支持k近邻查询的X*树索引结构

章德斌[1],曹丽君[2],梁永欣[1],张忠平[1]
ZHANG Debin[1],CAO Lijun[2],LIANG Yongxin[1],ZHANG Zhongping[1]
1.燕山大学 信息科学与工程学院,河北 秦皇岛 066004
2.河北科技师范学院,河北 秦皇岛 066004
1.College of Information Science and Engineering,Yanshan University,Qinhuangdao,Hebei 066004,China
2.Hebei Normal University of Science & Technology,Qinhuangdao,Hebei 066004,China

ZHANG Debin,CAO Lijun,LIANG Yongxin,et al.X*tree index structure for *k* nearest neighbor queries.Computer Engineering and Applications,2011,47(5):123-125.

Abstract: By analyzing the inefficiency of *k* nearest neighbor query in existing index structures,this paper presents X*tree index structure which is suitable to perform *k* nearest neighbor query.A new node splitting algorithm is adopted,and the Split History field is omitted.The experiment shows that it has better performance than X*tree in time and space complexity,and it is more suitable to *k* nearest neighbor query.

Key words: *k* nearest neighbor query;high dimensional index structure;node splitting;weighted overlap

摘 要: 通过分析已有的索引结构在进行k近邻查询时效率上的不足,提出了适合进行k近邻查询的X*树索引结构,采用了新的结点分裂算法,同时不需要额外存储结点分裂的历史信息。实验结果表明它比X树的时间和空间性能更好,更适合k近邻查询的应用。

关键词: k近邻查询;高维索引结构;结点分裂;带权重叠率
DOI:10.3778/j.issn.1002-8331.2011.05.037 文章编号:1002-8331(2011)05-0123-03 文献标识码:A 中图分类号:TP311

1 引言

在基于互联网的应用迅速涌现的背景下,大量的高维数据(如图像、生物信息、医学成像、时间序列等)需要通过索引结构来支持高效的k近邻查询。X树(eXtended node tree)[1]索引结构就是这一需求的产物,它针对R树[2]不适合对中高维数据进行索引的问题,提出了超结点(Supernode)概念,从而很好地解决了中等维度和高维数据集合严重重叠时的查询效率问题。对于维数高于16的数据来讲,X树检索性能一般可以比R树提高两个数量级。结点分裂历史(SplitHistory)信息是按照二叉树的结构存储在目录结点中的,被用来寻找结点的最小分裂,但是在花费了大量的CPU计算时间和占用大量额外空间之后,得到的分裂结果却不能保证一定是最优解,相反地,在很多情况下,得到的仅是次优解。

ESR树(Extended SRtree)[3]利用X树的设计思想来扩展SR树,通过引入超结点,充分利用了SR树和X树两者的优点,采用基于中心点选择子树和分裂结点的策略,与X树相比需要更少的计算次数。但是由于在插入和删除操作后需要同时更新最小边界矩形和超球体,因此在总体上的查询性能并不理想。文献[4]提出的Z树索引结构,通过重新插入策略来压缩超结点的数量和体积,以实现查询效率的提高。该方法虽然减少了超结点的数量,但是采用重新插入策略增加了索引结构维护更新的复杂度。

张军旗[5]提出一种支持最近邻查询的混合索引,首先通过聚类分解分割数据并建立树状索引;然后使用查询采样算法,对数据实际分布进行估计;最后根据数据分布的特性,把稀疏数据从树状索引中剪裁出来进行基于顺序扫描策略的索引,而分布比较密集的数据仍然保留在树状索引中。该混合索引是一种有效的度量空间索引结构。

针对已有的索引结构在进行k近邻查询时效率上的不足,提出了一种新的结点分裂算法,利用当前的索引结构结点构成的实际状况而不是利用分裂历史信息进行X树结点分裂。该算法不再要求每次得到最小分裂,而是仅仅将重叠控制在合理的范围之内,同样能够得到次优解,而时间和空间的花费要大大减少,尤其适合k近邻查询的应用。

2 基本概念

定义1(*d*维向量空间) 自然界中存在大量复杂的事物或现象,为了从不同角度描述这些复杂对象,需要用各个方面的特征及特征间的关系来共同描述。实数域上的一个*d*维向量就是实数域中*d*个数组成的有序数组$(a_1,a_2,\cdots,a_d)$。其中a_i称为该*d*维向量的分量。两个*d*维向量相等当且仅当这两个*d*维向量的所有分量都相等。以实数域中的数作为分量的*d*维向

基金项目:国家自然科学基金(the National Natural Science Foundation of China under Grant No.60773100);河北省教育厅科研计划项目(No.2006143)。
作者简介:章德斌(1976—),男,博士研究生,CCF会员,主要研究方向为数据挖掘;曹丽君(1971—),女,副教授,主要研究方向为数据挖掘;梁永欣(1976—),男,硕士研究生;张忠平(1972—),男,博士,博士后,副教授,硕士生导师,CCF会员。E-mail:zdbysu@163.com
收稿日期:2009-05-25;修回日期:2009-09-24;CNKI出版:2011-1-27;http://www.cnki.net/kcms/detail/11.2127.TP.20110127.1042.201105.123_468.html

量的全体,称为实数域上的d维向量空间,记为R^d。

定义2(距离函数DIST(Distance)) 是定义于d维向量空间R^d中的距离函数[6],数学表达式为:

$$DIST^2(q,p)=\sum_{i=0}^{d-1}(p_i-q_i)^2 \tag{1}$$

p和q是向量空间中的数据对象。

定义3(k近邻查询kNNQ(k Nearest Neighbor Query)) 给定查询数据q及正整数$k(k\geq 1)$,从数据集合D中找出距离q最近的k个数据。当k=1时,又称为最近邻查询。用公式表示为:

$$kNNQ(q,k)=\{o_0\cdots o_{k-1}\in D|\forall e\in D\backslash\{o_0\cdots o_{k-1}\},\ DIST(o_i,q)\leq DIST(e,q),0\leq i\leq k-1\} \tag{2}$$

定义4(最小边界矩形MBR(Minimum Bounding Rectangles)[2]) 包含d维空间对象的最小空间矩形称为该空间对象的最小边界矩形。它包围该结点的所有子树中的空间对象的最小边界矩形。数学表达式为:

$$MBR=[lb_0,ub_0]\times[lb_1,ub_1]\times\cdots\times[lb_{d-1},ub_{d-1}] \tag{3}$$

公式(3)中lb_i和ub_i分别表示空间对象在第i维上的起始位置和终止位置。

定义5(覆盖率) d维空间对象实际所占的空间与该对象的最小边界矩形的比值称为该空间对象的覆盖率。

定义6(结点的带权重叠率[1]) 落入空间重叠部分的数据项占全部数据项的百分比称为结点的带权重叠率,记作WeightedOverlap,数学表达式为:

$$WeightedOverlap=\frac{\left|\left\{p|p\in\bigcup_{I,J\in\{1\cdots m\},I\neq J}(R_I\cap R_J)\right\}\right|}{\left|\left\{p|p\in\bigcup_{I\in\{1\cdots m\}}R_I\right\}\right|} \tag{4}$$

定义7(分裂(Split)) 将结点$S=(mbr_1,mbr_2,\cdots,mbr_n)$分裂为两个子结点$t_1=(mbr_{t1},mbr_{t2},\cdots,mbr_{ts1})$和$t_2=(mbr_{j1},mbr_{j2},\cdots,mbr_{js2})$的过程称为分裂。数学表达式为:

$$Split(S)=\{(t_1,t_2)|S=t_1\cup t_2\text{且}t_1\cap t_2=\Phi\} \tag{5}$$

当且仅当$\|MBR(t_1)\cap MBR(t_2)\|=0$时为无重叠结点分裂;当且仅当$\|MBR(t_1)\cap MBR(t_2)\|\leq minimal$时为合理分裂,$minimal$为阀值,若重叠高于阀值则分裂是不合理的;当且仅当$-\varepsilon\leq|t_1|-|t_2|\leq\varepsilon$时为平衡的分裂。

为实现k近邻查询对众多查询路径的剪枝,引入了两个距离概念:MINDIST和MINMAXDIST[7],公式(6)、公式(8)和公式(9)中的lb_i和ub_i分别表示向量空间对象在第i维上的起始位置和终止位置。

定义8(最小距离(MINDIST))

$$\text{MIN}DIST^2(q,MBR)=\sum_{i=0}^{d-1}\left(\begin{cases}lb_i-q_i,q_i<lb_i\\q_i-ub_i,ub_i<q_i\\0,\text{otherwise}\end{cases}\right)^2 \tag{6}$$

定义9(最小最大距离(MINMAXDIST))

$$\text{MINMAX}DIST^2(q,MBR)=\min_{0\leq k\leq d}\left(|q_k-rm_k|^2+\sum_{\substack{i\neq k\\0\leq i<d}}|q_i-rM_i|^2\right) \tag{7}$$

$$rm_k=\begin{cases}lb_k,q_k\leq\dfrac{lb_k+ub_k}{2}\\ub_k,\text{otherwise}\end{cases} \tag{8}$$

$$rM_i=\begin{cases}lb_i,q_i\geq\dfrac{lb_i+ub_i}{2}\\ub_i,\text{otherwise}\end{cases} \tag{9}$$

3 X*树的基本操作

3.1 X*树目录结点结构

X*树的基本结构与X树相同,但结点结构不同。X*树每个结点中的数据项由有序对(MBR,Ptr)组成,其中Ptr是指向下一结点的指针,MBR为数据项所代表的最小边界矩形,在X*树的结点结构中不再使用结点分裂历史信息,因此X*树没有SplitHistory一项。X*树的目录结点结构如下所示。

$MBR_0\ Ptr_0\ \cdots\ MBR_{n-1}\ Ptr_{n-1}$

3.2 插入算法的基本步骤

插入是索引结构最重要的操作之一,由于各个结点之间的重叠和目录区域对数据空间覆盖的不完整性,使得选择插入结点成为难点。首先,找到一个适合容纳新数据的叶子结点,将新数据插入到叶子结点中;如果此叶子结点中容纳的数据超过了限制,则此结点按照一定的规则分裂为两个新的叶子结点。然后,对叶子结点的上层目录结点作相应的修改,使上层目录结点适应下层分裂和MBR的大小的变化。在子结点进行了结点分裂的情况下,需要在当前结点新增一个MBR,如果上层结点也发生溢出,则目录结点亦按一定的规则分裂;如果分裂失败,生成超结点。重复算法的前两步一直向上传播,如果根结点也发生分裂的话,则树的高度增加,并建立一个新的根结点,成为分裂后两个结点的上层结点。

3.3 删除算法的基本步骤

对于动态数据索引结构,删除算法是一个关键的操作。首先,查询到要删除的数据所在的叶子结点。然后删除此数据。如果因此造成叶子结点包含数据的数目小于限定值的话,则将此结点中的所有数据移出,删除此结点,然后把移出的数据再插入到索引中。

3.4 X*树的结点分裂算法

X*树分裂算法基本思想:首先对d维空间的每一维进行查找,看是否存在无重叠分裂,因为在无重叠的结点分裂的条件下对X*树的k近邻查询操作代价是最小的。若不存在,则使用拓扑方法,看是否能够找到结点覆盖率最大,结点带权重叠率最小,并且split能够是平衡的最优分裂;若不存在,使用简单聚类法,看是否存在合理分裂,若存在,则将旧的结点删掉,将新的两个结点加入到X*树中,然后返回TRUE;若不存在,则返回FALSE。

简单聚类法的基本步骤是:首先,用待分裂结点中距离最远的两个MBR作为结点的两个分裂集合$t1$和$t2$的种子;然后,对结点分裂集合$t1$,选取使得最小边界矩形增大最小的MBR加入结点分裂集合$t1$,即将与$t1$的种子距离更近的entry加入$t1$集合。同样地,结点分裂集合$t2$也如上操作。两个集合轮流进行以上的操作。直至所有entry都被加入这两个结点分裂集合为止。简单聚类法能尽可能产生无重叠分裂,使同一个结点内的各个entry更紧凑,有效地减少重叠区域,提高空间利用率,改善k近邻查询的效率。通过拓扑算法和简单聚类法的结合使用,在有结点合理分裂的情况下,找到合理结点分裂的概率得到了很大提高。在没有结点合理分裂的时候,生成超结点,使得在处理高维数据时的空间重叠极大地减少了,从而提高了索引结构的查询性能。

算法1 采用简单聚类法的X*树结点分裂算法

输入:待分裂结点地址in,新结点地址out1,out2;

输出:分裂成功返回TRUE,否则返回FALSE,创建超结点。

```
Begin
(1)SET_OF_MBR t1,t2;
(2)MBR r1,r2,seed1,seed2;
   //查找无重叠分裂
(3)for(i=0,i<d,i++){
(4)   if(overlapfree_split(in,t1,t2,i)){
(5)      *out1=t1;*out2=t2;
(6)      return TURE;
(7)   }
(8)}
   //试用拓扑方法做拓扑结点分裂
(9)topological_split(in,t1,t2);
(10)r1=t1->calc_mbr();r2=t2->calc_mbr();
    //拓扑方法失败,采用简单聚类结点分裂法
(11)if(overlap(r1,r2)>MAX_OVERLAP){
(12)  selectfarseeds(in,seed1,seed2);
(13)  for(i=0;i<M+1;i++){
(14)     if(DIST(entry[i].mbr,seed1)<=DIST(entry[i].mbr,seed2))
(15)        entry[i] is added to t1;
(16)     else entry[i] is added to t2;
(17)  }
(18)  if(t1->num_of_mbrs()<MIN_FANOUT||
       t2->num_of_mbrs()<MIN_FANOUT)
(19)     return FALSE;
(20)}
(21)*out1 = t1;*out2 = t2;
(22)return TURE;
End
```

3.5 k近邻查询算法

选用内存消耗较少的RKV算法[7]作为k近邻查询算法。基本思想是:从根结点开始,以深度优先的方式递归进行访问,对于每个被访问到的目录结点都先对其所有的子结点基于某种标准进行排序,以决定下一步要访问的子结点,并在排序后将那些到查询向量的最小距离已超过剪枝阈值的分支抛弃,通过抛弃该结点下的全部路径,从而可以大大地减少结点访问代价。深度优先地向下处理最小边界矩形中包含对象q的孩子结点,直到找到叶子结点,在叶子结点查询k近邻。通过搜索同一个双亲的叶子结点的近邻来实现k近邻查询,如果少于k个点,那么算法向上搜索叶子结点的双亲结点,从该结点可达的叶子结点查找其余的q的k近邻,当k个最近邻被发现时,算法结束。算法完成时近邻优先级队列result用于存放k近邻。

算法2 RKV算法

```
输入:数据对象q,近邻个数k,根结点地址na;
输出:存放k个近邻的优先级队列result。
Begin
(1)int i;
(2)NodeAdr *p=na;
(3)if(IsDataNode(p))  //叶子结点
(4)  for(i= 0;i<p.numOfEntries;i++){
(5)     if(dist(q,p[i])<=pruning_dist)
(6)           result.insert(p[i]);
(7)     if(result[k].dist<pruning_dist)
(8)           pruning_dist=result[k].dist;
(9)  }
(10)if(IsDirectoryNode(p)){ //目录结点
(11)  sort(p,CRITERION);
(12)  for(i=0;i<p.numOfEntries;i++){
(13)     if(MINDIST(q,p[i].MBR)<=pruning_dist)
(14)        RKV_algorithm(q,p[i].childnode);
(15)     if(MINMAXDIST(q,p[i].MBR)<=pruning_dist)
(16)        pruning_dist=MINMAXDIST(q,p[i].MBR);
(17)  }
(18)}
End
```

4 实验结果分析

主要通过比较实验的方法测试在X树与X*树索引结构上进行k近邻查询的结果及其性能。用C++编写了程序。运行机器配置为Intel Pentium D 3.1 GHz CPU,1 GB内存,160 GB硬盘,操作系统为Windows XP,编译器是Visual C++6.0。在实验中,采用仿真数据对X*树索引结构的查询效率和空间需求进行测试。

4.1 k近邻查询效率实验

在实验中,选择k=10,数据集大小一定,由随机数生成器产生的在高维空间中均匀分布的数据包含了10 000个多维空间中的点数据,对不同维数下的均匀分布数据分别进行了1 000次10近邻查询,取平均10近邻查询时间作为查询代价,图1中显示了实验X*树和X树索引结构的k近邻查询性能随着数据维数的变化而变化的折线图。

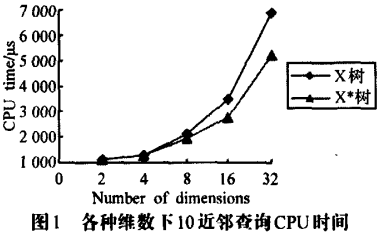


图1 各种维数下10近邻查询CPU时间

由图1中可以看出两种索引结构在数据空间维数较低(2维、4维、8维)的时候表现出了几乎完全相同的性能。这主要是因为在维数较低的时候,目录结点分裂基本是无重叠的,因而两种索引结构的性能很相近。随着数据维数的增加,X*树体现出了明显的性能优越性。在维数为16维的时候,X*树的k近邻查询响应时间是X树的80.78%,在维数为32维的时候,X*树的k近邻查询响应时间是X树的75.34%。这是因为在维数较高时,空间数据的重叠现象突出起来,导致分裂算法失败,生成超结点。而X*树的k近邻查询性能总体上优于X树,主要是由于在结点分裂时,采用了简单聚类法,使得同一个结点内的各个数据项更加紧凑,查询结果基本上只需在一个叶子结点内就可以选出,因此在k近邻查询时的剪枝效果就更好。

4.2 内存空间需求实验

在实验中,选择数据集大小一定,由随机数生成器产生的在高维空间中均匀分布的数据,包含了10 000个多维空间中的点数据,对不同维数下的均匀分布数据分别测试了X*树和X树索引结构实际占用的内存空间。由图2中可以看出,随着维数的增大,X*树和X树索引结构实际占用的内存空间也相

(下转174页)

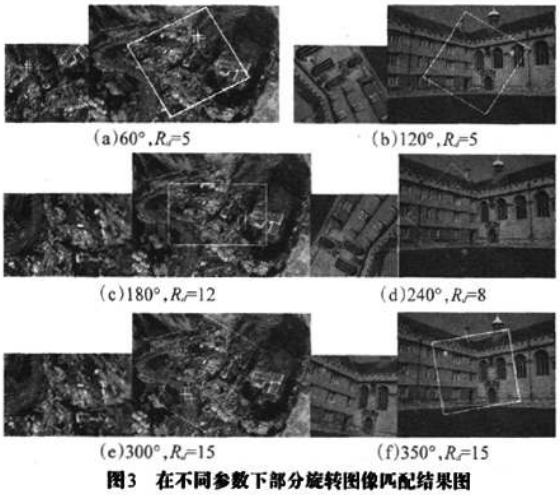
投影算法在不同参数下的准确率均为100%，实验部分结果如图3所示。此外，对最终的匹配点采用8个辅助点计算旋转角度，并用中间两个角度的平均值作为最终旋转角度，统计出实际转角、实验转角及二者误差角度，实验统计结果见表1，其中Max、Min和Ave分别表示在不同参数下的最大误差角度、最小误差角度和平均误差角度。结果表明，扩展的圆投影算法在不同参数下均能获得较好的匹配结果。

表1 不同参数下的旋转角度误差统计结果表

	R_r=5			R_r=8			R_r=12			R_r=15		
	Max	Min	Ave	Max	Min	Ave	Max	Min	Ave	Max	Min	Ave
A	1.88	0	0.63	1.82	0	0.64	1.84	0	0.70	1.84	0	0.68
B	1.85	0	0.72	1.74	0	0.59	1.63	0	0.72	1.59	0	0.60

(a)汶川映秀镇震后图像(800×600) (b)牛津大学校舍(1 024×768)

图2 基准图



(a)60°，R_r=5 (b)120°，R_r=5

(c)180°，R_r=12 (d)240°，R_r=8

(e)300°，R_r=15 (f)350°，R_r=15

图3 在不同参数下部分旋转图像匹配结果图

(左侧图像为模板图，右侧为配准结果)

5 总结

提出了一种鲁棒的圆投影改进算法用于任意角度旋转的景象匹配。该算法对原始的圆投影算法添加辅助点约束条件，提高了任意旋转角度的景象匹配的准确率，同时，辅助点有效帮助计算模版图像的旋转角度。将来研究目标是解决特定尺度的景象精确匹配问题。

参考文献：

[1] Tanaka K, Sano M, Ohara S, et al.A parametric template method and its application to robust matching[C]//IEEE Conference on Computer Vision and Pattern Recognition, 2000, 1: 620-627.

[2] Pang S N, Kim H C, Kim D, et al.Prediction of the suitability for image-matching based on self-similarity of vision contents[J].Image and Vision Computing, 2004, 22(5): 355-365.

[3] 陈宁江，李介谷.用归一化灰度组合法进行图像匹配[J].红外与激光工程，2000，29(5)：6-7.

[4] 王敬东，徐亦斌，沈春林.一种新的任意角度旋转的景象匹配方法[J].南京航空航天大学学报，2005，37(1)：6-10.

[5] Lin Y H, Chen C H, Wei C C.New method for subpixel image matching with rotation invariance by combining the parametric template method and the ring projection transform process[J].Optical Engineering, 2006, 45(6).

[6] Tsai D M, Chiang C H.Rotation-invariant pattern matching using wavelet decomposition[J].Pattern Recognition Letters, 2002, 23: 191-201.

[7] 罗三定，张凯.具有旋转不变性的异形模板匹配方法[J].计算机工程，2008，34(20)：215-217.

[8] Lina Yi-Hsien, Chen Chin-Hsing.Template matching using the parametric template vector with translation, rotation and scale invariance[J].Pattern Recognition, 2008, 41: 2413-2421.

(上接125页)

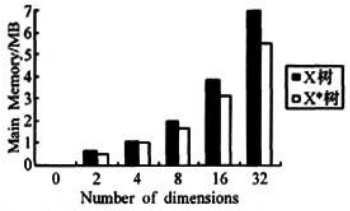


图2 两种索引结构在不同维数的内存空间需求

应增大，这是因为结点内各个数据项的MBR数组随着维数的增大而占用更多的空间，而且在任何维数，X树都比X*树占用了更多的内存空间，这是因为X树需要在全部目录结点的所有的数据项里额外存放结点分裂历史记录信息。

5 结束语

针对支持k近邻查询的索引结构进行了研究，在深入分析X树索引结构和k近邻查询算法的基础上提出了一种采用简单聚类法的结点分裂算法，利用当前的索引结构结点构成的实际状况而不是利用分裂历史信息进行X树结点分裂，实验结果表明X*树在时间和空间性能上都更好。

参考文献：

[1] Berchtold S, Keim D A, Kriegel H P.The X-Tree: An index structure for high dimensional data[C]//Proc of the 22nd VLDB Conference, 1996: 28-39.

[2] Guttman A.R-Trees: A dynamic index structure for spatial searching[C]//Proc ACM SIGMOD, Boston, MA, 1984: 47-57.

[3] 徐焕，林坤辉.ESR-Tree：一种多维对象的动态索引方法[J].计算机应用，2005，25(12)：2872-2874.

[4] 张强，赵政.Z树：一个高维度的数据索引结构[J].计算机工程，2006，33(15)：49-51.

[5] 张军旗.支持最近邻查找的高维空间索引[D].上海：复旦大学，2007：51-69.

[6] 夏火松.数据仓库与数据挖掘技术[M].北京：科学出版社，2004：171-176.

[7] Nick R, Stephen K, Frederic V.Nearest neighbor queries[C]//Proc of the ACM SIGMOD, SanJose, California, 1995: 71-79.