

Cubegrades: Generalizing Association Rules

Tomasz Imieliński (imielins@cs.rutgers.edu), Leonid Khachiyan *
(leonid@cs.rutgers.edu) and Amin Abdulghani
(aminabdu@cs.rutgers.edu)

Dept. of Computer Science, Rutgers University

Abstract.

Cubegrades are generalization of association rules which represent how a set of measures (aggregates) is affected by modifying a cube through specialization (rolldown), generalization (rollup) and mutation (which is a change in one of the cube's dimensions). Cubegrades are significantly more expressive than association rules in capturing trends and patterns in data because they can use other standard aggregate measures, in addition to COUNT. Cubegrades are atoms which can support sophisticated “what if” analysis tasks dealing with behavior of arbitrary aggregates over different database segments. As such, cubegrades can be useful in marketing, sales analysis, and other typical data mining applications in business.

In this paper we introduce the concept of cubegrades. We define them and give examples of their usage. We then describe in detail an important task for computing cubegrades: generation of significant cubes which is analogous to generating frequent sets. We also demonstrate how to evaluate simple cubegrade queries and conclude with a number of open questions and possible extensions of the work.

Keywords: database mining, cubegrades, association rules, cube generation, OLAP

1. Introduction

In the last few years a lot of interest has been expressed in database mining using association rules. In this paper we provide a different view of the association rules, which allows us to propose a significant generalization which we call *cubegrades*.

An example of a typical association rule states that, say, 23% of supermarket transactions¹ which buy bread and butter buy also cereal (that percentage is called confidence) and that 0.6% of all transactions buy bread and butter (this is called support) . This statement is typically represented as a probabilistic rule. But association rules can also be viewed as statements about how the cube representing the body of the rule is affected by specializing it by adding an extra constraint expressed by the rule's consequent. Indeed, the confidence of an association rule can be viewed as the ratio of the support drop, when the cube corresponding to the body of a rule (in our case the cube of transactions

* Supported in part by NSF Grants #IIS-0118635 and #CCR-9618796

¹ so called market basket data



buying bread and butter) is augmented with its consequent (in this case cereal). This interpretation gives association rules a “dynamic flavor” reflected in a hypothetical change of support affected by specializing the body-cube to a cube whose description is a union of body and consequent descriptors. For example our earlier association rule can be interpreted as saying that the count of transactions buying bread and butter drops to 23% of the original when restricted (rolled down) to the transactions buying bread, butter and cereal. In other words, an association rule states how the count of transactions supporting buyers of bread and butter is affected by buying cereal as well.

With such interpretation in mind, we can take a much more general view of association rules, when support (count) can be replaced by an arbitrary measure or aggregate and the specialization operation can be substituted with a different ‘delta’ operation. Conceptually, this is very similar to the notion of gradients used in algebra. In fact the term *cubegrade*, we choose is short for *cube gradient*. By definition the gradient of a function between the domain points X_1 and X_2 measures the ratio of the *delta change* in the function value over the *delta change* between the points. For a given point X and function $f()$, it can be interpreted as a statement of how a change in the value of $X(\delta X)$, affects a change of value in the function ($\delta f(X)$). They are “what if” formulae about how selected aggregates are affected by various cube modifications. For example, one could ask how the amount of cereal bought among buyers of bread is affected when we specialize the population of buyers of bread to buyers of bread and milk.

Similar questions and statements can be formed about such measures as MAX, MIN, SUM, AVG(average) of any numerical attribute aggregated over the population of a cube. While we give the formal definition of a cubegrade later, we would like to argue that if people accept association rules as the outcome of the discovery process, they should accept cubegrades as well, as significant and rich extensions of association rules. Cubegrades are statements which can be interpreted as “what if” formulae about how selected aggregates are affected by various cube modifications. Notice, that *cube specialization (roll-down)* is only one of several possible operations; other operations include roll up and mutation. Cubegrades with *rollup* operation show how different measures are affected by cube generalization; cubegrades with *mutation* hypothetically change one of the attribute(dimension) values in the cube (for example change the age from young to old) and determine how different measures are affected by such an operation. Cubegrades express how different subpopulations of the database are affected by different modifications of their definitions. The effect is measured by

the way selected aggregates change in response to operations such as specialization (roll down), generalization (roll up) and mutation.

As an illustration of the motivation behind cubegrades, consider the following extension of the market basket example. Consider a supermarket chain which maintains for each of its customers, the amount spent monthly on individual market basket items like milk, cereal, butter, etc. Also, assume the chain maintains demographic data on the customer including the area the customer lives in (suburban, urban or rural), the age group, and the income range for the customer. The basked data items, and the demographic data define the individual attributes in the customer database and grouping the database on one or more of the attributes (eg age, milk and cereal) would define the customer cubes. So, for example, we could have a cube of young customers buying milk and butter (the dimensions of the cube are age, sales of milk, and sales of butter) and the measure of interest is the average amount spent on cereal. Here, we use a cube to denote a hypercube in the multidimensional space defined by its member attributes. Using cubegrades, the supermarket would be able to evaluate how changes in the attributes of the cube affect some measure of interest. For example, the following kinds of questions can be expressed:.

Q1 How is the average amount of milk bought affected by different age categories among buyers of cereals?

Example answer: It increases by 20% for customers younger than 40 and drops by 5% among customers older than 40

Q2 What factors cause the average amount of milk bought to increase by more than 25% among suburban buyers?

Example answer: Buying cereal as well increases sales by 28%.

Q3 In cubes (population segments) where more than \$25 worth of milk is bought, how does cereal sales affect milk sales?

Example answer: It ncreases sales by 10% for urban buyers and by 28% for suburban buyers.

Q4 How do buyers in rural cubes compare with buyers in suburban cubes in terms of the the average amount spent on bread, milk, and cereal?

Example answer: They spend 10% more on bread, 5% less on milk etc.

From the above, it can be seen that cubegrades deal with computing different aggregate summaries in cubes and evaluating changes in those

aggregates due to changes in the structure of the cubes. In this thesis we cover cubegrades in details, defining a framework for cubegrades and presenting methods for mining using cubegrades. This includes defining the notion of cubegrades, introducing a language for expressing queries on cubegrades and defining a scheme for evaluating the expressed queries. An important contribution is the GBP (Grid based pruning) method which generalizes the concept of pruning based on `COUNT` used in association rules, to pruning based on `COUNT`, `MIN`, `MAX`, `SUM` and `AVG`. The new algorithm has low additional overhead and can replace `COUNT` based pruning in almost all cases.

Paper Organization: The organization of the rest of the paper is as follows. Section 2 discusses the related work in the area. In section 3, the concept for cubegrades as a generalization of rules is presented. Next, in section 4 we look at computational aspects of data mining using cubegrades. Since cubegrades are generalization of rules, existing association rule generation algorithms come to mind. However, here we aren't dealing solely with support. In this direction, we present a novel method for pruning cube queries involving arbitrary measures. In section 5 and 6 we evaluate the new pruning method and compare it both experimentally and analytically with existing pure support pruning. We conclude by looking at possible future extensions.

2. Related Work

The problem of mining association rules have attracted a lot of attention since its introduction (AIS93) . These works have covered a lot of ground including: (i) efficient algorithms for evaluating and generating rules(AMS⁺96) (ii) mining of quantitative and multi-level rules (iii) mining of rules with constraints and rule query languages and (iv) generalization of rules to correlations, causal structures and ratio rules.

However, what has remained the same in most of the works is that rules are restricted to the count (support) measure and only the specialization relationship between the LHS and RHS is expressed. There have been recent works to incorporate measures other than count, (NLHP98) (LNHP99). The objective in both of the papers is to extend the framework of rule querying in market basket data with new aggregates like `SUM`, `MIN`, `MAX` and `AVG` on the basket items. In (NLHP98) , constraints of the form $A(X) \oplus c$ are considered and in the second paper more complicated constraints of the form $A(X) \oplus B(X)$ are presented. Here the terms $A(X)$, $B(X)$ denote aggregate functions A , B on an itemset X , $\oplus \in \{<, \leq, >, \geq, =, ! =\}$ and c denotes a constant threshold. In both papers association rule queries are characterized and evaluated based

on the properties (*anti-monotonicity* for pruning and *succinctness*) of each of the individual constraints. Our work differs that we present a new general framework which encompasses association rules and other patterns with richer set of expressions and aggregates. The algorithms presented for evaluation are also novel in the level of pruning they provide.

In contrast, OLAP - an area which we believe shares the goal with association rules for “finding patterns in the data” - a variety of aggregate measures and operations such as rollups, drilldowns etc have been available for applications over cubes. Two of the main challenges here have been in putting a ‘discovery’ process on top of the cubes to quickly pinpoint cubes of interest and in generating or materializing a set of cubes.

Recently, there has been a lot of commercial and research interest to enhance OLAP products with discovery primitives. For example, Cognos Scenario uses decision trees to determine factors affecting a certain outcome (Cor98), and Pilot Software DSS uses clustering to define new hierarchies of attributes (Sof). In research, (HF95) have defined a methodology where association rules are used at multiple levels to find progressively correlations within dimensions. In (SAM98), the authors have proposed methods for finding exceptions and determining interesting cubes. More recently, (Sar99) another new operation was proposed to explain differences between two cubes. The work presented here continues to that end by dynamically discovering interesting cube changes and determining what factors affect these changes.

As for the generation of cubes, typically they are produced from relational-data in a top-down fashion where more detailed aggregates (ie more dimensions) are used to compute less detailed-aggregates. These algorithms take advantage of the commonality between a *parent* group by and a *child* group by by using a variety set of techniques of partitioning, sorting, or partial sorting (AAD⁺96), (RS97). However, in cases of large dimensionality of the data, most of these current algorithms could become very expensive. In addition, it may not even be known what cubes to precompute. In such scenarios, it may be better to evaluate the cubes dynamically based on a query. The query may have conditions on both dimensions and measures. So other approaches that statically materialize certain group-bys and dimensions (HRU96),(BPT97), (SDJ98) may not be sufficient. A recent work in that direction has been in (BR99) where a bottom-up approach for generating a dynamic subset selection of cubes have been proposed. The approach makes use of the support threshold in pruning away the un-necessary cubes. As part of the work we present an approach that provides a more general strategy involving measures other than COUNT.

3. Defining cubegrades

We start by looking at some basic notions. Define a *descriptor* to be an attribute value pair of the form $attribute=value$ if the attribute is a discrete attribute or $attribute \in interval$ if the attribute is a continuous attribute. A conjunction of k descriptors is denoted a k -conjunction.

For a given k -conjunction and a database of objects:

- the set of objects that satisfy the k -conjunction define the *cube* for that conjunction. Logically, a cube depicts a multidimensional view of the data.
- The attributes that constitute the k -conjunction define the *dimensions* of the cube.
- Attributes that aggregates over objects which satisfy the cube definition define the *measure* attributes of the cube. The measures are numerical.

A cube C' is defined to be a *specialization* or a *subcube* of another cube C if the set of records in C' is a subset of the set of records in C .

As an example, the cube of rural households earning between 45K to 60K is denoted by the conjunction (**areaType='rural' income=[45K-60K]**). The dimensions for this are **areaType** and **incomeRange**. If we look at the average amount spent on milk in the cube, then a measure for this cube would be **AVG(milk)**. A possible subcube of the cube would be (**areaType='rural' age=[24,39] income=[45K-60K]**). Note if the k -conjunction T is a superset of a m -conjunction T' ($k \geq m$) then the cube defined with T is a subcube for the cube described by T' . A numeric attribute such as income may be used both as a measure and as a dimension (in which case its discretized). For example, in the cube (**areaType='rural' income=[45K-60K]**) we may have the measure **AVG(income)**.

Association or propositional rules can be defined in terms of cubes. They can be defined as a quadruple (*Body, Consequent, Support, Confidence*) where *Body* and *Consequent* are cubes over disjoint sets of attributes, *support* is the number of records satisfying the body of the rule and *confidence* is the ratio of the number of records which satisfy the body and the consequent to the number of records which satisfy just the body. Another way of looking at an association rule is to consider it as a statement about a *relative change* of a measure (COUNT) when “restricting” or “drilling down” the Body cube to the Body + Consequent cube. The confidence of the rule measures how the consequent cube affects the support when drilling down the body

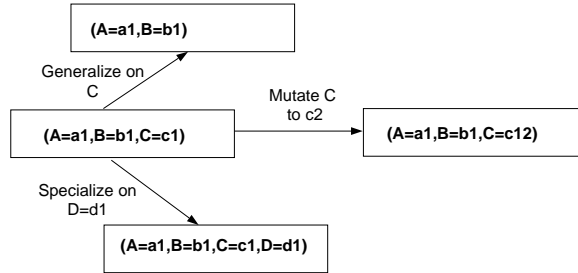


Figure 1. Scenarios of the three cubegrade operations

cube. In loose terms, this is similar to derivatives in calculus, where we measure how a change in the value of domain (in this case cube) affects a function (in this case count). There are two ways these association rules can be generalized:

- By allowing relative changes in other measures, instead of just confidence, to be returned as part of the rule. Thus, for example, the relative changes in measures such as **AVG**, **MAX** can be returned.
- By allowing cube modifications to be able to occur in different “directions” instead of just restrictions (or drill-downs). The additional directions we consider here include generalizations (or roll up) which modifies cubes towards the more general cubes with fewer descriptors, and mutations (or roll side), which modifies the descriptors of a subset of the attributes in the original cube definition with the others remaining the same. Consider again the cube (**areaType=‘rural’ income=[45K-60K]**). A generalization would be (**areaType=‘rural’**) while a mutation would be (**areaType=‘urban’ income=[45K-60K]**).

These generalized cube modifications are defined as *cubegrades*. A cubegrade expresses how a change in the structure of a given cube affects a set of predefined measures. The original cube which is being modified is referred to as the *source cube* and the modified cube as a *target cube*.

More formally, a *cubegrade* is a 5-tuple, $(Source-Cube, Target-Cube, Measures, Value, Delta-Value)$, where *Source-Cube* and *Target-Cube* are cubes, and *Measures* is the set of measures which are evaluated both in the *Source-Cube* as well as in the *Target-Cube*. *Value* is a function, $Value: Measures \rightarrow \mathcal{R}$, which evaluates measure $m \in Measures$

in the *Source-Cube*, and *Delta-Value* is also a function, *Delta-Value: Measures* $\rightarrow \mathcal{R}$, which computes the ratio of the value of $m \in \text{Measures}$ in the *Target-cube* versus the *Source-Cube*.

More formally, a *cubegrade* is a 5-tuple, (*Source-Cube*, *Target-Cube*, *Measures*, *Values*, *Delta Values*), where *Source-Cube* and *Target-Cube* are cubes, and *Measures* is the set of measures which are evaluated both in the *Source-Cube* as well as in the *Target-Cube*. *Values* is a list containing the numerical valuation of each measure $M \in \text{Measures}$ in the *Source-Cube* and *Delta Values* is a list containing the relative change in the valuation of $M \in \text{Measures}$ between the *Source-Cube* and *Target-cube*. The element $\text{Delta}M$ of *Delta Values* defines the relative change of value for the measure M . A cubegrade can visually be represented as a rule form:

SourceCube \rightarrow TargetCube [Measures, Values, Delta Values]

We distinguish three types of the cubegrades:

- Specializations: A cubegrade is a *specialization* if the set of descriptors of the target cube is a superset of the set of descriptors of the source cube.
- Generalizations: A cubegrade is a *generalization* if the set of descriptors of the target cube is a subset of the set of descriptors of the source cube.
- Mutations: A cubegrade is a *mutation* if the target cube and source cube have the same set of attributes but differ on the descriptor values (they are “union compatible” so to speak, as the term has been used in relation algebra).

Notice that, in principle, one can define a cubegrade over any pair of cubes. However, we will restrict ourselves only to the three types described above.

During the course of our discussion, we will be heavily using the term *factor* to denote the change of descriptors from source cube to target cube. Consider a cubegrade G described by source cube S and target cube T . Define a *descriptor change* as a change of descriptor from source cube to target cube. A *descriptor change* is a \cdot .

- *specialization*: if there is a descriptor $(A_i = a_i)$ present in target cube, while the source cube has no descriptor on attribute A_i . It is denoted as $+(A_i = a_i)$.

- *generalization*: if there is a descriptor $(A_i = a_i)$ present in source cube, while the target has no descriptor on attribute A_i . It is denoted as $-(A_i = a_i)$.
- *mutation*: if there is a descriptor $(A_i = a_i)$ present in source cube which is *mutated* in the target cube as descriptor $A_i = a'_i$.

As mentioned, a *factor* is defined to be set of descriptor changes from source cube S to target cube T . It is denoted as $\Delta(S, T)$. A *target* obtained by affecting *source* S with *factor* F is denoted as $S + F$.

It can be seen that cubes and association rules are both special cases of cubegrades. A cube is simply a cubegrade with just the source cube and the set of Values only. It can be viewed as a 3-tuple (*Source-Cube*, *Measures*, *Values*) where *Source-Cube* is a cube definition, *Measures* is a set of measures and *Values* are the values for *Measures* M in the cube. On the other hand, an association rule is a specialization cubegrade with target cube being a union of the body and the consequent, and COUNT as the measure, with support as the value of COUNT, and confidence as the value of DELTA_COUNT.

EXAMPLE 3.1.

Following are some examples of cubegrades. Notice that an attribute may interchangeably be used as a dimension or in a measure aggregate but on the same cubegrade.

- *The average age of buyers who purchase 20-30 dollars worth of milk monthly drops by 10% among buyers who also buy some cereal.*
 $(\text{salesMilk}=[\$20,\$30]) \rightarrow (\text{salesMilk}=[\$20,\$30], \text{salesCereal}=[\$1,\$5])$

$[\text{AVG}(\text{Age}), \text{AVG}(\text{Age}) = 23, \text{DeltaAVG}(\text{Age}) = 90\%]$

This is an example of a specialization cubegrade. We move from the cube of buyers who purchase 20-30 dollars worth of milk monthly to buyers who also buy some cereal. The $\text{AVG}(\text{AGE})$ in the target cube is 90% of the $\text{AVG}(\text{AGE})$ in the source cube. The factor for this cubegrade is $+\{\text{salesCereal} = [\$1, \$5]\}$.

- *The average amount spent on milk by urban buyers drops by 20% for buyers who are below 30.*

$(\text{areaType}=\text{'urban'}) \rightarrow (\text{areaType}=\text{'urban'}, \text{Age}=[18,30])$

$[\text{AVG}(\text{salesMilk}), \text{AVG}(\text{salesMilk}) = \$12.40, \text{DeltaAVG}(\text{salesMilk}) = 80\%]$

This is again an example of a specialization cubegrade. We look into the cube of urban buyers and see how the average amount

spent on milk is affected by specializing on a certain age group of urban buyers. Notice we are using AGE as a dimension attribute. The factor for this cubegrade is $+\{Age = [18, 30]\}$.

- *The average amount spent on milk drops by 30% when moving from suburban buyers to urban buyers.*

(areaType='suburban') → (areaType='urban')

[AVG(salesMilk), AVG(salesMilk) = \$12.40, DeltaAVG(salesMilk)=70%]

This is a mutation cubegrade where we observe how the average amount spent on milk changes moving from the cube of suburban buyers to the cube of urban buyers. The factor for this cubegrade is $\{areaType = 'suburban' - > areaType = 'urban'\}$.*

- *Urban buyers with incomes between \$50,000 to \$70,000 spend 10% more on milk than general urban buyers.*

(areaType='urban', income=[50K,70K]) → (areaType='urban')

[AVG(salesMilk), AVG(salesMilk) = \$13.78, DeltaAVG(salesMilk)=90%]

This is a generalization cubegrade where the cube of middle income urban buyers is rolled up or generalized to the cube of urban buyers. The change causes the average amount spent on milk in the target cube to be 90% of that in the source cube. The factor for this cubegrade is $-\{income = [50K, 70K]\}$

4. Query Processing

We think of cubegrade generation process as a two-way process between the user and system. The direction of the discovery is guided by the user through a query language similar to as in (IV99). Standard SQL wouldn't be sufficient in expressing cube and cubegrade queries as it is more tuned for queries dealing with relations rather than cubes. A CUBE-BY operator has been proposed for SQL(GBLP96), but the results of the query are still within a relational context. Queries become more un-natural if conditions are introduced on the cube descriptions and measures. Instead we propose *CubegradeQL* for the purpose. It can be used to query both cubes and cubegrades.

In a typical scenario, it is not expected that users would be asking for cubegrades per se. Rather, it may be more likely that they pose a query on *how* a given delta change affects a set of cubes, *which* cubes

are affected by a given delta change, or *what* delta changes affect a set of cubes in a prespecified manner . We distinguish the following three types of queries:

The how query: This query deals with how a delta change in a cube affects a certain measure? Does it go up or down? By how much? In the context of gradients in mathematics, it would be analogous to the question of finding $\Delta f(x)$, given $x_{initial}$ and Δx .

The which query: This type of query deals in finding cubes, which have measures affected in a prespecified way by the delta changes. This is analogous to the question of finding $x_{initial}$, given the gradient, i.e. both Δx and $\Delta f(x)$.

The what query: This type of query deals in finding what delta-changes on cubes affect a given measure in a prespecified manner. This is analogous to the question of finding $x_{initial}$ and Δx , given $\Delta f(x)$.

Example of queries that can be asked through the language are as follows:.

Q5 Find in which cubes with at least 1000 customers the average spending on cereal is less than \$20?

Example answer: cube S of buyers who make between 60-80K per year and who spend more than \$100 on soda monthly.

Explanation: This is an example of a cube query with no target cubes.

Q6 How is the average cereal sales affected by age, state, and income range in cubes with at least 1000 customers and where the average sales of cereal is between 25 to 40 dollars?

Example answer: It goes up for middle-age and suburban buyers for people earning between 50K and 85K.

Explanation: This belongs to *the how query* group. It asks how different values of **age**, **state**, and **income** affect the sale of cereals in certain cubes.

Q7 In which suburban cubes of more than 1000 customers the average amount of cereal bought was 50% more than their urban counterparts?

Example answer: Buyers of ages $\langle 20-30 \rangle$.

Explanation: This is an example of *the which query*. It asks for the cubes which are affected by change of area type. It is also an example of cubegrade mutation.

Q8 For cubes with more than 1000 customers and with average cereal sales less than \$20, what specializations would increase the average amount spent on cereal by 50%? In other words, which features substantially increase cereal consumption in these cubes.

Example answer: Buyers who also buy more than \$20 worth of cookies per month.

Explanation: This is an example of a *what query*. It asks for what specializations would lead average cereal sales to increase.

Q9 For cubes with more than 1000 customers and with average cereal sales less than \$20, in which specializations do customers spend twice as much on oats than on cereal?

Example answer: cube *S* of buyers who are between 40 and 50 years old, who make between 60-80K per year and who spend more than \$100 on food weekly.

Explanation: This is an example of a query with constraint of type *var-var type*. It looks for specializations which meet the condition that the amount spent on oat is twice as much as the amount spent on cereal.

A detailed description of the language will be given in a separate paper (Aion). However, here, we will describe the relevant components of **WHERE** clause of the language to better understand the evaluation process. The clause is basically a boolean condition consisting of one or more constraints connected through **AND/OR** connectives. Each constraint could be one of the following types:

- i) *Source Conjunct Condition:* This constraint specifies the set of valid conjuncts for the source cube. It is described using a list of conjunct patterns. If a cube conjunct satisfies any one of the conjunct patterns then it satisfies the condition. The conjunct description can contain a special descriptor pattern **A=*** which allows any descriptor on attribute **A** to appear in the conjunct (including the one with value **ALL**).
- ii) *Source Value Condition:* This specifies a constraint on a measure of the source cube. Examples would include constraints on **COUNT**, **AVG(Age)**, **MAX(income)** etc. The language allows any SQL-like relational expression to be compared with the measure.
- iii) *Delta Operation Condition:* This specifies the delta operation for generating the cubegrades.

- iv) *Target Conjunct Condition*: This condition is similar to the source conjunct condition discussed earlier.
- v) *Delta Value Condition*: This specifies a constraint on a delta-value of the cubegrade. The values are expressed as percentages. An example could be $\text{DELTA_COUNT}() \geq 0.8$ or $\text{DELTA_AVG}() \leq 0.6$.

The set of constraints mentioned above are very similar to the ones encountered for association rules. A natural question to ask is if cubegrades are generalization of association rules, can we use techniques from association rule generation to generate cubegrades. Fortunately the answer seems, yes, at least for a useful class of cubegrade queries. Just like association rules, generation of cubegrades can be divided into two phases: (i) generation of significant cubes (rather than frequent sets) (ii) computation of cubegrades from cubes (rather than computing association rules from frequent sets). In this paper we will focus on the first task and a companion paper will discuss the second task.

The first task is equivalent to the computation of cube queries where the conditions present are source conjunct conditions and source value conditions. In what follows, we assume that we are dealing with standard SQL aggregates `COUNT`, `MIN`, `MAX`, `SUM` and `AVG` for the measures and that the value constraints in the “where” clause are of the form $\text{agg}(\mathbf{A}) \oplus c$, where $\text{agg}(\mathbf{X})$ is a measure, $\oplus \in \{< . >, =, ! =\}$ and c is a constant threshold. In addition its further assumed that the dimension constraints are of the form $\mathbf{A} \oplus c$ where \mathbf{A} is a dimension, $\oplus \in \{< . >, =, ! =, in\}$ and c is a constant threshold (which could be a discretized interval). The quest for more arbitrary queries will be part of future work.

4.1. PRUNING FOR CUBE QUERIES

Query Monotonicity: In principle, cube queries are evaluated in a similar way as frequent sets. However, there is one fundamental difference: pruning which is so effectively used in frequent set generation, is no longer universally possible. A simple but critical observation, that if support of a cube is below a threshold then any specialization of such cube will have support below that threshold, no longer holds for arbitrary aggregate measures such as `MIN`, `MAX`, `AVG` etc. Consequently, there will be situations when no pruning will be possible. Our goal is to detect when such pruning is possible and use it whenever we can. We provide an algorithm to do just that and combine it with the cube generation process.

The fundamental property which allows pruning is called monotonicity of the query:

DEFINITION 4.1. Let \mathcal{D} be a database and $X \subseteq D$ be a cube. A query $Q(\cdot)$ is monotonic at X if the condition $Q(X)$ is FALSE implies $Q(X')$ is FALSE for any $X' \subset X$.

With this definition, a cube X would be pruned in the algorithm if the query Q is monotonic at X . However determining whether a query Q is monotonic in terms of this definition is an NP-hard problem for many simple class of queries. A theorem detailing this is given in the appendix.

The theorem shows that unless we are dealing with extremely simple queries stated in terms of MIN and MAX only, we do not expect that monotonicity of a query at a given cube X can be checked in time polynomial in the size of the database.

View Monotonicity: To work around this problem we define another notion of monotonicity called *view monotonicity* of a query. Suppose we have cubes defined on a set \mathcal{S} of dimension and measures. A *view* V on \mathcal{S} is an assignment of values to the elements of the set. If the assignment holds for the dimensions and measures in a given cube X , then V is a view for X on the set \mathcal{S} . So, for example, in a cube of rural buyers, with 20 people the average sales of bread is \$15 then the view on the set $\{\text{areaType}, \text{COUNT}(), \text{AVG}(\text{amtBread})\}$ for the cube is $\{\text{areaType}=\text{'rural'}, \text{COUNT}()=20, \text{AVG}(\text{amtBread})=15\}$. Extending the definition, a *view on a query* is an assignment of values for the set of dimension and measure attributes of the query expression.

DEFINITION 4.2. A query $Q(\cdot)$ is view monotonic on view V if for any cube X in any database \mathcal{D} s.t. V is the view for X , the condition Q is FALSE for X implies Q is FALSE for all $X' \subseteq X$.

An important property of view monotonicity is that the time and space required for checking it for a query depends on the number of terms in the query and not the size of the database or the number of its attributes. Since most of the queries typically have few terms, we feel it would be useful in most practical situations.

4.2. CHECKING FOR VIEW MONOTONICITY

Sketch: The approach we present for determining view monotonicity is borrowed from the Quantifier Elimination methods used in the first order theory of real addition with order, see e.g. (Bas97) (HRS93). The method is referred to as *Grid Base Pruning* or (*GBP*). We will go through a high-level description of the method and then fill in the relevant details. The grid is constructed such that each of its axes corresponds to a dimension or a measure used in the query. These axes

are divided into non-overlapping intervals based on the constants used in the query. The cartesian product of the intervals define individual cells of the grid. Every cell has a unique truth assignment w.r.t query and any possible view for the query will fall into exactly one cell. As an optimization we delete the “unsatisfiable TRUE cells” from the grid. These are the cells which have been assigned the value TRUE for the query but for which no views are possible. A simple example of an unsatisfiable cell is where the MAX of a measure is always less than the MIN.

When solving the view-monotonicity problem we have a query Q represented by the set of TRUE satisfiable cells in the grid and a cube X represented by a view V . The question is if Q is known to be FALSE for V , how can we check that Q is view-monotonic for V ? From the grid construction, if Q is FALSE for V then V falls in a FALSE cell. Q would be view-monotonic when no TRUE satisfiable cell in its grid is *reachable* from V . This generalizes to the *view reachability problem*. Conceptually, a cell C from a view V if there is a cube X with view V and subcube of X X' with view V' that falls in cell C . This can be shown to be checked efficiently.

Following we outline the steps needed to determine whether a given query, Q , is view monotonic at a given view V . Assume that the query is FALSE w.r.t the given view, for otherwise we have nothing to work with.

- Using the set of satisfiable true cells of the grid find a disjunctive (for instance, perfect) normal form for the query. We will refer to the cells of the obtained DNF as *DNF cells*.
- For each DNF cell check whether the cell is *reachable* from V . By definition, V is reachable to a given cell C if there exists a cube X' in some database \mathcal{D} such that
 - The view for $Q(X')$ maps to cell C .
 - There exists a cube $X \supset X'$ s.t. V is the view for $Q(X)$.

Grid Construction: Now for the gory details. We start with the construction of the grid. Every axis of the grid corresponds to a distinct dimension or a measure attribute in the query expression. Each is partitioned into a number of non-overlapping *intervals*. For a measure axis the intervals are defined by a sort on the thresholds used with the measure in the query expression. So, for example if in a query thresholds $c_1 < c_2 < \dots < c_k$ were used with $\text{agg}(A)$, each interval constructed would have endpoints c_i and c_j with $i = 0, 1, \dots, k$ and $i \leq j \leq i + 1$, where $c_0 = -\infty$ and $c_{k+1} = \infty$. Note that depending on

how the thresholds were used in the query, a given interval could be open, partially open or closed from both sides, or it may be a point.

Similarly, for a dimension axis, the intervals are defined based on the actual values c_i the attribute was compared with in the query. These intervals include constants corresponding to discrete attributes. In addition, two new special intervals ALL and NONE are added to the axes, to handle respectively, the situation when the attribute under consideration isn't part of a conjunct definition of the cube (i.e. it is summarized) and the situation when the value of the attribute is other than c_1, c_2, \dots, c_k . ALL can logically be viewed as a special interval encompassing all other intervals for the attribute. If the attribute was specified with only a wildcard pattern in the query then no axis for it is needed.

Selecting an interval for each of the axes and taking the Cartesian product of the selected intervals we obtain a *cell* in the grid. The input query is evaluated at a sample point (for example, the midpoint) of each cell and its truth value is assigned to the cell. The organization of the cells are such that the truth assignment of the query does not change within the cell.

Cell Satisfiability: Following the construction, we have with us a grid consisting of a set of cells, with each cell having a truth assignment. To be noted, is that any view possible from the query can be represented as a point in one of the cells and that the truth value of the view w.r.t query is equal to the truth value of the cell it maps to. In general, however, not every cell in the grid contains a view: some of the cells may be *unsatisfiable*. For example, suppose we have a query expression $\text{MIN}(A) \geq 6 \text{ OR } \text{MAX}(A) \leq 5$. One of the cells for this query is the Cartesian product of the intervals $\text{MIN}(A) = (6, \infty)$, $\text{MAX}(A) = (-\infty, 5)$ and the truth assignment in this cell is TRUE. However it is clear that there exists no set of real numbers such that the cell would be satisfied. Thus, before proceeding further with the grid we need to prune out the unsatisfiable cells.

To define *cell satisfiability* consider a cell C defined by a system of interval constraints on J dimension attributes D^1, \dots, D^J and on the measures of K distinct attributes A^1, \dots, A^K :

$$D^{(j)} \in [V^{(j)}] \quad (1)$$

$$\text{COUNT} \in I_{\text{COUNT}} \quad (2)$$

$$\text{MIN}(A^{(k)}) \in I_{\text{MIN}}^{(k)} \quad (3)$$

$$\text{MAX}(A^{(k)}) \in I_{\text{MAX}}^{(k)} \quad (4)$$

$$\text{SUM}(A^{(k)}) \in I_{\text{SUM}}^{(k)} \quad (5)$$

$$\text{AVG}(A^{(k)}) \in I_{\text{AVG}}^{(k)} \quad (6)$$

where $[V^{(j)}]$, $j = 1, \dots, J$, I_{COUNT} and $I_{\text{MIN}}^{(k)}$, $I_{\text{MAX}}^{(k)}$, $I_{\text{SUM}}^{(k)}$, $I_{\text{AVG}}^{(k)}$, $k = 1, \dots, K$, are given intervals. (Note some of the above constraints may be missing because the corresponding intervals are infinite.) We call C a *satisfiable cell* if there is a set $X = \{X_1 \dots, X_N\} \in D^{(1)} \times D^{(2)} \dots D^{(J)} \times \mathcal{R}^k$ of $N \in I_{\text{COUNT}}$ records whose values for the J dimension attributes and for the measures of the K attributes belong to the cell.

THEOREM 4.1. *The satisfiability of a cell defined by interval constraints on J dimensional and measures of K distinct attributes can be determined in $O(J + K)$ time.*

The proof of the theorem is given in the appendix. The method can be regarded as a simplification and generalization of the method suggested for one measure attribute cells in (RSSS98). The general idea is that for each of the J dimensions and K measure attributes we independently determine the interval on **COUNT** for which they are satisfiable. An intersection of the obtained intervals is taken and the cell would be satisfiable if this intersection contains an integral point. The constraints (1 - 6)) for the K measure attributes are shown to be reducable to a set of K independent linear systems on **COUNT**.

View Reachability: As mentioned previously, Q is view monotonic at V if and only if any cell C in the disjunctive normal form for the query is not reachable from the view V . A more rigorous definition of reachability is as follows:

Consider a satisfiable cell C defined by a system of interval constraints (1)-(6) on J dimension attributes D^1, \dots, D^J and measures of K distinct attributes $A^{(1)}, \dots, A^{(K)}$. Let V be a view defined by assigning some specific values to the dimension and measure attributes:

$$\begin{aligned} D^{(j)} &= [v^{(j)}] \\ \text{COUNT} &= \mathcal{C} \\ \text{MIN}(A^{(k)}) &= m^{(k)} \\ \text{MAX}(A^{(k)}) &= M^{(k)} \\ \text{SUM}(A^{(k)}) &= \Sigma^{(k)} \\ \text{AVG}(A^{(k)}) &= a^{(k)} \end{aligned}$$

Without loss of generality, we assume that $\Sigma^{(k)} = \mathcal{C}a^{(k)}$ for all $k = 1, \dots, K$, and that $V \not\subseteq C$. By definition, C is reachable from V if there is a set

$$X = \{X_1 \dots, X_N, X_{N+1}, \dots, X_{N+\Delta}\} \subset D^{(1)} \times D^{(2)} \dots D^{(J)} \times \mathcal{R}^K$$

of $\mathcal{C} = N + \Delta$ records such that

- $N \in I_{\text{COUNT}}$ and $\Delta \geq 1$
- X maps into V
- $X' = \{X_1, \dots, X_N\}$ maps into C .

THEOREM 4.2. *The reachability of V to C can be determined in $O(J + K \log K)$ time.*

The proof of the theorem is given in the appendix. It follows on a similar pattern as the satisfiability proof. The interesting part are the constraints on the measure attributes. The cell and view constraints on each of the K measure attributes are shown to be reducible to a set of at most 4 linear systems on `COUNT`. Hence, every measure attribute independently leads to a set of at most 4 intervals on `measure COUNT()`. We refer to these sets as *4-intervals*, irrespective of the actual number of intervals obtained. (In the proof, we show an example where the constraints lead to 2 disjoint intervals.) After obtaining separate 4-intervals for each measure attribute, we check if there is an interval value of `COUNT` shared by all K measure attributes. This can be done in $O(K \log K)$ time. The following overall result can now be stated.

THEOREM 4.3. *Let Q be a query in disjunctive normal form consisting of m conjuncts in J dimensions and K distinct measure attributes. Then the s -monotonicity of Q at a given view can be tested in $O(m(J + K \log K))$ time.*

Although for arbitrary Boolean queries Q the number of conjuncts m may grow exponentially in J and K , we believe that the suggested method is viable for practical considerations because for typical queries J and K would be relatively small. Also, as it turns out the theorem can be applied for any query given as a union of, not necessary disjoint, cells in DNF.

4.3. GENERATING THE CUBES

For generating the cubes, any of the algorithms mentioned in the literature like Apriori(AS94), BUC (BR99) etc.. can be employed for generating the significant cubes. The difference is that pruning for cubes would be based on *GBP* instead of support. Here, we will work with Apriori, although it should be noted that any frequent set generation algorithm can be employed. Fig. 4.3 shows an outline of the algorithm. It starts from an initial seed set and then iteratively generates candidate sets of larger conjuncts (smaller cubes). The initial seed set is the set

Input: Query Q , and database DB
Output: Set of cubes that satisfy Q
Algorithm Cube Evaluation:
 Construct grid G on input query Q ;
 Compute the initial seed set for Q ;
 Compute C_1 for the query;
 $i=1$; $Ans=\phi$;
 while (C_i not empty) {
 Conduct db scan on C_i ;
 Compute the tuples $\langle \text{cube}, \text{Measure}, \text{Value} \rangle$ for Q from C_i ;
 Add the tuples that satisfy Q to Ans ;
 Use GBP to identify the cubes to prune;
 and the set of significant cubes S_i ;
 construct C_{i+1} from S_i ;
 $i++$;
 }

Figure 2. Outline for the Cube Query Evaluation algorithm

of objects in the original database projected on the dimensions of the query. The first candidate set C_1 is computed by constructing cubes of conjunct length 1 (single descriptors). A scan is made and then tuples of the form $\langle \text{cube}, \text{Measure}, \text{Value} \rangle$ are evaluated for each candidate cube. If the tuple satisfies the query then it is added to the answer, otherwise GBP is used to check whether it can be pruned. The set of cubes that remain after pruning make up the *significant set* S_1 for the iteration. This set is used to construct the next candidate set C_2 , similar to as in Apriori. The algorithm continues by performing a scan on the new set.

EXAMPLE 4.1. *As an illustration of the approach consider the hypothetical query asking for cubes with 1000 or more buyers and with total milk sales less than \$50,000. In addition, the average milk sales per customer should be between 20 to 50 dollars and with maximum sales greater than \$75. This query can be expressed as follows:*

COUNT(*)>=1000 and AVG(salesMilk)>=20 and
 AVG(salesMilk)<50 and MAX(salesMilk)>=75 and SUM(saleMilk)<50K

The grid for this query consists of four axes: COUNT(), MAX(salesMilk), AVG(salesMilk) and SUM(salesMilk). Figure 4.1 shows two-dimensional sections of the grid, where each section is fixed on the dimensions COUNT() and MAX(salesMilk). Each cell is assigned its truth value

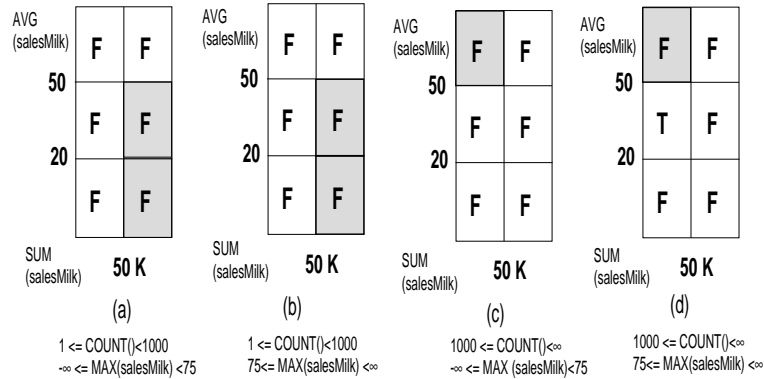


Figure 3. Example grid

w.r.t query. The shaded cells represent the unsatisfiable cells; while the rest of the cells are satisfiable.

After the grid has been constructed, the cube query evaluation algorithm can be used to get the cubes satisfying the query. The interesting part of the algorithm is to determine when to prune a cube and when not to prune.

Suppose that for a certain iteration we have a cube with the following view V ($C = 1200$, $\text{AVG}(\text{salesMilk}) = 50$, $\text{MAX}(\text{salesMilk}) = 80$, $\text{MIN}(\text{salesMilk}) = 30$, $\text{SUM}(\text{salesMilk}) = 60000$)

The query is FALSE at this view. To determine whether to prune this cube, we have to show that no cube with the above view V in any database can have a subcube which satisfies the query. The view reachability procedure checks for this condition by finding an interval on N , the count of records in the subset for which the query can be satisfied. For the example here it can be shown through the view reachability procedure (case (ii)) that the query can be satisfied when: $1000 \leq N < 1075$. Thus the cube shouldn't be pruned.

However if the view was ($C = 1200$, $\text{AVG}(\text{salesMilk}) = 57$, $\text{MAX}(\text{salesMilk}) = 80$, $\text{MIN}(\text{salesMilk}) = 30$, $\text{SUM}(\text{salesMilk}) = 68400$), then, there doesn't exist any interval for N for which V can be reached from the only true cell of the grid. Thus, this cube can be pruned.

5. Experimental Evaluation

To study evaluate GBP empirically, a variety of tests were performed. These tests had three main purposes: (i) To evaluate the performance of GBP and check its viability (ii) To sample scenarios where prunings

Table I. Parameters for SYNTH-DATA generation

Parameter Type	Parameters	value
Number of records	n	25000
Number of dimensions	d	7
Number of measures	m	1
Cluster size	c	50
Dimension domain	$D(i), i = 0 \dots d - 1$	Each dim domain has 25 values
Measure distribution	Mean μ_i and std $\sigma_i, i = 0 \dots m - 1$ generated using normal distribution	$\mu_1=60, \sigma_1=20$

offered by GBP are substantial (iii) and to see the how having multiple measure attributes in a query affects the pruning capability of GBP.

The experiments were conducted on a SUM Ultra-Enterprise with 4, 336 MHz processors and 2GB of memory with light to medium load. A commercial DBMS was used to store the source data for the experiments. In this section we start by describing the experiment setup and then describe each test separately.

5.1. EXPERIMENTAL SETUP

Experiments were performed both on synthesized data and real-life data. The synthetic data was generated using a scheme similar in nature to the one used in (Vir98) with the difference here that measure attributes also have to be dealt with. High-level input to the data included the number of dimensions d , number of measures m and the number of records n for the database. Each dimension $D(i)$ was constructed from a unique domain of size $\bar{D}(i), i = 0, 1, \dots, d - 1$, with every value in the domain equally probable. The measures were generated using a normal distribution with input parameters being the mean $\mu(i)$ and standard deviation $\sigma(i) i = 0, 1, \dots, d - 1$, of the measures. Rows were added to the data in the form of clusters of s contiguous rows, where s was the input cluster size. These clusters were generated such that j number of randomly selected attributes had a constant value throughout the cluster, while the remaining attributes were randomly assigned. The integer j was also a random integer selected between 1 to d , inclusive. For example if d was 5 and j 3, then 3 randomly selected dimensions say A_1, A_3, A_4 had a constant value throughout the cluster.

The real data set was the ‘‘Aus. Credit Dataset’’ available from the StatLog project (Pro). The dataset has a good mix of attributes –

Table II. Query Template for **SYNTH-DATA**

Aggregate	Operators	Constants
COUNT()	\geq	20, 250, 750
MIN, MAX, AVG	\leq, \geq	40, 60, 80
SUM	\leq, \geq	10000, 15000, 20000

continuous, nominal with small numbers of values, and nominal with larger numbers of values. The dataset consists of 690 records, and 14 attributes, 6 of which are numeric and 8 nominal. For reasons of confidentiality, the attributes in the dataset are coded and their names relabeled. The attributes are labeled **A1**, **A2** *ldots*. Since many of the attributes were numeric, we discretized them by arbitrarily dividing each of their ranges into 3 equal sub-ranges. The measure attributes were taken as **A2**, **A3** and **A7** with rest of the attributes being the dimensions

5.2. EVALUATING GBP PERFORMANCE

The purpose for the first experiment set, (**EXP:GBP-EVAL**) was to evaluate the pruning capabilities of GBP for a variety of cube queries and compare it relative to pruning based on pure support.

We start by describing the experiment for the synthetic data. The data set was generated with 7 dimensions, 1 measure, and 25000 records. The domain size for every dimension was 25. As for the the mean and standard deviation parameters for the selected measure, they were 60 and 25 respectively. Records were added in clusters of size 50. Table 5.1 summarizes these parameters.

An important task, here, was the selection of queries to run against the dataset. To ensure a good variety, the test queries generated were conjunctive queries with different combinations of **COUNT**, **MIN**, **MAX**, **SUM** and **AVG** measures. In general, every query had one or more constraints of type (**AGG()** $\{\leq, \geq\}$ *c*) where **AGG()** denotes an aggregate and *c* denotes a constant, with at least one **COUNT()** constraint present in the query. The template from which the queries were generated for the synthetic dataset is shown in Table (5.2). The thresholds selected for **MIN**, **MAX**, **AVG** aggregates were 40, 60, and 80. They denote **AVG(M1)-STD(M1)**, **AVG(M1)**, **qry AVG(M1)+STD(M1)** respectively. The thresholds chosen for **SUM** were obtained by multiplying the support count of 250 by the **MIN**, **MAX**, **AVG** thresholds namely, 40, 60, and

80. Queries were constructed by taking all possible combination of the constraints. This came to be 2401 queries per support constraint, for a total of 7203 queries.

To evaluate the performance of GBP we compared the execution of the queries using GBP with the execution using support pruning. The reduction in the execution times of the queries and the total number of conjuncts generated for the queries were noted. For presenting the results, we put queries into four different classes based on the number of conditions they had: $\text{COUNT}() + 1$ condition, $\text{COUNT}() + 2$ conditions, $\text{COUNT}() + 3$ conditions, $\text{COUNT}() + 4$ conditions. For example the query $\text{COUNT}() \geq 20 \text{ AND AVG}(M1) \geq 80$ has $\text{COUNT}() + 1$ condition, while $\text{COUNT}() \geq 250 \text{ AND AVG}(M1) \geq 80 \text{ AND SUM}(M1) \leq 10000$ has $\text{COUNT}() + 2$ conditions. Each class was evaluated separately.

Fig. (5.2) summarizes the executions times observed for the different classes. The summary is in the form of piechart. There are four possible slices based on the size of reduction compared with support pruning: < 1.25 times, $1.25 - 2.5$ times, $2.5 - 5$ times, and > 5 times. The figure shows that using $\text{COUNT}() + 1$ condition a very big chunk of the queries (about 84%) have minimal reduction. But as the number of conditions increase, this chunk gradually decreases in size. For $\text{COUNT}() + 4$ conditions, only 30% of the queries remain with minimal reduction. About 47% of the queries have a reduction of > 5 times.

Digging a little deeper, we looked at the number of conjuncts generated for the queries. Here, in constructing the piecharts one additional slice is added: non-satisfiable queries. These are the queries that are rejected by the system because they satisfiability constraints. Note that current support pruning methods do not check for these conditions. Fig. (5.2) shows the set of pie-charts summarizing the reductions observed for the number of conjuncts generated. Here again, we see that as the number of conditions a increase, there is a greater reduction in the number of conjuncts generated for the queries.

The two set of piecharts in Figs. (5.2,5.2) confirm what is expected of GBP. As the number of conditions increase, the reachability condition becomes more difficult to satisfy and thus queries can be pruned earlier. Also an increase in the number of conditions make it much more probable for a query to become unsatisfiable. One additional point one may note here is that the slice sizes for the execution time piecharts are slightly different then the slices for the number of conjuncts generated. In terms of the number of conjuncts generated, the reduction with increasing query sizes, seems greater. The reason could be explained by the fact that execution time also takes into account other overheads like database scans, subset checking etc.

Table III. Query Template for AUS-CREDIT

Aggregate	Operators	Constants
COUNT()	\geq	10, 40
MIN(A2), AVG(A2)	MAX(A2), \leq, \geq	20, 32, 44
SUM	\leq, \geq	800, 1280, 1760

To see the performance gain of GBP also holds on the real data set we repeated the experiment on the credit database. Queries were generated using the 10 selected dimensions. As before, only one measure attribute $A2$ was considered. The average of this attribute was 32 with standard deviation of 12 (rounded to whole numbers). Here again, the need was to generate conjunctive queries with different combinations of COUNT, MIN, MAX, SUM and AVG values. The description of the queries generated for the synthetic dataset is shown in Table 5.2. The constants used for comparing with MIN, MAX, AVG aggregates were 20, 32, and 44. They denote $AVG(A2) - STD(A2)$, $AVG(A2)$, query $AVG(A2) + STD(A2)$ respectively. The constants used for comparing with SUM were obtained by multiplying the support threshold of 40 by the above constants namely, 20, 32, and 44. With each query having at least one COUNT() constraint, every possible combination of the constraints were generated. A total of 4802 queries were generated.

The results evaluated for the dataset are shown in Figs. (5.2) and (5.2). Here again we see a reduction in execution time, and the number of conjuncts generated for a majority of the queries as the number conditions are increased. With COUNT() + 1 condition, 72% of the queries have reductions of size less than 1.25. Increasing the number of conditions to COUNT() + 4 conditions, only 19% of the queries have reductions less than 1.25.

One may at this point also raise a question that in cases where GBP offers no savings, what would be the overhead in computing the satisfiability and reachability set of constraints. The experiments that we have run show that it is a very little overhead. The maximum overhead we saw in our our executions was 5%.

5.3. QUERIES AND GBP PERFORMANCE

While running EXP:GBP-EVAL, it became interesting to find out when and for which kind of queries GBP seem to perform well for. This is

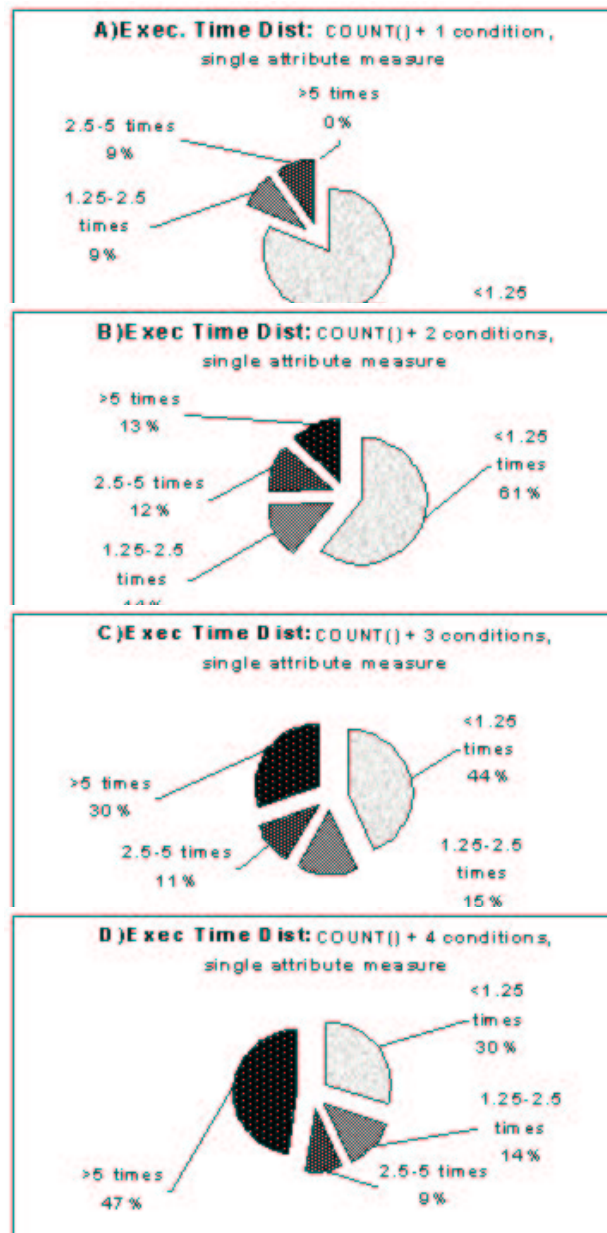


Figure 4. EXP:GBP-EVAL: SYNTH-DATA, Relative execution time using GBP compared with support pruning

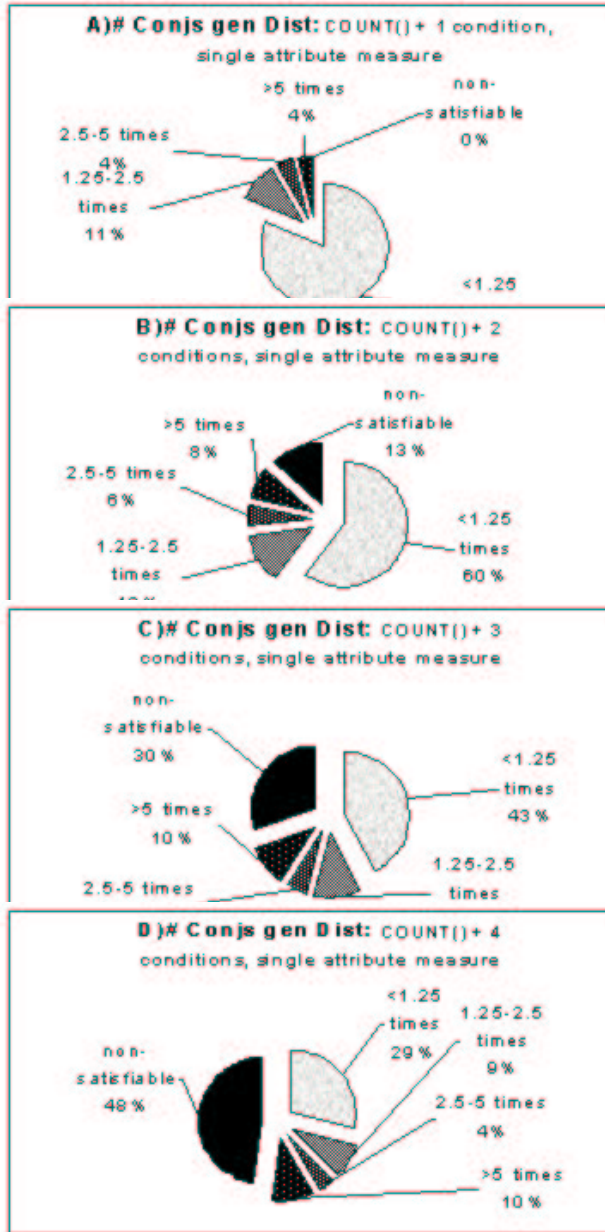


Figure 5. EXP:GBP-EVAL: SYNTH-DATA, Relative number of conjuncts generated using GBP compared with support pruning

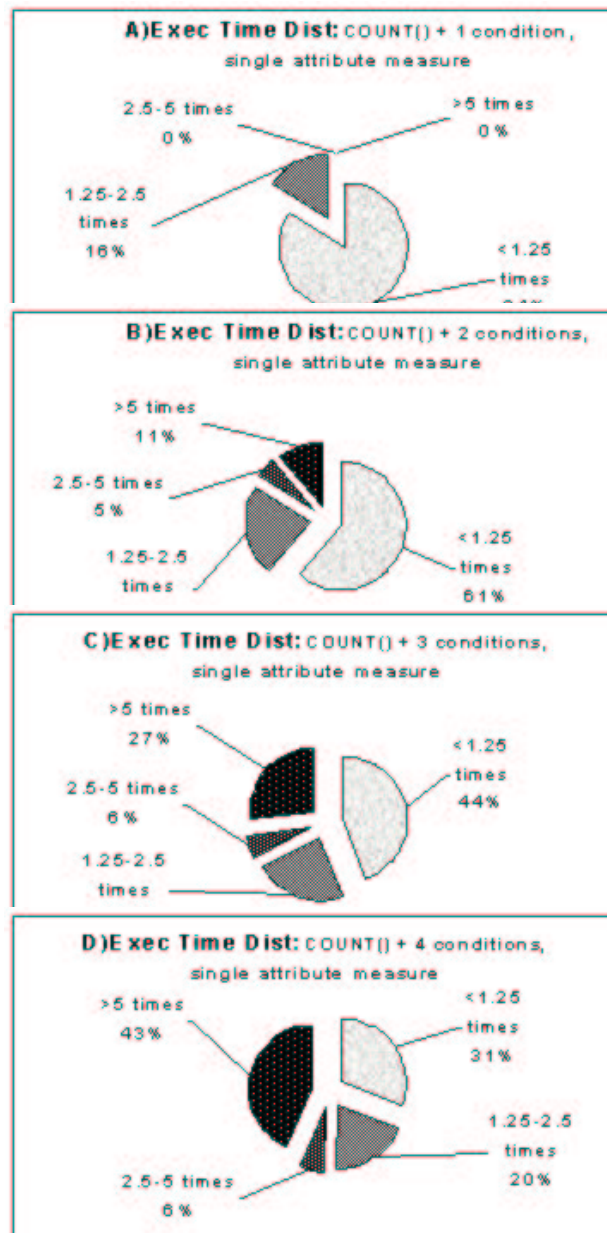


Figure 6. EXP:GBP-EVAL: AUS-CREDIT, Relative execution time using GBP compared with support pruning

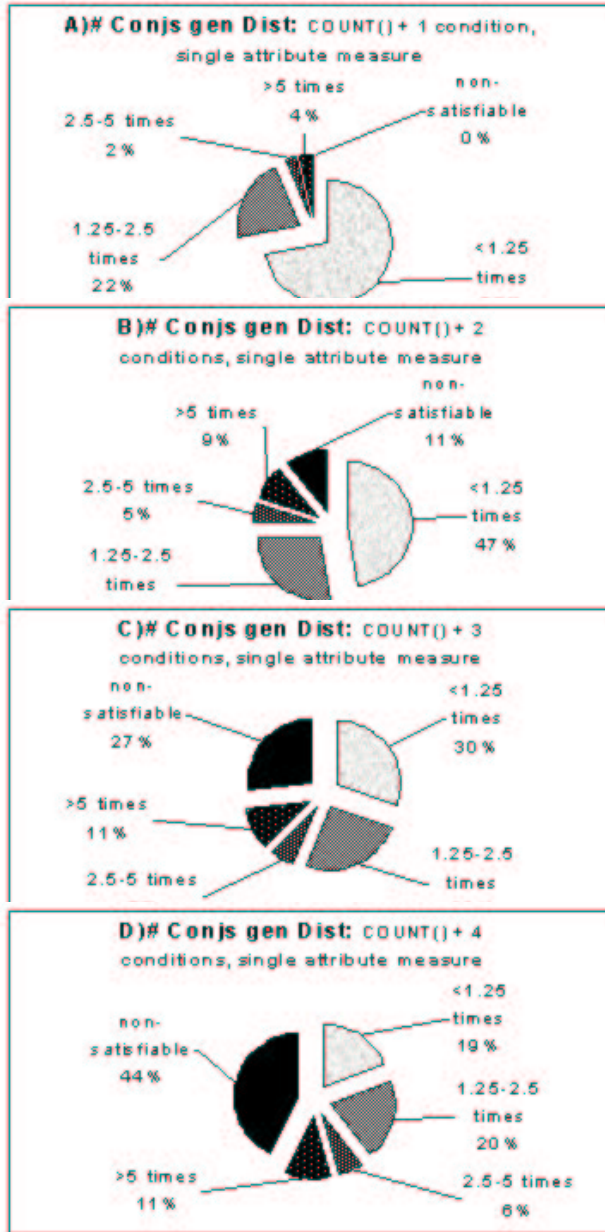


Figure 7. EXP:GBP-EVAL: AUS-CREDIT, Relative number of conjuncts generated using GBP compared with support pruning

probably a very open question not only due to the very large the space of queries but also because GBP performance is dependent on the data distribution. However, it seemed to be a useful exercise to look at how queries prune even it is for a small sample. For our experiment, we took the 7203 queries for the synthetic dataset, and the 4802 queries for the credit data set and compared their behaviors to see when they achieved more pruning and when they achieved less. It should be reiterated that this difference in query performance can only be considered if GBP is the pruning criteria. If support pruning is applied then all queries with the same threshold would have the same performance in terms of the total number of conjuncts generated by the algorithm.

Consider a query Q and consider another query $Q' = Q \wedge c$ where c is a constraint not present in Q . Clearly, the total number of conjuncts generated by GBP for Q' wouldn't exceed the number of conjuncts generated for Q . Q' is deemed as being *significant for the dataset* if for the given dataset, the number of conjuncts generated by GBP for Q' is some threshold $x\%$ less than the number of conjuncts produced for any significant predecessor (significant queries of the dataset having more general constraints than Q').

In table (IV) we list the significant queries for the synthetic data-set for the support threshold of 20. The parameter x was taken at 10%. It lists the queries, the total number of conjuncts generated using GBP, the total number of conjuncts generated using support pruning, and the number of conjuncts satisfying the query. Consider for example, the query $\text{SUM}(M1) \geq 20000 \text{ AND } \text{AVG}(M1) \leq 40 \text{ AND } \text{COUNT}() \geq 20$. This query produces 5930 conjuncts. Compare this with the whopping 187,069 conjuncts produced by the query $\text{COUNT}() \geq 20$. One may ask whether the pruning produced for the considered query due to the $\text{SUM}()$ constraint or to the $\text{AVG}()$ constraint? The answer is a combination of both. The query produces at least 55% less conjuncts than any of its other predecessor queries. The pattern obtained from the table signifies the strength of GBP: the pruning condition isn't dependent on a given constraint, rather pruning is evaluated on the query as a whole in the context of the underlying view. In the table, we also note for the queries the number of satisfied conjuncts for the dataset. This number is useful in comparing with the total of conjuncts produced by the pruning methods and indicates the extra amount of work needed by the pruning methods. For some of the queries, this number is 0. It should be stressed here, that this doesn't mean that the query is totally unsatisfiable but rather for the tested dataset the query was unsatisfiable. This underscores the basic assumption of GBP of looking at a view of a given cube in the data set, rather than the individual set of measure values.

We now look at other set of queries and datasets. For queries with $\text{COUNT} \geq 250$ in Table (V) there are a fewer significant queries. The reasons are similar to the one explained in the previous section. In every query the support threshold becomes the primary constraint on which the query is pruned. In Table (VI) queries with $\text{COUNT} \geq 750$ are shown. The number of significant queries for the dataset increase, due to the fact that queries become more restrictive and the reachability condition becomes harder to satisfy for the views; hence, many of the queries can be pruned earlier. A similar set of patterns was observed for the queries from the credit data-set. The number of significant queries generated for the support threshold for 10 was 46, and for the support threshold of 40 it was 18. Tables (VII, VIII) show a sample of those queries.

Query: $\text{COUNT} \geq 20$ + other conditions	Num. conjuncts generated by GBP (Support pruned generates 187069 conjuncts)	Num conjuncts satisfy
$\text{COUNT}() \geq 20$ only	187069	16909
$\text{AVG}(M1) \leq 60$	167379	8725
$\text{AVG}(M1) \geq 60$	165704	8184
$\text{MAX}(M1) \leq 60$	157540	0
$\text{MIN}(M1) \geq 60$	157070	0
$\text{MIN}(M1) \geq 40$ AND $\text{AVG}(M1) \leq 60$	147514	2
$\text{MIN}(M1) \geq 40$ AND $\text{MAX}(M1) \leq 60$	147375	0
$\text{MAX}(M1) \leq 80$ AND $\text{AVG}(M1) \geq 60$	145874	4
$\text{MIN}(M1) \geq 60$ AND $\text{MAX}(M1) \leq 80$	145736	0
$\text{AVG}(M1) \leq 40$	88522	0
$\text{AVG}(M1) \geq 80$	86879	0
$\text{MAX}(M1) \leq 40$	85979	0
$\text{MIN}(M1) \geq 80$	84856	0
$\text{SUM}(M1) \geq 10000$	13847	932
$\text{SUM}(M1) \geq 15000$	13646	686
$\text{SUM}(M1) \geq 20000$	13635	649
$\text{SUM}(M1) \geq 20000$ AND $\text{AVG}(M1) \leq 40$	5930	0
$\text{MAX}(M1) \leq 40$ AND $\text{SUM}(M1) \geq 20000$	5836	0

Table IV.: Significant queries for SYNTH-DATA, COUNT=20

Query: COUNT \geq 250 + other conditions	Num. conjuncts generated by GBP (Support prun. generates 13647 conjuncts)	Num conjuncts satisfy
COUNT() \geq 250 only	13647	688
SUM(M1) \geq 20000 AND AVG(M1) \leq 40	5930	0
MAX(M1) \leq 40 AND SUM(M1) \geq 20000	5836	0

Table V.: Significant Queries for SYNTH-DATA, COUNT=250

Query: COUNT \geq 750 + other conditions	Num. conjuncts generated by GBP (Support prun. generates 4576 conjuncts)	Num conjuncts satisfy
COUNT() \geq 750	4576	270
MIN(M1) \geq 80	2210	0
AVG(M1) \geq 80	2210	0
AVG(M1) \leq 40	2033	0
MAX(M1) \leq 40	2033	0
SUM(M1) \leq 20000	1516	0
SUM(M1) \leq 15000	1330	0
SUM(M1) \leq 10000	964	0

Table VI.: Significant Queries for SYNTH-DATA, COUNT=750

Query: COUNT \geq 10 + other conditions	Num conjs gen by GBP (Support prun. generates 32237 conjuncts)	Num conjs satisfy
COUNT() \geq 10	32237	31417
AVG(A2) \geq 32	27418	12269
MAX(A2) \leq 44 AND AVG(A2) \geq 32	23661	32
SUM(A2) \geq 1280	5783	5485
MAX(A2) \leq 44 AND SUM(A2) \geq 1280 AND AVG(A2) \geq 32	4970	0

Table VII.: Sample Significant Queries for AUS-CREDIT, COUNT=10

Query: COUNT ≥ 40 + other conditions	Num conjs gen by GBP (Support pruning generates 6076 conjuncts)	Num conjs satisfy
COUNT() ≥ 40	6076	5746
AVG(A2) ≤ 20	4204	0
SUM(A2) ≥ 1760	3910	3633
SUM(A2) ≥ 1760 AND AVG(A2) ≤ 20	1527	0

Table VIII.: Sample Significant Queries for AUS-CREDIT, COUNT=40

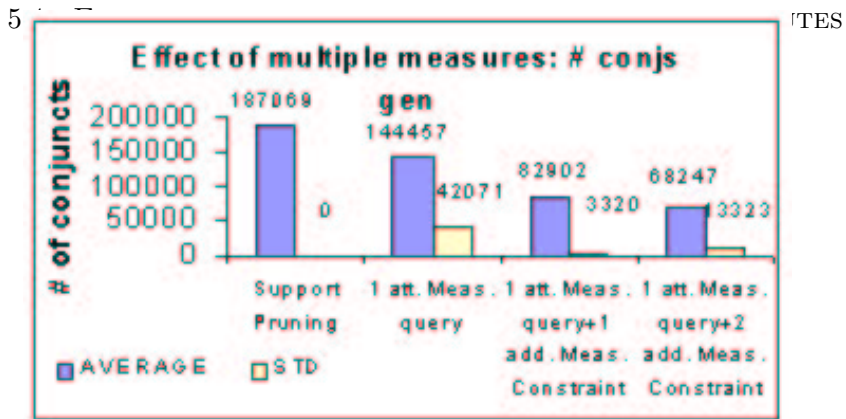


Figure 8. EXP:GBP-MULTI: CREDIT DATA

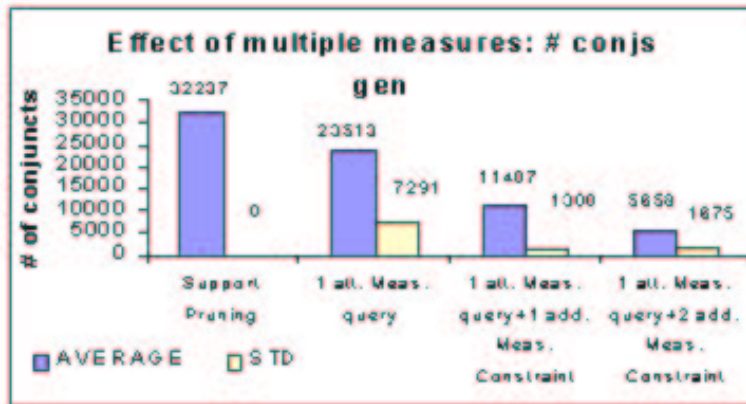


Figure 9. EXP:GBP-MULTI: AUS-CREDIT

The objective in this set of experiments was to see how multiple measure attributes in a query affect the pruning performance of GBP. One can infer that having multiple measure attributes will lead to a smaller set of satisfiable regions and hence a relatively smaller number of conjuncts would be generated. The experiments were conducted both on the synthetic dataset and the real-life credit database.

We start by looking at the setup and results for the synthetic data. Here, the dataset was similar to the one used in `EXP:GBP-EVAL` except that instead of having measures on a single attribute we could have measures on upto 3 attributes. The mean and std. devs of the new attributes were same as the mean and std. dev. of the attribute used earlier, namely 60 and 20. The test queries were produced by randomly selecting 50 queries from `EXP:GBP-EVAL` that generated 50000 or more conjuncts (We were interested the savings for large queries). A new measure attribute M_i was added into the original query(from the 50) by putting a constraint of type $\text{AVG}(M_i) \geq 80$ to it. The experiments ran by adding measure attributes M_2, M_3 , and M_4 to the queries. The results of the experiments are shown in Fig. (5.4). The average reduction is almost 2.5 times, if we 3 measure attributes are considered.

A good percentage of reduction was also observed, when the experiment was repeated for the credit dataset. Here, 50 random queries that generated 10000 or more conjuncts for the dataset in `EXP:GBP-EVAL` were selected. These queries also have constraints on only a single measure attribute. To have multiple attributes we added the constraints $\text{AVG}(A3) \geq 10$ and $\text{AVG}(A7) \geq 6$. The threshold of 10 is obtained by adding one standard deviation to the average of the attribute $A3$ in the dataset (and rounding it to a whole number). The threshold of $A7$ was similarly obtained. Fig. (5.4) shows the graph of the experiment. There is almost an average reduction of 5 times in the number of conjuncts produced using 1 measure attribute and using 3 measure attributes. Compare this with pure support pruning where no reduction would have been seen.

6. Quantifying the utility of GBP

In the previous section we experimentally looked at the performance of GBP for a few set of tests. One may argue that in certain cases the pruning achieved would be no better than pruning by support. Could we apriori identify these cases? In fact, a more general question would be how better is GBP for pruning a cube than pure support pruning? In this section we look at this question for the worst case scenario.

As an example, consider the evaluation of the following query Q ($\text{COUNT}() \geq 100$ AND $\text{SUM}(A) \geq 10\text{K}$ AND $\text{MAX}(A) \geq 150$) in a cube S where the view V is $\text{COUNT}()=1000$, $\text{MIN}(A)=20$, $\text{MAX}(A)=200$ $\text{SUM}(A)=22\text{K}$, $\text{AVG}(A)=22$. If support is to be used as the sole pruning criteria, then only those subcubes of S with support less than 100 can be pruned. On the other hand consider using the reachability test, the grid for the query has a single TRUE cell, $C1$. Applying the test, we get that $C1$ is reachable for subcubes with support bounds between 400 and 1000. This implies that any subcube with support below 400 cannot satisfy the query and can be pruned. We term this guarantee as the *derived threshold of query Q for view V* . This is in contrast to the threshold used in support pruning, (*base threshold*).

The *derived threshold* cannot be smaller than the *base threshold*. The derived threshold for a query can be computed using the algorithm shown in Figure 6.

Input: Cube query Q , a cube S with view V

Output: Derived support threshold: dev_support

Algorithm **Compute_dev_support:** {

Compute the grid G for the query and identify the true cells in the grid;

dev_support=MAXVALUE;

For each true cell C do {

Compute the COUNT intervals $[N_L, N_H]$ on C for which C is reachable from V using the reachability test;

If $N_L < \text{dev_support}$ then dev_support= N_L ;

}

return dev_support;

}

Figure 10. Computing the derived support

If the query had no support constraint, then using GBP, one would still be able to prune cubes with support less than 400. This would be in contrast to using support as the criteria where no pruning would be achieved. This shows how GBP may rely on measure constraints other than support to achieve pruning.

Of course, there may be queries and views in which GBP would have to rely on the support threshold for pruning. Consider the previous query Q and suppose instead of view V , we had view V' ($\text{COUNT}()=1000$, $\text{MIN}(A)=0$, $\text{MAX}(A)=750$, $\text{SUM}(A)=22\text{K}$, $\text{AVG}(A)=22$). Here, if the support constraint was not present then there could potentially be subcubes which may remain unpruned for $\text{COUNT}() < 100$. An example would be a subcube reachable from V' with view ($\text{COUNT}=50$, $\text{SUM}(A)=15\text{K}$, $\text{Min}(A)=0$, $\text{MAX}(A)=750$). A more extreme example would be a sub-

cube with view (COUNT=14, SUM(A)=10.5K, MIN(A)=750, MAX(A)=750). In such cases, specifying a support threshold would be useful for the purposes of query efficiency.

7. Discussion and Conclusion

In this paper we have introduced the concept of cubegrades. They are generalization of association rules which represent how a set of measures (aggregates) is affected by specializing (rolldown), generalizing (rollup) and mutating (which is a change in the cube's dimensions). Cubegrades are significantly more expressive than association rules in capturing trends and patterns in data since they use arbitrary aggregate measures, not just COUNT, as association rules do. Cubegrades are atoms which can support sophisticated "what if" analysis tasks dealing with behavior of arbitrary aggregates over different database segments. As such, cubegrades can be useful in marketing, sales analysis, and other typical data mining applications in business.

We have given an outline for evaluating cubegrades. Towards this, the GBP algorithm has been presented. The method provides efficient pruning techniques for the generation of cubegrades. The utility of GBP as a practical method was experimentally demonstrated.

Our work has also led us to identify several interesting research questions to be pursued further. Here are some of them not necessarily listed in the order of importance:

- Efficient testing when the cube query Q is satisfied for all specialization of a given cube

The procedure is very similar to testing monotonicity of Q on a view V . It involves testing whether the view V is reachable to any of the FALSE cells. If V is not reachable to any of the FALSE cells then Q is *reversely monotonic* on V .

This would be very useful not only in generating cubes but also in efficiently representing potentially very large set of cube query answers. In other words, one could say that a cube C and all *its specializations* belong to the answer; without having to explicitly list all of them.

- Define a cube query as *hard* at the cube C , if not only it cannot be pruned at C but cannot be pruned in any specializations of C .

It is natural to ask whether there are efficient tests for hardness of a query in a given cube. Are there any naturally defined syntactical classes of queries which are hard? This is related to a more general

question; since the GBP test incurs additional computation cost, it should be viewed as “investment” which will pay off only if it leads to some significant pruning. Is there a way to take advantage of the results of GBP tests for cubes which are generalizations of a given cube to reduce its cost for the specific cube?

- Building application programming interface (API) similar to the one proposed in (IVA99) to allow building more complex applications on the top of cubegrades. We realize that cubegrades, just as association rules, are only atoms from which the complex applications will have to be built. That is, just like association rules, massive volumes of cubegrades which can and will be generated cannot be presented to the analyst as the final product.
- Developing more sophisticated query language evaluation algorithms both for cubes and cubegrades. This would include allowing nesting subqueries, generalizing the constraints allowed in the query by allowing addition and multiplications between the measures.

Acknowledgment The authors thank N. Goyal for a simplification in the proof of Theorem C.1.

References

- S. Agarwal, R. Agrawal, P.M. Deshpande, A. Gupta, J. F. Naughton, R. Ramakrishnan, and S. Sarawagi. On the computation of multidimensional aggregates. In *Proceedings of the 22nd International Conference on Very Large Data Bases (VLDB'96)*, pages 506 – 521, Bombay, India, Sept 1996.
- R. Agrawal, T. Imielinski, and A. Swami. Mining associations rules between sets of items in large databases. In *Proceedings of ACM SIGMOD Conference on Management of Data (SIGMOD'93)*, pages 207 – 216, Washington D.C., May 1993.
- A. Abdulghani and T. Imielinski. Datamining with cubegrades: Querying, generation and application support. *Data Mining and Knowledge Discovery*, Submitted for Publication.
- R. Agrawal, H. Mannila, R. Srikant, H. Toivonen, and A. I. Verkamo. Fast discovery of association rules. In Usama M. Fayyad, Gregory Piatetsky-Shapiro, Padhraic Smyth, and Ramasamy Uthurusamy, editors, *Advances in Knowledge Discovery and Data Mining*, pages 307 – 328. AAAI Press, Menlo Park, CA, 1996.
- R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *vldb94*, pages 487 – 499, Santiago, Chile, 1994.
- S. Basu. An improved algorithm for quantifier elimination over real closed fields. In *Foundations of Computer Science (FOCS 1997)*, 1997.
- E. Baralis, S. Paraboschi, and E. Teniente. Materialized view selection in a multi-dimensional database. In *Proceedings of the 23rd International Conference on Very Large Data Bases (VLDB'97)*, pages 156 – 165, Athens, Greece, Aug 1997.

- K. Beyer and R. Ramakrishnan. Bottom-up computation of sparse and iceberg cubes. In *Proceedings of ACM SIGMOD Conference on Management of Data (SIGMOD'97)*, pages 359 – 370, Philadelphia, Pennsylvania, June 1999.
- Cognos Software Corporation. Cognos scenaron. <http://www.cognos.com>, 1998.
- J. Gray, A. Bosworth, A. Layman, and H. Pirahesh. Data Cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-totals. In *12th International Conference on Data Engineering (ICDE'96)*, pages 152 – 159, New Orleans, Louisiana, February 1996.
- J. Han and Y. Fu. Discovery of multiple level association rules from large databases. In *Proceedings of the 21st International Conference on Very Large Data Bases (VLDB'95)*, pages 420 – 431, Zurich, Switzerland, Sept 1995.
- J. Heintz, M.-F. Roy, and P. Solernó. On the theoretical and practical complexity of the existential theory of reals. *The Computer Journal*, 36(5), 1993.
- V. Harinarayan, A. Rajaraman, and J.D. Ullman. Implementing data cubes efficiently. In *Proceedings of ACM SIGMOD Conference on Management of Data (SIGMOD'96)*, pages 205 – 216, 1996.
- T. Imielinski and A. Virmani. M-sql: A query language for database mining. In *Data Mining and Knowledge Discovery*, 1999.
- T. Imielinski, A. Virmani, and A. Abdulghani. Dmajor-application programming interface for database mining. In *Data Mining and Knowledge Discovery*, 1999.
- L.V.S. Lakshmanan, R. Ng, J. Han, and A. Pang. Optimization of constrained frequent set queries with 2-variable constraints. In *Proceedings of ACM SIGMOD Conference on Management of Data (SIGMOD'99)*, pages 157 – 168, Philadelphia, Pennsylvania, June 1999.
- R. Ng, L.V.S. Lakshmanan, J. Han, and A. Pang. Exploratory mining and pruning optimizations of constrained association rules. In *Proceedings of ACM SIGMOD Conference on Management of Data (SIGMOD'98)*, pages 13 – 24, Seattle, Washington, June 1998.
- StatLog Project. Australian credit. <http://www.ncc.up.pt/liacc/ML/statlog/>.
- K. A. Ross and D. Srivastava. Fast computation of sparse datacubes. In *Proceedings of the 23rd International Conference on Very Large Data Bases (VLDB'97)*, pages 116 – 125, Athens, Greece, Aug 1997.
- K. A. Ross, D. Srivastava, P. J. Stuckey, and S. Sudarshan. Foundations of aggregation constraints. *Theoretical Computer Science*, 193:149 – 179, February 1998.
- S. Sarawagi, R. Agrawal, and N. Megiddo. Discovery-driven exploration of olap data cubes. In *6th International Conference on Extending Database Technology*, pages 168 – 182, Valencia, Spain, March 1998.
- S. Sarawagi. Explaining differences in multidimensional aggregates. In *Proceedings of the 25th International Conference on Very Large Data Bases (VLDB'99)*, pages 42 – 53, Edinburgh, Scotland, September 1999.
- A. Shukla, P.M. Deshpande, and J.F.Naughton. Materialized view selection for multidimensional datasets. In *Proceedings of the 24th International Conference on Very Large Data Bases (VLDB'98)*, pages 488 – 499, New York, New York, Aug 1998.
- Pilot Software. Decision support suite. <http://www.pilotsw.com>.
- A. Virmani. Discovery board: A query based approach to data mining. *PhD Thesis, Rutgers University*, April 1998.

Appendix

A. Complexity of Query monotonicity

THEOREM A.1. *For each of the following classes of queries, it is NP-hard to determine whether or not a given query Q is monotonic at a given cube X selected from a database \mathcal{D} :*

- i) Q contains a constraint on SUM*
- ii) Q contains a constraint on AVG*
- iii) Q contains constraints on COUNT and MAX on multiple attributes*
- iv) Q contains constraints on COUNT and MIN on multiple attributes.*

Proof:

i) Let \mathcal{D} be database of N records $X_1, X_2 \dots X_N$, each of which is a positive integer. Given a positive integer b , consider the query $\text{SUM}(\cdot) = b$. Assume without loss of generality that Q is FALSE at \mathcal{D} , i.e., $\text{SUM}(\mathcal{D}) = X_1 + \dots + X_N \neq b$. Then it is easy to see that Q is not monotonic at $X = \mathcal{D}$ if and only if there is a subset $Y \subset \mathcal{D}$ such that $\text{SUM}(Y) \doteq \sum\{X_i \mid i \in Y\} = b$. The latter equation is equivalent to the well-known NP-complete subset-sum problem.

ii) The proof is similar: Add an additional record with value $-b$ to \mathcal{D} and check whether or not the query $\text{AVG}(\cdot) = 0$ is monotonic at $X = \mathcal{D}$.

iii) Consider the vertex cover problem: Given a graph G with n vertices and m edges, determine whether G admits a vertex cover of size $k < n$. We show that this can be transformed to an equivalent problem of determining whether a query composed of constraints on COUNT and MAX is monotonic.

Consider the following construction of database \mathcal{D} from graph G . Each edge e_i , $i = 1, \dots, m$, in G represents an attribute for the database. The database consists of n records, each of which is the characteristic vector of a vertex: we write a 1 for attribute i in record j if edge i is incident on vertex j ; otherwise we set attribute i in record j to 0. Now it is easy to see that G has a k -vertex cover if and only if there exists a subset $Y \subset \mathcal{D}$ satisfying the query

$$[\text{COUNT}(Y) \leq k] \wedge [\wedge_{i=1}^m \text{MAX}(e_i)(Y) = 1] \quad (7)$$

Furthermore, since $k < n$, the above query is FALSE at $X = \mathcal{D}$. Hence (7) is equivalent to the NP-complete vertex cover problem.

iv) The proof is similar to that for (iii). \square

B. Satisfiability of a cell

Consider a cell C defined by a system of interval constraints on J dimension attributes D^1, \dots, D^J and on the measures of K distinct attributes A^1, \dots, A^K :

$$D^{(j)} \in [V^{(j)}] \quad (8)$$

$$\text{COUNT} \in I_{\text{COUNT}} \quad (9)$$

$$\text{MIN}(A^{(k)}) \in I_{\text{MIN}}^{(k)} \quad (10)$$

$$\text{MAX}(A^{(k)}) \in I_{\text{MAX}}^{(k)} \quad (11)$$

$$\text{SUM}(A^{(k)}) \in I_{\text{SUM}}^{(k)} \quad (12)$$

$$\text{AVG}(A^{(k)}) \in I_{\text{AVG}}^{(k)} \quad (13)$$

where $[V^{(j)}]$, $j = 1, \dots, J$, I_{COUNT} and $I_{\text{MIN}}^{(k)}$, $I_{\text{MAX}}^{(k)}$, $I_{\text{SUM}}^{(k)}$, $I_{\text{AVG}}^{(k)}$, $k = 1, \dots, K$, are given intervals. (Note some of the above constraints may be missing because the corresponding intervals are infinite.) We call C a *satisfiable cell* if there is a set $X = \{X_1, \dots, X_N\} \in D^{(1)} \times D^{(2)} \dots D^{(J)} \times \mathcal{R}^k$ of $N \in I_{\text{COUNT}}$ records whose values for the J dimension attributes and for the measures of the K attributes belong to the cell.

THEOREM B.1. *The satisfiability of a cell defined by interval constraints on J dimensional and measures of K distinct attributes can be determined in $O(J + K)$ time.*

Proof: To simplify the presentation, we prove the theorem under the assumption that C is defined by closed intervals, i.e.,

$$D^j \in [V^j] \quad (14)$$

$$\text{COUNT} \in I_{\text{COUNT}} \doteq [C_L, C_H] \quad (15)$$

$$\text{MIN}(A^{(k)}) \in I_{\text{MIN}}^{(k)} \doteq [m_L^{(k)}, m_H^{(k)}] \quad (16)$$

$$\text{MAX}(A^{(k)}) \in I_{\text{MAX}}^{(k)} \doteq [M_L^{(k)}, M_H^{(k)}] \quad (17)$$

$$\text{SUM}(A^{(k)}) \in I_{\text{SUM}}^{(k)} \doteq [S_L^{(k)}, S_H^{(k)}] \quad (18)$$

$$\text{AVG}(A^{(k)}) \in I_{\text{AVG}}^{(k)} \doteq [A_L^{(k)}, A_H^{(k)}] \quad (19)$$

where $j = 1, \dots, J$ and $k = 1, \dots, K$. We reduce the cell satisfiability problem to a system of $O(J + K)$ linear inequalities on N , the unknown count of X . The more general case of arbitrary (closed, open, or semi-open) intervals would only require some straight-forward modifications in the system we construct below.

We start with the case $K = 1, J = 0$. Suppose we have a set X of N numbers $\{X_1, \dots, X_N\}$ such that $\text{MIN}(X) \in [m_L, m_H]$ and $\text{MAX}(X) \in [M_L, M_H]$. We can assume without loss of generality that $m_L \leq M_L$, for otherwise we can replace M_L by m_L . Similarly, we assume that $m_H \leq M_H$. Further, assuming that the two intervals $[m_L, m_H]$ and $[M_L, M_H]$ are nonempty, we get $m_L \leq M_H$ as a necessary condition for the satisfiability of C .

Let $S = \text{SUM}(X)$, then

$$(N - 1)m_L + M_L \leq S \leq (N - 1)M_H + m_H. \quad (20)$$

Conversely, it is easy to see that any S satisfying (20) can be realized as the sum of N reals X_1, \dots, X_N satisfying the interval constraints (16) and (17). This is clearly true for $N = 1$. For $N \geq 2$, the interval $[(N - 1)m_L + M_L, (N - 1)M_H + m_H]$ can be expressed as

$$[m_L, m_H] + [M_L, M_H] + (N - 2)[M_L, M_H].$$

Hence any S satisfying (20) can be written as $X_1 + X_2 + X_3 + \dots + X_N$, where $X_1 \in [m_L, m_H]$, $X_2 \in [M_L, M_H]$, and $X_3, \dots, X_N \in [m_L, M_H]$. It remains to note that $m_L \leq \text{MIN}(X) \leq X_1 \leq m_H$, where the first inequality follows from the fact that $X_1, \dots, X_N \in [m_L, M_H]$. This proves (16). The proof for (17) is similar.

Since $\text{AVG}(X) = \text{SUM}(X)/N$, we have thus shown that C is satisfiable if and only if the following system of linear inequalities, denoted as $\text{CellSat}(N, S)$, is satisfied in variables $S \in \mathcal{R}$ and $N \in \mathcal{Z}^+$:

$$(N - 1)m_L + M_L \leq S \leq (N - 1)M_H + m_H \quad (21)$$

$$S_L \leq S \leq S_H \quad (22)$$

$$A_L N \leq S \leq A_H N \quad (23)$$

$$C_L \leq N \leq C_J \quad (24)$$

We mention in passing that if C is defined by a mixture of closed, open and semi-open intervals $I_{\text{MIN}}, I_{\text{MAX}}, I_{\text{SUM}}, I_{\text{AVG}}$, then some of the inequalities in the above system become strict.

Eliminating S from (21)-(24) we arrive at an equivalent system of at most 10 inequalities on N . This can be done by comparing each upper bound on S to each lower bound on S (which is a simple special case of the well-know Fourier-Motzkin elimination procedure). Assuming without loss of generality that $S_L \leq S_H$, we obtain at most 8 inequalities on N . The additional 2 inequalities on N come from (24). It remains to solve the resulting system and check whether the resulting interval on N contains an integral point. Clearly, it does if and only if C is a satisfiable cell.

Now, we turn to the case when the cell intervals are defined by interval constraints on the measures of $K \geq 2$ attributes. With the exception of COUNT, the constraints for each of the K attributes are independent of those for the remaining attributes. The only constraint common between attributes is that the records for all attributes have the same COUNT. So, to check the satisfiability of a cell C , we separately construct system $CellSat(N, S^k)$ for each attribute A^k , $k \in \{1, \dots, K\}$, and then using the method described above, independently eliminate $S^{(1)}, S^{(2)}, \dots, S^{(K)}$ from these systems. This results in a system of at most $8K + 2$ linear inequalities on N . Solving this system returns an interval on N and it can easily be checked whether it contains an integral point.

Clearly, the complexity of the above procedure can be bounded by $O(K)$ additions, subtractions, and comparisons.

For the case $J \geq 1$ we first note that the dimensions are independent of each other. For each dimension there are two possibilities. The first possibility is that the dimensional attribute considered is not aggregated as a measure attribute in the cell. In this case the interval constraints on the dimension is independent of the rest of the constraints and can be checked for validity separately in constant time. The second possibility is that the dimensional attribute is also aggregated as a measure attribute. In this case the interval constraint on the dimensional attribute is used in conjunction with the corresponding aggregate interval constraints of the measure attribute to tighten the bounds on m_L and M_H . Again this can be done in constant time. Here, note that the satisfiability of a dimension attribute maps to the satisfiability of an aggregate measure.

Thus it can be seen that the satisfiability of a cell defined by interval constraints on J dimensional and K measure interval constraints can be determined in $O(J + K)$ time. \square

C. Reachability of a cell from a view

Consider a satisfiable cell C defined by a system of interval constraints (8)-(13) on J dimension attributes D^1, \dots, D^J and measures of K distinct attributes $A^{(1)}, \dots, A^{(K)}$. Let V be a view defined by assigning some specific values to the dimension and measure attributes:

$$\begin{aligned} D^{(j)} &= [v^{(j)}] \\ \text{COUNT} &= \mathcal{C} \\ \text{MIN}(A^{(k)}) &= m^{(k)} \\ \text{MAX}(A^{(k)}) &= M^{(k)} \end{aligned}$$

$$\begin{aligned}\text{SUM}(A^{(k)}) &= \Sigma^{(k)} \\ \text{AVG}(A^{(k)}) &= a^{(k)}\end{aligned}$$

Without loss of generality, we assume that $\Sigma^{(k)} = \mathcal{C}a^{(k)}$ for all $k = 1, \dots, K$, and that $V \notin \mathcal{C}$. By definition, \mathcal{C} is reachable from V if there is a set

$$X = \{X_1, \dots, X_N, X_{N+1}, \dots, X_{N+\Delta}\} \subset D^{(1)} \times D^{(2)} \dots D^{(J)} \times \mathcal{R}^K$$

of $\mathcal{C} = N + \Delta$ records such that

- $N \in I_{\text{COUNT}}$ and $\Delta \geq 1$
- X maps into V
- $X' = \{X_1, \dots, X_N\}$ maps into \mathcal{C} .

THEOREM C.1. *The reachability of \mathcal{C} from V can be determined in $O(J + K \log K)$ time.*

Proof: As in the proof of Theorem B.1, we assume that \mathcal{C} is a closed cell defined by interval constraints (15)- (19). We start with the case $K = 1$, $J = 0$. Suppose we have a set X of $\mathcal{C} = N + \Delta$ reals $\{X_1, \dots, X_N, X_{N+1}, \dots, X_{N+\Delta}\}$ such that $\text{MIN}(X) = m$ and $\text{MAX}(X) = M$. Also suppose that the set $X' = \{X_1, \dots, X_N\}$ satisfies the inclusions $\text{MIN}(X') \in [m_L, m_H]$ and $\text{MAX}(X') \in [M_L, M_H]$. As before, we can assume without loss of generality that $m_L \leq M_L$, $m_H \leq M_H$, and that the two intervals $[m_L, m_H]$ and $[M_L, M_H]$ are nonempty. We can further assume that $m_L \geq m$ and $M_H \leq M$, for otherwise we can replace m_L by m and M_H by M . There are four possible cases:

- (i) m and M are outside the cell boundaries: $m_L > m$ and $M_H < M$
- (ii) m meets the cell boundary and M is outside the cell boundary: $m_L = m$ and $M_H < M$
- (iii) m is outside the cell boundary and M meets the cell boundary: $m_L > m$ and $M_H = M$
- (iv) m and M both meet the cell boundary: $m_L = m$ and $M_H = M$.

Case (i): $m_L > m$ and $M_H < M$. As mentioned in the proof of Theorem B.1, the inclusions $\text{MIN}(X') \in [m_L, m_H]$ and $\text{MAX}(X') \in [M_L, M_H]$ for $X' = \{X_1, \dots, X_N\}$ yield the following sharp bounds on $S = \text{SUM}(X')$:

$$M_L + m_L(N - 1) \leq S \leq m_H + M_H(N - 1). \quad (25)$$

Let $\delta = \Sigma - S$ be the sum of the remaining $\Delta = \mathcal{C} - N$ elements $X \setminus X' = \{X_{N+1}, \dots, X_{N+\Delta}\}$. Since $X \setminus X' \subset [m, M]$, $\text{MIN}(X \setminus X') = m$,

and $\text{MAX}(X \setminus X') = M$, it follows that

$$M + m(\mathcal{C} - N - 1) \leq \delta \leq m + M(\mathcal{C} - N - 1). \quad (26)$$

As before, it is easy to show that any real δ satisfying the above inequalities can be realized as the sum of Δ reals $X_{N+1}, \dots, X_{N+\Delta}$ such that $X_{N+1}, \dots, X_{N+\Delta} \in [m, M]$, $\text{MIN}\{X_{N+1}, \dots, X_{N+\Delta}\} = m$, and $\text{MAX}\{X_{N+1}, \dots, X_{N+\Delta}\} = M$. For $\Delta = 1$ note that case (i) is clearly not possible as only one of m and M would be outside the cell boundary. Accordingly, (26) is empty. For $\Delta = 2$ (26) clearly holds. Finally, for $\Delta \geq 3$, the interval $[M + m(\Delta - 1), m + (\Delta - 1)M]$ can be written as $m + M + (\Delta - 2)[m, M]$. Hence any real δ satisfying (26) can be represented as $X_{N+1} + X_{N+2} \dots + X_{N+\Delta}$ with $X_{N+1} = m$, $X_{N+2} = M$, and $X_{N+3}, \dots, X_{N+\Delta} \in [m, M]$. This shows that (26) gives the correct range for all possible values of δ .

Since $\delta = \Sigma - S$, (26) can be written as follows:

$$\Sigma - m - M(\mathcal{C} - N - 1) \leq S \leq \Sigma - M - m(\mathcal{C} - N - 1).$$

Combining the above inequalities with (25) and taking into account the cell bounds $\text{SUM}(X') \in I_{\text{SUM}} \doteq [S_L, S_H]$, $\text{AVG}(X') \in I_{\text{AVG}} \doteq [A_L, A_H]$, and $N \in I_{\text{COUNT}} \doteq [C_L, C_H]$, we conclude that C is reachable from V if and only if the following system of linear inequalities, denoted as $\text{ViewReach1}(N, S)$, can be satisfied in variables $S \in \mathcal{R}$ and $N \in \mathcal{Z}^+$:

$$M_L + m_L(N - 1) \leq S \leq m_H + M_H(N - 1) \quad (27)$$

$$\Sigma - m - M(\mathcal{C} - N - 1) \leq S \leq \Sigma - M - m(\mathcal{C} - N - 1) \quad (28)$$

$$S_L \leq S \leq S_H \quad (29)$$

$$A_L N \leq S \leq A_H N \quad (30)$$

$$C_L \leq N \leq \text{MIN}\{C_H, \mathcal{C} - 1\}. \quad (31)$$

Eliminating S from (27)-(31) we obtain an equivalent system of at most 16 inequalities on N . Solving this system we will be able to get an interval for N and check if this resulting interval contains an integral point. C is reachable from V if and only if the obtained interval on N contains an integral point.

Case (ii): $m_L = m$ and $M_H < M$. Now we split into two subcases: $m \in \{X_{N+1}, \dots, X_{N+\Delta}\} = X \setminus X'$ and $m \notin X \setminus X'$. If $m \in X \setminus X'$, then the arguments presented above for case (i) show that the reachability of V from C is still equivalent to the feasibility of $\text{ViewReach1}(N, S)$. The assumption that $m \notin X \setminus X'$, however, produces a different system of linear inequalities. Specifically, since $m \in \{X_1, \dots, X_N\}$, the upper bound on $S = \text{SUM}(X')$ in (25) becomes tighter:

$$M_L + m(N - 1) \leq S \leq m + M_H(N - 1). \quad (32)$$

On the other hand, since the requirement that $m \in \{X_{N+1}, \dots, X_{N+\Delta}\}$ is dropped, the upper bound on $\delta = X_{N+1} + \dots + X_{N+\Delta}$ in (26) increases:

$$M + m(\mathcal{C} - N - 1) \leq \delta \leq M(\mathcal{C} - N). \quad (33)$$

As before, it is easily seen that (32) and (33) provide exact ranges for all possible values of S and δ , respectively. The assumption $m \notin X \setminus X'$ thus leads to the following system of linear inequalities $ViewReach2(S, N)$:

$$M_L + m(N - 1) \leq S \leq m + M_H(N - 1) \quad (34)$$

$$\Sigma - M(\mathcal{C} - N) \leq S \leq \Sigma - M - m(\mathcal{C} - N - 1) \quad (35)$$

$$S_L \leq S \leq S_H \quad (36)$$

$$A_L N \leq S \leq A_H N \quad (37)$$

$$C_L \leq N \leq \min\{C_H, \mathcal{C} - 1\}. \quad (38)$$

Eliminating S independently from $ViewReach1(S, N)$ and $ViewReach2(S, N)$ results in two separate intervals on N such that C is reachable from V if and only if any one of these two intervals contains an integral point.

Case (iii): $m < m_L$ and $M_H = M$. This case is similar to (ii). We obtain two systems, the first of which is $ViewReach1(S, N)$ for the subcase $M \in X \setminus X'$. The other system, $ViewReach3(S, N)$, covers the subcase $M \notin X \setminus X'$ and is comprised of four inequalities

$$\begin{aligned} M + m_L(N - 1) &\leq S \leq m_H + M(N - 1) \\ \Sigma - m - M(\mathcal{C} - N - 1) &\leq S \leq \Sigma - m(\mathcal{C} - N) \end{aligned}$$

followed by (36), (37), (38).

Case (iv): $m = m_L$ and $M = M_L$. Now we have four subcases, three of which lead to the previously developed systems:

$ViewReach1(S, N)$ for $m \in X \setminus X'$ and $M \in X \setminus X'$

$ViewReach2(S, N)$ for $m \notin X \setminus X$ and $M \in X \setminus X'$

$ViewReach3(S, N)$ for $m \in X \setminus X$ and $M \notin X \setminus X'$.

The remaining subcase where both m and M do not belong to $X \setminus X'$ can be described by the inequalities

$$\begin{aligned} M + m(N - 1) &\leq S \leq m + M(N - 1) \\ \Sigma - M(\mathcal{C} - N) &\leq S \leq \Sigma - m(\mathcal{C} - N), \end{aligned}$$

which along with (36), (37), (38) form system $ViewReach4(S, N)$. Systems $ViewReach\{i\}(S, N)$, $i = 1, 2, 3, 4$, can then be solved for N . The given view C would be reachable from V if and only if any of the four intervals obtained on N contains an integral point.

Analogous systems $ViewReach\{i\}$ can also be constructed for cells C defined by any combination of closed, open, and semi-open intervals

I_{MIN} , I_{MAX} , I_{SUM} , and I_{AVG} . Combining all of the above cases, we thus conclude that for $K = 1$, in constant number of steps we can compute a system \mathcal{I} of at most four intervals such that C is reachable from V if and only if \mathcal{I} contains an integral point. We call \mathcal{I} a *4-interval* even if \mathcal{I} may actually contain fewer intervals.

EXAMPLE C.1. Consider the view of $\mathcal{C} = 19$ records $X = \{X_1, \dots, X_{19}\}$ with $m \doteq \text{MIN}(X) = 0$, $M \doteq \text{MAX}(X) = 75$, $\Sigma \doteq \text{SUM}(X) = 1000$, and consequently, $a \doteq \text{AVG}(X) = 1000/19$. Let C be the cell defined by the following intervals: $N = \text{COUNT}(X') \in [\mathcal{C}_L, \mathcal{C}_H] = [1, 19]$, $\text{MIN}(X') \in [m_L, m_H] = [0, 30]$, $\text{MAX}(X') \in [M_L, M_H] = [0, 50]$, $\text{SUM}(X') \in [S_L, S_H] = [-\infty, +\infty]$, and $\text{AVG}(X) \in [A_L, A_H] = [46.5, 50]$. Then systems *ViewReach1* and *ViewReach2* show that \mathcal{I} consists of 2 disjoint intervals and C is reachable from V either with $5.714 \leq N \leq 13.2$ or with $14.285 \leq N \leq 15$. Rounding the endpoints of N to integral values, we have C is reachable from V either with $N \in [6, 13]$ or with $N = 15$.

We continue with the proof of the theorem and turn to the case when the view (and the cell) intervals are defined by interval constraints on the measures of $K \geq 2$ attributes. Here again, with the exception of *COUNT*, the constraints for each of the K distinct measure attributes are independent of those for the remaining attributes. The only constraint common between attributes is that $N = \text{COUNT}(X')$ should be identical for all attributes. So, to check the reachability of V from C , it remains to separately construct the 4-interval $\mathcal{I}^{(k)}$ for each attribute $A^{(k)}$, $k = 1, \dots, K$ and then check whether the intersection of all the $\mathcal{I}^{(k)}$'s contains an integral point:

$$\mathcal{Z}^+ \cap \{\mathcal{I}^{(k)} \mid k = 1, \dots, K\} \neq \emptyset. \quad (39)$$

Note that by appropriately rounding the endpoints of each interval in $\mathcal{I}^{(k)}$, $k = 1, \dots, K$ we can make all of the endpoints integral. After such rounding, (39) becomes equivalent to determining whether or not the $\mathcal{I}^{(k)}$'s have a nonempty intersection:

$$\bigcap \{\mathcal{I}^{(k)} \mid k = 1, \dots, K\} \neq \emptyset.$$

The latter problem can easily be solved in $O(K \log K)$ time. Assume without loss of generality that each of the $\mathcal{I}^{(k)}$'s is defined by disjoint intervals and sort the endpoints of all these intervals to obtain a sorted array P of at most $8K$ points. Sweep through P from left to right, maintaining for each point $p \in P$ the difference between the numbers of the left and right endpoints $\leq p$. This difference is equal to the multiplicity of p , i.e., the number of the intervals containing p . Now

the sets $\mathcal{I}^{(k)}$ contain a common point if and only if P contains a point of multiplicity K . This proves the theorem for arbitrary K and $J = 0$.

Finally, for the case $J \geq 1$ we first note that within a view (and within a cell) the dimensions are independent of each other. Accordingly, we have the following alternatives for each dimension D^j , $j = 1, \dots, J$:

- If $D^{(j)}$ also appears as a measure attribute in the query, the dimension interval constraint in the cell and the view can be used in conjunction with the corresponding measure interval constraints to tighten the bounds on m_L and M_H . The problem then maps to checking the reachability of the measure attributes discussed above.
- If the interval value for $D^{(j)}$ in the view does not contain the interval value for $D^{(j)}$ in the cell then C is not reachable from V .

Thus it can be seen that the reachability problem for a cell defined by interval constraints on J dimensional and measures of K distinct attributes constraints can be determined in $O(J + K \log K)$ time.