Discovering branching and fractional dependencies in databases

Tsvetanka Georgieva

“St. St. Cyril and Methodius” University of Veliko Tarnovo, Department of Information Technologies, 3 Architect Georgi Kozarov Street, Veliko Tarnovo, Bulgaria

A R T I C L E   I N F O

Article history:
Received 18 October 2007
Received in revised form 25 March 2008
Accepted 3 April 2008
Available online 12 April 2008

Keywords:
Data dependencies
Data mining
Dependency discovery

A B S T R A C T

The discovery of dependencies between attributes in databases is an important problem in data mining, and can be applied to facilitate future decision-making. In the present paper some properties of the branching dependencies are examined. We define a minimal branching dependency and we propose an algorithm for finding all minimal branching dependencies between a given set of attributes and a given attribute in a relation of a database. Our examination of the branching dependencies is motivated by their application in a database storing realized sales of products. For example, finding out that arbitrary \( p \) products have totally attracted at most \( q \) new users can prove to be crucial in supporting the decision making.

In addition, we also consider the fractional and the fractional branching dependencies. Some properties of these dependencies are examined. An algorithm for finding all fractional dependencies between a given set of attributes and a given attribute in a database relation is proposed. We examine the general case of an arbitrary relation, as well as a particular case where the problem of discovering the fractional dependencies is considerably simplified.

© 2008 Elsevier B.V. All rights reserved.

1. Introduction

The functional dependency requires the values in a given set of attributes to uniquely determine the value of a given attribute in a database relation. In [6] a branching dependency that is a more general dependency than the functional dependency is introduced. It is not possible to use the branching dependency to decrease the size of the stored relation, as in the case of functional dependency, but it is possible to constrain the range of the values of the attributes. We consider a modified form of the branching dependency defined in [6]. The modified branching dependency enables us to determine the maximal number of the different values of a given attribute \( b \) corresponding to one or more different values of a given set of attributes \( A \) in the relation. The issue of discovering these dependencies is addressed since they are highly instrumental in data mining, which aims at discovering interesting and useful patterns in large databases to support future decision-making.

As a motivating example, we consider a database of products and the customers who purchased them, or Web pages and the users who visited them. Finding out that arbitrary \( p \) products have totally attracted at most \( q \) new users for a given period of time can prove to be crucial in supporting the decision making.

We obtain a fractional branching dependency by adding the requirement that \( b \) should functionally determine \( A \). In case of fractional dependency, we can determine the maximal number of the different values of \( b \), corresponding to a \( p \) number of different values of the attributes of \( A \), but we consider only these values of \( b \), that remain after the elimination of the values of \( b \) for which the maximum for \( p - 1 \) can be obtained. This is additional information that may be useful for domain specialists when analyzing the current content of the database. For example, it would be useful to discover that the \( p \)th newly
offered product (service, promotion sale, etc.) has attracted at most \( c \), new users that have not been attracted by any of the previously offered products, providing that each subsequent product has been selected in such a way as to attract a maximum number of new customers.

Among other reasons for discovering the branching and fractional dependencies, we can also point out storing some additional information for query optimization; extracting information for the purpose of restricting the range of the attributes values when conducting database reverse engineering.

The discovery of dependencies is an important activity in the area of data mining and is employed in many applications for knowledge discovery in databases. Functional dependencies are relationships between attributes of a database relation. Some functional dependencies are defined while the database is being designed and they are used to reduce the amount of redundant data. Then the constraints are set to maintain the referential integrity. But the constraints are few and often too general, in the sense that they are valid in all possible database states. The discovery of the functional dependencies that reflect the current content of the relation is an important database analysis technique [2,4,9,20,26,31]. Discovering the functional dependencies, which hold in the current instance of a relation, is primarily aimed at finding out valuable knowledge about the structure of the relation instance, assisting various specialists (managers, analysts, etc.) in making more precise and faster decisions.

In some cases, a given functional dependency may hold not only for a few tuples. This functional dependency can be thought of as approximate, i.e. it almost holds. Approximate functional dependencies also provide valuable knowledge of the structure of the current instance of the relation. This knowledge can prove instrumental in analyzing the data that is contained in the database by domain specialists. Different algorithms [5,13,14,20,23,26,30] have proposed various measures for approximate dependencies.

Functional and approximate dependencies are also utilized in the area of query optimization [13,30], and reverse engineering [1,8].

In the present paper, a minimal branching dependency is defined and some properties of the branching dependencies are examined. An algorithm for finding all minimal branching dependencies between a given set of attributes and a given attribute is proposed. Realization of the proposed algorithm by using a data cube is described.

In addition, we define a fractional dependency and a fractional branching dependency and we examine some properties of these dependencies. An algorithm for finding all fractional dependencies between a given set of attributes and a given attribute is proposed. We consider a particular case where some constraints are imposed on the data in the analyzed attributes which considerably simplifies the process of discovering the fractional dependencies. Besides, we consider the general case of an arbitrary relation.

The rest of the paper is organized as follows. In Section 2, we present a brief survey on the related work. In Section 3, the problem of discovering all minimal branching dependencies is formulated. In Section 4, the dependencies between attributes are analyzed for the purpose of revealing some valuable knowledge of the data that is contained in the database. In Section 5, an algorithm for finding all minimal branching dependencies is described. Section 6 features some details about the realization of the proposed algorithm for discovering all minimal branching dependencies. In Section 7, the problem of discovering all fractional dependencies is formulated. In Section 8, an algorithm for finding all fractional dependencies is described and some results from the execution of the algorithm are presented.

2. Related work

The basic definitions and properties connected with the functional dependencies in relational databases are dealt with in detail in [10]. Some theorems valid for functional dependencies have been summarized in relation to branching dependencies in [6]. Furthermore, some implications among branching dependencies are investigated. In [7] estimates have been made of the minimum number of tuples in a relation, for which the sets of attributes have been obtained, where \( (p,q) \)-dependence \((1 \leq p < q, \text{ integers})\) on sets of attributes \(A\).

We consider a more general branching dependency that allows determining the maximum number of the different values of a given attribute \( b \) corresponding to one or more different values of a given set of attributes \( A \) in the relation. In the present paper, the problem of finding all branching dependencies between a given set of attributes \( A \) and a given attribute \( b \) is addressed. For that purpose, a minimal branching dependency is defined in such a manner so that the validity of all branching dependencies between \( A \) and \( b \) can be established if all minimal branching dependencies are known. Moreover, some properties of the branching dependencies are proved which enables us to prune some values of \( p \) and \( q \) while trying to obtain the branching dependencies between \( A \) and \( b \) and to create an efficient algorithm.

We tackle the problem of finding all fractional dependencies between a given set of attributes \( A \) and a given attribute \( b \).

Our two conference papers [11,12] contain shorter versions of the presented results in a preliminary form. In the present paper, we examine the general case of an arbitrary relation, as well as a particular case where the problem of discovering the fractional dependencies is considerably simplified.

3. Functional and branching dependencies

Let \( R \) be a database relation and let \( \Omega \) be the set of attributes of the relation \( R \). The number of the attributes is \(|\Omega| = n\). We say that the functional dependency (FD) \( A \rightarrow b \) holds or is valid in \( R \), if for any two tuples \( r,s \in R \) we have: if \( r(a_k) = s(a_k) \) for all \( a_k \in A, A = \{a_1, \ldots, a_l\}, A \subset \Omega, k = 1, \ldots, l \), then \( r(b) = s(b) \) for \( b \in \Omega \). The functional dependency \( A \rightarrow b \) is called nontrivial if
We say that the functional dependency \( A \rightarrow b \) is \textit{minimal} if \( b \) is not functionally dependent on any subset of \( A \), i.e. if \( B \rightarrow b \) does not hold in \( R \) for any \( B \subset A \).

Since the relations in the real databases may contain repeats, we consider the relation as a multiset of tuples. Let \( \pi \) be the projection operator in relational algebra, let \( \delta \) be the duplicate-elimination operator [10] that returns the set of the tuples in the relation-multiset.

Let \( A \subset \Omega, b \in \Omega, b \not\in A \) and \( 1 \leq p \leq q \) be integers. The branching dependency is defined in [6] in the following way: we say that \( (p, q) \)-dependence on \( A \) if there are no \( q + 1 \) tuples such that they contain at most \( p \) different values in each \( a_i \in A, \)

\( k = 1, \ldots, l \), but \( q + 1 \) different values in \( b \). We also say that \( (p, q) \)-branching dependency holds and we denote it by \( A^{(p,q)} \).

For example, in the relation \( R \), shown in Table 1, if \( A = \{a_1, a_2\} \), then \( A^{(2,6)} \) does not hold, since the tuples with \( id \) in the set \( I = \{1, 2, 3, 4, 5, 7, 8\} \) are \( 7 \), they have two different values in each attribute \( a_1 \) and \( a_2 \) (i.e. the value of \( \max(|\delta(a_{a_1}(\sigma_{a_{a_2}(R)}))|, |\delta(a_{a_2}(\sigma_{a_{a_2}(R)}))|) \) is \( 2 \)), but they have seven different values in \( b \). On the other hand, \( A^{(2,5)} \) is valid.

In [6] the different values of each attribute are counted separately. In our considerations, the possible values of \( p \) are \( 1 \) and \( 2 \). If the branching dependency \( A^{(p,q)} \) holds, then there do not exist seven tuples with at most two different values in \( A \) (i.e. the value of \( |\delta(a_{a_1}(R))| \) is at most \( 2 \)), which have \( 7 \) different values in \( b \), where \( R \) is the relation, consisting of arbitrarily chosen seven tuples of \( R \).

It is easy to ascertain that for \( q < |\delta(a_{a_1}(R))| \):

1. if the branching dependency \( A^{(p,q)} \) holds, then the branching dependency \( A^{(p_{\hat{q}}, \hat{q})} \) holds for each \( q_{\hat{q}} > q \);
2. if the branching dependency \( A^{(p_{\hat{q}}, \hat{q})} \) holds, then the branching dependency \( A^{(p,q)} \) holds for each \( p_{\hat{q}} \leq p \).

Furthermore, if \( q = |\delta(a_{a_1}(R))| \), then \( A^{(p, q)} \) holds for each \( 1 \leq p \leq q \). These conclusions enable us to define a minimum branching dependency.

**Definition 1.** We say that the branching dependency \( A^{(p,q)} \) where \( 1 \leq p \leq q \) is \textit{minimal} if for \( q < |\delta(a_{a_1}(R))| \) the dependency \( A^{(p_{\hat{q}}, \hat{q})} \) does not hold for each \( q_{\hat{q}} > q, p \leq |\delta(a_{a_1}(R))| \). If \( q = |\delta(a_{a_1}(R))| \) the dependency \( A^{(p,q)} \) is called a minimal branching dependency, if there are \( q \) tuples such that they contain \( p \) different values in the attributes of \( A \) and \( q \) different values in \( b \), but there do not exist \( q \) tuples with \( p - 1 \) different values in the attributes of \( A \) and \( q \) different values in \( b \).

For example, in the relation shown in Table 1 the dependencies \( A^{(2,6)} \), \( A^{(4,6)} \) hold, as well as \( A^{(2,7)} \), \( A^{(5,8)} \), but only the former two are minimal.

**Proposition 1.** If \( p = q \) and \( |\delta(a_{a_1}(R))| > q \), then the branching dependency \( A^{(p,q)} \) holds if and only if the functional dependency \( A \rightarrow b \) is valid.

**Proof.** Let \( A^{(p,q)} \) be a valid branching dependency. We assume that the FD \( A \rightarrow b \) does not hold, i.e. there are two different tuples \( r \) and \( s \), such that \( r(a_k) = s(a_k) \) for all \( a_k \in A, k = 1, \ldots, l \) and \( r(b) \neq s(b) \). Since the number of the different values in \( b \) is at least \( p + 1 \), then besides \( r \) and \( s \) there are at least \( p - 1 \) tuples more, which contain different values in the attribute \( b \). We assume that \( r_t \) are these tuples where \( i = 1, \ldots, p - 1 \). Consequently, \( r, s, (r_t)_{i=1,\ldots,p-1} \) are \( p + 1 \) tuples with at most \( p \) different values in the attributes of \( A \), but \( p + 1 \) different values in the attribute \( b \). This conclusion contradicts the assumption that the branching dependency \( A^{(p,q)} \) holds. Hence, we can conclude that if \( r(a_k) = s(a_k) \) for all \( a_k \in A, k = 1, \ldots, l \), then \( r(b) = s(b) \), i.e. the FD \( A \rightarrow b \) is valid.
If the branching dependency $A \rightarrow b$ is valid, the values in the attributes of $A$ uniquely determine the value in the attribute $b$ and, therefore, it is not possible for $p + 1$ tuples to exist with at most $p$ different values in the attributes of $A$, but $p + 1$ different values in the attribute $b$. □

**Proposition 2.** If the branching dependency $A^{(p \cdot q)} \rightarrow b$ does not hold and $|\delta(p_b(R))| > q + 1$, i.e. there are at least $q + 2$ different values in the attribute $b$, then the branching dependency $A^{(p + 1 \cdot q + 1)} \rightarrow b$ also does not hold.

**Proof.** From the assumption that the branching dependency $A^{(p \cdot q)} \rightarrow b$ does not hold, we can conclude that there are $q + 1$ tuples $r_1, \ldots, r_{q+1}$, such that they have at most $p$ different values in the attributes in $A$ and $r_i(b) \neq r_j(b)$ for each $j_1 \neq j_2$, $1 \leq j_1, j_2 \leq q + 1$. Since there exist at least $q + 2$ different values in the attribute $b$, let $r_i(b) \neq r_j(b)$ for $j = 1, \ldots, q + 1$. Then the tuples $r_1, \ldots, r_{q+1}, r_1$ are $q + 2$ tuples, which have at most $p + 1$ different values in $A$ and $q + 2$ different values in the attribute $b$. Consequently, the branching dependency $A^{(p + 1 \cdot q + 1)} \rightarrow b$ does not hold. □

**Corollary 2.1.** If the branching dependency $A^{(1 \cdot q)} \rightarrow b$ does not hold and $|\delta(p_b(R))| > q + 1$, i.e. there are at least $q + 2$ different values in the attribute $b$, then the branching dependency $A^{(2 \cdot q + 1)} \rightarrow b$ also does not hold.

If we first obtain the minimal branching dependency for $p = 1$, i.e. $A^{(1 \cdot q)} \rightarrow b$, then since the branching dependency $A^{(1 \cdot q - 1)} \rightarrow b$ does not hold, from Corollary 2.1 it follows that the branching dependency $A^{(2 \cdot q - 1)} \rightarrow b$ also does not hold. Therefore, the next step should be testing the validity of the branching dependency $A^{(2 \cdot q - 1)} \rightarrow b$, and so on, until $p \leq |\delta(p_{A_{- B}}(R))|$ and $q \leq |\delta(p_b(R))|$.

4. **Analyzing the dependencies between attributes in databases using the information from the minimal branching dependencies discovered**

One of the ways to define the approximate dependency is based on the minimal number of tuples that need to be removed from the relation $R$ for the respective FD $A \rightarrow b$ to hold in $R$ [23]. The error $e(A \rightarrow b)$ is defined as $e(A \rightarrow b) = \min(|S|)$ for $S \subseteq R$ and $A \rightarrow b$ holds in $R \setminus S$/$|R|$. For a given value $\varepsilon$ such that $0 \leq \varepsilon \leq 1$ we say that $A \rightarrow b$ is an approximate (functional) dependency if $e(A \rightarrow b)$ is at most $\varepsilon$.

Let $\sigma$ be the selection operator in relational algebra and $v = (v_1, \ldots, v_l)$ be an arbitrary element of the set $\delta(p_{A_{- B}}(R))$. The error can be calculated by the following equation:

$$e(A \rightarrow b) = 1 - \sum_{v \in \delta(p_{A_{- B}}(R))} \max(|\pi_b(\sigma_{A \rightarrow v} \text{ and } b \neq b_v(R))|) / |R|.$$

If for an arbitrary element $v \in \delta(p_{A_{- B}}(R))$ the corresponding value $b_v$ of the attribute $b$ is different from $b_{\text{frequent}}$, which is one of the values satisfying the condition

$$|\pi_b(\sigma_{A \rightarrow v} \text{ and } b \neq b_v(R))| = \max(|\pi_b(\sigma_{A \rightarrow v} \text{ and } b \neq b_v(R))|) \text{ for } b_v \in \delta(p_b(\sigma_{A \rightarrow v}(R))),$$

$b_v$ is considered as exception.

When an approximate dependency $A \rightarrow b$ where $e(A \rightarrow b) \leq \varepsilon$ is found, it is interesting to establish whether the exceptions refer only to one or just a few different values of $A$ or the exceptions occur for mostly different values of $A$. This information may be used to choose one of the following two approaches to correction of the error:

1. deletion of a minimal number of tuples in the relation $R$ for the FD $A \rightarrow b$ to hold in $R$, i.e. ignoring the noise;
2. addition of a minimal number of attributes in $A$ to obtain the set of attributes $A'$, such that the FD $A' \rightarrow b$ holds in $R$ (in [3] an algorithm for finding such enlargement $A'$ is proposed).

The first approach is appropriate if the exceptions refer to only one or just a few different values of $A$ and they can be ignored by considering the exceptions as noise. This case corresponds to a minimal branching dependency $A^{(1 \cdot q)} \rightarrow b$ where $q$ is approximately equal to the number of the tuples in which the FD $A \rightarrow b$ is violated. The second approach is appropriate if the exceptions occur for mostly different values of $A$. Then we can draw different conclusions depending on the data itself. This case corresponds to a minimal branching dependency $A^{(1 \cdot q)} \rightarrow b$ where $q$ has a comparatively small value.

5. **An algorithm for finding the minimal branching dependencies**

The algorithm presented as Algorithm 1 starts with validity testing of the functional dependency. If FD $A \rightarrow b$ does not hold it computes the error.

**Algorithm 1**

*Input: relation $R$ with a set of attributes $\Omega$; $A \subseteq \Omega$, $A = \{a_1, \ldots, a_l\}$; $b \in \Omega$, $b \not\in A$*

*Output: all minimal branching dependencies between $A$ and $b$*
On line 5 the algorithm finds the maximal number of the different values in the attribute \( b \) that correspond to a value of \( A \). In this way it finds the minimal branching dependency \( A \rightarrow \{b \} \). If the number of the different values in \( b \) is at most \( q \), i.e. \( |\delta(p_b(R))| \leq q \), then the branching dependency \( A \rightarrow \{b \} \) is always valid. Therefore, the algorithm considers these values of \( q \), which are less than \( |\delta(p_b(R))| \) and for \( q = |\delta(p_b(R))| \) finds the minimal \( p \), for which there exist \( q \) tuples with \( p \) different values in the attributes of \( A \) and \( q \) different values in \( b \). On line 10 the algorithm finds the maximal number of the different values in the attribute \( b \), that correspond to \( p \) different values of \( A \) and in this way it discovers the minimal branching dependency \( A \rightarrow \{b \} \).

6. Realization of the algorithm for finding the minimal branching dependencies in databases

The proposed approach uses a data cube with dimensions corresponding to all the attributes in the relation \( R \) and a measure whose value is obtained by computing the number of the tuples in \( R \) with the different values in the attributes. The measure of the data cube is computed by grouping all possible combinations of the attributes in \( R \).

By means of a graphical interface the realization of the algorithm enables us to select the data cube, the attributes in the left-hand side (LSH), and the attribute in the right-hand side (RSH) of the dependency subject to analysis (Fig. 1a). There is also an opportunity for additional analysis by means of the proposed in [17] graphical representation of the different values of \( P(A = v) = |\sigma_{a_1 = v_1} \land \ldots \land a_l = v_l}(R)\)/\(|R|\) and the relevant values of \( H_b(\sigma_{A = v}(R)) = \sum_{t \in \delta(p_b(\sigma_{A = v}(R)))} P(b = t) \log \frac{1}{P(b = t)} \) in order to reveal some characteristics of the local structure (Fig. 1b).

For the realization of the algorithm we use the MDX (multidimensional expressions) [28] and Visual Basic languages. A database relation with unknown structure is considered. The values of the tuples in the attributes are randomly generated.

Analysis. Line 1 of the algorithm presented in the present paper tests the validity of the FD and if the FD is not valid, on line 4 the algorithm computes the error. For that purpose we use a data cube in which previously found different values of the attributes and the number of the tuples in the relation with relevant values of the attributes are stored. The aggregate values computed and stored in the data cube facilitate the verification of the validity of the FD \( A \rightarrow b \) by obtaining the number of the
different values in the attributes of the sets A and \( A \cup \{ b \} \), by computing the error \( e(A \rightarrow b) \), as well as discovering the minimal branching dependencies. Therefore, the usage of a data cube in the suggested algorithm is more effective than the usage of a relational table-based structure that requires multiple scans of the data.

Our survey of previously published practical algorithms for discovering dependencies suggests that no algorithm for discovering branching dependencies has been proposed so far. For all executions of line 10 in the proposed Algorithm 1, the following number of values are compared:

\[
\frac{\left( \delta(\pi_{n-1}...a(R)) \right)}{2} + \ldots + \frac{\left( \delta(\pi_{n-1}...a(R)) \right)}{p},
\]

where \( 2 < p < q \leq \left| \delta(\pi_{k}(R)) \right| \), consequently at most \( \left| \delta(\pi_{k}(R)) \right| - 2 \) values are summed.

If it is necessary to find all branching dependencies in a relation, we can initially eliminate those combinations from A and b, for which the FD \( A \rightarrow b \) is valid by applying an algorithm for discovering all functional dependencies in the relation.

7. Fractional and fractional branching dependencies

Let \( \triangleright \) be the natural join operator in relational algebra, \( v_i = (v_{j1}, \ldots, v_{ji}) \) be an arbitrary element of the set \( \delta(\pi_{n-1}...a(R)) \) and \( d_A = \delta(\pi_{n-1}...a(R)) \), \( d_k = \delta(\pi_{k}(R)) \). If FD \( A \rightarrow b \) is valid, each \( v_i \) uniquely determines the value of \( b \), else we examine the data in \( R \) and we try to extract more detailed information about the different values of \( b \), corresponding to one or more elements of \( \delta(\pi_{n-1}...a(R)) \). We consider the following example:

**Example 1.** There are given \( d_A \) number of tasks. Each task is subdivided into several subtasks. Each subtask can be used to conclude one or more tasks. The total number of the subtasks is \( d_b \). The complexity of one task is determined by the number of its subtasks. For each \( 1 \leq p \leq d_A \) we have to find the minimal number of subtasks that we need to solve in order to conclude \( p \) number of tasks with the maximal total number of subtasks (i.e. with maximal complexity).

Finding the minimal branching dependencies can solve this problem. Let us consider the following variant of example 1: Each subtask can be used to conclude exactly one task.

This case corresponds to a special type of branching dependencies. We define these dependencies in the following way:

**Definition 2.** A branching dependency \( A^{(\beta)} \rightarrow b \) is called fractional branching dependency, if it is a minimal branching dependency and the functional dependency \( b \rightarrow A \) holds.

**Proposition 3.** We assume that the functional dependency \( b \rightarrow A \) holds and the integers \( c_1, c_2, \ldots, c_k \), where \( c_k = \max \{ \delta(\pi_{n-k+1}...a(R_k)) \} \) for \( i = 1, \ldots, d_A \), \( k = 1, \ldots, d_A \), are sorted in a descending order, i.e. \( 1 \leq c_{k+1} < c_k \), for each \( k = 1, \ldots, d_A - 1 \). If the value \( c_k \) is obtained for the values \( (v_{k1}, \ldots, v_{ki}) \) of the attributes of \( A \), then the relation \( R_{k+1} \) is obtained in the following way: \( R_1 = R \), \( R_{k+1} = R_k \ \triangleright \ \sigma_{\pi_{k}(R_k)}(\pi_{n-k+1}...a(R_k)) \) for \( k = 1, \ldots, d_A \). Then the set FB = \( \{ A^{(1-c)}(b, A^{(2+c_1-c_2+\ldots-c_{k-1})}b, \ldots, A^{(p-c_1-c_2+\ldots-c_p)}b) \} \) contains all fractional branching dependencies between A and b.

**Proof.** We apply induction on \( p \). For \( p = 1 \) we obtain a valid fractional branching dependency, since \( c_1 \) is the maximal number of the different values in the attribute \( b \), corresponding to the values in the attributes of \( A \). We assume that for \( p = k \) the branching dependency \( A^{(k-1-c_2-c_3-\ldots-c_{k-1})}b \), obtained in the way described above, is a valid fractional branching dependency. We have to prove that for \( p = k + 1 \) the branching dependency \( A^{(k-c_1-c_2-c_3-\ldots-c_{k-1})}b \) is also a valid fractional branching dependency. For this purpose, first we assume that it is not valid, i.e. there exist \( c_1 + c_2 + \ldots + c_k + c_{k+1} + 1 \) tuples with at most \( k + 1 \) different values in the attributes of \( A \) and \( c_1 + c_2 + \ldots + c_k + c_{k+1} + 1 \) different values in the attribute \( b \). We denote these tuples by \( r_1, r_2, \ldots, r_{c_1}, r_{c_1+1}, \ldots, r_{c_1+c_2}, \ldots, r_{c_1+c_2+\ldots+c_k}, \ldots, r_{c_1+c_2+\ldots+c_k+c_{k+1}} \), where \( c = c_1 + c_2 + \ldots + c_k \). The tuples \( r_1, r_2, \ldots, r_{c_1}, r_{c_1+1} \) are c + 1, consequently they have at least \( k + 1 \) different values in the attributes of \( A \), since otherwise we obtain a contradiction with the induction assumption. Then \( r_{i,j} \in \{ c + 2, \ldots, c + c_{k+1} + 1 \} \) coincides in the attributes of \( A \) with some of the tuples \( r_i, r_j \in \{ 1, \ldots, c + 1 \} \). But the values \( c_1, c_2, \ldots, c_k, c_{k+1} \) are sorted in a descending order and they are obtained by computing the numbers of the different values in the attribute \( b \), corresponding to fixed values in the attributes of \( A \). Consequently, we obtain a contradiction with the selection of some of the values \( c_1, c_2, \ldots, c_k, c_{k+1} \).

On the other hand if \( A^{(k-c)}(b) \notin FB \), then it is not valid, because if we assume that \( q \neq c_1 + c_2 + \ldots + c_k \), then the dependency we are considering is either an invalid branching dependency, or is not a minimal branching dependency. Hence, it is not a valid fractional branching dependency.

Consequently, the values \( c_k, k = 1, \ldots, d_A \) can be obtained by an SQL (structured query language) query of the following type:

```sql
SELECT COUNT(DISTINCT b) AS c_k
FROM R
```
Let us consider another variant of example 1: Each subtask can be used to conclude one or more tasks. For each \(1 \leq p \leq d_A\) we have to find the minimal number of subtasks that we need to solve to conclude a \(p\) number of tasks. Each \(p\)th subsequent task is selected such that for its conclusion to remain the maximal number unsolved subtasks.

Finding the branching dependencies does not entail the solution of sample problems of this type which compels us to define other types of dependencies.

**Definition 3.** Let \(A \subseteq \Omega, b \in \Omega, b \not\subseteq A\) and \(1 \leq p < q_p\) be integers and the following conditions hold:

1. for \(p = 1, q_1 < d_b\): there are no \(q_1 + 1\) tuples with equal values in the attributes of \(A\) and \(q_1 + 1\) different values in the attribute \(b\) and \(q_1\) is the minimal with this property;
2. for \(2 \leq p \leq d_b, q_p < d_b\): each \(q_p + 1\) tuples with \(q_p + 1\) different values in \(b\), that contain \(q_p\) different values in \(A\) and \(q_p\) different values in \(b\), they have at least \(p + 1\) different values in \(A\) and \(q_p\) is the minimal with this property;
3. for \(q_p = d_b\): each \(q_p\) tuples with \(q_p\) different values in \(b\), that contain \(q_p\) different values in \(A\) and \(q_p\) different values in \(b\), they have at least \(p\) different values in \(A\).

Then we say that \((p, q_p)\) fractional dependency between \(A\) and \(b\) holds and denote it by \(A^{(p, q_p)}_b\).

The major problem that we address in this section is for a given set of attributes \(A\) and an attribute \(b\) to find all fractional dependencies between \(A\) and \(b\). This problem can be considerably simplified if we set several constraints on the data in the attributes of the relation which are analyzed. Therefore, first we consider a particular case, and then we consider the general case of an arbitrary relation.

### 7.1. Case of constraints on the data in the attributes analyzed

We consider the following particular case of the second variant of example 1: For each \(p\) all uncombined tasks have different number of unsolved subtasks.

**Proposition 4.** We assume that the integers \(c_1, c_2, \ldots, c_d\), where \(c_k = \max\{|\delta(\pi_b(\sigma_{a_1 = v_{i_1}} \text{ and } \ldots \text{ and } a_{i_p} = v_{i_p}(R_k)))\}|\) for \(i = 1, \ldots, d_A\), \(k = 1, \ldots, d_b\), are sorted in a descending order, i.e. \(1 \leq c_{k+1} \leq c_k\) for each \(k = 1, \ldots, d_A - 1\). If the value \(c_k\) is obtained for the values \((v_{k1}, \ldots, v_{kq})\) of the attributes of \(A\), then the relation \(R_k\) is obtained in the following way: \(R_1 = R, R_{k+1} = R \backslash \left\{ R_b \circ \psi_b(\sigma_{a_1 = v_{i_1}} \text{ and } \ldots \text{ and } a_{i_p} = v_{i_p}(R_k)) \right\}\) for \(k = 1, \ldots, d_A\). We assume that each relation \(R_k, k = 1, \ldots, d_A\) is such that for \(c_k > 1\) the values \((v_{k1}, \ldots, v_{kq})\) are the unique values of the attributes of \(A\), for which we can obtain \(c_k\), i.e. \(|\delta(\pi_b(\sigma_{a_1 = v_{i_1}} \text{ and } \ldots \text{ and } a_{i_p} = v_{i_p}(R_k)))|\) coincides with the value \(c_k\). Then the set \(B = \{A^{(1, c_1)}_b, A^{(2, c_1 + c_2)}_b, \ldots, A^{(p, c_1 + c_2 + \cdots + c_p)}_b\}\), where \(1 \leq p \leq d_A, c_1 + c_2 + \cdots + c_p = d_b\) contains all fractional dependencies between \(A\) and \(b\).

**Proof.** We use a representation of all tuples of the relation \(R\) that is shown in Table 2.

The tuples \(r_{i_1}, r_{i_2}, \ldots, r_{i_c}\) have the same values in the attributes of \(A\) and \(c\) different values in the attribute \(b\). Furthermore, their values in the attribute \(b\) do not coincide with any value of the attribute \(b\) of the tuples located in the first column of Table 2 from the first to the \(p\)th row, i.e. \(r_{ij}(b) \neq r_{i_j}(b)\) for each \(j_1, j_2\) and \(i \neq k\). The tuples in the second column and \(i\)th row of Table 2 have the same values in the attribute \(b\) just like some of the tuples in the rows above the \(i\)th row, i.e. \(r_{ij}^*(b) = r_{ij}(b)\) for some \(j \in \{1, \ldots, c\}\). They have the same values in the attributes of \(A\) just like the tuples in the first column of the same row of Table 2, i.e. \(r_{ij}^*(A) = r_{ij}(A)\). The three dots in the end of the \(i\)th row of Table 2 show that there can exist other tuples \(r_{ij}^*\), \(j \in \{1, \ldots, c\}\), coinciding in \(A\) and \(b\) with some of the tuples listed in the same cell of Table 2. The tuples in the different rows of Table 2 do not coincide in all attributes of \(A\), i.e. \(r_{i_j}(A) \neq r_{i_j}(A)\) for each \(i_1 \neq i_2, 1 \leq i_1, i_2 \leq d_A\).

Table 2 shows the example relation from Table 1 obtained in the way described above.

Let us assume that \(s_k = c_1 + c_2 + \cdots + c_k, k = 1, \ldots, d_A\). If we consider \(s_k\) tuples with \(s_k + 1\) different values in the attribute \(b\), that contain a \(s_k\) different values in the attributes of \(A\) and \(s_k\) different values in the attribute \(b\), they have to be from at least \(k + 1\) different rows in Table 2. Consequently they have at least \(k + 1\)
different values in the attributes of $A$. Besides, we can select $s_k$ tuples with $s_k$ different values in the attribute $b$, that have $k$ different values in the attributes of $A$, hence, it follows that $s_k$ is the minimal value with this property. Consequently, the fractional dependency $A \xrightarrow{(k_s, b)} b$ is valid for $k = 1, \ldots, p$.

In the next step of the proof, we assume that there exists a valid fractional dependency $A \xrightarrow{(k_s, b)} b \notin B$. If $k \in \{1, \ldots, p\}$, then there exists a valid fractional dependency $A \xrightarrow{(k_{s_1 + s_2 + \cdots + s_k}, b)} b \in B$, hence $A \xrightarrow{(k, b)} b$ is not valid. The fractional dependency $A \xrightarrow{(p, d_A)} b \in B$ is valid, consequently for $p < k \leq d_A$ the dependency $A \xrightarrow{(k, b)} b$ is not valid. □

We consider three approaches to solving the problem of discovering the fractional dependencies in this particular case.

7.1.1. First approach: using the relational model and applying Transact-SQL

The description of this approach to solving the problem of discovering the fractional dependencies is presented by Algorithm 2.

Algorithm 2

Input: relation $R$ with a set of attributes $\Omega$, which satisfy the conditions of proposition 4; $A \subset \Omega$, $A = \{a_1, \ldots, a_i\}$; $b \in \Omega$, $b \notin A$

Output: all fractional dependencies between $A$ and $b$

1. $q = \delta(\pi_b(\sigma_{a_1=\nu_1} \text{ and } \ldots \text{ and } a_i=\nu_i(R)))$, where $\delta(\pi_b(\sigma_{a_1=\nu_1} \text{ and } \ldots \text{ and } a_i=\nu_i(R))) = \max\{\delta(\pi_b(\sigma_{a_1=\nu_1} \text{ and } \ldots \text{ and } a_i=\nu_i(R)))\}$

2. Output $A \xrightarrow{(i, q)} b$

3. $p = 2$

4. $R_p = R \setminus (R \bowtie \pi_b(\sigma_{a_1=\nu_1} \text{ and } \ldots \text{ and } a_i=\nu_i(R)))$

5. While $R_p \neq \emptyset$ do

6. $c_p = \delta(\pi_b(\sigma_{a_1=\nu_1} \text{ and } \ldots \text{ and } a_i=\nu_i(R_p)))$, where $\delta(\pi_b(\sigma_{a_1=\nu_1} \text{ and } \ldots \text{ and } a_i=\nu_i(R_p))) = \max\{\delta(\pi_b(\sigma_{a_1=\nu_1} \text{ and } \ldots \text{ and } a_i=\nu_i(R_p)))\}$

7. $q = q + c_p$ \( \pi_b(\sigma_{a_1=\nu_1} \text{ and } \ldots \text{ and } a_i=\nu_i(R_p)))$

8. Output $A \xrightarrow{(i, q)} b$

9. $R_{p+1} = R_p \setminus (R_p \bowtie \pi_b(\sigma_{a_1=\nu_1} \text{ and } \ldots \text{ and } a_i=\nu_i(R_p)))$

10. $p = p + 1$

7.1.2. Second approach: using the bipartite undirected multigraph

A finite undirected graph $G$ [21] is a pair $\langle V, E \rangle$, where

- $V = \{v_1, v_2, \ldots, v_n\}$ is a finite set of vertices;
- $E = \{e_1, e_2, \ldots, e_m\}$ is a finite set of edges. Each element $e_k \in E$ ($k = 1, 2, \ldots, m$) is a pair of the elements of $V$.

If $e_k = \{v_i, v_j\} \in E$, then $v_i$ and $v_j$ are called neighbors.

Let $G(V, E)$ be an undirected graph. If the set of the edge contains duplicates, i.e. $E$ is a multiset, $G$ is called a multigraph.

An undirected graph $G(V, E)$ is called bipartite, if it has as vertex set $V$ the disjoint union of a set $V'$ and a set $V'' (V = V' \cup V''$, $V' \cap V'' = \emptyset)$, and the edges $(i, j) \in E$ are such that $i \in V'$ and $j \in V''$. We assume that $V' = \{v_1, v_2, \ldots, v_{d_A}\}$ and $V'' = \{v_{d_A+1}, \ldots, v_{d_A+d_B}\}$. Each vertex of $V'$ corresponds uniquely to an element of the set of the distinct values in the attributes of $A$ and each vertex of $V''$ corresponds uniquely to an element of the set of the distinct values in the attributes $b$. An edge $(i, j) \in E$, $i \in V'$, $j \in V''$ exists, if there is a tuple in the relation $R$ with values in the attributes of $A$, corresponding to a vertex $i$ of $V'$ and a value in the attribute $b$, corresponding to a vertex $j$ of $V''$. Then the number of the edges of the obtained bipartite multigraph $|E|$ is the number of tuples $m$ in the relation $R$. For the representation of the bipartite multigraph we can use an $(d_A \times d_B + 1)$ adjacency matrix $s$, defined in the following way:

<table>
<thead>
<tr>
<th>Table 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Representation of the tuples of the example relation</td>
</tr>
<tr>
<td>(1,1,1,1), (2,1,1,2), (3,1,1,3), (4,1,1,4)</td>
</tr>
<tr>
<td>(5,1,2,5), (7,1,2,6)</td>
</tr>
<tr>
<td>(8,2,1,7), (14,2,1,7)</td>
</tr>
<tr>
<td>(10,2,3,8)</td>
</tr>
<tr>
<td>(6,1,2,3)</td>
</tr>
<tr>
<td>(9,2,1,4)</td>
</tr>
<tr>
<td>(11,2,2,1)</td>
</tr>
<tr>
<td>(12,1,3,1)</td>
</tr>
<tr>
<td>(13,1,4,2)</td>
</tr>
</tbody>
</table>
the number of the edges between \(i\) and \(j\).
\[
s_{ij} = \begin{cases} 
1 & \text{if there exists an edge } (i, j), \\
0 & \text{otherwise}.
\end{cases}
\]

where \(i \in V\) and \(j \in V^p\);

\(s_{ib+1}\) is the number of the neighbor vertices of the vertex \(i \in V\), i.e. the number of the different values in the attribute \(b\) for given values in the attributes of \(A\).

The complexity of the procedure for creation of the adjacency matrix is \(\mathcal{O}(m \log m + d_A d_b)\).

For instance, the tuples of the relation from Table 1 can be represented by the bipartite multigraph, shown in Fig. 2. For this relation we have \(d_A = 7\), \(d_b = 8\), \(\delta(p_{1,2}(R)) = \{1, 1, 1, 2, 1, 2, 2, 2\}\), and \(\delta(p_9(R)) = \{1, 2, 3, 4, 5, 6, 7, 8\}\). The vertices of the multigraph are \(V = \{1', 2', 3', 4', 5', 6', 7', 8'\}\), where 1' corresponds to (1, 1); 2' – (1, 2); 3' – (2, 1); 4' – (2, 3); 5' – (2, 2); 6' – (1, 3); 7' – (1, 4); 1" – 1, 2", 3", 4", 5", 6", 7", 8"); where 1 corresponds to (1, 2, 1); 2' – (1, 2); 3' – (2, 1); 4' – (2, 3); 5' – (2, 2); 6' – (1, 3); 7' – (1, 4); 1" – 1, 2", 3", 4", 5", 6", 7", 8" – 8.

The bipartite multigraph obtained can be represented by the adjacency matrix shown in Table 4.

Finding the fractional dependencies by using this approach is described in Algorithm 2'.

Algorithm 2'

1. \(p = 0\)
2. \(q = 0\)
3. \textbf{While} \(E \neq \emptyset\) \textbf{do}
   4. \(s_{ib+1} = \max_{i \in E} \{s_{ib+1}\}\)
   5. \(q = q + s_{ib+1}\)
   6. \(p = p + 1\)
   7. \textbf{Output} \(A \rightarrow b\)
8. \textbf{For each} \((k, j) \in E, j \in V^p\) \textbf{do}
   9. \textbf{For each} \((i, j) \in E, i \in V\) \textbf{do}
      10. \(s_{ij} = 0\)
      11. \(s_{ib+1} = s_{ib+1} - 1\)

Fig. 2. Representing the tuples of the example relation by means of the bipartite multigraph.

Table 4

<table>
<thead>
<tr>
<th></th>
<th>1&quot;</th>
<th>2&quot;</th>
<th>3&quot;</th>
<th>4&quot;</th>
<th>5&quot;</th>
<th>6&quot;</th>
<th>7&quot;</th>
<th>8&quot;</th>
<th>51</th>
</tr>
</thead>
<tbody>
<tr>
<td>1'</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>2'</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>3'</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>4'</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>5'</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>6'</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>7'</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Fig. 3. The edges of the multigraph that are deleted during the first execution of lines 8–11 of Algorithm 2'.
On line 4, Algorithm 2′ finds the maximal degree of the vertices of $V'$. On lines 8–11, the algorithm deletes the edges of all vertices $j \in V''$, that are neighbors of the vertex $k$.

Fig. 3 shows the edges that are deleted during the first execution of lines 8–11 for the example from Fig. 2.

Algorithm 2′ establishes the maximal element by using pyramid sorting, and therefore, it requires a number of comparisons of the order $\Theta(d_k + \log_2 d_k)$. Line 4 is executed $p$ times, consequently, the total number of comparisons is of order $\Theta(p(d_k + \log_2 d_k))$, $1 \leq p \leq d_k$.

7.1.3. Third approach: using the data cube and applying MDX queries

The data cube can be considered as a relation with $n + 1$ attributes. The first $n$ attributes represent dimensions, the $(n + 1)$th attribute represents the measure of the cube. The measure of the cube is the count of the tuples in the initial relation with relevant values in the attributes. A data cell in the data cube $c[v_{i_1}, \ldots, v_{i_n}]$ stores the count of the corresponding tuples of the initial relation satisfying $a_1 = v_{i_1}, \ldots, a_n = v_{i_n}$. The summarizing cells in the data cube $c[all, v_{i_2}, \ldots, v_{i_n}]$, $c[v_{i_1}, all, \ldots, v_{i_n}]$, $c[v_{i_1}, v_{i_2}, \ldots, all]$ store the sums of the measure of the data cube in the cells that have $n − 1$ grouping attributes, and so on, the cells $c[all, all, \ldots, all, v_{i_n}]$, $c[v_{i_1}, all, \ldots, all]$ store the sums of the measure of the data cube in the cells that have one grouping attribute; the cell with the value “all” for all dimensions $c[all, all, \ldots, all]$ stores aggregated value of the measure of the data cube without grouping attributes.

The data cube stores precomputed aggregates of the data in order to accelerate the processing of the frequent queries. The total number of the cells in the cube is $\prod_{i=1}^{n} (|\delta_{\pi_{a_i}}(R)|) + 1)$. Building an $n$-dimensional data cube requires a $\Theta(2^n)$ [16,19,25] complexity. Methods have been proposed [8,15,22] for selecting only some of the data cube views to materialize in order to speed up query response time, given space storage and maintenance time constraints.

We denote by $M_A, M_{Ab}, M_b$ the following sets:

$$M_A = \{v_{i_1}, \ldots, v_{i_d}\} \text{ for } \forall c[v_{i_1}, \ldots, v_{i_d}, v_j, all, \ldots, all] > 0, \forall v_j \in M_b\},$$
$$M_{Ab} = \{v_{i_1}, \ldots, v_{i_d}, v_j\} \text{ for } \forall c[v_{i_1}, \ldots, v_{i_d}, v_j, all, \ldots, all] > 0, \forall v_j \in M_b, j = 1, \ldots, d_b\},$$
$$M_b = \{v_j \text{ for } \forall c[v_j, all, \ldots, all] > 0, j = 1, \ldots, d_b\}.$$

We obtain Algorithm 2′′ by using this approach to find the fractional dependencies.

Algorithm 2′′

1. $p = 1$
2. $M_p = \{v_j \text{ for } \forall c[v_{i_1}, \ldots, v_{i_d}, v_j, all, \ldots, all] > 0, \forall v_j \in M_b\}$, where $$|\{v_j \text{ for } \forall c[v_{i_1}, \ldots, v_{i_d}, v_j, all, \ldots, all] > 0, v_j \in M_b\}| = \max_{1 \leq j \leq d_k} \{|v_j \text{ for } \forall c[v_{i_1}, \ldots, v_{i_d}, v_j, all, \ldots, all] > 0, v_j \in M_b\}|$$
3. $q = |M_p|$
4. Output $A \rightarrow b$
5. $M_{\text{prune}} = M_b \setminus M_p$
6. While $M_{\text{prune}} \neq \emptyset$ do
7. $p = p + 1$
8. $M_p = \{v_j \text{ for } \forall c[v_{i_1}, \ldots, v_{i_d}, v_j, all, \ldots, all] > 0, \forall v_j \in M_{\text{prune}}\}$, where $$|\{v_j \text{ for } \forall c[v_{i_1}, \ldots, v_{i_d}, v_j, all, \ldots, all] > 0, v_j \in M_{\text{prune}}\}| = \max_{1 \leq j \leq d_k} \{|v_j \text{ for } \forall c[v_{i_1}, \ldots, v_{i_d}, v_j, all, \ldots, all] > 0, v_j \in M_{\text{prune}}\}|$$
9. $q = q + |M_p|$
10. Output $A \rightarrow^{(p,q)} b$
11. $M_{\text{prune}} = M_{\text{prune}} \setminus M_p$

After the data cube is built, it can be used for multiple executions of Algorithm 2′′ with different combinations of $A$ and $b$. The incremental maintenance can be used to compute and propagate only changes of source relations rather than recompute the entire data cube from the source relations [24]. Usually, the source changes are updated in the data cube at regular intervals.

7.1.4. Comparison of the three approaches described above with respect to the execution time

The usage of a bipartite undirected multigraph requires a large amount of memory space, but this approach efficiently eliminates vertices from the next considerations.

The third approach (using a data cube and applying MDX queries) provides better performance with fewer dimensions. We represent a comparison of the efficiency:

- $(A)$ with respect to the number of the distinct values $d_A$ in the attributes of $A$ and the number of the distinct values $d_b$ in the attributes of $A \cup \{b\}$ with fixed number of the distinct values $d_k$ in the attribute $b$ and a total number of the tuples in the relation $m$ (see Table 5);
Since SQL performs processing of the datasets, when the tuples are deleted from the next considerations and computations, the first approach performs better than the second approach provided the values of the parameters increase.

(B) with respect to the number of the tuples \( m \) in the initial relation with fixed number of the distinct values \( d_A \) in the attributes of \( A \) and the number of the distinct values \( d_B \) in the attributes of \( A \cup \{b\} \) (see Table 6).

If we fix \( d_A \) and increase \( m \), it can be seen that when we use a data cube and MDX queries, the value of \( m \) does not make an impact on the performance unlike the case where a relational structure and SQL queries are employed. The data cube provides fast access to precalculated aggregations and avoids multiple scans of the data at extracting from relational structure.

7.2. General case of an arbitrary relation

Let us consider the discovery of the fractional dependencies in the general case: For each \( p \) all unconcluded tasks have an arbitrary number of unsolved subtasks.

We assume that \( c_1 = \max\{d_\pi(\sigma_{a_1 \cdots a_d} \text{ and } \alpha_{a_j = v_j}(R))\} \) for \( i = 1, \ldots, d_A \). \( V^{(1)} \) contains those values in the attributes of \( A \), for which the value \( c_1 \) is obtained, i.e. \( V^{(1)} = \{v_j = (v_j, \ldots, v_j), \text{ where } (v_j, \ldots, v_j) \in (\delta_{a_1, \ldots, a_d}(R)) \text{ and } |d_\pi(\sigma_{a_1 \cdots a_d} \text{ and } \alpha_{a_j = v_j}(R))| = c_1\} \).

We consider the tree \( T \) whose nodes can be obtained in the following way (Fig. 4):

- the node (all. \( d_B \)) is the root of the tree \( T \);
- \((v_j, c_1)\) for \( v_j \in V^{(1)} \) are the nodes of the tree \( T \) from the first level, i.e. the child nodes of the root (all. \( d_B \));
- For each element \( v_j \in V^{(1)} \) we obtain the relation \( R'_1 = R \setminus \{R' \subset \sigma_{a_1 \cdots a_d}(R) \text{ and } \alpha_{a_j = v_j}(R)\} \). We find the value of \( c_2 = \max\{|d_\pi(\sigma_{a_1 \cdots a_d} \text{ and } \alpha_{a_j = v_j}(R'))| \text{ for } i = 1, \ldots, d_\pi\} \). We form the set \( V^{(2)} = \{v_i = (v_i, \ldots, v_i), \text{ where } (v_i, \ldots, v_i) \in (\delta_{a_1, \ldots, a_d}(R'_1)) \text{ and } |d_\pi(\sigma_{a_1 \cdots a_d} \text{ and } \alpha_{a_j = v_j}(R'_1))| = c_2\} \). Then \((v_i, c_2)\), \( v_i \in V^{(2)} \) are the child nodes of the node \((v_j, c_1)\).
- If \((v_j, c_k)\) is a node on a level \( k, k \geq 2 \), then its child nodes are obtained by using the relation \( R_{k+1} = R' \setminus \{R' \subset (\sigma_{a_1 \cdots a_d}(R'_k) \text{ and } \alpha_{a_j = v_j}(R'_k))\} \), computing the value of \( c_{k+1} = \max\{|d_\pi(\sigma_{a_1 \cdots a_d} \text{ and } \alpha_{a_j = v_j}(R_{k+1}))| \text{ for } i = 1, \ldots, d_\pi\} \), forming the set \( V^{(k+1)} = \{v_i = (v_i, \ldots, v_i), \text{ where } (v_i, \ldots, v_i) \in (\delta_{a_1, \ldots, a_d}(R_{k+1})) \text{ and } |d_\pi(\sigma_{a_1 \cdots a_d} \text{ and } \alpha_{a_j = v_j}(R_{k+1}))| = c_{k+1}\} \). Then \((v_i, c_{k+1})\), \( v_i \in V^{(k+1)} \) are the child nodes of the node \((v_j, c_k)\).

Since at each step some tuples of the relation are deleted, for each \( j \) there exists \( k_j \) for which \( R'_{k_j} = \emptyset \), hereby the leaves of the tree \( T \) can be obtained.

For instance, the tuples of the relation from Table 7 will result in obtaining the tree shown in Fig. 5.

**Proposition 5.** We assume that \( c_2 = \max\{c_2' \text{ for } v_j \in V^{(2)} \} \) \((v_j, c_2') \text{ that is a node of } T \text{ on the second level}\), then we consider the set of the nodes on the second level, that contain this value \( c_2 \) and we find \( V_2 = \{v_i \text{ for } \forall(v_i, c_2), v_i \in V^{(2)}\} \). We obtain the value \( c_3 = \max\{c_2' \text{ for } v_j \in V_2 \} \) that is a child node of \((v_j, c_2), v_j \in V_2\), we find \( V_3 = \{v_j \text{ for } \forall(v_j, c_3), v_j \in V^{(3)} \} \) that is a child node of \((v_j, c_2), v_j \in V_2\), and so on, \( c_p = \max\{c_p' \text{ for } v_j \in V_2 \} \) that is a child node of \((v_j, c_p-1), v_j \in V_2\), and so on, \( c_p = \max\{c_p' \text{ for } v_j \in V_2 \} \) that is a child node of \((v_j, c_p-1), v_j \in V_2\). Then the set \( B = (1^{c_1}b, 2^{c_1+c_2}b, \ldots, p^{c_1+c_2+\cdots+c_p}b) \) where \( 1 \leq p \leq d_A; \) \( c_1 + c_2 + \cdots + c_p = d_B \) contains all fractional dependencies between \( A \) and \( b \).

<table>
<thead>
<tr>
<th>( d_A )</th>
<th>( d_B )</th>
<th>( T ) (I approach)</th>
<th>( M ) (III approach)</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>3000</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>500</td>
<td>20,000</td>
<td>11</td>
<td>7</td>
</tr>
<tr>
<td>1000</td>
<td>100,000</td>
<td>33</td>
<td>13</td>
</tr>
<tr>
<td>10,000</td>
<td>100,000</td>
<td>36</td>
<td>38</td>
</tr>
<tr>
<td>30,000</td>
<td>300,000</td>
<td>61</td>
<td>96</td>
</tr>
<tr>
<td>200,000</td>
<td>450,000</td>
<td>107</td>
<td>114</td>
</tr>
<tr>
<td>300,000</td>
<td>450,000</td>
<td>117</td>
<td>148</td>
</tr>
<tr>
<td>400,000</td>
<td>500,000</td>
<td>153</td>
<td>186</td>
</tr>
</tbody>
</table>

**Table 5**
Response time in seconds for \( d_A = 100 \) and \( m = 1,000,000 \)

**Table 6**
Response time in seconds for \( d_A = 1500 \) and \( d_B = 20,000 \)
Proof. We apply an induction on \( p \). For \( p = 1 \) we obtain a valid fractional dependency, because \( c_1 \) is the maximal number of the different values in \( b \), corresponding to the values in the attributes of \( A \). We assume that for \( p = k \) the dependency obtained in the way described above \( A \rightarrow (k; c_1 + c_2 + \cdots + c_k) \) is a valid fractional dependency. We assume that \( r_1, r_2, \ldots, r_{c_k+1} \) are \( c + 1 \) tuples with \( c + 1 \) different values in \( b \), which have at least \( k + 1 \) different values in \( A \), where \( c = c_1 + c_2 + \cdots + c_k \) and they contain \((c - c_k)\)-element subset with exactly \( k - 1 \) different values in \( A \) and \( c - c_k \) different values in \( b \). We have to prove that for \( p = k + 1 \) \( A \rightarrow (k+1; c_1 + c_2 + \cdots + c_{k+1}) \) b is also a valid fractional dependency. Therefore, we consider the tuples \( r_1, r_2, \ldots, r_{c_k+1} \) with \( c + c_k + 1 \) different values in \( b \). If we assume that these tuples have at most \( k + 1 \) different values in \( A \), and thus we obtain a contradiction with the selection of some of the values \( c_1, c_2, \ldots, c_k, c_{k+1} \).

In the next step of the proof, we assume that there exists a valid fractional dependency \( A \rightarrow (kq) \) \( b \in B \). If \( q \in \{1, \ldots, p\} \), then there exists a valid fractional dependency \( A \rightarrow (kq) \) \( b \in B \), hence \( A \rightarrow (kq) \) \( b \) is not valid. Assuming the fractional dependency \( A \rightarrow (p_0d_B) \) \( b \in B \) is valid, then for \( p < k \leq d_A \) the dependency \( A \rightarrow (kq) \) \( b \) is not valid. \( \square \)

8. An algorithm for finding all fractional dependencies

In the algorithm represented as Algorithm 3, we utilize the function \( \text{Get} \_\text{tree}(R) \). This function creates a relation \( R \_\text{tree}(Id, a_1, \ldots, a_i, c, lvl, Parent) \), that contains the described tree structure: \( Id \) is a unique identifier for the tuples in the relation \( R \_\text{tree} \); \( c \) stores the values \( c_1, c_2, \ldots, c_p \); \( lvl \) determines the level of the relevant node of the tree structure; \( Parent \) stores NULL for the root of the tree and the value of the attribute \( Id \) of the parent of the relevant node for the other nodes of the tree.
Algorithm 3

Input: relation $R$ with a set of attributes $\Omega$; $A \subset \Omega$, $A = \{a_1, \ldots, a_t\}$; $b \in \Omega$, $b \notin A$

Output: all fractional dependencies between $A$ and $b$

1. $R_{tree} = Get\_tree(R)$
2. $c_1 = \pi_c(\sigma_{b \rightarrow 1}(R_{tree}))$
3. Output $A \rightarrow \rightarrow b$
4. $p = 2$
5. $c_p = \max\{\pi_c(\sigma_{b \rightarrow 0}(R_{tree}))\}$
6. $q = c_1 + c_p$
7. Output $A \rightarrow \rightarrow b$
8. $V_{tree} = \pi_{id}(\sigma_{b \rightarrow 0} \text{ and } c \rightarrow c_p(R_{tree}))$
9. While $V_{tree} \neq \emptyset$ do
   10. $p = p + 1$
   11. $c_p = \max\{\pi_c(\sigma_{b \rightarrow 0} \text{ and } Parent:V_{tree} \rightarrow 1(R_{tree}))\}$
   12. $q = q + c_p(p_{aq})$
   13. Output $A \rightarrow \rightarrow b$
   14. $V_{tree} = \pi_{id}(\sigma_{b \rightarrow 0} \text{ and } c \rightarrow c_p \text{ and } Parent:V_{tree} \rightarrow 1(R_{tree}))$

The temporary relation $V(\text{id}, a_1, \ldots, a_t, c, \text{lvl}, Parent)$ is used in the following function. The attribute $\text{id}$ has a seed value of 1 and an increment of 1 for each inserted record.

Function Get\_tree($R$)
1. $R_{tree} = \emptyset$
2. $V = \{(a_1 = \text{NULL}, \ldots, a_t = \text{NULL}, c = d_b, \text{lvl} = 0, \text{Parent} = \text{NULL})\}$
3. $k = 0$
4. $R^*_k = R$
5. While $k \geq 0$ do
6. If $\sigma_{b \rightarrow 0}(V) \neq \emptyset$
   7. $\text{id}_j = \min\{\pi_{id}(\sigma_{b \rightarrow 0}(V))\}$
   8. $R_{tree} = R_{tree} \cup \sigma_{\text{id} \rightarrow \text{id}}(V)$
   9. $(v_{j1}, \ldots, v_{jd}) = \pi_{a_1 \rightarrow \ldots \rightarrow a_t \rightarrow \text{id}}(\sigma_{\text{id} \rightarrow \text{id}}(V))$
   10. $V = V \setminus \sigma_{\text{id} \rightarrow \text{id}}(V)$
   11. $R^*_k = R^*_k \setminus (\{v_{j1} > \pi_b(\sigma_{a_1 = v_{j1} \text{ and } \ldots \text{ and } a_t = v_{jd}}(R^*_k))\})$
12. If $R^*_k \neq \emptyset$
   13. $c_{k+1} = \max\{\delta(\pi_b(\sigma_{a_1 = v_{j1} \text{ and } \ldots \text{ and } a_t = v_{jd}}(R^*_k)))) \text{ for } i = 1, \ldots, d_b\}$
   14. $V = V \cup \{(v_{j1}, \ldots, v_{jd}, c_{k+1}, k+1, \text{id})\}$, where $(v_{j1}, \ldots, v_{jd}) \in \delta(\pi_b(\sigma_{a_1 = v_{j1} \text{ and } \ldots \text{ and } a_t = v_{jd}}(R^*_k))))$ and $|\delta(\pi_b(\sigma_{a_1 = v_{j1} \text{ and } \ldots \text{ and } a_t = v_{jd}}(R^*_k))))| = c_{k+1}$
   15. $k = k + 1$
16. Else $k = k - 1$
17. Return $R_{tree}$

Fig. 6. Results for 1,000,000 tuples in the relation ($d_{a_1} = 8; d_x = 404,231; X = \{A_1, \ldots, A_y\}$).
Analysis. If the degree of each node of the tree is less than or equal to $e$ ($1 \leq e \leq d_A$), the height of the tree is $p$, $d_{Ab} = |\Deltaelong{(\pi_{x_{RN}, R})}|$, then the complexity of the function Get_tree(R) for creation of the described tree structure is $O(d_{Ab} + \log_2 d_{Ab})$, $1 \leq p \leq d_{Ab}$.

Since we apply sorting that in the common case has a complexity $O(d_{Ab} \log d_{Ab})$, the complexity of the algorithm after building the tree structure is $O(p d_{Ab} \log d_{Ab})$, $1 \leq p \leq d_{Ab}$.

The algorithm is realized (Fig. 6) by means of Transact-SQL [27,29] in order to enhance the performance, since SQL processes datasets when tuples, which have to be excluded from the next considerations and computations, are deleted.

9. Conclusion

Analyzing the dependencies between attributes existing at a specific time enables us to reveal valuable knowledge of the structure of the current instance of a relation. In the present paper, the problem of discovering all minimal branching dependencies between a given set of attributes and a given attribute is addressed. An algorithm for finding these dependencies is described.

A realization of the proposed algorithm is represented by using the previously created data cube in order to enhance the efficiency of computing the values needed for finding the branching dependencies. Besides, we implement the approach that is described in [17] to visualize the functional and approximate dependencies that enable us to reveal some characteristics of attributes and their local structure.

We tackle the problem of discovering all fractional dependencies between a given set of attributes and a given attribute. An algorithm for finding fractional dependencies is described in the general case of an arbitrary relation. We propose approaches to solving this problem with some constraints on the data in the analyzed attributes.

Usually, the domain specialist is interested in specified subset of attributes and wants to extract the relevant dependencies between chosen attributes. Sometimes it can prove useful for the specialist to grow as result all dependences between the attributes. Therefore, a possible future research direction is to find an apt generalization and to develop algorithms for discovering all branching and fractional dependencies in a database relation.

References

Tsvetanka Georgieva received her M.Sc. in Mathematics and Informatics from the University of Veliko Tarnovo, Bulgaria in 1997. Currently she is Assistant Professor at the University of Veliko Tarnovo, Bulgaria and teaches Databases and Modeling Information Systems. She is a member of John Atanasoff Society of Automatics and Informatics and Union of Bulgarian Mathematicians. She has published over 20 papers in refereed journals and international conferences, mainly in the fields of databases, collaborative filtering, and data warehousing. She joined several national research projects on the above areas. Her current research interests include multidimensional modeling, data mining, and collaborative filtering.