Querying Incomplete Information using Bag Relational Algebra

Lubomir Stanchev
Computer Science Department
Indiana University - Purdue University Fort Wayne
Fort Wayne, IN, USA
stanchel@ipfw.edu

***Abstract*—In this paper we introduce bag relational algebra with grouping and aggregation over a particular representation of incomplete information called c-tables, which was first introduced by Grahne in 1984. In order for this algebra to be closed and "well-defined", we adopt the closed world assumption as described by Reiter in 1978 and extend the tuple and table conditions to linear ones. It turns out that query answering over the described extension remains polynomial relative to the size of the certain information. Therefore, the proposed algebra can be implemented as part of a SQL engine that can query incomplete information. The execution time will be acceptable as long as the size of the incomplete information is small relative to the total size of the database.**

***Keywords*-incomplete information; c-tables; relational model; null values**

I. Introduction

Many times, when information is entered into databases, the values for some of the fields are left empty for various reasons. In some cases, partial information about the blank fields is available, but existing relational database technology does not allow for such information to be processed. Imielinski and Lipski in [1] were among the first to propose richer semantics for null values that allows for incomplete information to be processed. However, their model was based on set semantics. Later on, Libkin and Wong published a paper on querying incomplete information in databases with multisets ([2]), but included only a limited set of operations, which excluded grouping and aggregation. Other papers that tackle the problems of storing and querying incomplete information include [3], [4], [5], [6], [7]. However, they all fail to explore relational operations defined over bag semantics such as grouping and aggregation.

In this paper we try to fill a gap left in published research in the area of storing and querying incomplete information. More precisely, we show how bag relational algebra with grouping and aggregation can be applied over incomplete information represented as a particular variation of *c-tables*. A c-table consists of a set of *c-tuples* and a *global condition*, where every c-tuple is made up from a regular tuple that may include variables for some of its fields plus a local condition (See Table 1 for an example). The semantics of a c-table is determined by the set of relations it represents, where each representation corresponds to a possible valuation for the incomplete information. In order for the relational algebra to be closed and *well defined*, we define the semantics of a c-table to be over the *closed world assumption*, as defined in ([8]), and we extend local and global conditions to be linear. We will refer to the described c-tables as *linear c-tables*, where their exact semantics is given in Section 2.1.

Originally, c-tables were introduced by Grahne in [9] to have local and global conditions that didn't contain the "$+$" operator and the "$>$" relation. Later on, Grahne added the "$>$" relation in [3]. However, we are not aware of any published research that allows for the "$+$" operator to be part of the local conditions or the global condition of a c-table. However, introducing the "$+$" operator is required in order for the relational algebra with aggregation that we introduce to be closed.

The main contribution of the paper are the algorithms for performing the different relational operations over linear c-tables. While the implementation of the operations projection, selection and inner join are similar to the case of set semantics (see [1]), the algorithms that perform the monus, duplicate elimination, and grouping operations are novel.

A. Motivation

Real world requirements have shown the importance of storing and querying incomplete information. However, commercial database management systems (DBMSs) provide only limited support (e.g., only null values) for incomplete information. Part of the reason, why this is the case, is the lack of research in the area. Note that while the problem of storing incomplete information is somewhat solved, querying incomplete information remains an open research problem. We believe that this paper makes a significant step towards solving the later problem. Another hurdle towards the implementation of a DBMS that processes incomplete information is the intrinsic high cost of managing such information. However, note that the algorithms we present for performing the various algebraic operations are non-polynomial relative only to the size of the incomplete information. Taking into account the ever increasing speed of computational resources, we believe that incorporating tools that store and query incomplete information in commercial database engines is feasible. Moreover, we believe that the presented work can play a key part in such an implementation. For example, since the code for executing bag relational algebra operations is an important part of the kernel of a SQL engine, we believe that the presented work can be used to implement a full SQL engine that can query incomplete information stored as linear c-tables.

In what follows, in Section 2 we define a representation of incomplete information in terms of linear c-tables. We explore the fundamental properties of linear conditions and linear c-tables and present algorithms for their manipulation. In Section 3 we define bag relational algebra operations over linear c-tables and give algorithms for their implementation. In Section 4 the problem of grouping and aggregation over linear c-tables is explored and in Section 5 a summary or the presented work and areas for future research are outlined.

II. C-Tables with Linear Conditions

The problem of representing incomplete information in the relational model is almost as old as the relational model itself ([10], [11], [12], [13], [14]). When a null value appears in a relational table, its value can be interpreted as no information available, only partial information available, value not applicable, and so on. Most of the research on null values has concentrated on the first two meanings. Known representations of relational tables adapting these meanings for nulls include Codd tables, naive tables, Horn tables and c-tables. Codd tables are relational tables, where the values of some the fields can be null. Naive tables are an extension of Codd tables, where each null is given a label and nulls having the same label represent the same unknown value. C-tables are naive tables with a local condition associated which each tuple and a single global condition associated with each table. Horn tables are a special kind of c-tables, where the conditions that can appear are restricted to Horn clauses.

Grahne in [3] considered conditions over the system $\langle R, \{>, =\}\rangle$ (i.e., Boolean expressions with variables and constants defined over the set R extended with "$>$" and "$=$"). To the best of our knowledge, this is the most expressive system for expressing c-table conditions in published research. In this paper we explore c-tables whose conditions are over the system $\langle \mathbb{R}, \{>, =, +\}\rangle \cup \langle \mathbf{S}, \{=, \neq\}\rangle$, where $\mathbb{R}$ is used to denote the set of real numbers and **S** is the set of strings over some finite alphabet. While the "$+$" operator is introduced in order to make the algebra closed relative to aggregation, the system over strings is introduced to extend the expressive power of c-tables. Note that we don't explore conditions over $\langle \mathcal{Z}, \{>, =, +\}\rangle$, where $\mathcal{Z}$ is the set of integers, or over $\langle \mathbb{R}, \{>, =, *, +\}\rangle$. The reason is that, although those systems are more expressive, reasoning with them is much harder. For example, Fischer and Rabin have shown in [15] that the complexity of deciding whether a formula over the first system is satisfiable is super exponential. As well, the complexity of the fastest known algorithm for solving the same problem for the second system, which is presented in [16], is higher than exponential.

name	**school**	**condition**
John	y	$x = 1$
Mark	y	$x \neq 1$
q	z	TRUE

g.c. $(q \neq$"Mark"$) \wedge (q \neq$"John"$) \wedge (z \neq y)$

Table I
An example c-table

A. Definitions

Formally, we introduce a linear c-table T_C as a finite, unordered bag of linear c-tuples and a global condition. A linear c-tuple with attributes $\{A_i\}_{i=1}^{a}$ is the sequence of mappings from A_i to $D(A_i) \cup V_i$ plus a local condition, where i ranges from 1 to a, $D(A_i)$ denote the domain of A_i and V_i is used to represent a possibly infinite, countable, set of variables over $D(A_i)$. The local and global conditions can range over $\langle \mathbb{R}, \{>, =, +\}\rangle \cup \langle \mathbf{S}\ \{=, \neq\}\rangle$. Table 1 shows an example of a linear c-table.

We will refer to the part of a linear c-table where the data is stored as the *main part* and to the remaining parts as the *local condition part* and the *global condition part*, respectively. In our example, x, y, z and q are used to represent variables. Since our model is limited only to the domains of real numbers and strings, the domain of a variable that doesn't appear in the main part of a linear c-table can be inferred from the context in which it appears. In our example, we can use the local condition $x = 1$ to deduce that the domain of x is $\mathbb{R}$.

The example linear c-table contains the information that either there are no students or there are two students that study in different schools and the name of one of them is "John" or "Mark" and the name of the other one is neither "John" nor "Mark". Note that in this example and throughout the paper we will be using the closed world assumption. The assumption states that a database representation contains only the things that are known to be true. In our example, we have used this assumption to conclude that there are at most two students.

In order to formally define the semantics of a c-table, Imielinski and Lipski in [1] introduce a function called Rep that maps a c-table T_C to a possibly infinite set of relational tables. Intuitively, the meaning of the Rep function is that given a c-table T_C, the function returns all relational tables that T_C could represent under different valuations. In [1] this function is defined relative to the *open world assumption*. We define it relative to the closed world assumption as follows.

$$Rep(T_C) = \{T | \exists v, s.t. \ \ v(T_C) = T\} \tag{1}$$

In the definition v is a mapping that maps the variables in T_C to constants in the corresponding domains and is generalized to linear c-tuples as follows.

$$v(t_C) = \begin{cases} v(main(t_C)) & : \quad v(lc(t_C)) \wedge v(gc(T_C)) \\ \varepsilon & : \quad otherwise \end{cases} \tag{2}$$

In the formula the functions $main$, lc and gc are used to denote the main part and the local condition part of a linear c-tuple and the global condition part of a linear c-table. The symbol ε is used to represent the empty set. The value of the tuple $v(main(t_C))$ is calculated by substituting the variables in the main part of t_C with the values to which v maps them to. The mapping v is further extended to linear c-table T_C as shown in Equation 3, where $\{t_C^i\}_{i=1}^k$ are the c-tuples in T_C.

$$v(T_C) = \{\mid v(t_C^i) | i \in [1,k] \wedge v(t_C^i) \neq \varepsilon \mid\} \tag{3}$$

Note that it is also possible to define ordering for linear c-tables, but we leave this topic for future research. The presented definition of v applied to a linear c-table is novel and differs from the definitions presented in [1] and [3]. Unlike the cited papers, we define duplicate semantics for c-tables and use the closed world assumption. From now on, we will refer to linear c-tuples just as c-tuples and to linear c-tables just as c-tables.

B. Linear Condition Simplification

In the c-table normalization procedure that we will present in Section 2.3 a way to simplify linear conditions and check their satisfiability will be required. Note that a linear condition is a Boolean expression and, as such, can be expressed as a disjunction of *positive conjunctions*. A positive conjunction is a conjunction of positive atomic linear conditions, where a positive atomic linear condition is of the form $\bar{a} \cdot \bar{x} = \bar{b}$ or $\bar{a} \cdot \bar{x} < \bar{b}$, where $\bar{x}$ is a variable vector and $\bar{a}$ and $\bar{b}$ are vector constants. An atomic linear condition includes in addition negative conditions of the form $\bar{a} \cdot \bar{x} \neq \bar{b}$. An intuitive representation of a positive conjunction is a multi-dimensional polyhedra defining a semi-linear set. Therefore, a linear condition can be thought as a set of disjoint polyhedras. Note however that such a representation is not unique and therefore a unique canonical form for linear conditions could not exist.

Let us first consider the algorithm proposed in [17] for normalizing conjunctions of linear equalities and inequalities. More precisely, the paper represents a conjunction of atomic linear conditions by the system $A\bar{x} \leq \bar{b}$, $E\bar{x} = \bar{d}$, $\neg(\bar{c}_i\bar{x} = \bar{f}_i)$, where A and E are matrices, $\bar{b}$, $\bar{d}$, $\bar{c}_i$ and $\bar{f}_i$ are vectors and $\bar{x}$ is a variable vector. The normalization algorithm runs in polynomial time, can be implemented to run on parallel machines and relies on calls to a module that solves linear programs. As well, the algorithm recognizes sets of unsatisfiable atomic conditions and reports them as such. Part of the algorithm deals with elimination of redundant conditions, which is an extension of the research published in [18]. The pivot theorem from [17] follows.

Theorem 1: If two sets of atomic conditions over $(\mathbb{R}, +, >, =)$ define the same point set, where $\mathbb{R}$ is the set of real numbers, their canonical forms will have identical set of equality conditions, the same inequality conditions up to multiplication by a positive scalar and the same set of negative conditions.

INPUT: linear condition C
ALGORITHM:
1. Convert C into disjunctive normal form, i.e. $C = c_1 \vee c_2 \vee ... \vee c_n$, where each conjunction c_i is positive.
2. Normalize each conjunction c_i using the algorithm from [17].
3. Scan the conjunctions $\{c_i\}_{i=1}^n$ in order. In the first iteration mark the conjunction c_1. During the $k^{\texttt{th}}$ iteration find the intersection of c_k with each of the marked conjunctions. More precisely, if $g_1, \ldots, g_p$ are the marked conjunctions so far, then calculate the normal forms of $g_i \wedge c_k$, $g_i \wedge \neg c_k$ and $\neg g_i \wedge c_k$ ($i = 1$ to p) using the algorithm from [17] to form the new set of marked conjunctions. Note that the algorithm from [17] may return that a conjunction is unsatisfiable, in which case the conjunction should be dropped and not marked.
4. The simplified value for C will be $g_1 \vee \cdots \vee g_m$, where $g_1, \ldots, g_m$ are the marked conjunctions at the end of Step 3. If there are no marked conjunctions at the end of Step 3, then return $C =$ FALSE.

Figure 1. The linear condition normalization algorithm

The algorithm we present for simplifying a linear condition C is presented in Figure 1. The algorithm first break C into a set of positive conjunctions. Next, it divides those conjunctions further so that they don't overlap and the algorithm returns their normalized form in its final step. The algorithm runs in $O(m^c \cdot 3^n)$ time, where m is the length of C, n is the number of conjunctions in the disjunctive normal form of C (i.e. $n \leq (\sqrt{2})^m$) and c is a constant. In order to verify this, note that Step 1 takes $O(m \cdot n)$ time. Step 2 makes n calls to the procedure from [17], which runs in $O(m^c)$ time because the length of each conjunction is smaller then the length of C. Step 3 makes at most $\frac{3^n-1}{2} - 1$ calls to the procedure from [17]. The reason is that in Step 3, during the $k^{\texttt{th}}$ iteration ($k > 1$), there can be as much as $3^{(k-2)}$ processed conjunctions and therefore as much as $3^{(k-1)}$ calls to the normalization procedure from [17]. Therefore, Step 3 can do as much as $\sum_{k=2}^{n} 3^{k-1} = \frac{3^n-1}{2} - 1$ calls to the normalization procedure and each call can take at most $O(m^c)$ time.

Note that the above algorithm can be used to test the satisfiability of a linear condition. To do so, we only need to execute the first two steps, which can be done in $O(n \cdot m^c)$ time. The rest of the steps are only useful if we want to eliminate duplicate information by breaking up the input

A	B	condition
1	2	$x=1$
z	2	$x=2$
p	w	$x=t$

g.c.: $t \neq 1 \wedge t \neq 2$

A	B	condition
a	b	$((a=1)\wedge(b=2)\wedge(x=1))\vee$ $((a=z)\wedge(b=2)\wedge(x=2))\vee$ $((a=p)\wedge(b=w)\wedge(x=t))$

g.c.: $t \neq 1 \wedge t \neq 2$

Table II
A C-TABLE AND THE RESULT OF APPLYING NORMALIZATION STEPS 1, 2 AND 3 TO IT

linear condition into disjoint polyhedras.

The presented algorithm can be applied not only to conditions over the system ($\mathbb{R}$, $>$, $=$, $+$), but also to conditions over the system ($\mathbb{R}$, $>$, $=$, $+$) $\cup$ ($\mathbf{S}$, $=$, $\neq$). To do so, substitute each atomic conditions of the form $x \neq c$, where x is a string variable and c is a string constant with $x = c_1 \vee x = c_2 \vee \cdots \vee x = c_r \vee x = c_{r+1}$, where c_{r+1} is a newly introduced string constant and $\{c_i\}_{i=1}^r$ are the existing string constants in the condition excluding c. As well, substitute each atomic condition of the form $x \neq y$, where x and y are string variables with $\bigvee\limits_{i,j=1,r+2}^{i\neq j} (x = c_i \wedge y = c_j)$, where c_{r+1} and c_{r+2} are newly introduced constants and $\{c_i\}_{i=1}^r$ are the existing string constants. Alternatively, Step 1 of the algorithm can be modified to require the breaking of C into not necessarily positive conjunctions. This modification allows the direct application of the algorithm to a linear condition containing strings because the algorithm from [17] handles equality and week inequality conditions separately from inequality conditions.

In the rest of the paper, unless we explicitly specify otherwise, when we refer to the algorithm from Section 2.2, we will mean the algorithm which executes only the first two steps, where the first step breaks C into not necessarily positive conjunctions. The time complexity of such an algorithm applied to a linear condition over ($\mathbb{R}$, $>$, $=$, $+$) $\cup$ ($\mathbf{S}$, $=$, $\neq$) is $O((\sqrt{2})^m * m^c)$.

C. C-Table Normalization

Note that there may be different c-tables representing the same set of bag relational tables, i.e. it may be the case that $T'_C \neq T''_C$ but $Rep(T'_C) \equiv Rep(T''_C)$. If $Rep(T'_C) \equiv Rep(T''_C)$, then we will say that T'_C and T''_C are equivalent and write $T'_C \approx T''_C$. In this section we present a method for testing equivalence of c-tables by comparing their normalized forms. In the presented algorithm we use the concept of c-tuple unification.

Definition 1: The c-tuples t_C^1 and t_C^2 of the c-table T_C are unifiable iff the formula $lc(t_C^1) \wedge lc(t_C^2) \wedge gc(T_C)$ is not satisfiable.

INPUT: c-table T_C
ALGORITHM:
1. If for some $t_C \in T_C$, the expression $lc(t_C) \wedge gc(T_C)$ is not satisfiable, then remove t_C from T_C. Section 2.2 describes one way this can be done. Alternative methods are described in [19], [20].
2. If for some c-tuples $t'_C, t''_C \in T_C$, t'_C and t''_C are *unifiable*, then substitute them in T_C with a c-tuple with main part $\bar{X} = x_1, x_2, ..., x_n$, where n is the arity of T_C and x_i are newly introduced variables, and local condition $(\bar{X} = main(t'_C) \wedge lc(t'_C)) \vee (\bar{X} = main(t''_C) \wedge lc(t''_C))$. Repeat this step as many times as possible.
3. Propagate $gc(T_C)$ to all local conditions, i.e. for every $t_C \in T_C$ set the local condition of t_C to $lc(t_C) \wedge gc(T_C)$. Next, simplify all local conditions using the algorithm presented in Section 2.2. Remove all c-tuples t_C for which $lc(t_C)$ is not satisfiable. Put TRUE as the global condition of the resulting c-table.
4. For every c-tuple $t_C \in T_C$, if $main(t_C)$ contains the variable x for the attribute A_i of T_C, and $lc(t_C) \Rightarrow (x = c)$ is a valid expression, where c is a constant, then replace x with c in $main(t_C)$.

Figure 2. The c-table normalization algorithm

The idea behind this definition is that if two c-tuples have local conditions that can not both hold under any valuation, then at most one of the c-tuples could be present in any representation of the c-table and therefore the two c-tuples can be merged into one. The algorithm we propose for normalizing a c-table T_C is shown in Figure 2. Table 2 shows the result of applying the first three steps of the presented normalization algorithm to an example c-table. The properties of the algorithm can be summarized by the following theorem.

Theorem 2: The presented normalization algorithm is correct, i.e. $Norm(T_C) \approx T_C$ for any c-table T_C, where *Norm* is the normalization function as described by the algorithm. As well, the normalization procedure runs in $O(\sqrt{2}^{(v+r)\cdot m} \cdot ((v+r)\cdot m)^c \cdot (2^{v+r} + d) \cdot (v+n))$ time, where v is the number of *variable c-tuples* in T_C, m is the greater of the size of the longest c-tuple and the size of the global condition of T_C, d is the number of distinct local conditions, n is the number of regular c-tuples with distinct main parts, r is the highest count of regular c-tuples that have the same main part, but distinct local conditions and c is a constant. Note that we have used the term variable c-tuple to denote a c-tuple that has at least one variable appearing in its main part and regular c-tuple do denote a c-tuple that has no variables appearing in its main part.

Proof(Sketch): In order to prove the first part of the theorem, we need to show that each of the five steps are equivalence preserving, i.e. that if T_C is a c-table and

O_i is used to denote the application of the $i^{\texttt{th}}$ step, then $O_i(T_C) \approx T_C$, for $i = 1$ to 4. The proofs that the statement is true for $i = 1$, 3 and 4 are trivial. For $i = 2$, note that the algorithm examines pair of c-tuples that can not appear in the same representation, since $lc(t'_C) \wedge lc(t''_C) \wedge gc(T_C)$ is not satisfiable for them. We can therefore combine them and merge their conditions. In this way at most one of the two original c-tuples will appear in any representation and the set of representations of the c-table is not changed.

To prove the time bound, note that Step 1 of the algorithm takes $O((\sqrt{2})^m \cdot m^c \cdot d)$ time. The reason is that the algorithm from Section 2.2, which takes $O((\sqrt{2})^m \cdot m^c)$ time, needs to be applied to be applied to d distinct local conditions. Steps 2 will take $O(v^3 \cdot (\sqrt{2})^m \cdot m^c)$ time. The reason is that it takes $O(v^2)$ time to check for

To see, why this is the case, note that at most $n+v$ groups of c-tuples are candidates to be merged in the algorithm. The reason is that regular c-tuples that have different main parts can not be united. Each group will contain at most $r+v$ c-tuples and therefore at most $\binom{v+r}{2} + \cdots + \binom{v+r}{v+r} = 2^{v+r} - (v+r) - 1$ iterations of the algorithm can be performed on it because we first explore pairs of c-tuples, then triplets and s.o. Finally, not that each iteration can take as much as $O(\sqrt{2}^{(v+r)\cdot m} \cdot (m \cdot (v+r))^c)$ time because the algorithm from Section 2.1 may need to be applied to an expression as long as $(v+r)\cdot m$. Step 4 does the simplification of the local conditions and can be performed in $O((\sqrt{2})^{m\cdot(v+r)} \cdot ((v+r)\cdot m)^c \cdot 2^{v+r} \cdot (n+v))$ time because as much as $2^{v+r} \cdot (n+v)$ distinct local conditions of size at most $m \cdot (v+r)$ need to be simplified. Step 5 can be performed in $O((\sqrt{2})^{m\cdot(v+r)} \cdot (m \cdot (v+r))^c \cdot 2^{v+r} \cdot v)$ time because at most $2^{v+r} \cdot v$ c-tuples need to be checked. Therefore, the whole algorithm will take $O(\sqrt{2}^{(v+r)\cdot m} \cdot ((v+r)\cdot m)^c \cdot (2^{v+r} + d) \cdot (v+n))$ time. ■

Note that the above algorithm can be improved if we try to avoid doing the same computation more than once. For example, we can buffer existing results and use them in performing new calculations. This is a reasonable thing to do because most of the presented algorithms for performing relational algebra operations produce c-tuples with local conditions that have subexpressions in common. We believe that optimizing the above algorithm and making it as fast as possible is of crucial importance to the general problem of working with c-tables and therefore more research needs to be done in the area.

The algorithm we propose for checking the equivalence of two c-tables T'_C and T''_C is presented in Figure 3. The following theorem summarizes its properties.

Theorem 3: The proposed algorithm for checking $T'_C \approx T''_C$ is correct and works in time $O(((2^{v+r} \cdot (n+v))^2 + d) \cdot \sqrt{2}^{(v+r)\cdot 2m} \cdot ((v+r) \cdot 2m)^c))$, where n,m,v,r and d are equal to the maximum of the corresponding values for the two c-tables as defined in Theorem 2.

INPUT: c-tables T'_C and T''_C
ALGORITHM:
1. Normalize T'_C and T''_C.
2. If the number of c-tuples in the two normalized c-tables is equal try to match them. Two c-tuples t'_C and t''_C match iff $(main(t'_C) = main(t''_C)) \wedge lc(t'_C) \wedge lc(t''_C)$ is satisfiable. An efficient way to check this satisfiability is to first check that $main(t'_C) = main(t''_C)$ is satisfiable, which can be done in $O(r)$ time, where r is the arity of the c-tuples being compared.
3.T'_C and T''_C are equivalent iff there exists a one-to-one match between the c-tuples of the normalized c-tables.

Figure 3. The c-table comparison algorithm

Proof(Sketch): First, note that if two c-tables are equivalent, their normalized forms will have the same number of c-tuples. The reason is that the normalization procedure unifies all c-tuples that can appear as one in a representation. Second, note that if two c-tables are equivalent there must exist a one-to-one mapping that matches the c-tuples in their normal forms. To prove the time bound, note that Step 1 takes $O(\sqrt{2}^{(v+r)\cdot m} \cdot ((v+r)\cdot m)^c \cdot (2^{v+r} + d) \cdot (v+n))$. Step 2 takes $O(x^2 \cdot \sqrt{2}^y \cdot y^c)$ time where x is the number of c-tuples in the bigger c-table and y is the size of the condition being compared. Since $x = 2^{v+r} \cdot (n+v)$ and $y = (v+r) * m$, the time bound follows. ■

III. Bag Relation Algebra for C-tables

So far we have defined the syntax and semantics of c-tables and shown how c-tables can be normalized. Next, we will describe how relational algebra[1] can be extended to handle c-tables. In particular, thanks to using the closed world assumption, we are able to develop a sound and complete extension of relational algebra that is closed. By a closed algebra we mean that for an arbitrary operator q of the algebra, and for an arbitrary c-table T_C, it is the case that $q(T_C)$ is a c-table. By sound we mean that only correct answers will appear in the result of $q(T_C)$ or formally that $Rep(q(T_C)) \subseteq q(Rep(T_C))$ ($q(Rep(T_C))$ denotes the result of applying q to each table in $Rep(T_C)$). Lastly, by complete we mean that all correct answers will be in the result of $q(T_C)$ or that $q(Rep(T_C)) \subseteq Rep(q(T_C))$. In this section we define the semantics of *projection*, *selection*, *inner join*, *monus* and *duplicate elimination* over c-tables with duplicate semantics. The grouping and aggregation operations are discussed in the next section.

[1]Our choice of relational algebra is arbitrary, i.e. any language with the expressive power of relational algebra, e.g. relational calculus, can be used instead.

A	*B*	*condition*
2	x	$x \neq 3$
2	4	TRUE

g.c. $x \neq 2$

B	*C*	*condition*
4	1	TRUE
2	z	$z > 3$

g.c. TRUE

Table III
EXAMPLE R_1 AND R_2 C-TABLES

B	*condition*
4	TRUE
2	$z > 3$

g.c. TRUE

B	*C*	*condition*
4	1	TRUE $\wedge\ 1 > 2$
2	z	$z > 3 \wedge z > 2$

g.c. TRUE

Table IV
THE RESULT OF $\pi_B(R_2)$ AND $\sigma_{C>2}(R_2)$

A. Projection

If T_C is a c-table with attributes $\bar{A}$, then we denote the projection of the attributes $\bar{A}'$ over this c-table as $\pi_{\bar{A}'}(T_C)$. The c-table $\pi_{\bar{A}'}(T_C)$ is constructed from the c-table T_C by removing all columns in $\bar{A} - \bar{A}'$ and leaving the same local and global conditions. Note that this is duplicate preserving projection. In order to prove that projection is well defined we need to show that $Rep(\pi_{\bar{A}'}(T_C)) \equiv \pi_{\bar{A}'}(Rep(T_C))$ or that the sets $S_1 = \{T|\exists v, v(\pi_{\bar{A}'}(T_C)) = T\}$ and $S_2 = \pi_{\bar{A}'}(\{T|\exists v, v(T_C) = T\})$ are equivalent. Let $T \in S_1$. Then there exists a valuation v, s.t. $v(\pi_{\bar{A}'}(T_C)) = T$. Let T' be a table constructed from T by adding the columns for the attributes $\bar{A} - \bar{A}'$ and filling them with arbitrary values. By the definition of duplicate preserving projection over bag relational tables it follows that $\pi_{\bar{A}'}(T') = T = v(\pi_{\bar{A}'}(T_C))$. As well, note that T' is a representation of T_C for a valuation v' extending the valuation v for the attributes $\bar{A} - \bar{A}'$ and therefore $\pi_{\bar{A}'}(v'(T_C)) = T = v(\pi_{\bar{A}'}(T_C))$, i.e. there exists a valuation v' s.t. $\pi_{\bar{A}'}(v'(T_C)) = T$, which implies that $T \in S_2$. Proving the reverse direction is similar.

Table 3 shows the two example c-tables that we will use throughout this section. The left part of Table 4 shows the result of $\pi_B(R_2)$. The projection, as we have defined it, without normalizing the resulting c-table takes $O(s)$ time, where s denotes the size of the c-table on which the projection is applied.

B. Selection

We denote the selection over a c-table T_C as $\sigma_\gamma(T_C)$, where γ is a predicate formula over $(\mathbb{R}, >, =, +) \cup (\mathbf{S}, =, \neq)$, which references the variables $\{A_i\}_{i=1}^n$ that have the same names as the attributes of T_C. We construct $\sigma_\gamma(T_C)$ from T_C by keeping the same global condition and adding by conjunction the local condition $\gamma_{\theta(t_C)}$ to each c-tuple t_C from T_C, where $\theta(t_C)$ is a substitution that substitutes every variable A_i with $t_C[A_i]$ (the value for the attribute A_i in t_C). In order to prove that we have defined selection correctly, we need to show that $Rep(\sigma_\gamma(T_C)) \equiv \sigma_\gamma(Rep(T_C))$. But this is equivalent to proving that there exist valuations v and v' s.t. $v(\sigma_\gamma(T_C)) = \sigma_\gamma(v'(T_C))$. However, we have defined selection over c-tables in such a way that $v(\sigma_\gamma(T_C)) = \sigma_\gamma(v(T_C))$ for any valuation v, which proves that selection is well defined. The right part of Table 4 shows the result of $\sigma_{C>2}(R_2)$. The selection, as we have defined it, without normalizing the resulting c-table takes $O(s*m)$ time, where s is the size of the c-table being normalized and m is the size of the selection condition.

A	*B*	*C*	*condition*
2	x	1	$x \neq 3 \wedge x = 4 \wedge$ TRUE
2	x	z	$x \neq 3 \wedge x = 2 \wedge z > 3$
2	4	1	TRUE $\wedge$ TRUE

g.c. $x \neq 2\ \wedge$ TRUE

Table V
THE RESULT OF $R_1 \bowtie R_2$

C. Inner Join

Suppose we are given a c-table T'_C with attributes $\{\bar{A}, \bar{B}\}$ and a c-table T''_C with attributes $\{\bar{B}, \bar{C}\}$. We denote the inner join of T'_C and T''_C on the set of attributes $\bar{B}$ as $T'_C \bowtie_{\bar{B}} T''_C$. We would like to define inner join in such a way that $Rep(T'_C \bowtie_{\bar{B}} T''_C) \equiv Rep(T'_C) \bowtie_{\bar{B}} Rep(T''_C)$. The algorithm we propose for calculating inner join is shown in Figure 4.

```
**INPUT**: c-tables $T'_C$ and $T''_C$ with common attributes $\bar{B}$.
**ALGORITHM**:
**1**. Rename all the variables that occur in $T''_C$ in such a way so that $T'_C$ and $T''_C$ share no variables in common.
**2**. For each c-tuple $t'_C$ in $T'_C$ find all c-tuples $t''_C$ in $T''_C$ such that $\pi_{\bar{B}}(main(t'_C)) = \pi_{\bar{B}}(main(t''_C))$ is satisfiable. For each found c-tuple $t''_C$ insert a c-tuple in the resulting c-table with main part $(t'_C, \pi_{\bar{C}}(t''_C))$ and local condition $lc(t'_C) \wedge lc(t''_C) \wedge (t'_C[\bar{B}] = t''_C[\bar{B}])$.
**3** Add a global condition to the resulting c-table comprised of the conjunction of the global conditions of $T'_C$ and $T''_C$.
```

Figure 4. The algorithm for calculating $T'_C \bowtie_{\bar{B}} T''_C$

Table 5 shows the result of $R_1 \bowtie R_2$. The proof that inner join is well defined is trivial and is skipped in order to save space. Inner join, as we have defined it, without normalizing the resulting c-table takes $O(n' \cdot n'' \cdot (\sqrt{2})^m \cdot m^c)$ time, where n' and n'' are the sizes of the c-tables being joined and m is double the size of the longest local condition in them.

D. Monus

Monus is the difference operators for bags. In bag relational algebra over bag relational tables it is defined as: $T' \dot{-} T'' = \{t_{[k]} | t \in T' \wedge k = max(count(t, T') - count(t, T''), 0)\}$, where $t_{[k]}$ is used to denote the tuple t repeated k times and $count$ is a function that returns the number of occurrences of the tuple specified as the first parameter in the table specified as the second parameter. In order to extend this definition to c-tables, we propose the algorithm shown in Figure 5.

```
INPUT: c-tables $T'_C$ and $T''_C$.
ALGORITHM:
1. Rename all variables that occur in $T''_C$ in such a way so that $T'_C$ and $T''_C$ share no variables in common.
2. Copy $T'_C$ in the result c-table $V$.
3. Construct a matrix $X[i,j]$, where $1 \leq i \leq n = |T'_C|$, $1 \leq j \leq m = |T''_C|$ and the operator $|.|$ denotes the number of c-tuples in the specified c-table. Set $X[i][j] = [main(t'_i) = main(t''_j)] \wedge lc(t'_i) \wedge gc(T'_C) \wedge lc(t''_j) \wedge gc(T''_C)$, where $t'_i$ and $t''_j$ are the $i^{\text{th}}$ and $j^{\text{th}}$ c-tuples in $T'_C$ and $T''_C$ respectively relative to some ordering of the c-tuples.
4. Add, with conjunction, to the global condition of $V$ the expression: $\bigwedge_{j=1}^{m} [\bigvee_{i=1}^{n} (Y[1,j] = \cdots = Y[i-1,j] = Y[i+1,j] = \cdots = Y[n,j] = 0 \wedge Y[i,j] = 1)] \wedge \bigwedge_{i=1}^{n} [\bigvee_{j=1}^{m} (Y[i,1] = \cdots = Y[i,j-1] = Y[i,j+1] = \cdots = Y[i,m] = 0 \wedge Y[i,j] = 1)]$, where $Y$ is a newly introduced matrix of variables that is of the same size as $X$.
5. Add, with conjunction, to the local condition of the $i^{\text{th}}$ c-tuple of $V$ the condition $\neg[\bigvee_{j=1}^{m} (X[i,j] \wedge (Y[i,j] = 1))]$.
```

Figure 5. The algorithm for calculating $V = T'_C \dot{-} T''_C$

What the algorithm does is to first rename the variables of T''_C so that they are distinct from those in T'_C. Next, it calculates the matrix X and sets a restriction on the possible values for the matrix Y. The value of $X[i,j]$ contains the condition that must hold for the c-tuple t'_i to be deleted from T'_C and the c-tuple that "deletes" it to be t''_j. The matrix $Y[i][j]$ has the restriction that for each j, there exists exactly one i, s.t. Y[i][j]=1, and that for each i, there exists exactly one j, s.t. $Y[i][j] = 1$. As well, the elements of the matrix Y can only take the values 0 and 1. The matrix Y is used to enforce the condition that every c-tuple t''_j in T''_C can be used to delete at most one c-tuple of T'_C and that every c-tuple t'_i in T'_C can be deleted at most once. Lastly, the local conditions that we add to the result c-table do the deletions. They specify that if for some valuation both $X[i][j]$ and $Y[i][j] = 1$ hold, i.e. if the c-tuple t'_i in T'_C matches with the c-tuple t''_j in T''_C and the valuation is such that t_i can not be deleted by any tuple other then t_j, and t_j can only delete t_i, then t_i should be deleted from the result. This guarantees that, given a valuation v, each c-tuple in T'_C may disappear from the corresponding representation only if there exists a matching c-tuple in T''_C that evaluates to TRUE. Moreover, given a valuation v, every c-tuple in T''_C can delete at most one c-tuple from T'_C, i.e. the algorithm is correct and $Rep(T'_C \dot{-} T''_C) \equiv [Rep(T'_C) \dot{-} Rep(T''_C)] \cup \{\emptyset\}$. Here $\{\emptyset\}$ is used to represent the empty c-table. We don't have an exact equality in the above formula since we constructed the global condition of $T'_C \dot{-} T''_C$ in such a way that we allow for $\{\emptyset\}$ to be a possible representation. It is our believe that this is an intrinsic problem of monus when dealing with the closed world assumption. A demonstration of how monus can be applied over our example c-tables is shown in Table 6 for the expression $R_1 \dot{-} R_2$. Monus, as we have defined it, without normalizing the resulting c-table takes $O(s' \cdot s'')$ time, where s' and s'' are the sizes of the c-tables on which the operation is performed.

$x \neq 3 \wedge x \neq 2 \wedge$ $x = 4 \wedge$ TRUE $\wedge$ TRUE	$x \neq 3 \wedge x \neq 2 \wedge$ $x = 2 \wedge z > 3 \wedge$ TRUE
TRUE $\wedge x \neq 2 \wedge 4 = 4$ TRUE $\wedge$ TRUE	FALSE

A	*B*	*condition*
2	x	$x \neq 3 \wedge \neg((X[1,1] \wedge Y[1,1] = 1) \vee (X[1,2] \wedge Y[1,2] = 1))$
2	4	TRUE $\wedge \neg((X[2,1] \wedge Y[2,1] = 1) \vee (X[2,2] \wedge Y[2,2] = 1))$

g.c. $(x \neq 2) \wedge ((Y[1,1] = Y[2,2] = 1 \wedge Y[1,2] = Y[2,1] = 0) \vee (Y[1,2] = Y[2,1] = 1 \wedge Y[1,1] = Y[2,2] = 0))$

Table VI
SHOWS THE MATRIX X AND THE RESULT FOR $R_1 \dot{-} R_2$

A	*B*	*condition*
2	x	$x \neq 3 \wedge (x \neq 4 \vee$ FALSE$)$
2	4	TRUE $\wedge (x \neq 4 \vee x = 3)$

g.c. $x \neq 2$

Table VII
THE RESULT OF $\varepsilon(R_1)$

E. Duplicate Elimination

The last relational algebra operation that we will explore is duplicate elimination and we denote it as $\varepsilon(T_C)$. Note that duplicate elimination can be defined as a grouping by all the attributes in the relational case. In the case of c-tables, we define $\varepsilon(T_C) = group_{\bar{A}}(T_C)$, where $\bar{A}$ are the attributes of T_C. Note that the result of the group operation is a *nested c-table* (see Table 8). We define the semantics of a nested c-table and of the *group* operation in Section 4.1. Also, note that $\varepsilon(Rep(T_C)) \equiv group_{\bar{A}}(Rep(T_C)) \equiv Rep(group_{\bar{A}}(T_C)) \equiv Rep(\varepsilon(T_C))$, which proves that duplicate elimination is well defined. The fact that the equation $group_{\bar{A}}(Rep(T_C)) \equiv Rep(group_{\bar{A}}(T_C))$ holds follows from the fact that the *group* operation is well defined over c-tables.

The result of $\varepsilon(R_1)$, where R_1 is the c-table defined in Table 3, is shown in Table 7.

In this section we have presented algorithms for implementing bag relational algebra operations over c-tables with duplicates. The presented operators are complete in the sense that all bag relational algebra operations can be expressed in terms of the presented ones.

IV. Applying Aggregation to C-tables

To the best of our knowledge, no research has been published in the areas of applying grouping and aggregation to c-tables. We are aware of research on applying aggregation to fuzzy numbers [21] and to random variables [22], but the query results in those methods are approximations. On the other hand, the research done in constraint databases [23] has explored the problem of aggregating over constraint databases. Unfortunately, the operation of aggregation in most constraint database systems is not closed [24]. We are also aware of recent research in the area of auditing confidential information ([25]), which however deals only with aggregation over Boolean variable.

A. Basics of Grouping

In general, we would like to be able to evaluate a relational query of the form ${}_{\bar{A}}\mathcal{F}_{agg_1(B_1),\ldots,agg_n(B_n)}T_C$, where $\bar{A} \cup \{B_i\}_{i=1}^n$ are the attributes of T_C, $\bar{A} = \{A_i\}_{i=1}^k$ and each agg_i is one of the operations min, max, sum, $count$ and avg. In the relational case the above expression is evaluated by grouping the tuples that have the same values for $\bar{A}$ into a single tuple that has this common value for its first k fields and the remaining fields are calculated by applying the aggregations to the $\bar{B}$ fields of the tuples in the group. In order to extend this definition to c-tables, we will need to be able to group c-tuples and perform aggregation on c-tuples. In this subsection we will explore how the grouping can be done and in the next subsection aggregation will be presented.

Let us denote the result of grouping by the attributes $\bar{A}$ of T_C as $group_{\bar{A}}(T_C)$. The result of this operation will be a *nested c-table*, i.e. the value of a field in it may be a bag of values. For example, in the result of the above grouping operation, the values for the attributes in $\bar{A}$ will be single values and for the attributes $B_1, .., B_n$ - bag of values. The result of $group_B R_1$ is shown in Table 8, where R_1 is shown in Table 4. We will refer to the c-tuples of a nested c-table as *nested c-tuples*. Formally, a nested c-tuple with single valued attributes $\{A_i\}_{i=1}^a$ and multi-valued attributes $\{B_i\}_{i=a+1}^b$ is the sequence of mappings from A_i to $D(A_i) \cup V_i$ for i ranging from 1 to a plus the sequence of mapping from B_i to a bag of values over $D(B_i) \cup V_i$ for i ranging from $a+1$ to b plus a local condition over ($\mathbb{R}$, $>$, $=$, $+$) $\cup$ (**S** $=$, $\neq$). Note that $D(A)$ was used to denote the domain of A and V_i was used to represent a possibly infinite, countable, set of variables over $D(A_i)$ if $i \leq a$ and over $D(B_i)$ otherwise. The semantics of a nested c-table are similar to the semantics of a regular c-table as described in Section 2.1 (see Equations 1,2,3). The difference is that a

A	*B*	*condition*
2	x	$x \neq 3 \wedge (x \neq 4 \vee$ FALSE$)$
2	4	TRUE $\wedge\ (x \neq 4 \vee x = 3)$
2 2	4	$x \neq 3\ \wedge$ TRUE $\wedge\ x = 4$

g.c. $x \neq 2$

Table VIII
THE RESULT OF $group_B R_1$

nested c-table represents a set of nested bag relational tables (see [26]) under different valuations and consists of a bag of nested c-tuples.

The algorithm we propose for computing $V = group_{\bar{A}}T_C$, where $\bar{A} \cup \bar{B}$ are the attributes of T_C and $\bar{A} = \{A_i\}_{i=1}^k$ and $\bar{B} = \{B_i\}_{i=1}^n$ is shown in Figure 6. In it we have used the concepts of semi-unifiable c-tuples and c-tuple comparison as presented in Definitions 2 and 3.

Definition 2: The c-tuples $\{t_C^i\}_{i=1}^n$ of the c-table T_C are semi-unifiable relative to the set of attributes $\bar{A}$ iff the formula $\bigwedge_{i,j=1}^{n} \pi_{\bar{A}} main(t_C^i) = \pi_{\bar{A}} main(t_C^j)$ is satisfiable.

Definition 3: We will write $t' \prec_A t''$, where t' and t'' are c-tuples and $\bar{A}$ is a set of attributes iff $main(\pi_{\bar{A}}(t''_C))$ can be constructed from $main(\pi_{\bar{A}}(t'_C))$ by substituting some of the variables in $main(\pi_{\bar{A}}(t'_C))$ with constants.

A demonstration of how the algorithm can be applied to an example c-table is shown in Appendix A. In order to show why the proposed algorithm for calculating $group_{\bar{A}}(T_C)$ is correct, i.e. why $group_{\bar{A}}(Rep(T_C)) \equiv Rep(group_{\bar{A}}(T_C))$, let's look at the algorithm's steps. Step 1 initializes the algorithm by copying T_C into the result c-table. Step 2 clusters the c-tuples into e-bags, relative to the attributes of $\bar{A}$. Note that step 2 is equivalence preserving and that c-tuples from different e-bags cannot contribute to the same resulting nested c-tuple under any valuation. That is why it suffices to perform the *group* operation to the c-tuples in each e-bag and then merge the results. Step 3 normalizes the e-bags, where part of the normalization is the removal of the global condition of the c-table. Step 4 partitions each e-bag further into r-bags. What is done in this step is to partition the space over which the local conditions of the c-tuples in the e-bags is defined into non-overlapping polyhedras. Each r-bag corresponds to a single polyhedra, or to a set of disjoint polyhedras. Note that this operation is equivalence preserving. The additional constraint that all the conjunctions that form the $D[i]$ of a given r-bag appear in the same set of c-tuples' local conditions guarantees that the r-bags partition the possible valuations, i.e. under every valuation the local condition of at most one r-bag of every e-bag will be true. In other words, given an arbitrary valuation and an e-bag of r-bags, either non of the c-tuples' local conditions will be true or the local conditions of all the c-tuples in exactly one r-bag will be true. What remains is to determine which c-tuples can be grouped together in

INPUT: c-tables T_C with attributes $\bar{A} \cup \bar{B}$, where $\bar{A} = A_1, \ldots, A_k$ and $\bar{B} = B_1, \ldots, B_n$.
ALGORITHM:
1. Copy T_C into V_C.
2. Cluster the c-tuples of V_C into biggest bags of semi-unifiable tuples relative to $\bar{A}$ - we will call this *e–bags*. If a c-tuple belongs to more than one e-bag, then make copies of the c-tuple and put a copy in each e-bag. To do so, add the local condition $x = i$ to the i^{th} copy of the c-tuple for $i < u$ and the local condition $\bigwedge_{i=1}^{u-1} x \neq i$ to the u^{th} copy, where x is a newly introduced variable and u is the number of times the c-tuple is copied.
3. Apply the c-table normalization procedure to the set of c-tuples in each e-bag.
4. Partition each e-bag further into r-bags. To do so, apply Step 3 from the algorithm in Figure 1 to $\bigvee_{i=1}^{p} lc(t_i)$, to get the set of non-overlapping conjunctions $\{c_i\}_{i=1}^{w}$, where $\{t_i\}_{i=1}^{p}$ are the c-tuples in the e-bag. Let $C = \{c_i\}_{i=1}^{w}$. Rewrite the local condition of each t_i as a disjunction of c_is. Next, break C into equivalence classes, relative to the operation $\sim$. We define $c_i \sim c_j$ iff the set of the rewritten local conditions in which the two conjunctions appear is the same. Form the array D in such a way that $D[i]$ is a disjunction of all the conjunctions in the i^{th} equivalence class. Reconstruct V_C by substituting each e-bag with a bag of *r-bags*, where the c-tuples in i^{th} r-bag of a given e-bag will have the same local condition as the corresponding value of $D[i]$ and the main parts will correspond to the c-tuples that contained the local conditions that formed the equivalence class corresponding to $D[i]$.
5. From each r-bag form a set of vertices, where each vertex corresponds to a distinct c-tuple in the r-bag (i.e. for duplicate c-tuples we will have a single vertex). Find all spanning undirected graphs for the built vertices that are transitive and have the property that if there is an edge between the vertices n_1 and n_3 and there exists a third vertex n_2 such that $t_1 \prec_{\bar{A}} t_2$ and $t_2 \prec_{\bar{A}} t_3$, where t_1, t_2 and t_3 are the c-tuples corresponding to the vertices, then there are edges between n_1 and n_2 and between n_2 and n_3. Next, the set of nested c-tuples that correspond to each graph are created. They union yields the result of doing the grouping. More precisely, suppose we are examining a r-bag r and a graph G associated with it. Since G is transitive, it will contain a set of disjoint complete sub-graphs, where each such sub-graph will correspond to a resulting nested c-tuples. Let the vertices in the complete sub-graph belong to the c-tuples $\{t_i\}_{i=1}^{p}$, then the corresponding nested c-tuple, will have the single value $(x_1, \ldots, x_k)$ for the attributes $\bar{A}$, the bag of values $\{|\ \pi_{\bar{B}} main(t_i)|\}_{i=1}^{p}$, for the attributes $\bar{B}$ and local condition $L_r \wedge R_G \bigwedge_{i=1}^{p} [(\pi_{\bar{A}} main(t_i)) = (x_1, \ldots, x_k)]$. The condition L_r is the local condition of the r-bag r. The condition R_G is the condition that projection on the $\bar{A}$ attributes of the main parts of the c-tuples that correspond to nodes in G that are connected should be equal, while the projection on the $\bar{A}$ attributes of the main parts of the c-tuples that correspond to nodes in G that are not connected should be different.

Figure 6. The algorithm for calculating $V_C = group_{\bar{A}} T_C$

each r-bag and this is done in Step 5. The step constructs a set of graphs for each r-bag, where each graph corresponds to a valuation. In a graph, there is an edge between two vertices if under the corresponding valuation it is true that $\pi_{\bar{A}}(main(t'_C)) = \pi_{\bar{A}}(main(t''_C))$, where t'_C and t''_C are the c-tuples corresponding to the vertices. It can be easily shown that a graph is valid, i.e. a corresponding valuation exists iff (1) the graph is transitive (2) if there is an edge between the vertices n_1 and n_3 and there exists a third vertex n_2 such that $n_1 \prec_{\bar{A}} n_2$ and $n_2 \prec_{\bar{A}} n_3$, then there are edges between n_1 and n_2 and between n_2 and n_3. This is why all the graphs having these two properties are constructed and these graphs show which c-tuples in the r-bag will be grouped relative to the attributes $\bar{A}$ under different valuations.

To calculate the time bound for applying the presented grouping algorithm, we will adopt the parameter names used in Theorem 2. More precisely, v is the number of *variable c-tuples* in T_C, m is the greater of the size of the longest c-tuple and the size of the global condition of T_C, n is the number of regular c-tuples with distinct main parts, r is the highest count of regular c-tuples that have the same main part, but distinct local conditions, s is the number of attributes in T_C and c is a constant. Then Step 2 of the grouping will take $O(2^v + n) \cdot s$ time because it may take as much as $O(2^v \cdot s)$ time to partition the variable c-tuples and then $O(n \cdot s)$ time to determine the groups for the regular c-tuples. Note that we get this low time bound thanks to the fact that regular c-tuples with distinct main parts can not appear in the same e-bag. Step 3 takes $O((2^v + n) \cdot (\sqrt{2}^{(v+r)\cdot m} \cdot ((v + r) \cdot m)^c \cdot 2^{v+r} \cdot v)$ because we may have as much as $2^v + n$ e-bags to normalize and each e-bag may contain as much as $v + r$ c-tuples. Step 4 will take $O((2^v + n) \cdot 3^{\sqrt{2}^{m\cdot(v+r)}} \cdot (m \cdot (v + r))^c)$ time because the size of a c-tuple's local condition may grow to a size of $O(m \cdot (v + r))$ after the normalization procedure is applied. Step 5 takes $O((2^v + n) \cdot 2^{v+r} \cdot 2^{v+n})$ time because there maybe as much as 2^{v+r} r-bags in each e-bag and each r-bag may contain as

A	B	*condition*
$a_1 \dots a_k$	$b_1^1 \dots b_n^1$	c
	$\dots$	
	$b_1^p \dots b_n^p$	

Table IX
A COMPLEX C-TUPLE t_C

A	B	*condition*
$a_1 \dots a_k$	$x_1 \dots x_n$	$c \wedge con(x_1, agg_1, b_1^1, \dots, b_1^p) \wedge \dots \wedge con(x_n, agg_n, b_n^1, \dots, b_n^p)$

Table X
THE RESULT OF ${}_{\bar{A}}\mathcal{F}_{agg_1(B_1),\dots,agg_n(B_n)}(t_C)$

much as $n + v$ distinct c-tuples and therefore there are 2^{v+n} possible graphs for each r-bag.

B. Performing the Aggregation

Now that we have defined how grouping over c-tables can be done, performing aggregation is straightforward. Consider the expression ${}_{A_1,\dots,A_k}\mathcal{F}_{agg_1(B_1),\dots,agg_n(B_n)}T_C$, where the sets $\bar{A} = A_1, \dots, A_k$ and $\bar{B} = B_1, \dots, B_n$ are disjoint and their union yields all the attributes in T_C. We can calculate the result of performing the aggregation by first computing the value of the nested c-table $group_{\bar{A}}T_C$ and then aggregating over the $\bar{B}$ attributes by introducing new variables in the main parts of the result and moving the aggregations to the local conditions. More precisely, suppose the nested c-tuple shown in Table 9 is in the result of $group_{\bar{A}}T_C$. Then this c-tuple will appear in ${}_{\bar{A}}\mathcal{F}_{agg_1(B_1),\dots,agg_n(B_n)}T_C$ as shown in Table 10. The way the operator con is calculated, relative to the value of arg, is shown in Table 11. The fact that aggregation, as we have defined it, is well-defined follows from the correctness of the grouping algorithm and the correctness of the con operator as defined in Table 11.

(agg)	$con(x, agg, b_1, \dots, b_n)$
min	$\bigwedge_{i=1}^{n} (x \leq b_i)$
max	$\bigwedge_{i=1}^{n} (x \geq b_i)$
$count$	n
sum	$x = \sum_{i=1}^{n} b_i$
avg	$\underbrace{x + \cdots + x}_{n \text{ times}} = \sum_{i=1}^{n} b_i$

Table XI
EXPLAINS THE OPERATOR con

V. CONCLUSION AND FUTURE RESEARCH

In this paper we have presented algorithms for querying c-tables extended with linear conditions and over the closed world assumption. We have chosen this representation because it is the least expressive extension of c-tables over which bag relational algebra with grouping and aggregation is closed and can be well defined. As expected, the running time of the presented algorithms is polynomial relative to the size of the certain information and non-polynomial relative to the size of the incomplete information.

A major topic for future research is optimizing the presented algorithms for performing the relational operations. For example, in the relational case, the join between two tables can be performed in different ways and depending on the method we choose the time for performing the join will vary. The same applies for joining c-tables. In general, there are many ways of doing the presented relational algebra operations over c-tables. The purpose of this paper was to define their semantics by presenting example algorithms for doing the operations. Other possible extensions of the presented work follow.

- Speed up the presented algorithms by sacrificing their accuracy, i.e. explore approximate query answering for incomplete information.
- Explore integrity constraints for incomplete information and how they can be used for semantic query optimization.
- Extend research done in relational databases, such as research on view maintenance and transaction control management, to databases containing incomplete information.
- Explore querying c-tables with order.

VI. EXAMPLE OF DOING GROUPING

Here we show an example of performing the grouping. Suppose we have the c-table R shown in Table 12 and we want to calculate $group_{A,B}(R)$. Table 13 shows the two e-bags that will be constructed after applying Step 2 of the algorithm presented in Figure 6. Tables 14 and 15 show respectively the value of the C and D array that are constructed in Step 4. Table 16 shows the r-bags for the first and second e-bag that are constructing by Step 4. When processing the first r-bag of the first e-bag Step 5 of the algorithm will consider four possible graphs, which are shown in Figure 7, where nodes 1, 2 and 3 correspond to the first, second and third c-tuple of the first r-bag of the first e-bag. The corresponding resulting nested c-tuples constructed by Step 5 of the algorithm for the examined r-bag are shown in Table 17.

REFERENCES

[1] T. Imielinski and W. Lipski, “Incomplete information in relational databases,” *Journal of Association of Computing*, vol. 31, no. 4, pp. 761–791, October 1984.

A	B	C	condition
x	y	1	$(x+y=3) \vee (x>4) \vee (x<0)$
x	3	2	$(x+y=3 \wedge x<2) \vee (x<0)$
2	3	3	$x>5$
2	3	4	TRUE
3	4	5	TRUE

Table XII
EXAMPLE C-TABLE R

A	B	C	*condition*
x	y	1	$(x+y=3) \vee (x>4) \vee (x<0)$
x	3	2	$(x+y=3 \wedge x<2) \vee (x<0)$
2	3	3	$x>5$
2	3	4	TRUE

A	B	C	*condition*
x	y	1	$(x+y=3) \vee (x>4) \vee (x<0)$
3	4	5	TRUE

Table XIII
E-BAGS IN $group_{A,B}(R)$

$C[1,\ldots,6]$	$C[7,\ldots,11]$
$x<0 \wedge t=1$	$x<0 \wedge t \neq 1$
$0 \leq x<2 \wedge x+y=3 \wedge t=1$	$0 \leq x<2 \wedge x+y=3 \wedge t \neq 1$
$2 \leq x \leq 4 \wedge x+y=3 \wedge t=1$	$2 \leq x \leq 4 \wedge x+y=3 \wedge t \neq 1$
$4<x \leq 5 \wedge t=1$	$4<x \leq 5 \wedge t \neq 1$
$x>5 \wedge t>1$	$x>5 \wedge t \neq 1$
$x+y \neq 3 \wedge 0 \leq x \leq 4$	

$C[1,2]$	$C[3,4]$
$x<0 \wedge t \neq 1$	$x>4 \wedge t \neq 1$
$0 \leq x \leq 4 \wedge x+y=3 \wedge$ $t \neq 1$	$(0 \leq x \leq 4 \wedge x+y \neq 3 \wedge$ $t \neq 1) \vee (t=1)$

Table XIV
THE ARRAY C FOR THE TWO E-BAGS

i	$D[i]$	c-tuples corresponding to $D[i]$
1	$C[1] \vee C[2]$	$\{1,2,4\}$
2	$C[3] \vee C[4]$	$\{1,4\}$
3	$C[5]$	$\{1,3,4\}$
4	$C[7] \vee C[8]$	$\{2,4\}$
5	$C[11]$	$\{3,4\}$
6	$C[6] \vee C[9] \vee C[10]$	$\{4\}$

i	$D[i]$	c-tuples corresponding to $D[i]$
1	$C[1] \vee C[2] \vee C[3]$	$\{1,2\}$
2	$C[4]$	$\{2\}$

Table XV
THE ARRAY D FOR THE TWO E-BAGS

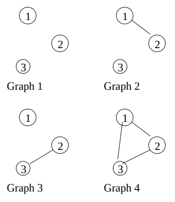

Figure 7. The four possible graphs

A	B	C	*condition*
x	y	1	$(x<0 \wedge t=1) \vee$ $(0 \leq x<2 \wedge x+y=3 \wedge t=1)$
x	3	2	
2	3	4	
x	y	1	$((2 \leq x \leq 4) \wedge (x+y=3) \wedge t=1) \vee$ $((4 \leq x \leq 5) \wedge (t=1))$
2	3	4	
x	3	1	$x>5 \wedge t=1$
2	3	3	
2	3	4	
x	3	2	$(x<0 \wedge t \neq 1) \vee$ $(0 \leq x<2 \wedge x+y=3 \wedge t \neq 1)$
2	3	4	
2	3	3	$x>5 \wedge t \neq 1$
2	3	4	
2	3	4	$(x+y \neq 3 \wedge 0 \leq x \leq 4) \vee$ $(2 \leq x \leq 4 \wedge x+y=3 \wedge t \neq 1) \vee$ $(4<x \leq 5 \wedge t \neq 1)$

A	B	C	*condition*
x	y	1	$(x<0 \wedge t \neq 1) \vee (0 \leq x \leq 4 \wedge$ $x+y=3 \wedge t \neq 1) \vee (x<4 \wedge t \neq 1))$
3	4	5	
3	4	5	$(0 \leq x \leq 4 \wedge x+y \neq 3 \wedge$ $t \neq 1) \vee (t=1)$

Table XVI
THE R-BAGS FOR THE TWO E-BAGS

A	B	C	*condition*
x	y	1	$y \neq 3 \wedge x \neq 2 \wedge R$
x	3	2	$y \neq 3 \wedge x \neq 2 \wedge R$
2	3	4	$y \neq 3 \wedge x \neq 2 \wedge R$
x	y	1 2	$x \neq 2 \wedge y=3 \wedge R$
2	3	4	$x \neq 2 \wedge y=3 \wedge R$
x	y	1	$x=2 \wedge y \neq 3 \wedge R$
2	3	2 4	$x=2 \wedge y \neq 3 \wedge R$
x	y	1 2 4	$x=2 \wedge y=3 \wedge R$

$R=((x<0 \wedge t=1) \vee (0 \leq x<2 \wedge x+y=2 \wedge t=1))$

Table XVII
THE CONTRIBUTION OF THE FIRST R-BAG OF THE FIRST E-BAG TO THE RESULT OF $group_{A,B}(R)$

[2] L. Libkin and L. Wong, "Some properties of query languages for bags," *Proceedings of Database Programming Languages*, pp. 97–114, 1994.

[3] G. Grahne, *The problem of Incomplete Information in Relational Databases*. Berlin: Springer-Verlag, 1991.

[4] R. Reiter, "A Sound and Sometimes Complete Query Evaluation Algorithm for Relational Databases with Null Values," *JACM*, vol. 33, no. 2, pp. 349–370, 1986.

[5] L. Y. Yuan and D.-A. Chiang, "A sound and Complete Query Evaluation Algorithm for Relational Databases with Null Values," *ACM*, 1988.

[6] L. Libkin, "Query language primitives for programming with incomplete databases," *Proceedings of DBPL*, 1995.

[7] P. Buneman, A. Jung, and A. Ohori, "Using powerdomains to generalize relational databases," *Theoretical Computer Science*, vol. 91, no. 1, 1991.

[8] R. Reiter, *On closed world databases, Logic and databases*. Plenum Press, 1978.

[9] G. Grahne, "Dependency satisfaction in databases with incomplete information," *Proceedings of International Conference on Very Large Data Bases*, pp. 37–45, 1984.

[10] J. A. Biskup, "A Formal approach to null values in database relations," *Advances in Database Theory*, pp. 299–341, 1981.

[11] E. F. Codd, "Understanding relations (Installment 7)," *FDT Bull. of ACM-SIGMOD*, vol. 3, no. 4, pp. 23–28, December 1975.

[12] ——, "Extending the database relational model to capture more meaning," *ACM Transactions on Database Systems*, vol. 4, no. 4, pp. 397–434, December 1979.

[13] J. Grant, "Null values in relational data base," *Information Processing Letters*, vol. 6, no. 5, pp. 156–157, October 1977.

[14] T. Imielinski and W. Lipski, "On representing incomplete information in a relational data base," *Proceedings of the 7th International Conference on Very Large Data Bases*, pp. 388–397, September 1981.

[15] M. Fischer and M. O. Rabin, "Super exponential complexity of Presburger arithmetic," *Project MAC Tech. Mem. 43. MIT*, 1974.

[16] S.Basu, "New Results on Qunatifier Elimination over Real Closed Fields and Applications to Constraint Databases," *JACM*, vol. 46, no. 4, pp. 537–555, 1999.

[17] J. L. Lassez and K. McAloon, "Applications of a Canonical Form of Generalized Linear Constraints," *Journal of Symbolic Computation*, vol. 13, pp. 1–24, 1992.

[18] J. L. Lassez, T. Huynh, and K. McAloon, "Simplification and elimination of redundant arithmetic constraints," *Proceedings of NACLP*, 1989.

[19] A. Tarski, "A Decision Method for Elementary Algebra and Geometry," *University of California Press*, 1951.

[20] F. Jerrante and C. Rackoff, " A Decision Procedure for the First Order Theory of Real Addition with Order," *SIAM Journal of Computing*, vol. 4, no. 1, pp. 69–76, 1975.

[21] G. Klir, U. Clar, and B. Yuan, *Fuzzy Set Theory. Foundations and Applications*. Prentice Hall, 1997.

[22] M. D. Springer, "The algebra of random variables," *Wiley series in probability and mathematical statistics*, 1979.

[23] G. Kuper, L. Libkin, and J. Paredaens, *Constraint Databases*. Springer, 1998.

[24] G. M. Kuper, "Aggregation in constraint databases," *Proceedings of the 1st International Workshop on Principles and Practice of Constraint Programming*, pp. 161–172, 1993.

[25] J. Kleinberg, C. Papadimitriou, and P. Raghavan, "Auditing Boolean Attributes," *PODS*, pp. 86–91, 2000.

[26] A. Makinouchi, "A consideration of normal form of non-necessarily-normalized relations in the relational data model," *Proceedings. of International. Conference. on Very Large Data Bases*, pp. 447–453, 1977.