基于关键维的高维空间划分策略*

周项敏+, 王国仁

(东北大学 信息科学与工程学院,辽宁 沈阳 110004)

Key Dimension Based High-Dimensional Data Partition Strategy

ZHOU Xiang-Min+, WANG Guo-Ren

(School of Information Science and Engineering, Northeastern University, Shenyang 110004, China)

+ Corresponding author: Phn: +86-24-83681250, E-mail:zxmhn@21cn.com, http://www.neu.edu.cn

Received 2003-08-19; Accepted 2004-01-07

Zhou XM, Wang GR. Key dimension based high-dimensional data partition strategy. *Journal of Software*, 2004,15(9):1361~1374.

http://www.jos.org.cn/1000-9825/15/1361.htm

Abstract: Index is one of the core components of content based similarity search and the data partition is the key factor affecting the performance of index. This paper proposes a new data partition strategy—key dimension based partition strategy on the basis of the traditional distance based partition strategy, and the index technique accordingly. The key dimension based data partition eliminates the overlaps between twin nodes, and the filtering between twin nodes by key dimension enhances the filtering ability of index. The data partition strategy and index technique proposed can greatly improve the filtering ability of index. Experimental results show that key dimension can be used to improve the performance of index, which is of great significance for accelerating the content based similarity search.

Key words: multidimensional index; metric space; key dimension; range search; *K*-NN search

摘 要: 索引技术是基于内容的相似性检索的核心内容,而数据的分割则是影响索引性能的关键因素.提出一种高维数据空间分割策略——在距离分割基础上基于关键维的二次分割,以及相应的索引技术.基于关键维的二次分割保证孪生兄弟节点的无重叠性,而在索引中根据选定的关键维进行孪生兄弟节点间的二次过滤,从而增强过滤效率.这种数据分片策略和索引技术使得索引的过滤效率成倍提高.实验结果显示,关键维能够很好地提高索引的相似性检索性能,对于加速基于内容的多媒体信息检索具有很大的意义.

关键词: 高维索引;度量空间;关键维;范围查询;最近邻查询

中图法分类号: TP311 **文献标识码**: A

* Supported by the National Natural Science Foundation of China under Grant No.60273079 (国家自然科学基金); the University Key Teacher Award Program for Outstanding Young Teachers in High Education Institute of the Ministry of Education of China (教育部高等学校优秀青年教师教学科研奖励计划基金)

作者简介: **周项敏**(1973－),女,河南伊川人,博士,主要研究领域为多媒体数据库;**王国仁**(1966－),男,博士,教授,博士生导师,主要研究领域为数据库理论和技术,分布与并行数据库,Web 数据管理技术,生物信息管理,面向对象数据库.

高维索引技术是加速相似性检索的关键技术之一.目前已经出现了很多高维索引方法,例如R-tree[1],R*-tree[2]以及R-tree的其他变种[3~5].R*-tree及其他R-tree变种都属于基于位置的空间访问方法,它们广泛用于地理信息系统中.但是,这种基于位置的索引方法存在着很大的局限性,只有在以下两个条件同时满足的情况下才适用:(1) 索引对象必须能够表示为一个高维向量空间的特征值;(2) 对象之间的相似性必须通过欧几里德距离来度量.对于颜色直方图等存在关联的距离函数,基于位置的索引方法就无能为力了[6].

实际应用中,大量图像信息属于度量空间.为满足度量空间中快速检索的需要,人们提出了基于距离的索引技术.基于距离的索引技术不同于基于位置的索引技术,它们处理对象之间的相对距离,而并不关心对象间的相对位置.在这类索引技术中,比较典型的有:M-tree[6],MVP-tree[7],MB+-tree[8],Slim-tree[9]等.其中VP-tree和MVP-tree是两个比较典型的基于度量空间的静态索引结构,而M-tree则第一次实现了基于度量空间的动态索引结构.MB+tree和Slim-trees对M-tree进行了改进,MB+tree利用两个单维的索引(B+tree和Block-tree)代替一个高维的索引结构,实现数据空间的不重叠分割.Slim-trees则利用一个建树后处理过程从而使节点个数和数据节点的覆盖半径最小化.

我们在M-tree和二元MVP-tree的基础上,提出了M^+-tree[10,11].M^+-tree具有如下特点:(1) 它继承了M-tree的对象提升机制、三角不等式过滤以及分支界限技术;(2) M^+-tree充分利用了MVP-tree的二次过滤思想;(3) M^+-tree提出了关键维和关键维转移的思想,从而进行数据空间的有效过滤,同时,又不需要进行额外的距离计算;(4) M^+-tree提出孪生兄弟节点的概念,使索引树的高度相对降低,并且在进行大量实验分析的基础上,对M^+-tree和M-tree的性能进行了系统的分析与评价.

本文在以前工作的基础上,详细介绍了基于关键维的二次数据分割策略、关键维的概念以及在此基础上的索引结构和检索技术.我们提出的关键维不仅仅局限于某个维,而是若干个维的线性组合.本文进行了大量的实验分析,通过实验证明k-NN查找中优先队列访问次数对查询性能的影响,从而证明队列访问是影响查询性能的一个不可忽视的因素.并且对M^+-tree的关键技术进行了更为详细的介绍.通过大量实验分析,与M-tree以及Slim-trees进行查询性能比较,证明了与Slim-trees相比,M^+-tree是一种更有效的改进技术.

本文第1节给出一些相关工作.第2节详细介绍基于关键维的二次分割策略.第3节详细介绍M^+-tree,包括M^+-tree的数据结构、关键技术和相关算法.第4节给出性能评价,包括理论分析和实验结果分析.第5节总结全文.

1 相关工作

MVP-tree是针对高维度量空间中数据提出的一种快速、有效的相似性查找解决方案.VP-tree[8]利用数据对象到特定受益点间的距离来分割数据空间,而MVP-tree[7]扩展了VP-tree的思想,将受益点个数增加到两个以上,同时采用距离预计算来减少距离计算次数.对于二元MVP-tree,在每一个中间节点,首先根据第1个受益点将数据空间分为两个子空间.然后,根据第2个受益点将数据空间进一步分割为4个子空间.MVP-tree的这种空间分割策略增加了索引的扇出数,降低了树的高度,因而极大地提高了查找性能.但它是一种静态索引结构,不能进行动态调整.为了保证索引性能,必须进行索引的重建.

与VP-tree和MVP-tree相比,M-tree[6]是一种动态索引结构.它是一个分页平衡树,采用自底向上的建树方法,引入结点上移和分裂机制,实现了索引结构的动态化,从而避免了静态索引结构的索引重建.M-tree第一次考虑了距离计算的复杂性,实现了范围查询,并在范围查询的基础上,采用启发式规则,实现了高维数据的最近邻查询.M-tree的出现具有划时代的意义,至今仍被看做是性能最优的索引之一.但由于M-tree的每个节点都对应一个超球体的区域,节点之间重叠非常大.同时,受磁盘页面大小的限制,M-tree索引树具有相对较高的高度.因此,本文提出了关键维的概念,在基于距离划分的基础上,进行基于关键维的次划分,从而将一个子空间进一步分割成一对孪生子空间,并且利用孪生兄弟节点间的数据重分配保证数据空间的利用率.

2 基于关键维的分割策略

2.1 基本概念

在介绍基于关键维分割策略之前,本节首先介绍与 M^+-tree 相关的概念.

定义 1(度量空间). 度量空间 M 在形式上定义为 $M=(O,d)$,其中 O 是特征值的范围,d 是一个距离函数.假设 O_x,O_y,O_z 是对象集中的 3 个对象,它们之间的相似性可以通过距离函数 d 来度量.距离函数 d 具有以下特性:

(1) 对称性.即 $d(O_x,O_y)=d(O_y,O_x)$;

(2) 非负性.即当 $O_x\neq O_y$ 时,$d(O_x,O_y)>0$;当 $O_x=O_y$ 时,$d(O_x,O_y)=0$;

(3) 三角不等式.即 $d(O_x,O_y)\leq d(O_x,O_z)+d(O_z,O_y)$.

定义 2(范围查询). 给定一个查询对象 q 和一个查询半径 r,其中 q 属于对象集 O,r 是一个非负值,范围查询就是查找所有满足以下条件的数据库对象:(1) $o\in O$;(2) $d(q,o)\leq r$.

定义 3(最近邻查询). 已知一个查询对象 q 和一个整数 k,其中 $q\in O$,$k\geq 1$,最近邻查询就是从数据库中找到距 q 最近的 k 个数据库对象.

定义 4(关键维). 在高维空间中,由于各个子空间中的数据分布不同,在进行相似性过滤时,各个维所起的作用不同,我们称对过滤影响最大的维为关键维.此处的维指的是高维空间中的某个维或者某几个维的线性组合.

定义 5(关键维转移). 由于各个子空间对过滤影响最大的维可能是各不相同的,因此不同的子空间有不同的关键维,我们称这种关键维的变化为关键维转移.

定义 6(孪生节点). 在 M^+-tree 中,每个中间节点入口项对应一对子树,这一对子树的两个根称为孪生节点.

2.2 基于距离和关键维的两步分割策略

数据空间的分割是最重要的研究课题之一.数据空间分割是否合理,直接影响到索引的性能.图 1 给出了 M-tree 中空间分割策略和 M^+-tree 中空间分割策略的对比情况.

M-tree 是根据数据空间中各个数据对象间的相对距离进行空间分割的.按照一定的空间分割策略,将一个数据空间分成两个子空间.M-tree 中,最小半径分割策略被认为是最优空间分割策略.图 1 中的子图(1)和(2)给出了 M-tree 中数据空间分割示意,子图(1)中,整个数据空间经过分割,成为两个超球体子空间,如子图(2)所示.

图 1 中(3)和(4)两个子图给出了 M^+-tree 中数据空间分割的一个示意.在 M^+-tree 中,与一个节点入口项相对应的是两个孪生的子空间.例如,在子图(3)和(4)中,空间 A 和空间 B 就是一对孪生的子空间.在 M^+-tree 中,这两个子空间对应的节点就称为孪生节点.因为左子空间关键维的最大值越小,关键维的过滤效率越好,右子空间关键维的最小值越大,关键维的过滤效率越好.为了提高关键维过滤的效率,使用两个关键维边界值,即左子空间关键维最大值和右子空间关键维最小值,将两个子空间分开.

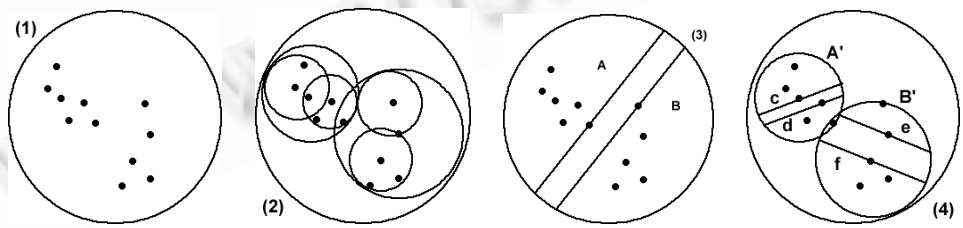

Fig.1 Data partition of M-tree vs. M^+-tree

图 1 M-tree 和 M^+-tree 数据分割

M^+-tree 采用基于距离的分割和基于关键维的分割两步分割法.第 1 步,将一个空间通过最小半径分割法分成两个子空间,这一步分割方法与 M-tree 中的空间分割方法相同,即为基于距离的分割.第 2 步,基于关键维的分割策略,按照关键维大小将这两个子空间进一步分割成 4 个子空间.如图 1 中子图(4)所示,子空间 A′被分成一对孪生子空间 c 和 d,同时,子空间 B′被分成一对孪生子空间 e 和 f.关键维分割实质上是利用超平面分割数据空间.

2.3 关键维的选择

关键维的选择遵循一个原则,即在保证无重叠分割的前提下,尽量将距离较近的对象分到同一子空间中.实验证明,当分割一个数据空间时,沿方差最大的维进行分割具有最好的查询性能,在 SS-tree 和 SR-tree 中,数据空间都是沿方差最大的维进行分割.因此,我们在这里选用一个数据空间中方差最大的维作为关键维.下面通过例子来说明关键维的选择.

关键维的选择是针对一个子集合来说的.假设有一个包含 4 个数据对象的 4 维数据空间,各个对象对应的向量为(0.2,0.3,0.4,0.5),(0.2,0.3,0.4,0.4),(0.2,0.3,0.4,0.6),(0.2,0.3,0.4,0.7).可以看到前 3 维的方差都为 0,而第 4 维的方差为 0.35,因此,第 4 维为关键维.

2.4 基于关键维的过滤原理

关键维的过滤是根据距离缩放和三角不等式原理来进行的.通过关键维过滤,一些无效的分支可以直接被过滤掉,同时又能保留所有正确的查询结果.

当特征向量的若干个维之间存在着关联时,数据之间的相似性需要通过非欧几里德距离来度量.通用距离函数可表示如下:

$$d=\sqrt{(x-y)\cdot A\cdot(x-y)^T}=\sqrt{\sum_{i=1}^{N}\sum_{j=1}^{N}a_{ij}\cdot(x_i-y_i)\cdot(x_j-y_j)}\ .$$

当矩阵 A 为对角阵时,通用距离函数即为加权的欧几里德函数.当 A 为单位矩阵时,则为欧几里德距离.即使图像之间通过非欧几里德距离来度量,基于关键维的过滤也是有效的.此时,关键维是特征向量中某几个维的线性组合.我们称该线性组合为扩展的关键维.下面一个简单的例子可用来说明关键维过滤的有效性.为了使问题简单化,我们假设特征向量为三维向量,并且关联矩阵为 A=[⟨1,0.8,0⟩,⟨0.8,1,0⟩,⟨0,0,1⟩].

假设有两个点P_1=(x_1,x_2,x_3),P_2=(y_1,y_2,y_3),则点P_1和P_2间的距离可以表示如下:

$$\begin{aligned}d(P_1,P_2)&=\sqrt{(P_1-P_2)\cdot A\cdot(P_1-P_2)^{\mathrm{T}}}=\sqrt{(x_1-y_1,x_2-y_2,x_3-y_3)\cdot A\cdot(x_1-y_1,x_2-y_2,x_3-y_3)^T}\\&=\sqrt{(x_1+0.8x_2-(y_1+0.8y_2))^2+(0.6x_2-0.6y_2)^2+(x_3-y_3)^2}\ .\end{aligned}$$

假设原数据空间为(d_1,d_2,d_3),则数据可以转化到一个新的数据空间中.假设新数据空间为(D_1,D_2,D_3).构造两个数据空间之间的联系.假设 D_1=d_1+0.8d_2,D_2=0.6d_2,D_3=d_3.此时,特征向量被转化到欧几里德空间中去.关键维的有效性可以利用欧几里德距离来得以验证.

下面通过数学公式来证明关键维过滤的有效性.假设 $O_i(X_1,X_2,X_3)$和 $O_j(Y_1,Y_2,Y_3)$是两个转化到空间(D_1,D_2,D_3)中的两个对象,则两个对象之间的距离可以表示为

$$D(O_i,O_j)=\sqrt{(x_1-y_1)^2+(x_2-y_2)^2+(x_3-y_3)^2}\ .$$

假设第 k 维为关键维,则$|X_k-Y_k|<D(O_i,O_j)$.假设查询半径为 r,如果 $D(O_i,O_j)\le r$,则必有$|X_k-Y_k|\le r$.对于 n 维向量可以用同样的方法依次类推.因此,无论是哪一种数据空间,都可以利用关键维进行过滤,也就是说,关键维的过滤不受数据空间的限制.

关键维过滤的过程可描述如下:假设有一对孪生兄弟节点,其所在的子空间关键维为第 k 维,孪生兄弟节点间关键维的分界值为 $L_{\max}$ 和 $R_{\min}$,其中 $L_{\max}$ 为左节点中所有对象关键维的最大值,$R_{\min}$ 为右节点中所有对象关键维的最小值.查询对象为 $O(x_1\ldots x_k\ldots x_n)$,查询半径为 r.当 $x_k-L_{\max}>r$ 时,表明左孪生节点中不包含符合条件的对象,因此,左节点可直接被过滤掉.当 $R_{\min}-x_k>r$ 时,表明右孪生节点中不包含查询结果,则可将右节点直接滤掉.对于最近邻查找的过滤原理相类似.

3 M$^+$-tree:一种高效的基于度量空间的索引技术

3.1 M$^+$-tree的结构

在 M-tree 的基础上,M$^+$-tree 对索引结构和算法作了全新的改进.M$^+$-tree 沿用了 M-tree 中的一些术语,与 M-tree 一样,M$^+$-tree 中的节点对象分为两种:路径对象和叶子对象.

M$^+$-tree 叶子结点中存储所有索引的数据库对象,它的入口项结构可表示为如下形式:

$$L(O_j,oid(O_j),d(O_j,P(O_j)),$$

其中,O_j 为数据库中媒体对象的特征值;$oid(O_j)$为数据库对象的标识符;$d(O_j,P(O_j))$为对象 O_j 距离其父节点中心的距离.$oid(O_j)$是唯一的,它唯一地表示数据库中的一个媒体对象.

路径结点存储的是中间结点对象,它的入口项结构可表示为如下形式:

$$R(O_r,r(O_r),d(O_r,P(O_r)),D_{\mathrm{NO}},\mathrm{leftTwinPtr}(T_{lt}(O_r)),M_{l\max},M_{r\min},\mathrm{rightTwinPtr}(T_{rt}(O_r))),$$

其中,O_r 为路径节点对象的特征值;$d(O_r,P(O_r)$为 O_r 距离其父节点的距离;D_{NO} 为关键维号;leftTwinPtr($T_{lt}(O_r)$)为指向孪生子树左子树的指针;rightTwinPtr($T_{rt}(O_r)$)为指向孪生子树右子树的指针;$M_{l\max}$ 为左子树关键维的最大值;$M_{r\min}$ 为右子树关键维的最小值.

M$^+$-tree 的路径对象有两个指针,每个中间节点入口项对应于两棵子树.这种结构使得 M$^+$-tree 的高度大大降低.图 2 给出了 M-tree 和 M$^+$-tree 的结构示意图.

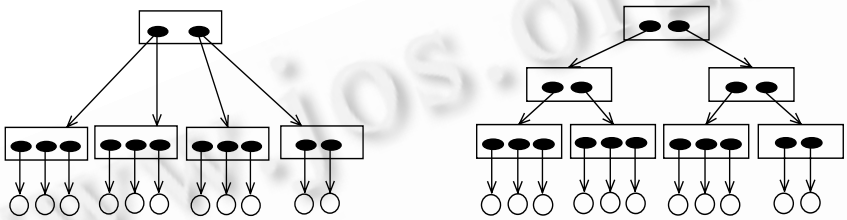

Fig.2 The structure of M+-tree vs. M-tree

图 2 M+-tree 树和 M-tree 树结构

3.2 节点插入

算法 1(子树选择).

输入:当前所在节点 N 和待插入的入口项 entry.

输出:最佳插入子节点.

步骤:

(1) 令 S 为节点 N 中所有入口项的集合;令 S_{in} 为与 entry 间的距离 d 小于入口项覆盖半径的所有入口项的集合;令右子树关键维的最小值为 R,左子树关键维的最大值为 L;

(2) 如果 S_{in} 不为空,选择 d 值最小的入口项.否则选择 d 值与覆盖半径之差最小的入口项,设该入口项为 entry(O'_n);

(3) 根据关键维进行孪生子节点选择.如果 entry 的关键维大于 R,则返回 entry(O'_n)的右子节点.如果 entry 的关键维小于 L,则返回 entry(O'_n)的左子节点.如果 entry 的关键维大于 L 且小于 R,则分别计算 entry 关键维与 L 和 R 之差的绝对值,如与 L 之差的绝对值较小,则返回左子节点,反之,返回右子节点.

在构造索引结构时,需要考虑许多问题,例如,数据节点上溢、下溢等.M$^+$-tree 的构造过程主要完成对象插入、节点分裂、孪生节点间对象重分配、对象提升以及路径对象的选取.

当一个节点对象插入到 M$^+$-tree 时,首先要从树的根结点出发,进行子树选择,找到节点要插入的位置.子树的选择遵循最优原则.首先,选择路径对象距离插入对象最近的子树.其次,尽量减少插入节点的半径增长.最后,如果插入任何子树分支都会引起该子树覆盖区域半径的增大,则选择半径增加量最小的子树.算法 1 和算法 2 分别描述了子树选择和插入算法.

算法 2(插入).

输入:当前所在节点 N 和待插入的入口项 entry.

步骤:

(1) 如果 N 不是叶子节点,则进行子树选择,选择最佳插入节点,并将对象插入;

(2) 否则,如果 N 是叶子节点,则进行以下操作:

(a) 如果 N 节点未满,则将对象直接插入;

(b) 否则,如果 Twin(N)为满,则将这一对孪生兄弟节点作为一个整体进行分裂,否则,进行孪生兄弟节点间的对象重分配.

3.3 节点分裂

M^+-tree 中的节点分裂分两步进行.第 1 步,将一对孪生节点当作一个大节点,根据其中各对象之间的相对距离进行分裂.这一步与 M-tree 的分裂过程类似.可以采用 M-tree 提供的任一种分裂策略.第 2 步,在两个子空间中进一步根据节点对象的关键维进行分裂.M^+-tree 按自底向上的方式进行分裂,采用了 M-tree 的节点提升机制和部分分裂机制.

当按关键维进行分裂时,如果节点是叶子节点,则直接将对象平均分配到左、右孪生叶子节点中.如果是中间节点,则可能引起部分子树的节点重分配.该问题可用对象重插入来解决.算法 3 给出了非根结点分裂算法的描述.

算法 3(非根结点分裂).

输入:该节点 N 在父节点中对应的入口项和要插入的入口项 E.

步骤:

(1) 令 S 为一个集合,其中包括 N 和 N 的孪生兄弟节点的所有入口项以及待插入的入口项 E,N_p 为父节点;

(2) 分配一个新的节点 N';

(3) 进行入口项对象的提升,选出两个父节点代表元素;

(4) 按最大最小半径的原则(该策略为 M-tree 中提到的一种最佳分裂策略),对象间的距离进行分裂得到两个入口项集合;

(5) 按关键维对这两个入口项集合进行二次分裂,生成两对孪生兄弟节点;

(6) 检查 N_p 节点的状态:

(a) N_p 为非根节点,如果 N_p 节点已满,而且其孪生兄弟节点 Twin(N_p)也为满,则对 N_p 和 Twin(N_p)进行分裂.若 N_p 满但 Twin(N_p)不满,则进行节点对象重分配;若 N_p 没满,则直接插入;

(b) N_p 为根结点,如果 N_p 节点已满,则进行根结点分裂,否则,直接插入对象.

3.4 节点重分配

由于 M^+-tree 的入口项对应的孪生兄弟节点是成对出现的,在进行数据库对象的插入操作时,这一对孪生兄弟节点不可能同时出现上溢.当一个对象插入一个节点空间时,该节点空间已满,其孪生兄弟节点可能空间还没有满.由于 M^+-tree 在进行分裂操作时,是把一对孪生兄弟节点当作一个整体同时进行分裂的,这样可以防止数据空间出现下溢现象,提高索引的空间利用率.M^+-tree 提出了节点重分配策略.

节点重分配就是在一对孪生兄弟节点中的其中一个节点出现上溢,而另一个空间还没满的情况下,在这对孪生兄弟节点之间重新分配它们的入口项,从而平衡孪生兄弟节点间的数据分配,以便推迟节点分裂的过程.

M^+-tree 的节点重分配过程采用两步进行:(1) 在建树的过程中,将孪生节点的入口项当作一个个的对象整体进行分配.采用平衡分配节点的原则将入口项平均分配到两个孪生兄弟节点中.(2) 在建树过程结束后,进行建树后处理.依次扫描索引数的各个节点,在存在重叠的孪生兄弟子树间调整子树包含的数据库对象,形成两棵完全不相交的子树.

M^+-tree 的两步节点重分配主要是为了节省建树时间.在建树过程中,孪生兄弟节点间的重分配非常频繁.要将一对孪生兄弟节点分成完全不重叠的两个区域,需要进行对象重插入,非常费时.而进行上述的第 1 步节点重分配操作却非常快.当建树结束后,只有少数孪生兄弟节点之间需要进行第 2 步节点重分配后处理.算法 4 和算法 5 分别给出了第 1 步和第 2 步节点重分配的算法描述.

算法 4(第 1 步节点重分配).

输入:一对孪生兄弟节点 LN 和 RN 以及待插入的入口项对象 E.

步骤:

(1) 令 S 为 LN 和 RN 中所有入口项对象以及对象 E 的集合,并从 LN 和 RN 中删除所有的入口项对象;

(2) 当 S 不为空时,循环进行以下操作,直到 S 为空;

(a) 从 S 中选择一个入口项 e,其中 e 包含的子树中所有对象的关键维最大值为最小,将 e 插入 LN,并将 e 从 S 中删除;

(b) 如果 S 不为空,则从 S 中重新选择一个入口项 e,其中 e 中包含的子树中所有对象的关键维的最小值为最大,将 e 插入 RN,并将 e 从 S 中删除;

算法 5(第 2 步建树后处理中节点重分配).

输入:一对孪生的兄弟节点 LN 和 RN.

步骤:

(1) 令 s 为 LN 和 RN 中包含的所有入口项的集合;

(2) 重新选关键维 keyNo,查找左子树关键维最大值 L_{Max},右子树关键维最小值 R_{Min};

(3) 从 LN 和 RN 中删除所有入口项;

(4) 令 S 为需要重插入的对象集合,将 S 初始化为空;

(5) 当 s 不为空时,进行以下操作:令子树关键维最大值 maxkey,最小值 minkey;

(a) 从 s 中选取入口项 e,其中 e 所对应的 maxkey 最小;

(I) 如果 maxkey 大于 L_{Max},且 LN 中入口项个数大于 0,则跳出循环;

(II) 否则,将 e 对应的子树中关键维值大于 L_{Max} 的对象放入集合 S,并从子树中删除.将 e 插入左子树 LN;

(b) 若 s 非空,从 s 中选取入口项 e,其中 e 所对应的 minkey 最大;

(I) 如果 minkey 小于 R_{Min},且 RN 中入口项个数大于 0,则跳出循环;

(II) 否则,将 e 对应的子树中关键维值小于 R_{Min} 的对象放入集合 S,并从子树中删除.将 e 插入右子树 RN;

(6) 若 s 非空,取出所有 s 中入口项 e,并将子树中包含的所有对象取出放入 S 集合中;

(7) 将集合 S 中的所有对象进行重插入.

3.5 M^+-tree的查询处理

查询性能的好坏是索引结构好坏的一个重要标志.M^+-tree 处理两类查询:范围查询(r-neighbor search)和最近邻查询(k-NN_Search).

范围查询首先从索引树的根结点出发,遍历所有可能包含查询结果的查询路径.在节点的每个入口项中,都保存有该入口项距离父节点的距离.为了减少距离计算,将该距离用于三角不等式.这样,一部分不包含查询结果的无效分支就会被过滤掉.对于未被滤掉的子树,需要计算查询对象与各个路径对象之间的距离,然后,根据三角不等式进行进一步的过滤.以上过程同 M-tree 中进行范围查询完全一样.接着进行第 2 步过滤,即基于关键维在孪生节点间进行过滤.这一步不需要任何距离计算,而且,对于任何可以滤掉的子树,不需要任何 I/O 访问.上面两步可以在中间节点间递归执行下去,直到叶子节点.通过计算这些数据库对象与查询对象之间的距离,比较该距离与查询半径 r 的关系,即可得到所有的查询结果.算法 6 给出了范围查找算法的描述.

对于 k-NN 查询,M^+-tree 采用了与 M-tree 中类似的查询机制,即采用启发式的方法依次去掉无效的子树分支.M^+-tree 使用一个优先队列 PR 存储指向有效子树的指针,同时利用一个 k 元素数组 NN 存放查询结果.

与 M-tree 相比,M^+-tree 在 k-NN 查询算法中加入了关键维过滤机制,同时,改善了对优先队列的操作.由于在 M^+-tree 中,孪生节点是成对出现的,因此,对于一对孪生兄弟节点,我们用一个标记 flag 来标识左右孪生兄弟节点是否都是有效的.如果一对孪生兄弟节点全有效,则将该标记置为 TRUE.只需要一次对列操作.在 k-NN 查询算法中,子过程 k-NN_NodeSearch 具有非常重要的地位,绝大部分查询操作都是在这个过程中实现的.算法 7 和算法 8 给出了最近邻查找算法和下一个优先节点查找算法的描述.

算法 6(范围查询).

输入:当前节点、查询对象和查询半径.

步骤:

(1) N 不是叶子节点,则进行以下操作:

(a) 对 N 中所有入口项利用三角不等式关系过滤,初步滤掉不符合条件的数据;

(b) 计算查询对象与未滤掉数据间的距离,进行第 2 次过滤;

(c) 过滤后未被滤掉的数据,进行关键维过滤;如果左分支符合条件,则对左分支进行递归范围查询;如果右分支符合条件,则对右分支进行递归范围查询;如果关键维未滤掉任何一个分支,则在两个子树上分别进行递归范围查询;

(2) 如果 N 是叶子节点,

(a) N 中的入口项,先通过三角不等式进行一次过滤;

(b) 对未被滤掉的数据通过对关键维的比较进行二次过滤;

(c) 对于二次过滤后仍未被滤掉的数据,计算该数据与查询对象间的距离,并将该距离与查询半径比较,将小于半径的数据放入查询结果集合;

(3) 返回查询结果.

算法 7(最近邻查询).

输入:指向根结点的指针 T,查询对象 Q 和查询结果返回个数 k.

步骤:

(1) 将 T 放入 PR,并将 NN 队列中各元素距离查询对象的距离初始化为无穷大;

(2) 如果 PR 不为空,则重复以下操作:

(a) 从 PR 中选择优先级最高的指针,得到下一步要查找的节点 Next_Node;

(b) 在 Next_Node 上调用 k-NN_NodeSearch;

(c) 如果 Next_Node 的标记为 TRUE,则在其孪生兄弟节点上调用 k-NN_NodeSearch;

(3) 返回查询结果集.

算法 8(最近邻查询中节点内查询算法描述).

输入:当前节点 N,查询对象 Q,查询结果返回个数 k.

步骤:

(1) 如果 N 不是叶子节点,则对所有 N 节点包含的入口项进行以下操作:

(a) 利用三角不等式进行一次过滤,初步滤掉不符合条件的数据;

(b) 若入口项未被滤掉,计算该入口项与查询对象间的距离;进行二次过滤;

(c) 若二次过滤仍未滤掉该数据,则进行关键维过滤,若一对孪生兄弟节点都可能包含结果元素,设置左节点的标记为 TRUE,并将左节点放入 PR,否则,将可能包含结果元素的节点标记 FALSE,并放入 PR;

(d) 如果 N 中元素距离查询对象的最大距离小于 $NN[k-1]$,则更新数组 NN,并将所有最小距离大于 $NN[k-1]$的子树从优先队列中删除;

(2) 如果 N 是叶子节点,则对所有 N 节点包含的对象进行以下操作:

(a) 利用三角不等式进行一次过滤,初步滤掉不符合条件的数据;

(b) 若该入口项未被滤掉,则计算该入口项与查询对象间的距离;

(c) 如果计算所得的距离小于 $NN[k-1]$,则更新数组 NN,并将所有最小距离大于 $NN[k-1]$的子树从优先队列中删除.

4 性能分析与评价

本节,我们从理论分析和实验评价两方面对 M-tree,M^+-tree 以及 Slim-trees 进行性能评估与比较.

4.1 理论分析

索引树的查询性能在很大程度上取决于树的高度.由于 M$^+$-tree 的中间节点入口项采用两层结构,使得每个中间节点入口项对应于两棵子树,从而使中间节点的扇出数增加将近 2 倍.随着扇出树的增加,树的高度降低了.而孪生兄弟节点之间的无重叠性保证了关键维过滤的有效性.这将大大减少查询时中间节点进行的距离计算次数和 I/O 访问次数.

理想状态下,可以对两种索引结构进行性能估算,性能估算参数包括:数据维数 n,磁盘页面大小 *pagesize*,入口项大小 *entrysize*,结点扇出 *fanout*,数据集大小 *DatasetSize*,树的高度 *Height*,I/O 次数 $C_{I/O}$,CPU 代价 C_{cpu}.

在两种索引结构中,树的高度为 $Height=\lfloor \log_{fanout}\ datasize \rfloor+1$.假设每一层都有 k 条路径,则 $C_{I/O}=k^{heigh}$.表 1 给出了两种索引结构中性能参数的估算公式.

Table 1 The comparison of parameters for M-tree and M$^+$-tree

表 1 M-tree 和 M$^+$-tree 中参数的比较

	M-tree	M$^+$-tree
entrysize	(*n*+2)**doublesize*	(*n*+4)**doublesize*
fanout	⌊*pagesize*/(*n*+2)**doublesize*⌋*66%	⌊*pagesize*/(*n*+4)**doublesize*⌋*2*66%
C_{cpu}	$C_{I/O}$**fanout*	$C_{I/O}$**fanout*/2

令数据集大小为 50 000,则表 2 给出了两种索引结构的实际参数值.

Table 2 The comparison of experiment parameters for M-tree and M$^+$-tree

表 2 两种索引的实验参数比较

Data dimension (*n*)	Maximal fanout (M)	Maximal fanout (M$^+$)	Height (M)	Height (M$^+$)
5	28	48	5	5
10	21	38	6	5
15	17	30	6	5
20	14	26	6	5
25	12	22	7	6
30	11	20	7	6
35	9	18	8	6
40	8	16	8	6

4.2 实验结果与评价

这部分给出了 M-tree,Slim-trees 和 M$^+$-tree 的范围查询和最近邻查询的性能测试结果,并对 3 种索引结构的性能进行分析比较.实验使用两类数据集:均匀数据和实际数据.从以下 4 方面比较它们的性能:(1) 数据集大小;(2) 数据空间维数;(3) 范围查询;(4) 最近邻查询.表 3 给出了性能测试中所使用的数据集描述,而表 4 给出了两种数据集特征描述.

Table 3 Data set used in the experiments

表 3 性能测试中的数据集

No.	Dim	Data set size	Type
(1)	10	10 000~80 000	Uniform
(2)	5~40	50 000	Uniform
(3)	10	50 000	Uniform
(4)	12	20 000	Uniform

Table 4 The representative about two data sets

表 4 两种数据集特征描述

Data set type	Property	Resource	Generation method
Uniform	Uniform	Synthetic	A time function
Real	Uneven	Real images	MPEG-7 tools

在均匀数据的范围查询和最近邻查询性能比较中,使用 10 维的数据集.数据各个维的特征值均匀分布在[0,1]的范围内.均匀数据是由一个时间函数随机产生的.当比较数据集大小对查询性能的影响时,数据集的大小从 10 000 变化到 80 000.当比较维数变化对查询性能的影响时,采用的数据维数从 5 增加到 40.在用实际数据进行查询性能分析的实验中,数据为 12 维,这些数据是利用 MPEG-7 特征提取工具提取得到的,通过标准化,将数据转化到[0,1]范围的空间中.所有测试使用的软硬件环境如下:(1) 硬件环境:PIV 1.5GHz CPU,256MB SDRAM 内存,80GB 硬盘;(2) 操作系统平台:Microsoft Windows 2000 Professional;(3) 数据库系统:对象数据库系统

Fish;(4) 编程环境:Microsoft Visual C++编译器.

4.2.1 优先队列访问与查询性能

M^+-tree 和 M-tree 以及 Slim-trees 的最近邻查询都采用启发式规则进行优先队列的访问,选择优先子树.对优先队列访问不仅需要许多运算,而且查找最优子树时需要对优先队列中各个子树进行排序的过程.这个过程代价相当大.

我们使用一组实际数据集来测试下面 4 种情况下 *k*-NN 查询的性能:(1) M-tree;(2) M^+-tree,但孪生兄弟节点分开处理;(3) M^+-tree,孪生兄弟节点作为一个整体进行处理;(4) Slim-trees.图 3(b)表明了 M^+-tree 中队列优化前后的性能对比.由图 3(c)和图 3(d)看出,在进行队列优化前后,M^+-tree 的 I/O 和距离计算次数几乎没变,但优化后的 M^+-tree 极大地减少了队列访问次数,响应速度极大地提高.可见,优先队列访问是影响索引最近邻查询的重要因素之一.

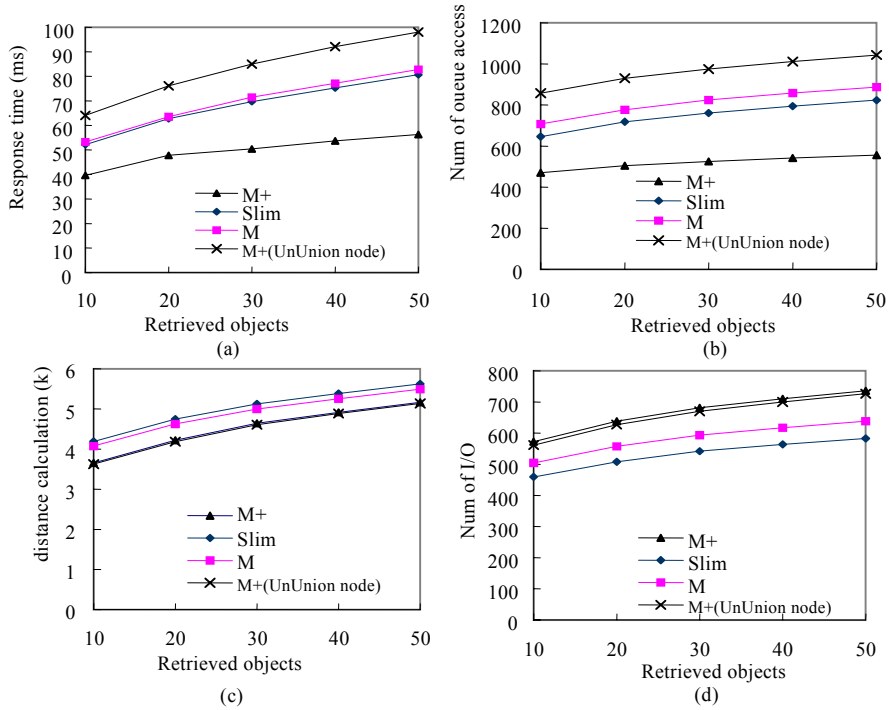

Fig.3 Effect of priority queue on indexes in *K*-NN search

图 3 最近邻查找中优先队列对查询性能的影响

4.2.2 数据集大小与查询性能

为了比较 M^+-tree,M-tree 以及 Slim-trees 的性能.我们研究了随着数据集大小变化这 3 种索引技术的性能变化.实验中,我们采用一个 10 维的均匀数据集.数据集的大小从 10 000 变化到 80 000,返回结果对象个数为 10.实验使用加权的欧几里德距离作为度量对象相似性的标准.我们利用 MPEG-7 工具包提取了 20 000 个图像的信息,并对 20 000 个图像信息进行分析,发现特征数据的每个维的值都在一个特定的范围内.因此,我们选用各个维的最大值的平方作为在该维上的权值,这样做可以使均匀数据更好地模拟真实数据.

图 4 反映了在最近邻查询中,当数据集大小发生变化时,M-tree,Slim -trees 和 M^+-tree 的性能对比情况.在研究 M-tree 和 M^+-tree 性能随数据集变化的实验中,主要考虑 4 个因素:(1) 查询相应时间;(2) 距离计算次数;(3) I/O 访问次数;(4) 优先队列的访问次数.

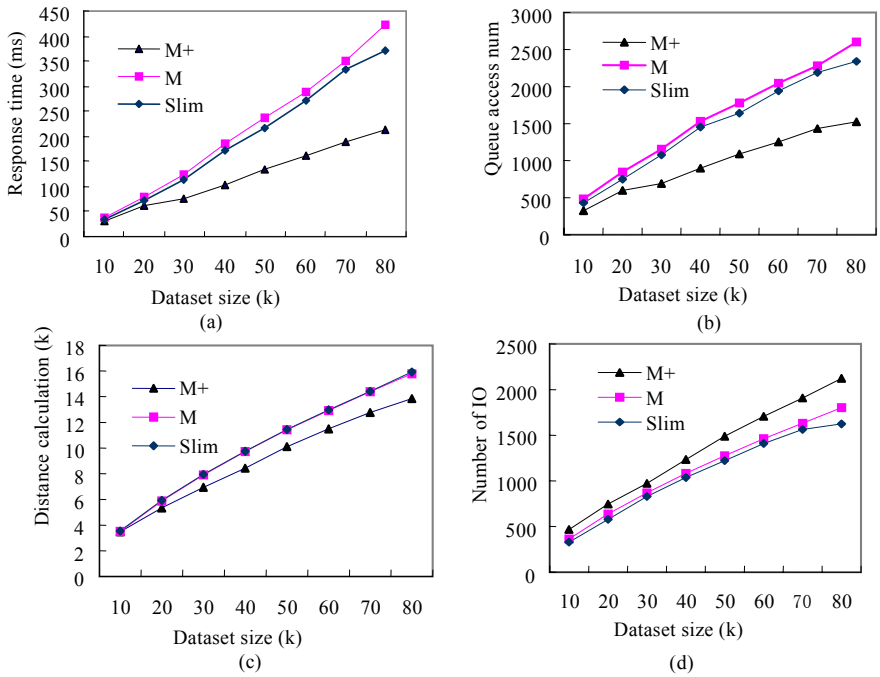

Fig.4 *K*-NN search of indexes with data set size changing

图 4 数据集大小变化时 3 种索引的最近邻查找

从图 4 可以看出,M^+-tree 的 I/O 操作比 M-tree 要大,然而,它的查询响应时间比 M-tree 要快.为什么会出现这种情况呢?这主要有几方面的原因:首先,在 M^+-tree 和 M-tree 中都采用了启发式原则.查询开始时,查询半径初始化为最大值.随着查询的进行,查询半径由大变小.但是,在查询半径非常大时,关键维的过滤效果非常差.因此 M^+-tree 有更多的距离计算和 I/O 次数.然而,启发式原则是通过大量对优先队列的操作而实现的,而且其中需要大量的排序和计算操作,这些操作代价相当大.而在 M^+-tree 中引入了一个孪生兄弟节点的概念,当进行对优先队列的操作时,一对孪生兄弟节点可以当作一个节点来处理,这种处理方法极大地减少了对优先队列的访问次数.这一点从图 4(b)中可以清楚地体现出来.由于队列操作的显著减少,而且,M-tree 需要较少的距离计算,因此,查询响应时间显著减少.随着数据集的增大,这种优越性体现得就越好.对于实际数据,M^+-tree 的优越性会越明显.

Slim-trees 继承了 M-tree 所有的查询算法,它的改进主要在于它减少了索引的节点个数,增加了节点的利用率.同时,由于节点个数的减少,I/O 次数和对优先队列的访问相应减少.但是由于节点利用率的提高,每个节点包含较多入口项,与 M-tree 相比,Slim-tree 的距离计算次数并没有减少.虽然 M^+-tree 的 I/O 次数略高于 M-tree 和 Slim-trees,但它的队列访问次数和距离计算次数都远远少于其他两种索引技术,响应速度比其他两种索引结构要快得多.从图 4 可以看出,与 Slim-trees 相比,M^+-tree 是一种更为有效的索引技术.

4.2.3 数据集维数与查询性能

图 5 显示了在最近邻查找中,随着数据空间维数的变化,M-tree,Slim-trees 以及 M^+-tree 的性能变化和对比情况.在比较维数对这 3 种索引技术性能的影响实验中,我们采用包含 50 000 数据的均匀数据集.数据空间的维数从 5 变化到 40.从图 5 中可以看出,3 种索引结构的距离计算次数和 I/O 次数都比较接近,特别是当维数升高到 25 以后,由于在 3 种索引结构中,所有的子空间都重叠在一起,因此,进行查询几乎需要遍历整个索引树.这时,查询的性能几乎完全取决于队列访问次数的多少.从图 5(b)可以看出,M^+-tree 的队列访问次数远远少于 M-tree 的访问次数,因此,M^+-tree 比其他两种索引技术有更好的查询性能.

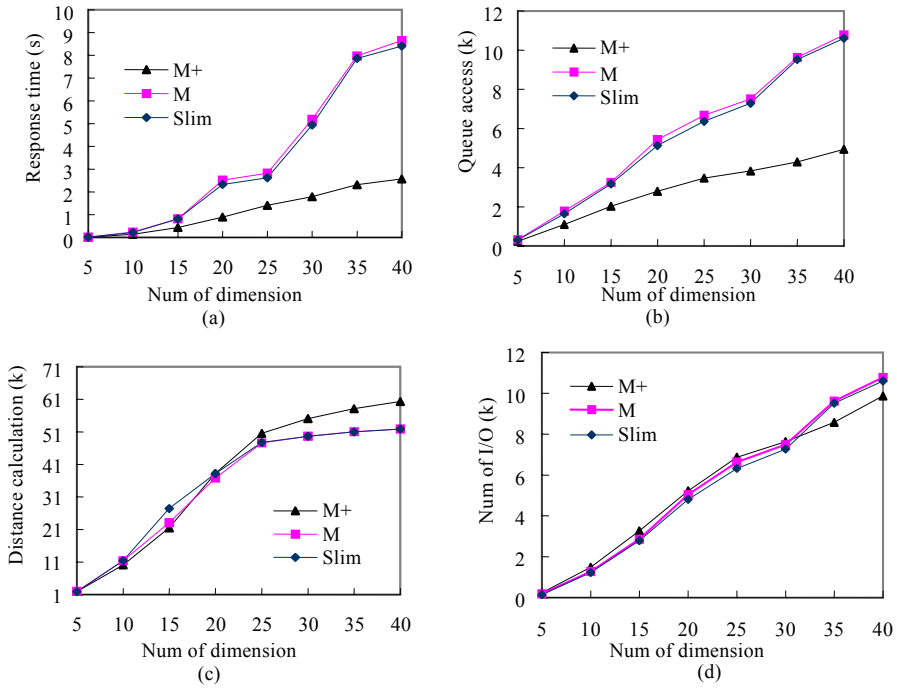

Fig.5 *K*-NN search for indexes by dimension varying
图 5 维数变化时 3 种索引技术的最近邻查找

4.2.4 范围查询

我们利用均匀分布的数据,对 3 种索引技术范围查询性能进行分析、比较,并给出实验对比结果.我们从以下 3 方面比较它们的性能:(1) 响应时间;(2) 距离计算次数;(3) I/O 次数.

图 6 显示了进行范围查询时,3 种索引的性能对比情况.从性能图上可以看出,当查询半径小的时候,M^+-tree 比 M-tree 需要更少的距离计算次数、I/O 次数 和查询响应时间,M^+-tree 的性能远比 M-tree 要好.当查询半径为 0 时,由于每次关键维过滤都能滤掉一半的数据,孪生兄弟节点间的过滤都是有效的,因此 M^+-tree 的性能远远高于 M-tree,查询响应速度比 M-tree 提高两倍以上,甚至高达 10 倍.随着查询半径的增加,关键维的过滤能力减弱,两种索引结构的性能对比逐渐减弱.

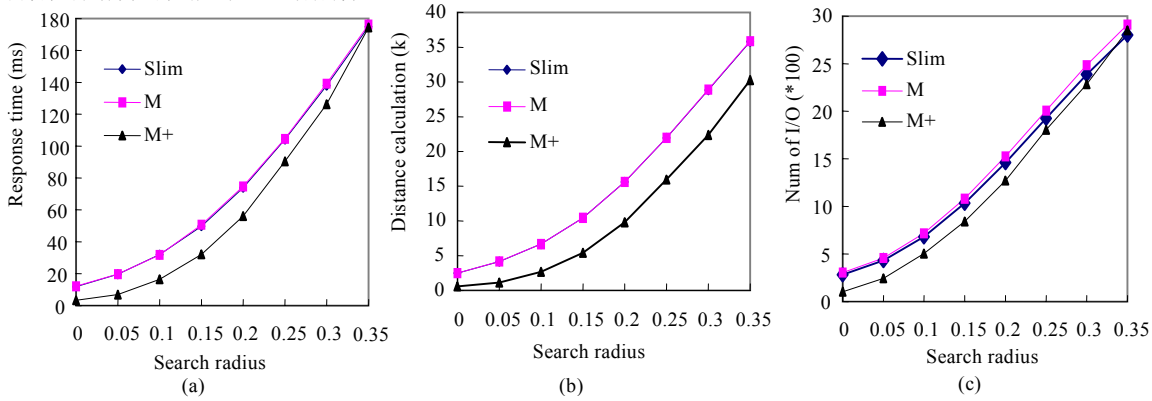

Fig.6 Range search for M-tree, Slim-tree and M^+-tree
图 6 M-tree,Slim-tree 和 M+-tree 的范围查询

从性能曲线图中可以看出,M^+-tree 的距离计算次数较少,随着查询半径的增大,逐渐趋于接近.同时,由于随着查询半径的增大,M^+-tree 的 I/O 操作次数增长较快,因此随着半径的增大,两条 I/O 性能曲线出现了交叉.由于

数据的高维性,而且距离计算次数相差较大,因此,尽管两者在某些情况下,M+-tree 需要较多的 I/O 操作,但总的查询性能仍然比 M-tree 要好.而 Slim-trees 的 I/O 次数比 M-tree 稍有减少,但距离计算次数没有丝毫减少,在查询响应上,几乎看不到有多大改善.对范围查找来说,M+-tree 是一种更为有效的索引技术.

4.2.5 最近邻查询

图 7 中显示了采用实际数据时,M+-tree 与 M-tree 及 Slim-trees 在进行最近邻查找时的性能以及三者的对比情况.对于最近邻查询,实验从距离计算次数、I/O 操作次数、优先队列访问次数以及查询响应时间 4 方面进行性能测试分析并比较.从图中可以看出,在最近邻查询中,M+-tree 具有最少的优先队列访问次数和距离计算次数,而且,与其他两种索引技术相比,这种改进程度非常大,只有 I/O 访问次数比其他两种索引技术多一些.总之,由于受其中两个影响查询性能的主要因素的影响,使用 M+-tree 索引进行相似性检索更快.

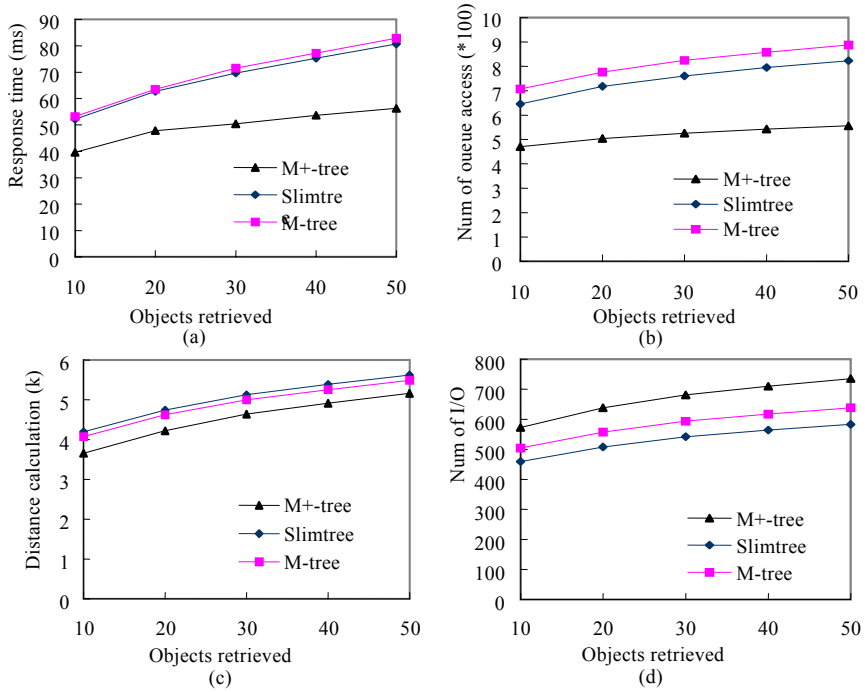

Fig.7 *K*-NN search for indexes

图 7 3 种索引技术的最近邻查询

5 结 论

高维索引是基于内容相似性查询的有效方法,而数据空间的分割直接影响索引的查询性能,本文针对基于内容查询和高维数据的特点,提出了一种新的数据分割策略,即在距离分割的基础上进行基于关键维的无重叠分割,并在此基础上将关键维过滤的思想引入索引技术中,提出了基于度量的高维索引方法——M+-tree.M+-tree 是一种动态可更新的分页平衡树.它以 M-tree 为基础,引入二元 MVP-tree 二次过滤的思想.同时,M+-tree 提出了两个新的概念——关键维和孪生节点.将关键维和关键维转移的理论以一种全新的方式运用到索引中去,同时利用关键维进行孪生兄弟节点间的有效过滤.本文同时给出了 M+-tree 的设计和性能评测结果.我们的实验结果表明了 M+-tree 的优越性,与 M-tree 相比,M+-tree 的速度大大提高了.在一般情况下,M+-tree 的查询性能比 M-tree 提高了 20%以上,甚至在某些情况下可以提高到 10 倍以上.

References:

[1] Guttman A. R-Trees: A dynamic index structure for spatial searching. In: Yormark B, ed. Proc. of the ACM SIGMOD Conf. Boston, 1984. 47~57.

[2] Berkmann N, Krigel HP. Schneider R, Seeger B. The R*-tree: An efficient and robust access method for points and rectangles. In: Hector GM, Jagadish HV, eds. Proc. of the ACM SIGMOD Conf. Atlantic, 1990. 322~331.

[3] Katayama N, Satoh S. The SR-tree: An index structure for high-dimensional nearest neighbor queries. In: Peckham J, ed. Proc. of the ACM SIGMOD Conf. Tucson, 1997. 369~380.

[4] White DA, Jain R. Similarity indexing with the SS-tree. In: Stanley YWS, ed. Proc. of the 12th Int'l Conf. on Data Engineering. New Orleans: IEEE Computer Society, 1996. 516~523.

[5] Lin K-I, Jagadish HV, Faloutsos C. The TV-tree: An index structure for high-dimensional data. VLDB Journal, 1994,3(4):517~542.

[6] Ciaccia P, Patella M, Zezula P. M-tree: An efficient access method for similarity search in metric spaces. In: Jarke M, Carey MJ, Dittrich KR, Lochovsky FH, Loucopoulos P, Jeusfeld MA, eds. Proc. of the 23rd VLDB Conf. Athens: Morgan Kaufmann Publishers, 1997. 426~435.

[7] Bozkaya T, Ozsoyoglu M. Distance-Based indexing for high-dimensional metric spaces. In: Peckham J, ed. Proc. of the ACM SIGMOD Conf. on Management of Data. Tucson, 1997. 357~368.

[8] Ishikawa M, Chen H, Furuse K, Yu JX, Ohbo N. MB+tree: A dynamically updatable metric index for similarity search. In: Lu HJ, Zhou AY, eds. Proc. of the 1st Int'l Conf. on Web Age Information Management. Lecture Notes in Computer Science 1846, Springer-Verlag, 2000. 356~373.

[9] Jr CT, Traina A, Seeger B, Faloutsos C. Slim-Trees: High performance metric trees minimizing overlap between nodes. In: Zaniolo C, Lockemann PC, Scholl MH, Grust T, eds. Proc. of the 7th Int'l Conf. on Extended Database Technology. Lecture Notes in Computer Science 1777, Konstanz: Springer-Verlag, 2000. 51~65.

[10] Zhou XM, Wang GR, Yu G. A research of index methods in metric space. Computer Science, 2002,29(B):265~267 (in Chinese with English abstract).

[11] Zhou X, Wang G, Yu JX, Yu G. M^+-tree: A new dynamical multidimensional index for metric spaces. In: Schewe KD, Zhou AY, eds. Proc. of the 4th Australian Database Conf. Adelaide, 2003. 161~168.

附中文参考文献:

[10] 周项敏,王国仁,于戈.度量空间中索引方法的研究.计算机科学,2002,29(B):265~267.