基于立方体计算的关键梯度分析*)

遇 辉[1,2] 唐世渭[1,2] 杨冬青[1] 李 囡[1]
(北京大学信息科学技术学院 北京 100871)[1]
(北京大学视觉与听觉信息处理国家重点实验室 北京 100871)[2]

摘 要 梯度分析是数据仓库和联机分析处理中的一项重要分析任务,在决策支持中发挥着重要作用。本文根据实际应用的需要,提出了一种新颖的关键梯度分析方法。借助立方体计算中的计数排序和分割策略,通过扩展补充路径,并利用插入排序方法,实现了高效的关键梯度分析算法。在模拟数据上进行了大量的实验,结果证明了算法的高效性和实用性。

关键词 关键梯度,梯度分析,立方体计算,数据仓库,联机分析处理

Significant Gradients Mining Based on Data Cube Computation

YU Hui[1,2] TANG Shi-Wei[1,2] YANG Dong-Qing[1] LI Nan[1]
(School of Electronics Engineering and Computer Science, Peking University, Beijing 100871)[1]
(National Laboratory on Machine Perception, Peking University, Beijing 100871)

Abstract Gradient analysis is an important data analysis task in data warehousing and online analytical processing, which has played an important role in the application of decision support. This paper considers a novel type of gradient analysis, significant gradient analysis. Significant gradient analysis is expressve, capable of capturing trends in data and answering "what-if" questions. The problem of mining significant gradients is challenging since the significant gradients can be widely scattered in the cube lattice, and do not present any monotonicity. To tackle the problem and develop techniques to speed up the search the state-of-the-art cube computation algorithm is extemded. An extensive performance study is reported to illustrate the effect of the approach.

Keywords Significant gradients, Gradient analysis, Cube computation, Data warehousing, Online analytical processing

1 引言

梯度分析是联机分析处理和数据仓库中的一项重要分析任务,它可用于分析数据中隐含的趋势,也可以回答 what-if 问题,在实际应用中具有重要的价值。目前,针对大型多维数据库和数据仓库的梯度分析研究已取得了很多成果。最先研究梯度分析的文献中将梯度分析称为方体关联规则[1],它是一种广义的梯度分析,也可以理解为扩展了的关联规则。在提出方体关联规则之后,人们又提出了约束梯度分析的概念[2],约束梯度分析相当于是方体关联规则的精简版。

在关联规则中,聚集函数只能是 count,而方体关联规则可以处理任意一种常用的聚集函数,如 AVG、MAX 等。方体关联规则比普通的关联规则更具有描述力,并且能够捕捉任意度量的趋势。但是,这种灵活性也使得它的计算复杂度增加,影响了它的可扩展性。之后提出的约束梯度分析引入了重要性约束 C_{sig}、探测约束 C_{prb} 和梯度约束 C_{grad}。重要性约束确保探查的单元在统计上具有一定的重要性,例如至少覆盖一定数量的基本单元或者一定量的总销售额。探测约束是针对维度信息的约束,它表示梯度分析中感兴趣的维属性。至于梯度约束,顾名思义,就是梯度分析中感兴趣的梯度幅度范围。加上了这些约束之后进行的梯度分析,会使得结果更清晰易懂,并且使得梯度的挖掘过程更加高效。

这两种梯度分析问题有一个共同之处,就是它们对梯度的定义都包括三种类型。假设 c_p 和 c_g 是构成梯度的两个单元,c_g/c_p 表示对应梯度的变化幅度,在广义梯度分析和约束梯度分析中,c_g 可以是 c_p 的祖先单元,可以是 c_p 的派生单元(即子单元),也可以是 c_p 的兄弟单元,在挖掘过程中,这三种梯度是同时挖掘的。但是,在某些应用需求中,并不需要同时挖掘三种类型的梯度,而只需要其中的一种或两种。在这种情况下,如果同时挖掘三种梯度,不仅会生成大量不感兴趣的结果,且费时费力。我们以对肺癌患病原因的分析为例,当把梯度约束 c_p 限制到派生单元时,会更有价值。假设 c_p 代表吸烟 10 年以上的人群,c_g 代表吸烟 10 年以上并且经常熬夜的人群,聚集函数是 AVG,代表平均患病率。如果发现 avg(c_g)/avg(c_p)的比值很大,则说明如果烟龄很长,若再加上经常熬夜,就会提高患肺癌的可能性。这样的知识对防治肺癌无疑是很有价值的。在实际应用中,还有很多类似这种针对派生梯度的挖掘需求,因此有必要研究专门针对派生梯度的挖掘方法。

此外,目前的梯度分析问题都是需要根据设置的梯度阈值来进行探查和分析,也就是说,整个分析过程非常依赖于阈值的设定。当用户对数据不够了解而无法指定合理的梯度阈值时,分析的结果就可能意义不大。设置阈值过大,会丢掉重要的信息;设置阈值过小,又可能会得到大量的结果,用户很难理解和分析。因此,如果能够找到一种与阈值无关的梯度分析方法,又能够保证呈现给用户最重要的梯度特征,将会很有价值。显然,用户会更加关心幅度变化大的梯度。如果对立方体格中的所有梯度进行排序,按照从大到小顺序呈现给

*)基金项目:国家自然科学基金项目(60473072);国家自然科学基金项目(60473051)。遇 辉 博士研究生。

用户，那么用户就可以从幅度最大的梯度开始探查过程，使得用户不会遗漏掉最关键的梯度特征。但这种排序方式会有一个很严重的问题，就是必须把所有的父子单元对的组合（即梯度）都计算出来并保存，否则无法进行全排序。当数据量很大的时候，对所有梯度的保存及全排序的时间和空间代价都是惊人的，因此必须采取一种折衷的办法。

基于上述两方面原因，本文提出一种与阈值无关的关键派生梯度的挖掘方法。通过将前 k 个增幅最大的梯度作为关键梯度呈现给用户，保证用户可以查看到最重要的梯度特征。同时，借助高效的立方体计算策略，通过增加补充路径的方式，利用插入排序，实现了关键派生梯度的高效挖掘算法，避免了梯度的完全保存和全排序过程，从而大大降低了算法的时间空间复杂度。由于呈现给用户的是“最关键”的前 k 个派生梯度特征，因此本文将它简称为关键梯度分析 SigGrad(for Significant Gradients)。

2 问题描述

首先介绍相关概念，然后给出关键梯度分析问题的定义。在数据仓库中，设基表 $B=(D1,\cdots,Dn,M)$，属性 $D1,\cdots,Dn$ 构成立方体的维度，M 是度量。基表 B 中的元组称为基元组。对于一个元组 t，$t.Di(1\leqslant i\leqslant n)$ 和 $t.M$ 分别是元组 t 在维 Di 和度量 M 上的值。一个元组 $c=(d1,\cdots,dn)$ 称为一个聚集单元（或简称单元），如果 $di\in Di\cup\{*\}(1\leqslant i\leqslant n)$，其中“*”表示在对应维上汇总，且 c 的覆盖集是与 c 的所有非 * 维度匹配的基元组集合，即 $cov(c)=\{t\in B|t.Di=c.Di$，如果 $c.Di\neq *,1\leqslant i\leqslant n\}$。一个聚集单元 c，如果所有的维都取特定值，即非 *，则 c 称为基本单元；如果它有 k 个非 * 维，那么就称 c 为一个 k-d 单元，其中 d 代表维度。聚集单元的完全集构成数据立方体 $CUBE(B)$。基于各个聚集单元之间覆盖集的关系，在 $CUBE(B)$ 之上定义偏序 $<$。如果一个聚集单元 c 与另一个聚集单元 c′ 满足 $cov(c)$ 包含 $cov(c')$，那么 c 是 c′ 的祖先节点，即 $c<c'$。换句话说，一个单元的祖先就是在它的某些维度上进行了概化的更汇总的单元。定义了偏序关系的 $CUBE(B)$ 就形成了数据立方体完整的格结构。

定义 1（关键梯度） 给定基表 B，一个重要性约束 C_{sig}，一个探测约束 C_{prb}，一个显示约束 k，关键梯度分析问题是要找到梯度变化幅度 $m(c_g)/m(c_p)$ 最大的前 k 个父子单元对的组合 (c_g, c_p)，其中 c_g 和 c_p 满足重要性约束，c_p 满足探测约束，且 c_g 是 c_p 的子单元。

表 1 基表举例

A	B	C	a tuples	x tuples P	pred
a1	b1	c1	7	1	11.20%
a1	b1	c2	9	2	22.20%
a1	b2	c1	4	1	25.00%
a1	b2	c2	7	3	42.86%
a1	bx	c1	10	2	20.00%
a1	b3	c2	8	3	37.50%
a2	b1	c1	5	0	0.00%
a2	b1	c2	11	1	9.00%
a2	b2	c1	7	2	25.57%
a2	b2	c2	15	3	20.00%
a2	b2	c2	15	3	0.00%
a2	bx	c2	12	3	25.00%

表 1 所示为一个有 3 个维的基表，分别是 A={a1,a2}，B={b1,b2,b3}，C={c1,c2}。设聚集函数是 AVG，所有可能的聚集单元总数是 $|A\cup\{*\}||B\cup\{*\}||C\cup\{*\}|=3*4*3=36$，聚集单元构成的格结构如图 1 所示。图上的每一条连线代表一个派生梯度关系，连接的两个单元是一对父子单元，连线上端的单元是探测单元 c_p，下端的单元是派生单元 c_g。假设 c_p 是 k-d 单元，那么 c_g 就是 $(k+1)$-d 单元，是在 c_p 的某个 * 维上取了一个特定的维成员值。每一条连线都对应了一个值，即 $avg(c_g)/avg(c_p)$，可以把这个值理解为连线的权重，代表了连线连接的两个单元的度量变化幅度。关键梯度分析问题就是要找到图 1 中前 k 个上升幅度（即权重）最大的连线，即梯度。

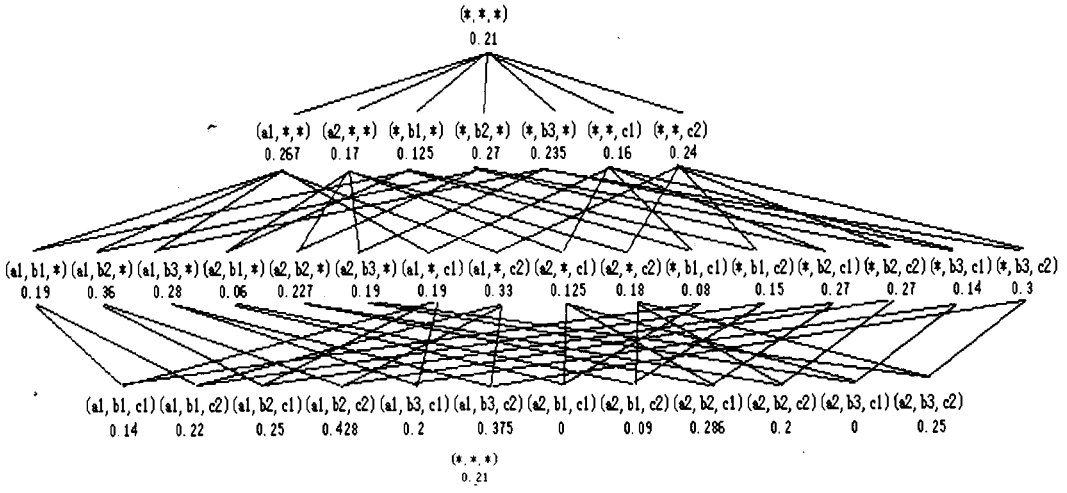

图 1 梯度搜索空间

连线的权重取值有三种情况：①大于 1，②小于 1，③等于 1。对于第一种情况，说明派生单元的聚集值比探测单元的聚集值大。以肺癌分析为例，说明派生单元新加的维成员属性在探测单元代表的属性组合上起到了正向的致病作用。例如，如果烟龄很长的话，若再经常熬夜，就会提高患肺癌的可能性。对于第二种情况，说明派生单元对应的新维成员相对于探测单元没有致病倾向，对防病会有好处。而第三种情况意味着派生单元与探测单元对应的两个人群的患病率相同，也就是说，派生单元的新加属性对是否患病不造成任何影响。如果我们把派生单元相对于探测单元新加的维成员称为派生

属性，那么关键梯度分析问题就是要找到前 k 个最关键的派生属性。假设显示约束 k 为 10，则上例共返回 10 个关键梯度，如图 2 所示。

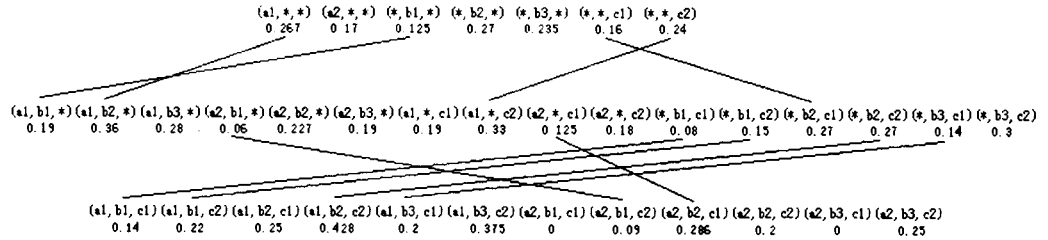

图 2　关键梯度

最极端情形是 k 值很大，以致超过了权重大于 1 的连线总数，那么这时的挖掘结果就是返回所有权重大于 1 的连线，这些连线对应的派生属性对它们的探测单元对应人群会有致病作用，致病的程度由连线的权重决定。在其他应用中，也可以将权重定义为派生单元与探测单元的差，即 $\mathrm{avg}(c_g)-\mathrm{avg}(c_p)$，等等。至于如何定义，要根据应用中的实际需要而定。

3　关键梯度分析算法

前一节已介绍，派生梯度实质上就是数据立方体格结构中的父子单元之间的连线，找到关键梯度等价于找到权重最大的前 k 条连线。如果数据立方体是完全物化的，那么上述查找过程仅仅是简单的查询问题。但是，在实际应用中，由于聚集单元通常是海量的，以致难以预先计算和存储，因此假设可提供的信息只有基表，我们需要从基表上计算关键梯度。实际上，这种情况与立方体计算面临的情况是一样的，那么如果能够借助立方体的计算过程同时也得到关键派生梯度，无疑将是高效的。近几年研究者们提出了很多高效的立方体计算方法[3~9]，BUC 算法是其中最高效的算法之一[8,9]，且由于它的计算策略是深度优先方式，可以在计算聚集单元的同时计算梯度，因此我们对它进行扩展，来实现关键梯度的高效挖掘。

在 BUC 算法中，将所有维按字母序排序，以决定遍历的顺序和路径。当维信息和序列关系确定之后，数据立方体的格结构（见图 3 左）就可以用一棵结构确定的遍历树来实现对所有聚集单元的完整且无重复的遍历。也就是说，每一个聚集单元被遍历一次，且仅有一次，如图 3 右所示。

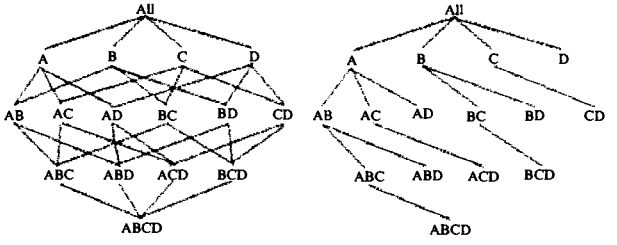

图 3　遍历路径

然而，当将这种遍历策略用于挖掘关键梯度时就会发现，图 3 右所示的遍历树结构只能保证聚集单元的遍历完整性，而没有遍历格上的所有边，即梯度。由于关键梯度之间不具备任何单调或弱单调性质，因此必须对格结构实施完全遍历，扩展 BUC 算法遍历的路径。同时，由于 BUC 方法利用了计数排序和分割策略，使得聚集单元的计算时间大大缩减，因此我们一方面要扩展 BUC 的遍历路径，另一方面还要保留 BUC 的计数排序和分割策略，使得关键梯度的挖掘过程不仅全面而且高效。

为了保留计数排序和分割策略，必须采用深度优先遍历策略。除了行走 BUC 的路径之外，在每一个探测单元，要扩展剩余的路径。下面，我们继续以表 1 为例说明算法的处理过程。首先，计算最汇总层单元的聚集值，如果(*，*，*)满足重要性阈值和探测阈值，深度优先遍历它的所有子孙单元。根据字母序，首先在维 A 上进行扩展，按照维 A 的成员对基表 B 进行分割和计数排序，得到两个派生子单元(a1，*，*)和(a2，*，*)的覆盖单元集和各自的聚集值。(a1，*，*)继续同样的过程，在(a1，*，*)的覆盖集上，按维 B 进行计数排序和分割，将(a1，*，*)的覆盖集分割为三块，分别对应(a1，b1，*)、(a1，b2，*)和(a1，b3，*)。类似进行下面的过程，深度优先遍历子单元(a1，b1，*)，对直到最细节层的(a1，b1，c1)和(a1，b1，c2)。之后，回溯到(a1，b2，*)，对(a1，b2，*)的覆盖集按维 C 进行分割，得到(a1，b2，c1)和(a1，b2，c2)。当处理完(a1，b3，*)分支之后，回溯到(a1，*，*)覆盖集，按下一个维(维 C)进行分割，得到(a1，*，c1)和(a1，*，c2)。这时，如果按照 BUC 的遍历策略，此处就会停止继续下行探查过程，那么(a1，*，c1)→(a1，b1，c1) (a1，*，c1) → (a1，b2，c1) (a1，*，c1) →(a1，b3，c1) (a1，*，c2) → (a1，b1，c2) (a1，*，c2) → (a1，b2，c2) (a1，*，c2) → (a1，b3，c2)这些派生梯度关系就会缺失。因此，此处对(a1，*，c1)和(a1，*，c2)各自的覆盖集都要再按照维 B 的成员进行分割，用来得到上面这些缺失的梯度关系。但是，这些补充的分割过程对应的遍历过程不会继续向下进行，以保证立方体格上的每一条边只遍历一次。

基于上述策略，在图 3 之上扩展补充路径，如图 4 所示。实线表示 BUC 遍历路径，每一个汇总层单元都按字母序继续下行遍历过程。虚线表示补充路径，不会在当前子单元立刻进行下行遍历。

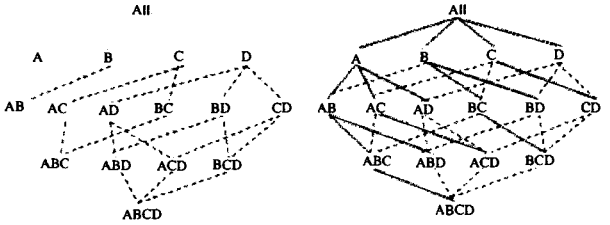

图 4　完全遍历路径

重要性约束确保探查的单元在统计上具有一定的重要性，例如至少覆盖一定数量的基本单元或者一定量的总销售

额。这种约束具有单调性质，在遍历时可以用于裁剪分支。由于算法是从最汇总层开始遍历过程，在第一遍扫描基表的同时，可以计算出各个 1-d 单元的覆盖集规模，找到其中不满足重要性约束 C_{sig} 的单元，那么由于覆盖关系在格结构中的单调性，这些不满足 C_{sig} 的单元的所有派生单元都可以裁剪掉，因为它们也一定不会满足 C_{sig}。由于探测约束是针对维度信息的约束，它表示梯度分析中感兴趣的维属性，因此可以将上面找到的不满足重要性约束的单元所对应的维属性信息添加到探测约束 C_{prb} 中，在后续遍历过程中由 C_{prb} 统一裁剪。

算法用一个长度为 k 的数组 outputEdge[k]存储当前最关键的 k 个梯度，outputEdge[k]初始值全部设为 0。为了使插入新梯度的过程更高效，outputEdge[k]中的梯度按照从大到小顺序排列，我们在遍历过程中遇到的每一个新梯度都采取插入排序的方式检测其是否应放入数组 outputEdge[k]中。也就是说，从向 outputEdge[k]中加入的第一个梯度开始，每次遍历到一个新的梯度，都将其与 outputEdge 中最小的 outputEdge[$k-1$]进行比较，如果小于 outputEdge[$k-1$]，则说明新的梯度一定不是前 k 最大梯度之一。否则，将新的梯度按照插入排序加入 outputEdge，使得插入后的 outputEdge[k]仍然是从大到小顺序，outputEdge[$k-1$]最小。

算法首先计算最汇总层单元(*，*，*)的聚集值，并且检查这个单元是否满足重要性阈值和探测阈值(算法第 1 行)。如果满足约束条件，则第 2 行就是对它的所有汇总维度进行分割，首先按维 A 进行分割，第 3 行获得 A 的维成员个数；下一步同时进行分割和计数排序，dataCount[d]中保存该维分割之后各个子块的大小。第 5 行到第 15 行是对分割之后的每一个子块进行处理。首先计算各个子块的聚集值，得到派生单元 c_{grad}，计算其与探测单元的比率，将这个比率与 outputEdge[$k-1$]进行比较。如果小于，则这个梯度对一定不是前 k 个最关键梯度；反之，则去掉 outputEdge[$k-1$]的值，将新梯度按照插入排序加入到 outputEdge[k]数组中。第 12 行的条件判断是在判断哪些维度需要继续下行遍历，哪些维度是为补充路径而分割的。根据 BUC 遍历策略，那些大于当前分割维度 d 的维，需要继续进行深度优先分割遍历。

算法 1(关键梯度)SigGrad(*input*, *dim*)

输入：Input：the relation to aggregate;

Dim：the starting dimension for depth-first search

输出：the first k edges that satisfy the two constraints;

过程：

```
1. if c_prb = Aggregate(input) not satisfy the constraint C_sig and C_prb,
   return;
2. for d = 0; d < numDims && input.d ! = *; d++ do
3.       Let C = cardinality[d];
4.       partition(input, d, C, dataCount[d]);
5.       Let m = 0;
6.       for i = 0; i < C; i++ do //for each partition
7.            Let c = dataCount[d][i];
8.            c_grad = Aggregate(input[m…m+c]);
9.            if c_grad/ c_prb > outputEdge[k-1] then
10.                 insert (c_grad, c_prb) and their ratio into outputEdge
                    in descending order
11.       end if
12.           if d >= dim then
13.                SigGrad(input[m…m+c], d+1);
14.           end if
15.           m += c;
16.      end for
17. end for
```

其中，numDims 表示维度的总数，cardinality[numDims]记录每一个维的成员个数，outputEdge 记录当前输出结果，dataCount[numDims]保存每一份分割的大小，dataCount[i]是每一个维成员对应的分割块大小。

4 实验与结果分析

实验的硬件环境为 P4 1.5GHz 的 CPU 和 512MB 的内存，软件环境为 Windows 2000(Professional)操作系统，所有代码均用 Visual C++(6.0)实现。

为了对算法的效率进行分析，我们根据实际数据的特点，生成了大量的模拟数据。模拟数据集的每一个基本单元包括两个数值：一个是总数，一个是其中属于目标类的个数。基本单元的患病率满足[0,1]区间的半边正态分布，均值为 0，标准方差是 1，即大多数 tuple 的患病率在 0 附近，患病率越高，tuple 数越少。

关键梯度分析问题的最直接的实现方法是首先计算满足探测约束条件的频繁模式，然后在频繁模式的基础上生成满足梯度约束条件的梯度规则，再对这些规则进行从大到小的排序，取出前 k 个最大的梯度，即为关键梯度的挖掘结果。这种方法与 SigGrad 方法的本质区别在于，SigGrad 方法利用了立方体计算中的计数排序和分割策略。因此，为了分析本文提出的基于立方体计算策略的挖掘算法的效率，我们对基于分割策略的深度优先算法与去掉分割策略的算法进行了实验比较，实验结果如图 5 所示。

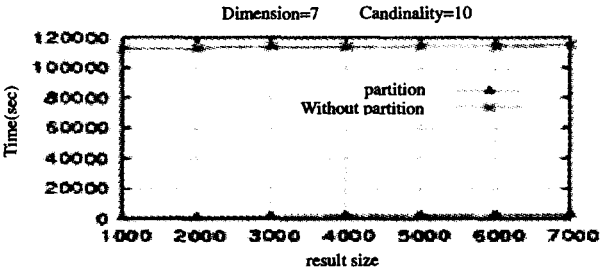

图 5 分割策略对算法效率的影响

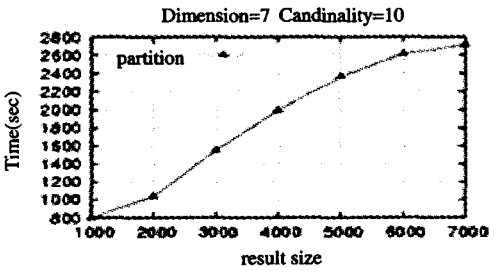

图 6 显示约束对运行时间的影响

可以看到，分割策略大大提高了算法的效率，这是因为每一个聚集单元值的计算过程都是在分割后的小数据集上进行，很大程度上降低了时间开销。图 6 单独给出了 SigGrad 算法在显示约束变化下运行时间的趋势，随着 k 的增大，程序运行过程中需要维护的当前结果就会越大，排序的开销增加，因此整个算法的运行时间也随之提高。这也证明了关键梯度分析的价值。由于用户通常关注的是少量的幅度大的梯度，即 k 比较小，因此算法会非常高效。

我们对算法在维数和维成员数上的可扩展性也进行了实验。将维度从 2 维变化到 10 维，将维成员的个数从 10 个变化到 1000 个，分别进行了实验，实验结果如图 7 所示。从实

(下转第 107 页)

移动对象数据库管理的重要方面，3D R-Tree 是可以提供对于高维数据有效管理的数据结构。本文基于 3D R-Tree 结合移动对象数据特点提出了新的插入代价参数，并利用最小代价优先搜索算法确定从根节点到 level=1 的节点的全局最优插入路径。在选择叶节点插入时，保证叶节点的空白区域增加最小，以减小在查询时访问无效节点的可能性。这样，虽然在插入记录时需要额外的节点访问，但与 3DR-Tree 相比提高了查询效率。移动对象数据库领域目前的大部分研究工作集中于区域查询处理，未来工作是研究移动对象数据库环境下的连接以及最邻近算法。

参 考 文 献

1 Satenis S, Jensen C S, Leutenegger S T. Indexing the positions of continuously moving objects. In: Proc. of the 202 ACM SIGMOD Int'l Conf. on Management of Data. New York: ACM Press, 2000. 331～342

2 Mokbel M F, Ghanem T M, Aref W G. Spatio-Temporal Access Methodes. IEEE, 2003

3 Vazirgiannis T M, Sellis T. Spatio-Temporal Indexing for Large Multimedia Applications. In: Proc. of the IEEE Conf. on Multimedia Computing and Systems, ICMCS, 1996

4 Pfoser D, Jensen C S, Theodoridis Y. Novel Approaches in Query Processing for Moving Object Trajectories. In: Proc. of the Intl Conf. on Very Large Data Bases(VLDB), 2000. 395～406

5 Nascimento M A, Silva J R O. Towards historical R-trees. In: Proc. of the ACM Symp on Applied Computing(SAC), 1998. 235～240

6 Tao Y, Papadias D. Efficient Historical R-trees. In: Proc of. the Intl Conf. on Scientific and Statistical Database Management(SSDBM), 2001. 223～232

7 Tao Y, Papadias D. MV3R-Tree: A Spatio-Temporal Access Method for Timestamp and Interval Queries. In: Proc. of the Intl. Conf. on Very Large Data Bases, (VLDB), 2001. 431～440

8 Song Z, Roussopoulos N. SEB-tree: An Approach to Index Continuously Moving Objects. In Mobile Data Management(MDM), 2003. 340～344

9 Chakka V P, Everspaugh A, Patel J M. Indexing Large Trajectory Data Sets with SETI. In: Proc. of the Conf. on Innovative Data Systems Research(CIDR), 2003

10 Beckmann N, Kriegel H, Schneider R, et al. The R*-Tree: An efficient and robust access method for points and rectangles. In: Proc. of the 1990 ACM -SIGMODE Conf.

(上接第 99 页)

验结果可以看出，利用立方体计算中的分割策略大大提高了算法的效率，维度越大，效果越显著。

此外，由于我们在算法中使用了插入排序方法来记录当前幅度最大的前 k 个梯度，对算法效率的提高也起到了相当大的作用。插入排序在最好情况下的时间复杂度是 $O(n)$，最坏情况下的时间复杂度是 $O(n^2)$，其中 n 表示数据立方体格中的总边数。采用插入排序的算法与直接比较的方法相比，优势显而易见。此处限于篇幅，不再详细列出实验结果。

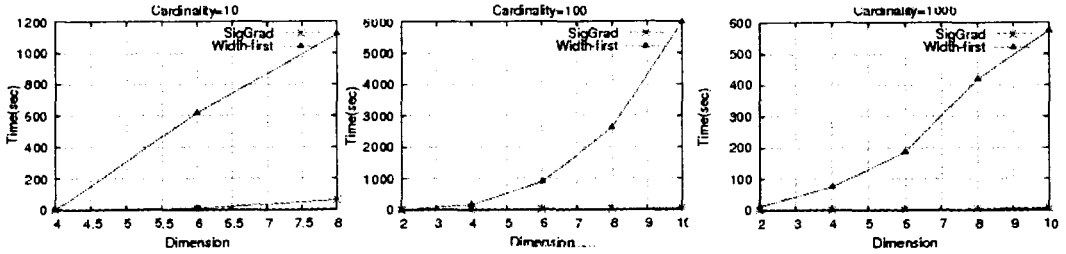

图 7 分割策略对算法效率的影响

小结 本文提出了一种与梯度阈值无关的关键梯度分析方法。我们通过在立方体计算过程中添加补充路径，实现了深度优先方式的关键梯度挖掘方法。由于利用了计数排序、分割策略以及插入排序，使得算法的效率得到很大提高。同时，由于返回的结果是最关键的梯度特征，使得用户不仅在最大程度上不会遗漏重要有价值的信息，也使得分析的结果更加简洁易懂。实验结果证明了算法的高效性和适用性。下一步的工作是针对其他两种梯度特征进行研究，提出合适的分析策略和相应的高效算法。

参 考 文 献

1 Imielinski T, Khachiyan L, Abdulghani A. Cubegrades: Generalizing association rules. In: Proc. of the 8th Int Conf. on Data Mining and Knowledge Discovery. ACM, 2002. 219～257

2 Dong G, Han J, Lam J, et al. Mining multi-dimensional constrained gradients in data cubes. In: Proc. 2001 Int Conf on Very Large Data Bases (VLDB'01), Roma, Italy, 2001

3 Zhao Y, Deshpande P M, Naughton J F. An array-based algorithm for simultaneous multidimensional aggregates. In: Proc. 1997 ACM-SIGMOD Int Conf. Management of Data (SIGMOD'97), 1997. 159～170

4 Han J, Pei J, Dong G, et al. Efficient computation of iceberg cubes with complex measures. In: Proc. 2001 ACM-SIGMOD Int Conf. Management of Data (SIGMOD'01), 2001. 1～12

5 Wang K, Jiang Y, Yu J X, et al. Pushing aggregate constraints by divide-and approximate. In: Proc. 2003 Int Conf. Data Engineering (ICDE'03), 2003. 291～302

6 Xin D, Han J, Li X, et al. Star-cubing: Computing iceberg cubes by top-down and bottom-up integration. In: Proc. 2003 Int Conf. on Very Large Data Bases (VLDB'03), 2003. 476～487

7 Ng R T, Wagner A S, Yin Y. Iceberg-cube computation with PC clusters. In: Proc. 2001 ACM-SIGMOD Int Conf. Management of Data (SIGMOG'01), 2001

8 Ross K A, Zaman K A. Optimizing selections over datacubes. In: Statistical and Scientific Database Management, 2000. 139～152

9 Beyer K, Ramakrishnan R. Bottom-up computation of sparse and iceberg cubes. In: Proc. 1999 ACM-SIGMOD Int Conf. Management of Data (SIGMOD'99), 1999. 359～370