Scalar Aggregation in FD-Inconsistent Databases

Marcelo Arenas[1], **Leopoldo Bertossi**[2], and **Jan Chomicki**[3]

[1] University of Toronto, Department of Computer Science, marenas@cs.toronto.edu
[2] Pontificia Universidad Catolica de Chile, Escuela de Ingeniería, Departamento de Ciencia de Computacion, bertossi@ing.puc.cl
[3] University at Buffalo, Department of Computer Science and Engineering, chomicki@cse.buffalo.edu

Abstract We consider here scalar aggregation queries in databases that may violate a given set of functional dependencies. We show how to compute consistent answers (answers true in every minimal repair of the database) to such queries. We provide a complete characterization of the computational complexity of this problem. We also show how tractability can be obtained in several special cases (one involves a novel application of the perfect graph theory) and present a practical hybrid query evaluation method.

1 Introduction

While integrity constraints capture important semantic properties of data, they are often unenforceable if data comes from different, autonomous sources (thus the integrated database may be inconsistent with the constraints). The notion of a *consistent query answer* [2] attempts to reduce this tension by using constraints to qualify query answers. A consistent answer is, intuitively, true regardless of the way the database is fixed to remove constraint violations. Thus answer consistency serves as an indication of its reliability.

Consistent query answers are potentially important in a datawarehouse context, where inconsistencies are likely to occur as the effect of the integration of data sources, with duplicate information, or delayed refreshment of the warehouse. In addition, it is in datawarehousing where aggregation queries are particularly important because they are used, in combination with OLAP methodologies, to better understand, in a global way, the peculiarities of clients, market and business behavior, and to support decision making.

In [2], in addition to a formal definition of a consistent query answer, a computational mechanism for obtaining such answers was presented. However, the queries considered were just first-order queries. Here we address in the same context the issue of *aggregation queries*. We limit, however, ourselves to single relations that possibly violate a given set of functional dependencies (FDs).

In defining consistent answers to aggregation queries we distinguish between queries with *scalar* and *aggregation* functions. The former return a single value for the entire relation. The latter perform grouping on an attribute (or a set of attributes) and return a single value for each group. Both kinds of queries use the same standard set of SQL-2 aggregate operators: `MIN`, `MAX`, `COUNT`, `SUM`, and `AVG`. In this paper, we address only aggregation queries with scalar functions.

Example 1. Assume we have the following database instance *Salary* (we are identifying the table with the database instance)

Salary	*Name*	*Amount*
	V.Smith	5000
	V.Smith	8000
	P.Jones	3000
	M.Stone	7000

and F is *Name* $\rightarrow$ *Amount*, meaning that *Name* functionally determines *Amount*, that is violated by the table *Salary*, actually by the tuples with the value *V.Smith* in attribute *Name*. If we pose the query `MIN(Amount)` to this database, we should get, independently of how the violation is fixed, the value 3000. Nevertheless, if we ask `MAX(Amount)`, we have a problem, because the maximum, 8000, comes from a tuple that participates in the violation of the functional dependency.

In [2] we defined an answer to a query posed to an inconsistent database as *consistent* when that same answer is obtained from every possible repair of the given database instance. Here, a repair is a new database instance that satisfies the given integrity constraints (ICs) and departs in a minimal way from the original database (see Section 2.1). In our case, the possible repairs are

$Salary_1$	*Name*	*Amount*
	V.Smith	5000
	P.Jones	3000
	M.Stone	7000

$Salary_2$	*Name*	*Amount*
	V.Smith	8000
	P.Jones	3000
	M.Stone	7000

In each repair `MIN(Amount)` returns the same value: 3000. On the other hand, `MAX(Amount)` returns a different value in each repair: 7000 or 8000. Thus, in the second case, there is no single consistent answer in the sense we had defined it. Nevertheless, an answer given by the initial database in the form of the interval $[6000, 9000]$, meaning that in every repair the maximum lies between 6000 and 9000, could be considered a consistent answer. In particular, we might be interested in getting, as a more accurate consistent answer, the smallest possible interval (the optimal lower and upper bounds), in this case the interval $[7000, 8000]$.

Example 2. Consider the *FD*: *StNumber* $\rightarrow$ *Name* and the inconsistent database instance

Jobs	*StNumber*	*Name*	*Activity*
	980134	*D.Singh*	*TeachAsst*
	980134	*F.Chen*	*ResAsst*
	980134	*D.Singh*	*Programmer*

This instance has two possible repairs

$Jobs_1$	*StNumber*	*Name*	*Activity*
	980134	*D.Singh*	*TeachAsst*
	980134	*D.Singh*	*Programmer*

$Jobs_2$	*StNumber*	*Name*	*Activity*
	980134	*F.Chen*	*ResAsst*

If we pose the query `COUNT(Jobs)` to these repairs, we obtain two different answers, 2 and 1, respectively. Thus, the optimal consistent answer is the interval [1,2]. □

Therefore, for aggregation queries we have to weaken a bit the notion of consistent query answer to allow answers that are not single values, but intervals.

In Section 2, we provide a general definition of consistent answer to an aggregation query with scalar functions. We also define a graph-theoretical representation of the database repairs, which is specifically geared towards FDs. In Section 3, we study the data complexity of the problem of computing consistent answers to aggregation queries in inconsistent databases. In Section 4, we show how to reduce the computational cost of computing such answers by decomposing the computation into two parts: one that involves standard relational query evaluation and one that computes the consistent answers in a smaller instance. In Section 5, we show that the complexity of computing consistent answers can be reduced by exploiting special properties of the given set of FDs or the given instances. In Section 6 we discuss related and further work.

2 Basic Notions

In this paper we assume that we have a fixed database schema containing only one relation schema R with the set of attributes U. We will denote elements of U by $A, B, \ldots$, subsets of U by $X, Y, \ldots$, and the union of X and Y by XY. We also have two fixed, disjoint infinite database domains: D (uninterpreted constants) and N (numbers). We assume that elements of the domains with different names are different. The database instances can be seen as first order structures that share the domains D and N. Every attribute in U is typed, thus all the instances of R can contain only elements either of D or N in a single attribute. Since each instance is finite, it has a finite active domain which is a subset of $D \cup N$. As usual, we allow built-in predicates over N that have infinite extensions, identical for all database instances. There is also a set of functional dependencies F over R that captures the semantics of the database. E.g., it may express the property that an employee has only a single salary. The instances of the database do not have to satisfy F (because the database may contain integrated data from multiple sources). A database that violates a given set of FDs is called *FD-inconsistent*.

2.1 Repairs

Given a database instance r, we denote by $\Sigma(r)$ the set of formulas $\{P(\bar{a}) \mid r \models P(\bar{a})\}$, where P is a relation name and $\bar{a}$ a ground tuple.

Definition 1. *The* distance $\Delta(r, r')$ *between data-base instances* r *and* r' *is the symmetric difference:* $\Delta(r, r') = (\Sigma(r) - \Sigma(r')) \cup (\Sigma(r') - \Sigma(r))$.

Definition 2. *For the instances* r, r', r'' , $r' \leq_r r''$ *if* $\Delta(r, r') \subseteq \Delta(r, r'')$, *i.e., if the distance between* r *and* r' *is less than or equal to the distance between* r *and* r''.

Definition 3. *Given a set of FDs* F *and database instances* r *and* r', *we say that* r' *is a* repair *of* r *w.r.t.* F *if* $r' \models F$ *and* r' *is* $\leq_r$*-minimal in the class of database instances that satisfy the set of FDs* F. □

We denote by $Repairs_F(r)$ the set of repairs of r w.r.t. F. Examples 1 and 2 illustrate the notion of repair. For a set of FDs, F, repairs are always obtained by deleting tuples from the table. For every instance r, the union of all repairs of r w.r.t. F is equal to r. These properties are not necessarily shared by other classes of ICs.

Definition 4. *The* core *of* r *is defined as* $Core_F(r) = \bigcap_{r' \in Repairs_F(r)} r'$. □

The *core* is a new database instance. If r consists of a single relation, then the core is the intersection of all the repairs of r. The core of r itself is not necessarily a repair of r. In example 1, the core is the table containing the tuples $(P.Jones, 3000)$ and $(M.Stone, 7000)$ only. In example 2, the core is empty.

2.2 Consistent Query Answers

First Order Queries Query answers for first order queries are defined in the standard way.

Definition 5. *Given a set of integrity constraints* F, *we say that a (ground) tuple* $\bar{t}$ *is a* consistent answer *to a query* $Q(\bar{x})$ *in a database instance* r, *and we write* $r \models_F Q(\bar{t})$ *(or* $r \models_F Q(\bar{x})[\bar{t}]$*), if for every* $r' \in Repairs_F(r)$, $r' \models Q(\bar{t})$. *If* Q *is a sentence, then true (false) is a* consistent answer *to* Q *in* r, *and we write* $r \models_F Q$ $(r \models_F \neg Q)$, *if for every* $r' \in Repairs_F(r)$, $r' \models Q$ $(r' \not\models Q)$.

Aggregation Queries The aggregation queries we consider are queries of the form: `SELECT f(...) FROM R`, where `f` is one of the aggregate operators `MIN`, `MAX`, `COUNT`, `SUM`, and `AVG`, applied to an attribute or the entire relation (as with the `COUNT(*)`). These queries return single numerical values by applying the corresponding *scalar function*, i.e., minimum for `MIN`, etc. In general, f will denote an aggregation query (or a scalar function itself). We write $r \models f = a$ to express that the aggregation query f returns the value a in the instance r.

Definition 6. *Given a set of integrity constraints* F, *we say that a numerical interval* $[a, b]$, *with* $-\infty < a \leq b < \infty$, *is a* consistent answer *to an aggregation query* f *in a database instance* r, *and we write* $r \models_F f \in [a, b]$ *(or* $r \models_F a \leq f \leq b$*) if for every* $r' \in Repairs_F(r)$, r' *returns to the query* f *a value* v *such that*

$a \leq v \leq b$. *If* $[a, b]$ *is a consistent answer, then* a *is called a* lower-bound-answer *and* b *an* upper-bound-answer. *An interval is an* optimal consistent answer *if no subinterval is a consistent answer. If* $[a, b]$ *is an optimal consistent answer, then* a *is called the greatest-lower-bound-answer (*glb-answer*) and denoted* $glb_F(f, r)$, *and* b *the least-upper-bound-answer (*lub-answer*) and denoted* $lub_F(f, r)$. □

We will be particularly interested in obtaining optimal consistent answers by querying the possibly inconsistent database, without computing and checking all possible repairs.

Note: Our notion of consistent query answer for aggregation queries with scalar functions has some shortcomings. For instance, while we guarantee that the value of the scalar function in every repair falls within the returned interval, clearly not every value in this interval will correspond to the value of the function obtained in some repair. Perhaps it is more natural for such queries to return a *set of values*, each corresponding to the value of the function in some repair. Along the same lines, one could represent such a set as an OR-object [12] or a C-table [11]. However, the interval-based representation is exponentially more compact than any explicit set-based representation.

Example 3. Consider the functional dependency $A \to B$ and the following database instance r_0 (columns represent tuples):

r_0							
A	1	1	2	2	$\cdots$	n	n
B	0	1	0	2	$\cdots$	0	2^{n-1}

The scalar function involving summing on the B attribute will assume each value between 0 and $2^n - 1$ in some repair of r_0. Therefore, any set-based representation of set of all of those values will be of exponential size. On the other hand, the interval-based representation $[0, 2^n - 1]$ has polynomial size. □

In addition to consistent answers, we will also consider other auxiliary notions of query answers in inconsistent databases.

Definition 7. *A value* v *is a* core answer *w.r.t.* F *to* f *in* r *if*

$$v = f\Big(\bigcap_{r' \in Repairs_F(r)} r'\Big).$$

A value v *is a* union answer *w.r.t.* F *to* f *in* r *if*

$$v = f\Big(\bigcup_{r' \in Repairs_F(r)} r'\Big).$$

Union answers are trivial for FDs, as the union of all the repairs of r is r itself, so the union answer reduces to $f(r)$.

2.3 Graph Representation

Given a set of FDs F and an instance r, all the repairs of r w.r.t. F can be succinctly represented as a graph.

Definition 8. *The* conflict graph $G_{F,r}$ *is an undirected graph whose set of vertices is the set of tuples in r and whose set of edges consists of all the edges (t_1, t_2) such that there is a dependency $X \rightarrow Y \in F$ for which $t_1[X] = t_2[X]$ and $t_1[Y] \neq t_2[Y]$. The* complement conflict graph $\bar{G}_{F,r}$ *is the complement of the conflict graph.*

Example 4. Consider a schema $R(AB)$, the set F of two functional dependencies $A \rightarrow B$ and $B \rightarrow A$, and an instance $r = \{(a_1, b_1), (a_1, b_2), (a_2, b_2), (a_2, b_1)\}$ over this schema. The conflict graph $G_{F,r}$ looks as follows:

$$\begin{array}{ccc} (a_1, b_1) & \text{——} & (a_1, b_2) \\ | & & | \\ (a_2, b_1) & \text{——} & (a_2, b_2) \end{array}$$

Proposition 1. *Each repair in $Repairs_F(r)$ corresponds to a maximal independent set in $G_{F,r}$ (or a maximal clique in $\bar{G}_{F,r}$) and vice versa.* □

The above graphs are geared specifically towards FDs. The repairs of other classes of constraints do not necessarily have similar representations.

2.4 Computational Complexity

Data Complexity The data complexity assumption [7, 15] makes it possible to study the complexity of query processing as a function of the size of the database instance.

Definition 9. *Given a class of databases $\mathcal{D}$, a class of queries $\mathcal{L}$ and a class of integrity constraints, the* data complexity *of computing consistent query answers is defined to be the complexity of (deciding the membership of) the sets $D_{F,\phi} = \{(D, \bar{t}) : D \models_F \phi[\bar{t}]\}$ for a fixed $\phi \in \mathcal{L}$ and a fixed finite set F of integrity constraints. This problem is* C-data-hard *for a complexity class C if there is a query $\phi \in \mathcal{L}$ and a finite set of integrity constraints F such that $D_{F,\phi}$ is C-hard.*

Upper and Lower Complexity Bounds We view computing glb- and lub-answers as an *optimization* problem. It is easy to see that for all SQL scalar aggregation queries the data complexity of this problem is in NPO - the class of optimization problems whose associated decision problems are in NP [4]. In several cases, we will show that computing glb- and lub-answers is in PO (polynomial-time computable optimization problems). To show intractability of computing a glb- (or lub)-answer to $f(r)$ for an aggregation query f, we will demonstrate that the decision problem $glb_F(\mathtt{f}, r)$ (or $lub_F(\mathtt{f}, r)$) θk (where $\theta \in \{\leq, \geq\}$) is NP-hard. If the latter is the case, then clearly computing the appropriate consistent answer is not in PO, unless P=NP.

3 Scalar Aggregation

Computing consistent answers by producing all the repairs of a database instance and then computing the aggregation queries for each of them may have a high complexity. The following instance r_1 with $2n$ tuples (columns represent tuples):

r_1							
A	1	1	2	2	$\cdots$	n	n
B	0	1	0	1	$\cdots$	0	1

has 2^n possible repairs for the single FD $A \to B$. So, in general, computing all repairs and then evaluating a query in each repair is not feasible. We have identified two ways of computing consistent answers by querying the given, inconsistent database instance, without having to compute all the repairs. Query transformation modifies the original query, Q, into a new query, $T(Q)$, that returns only consistent answers. We have applied this approach in [2] to restricted first order queries and universal integrity constraints. Except in some simple cases, this approach does not seem applicable to aggregation queries. For example, even when `MAX(A)` and `MIN(A)` queries can be written as first order queries, their resulting syntax does not allow to apply the methodology developed in [2] to them. Here, we use instead the fact that for FDs, the set of all repairs of an instance can be compactly represented as the conflict graph or its complement. We develop techniques and algorithms geared specifically towards this representation.

3.1 Core Answers

We start by considering *core answers*. For some aggregation operators, e.g., `COUNT` and `SUM` of nonnegative values, a core answer is a lower-bound-answer, but not necessarily a glb-answer. As we will see in Section 4, computing core answers to aggregation queries can be useful for computing consistent answers.

Theorem 1. *The data complexity of computing core answers for any scalar function is in PTIME.*

Proof: The core consists of all the isolated vertices in the conflict graph. □

In general, computing glb-answers and lub-answers is considerably more involved than computing core answers. We consider each aggregation operator in turn. In the following, r denotes an instance of a schema R.

3.2 `MIN` and `MAX`

Consider `MAX(A)` (`MIN(A)` is symmetric). In this case computing the lub-answer in r w.r.t. an arbitrary set of FDs F consists of evaluating `MAX(A)` in r. However, it is not obvious how to compute the glb-answer, namely the minimum of the set of maximums obtained by posing the query `MAX(A)` in every repair. Computing `MAX(A)` in $Core_F(r)$ gives us only a lower-bound-answer which does not have to be the glb-answer. We first provide a definition and prove a lemma which will also be useful later. Recall that U is the set of all attributes of the schema R.

Definition 10. *An FD $X \to Y$ is a* partition dependency *over R if $X \cup Y = U$ and $X \cap Y = \emptyset$.*

Lemma 1. *For any instance r of R and any partition dependency $d = X \to Y$ over R, the conflict graph $G_{d,r}$ is a union of disjoint cliques.*

Proof: Assume (t_1, t_2) and (t_2, t_3) are two edges in $G_{d,r}$ such that $t_1 \neq t_3$. Then $t_1[X] = t_2[X]$, $t_1[Y] \neq t_2[Y]$, $t_2[X] = t_3[X]$, and $t_2[Y] \neq t_3[Y]$. Therefore $t_1[X] = t_3[X]$. Also, $t_1[Y] \neq t_3[Y]$ because otherwise t_1 and t_3 would be the same tuple. So (t_1, t_3) is an edge in $G_{d,r}$. □

Theorem 2. *The data complexity of computing $glb_F($`MAX(A)`$, r)$ in r for a set of FDs F consisting of a single FD $X \to Y$ is in PTIME.*

Proof: Consider first the case where the FD is a partition dependency. Then by Lemma 1 the conflict graph $G_{F,r}$ is a union of disjoint cliques $C_1, \ldots, C_k$. Every repair picks exactly one tuple from each clique. Consider a tuple t in a clique C_j, $1 \leq j \leq k$. The value $t[A]$ is a maximum in a repair iff for every clique C_i, $1 \leq i \leq k$, there is a tuple t' in C_i such that $t'[A] \leq t[A]$. This condition can be tested in PTIME because the cliques are in our case just the connected components. Denote the set of all maximum values determined in this way as S. Then the glb-answer to `MAX(A)` is the minimum value in S.

A slight complication arises if the FD is not a partition dependency. The schema may contain some attributes other than those in XY. Let's call two tuples t and t' *XY-overlapping* if $t[XY] = t'[XY]$. There may be two different XY-overlapping tuples which are not in conflict although they are both in conflict with some other tuple. Thus, the conflict graph is not necessarily a union of disjoint cliques. However, it is easy to see that XY-overlapping tuples are always together in a repair. Therefore only the tuples with the maximum value of A among all XY-overlapping tuples can have a maximum value in a repair. All the remaining tuples can be removed without affecting the set of maximum values in repairs. If there is more than one tuple with the maximum value, an arbitrary one is selected. Denote the instance obtained in this way as r'. The conflict graph $G_{F,r'}$ is a union of disjoint cliques and the procedure described in the previous paragraph can be applied. □

Theorem 3. *There is a set of 2 FDs F_0 for which deciding whether $glb_{F_0}($`MAX(A)`$, r) \leq k$ in r is NP-data-hard.*

Proof: We reduce SAT to our problem. Consider a propositional formula $\varphi : C_1 \wedge \cdots \wedge C_n$ in CNF. Let $p_1, \ldots p_m$ be the propositional variables in φ. Construct a relation r with the list of attributes A, B, C, D and containing exactly the following tuples:

1. $(p_i, 1, C_j, 1)$ if making p_i true makes C_j true,
2. $(p_i, 0, C_j, 1)$ if making p_i false makes C_j true,
3. $(w, w, C_j, 2)$, $1 \leq j \leq n$, where w is a new symbol.

Consider also the FDs $A \to B$ (each propositional variable cannot have more than one truth value) and $C \to D$. The crucial observation is that the $glb_{F_0}($`MAX(D`
`)`$, r) = 1$ iff φ is satisfiable. □

3.3 COUNT(*) and SUM

We consider only COUNT(*): SUM is very similar.

Theorem 4. *If the set of FDs F is equivalent to a single dependency $X \rightarrow Y$, $X \cap Y = \emptyset$, the data complexity of computing $glb_F(\texttt{COUNT}(*), r)$ (or $lub_F(\texttt{COUNT}(*), r)$) in r is in PTIME.*

Proof: The glb-answer can be computed using the following set of SQL views (the lub-answer is obtained in a similar way):

```
CREATE VIEW S(X,Y,C) AS
  SELECT X,Y,COUNT(*) FROM R
  GROUP BY X,Y;
CREATE VIEW T(X,C) AS
  SELECT X, MIN(C) FROM S
  GROUP BY X;

SELECT SUM(C) FROM T;
```

□

To characterize the remaining cases, we prove two lemmas about maximum cliques in conflict graphs.

Lemma 2. *There is a set of 2 FDs F_1 for which the problem of determining the existence of a repair of r of size $\geq k$ is NP-data-hard.*

Proof: Reduction from 3-COLORABILITY. Given a graph $G = (N, E)$, with $N = \{1, 2, \ldots, n\}$, and given the colors w (white), b (blue) and r (red), we define the relation $P(A, B, C, D)$ by means of the following rules:

1. for every $1 \leq i \leq n$, $(i, w, i, w) \in P$, $(i, b, i, b) \in P$ and $(i, r, i, r) \in P$.
2. for every $(i, j) \in E$, $(i, w, j, b) \in P$, $(i, w, j, r) \in P$, $(i, b, j, w) \in P$, $(i, b, j, r) \in P$, $(i, r, j, w) \in P$ and $(i, r, j, b) \in P$.

We consider the set of functional dependencies $A \rightarrow B$ and $C \rightarrow D$. The crucial property is that G is 3-colorable iff there is a repair P' of P with exactly $n + 2 \cdot |E|$ tuples (the maximum possible number of tuples in a repair). □

Lemma 3. *There is a set of 2 FDs F_2 for which the problem of determining the existence of a repair of r of size $\leq k$ is NP-data-hard.*

Proof: Modification of the lower bound proof of Theorem 3. We build the instance by using the same tuples of the kind (1) and (2), as well as sufficiently many tuples of the kind (3), each with a different new symbol w. It is enough to have $3n(n + 1)$ such tuples, where n is the number of clauses. The formula is satisfiable iff there is a repair of size $\leq 3n$. □

The lemmas 2 and 3 imply the following theorems.

Theorem 5. *There is a set of two FDs F_1 for which determining whether $lub_{F_1}(\texttt{COUNT}(*), r) \geq k$ in r is NP-data-hard.*

Theorem 6. *There is a set of two FDs F_2 for which determining whether $glb_{F_2}(\texttt{COUNT}(*), r) \leq k$ in r is NP-data-hard.* □

The above results establish the intractability of determining lub-answers and glb-answers to `COUNT(*)` in a general setting. Similar results hold for `SUM`. We will see that the boundary between the tractable and the intractable can be pushed farther in several special cases.

3.4 COUNT(A)

We assume here that distinct values of A are counted (`COUNT (DISTINCT A)`).

Theorem 7. *There is a single FD $d_0 = B \rightarrow A$ for which determining whether $glb_{d_0}(\texttt{COUNT(A)}, r) \leq k$ in r is NP-data-hard.*

Proof: To see that the lower bound holds, we will encode an instance of the HITTING SET problem in r. For every set S_i in the given collection C and every element $x \in S_i$ we put the tuple (i, x) in r. There is in C a hitting set of size less than or equal to k iff there is a repair of r with at most k different values of the first attribute A. $\square$

Theorem 8. *There is a single FD $d_1 = B \rightarrow A$ for which determining whether $lub_{d_1}(\texttt{COUNT(A)}, r) \geq k$ in r is NP-data-hard.*

Proof: We reduce SAT to this problem. Let the instance r be the conjunction of clauses $\varphi : \; C_1 \wedge \ldots \wedge C_n$. Consider the functional dependency $X \rightarrow Y$ and the database instance $r(X, Y, A)$ with the following tuples:

1. $(p_i, 1, C_j)$ if making p_i true makes C_j true.
2. $(p_i, 0, C_j)$ if making p_i false makes C_j true.

Then, φ is satisfiable iff $lub_{d_1}(\texttt{COUNT(A)}, r) \geq n$. $\square$

3.5 AVG

Theorem 9. *If a set of FDs F is equivalent to a single dependency $X \rightarrow Y$, with $X \cap Y = \emptyset$, then the data complexity of the problem of computing $glb_F(\texttt{AVG(A)}, r)$ (or $lub_F(\texttt{AVG(A)}, r)$) in r is in PTIME.*

Proof:[1] First, the problems of finding the *glb* and *lub* answers for `AVG` with one functional dependency can be reduced in polynomial time to the following problem:

P1: There are m bins. Each bin contains objects of different colors. No two bins have objects of the same color. All objects of the same color have the same weight. One has to choose exactly one color for each bin in such a way that the sum of the weights of all objects of the chosen colors divided by the total number of such objects (i.e., the average weight AVG of objects of the chosen colors) is maximized.

To solve **P1**, consider the well-known "*2-OPT*" strategy of starting with an arbitrary selection $\langle c_1, c_2, ..., c_m \rangle$ of one color each from each of the m bins. The

[1] The proof of this theorem is due to Vijay Raghavan and Jeremy Spinrad.

2-OPT strategy is simply to replace a color from one bin with a different color from the same bin if so doing increases the value of the average weight of objects of the colors in the selection.

This *2-OPT* strategy can be shown to converge to the optimum. In addition, it can be designed in such a way the it runs in polynomial time. □

Theorem 10. *There is a set of two FDs F_3 for which determining whether $glb_{F_3}(\texttt{AVG(A)}, r) \leq k$ in r is NP-data-hard.*

Proof: We can use the same reduction as in theorem 3. There is a satisfying assignment iff there is a repair for which $\texttt{AVG(D)} = 1$ (otherwise the glb-answer is bigger than 1) iff $glb_{F_3}(\texttt{AVG(D)}, r) \leq 1$. □

Theorem 11. *There is a set of two FDs F_4 for which determining whether $lub_{F_4}(\texttt{AVG(A)}, r) \geq k$ in r is NP-data-hard.*

Proof: We reduce SAT to our problem. Change the tuples of the instance in the proof of theorem 3 as follows:

3'. $(w, w, C_j, -2)$, $1 \leq j \leq n$, where w is a new symbol.

There is a satisfying assignment iff $glb_{F_4}(\texttt{AVG(D)}, r) \geq 1$. □

3.6 Summary of Complexity Results

It is easy to show that each of the problems considered before belong to the class NP.

	glb-answer		lub-answer	
	$\|F\| = 1$	$\|F\| \geq 2$	$\|F\| = 1$	$\|F\| \geq 2$
`MIN(A)`	PTIME	PTIME	PTIME	NP-complete
`MAX(A)`	PTIME	NP-complete	PTIME	PTIME
`COUNT(*)`	PTIME	NP-complete	PTIME	NP-complete
`COUNT(A)`	NP-complete	NP-complete	NP-complete	NP-complete
`SUM(A)`	PTIME	NP-complete	PTIME	NP-complete
`AVG(A)`	PTIME	NP-complete	PTIME	NP-complete

4 Hybrid Computation

As we have seen, determining glb-answers and lub-answers is often computationally hard. However, it seems that hard instances of those problems are unlikely to occur in practice. We expect that in a typical instance a large majority of tuples are not involved in any conflicts. If this is the case, it is advantageous to break up the computation of an lub-answer to f in r into three parts: (1) the computation of f in the core of r, (2) the computation of an lub-answer to f in the complement of the core of r (which should be small), and (3) the combination of the results of (1) and (2). The step (1) can be done using a DBMS because the core of r can be computed using a first-order query (Theorem 1).

Definition 11. *The scalar function* f admits a g-decomposition *of its lub-answers (resp. glb-answers) w.r.t. a set of FDs* F *if for every instance* r *of* R*, the lub-answer (resp. glb-answer)* v *to* f *satisfies the condition* $v = g(f(Core_F(r)), v')$*, where* $v' = lub_F(f, r - Core_F(r))$ *(resp.* $v' = glb_F(f, r - Core_F(r))$*).*

Theorem 12. *The following pairs describe* g*-decompositions admitted by scalar functions* f*:*
1. $f = \texttt{MIN(A)}$, $g = \min$;
2. $f = \texttt{MAX(A)}$, $g = \max$;
3. $f = \texttt{COUNT(*)}$, $g = +$;
4. $f = \texttt{SUM(A)}$, $g = +$.

5 Special Cases

We consider here various cases when the conflict graph (or its complement) has some special form that could be used to reduce the complexity of computing answers to aggregation queries.

5.1 BCNF

We show here that if the set of FDs F has two dependencies and the schema R is in BCNF, computing lub-answers can be done in PTIME. This should be contrasted with Theorem 5 which showed that two dependencies without the BCNF assumption are sufficient for NP-hardness.

Lemma 4. *If* R *is in BCNF and* F *is equivalent to a set of FDs with 2 dependencies, then* F *is equivalent to a set of FDs with 2 partition dependencies* $X_1 \to Y_1$ *and* $X_2 \to Y_2$*.* □

Therefore, WLOG we can assume that $|F| = 2$ and $F = \{d_1, d_2\}$ where d_1 and d_2 are different partition dependencies. (The case of $|F| = 1$ has already been shown to be in PTIME, even without the BCNF assumption.)

Definition 12. *A* chord *in a cycle is an edge connecting two nonconsecutive vertices of the cycle.*

Lemma 5. *Every cycle of length* k *where* k *is odd and* $k > 3$ *in* $G_{\{d_1,d_2\},r}$ *has a chord.*

Proof: Such a cycle has two consecutive edges (t_1, t_2) and (t_2, t_3) that belong both to $G_{d_1,r}$ or both to $G_{d_2,r}$. Therefore, by Lemma 1 the edge (t_1, t_3), which is a chord, also belongs to one of those graphs, and consequently to $G_{\{d_1,d_2\},r}$. □

Note: For the above property to hold, it is essential for the cycle in the conflict graph to be odd. Example 4 shows an even cycle of length 4 that does not have a chord. That implies that conflict graphs in the case of two FDs are not necessarily *chordal* [5] and thus efficient algorithms for the computation of maximum independent set in such graphs [9] are not applicable.

Lemma 6. *Every cycle of length k where k is odd and $k > 3$ in the complement conflict graph $\bar{G}_{\{d_1,d_2\},r}$ has a chord.*

Proof: To give the idea of the proof, we consider the case of $R(A, B)$ and $d_1 = A \rightarrow B$ and $d_2 = B \rightarrow A$.

Assume $(t_1, t_2, \ldots, t_k, t_1)$ is a cycle in $\bar{G}_{\{d_1,d_2\},r}$. Let $t_i = (a_i, b_i)$, $1 \leq i \leq k$, where the a_i's and b_i's are distinct variables. We write down the formula ϕ that expresses the property that the consecutive vertices in the cycle are in $\bar{G}_{\{d_1,d_2\},r}$: $\phi \equiv \bigwedge a_i \neq a_{i+1} \wedge \bigwedge b_i \neq b_{i+1}$, where the indexes are interpreted cyclically, i.e., $k + 1 = 1$. Now we write down the formula ψ that expresses the property that there are no chords in the cycle. This formula is a conjunction of the formulas $\psi_{i,j}$ (for every pair (i, j) of nonconsecutive vertices in the cycle) that express the property that there is a conflict between t_i and t_j: $\psi_{i,j} \equiv (a_i = a_j \wedge b_i \neq b_j \vee b_i = b_j \wedge a_i \neq a_j) \equiv (a_i = a_j \vee b_i = b_j) \wedge (a_i \neq a_j \vee b_i \neq b_j)$. Therefore $\psi_{i,j}$ postulates at least one equality: $a_i = a_j$ or $b_i = b_j$.

Now the counting argument. Assume $\phi \wedge \psi$ is satisfiable. The formula ϕ postulates n inequalities between the a_i's and n inequalities between the b_i's. The formula ψ postulates $\frac{n(n-3)}{2}$ inequalities and the same number of equalities that involve either a_i's or b_i's. WLOG we assume that at least half of them, i.e., $\lceil\frac{n(n-3)}{4}\rceil$ involve a_i's. Therefore, for $n \geq 5$, the equalities imply together yet another equality. (The assumption that all the equalities holding have disjoint variables leads to contradiction.) Thus the total number of equalities is $\frac{n(n-3)}{2}+1$. Now

$$2n + \frac{n(n-3)}{2} + \frac{n(n-3)}{2} + 1 = n(n-1) + 1$$

and is greater than the number of 2-element sets consisting only of a_i's or b_i's. Therefore for some i and j, we have both $a_i = a_j$ and $a_i \neq a_j$ (or $b_i = b_j$ and $b_i \neq b_j$), which contradicts the satisfiability of $\phi \wedge \psi$. Thus an odd cycle of length ≥ 5 has to have a chord. □

Definition 13. *A graph is* perfect *if its chromatic number is equal to the size of its maximum clique.*

Strong Perfect Graph Conjecture: A graph G is perfect iff every odd cycle in G or $\bar{G}$ has a chord.

This conjecture has been shown to hold for many classes of graphs, including claw-free graphs [5].

Definition 14. *A graph is* claw-free *if it does not contain an induced subgraph (V_0, E_0) where $V_0 = \{t_1, t_2, t_3, t_4\}$ and $E_0 = \{(t_2, t_1), (t_3, t_1), (t_4, t_1)\}$.*

Lemma 7. *If R is in BCNF over $F = \{d_1, d_2\}$, then for every instance r of R, the conflict graph $G_{\{d_1,d_2\},r}$ is claw-free and perfect.*

Proof: Assume that the conflict graph contains a claw (V_0, E_0) where $V_0 = \{t_1, t_2, t_3, t_4\}$ and $E_0 = \{(t_2, t_1), (t_3, t_1), (t_4, t_1)\}$. Then two of the edges in E_0, say (t_2, t_1) and (t_3, t_1) come from one of $G_{d_1,r}$ or $G_{d_2,r}$. But the by Lemma 1, the edge (t_3, t_2) also belongs to that graph, and consequently to $G_{\{d_1,d_2\},r}$. Thus the subgraph induced by V_0 is not a claw.

As the conflict graph is claw-free, the Strong Perfect Graph Conjecture holds for it and Lemmas 5 and 6 yield together the fact that it is perfect. □

Theorem 13. *If R is in BCNF and the given set of FDs F is equivalent to one with at most two dependencies, computing $lub_F(\texttt{COUNT}(*), r)$ in any instance r of R can be done in PTIME.*

Proof: The theorem follows from Lemma 7 and the fact that in perfect claw-free graphs computing a maximum independent set can be done in $O(n^{5.5})$ [10]. □

What about $|F| > 2$? In this case the conflict graph does not have to be claw-free, so it is not clear whether the Strong Perfect Graph Conjecture holds for it. The conflict graph does not even have to be perfect. Take a conflict graph consisting of a cycle of length 5 where the edges corresponding to the dependencies d_1, d_2 and d_3 alternate. The chromatic number of this graph is 3, while the size of the maximum clique is 2.

5.2 Disjoint Union

Theorem 14. *If the instance r is the disjoint union of two instances that separately satisfy F, computing $lub_F(\texttt{COUNT}(*), r)$ can be done in PTIME.*

Proof: In this case, the only conflicts are between the parts of r that come from different databases. Thus the conflict graph is a bipartite graph. For bipartite graphs determining the maximum independent set can be done in PTIME. □

Note that the assumption in Theorem 14 is satisfied when the instance r is obtained by merging together two consistent databases in the context of database integration.

6 Related and Further Work

We can only briefly survey the related work here. A more comprehensive discussion can be found in [2]. The need to accommodate violations of functional dependencies is one of the main motivations for considering disjunctive databases [12, 14] and has led to various proposals in the context of data integration [1, 3, 8, 13]. A purely proof-theoretic notion of consistent query answer comes from Bry [6]. None of the above approaches considers aggregation queries.

Many further questions suggest themselves. First, is it possible to identify more tractable cases and to reduce the degree of the polynomial in those already identified? Second, is it possible to use approximation in the intractable cases? The INDEPENDENT SET problem is notoriously hard to approximate, but perhaps the special structure of the conflict graph may be helpful. Finally, it would be very interesting to see if our approach can be generalized to broader classes of queries and integrity constraints.

Finally, alternative definitions of repairs and consistent query answers that include, for example, preferences are left for future work. Also, one can apply further aggregation to the results of aggregation queries in different repairs, e.g., the average of all `MAX(A)` answers.

Acknowledgments: Work supported in part by FONDECYT Grant 1000593 and NSF grant INT-9901877/CONICYT Grant 1998-02-083. The comments of the anonymous referees are gratefully acknowledged.

References

1. S. Agarwal, A.M. Keller, G. Wiederhold, and K. Saraswat. Flexible Relation: An Approach for Integrating Data from Multiple, Possibly Inconsistent Databases. In *IEEE International Conference on Data Engineering*, 1995.
2. M. Arenas, L. Bertossi, and J. Chomicki. Consistent Query Answers in Inconsistent Databases. In *Proc. ACM Symposium on Principles of Database Systems (ACM PODS'99, Philadelphia)*, pages 68–79, 1999.
3. C. Baral, S. Kraus, J. Minker, and V.S. Subrahmanian. Combining Knowledge Bases Consisting of First-Order Theories. *Computational Intelligence*, 8:45–71, 1992.
4. D. P. Bovet and P. Crescenzi. *Introduction to the Theory of Complexity.* Prentice Hall, 1994.
5. A. Brandstädt, V. B. Le, and J. P. Spinrad. *Graph Classes: A Survey.* SIAM, 1999.
6. F. Bry. Query Answering in Information Systems with Integrity Constraints. In *IFIP WG 11.5 Working Conference on Integrity and Control in Information Systems.* Chapman &Hall, 1997.
7. A. K. Chandra and D. Harel. Computable Queries for Relational Databases. *Journal of Computer and System Sciences*, 21:156–178, 1980.
8. Phan Minh Dung. Integrating Data from Possibly Inconsistent Databases. In *International Conference on Cooperative Information Systems*, Brussels, Belgium, 1996.
9. Fanica Gavril. Algorithms for Minimum Coloring, Maximum Clique, Minimum Covering by Cliques, and Maximum Independent Set of a Chordal Graph. *SIAM Journal on Computing*, 1(2):180–187, 1972.
10. W-L. Hsu and G.L. Nemhauser. Algorithms for Minimum Covering by Cliques and Maximum Clique in Claw-free Perfect Graphs. *Discrete Mathematics*, 37:181–191, 1981.
11. T. Imieliński and W. Lipski. Incomplete Information in Relational Databases. *Journal of the ACM*, 31(4):761–791, 1984.
12. T. Imieliński, S. Naqvi, and K. Vadaparty. Incomplete Objects - A Data Model for Design and Planning Applications. In *ACM SIGMOD International Conference on Management of Data*, pages 288–297, Denver, Colorado, May 1991.
13. J. Lin and A. O. Mendelzon. Merging Databases under Constraints. *International Journal of Cooperative Information Systems*, 7(1):55–76, 1996.
14. R. van der Meyden. Logical Approaches to Incomplete Information: A Survey. In J. Chomicki and G. Saake, editors, *Logics for Databases and Information Systems*, chapter 10. Kluwer Academic Publishers, Boston, 1998.
15. M. Y. Vardi. The Complexity of Relational Query Languages. In *ACM Symposium on Theory of Computing*, pages 137–146, 1982.