, it provides a more efficient and reasonable attribute processing order that avoids duplication and conflict when processing $\varphi \in \Phi_{\text{CR}}$. The algorithm for generating the processing order of the attributes Q_Φ is shown in Algorithm 1.

$\mathcal{G}(\text{CR})$ represents the topological graph of attributes processing order, and $V'_A \subseteq V_A$ is the set of attributes involving Φ_{CR}. After initializing $\mathcal{G}(\text{CR})$ in Line 1, we construct the graph according to the rules in Φ_{CR} (Lines 2–7). Then, after initializing Queue and Q_Φ (Line 8), we add attributes with 0 in-degree in Queue to Q_Φ (Lines 9–13). If in-degree of all attributes in V'_A is not equal to 0, this indicates that the rules in Φ_{CR} conflict and Φ_{CR} will be return to be re-prepared (Lines 22 and 23). Finally, we return Q_Φ as the processing order of attributes (Line 27).

Algorithm 1 Generating the processing order of attributes Q_Φ

Input: the set of currency rules Φ_{CR}
Output: the topological graph of attributes processing order Q_Φ

1: $\mathcal{G}(\text{CR}) \leftarrow (V'_A, E), V'_A \leftarrow \phi, E \leftarrow \phi$
2: **for** each $\varphi \in \Phi_{\text{CR}}$ **do**
3: add A including in φ into V'_A;
4: **if** $\varphi \in \Phi_3$ **then**
5: add (A_{l_i}, A_{l_j}) into E;
6: **end if**
7: **end for**
8: Queue $\leftarrow \phi$, $Q_\Phi \leftarrow \phi$;
9: **for** each $A \in V'_A$ do **do**
10: **if** Indegree(A)= 0 **then**
11: add A into Queue;
12: **end if**
13: **end for**
14: **while** Queue.*noEmpty*() **do**
15: A= Queue.*pop*();
16: add A to Q_Φ;
17: delete A from V'_A and delete (A, A_{l_j}) from E;
18: **for** each $A \in V'_A$ **do**
19: **if** Indegree(A)= 0 **then**
20: add A to Queue;
21: **end if**
22: **if** all Indegree(A)$\neq$ 0, $A \in V'_A$ **then**
23: exit (-1) and adjust Φ by users;
24: **end if**
25: **end for**
26: **end while**
27: return Q_Φ;

Example 2 Consider currency rules r_1, r_2, r_3, r_4 in the motivating example. Figure 2 shows a topological graph of the processing order of the entity attributes, correspondingly.

As we can see in Fig. 2, a possible processing order of the attributes, which we can obtain from ***Info***, is as follows:

$$\text{Degree} \asymp^{curr} \text{Salary} \asymp^{curr} \text{Status} \prec^{curr} \text{Address}.$$

4.1.2 Creating EQB-Tree and ES-DLL

To improve efficiency in querying entities and tuples in dynamic datasets, we propose EQB-Tree and ES-DLL structures in the offline data preprocessing step.

Creating EQB-Tree We designed the structure of the EQB-Tree to find the corresponding entity of the tuples to be rapidly updated and to reduce the updating response time. Consider $D = \{e_i \mid i \in [1, N_e]\}$. The e_i node in the EQB-Tree carries important information regarding the entity e_i: $\mathcal{O}(e_i)^T$ =$\{e_i.key$, $e_i.addr$, $e_i.curInfoAddr\}$. In the structure $\mathcal{O}(e_i)^T$, $e_i.key$ represents the set of particular attribute values that help to distinguish e_i from other entities in $\mathcal{E}$, which can be generated by the similarity functions discussed in Ref. [28]. $e_i.addr$ represents the storage address of e_i in the dataset for completely querying all the tuples of e_i and $e_i.curInfoAddr$ represents the storage address of the currency graph in Ref. [2], which maintains the currency orders of the values of the different attributes of entity e_i.

Algorithm 2 presents the procedure for creating the EQB-Tree T_D. First, we initialize the EQB-Tree with the head node (Line 1), then we initialize the *key, addr, curInfoAddr* of the e_i node and insert this node to T_D (Lines 2–7). Lastly, we return T_D as the EQB-Tree for the entities in D (Line 8).

Example 3 Suppose another dataset ***Employee***

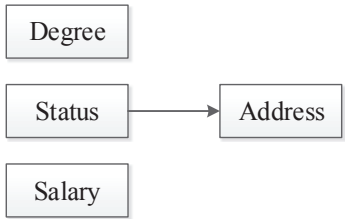


Fig. 2 Topological graph of the processing order of the entity attributes in the motivating example.

Algorithm 2 Bulid EQB-Tree

Input: D
Output: T_D
1: T_D = init Tree();
2: **for** each $e_i \in D$ **do**
3: acquire $e_i.key$ in $\mathcal{O}_{e_i}^T$;
4: initialize $e_i.addr$;
5: initialize $e_i.curInfoAddr$;
6: insert $Node(e_i)$ to T_D by $e_i.key$;
7: **end for**
8: return T_D

containing employee information for a company with the 21 entities that describe the employees {*Alice, Carina, Dean, Edward, George, Harry, July, Kelly, Linda, Mary, Nick, Peter, Qearl, Rose, Sweety, Tom, Victor, Wendy, Xavier, Yilia, Zoe*}. If $e_i.key$ = name, we can create the EQB-Tree of ***Employee***, as shown in Fig. 3. Accordingly, information regarding a certain entity can be efficiently queried using the structure of the EQB-Tree.

Creating ES-DLL After creating the EQB-Tree of the entities, we create and initialize the ES-DLL to efficiently insert new tuples into the dataset. Each node in the ES-DLL carries information regarding the structure $\mathcal{O}(e_i)^L = \{t_{j,e_i}, addr(t_{j+1,e_i}) \mid i \in [1, N_e], j \in [1, N_t]\}$, in which t_{j,e_i} represents one of the tuples in entity e_i, and $addr(t_{j+1,e_i})$ represents the storage address of the next tuple immediately following t_{j,e_i}. If t_{j,e_i} is the last tuple describing entity e_i, then $addr(t_{j+1,e_i}) = -1$. As noted above, the head address $e_i.addr$ is stored in the EQB-Tree, by which all records for e_i in the ES-DLL can be acquired.

Algorithm 3 presents the procedure for creating the ES-DLL L_D. First, we initialize a new external file for L_D (Line 1). Then, for each tuple of each entity in D, we initialize the $e_i.address$ with the current file pointer (Line 4) and each node of L_D can be written with the format t_j, $addr(t_{j+1})$, in which -1 represents the end of entity e_i and the function *getNextTupleAddress*() obtains the sum of the current file pointer and the length occupied by t_j, which is the address t_{j+1} in L_D (Lines 5–10). Lastly, we return L_D as the initiative ES-DLL (Line 13).

Example 4 Consider the above motivating example, we can create a linked list ES-DLL of the two entities *Tom* and *Alice* using Algorithm 3, as shown in Fig. 4.

4.1.3 Creating currency information for entities

Creating currency graphs and obtaining currency information is also a critical step in data preprocessing. Furthermore, currency graphs and scores represent the most important part of dynamic determination, as they provide the correct currency order for the tuples of the

Algorithm 3 Init ES-SLL L_D

Input: D
Output: L_D
1: L_D= init_list();
2: **for** each $e_i \in D$ **do**
3: **for** each $t_i \in \mathcal{T}$ **do**
4: $e_i.address = L_D.getFilePointer()$;
5: **if** t_j is the last tuple about e_i **then**
6: $addr(t_{j+1}) = -1$;
7: **else**
8: $addr(t_{j+1})$= *getNextTupleAddress*();
9: **end if**
10: insert t_j and $addr(t_{j+1})$ to L_D;
11: **end for**
12: **end for**
13: return L_D

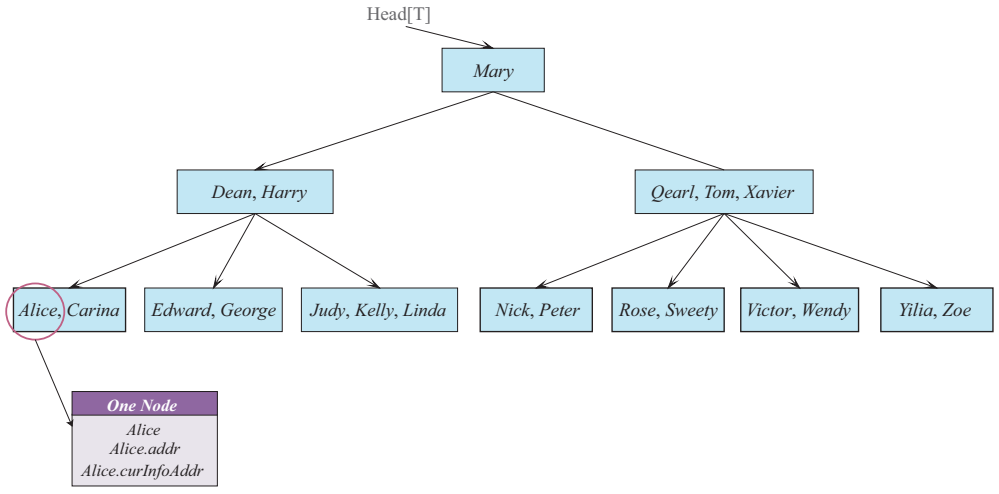


Fig. 3 EQB-Tree constructed in Example 3.

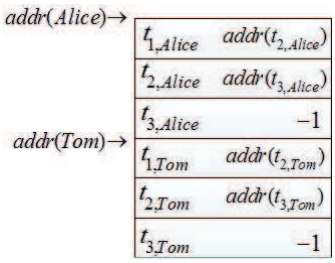


Fig. 4 ES-DLL created in the motivating example.

same entity. Algorithm 4 shows the pseudo code for obtaining entity Currency Information.

In Algorithm 4, first, we obtain currency graphs of the attributes involved in the course of determining the currency (Lines 1–14). After creating the currency graph of e on A (Line 1), we add it to $\mathcal{G}_e$ (Lines 2 and 3), and compute the currency score score_e for e (Lines 4–13). During this process, each tuple (vertex) of e with zero in-degree is added into the queue (Lines 4 and 5). In Lines 6–13, tuples are added with their currency scores, and the fresher is the data, the higher is the score. We must also determine the existence of any conflicts between the currency rules. If there is no tuple with a zero in-degree (Line 10) after all the vertexes and edges in $\mathcal{G}_e$ are deleted (Line 8), we know that loops exist. This indicates that the currency rules Φ used here lead to a conflicting result and Φ will be returned (to domain experts).

Algorithm 4 getCurInfo(e, $\mathcal{T}_e$, Φ, Q_Φ)

Input: the entity e, the tuples' set $\mathcal{T}_e$, the set of currency rules Φ, and the queue of currency dependence order Q_Φ.
Output: curInfo_e

1: $\mathcal{G}_{e,A} \leftarrow$ create $\mathcal{G}$ $(e, \mathcal{T}_e, \Phi)$
2: **for** each $A \in Q_\Phi$ **do**
3: add $\mathcal{G}_{e,A}$ into $\mathcal{G}_e$
4: Queue $\leftarrow \phi$; $count \leftarrow 0$
5: add all $t \in V$ and Indegree$(t) == 0$ into Queue
6: **while** Queue.*noEmpty*() **do**
7: t = Queue.*pop*() and score$(t[A]) \leftarrow$ ++*count*
8: delete t from V and delete (t, t_x) from E // (t, t_x) represents for the edge starting from t.
9: add all $t \in V$ and Indegree$(t) == 0$ into Queue
10: **if** all $t \in V$ and Indegree$(t) \neq 0$ **then**
11: exit (-1) and return Φ
12: **end if**
13: **end while**
14: **end for**
15: **for** each t in $\mathcal{T}_e$ **do**
16: score$(t) \leftarrow \Sigma_{A \in Q_\Phi}$(score$(t[A]) + |A| - |Q_\Phi|$)
17: add score(t) into score_e
18: **end for**
19: $\text{curInfo}_e \leftarrow (G_e, \text{score}_e)$
20: return curInfo_e

In Lines 15–18, the currency score is computed for each tuple in $\mathcal{T}_e$. Some of the attributes in Q_Φ may not be involved in the creation of the currency graph, which means that their currency cannot be determined by Φ or that they have no impact on the currency of the tuple. In these cases, they are given a currency score with the least value (score $=$ 1, generally). Finally, $\mathcal{G}_e$ and score_e are combined as curInfo_e in Line 19.

In Algorithm 4, $O(m \cdot |\mathcal{T}_e| \cdot |\Phi|)$ time is required to create $\mathcal{G}_{e,A}$ (Line 1) as discussed in Ref. [2]. The total time taken by the loop (Lines 2–13) is $O(m \cdot |\mathcal{T}_e|^2)$. Then, the computation of the currency score of e for all the attributes involved the costs $O(|\mathcal{T}_e| \cdot |\Phi|)$ (Lines 15–17). To put this all together, Algorithm 4 works in $O\big(m \cdot |\mathcal{T}_e| \cdot \max\{|\Phi|, |\mathcal{T}_e|\}\big)$ time.

Example 5 Based on Definition 3 and Algorithm 4, we can compute the currency scores of the entity *Alice* in the motivating example as follows: (1) Degree: score(*Bachelor*) = 1, score(*Master*) = 2; (2) Address: score(*3-DP*) = 1, score(*15-DP*) = 2, score(*Beijing*) = 3; (3) Salary: score(40) = 1, score(500) = 2, score(12 000)= 3. We cannot determine the currency orders of other attributes of the entity *Alice* by the currency rules above, so the score of those attributes is 1, which is the minimal positive integer. The scores of each tuple from the same entity all contribute to the currency of the tuple. So we can calculate the sum of the score of each attribute value in *Alice*, except for tID, eID, and Sex as follows: (1) score($t_{1,\ Alice}$) = 1+1+1+1+1+1= 6, score($t_{2,\ Alice}$) = 2+1+1+2+2+1= 9, score($t_{3,\ Alice}$) = 2+1+1+3+3+1= 11. Similarly, the currency score of *Tom* of the attributes involved in Φ are (1) Degree: score(*Bachelor*) = 1, score(*Master*) = 2, score(*PhD*) = 3; (2) Address: score(*1-DP*) = 1, score(*16-DP*) = 2, score(*16-DP*) = 3; (3) Salary: score(40) = 1, score(500) = 2, score(1000) = 3; (4) Status: score(*Single*) = 1, score(*Married*) = 2. Accordingly, we can determine the tuples' currency order as follows: (1) score($t_{1,\ Tom}$) = 1+1+1+1+1+1= 6, score($t_{2,\ Tom}$) = 2+1+1+2+2+1= 9, score($t_{3,\ Tom}$) = 3+1+1+3+3+2 = 13.

From the above, we find the tuple currency order of *Alice* to be $t_{1,\ Alice} \prec t_{2,\ Alice} \prec t_{3,\ Alice}$. Similarly the tuple currency order of *Tom* is $t_{1,\ Tom} \prec t_{2,\ Tom} \prec t_{3,\ Tom}$.

According to the above algorithms, by scanning the initial relation dataset once can we construct an EQB-Tree and ES-DLL for each entity, as well as obtain their currency graphs and scores. We propose the complete

preprocessing approach in Algorithm 5.

First, we initialize the index T_D and the linked list L_D in Line 1, which was introduced in Algorithms 2 and 3. In the loop from Line 2 to 14, the currency information of each entity curInfo$_e$ in the entity set $\mathcal{E}$ is computed by Algorithm 4 and written to the file, with the first address of the score$_e$ in disk memory (Lines 3 and 4). The tuples in $\mathcal{T}_e$ are sorted in ascending order, after which the head address $e.addr$ of the first tuple of e in the ES-DLL is scanned (Lines 5 and 6). Combined with $e.key$ and $e.curInfoAddr$, the node $\mathcal{O}(e)^T$ is inserted to T_D (Lines 7 and 8). In the loop from Lines 9–12, the head address of each tuple t in $\mathcal{T}_e$ and the next tuple immediately following t are inserted into L_D in proper order. The data in L_D are then written to the disk (Line 13). Lastly, the T_D and L_D of all the entities in D are returned.

4.2 Algorithms for currency determining

As mentioned in Section 3.2, after preprocessing the initial relations dataset offline, we can determine the currency information of the entities. Both different entities and tuples that refer to the same entities can be queried efficiently by the EQB-Tree and ES-DLL. We discuss our proposed updating and currency determination methods below.

4.2.1 Updating EQB-Tree and ES-DLL

When updating new tuples of entity e_i, first, we try to match it with an exist entity in the EQB-Tree. If e_i is an new entity not found in $\mathcal{E}$, the tuple is inserted into the file tail of ES-DLL, and the EQB-Tree is rebuilt. If e_i is found in the EQB-Tree of the initial dataset, the tuple is inserted into the tail of the ES-DLL of e_i, and the $e.address$ of e_i in $\mathcal{O}_e^T$ is updated if necessary. Algorithm 6 presents the update procedure for the EQB-Tree and ES-DLL.

First, we determine whether e represents a new entity to the initial dataset and if it does, we construct its $\mathcal{O}_e^L$ and insert it into the $\mathcal{O}_L$ of the dataset D (Lines 3 and 4). Then, the new entity e is inserted into the EQB-Tree with its head address in the ES-DLL (Lines 5 and 6). If e can be matched with a corresponding entity in $\mathcal{E}$, we update the new tuples of the entity into the tuple set $\mathcal{T}_e$ which is recorded in L_D with the loop from Line 8 to Line 20. First, we set the tuple ID number (tID) and entity ID number (eID) for the new tuple t_{new} and set its head file address into L_D in Lines 8 and 9. Thus, we have modified the address in the nodes in the ES-DLL to maintain the currency order. Next, we insert t_{new} into the correct position in T_D. *preTID* represents the current address of t, and if $preTID_{t[\text{new}]}$ is 0, t is inserted into the first line of $\mathcal{T}_e$, and the tuple immediately following t is the first tuple of the previous D, the address of which is recorded in T_D (Lines 10–12). If t_{new} is not the first tuple of e, the address of the next tuple immediately following t will be taken placed by t (Line 14), and the attribute tID of all the tuples

Algorithm 5 Preprocessing

Input: D, Φ, Q_Φ
Output: T_D, L_D
1: T_D := bulid EQB-Tree(); L_D := init ES-DLL();
2: **for** each $e \in \mathcal{E}$ **do**
3: curInfo(e) ← getCurInfo();
4: write curInfo(e) into curInfoFile() and get $e.curInfoAddr$;
5: sort $\mathcal{T}_e$ by $curInfo(e).score$;
6: $e.addr \leftarrow L_D.getFilePointer()$;
7: $\mathcal{O}_e^T \leftarrow$ init *Node* ($e.key$, $e.addr$, $e.curInfoAddr$);
8: $T_D.insert(\mathcal{O}_e^T)$;
9: **for** each $t \in \mathcal{T}_e$ **do**
10: $\mathcal{O}^L := \big(t, L_D.getNextTupleAddr(t)\big)$;
11: $L_D.insert(\mathcal{O}^L)$;
12: **end for**
13: $L_D.writeToDisk$;
14: **end for**
15: return T_D and L_D;

Algorithm 6 Updating ES-DLL and EQB-Tree

Input: e, t_{new}, L_D, T_D
Output: L_D, T_D
1: **if** $e.isNewEntity$ == True **then**
2: $e.addr = L_D.getFilePointer()$;
3: $\mathcal{O}^L := \big((1,|\mathcal{E}|+1, t_{\text{new}}), -1\big)$;
4: $L_D.insert(\mathcal{O}^L)$ and $L_D.writeToDisk()$;
5: $\mathcal{O}_e^T.setAddress(e.addr)$;
6: $T_D.insert(\mathcal{O}_e^T)$;
7: **else**
8: $t := (preTID_{t[\text{new}]}+1, eID, t_{\text{new}})$;
9: $addr(t) \leftarrow L_D.getFilePointer()$;
10: **if** $preTID_{t[\text{new}]} == 0$ **then**
11: $addr(t.next) \leftarrow e.addr$;
12: $\mathcal{O}_e^T.setAddress\big(addr(t)\big)$;
13: **else**
14: $addr(t.next) \leftarrow \mathcal{O}_e^L.setNexTupleAddr\big(addr(t)\big)$;
15: **end if**
16: **for** each $\mathcal{O}_{e,i}^L$, $i \in \big[preTID_{t[\text{new}]}+1, |\mathcal{E}|\big]$ **do**
17: $\mathcal{O}_{e,i}^L.t.eID$ +=1;
18: **end for**
19: $\mathcal{O}_e^L := \big(t, addr(t.next)\big)$;
20: $L_D.insert(\mathcal{O}^L)$ and $L_D.writeToDisk()$;
21: **end if**
22: return L_D, T_D

following t will increase by 1 (Lines 16 and 17). Then, we update the $\mathcal{O}^L$ of e with t in L_D (Lines 19 and 20).

Example 6 Consider Table 1 in the motivating example, we create a dynamic linked list of the two entities *Tom* and *Alice*, as presented in Fig. 5 according to Algorithm 6, and we efficiently update a new tuple t_{new} of *Tom* to the ES-DLL without moving any previous tuples. In addition, the data volume does not influence this update operation.

4.2.2 Updating currency graphs

Next, we discuss the process of updating the currency information when inserting a new tuple t_{new} into $\mathcal{T}_e$. First, we search the same entity in t_{new} using the keyword in the EQB-Tree index. After identifying the corresponding entity e from t_{new}, we update curInfo_e with the new information from t_{new}. When there is no identical entity recognized by t_{new} in the original dataset, we know that t_{new} describes a new entity. It is then inserted into the dataset, the EQB-Tree is updated, and the currInfo of e is initialized, accordingly.

After the entity e represented by t_{new} is recognized, the currency information currInfo is updated with t_{new}. Algorithm 7 presents the procedure for updating the currency graph for all the involved attributes of the entity e .

When the t_{new} describing e appears, it is added to $\mathcal{G}(e, A_k)$ as a new node. In the loop (Lines 1–9), the currency graph of e on A_k is determined based on the currency information curInfo_e (Line 2). Then, the t_{new} values of A_k are inserted into the vertex set of $\mathcal{G}(e, A_k)$, and both the in-degree and out-degree are initialized (Lines 3 and 4). In the nested-loop (Lines 5–8), based on each φ_{A_k} in Φ_{CR}, we can determine $t \prec_{A_k} t_{\text{new}}$ or $t_{\text{new}} \prec_{A_k} t$, and then the edges will be added into the graph correspondingly. Finally, it returns a new currency graph $\mathcal{G}(e, A)$.

We expect the loop between Lines 1 and 9 to execute $|Q_\Phi|$ times in total. The currency graphs are stored in

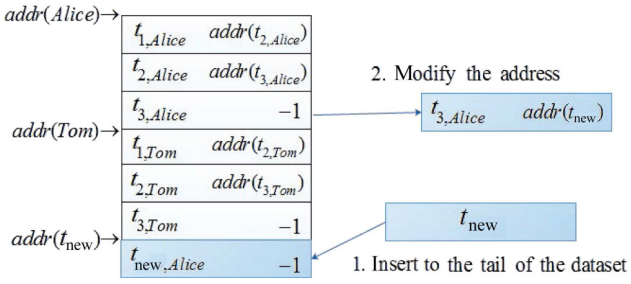


Fig. 5 The ES-DLL updated with the new tuple in the motivating example.

Algorithm 7 Updating currency graph $\mathcal{G}(e, A)$

Input: $t_{\text{new}}, \mathcal{G}(e) = (V_{A_k}, E), \Phi_{CR}$
Output: the new $\mathcal{G}(e, A)$ after updating

1: **for** each $A_k \in Q_\Phi$ **do**
2: get $\mathcal{G}(e, A_k)$ from $curInfo.\mathcal{G}_e$
3: add $\mathcal{T}_e.t_{\text{new}}[A_k]$ into $\mathcal{G}(e, A_k).V$
4: Outdegree($t_{\text{new}}[A_k]$) $\leftarrow 0$ and Indegree($t_{\text{new}}[A_k]$) $\leftarrow 0$
5: **for** each $t \in \mathcal{T}_e$, $t \prec_{A_k} t_{\text{new}}$ or $t_{\text{new}} \prec_{A_k} t$ can be inferred from $\varphi_{A_k} \in \Phi_{\text{CR}}$ **do**
6: add (t, t_{new}) or (t_{new}, t) into $\mathcal{G}(e, A_k).E$
7: Outdegree(t or t_{new}) += 1 and Indegree(t_{new} or t) += 1
8: **end for**
9: **end for**
10: return $\mathcal{G}(e, A)$

Hash access, with a cost of $O(1)$ in Line 2. We also expect that examining the satisfiability of one currency rule costs r time at most, so then the nested-loop (Lines 5–8) costs $r \cdot |\Phi|$. In total, the time complexity of Algorithm 7 is $O\big(r \cdot |\Phi| \cdot |Q_\Phi|\big)$.

4.2.3 Updating currency scores

After updating the currency graphs, we can update the currency score of entity e recognized by t_{new}, as shown in Algorithm 8. We then calculate the new currency score based on the functions of $\text{score}\big(t_l[A_k]\big)$, where $t_l[A_k]$ represents the tuple that has an edge (currency relationship order) with t_{new} on A_k in the currency graph. We note that in the directed graph $\mathcal{G}(e, A_k)$, $\big(t_l[A_k], t_{\text{new}}[A_k]\big)$ is an edge starting from $t_l[A_k]$, and vice versa.

If both the in-degree and out-degree values of $t_{\text{new}}[A_k]$ are 0, this means that the currency of $t_{\text{new}}[A_k]$ cannot be determined according to the rules in Φ_{CR}, so we initialize the score $\text{score}\big(t_{\text{new}}[A_k]\big)$ to 1 (Lines 2 and 3). When $t_{\text{new}}[A_k]$ has only out-neighbors, it becomes the most current, so the score of $t_{\text{new}}[A_k]$ is equal to the max score of $t_l[A_k]$ plus 1 (Lines 4 and 5). If $t_{\text{new}}[A_k]$ only has in-neighbors, we determine $t_{\text{new}}[A_k]$ to be the least current, whereupon we examine the minimum score in $\text{score}\big(t_l[A_k]\big)$. If the minimum scores in $(t_l[A_k])$ is greater than 1, the new $t_{\text{new}}[A_k]$ score equals the minimum score minus 1 (Lines 7 and 8), and if not, $\text{score}\big(t_{\text{new}}[A_k]\big)$ becomes 1, and the score of all $t_l[A_k]$ are increased by 1, accordingly (Lines 9–12). In other cases, the current value of $t_{\text{new}}[A_k]$ is neither the highest nor the lowest current. Obviously, the score is expected to be one point higher than the maximum score of $t_l[A_k]$ with an edge $\big(t_l[A_k], t_{\text{new}}[A_k]\big)$ (Line 16). Also, the score of the tuples $t_m[A_k]$ with an edge $\big(t_{\text{new}}[A_k], t_m[A_k]\big)$ (more current than $t_{\text{new}}[A_k]$) must be

Algorithm 8 Updating the currency score of entity e with t_{new}

Input: $t_{\text{new}}, \mathcal{G}(e, A_k) = (V_{A_k}, E)$, score_e
Output: the new score_e after updating

1: **for** each $A_k \in Q_\Phi$ **do**
2: **if** $\text{Indegree}(t_{\text{new}[A_k]}) = 0 \wedge \text{Outdegree}(t_{\text{new}[A_k]}) = 0$ **then**
3: $\text{score}(t_{\text{new}[A_k]}) \leftarrow 1$
4: **else if** $\text{Indegree}(t_{\text{new}[A_k]}) \neq 0 \wedge \text{Outdegree}(t_{\text{new}[A_k]}) = 0$ **then**
5: $\text{score}(t_{\text{new}[A_k]}) \longleftarrow (\max_{1 \leqslant l \leqslant N} \text{score}(t_l[A_k] \mid (t_l, t_{\text{new}}) \in E)) + 1$
6: **else if** $\text{Indegree}(t_{\text{new}[A_k]}) = 0 \wedge \text{Outdegree}(t_{\text{new}[A_k]}) \neq 0$ **then**
7: **if** $\min_{1 \leqslant l \leqslant N} \text{score}(t_l[A_k] \mid (t_{\text{new}}, t_l) \in E) > 1$ **then**
8: $\text{score}(t_{\text{new}[A_k]}) \longleftarrow \big(\min_{1 \leqslant l \leqslant N} \text{score}(t_l[A_k] \mid (t_{\text{new}}, t_l) \in E)\big) - 1$
9: **else**
10: $\text{score}(t_{\text{new}[A_k]}) = 1$
11: **for** each l from 1 to N **do**
12: $\text{score}\big(t_l[A_k] \mid (t_{\text{new}}, t_l) \in E\big) += 1$
13: **end for**
14: **end if**
15: **else**
16: $\text{score}(t_{\text{new}}[A_k]) \longleftarrow \max_{1 \leqslant l \leqslant N_{\text{in}}} \text{score}(t_l[A_k] \mid (t_{\text{new}}, t_l) \in E) + 1$
17: **for** each m from 1 to N_{out} **do**
18: $\text{score}(t_m[A_k]) += 1$
19: **end for**
20: **end if**
21: **end for**
22: $\text{score}_e \leftarrow (\sum_{A \in Q_\Phi} \text{score}(t_{\text{new}})) + |A| - |Q_\Phi|$
23: add $\text{score}(t_e)$ into curInfo_e.score
24: return score_e

one point higher than before (Lines 17 and 18). Then, we calculate the new score of e in Line 22 and update curInfo with the new score (Line 23). Lastly, we return the new score_e after updating (Line 25).

In the algorithm, the whole loop (Lines 1–21) is executed $|Q_\Phi|$ times. In this loop, it costs $O(1)$ time to obtain the number of both the in-degree and out-degree of $t_{\text{new}}[A_k]$, and the calculation of the *maximum* or *minimum* scores is executed in $O(N)$ $\big(O(|\mathcal{T}_e|)\big)$. The time costs of the loop in Lines 11 and 12 and Lines 17 and 18 are $O(N)$ and $O(N_{\text{out}})$, respectively. As such, the whole loop costs $O\big(|Q_\Phi| \cdot |\mathcal{T}_e|\big)$. In Lines 22 and 23, it takes $O\big(|Q_\Phi|\big)$ time to determine the total currency score of e and $O(1)$ time to update the score of the entity in curInfo. In total, the time complexity of Algorithm 8 is $O\big(|Q_\Phi| \cdot |\mathcal{T}_e|\big)$.

Example 7 Suppose that there are five tuples for the entity *Mary* in ***Info***. We take into consideration the attribute Degree regarding the educational background of the person via the following currency rule:

φ : $\forall t_1, t_2, t_3, t_4, (t_1[\text{Status}] = \textit{Bachelor}, t_2[\text{Status}] = \textit{Master}, t_3[\text{Status}] = \textit{PhD}, t_1[\text{Status}] = \textit{PostDoc} \rightarrow t_1 \prec_{\text{Degree}} t_2 \prec_{\text{Degree}} t_3 \prec_{\text{Degree}} t_4)$.

Figure 6 shows the new currency graph of *Mary* for Degree when a new tuple appears regarding *Mary* with t[Degree] = *Master* comes. To clarify here, we show only the edges connected with t_{new}. According to φ, we establish the currency graph $\mathcal{G}_{\text{Mary, Degree}}$, and the in-degree and out-degree of the node t_{new} are 3 and 2, respectively. As shown in Algorithm 8, $\text{score}(t_{\text{new}}[\text{Degree}]) = \text{score}(t_2[\text{Degree}]) + 1 = 3$. And the scores of tuples more current than t_{new}, namely t_4, t_5 are as follows: $\text{score}(t_4[\text{Degree}]) = \text{score}(t_5[\text{Degree}]) = \text{score}(t_{\text{new}}[\text{Degree}])' + 1 = 4$.

Example 8 Returning to the motivating example, when inserting the new tuple t_{new} into the correct location in ***Info***, according to the currency rules (r_1 to r_4), we compute the currency scores of t_{new} for each relevant attributes as follows: score (*Master*) = 2, score (*Programmer*) = 2, score (*Shanghai*) = 4, score (*Married*) = 2, score (12 000) = 3. So $\text{score}(t_{\text{new}})$ = 2+2+4+3+2 = 13. Therefore, we obtain $t_{1,\,Alice} \prec t_{2,\,Alice} \prec t_{3,\,Alice} \prec t_{\text{new}}$. The tuple t_{new} is then inserted right after $t_{3,\,Alice}$ in the order in which the tuples describing the same entity *Alice* maintain the correct data currency.

5 Experiments

In this section, we present our experiments, which we conducted on both real and synthetic datasets. In Section 5.1, we introduce our experimental settings. In Sections 5.2 through 5.5, we analyze the influence

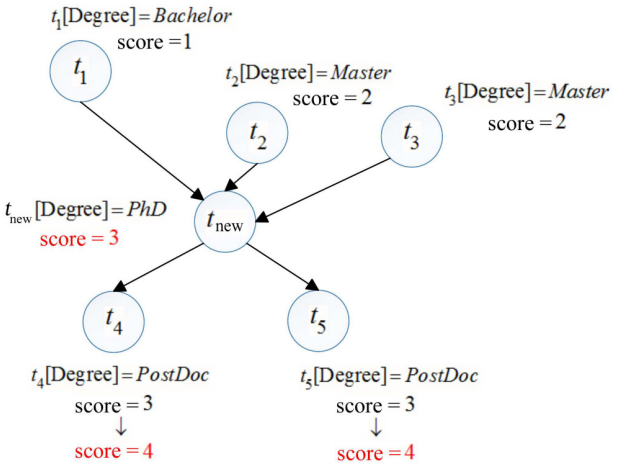


Fig. 6 The currency graph and score of *Mary* on the attribute Degree.

of four main parameters to determine the efficiency of the offline preprocessing and the online dynamic determination for each dataset.

5.1 Experimental settings

We ran the experiment on a computer with an Inter(R) 3.40 GHz Core i5 CPU and 8 GB of RAM, using Java in Eclipse.

5.1.1 Experimental data

We based the experiments on a real-life dataset, ***Student data***, which contains the personal information of 10 000 students both during their time in college and after graduation. The data schema in this dataset is as follows: {tID, eID, Name, Sex, Degree, Position, College, Address, Salary, Age}. To evaluate the efficiency of the algorithms on large-scale data, the synthetic data adhered to the same schema as that in ***Student data***. To effectively evaluate the impact of various parameters (listed below), we generated entities and the tuple number of each entity under different conditions.

5.1.2 Currency determining implementation

To identify the currency rules adopted in the experiments for both the real and synthetic data, we used the methods proposed in Ref. [23]. In addition, the semantic constraints of these currency rules also satisfy the definitions in Refs. [2, 23].

5.1.3 Algorithms

We implemented the following algorithms in both the preprocessing and dynamic determination steps of the model. As such, preprocessing generates the processing order of the attributes (Algorithm 1), creates the EQB-Tree (Algorithm 2) and the ES-DLL (Algorithm 3), and obtains currency information (Algorithm 4). Dynamic determination involves updating the currency graph and scores (Algorithms 7 and 8) and updating the EQB-Tree and ES-DLL (Algorithm 6).

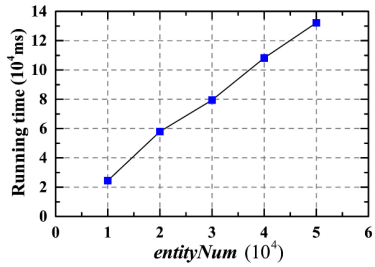
Here, we discuss the efficiency of four main parameters with respect to the algorithms, including the total number of entities, attributes, tuples referring to the same entity, and currency rules. The preprocessing running time includes the time spent creating and initializing the EQB-Tree and ES-DLL for all entities of the dataset and obtaining currency information regarding the entities. The dynamic determination running time includes the running times of Algorithms 7 and 8, which update the EQB-Tree and ES-DLL (Algorithm 6), respectively. We discuss situations during the determination process in which new tuples are inserted into different locations of the ES-DLL.

5.2 Impact of the total number of entities

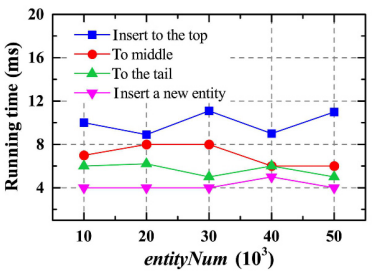
First, we evaluated the parameter *entityNum* with respect to the efficiency of the algorithms during both preprocessing and currency dynamic determination, with the conditions $attrNum = 10$, $tupleNum = 10$, $ruleNum = 40$.

In total, we processed 50 000 entities in the experiment. As the number of entities increased linearly, the preprocessing running time also increased linearly, as shown in Fig. 7a. During dynamic determination, we take into account four possible positions for updating a new tuple: insertion at the top, middle, and tail of the ES-DLL, and the insertion of a new entity.

As shown in Fig. 7b, the response time for updating a new tuple during the dynamic determination step is not affected by a change in the number of entities. The reason for this is that only a single scan is made over the whole dataset during preprocessing. This also validates Algorithm 7 (updating currency graph $\mathcal{G}(e, A_k)$) and Algorithm 8 (updating currency score score_e). The determination process runs no more than 12 ms. In addition, inserting the new tuple at the top of the linked list takes only a little more time than inserting it into either the middle or tail. Also, in cases such as these, it takes a minimum amount of time to insert the tuple into $\mathcal{E}$ if the tuple to be updated is a new entity.



(a) Time cost of preprocessing



(b) Time cost of currency determining

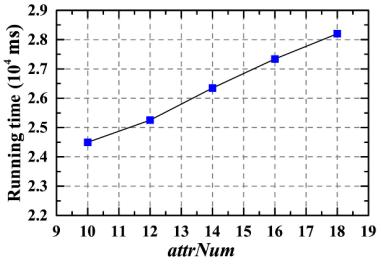
Fig. 7 ***entityNum*** **effects on efficiency.**

5.3 Impact of the number of attributes

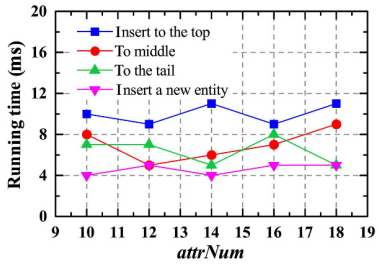
When experimentally evaluating the number of attributes, we generated at most eight attributes for the same tuple in ***Student Data***. Figure 8 shows the efficiency of the algorithms with different numbers of attributes and the following condition: *entityNum* = 10 000, *tupleNum* = 10, *ruleNum* = 40. As shown in Fig. 8a, the preprocessing running time increases linearly. With the small-scale increase in the number of attributes, the preprocessing running time increases gradually, and the response time during the dynamic determination of currency is also not affected by an increasing number of attributes in the tuples. With respect to updating the ES-DLL, inserting the new tuple at the top of the ES-DLL takes the most time, but is no more than 13 ms, as shown in Fig. 8b.

5.4 Impact of the number of tuples

Taking into account the number of tuples, we generated a maximum of 40 tuples for the same number of attributes in one entity. Figure 9 shows the effect of the number of tuples on the efficiency of the preprocessing and dynamic determination steps for the conditions *entityNum* = 10 000, *attrNum* = 10, *ruleNum* = 40. As shown in Fig. 9a, with a linear increase in the number of tuples in a given entity, the preprocessing time increases polynomially. It takes about 70 s to preprocess a dataset containing 500 000 tuples in a total of about 10 000 people. On the other hand, the dynamic

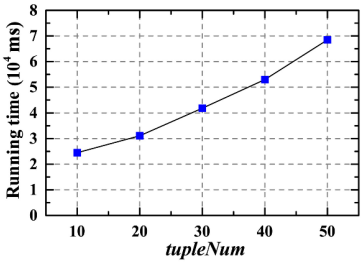


(a) Time cost of preprocessing

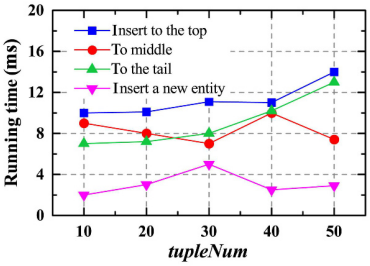


(b) Time cost of currency determining

Fig. 8 ***attrNum*** **effects on efficiency.**



(a) Time cost of preprocessing



(b) Time cost of currency determining

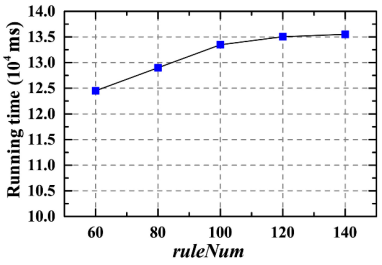
Fig. 9 ***tupleNum*** **effects on efficiency.**

updating time is independent of the total number of tuples, and takes no more than 14 ms. As shown in Fig. 9b, it takes less time to insert a tuple describing a new entity than it does to insert one into the other three positions in the linked list.

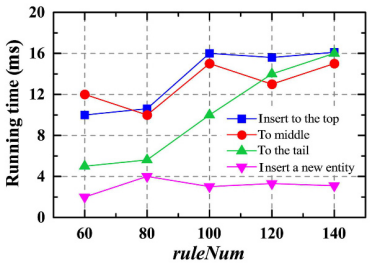
5.5 Impact of the number of currency rules

The number of currency rules used in the determination process also has an effect on the efficiency. With the other parameters set as follows: *entityNum* = 10 000, *attrNum* = 10, *tupleNum* = 10, we computed the running time for the total number of currency rules varying from 60 to 140, as shown in Fig. 10. From Fig. 10a, we can see that as the number of currency rules increases linearly, the preprocessing time also increases linearly at first, and then becomes stable once the total number of currency rules exceeds 90. The reason for this is that the number of attributes affects only the preprocessing time. In this experiment, when we held *attrNum* = 10 constant, the increase in the total number of currency rules either added to the semantic duplications in the tuples of a given entity or between different entities. Similarly, this setting tends to lead to a stable ES-DLL creation process time after the number of currency rules reaches a certain threshold.

In dynamic determination, the change in the total number of currency rules has almost no influence on the updating of a new entity. For the other three cases, the running time increases slightly as the number of



(a) Time cost of preprocessing



(b) Time cost of currency determining

Fig. 10 ***ruleNum* effects on efficiency.**

rules increases up to 100, and thereafter tends to take the same amount of time as when $ruleNum = 100$, as shown in Fig. 10b. That stable time cost tendency is consistent with the preprocessing result shown in Fig. 10a.

6 Conclusion

In this paper, we studied the dynamic determination of the currency of large-scale data and proposed a dynamic data currency model consisting of offline preprocessing and online dynamic determination. We designed and implemented several algorithms to optimize the attribute currency order of tuples referring to the same entity and to create and query currency information of entities. Using just a single scanning of the initial dataset, we can achieve the efficient determination and updating of data currency. In addition, we found the response time to be uninfluenced by the data scale within an appropriate range. In a set of reasonable experiments, we verified that our methods and algorithms are effective in determining the currency of dynamic data.

Acknowledgment

This paper was partially supported by the National Natural Science Foundation of China (Nos. U1509216 and 61472099), National Key Technology Research and Development Program (No. 2015BAH10F01), the Scientific Research Foundation for the Returned Overseas Chinese Scholars of Heilongjiang Province (No. LC2016026), and MOE-Microsoft Key Laboratory of Natural Language Processing and Speech, Harbin Institute of Technology.

References

[1] W. Fan, F. Geerts, S. Ma, N. Tang, and W. Yu, *Data Quality Problems beyond Consistency and Deduplication.* Springer Berlin Heidelberg, 2013, pp. 237–249.

[2] M. H. Li, J. Z. Li, and H. Gao, Evaluation of data currency, (in Chinese), *Chinese Journal of Computers*, vol. 35, no. 11, pp. 2348–2360, 2012.

[3] W. Fan, F. Geerts, and X. Jia, Conditional dependencies: A principled approach to improving data quality, in *British National Conference on Databases: Dataspace: the Final Frontier*, 2009, pp. 8–20.

[4] T. N. Herzog, F. J. Scheuren, and W. E. Winkler, *Data Quality and Record Linkage Techniques*. Springer Science & Business Media, 2007.

[5] W. Fan, F. Geerts, and J. Wijsen, Determining the currency of data, *Acm Transactions on Database Systems*, vol. 37, no. 4, pp. 71–82, 2012.

[6] M. Li and J. Li, A minimized-rule based approach for improving data currency, *Journal of Combinatorial Optimization* , vol. 32, no. 3, pp. 812–841, 2016.

[7] Y. Shen, B. Guo, Y. Shen, X. Duan, X. Dong, and H. Zhang, A pricing model for big personal data, *Tsinghua Science and Technology*, vol. 21, no. 5, pp. 482–490, 2016.

[8] C. Batini, C. Cappiello, C. Francalanci, and A. Maurino, Methodologies for data quality assessment and improvement, *ACM Computing Surveys*, vol. 41, no. 3, pp. 75–79, 2009.

[9] T. C. Godfrey, *Data Quality for the Information Age.* Artech House, Inc., 1996.

[10] R. Y. Wang and D. M. Strong, Beyond accuracy: What data quality means to data consumers, *Journal of Management Information Systems,* vol. 12, no. 4, pp. 5–33, 1996.

[11] Q. Gorz, An economics-driven decision model for data quality improvement—A contribution to data currency, in *Proc. 17th Americas Conference on Information Systems (AMCIS)*, Detroit, MI, USA, 2011, pp. 1–8.

[12] B. Heinrich and M. Klier, Assessing data currency—A probabilistic approach, *Journal of Information Science*, vol. 37, no. 1, pp. 86–100, 2011.

[13] C. Cappiello, C. Francalanci, and B. Pernici, A model of data currency in multi-channel financial architectures, in *International Conference on Information Quality*, 2002, pp. 106–118.

[14] B. Heinrich, M. Klier, and M. Kaiser, A procedure to develop metrics for currency and its application in CRM, *Journal of Data and Information Quality*, vol. 1, no. 1, pp. 1–28, 2009

[15] B. Heinrich and D. Hristova, A fuzzy metric for currency in the context of BIG DATA, in 22*nd European Conference*

on Information Systems (ECIS), 2014.

[16] C. Cappiello, C. Francalanci, and B. Pernici, Time related factors of data accuracy, completeness, and currency in multi-channel infor-mation systems, in *The Conference on Advanced Information Systems Engineering*, 2003, pp. 145–153.

[17] L. Bertossi, Consistent query answering in databases, *ACM Sigmod Record Homepage*, vol. 35, no. 2, pp. 68–76, 2006.

[18] J. Chomicki, Consistent query answering: Five easy pieces, in *Database Theory – ICDT 2007, International Conference,* Barcelona, Spain, January 10–12, 2007, pp. 1–17.

[19] X. L. Dong, L. Berti-Equille, and D. Srivastava, Truth discovery and copying detection in a dynamic world, *Proceedings of the Vldb Endowment*, vol. 2, no. 1, pp. 562–573, 2009.

[20] Y. Cao, W. Fan, and W. Yu, Determining the relative accuracy of attributes, in *ACM SIGMOD International Conference on Management of Data,* 2013, pp. 565–576.

[21] W. Fan, F. Geerts, N. Tang, and W. Yu, Inferring data currency and consistency for conflict resolution, in *2013 IEEE 29th International Conference on Data Engineering (ICDE)*, Brisbane, Australlia, 2013, pp. 470–481.

[22] W. Fan, J. Li, S. Ma, N. Tang, and W. Yu, Interaction between record matching and data repairing, in *ACM SIGMOD International Conference on Management of Data*, Athens, Greece, ACM, 2011, pp. 469–480.

[23] W. Fan, F. Geerts, N. Tang, and W. Yu, Conflict resolution with data currency and consistency, *Journal of Data and Information Quality*, vol. 5, nos. 1&2, pp. 1–37, 2014.

[24] X. Ding, H. Wang, Y. Gao, J. Li, and H. Gao, Determining the currency of dynamic data, in *Proceedings of the 2017 ACM TUR-C Conference,* ACM, 2017.

[25] P. Christen, A survey of indexing techniques for scalable record linkage and deduplication, *IEEE Transactions on Knowledge and Data Engineering*, vol. 24, no. 9, pp. 1537–1555, 2011.

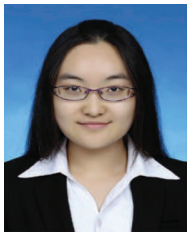
[26] M. Bodirsky and J. Kara. The complexity of temporal constraint satisfaction problems, in *ACM Symposium on Theory of Computing,* Victoria, British Columbia, Canada, 2008.

[27] H. Wang, J. Li, and H. Gao, Efficient entity resolution based on subgraph cohesion, *Knowledge and Information Systems*, vol. 46, no. 2, pp. 285–314, 2016.

[28] A. K. Elmagarmid, P. G. Ipeirotis, and V. S. Verykios, Duplicate record detection: A survey, *IEEE Transactions on Knowledge and Data Engineering*, vol. 19, no. 1, pp. 1–16, 2007.

Xiaoou Ding is currently a PhD student in School of Computer Science and Technology, Harbin Institute of Technology. She received the bachelor degree from Harbin Institute of Technology in 2015. Her research interests include data quality, data repairing and cleaning, and big data management.

Hongzhi Wang is a professor and doctoral supervisor at Harbin Institute of Technology. He was awarded Microsoft fellowship, Chinese excellent database engineer and IBM PhD fellowship. His research interests include big data management, data quality, graph data management, and web data mangement.



Yitong Gao received the master degree from Harbin Institute of Technology in 2016. Her research interests include big data management, data quality, and distributed algorithms.

Jianzhong Li is a professor and doctoral supervisor at Harbin Institute of Technology. He is a senior member of CCF. His research interests include database, parallel computing, wireless sensor networks, etc.

Hong Gao is a professor and doctoral supervisor at Institute of Technology. She is a senior member of CCF. Her research interests include database, parallel computing, wireless sensor networks, etc.