OLAP Dimension Constraints

Carlos A. Hurtado
University of Toronto
chl@cs.toronto.edu

Alberto O. Mendelzon
University of Toronto
mendel@cs.toronto.edu

ABSTRACT

In multidimensional data models intended for online analytic processing (OLAP), data are viewed as points in a multidimensional space. Each dimension has structure, described by a directed graph of categories, a set of members for each category, and a child/parent relation between members. An important application of this structure is to use it to infer summarizability, that is, whether an aggregate view defined for some category can be correctly derived from a set of precomputed views defined for other categories. A dimension is called heterogeneous if two members in a given category are allowed to have ancestors in different categories. In previous work, we studied the problem of inferring summarizability in a particular class of heterogeneous dimensions. In this paper, we propose a class of integrity constraints and schemas that allow us to reason about summarizability in general heterogeneous dimensions. We introduce the notion of frozen dimensions, which are minimal homogeneous dimension instances representing the different structures that are implicitly combined in a heterogeneous dimension. Frozen dimensions provide the basis for efficiently testing implication of dimension constraints, and are useful aid to understanding heterogeneous dimensions. We give a sound and complete algorithm for solving the implication of dimension constraints, that uses heuristics based on the structure of the dimension and the constraints to speed up its execution. We study the intrinsic complexity of the implication problem, and the running time of our algorithm.

1. INTRODUCTION

In multidimensional data models intended for online analytic processing (OLAP), data are viewed as points in a multidimensional space; for example, a sale of a particular item in a particular store of a retail chain can be viewed as a point in a space whose dimensions are items, stores, and time, and this point is associated with one or more *measures* such as price or profit. Dimensions themselves have structure; for example, along the store dimension, individual stores may be grouped into cities, which are grouped into states or provinces, which are grouped into countries. The relationship from elements at a finer granularity and those at a coarser granularity is called *rollup*; thus we would say that the city "Toronto" rolls up to the province "Ontario" and, transitively, it also rolls up to the country "Canada."

1.1 Heterogeneous Dimensions

The traditional approach to dimension modeling required every pair of elements of a given category to have ancestors in the same set of categories, a restriction referred to as *homogeneity*. For example, in a homogeneous dimension we cannot have some cities that rollup to provinces and some to states. A number of researchers and practitioners [11, 8, 13, 6] have dropped this restriction over the past few years, yielding *heterogeneous* dimensions, which are needed to represent more naturally and cleanly many practical situations. Moreover, heterogeneous dimensions permit more efficient storage of data by having fewer categories. A smaller number of categories might exponentially decrease the number of aggregate views we may need to handle and store in OLAP systems.

EXAMPLE 1. *The dimension instance of Figure 1, called* `location`*, represents the stores of a retailer. In our hypothetical scenario, the retailer has stores in Canada, Mexico, and USA. All the stores rollup to City, SaleRegion, and Country. However, while the stores in Canada rollup to Province, the stores in Mexico and USA rollup to State. The city Washington is an exception to the latter, since it rolls up directly to Country without passing through State. On the other hand, the states of Mexico and the provinces rollup to SaleRegion, while the states of USA do not necessarily rollup to SaleRegion.*

1.2 Summarizability

Cube views are simple aggregate queries that provide the basis for OLAP query formulation. A single-dimension cube view on a dimension d (e.g. the `location` dimension) is specified by picking a category within the hierarchy for d (e.g. the *Province* category) and a distributive [1] aggregate

[1]A distributive aggregate function `af` can be computed on a set by partitioning the set into disjoint subsets, aggregating each separately, and then computing the aggregation of these partial results with another aggregate function we will denote as $\texttt{af}^c$. Among the SQL aggregate functions, `COUNT`, `SUM`, `MIN`, and `MAX` are distributive. We have that $\texttt{COUNT}^c = \texttt{SUM}$; and for `SUM`, `MIN`, and `MAX`, $\texttt{af}^c = \texttt{af}$.

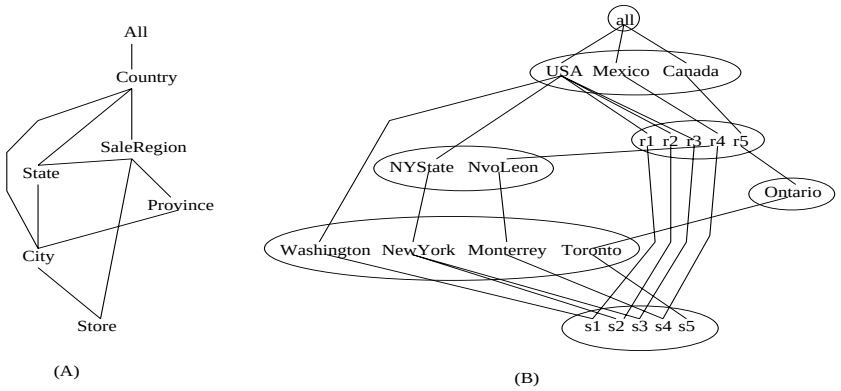


Figure 1: The dimension `location`: (A) hierarchy schema; (B) child/parent relation.

function (e.g. sum). This view, applied to a fact table, aggregates the raw data in it to the level of aggregation specified by the category; for example, it sums the sales of all stores grouped by province.

A key strategy for speeding up cube view processing is to reuse pre-computed cube views. In order to do this, the system must rewrite a cube view as another query that refers to pre-computed cube views. The process of finding such rewritings is known in the OLAP world as *aggregate navigation* [9]. The notion of summarizability was introduced to study aggregate navigation in statistical objects and OLAP dimensions [12, 11, 13, 6]. As originally stated, summarizability refers to whether a simple aggregate query (usually called *summarization* or *consolidation*) correctly computes a single-category cube view from another precomputed single-category cube view, in a particular database instance. In previous work [6] we extended summarizability to allow the combination of several cube views in the rewriting. The notion we use in this paper is: a category c of dimension d is summarizable from a set of categories $\{c_1, \ldots, c_n\}$ of dimension d if, for every fact table and every distributive aggregate function, the cube view for c can be computed (by a simple relational algebra expression) from the cube views on the c_i's. A formal definition is given in Section 3.

Just as database instances are modeled by database schemas, dimension instances (like the one in Figure 1(B)) are modeled by dimension schemas (basically the diagram in Figure 1(A)). *Testing summarizability* is the problem of deciding, given a dimension schema ds, a category c, and a set of categories S, whether c is summarizable from S in all the dimension instances represented by ds. In most dimension models in the literature, the dimension schema basically consists of the *hierarchy schema*, the DAG shown in Figure 1(A). Such models lack a language for describing integrity constraints on the schema other than the ones that are inherent in the hierarchy schema. This weakens the ability of OLAP systems to test summarizability.

EXAMPLE 2. *In the dimension* `location` *(depicted in Figure 1), we have that Country is summarizable from* $\{City\}$. *Intuitively, this happens because (i) all the stores rollup to Country passing through City. However, we cannot infer (i) just by analyzing the hierarchy schema of Figure 1 (A). This hierarchy schema may allow stores that rollup to Country passing through SaleRegions, without going though the category City.*

A new class of constraints is needed to express integrity constraints in OLAP dimensions, and to turn dimension schemas into adequate abstractions to model heterogeneity and to support the summarizability testing.

1.3 Related Work

Kimball [10] introduced the term *heterogeneity* to refer to the situation where several dimensions representing the same conceptual entity, but with different categories and attributes, are modeled as a single dimension table. Lehner et al. [11], and Pedersen and Jensen [13] account for heterogeneity, and propose different solutions to deal with summarizability. Lehner et al. propose transforming heterogeneous dimensions into homogeneous dimensions, which they say to be in *dimensional normal form* (DNF). The transformation is done by treating categories causing heterogeneity as attributes for tables outside the hierarchy. The proposed transformation flattens the child/parent relation, limiting summarizability in the dimension instance.

Pedersen and Jensen [14] model a particular class of heterogeneous dimensions, and propose transforming them into homogeneous dimensions by adding null members to represent missing parents. This solution has several drawbacks. First, the transformation algorithm proposed considers a restricted class of heterogeneous dimensions, and does not scale to general heterogeneous dimensions. In addition, null members may cause considerable waste of memory and computational effort due to the increased sparsity of the cube views.

Although database researchers have done abundant work on integrity constraints for a variety of data models, almost nothing has been said about integrity constraints in the context of OLAP dimension modeling. In previous work [6], we introduce *split constraints*, which are statements about possible categories the members in a given category may rollup to. Split constraints allow summarizability to be characterized only in a particular class of heterogeneous dimensions that keep a notion of ordering between the granularities defined by categories. Moreover, split constraints are insuffi-

cient for our problem because in the general case heterogeneity would be better captured by possible hierarchy paths, rather than possible sets of categories to which members rollup to. Goldstein [4] proposes to capture heterogeneity in database relations by means of *disjunctive existential constraints* (dec's). The main idea here is to model a relation as a combination of objects, each one determined by a set of non-null attributes that appear together. Dec's represent a particular class of split constraints. The constraints introduced by Husemann et al. [7] are also a subclass of split constraints. *Path constraints* [1, 2] seem to achieve the goal of describing certain forms of heterogeneity in semistructured data. Path constraints characterize the existence of paths associated with sequences of labels in semistructured data. However, path constraints also lack the entire expressiveness needed to characterize summarizability, and do not describe well the type of heterogeneity arising in OLAP applications. In particular, we cannot characterize summarizability with them. On the other hand, path constraints are interpreted over data which have many fewer restrictions in their structure than OLAP dimensions, yielding to a different treatment and complexity of their inference.

We refer the reader to the full version of this paper [5] for a more detailed study of the related work mentioned in this section.

1.4 Contributions and Outline

In this paper, we propose a class of constraints, *dimension constraints*, for the purpose of expressing integrity constraints in dimension schemas. We show that the hierarchy schema enriched with dimension constraints becomes an adequate abstract model to infer summarizability. In particular, we show that summarizability can be characterized using dimension constraints, turning the problem of testing summarizability into an inference problem over dimension constraints. We give a sound and complete algorithm for solving the implication of dimension constraints based on the notion of *frozen dimensions*. Frozen dimensions are minimal homogeneous dimension instances representing the different structures that are implicitly "mixed up" in the schema. They are inferred from the dimension schema, and provide a useful representation to understand heterogeneous schemas. We propose an algorithm that uses heuristics based on at the structure of the dimension schema and the constraints to speed up its execution. Finally, we study the intrinsic complexity of the implication problem, as well as the running time of the algorithm proposed.

The remainder of this paper is organized as follows. In Section 2 we review the main concepts related to heterogeneous dimensions. Section 3 introduces dimension constraints, and their relation with summarizability. The implication problem related to dimension constraints is studied in Section 4. In Section 5 we present the algorithm for testing implication of dimension constraints. Finally, in Section 6 we conclude and outline some prospects for future work. The proofs are presented in the full version of the paper [5].

2. MODELING HETEROGENEOUS DIMENSIONS

In this section, we describe our framework for modeling heterogeneous dimensions. In our approach, the dimension schema consists of a hierarchy schema and a set of integrity constraints.

2.1 Hierarchy Schema

The hierarchy schema provides the skeleton upon which dimension instances are defined. We extend the hierarchy schema given in earlier dimension models to allow multiple bottom categories, cycles, and shortcuts. Assume the existence of a set of categories $\mathbf{C}$.

DEFINITION 1. *A* hierarchy schema *is a tuple* $G = (C, \nearrow)$*, where* $C \subseteq \mathbf{C}$ *is a finite set of categories with a distinguished category All;* $\nearrow$ *is a binary relation on* C *(*$\nearrow^*$ *stands for the transitive and reflexive closure of* $\nearrow$*). The following conditions hold: (a) for each category* $c \in C$*,* $c \nearrow^* \texttt{All}$*; (b) for every category* $c \in C$*, it is not the case that* $c \nearrow c$*.*

A *bottom category* of a hierarchy schema $G = (C, \nearrow)$ is a category c_b such that there is no category $c' \in C$ where $c' \nearrow c_b$. A *shortcut* of a hierarchy schema G is a pair of categories c and c' of C such that $c \nearrow c'$ and there is a path from c to c' in G passing through some third category c''.

EXAMPLE 3. *The categories City and Country form a shortcut in the dimension* `location` *of Figure 1.*

Cycles are needed in certain schemas, as the following example shows.

EXAMPLE 4. *Consider a dimension with the following categories: Store, SaleDistrict, City, and* `All`*. Suppose that some cities have ancestors in SaleDistrict, while some sale districts have ancestors in City. As we will see in the next section, an edge* (c_1, c_2) *in the hierarchy schema allows the members in* c_1 *to have ancestors in* c_2*. Therefore, in order to model this dimension, we need the cycle*

SaleDistrict City SaleDistrict

in the hierarchy schema.

2.2 Dimension Instance

An instance of a dimension is obtained by specifying a member set for each category, a child/parent relation between members, and a set of functions that assign values to category attributes. For simplicity, we will associate a single attribute, called `Name`, to every category of a dimension. The attribute `Name` will contain names for members; we will model `Name` as a function that maps members into a set of values $\mathbf{V}$. We assume the existence of a set of members $\mathbf{M}$.

DEFINITION 2. *A* dimension instance *is a tuple* $d = (G,$ `MembSet`$, <,$ `Name`$)$ *where* $G = (C, \nearrow)$ *is a hierarchy schema;* `MembSet` $: C \to 2^{\mathbf{M}}$ *assigns a set of members* $\texttt{MembSet}_c$ *to*

each category c in C (we denote by $\texttt{MembSet}_d$ the union of these sets); $< \subseteq \texttt{MembSet}_d \times \texttt{MembSet}_d$ is the child/parent relation *between members (where $\ll$ stands for the transitive closure of $<$); and finally, $\texttt{Name} : \texttt{MembSet}_d \rightarrow \mathbf{V}$ is a function that assigns values to members. The conditions depicted in Figure 2 hold.*

Condition **(C1)** says that the edges in the hierarchy schema represent inter-category links, that must exist whenever we have a child/parent relationship between some pair of members in the categories. Condition **(C2)** states that each member in a category reaches no more than one member in any category above it. A dimension satisfying this restriction is also referred to as being partitioned [6] or being *strict* [13]. Condition **(C2)** appears as an inherent constraint in most dimension models [3, 11]. Condition **(C3)** states that the member sets are pairwise disjoint. This condition avoids redundant aggregates in the datacube. Condition **(C4)** says that `all` is the only member in $\texttt{MembSet}_{\texttt{All}}$. Condition (**C5**) states that a member cannot be a parent and an indirect ancestor of another member at the same time; in other words, we cannot have shortcuts in the dimension instance. Condition (**C6**) essentially says that categories do not straddle the descendant/ancestor relation; this implies that $<$ is acyclic. Condition (**C7**) replaces the condition of homogeneity, and states that any member rolls up to at least one category directly above its category.

(C1) (Connectivity) For every pair of categories c, c', and for every pair of members $x \in \texttt{MembSet}_c, x' \in \texttt{MembSet}_{c'}$, if $x < x'$, then $c \nearrow c'$.

(C2) (Partitioning) For every pair of categories c and c', if there exists a member $x \in \texttt{MembSet}_c$, and a pair of members $x_1, x_2 \in \texttt{MembSet}_{c'}$ such that $x \ll x_1$ and $x \ll x_2$, then $x_1 = x_2$.

(C3) (Disjointness) The member sets are pairwise disjoint.

(C4) (Top Category Constraint) $\texttt{MembSet}_{\texttt{All}} = \{\texttt{all}\}$.

(C5) (Shortcuts) For every pair of members x_a, x_b such that $x_a \in \texttt{MembSet}_{c_a}$, $x_b \in \texttt{MembSet}_{c_b}$ and $x_a < x_b$, there are no members $x_1, \ldots, x_n \in \texttt{MembSet}_d$ such that $x_a < x_1 < \ldots x_n < x_b$.

(C6) (Stratification) For every category c and every pair of members $x, x' \in \texttt{MembSet}_c$ it is not the case that $x \ll x'$.

(C7) (Up Connectivity) For every category $c \in (C \setminus \texttt{All})$, and for every member $x \in \texttt{MembSet}_c$, there exist a category $c' \in C$ and a member $x' \in \texttt{MembSet}_{c'}$ such that $c' \nearrow c$ and $x < x'$.

Figure 2: Conditions that a dimension instance must satisfy.

For simplicity, in the `location` dimension (Figure 1), the function `Name` is the identity; so we do not show it in the figure.

The child/parent relation induces a partial order relation $\leq$ between members defined as follows: $x_1 \leq x_2$ iff $x_1 \ll x_2$ or $x_1 = x_2$. We say that a member x_1 rolls up to a member x_2 whenever $x_1 \leq x_2$. Also, we say that a member x rolls up to a category c whenever there exists a member $x' \in \texttt{MembSet}_c$ such that $x \leq x'$. The rollup mapping from a category c_1 to a category c_2 of a dimension d, denoted $\Gamma_{c_1}^{c_2} d$, is defined as follows:

$$\Gamma_{c_1}^{c_2} d = \{(x_1, x_2) \mid x_1 \in \texttt{MembSet}_{c_1} \wedge x_2 \in \texttt{MembSet}_{c_2} \wedge x_1 \leq x_2\}$$

Notice that Condition **(C2)** forces the rollup mappings to be single-valued.

3. DIMENSION CONSTRAINTS

In this section we introduce a class of constraints, *dimension constraints*, that will serve to augment hierarchy schemas to model heterogeneous dimension instances. A dimension constraint is a Boolean combination of two sorts of atoms: *path atoms* and *equality atoms*. All the path atoms of a dimension constraint start from a unique category called the *root* of the constraint. Path atoms allow us to specify Boolean conditions on the paths the ancestors of the members in the root form.

EXAMPLE 5. *Consider the* `location` *dimension (Figure 1). The following dimension constraint states that all the stores rollup to City:*

$$Store\_City$$

The root of this constraint is the category Store. The constraint consists of the single path atom Store_City.

Equality atoms specify the values in the attributes of the ancestors of the members in the root. The combination of equality and path atoms is required to describe dependencies between paths that start from the root members and values of attributes. For instance, we may want to express that the existence of a path in the child/parent relation from a member x is determined by the value in an attribute of an ancestor of x.

EXAMPLE 6. *Consider the* `location` *dimension (Figure 1). The following dimension constraint asserts that for every store x, if x rolls up to Canada, then x has a parent in City which has a parent in Province:*

$$Store.Country \approx Canada \supset Store\_City\_Province.$$

In this constraint, Store_City_Province is a path atom that asserts that there exist members $x_c \in \texttt{MembSet}_{City}$ and $x_p \in \texttt{MembSet}_{Province}$ such that $x < x_c$ and $x_c < x_p$. The expression Store.Country≈Canada is an equality atom *which asserts that (a) there exists an ancestor y of x in $\texttt{MembSet}_{Country}$, and (b) $\texttt{Name}(y) = Canada$.*

3.1 Dimension Constraint Language

We now formalize the language of dimension constraints. Assume the existence of a possibly infinite set of constants $\mathbf{K}$.

DEFINITION 3. *Given a hierarchy schema $G = (C, \nearrow, \texttt{Name})$ and a category $c \in C$, the atoms over G with root c are defined as follows:*

- *A* path atom *is an expression of the form $c\_c_1\_\ldots\_c_n$, where $cc_1 \ldots c_n$ is a simple path (a path without cycles) in G.*
- *An* equality atom *is an expression of the form $c.c_i{\approx}k$, where $c_i \in C$ and $k \in \mathbf{K}$.*

A dimension constraint *over G with root c, where $c \neq \texttt{All}$, is a Boolean combination of atoms over G with root c.*

We consider the usual connectives $\neg, \wedge, \vee, \supset, \equiv$, and $\oplus$ for exclusive disjunction. In addition, $\bot$ and $\top$ will denote the false and the true proposition, respectively. Finally, $\bigodot A$, where A is a set of atoms, will express that there is exactly one true atom in A.

In order to define the semantics of dimension constraints, we introduce a function $\mathcal{S}$ that maps each dimension constraint into a FOL formula that refers to the components of a dimension. Given a dimension constraint α, $\mathcal{S}(\alpha)$ is defined as follows:

- If α is a path atom of the form $c\_c_1\_\ldots\_c_n$, then $\mathcal{S}(\alpha)$ is the following FOL expression: $\exists x_1 \ldots \exists x_n(\texttt{MembSet}_{c_1}(x_1) \wedge x < x_1 \wedge \texttt{MembSet}_{c_2}(x_2) \wedge x_1 < x_2 \wedge \ldots \wedge \texttt{MembSet}_{c_n}(x_n) \wedge x_{n-1} < x_n)$.
- If α is an equality atom of the form $c.c_i{\approx}k$, then $\mathcal{S}(\alpha)$ is the following FOL expression: $\exists x_i(\texttt{MembSet}_{c_i}(x_i) \wedge x \leq x_i \wedge \texttt{Name}(x_i) = k)$.
- If α is not an atom, then $\mathcal{S}(\alpha)$ is obtained from α by replacing every atom β with $\mathcal{S}(\beta)$.

Notice that x is the only free variable in $\mathcal{S}(\alpha)$. We can observe that an equality atom of the form $c.c{\approx}k$ basically means $\texttt{Name}(x) = k$; we will abbreviate an atom of this form as $c{\approx}k$.

DEFINITION 4. *Given a dimension instance $d = (G, \texttt{MembSet}, <, \texttt{Name})$ and a dimension constraint α with root c, d satisfies α, denoted $d \models \alpha$, if $\forall x(x \in \texttt{MembSet}_c \supset \mathcal{S}(\alpha))$ is true when the symbols $<$, MembSet, and Name that appear in $\mathcal{S}(\alpha)$, are interpreted as their corresponding elements in d.*

A *composed path atom* is an expression of the form $c.c_i$. A composed path atom is a shorthand for the following expression:

- If $c = c_i$, $c.c_i$ represents $\top$;
- else, $c.c_i$ represents the disjunction of all the path atoms with root c that end with c_i.

Intuitively, the atom $c.c_i$ expresses that x rolls up to c_i.

EXAMPLE 7. *Consider the dimension* location *(Figure 1). The dimension constraint Store.SaleRegion asserts that all the stores rollup to SaleRegion.*

Finally, a *dimension schema* is a tuple $ds = (G, \Sigma)$ where G is a hierarchy schema, and Σ is a set of dimension constraints over G. A dimension instance d is over a dimension schema $ds = (G, \Sigma)$ if the hierarchy schema of d is G, and $d \models \Sigma$. We denote by $\mathcal{I}(ds)$ the set of dimension instances over ds. A dimension schema ds *logically implies* a dimension constraint α, written $ds \models \alpha$, if every dimension instance d over ds satisfies α. The *implication problem* for dimension constraints, is the problem of determining, given a dimension schema ds and a dimension constraint α, whether $ds \models \alpha$.

EXAMPLE 8. *The dimension schema* locationSch*, depicted in Figure 3, models the* location *dimension (Figure 1). Notice that* locationSch *makes use of equality atoms to differentiate the structure of the stores in each country of* location*. Moreover,* locationSch *models the shortcut caused by Washington.*

3.2 Frozen Dimensions

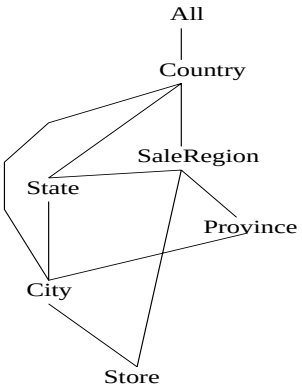
In essence, *frozen dimensions* are minimal homogeneous dimension instances conveyed by a dimension schema. The notion of the frozen dimension is essential for the algorithm we propose to test implication of dimension constraints.

Assume the existence of an injective function $\phi : \mathbf{C} \to \mathbf{M}$ that maps each category to a member. Moreover, given a dimension schema $ds = (G, \Sigma)$, $Const_{ds} : C \to \mathbf{K}$ is a function that assigns to each category c of ds the set of all constants k such that Σ contains an equality atom of the forms $c_i.c{\approx}k$ or $c{\approx}k$. We also assume the existence of a constant nk which is not mentioned in Σ.

DEFINITION 5. *Given a dimension schema ds, and a category c from it, a frozen dimension of ds with root c is a dimension instance $f = (G, \texttt{MembSet}, <, \texttt{Name}) \in \mathcal{I}(ds)$, such that the following hold: (a) $\texttt{MembSet}_c = \{\phi(c)\}$; (b) each category $c' \in C$ has at most the member $\phi(c')$ in $\texttt{MembSet}_{c'}$; (c) for every member $x \in \texttt{MembSet}_f$ such that $x \neq \phi(c)$, $\phi(c) \ll x$; and (d) for every category c' such that $\texttt{MembSet}_{c'} \neq \emptyset$, $\texttt{Name}(\phi(c')) \in Const(c') \cup \{\texttt{nk}\}$.*

Intuitively, whenever nk appears in $\texttt{Name}(\phi(c))$, it represents the set of constants $\{k \in \mathbf{K} \mid k \notin Const(c)\}$.

Frozen dimensions tell us a great deal about the semantics of dimension schemas, as the following example shows.



(a) $Store\_City$
(b) $Store.SaleRegion$
(c) $City{\approx}Washington \equiv City\_Country$
(d) $City{\approx}Washington \supset City.Country{\approx}USA$
(e) $State.Country{\approx}Mexico \vee State.Country{\approx}USA$
(f) $State.Country{\approx}Mexico \equiv State\_SaleRegion$
(g) $Province.Country{\approx}Canada$

Figure 3: The dimension schema `locationSch`.

EXAMPLE 9. *Figure 4 depicts the frozen dimensions of* `locationSch` *with root Store. Intuitively, this set illustrates the different structures of the stores in Mexico, USA, and Canada. The figure shows the subgraphs of the hierarchy schema induced by the non-empty categories of each frozen dimension. Also, we depict the value given by* `Name` *whenever the category has associated some constant different than* `nk` *in* $Const_{\texttt{locationSch}}$ *(categories City and Country).*

3.3 Dimension Constraints and Summarizability

In this section, we give a characterization of summarizability in terms of dimension constraints. We thus reduce testing summarizability to testing implication of dimension constraints.

A single-category cube view can be specified as $\texttt{CubeView}(d, F, c, \texttt{af}(m))$, where d is a dimension; F is a fact table containing facts at the base category c_b of d ($\texttt{MembSet}_{c_b}$ contains the members in the bottom categories); c is a category of d; `af` is an aggregate function; and m is a measure of F. The cube view $\texttt{CubeView}(d, F, c, \texttt{af}(m))$ represents the following aggregate view: $\Pi_{c,\texttt{af}(m)}(F \bowtie \Gamma^c_{c_b} d)$.

DEFINITION 6. *Given a dimension instance d, a set of categories $S = \{c_1, \ldots, c_n\}$, and a category c, c is* summarizable *from S in d iff for every fact table F, and distributive aggregate function* `af`*, we have:* $\texttt{CubeView}(F, d, c, \texttt{af}(m)) = \Pi_{c,\texttt{af}^c(m)}(\biguplus_{i \in 1 \ldots n}(\pi_{c,m}\Gamma^c_{c_i} d \bowtie \texttt{CubeView}(F, d, c_i, \texttt{af}(m))))$.

In order to characterize summarizability, we will use the shorthand $c.c_i.c_j$, where c, c_i, and c_j are categories. Formally, $c.c_i.c_j$ is defined as follows:

- If $c \neq c_i \neq c_j$ then $c.c_i.c_j$ represents the disjunction of all the path atoms that start with c, end with c_j, and contain c_i.
- If $c = c_i = c_j$ then $c.c_i.c_j$ represents $\top$.
- If $c = c_j$ and $c, c_j \neq c_i$ then $c.c_i.c_j$ represents $\bot$.
- If $c = c_i$ and $c, c_i \neq c_j$ then $c.c_i.c_j$ represents $c_1.c_j$.
- Finally, if $c \neq c_i, c_j$ and $c_i = c_j$ then $c.c_i.c_j$ represents $c.c_i$.

Intuitively, $c.c_i.c_j$ means that x rolls up to c_j passing through c_i. Now, we present the main result of the section.

THEOREM 1. *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 $d \models c_b.c \supset \bigodot_{c_i \in S} c_b.c_i.c$.*

The intuition behind Theorem 1 is that, in order for c to be summarizable from S, we need that every base member (i.e., a member in a bottom category) that rolls up to c, rolls up to c passing trough one and only one of the categories in S. Notice that Theorem 1 shows that summarizability can be characterized as a property of dimension instances themselves, avoiding the mention of fact tables.

EXAMPLE 10. *We have that Country is summarizable from $\{City\}$ in* `location` *(Figure 1) because*

$$\texttt{location} \models Store.Country \supset Store.City.Country.$$

However, Country is not summarizable from $\{State, Province\}$ in `location` *because*

$$\texttt{location} \not\models Store.Country \supset (Store.State.Country \oplus Store.Province.Country).$$

This is because the stores that belong to Washington rollup directly to Country without passing through states or provinces.

4. IMPLICATION

Before tackling the implication problem, we will investigate *satisfiability* in our setting. A dimension schema ds is satisfiable if $\mathcal{I}(ds) \neq \emptyset$.

PROPOSITION 1. *Every dimension schema is satisfiable.*

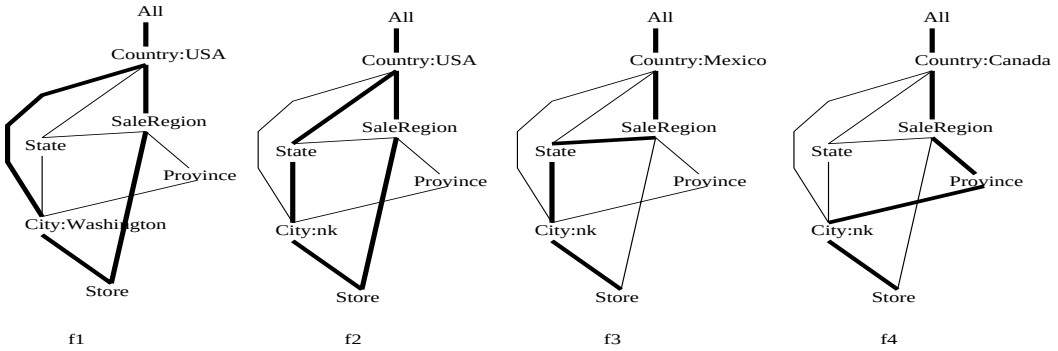


Figure 4: Frozen dimensions of `locationSch` with root *Store*.

Proposition 1 follows from the fact that there are no dimension constraints with root `All`. Therefore, given a dimension schema ds, we can always build a dimension instance over ds with a unique member `all` in the category `All`.

A category c is said to be *satisfiable* in a schema ds (we assume that c is a category of ds) iff there exists a dimension instance $d \in \mathcal{I}(ds)$ such that $\texttt{MembSet}_c \neq \emptyset$.

EXAMPLE 11. *Suppose we add the constraint*

$$\neg SaleRegion\_Country$$

to `locationSch`. *Then, SaleRegion would become unsatisfiable in the resulting schema, because condition* **(C7)** *of Definition 2 requires SaleRegion_Country.*

The *category satisfiability* problem is the problem of determining, given a dimension schema ds and a category c of ds, whether c is satisfiable in ds. Unsatisfiable categories can be dropped from the schema, providing a cleaner representation of the data. However, the fundamental importance of testing category satisfiability is its connection with testing implication.

THEOREM 2. *Given a dimension schema $ds = (G, \Sigma)$ and a dimension constraint α with root c, $ds \models \alpha$ iff c is unsatisfiable in $ds' = (G, \Sigma \cup \{\neg\alpha\})$.*

In view of Theorem 2, implication reduces to category satisfiability. Next, we show that frozen dimensions are minimal models for testing category satisfiability.

THEOREM 3. *Given a dimension schema ds and a category c belonging to it, c is satisfiable in ds iff there exists a frozen dimension of ds with root c.*

Given a dimension schema $ds = (G, \Sigma)$ and a category c, a candidate frozen dimension of ds with root c can be built by first choosing a subgraph of G, and then selecting the constants for `Name` using the function *Const*. The number of candidate frozen dimensions generated in this way is finite, and the test of whether one of them is a frozen dimension can be done in polytime. Consequently, Theorem 3 establishes an algorithm to solve category satisfiability.

THEOREM 4. *Category satisfiability is NP-Complete.*

The NP-hardness of category satisfiability follows from a straightforward reduction from SAT. The proof of membership in NP uses Theorem 3. In addition, from Theorem 2 and Theorem 3, it follows that testing implication of dimension constraints is CoNP-complete.

5. THE DIMSAT ALGORITHM

In this section, we provide an algorithm, called `DIMSAT`, to solve category satisfiability efficiently.

In order to describe the algorithm we need to introduce the notion of *subhierarchy*.

DEFINITION 7. *Given a hierarchy schema $G = (C, \nearrow)$, a* subhierarchy *of G with root c is a pair $(C', \nearrow')$ such that (a) $C' \subseteq C$ and $\nearrow' \subseteq \nearrow$; (b) $c, \texttt{All} \in C'$; and (c) for all categories $c' \in C'$, $c \nearrow'^{*} c'$ and $c' \nearrow'^{*} \texttt{All}$.*

Given a dimension schema $ds = (G, \Sigma)$, and a subhierarchy g of G, we say that g *induces a frozen dimension* in ds iff there exists a frozen dimension f of ds such that g is the graph obtained from the child/parent relation $<$ of f by renaming every member x in $\texttt{MembSet}_f$ with $\phi^{-1}(x)$.

The algorithm `DIMSAT` builds subhierarchies and tests whether each of them induces at least one frozen dimension in the dimension schema given. When a subhierarchy is built, each path atom p in the constraints is replaced by a truth value given by whether p appears in the subhierarchy; the equality atoms over categories that do not appear in the subhierarchy are replaced by $\bot$. In this form, Σ is reduced to a set of constraints that do not mention path atoms. This set is then tested over the candidate frozen dimensions induced by the subhierarchy. In addition, the algorithm prunes the subhierarchies to be explored by taking into account shortcuts, cycles, and *into* constraints. *Into* constraints are dimension constraints of the form c_c'; intuitively, an *into* constraint

states that all the members of c have a parent in c'. We conjecture that this optimization should have a major impact in practice, since we will frequently have heterogeneity arising as an exception, having most of the edges of the schema associated with *into* constraints.

The following definition is useful, as we wish to discard the constraints in Σ that are irrelevant when finding a frozen dimension. Given a dimension schema $ds = (G, \Sigma)$, and a category c of ds, $\Sigma(ds, c)$ is the set containing the dimension constraints α of Σ such that the root c' of α satisfies $c \nearrow^* c'$.

The `DIMSAT` algorithm uses a procedure called `CHECK`, that tests whether a subhierarchy induces a frozen dimension. The main idea behind `CHECK` is as follows: when a subhierarchy g is built, all the path atoms that appear in the dimension expression $\Sigma(ds, c)$ are replaced by their truth values in g. Doing this, $\Sigma(ds, c)$ is turned into a dimension expression that mentions only equality atoms that refer to the categories in the subhierarchy. In order to test whether a candidate frozen dimension f built over g is a frozen dimension, we need only to test whether the assignment of constants to categories in f satisfies $\Sigma(ds, c)$. In this form, we evaluate the path atoms (and some of the equality atoms as well) only once for all the candidate frozen dimension built over the same subhierarchy.

We next define the circle operator, that replaces the truth value of each path atom p in a set of dimension constraints, according to whether p exists in a given subhierarchy.

DEFINITION 8. *Given a set of dimension constraints Σ, and a subhierarchy g of G, $\Sigma \circ g$ is the set of dimension constraints resulting from Σ by: (a) renaming every path atom p with $\top$ if p is a path in g, and with $\bot$ otherwise; and (b) renaming every equality atom $c_i.c_j \approx k$, such that there is no path from c_i to c_j in g, with $\bot$.*

EXAMPLE 12. *The dimension constraints $\Sigma($`locationSch`, Store$)$ are depicted in Figure 5 (left). Now, let g be the subhierarchy represented as f_2 in Figure 3. The dimension constraints $\Sigma($`locationSch`, Store$) \circ g$ are depicted in Figure 5 (right).*

Notice that the dimension constraints $\Sigma(ds, c) \circ g$ contain only equality atoms. Now, given a dimension schema $ds = (G, \Sigma)$ and a subhierarchy $g = (C', \nearrow')$ of G, a c-assignment for g is a injective function $\texttt{ca} : C' \rightarrow \mathbf{K} \cup \{\texttt{nk}\}$ such that for all $c' \in C'$, $\texttt{ca}(c') = k$ implies that $k \in Const_{ds}(c')$. Intuitively, a c-assignment selects one constant in $Const(c')$ for each category c' in the subhierarchy. We say that a c-assignment `ca` satisfies a set of dimension constraints Σ that mention only equality atoms, denoted $\texttt{ca} \models \Sigma$, if Σ is true when we replace each equality atom in Σ with its truth value given by `ca`. For example, if an equality atom p is $c.c_i \approx k$, and we have that $\texttt{ca}(c_i) = k$ then we replace p with $\top$.

PROPOSITION 2. *Given a dimension schema $ds = (G, \Sigma)$, and a subhierarchy g of G with root c, g induces a frozen dimension iff (a) g has no cycles or shortcuts, and (c) there exists a c-assignment `ca` of g such that $\texttt{ca} \models \Sigma(ds, c) \circ g$.*

Algorithm `DIMSAT`(ds, c)
Input: A dimension schema $ds = (G, \Sigma)$ and a category $c \in C$.
Output: Whether c is satisfiable in ds.
(1) `FIND` := *false*, $Pr := \Sigma(c, ds)$
(2) $g.C := \{c\}$, $g.Out(c) := \emptyset$, $g.Top := \{c\}$, $g.In^*(c) := \emptyset$
(3) `EXPAND`$(g, c, \emptyset)$
(4) *return*(`FIND`)
end `DIMSAT`

Procedure `CHECK`(g)
Input: A subhierarchy g of G
Local Vars: Pr', `ca`
Global Vars: `FIND`
(1) $Pr' := Pr \circ g$
(2) For every c-assignment `ca` of g do
(3) `FIND` := $(\texttt{ca} \models Pr')$
(4) If `FIND` then return()
(5) endFor
end `CHECK`

Procedure `EXPAND`(g, c, R)
Input: a category c, and a list of categories R
Local Vars: *ctop*, Ss, Sc, S, P, S'
Global Vars: G, `FIND`
(1) If $R \neq \emptyset$ then
(2) $g.Top := (g.Top \setminus \{c\}) \cup (R \setminus g.C)$
(3) $g.C := g.C \cup R$; $g.Out(c) := R$
(4) For every $c' \in R$ do $g.In^*(c') := g.In^*(c)$
(5) EndIf
(6) If $Top = \{\texttt{All}\}$ then
(7) `CHECK`(g)
(8) If `FIND` then exit() else return()
(9) EndIF
(10) Choose a category $ctop \neq \texttt{All} \in g.Top$
(11) $Ss := \{c' \in G.Out(ctop) \mid g.In(c') \cap g.In^*(ctop) \neq \emptyset\}$
(12) $Sc := G.Out(ctop) \cap g.In^*(ctop)$
(13) $S := G.Out(ctop) \setminus (Ss \cup Sc))$
(14) $Into := \{c' \in G.Out(ctop) \mid ctop\_c' \in \Sigma\}$
(15) If $((Into \not\subseteq S)$ or $(S = \emptyset))$ then return()
(16) For every non-empty set $S' \subseteq (S \setminus Into)$ do
(17) `EXPAND`$(g, ctop, S' \cup Into)$
(18) endFor
end `EXPAND`

Figure 6: Algorithm `DIMSAT`.

$\Sigma(\texttt{locationSch}, Store)$	$\Sigma(\texttt{locationSch}, Store) \circ g$
(a) $Store\_City$	(a) $\top$
(b) $Store.SaleRegion$	(b) $\top$
(c) $City \approx Washington \equiv City\_Country$	(c) $City \approx Washington \equiv \bot$
(d) $City \approx Washington \supset City.Country \approx USA$	(d) $City \approx Washington \supset City.Country \approx USA$
(e) $State.Country \approx Mexico \vee State.Country \approx USA$	(e) $State.Country \approx Mexico \vee State.Country \approx USA$
(f) $State.Country \approx Mexico \equiv State\_SaleRegion$	(f) $State.Country \approx Mexico \equiv \bot$
(g) $Province.Country \approx Canada$	(g) $Province.Country \approx Canada$

Figure 5: (Left) $\Sigma(\texttt{locationSch}, Store)$**. (Right)** $\Sigma(\texttt{locationSch}, Store) \circ g$**.**

We are now able to introduce the DIMSAT algorithm. DIMSAT, depicted in Figure 6, is basically a backtracking algorithm that explores subhierarchies. The procedure EXPAND constructs subhierarchies of G with root c, that have no cycles or shortcuts and satisfy the into constraints given in Σ. When one of such subhierarchies g is built, EXPAND calls CHECK(g) to decide whether g induces a frozen dimension. If so, CHECK makes FIND = *true*, and EXPAND exits, aborting all previous calls to EXPAND, and returning the control of the execution to DIMSAT. If not, EXPAND returns, and backtracks to a previous state in the search; we assume that when this occurs, g is restored to the form it had before EXPAND was called.

Let us now explain some aspects of EXPAND. The subhierarchy being built is kept in the variable g, which has four components: $g.C$, containing the categories of g; $g.Out$, which contains for every category $c' \in g.C$, the categories directly above c' in g; $g.Top$, which has the categories in $g.C$ with no edges from them in g; and $g.In^*$, which keeps for every category $c' \in g.C$, the categories that reach directly or indirectly c' in g. As we will see, $g.In^*$ is essential for recognizing shortcuts. In each step in the recursion, EXPAND is called with parameters g, c, and R, where c is a category, and R is a set of categories. Initially, EXPAND is called by DIMSAT with $R = \emptyset$; in this case $\{c\}$ is kept as $g.Top$. In an execution of EXPAND, Line (6) detects whether $g.Top = \{\texttt{All}\}$. If so, CHECK(g) is called. If not, EXPAND chooses a top category $ctop \in g.Top$, and tries all possible calls EXPAND(g, c, R), where R is any combination of categories directly above *ctop* in G such that the following hold: R does not produce shortcuts or cycles (note that the categories that potentially cause shortcuts and cycles are computed in lines (11) and (12), respectively); and R contains all categories c' such that the *into* constraint $ctop\_c'$ is in Σ. In this form, EXPAND takes into account the *into* constraints in order to prune the subhierarchies to be explored, and shortens the loop of Line (16).

EXAMPLE 13. *Consider the execution of*

DIMSAT(locationSch, $Store$).

Figure 7 shows g in the successive instances of EXPAND*. The subhierarchy g with which* EXPAND *calls* CHECK *the first time is delimited by a box. Notice that $g.Top$ is the category written with a large font in each subgraph.*

PROPOSITION 3. *Every execution of* DIMSAT(ds, c) *terminates, and correctly outputs whether c is satisfiable in ds.*

We end this section by giving the asymptotic time complexity of DIMSAT. Let N be the number of categories in ds, and let N_K be the maximum number of constant $Const_{ds}$ may assign to a category. In addition, N_Σ stands for the size of Σ.

PROPOSITION 4. DIMSAT *runs in time* $O(2^{N^2 + N \log N_K} N^3 N_\Sigma)$.

Notice that if the dimension schema does not have equality atoms, $N_K = 1$, so DIMSAT runs in time $O(2^{N^2} N^3 N_\Sigma)$.

6. CONCLUSION

Dimension constraints have a practical motivation, can express summarizability, and have a relatively efficient inference problem (CoNP-complete) compared with other classes of path-like constraints that have been studied. Moreover, from the study of the running time of DIMSAT given in the full version of this paper [5], we conjecture that in most practical situations DIMSAT should yield execution times of the order of a few seconds. We believe these properties should make dimension constraints useful in a broad set of practical settings.

Although the first and most direct motivation for introducing dimension constraints is to support aggregate navigation, they are also helpful in the design stage of data cubes. As in traditional database systems, the design of dimensions for OLAP should be driven by the semantic information provided in the schema. Dimension constraints provide the means to capture such semantic information. In addition, dimension constraints may play an important role in the problem of selecting views to materialize in data cubes by supplying meta-data to support the test of whether a selected set of views is sufficient to compute all the required queries.

Dimension constraints can be extended in several directions. We could consider further built-in predicates over attributes, such as an order relation, to extend equality atoms. We would then be able to express dependences such as: "if the value of the price of a product is less than a given amount, the product rolls up to some particular path in the hierarchy schema". In addition, if we relax the partitioning constraint, summarizability can no longer be characterized with dimension constraints. Further extensions to dimension constraints are needed to support summarizability inference and aggregate navigation in such dimensions.

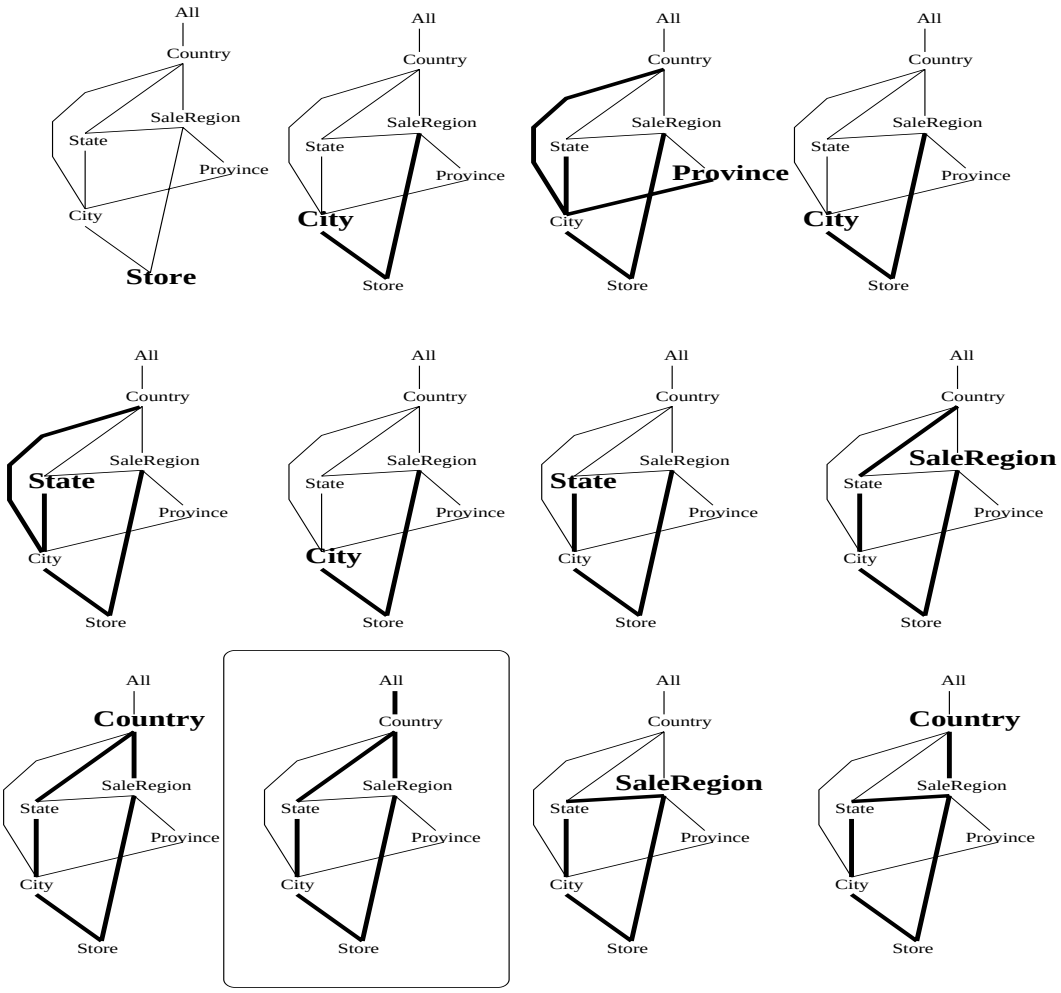


Figure 7: The variable g in an execution of DIMSAT(locationSch, $Store$).

Acknowledgments

This research was supported by the Natural Science and Engineering Research Council and the Institute for Robotics and Intelligent Systems of Canada. We thank Renée Miller, Ken Sevcik, and the anonymous reviewers for their fruitful suggestions.

7. REFERENCES

[1] S. Abiteboul and V. Vianu. Regular path queries with path constraints. In *Proceedings of the 16th ACM Symposium on Principles of Database Systems*, Tucson, Arizona, USA, 1997.

[2] P. Buneman, W. Fan, and W. S. Path constraints on semistructured and structured data. In *Proceedings of the 17th ACM Symposium on Principles of Database Systems*, Seattle, Washington, USA, 1998.

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

[4] B. A. Goldstein. Constraints on null values in relational databases. In *Proceedings of the 7th International Conference on Very Large Data Bases*, Cannes, France, 1981.

[5] C. Hurtado and A. Mendelzon. OLAP dimension constraints (extended version). In *ftp.db.toronto.edu/pub/papers/fullpods02.ps.gz*.

[6] C. Hurtado and A. Mendelzon. Reasoning about summarizability in heterogeneous multidimensional schemas. In *Proceedings of the 8th International Conference on Database Theory*, London, UK, 2001.

[7] B. Huseman, J. Lechtenborger, and G. Vossen. Conceptual data warehouse design. In *Proceedings of the International Workshop on Design and Management of Data Warehouses (DMDW)*, Stockholm, Sweden, 2000.

[8] 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.

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

[10] R. Kimball. *The Data Warehouse Toolkit*. J.Wiley and Sons, Inc, 1996.

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

[12] H. J. Lenz and A. Shoshani. Summarizability in OLAP and statistical databases. In *Proceedings of the 9th SSDBM Conference*, Olympia, Washington, USA, 1997.

[13] T. B. Pedersen and C. S. Jensen. Multidimensional data modeling for complex data. In *Proceedings of the 15th IEEE International Conference on Data Engineering*, Sydney, Australia, 1999.

[14] T. B. Pedersen, C. S. Jensen, and D. 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.