Containment of Aggregate Queries*

Sara Cohen[1], Werner Nutt[2], and Yehoshua Sagiv[1]

[1] School of Computer Science and Engineering
The Hebrew University of Jerusalem
Jerusalem 91904, Israel
{sarina,sagiv}@cs.huji.ac.il
[2] School of Mathematical and Computer Sciences
Heriot-Watt University, Edinburgh EH14 4AS
Scotland, UK
nutt@macs.hw.ac.uk

Abstract. The problem of deciding containment of aggregate queries is investigated. Containment is reduced to equivalence for queries with *expandable* aggregation functions. Many common aggregation functions, such as *max*, *cntd* (count distinct), *count*, *sum*, *avg*, *median* and *stdev* (standard deviation) are shown to be expandable. It is shown that even in the presence of integrity constraints, containment can be reduced to equivalence. For conjunctive *count* and *sum*-queries, simpler characterizations for containment are given, that do not require checking equivalence. These results are built upon in order to solve the problem of finding maximally-contained sets of rewritings for conjunctive *count*-queries.

1 Introduction

Rewriting queries using views is a fundamental problem in databases. Research in view usability attempts to answer the questions "what can be computed from what, and how?" Such techniques have applications in a number of areas. In query optimization, the execution of a query can be accelerated if results from previous queries can be used to compute answers [23,2]. In designing information systems that should periodically process a huge number of a priori known queries, it can be beneficial to store beforehand intermediate results that can be useful for as many queries as possible [17,20]. In integrating heterogeneous information sources, one approach is to model the contents and the relationships of a set of sources by setting up one rich schema and to describe the sources as views on this schema. In order to answer queries issued against the schema we must rewrite them using the "views" [16].

While the focus of this work was for a long time on non-aggregate queries, interest in aggregate queries has been motivated recently by the rapid expansion of data warehousing and decision support, where aggregate queries typically occur. Optimization based on the reuse of previously computed results is particularly

* This work was supported by the Israel Science Foundation (Grant 96/01-1) and the British EPSRC (Grant GR/R74932/01).

D. Calvanese et al. (Eds.): ICDT 2003, LNCS 2572, pp. 111–125, 2003.

promising for aggregate queries, since often a huge amount of data is touched to produce a single aggregate value. In fact, most existing data warehouses make use of this idea in a rather ad hoc way in their optimization algorithms [13].

There are several papers that provide characterizations for equivalence of aggregate queries [18,6,9,4]. Algorithms for rewriting aggregate queries have been presented [6,7,10,21]. However, these algorithms investigated the problem of finding a rewriting that is *equivalent* to the given query. They also did not use unions of aggregate queries to rewrite aggregate queries and thus, could not find rewritings in some cases, even though such rewritings existed.

This paper investigates the problem of deciding containment of aggregate queries. To the best of our knowledge, there are no previous results on this topic. See Section 5 for related work on containment of non-aggregate queries. We show how it is possible to reduce containment of a wide class of aggregate queries to equivalence. We build on our containment results and present a method for finding maximally-contained sets of rewritings of *count*-queries. Maximally-contained sets of rewritings are of importance in the context of information integration. This solves an open problem mentioned in [14].

Section 2 contains basic definitions. We present our containment results in Section 3. A method for computing maximally-contained sets of rewritings is detailed in Section 4. Section 5 concludes. An extended version of this paper, containing proofs and additional examples, is available on the Web [5].

2 Aggregate Queries

We introduce aggregate queries and review their basic properties. Our aggregate queries can contain disjunction, negation and comparisons.

2.1 Syntax of Queries

Let P be a fixed relational vocabulary and let C be a set of constants. We assume that there is a linear ordering defined over C, which is either dense (like the rational numbers) or discrete (like the integers). We will usually define a *database* $\mathcal{D}$ as a *set* of ground atoms of the form $p(c_1, \ldots, c_k)$, where $p \in P$ is a predicate of arity k and $c_1, \ldots, c_k \in C$. At times we will consider databases that are *bags* of ground atoms, but then we will state this explicitly.

A *condition* $\varphi(\bar{z}, \bar{x})$, where $\bar{z}$, $\bar{x}$ are tuples of variables, is a conjunction of positive and negated atomic formulas over P, $<$ and $\leq$, with constants from C. The comparisons are interpreted w.r.t. the ordering over C. The variables in $\bar{x}$ are exactly the *free* variables in $\varphi(\bar{z}, \bar{x})$ and the variables in $\bar{z}$ are *bound* by existential quantifiers. A *query* has the form

$$q(\bar{x}) \leftarrow \varphi_1(\bar{z}_1, \bar{x}) \vee \cdots \vee \varphi_n(\bar{z}_n, \bar{x})$$

where $\varphi_i(\bar{z}_i, \bar{x})$ is a condition, for all i. We call $q(\bar{x})$ the *head* of the query and $\varphi_1(\bar{z}_1, \bar{x}) \vee \cdots \vee \varphi_n(\bar{z}_n, \bar{x})$ the *body* of the query. Each condition $\varphi_i(\bar{z}_i, \bar{x})$ is a

disjunct of the query. We sometimes use the notation $q(\bar{x})$ or q as a short-hand for denoting the query above. We assume that all queries are *safe* [22]. Note that all variables not appearing in $\bar{x}$ are implicitly existentially quantified.

We assume that there is a fixed set of constants C_{agg}. If S is a domain, we denote by $\mathcal{M}(S)$ the set of finite bags over S. A *k-ary aggregation function* is a function $\alpha\colon \mathcal{M}(C^k) \to C_{\text{agg}}$ that maps bags of k-tuples of values in C to values in C_{agg}. In this paper we consider a wide class of aggregation functions, with the property of being *expandable*. Such aggregation functions include the unary aggregation functions *max*, *sum*, *avg*, *median*, *stdev* and *cntd* which map a bag of real numbers to the maximum, sum, average, median, standard deviation or number of distinct elements.[1] We also consider the expandable aggregation function *count* which maps a bag of tuples to its cardinality.

An *aggregate query* has the form $q(\bar{x}, \alpha(\bar{y})) \leftarrow \varphi_1(\bar{z}_1, \bar{x}, \bar{y}) \vee \cdots \vee \varphi_n(\bar{z}_n, \bar{x}, \bar{y})$, where α is an aggregation function and $\bar{x}$ and $\bar{y}$ are free variables appearing in φ_i, for all i. We call $\bar{x}$ the *grouping variables* and $\bar{y}$ the *aggregation variables*. We will also refer to the query q as an *α-query* since the aggregation function α appears in q's head.

We distinguish the class of conjunctive queries. A query q is conjunctive if its body contains a single disjunct $\varphi(\bar{z}, \bar{x})$ that is a conjunction of *positive* atomic formulas over P and C. Note that conjunctive queries do not contain comparisons.

Example 2.1. Suppose that our database contains the relations `study(student, course, grade)` and `teach(prof, course)`. The conjunctive query q finds the average grade of the students in Prof. Lau's courses:

$$q(c, avg(g)) \leftarrow \texttt{study}(s, c, g) \wedge \texttt{teach}(\texttt{Lau}, c)\,.$$

2.2 Semantics of Queries

We define in which way an aggregate query q, for a database $\mathcal{D}$, gives rise to a *set* of tuples $q^{\mathcal{D}}$. Aggregate queries are evaluated in two phases. In the first phase, the query retrieves a bag of tuples from the database. The tuples are then grouped into equivalence classes, and an aggregation function is applied to each equivalence class.

An *assignment* γ for a condition $\varphi(\bar{z}, \bar{x}, \bar{y})$ is a mapping of the variables in $\bar{z}, \bar{x}, \bar{y}$ to constants in C. *Satisfaction* of a condition by an assignment w.r.t. a database is defined in the usual way.

Consider an aggregate query $q(\bar{x}, \alpha(\bar{y})) \leftarrow \varphi_1(\bar{z}_1, \bar{x}, \bar{y}) \vee \cdots \vee \varphi_n(\bar{z}_n, \bar{x}, \bar{y})$. Let Γ_i be the set of satisfying assignments of $\varphi_i(\bar{z}_i, \bar{x}, \bar{y})$ with respect to $\mathcal{D}$. For a tuple of values $\bar{d}$ from C, let $\Gamma_{i,\bar{d}}$ be the set of assignments in Γ_i that map $\bar{x}$ to $\bar{d}$, i.e.,

$$\Gamma_{i,\bar{d}} := \{\gamma \in \Gamma_i \mid \gamma(\bar{x}) = \bar{d}\}.$$

[1] Results for *min* are analogous to results for *max* and, thus, are not presented.

In the sets $\Gamma_{i,\bar{d}}$, we collect those satisfying assignments of φ_i that agree on $\bar{d}$.

For each set $\Gamma_{i,\bar{d}}$, we define $\Gamma_{i,\bar{d}}(\bar{y})$ to be the bag of tuples of constants for $\bar{y}$ derived by applying the assignments in $\Gamma_{i,\bar{d}}$ to $\bar{y}$. Formally,

$$\Gamma_{i,\bar{d}}(\bar{y}) := \{\!\{\gamma(\bar{y}) \mid \gamma \in \Gamma_{i,\bar{d}}\}\!\}$$

is the bag of tuples obtainable by restricting assignments in $\Gamma_{i,\bar{d}}$ to $\bar{y}$.

Now we define the result of evaluating q over $\mathcal{D}$, denoted $q^{\mathcal{D}}$ as

$$\left\{(\bar{d}, \alpha(\Gamma_{1,\bar{d}}(\bar{y}) \uplus \cdots \uplus \Gamma_{n,\bar{d}}(\bar{y}))) \mid \bar{d} = \gamma(\bar{x}) \text{ for some } i \text{ and } \gamma \in \Gamma_i\right\} \qquad (1)$$

where $\uplus$ is the bag union operator. Note that we have computed the function α on the bag corresponding to each mapping of $\bar{x}$ to constants. Observe that if a mapping satisfies more than one disjunct, then it contributes more than one value to the bag to which we apply α. Thus, for example, the queries $q(x, sum(y)) \leftarrow a(x,y) \vee a(x,y)$ and $q'(x, sum(y)) \leftarrow a(x,y)$ are not equivalent since q will always return an aggregate value twice that of q'.

For the special case of conjunctive α-queries, we also consider evaluating queries over databases that are *bags* of ground atoms. Queries evaluated over a database defined as a bag are said to be evaluated under *bag-semantics*. Otherwise, the evaluation is under *set-semantics*. When evaluating a conjunctive α-query under bag-semantics, we proceed as above (i.e., as under set semantics), except that there is only one $\Gamma_{\bar{d}}$ for each $\bar{d}$ (since there is only one disjunct) and $\Gamma_{\bar{d}}$ can be a bag of assignments. Consider the conjunctive α-query

$$q(\bar{x}, \alpha(\bar{y})) \leftarrow a_1(\bar{w}_1) \wedge \cdots \wedge a_k(\bar{w}_k),$$

where $\bar{w}_i$ are variables from $\bar{x}$, $\bar{y}$ or are bound variables. Let γ be an assignment that satisfies the body of q and maps $\bar{x}$ to $\bar{d}$. Suppose that $a_i(\gamma(\bar{w}_i))$ appears in $\mathcal{D}$ a total of n_i times. Then, γ will be in $\Gamma_{\bar{d}}$ a total of $n_1 \times \cdots \times n_k$ times. The rest of the computation proceeds as in Equation (1). Note that the output of an aggregate query is always a set, even if it is evaluated over a bag database.

2.3 Containment and Equivalence

Containment and equivalence of aggregate queries are defined in the natural way. An aggregate query q is *contained* in an aggregate query q' if over every database the set of results returned by q is a subset of the results returned by q'. Formally, q is contained in q', denoted $q \subseteq q'$, if $q^{\mathcal{D}} \subseteq {q'}^{\mathcal{D}}$ for all databases $\mathcal{D}$. Two queries q and q' are *equivalent*, denoted $q \equiv q'$, if over every database they return the same sets of results. Obviously, two queries are equivalent if and only if they contain each other.

For conjunctive queries q and q', we say that q is *bag-contained* in q', denoted $q \subseteq_{\mathrm{b}} q'$, if over every bag database $\mathcal{D}$, we have $q^{\mathcal{D}} \subseteq {q'}^{\mathcal{D}}$. We say that q and q' are *bag-equivalent*, denoted $q \equiv_{\mathrm{b}} q'$, if $q \subseteq_{\mathrm{b}} q'$ and $q' \subseteq_{\mathrm{b}} q$.

3 Containment of Aggregate Queries

This section contains characterizations of containment among queries with a wide class of aggregation functions. In Section 4 we show how our containment results can be used to create rewritings of queries using views. Our criteria involve reducing containment of aggregate queries to equivalence of aggregate queries.

3.1 Queries with Expandable Aggregation Functions

We present the class of *expandable aggregation functions*. We show that they are common and reduce containment of queries with expandable aggregation functions to equivalences of queries.

Let B be a bag of constants and c be a constant. We denote by $|B|_c$ the multiplicity of c in B. Given a non-negative integer n, we define the *n-expansion* of B. Intuitively, the n-expansion of B, denoted $B \otimes n$, is the bag that contains the same constants as those in B, but each constant has a multiplicity n times larger than in B. Formally, for all constants c, $|B \otimes n|_c = |B|_c \times n$.

Aggregation functions can be characterized by their behavior on expanded bags. We say that an aggregation function α is *expandable* if for all bags B and B' and for all positive integers n we have

$$\alpha(B \otimes n) = \alpha(B' \otimes n) \quad \Longleftrightarrow \quad \alpha(B) = \alpha(B')\,.$$

Proposition 3.1 (Expandable Functions). *The aggregation functions max, cntd, count, sum, avg, stdev and median are expandable.*

However, not all aggregation functions are expandable.

Example 3.2. The function *prod* (product) is not expandable. As an example, consider the bags $B = \{\!\{-2\}\!\}$ and $B' = \{\!\{2, 1\}\!\}$. Clearly, $prod(B) \neq prod(B')$. However, $prod(B \otimes 2) = prod(B' \otimes 2) = 4$. Hence, *prod* is not expandable. Observe that *prod* is expandable when the domain of the bags contains only non-negative numbers. This example also shows that the function *parity* is not expandable.

Given the aggregate queries

$$q(\bar{x}, \alpha(\bar{y})) \leftarrow \varphi_1(\bar{z}_1, \bar{x}, \bar{y}) \vee \cdots \vee \varphi_n(\bar{z}_n, \bar{x}, \bar{y})$$
$$q'(\bar{x}, \alpha(\bar{y}')) \leftarrow \varphi'_1(\bar{z}'_1, \bar{x}, \bar{y}') \vee \cdots \vee \varphi'_m(\bar{z}'_m, \bar{x}, \bar{y}')$$

we say that a query p is a *product* of q and q' if p is defined as

$$p(\bar{x}, \alpha(\bar{y})) \leftarrow \bigvee_{1 \leq i \leq n,\ 1 \leq j \leq m} \varphi_i(\bar{z}_i, \bar{x}, \bar{y}) \wedge \varphi'_j(\theta(\bar{z}'_j), \bar{x}, \theta(\bar{y}'))\,,$$

where θ is a substitution that maps the variables in $\bar{y}'$ and $\bar{z}'_j$ to distinct unused variables. A product of q and q' is essentially a join on the grouping variables

of q and q'. Note that two products of q and q' are only distinguished by the substitution θ and thus, are isomorphic. Slightly abusing notation, we will write $q \otimes q'$ to refer to an arbitrary product of q and q'. This is justified because the specific variables occurring in a query are never important for our arguments.

Containment of aggregate queries can be reduced to equivalence of appropriate aggregate queries, if the aggregation function is expandable.

Theorem 3.3 (Containment). *Consider the aggregate queries $q(\bar{x}, \alpha(\bar{y}))$ and $q'(\bar{x}, \alpha(\bar{y}'))$. Suppose that α is an expandable function. Then for any database $\mathcal{D}$*

$$q^{\mathcal{D}} \subseteq {q'}^{\mathcal{D}} \iff (q \otimes q)^{\mathcal{D}} = (q' \otimes q)^{\mathcal{D}}.$$

Proof. (Sketch) Consider a database $\mathcal{D}$ and a tuple $\bar{d}$. Suppose that q computes the bag B of values for $\bar{d}$ and q' computes the bag B' of values for $\bar{d}$. It is not difficult to show that in this case, $q \otimes q$ will compute the bag $B \otimes |B|$ for $\bar{d}$ and $q' \otimes q$ will compute the bag $B' \otimes |B|$ for $\bar{d}$. Since α is an expandable aggregation function, $\alpha(B) = \alpha(B')$ if and only if $\alpha(B \otimes |B|) = \alpha(B' \otimes |B|)$, for $|B| > 0$.

"$\Leftarrow$" Suppose that $q \otimes q \equiv q' \otimes q$. If q returns an aggregate value for $\bar{d}$, then $|B| > 0$. Therefore, $\alpha(B \otimes |B|) = \alpha(B' \otimes |B|)$ implies that $\alpha(B) = \alpha(B')$, i.e., q and q' return the same aggregate value for $\bar{d}$.

"$\Rightarrow$" Suppose that $q \subseteq q'$. If q does not return an aggregate value for $\bar{d}$, then both $q \otimes q$ and $q' \otimes q$ will not return an aggregate value. Otherwise, q returns an aggregate value for $\bar{d}$, and q' returns the same aggregate value. Therefore, from $\alpha(B) = \alpha(B')$, we conclude that $\alpha(B \otimes |B|) = \alpha(B' \otimes |B|)$, i.e., $q \otimes q$ and $q' \otimes q$ return the same value. □

Corollary 3.4 (Decidability). *Checking if q is contained in q' is decidable if*

- *q and q' are count, sum or max-queries* or
- *q and q' are conjunctive avg or cntd-queries without constants.*

Proof. This follows from Theorem 3.3 and the decidability of equivalence for the classes above (see [18,6,9,4]). □

Theorem 3.5 (Bag-Containment). *Let α be an expandable function and q and q' be conjunctive α queries. Then, $q \subseteq_{\mathrm{b}} q'$ if and only if $q \otimes q \equiv_{\mathrm{b}} q' \otimes q$.*

Proof. The proof is analogous to the proof of Theorem 3.3. □

If α is not an expandable function, the reduction may not be correct.

Example 3.6. Consider the queries

$$q(\mathit{parity}(y)) \leftarrow p(y) \vee p(y)$$
$$q'(\mathit{parity}(y)) \leftarrow p(y)\,.$$

Let k be the number of values d such that $\mathcal{D}$ contains $p(d)$. Then, q always returns the value "even", since it evaluates the parity of a bag with $2k$ values.

The query q' returns the parity of k since it evaluates the parity of a bag with k values. Thus, $q \not\subseteq q'$, since if k is odd, the queries q and q' will return different values. Consider the queries $q \otimes q$ and $q' \otimes q$. Clearly, $q \otimes q$ will always return the value "even", since the aggregation function *parity* is evaluated over a bag with cardinality $4k^2$. Similarly, $q' \otimes q$ will always return the value "even", since it computes the parity of a bag of size $2k^2$. Therefore, the queries $q \otimes q$ and $q' \otimes q$ are equivalent, even though $q \not\subseteq q'$.

3.2 Integrity Constraints

Even in the presence of arbitrary integrity constraints it is possible to reduce containment of aggregate queries to equivalence. We write $q \subseteq_I q'$ if for all databases $\mathcal{D}$ satisfying the integrity constraints I, it holds that $q^{\mathcal{D}} \subseteq {q'}^{\mathcal{D}}$. Similarly, we write $q \equiv_I q'$ if $q \subseteq_I q'$ and $q' \subseteq_I q$.

Corollary 3.7 (Reduction w.r.t. Integrity Constraints). *Consider the α-queries q and q' and the integrity constraints I. Suppose that α is an expandable aggregation function. Then $q \subseteq_I q'$ if and only if $(q \otimes q) \equiv_I (q' \otimes q)$.*

Proof. Theorem 3.3 allows us to reduce containment with respect to a set of integrity constraints I to equivalence with respect to I. □

It is possible to extend previous characterizations of equivalence of queries from [18,6,4] to equivalence with respect to a set of functional dependencies F. For example, it can be shown that if q and q' are conjunctive *count*-queries, then $q \equiv_F q'$ if and only if the chases of q and q' with respect to F are isomorphic.

Corollary 3.8 (Decidability). *Given a set of functional dependencies F, the problem of checking whether $q \subseteq_F q'$ is decidable if q and q' are count, sum or max-queries.*

Example 3.9. Consider the queries q and q' which find the number of visitor advisors of a student and the number of advisors of a student, respectively:

$$q(s, count) \leftarrow \texttt{advise}(p, s) \wedge \texttt{visitor}(p)$$
$$q'(s, count) \leftarrow \texttt{advise}(p, s)\,.$$

In general, $q \not\subseteq q'$. Suppose that every student can have at most one advisor, i.e., the first column of `advise` is functionally dependent on the second. Let F be the set of functional dependencies that contains exactly this dependency. We will show that that $q \subseteq_F q'$ even though applying the chase to q and q' has no effect. To prove that $q \subseteq_F q'$, we consider the products $q \otimes q$ and $q' \otimes q$, i.e.,

$$(q \otimes q)(s, count) \leftarrow \texttt{advise}(p, s) \wedge \texttt{visitor}(p) \wedge \texttt{advise}(p', s) \wedge \texttt{visitor}(p')$$
$$(q' \otimes q)(s, count) \leftarrow \texttt{advise}(p, s) \wedge \texttt{advise}(p', s) \wedge \texttt{visitor}(p').$$

By Corollary 3.7, $q \subseteq_F q'$ if and only if $(q \otimes q) \equiv_F (q' \otimes q)$. Indeed, applying the chase to $(q \otimes q)$ and $(q' \otimes q)$ results in isomorphic, and thus, equivalent queries.

3.3 Containment of Conjunctive Count-Queries

We show how the results in the previous section give rise to simpler characterizations for conjunctive *count*-queries.

A *bound atom* is an atom with at least one argument that is a bound variable. Otherwise, the atom is *free*. We sometimes write a conjunctive query as

$$q(\bar{x}, \alpha(\bar{y})) \leftarrow \varphi_f(\bar{x}, \bar{y}) \wedge \varphi_b(\bar{z}, \bar{x}, \bar{y})\,.$$

where φ_f is a conjunction of free atoms and φ_b is a conjunction of bound atoms. Note that for the special case of *count*-queries, $\bar{y}$ is an empty tuple of variables.

Theorem 3.10 (Conjunctive Count-Queries). *For the conjunctive queries*

$$q(\bar{x}, count) \leftarrow \varphi_f(\bar{x}) \wedge \varphi_b(\bar{z}, \bar{x})$$
$$q'(\bar{x}, count) \leftarrow \varphi'_f(\bar{x}) \wedge \varphi'_b(\bar{z}', \bar{x}).$$

it holds that $q \subseteq_{\mathrm{b}} q'$ *if and only if* $q \equiv_{\mathrm{b}} q'$*, and* $q \subseteq q'$ *if and only if both the following conditions hold*

- *the set of atoms in* $\varphi'_f(\bar{x})$ *is contained in the set of atoms in* $\varphi_f(\bar{x})$ and
- $\varphi'_b(\bar{z}', \bar{x})$ *is isomorphic to* $\varphi_b(\bar{z}, \bar{x})$.[2]

When considering the general problem of aggregate query containment, we assumed that both queries had the same grouping variables. This was without loss of generality since equalities between grouping variables and between grouping variables and constants could be expressed using comparisons in the bodies of the queries. In this section we discuss only conjunctive *count*-queries and such queries do not have comparisons in their bodies. Therefore, we characterize containment of queries with different grouping terms. (Both queries have the same number of grouping terms, however.) A query has the form

$$q(\bar{s}, count) \leftarrow \varphi_f(\bar{x}) \wedge \varphi_b(\bar{z}, \bar{x})\,,$$

where $\bar{s}$ is a tuple of terms and $\bar{x}$ are exactly all the variables in $\bar{s}$.

Let $q(\bar{s}, count) \leftarrow \varphi_f(\bar{x}) \wedge \varphi_b(\bar{z}, \bar{x})$ and $q'(\bar{s}', count) \leftarrow \varphi'_f(\bar{x}') \wedge \varphi'_b(\bar{z}', \bar{x}')$ be queries. A mapping θ from the variables in q' to the terms in q is a *containment mapping* if the following conditions hold

1. $\theta(\bar{s}') = \bar{s}$, i.e., θ maps the head of q' to the head of q;
2. θ is injective on the bound variables in q';
3. the set of atoms in $\theta(\varphi'_b(\bar{z}', \bar{x}'))$ is the same as the set of atoms in $\varphi_b(\bar{z}, \bar{x})$, i.e., θ maps bound atoms in q' to bound atoms in q;
4. $\theta(\varphi'_f(\bar{x}'))$ is contained in $\varphi_f(\bar{x}) \wedge \varphi_b(\bar{z}, \bar{x})$, i.e., the images of the free atoms in q' appear in q.

The following corollary follows from Theorem 3.10 and will be used in Section 4 when discussing rewritings of queries.

Corollary 3.11 (Conjunctive Count-Queries (2)). *Let* q *and* q' *be conjunctive count-queries, possibly with different terms in their heads. Then* $q \subseteq q'$ *if and only if there is a containment mapping from* q' *to* q.

[2] Note that our isomorphisms ignore duplicate occurrences of atoms in a conjunction.

3.4 Containment of Conjunctive Sum-Queries

Theorem 3.3 characterizes containment of *sum*-queries, since *sum* is an expandable function. We present a simpler condition for the case where the *sum*-queries are conjunctive. A *sum*-query of the form $q(\bar{x}, sum(y)) \leftarrow \varphi(\bar{z}, \bar{x}, y)$ is associated with a *count*-query q_c defined as $q_c(\bar{x}, y, count) \leftarrow \varphi(\bar{z}, \bar{x}, y)$.

Theorem 3.12 (Conjunctive Sum-Queries). *Let q and q' be conjunctive sum-queries. Then $q \subseteq q'$ if and only if $q_c \subseteq q'_c$, and $q \subseteq_b q'$ if and only if $q_c \subseteq_b q'_c$.*

Proof. (Sketch) This follows from the the fact that $q \equiv q'$ if and only if $q_c \equiv q'_c$ and $q \equiv_b q'$ if and only if $q_c \equiv_b q'_c$, shown in [18].

4 Rewriting Conjunctive Count-Queries

The problem of answering queries using views has been studied for different classes of queries [1,8,11,15]. See [14] for a comprehensive survey of related work. The ability to answer queries using views is useful for query optimization, for maintaining independence of physical and logical database schemas and in the context of data integration. We focus here on finding *maximally-contained sets of rewritings*, a problem that arises when integrating data from varied sources.

We present a sound and complete method to compute maximally-contained sets of rewritings for conjunctive *count*-queries. Our results are also immediately applicable to non-aggregate queries evaluated under bag-set semantics. A similar method can be used to compute maximally-contained sets of rewritings for *sum*-queries. We do not show how to find such sets because of space limitations.

Rewriting aggregate queries has been considered in [6,21,9]. However, to the best of our knowledge, the problem of finding maximally-contained sets of rewritings of aggregate queries has not been dealt with at all. In fact, in [14], this problem is mentioned as an interesting open problem.

4.1 Definitions

Let V be a set of queries, called *views*. The queries in V are defined over the relational vocabulary P and use the constants in C. We say that a predicate v is *defined* in V if there is a view in V which has the predicate v as its head. We also call v a *view predicate*.

We now allow queries to be defined over the relational vocabulary V. Let r be such a query. Any view predicate v in r's body is considered an IDB predicate. Thus, the extension of v with respect to a database $\mathcal{D}$ contains exactly the tuples derived by evaluating v on $\mathcal{D}$, i.e., $v^{\mathcal{D}}$. We denote the result of evaluating r on $\mathcal{D}$, using the view definitions in V, as $r^{\mathcal{D}_V}$.

A query that is defined over P is said to be defined over the *base predicates*. A query that is defined over V is said to be defined over the *view predicates*. Let q be a query defined over P and let r, r' be queries defined over V. We say that

r is *contained modulo* V in q, denoted $r \subseteq_V q$, if for all database $\mathcal{D}$, $r^{\mathcal{D}_V} \subseteq q^{\mathcal{D}}$. Similarly, $r \subseteq_V r'$ if $r^{\mathcal{D}_V} \subseteq r'^{\mathcal{D}_V}$. We define *equivalence modulo* V, denoted $\equiv_V$, in a similar fashion. If $r \subseteq_V q$ we say that r is a *rewriting* of q using V. Note that our notion of a rewriting differs from the standard definition in that r must be contained in q and need not be equivalent to q.

Let q be a query, let $\mathcal{R}$ be a set of rewritings of q using views V and let R be a subset of $\mathcal{R}$. We say that R is a *maximally-contained set of rewritings* of q with respect to $\mathcal{R}$ if for every $r \in \mathcal{R}$ there is an $r' \in R$ such that $r \subseteq_V r'$.

4.2 Class of Views and Class of Rewritings

We consider the problem of finding rewritings of a conjunctive *count*-query given a set of views. A count-query is sensitive to multiplicities, and count-views are the only type of aggregate views that do not lose multiplicities.[3] Thus, we only use count-views when rewriting count-queries. Even if we restrict ourselves to using *count* views, there may be an infinite number of aggregate terms that can be usable in the head of a rewriting of a count-query. However, we will restrict ourselves to a specific class of queries that is both natural and expressive.

Finding a rewriting of a *count*-query q using a set of views V involves two steps:

- **Generate:** Create a query r over the view predicates V.
- **Test:** Determine if r is contained in q.

Observe that it is might not possible to check directly if $r \subseteq_V q$ since r may not even be a *count*-query and r uses the view predicates and not the base predicates. Therefore, we have no characterizations of containment that can allow us to determine directly if $r \subseteq_V q$. In order to overcome this problem, we will restrict ourselves to choosing r from the class of queries $\mathit{Unf}(V)$, defined below. We will show that if $r \in \mathit{Unf}(V)$, then it is possible to find a *count*-query r' over the base predicates that is equivalent to r. Thus, we will be able to check whether $r \subseteq_V q$ by checking whether $r' \subseteq q$, using the characterizations in Theorem 3.10 and Corollary 3.11.

Let $V = \{v_i(\bar{s}_i, \mathit{count}) \leftarrow \varphi_i(\bar{z}_i, \bar{x}_i)\}_{i \in \mathcal{I}}$ be a set of count-views. Let r be a query defined over V of the form

$$r(\bar{s}, \mathit{sum}(\prod_{j=1}^{n} w_j)) \leftarrow v_1(\theta_1(\bar{s}_1), w_1) \wedge \cdots \wedge v_n(\theta_n(\bar{s}_n), w_n)\,. \tag{2}$$

We call θ_i the *instantiation* of v_i. Note that w_i is the variable that replaces the aggregation term *count* in the head of v_i. A view in V can appear any number of times in r. The *unfolding* of r, denoted r^u, is derived by

1. replacing each view atom $v_i(\theta_i(\bar{s}_i), w_i)$ in the body of r by $\varphi_i(\theta'_i(\bar{z}_i), \theta_i(\bar{)}x_i)$ where θ'_i is an injective mapping of $\bar{z}_i$ to unused variables *and*

[3] Although sum-views are sensitive to multiplicities, they lose these values. For example, sum-queries ignore occurrences of zero values.

2. replacing the aggregation function in the head of r with the function *count*.

Thus, r^u has the form

$$r^u(\bar{s}, count) \leftarrow \varphi_1(\theta'_1(\bar{z}_1), \theta_1(\bar{x}_1)) \wedge \cdots \wedge \varphi_n(\theta'_n(\bar{z}_n), \theta_n(\bar{x}_n))$$

We say that r has the *unfolding property* if for all databases $\mathcal{D}$ it holds that $(r^u)^{\mathcal{D}} \equiv_V r^{\mathcal{D}_V}$. Intuitively, this property states that evaluating r over $\mathcal{D}$ by taking V into consideration will yield the same result as evaluating the unfolding of r on the relations in the database. This is a natural property that is required when rewriting non-aggregate queries. In [6] it has been shown that for a query to have the unfolding property it must be of the form presented in Equation 2. We denote as $Unf(V)$ the class of queries with the form presented in Equation 2 such that $v_i \in V$, for all i.

Now, given a *count*-query q and a query r in $Unf(V)$, it is possible to determine if $r \subseteq_V q$, since it is possible to find a *count*-query over the base predicates that is equivalent to r. This *count*-query is simply the unfolding of r, i.e., r^u.

Example 4.1. Consider the query q and the views $V = \{v_1, v_2, v_3\}$ defined below:

$$\begin{aligned} q(p, g, count) &\leftarrow \texttt{study}(s, c, g) \wedge \texttt{teach}(p, c) \\ v_1(p, c, count) &\leftarrow \texttt{teach}(p, c) \wedge \texttt{full\_time}(p) \\ v_2(s, c, g, count) &\leftarrow \texttt{study}(s, c, g) \\ v_3(p, g, count) &\leftarrow \texttt{teach}(p, c) \wedge \texttt{visitor}(p) \wedge \texttt{study}(s, c, g)\,. \end{aligned}$$

Query q computes the number of students with a given grade, for a given professor. View v_1 returns information about the courses that full-time professors teach. (The *count* value will always be one in this view since all the variables in the view are free.) View v_2 returns students with courses that they studied and their respective grades. The view v_3 computes the number of students with a given grade, for a given visiting professor.

Consider the query r_1 defined over the view predicates and its unfolding r_1^u

$$\begin{aligned} r_1(p, g, sum(w_1 \times w_2)) &\leftarrow v_1(p, c, w_1) \wedge v_2(s, c, g, w_2) \\ r_1^u(p, g, count) &\leftarrow \texttt{teach}(p, c) \wedge \texttt{full\_time}(p) \wedge \texttt{study}(s, c, g)\,. \end{aligned}$$

By applying Theorem 3.10 we derive that $r_1^u \subseteq q$ and therefore, $r_1 \subseteq_V q$.

Consider also the query r_2 defined over the views and its unfolding r_2^u

$$\begin{aligned} r_2(p, g, w_3) &\leftarrow v_3(p, g, w_3) \\ r_2^u(p, g, count) &\leftarrow \texttt{teach}(p, c) \wedge \texttt{visitor}(p) \wedge \texttt{study}(s, c, g)\,. \end{aligned}$$

Once again, using Theorem 3.10, we can verify that $r_2^u \subseteq q$ and thus, $r_2 \subseteq_V q$.

In this manner, we found two rewritings of q using the views. It is possible to show that $\{r_1, r_2\}$ is a maximally-contained set of rewritings of q w.r.t. $Unf(V)$.

However, the following example will show that if only queries from $Unf(V)$ are considered, it may not be possible to find a finite maximally-contained set of rewritings for a given query.

Example 4.2. Consider the following query and views

$$q(x, count) \leftarrow a(x, z) \wedge c(x, u)$$
$$v_1(x, y, count) \leftarrow a(x, z) \wedge b(x, y)$$
$$v_2(x, count) \leftarrow c(x, u).$$

The query r_1 has the unfolding property (i.e., $r_1 \in \mathit{Unf}(V)$), but is not a rewriting of q:

$$r_1(x, sum(w_1 \times w_2)) \leftarrow v_1(x, y, w_1) \wedge v_2(x, w_2).$$

Intuitively, r_1 is not a rewriting of q since the atom $b(x, y)$ is bound in r_1^u and does not appear in q. However, for any constant d, the query $r_d \in \mathit{Unf}(V)$

$$r_d(x, sum(w_1 \times w_2)) \leftarrow v_1(x, d, w_1) \wedge v_2(x, w_2)$$

is a rewriting of q. Therefore, if we consider only queries with the unfolding property, a maximally-contained set of rewritings w.r.t. $\mathit{Unf}(V)$ will contain an infinite number of queries (one for each constant).

In order to ensure the existence of a finite maximally-contained set of rewritings, we will choose our rewritings from a class that is more expressive than $\mathit{Unf}(V)$, called $\mathit{PUnf}(V)$. Intuitively, $\mathit{PUnf}(V)$ contains projections of queries in $\mathit{Unf}(V)$. Formally, let $p(\bar{s}, sum(\prod_{j=1}^{n} w_j))$ be a query in $\mathit{Unf}(V)$. We say that the query r

$$r(\bar{s}', w) \leftarrow p(\bar{s}, w) \tag{3}$$

is a *k-projection* of p if

1. $\bar{s}'$ contains exactly the first k terms in $\bar{s}$, i.e., $\bar{s} = (\bar{s}', \bar{t})$ for some tuple of terms $\bar{t}$ *and*
2. every variable in $\bar{s}$ and not in $\bar{s}'$ appears only in free atoms in p^u.

If r is a k-projection of p, we say that p is the *k-projectee* of r. When evaluating a query such as r, the projection is taken as a *set-projection*, i.e., duplicates are removed. Property 2 ensures that each tuple in a k-projection is associated with a single aggregation value (see [5]).

Given a set of views V, the class $\mathit{PUnf}(V)$ contains all k-projections of queries in $\mathit{Unf}(V)$, for all k. The reader will note that any query in $\mathit{Unf}(V)$ can be expressed in $\mathit{PUnf}(V)$ by simply taking $\bar{s}'$ as $\bar{s}$.

Example 4.3. Recall the query q and views v_1 and v_2 from Example 4.2. Let $p \in \mathit{Unf}(V)$ and $r \in \mathit{PUnf}(V)$ be the queries

$$p(x, y, sum(w_1 \times w_2)) \leftarrow v_1(x, y, w_1) \wedge v_2(x, w_2)$$
$$r(x, w) \leftarrow p(x, y, w)\,.$$

It is not difficult to see that $r \subseteq_V q$ and for all d, it holds that $r_d \subseteq_V r$. (We show formally how to check for containment below.) In fact, one can show that the set $\{r\}$ is a maximally contained rewriting of q w.r.t. $\mathit{PUnf}(V)$.

Given a query q and a set of views V, we will be interested in finding a maximally-contained set of rewritings of q w.r.t. $PUnf(V)$. We extend the definition of a containment mapping (page 118) to allow us to check if a query in $PUnf(V)$ is contained in a *count*-query. Consider the queries $p(\bar{s}, count)$ and $p'(\bar{s}', count)$. Suppose that the tuples $\bar{s}$ and $\bar{s}'$ are at least of size k. We say that a mapping θ from the variables in p' to the terms in p is a k-containment mapping if it fulfills requirements 2 through 4 of containment mappings and also maps the first k terms in $\bar{s}'$ to the first k terms in $\bar{s}$. Our k-containment mappings can be used to check if a query in $PUnf(V)$ is contained in a *count*-query.

Lemma 4.4 (Containment of $PUnf(V)$ Queries). *Let q be a conjunctive count-query, V be a set of views and r be a query in $PUnf(V)$. Suppose that p is the k-projectee of r. Then r is a rewriting of q, i.e., $r \subseteq_V q$, if and only if there exists a k-containment mapping from q to p^u.*

Lemma 4.4 implies an algorithm for checking if a query in $PUnf(V)$ is a rewriting. Hence, when searching for a maximally-contained set of rewritings, we simply have to generate queries in $PUnf(V)$ and check if they are rewritings. For this procedure to terminate, we show that it is sufficient to generate a finite number of queries. In fact, it is possible to bound the number of views being used and the number of constants needed. This is similar to the conjunctive case. Using Lemma 4.4 and 4.5, we can show that it is possible to find a maximally-contained set of rewritings of a given query w.r.t. the class $PUnf(V)$.

Lemma 4.5 (Size of Rewriting). *Let q be a query and let $r \in PUnf(V)$ be a rewriting of q. Suppose that the body of q contains n atoms. Then there is a rewriting r' of q such that $r \subseteq_V r'$ and the projectee of r'*

- *contains at most n views* and
- *does not contain any constant not appearing in either V or q.*

Theorem 4.6 (Maximally-Contained Set of Rewritings). *Let q be a query and V be a set of views. It is possible to find a maximally-contained set of rewritings of q w.r.t. $PUnf(V)$ of finite size.*

Theorem 4.6 suggests an exponential algorithm for finding a maximally-contained set of rewritings. It can be improved using standard rewriting algorithm techniques, such as those appearing in [7,19]. Such algorithms have been shown to run well in practice [19].

5 Conclusion

The class of expandable aggregation functions has been introduced and we have shown how to reduce containment of queries with expandable aggregation functions to equivalence of queries. This reduction holds even in the presence of arbitrary integrity constraints. Simpler characterizations for containment of

count and *sum*-queries have been presented. The problem of finding maximally-contained sets of rewritings has been solved for conjunctive *count*-queries.

Containment of non-aggregate conjunctive queries under bag and bag-set semantics has been studied in [3,12]. In these papers, a query q is contained in a query q' if for all databases $\mathcal{D}$, it holds that $q^{\mathcal{D}}$ is a *sub-bag* of $q'^{\mathcal{D}}$. For *count*-queries, this is similar to defining that $q(\bar{x}, count)$ is contained in $q'(\bar{x}, count)$ if for all databases $\mathcal{D}$ and tuples $\bar{d}$, if $(\bar{d}, c) \in q^{\mathcal{D}}$, then there is a $c' \geq c$ such that $(\bar{d}, c') \in q'^{\mathcal{D}}$. It has been shown independently by [3] and by [12] that containment of conjunctive queries under bag semantics is decidable if the queries do not contain any repeated predicates. Undecidability of containment of unions of conjunctive queries under bag semantics has been proven in [12].

Our definition of containment differs from the definition above and we feel that it is more natural in the context of finding maximally-contained sets of rewritings. For our definition of containment, we know that if $q \subseteq q'$ then every result of applying q on a database would also be achieved by q'. The corresponding definition of containment in [3,12] would only allow us to derive lower bounds on the *count* values of q'. Moreover, our definition carries over easily to other types of aggregation functions.

We leave for future research the problem of deciding containment of queries with a `HAVING` clause, as well as queries containing aggregation functions that are not expandable, such as *prod* and *parity*. Finding tight upper and lower bounds for the complexity of containment is another important open problem.

References

[1] S. Abiteboul and O. Duschka. Complexity of answering queries using materialized views. In J. Paredaens, editor, *Proc. 17th Symposium on Principles of Database Systems*, pages 254–263, Seattle (Washington, USA), June 1998. ACM Press.

[2] S. Chaudhuri, S. Krishnamurthy, S. Potamianos, and K. Shim. Optimizing queries with materialized views. In P.S.Yu and A. Chen, editors, *Proc. 11th International Conference on Data Engineering*, Taipei, Mar. 1995. IEEE Computer Society.

[3] S. Chaudhuri and M. Vardi. Optimization of real conjunctive queries. In *Proc. 12th Symposium on Principles of Database Systems*, Washington (D.C., USA), May 1993. ACM Press.

[4] S. Cohen, W. Nutt, and Y. Sagiv. Equivalences among aggregate queries with negation. In *Proc. 20th Symposium on Principles of Database Systems*, pages 215–226, Santa Barbara (California, USA), May 2001. ACM Press.

[5] S. Cohen, W. Nutt, and Y. Sagiv. Containment of aggregate queries (extended version), Oct. 2002. Available at `http://www.cs.huji.ac.il/~sarina/papers/agg-containment-long.ps`.

[6] S. Cohen, W. Nutt, and A. Serebrenik. Rewriting aggregate queries using views. In C. Papadimitriou, editor, *Proc. 18th Symposium on Principles of Database Systems*, pages 155–166, Philadelphia (Pennsylvania, USA), May 1999. ACM Press.

[7] S. Cohen, W. Nutt, and A. Serebrenik. Algorithms for rewriting aggregate queries using views. In *Symposium on Advances in Databases and Information Systems, Enlarged Fourth East-European Conference on Advances in Databases and Information Systems*, pages 65–78, Prague (Czech Republik), Sept. 2000.

[8] O. Duschka and M. Genesereth. Answering recursive queries using views. In *Proc. 16th Symposium on Principles of Database Systems*, pages 254–263, Tucson (Arizona, USA), May 1997. ACM Press.

[9] S. Grumbach, M. Rafanelli, and L. Tininini. Querying aggregate data. In C. Papadimitriou, editor, *Proc. 18th Symposium on Principles of Database Systems*, pages 174–183, Philadelphia (Pennsylvania, USA), May 1999. ACM Press.

[10] S. Grumbach and L. Tininini. On the content of materialized aggregate views. In *Proc. 19th Symposium on Principles of Database Systems*. ACM Press, 2000.

[11] A. Gupta and I. S. Mumick, editors. *Materialized Views—Techniques, Implementations and Applications*. MIT Press, 1999.

[12] Y. Ioannidis and R. Ramakrishnan. Beyond relations as sets. *ACM Transactions on Database Systems*, 20(3):288–324, 1995.

[13] R. Kimball. *The Data Warehouse Toolkit*. John Wiley and Sons, 1996.

[14] A. Levy. Answering queries using views: A survey. *The VLDB Journal*, 10(4):270–294, 2001.

[15] A. Levy, A. Mendelzon, Y. Sagiv, and D. Srivastava. Answering queries using views. In *Proc. 14th Symposium on Principles of Database Systems*, pages 95–104, San Jose (California, USA), May 1995. ACM Press.

[16] A. Levy, A. Rajaraman, and J. Ordille. Querying heterogeneous information sources using source descriptions. In *Proc. 22nd International Conference on Very Large Data Bases*, pages 251–262, Bombay (India), Sept. 1996.

[17] F. Llirbat, F. Fabret, and E. Simon. Eliminating costly redundant computations from SQL trigger executions. In *Proc. 1997 ACM SIGMOD International Conference on Management of Data*, pages 428–439, Tucson (Arizona, USA), June 1997.

[18] W. Nutt, Y. Sagiv, and S. Shurin. Deciding equivalences among aggregate queries. In J. Paredaens, editor, *Proc. 17th Symposium on Principles of Database Systems*, pages 214–223, Seattle (Washington, USA), June 1998. ACM Press. Long version as Report of Esprit LTR DWQ.

[19] R. Pottinger and A. Levy. A scalable algorithm for answering queries using views. In P. 26th International Conference on Very Large Data Bases, editor, *Proc. 26th International Conference on Very Large Data Bases*, Cairo (Egypt), 2000. Morgan Kaufmann Publishers.

[20] K. Ross, D. Srivastava, and S. Sudarshan. Materialized view maintenance and integrity constraint checking: Trading space for time. In *Proc. 1996 ACM SIGMOD International Conference on Management of Data*, pages 447–458, Montreal (Canada), June 1996.

[21] D. Srivastava, S. Dar, H. Jagadish, and A. Levy. Answering queries with aggregation using views. In *Proc. 22nd International Conference on Very Large Data Bases*, Bombay (India), Sept. 1996. Morgan Kaufmann Publishers.

[22] J. Ullman. *Principles of Database and Knowledge-Base Systems, Vol. I: Classical Database Systems*. Computer Science Press, New York (New York, USA), 1988.

[23] H. Yang and P.-A. Larson. Query transformation for PSJ queries. In *Proc. 13th International Conference on Very Large Data Bases*, pages 245–254, Brighton (England), Sept. 1987. Morgan Kaufmann Publishers.