Cubegrades: Generalizing Association Rules

Tomasz Imieliński Leonid Khachiyan *
Amin Abdulghani
Rutgers University
Department of Computer Science
New Brunswick, N.J. 08903 USA

September 26, 1999

1 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 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 formally define cubegrades, show methods to generate them by using efficient pruning algorithms, and finally define two query languages to generate and retrieve sets of cubegrades which satisfy user defined conditions. We also demonstrate how to evaluate simple cubegrade queries and conclude with a number of open questions and possible extensions of the work.

2 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 transactions[1] which buy bread and butter buy also cereal (that percentage is

*Supported in part by NSF Grant CCR-9618796

[1] so called market basket data

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 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) is replaced by an arbitrary measure or aggregate. This leads us to the concept of cubegrades which we introduce in this paper. For example, one could ask how the average age of buyers of bread is affected when we specialize the population of buyers of bread to buyers of bread and milk (if, of course age is stored along with transactions).

Similar questions, statements can be formed about other aggregates and measures such 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; another operations include roll up and mutation. Cubegrades with *rollup* operation show how different measures (aggregates) are affected by cube generalization; cubegrades with *mutation* hypothetically change one of the attribute values in the cube and determine how different measures are affected by such operation. Cubegrades state how different subpopulations of the database are affected by different modifications of their definitions. The effect is measured by how 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 data example. Consider a supermarket chain which maintains for each of its customers the amount she has spent monthly on different items. Also, assume the company maintains demographic data on the customer such as the area the customer lives in (suburban, urban or rural), the age group, and the approximate income range for the customer. Now, suppose the supermarket wants to explore the trend of sales in different customer cubes,

described by some set of conditions on demographics or buying habits, and see how these trends are affected by changes in the cube description. Using cubegrades, analysts would be able to express the following kinds of questions on the data:

Q1 How is the average age of buyers of salsa affected by buying soda as well?

Example answer: Drops by 10% from 26 to 24.

Q2 How is the average amount of milk bought affected by customer age among buyers of cereals?

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

Q3 How is the average amount of detergents bought affected by foods (vegetables, meat, and fish)?

Example answer: Drops by 30% for vegetables etc.

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.

As we can see, the cubegrades deal with evaluating different aggregate summaries in cubes and changes in those aggregates due to changes in the structure of the cubes. They are hybrid of the queries for association rules and OLAP. In this paper we layout a paradigm for asking such cubegrade queries. This includes defining the concept behind cubegrades; introducing a language CGQL for expressing queries on cubegrades; and defining a scheme for evaluating the expressed queries.

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 layed out. Section 4 presents the CGQL query language. In section 5 we look at how to evaluate cubegrade queries.

3 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) 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 as mentioned earlier most of these rules are restricted to the count aggregate and can only express relative changes from body of the rule to body and consequent. Only recently have this aspect come to attention [NLHP98]

and [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`. 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 queries are characterized and evaluated based on the properties (*anti-monotonicity* and *succinctness*) of each of the individual constraints. The drawback with the strategies presented is that arbitrary Boolean conditions would be difficult to implement.

In contrast, in OLAP - an area which we believe is closely intertwined with association rules and which 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. The aim of this paper is to make such measures and operations available on rules. Instead of operations invoked on specific cubes and directed by the users (user-centric) we make these operations native to rules as part of a 'discovery' process. In the context of OLAP, this could be considered as building upon recent directions for providing automated analysis tools on top of cubes, while minimizing user input [SAM98], [Sar99].

4 Definition of cubegrades

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 a $k-$*conjunct*.

For a given k-conjunct and a database of records:.

- the set of records that satisfy the k-conjunct define the *cube (segment)* for that conjunct.
- The attributes that constitute the k-conjunct define the *dimensions* of the cube.
- Attributes that are aggregates over objects which satisfy the cube definition define the *measure attributes* of the cube. The measure attributes 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.

For example the conjunct **(areaType='rural' incomeRange=[45K-60K])** defines a cube of rural households earning between 45K to 60K. The dimensional attributes here are `areaType` and `incomeRange`. A possible measure attribute for this cube could be `AVG(milk)`, which gives the average amount spent on milk in the cube. The cube **(areaType='rural' incomeRange=[45K-60K])** is a subcube of **(areaType='rural')**. Note if the k-conjunct T is a subset of a l-conjunct T' ($k \leq l$) then the cube defined with T' is a subcube for the cube T.

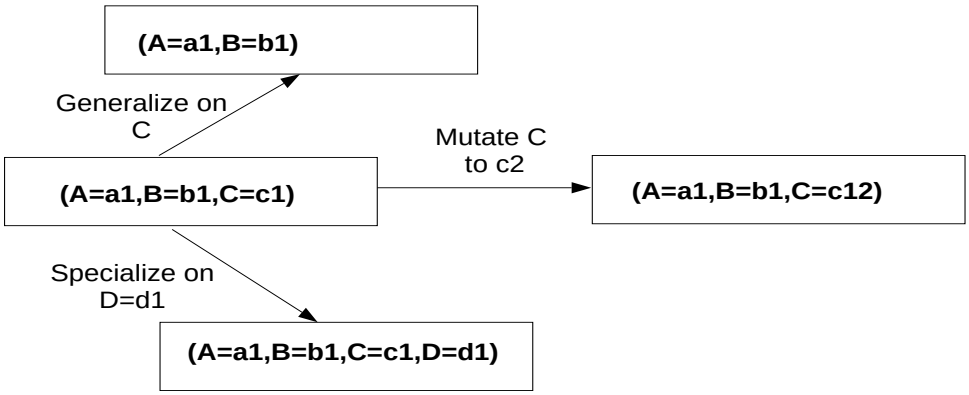

Figure 1: Scenarios of the three cubegrade operations

A numeric attribute such as income may be used both as a measure attribute and a descriptor (in which case its discretized).

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 an aggregate (`COUNT`) when "restricting" or "drilling down" the Body cube to the Body + Consequent cube. The confidence of the rule measures the impact of the consequent cube on the the drill down of the body cube. There are two ways these 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 values in a subset of the attributes in the original cube definition with the others remaining the same.

These generalized cube modifications are defined as *Cubegrades*. A cubegrade expresses how a modification in the structure of a given cube affects a predefined measure. The original cube which is being modified is referred to as the *source cube* and the modified cube as *target 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 values of each measure `M` $\in$ *Measures* in the *Source-Cube* and *Delta Values* is a list containing the relative change in the value of `M` $\in$ *Measures* between the *Source-Cube* and *Target-cube*. The element `DeltaM` of *Delta Values* defines the relative change of value for the measure `M`. Thus, for example, if `AVG(Salary)` = 40K and `DeltaAVG(salary)` is 110% then it means that the `AVG(Salary)` in the *source cube* is 40 and the `AVG(Salary)` in the *target cube* is 10% higher, ie 44K. A cubegrade can visually be represented as a rule form:

`SourceCube` $->$ `TargetCube [Measures, Values, Delta Values]`

Finally, we will distinguish three types of the cubegrades:

- Specializations
- Generalizations
- Mutations

A cubegrade is called a *specialization* if the set of descriptors of the target cube is a superset of the set of descriptors of the source cube. It is called a *generalization* if the set of descriptors of the target cube is a subset of the set of descriptors of the source cube. Finally, it is called *mutation* if the target cube and source cube have the same set of attributes (they are "union compatible" so to speak, as the term has been used in relational 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.

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 `DeltaCOUNT`.

Example 4.1 *Following are some examples of cubegrades.*

- *The average age of buyers who purchase 20-30 dollars worth of milk monthly drops by 10% among buyers who also buy some cereal.*

  (salesMilk=[$20,$30]) $->$ **(salesMilk=[$20,$30], salesCereal>$0)**

  `[AVG(Age), AVG(Age) = 23, DeltaAVG(Age) = 90%]`

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

  (areaType='urban') $->$ **(areaType='urban', Age=[18,30])**

  `[AVG(salesMilk), AVG(salesMilk) = $12.40, DeltaAVG(salesMilk)= 80%]`

5 Query Languages: CubeQL and CubegradeQL

As pointed out in [IV99] is it important to help a user to selectively generate association rules and then later, query those which have been persistently stored. A similar need exists for cubegrades. In this section we define the concept of cube and cubegrade query languages. We distinguish between the two here for the sake of syntax simplicity. We start by defining *CubeQL* for cubes and then define *CubegradeQL* for cubegrades.

5.1 CubeQL

A *CubeQL* query can be expressed using the following syntax:

```
GET CUBES
 FROM DB
 WHERE  <conditions>
```

The query generates cubes over the elements of the database satisfying the conditions described in the "where " clause. The type of conditions possible in the where clause are as follows:.

- *cube descriptor condition*: This defines a Boolean condition on the descriptor patterns that can appear in the *answer cubes* . A descriptor pattern could be one of the following:.

  A_i=v: A descriptor for attribute A_i matches this pattern if A_i has the value v in the cube.

  A_i IN <interval>: A descriptor for attribute A_i matches this pattern if the descriptor's discretized interval is a subset of the pattern's interval. This assumes that A_i is a discretized attribute.

  $A_i = *$: This matches any descriptor for A_i. It can be used for both categorical or discretized dimensions.

  A Boolean formula over the descriptor patterns is called a <`conjunct pattern`>. The condition is expressed as < `CUBE MATCHES` <`conjunct pattern`> >. A cube would satisfy the condition if the cube's descriptors exactly match the Boolean condition on the descriptor patterns.

  As an example consider the following condition:.

```
CUBE MATCHES (areaType='urban' and
     (salesBread=* or  salesCereal=[35,50]
        or salesMilk=*))
```

This condition is true for cubes matching the pattern **(areaType='urban')**, and any combination of the following patterns **(salesBread=*), (salesCereal=[35,50]) (salesMilk=*)**. Examples of cubes that satisfy this would include **(areaType='urban', salesBread=[5,10])** or **(areaType='urban', salesBread=[15,20])** . Cubes not satisfying the pattern include **(areaType = 'urban')**, **(salesBread=[5,10])** or **(areaType='urban', salesCheese = [10,50])**.

- *Measures condition*: This specifies the list of measures to be computed for the cube. Its syntactically represented as `MEASURES = <measure list>`. All the Values computed for the cube must have a corresponding measure entry in the `<measure list>`.

- *Value condition*: This specifies a Boolean condition on the measure values for the cube. Examples include:. `AVG(salesMilk)>50`, or `AVG(salesMilk) < MIN(salesBread)*1.5`.

- *Length condition*: This is a condition on the conjunct length of the cube. The length of the cube is given by the keyword `LENGTH`.

The "where" clause is an arbitrary Boolean condition built by combining the different condition types through `AND/OR` operators. For an empty clause all possible descriptors on all the attributes in the database are allowed in the cube definition and *Measures* is defaulted to contain an empty list.

Consider for example the following *cube query* and its *CubeQL* formulation.

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.

CubeQL syntax

```
GET CUBES
FROM customer
WHERE CUBE MATCHES (areaType=* or age=* or income=* or
                    salesBread=* or  salesMilk=* or
                    salesCookies=* or saleSoda=*)
      AND MEASURES = {COUNT(*), AVG(salesCereal)}
      AND COUNT(*)>1000 and AVG(salesCereal)<20
```

In this query the `MATCHES` clause specifies the set of descriptor patterns for the cube (ie, the constraints on the dimensions of the cube) , the `COUNT` and `AVG` conditions define the Value conditions and the `MEASURE` clause specifies the set of measure to return with the answer cubes.

5.2 CubegradeQL

Next, we propose the *CubegradeQL* (*Cubegrade Query Language*), a language for manipulating the cubegrades. Its syntax is shown below:.

```
GET [SPECIALIZE(X)|GENERALIZE(X)|MUTATE(X)] CUBEGRADES
 FROM DB
 WHERE  <condition>
```

The query generates cubegrades over the elements of the database satisfying the conditions described in the "where " clause. The type of conditions possible in the "where" clause are from the following:.

- *Source cube and target cube conditions*: These are similar to the cube descriptor conditions discussed for cube queries . They are expressed as Boolean conditions on the descriptors that can appear in the *source cube* or the *target cube* of the cubegrade. Syntactically, they are expressed as <`SOURCE_CUBE MATCHES` < `conjunct description` > > or < `TARGET_CUBE MATCHES` < `conjunct description` > >. For example, consider the condition:

  ```
  SOURCE_CUBE MATCHES (areaType='urban' and
       (salesBread=* or  salesCereal=[35,50]
          or salesMilk=*))
  ```

  This is satisfied when source is **(areaType='urban' salesBread=[5,10])** but is not satisfied when it is (**areaType='urban'**), **(salesBread=[5,10])** or **(areaType='urban' salesCookies=[10,50])**

- *Join conditions:* These define the conditions between the `SOURCE` and `TARGET` cubes in the cubegrade. In principle any condition can be defined for the join. A Cartesian product operation between the source and target cubes is implied when no condition is present.

  For our purposes we deal with three specific types of cubegrades

  Since arbitrary join conditions would computationally be prohibitive we explicitly define three join operations that would be of use to us, namely specialization, generalization, and mutation. The operations are *implicit* in case the type of the cubegrade is clearly specified in the `GET CUBEGRADE` clause.

  - Specialization: Here the join condition specifies that the target cube has to have more descriptors including the ones in the source cube. Its written out as `TARGET_CUBE SPECIALIZES SOURCE_CUBE`.

    We further define here a useful variant of `SPECIALIZES` operation, `SPECIALIZES(X)` which will extend the source cube by those dimensions in X which are not already present in the source cube. The extension of the source cube will be on all possible combinations of values in X.

- Generalization: Here the join specifies that the target cube has a subset of the descriptors of the source cube. Its represented as `TARGET_CUBE GENERALIZES SOURCE_CUBE`.

  We further distinguish more specific subset relations, namely the relation `GENERALIZES(X)` "where" X is a set of attributes which will be dropped from the source cube. This corresponds to the generalization of the source cube by projecting out the dimensions in X.

- Mutation: Here the join condition specifies that the target cube has to have the same set of attributes as the source cube but with different descriptors. Syntactically, it can be expressed as `TARGET_CUBE UNION-COMPATIBLE SOURCE_CUBE`.

  The variant we further introduce here is `UNION-COMPATIBLE(X)` which restricts the source and target cubes to not only being union compatible but to also agree on the attributes in X (those of which also present in source cube). This restriction allows mutating only selected dimensions not in X.

  We may also utilize user defined functions to define the "mutated" values and their dependence on the original source values. Such functions can for example be defined over numeric domains by modifying age, salary, location, time etc (for example increasing salary, moving time and/or location) and observing their effect on selected source cube aggregates.

- *Measures conditions*: This is similar to the `MEASURES` clause for *CubeQL*. It specifies the set of measures for which the Values and the Delta values are to be computed.

- *Value and Delta-Value conditions:* This defines conditions on the Values and the the Delta Values for the source and the target cubes respectively.

- *Length conditions:* This specifies the constraints on the length of the conjuncts for source and target cubes. The keyword `SOURCE_LENGTH` gives the length of the source cube while `TARGET_LENGTH` gives the length of the target cube.

A $<$ *condition* $>$ in the "where" clause is an arbitrary Boolean condition built by ANDing/ORing the above condition types. For an empty "where" clause all possible descriptors on all the attributes in the database are allowed in the source cube and target cube definitions, the join condition is implied to be a specialization and *Measures* is defaulted to contain an empty list.

Notice, that both CubeQL and CubegradeQL allow only simple "selection" queries, no nesting and join operations (other than between source and target cubes) are provided. These extensions we will consider in the future work.

We conclude this section with some examples of queries, some of them formulated in CubegradeQL:

Q6 What specializations on the cubes considered in Q5 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 in S who also buy more than $20 worth of cookies per month.

Explanation: This is a specialization cubegrade query.

CubegradeQL syntax:

```
GET SPECIALIZATION CUBEGRADES
FROM customer
WHERE SOURCE_CUBE MATCHES (areaType=* or age=* or income=*
                  or salesBread=* or  salesMilk=*
                  or salesCookies=* or salesSoda=*)
      AND TARGET_CUBE MATCHES  (areaType=* or age=*
                  or income=* or salesBread=*
                  or  salesMilk=* or salesCookies=*
                  or salesSoda=*)
      AND TARGET_CUBE SPECIALIZE SOURCE_CUBE
      AND MEASURES = {COUNT(*), AVG(salesCereal)}
      AND COUNT(*)>1000 and AVG(salesCereal)<20
      AND DeltaAVG(salesCereal)>1.5 and DeltaCOUNT(*)>0.5
```

If the specialization is on a set of selected attributes (for example `salesMilk`, and `salesBread`) then the expression becomes:.

```
GET SPECIALIZATION(salesMilk, salesBread)  CUBEGRADES
FROM customer
WHERE SOURCE_CUBE MATCHES (areaType=* or age=* or income=* or
                  salesBread=* or  salesMilk=* or
                  salesCookies=* or salesSoda=*)
      AND MEASURES = {COUNT(*), AVG(salesCereal)}
      AND COUNT(*)>1000 and AVG(salesCereal)<20
      AND DeltaAVG(salesCereal)>1.5 and DeltaCOUNT(*) >0.5
```

Q7 Which cubes of more than 1000 customers showed an increase in the average amount of cereal bought, in 1998 as compared to 1997?

Example answer: Young buyers of ages <20-30> who shop at least once per week.

Explanation: It is a mutation cubegrade query

CubegradeQL syntax:

```
GET  CUBEGRADES
FROM customer
WHERE SOURCE_CUBE MATCHES (areaType=* or age=*
```

```
                or income=* or salesBread=*
                or  salesMilk=* or salesCookies=*
                or numVisits=*)
AND TARGET_CUBE MATCHES  (areaType=* or age=*
                or income=* or  salesBread=*
                or  salesMilk=* or salesCookies=*
                or numVisits=*)
AND TARGET_CUBE UNION-COMPATIBLE SOURCE_CUBE
AND MEASURES = {COUNT(*), AVG(salesCereal)}
AND COUNT(*)>1000 and AVG(salesCereal)<20
AND DeltaAVG(salesCereal)>1.5 and DeltaCOUNT(*)>0.5
```

Q8 Find in which cubes suburban customers generating more than 40% sales than their urban counterparts?

Example answer: The cube of cheese buyers.

Explanation: This is a mutation cubegrade query

Q9 For the cubes considered in Q5 find in which specializations do customers spend twice as much on oats then 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.

Q10 In which cubes customers aged between 20 to 30 generate more than 40% of the total sales?

Example answer: Buyers of chocolate and icecream.

Q11 In which cubes of at least 1000 customers the age bracket of <40,50> had the highest average spending on salsa as compared to the other age brackets in the same cube.

Example answer: Buyers of crackers and cheese.

Explanation: This is an example of a nested mutation cubegrade query.

6 Query Evaluation for cube and cubegrade queries

In this section, we look at how to evaluate CubeQL queries. The CubeQL evaluation will be useful in evaluating general cubegrade queries expressed in CubegradeQL.

6.1 Pruning for cube queries

In principle, cube queries are evaluated in a similar way as association rule queries. 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 6.1 *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, as the following the theorem shows, determining whether a query Q is monotonic in terms of this definition is an NP-hard problem.

Theorem 6.1 *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 $X_1, X_2 \ldots X_N$, each of which is a positive integer. Given a positive integer b, consider the query $\texttt{SUM}(\cdot) = b$. Assume without loss of generality that Q is `FALSE` at $\mathcal{D}$, i.e., $\texttt{SUM}(\mathcal{D}) = X_1 + \ldots + X_N \neq b$. 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 $\texttt{SUM}(Y) \doteq \sum\{X_i \mid i \in Y\} = b$. 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 $\texttt{AVG}(\cdot) = 0$ 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, $i = 1, \ldots, m$, 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

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

Furthermore, since $k < n$, the above query is `FALSE` at $X = \mathcal{D}$. Hence (1) is equivalent to the NP-complete vertex cover problem.

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

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 checked in time polynomial in the the size of the database.

To workaround this problem we define another notion of monotonicity called *structural view (s-view) monotonicity*. Suppose we have a set $\mathcal{S}$ of dimension attributes and measures. Define a *view* on $\mathcal{S}$ as an assignment of values to the elements in the set. If the assignment is such that the values hold for the dimension attributes and measures in a given cube X, then this is a view for X on the set $\mathcal{S}$. So, for example, if in a cube the average sales of bread is \$15 then the view on the set {`areaType, AVG(amtBread)`} for the cube is {`areaType=ALL, AVG(amtBread)=15`}. The value `ALL` for the attribute `areaType` is a special value denoting that all values are included for that attribute. If its restricted to some value like `rural` then the view on the set becomes {`areaType=ALL, AVG(amtBread)=15`}. Extending the definition, a view on a query is an assignment of values for the set of attributes and measures in the query expression.

Definition 6.2 *A query $Q(\cdot)$* s-view monotonic *on view V if for any cube X in any database $\mathcal{D}$ s.t. V is view for X, the condition Q is* `FALSE` *for X implies Q is* `FALSE` *for all $X' \subseteq X$.*

An important property of s-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.

The approach that we present for determining structural monotonicity is borrowed from the Quantifier Elimination methods used in the first order theory of real addition with order, see e.g. [Bas97] [HRS93]. In what follows, we assume that no cube length conditions are present and that we are dealing with standard SQL aggregates `COUNT, MIN, MAX, SUM` and `AVG` for the measures. In addition, it is assumed that that the value constraints in the "where" clause are of the form `agg(A)` $\oplus$ `c`, where `agg(X)` is a measure, $\oplus \in \{<\ .\ >, =, !=\}$ and c is a constant threshold.

We refer to the approach as *Grid Base Pruning* or (*GBP*). It involves construction of a multidimensional grid. Every axis of the grid corresponds to a distinct dimensional attribute or a measure in the query expression. Each is partitioned into a number of non-overlapping *intervals*. For an aggregate axis the intervals are defined by a sort on the thresholds used with the aggregate in

the query expression. So, for example if in a query thresholds $c_1 < c_2 < \ldots < c_k$ were used with `agg(A)`, each interval constructed would have endpoints c_i and c_j with $i = 0, 1, \ldots, k$ 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 dimensional axis, the intervals are defined based on the actual values c_i the attribute was compared with in the query. These intervals also 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 $c_1, c_2, \ldots, c_k$. `ALL` can logically be viewed as a special interval encompassing all other intervals for the attribute.

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.

Following this 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 `MIN(A)`≥ 6 `OR MAX(A)`≤ 5. One of the cells for this query is the Cartesian product of the intervals `MIN(A)=(6,`∞`), 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. In section 6.1.1 we give an $O(J + K)$-time algorithm to check the satisfiability of a cell defined by interval constraints on J distinct dimensional attributes and the aggregates of K distinct attributes. Our algorithm can be regarded as a simplification and generalization of the method suggested for one attribute cells in [RSSS98].

Now, we can outline the steps needed to determine whether a given query, Q, is s-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 V is *reachable* from the cell. By definition, V is reachable from 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).

It is easy to see that Q is s-view monotonic at V if and only if V is not reachable from any cell C in the disjunctive normal form for the query. As we show in section 6.1.2, the reachability of a view from a cell can be checked in $O(J + K \log K)$ time, where as before J denotes the number of distinct dimensions in the query expression and K is the number of distinct attributes used as measures in the query. (For example `MIN(Age), AVG(Age)` are measures on a single attribute `Age`). Hence we obtain the following result.

Theorem 6.2 *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.

6.1.1 Satisfiability of a cell

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

$$D^{(j)} \in [V^{(j)}] \tag{2}$$

$$\texttt{COUNT} \in I_{\texttt{COUNT}} \tag{3}$$

$$\texttt{MIN}(A^{(k)}) \in I^{(k)}_{\texttt{MIN}} \tag{4}$$

$$\texttt{MAX}(A^{(k)}) \in I^{(k)}_{\texttt{MAX}} \tag{5}$$

$$\texttt{SUM}(A^{(k)}) \in I^{(k)}_{\texttt{SUM}} \tag{6}$$

$$\texttt{AVG}(A^{(k)}) \in I^{(k)}_{\texttt{AVG}} \tag{7}$$

where $[V^{(j)}],\ j = 1, \ldots, J,\ I_{\texttt{COUNT}}$ and $I^{(k)}_{\texttt{MIN}},\ I^{(k)}_{\texttt{MAX}},\ I^{(k)}_{\texttt{SUM}},\ I^{(k)}_{\texttt{AVG}},\ k = 1, \ldots, K$, 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 $X = \{X_1 \ldots, X_N\} \in D^{(1)} \times D^{(2)} \ldots D^{(J)} \times \mathcal{R}^k$ of $N \in I_{\texttt{COUNT}}$ records whose values for the J dimension attributes and for the measures of the K attributes belong to the cell.

Theorem 6.3 *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.,

$$D^j \in [V^j] \tag{8}$$

$$\texttt{COUNT} \in I_{\texttt{COUNT}} \doteq [C_L, C_H] \tag{9}$$

$$\texttt{MIN}(A^{(k)}) \in I^{(k)}_{\texttt{MIN}} \doteq [m^{(k)}_L, m^{(k)}_H] \tag{10}$$

$$\texttt{MAX}(A^{(k)}) \in I^{(k)}_{\texttt{MAX}} \doteq [M^{(k)}_L, M^{(k)}_H] \tag{11}$$

$$\texttt{SUM}(A^{(k)}) \in I^{(k)}_{\texttt{SUM}} \doteq [S^{(k)}_L, S^{(k)}_H] \tag{12}$$

$$\texttt{AVG}(A^{(k)}) \in I^{(k)}_{\texttt{AVG}} \doteq [A^{(k)}_L, A^{(k)}_H] \tag{13}$$

where $j = 1, \ldots, J$ and $k = 1, \ldots, K$. 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 $\{X_1, \ldots, X_N\}$ such that $\texttt{MIN}(\texttt{X}) \in [m_L, m_H]$ and $\texttt{MAX}(\texttt{X}) \in [M_L, M_H]$. 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 $S = \texttt{SUM}(\texttt{X})$, then

$$(N-1)m_L + M_L \leq S \leq (N-1)M_H + m_H. \tag{14}$$

Conversely, it is easy to see that any S satisfying (14) can be realized as the sum of N reals $X_1, \cdots, X_N$ satisfying the interval constraints (10) and (11). 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 (14) can be written as $X_1 + X_2 + X_3 + \cdots + X_N$, where $X_1 \in [m_L, m_H]$, $X_2 \in [M_L, M_H]$, and $X_3, \cdots, X_N \in [m_L, M_H]$. It remains to note that $m_L \leq \texttt{MIN}(\texttt{X}) \leq X_1 \leq m_H$, where the first inequality follows from the fact that $X_1, \cdots X_N \in [m_l, M_H]$. This proves (10). The proof for (11) is similar.

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

$$(N-1)m_L + M_L \leq S \leq (N-1)M_H + m_H \tag{15}$$

$$S_L \leq S \leq S_H \tag{16}$$

$$A_L N \leq S \leq A_H N \tag{17}$$

$$C_L \leq N \leq C_J \tag{18}$$

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

Eliminating S from (15)-(18) 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 (18). 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 $A^k,\ k \in \{1, \ldots, K\}$, and then using the method described above, independently eliminate $S^{(1)}, S^{(2)}, \cdots S^{(K)}$ 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. □

6.1.2 Reachability of a view from a cell

Consider a satisfiable cell C defined by a system of interval constraints (2)-(7) on J dimension attributes $D^1, \ldots, D^J$ and measures of K distinct attributes $A^{(1)}, \cdots, A^{(K)}$. Let V be a view defined by assigning some specific values to the dimension and measure attributes:

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

$$\texttt{SUM}(A^{(k)}) = \Sigma^{(k)}$$
$$\texttt{AVG}(A^{(k)}) = a^{(k)}$$

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

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

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

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

Theorem 6.4 *The reachability of V from C can be determined in $O(J+K \log K)$ time.*

Proof: As in the proof of Theorem 6.3, we assume that C is a closed cell defined by interval constraints (9)- (13). We start with the case $K = 1,\ J = 0$. Suppose we have a set X of $\mathcal{C} = N + \Delta$ reals $\{X_1, \ldots, X_N, X_{N+1}, \ldots X_{N+\Delta}\}$ such that $\texttt{MIN}(\texttt{X}) = m$ and $\texttt{MAX}(\texttt{X}) = M$. Also suppose that the set $X' = \{X_1, \ldots, X_N\}$ satisfies the inclusions $\texttt{MIN}(\texttt{X}') \in [m_L, m_H]$ and $\texttt{MAX}(\texttt{X}') \in [M_L, M_H]$. 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 6.3, the inclusions $\texttt{MIN}(\texttt{X}') \in [m_L, m_H]$ and $\texttt{MAX}(\texttt{X}') \in [M_L, M_H]$ for $X' = \{X_1, \ldots, X_N\}$ yield the following sharp bounds on $S = \texttt{SUM}(\texttt{X}')$:

$$M_L + m_L(N-1) \leq S \leq m_H + M_H(N-1). \tag{19}$$

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

$$M + m(\mathcal{C} - N - 1) \leq \delta \leq m + M(\mathcal{C} - N - 1). \tag{20}$$

As before, it is easy to show that any real δ satisfying the above inequalities can be realized as the sum of Δ reals $X_{N+1}, \cdots, X_{N+\Delta}$ such that $X_{N+1}, \ldots, X_{N+\Delta} \in$

$[m, M]$, $\texttt{MIN}\{\texttt{X}_{\texttt{N}+1}, \ldots, \texttt{X}_{\texttt{N}+\Delta}\} = m$, and $\texttt{MAX}\{\texttt{X}_{\texttt{N}+1}, \ldots, \texttt{X}_{\texttt{N}+\Delta}\} = M$. 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, (20) is empty. For $\Delta = 2$ (20) 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 (20) can be represented as $X_{N+1} + X_{N+2} \cdots + X_{N+\Delta}$ with $X_{N+1} = m$, $X_{N+2} = M$, and $X_{N+3}, \ldots, X_{N+\Delta} \in [m, M]$. This shows that (20) gives the correct range for all possible values of δ.

Since $\delta = \Sigma - S$, (20) can 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 (19) and taking into account the cell bounds $SUM(X') \in I_{\texttt{SUM}} \doteq [S_L, S_H]$, $\texttt{AVG}(\texttt{X}') \in I_{\texttt{AVG}} \doteq [A_L, A_H]$, and $N \in I_{\texttt{COUNT}} \doteq [C_L, C_H]$, we conclude that V is reachable from C if and only if the following system of linear inequalities, denoted as $ViewReach1(N, S)$, can be satisfied in variables $S \in \mathcal{R}$ and $N \in \mathcal{Z}^+$:

$$M_L + m_L(N - 1) \leq S \leq m_H + M_H(N - 1) \tag{21}$$
$$\Sigma - m - M(\mathcal{C} - N - 1) \leq S \leq \Sigma - M - m(\mathcal{C} - N - 1) \tag{22}$$
$$S_L \leq S \leq S_H \tag{23}$$
$$A_L N \leq S \leq A_H N \tag{24}$$
$$C_L \leq N \leq \texttt{MIN}\{\texttt{C}_\texttt{H}, \mathcal{C} - 1\}. \tag{25}$$

Eliminating S from (21)-(25) 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. V is reachable from C 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: $m \in \{X_{N+1}, \ldots, X_{N+\Delta}\} = X \setminus X'$ 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 $ViewReach1(N, S)$. The assumption that $m \notin X \setminus X'$, however, produces a different system of linear inequalities. Specifically, since $m \in \{X_1, \ldots, X_N\}$, the upper bound on $S = \texttt{SUM}(\texttt{X}')$ in (19) becomes tighter:

$$M_L + m(N - 1) \leq S \leq m + M_H(N - 1). \tag{26}$$

On the other hand, since the requirement that $m \in \{X_{N+1}, \ldots, X_{N+\Delta}\}$ is dropped, the upper bound on $\delta = X_{N+1} + \ldots + X_{N+\Delta}$ in (20) increases:

$$M + m(\mathcal{C} - N - 1) \leq \delta \leq M(\mathcal{C} - N). \tag{27}$$

As before, it is easily seen that (26) and (27) 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 $ViewReach2(S, N)$:

$$M_L + m(N-1) \leq S \leq m + M_H(N-1) \tag{28}$$
$$\Sigma - M(\mathcal{C} - N) \leq S \leq \Sigma - M - m(\mathcal{C} - N - 1)) \tag{29}$$
$$S_L \leq S \leq S_H \tag{30}$$
$$A_L N \leq S \leq A_H N \tag{31}$$
$$C_L \leq N \leq min\{C_H, \mathcal{C} - 1\}. \tag{32}$$

Eliminating S independently from $ViewReach1(S, N)$ and $ViewReach2(S, N)$ results in two separate intervals on N such that V is reachable from C 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 $M \in X \backslash X'$. The other system, $ViewReach3(S, N)$, covers the subcase $M \notin X \setminus X'$ and is comprised of four inequalities

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

followed by (30), (31), (32).

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

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

which along with (30), (31), (32) form system $ViewReach4(S, N)$. Systems $ViewReach\{i\}(S, N)$, $i = 1, 2, 3, 4$, can then be solved for N. The given view V would be reachable from C 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 $I_{\texttt{MIN}}$, $I_{\texttt{MAX}}$, $I_{\texttt{SUM}}$, and $I_{\texttt{AVG}}$. 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 V is reachable from C 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 6.1 *Consider the view of* $\mathcal{C} = 19$ *records* $X = \{X_1, \ldots, X_{19}\}$ *with* $m \doteq \texttt{MIN(X)} = 0$, $M \doteq \texttt{MAX(X)} = 75$, $\Sigma \doteq \texttt{SUM(X)} = 1000$, *and consequently,* $a \doteq \texttt{AVG(X)} = 1000/19$. *Let* C *be the cell defined by the following intervals:* $N = \texttt{COUNT(X')} \in [\mathcal{C}_L, \mathcal{C}_H] = [1, 19]$, $\texttt{MIN(X')} \in [m_L, m_H] = [0, 30]$, $\texttt{MAX(X')} \in [M_L, M_H] = [0, 50]$,

$\texttt{SUM}(\texttt{X}') \in [S_L, S_H] = [-\infty, +\infty]$, *and* $\texttt{AVG}(\texttt{X}) \in A_L, A_H] = [46.5, 50]$. *Then systems* $ViewReach1$ *and* $ViewReach2$ *show that* $\mathcal{I}$ *consists of* 2 *disjoint intervals and* V *is reachable from* C *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* V *is reachable from* C *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 $N = \texttt{COUNT}(\texttt{X}')$ 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 $A^{(k)},\ k = 1, \ldots, K$ and then check whether the intersection of all the $\mathcal{I}^{(k)}$'s contains an integral point:

$$\mathcal{Z}^+ \cap \{\mathcal{I}^{(k)} \mid k = 1, \ldots, K\} \neq \emptyset. \tag{33}$$

Note that by appropriately rounding the endpoints of each interval in $\mathcal{I}^{(k)},\ k = 1, \ldots, K$ we can make all of the endpoints integral. After such rounding, (33) becomes equivalent to determining whether or not the $\mathcal{I}^{(k)}$'s have a nonempty intersection:

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

The latter problem can easily be solved in $O(K \log K)$ comparisons. For instance, this can be done by the following divide-and-conquer strategy:

Find the median of the endpoints of all intervals in $\mathcal{I}^{(k)},\ k = 1, \ldots, K$. Check whether the median belongs to $\cap\{\mathcal{I}^{(k)} \mid k = 1, \ldots, K\}$. If not, recursively check in each of the two intervals on either side of the median for a point common to all $\mathcal{I}^{(k)},\ k = 1, \ldots K$.

Since each of the two smaller subproblems contains approximately half of the total number of endpoints in $\mathcal{I}^{(k)},\ k = 1, \ldots, K$, the overall running time of the procedure is $O(K \log K)$. 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 $D^j,\ j = 1, \ldots, J$:

- 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 V is not reachable from C.

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.

Operator	Aggregate				
	min(X.A)	max(X.A)	count()	sum(X.A)	avg(X.A)
$> v$	No	Yes	Yes	Yes	No
$= v$	No	No	No	No	No
$< v$	Yes	No	No	No	No

Table 1: Monotonic characterization of constraints

6.2 Query Monotonicity

We have looked at how to use the GBP method for determining the structural monotonicity of a query Q on a given view V. A closely related question is whether or not Q is monotonic *for any* view V. This notion of monotonicity was introduced in [NLHP98], where Q was defined to be monotonic (anti-monotonic in their context) if

$$Q(X) \ \texttt{FALSE} \implies Q(X') \ \texttt{FALSE}$$

for any cubes $X' \subseteq X \subseteq \mathcal{D}$ in any database $\mathcal{D}$. Table 1 shows some examples of monotonic and non-monotonic queries discussed in [NLHP98]. The GBP method can easily be extended to determine the monotonicity of an arbitrary cube query.

Consider a cube query $Q(\cdot)$ and the grid G for that query. Define a cell $\bar{C} \in G$ given by the intervals $[\bar{V}^{(j)}],\ j = 1, \ldots, J,\ \bar{I}_{\texttt{COUNT}}$ and $\bar{I}^{(k)}_{\texttt{MIN}},\ \bar{I}^{(k)}_{\texttt{MAX}},\ \bar{I}^{(k)}_{\texttt{SUM}},\ \bar{I}^{(k)}_{\texttt{AVG}},\ \ k = 1, \ldots, K$, to be *reachable* from a cell $C \in G$ given by the intervals $[V^{(j)}],\ j = 1, \ldots, J,\ I_{\texttt{COUNT}}$ and $I^{(k)}_{\texttt{MIN}},\ I^{(k)}_{\texttt{MAX}}, I^{(k)}_{\texttt{SUM}},\ I^{(k)}_{\texttt{AVG}},\ \ k = 1, \ldots, K$, if there is a set

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

of $N + \Delta \in \bar{I}_{\texttt{COUNT}}$ records such that

- $N \in I_{\texttt{COUNT}}$ and $\Delta \geq 1$
- $\{X_1 \ldots, X_N, X_{N+1}, \ldots, X_{N+\Delta}\}$ maps into $\bar{C}$
- $\{X_1, \ldots, X_N\}$ maps into C.

Q would be monotonic if the `FALSE` cells in its grid are not reachable from any of the `TRUE` cells. Thus, to prove the monotonicity we can construct an algorithm which takes every pair $(C, \bar{C})$ of satisfiable `FALSE` and `TRUE` cells in the grid and check whether the `FALSE` cell $\bar{C}$ is reachable from the `TRUE` cell C. If the answer is positive then the query is not monotonic. As the following theorem shows, the reachability of $\bar{C}$ from C can be tested in polynomial time in the bit model of computation.

Theorem 6.5 *Given two (satisfiable) cells C and $\bar{C}$ defined by rational intervals on J dimensional and the measures of K distinct attributes, the reachability of $\bar{C}$ from C can be checked in polynomial time.*

The proof is similar to the proof of Theorem 6.4 and we only briefly sketch it here. The proof starts with the case $K = 1$, i.e. when the cells C and $\bar{C}$ are defined by some rational intervals $I_{\texttt{COUNT}}$, $I_{\texttt{MIN}}$, $I_{\texttt{MAX}}$, $I_{\texttt{SUM}}$, $I_{\texttt{AVG}}$ and $\bar{I}_{\texttt{COUNT}}$, $\bar{I}_{\texttt{MIN}}$, $\bar{I}_{\texttt{MAX}}$, $\bar{I}_{\texttt{SUM}}$, $\bar{I}_{\texttt{AVG}}$, respectively, on the measures of a single attribute. Analogously to the proof of Theorem 6.4, it can be shown that the reachability of $\bar{C}$ from C is equivalent to a constant number of systems of linear inequalities in N and Δ. Unlike Theorem 6.4, however, we cannot conclude that sum $N + \Delta$ is constant and consequently, each of the above systems defines a 2-dimensional rational polytope in the plane (N, Δ). It remains to check whether any of these polytops contains an integral point (N, Δ). This can be done in time polynomial in the length of the binary encoding of the endpoints of $I_{\texttt{COUNT}}$, $\bar{I}_{\texttt{COUNT}}$, $I_{\texttt{MIN}}$, $\bar{I}_{\texttt{MIN}}$, $I_{\texttt{MAX}}$, $\bar{I}_{\texttt{MAX}}$, $I_{\texttt{SUM}}$, $\bar{I}_{\texttt{SUM}}$, $I_{\texttt{AVG}}$ $\bar{I}_{\texttt{AVG}}$ due to the fact that integer programming in fixed dimension is polynomial-time solvable, see e.g. [Sch86].

Next we turn to the case when the cell intervals are defined by interval constraints on the measures of $K \geq 2$ distinct attributes. Here again, 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 cardinalities N and Δ should be identical for all attributes. So, to check the reachability of $\bar{C}$ from C, we can separately construct the constant number of rational polytops on N and Δ for each attribute A^k, $k = 1, \ldots, K$, and then check whether the intersection of some polytops for each of the K attributes contains an integral point (N, Δ). The latter problem can be solved in polynomial time as follows. Each linear inequality in each of the above polytops defines a line in the plane (N, Δ). There is a total of $O(K)$ such lines, and their arrangement in the plane can be constructed in $O(K^2)$ time by using standard methods from computational geometry [Hal97]. Pick a sample point in each polyhedral cell in the arrangement and test whether the point, and hence the cell, belongs to at least one polytope for each attribute $A^1, \ldots, A^K$. If the answer is positive, use 2-dimensional integer programming to test whether the cell contains an integral point. Since there are $O(K^2)$ cells, and the test for each of them runs in polynomial time in the bit model of computation, the overall running time of the algorithm is also polynomial.

Finally, the case for $J \geq 1$ is similar to the view reachability proof.

Theorem 6.5, although interesting from theoretical point of view, is not very useful practically because of the high running time of the algorithm and the fact that very few queries can be expected to be monotonic for all possible views.

6.3 Generating the cubes

For generating the cubes, algorithms employed for generating frequent sets can be used. The only difference is that pruning for cubes would be based on *GBP* instead of support. Fig. 2 shows an outline of cube query evaluation based on Apriori. It starts from an initial seed set and then iteratively generates candidate sets of larger conjuncts (smaller cubes). The initial seed set is typically the whole database except under situations when the query asks for subcubes for a specific cube, in which case the initial seed set can be set to the common super cube.

The first candidate set C_1 is computed by constructing cubes of conjunct length 1. A database scan is made to compute the measures for the cube. After the scan every potential conjunct is paired with a $Measure$ of Q and the resulting $< cube, Measure, Value >$ tuple is evaluated on the query. If the tuple satisfies the query then it is added to the answer, otherwise another test is made to check whether it can be pruned using GBP. The set of cubes that remain after pruning make up the *significant set* S_1 for the current iteration. This set is used to construct the candidate set C_2, for the next iteration, similar to as in Apriori. The algorithm continues by performing a scan on the new 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=ϕ;
while (C_i not empty) {
Conduct db scan on C_i;
Compute the tuples <cube, Measure, Value > 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

Example 6.2 *As an illustration of how our approach for cube queries work consider the query asking for the cubes satisfying the following expression on the amount of milk sold in the cube.*

```
Q12: COUNT(*)>=1000 and AVG(salesMilk)>=20 and
     AVG(salesMilk)<50 and  MAX(salesMilk)>75 and SUM(salesMilk)<50K
```

The grid for this query consists of four axes: `COUNT()`, `MAX(salesMilk)`, `AVG(salesMilk)`, *and* `SUM(salesMilk)`. *Figure 3 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 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.

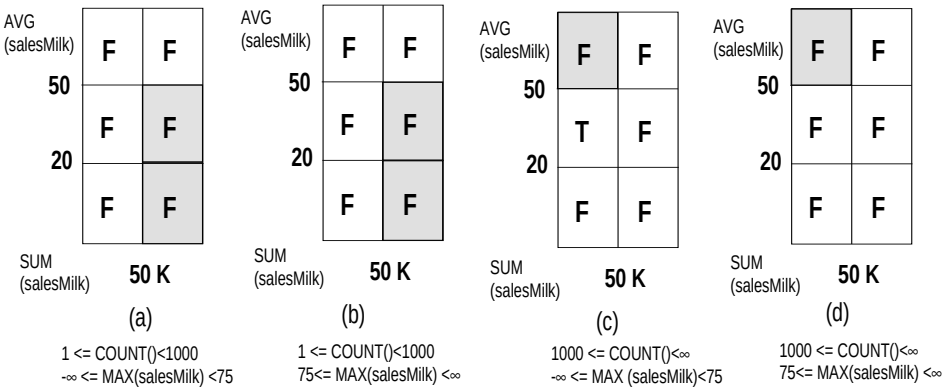

Figure 3: Example grid

Suppose that for a certain iteration we have a cube with the following view V:.

$$\begin{aligned}
&\mathcal{C} = 1200\\
&\texttt{AVG}(\texttt{salesMilk}) = 52\\
&\texttt{MAX}(\texttt{salesMilk}) = 80\\
&\texttt{MIN}(\texttt{salesMilk}) = 30\\
&\texttt{SUM}(\texttt{salesMilk}) = 60000
\end{aligned}$$

The query is `FALSE` *at this view. However, to determine whether to prune this cube, we have to show that no cube with the above view V in any database can have have a subset 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 V of the cube was:.

$$\begin{aligned}
&\mathcal{C} = 1200\\
&\texttt{AVG}(\texttt{salesMilk}) = 52\\
&\texttt{MAX}(\texttt{salesMilk}) = 80\\
&\texttt{MIN}(\texttt{salesMilk}) = 30\\
&\texttt{SUM}(\texttt{salesMilk}) = 64000
\end{aligned}$$

Then, there doesn't exist any interval for N for which V can be reached from the the only true cell of the grid. Thus, this cube can be pruned.

One of the optimizations that can be made in the evaluation is to amortize some operations involved in performing the pruning. The observation is that if there is a a view V reachable from another view $\bar{V}$ and $\bar{V}$ is reachable from a cell C then V is also reachable from cell C (*transitivity of view reachability*). Now, suppose we have a query Q, and a cube X with view V on the query Q and it is tested that V is not reachable from a `TRUE` cell C. Then from transitivity, it also holds that any view belonging to a subset of X is also not reachable from C. Thus, in this case when performing GBP tests on the views for subcubes of X we can save on some costs by not checking their reachability from C.

7 Evaluating Cubegrade Queries

As promised, we evaluate cubegrade queries by reducing them to the cube queries whose evaluation we have already described.

Before proceeding, we define some basic query transformations which will make it possible for us to reduce cubegrade query evaluation to repetitive cube query evaluations. We start by first generating all possible candidates for source cubes of the cubegrades by transforming the original query Q into another query $Q[source]$. This is done by making all descriptor conditions which involve *target cube* true and making all *Delta Value* conditions true. The resulting query $Q[source]$ is a cube query and can be evaluated using the methods described in the section on CubeQL evaluation.

Now, given a specific source cube C, define $Q[C]$ as the query which results from Q by substituting into the "where" clause all the values of all the *Measures*, evaluating the *source cube* descriptors on C, as well as by performing the step which we call "Delta elimination" which replaces all the relative *Delta Values* (expressed as fractions) by the regular less than, greater than conditions. This is possible, since all the measures in *Measures* have known values for X, thus the *Delta Values* can now be expressed as conditions on *Values* of the *target cube*. For example if `AVG(Salary)` on X is `40K` and the condition on the `DeltaAVG(Salary)` is of the form `DeltaAVG(Salary)` $>$ `0.10`, this can be translated now to `AVG(Salary)` $>$ `44K` where `AVG(Salary)` refers now to the *target cube*.

Finally, we will have to transform all the *join conditions* in Q into the *target cube* conditions since the source cube is already specified. Notice, thus, that $Q[X]$ is the cube query specifying only target cubes.

Below, we present the general algorithm:

```
INPUT: A cubegrade query Q

OUTPUT: The set of cubegrades which satisfy Q

Evaluate Q[source];
For each cube S in Q[source]
   evaluate Q[S];
```

```
For each T in Q[S]
 Form the cubegrade
 <S, T, Measures, Values, Delta-Values>
 where Delta Values have to be calculated
 as ratios of the Measures evaluated on the target and
 on the source cubes respectively
```

Notice that the above algorithm can handle arbitrary cubegrade queries, even if the cubegrade is not a specialization, generalization or a mutation, in other words, target and source cubes are not "closely" related.

Example 7.1 *Consider the evaluation of the cubegrade query* **Q6** *given in section 5.2. We start by first transforming the query to a pure cube query, Q[source] to generate the set potential candidates for the source cube.*

```
GET  CUBE
FROM customer
WHERE CUBE MATCHES (areaType=* or age=* or income=* or
                    salesBread=* or  salesMilk=* or
                    salesCookies=* or salesSoda=*)
      AND MEASURES = {COUNT(*), AVG(salesCereal)}
      AND COUNT(*)>1000 and AVG(salesCereal)<20
```

Next, for a source cube S that satisfies Q[source], a query on the target cube is generated by performing source descriptor substitution, " Delta elimination" and finally transforming the join conditions into conditions for the target cube. For example, consider the case of a candidate source cube of buyers who make between 60-80K per year and spend more than $100 on soda with `COUNT(*)=1500` *and* `AVG(salesCereal)=18`*. The query generated for this cube would be:.*

```
GET  CUBE
FROM customer
WHERE CUBE MATCHES (income in [60K,80K] and salesSoda in [100,MAX]
                    and (areaType=* or age=*
                    or salesBread=* or  salesMilk=*
                    or salesCookies=* or salesSoda=*))
      AND MEASURES = {COUNT(*), AVG(salesCereal)}
      AND AVG(salesCereal)>27 and COUNT(*) >750
```

This CubeQL query is then evaluated to generate the target cubes and hence finally a set of cubegrades for the answer of **Q6**. *For example if a cube generated for the source cube considered earlier, is the cube of buyers who make between 60-80K per year, spend more than $100 on soda and spend more than $20 on cookies with* `COUNT(*)=1125` *and* `AVG(salesCereal)=32` *then the answer set contains the following cubegrade.*

(income=[60K,80K], salesSoda>$100) − >
(income=[60K,80K], salesSoda>$100, salesCookies>$20)

```
[COUNT(*), COUNT(*) = 1500, DeltaCOUNT(*)=75%]
[AVG(salesCereal), AVG(salesCereal)=18, DeltaAVG(salesCereal)=180%]
```

8 Discussion and Conclusion

In this paper we have introduced cubegrades, showed how to generate them using efficient pruning techniques with the GBP as the main foundation, and finally defined a query language to query and retrieve (previously stored) cubegrades and cubes.

Cubegrades 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.

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 ¿from any of the `FALSE` cells. If V is not reachable from 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 CubeQL and CubegradeQL. Generalizing CubeQL and CubegradeQL to allow nesting subqueries, introducing "join operation" to the Cube querying etc.

  It would be interesting to see if the methods presented here can practically be extended to handle more general query conditions including where addition and multiplication between the aggregates are allowed.

References

[AIS93] 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.

[AMS+96] 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.

[Bas97] S. Basu. An improved algorithm for quantifier elimination over real closed fields. In *Foundations of Computer Science (FOCS 1997)*, 1997.

[Hal97] D. Halperin. Arrangements. In J. Goodman and J. O'Rourke, editors, *Handbook of discrete and computational geometry*, pages 389 – 412, 1997.

[HRS93] 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.

[IV99] T. Imielinski and A. Virmani. M-sql: A query language for database mining. In *Data Mining and Knowledge Discovery*, 1999.

[IVA99] T. Imielinski, A. Virmani, and A. Abdulghani. Dmajor-application programming interface for database mining. In *Data Mining and Knowledge Discovery*, 1999.

[LNHP99] 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, Zurich, Switzerland, Sept 1999.

[NLHP98] 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, Zurich, Switzerland, Sept 1998.

[RSSS98] K. A. Ross, D. Srivastava, P. J. Stuckey, and S. Sudarshan. Foundations of aggregation constraints. *Theoretical Computer Science*, 193:149 – 179, Feburary 1998.

[SAM98] 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.

[Sar99] 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.

[Sch86] A. Schrijver. *Theory of Linear and Integer Programming*. Wiley-Interscience series, 1986.