PAPER

Currency Preserving Query: Selecting the Newest Values from Multiple Tables

Mohan LI[†a)], *Nonmember and* **Yanbin SUN**[†b)], *Member*

SUMMARY In many applications, tables are distributively stored in different data sources, but the frequency of updates on each data source is different. Some techniques have been proposed to effectively express the temporal orders between different values, and the most current, i.e. up-to-date, value of a given data item can be easily picked up according to the temporal orders. However, the currency of the data items in the same table may be different. That is, when a user asks for a table *D*, it cannot be ensured that all the most current values of the data items in *D* are stored in a single table. Since different data sources may have overlaps, we can construct a conjunctive query on multiple tables to get all the required current values. In this paper, we formalize the conjunctive query as currency preserving query, and study how to generate the minimized currency preserving query to reduce the cost of visiting different data sources. First, a graph model is proposed to represent the distributed tables and their relationships. Based on the model, we prove that a currency preserving query is equivalent to a terminal tree in the graph, and give an algorithm to generate a query from a terminal tree. After that, we study the problem of finding minimized currency preserving query. The problem is proved to be NP-hard, and some heuristics strategies are provided to solve the problem. Finally, we conduct experiments on both synthetic and real data sets to verify the effectiveness and efficiency of the proposed techniques.

key words: *data quality, data currency, referential integrity constraints*

1. Introduction

Low quality data can severely impact on the usability of data, and may lead to huge losses. For example, the poor quality of customer data costs the U.S. businesses $611 billion a year [1]. One of the issues that is worth paying attention to is that with the elapse of time the quality of data deteriorates rapidly [2], that is, some values which are once correct may become wrong since the real world is changing. A survey revealed that 2% of the records in a common customer database will become obsolete in one month. The problem is so-called data currency problem, and low currency has been considered as one of the important factors bringing down the quality of data [3].

In many circumstances, such as data integration or web databases, many tables are distributively stored in different data sources. These data sources may have overlaps, but can hardly cover the entire required data set [4], and the frequency of updates on each data source might be different. In some applications, the tables which are less important may not be updated as frequently as the important ones for the purpose of reducing the cost. This situation happens especially in the case of managing big data. Moreover, copying between data sources is common in practice [5], [6], that is, some sources obtain their data by copying from other sources. If a source *S* obtains data by copying from source S', the copied values provided by *S* may be less up-to-date than the value in S'. Consequently, a data source *S* may have obsolete values either because the values are less important, or because the values are copied from other sources.

To get all the most current values of a target data item, some techniques have been proposed to find the temporal orders between different values. On one hand, the copy detection methods are proposed [5], [7]–[9], and the detected copy relationships can be considered as the temporal orders between different data sources. On the other hand, a rule based model of data currency has been studied [3], [10]. Using the rules, the temporal orders of the values can be inferred, and the most current value can be easily picked up according to the temporal orders.

Unfortunately, we cannot always ensure that there is a single table containing all the most current values of a target table. Therefore, we may need to construct a conjunctive query on multiple tables to get all the required current values. We name this kind of query currency preserving query. The following example illustrates the motivation of finding a currency preserving query.

Example 1 *Figure 1 shows an example of multiple tables and the corresponding temporal orders.* D_1 *to* D_5 *are five tables containing the information of employees.* D_1 *is the target table required by users, and* D_2 *to* D_5 *are the tables stored on other data sources. The overlaps of different tables are the columns with the same name. Assume that the values of Addr and Sal in* D_1 *are obsolete since they are not updated timely.*

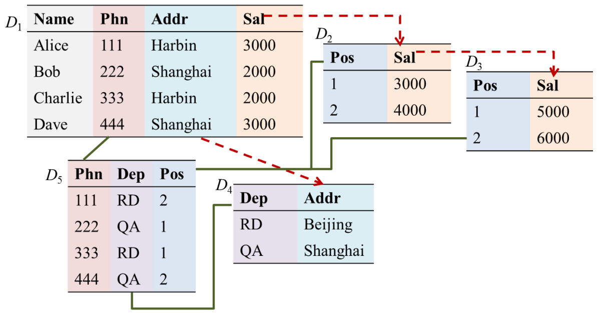
The corresponding temporal orders are shown by the red dashed arrows. According to the temporal orders, we have following conclusions. (1) The values of Sal in D_1 *are* ***no more up-to-date*** *than the values of Sal in* D_2*. (2) The values of Sal in* D_2 *is no more up-to-date than the values of Sal in* D_3*. (3) The values of Addr in* D_1 *is no more up-to-date than the values of Addr in* D_4*. Therefore, we need to query* D_3 *and* D_4 *to get the most up-to-date value of Addr and Sal when a user asked for table* D_1*.*

However, to construct the query, the temporal orders can only indicate which table have the newest values, but

Manuscript received February 7, 2018.
Manuscript revised July 1, 2018.
Manuscript publicized August 31, 2018.

[†]The authors are with the Guangzhou University, Guangzhou, P.R. China.
a) E-mail: limohan.hit@gmail.com
b) E-mail: yanbin_hit@foxmail.com (Corresponding author)

DOI: 10.1587/transinf.2018EDP7058



Name	Phn	Addr	Sal
Alice	111	Harbin	3000
Bob	222	Shanghai	2000
Charlie	333	Harbin	2000
Dave	444	Shanghai	3000

Pos	Sal
1	3000
2	4000

Pos	Sal
1	5000
2	6000

Phn	Dep	Pos
111	RD	2
222	QA	1
333	RD	1
444	QA	2

Dep	Addr
RD	Beijing
QA	Shanghai

Fig. 1 An example of multiple tables and the currency orders

cannot tell us the correspondences of those values. For instance, we do not know the salary 5000 *in* D_3 *corresponds to which employee in* D_1. *Therefore, we need to leverage other information to ensure that the newest values scattered in* D_1, D_3 *and* D_4 *can be correctly combined together.*

Assume that we have four referential integrity constraints expressed by inclusion dependencies [11], which are

$$D_1[Phn] \subseteq D_5[Phn], \qquad D_5[Dep] \subseteq D_4[Dep],$$
$$D_5[Pos] \subseteq D_2[Pos], \qquad D_5[Pos] \subseteq D_3[Pos].$$

These constraints indicate the relationships of foreign keys, and are shown by the green solid lines in Fig. 1. Please note that for brevity we assume that the attributes in the left hand side and right hand side have the same names, but the assumption is not necessary in practice. The referential integrity constraints can be given by domain experts, and some techniques, such as mappings relationships discovering [12] and data polygamy [13], can help us to derive the possible referential integrity constraints.

Using the temporal orders and the dependencies, we can construct a currency preserving query Q *which returns all the most up-to-date values of* D_1, *that is,*

$$ans(X_1, X_2, X_3, X_4) \leftarrow R_1(X_1, X_2, Y_1, Y_2), R_3(Y_3, X_4), R_4(Y_4, X_3), R_5(X_2, Y_4, Y_3),$$

where R_i *is the schema of* D_i. *The entities (i.e. employees) in the result of* Q *is the same as those in* D_1, *but the values are the most up-to-date ones.*

Some works focus on the query rewriting considering the access restrictions and integrity constraints [14]–[16]. These works can give the proofs of answerability of a query when the search space is limited by access restrictions and can be extended through the integrity constraints. However, the temporal orders between data sources are quite different from the access restrictions. because the access restrictions indicate two states, 0 or 1, but the temporal orders indicate partial orders. More precisely, the temporal orders do not restrict the condition of accessing a value, but determine the priorities of selecting a value, that is, we always want to select a value that is more up-to-date than other values. Therefore, these works cannot be used to solve the problem of generating currency preserving query.

In this paper, we focus on the two key challenges of generating the currency preserving query, that is, how to handle the temporal orders and how to leverage the referential integrity constraints. The main contributions are as follows.

1. We formally define the currency preserving query. Given a target table, the currency preserving query aims to return all the most up-to-date values of the target table.
2. We propose a graph model to represent the temporal orders and the referential integrity constraints between different tables. The currency preserving query is proved to be equivalent to a special tree, namely terminal tree. An algorithm for generating currency preserving query from a terminal tree is also given.
3. We study the problem of generating the minimized currency preserving query. We prove that the problem is equivalent to finding a minimized terminal tree in the graph, and is NP-hard. Two heuristic strategies are provided to solve the problem. Moreover, we discuss two general cases of the graph model, and provide the strategies of dealing with those cases.
4. We conduct experiments on both synthetic and real data sets to verify the effectiveness and efficiency of the proposed techniques.

The paper is organized as follows. Section 2 formally defines the currency preserving query. Section 3 builds the graph model and give the algorithm of transforming a terminal tree to the corresponding currency preserving query. Section 4 studies the problem of generating the minimized currency preserving query. Section 5 experimentally evaluates our approach. Section 6 discusses the related work, and Sect. 7 concludes the paper.

2. Currency Preserving Query

2.1 Preliminaries

Let $\mathbb{R} = \{R_1, \ldots, R_n\}$ be a set of schemas, where $R_i = (A_1, \ldots, A_{|R_i|})$ is a set of attributes. $\mathbb{D} = \{D_1, \ldots, D_n\}$ is an instance of $\mathbb{R}$, and the schema of the ith table D_i is R_i. For each tuple $t \in D_i$, $t \in dom(R_i[A_1]) \times \ldots \times dom(R_i[A_{|R_i|}])$, where $R_i[A_j]$ is the attribute belonging to the ith schema with name A_j, (e.g. $R_1[Sal]$ is the attribute "*Sal*" in schema R_1) and $dom(R_i[A_j])$ represents the domain of $R_i[A_j]$.

We say that the attribute (a.k.a. column) A_j of table D, denoted by $D[A_j]$, describes a set of objects. For example, the object represented by *Addr* of the first tuple in D_1 of Fig. 1 is the address of the Alice, and objects represented by of $D_1[Addr]$ are the addresses of the employees.

A conjunctive query [11] Q is in the form of

$$ans(\vartheta) \leftarrow R_{c_1}(\vartheta_1), \ldots, R_{c_h}(\vartheta_h).$$

where the symbols are defined as follows:

- $ans(\vartheta)$ is called the head of Q, denoted by $head(Q)$, and ϑ is a sequence of variables which are called head variables;
- the right-hand side of the symbol $\leftarrow$ is the body of Q, denoted by $body(Q)$, where R_{c_i} is a schema belongs to $\mathbb{R}$, and ϑ_i is a sequence of variables, which are called body variables;
- each variable occurring in ϑ must also occur at least once in $\vartheta_1, \ldots, \vartheta_h$.

Please note that in general definition of conjunctive query, constants can also occur in $\vartheta, \vartheta_1, \ldots, \vartheta_h$, but here we do not consider constants since they will not be used in solving our problem.

Temporal orders. Some columns in different tables may describe the same set of objects. Since the frequencies of updates are not all the same, these columns may not be consistent in data currency. That is, we may find temporal orders between these columns. Temporal order is a partial order, and is represented by the symbol $\preceq$ [3]. $D_i[A] \preceq D_j[B]$ means that the column A of D_i and the column B of D_j describe the same set of objects, and A is no more up-to-date than B. More precisely, for each value a in $D_i[A]$, $D_j[B]$ contains a value that is either equal to or more up-to-date than a. We assume that the temporal orders have the property of transitivity, and the symbols $\prec$ and $\asymp$ represent the relationships of strictly less and equal, respectively. Let Σ be the set of temporal orders with the form $D_i[A] \preceq D_j[B]$, where A and B are two attributes in R_i and R_j. As is discussed in Sect. 1, Σ can be effectively obtained by the previous works [3], [5], [7]–[10], thus we consider Σ as an input of our problem.

Referential integrity constraints. As is shown in Example 1, we need referential integrity constraints to link the newest values scattered in different tables. Inclusion dependencies (INDs for short) are used to formalize the constraints. The syntax of an IND is $R_i[\alpha] \subseteq R_j[\beta]$, where $\alpha \subseteq R_1$ and $\beta \subseteq R_2$ are two subsets of attributes. $\mathbb{D}$ satisfies $R_i[\alpha] \subseteq R_j[\beta]$ if for any tuple $t_1 \in D_i$ there exists a tuple $t_2 \in D_j$ such that $t_1[\alpha] = t_2[\beta]$. The set of referential integrity constraints defined on $\mathbb{R}$ is denoted by Ψ. To simplify the discussion, we assume that the primary key of any R_i has only one attribute, that is, for each $R_i[\alpha] \subseteq R_j[\beta] \in \Psi$, $|\alpha| = 1$ and $|\beta| = 1$. By combining the primary key attributes, our methods can deal with the cases that the primary key has more than one attributes.

2.2 Problem Definition

The most current values can be found according to the temporal orders in Σ. Since each end of $\preceq$ is an attribute in $\mathbb{R}$, we can define the most current comparable attribute to indicate the position of the most current values.

Definition 1 (Most Current Comparable Attribute) *Given $\mathbb{R}$, $\mathbb{D}$, $A \in R_i$ and $A' \in R_j$, we say that $D_i[A]$ and $D_j[A']$ are comparable attributes if $D_i[A] \preceq D_j[A']$ or $D_j[A'] \preceq D_i[A]$ according to the temporal orders in Σ. $D_i[A]$ is a comparable attribute of itself.*

$D_i[A]$ and $D_j[A']$ are equivalent comparable attributes if $D_i[A] \preceq D_j[A']$ and $D_j[A'] \preceq D_i[A]$, that is, $D_i[A] \asymp D_j[A']$.

$D_j[A']$ is the <u>most current comparable attributes</u> of $D_i[A]$ if $D_i[A] \preceq D_j[A']$, and one of the following two conditions are satisfied.

- *$\nexists A'' \in R_k$ such that $D_j[A'] \preceq D_k[A'']$, or*
- *for each $D_k[A'']$ satisfying $D_j[A'] \preceq D_k[A'']$, we have $D_j[A'] \asymp D_k[A'']$.*

Intuitively, Most Current Comparable (MCC for short) attributes are the "maximal" attributes in the temporal orders. If $D_j[A']$ is an MCC attributes of $D_i[A]$, then any other attributes can only have three relationships with $D_j[A']$, that is, (1) not comparable, (2) not up-to-date than $D_j[A']$, or (3) equivalent comparable with $D_j[A']$. We use $MCC(D_i[A])$ to represent the set of MCC attributes of $D_i[A]$. Thus, if a user requires the values of $D_i[A]$, the most up-to-date values which can be found in $\mathbb{D}$ is the values of $MCC(D_i[A])$. Please note that $D_i[A]$ may have multiple MCC attributes. If a user requires a table D_T with schema R_T, then, under the ideal situation, we need to generate a new table D_T^c which contains all the most up-to-date values corresponding to D_T. More precisely, D_T^c should satisfy following two conditions: (1) with schema R_T, (2) the values of D_T^c are from the MCC attributes of D_T.

Our work aims to construct a query to find D_T^c for a given table D_T. The query is named Currency Preserving Query (CPQ for short). Given $\mathbb{R}$, $\mathbb{D}$, Σ, Ψ and D_T, a currency preserving query can be found in two steps: (1) find the MCC attributes for each attribute in D_T, (2) use the referential integrity constraints to link all the MCC attributes. Now we first use an example to illustrate the intuitive idea of generating CPQ, then give a formal definition of the CPQ generating problem.

Example 2 *Consider $\mathbb{D} = \{D_1, D_2, D_3, D_4, D_5\}$ in Fig. 1, assume that $D_T = D_1$, $R_1 = \{Name, Phn, Addr, Sal\}$. Remind that the red dash lines indicate three temporal orders $D_1[Sal] \preceq D_2[Sal]$, $D_2[Sal] \preceq D_3[Sal]$ and $D_1[Addr] \preceq D_4[Addr]$. Thus we have*

$$MCC(D_1[Name]) = \{D_1[Name]\},$$
$$MCC(D_1[Phn]) = \{D_1[Phn]\},$$
$$MCC(D_1[Addr]) = \{D_4[Addr]\},$$
$$MCC(D_1[Sal]) = \{D_3[Sal]\}.$$

First, we link $MCC(D_1[Name])$, $MCC(D_1[Phn])$ and $MCC(D_1[Addr])$. According to the green solid lines which indicate the INDs $D_1[Phn] \subseteq D_5[Phn]$ and $D_5[Dep] \subseteq D_4[Dep]$, we first use D_1 and D_5 to perform a equal join on Phn, then use the result and D_4 to perform an equal join on Dep. After that, according to $D_5[Pos] \subseteq D_3[Pos]$, we continue to join the result and D_3 on Pos to link $D_1[Sal]$. Now all the MCC attributes are linked, and the query is

$$ans(X_1,X_2,X_3,X_4) \leftarrow R_1(X_1,X_2,Y_1,Y_2), R_3(Y_3,X_4), R_4(Y_4,X_3), R_5(X_2,Y_4,Y_3).$$

It is the same one shown by Example 1.

In Example 2, two tables can perform an equal join only on the attributes contained by an IND. The two attributes occurring in the same IND are called joinable attributes. In other words, $D_i[A]$ and $D_j[B]$ are the joinable attributes if there exists an IND $R_i[A] \subseteq R_j[B]$ in Ψ.

Base on the comparable attributes and joinable attributes, the currency preserving query can be defined as follows.

Definition 2 (Currency Preserving Query) *A currency preserving query Q_{MCC} is a conjunctive query,*

$$ans(\vartheta) \leftarrow R_{c_1}(\vartheta_{c_1}), \dots, R_{c_h}(\vartheta_{c_h}),$$

which satisfies following conditions.

1. *The number of variables in ϑ is $|R_T|$.*
2. *Let $\vartheta^{(i)}$ represent the ith variable in ϑ, then for an arbitrary attribute $A_i \in R_T$, there exists one attribute $A_{c_jk} \in R_{c_j}$ such that (1) R_{c_j} occurs in $body(Q_{MCC})$, (2) $D_{c_j}[A_{c_jk}] \in MCC(D_T[A_i])$, (3) $\vartheta^{(i)} = \vartheta^{(k)}_{c_j}$, that is, the kth variable in ϑ_{c_j} is same as the ith variable in ϑ.*
3. *For each $R_{c_{j_1}}$ and $R_{c_{j_2}}$ ($j_1 \neq j_2$) in $body(Q_{MCC})$, there must exist $R_{c_{i_1}}, \dots, R_{c_{i_k}}$ in $body(Q_{MCC})$ such that (1) $j_1 = i_1$, (2) $j_2 = i_k$, (3) $R_{c_{i_l}}$ and $R_{c_{i_{l+1}}}$ have at least one pair of joinable attributes.*
4. *If there exists two body variables $\vartheta^{(k)}_{c_i} = \vartheta^{(l)}_{c_j}$, then $D_{c_i}[A_{c_ik}]$ and $D_{c_j}[A_{c_jl}]$ are joinable attributes.*

In Definition 2, the first and second conditions ensure that the result have R_T columns and the ith column corresponds to an MCC attribute of the ith attribute of D_T. The third condition guarantees that all the schemas in $body(Q_{MCC})$ are not isolated, that is, any two tables have to be linked by a sequence of joinable attributes. The fourth condition restricts that two tables can only perform join on a pair of joinable attributes.

The problem of this paper is that given a target table D_T with schema R_T, how to use Ψ and Σ to get a CPQ Q_{MCC} for D_T such that executing Q_{MCC} on $\mathbb{D}$ will return D^c_T. The formal definition is as follows.

Input: D_T, R_T, $\mathbb{R} = \{R_1, \dots, R_n\}$, $\mathbb{D} = \{D_1, \dots, D_n\}$, Σ and Ψ.

Output: a CPQ Q_{MCC}.

In following sections, we first give the graph model representing the temporal orders and INDs, then provide a method of generating CPQ based on the model and study the problem of minimized CPQ.

3. A Graph Model for Generating CPQ

3.1 Schema Currency Graph

Schema Currency Graph (SCG for short) is defined for representing the temporal orders and INDs. The formal definition is as follows.

Definition 3 (Schema Currency Graph) *A schema currency graph $G_{\mathbb{R},\mathbb{D},\Sigma,\Psi} = (V,(E_1,E_2,E_3))$ is constructed based on $\mathbb{R}$, $\mathbb{D}$, Σ and Ψ, where V is the node set, E_1, E_2 and E_3 are three disjoint edge sets. The detailed definitions of V, E_1, E_2, and E_3 are as follows.*

- $|V| = |\mathbb{R}| = \sum_{i=1}^{n} |R_i|$. *The nodes in V have one-to-one correspondence to the attributes in $\mathbb{R}$, and the attribute corresponding to node v is denoted by $attr(v)$. More precisely, $attr(v) = R_j[A_i]$ if v corresponding to $A_i \in R_j$.*
- *E_1 is the schema edge set consisting of undirected edges for linking the attributes belonging to the same schemas. $\forall v_i, v_j \in V$, a schema edge $(v_i, v_j) \in E_1$ only if $attr(v_i)$ and $attr(v_j)$ belong to the same schema R_k.*
- *E_2 is the temporal edge set consisting of directed edges denoting the temporal orders. $\forall v_i, v_j \in V$, assume that $attr(v_i) = R_k[A]$, and $attr(v_j) = R_l[B]$, there exists a temporal edge (v_i, v_j) from v_i to v_j in E_2 if $\exists D_k[A] \preceq D_l[B] \in \Sigma$. There exists two temporal edges $(v_i, v_j), (v_j, v_i) \in E_2$ if $\exists D_k[A] \asymp D_l[B] \in \Sigma$.*
- *E_3 is the dependency edge set consisting of undirected edges denoting the INDs. $\forall v_i, v_j \in V$, a dependency edge $(v_i, v_j) \in E_3$ if there exists $attr(v_i) \subseteq attr(v_j) \in \Psi$.*

Please note that the set of three types of edges in SCG can be empty, which are special cases of SCG. For example, a graph without schema edges is a special cases of SCG. The situation behind this kind of SCG is that all schemas have only one attribute. In other words, each schema only has one attribute and thus each table only has one column. In this case, there are no two attributes belonging to the same schema, so there is no schema edge in the NSCG.

According to the definition of schema edges, the simplest way to organize the schema edges is to make the attributes in the same schema comprise a clique. The organization of schema edges can be substituted for other definitions according to practical working conditions, i.e., the attributes in the same schema can also comprise a path, or a tree, or any other structures.

Example 3 *Consider the case that the columns of a table has an access constraint "ϕ: if you want to access both A and B, you must access C". Specifically, A and B can indicate the patient's name and illness, and C indicates the authorization status. Only the authorized doctor can see the patient's name and disease at the same time. In this case, a better way is to organize A, B and C as a path shown by Fig. 2 rather than a clique. C is in the path between A and*

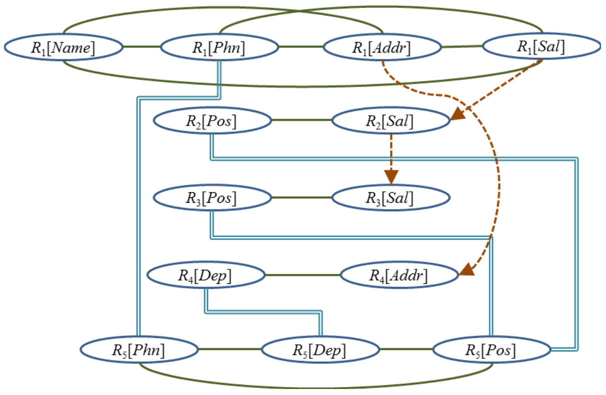


Fig. 3 Schema currency graph, SCG

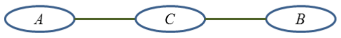


Fig. 2 An example of schema edges of A, B and C

B to ensure that C must in the TerTree if both A and B are in the TerTree, that is, to ensure the access constraint ϕ to be true.

However, enumerating different practical working conditions is beyond the topic of this paper, thus we only use the simple structure, i.e. clique, to represent the nodes belonging to the same schema. Since the attributes and the nodes have one-to-one correspondence, we will alternately use the terminologies "attribute" and "node" on the basis of the context in the rest part of this paper.

Figure 3 shows the corresponding SCG of Fig. 1, where the green solid single lines represent the schema edges, the brown dash arrows represent the temporal edges, and the blue solid double lines represent dependency edges.

It is easy to observe that the paths comprised by temporal edges can be used to determine the temporal orders of the attributes on the path. Thus we can further define temporal connected component to indicate which attributes are equivalent comparable attributes.

- v_i is temporal reachable from v_j if there is a path comprised only by temporal edges from v_j to v_i in SCG $G_{\mathbb{R},\mathbb{D},\Sigma,\Psi}$. For example, $R_3[Sal]$ is temporal reachable from $R_1[Sal]$ in Fig. 3, but $R_1[Sal]$ is not temporal reachable from $R_3[Sal]$.
- v_i and v_j are temporal connected if they are temporal reachable from each other. Assume that $attr(v_i) = R_k[A]$ and $attr(v_j) = R_l[B]$, if v_i and v_j are temporal connected, then we have $D_k[A] \preceq D_l[B]$ and $D_l[B] \preceq D_k[A]$ according to the transitivity of $\preceq$, that is, $D_k[A] \asymp D_l[B]$.
- A subgraph G' of $G_{\mathbb{R},\mathbb{D},\Sigma,\Psi}$ is a temporal connected component if $\forall v_i$, v_j in G', v_i and v_j are temporal connected.
- G' is maximal temporal connected component if $\nexists G'' \neq G'$ such that (1) G' is a subgraph of G'', and (2) G'' is a temporal connected component.

For each $A \in R_T$, the MCC attributes corresponds to the farthest temporal reachable node from A, which can be expressed by Terminal Node (TN for short) in SCG.

Definition 4 (Terminal Node) *Given SCG $G_{\mathbb{R},\mathbb{D},\Sigma,\Psi} = (V, (E_1, E_2, E_3))$, $v \in V$ is a terminal node if and only if $\exists v' \in V$ s.t. $attr(v')$ is in R_T and one of the following conditions holds.*

1. *$v \neq v'$, v is temporal reachable from v', and either (a) v's out-degree of temporal edges is 0, or (b) for each v'', if v'' is temporal reachable from v, then v and v'' are temporal connected.*
2. *$v = v'$, and for each v'', if v'' is temporal reachable from v, then v and v'' are temporal connected.*

Theorem 1 *Assume that node v_A corresponds to $R_i[A]$, and $v_{A'}$ corresponds to $R_T[A']$, v_A is a TN of $v_{A'}$ if and only if $D_i[A] \in MCC(D_T[A'])$.*

Proof 1 *$\Rightarrow$: If v_A satisfies condition 1, then $v_A \neq v_{A'}$ and $D_T[A'] \preceq D_i[A]$. According to the transitivity of $\preceq$, condition 1(a) in the definition of TN means that and $\nexists A''$ s.t. $D_T[A] \preceq D_i[A'']$. If condition 1(a) is not satisfied, condition 1(b) will ensure that for each $D_T[A] \preceq D_i[A'']$, we have $D_T[A] \asymp D_i[A'']$. If v_A satisfies condition 2, then $v_A = v_{A'}$ (i.e., $R_i[A] = R_T[A]$) and an arbitrary $D_i[A'']$ satisfying $D_T[A] \preceq D_i[A'']$ is an equivalent comparable attribute of $R_T[A]$. According to Definition 1, $D_i[A] \in MCC(D_T[A'])$.*

$\Leftarrow$: If $D_i[A] \in MCC(D_T[A'])$, there are two mutually exclusive cases: (a) $\nexists D_j[A']$ such that $D_i[A] \preceq D_j[A']$, or (b) for each $D_j[A']$ which satisfies $D_i[A] \preceq D_j[A']$, we have $D_i[A] \asymp D_j[A']$. For case (a), the out-degree of the node corresponding to $D_i[A]$ must be 0, thus the condition 1(a) in the definition of TN is satisfied. For case (b), $D_i[A] \asymp D_j[A']$ means temporal connected in the SCG. Therefore, if $D_i \neq D_T$ then condition 1(b) of the definition of TN is satisfied, otherwise condition 2 of the definition of TN is satisfied.

Given an attribute A in R_T, the corresponding TNs can be easily found by scanning the SCG once. Based on the TNs, we can further refine SCG to get Normalized Schema

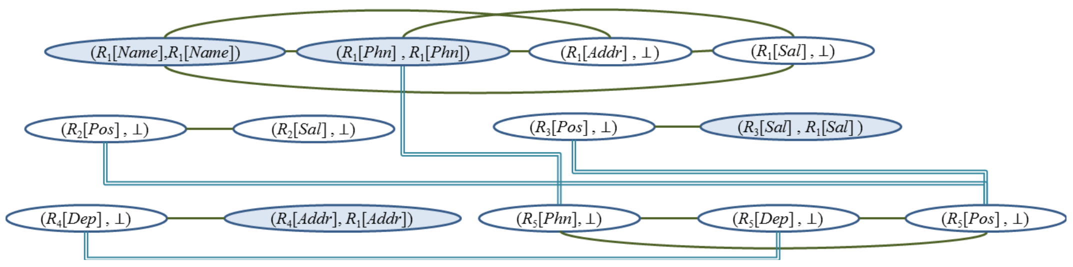


Fig. 4 Normalized schema currency graph, NSCG

Currency Graph (NSCG for short).

Definition 5 (Normalized Schema Currency Graph) *The normalized schema currency graph $G_{\mathbb{R},\mathbb{D},\Sigma,\Psi,N} = (V,(E_s, E_d))$ is built based on SCG $G_{\mathbb{R},\mathbb{D},\Sigma,\Psi}$.*

- $|V| = |\mathbb{R}| = \sum_{i=1}^{n} R_i$, *and each node $v \in V$ corresponds to an attribute $attr(v)$ in $\mathbb{R}$. Every $v \in V$ has a label $(attr(v), ter(v))$, where $attr(v)$ can be considered as the identifier of v, $ter(v)$ represents whether v is the a TN, if $\exists v' \in R_T$ s.t. v is a TN of v', then $ter(v) = v'$, otherwise $ter(v) = \perp$.*
- *E_s equals to the schema edge set of $G_{\mathbb{R},\mathbb{D},\Sigma,\Psi}$, and (v_i, v_j) is a schema edge is denoted by $(v_i, v_j)_/ \in E_s$, where the subscript "/" indicates the type of the edge.*
- *E_d equals to the dependency edge set of $G_{\mathbb{R},\mathbb{D},\Sigma,\Psi}$, and (v_i, v_j) is a dependency edge is denoted by $(v_i, v_j)_{//} \in E_d$, where "//" indicates the type of the edge.*

Figure 4 shows the corresponding NSCG of Fig. 3. The four colored nodes indicate the TNs, and the other nodes are ordinary nodes. Single lines represent the schema edges, and double lines represent the dependency edges.

3.2 The Algorithm of Generating CPQ

To simplify the discussion, first we assume that each attribute of R_T corresponds to a single TN. In Sect. 4, we will further discuss the case of multiple TNs.

Since the TN is unique for each attribute, there are exactly $|R_T|$ TNs in a given NSCG. If all the TNs are connected either by schema edges or by dependency edges, then we can find a tree which contains all the TNs. The tree is called Terminal Tree (TerTree for short).

Definition 6 (Terminal Tree) *Let ρ be a tree in NSCG, the node set of ρ is denoted by V_ρ. ρ is a terminal tree if and only if ρ contains exactly $|R_T|$ TNs, and $ter(\rho) = R_T$, where $ter(\rho) = \{ter(v) | v \in V_\rho, ter(v) \neq \perp\}$.*

Example 4 *In Fig. 4, the edges $(R_1[Phn], R_1[Phn])$, $(R_1[Phn], R_5[Phn])$, $(R_5[Phn], R_5[Dep])$, $(R_5[Dep], R_4[Dep])$, $(R_4[Dep], R_4[Addr])$, $(R_5[Phn], R_5[Pos])$, $(R_5[Pos], R_3[Pos])$, $(R_3[Pos], R_3[Sal])$ and the corresponding nodes comprise a TerTree. It is easy to observe that a NSCG can have multiple TerTrees.*

Algorithm 1 Generating CPQ from a TerTree

```
Input: TerTree $\rho$
Output: a query $Q_\rho$
1: $Head(Q_\rho) = ans(X_1, \ldots, X_{|R_T|})$ //Initialize the head of the query
2: for each $v \in V_\rho$ do
3:     if $attr(v) = R_{c_i}[A_{c_i j}]$ then
4:         if $R_{c_i}$ do not belong to the body of $Q_\rho$ then
5:             Add $R_{c_1}(Y_{c_1 1}, Y_{c_1 2}, \ldots, Y_{c_1 |R_{c_1}|})$ to the body of $Q_\rho$
6:         $cor(v) = Y_{c_i j}$ //$cor(v)$ is the corresponding variable of $v$
7: for each terminal node $u \in V_\rho$ do
8:     if $ter(u)$ is the $k$th attribute of $R_T$ then
9:         change $cor(v)$ into $X_k$
10: for each $v, v' \in V_\rho$ do
11:     if $\exists e \in E_{\rho_d}$ and $cor(v') \neq cor(v)$ then
12:         //$E_{\rho_d}$ is the dependency edge set of $\rho$
13:         change $cor(v')$ into $cor(v)$
```

A TerTree connects all the TNs, and each TN corresponds to an MCC attribute of D_T. Remind that the aim of finding CPQ is to link all the MCC attributes of R_T, thus we can generate a CPQ from a TerTree. Here we provide an method for generating a query from a TerTree, and prove that the query generated by our method is a CPQ.

The method for generating a query from a TerTree is shown by Algorithm 1. Each step is explained as follows. Initially let $head(Q_\rho) = ans(X_1, X_2, \ldots, X_{|R_T|})$ and $body(Q_\rho)$ be empty. Then, for each $v \in V_\rho$, $R_{c_i}(Y_{c_i 1}, Y_{c_i 2}, \ldots, Y_{c_i |R_{c_i}|})$ is added to $body(Q)$ if $attr(v) = R_{c_i}[A_{c_i j}]$. The variable $Y_{c_i j}$ corresponds to $R_{c_i}[A_{c_i j}]$. For each $v \in V_\rho$, if $ter(v) = R_T[A_k]$ and $attr(v) = R_{c_i}[A_{c_i j}]$, then change variable $Y_{c_i j}$ into X_k. Finally, for each pair of nodes connected by a dependency edge (i.e., joinable attributes), change their corresponding variables to be the same one. The loop in Line 1 needs to check each pair of nodes, thus the time complexity of Algorithm 1 is $O(|V_\rho|^2)$. Following theorem ensures the correctness of Algorithm 1.

Theorem 2 *The output of Algorithm 1 is a CPQ.*

Proof 2 *We prove that Q_ρ satisfies all the conditions in Definition 2. In Line 1, the head variables are set to be $X_1, \ldots, X_{|R_T|}$ and do not be changed in following steps, thus condition 1 of Definition 2 is satisfied. Theorem 1 indicates that each TN corresponds to an MCC attribute of R_T. The loops in Line 1 ensures that all the MCC attributes (i.e., TN) exists in $body(Q_\rho)$. The loops in Line 1 ensures that for each $A_k \in R_T$, A_k's corresponding variable is the same to its MCC attributes. Thus, condition 2 in Definition 2 is sat-*

isfied. Since ρ is a tree, all the nodes must be connected. Thus, there do not exists isolated attributes, the condition 3 of Definition 2 is satisfied. Line 1 set all the variables to be different at the beginning, and two body variable is set to be same one only if they are joinable variable in Line 1. Therefore, condition 4 in Definition 2 is satisfied. In conclusion, all the conditions of Definition 2 is satisfied, Q_ρ is a CPQ.

Example 5 illustrates how to generate a CPQ from TerTree.

Example 5 *Consider the NSCG in Fig. 4 and the TerTree ρ in Example 4. The CPQ is generated as follows.*

1. *head(Q_ρ) is set to be $ans(X_1,X_2,X_3,X_4)$ and body(Q_ρ) is empty.*
2. *The attributes involved in ρ belong to the schemas R_1, R_3, R_4, R_5, thus we add $R_1(Y_{11},Y_{12},Y_{13},Y_{14})$, $R_3(Y_{31}, Y_{32})$, $R_4(Y_{41},Y_{42})$ and $R_5(Y_{51},Y_{52},Y_{53})$ are to body(Q_ρ).*
3. *For each terminal node v, i.e. v satisfying $ter(v) \neq \perp$, we set cor(v) the same to cor(ter(v)), where cor(v) is the corresponding variable of v. That is, we change $Y_{11},Y_{12},Y_{32},Y_{42}$ into X_1,X_2,X_4,X_3, respectively. Now, body(Q_ρ)=$R_1(X_1,X_2,Y_{13},Y_{14})$,$R_3(Y_{31},X_4)$,$R_4(Y_{41},X_3)$, $R_5(Y_{51},Y_{52},Y_{53})$.*
4. *We change the variables corresponding to the pairs of nodes connected by dependency edges into the same one. That is, we change Y_{51},Y_{52},Y_{53} into X_2,Y_{41},Y_{31}, respectively. Now, body(Q_ρ) = $R_1(X_1,X_2,Y_{13},Y_{14})$, $R_3(Y_{31},X_4)$,$R_4(Y_{41},X_3)$,$R_5(X_2,Y_{41},Y_{31})$.*

Finally, we get a CPQ

$$ans(X_1,X_2,X_3,X_4) \leftarrow R_1(X_1,X_2,Y_{13},Y_{14}),R_3(Y_{31},X_4), R_4(Y_{41},X_3),R_5(X_2,Y_{41},Y_{31})$$

It is the same one with Example 1 and 2 if we substitute the variables $Y_{13},Y_{14},Y_{31},Y_{41}$ by Y_1,Y_2,Y_3,Y_4, respectively.

Since the INDs indicate referential integrity constraints, i.e. the foreign key relationships, Ψ does not contain two INDs $R_i[A] \subseteq R_j[B]$ and $R_k[A'] \subseteq R_l[B']$ in such that $i = k$ and $j = l$. In this case, we can prove that a CPQ can be equally transformed into a TerTree. Here we first provide the method of transforming Q into a graph denoted by G_Q, then prove that G_Q is a TerTree in NSCG. The transforming method is as follows. First, initialize each schema R_{c_i} in *body*(Q) to a set of nodes $v_1,\dots,v_{|R_{c_i}|}$, and add a schema edge $(v_i,v_{i+1})_/$ for each neighboring pair of v_i,v_{i+1} to G_Q. Each node corresponds to a variable in R_{c_i}. Assume that the jth body variable y in R_{c_i} corresponds to a node v_y. If y is different with all the other variables, then the label of v_y is $(R_{c_i}[A_{c_ij}],\perp)$. If y is same to the kth head variable, then the label of v_y is $(R_{c_i}[A_{c_ij}],R_T[A_k])$. If y is same to the hth body variable of R_{c_l} corresponding to node v'_y, then the label of v_y is $(R_{c_i}[A_{c_ij}],\perp)$, and a dependency edge between $(v_y,v'_y)_{//}$ is added to G_Q.

Theorem 3 *Given CPQ Q, G_Q is a TerTree if there do not exist two INDs $R_i[A] \subseteq R_j[B]$ and $R_k[A'] \subseteq R_l[B']$ in Ψ such that $i = k$ and $j = l$.*

Proof 3 *It is obvious that the edge set of G_Q is a subset of the corresponding NSCG. According the generating method of G_Q, the nodes corresponding to same schema is lined as a path. Condition 3 of Definition 2 ensures that all the nodes in G_Q is connected, thus G_Q is a connected subgraph of the NSCG. Moreover, each pair of schemas only have one pair of joinable attributes, since there do not exist two INDs $R_i[A] \subseteq R_j[B]$ and $R_k[A'] \subseteq R_l[B']$ in Ψ such that $i = k$ and $j = l$, Therefore, G_Q is acyclic. Consequently, G_Q is acyclic connected subgraph of NSCG, that is, G_Q is a tree.*

According to Definition 2, each attribute in R_T has an MCC attribute in body(Q), the generating method of G_Q ensures that each of these MCC attributes corresponds to a TN in G_Q. Thus, G_Q contain all the TNs in the corresponding NSCG, since the MCC attribute of each attribute is unique. In summary, G_Q is a TerTree.

According to Theorem 2 and 3, CPQ and TerTree are transformable from each other. Therefore, to find a CPQ, we can first find a TerTree in NSCG, then use Algorithm 1 to get a CPQ.

4. Minimized Currency Preserving Query

4.1 Strategies of Finding Minimized CPQ

Given a NSCG, there might be more than one TerTrees. Different CPQs may be generated from these TerTrees. Since a CPQ needs to be performed on different columns of multiple tables, obtaining these columns may lead to high cost. In many applications, especially on the column-oriented database management systems, a better choice is that we only obtain the related columns for a CPQ. More concretely, the columns of MCC attributes and joinable attributes is needed, but the other nodes do not have to be in CPQ. Thus, the reading and transmission expense will be much lower than loading the whole database, and D_T^c can be quickly returned. Therefore, we need to study the problem of minimized CPQ.

Definition 7 (Minimized CPQ) *A CPQ Q is a minimal CPQ if and only if both of the following conditions hold. (1) Q is a CPQ. (2) $\nexists Q'$ such that Q' is a CPQ, and $BC(Q') \subset BC(Q)$, where $BC(Q)$ is the set of MCC attributes and joinable attributes in the schemas in body(Q).*

Q is a minimized CPQ if and only if both of the following conditions hold. (1) Q is a minimal CPQ. (2) $\nexists Q'$ such that Q' is a CPQ, and $|BC(Q')| < |BC(Q)|$.

Since we have proved the correspondence of CPQ and TerTree, we can try to find a TerTree ρ_m which has the least nodes, and use ρ_m to generate a CPQ. Therefore, finding

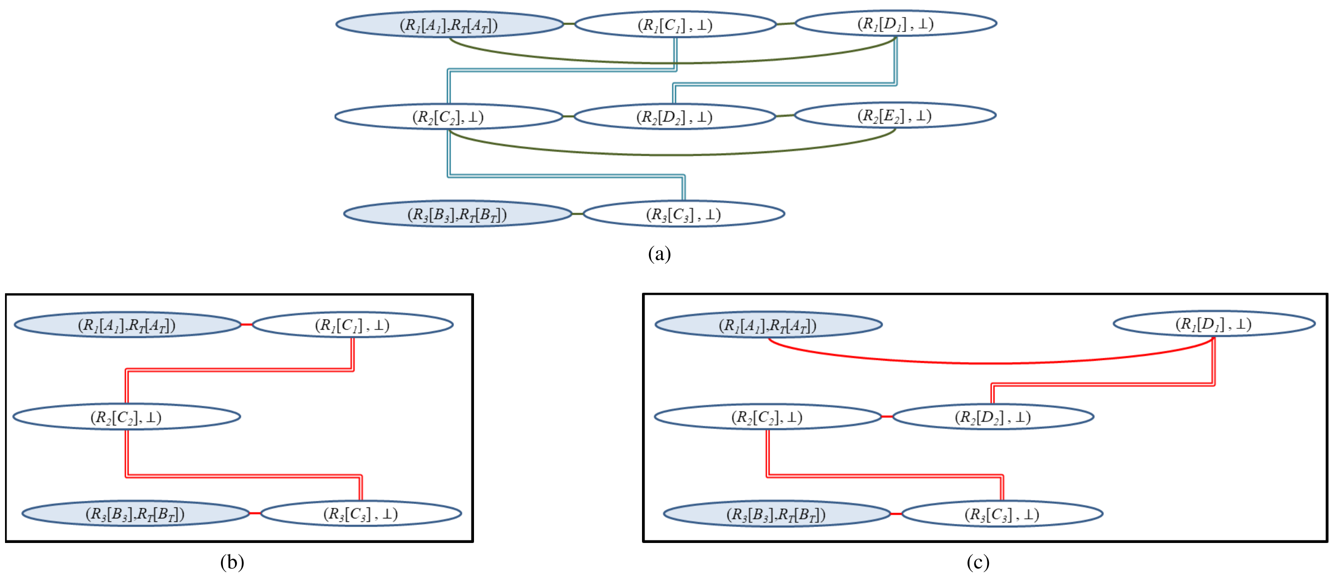


Fig. 5 An example that the dependency edges are not bridges

a minimized CPQ can be transformed into finding a minimized TerTree. The minimized TerTree problem (MTTP for short) is defined as follows.

Input: A NSCG $G_{\mathbb{R},\mathbb{D},\Sigma,\Psi,N}$,
Output: A TerTree ρ_m s.t. there do not exist a tree in $G_{\mathbb{R},\mathbb{D},\Sigma,\Psi,N}$ satisfying $|V_{\rho'}| < |V_{\rho_m}|$.

Theorem 4 *The minimized TerTree problem is NP-hard.*

Proof 4 *This is a proof sketch. We use the minimized Steiner Tree problem (MSTP for short) [17], [18] to do the reduction. The MSTP is that given a graph $G = (V = (V_1 \cup V_2), E)$, find the minimized tree T containing the node set V_1.*

First, we reduce MSTP to MTTP. Given a graph $G = (V = (V_1 \cup V_2), E)$, we can find the bridges in G in linear time using Tarjan's Algorithm [19]. We build G': (1) the node set of G' is $V_1 \cup V_2$; (2) the bridges of G are the dependency edges of G', and other edges of G are schema edges of G'; (3) the nodes in V_1 be TNs of G'. It is easy to verify that finding a Steiner tree in G is equivalent to finding a TerTree in G'. Example 3 have showed that the dependency edge can be any structure, so in general case, the two components at both ends of the bridge can also be any kind of graph structure, that is, can be a clique, a tree, a single node, a path, and so on. If there is no bridge in graph, then we can regard it as a special case, that is, there are only schema edges but no constraint edge in the NSCG.

Now we reduce MTTP to MSTP. Given a NSCG G, we change all the edges in G to be the same, and consider the set of TNs to be the necessary node set (i.e., V_1) in MSTP. Then, finding a TerTree in G is equivalent to finding a Steiner tree in G'.

Since MSTP is NP-hard, MTTP is also NP-hard.

Please note that in the NSCG, the dependency edge does not have to be a bridge, and thus the step of finding the bridge in the proof is not necessary. It is just a operation to make the result of the construction look more similar to an ordinary NSCG. In fact, in extreme cases we can even treat each Steiner tree problem as a TerTree problem without dependency edges. We use the bridge as a dependency edge is mainly because that the bridge connects two relatively dense edges in the graph. This property is most similar to the dependency edge in the NSCG. Therefore, it is most appropriate to use a bridge to generate a dependency edge. Figure 5 (a) gives an example to illustrate a situation where the dependency edges are not a bridge. In the figure, two dependencies $R_1[C_1] \subseteq R_2[C_2]$ and $R_1[D_1] \subseteq R_2[D_2]$ add two dependency edges between schema R_1 and R_2. Thus either edge of the two edges is not a bridge in the corresponding NSCG. This may lead to more than one ways to generate CPQ. More precisely, we can generate two different CPQs by the two TerTrees shown by Figs. 5 (b) and 5 (c). The cost of the different CPQs are also different.

We can transform the problem of minimized Steiner tree to the problem minimized TerTree. Since the minimized TerTree problem is NP-hard, we give two approximate methods to solve it.

Method based on the minimum spanning tree. The details of this method are shown by Algorithm 2. The main ideas of each step are as follows. (1) Initially let ρ_m be empty. (2) We randomly pick a TN, and add it to ρ_m. (3) A node v' which is nearest to ρ_m is selected, and the corresponding shortest path is added to ρ_m. (4) Step (3) is repeated until all the TNs are added to ρ_m. In Line 2, the nodes which belong to the same schemas with the nodes in ρ_m have the priority for selection. This is because that Definition 2 ensures the distance between these nodes and ρ_m to be 1, but if the schema edges are organized by other ways rather than clique, this step is not necessary. In the worst cases, Algorithm 2 need to compute the shortest path for each TN. If we use Dijkstra's Algorithm with a implementation based

Algorithm 2 Generating CPQ from a TerTree

```
Input: NSCG $G_{\mathbb{R},\mathbb{D},\Sigma,\Psi,N}$
Output: Approximate Minimized TerTree $\rho_m$
 1: $\rho_m = \emptyset$, $S_{\rho_m} = \emptyset$ //$S_{\rho_m}$ is the set of schemas corresponding to the nodes in $\rho_m$
 2: random select a terminal node $v$ and add it to $\rho_m$
 3: $S_{\rho_m} = S_{\rho_m} \cup \{S_v\}$ //$S_v$ is the schema corresponding to $v$
 4: repeat
 5:     if $\exists$ terminal nodes which belongs to the schemas in $S_{\rho_m}$ but not in $\rho_m$ then
 6:         add these nodes into $\rho_m$
 7:     add edges to make the nodes in the same schema be a path
 8:     select the terminal node $v'$ which is nearest to $\rho_m$
 9:     add $v'$ and its corresponding shortest path into $\rho_m$
10:     $V'$ is the set of the new added nodes
11:     $S_{\rho_m} = S_{\rho_m} \cup \{S_{V'}\}$ //add the new schema to $S_{\rho_m}$
12: until all the terminal nodes are added to $\rho_m$
```

on a priority queue, the shortest paths for a given starting node can be computed in $O(|V|\log|V|)$ time. Thus, the time complexity of Algorithm 2 is $O(|R_T||V|\log|V|)$, where V is the node set of NSCG. The approximation ratio is 2, and the proof is similar to finding the minimized Steiner tree [20], thus we do not repeat it here.

Method based on shortest paths. If $|R_T|$ is large, and close to $|V|$, then the time complexity of Algorithm 2 is $O(|V|^2\log|V|)$. In this case, Algorithm 2 might not be efficient enough, thus we give a simpler method for quickly finding a TerTree ρ. The main step is as follows. Initially let ρ be empty. First, we random choose a TN to be v_{start}. Then, compute the shortest paths from v_{start} to all the other nodes in NSCG. After that, add each TN and their corresponding path to ρ. Finally, we cut the redundant edges to make sure that ρ is a tree.

Since we only need to compute the shortest path for a given starting node once, the time complexity is $O(|V|\log|V|)$. This method is faster than Algorithm 2, but it may generate a larger TerTree.

4.2 Discussions on General Cases of NSCG

4.2.1 Multiple MCC Attributes

In the above discussions, we assume that each attribute corresponding to a unique MCC attribute. However, the assumption cannot always be satisfied in practice. In this subsection, we discuss how to deal with the case of multiple MCC attributes. There are two reasons for an attribute corresponding to multiple MCC attributes. (1) Given $A \in R_T$, if $D_i[B_1] \in MCC(D_T[A])$, then for each $D_j[B_2] \asymp D_i[B_1]$, $D_j[B_2]$ also belongs to $MCC(D_T[A])$. In this case, there are more than one MCC attributes exist in the same temporal connected component. (2) Some MCC attributes are not comparable. That is, $\exists D_i[B_1], D_j[B_2] \in MCC(D_T[A])$ such that neither $D_j[B_2] \preceq D_i[B_1]$ nor $D_i[B_1] \preceq D_j[B_2]$. Based on these two reasons, we give three strategies to deal with multiple MCC attributes.

Strategy 1. The most intuitive strategy is to enumerate all the single TN set.

Definition 8 (Single TN Set) *Given a NSCG $G_{\mathbb{R},\mathbb{D},\Sigma,\Psi,N}$, the set of TNs is denoted by V_T, V_{TS} is a single TN set iff (1) $V_{TS} \subseteq V_T$, (2) $|V_{TS}| = |R_T|$, (3) For each $A \in R_T$, $M_A \cap V_{TS} \neq \emptyset$, where M_A is the TN set of A.*

V_{TS} is a single TN set means that V_{TS} contains exactly one TN of each attribute in R_T. Therefore, we can enumerate all the single TN sets. For each single TN set V_{TS}, we build a new NSCG $G^{(V_{TS})}_{\mathbb{R},\mathbb{D},\Sigma,\Psi,N}$. The only difference between $G^{(V_{TS})}_{\mathbb{R},\mathbb{D},\Sigma,\Psi,N}$ and the original NSCG is that, in $G^{(V_{TS})}_{\mathbb{R},\mathbb{D},\Sigma,\Psi,N}$, the nodes in V_{TS} are TNs while the nodes in $V_T \setminus V_{TS}$ are ordinary ones. We use the methods in Sect. 4.1 to generate TerTrees, and chose the one has the least nodes as the final TerTree. A total of $\prod_{A\in R_T}|M_A| \leq |V|^{|R_T|}$ single TN sets need to be enumerated. If $|M_A|$ can be considered as a constant c, then the number of single TN sets are $c^{|R_T|}$.

Strategy 2. Strategy 1 can be further optimized, that is, we can consider the MCC attributes exist in the same temporal connected component only once. More precisely, we enumerate equivalent TN set.

Definition 9 (Equivalent TN Set) *V_{TE} is a equivalent TN set iff (1) $V_{TE} \subseteq V_T$, (2) For each $A \in R_T$, $M_A \cap V_{TE}$ is a maximal temporal connected component.*

Based on Definition 9, Strategy 1 can be improved as follows. First, we enumerate all the equivalent TN sets. For each equivalent TN set V_{TS}, we build a new NSCG $G^{(V_{TE})}_{\mathbb{R},\mathbb{D},\Sigma,\Psi,N}$. The method of building $G^{(V_{TE})}_{\mathbb{R},\mathbb{D},\Sigma,\Psi,N}$ is the same to building $G^{(V_{TS})}_{\mathbb{R},\mathbb{D},\Sigma,\Psi,N}$. Then, we use the methods in Sect. 4.1 to generate TerTrees, but once a TN v is added to the TerTree, all the nodes v' that $ter(v') = ter(v)$ in $G^{(V_{TS})}_{\mathbb{R},\mathbb{D},\Sigma,\Psi,N}$ is changed into ordinary nodes, that is, change $ter(v')$ to be $\perp$. Finally, we retain the TerTree which has the least nodes. A total of $\prod_{A\in R_T}|SC_A|$ equivalent TN sets need to be enumerated, where $|SC_A|$ is the number of maximal temporal connected component in M_A. Finding the connected component can be finished in linear time, thus this strategy can decrease the time cost greatly.

Strategy 3. If a user has a good tolerance to large TerTrees, the following heuristic strategy can quickly generate a TerTree. That is, we use the methods in Sect. 4.1 to find a TerTrees on the original NSCG, but once a TN v is added to the TerTree, each node v' that $ter(v') = ter(v)$ is changed into ordinary nodes. The strategy is quite like Strategy 2, but we do not need to enumerate any equivalent TN sets. If a node is changed into an ordinary node, it is also considered to be an ordinary node in the final TerTree. This strategy can eliminate the time cost of enumeration, but the shortage is that it may generate a large TerTree.

4.2.2 NSCG without Complete CPQ

In some cases, we cannot find a complete CPQ. That is, there is no complete TerTree in NSCG. In this case, we try to find an approximate TerTree ρ, and perform the corresponding

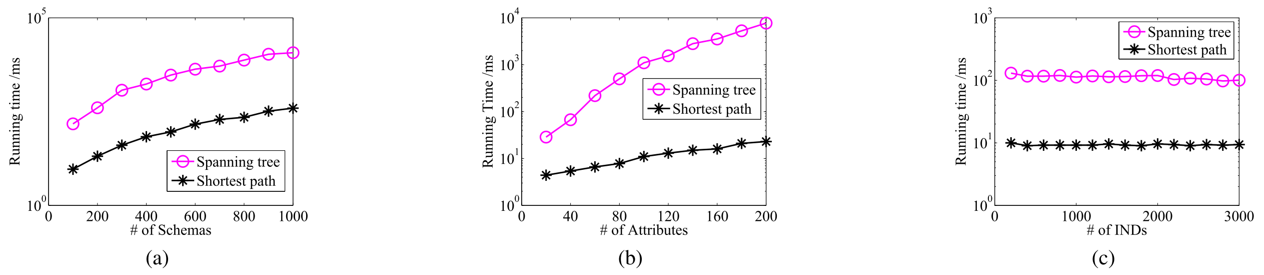


Fig. 6 Running time of finding TerTree in NSCG with single MCC

query Q_ρ to return partial newest values of D_T. The method of generating an approximate TerTree is as follows.

1. We use the above methods to try to find a partial TerTree ρ. When selecting TNs, we do not add those TNs which do not have any paths to ρ.
2. For each $A \in R_T$ do not have any TN in ρ, we use another newer (but not the newest) attribute to substitute $MCC(A)$. That is, we check the attributes which is more up-to-date than A according to the temporal orders from new to old. If we find $D_T[A] \preceq D_i[A']$, and $R_i[A']$ can be connected to ρ in NSCG, then we add $R_i[A']$ into ρ.
3. The process of Step 2 is repeated until each $A \in R_T$ has either a TN or a substitution of TN in ρ.

5. Experimental Results

We conduct the experiments on both real-life and synthetic data sets. The codes are written in C++ and run on a machine with 3.10GHz Intel CPU and 4GB of RAM.

NBA player is a real-life data set containing the information of NBA player and teams from 1946 to 2013. The tables keep the information of players, teams, coaches and arenas in different periods. NBA player totally contains 139 tables (i.e. $|\mathbb{R}| = 139$), and each table has 3 to 8 attributes. The temporal orders and the referential integrity constraints are manually derived. Table 1 shows the details of these tables. Some tables are with the same schema, and the number of tables with same schema are denoted by #T.

SYN is the set of synthetic data. We randomly generate different sizes of attributes and dependencies to test the efficiency and effectiveness of the proposed technique.

The characteristics of the above two data sets can be generalized to the general situation. The NBA player dataset is a real dataset that meets the problems studied in this paper. It is also used by many other papers that study the data quality and is therefore authoritative. The data set SYN is only used to test and compare the relative merits of different methods. Because the data was generated randomly and we did not make any presuppositions about it, the experimental results on this data set can be generalized.

Table 1 Description of data sources

	Schema	#T
1	Abbreviation(Team, TeamAbbr)	1
2	Player(Rk, Player, Age, Tm, Lg, G, PTS)	67
3	Team(Franchise, Lg, Yrs, G, W, L, Champ)	6
4	Arena(Team, Arena, Location, Capacity)	13
5	Coaches(Lg, Team, Coaches)	52

5.1 Efficiency

The parameters which influence the efficiencies of our approaches are the number of schemas (*sNum*), the number of attributes (*aNum*), the number of INDs (*indNum*), the number of MCC attributes corresponding to each attribute (*mccNum*), the number of maximal temporal connected components (*sccNum*), and the number of nodes in TerTree when generating the CPQ. First, we test the efficiencies of the methods of finding TerTree in both single and multiple MCC cases, and then, we test the efficiency of generating the CPQ using Algorithm 1. All the results are the average results of 5 runs.

NSCG with single MCC. Figure 6 shows the efficiencies of the methods of finding TerTree when varying *sNum*, *aNum* and *indNum*. In these experiments, each attribute corresponds to a unique MCC attribute. The x-axis indicates the variety of parameters, and the y-axis indicates the running time. The lines with label "Spanning Tree" shows the results of the method based on the minimum spanning tree. The lines with label "Shortest Path" shows the results of the method based on the shortest paths. Figure 6 (a) shows the running time when varying *sNum*. We set *aNum* to be 10, and varying *sNum* from 0 to 1,000. To keep the nodes in NSCG connected, *indNum* should increase with the increase of *sNum*, thus we set $indNum = 2 \times sNum$. The running time increases with the increase of *sNum*, but the time cost of Shortest Path is much less than Spanning Tree. In the experiments shown by Fig. 6 (b), $sNum = 10$, $indNum = 20$. *aNum* of all the schemas is the same, varying from 0 to 200. It's easy to see that $|R_T| = aNum$, $|V| = aNum \times sNum$, Thus, the time complexity of Spanning Tree based strategy is $O(|R_T||V|\log|V|) = O(aNum \times (aNum \times sNum) \times \log(aNum \times sNum))$. The time complexity of Shortest Path based strategy is $O((aNum \times sNum) \times \log(aNum \times sNum))$. Therefore, the increase of the running time of Spanning Tree is more dramatic than Shortest Path. The same trend can

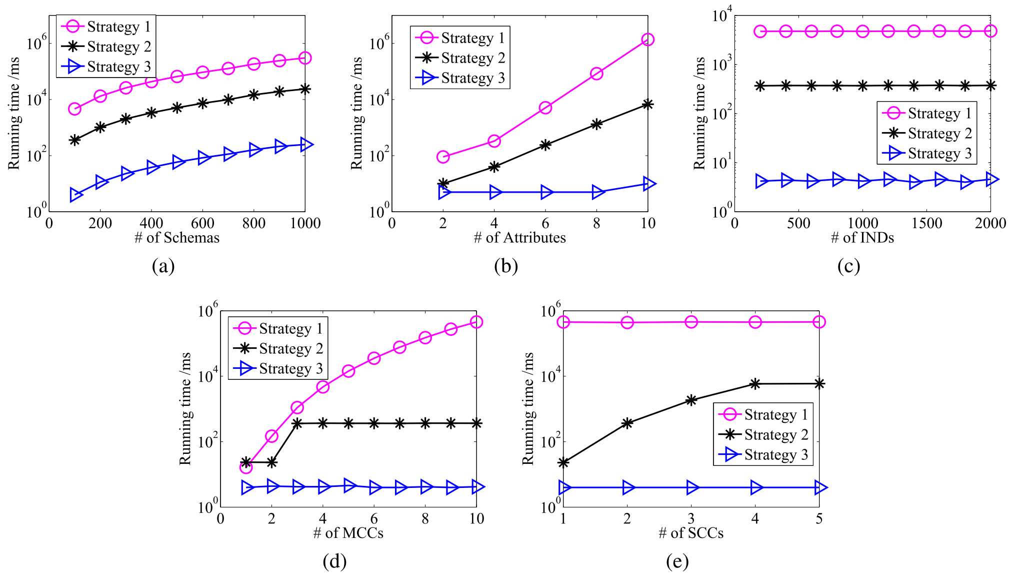


Fig. 7 Running time of finding TerTree in NSCG with multiple MCC

also be observed in the experimental results. In the experiments shown by Fig. 6 (c), $sNum = 100$, $aNum = 10$ and $indNum$ is increased from 20 to 3,000. Since $indNum$ only influence the density of NSCG, and the density of NSCG does not affect the time complexity of the two strategies, the running time remains stable.

NSCG with multiple MCC. Figure 7 shows the efficiencies of finding TerTree. when each attribute corresponds to more than one MCC attribute. We use the shortest paths based method as the basic method. The results labeled by "Strategy 1" to "Strategy 3" are returned by the 3 strategies in Sect. 4.2.1. According to the time complexity of each strategies, the time cost of Strategy 1 grows exponentially with base $mccNum$, the time cost of Strategy 2 grows exponentially with base $sccNum$, and the time cost of Strategy 3 does not relate to $mccNum$ and $sccNum$. Figure 7 (a) shows the running time when varying $sNum$. We set $aNum$ to be 5, $indNum = 2 \times sNum$, $mccNum = 4$ and $sccNum = 2$ for all the attributes in R_T. The running time of the three strategies increases with the increase of $sNum$. Strategy 1 has the lowest efficiency, and Strategy 3 is the highest performer. In the experiments shown by Fig. 7 (b), $sNum = 10$, $indNum = 20$, $mccNum = 4$ and $sccNum = 2$. The time cost increases with the increase of $aNum$. The increase trend of Strategy 1 is sharper than Strategy 2 and Strategy 3. Since $aNum = |R_T|$, and $|R_T|$ is the exponent of the time complexities of Strategy 1 and Strategy 2, the running time of "Strategy 3" grows much slower than the other two strategies. Figure 7 (c) shows the efficiencies when varying $indNum$. In the experiments, $sNum = 100$, $aNum = 5$, $mccNum = 4$ and $sccNum = 2$. Similar to the results of Fig. 6 (c), the increase of $indNum$ does not affect the efficiencies much. In the experiments of Fig. 7 (d), $sNum = 100$, $aNum = 5$ and $indNum = 200$. We increase $mccNum$ from 1 to 10. When $mccNum \leq 2$, the TNs can hardly be partitioned into more than one connected components, thus we set $sccNum = 1$ in these cases. When $mccNum > 2$, $sccNum$ is set to be 2. There is a significant trend for an increase of the running time of Strategy 1. The time cost of Strategy 2 is stable when $sccNum$ is not changed, and the time cost of Strategy 3 is unrelated to $mccNum$. In the experiments shown by Fig. 7 (e), $sNum = 100$, $aNum = 5$, $indNum = 200$, $mccNum = 10$, and $sccNum$ grows from 1 to 5. It is easy to observe that $sccNum$ only influence Strategy 2, and the time costs of the other two strategies are stable when $sccNum$ increases. The observation is consistent with the time complexity analysis in Sect. 4.2.

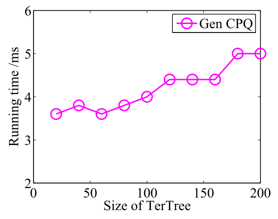


Fig. 8 Running time of generating CPQ

Generating CPQs. Figure 8 shows the efficiency of generating a CPQ from a TerTree. Since the running time is only related to the size (i.e., the number of nodes) of TerTree, we increase the size of TerTree from 20 to 200. The time cost grows with the increase of size of TerTree, but is much lower than the cost of finding a TerTree. When there are 200 nodes in the TerTree, it only needs 5 ms to generate the corresponding CPQ.

5.2 Effectiveness

We first use the data set SYN to evaluate the impacts of parameters $sNum$, $aNum$, $indNum$, $mccNum$ and $sccNum$ on the size of TerTree. Then, use the real-life data set NBA Player to test the effectiveness of the generated CPQ. Since

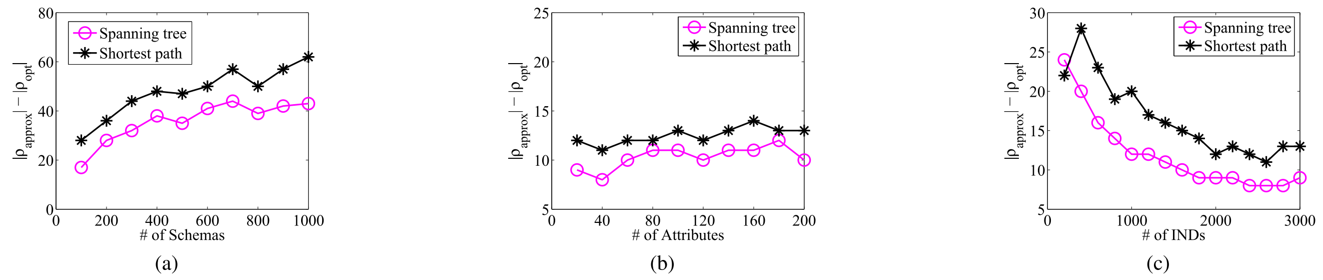


Fig. 9 Size of TerTree in NSCG with single MCC

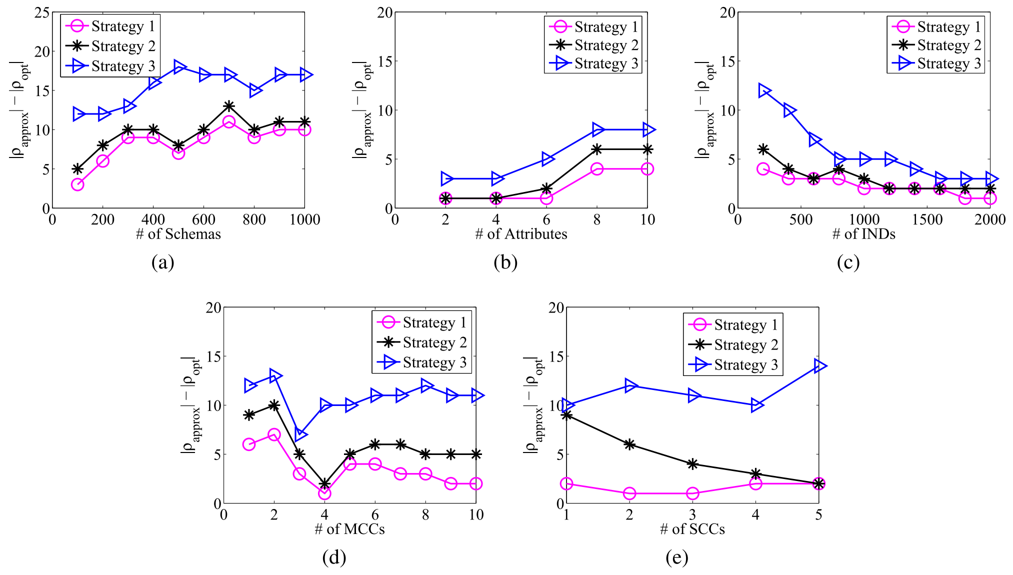


Fig. 10 Size of TerTree in NSCG with multiple MCC

the time cost of the exact method of finding minimized TerTree is too high, and a TerTree contains at least $|R_T|$ nodes, we use $|R_T|$ as an approximation of the exact result.

NSCG with single MCC. Figure 9 shows the impact of *sNum*, *aNum* and *indNum* in the case of single MCC. The x-axis shows the variation of parameters, and the y-axis $|\rho_{app}| - |\rho_{opt}|$ shows the difference between the result of our methods (ρ_{app}) and the exact result (ρ_{opt}). Figure 9 (a) shows the growth trend when *sNum* increases. We set $aNum = 10$, $indNum = 2 \times sNum$, and increase *sNum* from 0 to 1000. $|\rho_{app}| - |\rho_{opt}|$ increases with the growth of *sNum*. The results of Spanning Tree based strategy are better than the Shortest Path based one. Figure 9 (b) shows the growth trend when *aNum* increases. We set $sNum = 10$, $indNum = 20$, and increase *aNum* from 20 to 200. It can be observed that $|\rho_{app}| - |\rho_{opt}|$ slightly increases. *aNum* has almost no effect on the size of TerTree, because that we organize the attributes belonging to same schema as a clique, which makes these nodes reach each other in one hop. However, *sNum* increases means that the nodes belonging to different schemas increase, thus the size of TerTree has a stronger growth. In the experiment shown by Fig. 9 (c), we set $sNum = 100$, $aNum = 10$, and study the influence of *indNum* on size of TerTree. As we have discussed, *indNum* influence the density of NSCG. It is easier for the approximate method to find a small TerTree in a dense NSCG. Thus, $|\rho_{app}| - |\rho_{opt}|$ is more close to 0 when *indNum* increases. Consequently, $|\rho_{app}| - |\rho_{opt}|$ decreases with the increase of *indNum*.

NSCG with multiple MCC. Figure 10 shows the size of TerTree when each attribute in R_T corresponds to multiple TNs. For the efficiency reasons, we use the method based on shortest paths as the basic method of finding TerTree. Figure 10 (a) illustrates the results when *sNum* increases, where $aNum = 5$, $indNum = 2 \times sNum$, $mccNum = 4$ and $sccNum = 2$. It can be observed that the sizes of TerTrees generated by the three strategies deviate from $|\rho_{opt}|$ with the increase of *sNum*. The size of TerTrees generated by Strategy 3 is obviously larger than the TerTrees generated by the other two strategies. Strategy 1 generates better TerTrees than Strategy 2. The results shown by Fig. 10 (b) are the size of TerTree when varying *aNum*. We set $sNum = 10$, $indNum = 20$, $mccNum = 4$, $sccNum = 2$, and increase *aNum* from 2 to 10. Similar to the case of single MCC, the size of TerTree increases with the increase of *aNum*. Strategy 1 get the best results, followed by Strategy 2, and Strategy 3 always generates the largest TerTrees. The parameters settings of the experiments are shown by Fig. 10 (c) are

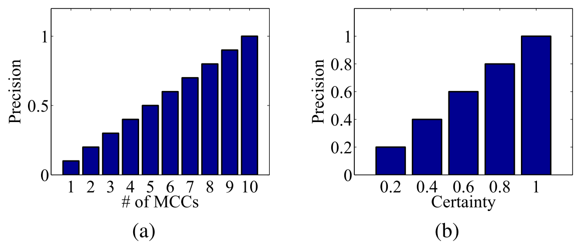


Fig. 11 Precision of CPQ on real dataset

$sNum = 100$, $aNum = 5$, $mccNum = 4$ and $sccNum = 2$. Similar to Fig. 9 (c), $|\rho_{app}| - |\rho_{opt}|$ decreases with the increase of *indNum*. Figure 10 (d) shows the experiments when varying *mccNum*. We set $sNum = 100$, $aNum = 5$, $indNum = 200$, and vary *mccNum* from 1 to 10. Similar to Fig. 7 (d), when $mccNum \leq 2$, we set $sccNum = 1$, otherwise set $sccNum = 2$. It can be observed that *mccNum* has almost no impact on the size of TerTree. The changing trends of Strategy 1 and Strategy 2 are similar, but the results of Strategy 1 are better than Strategy 2. Strategy 3 always generates the largest TerTrees. Figure 10 (e) shows the results when varying *sccNum*. We set $sNum = 100$, $aNum = 5$, $indNum = 200$, $mccNum = 10$, and increases *sccNum* from 1 to 5. We observe that *sccNum* only influence Strategy 2, and the other two strategies are not affected. The results of Strategy 2 are quite similar to Strategy 3 at first. With the increase of *sccNum*, the results are more and more close to Strategy 1. When $sccNum = 5$, each connected component only has two nodes, thus the results of Strategy 2 nearly equal to Strategy 1.

Effectiveness of generated CPQs. We use the proposed methods to find TerTrees and to generate the corresponding CPQs for the data set NBA Player, and test whether the CPQs can always return the most up-to-date values. The results are shown by Fig. 11. First, we consider the case discussed in Sect. 4.2.2, that is, the NSCG has no complete CPQ. Let $|cmp_Q|$ be the completeness of query Q, that is, $|cmp_Q|$ equals to the number of attributes in R_T which have one TN in Q. Let precision be the most up-to-date values which can be returned by Q. We increase $|cmp_Q|$, and run Q on NBA Player. The changing trend of precision is shown by Fig. 11 (a). Since the MCC attributes must contain all the up-to-date values of the target attribute, the precision is equivalent to $|cmp_Q|$. After that, we consider the case that the temporal orders are uncertain, that is, the MCC attributes only contain partial up-to-date values of the target attribute. We change the percents of up-to-date values contained by the MCC attributes, and observe the changing trend of precision. The results are shown by Fig. 11 (b). Precision is proportional to the percent of up-to-date values contained by the MCC attributes.

6. Related Works

Recently, some approaches have studied how to determine the data currency (i.e. temporal orders) based on rules [3], [10]. Based on the proposed rules, their methods can deduce temporal orders of the tuples representing the same entity. Combining with the techniques of entity resolution on different data sources [21], the temporal orders can be extended to the orders on different sources.

There are also a lot of works focusing on source dependencies, source selection, and capturing uncertain relationships in different types of networks [5], [7]–[9], [22]–[24]. These works propose methods aiming at finding the relationships of different sources and evaluating the qualities of the sources according to the relationships. The copy relationships and the overlaps detected by these methods can easily be extended to the temporal orders of columns on different sources.

Some works aim at finding the most current replica of the target data [25]–[27]. These works focus on how to find up-to-date data according to the global time stamps and the data updating model. However, the works assume that data has exact time stamps and updating model, and the aims are finding a current replica from a distributed storage system. Our work is different from these approaches because we do not assume all the required data items are stored in the same replica, and we have no requirement for the time stamps and updating model. The methods proposed in [28] focus on integrating data according to the time stamps and user defined preference rules. It does not focus on finding the most up-to-date values for a given target table, thus is different from our work.

Another branch of works focus on the query rewriting considering the access-restrictions and integrity constraints [14]–[16]. These works extend the search space through the integrity constraints, when the search space is limited by access restrictions, However, these works are different with our research, because that the temporal orders between data sources are quite different from the access restrictions. The access restrictions are a kind of 0-1 restrictions indicating the access methods of data items, but temporal orders are partial orders on different sources determining the priorities of selecting a value. Therefore, these works cannot solve the problem of generating currency preserving query.

7. Conclusions and Future Work

We studied the problem of generating currency preserving query. First, we proposed a graph model to represent the distributed table and their relationships. Based on the model, we proved that a currency preserving query is equivalent to a terminal tree in the graph, and gave an algorithm of transforming a terminal tree to the corresponding query. Then, we proved that generating the minimized currency preserving query is NP-hard, and provided heuristics strategies to solve the problem. Finally, we conducted experiments on both synthetic and real data sets to verify the effectiveness and efficiency of the proposed techniques.

In future works, we will study how to deal with conditional temporal orders, that is, the case that the temporal orders only exists in certain conditions. We will introduce the operation of selection into the currency preserving query, and extend our approaches by leveraging conditional temporal orders.

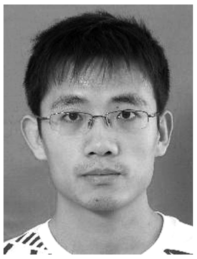
Acknowledgments

This work is supported by the National Natural Science Foundation of China (No. 61702220, No. 61702223), the Fundamental Research Funds for the Central Universities (No.11617303, 11617302), and Zhuhai Top Discipline Information Security.

References

[1] W. Eckerson, "Data warehousing special report: Data quality and the bottom line," Applications Development Trends, vol.1, no.1, pp.1–9, 2002.
[2] L.L. Pipino, Y.W. Lee, and R.Y. Wang, "Data quality assessment," Commun. ACM, vol.45, no.4, pp.211–218, 2002.
[3] W. Fan, F. Geerts, and J. Wijsen, "Determining the currency of data," ACM Trans. Database Syst. TODS, vol.37, no.4, pp.1–46, 2012.
[4] M. Salloum, X.L. Dong, D. Srivastava, and V.J. Tsotras, "Online ordering of overlapping data sources," Proc. VLDB Endowment, vol.7, no.3, pp.133–144, 2013.
[5] X.L. Dong, L. Berti-Equille, and D. Srivastava, "Integrating conflicting data: The role of source dependence," Proc. VLDB Endowment, vol.2, no.1, pp.550–561, 2009.
[6] X. Li, X.L. Dong, K. Lyons, W. Meng, and D. Srivastava, "Truth finding on the deep web: Is the problem solved?," Proc. VLDB Endowment, vol.6, no.2, pp.97–108, 2012.
[7] X.L. Dong, L. Berti-Equille, and D. Srivastava, "Truth discovery and copying detection in a dynamic world," Proc. VLDB Endowment, vol.2, no.1, pp.562–573, 2009.
[8] X.L. Dong, L. Berti-Equille, Y. Hu, and D. Srivastava, "Global detection of complex copying relationships between sources," Proc. VLDB Endowment, vol.3, no.1-2, pp.1358–1369, 2010.
[9] T. Rekatsinas, X.L. Dong, and D. Srivastava, "Characterizing and selecting fresh data sources," SIGMOD, pp.919–930, ACM, 2014.
[10] W. Fan, F. Geerts, N. Tang, and W. Yu, "Conflict resolution with data currency and consistency," J. Data Inf. Quality, vol.5, no.1-2, article no.6, Aug. 2014.
[11] S. Abiteboul, R. Hull, and V. Vianu, Foundations of Databases, Addison-Wesley Reading, Boston, USA, 1995.
[12] Y. Wang and Y. He, "Synthesizing mapping relationships using table corpus," Proc. 2017 ACM International Conference on Management of Data, pp.1117–1132, ACM, 2017.
[13] F. Chirigati, H. Doraiswamy, T. Damoulas, and J. Freire, "Data polygamy: The many-many relationships among urban spatio-temporal data sets," Proc. 2016 International Conference on Management of Data, pp.1011–1025, ACM, 2016.
[14] M. Benedikt, J. Leblay, and E. Tsamoura, "PDQ: Proof-driven query answering over web-based data," Proc. VLDB Endowment, vol.7, no.13, pp.1553–1556, 2014.
[15] M. Benedikt, J. Leblay, and E. Tsamoura, "Querying with access patterns and integrity constraints," Proc. VLDB Endowment, vol.8, no.6, pp.690–701, 2015.
[16] M. Benedikt, B.T. Cate, and E. Tsamoura, "Generating low-cost plans from proofs," PODS, pp.200–211, 2014.
[17] R.M. Karp, Reducibility among Combinatorial Problems, Springer, 1972.
[18] M.R. Garey and D.S. Johnson, Computers and Intractability, A guide to the theory of NP-completeness, WH Freeman and Co., San Francisco, CA, 1979.
[19] R.E. Tarjan, "A note on finding the bridges of a graph," Inform. Process. Lett., vol.2, no.6, pp.160–161, 1974.
[20] V.V. Vazirani, Approximation Algorithms, Springer, Berlin, Heidelberg, Germany, 2013.
[21] 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, 2007.
[22] X.L. Dong, B. Saha, and D. Srivastava, "Less is more: Selecting sources wisely for integration," Proc. VLDB Endowment, vol.6, no.2, pp.37–48, 2012.
[23] M. Han, M. Yan, J. Li, S. Ji, and Y. Li, "Neighborhood-based uncertainty generation in social networks," J. Comb. Optim., vol.28, no.3, pp.561–576, 2014.
[24] M. Han, M. Yan, J. Li, S. Ji, and Y. Li, "Generating uncertain networks based on historical network snapshots," Computing and Combinatorics, Lecture Notes in Computer Science, vol.7936, pp.747–758, Springer, 2013.
[25] B. Yan, D. Qian, and Y. Huang, "Data currency in replicated distributed storage system," 2009 IEEE International Conference on Networking, Architecture, and Storage, pp.57–63, 2009.
[26] R. Akbarinia, E. Pacitti, and P. Valduriez, "Data currency in replicated DHTs," Proc. 2007 ACM SIGMOD international conference on Management of data, New York, NY, USA, pp.211–222, ACM, 2007.
[27] C.H.C. Leung and K. Wolfenden, "Analysis and optimization of data currency and consistency in replicated distributed databases," The Computer Journal, vol.28, no.5, pp.518–523, 1985.
[28] B. Alexe, M. Roth, and W.-C. Tan, "Preference-aware integration of temporal data," Proc. VLDB Endowment, vol.8, no.4, pp.365–376, 2014.

Mohan Li received her B.S., M.S. and Ph.D degree in Computer Science from Harbin Institute of Technology (HIT), Harbin, China. She is currently a assistant professor in Guangzhou University, China. Her research interests include data quality and data cleaning.



Yanbin Sun received the B.S., M.S. and Ph.D degree in Computer Science from Harbin Institute of Technology (HIT), Harbin, China. He is currently a assistant professor in Guangzhou University, China. His research interests include Information-centric networking and information systems.