浓缩数据立方中约束立方梯度的挖掘*

冯玉才, 刘玉葆+, 冯剑琳

(华中科技大学 计算机科学技术学院,湖北 武汉 430074)

Mining Constrained Cube Gradient for the Condensed Data Cube

FENG Yu-Cai, LIU Yu-Bao+, FENG Jian-Lin

(College of Computer Science, Huazhong University of Science and Technology, Wuhan 430074, China)

+ Corresponding author: Phn: 86-27-87522500 ext 8004, E-mail: yubliu@263.net

http://www.dm2.com.cn

Received 2002-06-18; Accepted 2002-12-04

Feng YC, Liu YB, Feng JL. Mining constrained cube gradient for the condensed data cube. *Journal of Software*, 2003,14(10):1706~1716.

http://www.jos.org.cn/1000-9825/14/1706.htm

Abstract: Constrained cube gradient mining is an important mining task and its goal is to extract the pairs of gradient-probe cell satisfy the gradient constraint from a data cube. However, previous work are explored for a general data cube. In this paper, the problem of the mining constrained cube gradient for a condensed cube is studied. An algorithm named as eLiveSet for the problem is developed through the extension of the existing efficient mining algorithm LiveSet-driven. The experimental results show that the algorithm is more effective than the existing algorithm on the performance of mining constrained cube gradient.

Key words: data cube; cube gradient; condensed cube; cube computation; association rule

摘 要: 约束立方梯度挖掘是一项重要的挖掘任务,其主要目的是从数据立方中挖掘出满足梯度约束的梯度-探测元组对.然而,现有的研究都是基于一般数据立方的.研究了浓缩数据立方中约束数据立方梯度的挖掘问题.通过扩展LiveSet驱动算法,提出了一个eLiveSet算法.测试表明,该算法在立方梯度挖掘效率上比现有算法要高.

关键词: 数据立方;立方梯度;浓缩数据立方;数据立方计算;关联规则

中图法分类号: TP311 **文献标识码**: A

1 Introduction

Recently, there have been growing interests in multidimensional analysis of relational databases, transactional

* Supported by the e-Government Project of the Ministry of Science and Technology of China under Grant No.2001BA110B01 (国家科技部电子政务项目)

FENG Yu-Cai is a professor and doctoral supervisor at the College of Computer Science, the Huazhong University of Science and Technology. His research areas are the database and the information system. **LIU Yu-Bao** is a Ph.D. candidate at the College at the Computer Science, the Huazhong University of Science and Technology. His research interests are the data mining and the data warehouse. **FENG Jian-Lin** received his Ph.D. degree from the College of Computer Science, the Huazhong University of Science and Technology. His research interests are the database and the data warehouse.

databases, and data warehouses[1,2]. Most of such analyses involve data cube-based summary or transaction-based association analysis. However, many interesting applications may need to analyze the changes of measures in multidimensional space. For example, one may want to ask what the changes of the average house price in the Vancouver area in year 2000 are compared against 1999, and the answer could be "the average price for those sold to professionals in the West End went down by 20%, while those sold to business people in Metrotown went up by 10%, etc." Expressions such as "professionals in the West End" correspond to cells in data cube and describe sectors of the business modeled by the data cube. The problem of mining changes of measures in a multidimensional space was first proposed by Imielinski, *et al*. as a cubegrade problem[3], which can be viewed as a generalization of association rules[4] and data cubes[5]. An example of a typical association rule states that, 23% of supermarket transactions that buy bread and butter also buy cereal (that percentage is called confidence) and that 0.6% of all transactions buy bread and butter (this is called support). This rule can also be interpreted as saying that the count of transactions buying bread and butter drops to 23% of the original when the restricted to (drill-down) the transactions buying bread, butter and cereal. This interpretation leads to a much more general view of association rules, when support (count) is replaced by an arbitrary aggregate measure. This leads to the concept of cubegrade that studies how changes of measures (aggregates) of interest are associated with the changes in the underlying characteristics of sectors, where the changes in sector characteristics are expressed in terms of dimensions of the cube and are limited to specialization (drill-down), generalization (roll-up), and mutation (a change in one of the cube's dimensions). 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. Typically, cubegrades can 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: Rise by 20% for customers younger than 40 and drops by 5% among customers older than 40.

Cubegrades can also support sophisticated "what if" analysis etc. and be useful in marketing, sales analysis, and other typical data mining applications in business[3]. The constrained cube gradient mining[6] represents a confined but interesting version of the cubegrade problem. The goal of constrained cube gradient mining is to extract the pairs of gradient-probe cell characteristics associated with big changes in measure from a data cube.

However, the previous studies are mainly explored for a general data cube. It is well known that the size of a cube is always huge and the cost of cube computation is also expensive. So the process of constrained cube gradient mining would be expensive. In this paper, we consider the problem of mining constrained cube gradient for a condensed data cube. The condensed cube[7] is a novel and efficient data organization approach. The basic spirit of the condensed cube is to condense a number of cells of a cube into a cell, that is, a single tuple. Thus a condensed cube can reduce dramatically the size of a data cube itself and hence the cube computation time. Since the existing mining algorithms are explored for a general cube, it is not applicable to our problem where the probe cells and gradient cells are stored in a condensed format. A new algorithm named as eLiveSet is developed for our problem by the extension of the existing LiveSet-driven algorithm (short for LiveSet)[6]. There are some related works regarding our problem including the discovery-driven exploration of cube[8] and the cube computation[9,10].

This paper is organized as follows: an overview of condensed cube is introduced in section 2. The problem definition is given in section 3. The mining algorithm for our problem is discussed in section 4. The experimental studies are given in section 5. The conclusions are in section 6.

2 An Overview of Condensed Cube

The CUBE BY operator[5] is a multidimensional extension of relational GROUP BY operator. While the semantics of the CUBE BY operator is to partition a relation into groups based on the values of the attributes specified in the CUBE BY operator and then apply aggregations functions to each of such groups, the CUBE BY operator computes GROUP BY corresponding to all possible combinations of attributes in the CUBE BY operator. In general, a CUBE BY operator on n attributes computes 2^n GROUP BYs, or cuboids. The grouping attributes are called dimensions and the attributes that are aggregated are called measures. A tuple with dimension attributes and measure attributes in a data cube is called a cell. While the aggregation is applied to groups of relation table tuples obtained by partitioning the relation table on the cuboid attributes, there exist such partitions that contain only one tuple that is named as a single tuple. Assumed that the relation table R with 3-dimension attributes (A,B,C) and one measure attributes M in Fig.1 is the running example relation table, the aggregate function is SUM function. When the relation table R is partitioned on dimension A (i.e. cuboid A), the partition in which the dimension value of A is equal to 0 contains a single tuple (0,1,1,50). The corresponding cell aggregated from this tuple is (0,*,*,50), where "*" denotes the value "all", i.e., aggregated to the highest level on this dimension. Given a set of dimension attributes $SD \subset A$[⇨], if r is the only tuple in its partition when the relation table is partitioned on SD, we say tuple r is a single tuple on SD, and SD is called the single dimensions of r. For example, $\{A\}$ is the single dimension of the single tuple (0,1,1,50). If a relation table tuple r is a single tuple on SD, then r is also a single tuple on any superset of r. For example, the tuple (0,1,1,50) is also a single tuple on the any superset of $\{A\}$, i.e., $\{AB\}$, $\{AC\}$ and $\{ABC\}$[⇨⇨]. We associate each single tuple a set of SDs, SDSET to represent the fact that a tuple can be a single tuple on different dimensions. The single tuple r and its SDSET can express many cells aggregated from this single tuple. For example, the single tuple (0,1,1,50) and its SDSET=$\{\{A\},\{AB\},\{AC\},\{ABC\}\}$ can express the four cells: (0,*,*,50), (0,1,*,50), (0,*,1,50) and (0,1,1,50) and the four cells can be viewed as being condensed into the single tuple (0,1,1,50).

Given a single tuple $r(r(a_1),...,r(a_n))$ ($r(a_k)$ denotes the value of dimension a_k ($1 \le k \le n$)) and its single dimensions $SD=\{a_i,...,a_j\}$ ($1 \le i \le j \le n$) or its SDSET, the complete set of cells condensed by a single tuple r is denoted as ExpandSet(r) and it can be computed by the following Expand operator:

- $Expand(r,SD)=r'$ such that $m(r')=m(r)$ and $r'(a_k)=r(a_k)$ for $a_k \in SD$ and $r'(a_k)=*$ for $a_k \notin SD$, where $m(r)$ denotes the measure of r and.
- $Expand(r,SDSET)=\{r' | r'=Expand(r,SD_i), SD_i \in SDSET\}$.

For example, given a single tuple r=(2,3,1,60) and its single dimensions $SD=\{A\}$, $ExpandSet(r)$={(2,*,*,60), (2,3,*,60), (2,*,1,60), (2,3,1,60)}. In general, for the given single tuple $r(r(a_1),\ldots,r(a_n))$ and its $SD=\{a_i,\ldots,a_j\}$ ($1 \le i \le j \le n$), there are the number of 2^{n-j} cells contained in $ExpandSet(r)$, in other words, 2^{n-j} cells are condensed into the single tuple r. Note that all the cells in $ExpandSet(r)$ have the same aggregation value $m(r)$ since all of them are only aggregated from the same single tuple.

The value of dimension a_k($1 \le k \le n$) of any cell r' in $ExpandSet(r)$ can also be defined as the following Expand principles:

(1) $r'(a_k)=r(a_k)$, if $a_k \in SD$;

(2) $r'(a_k)=r(a_k)$ or $r'(a_k)=*$, if $k>j$;

[⇨] Notice that we require $SD \neq A$, where A denotes the all dimensions of a cube.

[⇨⇨] In this paper, whenever there is no confusion, we use the concatenation of dimension names to represent the set consisting of those dimensions. For example, the $\{AB\}$ is a short of $\{A,B\}$.

(3) $r'(a_k)$=*, if $a_k \notin SD$ and $1 \le k < j$.

It is clear that the definition of Expand principles is equivalent to the definition of Expand operator. For example, suppose r=(A=1, B=2, C=3, D=4) is a single tuple and its SD={AC}, each dimension value of the cells in *ExpandSet*(r) are as follows: A=1, B=*, C=3, and D=4 or D=*, i.e., *ExpandSet*(r)={(A=1, B=*, C=3, D=4), (A=1, B=*, C=3, D=*)}. According to the Expand principles, we have also two important properties on a single tuple r:

(1) The non-* dimension values of the cells in *ExpandSet*(r) are all derived from the single tuple r and equal to the corresponding dimensions values of r.

(2) All of cells in *ExpandSet*(r) share the same non-* dimension values on SD and the cells that have the same non-* dimension values on SD are contained in the set *ExpandSet*(r).

In a condensed cube, we only need to physically store the single tuple together with an extra field to store the single dimensions information of the single tuple. The cells can be expressed by the single tuple are not stored physically. When needed, these cells can be generated through the expand operator of the single tuple. The SD fields of these non-single tuples are equal to ∅ in a condensed cube. These non-single tuples can be viewed as the general cells in a general data cube since they don't condense any cells. The illustration of the condensed cube and the general cube of R is shown in Fig.1. A condensed cube can be computed through the BU-BST algorithm[7], which is basically a modified version of the original BUC algorithm[9].

TID	*A*	*B*	*C*	*M*
1	0	1	1	50
2	1	1	1	100
3	2	3	1	60
4	4	5	1	70
5	4	5	2	80

(a) The relation table R

TID	*A*	*B*	*C*	*M*	*SD*
1	0	1	1	50	{A}
2	4	*	*	150	∅

(b) The condensed cube of R

TID	*A*	*B*	*C*	*M*
1	0	*	*	50
2	0	1	*	50
3	0	*	1	50
4	0	1	1	50

(c) The general cube of R

Fig.1

3 The Problem of Mining Constrained Cube Gradient

3.1 The problem definition

Given two distinct cells c_1 and c_2 of a data cube D of a given relation table R with n dimensions, c_1 is an ancestor of c_2 and c_2 is a descendant of c_1 iff on every dimension attribute, either c_1 and c_2 share the same value, or c_1 has value '*', where '*' indicates 'all'; c_1 is a sibling of c_2, and vice versa, iff c_1 and c_2 have identical values in all dimensions except one dimension in which neither has value *. The single tuple has not descendant cell because each dimension has the non-* value, i.e., the specific value. For simplicity, we sometimes say c_1 is similar to c_2 if c_1 is a descendant, an ancestor or a sibling of c_2.

A significance constraint C_{sig} is usually defined as conditions on measure attributes. The significance constraint is assumed to be anti-monotonic in this paper. Anti-montonicity is very useful for the pruning of cells in the cube computation algorithm. It states that if a cell c does not satisfy an anti-monotonic constraint, none of c's descendants can do so. Some methods for deriving weaker anti-monotonic constraints from non-anti-monotonic constraints are discussed in the Ref.[10]. A probe constraint C_{prb} is usually defined as conditions on dimension attributes and is used to select a set of user-desired cells. A cell c is significant iff $C_{sig}(c)$=true, and a cell c is a probe cell iff c is significant and $C_{prb}(c)$=true. The complete set of probe cells is denoted as P. The set of significant cells that may have gradient relationship with a set of probe cells, P, are called the gradient cells of P. The gradient constraint has the form $C_{grad}(c_g, c_p) \equiv (g(c_g,c_p)\ \theta\ v)$, where θ is in {≤,≥,<,>}, v is a constant value, and g is a gradient function. In this paper, the gradient constraint form is defined as follows: "$m(c_g)-m(c_p)\ \theta\ v$", where $m(c)$ is

a measure value for a cell c. A gradient cell c_g is interesting with respect to a probe cell $c_p \in P$ iff c_g is significant, c_g and c_p satisfy similar relationship and $C_{grad}(c_g,c_p)$=true. Formally, given a relation table R, a significant constraint C_{sig}, a probe constraint C_{prb} and a gradient constraint $C_{grad}(c_g,c_p)$, the constrained cube gradient problem is to find all the interesting gradient-probe pairs (c_g,c_p) such that $C_{grad}(c_g,c_p)$=true.

Example 1. Supposed $C_{prb} \equiv (A=4,B=*,C=*)$, $C_{sig} \equiv (M>50)$, $C_{grad}(c_g,c_p) \equiv (m(c_g)-m(c_p))>0$, the cells c_g=(4,*,*, 150), c_{p1}=(4,5,1,70) and c_{p2}=(4,5,2,80) are significant. c_g is an ancestor of c_{p1} and c_{p2}, c_{p1} is a sibling of c_{p2} and (c_g, c_{p1}), (c_g,c_{p2}) are interesting gradient-probe cell pairs.

For a single tuple c, if c is significant, all of the cells condensed into c are significant because they have the same measure to c. We say c satisfy the probe constraint C_{prb}, if each dimension of c has the dimension values that satisfy the corresponding dimension value constraints of C_{prb}.

Given a relation table R, a significant constraint C_{sig}, a probe constraint C_{prb} and a gradient constraint C_{grad}, the goal of our problem is to find all interesting gradient-probe cell pairs (e_g,e_p) such that $C_{grad}(e_g,e_p)$=true, where $e_g \in ExpandSet(c_g)$ and $e_p \in ExpandSet(c_p)$[⇨].

3.2 The LiveSet algorithm

The main framework of LiveSet algorithm is as follows: (1) Apply a cube computation algorithm, such as H-cubing[10] or BUC, to compute the set of probe cells P from the relation table R using both the significance and probe constraints. (2) Produce the interesting gradient-probe cell pairs, that is, to determine which gradient cell should be associated with which probe cells.

The live set method is used to in the second step. In general, the live set of a gradient cell c_g, which is denoted as $LiveSet(c_g)$, is the set of probe cells c_p such that it is possible that (c_g',c_p) is an interesting gradient-probe pair, for some descendant cell c_g' of c_g. As generating the interesting gradient-probe cell pairs, we only need to compare the gradient cell, which is generated by a cube computation algorithm, with its related probe cells (i.e. live set)[⇨] but the complete set of probe cells P. Thus how to derive the live set is a key problem for the LiveSet-style algorithms. In LiveSet algorithm, the matching analysis method is used to derive the LiveSet of the gradient cells. Let $c_p=(d_{p1}, d_{p2},...,d_{pm})$ be a probe cell and $c_g=(d_{g1},d_{g2},...,d_{gm})$ be a gradient cell. The number of solid-mismatches between c_p and c_g is the number of dimensions in which both values are not * but are not matched, i.e., of different values. The number of *-mismatches between c_p and c_g is the number of dimensions in which c_p is * but c_g is not. It is noticed that the notion of *-mismatches is not symmetric, i.e., if c_g has * value on a dimension but c_p has a non-* value on the same dimension, this is not considered a *-mismatch. A probe cell c_p is matchable with a gradient cell c_g if either c_g and c_p have no solid-mismatch, or they have exact one solid-mismatches but no *-mismatch.

Example 2. In the example1, the probe cell c_p=(4,5,1,70) is matchable with c_{g1}=(4,*,*,150). However, c_p is not matchable with c_{g2}=(1,1,1,100) since there are two solid-mismatches between c_p and c_{g2}.

Property 1. Assume that c_{g1} and c_{g2} are gradient cells and c_{g2} is a descendant of c_{g1}. Then $LiveSet(c_{g2}) \subseteq LiveSet(c_{g1})$.

This property ensures that we can produce the live set of a descendant cell from that of the ancestor cell.

Property 2. Let c_p is a probe cell and c_g is a gradient cell. If c_p is matchabel with c_g then $c_p \in LiveSet(c_g)$ otherwise $c_p \notin LiveSet(c_g)$.

This property shows that the matching analysis method can be used to derive the live set.

⇨ As the gradient cell c_g (or the probe cell c_p) is a non-single tuple, the set $ExpandSet(c_g)$ (or $ExpandSet(c_p)$) only contains one tuple, i.e., the c_g (or c_p) itself and then e_g (or e_p) is equal to c_g (or c_g).

⇨ The initialized gradient cell c_g is set to (*,*…,*) and the $LiveSet(c_g)$ is set to the complete probe cell set P.

Example 3. Suppose the set of probe cells P has four cells with three dimensions and one measure: P={(0,*,*,50), (0,1,*,50), (0,*,1,50), (0,1,1,50)}, the gradient cell c_g=(4,*,*,150) and $LiveSet(c_g)=P$. c_g'=(4,5,*,150) is a descendant of c_g. According to property1 and property2, the $LiveSet(c_g')=\varnothing$, i.e., it is the result of pruning the cells (0,*,*,50), (0,1,1,50), (0,*,1,50) and (0,1,*,50) from $LiveSet(c_g)$.

4 The Mining Algorithm

In this section, the eLiveSet algorithm for our problem is discussed. In particular, two key techniques of eLiveSet algorithm are introduced. The first technique is on how to derive the live set that contains the single tuples. The other technique is the partial expansion technique of single tuples. Before the end of this section, the complete description of eLiveSet algorithm is given.

4.1 Derive the live set

Intuitively, the live set concept describes the related probe cells of a gradient cell. The basic cell of a single tuple represents a special cell that is an ancestor of the other cells in ExpandSet. The potential concept defines the conditions that a gradient cell can be processed further in the depth-first order.

Definition 1. Given a gradient cell c_g, the live set of c_g is defined as following:

(1) The gradient cell c_g is a non-single tuple. For a probe cell c_p that is a single tuple, we say $c_p \in LiveSet(c_g)$ if there exists a certain cell $c_p' \in ExpandSet(c_p)$ and (c_g', c_p') is a possibly interesting gradient-probe cell pair for some descendant cells c_g' of c_g. On the other hand, if c_p is a non-single tuple, we say $c_p \in LiveSet(c_g)$ if (c_g', c_p) is a possibly interesting gradient-probe cell pair for some descendant cells c_g' of c_g.

(2) The gradient cell c_g is a single tuple. Due to c_g likely condenses many gradient cells, we define $LiveSet(c_g)=\cup LiveSet(c_{gi})$ where $c_{gi} \in ExpandSet(c_g)$ and $1 \le i \le |ExpandSet(c_g)|$ and $|ExpandSet(c_g)|$ denotes the cardinal number of the set *ExpandSet*.

Definition 2. Given a single tuple r, the cell in $ExpandSet(r)$ that is obtained by taking the * value for each dimension of r except the dimensions in SD of r, is called the basic cell of the single tuple r and is denoted as c_b. It is clear that $c_b \in ExpandSet(r)$. For example, assume that c_g=(A=1,B=2,C=3,D=4) is a single tuple and its SD={B,C}, then the basic cell of c_g, c_b=(*,2,3,*). It is also clear that c_b is an ancestor of the other cells in $ExpandSet(r)$ because the other cells is obtained by taking the non-* value for the dimensions that is not in SD from c_b.

Definition 3. A gradient cell c_g is potential to expand higher dimension if it satisfy the following conditions: (1) c_g is not a single tuple; (2) $C_{sig}(c_g)$=true; (3) $C_{grad}(c_g,c_p)$=true, for some probe cell c_p in $LiveSet(c_g)$.

Lemma 1. Given a gradient cell c_g and a probe cell c_p that is a single tuple. If c_p is not matchable with c_g, then for any cell $c \in ExpandSet(c_p)$, c is not similar to c_g' that is a descendant of c_g.

Proof. Suppose c_p is not matchable with c_g. Then c_p and c_g have at least two solid-mismatches (there are not *-mismatches between c_p and c_g because c_p is a single tuple and has no * dimension value). Without loss of generalization, we assume that c_p has different non-* values with c_g on dimensions $(a_1,\ldots,a_i)$ ($i \ge 2$). Both c_g' and c_g have the same non-* values on dimensions $(a_1,\ldots,a_i)$ because c_g' is a descendant of c_g. Thus, c_p also has different non-* values with c_g' on dimensions $(a_1,\ldots,a_i)$ ($i \ge 2$). Two cases raise here, i.e., $\{a_1,\ldots,a_i\} \cap SD \ne \varnothing$ and $\{a_1,\ldots,a_i\} \cap SD = \varnothing$, where SD is the single dimensions of the single tuple c_p.

(1) Suppose $\{a_1,\ldots,a_i\} \cap SD \ne \varnothing$. Without loss of generalization, we suppose $\{a_1,\ldots,a_i\} \cap SD = \{a_1,\ldots,a_j\}$ ($j \ge 1$). From the property of the single tuple, it is known that all of cells in $ExpandSet(c_p)$ share the same non-* values on dimensions $(a_1,\ldots,a_j)$. Thus, c has different non-* values with c_g' on dimensions $(a_1,\ldots,a_j)$. Suppose $j>1$, i.e., c and c_g' have at least two different non-* values on dimensions $(a_1,\ldots,a_j)$, c is not similar to c_g'. Suppose $j=1$. Since c_p and c_g' have at least two different non-* dimension values, there is at least another dimension a_d, where $1 \le d \le i$, in

which c_p and c_g' have different non-* value. The value of dimension a_d of any cell $c \in ExpandSet(c_p)$ is either equal to a non-* value $c_p(a_d)$, where $c_p(a_d)$ denotes the value of c_p on dimension a_d, or equal to *. If equal to a non-* value, c and c_g' have different non-* dimension value on dimension a_k and a_d. c is not similar to c_g'. If equal to * value, c and c_g' have different non-* dimension value on a_j except the dimension a_d in which c has * but c_g' has non-*. Thus, c is also not similar to c_g'.

(2) Suppose $\{a_1,\ldots,a_i\} \cap SD = \varnothing$ ($i \geq 2$). According to the values of dimensions $(a_1,\ldots,a_i)$, the cells in $ExpandSet(c_p)$ can be divided into three types: (a) the cells in which the values of dimensions $(a_1,\ldots,a_i)$ are all equal to *; (b) the cells in which only one dimension value is non-* and the others are *; (c) the cells in which more than two dimension values are non-*. For the case of (a), suppose c is similar to c_g', there is only a case that c is an ancestor of c_g' (Because the values of dimensions $(a_1,\ldots,a_i)$ of cell c are * value, whereas the values of dimensions $(a_1,\ldots,a_i)$ of cell c_g' are non-*). Then c and c_g' should share the same value or c has * on the remaining dimensions $A-\{a_1,\ldots,a_i\}$, where A denotes the all dimensions of cube. Due to c has non-* dimension value on SD that is contained in $A-\{a_1,\ldots,a_i\}$, c and c_g' should share the same non-* value on SD. Then c_g' should be contained in $ExpandSet(c_p)$ according to the property(2) of the single tuple. If $c_g' \in ExpandSet(c_p)$, the non-* values of dimensions of c_g' are all derived from the corresponding dimension values of c_p. As c_g is an ancestor of c_g', the non-* values of the corresponding dimensions of c_g should be same to that of c_g'. Then the non-* dimension values of c_g are also derive from c_p. So c_p and c_g surely have no two solid-mismatches. It is in contradiction to the supposition: c_p is not matchable with c_g. c is not similar to c_g'. For the case of (b), suppose the only one different non-* value dimension is a_j ($1 \leq j \leq i$) and the * value dimension is a_k ($1 \leq k \leq i$, $k \neq j$). Then c and c_g' have one different non-* value on dimension a_j except one dimension a_k in which c has * but c_g' has non-* value, c is not similar to c_g'. For the case of (c), c and c_g' have more than two different non-* dimension values, c is not similar to c_g'. □

Lemma 2. Given two gradient cells c_g, c_g' and c_g' is obtained from c_g in the depth-first order, we have $LiveSet(c_g') \subseteq LiveSet(c_g)$.

Proof. Two cases raise here:

(1) Suppose c_g' is a non-single tuple. For every $c_p \in LiveSet(c_g')$, there are two special cases: c_p is either a single tuple or a non-single tuple. In the case that c_p is a non-single tuple. Since $c_p \in LiveSet(c_g')$, it is known that (c_g'', c_p) is a possible gradient-probe cell pairs, where c_g'' is a descendant of c_g'. Due to c_g' is obtained from c_g in the depth-first order, i.e., c_g' is obtained by taking specific values for some dimensions in which c_g has * value, c_g' is a descendant of c_g. Since c_g'' is a descendant of c_g' and c_g' is a descendant of c_g, c_g'' is also a descendant of c_g according to the definition of descendant. Due to (c_g'', c_p) is a possible gradient-probe cell pairs, we also have $c_p \in LiveSet(c_g)$. On the other hand, assume that c_p is a single tuple. It is known that there exists a certain cell $c_p' \in ExpandSet(c_p)$ and (c_g'', c_g') is a possible gradient-probe cell pairs, where c_g'' is a descendant of c_g' and c_g. Thus, we have $c_p \in LiveSet(c_g)$. In a word, for every $c_p \in LiveSet(c_g')$, we have $c_p \in LiveSet(c_g)$. Therefore, $LiveSet(c_g') \subseteq LiveSet(c_g)$ is held.

(2) Suppose c_g' is a single tuple, we have $LiveSet(c_g') = \cup LiveSet(c_{gi}')$ where $c_{gi}' \in ExpandSet(c_g')$ and $1 \leq i \leq |ExpandSet(c_g')|$. Assume that c_b' is the basic cell of c_g', we have $c_b' \in$ ExpandSet(c_g'). For any cell c_{gi}' in ExpandSet(c_g') that is not equal to c_b', it is known that c_{gi}' is a descendant of c_b'. Due to both c_{gi}' and c_b' are non-single tuples, we have $LiveSet(c_{gi}') \subseteq LiveSet(c_b')$ according to the above proof of (1) of Lemma 2. So $\cup LiveSet(c_{gi}') \subseteq LiveSet(c_b')$ where $c_{gi}' \neq c_b'$ and $1 \leq i \leq |ExpandSet(c_g')|$. Then $(\cup LiveSet(c_{gi}')) \cup (LiveSet(c_b')) = LiveSet(c_b')$, that is, $\cup LiveSet(c_{gi}') = LiveSet(c_b')$ for every cell $c_{gi}' \in ExpandSet(c_g')$. Again, c_g' is a single tuple. Then c_g' is obtained by partitioning c_g on the dimensions of SD of c_g' in the depth-first order and c_g has * value on some dimensions of SD. Since the basic cell of c_g', c_b' is obtained by take more special value (i.e., non-* value) for the dimensions of SD of c_g', c_b' is a descendant of c_g. Similarly, according to the above proof of (1) of Lemma 2, we

also have $LiveSet(c_b')\subseteq LiveSet(c_g)$. Therefore we have $LiveSet(c_g')=\cup LiveSet(c_{gi}')=LiveSet(c_b')\subseteq LiveSet(c_g)$. □

From Lemmas 1 and 2, we can derive $LiveSet(c_g')$ by pruning the single tuples that are not matchable with c_g' or the basic cell of c_g', c_b' from $LiveSet(c_g)$. For the non-single tuples in the $LiveSet(c_g)$, they can be pruned from $LiveSet(c_g)$ by the propery2 in the section3. In a word, for a given potential gradient cell c_g and it's descendant cell c_g', we can derive $LiveSet(c_g')$ from $LiveSet(c_g)$ by pruning the cells that are not matchable with c_g' or c_b' from $LiveSet(c_g)$.

4.2 Partial expansion technique

In order to produce all interesting gradient-probe cell pairs, as the gradient cells or probe cells are single tuples, we should expand the single tuples. We observe that it is not necessary to compute the all cells condensed into c_g or c_p. In particular, the cells in ExpandSet that are not meaningful, i.e, not similar to the cells to be compared are not computed. For example, suppose c_p, c_g have dimensions A, B, C, D and c_p=(1,2,3,3), c_g=(2,3,2,3) and the single dimensions SDs of c_p, c_g are both $\{A\}$. The dimension values of the cells of $ExpandSet(c_p)$ and $ExpandSet(c_g)$ are shown in the two rectangle frames in Fig.2. We scan and compare the values of dimension A, B, C, and D in the two rectangle frames according to the order of dimension lexicography. The cells in $ExpandSet(c_g)$ and the cells in $ExpandSet(c_p)$ have different non-* value on dimension A. Thus, there is only one possibly meaningful relationship between the cells in $ExpandSet(c_p)$ and the cells in $ExpandSet(c_g)$, that is, the sibling relationship. In general, both the different dimension values on one dimension in *ExpandSets* are non-* value, the meaningful relationship is sibling and if one dimension value is * value whereas the other dimension value is non-*, the meaningful relationship is ancestor/descendant. For the *ExpandSets* in Fig.2, according to the definition of sibling, the cells in $ExpandSet(c_p)$ should share same values with the cells in $ExpandSet(c_g)$ on the remaining dimensions. Thus, the meaningful values of dimensions of the cells in $ExpandSet(c_p)$ are as follow: A=1, B=*, C=* and D=3 or D=*, i.e., $ExpandSet(c_p)$= {(1,*,*,3), (1,*,*,*)}. $ExpandSet(c_p)$ should contain 2^3=8 cells if the single tuple c_p takes the complete expansion. Similarly, we have $ExpandSet(c_g)$={(2,*,*,3), (2,*,*,*)}. The similar gradient-probe cell pairs are ((1,*,*,3), (2,*,*,3)) and ((1,*,*,*), (2,*,*,*)). In the case that either c_g or c_p is a non-single tuple, it is easy to determine the meaningful dimension values in *ExpandSet* because each dimension of a non-single tuple has a definite dimension value. The above method is named as partial expansion technique in this paper. The main spirits of partial expansion technique is to recognize the possibly meaningful relationship between the cells in $ExpandSet(c_p)$ and the cells in $ExpandSet(c_g)$ and then utilize the relationship to determine the meaningful values of the dimensions in *ExpanSets* and hence reduce the size of *ExpandSets*. Thus, the key of partial expansion is to recognize the possibly meaningful relationship. Intuitively, the relationship can be recognized through scanning and comparing the dimension values. In the case that there are not dimension values that satisfy the possibly meaningful relationship, the single tuples need not to expand and there no similar gradient-probe cell pairs to be generated. The probe constraint can be used to further filter the meaningful dimension values in $ExpandSet(c_p)$. For example, in the Fig.2, if the $C_{prb}\equiv$(A=1,B=2, C=*, D=*), then the dimension value of dimension B should not take the "Or" value but take the non-value "2".

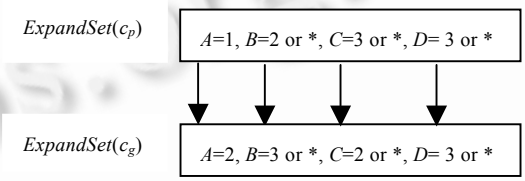

Fig.2 An example of partial expansion of two single tuples

The complete procedure of generating the interesting gradient-probe cell pairs using the partial expansion technique is given as following:

Procedure: GIP (Generating the Interesting gradient-probe cell pairs using Partial expansion)

Input: The gradient cell c_g, the probe cell c_p, a C_{prb}, a C_{grad}, and the number of dimension of cube n.

Output: The interesting gradient-probe cell pairs between c_g and c_p.

Method:

1. If $C_{grad}(c_g,c_p)$=false, return;
2. For each dimension a_k ($1\le k\le n$) {
3. Apply the C_{prb} to determine the dimension values of a_k in $ExpandSet(c_p)$;
4. If the meaningful relationship is not recognized
5. Compare the dimension values of a_k between $ExpandSet(c_g)$ and $ExpandSet(c_p)$ and recognize the meaningful relationship;
6. Else
7. Use the relationship to determine the meaningful values of dimension a_k in *ExpandSets* and if there not meaningful dimension values that satisfy the possibly meaningful relationship then return;}
8. Output the interesting gradient-probe cell pairs (c,c') where $c\in ExpandSet(c_g)$, $c'\in ExpandSet(c_p)$;

The computational complexity of the GIP procedure can be measured in terms of the number of execution times of key operation. The comparison operation of dimensions of cells is taken as the key operation. The total number of execution times is equal to $n+|ExpandSet(c_g)|\times|ExpandSet(c_p)|\times n$, where the *ExpandSets* contain no such cells that are not meaningful.

4.3 The description of eLiveSet algorithm

Similar to the framework of the LiveSet algorithm, eLiveSet algorithm also consists two phases: (1) Computing the set of probe cells P from the relation table R using BU-BST algorithm with the significance and probe constraints, (2) Produce the interesting gradient-probe cell pairs.

Algorithm: eLiveSet (short for extending LiveSet algorithm).

Input: A relation table R, a C_{sig}, a C_{prb} and a C_{grad}.

Output: The complete set of gradient-probe cell pairs satisfying the constraints.

Method:

1. Apply the BU-BST algorithm to compute set of probe cells P from R utilizing the C_{sig} and C_{prb} constraints.
2. Initialize the potential gradient cell to cell c_g=(*,*,…,*) and $LiveSet(c_g)=P$.
3. For every gradient cell c_g do {
4. If ($C_{sig}(c_g)$=true)
5. For every $c_p\in LiveSet(c_g)$ {
6. If both the gradient cell c_g and probe cell c_p are single tuples (i.e., their SDs are not equal to ∅), invoking the GIP procedure to output the interesting gradient-probe cell pairs (e,e'), where $e\in ExpandSet(c_g)$ and $e'\in ExpandSet(c_p)$;
7. If both the gradient cell c_g and probe cell c_p are not single tuples and c_g is similar to c_p and $C_{grad}(c_g, c_p)$=true, output (c_g,c_p);
8. If the gradient cell c_g is a single tuple but the probe cell c_p is not, invoking the GIP procedure to output the interesting gradient-probe cell pairs (e,c_p), where $e\in ExpandSet(c_g)$;
9. If the gradient cell c_p is a single tuple but the probe cell c_g is not, invoking the GIP procedure to output the interesting gradient-probe cell pairs (c_g,e'), where $e'\in ExpandSet(c_p)$; }
10. Use the measure of c_g to prune $LiveSet(c_g)$.
11. If $LiveSet(c_g)$ is empty or c_g has no potential to grow, terminate this branch and backtrack to process the next

cell according to the depth-first order.

12. If c_g has potential to grow, expand it to the next level, according to the depth-first order. If a descendant cell c_g' of c_g is processed from this expansion, derive *LiveSet*(c_g') from *LiveSet*(c_g) according to the method discussed in section 4.1.}

Algorithm Explanation: (1) Although the gradient function is assumed as the form '$m(c_g)-m(c_p)\ \theta\ v$', our algorithm is still applicable to the other form gradient functions by modifying the computation part of the gradient function. In addition, if v is not a const value but some interval such as $1\leq v\leq 2$, we should modify the gradient test part by testing not only the lower bound but also the higher bound. (2) We can use the measure of c_g to further prune *LiveSet*(c_g) in the step 10. In particular, for any probe cell c_p in *LiveSet*(c_g), if $C_{grad}(c_g,c_p)$=false, c_p can be pruned from *LiveSet*(c_g). In our study, the probe cells in *LiveSet*(c_g) are assumed to be stored in the ascending order according to the measures. If the measure of one probe cell c_p can not satisfy the constraint $C_{grad}(c_g,c_p)$, all the probe cells following it will not satisfy the gradient constraints function and hence can be pruned from *LiveSet*(c_g). (3) In the extreme case that there are not single tuples in the condensed cube, eLiveSet algorithm degenerates into the LiveSet algorithm.

5 Experiments

In this section, we present our experimental results on the performance (in terms of time) of mining constrained cube gradient from a data cube. All experiments are conducted on a PC platform with an Intel Pentium Ⅲ 500M CPU, 218M RAM and Windows 2000 OS. Two mining algorithms are used in our experiments. The first algorithm is eLiveSet that is constructed on a condensed cube that is computed by the BU-BST algorithm. The other is LiveSet that is constructed on a general cube that is computed by BUC algorithm. All experiments are performed using synthetic (algorithmically generated) datasets. To study the effects of data skew, we generate the first dataset of 1M (=1 024) tuples following the Zipf distribution with different zip factors. A zip factor of 0 means the data is uniformly distributed. In the first dataset, the number of dimension is set to 9 and the cardinality of all attributes is set to 1000. The aggregate function used in the cube computation algorithm is SUM function. The performance of the algorithms with different constraints is shown in Figs.3, 4 and 5 respectively. In Fig.3, the significance constraints C_{sig}>0 and the gradient constraints C_{grad}=$m(c_g)-m(c_p)$>0. The value of the non-* value dimension in probe constraint C_{prb} is set to [1,500]. The non-* value dimensions in C_{prb} are fixed on the first k dimensions ($1\leq k\leq 9$), respectively. Fig.3 shows the runtime of both algorithms are less with the increase of the number of non-* value dimensions in C_{prb}. However, eLiveSet algorithm is faster than LiveSet algorithm. The main reason is that many cells are condensed into the single tuples and the search space of the cells including the probe cells and the gradient cells to be handled reduces dramatically in eLiveSet algorithm. Just as shown in Figs.4 and 5, the constraints would reduce also dramatically the search space of the cells. However, in the uniform datasets, the condensed cube still plays an important role. Figures 4 and 5 show the runtime of both algorithms are less with the increase of the threshold of C_{sig} and C_{grad}. In Fig.6, we increase the skew in the distribution of distinct values in the dataset. The test shows the runtime of both algorithms is larger with the increase of the Zipf. When the skew is high, the number of single tuples would be smaller and the number of gradient cells and probe cells would be also smaller. However, the time of cube computation would be larger with the increase of the skew and affect the performance of the algorithms.

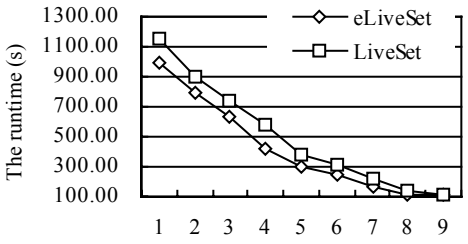

Fig.3 The scalability with probe cells

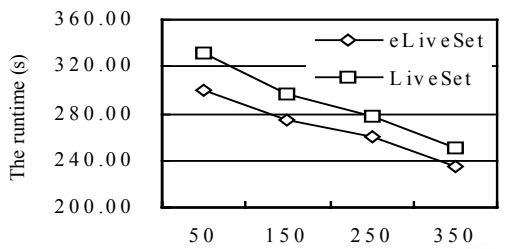

Fig.4 The scalability with significance threshold

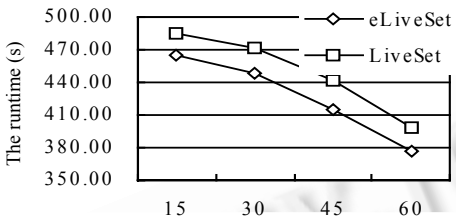

Fig.5 The scalability with gradient threshold

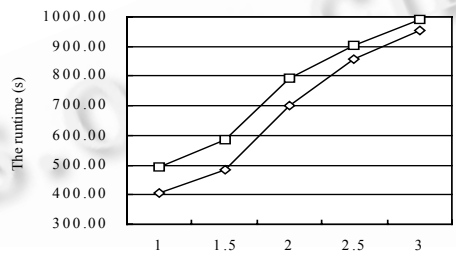

Fig.6 The scalability with the Zipf distribution

6 Conclusions

In this paper, we study the problem of mining constrained cube gradient for a condensed cube. A new algorithm, eLiveSet, is developed for our problem by the extension of the existing efficient algorithm LiveSet. The performance tests of algorithms are also reported. There are many interesting issues in our future wok, for example, we can mine the cube gradient in an interactive mode but in batch mode, as shown in this paper. Further enhancing the performance of eLiveSet algorithm is also our interesting topic.

Acknowledgement The authors would like to thank the anonymous referees for providing many useful comments.

References:

[1] Han J. Towards on-line analytical mining in large databases. SIGMOD Record, 1998,27(1):97~107.
[2] Palpanas T. Knowledge discovery in data warehouses. SIGMOD Record, 2000,29(3):88~100.
[3] Abdulghani AA. Cubegrades-Generalization of association rules to mine large datasets [Ph.D. Thesis]. New Brunswick: The State University of New Jersey, 2001.
[4] Agrawal R, Tomasz I, Arun NS. Mining association rules between sets of items in large databases. In: Buneman P, ed. Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data. New York: ACM Press, 1993. 207~216.
[5] Gray J, Bosworth A, Layman A, Pirahesh H. Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-total. In: Stanley YW, ed. Proceedings of the 12th International Conference on Data Engineering. Washington: IEEE Computer Society Press, 1996. 152~159.
[6] Dong G, Han J, Lam J, Pei J, Wang K. Mining multi-dimensional constrained gradients in data cubes. In: Peter MG, ed. Proceedings of the 27th International Conference on Very Large Data Bases. San Fransisco: Morgan Kaufmann Publishers, 2001. 321~330.
[7] Wang W, Feng J, Lu H, Jeffrey XY. Condensed cube: An effective approach to reducing data cube size. In: Rakesh A, ed. Proceedings of the 18th International Conference on Data Engineering. Washington: IEEE Computer Society Press, 2002. 155~165.
[8] Sarawagi S, Agrawal R, Megiddo N. Discovery-Driven exploration of OLAP data cubes. In: Hans-Jörg Schek, ed. Proceedings of the 6th International Conference on Extending Database Technology. New York: Springer-Verlag, 1998. 168~182.
[9] Beyer K, Ramakrishnan R. Bottom-Up computation of sparse and iceberg CUBE. In: Davidson DB, Delis A, eds. Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data. New York: ACM Press, 1999. 359~370.
[10] Han J, Pei J, Dong G., Wang K. Efficient computation of iceberg cubes with complex measures. In: Sellis T, ed. Proceedings of the 2001 ACM SIGMOD International Conference on Management of Data. New York: ACM Press, 2001. 1~12.