Capturing Summarizability with Integrity Constraints in OLAP

Carlos A. Hurtado
Universidad de Chile
churtado@dcc.uchile.cl

Claudio Gutiérrez
Universidad de Chile
cgutierr@dcc.uchile.cl

Alberto 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 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 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 *structural 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 [15, 12, 17, 9] have dropped the homogeneity restriction over the past few years, yielding *structural heterogeneous* dimensions, which are needed to represent more naturally and cleanly many practical situations. In addition, 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*

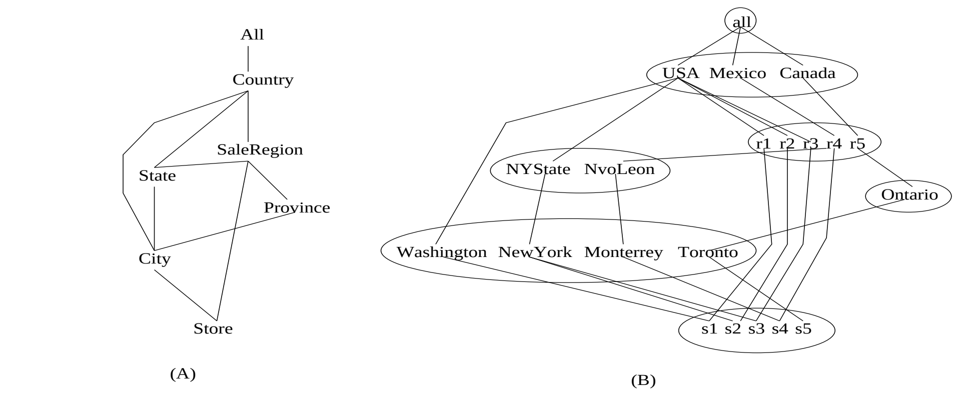


Figure 1. The dimension `location`: (A) hierarchy schema; (B) child/parent relation.

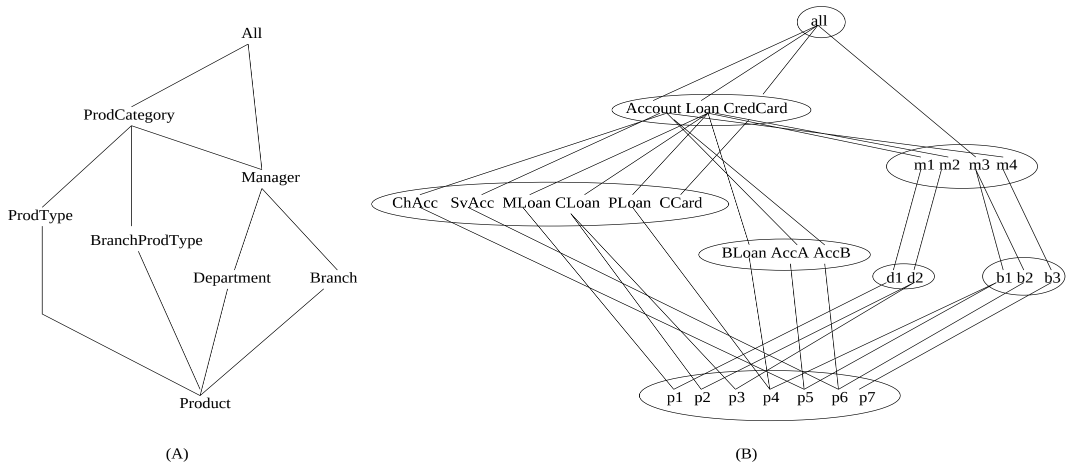


Figure 2. The dimension `product`: (A) hierarchy schema; (B) child/parent relation.

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 $SaleRegion$.

Example 2 *Figure 2 depicts a heterogeneous dimension, called* `product`*, which models financial services offered by a bank, such as: accounts, credit cards, loans. 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 $ProductCategory$.*

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 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.

[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`, $\texttt{af}^c = \texttt{af}$.

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* [13]. The notion of summarizability was introduced to study aggregate navigation in statistical objects and OLAP dimensions [16, 15, 17, 9]. 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 [9] 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 3 *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 [14] 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. [15], and Pedersen and Jensen [17] 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.

The dimension model of Jagadish et al. [12] allows several bottom categories where members may be placed, which intuitively allows such members to have ancestors in different sets of categories. In this model, the heterogeneity of a schema can be only modeled by splitting the categories of the schema, which may increase exponentially the number of categories, and may impose unnatural restriction on tha way members are grouped into categories. Their model is subsumed by the model we present in this paper.

Pedersen and Jensen [18] 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 some dimensions, we may need to place several different nulls in some categories, which leads to a considerable waste of memory and computational effort due to the increased sparsity of the cube views. As an example, Figure 3 shows the dimension resulting from an attempt to transform `location` (Figure 1) by inserting null members. Notice that the rollup mapping $\Gamma_{Province}^{SaleRegion}$ becomes a many-to-many relation, which limits summarizability in the dimension.

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 [9], we introduced *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 insufficient 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 [6] proposes to capture heterogeneity in database relations by means of

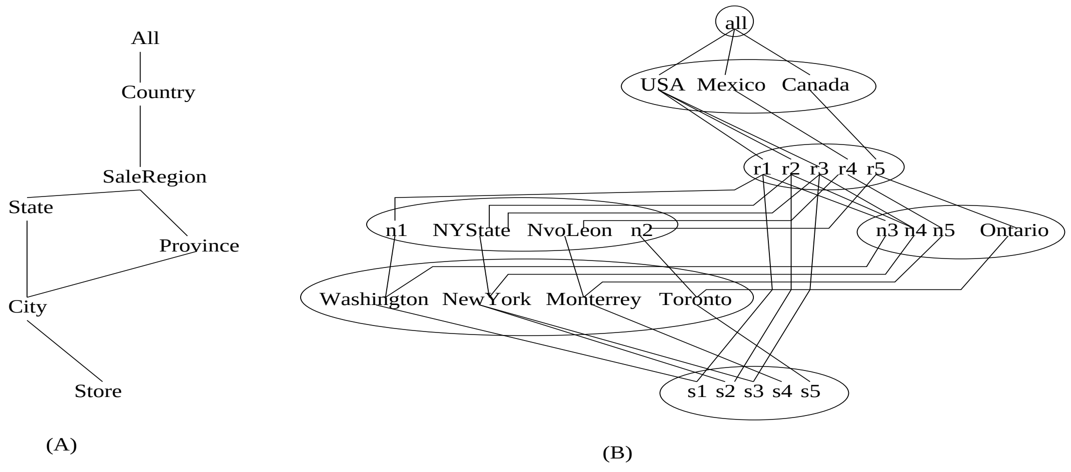


Figure 3. An attempt to fill missing parents with null values in the dimension `location`: (A) hierarchy Schema; (B) child/parent relation.

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. [11] are also a subclass of split constraints. *Path constraints* [1, 4] 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.

In Section 7, we present a more detailed study of the related work mentioned in this section.

1.4 Contributions

In this paper, we introduce a model for heterogeneous dimensions. The model, formalized with graph-theoretic notions, yields a new approach to represent the hierarchical structure of dimensions.

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 the structure of the dimension schema and the constraints to speed up its execution. We study the intrinsic complexity of the implication problem, and the running time of the algorithm proposed with experiments.

Finally, we study of the relationship between dimension constraints and other known classes of integrity constraints presented in the database literature.

1.5 Outline

The remainder of this paper is organized as follows. In Section 2 we present a model for heterogeneous dimensions, and formalize cube views and the notion of summarizability. Section 3 introduces dimension constraints, and dimension schemas. The implication problem related to dimension constraints is studied in Section 4. The relationship between dimension constraints and summarizability is shown in Section 5. In Section 6 we present the algorithm for testing implication of dimension constraints and its implementation. In Section 7 we compare dimension constraints with other known classes of integrity constraints. Finally, in Section 8 we conclude and outline some prospects for future work.

The proofs are presented in appendices.

2 Modeling Heterogeneous Dimensions

In this section, we give formalize heterogeneous dimensions. We define summarizability and its essential properties.

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 $\phi : V_1 \to V_2$ 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).

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* $d : (M, <) \to (C, \nearrow)$ *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 [5, 10, 12, 15], 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 Summarizability

We will formalize summarizations using relational algebra with bag semantics extended with the *generalized projection operator* [7, 2], to express aggregation. Besides the usual operators ($\sigma, \bowtie, \times$, etc.), the algebra includes the *additive union* $\uplus$ which adds the multiplicity of the tuples. The generalized projection operator, Π_A, is an extension of the duplicate-eliminating projection, where A can include both regular and aggregate attributes.

Given a dimension d, we assume the existence of distinguish category c_{base}, called *base category*, which contains all the members that are in the bottom categories of d. For every category for every category c of d we have:

$$\Gamma_{c_{base}}^{c}(d) = \biguplus_{\text{Every bottom category } c_b \text{ of } d} \Gamma_{c_b}^{c_i}(d).$$

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

Our definition of summarizability is based on the equivalence of two queries, the cube view and the summarization.

Definition 2 (Summarizability) *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* $\texttt{af}$*, we have:* $\texttt{CubeView}_{c,\texttt{af}(m)}(F, d) = \Pi_{c,\texttt{af}^c(m)}(\biguplus_{i \in 1\ldots n}(\pi_{c,m}\Gamma^{c}_{c_i} d \bowtie \texttt{CubeView}_{c_i,\texttt{af}(m)}(F, d_i)))$.

The following proposition gives a characterization of summarizability that avoids the mention of fact tables.

Proposition 1 (Summarizability) *A category c is summarizable from a set of categories S in a dimension instance d iff $\Gamma^{c}_{c_{base}} = \biguplus_{c_i \in S} \pi_{c_{base},c}(\Gamma^{c_i}_{c_{base}}(d) \bowtie \Gamma^{c}_{c_i}(d))$.*

The next corollary follows from Proposition 1.

Corollary 1 (Summarizability and Bottom Categories) *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: $\Gamma^{c}_{c_b}(d) = \biguplus_{c_i \in S} \pi_{c_b,c}(\Gamma^{c_i}_{c_b}(d) \bowtie \Gamma^{c}_{c_i}(d))$.*

The corollary easily follows from Proposition 1, and definition of $\Gamma^{c_i}_{c_{base}}$ for a category c.

Example 4 *Consider the dimension* `product` *depicted in Figure 2. In this dimension, $ProdCategory$ is summarizable from*

$$\{BranchProdType, Department\}.$$

However, in the dimension `product`*, $ProdCategory$ is not summarizable from*

$$\{ProdType, Department\}$$

because we would twice add to $ProdCategory$ the sales of products that rollup to $ProdCategory$ passing through $ProdType$ and $Department$ at the same time.

By making $|S| = 1$ in Corollary 1, we have that a category c is summarizable from a single category c_1 in a dimension d iff for every bottom category c_b of d we have $\Gamma^{c}_{c_b}(d) = \pi_{c_b,c}(\Gamma^{c_1}_{c_b}(d) \bowtie \Gamma^{c}_{c_1}(d))$.

3 Dimension Constraints

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

3.1 Dimension Constraint Language

Definition 3 (Dimension Constraint) *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.*

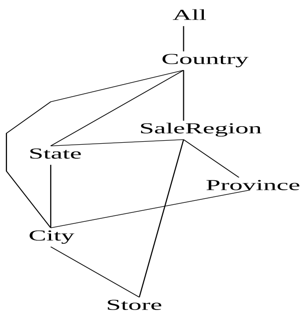
Dimension constraints consider the usual connectives $\neg, \wedge, \vee, \Rightarrow, \Leftrightarrow$, and $\oplus$ for exclusive disjunction. As usual, $\bot$ (resp. $\top$) will denote the false (resp. true) proposition. In addition, given a set of atoms A, $\bigodot_A$ denotes that there is exactly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

Definition 4 (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 $d(x_i) \in c_i$.*



All
Country
SaleRegion
State
Province
City
Store
(a) $\langle Store, City\rangle$
(b) $\langle Store, .., SaleRegion\rangle$
(c) $\neg\langle City, State\rangle \vee \langle City, Province\rangle$

Figure 4. The dimension schema `locationSch`$_\texttt{p}$.

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.*

A *composed path atom* is an expression of the form $\langle c, .., c_i\rangle$ which is a shorthand for the following expression: if $c = c_i$, $\langle c, .., c_i\rangle$ represents $\top$; else, $\langle c, .., c_i\rangle$ represents the disjunction of all the path atoms with root c that end with c_i. Intuitively, the atom $\langle c, .., c_i\rangle$ expresses that every root member roll up to c_i.

Example 5 *Consider the dimension* `location` *(Figure 1). The dimension constraint*

$$\langle Store, .., SaleRegion\rangle$$

asserts that all the stores rollup to $SaleRegion$.

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 $d \models \Sigma'$.

3.2 Dimension Schema

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

Definition 5 (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 $I(D)$.*

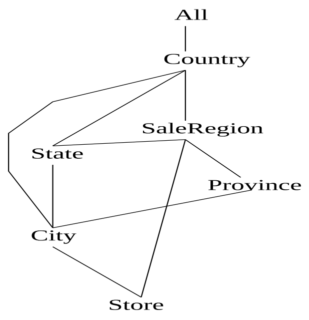
We shall now introduce some examples of dimension schemas. Our first schema, `locationSch`$_\texttt{p}$, provides an abstract model for $location$ (Figure 1), and is depicted in Figure 4. Notice that the constraint (a) of `locationSch`$_\texttt{p}$ is an *into* constraint.

The next schema we introduce, `locationSch`, makes use of equality atoms to differentiate the structure of the stores in each country of $location$. This schema is depicted in Figure 5.

Finally, we give a dimension schema, `productSch`, that models the product dimension of Figure 2. This schema is depicted in Figure 6.

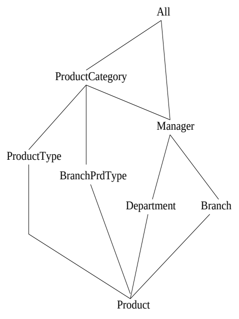
We end this section by investigating *satisfiability* in our setting. Formally, we say that a dimension schema D is satisfiable if $I(D) \neq \emptyset$.

Proposition 2 (Satisfiability) *Every dimension schema is satisfiable.*



(a) $\langle Store, City\rangle$
(b) $\langle Store, .., SaleRegion\rangle$
(c) $\langle City = Washington\rangle \equiv \langle City, Country\rangle$
(d) $\langle City = Washington\rangle \Rightarrow \langle City.Country = USA\rangle$
(e) $\langle State, .., Country = USA\rangle \vee \langle State, .., Country = Mexico\rangle$
(f) $\langle State, .., Country = Mexico\rangle \equiv \langle State, SaleRegion\rangle$
(g) $\langle Province, .., Country = Canada\rangle$

Figure 5. The dimension schema `locationSch`**.**



(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) $\langle Branch = b_3\rangle \Leftrightarrow \langle Branch, Manager, ProdCategory\rangle$

Figure 6. The dimension schema `productSch`**.**

3.3 Classes of Dimension Schemas

The model we have presented subsumes the dimension models presented in the literature. The following definition formalizes two classes of dimension schemas that arise in OLAP.

Definition 6 (Classes of Dimension Schemas) *Let $D = (H, \Sigma)$ be a hierarchy schema.*
1. D is canonical *if H has no shortcuts and Σ is equivalent to $\{\langle c, c'\rangle \mid c \nearrow c'\}$.*
2. D is balanced *if D is canonical and H has a source.*

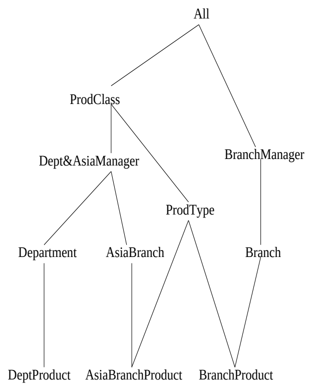
Example 6 *Figure 7 shows a canonical schema dimension that models the bank products.*

A dimension instance d is *homogeneous* if for every pair of categories $c_1 \nearrow c_2$ it holds that the rollup mapping $\Gamma_{c_1}^{c_2} d$ is a total function. Note that the constraint $\langle c, c'\rangle$ where $c \nearrow c'$ forces the rollup mapping from c to c' to be total. Therefore, canonical schemas convey all the homogeneous instances over its hierarchy schema. In this sense, in canonical schemas, Σ captures exactly homogeneity. Also notice that we have defined a canonical schema to be shortcut-free, because otherwise Σ would force the categories from which the shortcut start to be empty in every dimension conveyed by the schema.

Given two classes of schemas S_1, S_2, we define $S_1 \subseteq S_2$ iff for each schema in S_1, there is an equivalent schema in S_2. Then it holds Balanced Schemas $\subseteq$ Canonical Schemas $\subseteq$ Dimension Schemas.

4 Implication

A dimension schema D *logically implies* a dimension constraint α, written $D \models \alpha$, if every dimension instance d over D satisfies α. In our context, the *implication problem* is the problem of determining, given a dimenion schema D and a dimension constraint α, whether $D \models \alpha$.



(e) $\langle c, c' \rangle$, for all edges (c, c') in the hierarchy schema.

Figure 7. A canonical schema for the bank products.

4.1 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 [10] for details.)

Frozen dimension are important because, as we will show next in this section, 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).

Let D be a dimension schema and c a constant of it, $\texttt{Const}_D(c)$ be the set of constants k that occur in atoms of the form $\langle c_i, .., c = k \rangle$ in D.

Definition 7 (Frozen Dimension) *Given a dimension schema D and $c \in C$, a* frozen dimension *with root c is a dimension instance $d : (M, <) \to (C, <)$ of D such that:*
1. d is injective (i.e., each category has at most one member);
2. $d^{-1}(c)$ is a source of $(M, <)$;

There could be infinitely many frozen dimensions, but there are only finitely many up to isomorphism, where isomorphism is defined as follows: d is iso to d' iff there exists a graph mapping $f : (M, <) \to (M', <')$ such that $d = d' \circ f$, and if $k \in \texttt{Const}_D(c_j)$ and $d(k) = c_j = d'(k)$, then $f(x) = x$.

From now one, we will consider frozen dimensions up to isomorphism. We introduce an injective function $\texttt{nk} : \mathbf{C} \to \mathbf{M}$ which assigns a fix member to each category which does not have a constant member in a frozen dimension.

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

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

Example 7 *Consider the dimension schemas $\texttt{locationSch}_\texttt{p}$. The set*

$$\texttt{Frozen}(\texttt{locationSch}_\texttt{p}, \texttt{Store})$$

consists of the dimensions depicted in Figure 8. The figure shows the subgraphs induced by the nonempty edges in the child/parent relation of each frozen dimension.

The set $\texttt{Frozen}(\texttt{locationSch}, \texttt{Store})$ contains the dimensions of Figure 9. Here, we present the frozen dimensions similarly to Figure 8 but we depict the member in a category c, whenever the category has associated some constant that appears in the constraints. Notice that this set illustrates the different structures stores in $Mexico$, USA, and $Canada$ have.

The set $\texttt{Frozen}(\texttt{productSch}, \texttt{Store})$ is depicted in Figure 10.

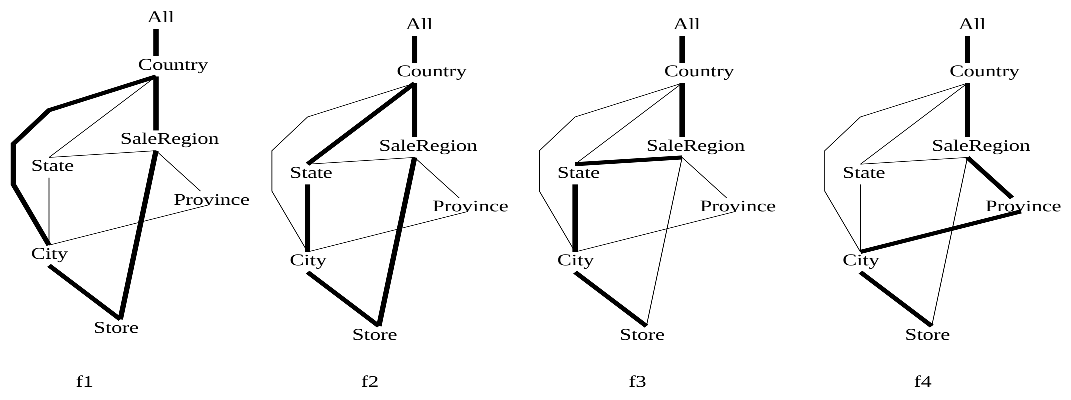


Figure 8. Frozen dimensions of $\texttt{locationSch}_{\texttt{p}}$ with root $Store$.

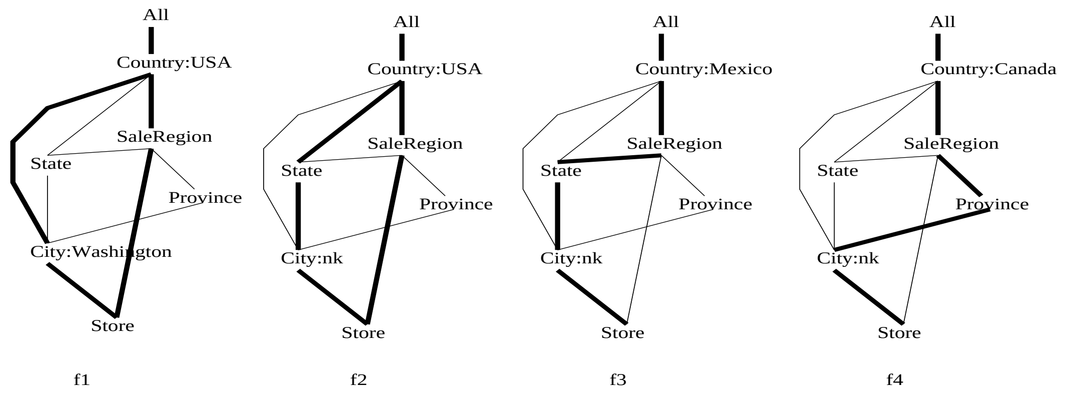


Figure 9. Frozen dimensions of `locationSch` with root $Store$.

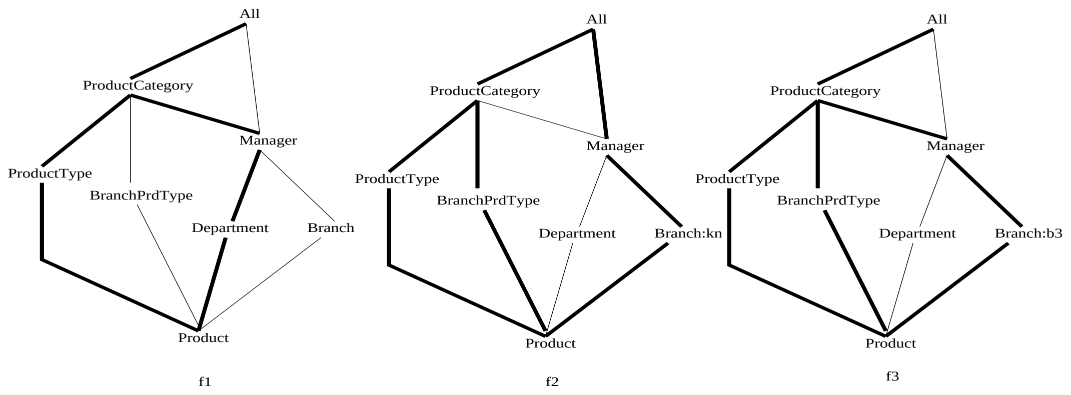


Figure 10. Frozen dimensions of `productSch` with root $Product$.

4.2 Dimension Tuples

Just as a relational table can be viewed as a set of tuples, an OLAP dimension may be viewed as a set of small pieces of data we will call *dimension tuples*. The notion of dimension tuple will serve to simplify several proofs in this thesis.

Definition 8 (Dimension Tuple) *A dimension tuple of a dimension instance d is the restriction of d to the upgraph of $dom(d)$ defined by a particular member x in $dom(d)$.*

It is easily verified that the preceding definition is sound, i.e., any dimension tuple satisfies conditions of Definition 1. Notice that every member x of a dimension d defines a dimension tuple which will be denoted by $\texttt{DimTuple}(d, x)$. Moreover, we can view a dimension as a set having one dimension tuple for each leaf member.

The following lemma says that in order to test if a dimension instance d satisfies a dimension constraint with root c, we just need to check whether the dimension tuple of each member in $\texttt{MembSet}_c$ satisfies the dimension constraint.

Lemma 1 (Dimension Constraints and Dimension Tuples) *Given a dimension instance d and a dimension constraint α with root c, $d \models \alpha$ iff for every member $x \in d^{-1}(c)$, $\texttt{DimTuple}(d, x) \models \alpha$.*

Another result we will need to simplify further proofs is the following:

Lemma 2 (Dimension Constraints and Dimension Tuples) *Given a dimension instance d and a dimension constraint α such that $d \models \alpha$, then for every member x of d, $\texttt{DimTuple}(d, x) \models \alpha$.*

From a dimension tuple t over a dimension schema D we can obtain a frozen dimension of D, denoted by $\texttt{TFrozen}(D, t)$, as follows: for every member x of t, let c be the category to which x belongs, if $x \notin \texttt{Const}_D(c)$, then replace x with $\texttt{nk}(\texttt{c})$

4.3 Category Satisfiability

A category c is said to be *satisfiable* in a schema D (we assume that c is a category of D) if there exists a dimension instance $d \in I(D)$ such that $d^{-1}(c) \neq \emptyset$.

Example 8 *Suppose we add the constraint $\neg\langle SaleRegion, Country\rangle$ to* `locationSch`*. Then, $SaleRegion$ would become unsatisfiable in the resulting schema, because every member in a dimension must reach* `all`*, and consequently, every dimension instance of the hierarchy schema of* `locationSch` *should satisfy $\langle SaleRegion, Country\rangle$.*

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

Theorem 1 (Cat. Satisfiability and Implication) *Given a dimension schema D and a dimension constraint α with root c, $D \models \alpha$ iff c is unsatisfiable in $D' = (H, \Sigma \cup \{\neg\alpha\})$.*

In view of Theorem 1, any algorithm for solving category satisfiability can be used to solve implication. The converse is also true; however, it requires expressing the theorem a little bit differently. Next, we show the importance of frozen dimensions for testing category satisfiability and implication.

4.4 Testing Category Satisfiability

The following theorem proves that frozen dimensions are minimal models [3] for testing category satisfiability.

Theorem 2 (Cat. Satisfiability and Frozen Dimensions) *Given a dimension schema D and a category c of D, c is satisfiable in D iff $\texttt{Frozen}(\texttt{D}, \texttt{c}) \neq \emptyset$.*

Given a dimension schema $D = (H, \Sigma)$ and a category c, a candidate frozen dimension of D with root c can be built by first choosing a subgraph of H, and then selecting the members using the functions $\texttt{Const}_D$ and $\texttt{nk}$. 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 2 establishes an algorithm to solve category satisfiability. In Section 6 we present such an algorithm in details.

Similarly, frozen dimensions can be directly used to test implication, as the following theorem shows.

Theorem 3 (Implication and Frozen Dimensions) *Given a dimension schema D, and a dimension constraint α with root c, $D \models \alpha$ iff for every frozen dimension $f \in \texttt{Frozen}(\texttt{D}, \texttt{c})$, $f \models \alpha$*

We now give the intrinsic complexity of implication and category satisfiability.

Theorem 4 (Complexity) *Category satisfiability is NP-complete, implication is CoNP-Complete.*

From the proof of Theorem 4 it is easily verified that including composed path atoms into dimension constraints does not add extra complexity to the problems.

In canonical schemas, category satisfiability becomes trivial since all the categories are satisfiable.

Proposition 3 (Cat. Satisfiability in Canonical Schemas) *Every category of a canonical schema D is satisfiable in D.*

5 Reasoning about Summarizability

In this section, we give a characterization of summarizability in terms of dimension constraints. In this form, we turn the problem of testing summarizability into testing implication inside our class of constraints.

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

- If $c \neq c_i \neq c_j$ then $\langle c, .., c_i, .., c_j \rangle$ 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 $\langle c, .., c_i, .., c_j \rangle$ represents $\top$.
- If $c = c_j$ and $c, c_j \neq c_i$ then $\langle c, .., c_i, .., c_j \rangle$ represents $\perp$.
- If $c = c_i$ and $c, c_i \neq c_j$ then $\langle c, .., c_i, .., c_j \rangle$ represents $\langle c, .., c_j \rangle$.
- Finally, if $c \neq c_i, c_j$ and $c_i = c_j$ then $\langle c, .., c_i, .., c_j \rangle$ represents $\langle c, .., c_i \rangle$.

Intuitively, the dimension constraint $\langle c, .., c_i, .., c_j \rangle$ means that for all member $x \in \texttt{MembSet}_c$, x rolls up to c_j passing through c_i.

Theorem 5 (Summarizability and Dimension Constraints) *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 $d \models \langle c_b, .., c \rangle \Rightarrow \bigodot_{c_i \in S} \langle c_b, .., c_i, .., c \rangle$.*

The intuition behind Theorem 5 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 trough one and only one of the categories in S. Notice that Theorem 5 shows that summarizability can be characterized as a property of dimension instances themselves, avoiding the mention of fact tables.

Example 9 *In the dimension* `product` *(Figure 2), we have that $ProdCategory$ is summarizable from*

$$\{BranchProdType, Department\}$$

because

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

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

$$\texttt{location} \models \langle Store, .., Country \rangle \Rightarrow \langle Store, .., City, .., Country \rangle.$$

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

$$\texttt{location} \not\models \langle Store, .., Country \rangle \Rightarrow (\langle Store, .., State, .., Country \rangle \oplus \langle Store, .., Province, .., Country \rangle).$$

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

From Theorem 5, it follows that a category c is summarizable from a set of categories S in a dimension schema D iff for every bottom category c_b of D we have $D \models \langle c_b, .., c \rangle \Rightarrow \bigodot_{c_i \in S} \langle c_b, .., c_i, .., c \rangle$. Therefore, testing summarizability reduces to testing implication of the preceding constraint, for each bottom category.

We now study the intrinsic complexity of testing summarizability.

Theorem 6 (Complexity of Testing Summarizability) *Testing summarizability is coNP-complete.*

Proposition 4 (Summarizability and Canonical Schemas) *Given a canonical schema $D = (H, \Sigma)$, a category c of D, and a set of categories S of D, c is summarizable from S in D iff for every bottom category c_b of D, if $c_b \nearrow^* c$, then there is exactly one category $c' \in S$ such that $c_b \nearrow^* c'$ and $c' \nearrow^* c$ in H.*

From Proposition 4 it easily follows that testing summarizability in cannonical schemas is in polytime.

6 The DIMSAT Algorithm

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

6.1 Description of the Algorithm

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

Definition 9 (Subhierarchy) *Given a hierarchy schema H:*

- *a* subhierarchy *of H with root c is a subgraph of H whose source is c and whose sink is* `All`.
- *let $D = (H, \Sigma)$ be a dimension schema, and g be a subhierarchy of H, we say that g* induces a frozen dimension *in D iff there exists a frozen dimension f of D such that $g = \mathtt{ran}(f)$.*

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 $\perp$. 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 $\langle c, c' \rangle$; intuitively, an *into* constraint states that all the members of c have a parent in c'. We conjecture that this optimization should be useful in practice, since in many situations heterogeneity may arise 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 $D = (H, \Sigma)$, and a category c of D, $\mathtt{Prop}(D, c)$ 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 $\mathtt{Prop}(D, c)$ are replaced by their truth values in g. Doing this, $\mathtt{Prop}(D, c)$ 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 $\mathtt{Prop}(D, c)$. 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 10 *Given a set of dimension constraints Σ, and a subhierarchy g of H, $\Sigma \circ g$ 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 $\perp$ otherwise; and (b) renaming every equality atom $c_i.c_j = k$, such that there is no path from c_i to c_j in g, with $\perp$.*

Example 11 *The dimension constraints $\mathtt{Prop}(\mathtt{locationSch}, Store)$ are depicted in Figure 11 (left). Now, let g be the subhierarchy represented as f_2 in Figure 5. The dimension constraints $\mathtt{Prop}(\mathtt{locationSch}, Store) \circ g$ are depicted in Figure 11 (right).*

$\mathtt{Prop}(\mathtt{locationSch}, Store)$	$\mathtt{Prop}(\mathtt{locationSch}, Store) \circ g$
(a) $\langle Store, City \rangle$	(a) $\top$
(b) $\langle Store, .., SaleRegion \rangle$	(b) $\top$
(c) $\langle City = Washington \rangle \equiv \langle City, Country \rangle$	(c) $\langle City = Washington \rangle \equiv \bot$
(d) $\langle City = Washington \rangle \Rightarrow \langle City, .., Country = USA \rangle$	(d) $\langle City = Washington \rangle \Rightarrow \langle City.Country = USA \rangle$
(e) $\langle State, .., Country = Mexico \langle \vee \langle State, .., Country = USA \rangle$	(e) $\langle State, .., Country = Mexico \rangle \vee \langle State, .., Country = USA \rangle$
(f) $\langle State, .., Country = Mexico \rangle \equiv \langle State, SaleRegion \rangle$	(f) $\langle State, .., Country = Mexico \rangle \equiv \bot$
(g) $\langle Province, .., Country = Canada \rangle$	(g) $\langle Province, .., Country = Canada \rangle$

Figure 11. (Left) $\mathtt{Prop}(\mathtt{locationSch}, Store)$**. (Right)** $\mathtt{Prop}(\mathtt{locationSch}, Store) \circ g$**.**

Notice that the dimension constraints $\mathtt{Prop}(D, c) \circ g$ contain only equality atoms. Now, given a dimension schema $D = (H, \Sigma)$ and a subhierarchy $g = (C', \nearrow')$ of H, a c-assignment for g is a injective function $\mathtt{ca} : C' \to \mathtt{Const} \cup \{\mathtt{nk}\}$ such that for all $c' \in C'$, $\mathtt{ca}(c') = k$ implies that the there is an atom of the form $\langle c, .., c = k \rangle$ in Σ.

We say that a c-assignment ca satisfies a set of dimension constraints Σ that mention only equality atoms, denoted $\mathtt{ca} \models \Sigma$, 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 $\langle c, .., c_i = k \rangle$, and we have that $\mathtt{ca}(c_i) = k$ then we replace p with $\top$.

Lemma 3 *Given a dimension schema $D = (H, \Sigma)$, and a subhierarchy g of H 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* $\mathtt{ca} \models \mathtt{Prop}(D, c) \circ g$.

The proof of the lemma is straightforward, so we skip it.

We are now able to introduce the DIMSAT algorithm. DIMSAT, depicted in Figure 12, is basically a backtracking algorithm that explores subhierarchies. The procedure EXPAND constructs subhierarchies of H 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; $g.\mathtt{Out}$, 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 $g.\mathtt{In}^*$, which keeps for every category $c' \in g.C$, the categories that reach directly or indirectly c' in g. As we will see, $g.\mathtt{In}^*$ 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 $g.Top = \{\mathtt{All}\}$. If so, CHECK(g) is called. If not, EXPAND chooses a top category $ctop \in g.Top$, and tries all possible calls $\mathtt{EXPAND}(g, c, R)$, where R is any combination of categories directly above $ctop$ in H 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 $\langle ctop, c' \rangle$ 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 12 *Consider the execution of*

$$\mathtt{DIMSAT}(\mathtt{locationSch}, Store).$$

Figure 13 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.*

Correctness of DIMSAT is proved in Appendix D.

We end this section by giving the asymptotic time complexity of DIMSAT. Let N be the number of categories in D, and let N_K be the number of constant in the schema. In addition, N_Σ stand for the size of Σ.

```
Algorithm DIMSAT$(D, c)$
Input: A dimension schema $D = (H, \Sigma)$ and a category
$c \in C$.
Output: Whether $c$ is satisfiable in $D$.
(1) FIND $:=$ $false$, $Pr :=$ Prop$(c, D)$
(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 $H$
Local Vars: $Pr'$, ca
Global Vars: FIND
(1) $Pr' := Pr \circ g$
(2) For every c-assignment ca of $g$ do
(3)     FIND $:=$ (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: $H$, 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 $Top = \{$All$\}$ then
(7)     CHECK(g)
(8)     If FIND then exit() else return()
(9) EndIF
(10) Choose a category $ctop \neq$ All $\in g.Top$
(11) $Ss := \{c' \in H.$Out$(ctop) \mid$
             $g.$In$(c') \cap g.$In$^*(ctop) \neq \emptyset\}$
(12) $Sc := H.$Out$(ctop) \cap g.$In$^*(ctop)$
(13) $S := H.$Out$(ctop) \setminus (Ss \cup Sc))$
(14) $Into := \{c' \in H.$Out$(ctop) \mid \langle ctop, c' \rangle \in \Sigma\}$
(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 12. Algorithm DIMSAT.

Proposition 5 (Complexity of DIMSAT) DIMSAT *runs in time* $O(2^{N^2+N\log N_K} N^3 N_\Sigma)$.

From the proof of Proposition 5, it follows that the time complexity of DIMSAT can be expressed in terms of the number of subhierarchies of the schema which match the into constraints. Let W be this number, then we have that DIMSAT runs in time $O(W 2^{N \log N_K} N^3 N_\Sigma)$. If the schema does not have equality atoms, the complexity turns to $O(W N^3 N_\Sigma)$.

6.2 Implementation

To assess the performance of DIMSAT we implemented it using Java, and performed experiments on a Pentium IV computer, with CPU clock rate of 2.4 GHz, 512MB RAM, and running Windows XP.

Firstly, we performed experiments with the three dimension schemas introduced in Section 3: $\texttt{locationSch}_\texttt{p}$ (Figure 4); locationSch (Figure 5); and productSch (Figure 6). We ran DIMSAT to test category satisfiability of the bottom categories of the aforementioned schemas.

For each of the schemas we also ran a variation of DIMSAT, called FROZEN, used to compute the whole set of frozen dimension. Recall that DIMSAT halts when a particular frozen dimension is found, which may require the exploration of a particular subset of subhierarchies of the schema. In contrast, when the algorithm returns *false*, it builds the entire set of subhierarchies of the schema. Consequently, there could be differences between the running times of DIMSAT when it returns *true* and when it returns *false*. Since FROZEN does no halt until all the frozen dimensions are found, its running time approximates the time DIMSAT would take in its worst-case executions. The results of the experiments are shown in Figure 14. The first column shows the time (seconds) spent to load the dimension schema; the last two columns show the remaining times taken by DIMSAT and FROZEN. It can be seen that the overall cost of computing the frozen dimensions was less than .1 second in all of the three schemas.

To study scalability of DIMSAT, we performed similar experiments with five more complex dimension schemas. Their hierarchy schemas are lattices of adjacent squares, where the categories are placed in the corners of them. The schemas have disjunctive constraints which do not impose any restriction on how members rollup to the categories directly above

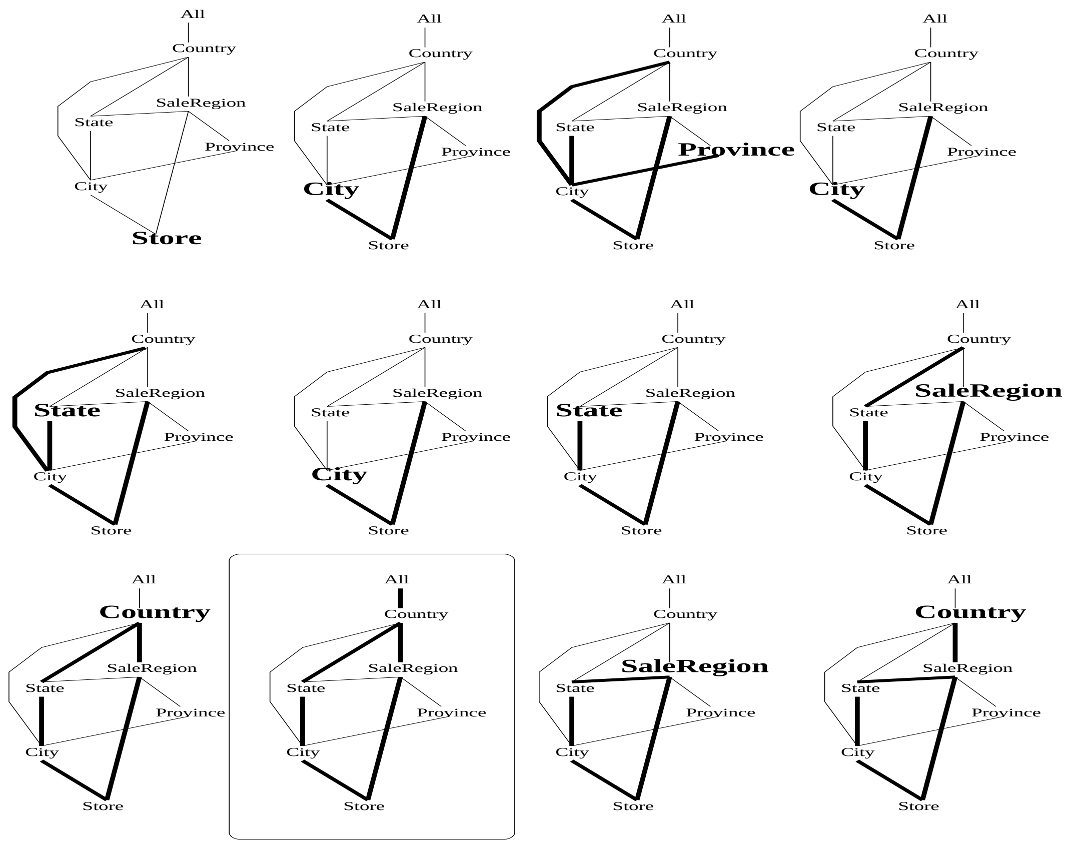


Figure 13. A series of subhierarchies in an execution of $\mathtt{DIMSAT}(\mathtt{locationSch}, Store)$.

	Time Load	`DIMSAT`	`FROZEN`
$\mathtt{locationSch}_{\mathtt{p}}$	.04	.01	.02
`locationSch`	.06	.01	.03
`productSch`	.05	.02	.03

Figure 14. Running times (seconds) of `DIMSAT` for the schemas introduced in Section 3.

	Num Cat.	**Size** Σ	**Num Frozen Dims.**	**Time Load**	`DIMSAT`	`FROZEN`
`lattice1`	8	9	15	.05	.01	.01
`lattice2`	12	18	207	.05	.01	.08
`lattice3`	16	27	2895	.06	.01	.7
`lattice4`	20	36	40735	.09	.01	12.8
`lattice5`	24	45	573951	.1	.01	368.1

Figure 15. Running times (seconds) of `DIMSAT` for the schemas `lattice1`-`lattice5`.

	Num Frozen Dims.	**Time Load**	`DIMSAT`	`FROZEN`	**Gain** `FROZEN`
`lattice1`′	15	.05	.01	.01	0
`lattice2`′	153	.05	.01	.07	.01
`lattice3`′	1494	.07	.01	.3	.4
`lattice4`′	13657	.1	.01	4.6	8.2
`lattice5`′	120038	.12	.01	56.7	311.1

Figure 16. Running times (seconds) of `DIMSAT` for the schemas `lattice1`′-`lattice5`′.

them. Because of this, the number of frozen dimensions of the schemas are the same as their numbers of subhierarchies. The schemas, called `lattice1`, `lattice2`, `lattice3`, `lattice4`, and `lattice5`, have respectively, 8, 12, 16, 20 and 24 categories.

The results of the experiments are shown in Figure 15. The size of each set of constraints is measured as the number of atoms and logic operators that arise in the constraints. Notice that the number of frozen dimensions grows exponentially by a factor of 4 between two consecutive schemas in the table. The computation of the frozen dimension for schemas `lattice1`-`lattice4` take less than a second. This time considerably increases in the last schema (6 minutes aprox.).

From Figure 15 and the analytic study of the complexity of `DIMSAT` we conjecture that in a lattice of 28 categories with structure similar to `lattice1`-`lattice5`, the algorithm `FROZEN` would take more than two hours, and it would take around five days in a lattice of 32 categories. Nevertheless, it is important to notice that it would be very unusual to have schemas with such amounts of subhierarchies in real-world applications. But even if those schema arose, `DIMSAT` may be improved to allow handling schemas with high degree of complexity. For instance, `DIMSAT` may be speed up by precomputing and storing the set of frozen dimensions, turning the algorithm to polytime on the number of frozen dimensions of the schema.

Finally, we investigated the effect of the pruning heuristic on the running time of `DIMSAT`. As explained in Section 6.1, homogeneous edges in `lattice5` properly modeled with *into constraints* may considerably reduce the number of subhierarchies explored by `DIMSAT` and `FROZEN`. In order to assess this claim we tested the algorithm with schemas `lattice1`′-`lattice5`′, which are obtained from `lattice1`-`lattice5` by adding into constraints to some of their edges. In particular, we add one-five *into constraints* to the schemas `lattice1`′-`lattice5`′, respectively. The results, shown in Figure 16, provide evidence that the running times of the algorithms reduce significantly. In particular, the usage of *into constraints* for pruning yields a speed up of 8.2 seconds and 5 minutes for schemas `lattice4`′ and `lattice5`′, respectively. Notice that the frozen dimensions of the schemas reduces w.r.t. schemas `lattice1`-`lattice5`. Because of the pruning strategy, the algorithm avoid the cost of building subhierarchies that violate the into constraints and it only builds subhierarchies that induce frozen dimensions.

7 Related Database Constraints

In this section we briefly compare dimension constraints with other classes of constraints known in the database world.

As explained in several papers (e.g., [12]) OLAP dimension may be modeled as a set of normalized tables, one for each category, containing the rollup mappings that start from the category, along with the the attributes of the category. Therefore the framework presented in this paper may be formalized using a relational database setting.

Let us first clarify the relationship between dimension constraints and First Order Logic (FOL) constraints, that may expressed over the relational representation described above. An important property of the relation $<$ of a dimension instance is that the size of its largest path should be smaller than the size of the largest path without cycle in the hierarchy schema.

This turns the ancestor/descendant relation $\ll$ to be FOL definable. Consequently, the conditions that a dimension must satisfy given in Definition 1 can be defined with FOL sentences over the relational representation. In addition, it is easily verified that dimension constraints are FOL constraints; therefore, our entire framework is a fraction of FOL. Essentially, the partitioning property (Condition c of Definition 1 (3)) turns frozen dimensions into minimal models for testing implication of dimension constraints, which makes the inference test tractable and coNP-complete.

Abiteboul et al. [1] study a class of FOL constraints called *embedded constraints* that formalizes a wide variety of constraints studied in the database literature. Embedded constraints essentially say that the presence of some tuples in the instance implies the presence of some tuples in the instance or implies that certain tuple components are equal. Dimension constraints cannot be expressed with embedded constraints, since we cannot express with them constraints that assert dependences such as “some tuples or some other tuples appear in the instance”.

Example 13 *Consider the dimension constraint $\langle c, c_1\rangle \vee \langle c, c_2\rangle$. This constraint is equivalent to the following FOL expression:*

$$\forall x(\texttt{MembSet}_c(x) \Rightarrow \exists x_1 \exists x_2(\Gamma_c^{c_1}(x, x_1) \vee \Gamma_c^{c_2}(x, x_2))).$$

This constraint cannot be expressed with an embedded constraint, since an embedded constraint is an expression of the form

$$\forall x_1, \ldots, x_n(\phi(x_1, \ldots, x_n) \Rightarrow \exists z_1, \ldots, z_k \psi(y_1, \ldots, y_m)),$$

where $\{z_1, \ldots, z_k\} = \{y_1, \ldots, y_m\} - \{x_1, \ldots, x_n\}$, and ϕ and ψ are conjunctions of atoms.

Dimension Constraints restrict data in a similar fashion as a class of constraints (which are not embedded constraints) called disjunctive existential constraints (dec’s) [6]. 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. Disjunctive existential constraints are used to characterize the possible sets of non-null attributes that may occur in the tuples of a relation; and hence, the possible objects that are mixed in the relation. Formally, a dec has the form $X \vdash \{Y_1, \ldots, Y_n\}$, and means that whenever a tuple is non-null for the set of attributes X, it must be non-null for all the attributes in at least one set of attributes $Y_1, \ldots, Y_n$.

In order to clarify this relationship, let us sssume that the dimension is represented as a single relational table, where we each category is an attribute, and the base category c_{base} is the key of the table. The partitioning restriction (Condition c of Definition 1 (3)) causes the base category to be a key in thus table, also, the edges in the hierarchy schema may be regarded as functional dependences, properly interpreted to deal with null values. It is important to note that this representation is not always possible, as shown in [8], because of heterogeneity.

A dec $X \vdash \{Y_1, \ldots, Y_n\}$ over this table can be expressed with the following dimension constraint over d:

$$(\bigwedge_{l \in X} \langle c_{base}, .., c_j\rangle) \Rightarrow \bigvee_{i \in 1..n}(\bigvee_{c' \in Y_i} \langle c_{base}, .., c'\rangle)$$

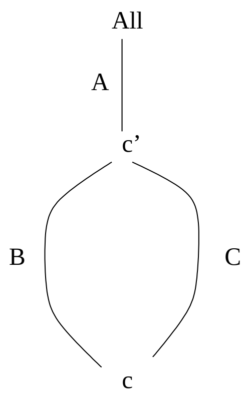


Figure 17. Graphical representation of a path constraint interpreted in a dimension instance.

By definition, a dec of the form $X \vdash \emptyset$ means $\top$. Therefore, we cannot express split constraints of the form $(\bigvee_{c \in X} \neg\langle c_{base}, .., c_j\rangle)$, because the absence of positive atoms in the clauses of their bodies makes their corresponding dec’s $\top$. In particular, the constraint $\langle c_{base}, .., c\rangle \Rightarrow (\neg\langle c_{base}, .., c_1\rangle \vee \neg\langle c_{base}, .., c_2\rangle)$ states that every member of c_{base} that rolls up to c does not rollup to c_1 and c_2 at the same time. This constraint cannot be expressed with dec's. Moreover, as shown by Hurtado and Mendelzon [9], we need constraints of this form to characterize summarizability of c from $\{c_1, c_2\}$ in a hierarchical dimension instance.

The dimension schemas introduced by Husemann et al. [11] can be easily represented with split schemas. However, the converse is not true. Consider a hierarchy schema with categories c, c_1, c_2, such that $c \nearrow c_1$, and $c \nearrow c_2$. The model of Husemann et al. allows us to express only one of the following split constraints: $\langle c, c_1\rangle \wedge \langle c, c_2\rangle$ (the categories are optional), or $\langle c, c_1\rangle \oplus \langle c, c_2\rangle$ (the categories are alternatives). There are a variety of situations in between that cannot be expressed with them. For instance, we may have stores that rollup to $Province$ and do not rollup to $State$, and stores that rollup to $Province$ and $State$, yielding the following split constraint: $\langle Store, .., State\rangle \Rightarrow \langle Store, .., Province\rangle$. Here $Province$ and $State$ are neither optional nor alternative categories. We conclude that, unlike the constraints introduced by Husemann et al. [11], split constraints incorporate the whole expressiveness of the boolean connectives.

We turn now to study the relationship between path constraints [1, 4] and dimension constraints. A path constraint is interpreted over a semistructured data instance (sdi) which, following Buneman et al. [4], can be abstracted as a pair $\sigma = (r, E)$, where r is a constant denoting the "root" of the data instance, and E is a a finite set of binary relations denoting the edge labels. We can represent a dimension instance d with a sdi by making $r = \texttt{All}$, and reversing the child/parent relation $<_d$ to represent the edges of σ. In path constraints, a *path* $\alpha(x, y)$ is a predicate that states the existence of a path α from x to y in σ. In the dimension instance representing σ, α can be viewed as a sequence of categories. Path constraints are expressions of the form:

$$\forall x \forall y (A(r, x) \wedge B(x, y) \Rightarrow C(x, y),$$

where A, B, C are paths [2]. This constraint (see Figure 17) basically states that if there is a path A from `all` to a member x in c', and a path B from x to a member y in c, then there must be a path C from x to y. We can express this path constraint with the following dimension constraint: $AB \Rightarrow C$. Consider now the dimension constraint β: $AB \Rightarrow \neg C$. This constraint could be needed to characterize summarizability of c' from a pair of categories $c_1 \in B$, $c_2 \in C$. In order to represent β with a path constraint we would need to have $\forall x \forall y (A(r, x) \wedge B(x, y) \Rightarrow \neg C(x, y)$ in the language of path constraints, which is not the case. In conclusion, unlike dimension constraints, path constraints lack the full expressiveness of the Boolean operators.

8 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 this paper, we conjecture that in most practical situations `DIMSAT` should yield execution times of the order of a less than a second. 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.

Acknowledgments

This research was supported by Millenium Nucleus, Center for Web Research (P01-029-F), Mideplan, Chile. We thank Renée Miller, Ken Sevcik, Anthony Bonner, John Mylopoulos, and Laks Lakshmanan for their fruitful suggestions.

[2] We consider only path constraints in *forward form* because *backward path constraints* need cycles in the instance to be satisfied.

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] R. Agrawal, A. Gupta, S. Sarawagi, P. Deshpande, S. Agarwal, J. Naughton, and R. Ramakrishnan. On the computation of multidimensional aggregates. In *Proceedings of the 22nd International Conference on Very Large Data Bases*, Bombay, India, 1996.

[3] E. Borger, E. Gradel, and Y. Gurevich. *The Classical Decision Problem*. Springer, Berlin, 1996.

[4] 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.

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

[6] 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.

[7] A. Gupta, V. Harinarayan, and D. Quass. Generalized projections: A powerful approach to aggregation. In *Proceedings of the 21st International Conference on Very Large Data Bases*, Zurich, Switzerland, 1995.

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

[9] 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.

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

[11] 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.

[12] 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.

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

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

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

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

[17] 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.

[18] 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.

A Proofs of Theorems

Theorem 1 Given a dimension schema D and a dimension constraint α with root c, $D \models \alpha$ iff c is unsatisfiable in $D' = (H, \Sigma \cup \{\neg\alpha\})$.

Proof of Theorem 1 First, we prove that $(*)$ given a dimension instance d, a dimension constraint α, and a member $x \in d^{-1}$, then $\texttt{DimTuple}(d, x) \not\models \alpha$ iff $\texttt{DimTuple}(d, x) \models \neg\alpha$. This follows directly from the fact that $\texttt{DimTuple}(d, x)$ has only x in d^{-1}.

(If) Assume $D \not\models \alpha$. Then there exists a dimension instance $d \in I(D)$ such that $d \not\models \alpha$. Therefore, by Lemma 1, there exists at least one member $x \in d^{-1}$ in d such that $\texttt{DimTuple}(d, x) \not\models \alpha$. By $(*)$, $\texttt{DimTuple}(d, x) \models \neg\alpha$. Now, because $d \in I(D)$, $d \models \Sigma$, and by Lemma 2, we have $\texttt{DimTuple}(d, x) \models \Sigma$. Therefore $\texttt{DimTuple}(d, x)$ is over H, $\texttt{DimTuple}(d, x) \models \Sigma$, and $\texttt{DimTuple}(d, x) \models \neg\alpha$. Consequently c is satisfiable in D', yielding a contradiction.

(Only If) Assume that c is satisfiable in D'. Then there exists a dimension d over H such that $d \models \Sigma$ and $d^{-1} \neq \emptyset$ in d. Hence $d \models \neg\alpha$. Now, choose a member $x \in d^{-1}$. From Lemma 1, it follows that $\texttt{DimTuple}(d, x) \models \neg\alpha$. By $(*)$, $\texttt{DimTuple}(d, x) \not\models \alpha$. Now, from Lemma 2, and the fact that $d \models \Sigma$, it follows that $\texttt{DimTuple}(d, x) \models \Sigma$. Therefore $\texttt{DimTuple}(d, x) \models \Sigma$, and $\texttt{DimTuple}(d, x) \not\models \alpha$, and hence $D \not\models \alpha$ leading to a contradiction.

•

Theorem 2 Given a dimension schema D and a category c of D, c is satisfiable in D iff $\texttt{Frozen}(\texttt{D}, \texttt{c}) \neq \emptyset$.

Proof of Theorem 2 **(If)** It is direct since a frozen dimension d in $\texttt{Frozen}(\texttt{D}, \texttt{c})$ is a dimension instance of D and has a member in d^{-1}.

(Only If) Assume that c is satisfiable in D. Then there exists at least one dimension instance $d \in I(D)$ such that $d^{-1} \neq \emptyset$. Now, consider a member $x \in d^{-1}$ in d, and let $d' = \texttt{DimTuple}(d, x)$. From Lemma 2, it follows that $d' \models \Sigma$, and hence $d' \in I(D)$. Thus $\texttt{TFrozen}(D, x) \in \texttt{Frozen}(\texttt{D}, \texttt{c})$, leading to a contradiction. •

Theorem 3 Given a dimension schema D, and a dimension constraint α with root c, $D \models \alpha$ iff for every frozen dimension $f \in \texttt{Frozen}(\texttt{D}, \texttt{c})$, $f \models \alpha$

Proof of Theorem 3 **(If)** Assume that there exists a dimension $d \in I(D)$ such that $d \not\models \alpha$. Then, by Lemma 1, there exists at least a member $x \in d^{-1}$, such that $d' = \texttt{DimTuple}(d, c, x) \not\models \alpha$. Now, let $f = \texttt{TFrozen}(D, d')$. It is easily verified that $f \not\models \alpha$. Consequently, $f \in \texttt{Frozen}(\texttt{D}, \texttt{c})$, and $f \not\models \alpha$, yielding a contradiction.

(Only If) Assume that there exists a frozen dimension $f \in \texttt{Frozen}(\texttt{D}, \texttt{c})$ such that $f \not\models \alpha$, then $D \not\models \alpha$, leading to a contradiction. •

Theorem 4 Category satisfiability is NP-complete, implication is CoNP-complete.

Proof of Theorem 4 First, we will prove that implication is NP-complete.

(NP-hard) We will show a straightforward reduction from SAT to the problem of testing category satisfiability. From a propositional formula α, we will build the following dimension schema $D = (H, \Sigma)$: the hierarchy schema H has a base category c_{base}, and for each propositional variable p of α, H has a category c_p. Every category c_p is connected with $\texttt{All}$, and c_{base} is connected with each category c_p. The set of constraints Σ is as follows: we have α' where α' is obtained from α by replacing every variable p with $\langle c_{base}, c_p\rangle$. It is easy to see that α is propositionally satisfiable iff c_{base} is satisfiable in D.

(NP) Given a category c of a schema $D = (H, \Sigma)$, a candidate frozen dimension in $\texttt{Frozen}(\texttt{D}, \texttt{c})$ can be specified by a pair $(g, \texttt{ca})$, where g is a candidate frozen graph of H whose root is c, and $\texttt{ca}$ assigns a member, that could be $\texttt{nk}(\texttt{c}')$ or some constant mentioned in a equality atom, to a each category in g. The child/parent relation of f is defined by the edges of g. A nondeterministic algorithm for testing if c is satisfiable in D needs only guess a candidate frozen dimension f, and test in polytime whether f satisfy conditions of Definition 1, and whether $f \models \Sigma$.

Notice that, even if Σ contains composed path atoms, the test $f \models \Sigma$ can be done in polytime. Thus, including composed path atoms in dimension constraints does not add extra complexity to the problem.

By Theorem 1, testing whether $D \not\models \alpha$ is equivalent to testing whether c is satisfiable in $(H, \Sigma \cup \{\neg\alpha\})$. By Theorem 4, testing category satisfiability is NP-complete, hence testing implication is coNP-complete. •

Theorem 5 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 $d \models \langle c_b, .., c\rangle \Rightarrow \bigodot_{c_i \in S}\langle c_b, .., c_i, .., c\rangle$.

Proof of Theorem 5 Let us denote by R_{c_b} the expression $\biguplus_{c_i \in S} \pi_{c_b}(\Gamma_{c_b}^{c_i} d \bowtie \Gamma_{c_i}^{c} d)$. From Corollary 1, it suffices to prove that for every bottom category c_b of d, (a) $\Gamma_{c_b}^{c} d = R_{c_b}$ iff (b) $d \models \langle c_b, .., c\rangle \Rightarrow \bigodot_{c_i \in S} \langle c_b, .., c_i, .., c\rangle$.

Recall the notion of dimension tuple, `DimTuple`, from Section 4.2. The following two statements are easily verified: (1) given a dimension instance d, two categories c_a and c_b of d, and a member $e \in \texttt{MembSet}_{c_a}$, $\texttt{DimTuple}(d, e) \models \langle c_a, .., c_b\rangle$ iff $\exists y((x_a, y) \in \Gamma_{c_a}^{c_b} d)$; and (2) given a dimension instance d, three categories c_a, c_b, and c_g of d, and a member $e \in \texttt{MembSet}_{c_a}$, $\texttt{DimTuple}(d, e) \models \langle c_a, .., c_b, .., c_g\rangle$ iff $\exists y \exists z((e, y) \in \Gamma_{c_a}^{c_b} d \wedge (y, z) \in \Gamma_{c_b}^{c_g} d)$.

(If) Assume (a) is false, then there are three cases to consider.

- (Case 1) There is a pair (e_b, e) in $\Gamma_{c_b}^{c} d$ such that (e_b, e) is not in R_{c_b}. Then from (1), it follows that $\texttt{DimTuple}(d, e_b) \models \langle c_b, .., c\rangle$. Because (b) holds, the constraint mentioned in (b) also holds for $\texttt{DimTuple}(d, e_b)$. Thus there is exactly one category $c_i \in S$ such that $\texttt{DimTuple}(d, e_b) \models \langle c_b, .., c_i, .., c\rangle$. By (2), $\exists y \exists z((e_b, y) \in \Gamma_{c_b}^{c_i} d \wedge (y, z) \in \Gamma_{c_i}^{c} d)$; and because d is partitioned, $\exists y((e_b, y) \in \Gamma_{c_b}^{c_i} d \wedge (y, e) \in \Gamma_{c_i}^{c} d)$. Therefore the tuple (e_b, e) appears in $\pi_{c_b, c}(\Gamma_{c_b}^{c_i} d \bowtie \Gamma_{c_i}^{c} d)$, and hence (e_b, e) appears in R_{c_b}, leading to a contradiction.

- (Case 2) There is a tuple (e_b, e) in $\Gamma_{c_b}^{c}$ that occurs at least twice in R_{c_b}. Then there are two categories c_i, c_j in S such that (e_b, e) appears in $\pi_{c_b, c}(\Gamma_{c_b}^{c_i} d \bowtie \Gamma_{c_i}^{c} d)$, and (e_b, e) appears in $\pi_{c_b, c}(\Gamma_{c_b}^{c_j} d \bowtie \Gamma_{c_j}^{c} d)$. Hence $\exists y(\Gamma_{c_b}^{c_i}(e_b, y) \wedge \Gamma_{c_i}^{c}(y, e))$, and $\exists z(\Gamma_{c_b}^{c_j}(e_b, z) \wedge \Gamma_{c_j}^{c}(z, e))$. By (2), $\texttt{DimTuple}(d, e_b) \models \langle c_b, .., c_i, .., c\rangle$, and $\texttt{DimTuple}(d, e_b) \models \langle c_b, .., c_j, .., c\rangle$. Because $(e_b, e) \in \Gamma_{c_b}^{c} d$, and by (2), $\texttt{DimTuple}(d, e_b) \models \langle c_b, .., c\rangle$. Hence the constraint mentioned in (b) does not hold in $\texttt{DimTuple}(d, e_b)$, and thus (b) is false, leading to a contradiction.

- (Case 3) There is a tuple (e_b, e) in R_{c_b} which is not in $\Gamma_{c_b}^{c} d$. Then there is a category $c_i \in S$ such that $\exists y(\Gamma_{c_b}^{c_i}(e_b, y) \wedge \Gamma_{c_i}^{c}(y, e))$. Thus, $(e, e_b) \in \Gamma_{c_b}^{c} d$, yielding a contradiction.

(Only If) Assume (b) is false. Then, there are two cases to consider.

- (Case 1) The dimension d has a base member e_b such that $\texttt{DimTuple}(d, e_b) \models \langle c_b, .., c\rangle$, and for all $c_i \in S$, $\texttt{DimTuple}(d, e_b) \not\models \langle c_b, .., c_i, .., c\rangle$. Then from (1) and (2), it follows that $(e_b, e) \in \Gamma_{c_b}^{c} d$, and for all $c_i \in S$ it is not the case that $\exists y \exists z((e_b, y) \in \Gamma_{c_b}^{c_i} d \wedge (y, z) \in \Gamma_{c_i}^{c} d)$. Thus (e_b, e) does not appear in R_{c_b}, and hence (a) does not hold, yielding a contradiction.

- (Case 2) The dimension d has a base member e_b such that $\texttt{DimTuple}(d, e_b) \models \langle c_b, .., c\rangle$, and there exist at least two categories $c_i, c_j \in S$ such that $\texttt{DimTuple}(d, e_b) \models \langle c_b, .., c_i, .., c\rangle$, and $\texttt{DimTuple}(d, e_b) \models \langle c_b, .., c_j, .., c\rangle$. Therefore, from (1) and (2), it follows that $(e_b, e) \in \Gamma_{c_b}^{c} d$, and there are two categories $c_i, c_j \in S$ such that $\exists y \exists z(\Gamma_{c_b}^{c_i}(e_b, y) \wedge \Gamma_{c_i}^{c}(y, z))$, and $\exists v \exists w((e_b, v) \in \Gamma_{c_b}^{c_j} d \wedge (v, w) \in \Gamma_{c_i}^{c} d)$. Because d is partitioned, there is a member e in d^{-1} such that $\exists y((e_b, y) \in \Gamma_{c_b}^{c_i} d \wedge (y, e) \in \Gamma_{c_i}^{c} d)$, and $\exists v((e_b, v) \in \Gamma_{c_b}^{c_j} d \wedge (v, e) \in \Gamma_{c_i}^{c} d)$. Hence (e_b, e) appears at least twice in R_{c_b} and appears once in $\Gamma_{c_b}^{c} d$, leading to a contradiction.

-

Theorem 6 Testing summarizability is coNP-complete.

Proof of Theorem 6 **(coNP-hard)** We will present a polytime transformation from VALIDITY, which is known to be coNP-complete. In VALIDITY we are given a proposition P and we are asked whether P is valid, i.e., whether P is satisfied by all truth assignments. From the instance P of VALIDITY we obtain the dimension schema $D = (H, \Sigma)$. The hierarchy schema H is as follows: $C = \{c_b, c, c', \texttt{All}\} \cup C_P$, where C_P is a set containing one category c_i for each propositional variable p_i in P. The relation $\nearrow$ contains the pairs $c_b \nearrow c, c_b \nearrow c'$, $c \nearrow \texttt{All}$, $c' \nearrow All$. Also, for each category $c_i \in C_P$ we have $c' \nearrow c_i$ and $c_i \nearrow \texttt{All}$. The set Σ contains the following constraints: $\langle c_b, c\rangle \oplus \langle c_b, c'\rangle$, $\langle c, \texttt{All}\rangle$, $\langle c', \texttt{All}\rangle$, and $\neg\alpha_P$, where α_P is the constraint obtained from P by replacing each propositional variable c_i with $\langle c', c_i\rangle$.

Now, consider the instance of *testing summarizability* in which we are asked whether `All` is summarizable from c in D. It is easily verified that `All` is summarizable from c in D iff c' is unsatisfiable in D. In addition, we can easily see that c is unsatisfiable in D iff P is valid. Therefore `All` is summarizable from c in D iff P is valid.

(coNP) In order to test summarizability, we need to test the implication of the summarizability condition. First we will show that testing implication of a constraint which is a Boolean combination of atoms of the form $\langle c, .., c_i\rangle$ or $\langle c, .., c_i, .., c_j\rangle$ (in this proof we will refer to such atoms as complex atoms) is in coNP. It is enough to prove that testing category satisfiability in a dimension schema with complex atoms is in NP. An NP algorithm that solves the problem needs only to guess a candidate frozen dimension d (see the proof of Theorem 4) and tests whether d does not have cycles and shortcuts and satisfies the set of constraints of the schema. Any complex atom that appears in the set of constraint can be evaluated in d in polytime, since this task essentially requires the computation of the transitive closure of the hierarchy of d. The remainder atoms can be evaluated in polytime as well. Thus, the constraints can be evaluated in polytime.

Since the size of the summarizability constraint, without replacing its complex atoms, is polynomial on the size of the dimension schema, testing summarizability is coNP.

•

B Proofs of Propositions

Proposition 1 A category c is summarizable from a set of categories S in a dimension instance d iff $\Gamma^{c}_{c_{base}} = \biguplus_{c_i \in S} \pi_{c_{base},c}(\Gamma^{c_i}_{c_{base}} \bowtie \Gamma^{c}_{c_i})$.

Proof of Proposition 1 The condition of Definition 2 is equivalent to the following condition:

$$\Pi_{c,m=\mathtt{af}(m)}(\Gamma^{c}_{c_{base}} \bowtie f_{base}) = \Pi_{c,m=\mathtt{af}^c(m)}(\biguplus_{i=1\ldots n} \pi_{c,m}(\Gamma^{c}_{c_i} \bowtie (\Pi_{c_i,m=\mathtt{af}(m)}(\Gamma^{c_i}_{c_{base}} \bowtie f_{base}))))$$

Because the dimension is partitioned, i.e., each element in c_i rolls up to a unique element in c, we can push up the projection $\Pi_{c_i,m=af(m)}$ in the right-side expression yielding:

$$\Pi_{c,m=\mathtt{af}(m)}(\Gamma^{c}_{c_{base}} \bowtie f_{base}) = \Pi_{c,m=\mathtt{af}^c(m)}(\biguplus_{i=1\ldots n} \pi_{c,m}\Pi_{c_i,m=\mathtt{af}(m)}(\Gamma^{c}_{c_i} \bowtie (\Gamma^{c_i}_{c_{base}} \bowtie f_{base}))).$$

And because af is distributive, we have:

$$\Pi_{c,m=\mathtt{af}(m)}(\Gamma^{c}_{c_{base}} \bowtie f_{base}) = \Pi_{c,m=\mathtt{af}(m)}(\biguplus_{i=1\ldots n} \pi_{c,m}(\Gamma^{c}_{c_i} \bowtie (\Gamma^{c_i}_{c_{base}} \bowtie f_{base}))).$$

Because π does not eliminate duplicates, we can replace $\pi_{c,m}$ with $\pi_{c_{base},c,m}$ yielding:

$$\Pi_{c,m=\mathtt{af}(m)}(\Gamma^{c}_{c_{base}} \bowtie f_{base}) = \Pi_{c,m=\mathtt{af}(m)}(\biguplus_{i=1\ldots n} \pi_{c_{base},c,m}(\Gamma^{c}_{c_i} \bowtie (\Gamma^{c_i}_{c_{base}} \bowtie f_{base}))).$$

Because $\bowtie$ is associative, we have:

$$\Pi_{c,m=\mathtt{af}(m)}(\Gamma^{c}_{c_{base}} \bowtie f_{base}) = \Pi_{c,m=\mathtt{af}(m)}(\biguplus_{i=1\ldots n}(\pi_{c_{base},c,m}(\Gamma^{c}_{c_i} \bowtie \Gamma^{c_i}_{c_{base}})) \bowtie f_{base}).$$

Finally, the condition of Definition 2 is equivalent to:

$$(*)\ \Pi_{c,m=\mathtt{af}(m)}(\Gamma^{c}_{c_{base}} \bowtie f_{base}) = \Pi_{c,m=\mathtt{af}(m)}(R \bowtie f_{base}),$$

where R is the query $\biguplus_{i\in 1\ldots n} \pi_{c_{base},c}(\Gamma^{c_i}_{c_{base}} \bowtie \Gamma^{c}_{c_i})$.

We now prove both directions of the proposition.

(If) It is direct by replacing R with $\Gamma^{c}_{c_{base}}$ in $(*)$.

(Only If) Assume $(*)$, and suppose that $R \neq \Gamma^{c}_{c_{base}}$. Then there is a tuple $t = (x_b, x_c)$ such that the number of occurrences of t in R is different than the number of occurrences of t in $\Gamma^{c}_{c_{base}}$. Now, let af be the aggregate function sum, and let f_{base} be the fact table with a single tuple $(x_b, 1)$. Then it is direct that the right and left side query of $(*)$ compute different values for the measure m at x_c, yielding a contradiction. •

Proposition 2 Every dimension schema is satisfiable.

Proof of Proposition 2 Given a dimension schema $D = (H, \Sigma)$, let d be a dimension instance with hierarchy schema H, and a unique member all. It is easy to see that the relation $<$ of d inducs a hierarchy domain and d satisfies conditions of Definition 1. Moreover, because all the categories of d, except All, are empty, and there cannot be a dimension constraint with root All, trivially $d \models \Sigma$. Hence $d \in I(D)$, and D is satisfiable. •

Proposition 3 Every category of a canonical schema is satisfiable.

Proof of Proposition 3 Given a canonical schema $D = (H, \Sigma)$ (recall that Σ here is a set of constraints that represents the condition of homogeneity), let D' be the schema (H, Σ') such that Σ' has a constraint $\langle c_i, c_j \rangle$ for every edge in H. We have that a category c is satisfiable in D iff c is satisfiable in D', simply because $I(D) = I(D')$. Now consider the dimension f whose hierarchy schema is H. The dimension f has a member $\mathtt{nk}(\mathtt{c}')$ for every category in $Out^*(c)$, and its child/parent relation $<$ is defined as follows: for every pair of members x, y of f, $x < y$ iff y's category is directly above x's category in H. It is easily verified that f satisfies conditions of Definition 1. Also, f satisfies all the into constraints in Σ'. Thus f is a frozen dimension in $\mathtt{Frozen}(\mathtt{D}, \mathtt{c})$, and hence c is satisfiable in D. •

Proposition 4 Given a canonical schema $D = (H, \Sigma)$, a category c of D, and a set of categories S of D, c is summarizable from S in D iff for every bottom category c_b of D, if $c_b \nearrow^* c$, then there is exactly one category $c' \in S$ such that $c_b \nearrow^* c'$ and $c' \nearrow^* c$ in H.

Proof of Proposition 4 Consider the constraint $\langle c_b, .., c \rangle \Rightarrow \bigodot_{c_i \in S} \langle c_b, .., c_i, .., c \rangle$, where c_b is a bottom category of D. Now consider the atom $\langle c_b, .., c \rangle$ that appears in $\Xi(c_b, c, S)$. Because D is canonical, $D \models \langle c_b, .., c \rangle$ means that $c_b \nearrow^* c$ in D. Similarly, $D \models \langle c_b, .., c', .., c \rangle$ means that $c_b \nearrow^* c'$ and $c' \nearrow^* c$ in H. Thus $D \models \Xi(c_b, c, S)$ asserts that there is exactly one category $c' \in S$ such that $c_b \nearrow^* c'$ and $c' \nearrow^* c$ in H. •

Proposition 5 DIMSAT runs in time $O(2^{N^2 + N \log N_K} N^3 N_\Sigma)$.

Proof of Proposition 5 The execution time of DIMSAT can be decomposed into the time T_1 due to executions of EXPAND without considering subcalls to CHECK, plus the time T_2 due to executions of CHECK.

Let us first examine T_1. If we do not consider the execution of CHECK, an arbitrary execution of EXPAND takes $O(N N_{out})$ steps in lines 1-14, plus the time for the executions of further calls to EXPAND in Line 16. Here, N_{out} stands for the maximum out degree of the categories in D. If we assume that all sets are implemented as bit vectors, each union, difference, and intersection operation takes $O(N)$. Thus, the time $O(N N_{out})$ of lines 1-14 is basically given by lines 4 and 11, where at most N_{out} operations are done.

Therefore, T_1 is in $O(\alpha N N_{out})$, where α is the number of calls made to EXPAND in the entire execution of the algorithm. Now, when constructing a candidate subhierarchy, EXPAND is called at most once for each category in the subhierarchy. There are at most 2^N candidate subhierarchies, so $\alpha < 2^N N$. Thus T_1 is in $O(2^N N^3)$

Now, we study T_2. The cost of a single call of CHECK is (a) the time used to compute $Pr' := Pr \circ g$, plus (b) the time that Loop 2-4 takes. A naive procedure to compute $Pr' := Pr \circ g$ is as follows: while we are doing a depth-first traversal of g, we delete from each path the edge being traversed; in the same step, we mark the equality atom that refers to the node we visit. At the end, we make false the equality atoms unmarked and the path atoms that remain in the list. This takes $O((N_\Sigma) N^2)$, which is the order of (a). Time (b) is in the order of the number of c-assignments times the cost of evaluating each c-assignment. The number of c-assignments is at most ${N_K}^N$, and the evaluation of each of them takes at most $N N_\Sigma$ steps. Therefore, Time (b) is in $O({N_K}^N N_\Sigma)$. Consequently, each execution of CHECK takes $O(N_\Sigma N^2 + {N_K}^N N N_\Sigma)$ steps. Hence each execution of CHECK takes $O({N_K}^N N^2 N_\Sigma)$ steps. Consequently, we have that T_2 is in $O(2^{N^2}({N_K}^N N^2 N_\Sigma))$.

Therefore $T_1 + T_2$ is in $O(2^{N^2 + N \log N_K} N^3 N_\Sigma)$.
•

C Proofs of Lemmas

Lemma 1 Given a dimension instance d and a dimension constraint α with root c, $d \models \alpha$ iff for every member $x \in d^{-1}(c)$, $\mathtt{DimTuple}(d, x) \models \alpha$.

Proof of Lemma 1 **(If)** Assume that $d \not\models \alpha$. Then there must exist a member $m \in \mathtt{MembSet}_c$ such that $\alpha[x/m]$ does not hold in d, where $\alpha[x/m]$ is the FOL expression obtained from α by renaming the unique free variable x with m. Now consider a single atom at in α. It is easily verified that **(*)** $at[x/m]$ holds in d iff $at[x/m]$ holds in $\mathtt{DimTuple}(d, m)$. Thus $\alpha[x/m]$ does not hold in $\mathtt{DimTuple}(d, m)$. Now, from the fact that $\mathtt{DimTuple}(d, m)$ has a unique member m in $\mathtt{MembSet}_c$, it follows that $\mathtt{DimTuple}(d, m)$ does not satisfy $\forall x \in d^{-1}(c)[\alpha]$. Thus, $\mathtt{DimTuple}(d, m) \not\models \alpha$, leading to a contradiction. **(Only If)** Assume that there exists a member $m \in d^{-1}(c)$ such that $\mathtt{DimTuple}(d, m) \not\models \alpha$. Then $\alpha[x/m]$ does not hold

in $\texttt{DimTuple}(d, m)$. Now, from (*), it follows that $\alpha[x/m]$ does not hold in d. Hence, d does not satisfy $\forall x \in d^{-1}(c)\alpha$. Consequently, $d \not\models \alpha$, yielding a contradiction.
•

Lemma 2 Given a dimension instance d and a dimension constraint α such that $d \models \alpha$, then for every member x of d, $\texttt{DimTuple}(d, x) \models \alpha$.

Proof of Lemma 2 Assume there is a member m of d such that $\texttt{DimTuple}(d, m) \not\models \alpha$. Then there are three cases: (Case 1) $m \in d^{-1}(c)$. In this case, we use Lemma 1 to reach a contradiction. (Case 2) $m \notin d^{-1}(c)$ and m rolls up to c. Let m' be the ancestor of x in $d^{-1}(c)$. Then, from Lemma 1, it follows that $\texttt{DimTuple}(\texttt{DimTuple}(d, m), m')$ does not satisfy α. But $\texttt{DimTuple}(\texttt{DimTuple}(d, m), m') = \texttt{DimTuple}(d, m')$. Hence $\texttt{DimTuple}(d, m') \not\models \alpha$, and using Lemma 1, $d \not\models \alpha$, yielding a contradiction. (Case 3) m does not rollup to c. Then, $\texttt{DimTuple}(d, m)$ has no member in $d^{-1}(c)$, and it is direct that $\texttt{DimTuple}(d, m) \models \alpha$, yielding a contradiction. •

Lemma 3 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 $\texttt{ca}$ of g such that $\texttt{ca} \models \texttt{Prop}(ds, c) \circ g$.

D Correctness of DIMSAT

In this section we prove the correctness of DIMSAT. Let us define the procedure EXPAND′, which is obtained from EXPAND by replacing Line 7 (where CHECK is called) with an operation that adds g to a global variable $Result$.

Next, we prove some properties of EXPAND′.

We will denote by P_i, where i can be a number, or a letter, a particular instance (or execution) of a procedure P. If p is a parameter of P, p_i stands for the actual parameter that corresponds to p in the instance P_i of P. In particular, given an instance $\texttt{EXPAND}_i(c_i, R_i)$ of EXPAND, g_i and $ctop_i$ stand for the variables g and $ctop$ immediately after the execution of Line 11. The *call graph* of an execution of a recursive procedure P is a tree whose nodes are instances of P, and each edge (P_i, P_j) represents that P_i called P_j during the execution. The children of a node are ordered according to their occurrence in the execution. Consider an instance $\texttt{EXPAND}_n(c_n, R_n)$ of EXPAND, and let τ be the path from the root to that instance in the call graph of $\texttt{EXPAND}(c, \emptyset)$, then $CatP_n$ stands for the set of categories c_i such that there is an instance $\texttt{EXPAND}_i(c_i, R_i)$ in τ.

Lemma 4 (Properties of EXPAND′) *Given an instance* $\texttt{EXPAND'}_\texttt{i}(\texttt{c}_\texttt{i}, \texttt{R}_\texttt{i})$ *in the call graph* CG *of* $\texttt{EXPAND'}(\texttt{c}, \emptyset)$, *the following hold:*

1. *The category c reaches every category in $g_i.C$, and each category in $g_i.C$ reaches a level in $g_i.Top$ in g_i.*

2. *g_i has no cycles or shortcuts.*

3. *For every into constraint $\langle c_1, c_2 \rangle \in \Sigma$ such that $c_1 \in (g_i.C \setminus g.Top)$, we have that (c_1, c_2) is an edge of g_i.*

4. *$g_i.Top \neq \emptyset$.*

5. *Let* $\texttt{EXPAND'}_\texttt{j}(\texttt{c}_\texttt{j}, \texttt{R}_\texttt{j})$ *be the instance that called* $\texttt{EXPAND'}_\texttt{i}(\texttt{c}_\texttt{i}, \texttt{R}_\texttt{i})$, *then $c_i \notin CatP_j$.*

Proof of Lemma 4

1. We omit the proof since the statement can be proved by a straightforward induction on the length of a path in CG.

2. We will prove this using induction on the length of the execution path. Base Case: it is direct since there is a unique category in $g.C$, and this category is also in $g.Top$. Induction step: assume that the instance $\texttt{EXPAND'}(\texttt{c}_\texttt{j}, \texttt{R}_\texttt{j})$ called $\texttt{EXPAND'}(\texttt{c}_\texttt{i}, \texttt{R}_\texttt{i})$. Suppose g_i has a cycle, then because g_j has no cycles, at least one category $c_n \in R_i$ must be in the cycle. But, because $c_n \notin Sc_j$ (where Sc_j is computed in Line 12) in $\texttt{EXPAND'}_\texttt{j}(\texttt{c}_\texttt{j}, \texttt{R}_\texttt{j})$, $c_n \notin g_j.In^*(ltop_j)$. Now from Line 4 in $\texttt{EXPAND'}_\texttt{i}(\texttt{l}_\texttt{i}, \texttt{R}_\texttt{i})$, it follows that $g_i.In^*(c_n) = g_j.In^*(ctop_j) \cup \{ctop_j\}$, and hence $c_n \notin g_i.In^*(c_n)$, and therefore c_n cannot be in a cycle, leading to a contradiction. Now, suppose g_i has a shortcut, then because g_j has no shortcuts, there must be a shortcut (c', c_n), for some $c_n \in R_i$ in g_i. Therefore, $g_i.In(c_n) \cap g_i.In^*(c_i) \neq \emptyset$. And from

$c_i = ctop_j$, $g_i.In(c_n) \subseteq H.In(c_n)$, and $g_i.In^*(ctop_j) = g_j.In^*(ctop_j)$, t follows that $H.In(c_n) \cap g_j.In^*(ctop_j) \neq \emptyset$. Now, because of Line 11, $c_n \in Ss_j$. Moreover, because of Line 13, $c_n \notin S$; and because of Line 16, $c_n \notin R_i$, yielding a contradiction.

3. We will prove this by induction on the length of the execution path. Base Case: it is direct since there is a unique category in $g.C$, and it is also in $g.Top$. Induction step: assume that the instance $\mathtt{EXPAND'(c_j, R_j)}$ called $\mathtt{EXPAND'(c_i, R_i)}$. Assume that there exists a category c', $c' \in g_i.C \setminus g_i.Top$, such that $\langle c', c'' \rangle \in \Sigma$, and $c'' \notin g_i.\mathtt{Out}(c')$. Notice that because g_j satisfies the statement of the lemma, it must be the case that $c' = c_i$, and then $c' = ctop_j$. Now, because $\mathtt{EXPAND'_i(c_i, R_i)}$ was called with $R_i = S'_j \cup Into_j$, and $c'' \in Into_j$, we have that $c'' \in R_i$. Hence $c'' \in g_i.\mathtt{Out}(c_n)$, yielding a contradiction.

4. Assume that $g_i.Top = \emptyset$. This instance must have been called by another instance $\mathtt{EXPAND'(c_j, R_j)}$ such that $g_j.Top = \{c_i\}$, otherwise $g_i.Top$ would not be empty (because in Line 2, where $g_i.Top$ is computed, $g_j.Top \setminus \{c_i\} \subseteq g_i.Top$). Now, from Lemma 4 (1), it follows that $g_j.C \subseteq g_j.In^*(c_i)$. On the other hand, because $g_i.Top = \emptyset$, $R_i \subseteq g_j.C$ (again this follows from Line 2 where $g_i.Top$ is computed). Then, we have $R_i \subseteq g_j.In^*(c_i)$. Therefore, $R_i \subseteq g_i.In^*(l_i)$, and because $R_i = g_i.Out(c_i)$, we have a cycle in g_i, which contradicts Lemma 4 (2).

5. It is direct by induction on the length of the paths, that for every path $\tau' = \mathtt{EXPAND}(c, \emptyset) \ldots \mathtt{EXPAND}_m(c_m, R_m)$ of τ, $g_m.Top = g_m.C \setminus C_{\tau'}$. Now, let τ be the path from the root to $\mathtt{EXPAND}_i(c_i, R_i)$, and assume that there exist an instance $\mathtt{EXPAND}_j(c_j, R_j)$ in τ such that $c_i = c_j$; then there must be a sub-path $\tau' = \mathtt{EXPAND}(c, \emptyset) \ldots \mathtt{EXPAND}_i(c_i, R_i) \ldots \mathtt{EXPAND}_k(c_k, R_k)\ \mathtt{EXPAND}_j(c_j, R_j)$ of τ (we assume without loss of generality that $\mathtt{EXPAND}_i(c_i, R_i)$ occurs before than $\mathtt{EXPAND}_k(c_k, R_k)$). Let τ'' be τ' without $\mathtt{EXPAND}_j(c_j, R_j)$. Then, we have $g_k.Top = g_k.C \setminus C_{\tau''}$. But because $\mathtt{EXPAND}_j(c_j, R_j)$ is called by $\mathtt{EXPAND}_k(c_k, R_k)$ in Line 17, $ctop_k = c_j$, and hence $c_j \in g_k.Top$. Then, $c_j = c_i$, and $c_j \in C_{\tau''}$, yielding a contradiction.

•

Lemma 5 (Correctness of `EXPAND`**)** *Given a dimension schema $D = (H, \Sigma)$, and a category c, after an execution of $\mathtt{EXPAND'(c, \emptyset)}$, the variable $Result$ contains all the subhierarchies g of H with root c such that: (a) g has no cycles or shortcuts and (b) for every into constraint $\langle c_1, c_2 \rangle \in \Sigma$, such that c_1 is a category of g we have that (c_1, c_2) is an edge of g.*

Proof of Lemma 5 Let us introduce some notation for the proof. First, we denoted the call graph of $\mathtt{EXPAND'(c, \emptyset)}$ by CG. In addition, let Q be the set containing all the subhierarchies g of H with root c such that: (a) g has no cycles or shortcuts and (b) for every into constraint $\langle c_1, c_2 \rangle \in \Sigma$, such that c_1 is a category of g we have that (c_1, c_2) is an edge of g. Finally, given two subhierarchies g and g' of H, g' is a l-subgraph of g if: $g'.C \subseteq g.C$; and for every category $c \in (g'.C \setminus g'.Top)$, $g.Out(c) = g'.Out(c)$.

We need to prove that $Result = Q$.

First, we will prove that $Result \subseteq Q$. Consider a leaf $\mathtt{EXPAND'_i(c_i, R_i)}$ of CG. From Lemma 4 (1), Lemma 4 (2), Lemma 4 (3), and the fact that $g_i.Top = \{\mathtt{All}\}$, it easily follows that $g_i \in Q$, and hence $Result \subseteq Q$.

We now prove that $Q \subseteq Result$. First, we will prove (i) if there exists an instance $\mathtt{EXPAND'_i(c_i, R_i)}$ in CG where g_i is a l-subgraph of g, and $g_i.C \subset g.C$, then $g_i.Top \neq \{\mathtt{All}\}$. Assume that this is false, i.e., $g_i.Top = \{\mathtt{All}\}$. Now let c' be a category in $g.C$ which is not in $g_i.C$ (note that $c' \neq \mathtt{All}$). Now consider a path $p = c''c_1 \ldots c_n c'$ from some category $c'' \in g_i.C$ to c' in g. Note that `All` cannot be in this path, because there is no category above `All` in g. This path must exist because c' is at least connected from c in g (see definition of subhierarchy). Because g_i is a l-subgraph of g, and $c'' \in g_i.C \setminus g_i.Top = g_i.C \setminus \{\mathtt{All}\}$, we have that $(c''c_1)$ is an edge in g_i, hence $c_1 \in g_i.C$. We repeat this argument with c_1, c_2, instead of c'', c_1, yielding $c_2 \in g_i.C$. Repeating the argument along the path, we reach $c' \in g_i.C$, yielding a contradiction.

Now, we prove (ii) if there is an instance $\mathtt{EXPAND'_i(c_i, R_i)}$ in CG where g_i is a l-subgraph of g, and $g_i.C \subset g.C$, then there is a child $\mathtt{EXPAND'_j(c_j, R_j)}$ of $\mathtt{EXPAND'_i(c_i, R_i)}$ such that g_j is a l-subgraph of g. Because (i) we reach Line 10 in $\mathtt{EXPAND'_i(c_i, R_i)}$, and because Lemma 4 (4), and (i) there is at least one category $ctop$ in $g_i.ctop$, and $ctop \neq \mathtt{All}$. Now, we call $\mathtt{EXPAND'_j(ctop_i, S'_i \cup Into_i)}$, where $S'_i \cup Into_i$ is the set of categories in $g.\mathtt{Out}(ctop_i)$. Note that $Into_i \subseteq g.\mathtt{Out}(ctop_i)$ because $g \in Q$. Now, we have $c_j = ctop_i$, and $R_j = S'_i \cup Into_i$, and we have that g_j is also a l-subgraph of g.

We will prove the existence of a path τ in the execution graph that ends with a leaf $\mathtt{EXPAND'_n}$ (c_n, R_n) of CG, such that g_i corresponds to g. We star from the root $\mathtt{EXPAND'(c, \emptyset)}$ and using (ii) we have the path $\mathtt{EXPAND'(c, \emptyset) EXPAND'_1(c_1, R_1)}$, where g_i is a l-subgraph of g. We repeat this argument until the last instance in the path, say, $\mathtt{EXPAND'_n(c_n, R_n)}$ satisfies

$g_n.C = g.C$. Because g_n is a l-subgraph of g, we have that $g_n = g$. And hence $g_n.Top = \{\texttt{All}\}$. And hence in Line 7 we add g to $Result$.

●

We now prove the correctness of `DIMSAT`.

Theorem 7 (Correctness of `DIMSAT`) *Every execution of* `DIMSAT`(D, c) *stops, and correctly outputs whether c is satisfiable in D.*

Proof of Theorem 7 First, we will prove that `DIMSAT`(D, c) stops. It is enough to show that `EXPAND` when called the first time with c and $\emptyset$ stops. We will show that the call graph of `EXPAND`$(c, \emptyset)$, CG, is finite. It is easy to see that every node `EXPAND`$_i(c_i, R_i)$ of CG has a finite number of children, because the number of calls to `EXPAND` in Line 16 is at most the number of subsets of $Out(ctop_i)$ in H, which is finite. Hence, in order for CG to be infinite, there must exist an infinite path in CG starting from the root `EXPAND`$(c, \emptyset)$. Now, because of Lemma 4 (5), we cannot have an infinite-length path in CG, otherwise the number of categories in the graph would be infinite. Hence CG is finite. Now, we can easily see that each instance of `EXPAND` in CG takes a finite amount of time, plus the time that takes the execution of its descendants. Therefore `DIMSAT` stops.

Now, we show that `DIMSAT`(D, c) correctly outputs whether c is satisfiable in D. First, we prove that if c is satisfiable in D, then `DIMSAT`(D, c) outputs $true$. Assume not, then there is a non-empty set S containing every subhierarchies g of H with root c such that: (a) g has no cycles or shortcuts; and (b) for every into constraint $\langle c_1, c_2 \rangle \in \Sigma$, if c_1 is a category of g, then (c_1, c_2) is an edge of g. Then, because of Lemma 5, we have that `CHECK` is called with every subhierarchy in S. In particular, `CHECK` is called with a subhierarchy $g' \in S$ that induces a frozen dimension in D. Moreover, `CHECK`(g') returns $false$, which contradicts Proposition 3. It remains to prove that if `DIMSAT`(D, c) outputs $true$, then c is satisfiable in D. Assume that `DIMSAT`(D, c) outputs $true$. It is easily verified from Lemma 5 and Proposition 3 that the last graph g with which `CHECK` is called is a subhierarchy of H with root c, and g induces a frozen dimension in ds. Therefore, c is satisfiable in D.

●