Mining Multi-Dimensional Constrained Gradients in Data Cubes*

Guozhu Dong Jiawei Han Joyce Lam Jian Pei Ke Wang

Wright State University, OH, USA
gdong@cs.wright.edu

Simon Fraser University, B.C., Canada
{han, lamd, peijian, wangk}@cs.sfu.ca

Abstract

Constrained gradient analysis (similar to the "cubegrade" problem posed by Imielinski, et al. [9]) is to *extract pairs of similar cell characteristics associated with big changes in measure* in a data cube. Cells are considered similar if they are related by roll-up, drill-down, or 1-dimensional mutation operation. Constrained gradient queries are expressive, capable of capturing trends in data and answering "what-if" questions.

To facilitate our discussion, we call one cell in a gradient pair *probe cell* and the other *gradient cell*. An efficient algorithm is developed, which pushes constraints deep into the computation process, finding all gradient-probe cell pairs in one pass. It explores bi-directional pruning between probe cells and gradient cells, utilizing *transformed* measures and dimensions. Moreover, it adopts a hyper-tree structure and an H-cubing method to compress data and maximize sharing of computation. Our performance study shows that this algorithm is efficient and scalable.

1 Introduction

In recent years, there have been growing interests in multi-dimensional analysis of relational databases, transactional databases, and data warehouses. Most of such analyses involve data cube-based summary or transaction-based association analysis. However, many interesting applications may need to analyze the *changes* of *sophisticated measures* in multidimensional space. For example, one may want to ask what are the *changes* of the *average* house price in the Vancouver area in year 2000 compared against 1999, and the answer could be "the average price for those sold to professionals in the West End went down by 20%, while those sold to business people in Metrotown went up by 10%, etc." Expressions such as "professionals in the West End" correspond to cells in data cubes and describe sectors of the business modeled by the data cube.

The problem of mining *changes* of *sophisticated measures* in a multidimensional space was first proposed by Imielinski, et al. [9] as a *cubegrade* problem, which can be viewed as a generalization of association rules and data cubes. It studies how changes in a set of measures (aggregates) of interest are associated with the changes in the underlying characteristics of sectors, where changes in sector characteristics are expressed in terms of dimensions of the cube and are limited to specialization (drill-down), generalization (roll-up), and mutation (a change in one of the cube's dimensions). For example, one may want to ask "what kind of sector characteristics are associated with major changes at average house price in the Vancouver area in 2000," and the answer will be pairs of sectors, associated with major changes at average house price, including for example "the sector of professional buyers in the West End area of Vancouver" vs. "the sector of all buyers in the entire area of Vancouver" as a specialization (or generalization).

The cubegrade problem is significantly more expressive than association rules since it captures the trends in data and handles arbitrary measures, not just COUNT, as association rules do. The problem is interesting and has broad applications, such as trend analysis, answering "what-if" questions, discovering exceptions or outliers, etc. However, it also poses serious challenges on both understandability of results and on computational efficiency and scalability, as illustrated below.

1. A data cube may have many dimensions. Even though each dimension may involve only a small number of values, the total number of cells of the cube may still be quite huge. In a transactional database, if we consider each item (such as milk or bread) as one independent dimension, as in [9], we may need to handle thousands of dimensions, and the curse of dimensionality will be even worse than that of classical data cubes which usually contain only dozens of dimensions. An effective compromise to this problem is to compute iceberg cubes instead of the complete cubes [3]. To this end, we need to introduce a *significance constraint* for pruning the huge number of trivial cells in the answer set.

* Work supported in part by NSERC and NCE/IRIS-3 of Canada.

**Proceedings of the 27th VLDB Conference,
Roma, Italy, 2001**

2. The cubegrade problem needs to compare each cell in the cube with its associated cells generated by specialization, generalization, and mutation. Even when considering only iceberg cubes, it may still generate a very large number of pairs. Since for each analysis task, a user is often interested in examining only a small subset of cells in the cube, it is desirable to enforce certain *probe constraints* to select a subset of cells (called *probe cells*) from all the possible cells as starting points for examination. By doing so, the study is focused only on these cells and their relationships with corresponding siblings, ancestors, and descendants.
3. Furthermore, a user is usually interested in only certain types of changes between the cells (sectors) under comparison. For example, one may be interested in only those cells whose average profit increases by more than 40% compared to that of the probe cells, etc. Such changes can be specified as a threshold in the form of ratio/difference between certain measure values of the cells under comparison. We will call the cell that captures the change from the probe cell *the gradient cell*, and call such constraints the *gradient constraints*.

Based on this discussion, one can see that to mine interesting gradients in a multidimensional space, it is often necessary to have the following three kinds of constraints: (1) *significance constraint* ensures that we examine only the cells which have certain statistical significance in the data, such as containing at least certain number of base cells or at least certain total sales; (2) *probe constraint*, on dimensional attributes confines the set of probe cells that our gradient analysis will focus on; and (3) *gradient constraint* specifies the user's range of interest on the gradient (i.e., measure change). Enforcing these constraints may lead to interesting, clearly understandable answers as well as possibility to derive efficient methods for gradient analysis in a multidimensional space. In this context, the problem of multidimensional gradient analysis with such constraints represents a confined but interesting version of the cubegrade problem, which we call the *constrained* (*multidimensional*) *gradient analysis*.

The remaining of the paper is organized as follows. Section 2 defines the constrained gradient analysis problem and presents an example. Section 3 discusses the rudimentary algorithm and its deficiencies. Section 4 presents the LiveSet-Driven algorithm, including the techniques for pruning probe cells and gradient cells. A performance analysis is presented in Section 5. Section 6 discusses extensions of our method to deal with various kinds of situations and compares our work with other related works. We conclude our study in Section 7.

2 Problem Definition and Assumptions

Let $\mathcal{D}$ be a relational table, called the *base table*, of a given cube. The set of all *attributes* $\mathcal{A}$ in $\mathcal{D}$ are partitioned into two subsets, the *dimensional attributes* DIM and the *measure attributes* M (so $DIM \cup M = \mathcal{A}$ and $DIM \cap M = \emptyset$). The measure attributes functionally depend on the dimensional attributes in $\mathcal{D}$ and are defined in the context of data cube using any of these five SQL aggregate functions: COUNT, SUM, AVG, MAX, and MIN.

A tuple with schema $\mathcal{A}$ in a multi-dimensional space (i.e., in the context of data cube) is called a **cell**. Given three distinct cells c_1, c_2 and c_3, c_1 is an **ancestor** of c_2, and c_2 a **descendant** of c_1 iff on every dimensional attribute, either c_1 and c_2 share the same value, or c_1 has value "$*$" (where "$*$" indicates "all", i.e., aggregated to the highest level on this dimension); c_2 is a **sibling** of c_3, and vice versa, iff c_2 and c_3 have identical values in all dimensions except one dimension in which neither has value "$*$". A cell which has k non-* values is called a k-**d cell**.

A tuple $c \in \mathcal{D}$ is called a **base cell**. A base cell does not have any descendant. A cell c is an **aggregated cell** iff it is an ancestor of some base cell. For each aggregated cell c, its values on the measure attributes are derived from the complete set of descendant base cells of c.

As mentioned in Section 1, the specification of a constrained gradient analysis problem requires three constraints: a *significance constraint* C_{sig}, a *probe constraint* C_{prb}, and a *gradient constraint* C_{grad}. Both C_{sig} and C_{prb} are unary (defined over cells). A cell c is **significant** iff $C_{sig}(c) = true$, and a cell c is a **probe cell** iff c is significant and $C_{prb}(c) = true$. The complete set of probe cells is denoted as $\mathcal{P}$. The set of significant cells which may have gradient relationship with a set of probe cells, $\mathcal{P}$, are called the **gradient cells** of $\mathcal{P}$.

Significance constraints are usually defined as conditions on measure attributes. These constraints do not have to be anti-monotonic[1], and can be, for example, on a measure defined by AVG. In [8], methods for deriving *weaker anti-monotonic* constraints from non-anti-monotonic constraints and for efficiently computing iceberg cubes were discussed. We will use such weaker anti-monotonic constraints for pruning candidate cells.

We assume that a **probe constraint** is in the form of an SQL query, which will "select" a set of user-desired cells.

A **gradient constraint** is binary (defined over pairs of cells). It has the form $C_{grad}(c_g, c_p) \equiv (g(c_g, c_p)\ \theta\ v)$, where θ is in $\{<, >, \geq, \leq\}$, v is a constant value, and g is a *gradient function*. $g(c_g, c_p)$ is defined iff c_g is either an ancestor, a descendant, or a sibling of c_p. A gradient cell c_g is **interesting** with respect to a probe cell $c_p \in \mathcal{P}$ iff c_g is significant and $C_{grad}(c_g, c_p) = true$.

In this paper we mainly consider gradient constraints defined using the ratio of two measure values such as "$m(c_g)/m(c_p)\ \theta\ v$", where $m(c)$ is a measure value for a cell c. Most of the results derived for ratio can be easily extended to difference, "$m(c_g) - m(c_p)\ \theta\ v$" (see Section 6).

[1] Anti-monotonicity is very useful for pruning. It states that *if a cell c does not satisfy an (anti-monotonic) significance constraint C_{sig}, none of c's descendants can do so.* For example, the constraint "$count > 10$" is anti-monotone. Anti-monotonicity-based pruning forms the foundation for most algorithms for computing iceberg cubes.

Problem definition. Given a base table $\mathcal{D}$, a significance constraint C_{sig}, a probe constraint C_{prb}, and a gradient constraint $C_{grad}(c_g, c_p)$, the **constrained gradient analysis** problem is to *find all the* interesting *gradient-probe pairs* (c_g, c_p) *such that* $C_{grad}(c_g, c_p) = true$. □

Example 1 (Constrained average gradient) Let the *base table* $\mathcal{D}$ be a sales table with the schema

$$sales(year, city, cust\_grp, prod\_grp, cnt, avg\_price).$$

Attributes $year$, $city$, $cust\_grp$, and $prod\_grp$ are the *dimensional attributes*; and cnt and avg_price are the *measure attributes*.

c_1	(00, Vancouver, Business, PC, 300, \$2100)
c_2	$(*, Vancouver, Business, PC, 2800, \$1900)$
c_3	$(*, Toronto, Business, PC, 7900, \$2350)$
c_4	$(*, *, Business, PC, 58600, \$2250)$

Table 1: A set of base and aggregated cells

Table 1 shows a set of base and aggregated cells. Tuple $c_1 \in \mathcal{D}$ is a *base cell*, while tuple c_2 is an *aggregated cell*. Tuple c_3 is a sibling of c_2, c_4 is an ancestor of c_2, and c_1 is a descendent of c_2.

Suppose the significance constraint is $C_{sig} \equiv (cnt \geq 100)$. All cells (including base and aggregated ones) with cnt no less than 100 are regarded as *significant*. Suppose the probe constraint is $C_{prb} \equiv (city = \text{“}Vancouver\text{”}, cust\_grp = \text{“}Business\text{”}, prod\_grp = *)$. The set of *probe cells* $\mathcal{P}$ contains the set of aggregated tuples about the sales of the Business customer group in Vancouver, *for every product group*, provided the cnt in the tuple is greater than or equal to 100. It is easy to see $c_2 \in \mathcal{P}$.

Let $C_{grad}(c_g, c_p) \equiv (avg\_price(c_g)/avg\_price(c_p) \geq 1.4)$. The constrained gradient analysis problem is to find all pairs (c_g, c_p), where c_p is a probe cell in $\mathcal{P}$, c_g is a sibling, ancestor, or descendant of c_p, c_g is a significant cell, and c_g's average price is at least 40% more than c_p's. □

If a data cube is completely materialized, the query posed in Ex. 1 becomes a relatively simple retrieval of the pairs of computed cells that satisfy the constraints. Unfortunately, the number of aggregated cells is often too huge to be precomputed and stored. Thus we assume only the base table is available, and it is our task to compute from it the gradient-probe pairs efficiently.

3 The All-Significant-Pairs Algorithm

In this section, we introduce a rudimentary algorithm: **All-Significant-Pairs**, which first computes iceberg cube $\mathcal{I}$ from $\mathcal{D}$ using the significance constraint C_{sig}, then applies the probe constraint C_{prb} to select the set of significant probe cells, $\mathcal{P}$, from $\mathcal{I}$, and finally for each probe cell, $c_p \in \mathcal{P}$, computes the set of gradient cells on $\mathcal{I}$ by enforcing the gradient constraint $C_{grad}(c_g, c_p)$.

Further optimization can be explored to prune the search for ancestors and/or descendants of a probe cell c_p based on the anti-monotonic relationships between them. If the gradient measure is an anti-monotonic function, one can explore the following property: *if the measure m of a cell c is no greater than τ, none of c's descendants can have the measure m greater than τ; and if the measure m of a cell c is no less than τ, none of c's ancestor can have the measure m less than τ*. If the gradient measure is not an anti-monotonic function, such as average and sum of positive or negative elements, one can explore a weaker but anti-monotonic constraint to prune its ancestors and/or descendants. For example, for average, one can explore the property of top-k average [8]: *if the top-k average of the base cells in a cell c is no greater than τ, where k is the significance constraint threshold value, then none of c's significant descendants can have the $average$ value greater than τ*. Similarly, one can derive many other interesting properties to facilitate pruning for constraints involving some complex measures.

Example 2 (All-Significant-Pairs) Let's examine how to perform constrained gradient analysis for the problem specified in Ex. 1, using the *All-Significant-Pairs* method.

First, compute all the significant aggregate cells in the data cube by enforcing the significance constraint $C_{sig} \equiv (cnt \geq 100)$ and applying an efficient iceberg cube computation algorithm, such as H-Cubing [8]. This will yield a set of cells, e.g. $c_1 = (00, Vancouver, Business, PC, 300, \$200)$, $c_2 = (*, Vancouver, Business, PC, 800, \$150)$, $c_3 = (*, Toronto, Business, PC, 900, \$250)$, $c_4 = (*, *, Business, PC, 8600, \$225)$, and so on. Let this computed iceberg cube from $\mathcal{D}$ be $\mathcal{I}$. The gradient computation will then be performed on $\mathcal{I}$ only.

Second, apply the probe constraint C_{prb} to select the set of significant probe cells, $\mathcal{P}$, from $\mathcal{I}$. For each probe cell, $c_p \in \mathcal{P}$, compute the set of gradient cells on $\mathcal{I}$ by enforcing the gradient constraint $C_{grad}(c_g, c_p)$, and performing possible pruning of ancestors and/or descendants of the gradient cell currently under examination. For example, suppose our search is from top-level down (i.e., first computing high-level cells and then their descendants). If a cell c_g's top-k average value[2] (where $k = 100$, the minimum support threshold, i.e., significance constraint) is no more than $c_p \times 1.4$, then c_g and all of its descendants can be pruned since none of them can satisfy the gradient constraint. □

The algorithm All-Significant-Pairs is not presented here due to lack of space. The method computes the *complete* set of significant cells in the iceberg cube using C_{sig}. However, when the probe constraint is sharp, only a small portion of such iceberg cube cells will be useful in the derivation of constrained gradients. For example, if the probe set contains only one cell c_p, the constrained gradient analysis will need to analyze only c_p's siblings, descendents, and ancestors. Moreover, the search for gradient cells is done in a one-search-loop-per-probe-cell fashion. A

[2] The efficient computation of top-k average has been discussed in [8].

huge amount of repeated work is performed for probe cells which are similar. It may involve computing the set of gradient cells $|\mathcal{P}|$ times, where $|\mathcal{P}|$ is the number of probe cells in $\mathcal{P}$, which is costly.

4 The LiveSet-Driven Algorithm

In this section we present a better algorithm, *LiveSet-Driven*, which overcomes the deficiencies of the All-Significant-Pairs algorithm. Its main spirit is to use a set of relevant probe cells, called $LiveSet$, to prune potential gradient cells. The technical issues include how to derive "tighter" $LiveSet$ and how to use it for pruning.

Here we present an overview of the method. To avoid the waste of resource for computing cells unrelated to probe cells, we first compute the set of iceberg probe cells $\mathcal{P}$ from $\mathcal{D}$, using both the significance and probe constraints. The second step utilizes the set of derived iceberg probe cells $\mathcal{P}$ to efficiently constrain the search for interesting gradient-probe cell pairs. This is similar to the golden rule of pushing selection deeply in relational query processing. To make the computation of the second step efficient, several techniques are developed as outlined below.

1. Using sets of probe cells to constrain the processing: To avoid the costly repetition of computation in the *All-Significant-Pairs* algorithm, set-oriented processing is explored. Roughly speaking, we associate with each gradient cell the set of all possible probe cells that might co-occur in interesting gradient-probe pairs with some descendants of the gradient cell, and use that set to prune future gradient cell search space.
2. Iceberg growth from low to high dimensions: The multi-dimensional space should be explored in a progressive and confined manner, using an "iceberg growth approach": Start at lower dimensional cells and proceed to higher ones. There are usually a smaller number of lower dimensional cells than that of the higher dimensional ones. The anti-monotonicity property of significance constraints (or their weaker versions) and the (transformed) gradient cell constraints can be used to prune the remaining search space: if a k-d cell fails to satisfy a constraint, so will all of its descendants (higher dimensional cells). All three types of constraints, i.e., the probe, significance and gradient constraints, are used in this iceberg growth process.
3. Dynamic pruning of probe cells during the growth: During dimension growth, increasingly more probe cells fail to be associated with the higher dimensional gradient cells due to dimension value mismatch or the relevant measure value being out of the gradient range. Thus, one can prune the set of probe cells associated with the gradient cells in the growth. The search terminates when either no significant gradient cells can be generated or none of the probe cells can proceed further. The pruning of probe cells increases the power to prune gradient cells.
4. Incorporation of compressed data structure, H-tree, and efficient iceberg growth algorithm, H-cubing: For efficient computation of iceberg cubes, we also incorporate a compressed data structure, H-tree, and extend an efficient iceberg growth algorithm, H-cubing. This data structure and algorithm were shown to be highly efficient for computing iceberg cubes with complex measures [8]; they allow us to do maximal sharing between cells in the computation. This further enhances the efficiency of constrained gradient analysis.

4.1 Pruning gradient cells and probe cells using gradient constraints

Suppose $\mathcal{P}$, the set of probe cells, has been computed. The next step in the computation is to determine which cell should be associated with which probe cell to produce valid gradient-probe pairs. The computation will start from low dimensions and proceed to higher dimensions, in a depth-first manner. Information on low dimension gradient cells will be used to prune higher dimension cells.

Definition 1 *The* **live set** *of a gradient cell c_g, denoted as $LiveSet(c_g)$, is the set of probe cells c_p such that it is possible that $(c_{g'}, c_p)$ is an interesting gradient-probe pair, for some descendant cell $c_{g'}$ of c_g.*

From this definition it is clear that the smaller $LiveSet$ is, the more gradient cells can be pruned. The determination of $LiveSet$ involves the gradient constraint and the matches between dimensions of gradient and probe cells. This section only deals with the former, and the next section deals with the latter.

Interestingly, pruning can be done in both directions between $LiveSet(c_g)$ and c_g: (1) $LiveSet(c_g)$ can be used to determine if c_g and its descendants have the potential to be interesting gradient cells w.r.t. (any probe cell in) $LiveSet(c_g)$; if not, c_g can be pruned. (2) Information about c_g can also be used to prune probe cells c_p in $LiveSet(c_g)$. This involves checking whether c_g and its descendants have the potential to be interesting gradient cells w.r.t. c_p. If the answer is no, c_p can be pruned from the $LiveSet(c_g)$.

Definition 2 *Let c_g be a gradient cell, C_p a set of probe cells, and C_{grad} the gradient constraint. We say c_g and its descendants have* potential *to be interesting gradient cells w.r.t. C_p if the following is true:*

(1) If the gradient constraint is anti-monotone (such as sum), then $C_{grad}(c_g, c_p)$ is satisfied for some $c_p \in C_p$.

(2) If the gradient constraint is not anti-monotone, such as $(avg\_price(c_g)/avg\_price(c_f) \geq v)$, then a transformed, weaker constraint can be potentially satisfied for some $c_p \in C_p$, such as $(avg^k\_price(c_g)/avg\_price(c_p) \geq v)$, where avg^k represents top-k average and k is the minimum support threshold (i.e., significance constraint). Observe that the

avg^k constraint is a weaker anti-monotonic constraint constructed for the non-anti-monotonic avg constraint.

We say a gradient cell c_g is a **potential cell**, *or* **has potential to grow**, *if (i) c_g is significant and (ii) c_g and/or its descendants have* potential *to be interesting gradient cells w.r.t. $LiveSet(c_g)$.*

Observation. Some non-antimonotonic constraint can be transformed into a weaker, anti-monotonic constraint for pruning. For (2) above, we use $avg^k\_price(c_g)$ as an upper estimate of $avg\_price(c_{g'})$ for all significant descendant cells $c_{g'}$ of c_g.

Example 3 Using the schema of Ex. 1, suppose $C_{grad}(c_g, c_p) \equiv (avg\_price(c_g)/avg\_price(c_p) \geq 1.4)$. Assume the set of probe cells $\mathcal{P}$ has been derived using the two constraints C_{sig} and C_{prb}. Let c_g be the 1-d cell $(00, *, *, *)$, which is assumed to be significant.

Suppose that initially[3] $LiveSet(c_g)$ is the following subset $\{c_{p_1}, c_{p_2}, c_{p_3}\}$ of $\mathcal{P}$, where c_{p_1} = (00, "$Vancouver$", "$Business$", $*$, 2800, \$1500), c_{p_2} = (99, "$Toronto$", $*$, PC, 7900, \$3000), and c_{p_3} = (00, "$Toronto$", "$Education$", PC, 450, \$2000).

We examine two scenarios: (i) $avg^k\_price(c_g)$ = \$2500. Since $avg^k\_price(c_g)/avg\_price(c_{p_1}) = 2500/1500 > 1.4$, c_g has potential to grow. However, because $2500/3000 < 2500/2000 < 1.4$, c_{p_2} and c_{p_3} can both be pruned from $LiveSet(c_g)$. (ii) $avg^k\_price(c_g)$ = \$2000. Since $avg^k\_price(c_g)/avg\_price(c_p) < 1.4$ for each $c_p \in LiveSet(c_g)$, c_g does not have potential to grow, and can thus be pruned. □

Let's consider how to use a set C_p of probe cells to prune gradient cells, where $avg\_price(c_p)$ is known for every c_p in C_p. Given a gradient cell c_g, clearly it is not efficient to check against all individual probe cells c_p in $LiveSet$ whether the condition $avg^k\_price(c_g)/avg\_price(c_p) \geq 1.4$ holds. Fortunately, one can derive an overall gradient cell constraint for set C_p, $C_{gcell}(C_p)$, which specifies a range of measure values (such as average prices) for c_g and which must be satisfied by a gradient cell c_g if c_g *might co-occur in interesting gradient-probe pairs with any probe cell in C_p*. In general, we have the following:

Property 4.1 (gradient cell constraint for a set of probe cells) If $C_{grad} \equiv (m(c_g)/m(c_p)\ \theta\ v)$, where θ is in $\{<, >, \geq, \leq\}$, v is a constant value, and $m(c_p) > 0$, then the gradient cell constraint corresponding to a set of probe cells C_p is $C_{gcell}(C_p)$, where

$$C_{gcell}(C_p) \equiv \begin{cases} m(c_g)\ \theta\ v \times min\{m(c_p) | c_p \in C_p\} \\ \qquad if\ \theta\ \in \{>, \geq\} \\ m(c_g)\ \theta\ v \times max\{m(c_p) | c_p \in C_p) \\ \qquad if\ \theta\ \in \{<, \leq\} \end{cases} \quad (1)$$

□

[3]The next section will discuss how $LiveSet$ is derived.

This property can be used to derive a gradient cell constraint from a set of probe cells.

4.2 Pruning probe cells by dimension matching analysis

In this subsection, we describe what probe cells should be associated with a gradient cell, and how to prune the associated probe cells when the processing goes from a gradient cell to a descendant one; both will be from a dimension-matching perspective.

The dimension matching analysis is made possible under the assumption that we are only interested in gradient-probe pairs involving ancestor-descendant, descendant-ancestor, and sibling-sibling pairs.

Let c_g be a gradient cell. Recall that $LiveSet(c_g)$ denotes the set of probe cells c_p such that it is possible that $(c_{g'}, c_p)$ is an interesting gradient-probe pair for some descendant cell $c_{g'}$ of c_g. Hence, from a dimensional perspective, a probe cell c_p can be in $LiveSet(c_g)$ if (i) c_p is an ancestor or descendant of c_g, or c_g itself; or (ii) c_p is a sibling of some descendant of c_g or a sibling of c_g. It turns out that these conditions can be captured by a notion of "matchable," defined next.

Let $c_p = (d_{t1}, d_{t2}, \ldots, d_{tm})$ be a probe cell and $c_g = (d_{g1}, d_{g2}, \ldots, d_{gm})$ be a gradient cell. The number of **solid-mismatches** between the two cells c_p and c_g is the number of dimensions in which both values are not $*$ but are not matched (i.e., of different values). The number of **$*$-mismatches** between c_p and c_g is the number of dimensions in which c_p is $*$ but c_g is not. (Observe that the notion of $*$-mismatches is not symmetric and the cells are *playing certain roles*.) A probe cell c_p is **matchable** with a gradient cell c_g if either c_g and c_p have no solid-mismatch, or they have exact one solid-mismatch but no $*$-mismatch.

Example 4 Consider the 4-d probe cell $c_p = (a, b, *, d)$. c_p is matchable with its ancestor gradient cell $c_{g1} = (*, *, *, d)$ since c_{g1} contains neither $*$-mismatch nor solid-mismatch; c_p is matchable with its sibling $c_{g2} = (f, b, *, d)$ since c_{g2} contains only one solid-mismatch but no $*$-mismatch; c_p is matchable with $c_{g3} = (*, g, *, d)$ since c_{g3} contains one solid-mismatch but no $*$-mismatch; c_p is matchable with $c_{g4} = (a, *, c, d)$ since c_{g4} contains no solid-mismatch; and also c_p is matchable with its descendant $c_{g5} = (a, b, c, d)$ since c_{g5} contains only one $*$-mismatch. However, it is not matchable with $c_{g6} = (*, c, e, d)$ since c_{g6} contains one solid-mismatch and one $*$-mismatch. As illustrated above, c_p is the sibling of a descendant of the gradient cell (c_{g3} or c_{g4}). □

Property 4.2 (correctness of dimension analysis) c_p is matchable with c_g iff c_p is c_g, an ancestor/descendant of c_g, or it is a sibling of c_g or of some descendant of c_g.

Rationale. For the "only if": Suppose c_p is matchable with c_g. Two cases arise: (a) c_g and c_p have no solid-mismatch. Let c' be obtained by taking the more specific value, for

each dimension, from c_g and c_p. (Non-* values are not comparable, and each non-* value is more specific than the * value.) Then c' is a descendant of c_g and c' is a descendant of c_p. Hence c_p is an ancestor of some descendant of c_g. There are special cases here: if $c' = c_g$, then c_p is an ancestor of c_g; if $c' = c_p = c_g$, then c_p is c_g. (b) c_g and c_p have exactly one solid-mismatch but no $*$-mismatch. Let c' be obtained by taking the more specific value, for each dimension, from c_g and c_p, except that c' takes the value of c_g for the dimension of the solid-mismatch. So c' is a descendant of c_g. Since there is no *-mismatch between c_p and c_g, each of the specific value also occurs in c_p. Clearly c_p and c' have exactly one solid-mismatch, and so c_p is a sibling of c'. Observe that c' can be c_g; in that case c_p is a sibling of c_g.

We omit the details of the "if." The non-trivial cases are illustrated in Ex. 4. □

We now discuss how dimension analysis is used for pruning $LiveSet$ when the processing goes from a gradient cell to a descendant one.

Property 4.3 (relationship between livesets of ancestor-descendant cells) Let c_{g1} and c_{g2} be two gradient cells such that c_{g2} is a descendant of c_{g1}. Then $LiveSet(c_{g2}) \subseteq LiveSet(c_{g1})$.

Rationale. Let c_p be a probe cell such that (c_{g3}, c_p) might exist as an interesting gradient-probe cell pair for some descendant cell c_{g3} of c_{g2}. Since c_{g3} is a descendant of c_{g1} as well, the fact in the last statement implies that c_p is also in $LiveSet(c_{g1})$. □

This property ensures that we can produce the $LiveSet$ of a descendant cell from that of the ancestor cell. The way to do that is simply to do a dimension matching analysis, plus a gradient-based pruning. We illustrate the dimension-matching based pruning using the following example.

Example 5 Let $c_{g1} = (*, *, c, *)$ be a gradient cell and let $c_{g2} = (*, b, c, *)$, which is a descendant of c_{g1}. Suppose $LiveSet(c_{g1}) = \{(*, *, *, *), (a, b, c, *), (*, b1, c, *), (a, b1, c, *), (*, b1, c1, *)\}$. Then $LiveSet(c_{g2}) = \{(*, *, *, *), (a, b, c, *), (*, b1, c, *), (a, b1, c, *)\}$, i.e., it is the result of pruning $(*, b1, c1, *)$ from $LiveSet(c_{g1})$. □

Notice that if we consider the expansion of gradient cells following a particular order, more pruning of the probe cells can be done. For example, if the dimensions are expanded from left to right, some descendants of c_g will be processed before c_g is processed (observe that the ancestor-descendant relationship is many-to-many). A technical definition capturing such traversal-order dependent pruning was obtained in our own study, but is omitted here.

In this study, we assume that the set of probe cells, and hence the $LiveSet$, is usually a small set, which can be sorted in value ascending order according to certain measure values (see the next subsection) to facilitate pruning using gradient constraint. In case there is a large set, tree structure or hash table can be adopted for fast accessing.

4.3 The LiveSet-Driven Algorithm

Based on the above discussion, the $LiveSet$-driven algorithm is worked out for computing all the gradient-probe pairs which satisfy all the constraints. Our method starts with the 0-d cell of the cube, carrying the initial set of probe cells, $\mathcal{P}$, as its $LiveSet$, and proceeds to higher dimensional gradient cells. Along the way, it uses the given constraints to prune the gradient cells which cannot satisfy the $LiveSet$, and to prune the cells in the $LiveSet$ which cannot pass either gradient constraints or dimensional matching analysis. The processing along any branch terminates when the $LiveSet$ becomes empty, or when the gradient cell has no potential to generate any interesting pairs.

Let's examine an example in more detail.

Example 6 (LiveSet-Driven) For the same base table schema $\mathcal{D}$ in Ex. 1, we examine how to perform constrained gradient analysis by the *LiveSet-Driven* algorithm. Let the gradient constraint be $C_{grad}(c_g, c_p) \equiv (ave\_price(c_g)/avg\_price(c_p) \geq 1.2)$, and the significance constraint be $C_{sig} \equiv (cnt \geq 100)$.

Let the set $\mathcal{P}$ of probe cells be given in Table 2, sorted in avg_price ascending order. Notice this order is important since once a probe cell in the table cannot satisfy the gradient constraints, all the cells following it cannot satisfy it either (since they carry an even larger measure value) and thus can all be pruned immediately.

$(00, Vancouver, Education, PC, 100, 1500)$
$(99, Toronto, *, PC, 4000, 1800)$
$(*, Montreal, Business, PC, 1500, 8000)$
$(*, Edmonton, *, Ski, 2000, 10000)$
$(*, Whisler, *, Ski, 1000, 10050)$

Table 2: The set of probe cells, $\mathcal{P}$.

The set of all probe cells $\mathcal{P}$ is the initial $LiveSet$ for the 0-d gradient cell $c_0 = (*, *, *, *)$. Since 1500 is the lowest avg_price value among all current probe cells, it is taken as the global gradient lower bound. Suppose the top-100 average of the 0-d cell c_0 is 4000 and its count is 50000. Then c_0 has potential to grow, because $4000 \geq 1.2 \times 1500 = 1800$ and $50000 \geq 100$. Now, the top-100 average of c_0 is used to prune the probe cells to generate a tighter $LiveSet$ for c_0: Since the fourth cell $(*, Edmonton, *, Ski, 2000, 10000)$ cannot satisfy the gradient constraint due to $4000 < 1.2 \times 10000$, this cell and all the remaining in the $LiveSet$ will be pruned. The actual average value of c_0 will decide which probe cell will be paired with this cell to become an interesting gradient-probe pair.

The computation then proceeds to process 1-d cells, 2-d cells, and so on, in a depth first manner. To avoid repetition and for the sake of clarity, we now show how the processing is done for a typical 3-d cell.

Suppose the first three probe cells are all alive after processing the 2-d gradient cell $c_2 = (00, Toronto, *, *)$, and the processing goes from this 2-d cell to the 3-d cell $c_3 = (00, Toronto, *, PC)$.

Probe cell	# of mismatches
$(00, Vancouver, Edcationu, PC, 100, 1500)$	1
$(99, Toronto, *, PC, 4000, 1800)$	1
$(*, Montreal, Business, PC, 1500, 8000)$	$1, 1*$

Table 3: Number of mismatches in probe cells.

We first prune the $LiveSet$ of c_2 using dimensionality matching with c_3. The number of mismatches of each probe cell w.r.t. c_3 is presented in Table 3, where 1 indicates that there is one solid mismatch, and $1*$ indicates that there is one *-mismatch. Table 3 indicates that the first two probe cells remain alive w.r.t. the 3-d gradient cell c_3.

The actual average value of c_3 decides which probe cell should be paired with this cell to become an interesting gradient-probe pair. If $avg\_price(c_3) = 1850$, then c_3 and the first probe cell form an interesting gradient-probe cell pair, but not c_3 and the second.

The minimum average of the cells in $LiveSet$, 1500, and the top-100 average of c_3, will decide if the processing should continue with c_3's descendants. If the top-100 average of c_3 is less than $1800 = 1.2 * 1500$, computation stops for this branch. Otherwise, it continues. Suppose the top-100 average of c_3 is 1900. Then we go back to prune the current $LiveSet$ of c_3. Because $1900 < 1800 * 1.2$, we can indeed prune the second probe cell, $(99, Toronto, *, PC, 4000, 1800)$, from the $LiveSet$.

In summary, the processing of a gradient cell c involves: (1) derive an initial $LiveSet$ from the $LiveSet$ of the ancestor of the cell c by dimension matching, (2) compute the necessary measures and top-k average measures of c, check them against the $LiveSet$ for answers, and decide if the descendants of c may require processing. (3) if processing of descendants is needed, prune $LiveSet$ using the gradient constraint and the top-k average values. □

We now present the LiveSet-Driven algorithm.

Algorithm 1 (LiveSet-Driven)

Input: A base relational table $\mathcal{D}$, a significance constraint C_{sig}, a probe constraint C_{prb}, and a gradient constraint C_{grad}.

Output: The complete set of gradient-probe pairs in the data cube derived from $\mathcal{D}$ that satisfy the three constraints.

Method:

1. Apply an iceberg cube computation algorithm to compute the set of iceberg probe cells $\mathcal{P}$ from $\mathcal{D}$ using significance constraint C_{sig} and probe constraint C_{prb};
2. Derive gradient cell constraint C_{gcell} for $\mathcal{P}$;
3. Initialize the potential gradient cell to $c = (*, ..., *)$. Initialize $LiveSet(c) = \mathcal{P}$.
4. Use a bottom-up, depth-first iceberg cubing method (we use H-cubing but other methods can be employed, e.g. BUC) to find all interesting gradient-probe pairs. In depth-first processing, values in each dimension are ordered, and the dimensions are also ordered.

   for every value in each dimension do {

   1 If c is significant, for each live probe cell c_p in $LiveSet(c)$, output the gradient-probe pair (c, c_p) if the pair passes the gradient cell constraint.

   2 Use the measure (or transformed measure such as top-k) value of c to prune $LiveSet(c)$.

   3 If $LiveSet(c)$ is empty or c has no potential to grow, terminate this branch and backtrack to process the next cell according to the depth-first order.

   4 If c has potential to grow, expand it to the next level, according to the depth-first order.
   If a descendant cell c' of c is processed from this expansion, derive $LiveSet(c')$ from $LiveSet(c)$ using the matchability test.

   } □

5 Performance Analysis

In this section, we report our experimental results on computing gradients in data cubes.

All experiments were conducted on a PC with an Intel Pentium III 700MHz CPU and 256M main memory, running Microsoft Windows/NT. All programs were coded in Microsoft Visual C++ 6.0. The experiments were conducted on synthetic data sets generated using the data generator described in [8]. The results are similar. Limited by space, except for performance with respect to the number of tuples, we report here only results on some typical data sets, with 10 dimensions and between $10,000$-$20,000$ tuples. The cardinality for every dimension is set to 10[4]. The measures are in range of $[100, 1000]$. The noise factor is set to 20% and repeat factor is 200.

The first data set we use has $10,000$ tuples. We test the scalability of the algorithms with respect to number of probes in Figure 1, significance threshold in Figure 2, and gradient threshold in Figure 3.

Figure 1 shows the scalability of the two algorithms, All-Significant-Pairs and LiveSet-Driven, with respect to the number of probe cells. We set the significance threshold to 10, the number of bins to 3 for top-k average, and the gradient threshold is 1.2. The number of probes varies from 1 to $1,000$. When the number of probes is small, both algorithms have similar performance. However, as the number of probes grows, the pruning power of LiveSet-Driven algorithm takes effect. It prunes unfruitful searches and keeps the runtime low. In contrast, the All-Significant-Pairs algorithm does not scale well under large number of probes.

[4]The smaller the cardinality, the denser the data cube, and thus the larger number of cells satisfy the constraints.

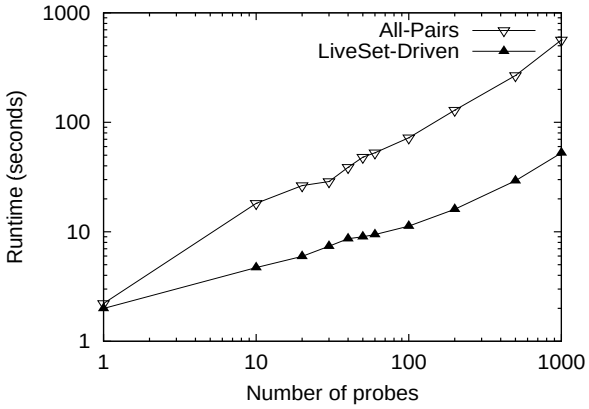

Figure 1: Scalability over number of probe cells

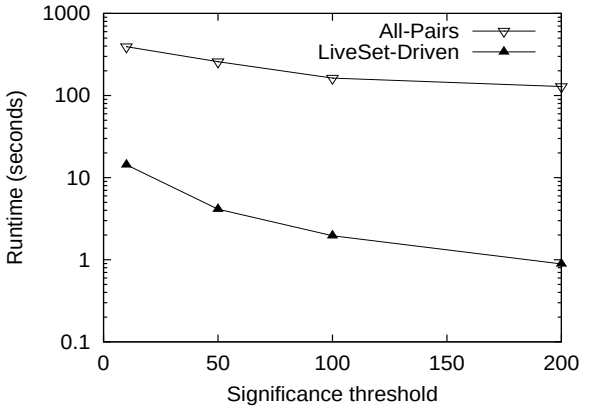

Figure 2: Scalability w.r.t. significance threshold

Figure 2 shows the scalability of both algorithms with respect to the significance threshold. The gradient threshold is set to 1.2, the number of bins to 3 and the number of probes to 50. LiveSet-Driven achieves good scalability by pruning many cells in the search whereas All-Significant-Pairs checks a huge number of pairs of cells, thus requires exponential runtime.

Figure 3 shows the scalability of All-Significant-Pairs and LiveSet-Driven with respect to various gradient thresholds. We fix the significance threshold to 10, number of bins to 3 and number of targets to 50. As the gradient threshold goes down, the number of cells that All-Significant-Pairs has to check increases dramatically, and thus its runtime increases dramatically as well.

Figure 4 shows a scaling-up experiment with respect to various number of tuples, varying up to $20,000$. We set the significance threshold to 1% of the number of tuples, the gradient threshold to 2, the number of bins to 3 and the number of probes to 100. While both algorithms are scalable, LiveSet-Driven naturally is more efficient.

We also analyzed the number of cells explored by each algorithm during the mining process on a $10,000$-tuple

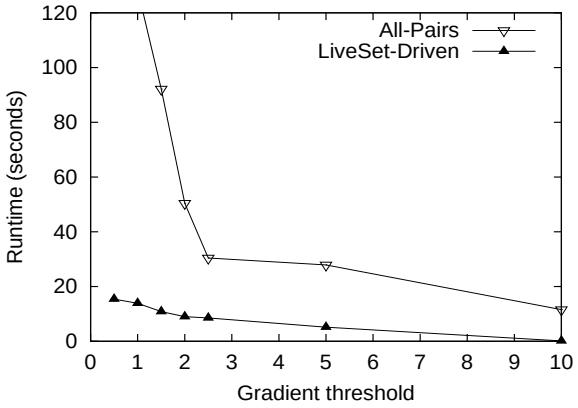

Figure 3: Scalability w.r.t. gradient threshold

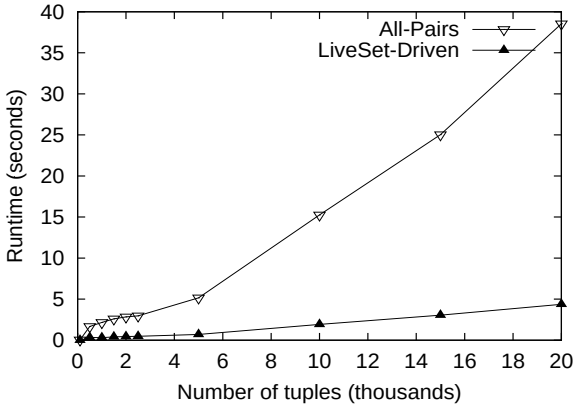

Figure 4: Scalability w.r.t. number of tuples

dataset with 50 probe cells. Figure 5 presents the number of cells that the two algorithms explored with respect to various gradient thresholds. It confirms that LiveSet-Driven achieves better pruning than All-Pairs. As shown in the figure, LiveSet-Driven on average explores only about one tenth of the cells All-Significant-Pairs does. That explains the difference of efficiency and scalability between the two algorithms.

Similar statements can be made about Figure 6, where the significance threshold varies from 10 to $1,000$. LiveSet-Driven explores a substantially smaller subset of cells that All-Significant-Pairs examines.

6 Discussion

Here we examine some possible extensions or refinements of the method and compare our study with related works.

6.1 Possible extensions of the method

1. Mining constrained gradients for more restricted relationship or under a subset of dimensions.

Our method searches for ancestors, descendants, and siblings at the same time. In some applications, people

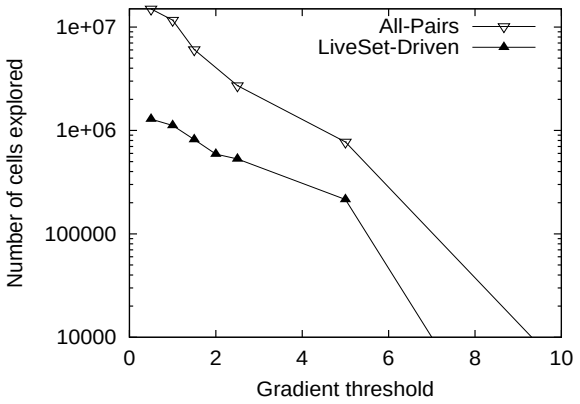

Figure 5: Using gradient in pruning

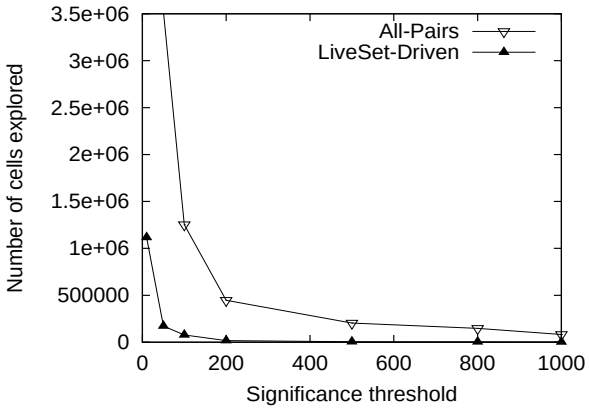

Figure 6: Significance threshold and pruning

may be interested in only one or two kinds but not all kinds. This can be easily addressed by modifying the definition of $LiveSet$ for potential gradient cells. For each special case, more probe cells will be removed (and thus more efficient) than the general case, and there is no need to change other parts of the algorithm.

Similar extensions can be worked out if a user would like to find the constrained gradients only in relevance to a small subset of dimension combinations, such as $\{D_i, \ldots, D_j\}$ in a data cube. In this case, starting with the 0-d cell, $(*, \ldots, *)$, the set of (non-*) gradient cells to be considered and tested will be confined to only those in a subset of dimensions $\{D_i, \ldots, D_j\}$.

2. Finding multi-dimensional gradients constrained by an "interval".

Our algorithm searches for multi-dimensional gradients by checking a single gradient constraint, such as $C_{grad}(c_g, c_p) \equiv (g(c_g, c_p)\ \theta\ v)$, where θ is in $\{<, >, \geq, \leq\}$, v is a constant value, and g is a *gradient function*. In many cases, the desired constraint could be an interval, such as $1.4 \leq g(c_g, c_p) \leq 2.5$. In such cases, one can modify the gradient testing part of the algorithm by testing not only the lower bound on the top-k average of the measure (i.e., no less than $1.4 \times avg(c_p)$) but also the upper bound on the bottom-k average of the measure (i.e., no more than $2.5 \times avg(c_p)$). Whether it is more efficient to prune the search space using both upper and lower bounds or using only one of them and postponing the evaluation of the other after the constraint evaluation will depend on the gradient constraint values and the data set.

3. Replacing ratio-based gradients by differences as "gradient" constraint.

Although ratio-based gradients are handled in our algorithm, with slight modifications, we can handle gradients defined with differences, such as,

$$C^{dif}_{grad}(c_g, c_p) \equiv (avg\_price(c_g) - avg\_price(c_p) \geq 400).$$

Similarly, our algorithm can easily be extended to find similar patterns (i.e., measures which are similar when some dimension values change).

4. What will happen if avg is replaced by sum or count?

The measure "average" has been used in our gradient analysis since it is natural to define interesting gradients as substantial changes on "average". However, if avg is replaced by sum or count, an ancestor cell should naturally have much bigger sum or count values than its descendants. The simple gradient definition, such as $g(c_g, c_p) \geq v$, may not be so interesting, and some "normalized" definition of gradients, will make more sense. Our algorithm can also be made to work with corresponding modifications.

5. Extension of our model from cube to transaction-based association rules.

Our model, though studied in the context of data cubes, can be extended to mining transaction-based association rules with complex measures. A simple method is to consider each distinct item in a transaction as one dimension and consider the average sales or price as a complex measure. Our model and algorithm are still applicable. However, the "curse of dimensionality" poses challenges on efficiency, and further studies are needed to improve performance.

6.2 Related work

The closest work related to our study is that on the cubegrade problem by Imielinski, et al. [9]. A cubegrade query asks for association-type rules that describe changes in measure values associated with changes in dimension descriptions of cuboids. It deals with questions such as "what cube changes are associated with significant measure changes." Cubegrade queries can also have constraints that restrict the attributes in the gradient cells, other than those allowed by roll-up, drill-down, and mutation. Our constrained gradient analysis does not have user-defined constraints on gradient cells. However, they can be easily dealt with by adding more power to prune $LiveSet$. Thus adding user-defined constraints will actually lead to more efficient processing.

The main contributions of [9] are the cubegrade framework and the proposed language. It considered a relativized notion of monotonicity (w.r.t. a cube or a constrained cube), the so-called structural monotonicity, which can be tested quite efficiently. Similar to our *all-significant-pairs* approach, the evaluation strategy proposed in [9] uses multiple loops: for each probe cell, search through the entire space for potential gradient cells. It will have a serious efficiency problem if we generalize the notion of "comparable" cells as we discussed above, because the search space per probe cell will be large, and this search will be repeated once per probe cell.

There are also a few other studies on efficient exploration of interesting cells in data cubes or interesting rules in multi-dimensional space.

[12] considers discovery-driven exploration of OLAP data cubes. It computes anticipated value for a cell using the neighborhood values, and a cell is considered an exception if its value is significantly different from its anticipated value. This is rather different from the "interestingness" defined here based on a user-specified gradient ratio in relevance to a cell's ancestors, descendants, and siblings. For computation, the former ([12]) is on interactive exploration of *computed* cube cells; whereas the latter (our study) is on computing (nonmaterialized) cells (more exactly, pairs of cells) satisfying certain constraints. It is an interesting issue to see whether our computation can be used as a filtering process and feed the results into the statistical analysis of neighborhood cells to reduce the overall processing cost of discovery-driven exploration of OLAP data cubes.

[2] considers how statistics (a measure) of one group of tuples differs from the same measure of a supergroup. It shows that, by adopting such difference or ratio measure, the number of association rules can be reduced substantially and only the interesting rules are preserved. This shares a similar motivation as our study here. However, our study provides a general mechanism to specify constraints and any kind of measures and/or gradients in relevance to ancestors, descendants and siblings. Therefore, it provides a more general model, as well as an efficient constraint-pushing and computation method. We believe that our method can serve as an efficient preprocessing step for subsequent statistical studies on mined interesting gradients or rules.

Our study is also closely related to (1) data cube and iceberg cube computation methods proposed in previous studies, such as [1, 4, 6, 3, 8], as well as (2) constraint-based data mining methods, such as [13, 10, 5, 7, 11]. This study can be considered as an extension and integration of both mechanisms towards efficient, multi-dimensional, constrained gradient analysis.

7 Conclusions

In this paper, we have studied issues and methods on efficient mining of multi-dimensional, constrained gradients in data cubes. Constrained gradients are substantial changes in a set of measures (aggregates) of interest associated with the changes in the underlying characteristics of cube cells, where changes in characteristics are expressed in terms of the dimensions and are limited to specialization, generalization, and 1-d mutation. To ensure only interesting changes of relevant cells are studied, we show that it is necessary to introduce three kinds of constraints: *significant constraints*, *probe constraints*, and *gradient constraints*.

An efficient algorithm, $LiveSet$-driven, has been developed which explores set-oriented processing and the maximal pushing of the constraints as deeply as possible in the early stage of the mining process to prune the search space. Moreover, we also adopt a compressed hyper-tree structure to represent the base table of a data cube, and to achieve "maximal" sharing of computation among different cells. Our performance study shows that this method is efficient and scalable. It outperforms another method which relies on the iceberg cube computation of all-significant-pairs.

There are also many interesting issues which call for further studies, including the efficient mining of association rules with complex measures, and the analysis of obtained gradient-probe pairs to extract truly interesting rules.

References

[1] S. Agarwal, R. Agrawal, P. M. Deshpande, A. Gupta, J. F. Naughton, R. Ramakrishnan, and S. Sarawagi. On the computation of multidimensional aggregates. *VLDB'96*.

[2] Y. Aumann and Y. Lindell. A statistical theory for quantitative association rules. *KDD'99*.

[3] K. Beyer and R. Ramakrishnan. Bottom-up computation of sparse and iceberg cubes. *SIGMOD'99*.

[4] S. Chaudhuri and U. Dayal. An overview of data warehousing and OLAP technology. *ACM SIGMOD Record*, 26:65–74, 1997.

[5] G. Dong and J. Li. Efficient mining of emerging patterns: Discovering trends and differences. *KDD'99*.

[6] M. Fang, N. Shivakumar, H. Garcia-Molina, R. Motwani, and J. D. Ullman. Computing iceberg queries efficiently. *VLDB'98*.

[7] G. Grahne, L. Lakshmanan, and X. Wang. Efficient mining of constrained correlated sets. *ICDE'99*.

[8] J. Han, J. Pei, G. Dong, and K. Wang. Efficient computation of iceberg cubes with complex measures. *ACM-SIGMOD'01*.

[9] T. Imielinski, L. Khachiyan, and A. Abdulghani. Cubegrades: Generalizing association rules. *Tech. Rep., Dept. Computer Science*, Rutgers Univ., Aug. 2000.

[10] R. Ng, L. V. S. Lakshmanan, J. Han, and A. Pang. Exploratory mining and pruning optimizations of constrained associations rules. *ACM-SIGMOD'98*.

[11] J. Pei, J. Han, and L. V. S. Lakshmanan. Mining frequent itemsets with convertible constraints. *ICDE'01*.

[12] S. Sarawagi, R. Agrawal, and N. Megiddo. Discovery-driven exploration of OLAP data cubes. *EDBT'98*.

[13] R. Srikant, Q. Vu, and R. Agrawal. Mining association rules with item constraints. *KDD'97*.