多维向量动态索引结构研究*

周学海[1], 李 曦[1], 徐海燕[2], 龚育昌[1], 赵振西[1]

[1](中国科学技术大学 计算机科学与技术系,安徽 合肥 230027);
[2](浙江大学 计算机科学与工程系,浙江 杭州 310027)
E-mail: xhzhou@ustc.edu.cn

摘要: 多维向量的索引技术是多媒体数据库系统中的关键技术之一.集中研究基于向量空间模型的动态索引结构,以解决在图像数据库系统中按内容快速检索图像的对象问题.在分析研究 R-Tree 和 R*-Tree 的基础上,提出了 ER-Tree 动态索引结构.该索引树用超球体划分多维向量空间,以有利于计算最近邻;吸取 R*-Tree 树的重插技术,以增强索引树对数据集整体特征的表达能力,从而提高检索效率;通过引入插入安全点和删除安全点概念,有效地提高建树的效率.同时,给出了基于该结构的特征向量插入算法.实验结果表明,所提出的索引结构建树的效率比 R*-Tree 提高 10 倍,检索的有效性也有明显的提高.

关 键 词: ER-Tree;动态索引结构;相似性检索
中图法分类号: TP311 文献标识码: A

随着多媒体信息的增多,如何有效地检索多媒体对象越来越受到人们的关注.一般采用的方法是:在多媒体对象存档的同时提取对象的特征,建立多媒体对象的特征库.在进行检索时,通过进行特征的相似性检索来存取多媒体对象.为了提高检索的效率,需要使用索引机制以加速搜索的过程.

最早提出的最近邻检索算法是 Elias's 算法.Rivest,Cleary 以及 Arya 和 Mount[1]在假设点集为均匀分布的情况下分析了这类算法的平均复杂度为 $O(2^d)$.目前最好的静态索引结构和检索算法是 Arya 和 Mount[1]在 Friedman,Bentley 和 Finkel 等人工作的基础上提出来的.但在实际应用中,我们可能并不知道全部数据集,需要动态的索引结构来支持这类应用.

直到最近,数据库领域的研究者们开始对低维数据集的索引和搜索类型(包括最近邻和相似性检索)进行研究.许多索引结构如 grid file[2]、线性四分树[2]并不能扩展到高维数据.可以扩展到高维数据集,并得到了普遍应用的动态索引结构有 R-Tree 和 R*-Tree[3].

本文集中研究基于向量空间模型的相似性检索所需的动态索引结构.在向量空间模型中,对象由固定维数的向量表示.对于给定的一组对象,研究其动态索引结构.本文第 1 节描述了我们所提出的索引结构——ER-Tree,并给出了实验结果,最后给出了本文的小结.

1 多维数据集的动态索引结构 ER-Tree (Extended R*-Tree)

动态索引结构的关键是多维特征空间划分策略以及数据集整体特征的表达,这两方面直接影响到特征向量检索、插入和删除算法的效率.R-Tree 和 R*-Tree 均采用超立方体对多维向量空间进行划分,R*-Tree 的节点插入效率低于 R-Tree,基于 R*-Tree 的检索算法明显优于 R-Tree.我们在对已有的高维数据集动态索引结构加以

* 收稿日期: 2000-03-21; 修改日期: 2001-02-16
基金项目: 国家自然科学基金资助项目(69503005);安徽省自然科学基金资助项目(99043302)

作者简介: 周学海(1966-),男,江苏武进人,博士,教授,主要研究领域为信息系统,计算机体系结构;李曦(1963-),男,安徽合肥人,博士,高级工程师,主要研究领域为计算机体系结构;徐海燕(1974-),女,博士,主要研究领域为超媒体系统;龚育昌(1943-),女,教授,博士生导师,主演研究领域为数据库技术;赵振西(1937-),男,教授,博士生导师,主要研究领域为智能软件开发,多媒体技术.

分析的基础上,对 R*-Tree 进行了有效的改进,采用超球体划分多维特征空间,以便有效地计算最近邻,通过引入 R*-Tree 的重插技术来提高该结构对数据集的整体表达能力,已达到提高检索效率的目的.为了克服 R*-Tree 的插入、删除算法较慢的缺陷,我们引入插入安全点和删除安全点的概念,以期能够提高特征向量插入和删除算法的效率.实验结果表明,本文所采用的方法是有效的,具体实验结果见第 1.3 节.

1.1 ER-Tree-多维向量动态索引树

在 ER-Tree 树中,用超球体来对特征空间进行划分,特征向量存储在叶节点,内部节点用于特征空间的划分.基本结构如图 1 所示.

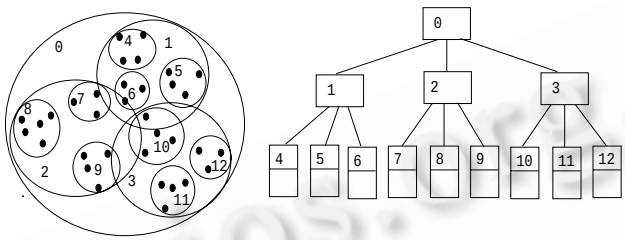


Fig.1 Feature space partition and ER-Tree structure

图 1 特征空间的划分及相应的 ER-Tree 结构

1.1.1 ER-Tree 树及其安全点定义

定义 **1.1**. 设桶的最大容量为 *M*,最小容量为 *m*,则 ER-Tree 具有以下特征:

(1) 所有叶节点处于同一层;

(2) 叶节点所含特征向量的个数在 *m* 和 *M* 之间;

(3) 除根节点以外,非叶节点所含子节点数在 *m* 和 *M* 之间;

(4) 非叶节点所表示的超球体,完全包含其子节点所表示的超球体;

(5) 根节点若为非叶节点,则必有两个子节点.

定义 **1.2**. 在节点最大容量为 *M*,最小容量为 *m* 的 ER-Tree 树上:

(1) 容量为 *M* 的节点称为插入不安全节点,否则称为插入安全点;

(2) 容量为 *m* 的节点称为删除不安全节点,否则称为删除安全点.

1.1.2 ER-Tree 树节点的数据结构

ER-Tree 树中的叶节点和内部节点用统一的数据结构表示.用 C 语言描述如下:

```
struct ERNODE
{
    int nHighLevel;                    //节点所在的层号,叶节点的层号为 0
    union{
        struct ERNODE*ptrChildNode[_MAX_INTERNAL_CAPACITY];
        FEATURE_DATA*ptrData[_MAX_LEAF_CAPACITY];
    } uNextNode;
    int nNodeCapacity;                 //节点所含的子节点数
    int nPointNum;                     //该节点所代表子树所含的点数
    float fRadius;                     //节点的半径
    float fCentroid[_DIMENTION];       //节点的中心坐标
};
```

*n*HighLevel:标识当前节点在树中的层号;*u*NextNode:指向其下一层节点,若当前节点为叶节点,则指向特征向量,若当前节点为非叶节点,则指向其子节点;*n*NodeCapacity:表示当前节点所含的子节点数;*f*Radius:表示当前超球体的半径;*f*Centroid:表示当前超球体的中心位置.

1.2 特征向量的插入

R*-Tree 通过重插技术改善了对数据集整体特征的表达,提高了系统的查询效率,但正是因为进行了重插,使得节点分裂的可能性增加,从而使得插入算法要多次扫描索引树,插入一个特征向量的平均时间增加.为了克服这一缺陷,我们引入了插入安全点[4,5]和删除安全点的概念,使得分裂只涉及父子两级节点,插入一个节点只需扫描一次索引树.

ER-Tree 树的基本操作有插入操作和删除操作.

特征向量插入索引树的方法借鉴 R*-Tree 的重插技术以及预操作思想,在从根向叶节点的搜索过程中,若发现插入不安全点,则立即进行分裂处理,使得整个操作只需从上到下扫描一遍,其过程如下:

(1) 若根为插入不安全节点,则对其分裂;

(2) 运用子树选择算法选择子节点,如果该节点为插入不安全节点,调用溢出处理过程对该节点进行处理.调整所选节点的中心和半径,重复步骤(2),直到某一叶节点.

(3) 将待插特征向量插入.

显然,该算法插入一向量只需要从根节点开始一遍扫描 ER-Tree 树,不存在分裂自底向根的传播过程,因此,其插入效率明显优于 R*-Tree.该算法涉及子树选择策略以及节点溢出处理策略.

1.2.1 子树选择策略

子树选择算法用来选择插入路径,其返回值为所选子节点的地址.选择子节点的基本策略为:

规则 1. 若当前节点为叶节点,返回.

规则 2. 若当前节点为内部节点,且其所在层 *n*HighLevel 为 1,则从其子节点中选择在插入点插入后与其兄弟节点覆盖最小的节点.若存在覆盖相等的节点,则选择半径增加最小的,若半径增加也相同,则选择半径小的叶节点,若半径也相同,则选择包含点少的叶节点,若所含点数也相同,则选择前面的节点.

规则 3. 若当前节点为内部节点,且其所在层 *n*HighLevel>1,则从其子节点中选择到该节点中心点最近的节点,若这样的节点有若干个,则选择节点半径较小的,若半径也相同,则选择所含节点数较少的,否则选择前面的节点.

1.2.2 节点溢出处理

考虑数据分布的全局性,我们借鉴 R*-Tree 的重插技术来处理节点溢出.当在选择插入路径时,所选节点为不安全插入节点,若当前节点层未进行过重插处理,则对该节点进行重插处理,否则对其进行分裂处理.若当前节点层已进行重插处理后,仍然是不安全节点,则进行分裂处理.分裂的基本策略为:

(1) 选择在该节点所含的项中,距离最远的两项作为种子项,形成两组,每组含有一项,我们用 group1,group2 表示.

(2) 若当前节点为叶节点,则该节点中的其他项按规则 3 选择相应的组插入,在插入过程中,若其中一组所含项数等于 $M-m-1$,则将其余项插入另一组,然后与其父节点相连.

(3) 若当前节点为内部节点,则对该节点中的其他项按规则 2 选择相应的组插入,在插入过程中,若其中一组所含项数等于 $M-m-1$,则将其余节点插入另一组,若当前节点为根节点,则生成新的根节点,否则将划分好的两组与其父节点相连.并调整父节点.

1.3 实验结果

在 SUN SPARCII(32M 内存)、Sunos4.1.2 条件下我们对所提出的索引结构进行了实验.选用:

· 均匀分布的 2D,3D,4D,5D,6D,7D,8D,9D,10D 50000 点.

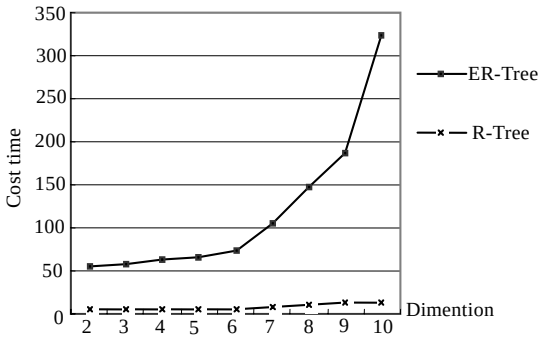
· 正态分布的 2D,3D,4D,5D,6D,7D,8D,9D,10D 50000 点.

进行了 ER-Tree,R-Tree 和 R*-Tree 创建索引的性能比较以及检索效率的比较.

1.3.1 创建检索树性能比较

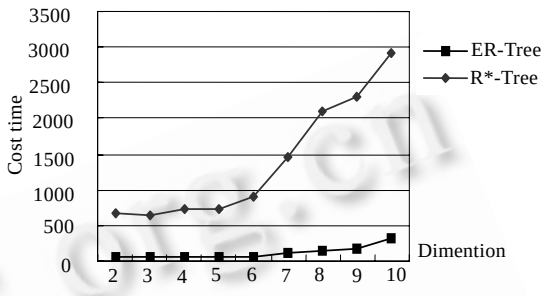
图 2(a)~图 2(d)分别列出了均匀分布和正态分布的 50 000 个点,当取 M=30 时,创建 R-Tree,R*-Tree 和 ER-Tree 索引结构所需时间(单位为秒).由实验结果可知,R-Tree 建树耗费最小,R*-Tree 建树耗费最大,ER-Tree

建树过程大约比 R*-Tree 快 10 倍.R-Tree 之所以建树过程较快,主要是因为其子树选择算法较简单、计算量较小,但从后面的实验可知,R-Tree 的检索性能是最差的,而 ER-Tree 和 R*-Tree 子树选择算法计算量相当,ER-Tree 之所以建树较快,得益于我们引入的插入安全点,使得插入一个节点只需一遍扫描树,在节点分裂时,无须向上传播.



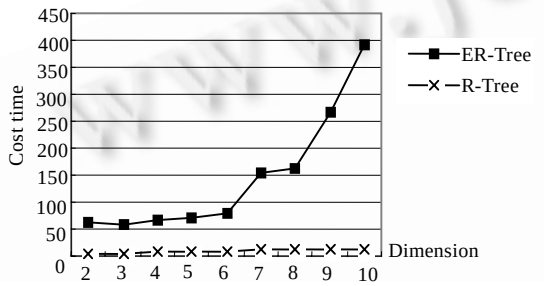
(a) Time of creation ER-Tree and R-Tree (uniform distribution)

(a) 创建 ER-Tree 和 R-Tree 所需时间(均匀分布)



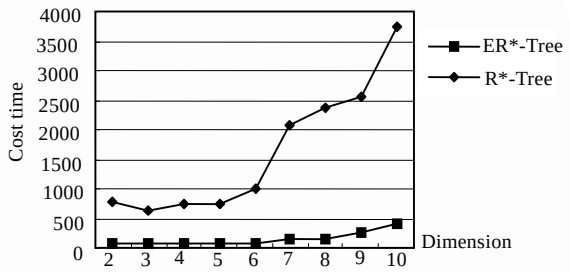
(b) Time of creation ER-Tree and R*-Tree (uniform distribution)

(b) 创建 ER-Tree 和 R*-Tree 所需时间(均匀分布)



(c) Time of creation ER-Tree and R-Tree (gaussian distribution)

(c) 创建 ER-TREE 和 R-TREE 所需时间(正态分布)



(d) Time of creation ER-Tree and R*-Tree (gaussian distribution)

(d) 创建 ER-Tree 和 R*-Tree 所需时间(正态分布)

Fig.2

图 2

1.3.2 K-最近邻检索算法

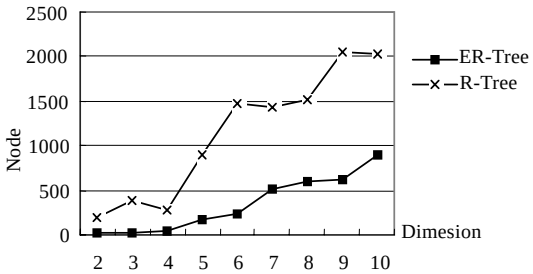
我们比较了 ER-Tree,R*-Tree 和 R-Tree 的 3 种最近邻检索算法的效率.一种方法是一般的查询算法(K-NN 算法),另一种方法是基于优先级的检索策略(PRK-NN),最后一种是近似检索算法.具体算法参见文献[5],实验结果如图 3 所示.

(1) 精确的 K-最近邻检索性能比较图 3(a)~图 3(f)分别列出了 M=30 时均匀分布和正态分布情况下,进行 10-最近邻检索各类树所需访问的节点数(单位:个).实验结果表明,无论是正态分布还是均匀分布,ER-Tree 的检索效率明显优于 R*-Tree,R-Tree 的检索效率是最差的.无论是 R*-Tree 还是 ER-Tree,基于优先级的 K-NN 检索算法优于标准的 K-NN 算法.

(2) 近似检索的性能测试

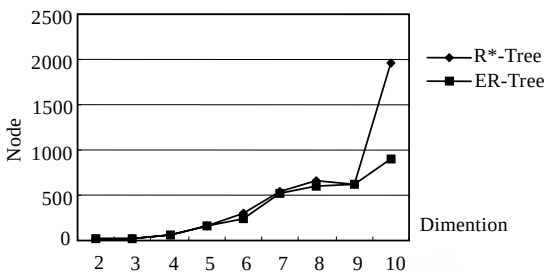
表 1 列出了在概率误差为 1‰的情况下,ε的取值以及对查询性能的影响.其中 Exact 表示 ER-Tree 在精确检索的情况下基于优先级的 K-NN 算法需访问的叶节点数(单位:个),LeafNum 表示近似检索时需访问的节点数(单位:个).

测试结果表明,如果查询结果允许有 1‰的误差,那么访问叶节点的数量平均可减少 20~30%.



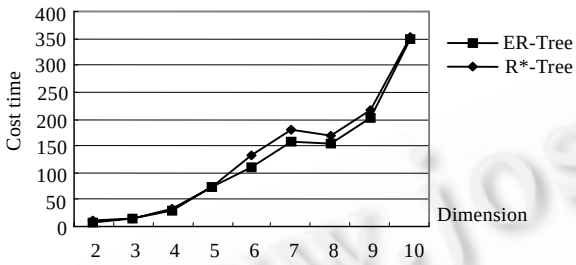
(a) Performance of K-NN retrieval on ER-Tree and R-Tree (uniform distribution)

(a) 均匀分布 ER-Tree 和 R-Tree 的 K-NN 检索性能



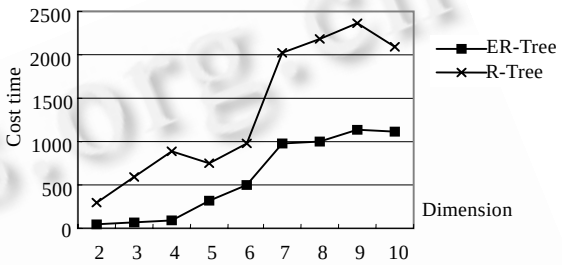
(b) Performance of K-NN retrieval on ER-Tree and R*-Tree (uniform distribution)

(b) 均匀分布 ER-Tree 和 R*-Tree 的 K-NN 检索性能



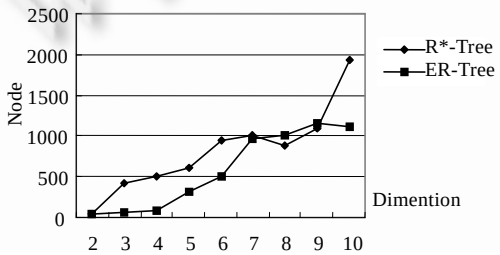
(c) Performance of PRK-NN retrieval on ER-Tree and R*-Tree (uniform distribution)

(c) 均匀分布 ER-Tree 和 R*-Tree 的 PRK-NN 检索性能



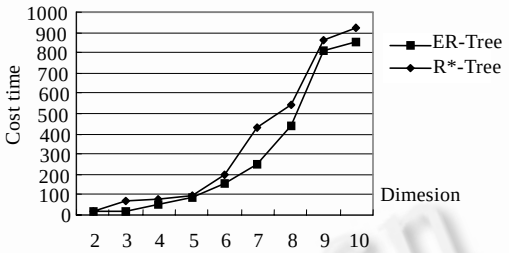
(d) Performance of K-NN retrieval on ER-Tree and R-Tree (gaussian distribution)

(d) 正态分布 ER-Tree 和 R-Tree 的 K-NN 检索性能



(e) Performance of K-NN retrieval on ER-Tree and R*-Tree (gaussian distribution)

(e) 正态分布 ER-Tree 和 R*-Tree 的 K-NN 检索性能



(f) Performance of PRK-NN retrieval on ER-Tree and R*-Tree (gaussian distribution)

(f) 正态分布 ER-Tree 和 R*-Tree 的 PRK-NN 检索性能

Fig.3

图 3

Table 1 Performance of approximate retrieval

表 1 近似检索的性能测试结果

Dimesion①	Uniform distribution②			Gaussion distribution③		
	Exact	LeafNum	ε	Exact	LeafNum	ε
2	7.1	6.9	0.01	20.2	20.1	0.01
3	14.2	13.1	0.02	21.3	20.9	0.01
4	30.2	26.3	0.06	51.4	35.8	0.04
5	73.3	53.2	0.10	85.0	62.4	0.03
6	109.5	70.8	0.21	159.5	108.2	0.14
7	158.6	95.5	0.20	249.2	157.9	0.21
8	154.2	89.4	0.31	437.7	253.3	0.30
9	201.8	100.2	0.32	811.1	446.5	0.36
10	350.5	160.5	0.42	853.3	427.4	0.38

①维数,②均匀分布,③正态分布.

2 小 结

本文对动态索引结构进行了细致的研究,提出了一种新的动态索引结构.通过研究对中维和高维数据集进

行 K-最近邻检索,我们得到如下结论:搜索时间随着数据集维数的增长呈指数增长.在均匀分布的情况下,许多算法搜索时间随着维数呈指数增长.在实际应用中,系统的性能不仅与数据集的分布有关,而且与数据集的大小和索引结构有关;对中维和高维检索,近似的最近邻将会极大地提高系统的性能,并且近似查询在许多应用中相当普遍.实验表明,我们所提出的动态索引结构 ER-Tree,其索引创建时间比 R*-Tree 提高 10 倍,检索性能也明显优于 R-Tree 和 R*-Tree.

ER-Tree 的创新特色为:以超球体对多维向量空间划分,并采用重插技术,有效地提高了检索效率.引入安全点和不安全点的概念,提高了索引树的建树效率.

References:

[1] Sunil, A. Nearest neighbor searching and applications [Ph.D. Thesis]. Computer Vision Laboratory, Center for Automation Research, University of Maryland, College Park, MD 20742-3275, 1995.

[2] Hanan, S. The Design and Analysis of Spatial Data Structures. New York: Addison-Wesley Publishing Company, Inc., 1989.

[3] Beckmann, N., Kriegel, H-P., Schneider, R., *et al*. The R*-tree: an efficient and robust access method for points and rectangles. In: Clifford, J., King, R., eds. Proceedings of the ACM SIGMOD International Conference on the Management of Data. Denver, Colorado: ACM Press, 1991. 322~331.

[4] Gong, Yu-chang, Wang, Wei-hong. e-B+-tree: an indexing organization optimized for DBMS supporting multi-users. Journal of Software, 1996,7(5):314~320 (in Chinese).

[5] Zhou, Xue-hai. Research on image database system with content-based retrieval [Ph.D. Thesis]. Hefei: University of Science and Technology of China, 1997 (in Chinese).

附中文参考文献:

[4] 龚育昌,王卫红.e-B+树:面向多用户数据库系统优化的索引技术.软件学报,1996,7(5):314~320.

[5] 周学海.基于内容检索的图象数据库系统研究[博士学位论文].合肥:中国科学技术大学,1997.

Research on Dynamic Indexing Structure for Multi-Dimensional Vectors*

ZHOU Xue-hai[1], LI Xi[1], XU Hai-yan[2], GONG Yu-chang[1], ZHAO Zhen-xi[1]

[1](*Department of Computer Science and Technology, University of Science and Technology of China, Hefei* 230027,*China*);

[2](*Department of Computer Science and Engineering, Zhejiang University, Hangzhou* 310027, *China*)

E-mail: xhzhou@ustc.edu.cn

Abstract: Multi-Dimentional vectors indexing is one of the key technologies in multimedia database systems. This paper focuses on the dynamic indexing structure based on the vector space. An ER-Tree indexing structure is proposed which is efficient for medium and high dimensional vectors. In the ER-Tree, the efficiency of retrieval could be improved by the ways of partition of the vector space with hyper-sphere and R*-Tree's re-insert technology introduced to enhance ER-Tree's capability to represent the data-sets' features. By introducing safe-inserted-node and safe-deleted-node concept into the structure, the performance of tree creation is reenforced. The algorithms are also given for the creation of ER-Tree. The empirical test results show that (1) the tree creation algorithm proposed is about 10 times faster than R*-Tree. (2) the effectiveness of retrieval is highly improved.

Key words: ER-Tree; dynamic indexing structure; similarity retrieval

* Received March 21, 2000; accepted February 16, 2001

Supported by the National Natural Science Foundation of China under Grant No.69503005; the Natural Science Foundation of Anhui Province of China under Grant No.99043302