PrefixCube 计算的优化*)

冯玉才　方　琼　李　曲　冯剑琳

(华中科技大学计算机科学与技术系　武汉430074)

摘　要　基本单元组浓缩把那些由同一条基表元组聚集计算得到的立方元组浓缩成一条，从而减小数据立方的体积。共享前缀技术通过消除元组之间的前缀冗余来进一步压缩数据立方。PrefixCube 就是将基本单元组的浓缩与共享前缀结合，而提出的一种有效的数据立方组织形式。在本文中，我们以批处理模式进一步优化计算 PrefixCube，从而减少 PrefixCube 的计算时间代价。通过在模拟数据集和真实数据集上的实验证明，在大多数数据集上，以批处理模式计算 PrefixCube 要优于一般模式计算 PrefixCube。

关键词　联机分析处理，前缀立方，数据小方，基本单元组

Optimizing the Computation of PrefixCube

FENG Yu-Cai　FANG Qiong　LI Qu　FENG Jian-Lin

(Department of Computer Science, Huazhong University of Science and Technology, Wuhan430074)

Abstract　BST Condensing is an effective approach to reducing cube size, which condenses those tuples, aggregated from the same single base relation tuple, into one physical tuple. Prefix-sharng technique can further reduce the size of a data cube, by eliminating prefix redundancies existing among cube tuples. PrefixCube is proposed to be an efficient cube structure by augmenting BST condensing and prefix-sharing. In this paper, we optimize the computation of PrefixCube through batch mode processing to reduce the computation time cost. Through extensive experiments, using both synthetic and real world dataset, the batch-mode computation of PrefixCube is proved to outperform the normal-mode computation on most of datasets.

Keywords　OLAP, PrefixCube, Cuboid, Base single tuple

1　引言

为了有效地支持决策支持系统的查询应用，Jim Gray 等人在1996年提出了数据立方(DATA CUBE)和立方算子(CUBE BY)的概念[1]，CUBE BY 是传统的 GROUP BY 算子的多维扩展，它计算 CUBE BY 子句中各属性的所有可能组合对应的 GROUP BY。一个 GROUP BY 操作对应一个数据小方(cuboid)。

CUBE BY 操作的代价十分昂贵，其计算的结果集也十分巨大，尤其当 CUBE BY 子句中的属性个数和基本关系表(简称基表)中的元组个数很多时，更是如此。假设存在一个 N 维的基表 R，则一个 k 维数据小方($0<=k<=N$)中元组的个数，是 R 在这 k 个维上所有不同属性值组合的个数。当 R 很稀疏时，一个数据小方的大小就接近于基表 R 的大小，因此，存储数据立方所需的 I/O 代价就变得十分重要。数据立方的巨大体积使得数据立方计算成为一种十分耗时的操作。

最近，几种减数据立方体积和数据立方计算代价、存储代价的方法被相继提出，它们包括浓缩数据立方(Condensed Cube)[8]，Dwarf[7]，Quotient Cube[6]和 QC-trees[9]。这几种方法的基本思想都是尽可能地消除存在于数据立方元组之间的冗余。元组之间通常存在两种冗余，在 Dwarf 一文中被明确地定义为前缀冗余(prefix redundancy)和后缀冗余(suffix redundancy)，在其他几种方法中也或多或少非正式地提到这两种冗余。

假设存在一个包含三个维 A,B,C 的数据立方，则维 A 上的每一个维值都会出现在数据小方(A)，(AB)，(AC)和(ABC)中，而且除了数据小方(A)，在其他几个小方中同一个维值还可能会出现多次。这种冗余就被称为前缀冗余。前缀冗余又可以进一步分为小方间的前缀冗余(inter-cuboid prefix redundancy)和小方内的前缀冗余(intra-cuboid prefix redundancy)。

当属于不同数据小方的元组实际上是由同一组基表元组计算得到时，就会产生后缀冗余。举一个较为特殊的例子，假设基表 R 仅包含一条元组 $r(a_1, a_2, \cdots, a_n, m)$，则由 R 计算得到的数据立方中包含 2^n 条元组，分别为 $(a_1, a_2, \cdots, a_n, V_1)$，$(a_1, *, \cdots, *, V_2)$，$(*, a_2, *, \cdots, *, V_3)$，$\cdots$，$(*, *, \cdots, *, V_m)$，其中 $m=2^n$，$*$ 代表 ALL 值。由于基表 R 中仅有一条元组，则我们可以得到 $V_1=V_2=\cdots\cdots=V_m=aggr(r)$。因此在这个数据立方中，我们实际上只需保存一条元组 $(a_1, a_2, \cdots, a_n, V)$，$V=aggr(r)$，再保存一些附加信息用以说明这条元组代表一组立方元组。针对该数据立方的任何查询，我们都可以直接返回 V 值，而不需要再做任何进一步的聚集计算。在这个例子中，R 对应的数据立方中的 2^n 条元组可以被浓缩成一条元组。总的来说，已知由同一组基表元组计算

*)本文研究得到国家自然科学基金(项目编号60303030)的资助。冯玉才　博士生导师，主要研究方向是数据库与多媒体。方　琼　硕士研究生，主要研究方向是数据仓库与 OLAP。李　曲　博士研究生，主要研究方向是数据仓库和数据挖掘。冯剑琳　博士，主要研究方向是数据仓库和数据挖掘。

聚集得到的立方元组，都可以被浓缩成一条元组。浓缩数据立方只是将那些由同一条基表元组计算聚集得到的立方元组浓缩成一条元组，而 Dwarf 等方法则扩展到浓缩那些由同一组基表元组计算聚集得到的立方元组。浓缩数据立方中这样的基表元组被称为基本单元组(Base Single Tuple，或 BST)，计算得到的数据立方也因此被称为基本单元组浓缩数据立方(BST Condensed Cube)。由于当基表很稀疏时，会存在很多这样的基本单元组，从而能够极大地减小数据立方的体积，也使得 Dwarf 实现的浓缩由同一组基表元组计算得到的立方元组，变得不是很有必要。

PrefixCube 是将基本单元组的浓缩和小方内的前缀共享结合，而提出的一种有效的数据立方组织结构。它将数据立方按数据小方聚簇，然后消除小方内的前缀冗余，从而有效地减小了数据立方的体积。PrefixCube 在数据立方压缩比例，数据立方恢复和更新代价以及数据立方查询特性上得到了很好的平衡[12]。从一方面看，它有效地减小了数据立方的体积，降低了数据立方的计算代价；但是从另一方面看，为了识别元组间的共享前缀，在计算 PrefixCube 的同时需要进行大量的元组间的比较，又消耗了更多数据立方计算的时间。由此，在本文中我们提出了以批处理模式优化计算 PrefixCube，从而在保证有效压缩数据立方体积的前提下，进一步降低了 PrefixCube 的计算时间代价。

本文在第2节中回顾了基本单元组浓缩机制，并说明如何将基本单元组的浓缩与批处理模式结合；然后在第3节中介绍了 PrefixCube 结构，以及用批处理模式构建 PrefixCube 的方法；在第4节中我们通过实验，分析了批处理模式对 PrefixCube 计算时间的影响；最后对全文进行了总结。

2 基本单元组浓缩数据立方

前文已经指出，浓缩数据立方将由同一条基本单元组计算聚集得到的立方元组浓缩成一条元组，并用这一条基本单元组来代表这一组被浓缩的立方元组。基本单元组的形式定义如下：

定义1(基本单元组，BST) 给定一组维属性 $SD \subset \{D_1, D_2, \cdots, D_n\}$，将基表在 SD 上进行划分，若在某一个分组中仅包含一条元组 r，则我们称 r 是 SD 上的基本单元组，SD 称为 r 的单值维集(the single dimension set)。

表1 一个基本关系 R

	A	B	C	M
t1	8	1	1	100
t2	1	8	1	50
t3	1	2	3	60

表1是一个3维的基表 R，包含三个维属性 A，B，C 和一个度量属性 M。基于定义1，我们可以看到元组 $t1$ 在 $\{A\}$ 上是基本单元组，因为将基表 R 在维 A 上做划分时，在分组 $A=8$ 中仅包含 $t1$ 这一条元组。元组 $t2$ 在 $\{A\}$ 上不是基本单元组，因为 $t2$ 和 $t3$ 都属于同一分组 $A=1$。而 $t2$ 在 $\{B\}$ 上则是基本单元组。

一条基表元组可以在不止一个维集上是基本单元组。如表1中，$t1$ 在维集 $\{A\}$ 和 $\{B\}$ 上都是基本单元组。如果我们发掘每一条基本单元组所有可能的单值维集，就可以得到唯一一个最小基本单元组浓缩数据立方(minimal BST condensed cube)。文[8]中提出了一种计算最小浓缩数据立方的算法 MinCube，但是这种算法的计算代价十分巨大，而且当我们从最小浓缩数据立方中恢复被浓缩的普通立方元组时，我们还必须检查从同一条基本单元组的不同单值维集上是否会恢复出同样的立方元组。因此，我们并不希望发现基本单元组所有的单值维集来得到最小浓缩数据立方，而是一条基本单元组仅对应一个单值维集并且仍然能够得到较高的浓缩比例。所以，我们选择 BU-BST 浓缩数据立方[8]作为我们的基础，它是由 BottomUpBST(简称 BU-BST)算法计算得到的。

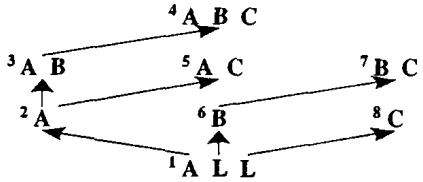


图1 BU-BST 处理树

BU-BST 算法按自底向上的顺序计算浓缩数据立方，如图1所示。图中的每个结点代表一个数据小方，结点包含的属性表示对应小方的聚集属性，结点边标注的数字指明 BU-BST 的计算顺序。BU-BST 算法基本上是 BUC 算法[2]的改进，我们又进一步改进了算法，将 BU-BST 计算与发现共享前缀结合，一次计算一组属于同一小方的、且共享某个前缀的元组，从而减少了为识别元组间的共享前缀所作的比较，即实现了元组的批量生成，。我们仍然称新改进的算法为 BU-BST，并且直接给出了新版本的 BU-BST 算法。从算法的描述我们可以看到，BU-BST 计算能够很自然地与元组的批量生成结合。

BU-BST 预先定义了维的计算序。为描述的简单起见，我们将这个维顺序映射成0到 *numDims*-1之间的整数，其中 *numDims* 表示维的个数。在计算过程中，我们用一个位图 *CID* 来表示当前计算的数据小方的聚集属性，若当前生成的元组是基本单元组，则 *CID* 表示该基本单元组的单值维集。*CID* 包含 *numDims* 个 bit 位，每一位对应数据立方的一个维。计算开始时，*CID* 的所有位都初始化为0，在计算过程中，若某一维变成聚集属性维，则将 *CID* 中的对应位置1。

算法1 BottomUpBST(input, dim, CID)

```
1: //Output normal tuples or BSTs in batches
   k=0; C=cardinality[dim];
2:for i = 0 ; i < C ; i++ do // For each partition
3:    c = dataCount[dim][i];
4:    if c == 1 then
5:       if dim == numDims - 1 then
6:          outputNaiveBST(input+k, dim);
7:       else
8:          outputBST(input+k, dim);
9:       end if
10:   else
11:      if c > 1 then
12:         outputNormalTuple(input+k, c, dim);
13:      end if
14:   end if
15:   k += c;
16:endfor
17://recursively process each partition
   k = 0; C = cardinality[dim];
18:for i = 0 ; i < C ; i++ do // For each partition
19:   c = dataCount[dim][i];
20:   if c > 1 then
21:      for d = dim+1 ; d < numDims ; d++ do
22:         C = cardinality[d];
23:         Partition(input, d, C, dataCount[d]);
24:         BottomUpBST(
               input[k…k+c], d+1, CID | (1<<d) );
25:      end for
26:   end if
27:   k += c;
28: end for
```

BU-BST 算法的具体描述见算法1。与以往的 BU-BST 算

法不同，对于一个输入分组 input，我们首先批量生成一组立方元组(行2～16)。注意，虽然这一组立方元组是在 dim 维上对 input 做划分得到的，但此处并不真地划分 input，而是利用保存在 datacount 中的 input 按维值排序的信息，从而得到 input 对应的一组立方元组。可以看到，这一组立方元组都属于数据小方 cuboid(*CID*)，并且它们共享了从0到 dim 维之间所有聚集属性维的维值。若某一分组中元组的个数为1，且已经划分到最后一维，由于该元组没有任何浓缩效果，则我们称这条元组为伪基本单元组(naive BST)，把它作为普通元组插入到 PrefixCube 中(行6)；若还未划分到最后一维，则我们找到一条基本单元组，将其直接插入到 PrefixCube 中(行8)，该基本单元组的单值维集即为 *CID*；若分组中元组的个数大于1，则生成了一条普通立方元组，也将其插入到 PrefixCube 中(行12)。在 *outputNaiveBST*，*outPutBST* 和 *outputNormalTuple* 中均调用了 packingPrefix 算法(见算法2)，以实现元组的批量拼装。可以看到，若对某个仅包含一条基本单元组的分组做进一步划分，得到的立方元组都会被浓缩到该基本单元组中，因此我们在下面递归调用 BottomUpBST 时加入了判断条件 $c>1$(行20)，从而不再处理那些仅包含1条元组的分组。我们称那些被浓缩到某条基本单元组中的立方元组为该基本单元组的祖先元组(ancestor tuple)。由于一条基本单元组的所有祖先元组都共享了基本单元组在单值维集上的前缀维值，因此基本单元组的浓缩实质上也是一种特殊的前缀共享。当批量生成了当前维对应的一组元组后，再对 *input* 做划分(行23)，并循环处理 *dim* 维对应的每个分组，每一个分组作为新的输入递归调用 *BottomUpBST*(第24行)。其中，*dataCount*[*dim*]保存了维 *dim* 的所有维值对应的元组个数(行3，行19)。当所有划分都处理完以后，我们再重复处理下一个维，直到计算完整个数据立方。

表2为由表1计算得到的 BST 浓缩数据立方。*SD* 表示每条基本单元组的单值维集，*CID* 表示每条普通立方元组所属的数据小方。

表2 BU-BST 浓缩数据立方

	A	B	C	M	SD	CID
ct1	*	*	*	210		ALL
ct2				110		A
ct3	8	1	1	100	A	
ct4				60	AB	
ct5				50	AB	
ct6			1	50		AC
ct7			3	60		AC
ct8	*	1	1	100	B	
ct9	*	2	3	60	B	
ct10	*	8	1	50	B	
ct11	*	*	1	150		C
ct12	*	*	3	60		C

3 共享前缀的浓缩数据立方

我们称由 BU-BST 算法计算得到的浓缩数据立方为 BU-BST 浓缩数据立方(BU-BST condensed cube)。BU-BST 浓缩数据立方具有以下特性：

1) 每一条基本单元组与且仅与一个单值维集相关联。

2) 一条基本单元组能且仅能浓缩某一部分数据小方中的元组，这些数据小方的聚集属性集都以该基本单元组的单值维集作为前缀。

3) 一条立方元组要么是一条普通立方元组，要么被浓缩到一条唯一的基本单元组中。

基于以上特点，一个 BU-BST 浓缩数据立方可以被分成两个互不相交的子立方：一个由所有的普通立方元组组成，称为普通子立方，另一个由所有的基本单元组组成，称为单元组子立方。

尽管 BU-BST 浓缩机制可以有效地减小数据立方的体积，在 BU-BST 浓缩数据立方中仍然存在冗余。如表2中，元组 *ct*4和 *ct*5，*ct*6和 *ct*7之间都分别存在小方内的前缀冗余；而元组 *ct*2，*ct*4和 *ct*6之间又存在小方间的前缀冗余。另外，BU-BST 浓缩的效果与提前给定的维计算顺序有很大关系。将维按基数从大到小排序时，BU-BST 浓缩机制的效果最好，因为这样最利于尽早发现尽可能多的基本单元组。所以文[12]中提出了一种 PrefixCube 结构，即共享前缀的浓缩数据立方。它在 BU-BST 浓缩的基础上消除小方内的前缀冗余，不仅能进一步减小数据立方的体积，而且使得 BU-BST 算法对基表数据的分布不再那么敏感。

3.1 PrefixCube

构建 PrefixCube 的中心思想是：将普通子立方中的元组按它们的聚集维集聚簇，将单元组子立方中的基本单元组按它们的单值维集聚簇，然后消除每个聚簇内的前缀冗余。具有相同聚集维集的普通立方元组构成的小方称为普通数据小方(normal cuboid)，具有相同单值维集的基本单元组构成的小方称为虚数据小方(virtual cuboid，或 v-cuboid)，虚数据小方的单值维集实际上与聚集维集的作用类似。对每个普通数据小方，我们分别建前缀树，称为 *N-prefixTree*；对每个虚数据小方，我们也类似地建前缀树，称为 *V-prefixTree*，所有的前缀树就构成了整个 PrefixCube。

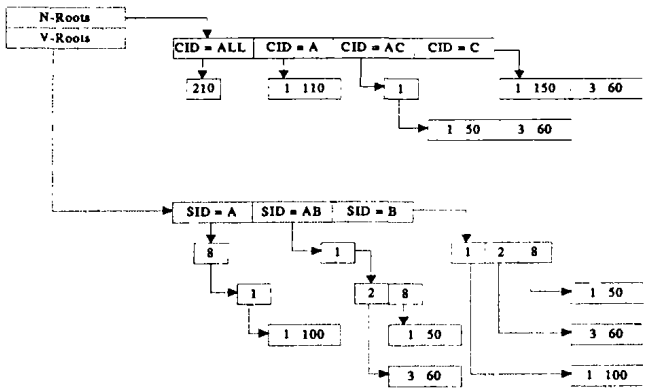


图2 PrefixCube 结构

图2是基表1对应的 PrefixCube 结构。PrefixCube 的头节点 *N-Roots* 指向一组 *N-prefixTree*，每一棵 *N-prefixTree* 对应一个普通数据小方；头节点 *V-Roots* 指向一组 *V-prefixTree*，每一棵 *V-prefixTree* 对应一个虚数据小方。除普通数据小方 cuboid(*CID*=*ALL*)外，*N-prefixTree* 的高度与其应普通数据小方的聚集属性个数相同；*V-prefixTree* 的高度则等于其对应的虚数据小方单值维集中维的个数加上单值维集的最后一个维之后的所有维的个数。前缀树的一层就对应一个维。

在每一棵前缀树中，非叶节点包含一组形如[维值，指针]的单元，一个单元对应相应维的一个维值。同一节点内的所有单元按维值大小升序排列。每个单元的指针指向下一层的一个节点，被指向的节点包含下一个维中所有与该单元维值相关联的维值。例如，在虚数据小方 *AB* 对应的前缀树 V-prefixTree(*SID*=*AB*)中，单元[1]指向包含单元[2]和[8]的节

点。前缀树的叶节点中包含一组形如[维值,聚集值]的单元。从一个数据小方对应前缀树的根节点扫描到某个叶节点的某一个单元,就可以得到属于该数据小方的一条元组,在叶节点对应的单元中保存了该元组最后一个聚集维的维值和度量值。如在 V-prefixTree($SID=AB$)中,沿路径{1,2}扫描到达叶节点的一个单元[3,60],表示在虚数据小方 v-cuboid($SID=AB$)中包含一条基本单元组(1, 2, 3, 60),即表2中的 $ct5$。

3.2 批处理模式计算 PrefixCube

在本节中我们将介绍如何以批处理模式计算 PrefixCube。由 BU-BST 算法的计算特性可以看到,一个包含 d 个分组维属性的分组,若在 $d+1$维上对其做进一步地划分,则在 $d+1$维上生成的一组立方元组一定满足如下两个条件:

1) 共享了前 d 个分组维属性上的维值。

2) 其中的普通立方元组一定属于同一普通数据小方,其中的基本单元组一定属于同一虚数据小方。

如表1,在{A}上对基表进行划分,则在分组 $A=1$中包含两条元组(1, 2, 3, 60)和(1, 8, 1, 50)。若将该分组在维 B 上做进一步的划分,则生成了两条基本单元组,即表2中的 $ct4$(1, 2, 3, 60)和 $ct5$(1, 8, 1, 50),这两条基本单元组在分组维属性 A 上具有相同的维值1,并且它们都属于虚数据小方 v-cuboid($SID=AB$)。因此,在拼装这样一组元组时,我们仅当插入第一条普通立方元组和第一条基本单元组时,才需要将它们分别与前一条插入的元组进行比较以识别共享前缀,对于该组中余下的普通立方元组和基本单元组,我们不需要做任何比较,只是标记它们的共享前缀,然后直接插入元组的后缀部分。因此,减少了组内元组之间的比较,加快了数据立方的计算速度。仍见上例中的基本单元组 $ct4$和 $ct5$,插入第一条基本单元组 $ct4$后,我们记住该组元组的共享前缀{1},然后直接插入余下元组 $ct5$除共享前缀以外的后缀部分(8, 1, 50),而无需任何比较操作。

另外,由于在 BU-BST 计算的过程中,每次处理一个分组之前,都会将该分组中的元组按维值的升序排序,这就使得属于同一小方(普通小方或是虚小方)中的元组是按序生成的。因此,给定一个属于某数据小方的立方元组 t,在所有属于该小方且在 t 之前生成的立方元组中,一定是最后一条生成的元组与 t 之间存在最长的共享前缀。将该想法稍作改进,由于同一组生成的元组具有相同的共享前缀,而且组间和组内的元组都是按序生成的,因此我们只需要将当前生成的一组元组中的第一条普通立方元组和第一条基本单元组,与对应数据小方中已经插入的最后一组元组中的第一条普通立方元组或基本单元组比较,即可找到该组元组与所有已经插入的元组之间的最长共享前缀。

下面我们会给出实现批处理模式的计算和拼装的算法 packingPrefix。packingPrefix 算法的具体描述见算法2。BU-BST 算法一次生成一组元组,这组元组满足3.2节中描述的两个条件。组内的每一条元组,作为一个输入调用算法 packingPrefix。给定一个输入元组 *tuple*,我们首先检查其对应的缓冲页 page[*cid*]是否为空(行1)。若页为空,说明 *tuple* 开始了一个新的共享前缀组(prefix group),将 *tuple* 设为当前共享前缀组的首元组(leading tuple),并将其相对于页首的偏移地址保存在 page[*cid*]的偏移地址槽(offset slot)中,同时记住 *tuple* 为最后插入的元组(*prevTuple*),作为以后判断共享前缀的比较标准,最后返回上层调用 BU-BST(行1~4)。若 page[*cid*]不为空,说明该缓冲页中已经拼装了一些元组,此时,若 *tuple* 是即将拼装的一组元组中的第一条普通立方元组或第一条基本单元组,则将 *tuple* 与 *prevTuple* 比较,得到它们之间的共享前缀 *prefix* 和 *tuple* 的后缀部分 *suffix*(行6)。若 *prefix* 不为空,且 page[*cid*]仍然有足够的空间可以装下 *suffix*,则将 *suffix* 拼装到 page[*cid*]中,返回(行9)。若 *prefix* 不为空,但 page[*cid*]没有足够的空间装下 *suffix*,则我们将当前缓冲页写到磁盘,并认为 *tuple* 在新的一页中开始了一个新的共享前缀组,将 *tuple* 设为当前共享前缀组的首元组,并将 *tuple* 保存到 *prevTuple* 中,返回(行13~17)。如果 *prefix* 为空,则表示当前共享前缀组结束,*tuple* 开始了一个新的共享前缀组。此时,若 page[*cid*]没有足够的空间装下 *tuple*,则先将 page[*cid*]写到磁盘(行13~14),然后重新利用该页,将 *tuple* 设为一个新的共享前缀组的首元组,且将 *tuple* 保存到 *prevTuple* 中,返回(行16~17)。若 *tuple* 不是当前拼装的一组元组中的第一条元组,则可以确定 *tuple* 一定属于当前共享前缀组,我们只需直接检查 page[*cid*]是否有足够的空间装下 *suffix*,若有,就直接将 *suffix* 拼装入该页中,返回(行19, 20);否则将 page[*cid*]写入磁盘,并且认为 *tuple* 在新的一页开始了一个新的共享前缀组,将 *tuple* 保存到 *prevTuple* 中,返回(行22~24)。需要说明的是,与 *tuple* 和 *suffix* 一同保存到 page[*cid*]中的还有一个 *size*,用于记录当前 *tuple* 或 *suffix* 的长度,以便于元组的恢复。

算法2 packingPrefix(*tuple*, *cid*)

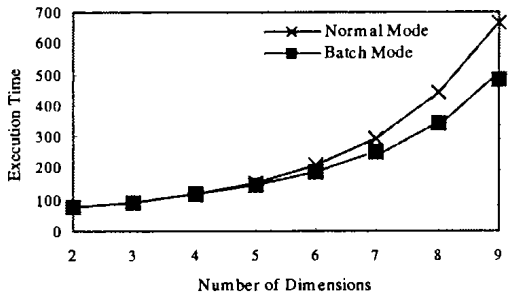
```
1:if page[cid] is empty then
2:     set tuple as the leading tuple of current prefix group;
3:     prevTuple= tuple; return;
4:end if
5:if is1stNormal or is1stBST then
6:     prefix=tuple & prevTuple; suffix=tuple - prefix;
7:     if prefix is non-empty then
8:          if page[cid] has room to hold suffix then
9:               append suffix to page[cid]; return;
10:         else flush page[cid];
11:         endif
12:    endif
13:    if page[cid] has no room to hold tuple then
14:         flush page[cid];
15:    endif
16:    set tuple as the leading tuple of a new prefix group;
17:    prevTuple = tuple; return;
18:else
19:    if page[cid] has room to hold suffix then
20:         append suffix to page[cid]; return;
21:    else
22:         flush page[cid];
23:         set tuple as the leading tuple of a new prefix group;
24:         prevTuple = tuple; return;
25:    endif
26:endif
```

还需要注意的是,在缓冲页的偏移地址槽中仅保存共享前缀组的首元组的偏移地址,因为一个共享前缀组中的所有元组,它们第一维的维值一定合并到了该组的首元组中。而由于查找一条立方元组 t 时,我们还需要从它之前的元组中恢复 t 被共享的前缀维值,因此我们可以首先找到 t 所属的共享前缀组,然后从首元组开始扫描,在查找 t 的同时逐个恢复其前缀维值。当 t 被找到,它所有的维值也都恢复了。

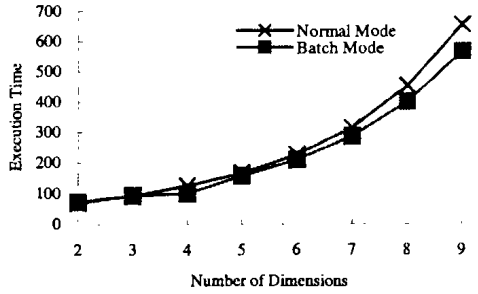
其次,如文[12]中提到的,packingPrefix 算法的瓶颈在于为了识别元组之间的共享前缀而需要进行大量的元组间的比较。而以批处理模式计算 PrefixCube 时,仅当元组 t 为当前要拼装的一组元组中的第一条普通立方元组或基本单元组时,才将 t 与 *prevTuple* 比较,判断它们之间是否存在共享前缀。否则,不需要进行任何元组间的比较,直接识别元组的前缀,并将其后缀部分拼装。另外,对于仅包含一个聚集维属性的数据小方,如 cuboid(A)和 v-cuboid(A)等,其中的元组之间不

存在共享前缀，所以我们也无需做任何比较。

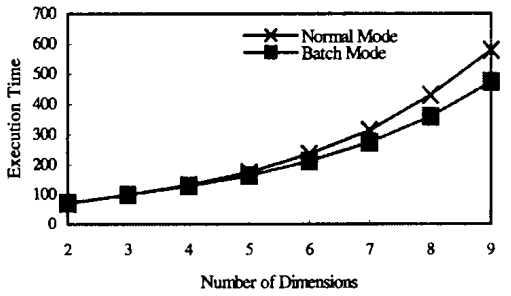
最后，当一条新插入的元组 t 到来，并且发现当前共享前缀组所在的缓冲页已满，我们将这一页写到磁盘，重置页的状态，然后不论元组 t 与先前插入的元组之间是否存在共享前缀，仍将 t 作为一个新的共享前缀组的首元组拼装到缓冲页中。也就是说，我们不考虑不同缓冲页之间的前缀共享，以保证每一个小方的每一页都是自包含的(self-contained)[3,4]。我们将前缀共享限制在页级别，因此一个共享前缀组最大不会超过一页；由于每一页都是自包含的，查找并恢复一条元组都可以在一页内实现，而且扫描范围至多不会超过一页。



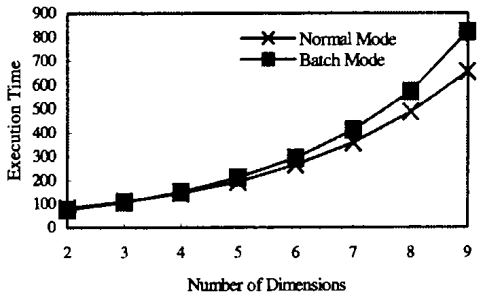
(a) 基数=50



(b) 基数=100



(c) 基数=200



(d) 基数=500

图3 均匀分布数据集

4 实验分析

我们通过比较一般模式下和批处理模下 PrefixCube 的计算时间，来分析批处理模式的引入对 PrefixCube 计算时间效率的影响。

实验的平台为：Pentium IV 1.8G Hz CPU，512MB 内存和80G IDE 硬盘。在实验过程中，我们使用了两组数据集，分别是模拟生成的均匀分布数据集和包含1985年9月不同气象站采集的天气数据集[11]。一般模式下 PrefixCube 的计算是按文[12]给出的算法实现的，这两种模式下计算得到的 PrefixCube 以同样的物理结构存储，并且缓冲页的大小均设为8kB，从而利于它们在计算时间上的比较。

4.1 均匀分布数据集

第一组实验采用均匀分布的模拟数据集，研究在一般模式下和批处理模式下 PrefixCube 的计算时间效率。我们首先固定每个数据集的元组数目为1M，然后从2到9变化 CUBE-BY 属性的个数，每个维的基数都设为一样，从而生成8个均匀分布的数据集。如图3所示，我们给出了四组实验结果，基数分别为50，100，200和500。从图中可以看到：

1) 无论基数大小，在维个数较少的数据集上(即维个数为2和3)，批处理模式下的计算速度和一般模式下的计算速度相当。这是因为数据集的维个数较少，则生成的立方元组的维数也相应的较少，所以即使在一般模式下逐个进行元组间的比较，也不会花费太多的计算时间。

2) 在基数为50，100和200这三组数据集的实验中，随着数据集维个数的增多，批处理模式下的计算速度逐渐优于一般模式下的计算速度。原因是随着维个数的增多，用于元组间的比较所花费的计算代价就越来越大，从而使得元组的批量生成逐渐体现出较为明显的优势。在9个维的数据集上，当基数分别为50，100和200时，批处理模式下 PrefixCube 的计算速度比一般模式下 PrefixCube 的计算速度分别快180秒，88秒和106秒。

3) 在基数为500的数据集的实验中，随着数据集维个数的增多，批处理模式下的计算速度反而劣于一般模式下的计算速度。原因是数据集元组的个数不变，随着维基数的不断增大，数据集相对地就越来越稀疏。因此在对数据集进行划分时，每个分组中包含的元组数目就会减少，而一个分组中元组数目的多少直接决定了有多少条属于同一小方的立方元组共享了某个前缀，也即决定了可同一批生成的元组数目。当同一批生成的元组数目较少时，批量生成并不能节省更多的计算时间，反而会因为要对每个分组多进行一次扫描而花费了更多的时间。

可以看到，与 PrefixCube 的一般计算模式相比，PrefixCube 的批处理计算模式在不太稀疏的数据集上取得了较为明显的优势，并提高了 PrefixCube 的计算时间效率；而随着数据集越来越稀疏，批处理模式的性能逐渐减小，在数据集达到某个稀疏程度时，批处理模式的性能劣于一般模式。

4.2 真实数据集

这一组实验采用了真实的气象数据集。气象数据集包含1,015,367条元组，其维属性依次为：station-id(7037)，longitude(352)，solar-altitude(179)，latitude(152)，present-weather(101)，day(30)，weather-change-code(10)，hour(8)和 brightness(2)。我们分别投影头 k 个维($2 <= k <= 9$)，从而生成了8个数据集，数据集的维个数依次从2到9变化。图4是在不同的数据集上 PrefixCube 的计算时间结果。

(下转第96页)

9 Beddow J. Shape Coding of Multidimensional Data on a Mircocomputer Display. In: Visualization '90, SanFrancisco, CA, 1990. 238～246
10 Levkowitz H. Color icons: Merging color and texture perception for integrated visualization of multipleparameters. In: Visualization '91, San Diego, CA, Oct. 1991
11 Keim D A, Kriegel H-P. VisDB: Database Exploration using Multidimensional Visualization. ComputerGraphics & Applications, Sept. 1994. 40～49
12 Keim D A, Kriegel H-P, Ankerst M. Recursive Pattern: A Technique for Visualizing Very Large Amounts of Data. In: Proc. Visualization '95, Atlanta, GA, 1995. 279～286
13 Ankerst M, Keim D A, Kriegel H P. Circle Segments: A Technique for Visually Exploring Large Multidimensional Data Set. In: Proc. Visualization '96, 1996
14 LeBlanc J, Ward M O, Wittels N. Exploring N-Dimensional Databases. In: Visualization '90, San Francisco, CA, 1990. 230～239
15 Johnson B. Visualizing Hierarchical and Categorical Data: [Ph. D. Thesis]. Department of Computer Science, University of Maryland, 1993
16 Robertson G G, Mackinlay J D, Card S K. Cone Trees: Animated 3D Visualizations of HierarchicalInformation. In: Proc. Human Factors in Computing Systems CHI '91 Conf., New Orleans, LA, 1991. 189～194
17 Battista G D, Eades P, Tamassia R, Tollis I. Annotated Bibliography on Graph Drawing Algorithms. Computational Geometry: Theory and Applications, 1994, 4: 235～282
18 Buja A, Swayne D F, Cook D. Interactive High-Dimensional Data Visualization. Journal of Computationaland Graphical Statistics, 1996, 5(1): 78～99
19 Fishkin K, Stone M C. Enhanced Dynamic Queries via Movable Filters. In: Proc. Human Factors in ComputingSystems CHI '95 Conf., Denver, CO, 1995. 415～420
20 Ahlberg C, Wistrand E. IVEE: An Information Visualization and Exploration Environment. In: Proc. Int. Symp. onInformation Visualization, Atlanta, GA, 1995. 66～73
21 Wilhelm A, Unwin A R, Theus M. Software for Interactive Statistical Graphics - A Review. In: Proc. Int. Softstat '95 Conf., Heidelberg, Germany, 1995
22 Leung Y, Apperley M. A Review and Taxonomy of Distortion-oriented Presentation Techniques. In: Proc. HumanFactors in Computing Systems CHI '94 Conf., Boston, MA, 1994. 126～160
23 Mackinlay J D, Robertson G G, Card S K. The Perspective Wall: Detail and Context Smoothly Integrated. In: Proc. Human Factors in Computing Systems CHI '91 Conf., New Orleans, LA, 1991. 173～179
24 Apperley M, Spence I T. A Bifocal Display Technique for Data Presentation. In: Proc. Eurographics, 1982. 27～43
25 Rao R, Card S K. The Table Lens: Merging Graphical and Symbolic Representation in an InteractiveFocus + Context Visualization for Tabular Information. In: Proc. Human Factors in Computing Systems CHI '94Conf., Boston, MA, 1994. 318～322
26 Sarkar M, Brown M. Graphical Fisheye Views. Communications of the ACM, 1994, 37(12): 73～84
27 Munzner T, Burchard P. Visualizing the Structure of the World Wide Web in 3D Hyperbolic Space. In: Proc. VRML'95 Symp, San Diego, CA, 1995. 33～38
28 Alpern B, Carter L. Hyperbox. In: Proc. Visualization '91, San Diego, CA, 1991. 133～139
29 Keim D A. Designing Pixel-Oriented Visualization Techniques: Theory and Applications. IEEE trans. on visualization and computer graphics. 2000
30 Abello J, Korn J. MGV: A system for visualizing massive multidigraphs. Transactions onVisualization and Computer Graphics, 2001
31 Kreuseler M, Lopez N, Schumann H. A Scalable Framework for information Visualization. In: Proc. InfoVis'2000, Salt Lake City, 2000. 27～36
32 Stasko J, Zhang E. Focus + Context Display and Navigation Techniques for Enhancing Radial, Space-Filling Hierarchy Visualizations. In: Proc. IEEE Information Visualization 2000, Salt Lake City, UT, Oct. 2000. 57～65

(上接第85页)

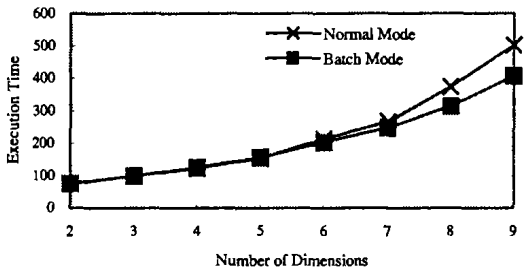


图4 气象数据集

从图中可以看到，在维个数较少的数据集上，一般模式下和批处理模式下计算 PrefixCube 所用的时间相当，随着维个数的增加，批处理模式计算逐渐优于一般模式下的计算。当数据集的维个数为9时，一般模式计算 PrefixCube 需要501秒，而批处理模式计算 PrefixCube 仅需要405秒，比前者少了96秒。这是因为，天气数据集是按维基数从大到小排序的，则在头几维上做划分时，由于维的基数较大，分组中元组的个数就相对较少，因此成批生成的元组个数就少，从而并不能给计算时间的加快带来多少好处；而随着数据集的维个数逐渐增多，由于后几维的基数较小，则在其上做划分后分组中元组的个数要多一些，成批生成的元组个数也会多一些，此时才体现出了批处理模式的优势，降低了 PrefixCube 计算的时间代价。

结论 在文[12]中提出的 PrefixCube 结构，不仅能有效减小数据立方的尺寸大小，而且在立方尺寸压缩比例，立方恢复和更新，以及立方查询这三者之间得到了很好的平衡。可是在计算 PrefixCube 时，为了识别元组间的共享前缀而不得不进行大量的元组之间的比较，因此降低了 PrefixCube 的计算时间效率。在本文中，我们提出了以批处理模式计算 PrefixCube 的方法，实现了批量生成一组属于同一小方、且共享某个前缀的元组，从而减少了组内元组之间的比较，有效地缩短了 PrefixCube 的计算时间。我们的实验结果表明，在大多数数据集上，批处理模式的引入都能有效地降低 PrefixCube 的计算时间代价。在真实气象数据集的实验中，在最大的9维气象数据集上，批处理模式计算 PrefixCube 所需的时间比一般模式下计算 PrefixCube 所需的时间要少得多。

参考文献

1 Gray J, Bosworth A, Layman A, Pirahesh H. Data Cube: A Relational Operator Generalizing Group-By, Cross-Tab, and Sub-Totals. In: Proc. of the Int. Conf. on Data Engineering, 1996
2 Beyer K, Ramakrishnan R. Bottom-Up Computation of Sparse and Iceberg CUBEs. In: Proc. of the ACM SIGMOD Int. Conf. on Management of Data, 1999
3 Poess M, Potapov D. Data Compression in Oracle. In: Proc. of Int. Conf. on Very Large Data Bases, 2003
4 Table Compression in Oracle9i Release 2: A Performance Analysis. Oracle Whitepaper, 2003
5 Gray J, Sundaresan P, Englert S, Baclawski K, Weinberger P J. Quickly Generating Billion-Record Synthetic Databases. In: Proc. of the ACM SIGMOD Int. Conf. on Management of Data, 1994
6 Lakshmanan L V S, Pei J, Han J. Quotient Cube: How to Summarize the Semantics of a Data Cube. In: Proc. of Int. Conf. on Very Large Data Bases, 2002
7 Sismanis Y, Deligiannakis A, Roussopoulos N, Kotidis Y. Dwarf: Shrinkingthe PetaCube. In: Proc. of the ACM SIGMOD Int. Conf. on Management of Data, 2002
8 Wang W, Feng J, Lu H, Yu J X. Condensed Cube: An Effective Approach to Reducing Data Cube Size. In: Proc. of the Int. Conf. on Data Engineering, 2002
9 Lakshmanan L V S, Pei J, Zhao Y. QC-Trees: An Efficient Summary Structure for Semantic OLAP. In: Proc. of the ACM SIGMOD Int. Conf. on Management of Data, 2003
10 Feng J, Si H, Feng Y. Indexing and Incremental Updating Condensed Data Cube. In: Proc of the Int. Conf. on Scientific and Statistical Database Management, 2003
11 Hahn C, Warren S, London J. Edited synoptic coud report from ships and land stations over the globe. http://cdiac.esd.ornl.gov/cdiac/ndps/ndp026b.html. http://cdiac.esd.ornl.gov/ftp/ndp026b/SEP85L.DAT.Z
12 Feng J, Fang Q, Ding H. PrefixCube: Prefix-sharing Condensed Data Cube. To appear: ACM International Workshop on Data Warehousing and OLAP, 2004