CSA-Tree:一种改进的高维主存索引树

梁俊杰[1),2)] 冯玉才[1)]

[1)](华中科技大学计算机学院 武汉 430074)

[2)](湖北大学数学与计算机科学学院 武汉 430062)

摘 要 主存技术的不断进步,使得主存多媒体数据库的实现成为可能.研究表明,主存多媒体数据库系统性能深受处理器缓存未命中的影响,缓存感知型主存索引是提高数据检索效率的有效手段.针对SA-Tree不适用于主存存取的缺点,提出它的变体CSA-Tree.CSA-Tree利用PCA降维技术,将树的各层节点采用不同的维度表示,这样不仅提高了缓存空间的利用率,还降低了CPU负载,从而提高了索引查询效率.大量实验证明,CSA-Tree在主存环境中具有良好的高维数据检索性能.

关键词 高维主存索引;L2-cache未命中;距离计算;KNN查询;主成分分析

中图法分类号 TP311

CSA-Tree: An Optimized High-Dimensional Index Tree for Main Memory Access

LIANG Jun-Jie[1),2)] FENG Yu-Cai[1)]

[1)](*College of Computer Science & Technology, Huazhong University of Science and Technology, Wuhan* 430074)

[2)](*Faculty of Mathematics & Computer Science, Hubei University, Wuhan* 430062)

Abstract In main-memory databases, the number of processor cache misses has a critical impact on the performance of the system. Cache-conscious indices are designed to improve performance by reducing the number of processor cache misses that are incurred during a search operation. Considering the disadvantage of SA-Tree inefficient for main memory access, the authors present its variant called CSA-Tree, which is a multi-level structure where each level of the tree represents the data space at different dimensionalities by using Principal Component Analysis. Each level of the tree serves to prune the search space more efficiently as the reduced dimensions can better exploit the small cache line size. Moreover, the distance computation on lower dimensionality is less expensive. Extensive experiments show that the CSA-Tree is superior in most cases compared with other methods.

Keywords high-dimensional main memory index; L2-cache misses; distance computation; K-Nearest Neighbor queries; principal component analysis

1 引 言

高维数据在很多领域得到了广泛应用,且这些应用大都要求搜索相似性对象,KNN(K-Nearest Neighbor)查询是其中常见的一种查询方式.提高高维数据检索效率常用的方法是创建索引,目前已有很多学者对高维索引提出了很多解决方法[1-3].例如:SA-Tree(Spatial Approximation Tree)就是目前普遍采用的一种高维索引树,它是根据Voronoi

收稿日期:2005-02-019;修改稿收到日期:2006-09-15.本课题得到国家"八六三"高技术研究发展计划项目基金([2005]555)资助.**梁俊杰**,女,1974年生,博士研究生,主要研究方向为多媒体数据库、基于内容的图像检索、高维索引技术等.E-mail:ljjhubu@smail.hust.edu.cn.**冯玉才**,男,1946年生,教授,博士生导师,主要研究领域为数据库、多媒体技术、GIS等.

图思想设计的一种适用于 KNN 查询的动态多级树[4-6].然而,它们都是针对常驻磁盘设计的.随着 RAM 技术的不断发展,主存高维索引的实现日益成为可能.为此,人们提出高维主存索引(High-dimensional Main Memory Index)[7-9].由于高维主存索引的搜索代价很大程度上是由 L2-cache 未命中(L2-cache Misses)引起[10],所以,现有的高维主存索引都设计成高速缓存感知型(Cache Concious)[8-9,11].但是,这些索引大多适用于静态数据,或者适用于低维数据,使得在维数大于 20 的动态环境下,索引的搜索效率大大降低,主要原因是没有充分利用缓存空间.

为了克服上述缺点,本文提出 SA-Tree 的缓存感知型变体 CSA-Tree(Cache-conscious Spatial Approxi- mation Tree).CSA-Tree 通过数据降维处理,不仅可以充分利用有限的缓存空间,还降低了距离计算复杂度,从而提高了索引查询效率.CSA-Tree 是一个多级树,树的每层节点都是采用不同的维度来表示的:从树根到树叶,节点的维度呈递增形式,叶节点的维度为最大(全维).树的层数和每层节点的维度是通过主成分分析方法 PCA(Principal Component Analysis)确定的.文中给出了 CSA-Tree 的构造算法以及 KNN 查询算法.通过将本文提出的方法和其它高维索引方法进行实验分析比较,发现 CSA-tree 的很多性能指标都是较好的.

本文第 2 节介绍背景知识;第 3 节提出 CSA-Tree 的构造算法和检索算法,并进行详细的算法分析;第 4 节提供实验设计和分析;最后是对本文工作的总结.

2 背景知识

2.1 主成分分析法 PCA(Principal Component Analysis)

PCA[12]是人们普遍采用的一种数据降维方法,它可以将高维空间的点映射到低维空间中,即用数据集的少数几个综合特征(维)来描述源特征(维)[13-14].采用 PCA 方法对数据降维的最大好处是,主成分能够最大限度地保留原始数据集的变化,均方误差(meansquare error)可由系统控制.这就为建立在降维空间中的索引结构的正确性(即通过索引检索的结果和在源空间中的检索结果是一致的)提供了有力保障.

PCA 性质如下.

性质 1. 假设 p',q' 为 S_d 空间的两个点,p_{k1},p_{k2} 是 p 分别转换到 PCA_{k1} 和 PCA_{k2} 空间的点.类似地,定义 q_{k1},q_{k2}.若 $0<k1<k2\leq d$,则 $Dist(p_{k1},q_{k1})\leq Dist(p_{k2},q_{k2})$.

性质 2. 由于前 k 个主成分最能体现数据集特征,因此 $Dist(p_d,q_d)\approx Dist(p_k,q_k)$,即使 $k\ll d$.

文献[12-14]给出了以上两个性质的详细说明.在文献[14]中,PCA 用于数据聚类以及为每类寻找最优维数.我们使用 PCA 方法是为了构造索引树结构,为每级树节点设计不同的维度,最终实现检索快速剪枝,减少距离计算和 L2-cache 未命中.

为叙述方便,现将本文中出现的符号及其描述的含义归纳如表 1 所示.

表 1 本文用到的符号及其含义

符号	含义
$S(n,d)$ 或 S_d	源数据空间,n:数据点个数,d:维数
$PCA(n,k)$ 或 PCA_k	PCA 空间,n:数据点个数,k:维数($1\leq k\leq d$)
$p'(\boldsymbol{x}_1,\boldsymbol{x}_2,\cdots,\boldsymbol{x}_d)$	源空间点向量,$\boldsymbol{x}_1$:第一维向量,…,$\boldsymbol{x}_d$:第 d 维向量
$p(\boldsymbol{y}_1,\boldsymbol{y}_2,\cdots,\boldsymbol{y}_k)$	PCA 空间点向量,$\boldsymbol{y}_1$:第一维向量,…,$\boldsymbol{y}_k$:第 k 维向量
$Dist(p,q)$	p,q 两点间的欧氏距离
$N(p)$	p 的邻居点集合
L	树的层数(树高)≥1
m_i	第 i 层树节点的维度($1\leq i\leq L$)
p_{m_i}	第 i 层节点 p
PCs	主成分集
$\boldsymbol{m}$	维度向量 $\boldsymbol{m}=(\boldsymbol{m}_1,\boldsymbol{m}_2,\cdots,\boldsymbol{m}_{L-1})$
Root(或 r)	根节点
TreeNode	树节点(除根以外)
Entry 或 Point	点数据
q	查询点
K	KNN 查询结果集的点个数
Q_i	前 i 个主成分的累积贡献率

2.2 SA-Tree

SA-Tree(Spatial Approximation Tree)[4]的构造思想来源于 Voronoi 图,它是一种广泛用于最近邻搜索的方法.在 Voronoi 图中,点 p 的 Voronoi 胞腔定义为距离 p 最近的区域.这样,对于某个查询点 q,最近邻搜索只需要判断 q 在哪个 Voronoi 胞腔内.利用 Voronoi 图,将空间每个点作为一个节点,在两个具有公共胞腔边框的点之间画一条边,这样可以构成一个图(Delaunay graph)表示 Voronoi 图的邻居关系.在 Delaunay 图中,搜索 q 点的最近邻可以从图中任意点开始,在该点的邻居点中沿着最接近 q 的方向搜索,一旦发现某个点 p 的所有邻居距离 q 都比 p 距离 q 还要远,这时 p 即为 q 的最近邻点,搜索停止.这种搜索方法是利用了 Voronoi 性质,并且凡是在满足该性质的图中搜索最近邻点都可以采用这种搜索方法[2].

为了近似 Delaunay 图，SA-Tree 以空间某个点为根节点 r，寻找空间分布与它最近的节点（即邻居 $N(r)$）作为它的子节点，这些子节点就组成树的下一层节点. 照此方法，依次构造树的每一层节点，这样整个索引树就建立起来了. 它的优点在于：在维数较高（$d>20$）的度量空间（metric space）中，即使搜索范围较大，仍然具有较高的搜索效率.

3　CSA-Tree

维数灾难（dimensionality curse）是阻碍高维数据搜索效率提高的一个主要瓶颈. 在基于主存的高维索引结构中，高维又带来了新问题：一个 L2 高速缓存栈（L2-cache line）可能无法容纳一个高维点数据. 为了使 CSA-Tree 适用于主存存取环境，我们应用 PCA 方法对原始数据进行降维，为索引树的各级节点选择不同的维度表示，这样做的直接好处是可以大大降低索引树的大小，使之有可能完全或部分存储在缓存中，从而提高 L2-cache 命中率.

3.1　CSA-Tree 结构

CSA-Tree 的提出来自于对以下三个方面的考虑：(1)降维是解决维数灾难的主要技术手段. 特别是，通过降维有可能将一个或多个高维空间的点数据存储在一个 L2 高速缓存栈中. (2)如何确定降维后的空间维数大小是一个非常棘手的问题. 特别是，在不降低搜索精度前提下，如何确定降维后的空间维数. (3)对以上两个问题，PCA 方法提供了较好的解决办法，因为 PCA 方法提取出来的前几个主成分能够很好地反映源数据集的绝大部分信息，并且上节我们已讨论过 PCA 方法有很多很好的性质. 所以，在建立索引之前，我们先用 PCA 方法对数据集进行降维处理.

定义 1.　假设 $p(x_1,x_2,\cdots,x_k)$ 为 PCA 空间中任意一点，定义 $\prod(p,k')$ 为将 p 映射到前 $k'(1\leq k'\leq k)$ 维主成分的操作符，即

$$\prod(p,k')=\prod((x_1,x_2,\cdots,x_k),k')=(x_1,x_2,\cdots,x_{k'}).$$

CSA-Tree 和 SA-Tree 有很多相同之处，它们都是多层树结构. CSA-Tree 的构造方法在 3.2 节详细介绍. 但是，不同于 SA-Tree 的所有节点采用全维的表示形式，CSA-Tree 树的每层节点的维度都不相同，越靠近根节点的维度越小，而叶节点的维度最大（全维），即若 $L_i<L_j$，则 $m_i<m_j$. 下面我们将讨论如何确定 CSA-Tree 树的层数 L 以及每层树节点的维度 $m_i(1\leq i\leq L)$.

定义 2.　CSA-Tree 节点表示形式为

$$p_{m_i}=\prod(p,m_i)=\prod((x_1,x_2,\cdots,x_k),m_i),\quad 1\leq i\leq L.$$

定义 3.　给定点集合 S 和点 $a\in S$，则点 a 的邻居 $N(a)=\{S'\subseteq S\mid \forall p\in S',\forall v\in S'-\{p\}:Dist(p,a)<Dist(p,v)\}$.

CSA-Tree 是根据 PCA_k 空间点的相似性构造的. 由图 1 可以看出，每个树节点（TreeNode）中存放一个或多个节点数据（Entry），而每个节点的子节点又是由其邻居组成. 根据这一思想，在构造 CSA-Tree 时，首先，从 PCA_k 中选出一个点作为根节点数据，记为 r_{m_1}（定义 2）；然后，找出 r 的邻居 $N(r)=\{r_1,r_2,\cdots,r_t\}$，并将它们作为 r 的儿子加入树的第二层；依次下去，直到所有点都加入树中，整个索引树就构造完成. 值得提醒的是，在构造 CSA-Tree 过程中，每层节点的距离计算都是基于该层维度基础上的，即点 p_{m_i} 的儿子是针对 $PCA_{m_{i+1}}$ 而言的.

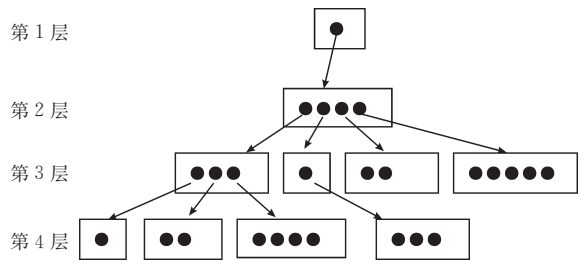

图 1　CSA-Tree 结构

接下来，我们具体讨论节点数据（Entry）的结构. 每个节点都是一个 4 元组 (p_{m_i},R,num,ptr)，其中 p_{m_i} 表示点特征向量，R 表示该点的最大覆盖半径（节点与其子节点的最大距离），num 表示子节点的个数，ptr 是指向第一个子节点的指针. 为什么只存储指向第一个子节点的指针？这是因为每个子节点存储的内容基本相似，所以我们可以将每个父节点的所有子节点存储在一片连续的存储空间中，并且为每个子节点分配相同大小的存储单元，这样对于每个父节点来说，根据第一个子节点的地址以及偏移地址就可以确定其它子节点的位置，因此只需保存第一个子节点地址即可，这也是为了节省存储空间. 另外，CSA-Tree 的根节点 Root 除包含点数据 $r(r_{m_1},R,num,ptr)$ 以外，还需要存储一些和整棵树有关的信息：树的层数 L；反映每层节点维度的向量 $\boldsymbol{m}=(m_1,m_2,\cdots,m_{L-1})$；用于提取主成分向量的特征矩阵 ***Eigenmatrix***. 由于叶节点的维数 $m_L=d$，

所以 m 中不包含叶节点的维度.

定义 4. 假设 PCs 对应的特征值为 $\lambda=(\lambda_1,\lambda_2,\cdots,\lambda_d)$,则 CSA-Tree 前 l 层维度的累积贡献率为[12] $Q_l=\sum_{i=1}^{l}\lambda_i\Big/\sum_{j=1}^{d}\lambda_j$.

要想构建 CSA-Tree,关键问题是如何确定树的层数和每层节点的维度.这里,我们假设树中每个节点的扇出值相同,具体确定方法可以参考我们的实验,例如对于 64 维数据,实验结果显示最佳叶节点大小为 2～3K,因此每个叶节点的最大扇出值为 8～12.这也是树中非叶子节点的最大扇出值,而这些节点的大小可以根据它们的维度适当地做些调整.下面我们给出一个简单的算法确定 L 和 m 的值.

(1)确定树层数 L.假设树节点的最大扇出值为 f;根据点的个数 n 和 f 确定 CSA-Tree 的层数 $L=\lceil\log_f n\rceil$.

(2)确定维度向量 $\boldsymbol{m}$.根据定义 4,我们可以为每层确定一个累积贡献率阈值 $Q_l^*=(1-\Delta)\times l/L+\Delta$, $1\leq l\leq L$,则满足 $Q_k\geq Q_l^*$ 的最小 k 值就是第 l 层的维度 m_l,其中 Δ 是累积贡献率的起步值.上述公式是根据我们的实验结果分析得来的,我们的实验数据是从网上 UCI KDD ATCHIVE 图像库提取的颜色特征值,根据这些图像的颜色特征计算它们的主成分,发现第一主成分的累积贡献率都会超过 70%,为了使各层的维度分布较均匀,我们设定每层的维度贡献率至少为 70%(即 $\Delta=0.7$),这样的维度表示才有意义.当然,根据具体应用领域,可以修改该参数值,使得索引树的各层维度分布尽可能均匀并足以反映整个数据集的变化幅度,以此来保证一定的查询精度.

表 2 CSA-Tree 维度分布及累积贡献率($n=1000$,$\Delta=0.7$)

数据集	真实数据集		均匀数据集	
	m_l	Q_l/%	m_l	Q_l/%
$l=1$	2	85.53	48	75.00
$l=2$	2		51	79.69
$l=3$	2		54	84.38
$l=4$	3	91.90	56	87.50
$l=5$	3		59	92.19
$l=6$	5	96.66	62	96.88
$l=7$	64	100	64	100

我们用一个实验来说明上述算法的可行性.该实验使用了两种数据集,一种是从 UCI KDD ATCHIVE 提取的 1000 幅图像颜色直方图真实数据,另一种是均匀分布的人造数据,并且都是 64 维的数据集.从表 2 的 PCs 的累积贡献率可以清楚地看出,真实数据集的前几维 PCs 就足以能体现数据集的整个变化幅度了,如:前 2 维反映了数据集 85%的变化幅度.但是,由于均匀分布的数据集的变化也是均匀分布的,所以无法只使用前几维来反映整个数据集的变化情况,而前 48 维才反映了数据集 75%的变化幅度.假设我们设计的 CSA-Tree 的层数 $L=7$,则以上两种数据集对应的 CSA-Tree 每层的维度及其累积贡献率如表 2 所示.

3.2 CSA-Tree 构造

对于一个给定的数据集,我们采用自顶向下的方式构造它的 CSA-Tree 索引树.首先,将 S_d 的数据点映射到 PCA_k,确定特征矩阵 ***Eigenmatrix*** 和维度向量 $\boldsymbol{m}$.然后,选出根节点 r,初始化 r 的 ***Eigenmatrix***、$\boldsymbol{m}$ 和 L 参数.最后,在 PCA_{m_2} 空间搜索 r 的邻居点 $N(r)$,将这些点作为 r 的子节点加入树中,这就是 CSA-Tree 第二层节点.依此类推,建立树的下一层节点,直到所有节点都加入树中为止.算法描述如下:

Algorithm BuildTree(dataset S)

Input: a high-dimensional dataset

Output: CSA-tree

1. Transform S into PCA space.
2. Select the centroid of S as root r, denoting a.
   Initialize a with ***Eigenmatrix***, L and vector $\boldsymbol{m}=(m_1,m_2,\cdots,m_{L-1})$.
3. Let the level of the tree is l, for each $a_i\in N(a)$, project a_i on m_l dimensions and add a_i as a child of a.
4. For each $a_i\in N(a)$, replace a with a_i. Repeat from step 3.

不难发现,CSA-Tree 并不是一棵平衡树,这是因为数据集分布不均匀造成的.在文献[4]中实验说明了数据维度对索引树的高度、树叶平均深度和节点最大扇出值的影响要比数据点个数带来的影响大.在数据集大小相同的情况下,维度越高,构造的索引树就越低(层数少),每个节点(TreeNode)的扇出值越大.但是,高维情况下节点的扇出值越大,存储节点所需的空间也就越大,这会增加 KNN 搜索时 L2-cache 未命中概率.因此,经过 PCA 降维处理后构造的 CSA-Tree 虽然不是一个平衡树,但是它可以降低每个节点的子节点个数(扇出值),又因为降维后数据表示非常简单,大大缩小节点存储空间,所以 CSA-Tree 更适合于主存环境.

为了尽可能降低 CSA-Tree 的高度，我们在设计 CSA-Tree 时采用了特殊的处理. 不同于根节点是从数据空间中任意选取的一个点[4-6]，CSA-Tree 选择空间点分布的几何中心作为根节点，为了简化计算，只需在前几维 PCA 空间统计特征均值向量，再找到距离该值最近的点，以该点作为 CSA-Tree 的根节点. 这样构造的树结构，在点个数一定的情况下，可以增大树的宽度，使得各个子树的高度差尽量小，不会给树的搜索效率带来太大的影响.

3.3　KNN 查询算法

根据 CSA-Tree 的构造可以看出，在 KNN 查询过程中，能实现快速剪枝，引导搜索至最佳路径. 回顾 PCA 性质 1，不难得出此结论，由于低维空间中两点间的距离要小于它们在高维空间中的距离，所以通过在低维空间中将不满足条件的分支剪去，就不必搜索相应的高维空间了. 具体来说，KNN 查询时，从 CSA-Tree 根节点开始，先在上层节点（低维空间）中搜索满足条件的节点，对于不满足条件的节点 $p(Dist(q,p)>R(p)+MaxDist(KNNList))$，可以直接将 p 的后续分支剪去，即不必搜索 p 的子节点. 这是因为，若在低维空间 p 与 q 间的距离已超出搜索范围的话，则它们在高维空间中的距离值将会更大，即更不可能存在满足条件的点（注：搜索半径不随维度变化而变化[3,15]）. 更重要的是，降维使得开始阶段搜索的节点的维度大大降低，这样不仅减少距离计算的复杂度，还能充分利用 L2-cache 空间以减少缓存未命中次数，这些正是主存高维索引希望达到的目标.

CSA-Tree 的 KNN 查询根据节点的最大覆盖半径 R 确定儿子分支是否需要检索，以及覆盖半径内与查询节点的最小距离确定是否需要剪去儿子节点的下一个分支，所以它的剪枝速度很快. 算法如下.

```
Algorithm KNN-search (Root r, Query q, K)
KNNList ← NewList(K)
PriorityQueue ← NewPriorityQueue()
                //Order by the distance to q (closer first)
PriorityQueue ← {r}
While not EMPTY(PriorityQueue) and MinDist(Pri-
      orityQueue) < MaxDist(KNNList) do
a ← the first element of PriorityQueue
If Dist(a,q) < MaxDist(KNNList) then
   Remove the element with largest distance from
   KNNList
   Insert(KNNList,a,d(a,q))
If Dist(a,q) ≤ R(a) + MaxDist(KNNList) then
   d_min ← min_{c∈{a}∪N(a)} {Dist(c,q)}
   for each b∈N(a) do
      if Dist(b,q) ≤ d_min + 2 * MaxDist(KNNList) then
         KNN-search(b,q,K)
```

为了快速方便地实现 KNN 查询，算法中设计了两个列表：

（1）PriorityQueue：待查询节点列表.

（2）KNNList：结果节点列表.

其中，PriorityQueue 主要用于引导搜索沿着最有可能的分支方向进行，距离 q 越近的节点，越先得到搜索. KNNList 用于存储目前为止找到的最近邻节点集，该集合与 q 间的最大距离在搜索索引树时还可用于剪枝判断，以减少距离计算次数. 这两个列表中的数据都是根据与 q 的距离升序排列的.

3.4　CSA-Tree 插入

为使 SA-Tree 适用于动态环境，文献[5-6]给出了很多很好的插入和删除算法，像溢出桶、重构和时间戳等技术，这些算法都在某种程度上实现了动态概念. 但是，究竟哪一种方法更适用于主存数据库，还需要近一步的研究.

事实上，CSA-Tree 的插入操作很容易实现. 首先，利用根节点存储的 ***Eigenmatrix*** 将新插入节点映射到 PCA 空间. 然后，在 CSA-Tree 中寻找插入位置，若该位置有可用空间，直接将其插入；若该位置已满，则需要溢出处理. 需要指出，为新节点 p 寻找插入位置时，我们是从树的根节点开始，从上到下寻找 p 的最近邻点，然后将 p 作为它的子节点插入树中.

由于 CSA-Tree 采用了链表和顺序表混合的存储结构（即同层子节点顺序存储），这样在插入一个新节点时，对应的父节点信息需要修改，主要是修改其中的最大覆盖半径 R 和子节点个数 num 的参数值，这样利用父节点中的指向子节点序列的头指针可以访问所有的子节点.

上述讨论的插入过程，没有涉及到特征矩阵修改操作，因此在经历大量新数据插入后，原特征矩阵就不能很好地反映新的数据空间变化情况. 但是，对特征矩阵的修改，将引起 PCA 空间点的表示也需要相应的修改，这意味着整个索引树可能重建. 因此，在保证索引树一定的检索精度前提下，选择合适的时机修改特征矩阵和重构索引树，对降低构造代价具有很重要的意义. 在我们的实验中，采用了两种方

法:(1)插入阈值:当新插入的点个数达到一定的预定义值(如原空间点总数的 30%)时,重新计算特征矩阵.(2)距离变化阈值:假设索引树中任意点 pt $(1\leq t\leq n)$表示为 pt_{m_i} $(1\leq i<L)$,则整个数据空间经 CSA-Tree 索引后的平均距离变化值为

$$V=\sum_{t=1}^{n}\Big(\sum_{j=m_{i+1}}^{d}\Big|\prod(pt,j)-\prod(r,j)\Big|\Big)\Big/n.$$

每插入一个新数据 p_{m_i},我们可以得到新的距离变化值平均值为

$$V'=n\times V+\sum_{j=m_{i+1}}^{d}\Big|\prod(p,j)-\prod(r,j)\Big|\Big/n+1.$$

当 $V'-V/V$ 大于某个给定阈值(与新插入节点的分布有关)时,需要重新构造索引树以保证检索精度,因为距离变化值实质上是原数据在 CSA-Tree 索引后的距离损失值.

3.5 算法分析

为了方便讨论 CSA-Tree 构造算法和查询算法的时空复杂度,假设 CSA-Tree 子树和 S_d 空间的点分布情况是一致的.在 CSA-Tree 中,树的每一层节点的维度是不同的,这样树中点的分布情况必然和 S_d 中点的分布情况有所差别,但考虑到采用 PCA 降维方法能够最大程度地保留原数据集的变化,因此我们可以忽略这种差异.对于 S_d 中的任意两点 q_1,q_2,假设 $Dist(q_1,q_2)=\chi$ 的概率为 p_x,从而有 $\int_0^{\infty}p_x\mathrm{d}x=1$.那么,$Dist(q_1,q_2)<\chi$ 的概率为 $P_x=1-\int_x^{\infty}p_x\mathrm{d}x$.

首先分析 CSA-Tree 构造算法的时空复杂度.在时间复杂度的分析中主要考虑距离比较的执行次数.假设空间中点的个数为 n,则整棵树中点的平均邻居点个数为 $\Theta(\log n)$,树的平均深度为 $\Theta\Big(\frac{\log n}{\log\log n}\Big)$.在 CSA-Tree 的构造算法中,确定各点的邻居节点算法和文献[4]中 SA-Tree 的算法是一致的,故没有在本文列出.在候选集 $Q(a)$ 中寻找点 a 的邻居 $N(a)$,假设 $Q(a)$ 中点的个数为 n_Q,那么寻找 a 的邻居点所需的距离比较次数为 $n_Q\log n_Q$,为 a 的邻居点构造它们的邻居点的候选集所需的距离比较次数也为 $n_Q\log n_Q$,平均有 $\frac{n_Q}{\log n_Q}$ 个点进入到 a 的邻居点的候选集中.这样,从根节点开始,对于每一个点执行上述操作,可以得到构造算法的时间复杂度的递推公式:

$$B(n)=2n\log n+\log nB\Big(\frac{n}{\log n}\Big)=\Theta\Big(\frac{2n\log^2 n}{\log\log n}\Big).$$

需要指出的是,CSA-Tree 中,在寻找点 a 的邻居 $N(a)$的过程中,对于处于树中不同层次的点 a,其距离计算花费的时间是不同的.距离根节点越近的点,其维度越小,距离计算花费时间越少;叶子节点的维度最大,距离计算花费时间也就最多.而 SA-Tree 由于没有对点数据进行降维处理,树中每个层次点数据的维度都一样(全维),点的距离计算时间与 CSA-Tree 的叶子节点是一样的,因此 CSA-Tree 的构造算法在时间性能方面要优于 SA-Tree.

容易看出,CSA-Tree 的空间复杂度和 SA-Tree 一样,为 $O(n)$,但是这里没有考虑降维处理对节点大小的影响.PCA 降维后,CSA-Tree 中各层的节点(叶子节点除外)相比 SA-Tree 的节点占用的空间要小很多.

下面分析 CSA-Tree 查询算法的时间复杂度.文献[4]指出,对于最近邻查询,SA-Tree 的查询复杂度计算公式为

$$Q(n)=\log(n)(1+P_{2r+\varepsilon}Q(n/\log n)).$$

在 CSA-Tree 中,主要实现的查询算法是 K 近邻查询.通过分析可以发现,K 近邻查询的时间复杂度和最近邻查询的时间复杂度大致相同.但是在这里不能认为 CSA-Tree 查询算法的时间复杂度和 SA-Tree 完全一样,由于降维的影响,利用 CSA-Tree 进行查询的剪枝速度比 SA-Tree 要快,这可以从本文后面的实验结果中看出.所以,我们必须将 $Q(n)$计算公式中的 $P_{2r+\varepsilon}Q(n/\log n)$项乘上某个权重 $\omega(\omega<1)$,ω 表示降维后实际进入的 CSA-Tree 子树个数要小于同样情况下利用 SA-Tree 查询时进入的子树个数.需要说明的是,ω 是不确定的,可以根据具体应用的空间数据分布来加以推测.因此,CSA-Tree 查询算法的时间复杂度公式为

$$Q(n)=\log(n)(1+\omega P_{2r+\varepsilon}Q(n/\log n)).$$

解这个公式可得 CSA-Tree 查询算法的时间复杂度

$$Q(n)=n^{1-\Theta\left(\frac{\log(1/\omega P_{2r+\varepsilon})}{\log\log n}\right)}=n^{1-\Theta(1/\log\log n)}.$$

在 CSA-Tree 中插入一个点 p,实质上就是为 p 查找其最近邻点的过程,所以,CSA-Tree 插入算法的时间复杂度和最近邻查询的时间复杂度是一样的.

4 实验结果与分析

我们的实验环境是:Intel Celeron Ⅱ 1000MHz,

256MB SDRAM，128KB L2-Cache. 操作系统为 Win 2000 Server，开发环境为 VC6.0. 选用了 Intel 公司提供的 emon 软件，测试 L2-cache 未命中次数. 实验数据是从 UCI KDD ATCHIVE 图像库提取的颜色特征值. 在实验中为了消除因查询图像的不同而造成的检索时间不同的影响，我们将测出 100 次不同检索所消耗的平均时间，作为实验结果数据. 除此以外，为了让 CSA-Tree 全部或最上面几层树节点真正存储在缓存中，达到主存索引检索目的，可以在执行多次检索后开始测试实验数据.

表 3　节点大小和查询消耗时间及 L2-cache 未命中关系表(n=7000)

节点大小/K	消耗时间/s	L2-cache 未命中
1	0.62	2880
2	0.56	2152
3	0.57	2356
4	0.59	2991
5	0.61	3010
6	0.68	3121
7	0.70	3286
8	0.72	3480

尽管一维索引树节点的最佳大小为一个缓存栈的大小，但是这并不适用于高维索引树，因为对一个 64 维的点数据来说，它的大小就为 256Bytes，而目前通用的处理器的一个缓存栈的大小只有 64Bytes，那也就是说一个缓存栈连一个高维节点都存储不下. 文献[8]指出二维索引树最优的树节点大小为 256～512Bytes，说明随维度增加，树节点大小应适当增大. 因此，高维索引树节点的大小要比一个缓存栈大得多，实验将从大小为 1K 的树节点开始，逐渐增加树节点大小，通过测量检索消耗时间和 L2-cache 未命中来确定 CSA-Tree 树节点的最佳大小. 由表 3 可以看出，CSA-Tree 树节点的最佳大小为2～3K，当节点大小小于 2K 时，每个树节点存储的点数据相对较少，检索过程需要访问的树节点就较多，所以检索消耗时间较长. 随着树节点的增大，检索时间逐渐减少. 但是当增大到 3K 以上时，检索性能又开始下降，这是因为太大的节点增加了每个节点的 L2-cache 未命中几率，所以总体性能下降. 需要指出，最佳节点大小指的是树叶节点的大小，对于树中其它节点的大小，可以根据它们的维度在此基础上适当地做些调整. 以下的实验将 CSA-Tree 和其它方法进行比较分析，由于现有的高维索引树有很多种，为使比较具有一定的普遍性，将选择几种常用的高维索引树作为比较对象，如 SA-Tree，M-Tree，CR-Tree. 实验采用来自 1000 幅图像提取的 64 维颜色特征数据，KNN 查询结果集大小 K=10. CSA-Tree 叶节点大小设计为 2K，节点扇出值固定为 8，累积贡献率的起步值 Δ=0.7.

实验 1(图 2)考察了维度对查询时间的影响. 根据 1000 幅图像的不同维(8，16，24，32，40，48，56，64)颜色特征向量，统计检索消耗时间. 由图 2 的数据可以看出，随维度的增加，CSA-Tree 检索的时间增长缓慢，而其它方法的时间增长较快；并且在维数相同的情况下，其它方法消耗的时间远远大于 CSA- Tree 检索时间. 这主要是因为 CSA-Tree 利用了 PCA 降维处理，在较小的距离信息损失情况下，降低了维度增加带来的影响，并且降维后的距离计算复杂性也降低了，所以应用 CSA-Tree 检索消耗的时间少.

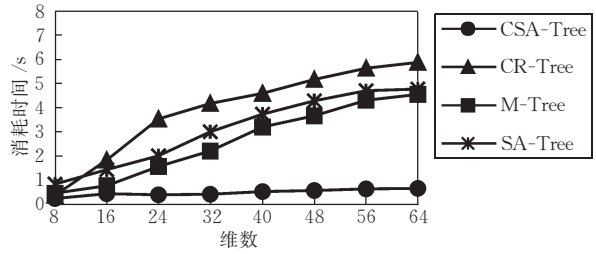

图 2　维数和查询消耗时间(n=1000)

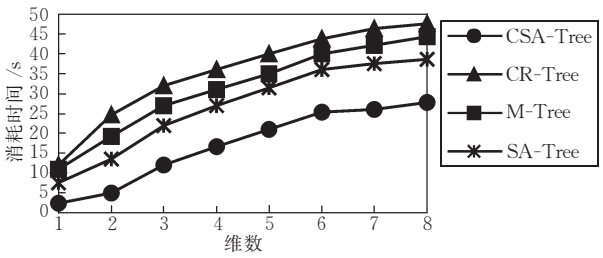

图 3　数据集大小和查询消耗时间(dim=64)

实验 2(图 3)考察了数据量对查询时间的影响，使用数据量从 1000 条变化到 8000 条. 可以看出，运用 CSA-Tree 进行检索具有很好的扩展性，主要因为 CSA-Tree 的 KNN 查询算法能实现快速剪枝，避免了非候选点的距离计算，所以图像集的大小对其检索不会带来太大影响. 并且，由于利用了 Voronoi 图思想 CSA-Tree 将数据空间划分成不同的邻居区域，更加有利于 KNN 检索剪枝，所以距离计算次数少了 50%之多(如图 4).

实验表明 CSA-Tree 最上面几层节点的维度一般只有很少几维(小于 6)，远远小于原图像数据

的全维表示，所以查询过程中经常需要检索的上层节点就可以存储在缓存中，这样可以大大减少 L2-cache 未命中次数，又因为 CSA-Tree 能实现快速剪枝，所以大部分的中间节点和叶节点在搜索过程中很可能不需要检索，所以 CSA-Tree 的 L2-cache 未命中率很小(图 5).

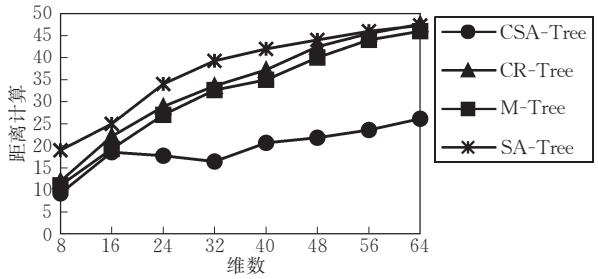

图 4 维数和距离计算次数(n=1000)

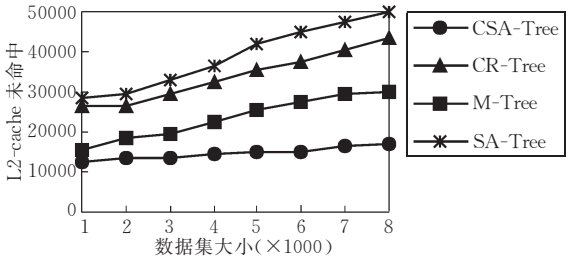

图 5 L2-cache 未命中和数据集大小(维数=64)

5 总 结

本文提出了一种优化的高维主存索引树 CSA-Tree. 它是一个每层节点采用不同的维度表示的多级树结构. 通过 PCA 降维技术的应用，简化节点的向量表示，降低索引树的大小，这样检索过程中访问频率较高的上层树节点可以直接存储在 L2-cache 中. 在 KNN 检索过程中，由于降维带来的距离计算代价和 L2-cache 未命中的减少，使得检索效率得到了很大程度的提高. 为了说明 CSA-Tree 的可行性，本文还设计了大量实验，将 CSA-Tree 和其它高维索引检索方法进行比较分析，说明 CSA-Tree 在很多方面有很好的表现. 今后还需要进一步研究如何优化动态 CSA-Tree 算法以及并发性控制等方面的内容.

参 考 文 献

[1] Bohm C, Berchtold S, Keim D. Searching in high-dimensional spaces: Index structures for improving the performance of multimedia databases. ACM Computing Surveys, 2001, 33(3): 322-373

[2] Hjaltson G R, Samet H. Index-driven similarity search in metric spaces. ACM Transactions on Database Systems, 2003, 28(4): 517-580

[3] Chavez E, Navarro G, Yates R b, Marroquin J L. Searching in metric spaces. ACM Computing Surveys, 2001, 33(3): 273-321

[4] Navarro G. Searching in metric spaces by spatial approximation//Proceedings of the String Processing and Information Retrieval and International Workshop on Groupware. Cancun, Mexico, 1999: 141-148

[5] Navarro G, Reyes N. Dynamic spatial approximation trees//Proceedings of the XXI Conference of the Chilean Computer Science Society. Punta Arenas, Chile, 2001: 213-222

[6] Navarro G, Reyes N. Improved deletion in dynamic spatial approximation//Proceedings of the XXIII Conference of the Chilean Computer Science Society. Chillan, Chile, 2003: 13-22

[7] Bohannom P, Mcllroy P, Rastogi R. Main-memory index structures with fixed-size partial keys//Proceedings of the ACM SIGMOD Conference. Santa Barbara, CA, USA, 2001: 163-174

[8] Kim K, Cha S K, Kwon K. Optimizing multidimensional index trees for main memory access//Proceedings of the ACM SIGMOD Conference. Santa Barbara, CA, USA, 2001: 139-150

[9] Rao J, Ross K. Making B^+-trees cache conscious in main memory//Proceedings of the ACM SIGMOD Conference. Philadelphia, PA, 2000: 194-205

[10] Ailamaki A, DeWitt D J, Hill M D, Wood D A. DBMSs on a modern processor: where does time go//Proceedings of the 25th VLDB Conference. Edinburgh, Scotland, 1999: 266-277

[11] Rao J, Ross K A. Cache conscious indexing for decision-support in main memory//Proceedings of the 25th VLDB Conference. Edinburgh, Scotland, 1999: 78-89

[12] Jolliffe I T. Principal Component Analysis. New York: Springer-Verlag, 1986

[13] Hui J, Ooi B C, Shen H, Yu C, Zhou A. An adaptive and efficient dimensionality reduction algorithm for high-dimensional indexing//Proceedings of the 19th ICDE Conference. Bangalore, India, 2003: 87-98

[14] Chakrabarti K, Mehrotra S. Local dimensionality reduction: A new approach to indexing high dimensional spaces//Proceedings of the 26th VLDB Conference. Cairo, Egypt, 2000: 89-100

[15] Bozkaya T, Ozsoyoglu M. Distance-based indexing for high-dimensional metric spaces//Proceedings of the ACM SIGMOD Conference. Tucson, Arizona, 1997: 357-368

LIANG Jun-Jie, born in 1974, Ph.D. candidate, assistant professor. Her research interests include content-based multimedia information retrieval, high-dimensional index and DBMS.

FENG Yu-Cai, born in 1946, professor, Ph. D. supervisor. His current research interests include multimedia signal processing, GIS and DBMS.

Background

In main memory systems, distance computations and L2 cache misses contribute significantly to the overall cost. Presently, several main memory indexing schemes have been designed to be cache conscious. These schemes are targeted at single or low dimensional data, which can not efficiently deal with high-dimensional data. In high-dimensional data space, distance calculations are computationally expensive. Therefore, the authors present an efficient main memory index that can exploit the L2 cache effectively and minimize the distance computation to improve the performance.

This subject is supported by the National High Technology Development Program (863 Program) of China under Grant No. [2005]555 whose name is "Large-scale Universal DBMS". The project focuses on research and development of database management system. The team has made a lot of progress in the area of DBMS and published nearly 30 papers in international and domestic journals or conference proceedings. Many multidimensional indexing methods are proposed, of which some have been used to manage multimedia data in practice. Some optimized strategies are put forward to facilitate similarity search in high-dimensional spaces. This paper proposes a novel multi-tier index structure, that can facilitate KNN search in main memory environment.