Computing Cube View Dependences in OLAP Datacubes

Carlos Hurtado and Claudio Gutiérrez
Department of Computer Science, Universidad de Chile
{churtado,cgutierr}@dcc.uchile.cl

Abstract

A common technique for speeding up OLAP query processing is to materialize (pre-compute) some aggregate views, called cube views, and use them for the derivation of other cube views. The derivations are inferred from the dimension hierarchies, and are usually represented as a relation between cube views called cube dependence relation*, also called* summarizability relation *for a single dimension hierarchy. In this paper, we study the problem of computing the summarizability relation of dimension schemas that model structural irregularities of dimensions by means of integrity constraints. We study the intrinsic complexity of the problem, and present an algorithm which uses deep structural properties of dimension hierarchies and constraints to solve the problem. Finally, we give an extension of the algorithm to compute the dependence relation for the multidimensional case.*

1 Introduction

Multidimensional organization of information is not the only need of OLAP users. They need also to explore facts aggregated at different granularities. The queries that define the facts at each granularity are referred to as *cube views* [HM02]. A set of cube views is called a *data cube* [GBLHP96, HRU96]. In a typical OLAP system, users formulate cube views essentially by selecting a *granularity* (a list of category attributes, called simply *categories*) from the list of dimensions in the data model at hand.

A common technique for speeding up OLAP query processing is to materialize (pre-compute) some aggregate views and use them for the derivation of other aggregate queries. In order to take advantage of materialized views, the system should rewrite a selected aggregate query as another query that refers to materialized views. The process of finding such rewritings is known in the OLAP world as *aggregate navigation* [Kim95]. Since the cost of computing a cube view is essentially linear in the number of facts accessed in the computation, aggregate navigation can speed up this processing by a factor which approximates the ratio of raw facts referenced by the cube view to aggregated facts referenced by the rewriting (which can be up to several orders of magnitude in real scenarios [Kim95]).

The fundamental ideas in OLAP aggregate navigation were already present in the early 1980's in graph models for statistical data bases (e.g., the Statistical Object Representation Model (STORM [RS90]). In order to study aggregate navigation in these models, Rafanelli and Shoshani [RS90] introduced the notion of summarizability, which refers to the ability of an aggregate query called *summarization* (also known in the OLAP world as a *consolidation* or a *roll up operation*) to correctly compute a cube view from another cube view where both cube views are defined in a single dimension (and their granularities consist of a single category of the dimension).

Summarizability can be thought of as a property of a pair of categories in a given dimension [HM02]. As an example, in Figure 1, we may say that ProdCategory is summarizable from ProdType in the dimension `product`. This holds because the cube view at ProdCategory can be computed from the cube view at ProdType, using a summarization which is correct independently of the fact table upon which the cube views are defined (sales at product .)

In traditional OLAP, dimension schemas are hierarchy schemas like that of Figure 1(A). Formally, a category c is summarizable from another category c' in a dimension schema, whenever c is summarizable from c' in all the dimensions that are instances of the schema.

For $n > 1$ dimensions, aggregate navigation is based on the notion of cube view *dependence* first introduced by Harinarayan et al. [HRU96]. In the context of a fixed data cube, $\texttt{CubeView}_G$ denotes the cube view at granularity G. Harinarayan et al. write that $\texttt{CubeView}_{G'}$ depends on $\texttt{CubeView}_G$, iff $\texttt{CubeView}_{G'}$ can be answered using only the results of $\texttt{CubeView}_G$. In this context, a summarization from a fact table f (of granularity $G' = \langle {c'}_1, \ldots, {c'}_n \rangle$ and measure m) to a granularity $G = \langle c_1, \ldots, c_n \rangle$ in a list of dimensions $d_1, \ldots, d_n$ can be expressed with the following SQL query:

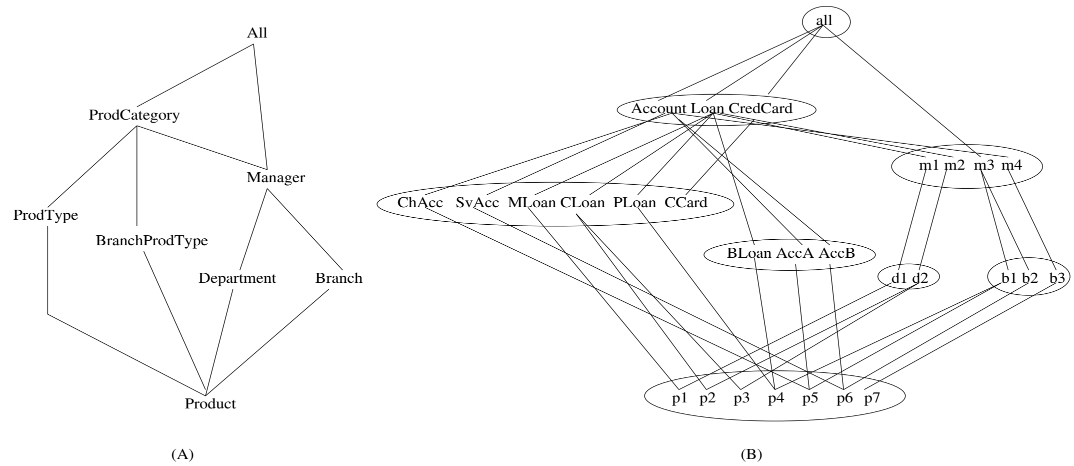

Figure 1. The dimension `product`: (A) hierarchy schema; (B) child/parent relation. In this dimension all products are classified through the hierarchy path of categories: $Product$-$ProdType$-$ProdCategory$-`All`. On the other hand, some types of products, like personal loans and some sorts of accounts, are handled by branches, whereas other types of products, like mortgage and corporate loans, are handled by departments. The products that are handled by branches are also classified according to the category $BranchProdType$. There is a manager in charge of each branch and department. Finally, it happens that some managers handle products in only one category, which explains the edge from $Manager$ to $ProductCategory$.

SELECT $G, sum(m)$
FROM $f, \Gamma^{c_1}_{c'_1}(d_1)$ as $R_1, \ldots, \Gamma^{c_n}_{c'_n}(d_n)$ as R_n
WHERE $f.c'_1 = R_1.c'_1, \ldots, f.c'_n = R_n.c'_n$
GROUP BY G

where $\Gamma^{c_i}_{c'_i}(d_i)$ denotes the rollup mappings from c'_i to c_i in the dimensions d_i. As an example, in the dimension `product` given in Figure 1, we have that $\Gamma^{Department}_{Product}(\texttt{product}) = \{(p_1, d_1), (p_2, d_2), (p_3, d_2)\}$. Intuitively, this summarization aggregates the fact table f to the granularity G using each of the $\Gamma^{c_i}_{c'_i}(d_i)$'s.

In this paper, we view *dependence* as a natural extension of summarizability for multidimensional granularities. Dependence can be thought of as a property of a pair of granularities in a list of dimension schemas $D_1, \ldots, D_n$ whenever the dependence holds in each list of dimensions $d_1, \ldots, d_n$, where d_i is an instance of D_i. It turns out that in traditional OLAP, the dependence relation can be characterized as follows: a granularity $\langle c_1, \ldots, c_n\rangle$ depends on a granularity $\langle c'_1, \ldots, c'_n\rangle$ in a list of dimension schemas $D_1, \ldots, D_n$ iff for each i, we have c'_i is an ancestor of c_i in D_i. This leads to a straightforward procedure to compute the dependence relation from the list of hierarchy schemas involved. The transitive transitive and reflexive reduction of the dependence relation, known as *cube dependence graph* is an useful representation for the system and user to navigate throughout aggregates at different granularities.

1.1 Problem Statement

A central drawback of traditional dimension models is that they do not account for structural heterogeneity. Such models require every member in a category c in the hierarchy schema to have a parent in every category directly above c, a condition we refer to as *homogeneity*. In other words, the rollup mappings are required to be total functions. This restriction is unnatural in several situations. For example, we may need to represent members that have parents (resp., ancestors) in different sets of categories. Heterogeneous dimensions also yield fewer numbers of categories, which might exponentially decrease the number of cube views we may need to handle and store. In [Hur02] there is a simple example in which the use of a heterogeneous model reduces the number of cube views from 4320 to 192.

In previous work [HM02] we showed that the hierarchy schema itself is not expressive enough to support summarizability reasoning in the presence of heterogeneity. In order to overcome this limitation, we introduced a class of integrity constraints, *dimension constraints*. In this model, a hierarchy schema along with a set of dimension constraints form a *dimension schema*. These schemas extend well known classes of OLAP dimension schemas, such as the model of Jagadish el al. [JLS99] which partially solves the limitations of traditional OLAP models by allowing several bottom categories (which allows unbalancedness), but keeping the homogeneity restriction. Different classes of dimension schemas are classfied in [Hur02] (where for "traditional OLAP" we refer to the basic class of homogenoeus schemas with a single bottom category (balancedness).)

Example 1 *We could model the dimension* `product` *(Figure 1) with the dimension schema* `productS`*. This schema consists of the hierarchy schema of Figure 1 (A) along with the following set of constraints:*
(a) $\langle Product, ProdType\rangle$
(b) $\langle Product, Branch\rangle \oplus \langle Product, Department\rangle$
(c) $\langle Product, Branch\rangle \equiv \langle Product, BranchProdType\rangle$
(d) $\langle Department, Manager, ProdCategory\rangle$
(e) $\langle Branch = b_3\rangle \equiv \langle Branch, Manager, ProdCategory\rangle$

The constraints refer essentially to relationships among members of the categories. For example, (a) says that all members of Product must have a parent in ProdType; (b) says that the member in Product must have a parent in Branch or in Department, but not in both; (e) says that the member b_3 *in Branch is the only one having a path via Manager to ProdCategory.*

When it comes to perform aggregate navegation in non traditional OLAP schemas, two main problems arise:

1. The dependence (resp. summarizability) relation captures just a small set of the useful rewritings, because summarizations do not consider derivations from combinations of cube views. Summarizations from a single granularity can be of little help for aggregate navigation because we usually need to combine several cube views in the derivation. Thus, dependence (resp. summarizability) shall be a property of a granularity from a set of granularities in a list of dimension instances.

Example 2 *Assume a fact table* $Sales(Product, Amount)$ *containing the total sales per product, where Product is the bottom category in the dimension* `product` *(see Figure 1) and Amount is a measure. Assume two materialized cube views: the first,* $SalesC$*, containing the total sales per branch product type; and the second,* $SalesD$*, containing the total sales per department.*

These cube views can be used to compute the cube view the total sales per Product Category using the following extension of a summarization:

SELECT $ProdCategory, sum(Amount)$
FROM
  (SELECT $ProdCategory, sum(Amount)$
  FROM $SalesC, \Gamma^{ProdCategory}_{BranchProdType}$(`product`) as R
  WHERE $SalesC.BranchProdType =$
    $R.BranchProdType$
  GROUP BY $ProdCategory$
  UNION ALL
  SELECT $ProdCategory, sum(Amount)$
  FROM $SalesD, \Gamma^{ProdCategory}_{Department}$(`product`) as S
  WHERE $SalesD.Department = S.Department$
  GROUP BY $ProdCategory$)
GROUP BY $ProdCategory$

Thus we could say that $ProdCategory$ *is summarizable from* $\{BranchProdType, Department\}$ *in the dimension* `product`.

2. Even if we consider the traditional notion of dependence (as explained before), the straightforward procedure to compute it becomes unsound when applied to non traditional OLAP. For example, the summarizability relation of the dimension schema `productS` (Example 1) is not equivalent to the transitive and reflexive clousure of its relation between categories (hierarchy schema): although Departament is a child of Manager, Manager is not summarizable from Departament.

Furthermore, the problem has non-trivial complexity especially if we consider combinations of summarizations in an extended notion of dependence.

Example 3 *In Figure 2 there are two hierarchy schemas (A) and (B). Assume first a classical interpretation, that is, (A) has the constraint* $\langle c, a\rangle \wedge \langle c, b\rangle$*, and (B) has* $\langle f, d\rangle \wedge \langle f, e\rangle$*. In this case both dimension schemas are traditional and the dependence relation can be easily computed using the direct product of both graphs. For example,* $(\texttt{all}, \texttt{all})$ *depends on* (a, d)*. Another dependence is on* (b, e)*, etc.*

Now, assume the same constraints for (A), and $\langle f, d\rangle \oplus \langle f, e\rangle$ *(exclusive or) for (B). In this case none of the aforementioned dependences hold. Now if we consider an extended notion of dependence which allow combinations of cube views whose facts are aggregated in a similar fashion as done in Example 2 a whole new bunch of dependences arise. For example,* $(\texttt{all}, \texttt{all})$ *depends on both* (a, d) *and* (a, e)*, because all base facts (at granularities* c, f*) that are aggregated to* $(\texttt{all}, \texttt{all})$ *are also aggregated to one of* (a, d) *or* (a, e) *but not to both (which avoids double counting).*

Finally, if we consider the constraints $\langle c, a\rangle \oplus \langle c, b\rangle$ *and* $\langle f, d\rangle \oplus \langle f, e\rangle$*, none of the previous dependences hold and it turns out that* $(\texttt{all}, \texttt{all})$ *depends on* $(a, d), (a, e), (b, d), (b, e)$.

The reader can guess from this elementary example how complex can become to compute all correct summariza-

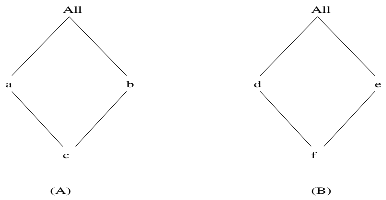

Figure 2. (A) The hierarchy schema of D_1. (B) The hierarchy schema of D_2.

tions when a non traditional schema is used, i.e, arbitrary Boolean combinations of statements are allowed.

1.2 Related Work

The notion of summarizability is from Rafanelli and Shoshani [RS90]. Summarizability was originally stated as a property of rollup mappings in a statistical object. Dependence has been formalized (for traditional homogeneous and balanced OLAP models) by Harinarayan et al. [HRU96]. As best we know, treating heterogeneity with null values is the approach taken by commercial OLAP systems nowadays. Null values may cause an exponential growth of the aggregates handled in data cubes. In addition, the facts associated with null values have no meaning for the users that query the cube. Pedersen et al. [PJE99] propose transforming a particular class of heterogeneous dimensions into balanced dimensions by adding null members to represent missing parents. The algorithm proposed, however, considers a restricted class of heterogeneous dimensions, and does not work with general heterogeneous dimensions.

The problem of computing the summarizability relation can be viewed as a particular version of the problem of finding rewritings of aggregate queries using aggregate views. Almost all the research that has addressed the problem of aggregate query rewriting considers relational data. Two papers address the problem in the context of multidimensional data. Chaudhuri [CD96] gives a procedure to compute aggregate rewritings of cube views with a select condition, defined over star schemas. The procedure considers rewriting using a single view. Albrecht et al. [AGL99] extend the framework to consider cube views defined over dimension hierarchies. These approaches consider only traditional dimension schemas.

We refer the reader to [Hur02] (Chapter 1) for an extended description of related work.

1.3 Contributions and Outline

In this paper we present a notion of dependence for non-traditional schemas which allow combinations of cube views in summarizations. This notion of dependence naturally extends summarizability, as introduced in previous work [HM02]. It also formalizes and extends the notion of *cube view dependence* given by Harinarayan et al. [HRU96], in order for it to account for combinations of cube views in summarizations.

We study the problems of computing the summarizability relation of categories from a given dimension schema. We tackle the problem incrementally, studying the different inference problems related to it, and their intrinsic complexity. We show that the problem cannot be solved in polynomial time and give lower an upper complexity bounds. We provide an algorithm for computing the summarizability relation using deep structural properties of dimension schemas and contraints. We show with experiments that the algorithm outperforms a naive approach (which consists in the iterative application of the summarizability test based on previous work on dimension constraint implication [HM02]) and performs well for schemas of reasonable size.

Finally, we give a characterization of dependence in terms of the dimension schemas involved. This yields an algorithm for computing the dependence relation, which also uses structural properties of the dimension schemas involved. We describe the algorithm and study its time complexity.

The remainder of this paper is organized as follows. In Section 2 we review the main concepts that will be used in the paper. Section 3 presents the algorithms for computing the summarizability relation. In Section , we give the characterization of dependence and an algorithm for computing the dependence relation. Finally, in Section 5 we conclude and outline some prospects for future work. The proofs and details are presented in [Hur02].

2 Preliminaries

In this section, we give the formal definitions and notations, introduced in previous work, necessary to state the problem. We review basic background on schemas, define the summarizability relation and its essential properties. We will give the definitions for a single dimension. In Section 4 we will show how this can be extended to multiple dimensions.

2.1 Graph Notation

It is convenient to refresh some elementary graph concepts. A (directed) graph G is a pair of sets (V, E) where

$E \subseteq V \times V$. Elements $v \in V$ are called *vertices* and pairs $(u, v) \in E$ (directed) *edges*; u and v are *adjacent* vertices. A *path* in G from v to w is a sequence of vertices $v = v_0, \ldots, v_n = w$ such that $(v_i, v_{i+1}) \in E$. We say that v *reaches* w. The *length* of the path is n. A *cycle* is a path with $v = w$. A *dag* is a directed acyclic graph. A *sink* in a dag is a distinguished vertex w reachable from every other vertex in the graph. A *source* in a dag is a distinguished vertex v from which every other vertex of the graph is reachable. A *shortcut* in a dag is a path of length > 1 between two adjacent vertices. Given a vertex v of G, an *upgraph* is the subgraph of G generated by v and all the vertices reachable from it. Given two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$, a *graph morphism* is a function $\phi : V_1 \rightarrow V_2$ preserving edges, that is, $(u, v) \in E_1$ implies $(\phi(u), \phi(v)) \in E_2$. The morphism ϕ is called an *isomorphism (resp. monomorphism, epimorphism)* if ϕ as a function is bijective (resp. injective, onto). The *direct product* $G_1 \times G_2$ is the graph (V, E) such that, where $V = V_1 \times V_2$ and $E = \{((v_1, v_2), (v'_1, v'_2)) : (v_1, v'_1) \in E_1 \wedge (v_2, v'_2) \in E_2\}$.

2.2 Dimensions

Definition 1 *Assume the existence of (possibly infinite) sets* $\mathbf{C}$ *(categories), and* $\mathbf{M}$ *(members). Let* $C \subseteq \mathbf{C}$ *and* $M \subset \mathbf{M}$.

1. *A* hierarchy schema *is a* dag $H = (C, \nearrow)$ *having a distinguished category* `All` $\in C$ *which is a sink.*

2. *A* hierarchy domain *is a* dag $h = (M, <)$ *having a distinguished member* `all` $\in M$ *which is a sink, and without shortcuts. (*$\ll$ *will denote the transitive closure of* $<$*; its reflexive and transitive closure, denoted* $\leq$*, is called* rollup relation*.)*

3. *A* dimension instance d over a hierarchy schema $(C, \nearrow)$ *is a graph morphism* $d : (M, <) \rightarrow (C, \nearrow)$ *such that: (a)* $(M, <)$ *is a hierarchy domain; (b)* $d($`all`$) =$ `All`*; and (c) for all* x *and* $y \neq z$*, if* $x \ll y \wedge x \ll z$ *then* $d(y) \neq d(z)$.

The last condition in item 2 (no shortcuts) avoids redundancies (transitive edges) in the representation of the data. The fact that d is a graph morphism in item 3 states that whenever we have a relationship $m_1 < m_2$ between some pair of members $m_1 \in c_1$ and $m_2 \in c_2$, then there is an edge $c_1 \nearrow c_2$ in the hierarchy schema representing links between categories c_1 and c_2. Condition c of item 3 is a basic restriction in OLAP data modeling [CT97, HM02, JLS99, LAW98], and states that the rollup relation $\leq$ is functional (i.e., single valued) between every pair of categories. This motivates to introduce the *rollup mapping* between two categories c_1 and c_2 of a dimension d, denoted $\Gamma_{c_1}^{c_2}(d)$, which is the restriction of $\leq$ to $d^{-1}(c_1)$ and $d^{-1}(c_2)$.

2.3 Constraints and Dimension Schemas

In our framework, a dimension schema consists of a hierarchy schema along with a set of *dimension constraints*.

Definition 2 (Dimension Constraints) *Let* $H = (C, \nearrow)$ *be a hierarchy schema,* $c \in C$*,* $K \subseteq \mathbf{M}$*. The language of constraints (with root c) has the following atoms: (1) Path atoms:* $\langle c, c_1, \cdots, c_n \rangle$*, where the* c_j *must satisfy that* $cc_1 \cdots c_n$ *is a path in H; (2) Equality atoms:* $\langle c, \ldots, c' = k \rangle$*, where* c' *is such that there is a path from* c *to* c'*, and* $k \in K$.

A dimension constraint with root c *is a Boolean combination* ϕ *of atoms of the above kind.*

Dimension constraints consider the usual connectives $\neg, \wedge, \vee, \Rightarrow, \Leftrightarrow$, and $\oplus$ for exclusive disjunction. As usual, $\perp$ (resp. $\top$) will denote the false (resp. true) proposition. In addition, given a set of atoms A, $\bigodot_A$ denotes that there is exactly one true atom in A.

Definition 3 (Semantics of Constraints) *Let* $d : (M, <) \rightarrow (C, \nearrow)$ *be a dimension instance, and* ϕ *a constraint with root c. Then* $d \models \phi$ *if and only if*
for all $m \in d^{-1}(c)$*,* $d \models \phi[c/m]$*,*
where $d \models \phi[c/m]$ *is defined recursively as follows:*

1. $d \models \langle c, c_1, \ldots, c_n \rangle[c/m]$ *iff there is a path* $mx_1 \cdots x_n$ *in* $(M, <)$ *with* $d(x_i) \in c_i$.

2. $d \models \langle c, \ldots, c' = k \rangle[c/m]$ *iff* $d(k) \in c'$ *and* $m \leq k$.

3. $d \models (\phi \wedge \psi)[c/m]$ *iff* $d \models \phi[c/m]$ *and* $d \models \psi[c/m]$. *Similarly for* $\vee$ *and the other Boolean connectives.*

Given a hierarchy schema H and two sets of constraints Σ, Σ' over H, we say that Σ is equivalent to Σ', if for all dimension instances d over H it holds: $d \models \Sigma$ iff $d \models \Sigma'$.

Now we are ready to introduce the concept of Dimension Schema. The following definition extends Definition 1 (1) in the presence of constraints.

Definition 4 (Dimension Schema) *A* dimension schema *is a pair* (H, Σ) *where* H *is a hierarchy schema and* Σ *is a set of constraints.*

A dimension instance d over a dimension schema $D = (H, \Sigma)$ *is a dimension instance* d *over* H *such that* $d \models \Sigma$. *The set of dimensions instances over* D *will be denoted by* $I(D)$.

2.4 Frozen Dimension

Intuitively, a *frozen dimension* is a minimal dimension instance conveyed by a dimension schema. They are minimal because they contain at most one member per each category and have a single bottom category. Each frozen dimension shows a structure (upgraph of some bottom category of the hierarchy schema), along with some constants that appear in the schema and other arbitrary members (we refer the reader to previous work [HM02] for details.) Frozen dimension are important because, as showed in previous work, in order to test implication of a constraint (with root is c) from a dimension schema, we only need to test whether the constraint holds for each of the frozen dimensions of the schema (whose upgraph start from c). In addition, as we will see in the next section, the notion of frozen dimension is essential for studying algorithmic aspects of summarizability in dimension schemas.

Definition 5 (Frozen Dimension) *Let $D = (H, \Sigma)$ and $H = (C, \nearrow)$. Let $\texttt{NotKnown} : \mathbf{C} \rightarrow \mathbf{M}$ an injective function that assigns a fix member to each category.*

A frozen dimension *with root c is a dimension instance $d : (M, <) \rightarrow (C, <)$ such that: (1) d is injective; (2) $c \in Im(d)$ and $d^{-1}(c)$ is a source in $(M, <)$; and (3) all $x \in M$ appear in an atom $c = k$ in Σ or $x \in \texttt{NotKnown}(\texttt{d}(\texttt{x}))$.*

We denote by $\texttt{FrozenDim}(D, c)$ the set of frozen dimension of D with root c, and by $\texttt{FrozenDim}(D)$ the union of all $\texttt{FrozenDim}(D, c)$ for all categories c of D.

Example 4 *Figure 3 depicts the frozen dimensions of the schema* productS *(Example 1) with root Product. The figure shows the subgraphs of the hierarchy schema induced by the non-empty categories of the frozen dimensions.*

2.5 The Summarizability Relation

Let d be a dimension instance, $S = \{f_1, \ldots, f_n\}$ a set of fact tables (where all f_j have measure m, and the granularity of f_j is a category c_j in d), and c a fixed category in d. A *summarization from S to c in d* is the following query:

SELECT c, $\texttt{af}(m)$
FROM $f_1, \ldots, f_n, \Gamma^{c}_{c_1}(d)$ as $R_1, \ldots, \Gamma^{c}_{c_n}(d)$ as R_n
WHERE $f_i.c_i = R_i.c_i, \ldots, f_n.c_n = R_n.c_n$
GROUP BY c

We will abbreviate this query as $\texttt{SUMM}_c(d, S)$. Intuitively, this summarization aggregates the set $S = \{f_1, \ldots, f_n\}$ of fact tables to the granularity c using the rollup mappings of the dimension d.

Our definition of summarizability is based on the equivalence of two queries, the cube view and the summarization. In previous work [HM02] we presented the following characterization of summarizability in terms of dimension constraints, which shows that summarizability can be characterized as a property of dimension instances themselves, avoiding the mention of fact tables and aggregate views.

Theorem 1 (Summarizability) *A category c is summarizable from a set of categories S in a dimension instance d iff for every bottom category c_b of d we have $d \models \langle c_b, \cdots, c\rangle \Rightarrow \bigodot_{c_i \in S}\langle c_b, \cdots, c_i, \cdots, c\rangle$.*

The intuition behind Theorem 1 is that, in order for c to be summarizable from S, it must be the case that every base member (i.e., a member in a bottom category) that rolls up to c, rolls up to c passing through one and only one of the categories in S. This avoids repetitions.

Example 5 *In the dimension* product *(Figure 1), we have that $ProdCategory$ is summarizable from $\{BranchProdType, Department\}$ because*

$$\texttt{product} \models \langle Product, \cdots, ProdCategory\rangle \Rightarrow \langle Product, \cdots, BranchProdType, \cdots, ProdCategory\rangle \odot \langle Product, \cdots, Department, \cdots, ProdCategory\rangle.$$

Now we are ready to define the summarizability relation in the setting of dimension schemas.

Definition 6 (Summarizability Relation) *Let D be a dimension schema, d a dimension instance of D, S a set of categories, and c a category.*

We say that c is summarizable from S in D *if c is summarizable from S in every dimension instance d of D. Moreover, we say that c is* minimally summarizable *from S in D if c is summarizable from S in D, and there is no proper subset $S' \subset S$ such that c is summarizable from S' in D.*

The summarizability relation *of a dimension schema D is the relation containing the pairs (c, S) such that c is minimally summarizable from S in D.*

Let c and c' be two categories in a dimension schema D. The following are some basic properties of the summarizability relation: the pair $(c, \{c\})$ is always in the summarizability relation of D; and if (c, S) is in the summarizability relation of D, then for all $c' \in S$, $c' \nearrow^* c$.

3 Computing the Summarizability Relation

The problem of *computing the summarizability relation* is, given a dimension schema, to compute its summarizability relation. We will first address the problem of *finding summarizations*, in which we have to compute the projection of the summarization relation for a specific category, which represents all the possible minimal summarizations that correctly compute the cube view at c from cube views at categories in S. Formally, we are asked to compute the sets $S' \subseteq S$ such that c is minimally summarizable from S' in D, for a given dimension schema D, category c, and set of categories S.

In remainder of the section, we assume D is a dimension schema, c a category, and S a set of categories.

3.1 Complexity bounds

In previous work [HM02] it was shown that testing summarizability (i.e., deciding whether a category is summarizable from a set of categories in a dimension schema) is coNP-complete. Another complex problem is to determine whether there exists a set $S' \subseteq S$ such that c is summarizable from S' in D:

Proposition 1 (Bounds on Finding Summs.) *Given D, c, S, determining whether there exists a set $S' \subseteq S$ such that c is summarizable from S' in D is NP-hard.*

The proof of the proposition is by a transformation from *exact cover by three sets*. It is easy to see that the decision problem is in Σ_2P. An NP^{NP} algorithm for the decision problem needs only to guess a subset S' of S, and check using an oracle in NP, whether c is minimally summarizable from S in D. It is not known if the decision problem is Σ_2P-hard for DC schemas.

It is worth noting that the complexity of the problem continues to be NP-hard even for a class of homogeneous schemas introduced by Jagadish et al [JLS99]. (For further discussion see [Hur02].)

3.2 The naive algorithm

Given a dimension schema D, a category c of D, and a set S of categories, the simplest idea is to compute the sets $S' \subseteq S$ such that (c, S') belongs to the summarizability relation of D, using some algorithm to test summarizability of c from each candidate set S' in 2^S. We need to enumerate the power set 2^S to avoid repetitions. This can be easily implemented using Rymon's set-enumeration [Rym92][1]

The cost of this naive algorithm can be estimated by the number of nodes in the tree visited times the cost of testing summarizability. This test can be done by testing whether the summarizability condition of Theorem 1 is inferred from the dimension schema. As mentioned above, this problem is coNP-complete; we may use the algorithm `DIMSAT` [HM02] to check constraint inference, which runs in time $O(f_{\texttt{DIMSAT}})$, where $f_{\texttt{DIMSAT}}(N, |\Sigma|) = 2^{p(N,|\Sigma|)}$, with $p(x, y)$ a quadratic polynomial. Thus, the complexity depends on the number N of categories of D and the size $|\Sigma|$ of the constraints.

Since we can easily construct instances of the problem where the size of the solution is exponential on the size $|S|$ of S, even if we use some heuristic rules to reduce the size of the search space, we may need an exponential number of calls to `DIMSAT` in the worst case. (Observe that $|S| \leq N$)

Proposition 2 *The naive algorithm runs in time $O(2^{|S|} N f_{\texttt{DIMSAT}}(N, |\Sigma|))$.*

The naive algorithm follows a two-step strategy of choosing a candidate rewriting and proving its correctness. A better approach to solve the problem needs more structural information about the schema. This is what we do next.

[1] In Rymon's set-enumeration, the power set is organized as a tree, where the children of each node are generated by appending terms in a given ordering.

3.3 Summarizability and Frozen Dimensions

The results presented in this section will be fundamental to the algorithms we propose for finding summarizations, and for computing the summarizability relation.

In the sequel, a subgraph of a hierarchy schema induced by the non-empty categories of a frozen dimension will be called a *frozen graph*. We denote by $\texttt{FrozenG}(D, c)$ the set of frozen graphs of D with root c. The following proposition characterizes summarizability in terms of frozen graphs.

Proposition 3 (Summ. and Frozen Dims.) *Given a dimension schema D, a category c of D, and a set of categories S of D,*

(1) c is summarizable from S in D iff for every bottom category c_b of D, and frozen graph $g \in \texttt{FrozenG}(D, c_b)$ such that c is a category of g, there exists exactly one category $c' \in S$ such that $c_b \nearrow^* c'$ and $c' \nearrow^* c$ in g;

(2) c is minimally summarizable from S in D iff (i) c is summarizable from S in D, and (ii) for every category $c' \in S$ there is a bottom category c_b of D, and a frozen graph $g \in \texttt{FrozenG}(D, c_b)$, such that $c' \nearrow^* c$ in g.

As the reader is surely guessing, the exact set cover problem is playing around. Given D, c, S, define the following instance of exact set cover $\mathcal{XC}(D, c, S) = (\mathcal{X}, \mathcal{C})$: (a) $\mathcal{X}$ is the set of all frozen graphs $g \in \texttt{FrozenG}(D, c_b)$ containing the vertex c, where c_b is a bottom category of D; (b) for each $c_i \in S$, define C_i as the set of frozen graphs $g \in \mathcal{X}$ such that c_i is in g and $c_i \nearrow^* c$ in g. Then $\mathcal{C}$ is the union of all C_i's.

The next theorem follows from Proposition 3.

Theorem 2 *A category c is minimally summarizable from S' ($S' \subseteq S$) in D iff S' is an exact cover in $\mathcal{XC}(D, c, S)$.*

Example 6 *Consider the dimension schema* `productS` *(Example 1). Figure 3 (above) shows the set of frozen graphs* $\texttt{FrozenG}(\texttt{productS}, Product)$. *The instance of* finding exact covers

$$\mathcal{XC}(\texttt{productS}, ProdCategory, \{Department, Branch, BranchProdType, ProdType\})$$

is depicted in Figure 3 (below). Here we have that $\mathcal{X}$ is the set $ProdCategory'$ which contains the elements $\{f_1, f_2, f_3\}$. In addition, $\mathcal{C}$ consists of the sets $Department'$, $Branch'$, $BranchProdType'$, and $ProdType'$.

3.4 Algorithms

We can avoid the exponential number of calls to `DIMSAT` in the naive algorithm by performing the following steps, which are the main modules of the algorithm $\texttt{FINDSUMMS}(D, c, S)$: (1) Compute $\texttt{FrozenG}(D, c_b)$ for each bottom category c_b, and construct an instance of *finding exact covers*; (2) Solve this instance of *finding exact covers*. The correctness of the algorithm follows from Theorem 2.

Let N_F be the number of bottom frozen graphs of D, and let $|S|$ be the size of the set of categories S.

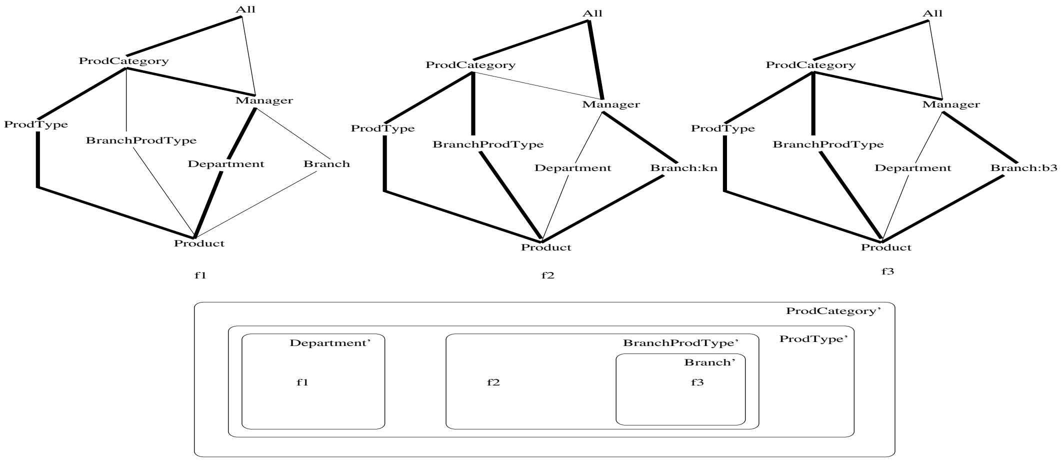

Figure 3. Transformation of ***finding summarizations*** **to** ***finding exact covers*****.**

Proposition 4 (Complexity of `FINDSUMMS`**)** `FINDSUMMS` *runs in time* $O(N f_{\texttt{DIMSAT}}(N, |\Sigma|) + 2^{|S|} N_F |S|)$.

The first term in the expression of Proposition 4 represents the cost of computing the instance of *finding exact covers*. In order to do this we compute the frozen dimension which can be done with a mild modification of `DIMSAT` [HM02]. Thus, instead of the exponentiall number of calls to `DIMSAT`, we only need a single call for each bottom category. The second term represents the cost of computing the exact covers.

Finally, we present the algorithm `COMPSUMMAREL` to compute the summarizability relation. `COMPSUMMAREL` first computes the bottom frozen graphs in order to build an instance of findig exact cover, for each category c and set of categories below c (in the same manner as done in `FINDSUMMS`). Finally, it iterates over each category finding the exact covers.

Proposition 5 (Complexity of `COMPSUMMAREL`**)** `COMPSUMMAREL` *runs in time*
$O(N f_{\texttt{DIMSAT}}(N, |\Sigma|) + 2^N N_F N^2))$.

The first term in the expression of Proposition 5 represents the cost of computing the instances of *finding exact covers*. This step is dominated by the computation of the bottom frozen graphs of the schema; thus the cost of this processing is similar to the cost of computing the single instance of *finding exact covers* in `FINDSUMMS`. The second term in the expression of Proposition 5 represents the cost of finding the exact covers.

3.5 Implementation

To assess the performance of the `COMPSUMMAREL` algorithm we implemented it using Java, and performed some simple experiments on a Pentium II computer, with CPU clock rate of 266 MHz, 128MB RAM, and running Windows 2000. We run the algorithm for four dimension schemas: (1) `productS`, which is the dimension schema of Example 1; (2) `location`, a schema that models stores, introduced in [HM02], with 7 categories, 7 constraints, yielding 4 frozen dimensions; (3) `lattice1`, which is a lattice of 8 categories in the corners of tree adjacent squares, having a disjunctive constraint in each node; and (4) `lattice2` which is similar to `lattice1`, but with 12 categories.

	Load	`COMPSUMMAREL`	**Naive**
`productS`	.22	.22	3.85
`location`	.22	.17	1.59
`lattice1`	.17	.16	1.92
`lattice2`	.21	2.64	41.52

Figure 4. Running times (seconds) for computing the summarizability relation.

Figure 4 shows the running times in seconds for `COMPSUMMAREL` and the naive algorithm, with each of the aforementioned schemas. The first column shows the time the algorithms load the dimension schema and computing the transitive closure of the hierarchy schema. The table shows that `COMPSUMMAREL` clearly outruns the naive algorithm. The implementation of `COMPSUMMAREL` uses Rymon's set enumeration to find exact covers and it does not uses rules to prune the space of sets to be visited, which allows us to compare the strategy in `COMPSUMMAREL` with the naive approach. So the running times of `COMPSUMMAREL` that we show in Figure 4 could be improved considerably.

4 Computing the dependence relation

Until now we presented with details the case of one dimension. We show in this section that all results hold for several dimensions.

4.1 The Dependence Relation

The dependence relation extends the summarizability relation to deal with many dimensions.

Definition 7 (Dependence) *A granularity G* depends *on a set of granularities $S = \{G_1, \ldots, G_p\}$ in a list of dimensions $\delta = \langle d_1, \ldots, d_n\rangle$ (G is a granularity of δ, and S contains granularities of δ) if for every base fact table f_{base} of the list of dimensions*

$$\texttt{CubeView}_G(\delta, f_{base}) = \texttt{SUMM}_G(\delta, \{\texttt{CubeView}_{G_1}(\delta, f_{base}), \ldots, \texttt{CubeView}_{G_p}(\langle\delta, f_{base}\rangle)\}),$$

where $\texttt{SUMM}_G$ is a summarization (which extends for $n > 1$ dimensions the summarization defined in Section 2.5) that aggregates the cube views at granularities $G_1, \ldots, G_p$ to G using the rollup mappings of the dimensions.

Dependence and minimal dependence in list of dimension schemas are mild extensions of the notions of summarizability and minimal summarizability in a dimension schema (Definition 6). The *dependence* relation of a list of dimension schemas $\mathcal{D}$, is the set of pairs (G, S) such that G depends minimally on S in $\mathcal{D}$. Notice that, if $\mathcal{D}$ has a single dimension schema, the dependence and the summarizability relation of $\mathcal{D}$ are the same.

4.2 Dependence and Frozen Dimensions

In this section, we give a sound and complete algorithm for computing the dependence relation. First, we will characterize dependence in terms of frozen graphs (recall the notion of frozen graph from Section 3.3). In order to do this, we will use the *direct product* of two frozen graphs, as defined in Section 2.1.

The following theorem is a generalization of Proposition 3. It characterizes dependence in term of the frozen graphs of the dimension schemas involved.

Proposition 6 (Dep. and Frozen Dims.) *Given a list of DC dimension schemas $\mathcal{D} = \langle D_1, D_2\rangle$, a granularity $G = \langle c_1, c_2\rangle$ in $\mathcal{D}$, and a set S containing granularities in $\mathcal{D}$:*

(1) G depends on S in $\mathcal{D}$ iff for every bottom categories c_{b1} of D_1 and c_{b2} of D_2, and frozen graphs $g_1 \in \texttt{FrozenG}(D_1, c_{b1})$, and $g_2 \in \texttt{FrozenG}(D_2, c_{b2})$, such that G is a node of $g_1 \times g_2$ we have that there is exactly one granularity G' in S such that $\langle c_{b1}, c_{b2}\rangle \nearrow^* G'$ and $G' \nearrow^* G$ in $g_1 \times g_2$.

(2) G depends minimally on S in Δ iff (i) G depends on S in Δ, and (ii) for every granularity $G' \in S$ there are bottom frozen graphs g_1 of D_1 and g_2 of D_2 such that $G' \nearrow^* G$ in $g_1 \times g_2$.

Since, a traditional dimension schemas has a single frozen graph, Proposition 6 generalizes the result by Harinarayan et al. [HRU96] in which dependence is characterized using the direct product of hierarchy schemas.

Recall from Section 3.3 the instance of *finding exact covers*, $\mathcal{XC}(D, c, S) = (\mathcal{X}, \mathcal{C})$, obtained from the instance of *finding summarizations* of c from S in D. We will define $\mathcal{XC}(\mathcal{D}, G, S)$ for the multidimensional case. However, instead of obtaining it by performing the direct product of bottom frozen graphs, we derive it by combining instances of *finding exact covers*, one for each dimension in the list of dimensions $\mathcal{D}$. In this form, we avoid computing and processing direct products of the frozen graphs.

The next Therem follows from Proposition 6.

Theorem 3 (Dependence and Exact Covers) *A granularity $G = \langle c_1, c_2\rangle$ depends minimally on a set of granularities S' ($S' \subseteq S$) in $\langle D_1, D_2\rangle$ iff S' is an exact cover in $\mathcal{XC}(DS, \langle c_1, c_2\rangle, S) = (\mathcal{X}, \mathcal{C})$, where $\mathcal{X}$ and $\mathcal{C}$ are as defined above.*

4.3 Algorithm

Theorem 3 leads to an algorithm, called `COMPDEPREL`, to compute the dependence relation in a list of DC dimension schemas. The algorithm first computes an array of instances of finding exact cover for each dimension schema, and then it combines the arrays to produce another array of instances of finding exact covers. Finally, the exact covers are computed yielding the dependence relation. Let N_1 and N_2 be the number of categories of the two dimension schemas, let N_G be the number of granularities ($N_G = N_1 \times N_2$), and let N_F be the number of direct products of frozen graphs in the schemas.

Proposition 7 (Complexity of `COMPDEPREL`) *Algorithm `COMPDEPREL` runs in time $O(N_{bott1} f_{\texttt{DIMSAT}}(N_1, |\Sigma_1|) + N_{bott2} f_{\texttt{DIMSAT}}(N_2, |\Sigma_2|) + 2^{N_G} N_F {N_G}^2)$.*

The first and second terms in the expression of Proposition 7 represent the cost of generating the instances of *findig exact covers* for the first and second schema. The last term represents the cost of finding the exact covers. The algorithm `COMPDEPREL` (and Proposition 7) can be easily generalized to deal with lists of $n > 2$ dimensions.

5 Conclusion

Traditional OLAP aggregate navigators do not model heterogeneity; designer must add null values to force homogeneity. Saving storage space is of central importance in OLAP because of the exponential explosion of the aggregates in data cubes, and the limited time window systems have for maintaining the aggregates. Although our framework requires more complex inference algorithms, it allows us to perform aggregate navigation without additional overhead on the data instance.

The test of the proposed algorithm for complex dimension schemas, along with the implementation of the algorithm for computing the dependence relation are obvious tasks for future work.

Acknowledgments

This research was supported by Millenium Nucleus, Center for Web Research (P01-029-F), Mideplan, Chile.

References

[AGL99] J. Albrecht, H. Guenzel, and W. Lehner. Set-derivability of multidimensional aggregates. In *Proceeding of the First International Conference on Data Warehousing and Knowledge Discovery (DaWaK)*, Florence, Italy, 1999.

[CD96] S. Chaudhuri and U. Dayal. Data warehousing and OLAP for decision support. In *Tutorial of the 22nd International Conference on Very Large Data Bases*, Mumbai, India, 1996.

[CT97] L. Cabibbo and R. Torlone. Querying multidimensional databases. In *Proceedings of the 6th International Workshop on Database Programming Languages*, East Park, Colorado, USA, 1997.

[GBLHP96] J. Gray, A. Bosworth, A. Layman, and H. H. Pirahesh. Data cube : A relational operator generalizing group-by, cross-tab and sub-totals. In *Proceedings of the 12th IEEE International Conference on Data Engineering*, New Orleans, Los Angeles, USA, 1996.

[HM02] C. Hurtado and A. Mendelzon. OLAP dimension constraints. In *Proc. PODS 2002*, Madison, USA, 2002.

[HRU96] V. Harinarayan, A. Rajaraman, and J. Ullman. Implementing data cubes efficiently. In *Proceedings of the 1996 ACM International Conference on Management of Data*, Montreal, Canada, 1996.

[Hur02] C. Hurtado. Structurally heterogeneous OLAP dimensions. In *Doctoral Thesis, Computer Science Dep. Toronto. URL: www.cs.toronto.edu/˜chl/thesiss.ps*, 2002.

[JLS99] H. V. Jagadish, L. V. S. Lakshmanan, and D. Srivastava. What can hierarchies do for data warehouses? In *Proc. of the 25th International Conference on Very Large Data Bases*, Edinburgh, Scotland, UK, 1999.

[Kim95] R. Kimball. The aggregate navigator. *DBMS and Internet Systems Magazine, http://www.dbmsmag.com*, November 1995.

[LAW98] W. Lehner, H. Albrecht, and H. Wedekind. Multidimensional normal forms. In *Proceedings of the 10th Statistical and Scientific Database Management Conference*, Capri, Italy., 1998.

[PJE99] T. B. Pedersen, C. S. Jensen, and Dyreson C. E. Extending practical pre-aggregation in on-line analytical processing. In *Proceedings of the 25th International Conference on Very Large Data Bases*, Edinburgh, Scotland, 1999.

[RS90] M. Rafanelli and A. Shoshani. Storm: A statistical object representation model. In *Proceedings of the 5th Statistical and Scientific Database Management Conference (SSDBM)*, Charlotte, N.C., USA, 1990.

[Rym92] R. Rymon. Search through systematic set enumeration. In *Proceedings of the Third International Conference on Principles of Knowledge Representation and Reasoning (KR)*, pages 539–550, San Mateo, California, 1992. Morgan Kaufmann.