Rav-tree:一种有效支持反向近似近邻查询的索引结构

李博涵[1] 郝忠孝[1,2]
(哈尔滨理工大学计算机科学与技术学院 哈尔滨 150080)[1]
(哈尔滨工业大学计算机科学与技术学院 哈尔滨 150001)[2]

摘 要 空间数据库的索引结构是实现有效数据查询的前提和基础。空间数据反向近似近邻查询是空间查询的一个新方向,它避免了精确查询中过多的距离计算,从而能够在效率与准确性上取得平衡。提出的Rav-tree不同于基于启发式规则的索引结构,首先利用局部近似,然后根据Voronoi cell区域和估计圆的方法实现近似近邻查询,并利用过滤结果和分域查询得到初步的候选集,最终通过反向近似近邻查询(RANNQuery)算法得到RANN集,并完整地给出基于Rav-tree的ANN查询算法和RANN查询算法。实验结果表明,Rav-tree对RANN等查询具有较好的查询效率和查全率。

关键词 索引结构,反向近似近邻,分域查询,区域估计
中图法分类号 TP311.13 **文献标识码** A

Rav-tree:An Efficient Index Structure for Reverse Approximate Nearest Neighbor Query

LI Bo-han[1] HAO Zhong-xiao[1,2]
(College of Computer Science and Technology, Harbin University of Science and Technology, Harbin 150080, China)[1]
(College of Computer Science and Technology, Harbin Institute of Technology, Harbin 150001, China)[2]

Abstract Index structure is the precondition and foundation in the efficient data query. The reverse approximate nearest neighbor query is a new issue in the area of spatial query. This approach can avoid much metric distance computation in exact query, and acquire a better tradeoff between the efficiency and precision. The Rav-tree is different from the index structures based on the heuristic rules. It applies partial Voronoi cell approximation with estimated circles to filter the results of approximate nearest neighbor query. The final result set of RANN is reached through the algorithms of ANN query and Division query with primary candidates. The experimental results indicate that the Rav-tree is an effective index structure and has better efficiency and recall for the query such as RANN query.

Keywords Index structure, Reverse approximate nearest neighbor, Division query, Region estimation

近邻查询是空间数据库查询领域的一个重要问题,近年来在GIS、Web查询引擎、设施定位、多媒体数据库等方面应用广泛,也是查询的核心技术之一。目前空间数据查询的主要分支有近邻[1]、反向(k)近邻[2,3]、k最近对查询[4]等。这些查询主要是针对精确的查询结果集,而实际的应用中有时可以利用查询的近似结果,以查询效率为第一目标,甚至可以忽略一些结果。基于Z曲线的近似最近对[5]查询虽可以解决高维空间满足一定"相似"程度的最近对查询,但是降维过程中采用不同曲线使查询性能差异较大,而且由于Z曲线的聚类性较差,对结点重叠的判定就容易产生较大误差。

已有的索引结构多为采用基于启发式规则的算法,这些算法为了精确地匹配一个查询结果,即使索引结点中有符合的结果,仍然会遍历查询整个路径,直至叶结点。更为严重的是,一旦启发式规则失效,并且是出现在索引结构的较高层上,那么不必要的路径选择就会直接影响查询效率。ANN-tree是一种不借助任何启发式规则的索引结构,可以实现比较精确的近似近邻查询。但该索引结构利用圆型(球型)区域实现对Voronoi cell区域的估计方法存在局限性[6]。该方法仅考虑了单独的Voronoi cell的估计,不能有效地处理周围的Voronoi cell的影响,并且在高维的情况下近似扩展系数值不易确定。目前,近邻查询中已有的关于反向近似查询解决方式难以解决存储近邻记录代价较高且查询效率较低的问题,或者根本无法实现反向近似近邻查询。反向近似近邻查询是对近邻查询的重要补充,尤其适用于需要快速反应的实际应用。

基于以上原因,本文在已有的ANN-tree和Voronoi cell的基础上,对原有索引结构进行改造,提出了一种能有效支持反向近似近邻查询的索引结构——Rav-tree。它引入了Voronoi cell区域估计和Voronoi估计圆等概念,不仅支持低维反向近似近邻查询,也支持多维。理论分析和仿真验证均证明了Rav-tree实现近邻查询和反向近似近邻查询的有效性[1)]。

到稿日期:2009-02-20 返修日期:2009-05-02 本文受国家自然科学基金项目(60673136),黑龙江省自然科学基金项目(F200601)资助。
李博涵(1979—),男,博士研究生,CCF会员,主要研究方向为空间数据库的数据索引与查询优化,E-mail:bohanli@hrbust.edu.com;**郝忠孝**(1940—),男,教授,博士生导师,主要研究方向为数据库理论及应用。

1)针对本文的查询有效性概率包含查询效率和最终结果的查全率。

1 基础知识

定义 1(RNN 查询)[1] 假设 d 维数据集 P 和查询点 p，反向最近邻(RNN)查询就是找出 P 的子集 RNN(q)，即

$$RNN(p)=\{p_i\in P \mid \forall p_j\in P: dist(p,p_i)\leqslant dist(p_i,p_j)\},i\neq j$$

定义 2(RkNN 查询)[3] 假设 d 维数据集 P 和查询点 p，反向 k 最近邻(RkNN)查询检索所有将 p 作为 k 个最近邻之一的数据点，即

$$RkNN(p)=\{p_i\in P \mid \forall dist(p,p_i)\leqslant dist(p_i,p_k)\}$$

p_k 为 p 的第 k 个最远的 NN。

特别地，当 $k=1$ 时，即为 RNN 查询。

图 1 是一个 kNN 与 RkNN 的简单实例。如图 1(a)所示，在一维情况下，图中有 4 个数据点，显然 $NN(p)=p_2$，然而 p_2 却不是 p 的 RNN，即 $p_2\notin RNN(p)$。这是由于 $NN(p_2)=p_1$，因此 $p\notin NN(p_2)$。所以，根据定义 1 可知 p_2 不是 p 的 RNN。进一步判断可知，p 是 p_3 的 NN，因此只有 p_3 将 q 作为其 RNN，故 $RNN(p)=p_3$。图 1(b)是二维情况下对 2NN 和 R2NN 的查找。根据定义 2 可知 $2NN(p)=\{p_2,p_3\}$，而 $R2NN(p)=p_3$。假设 qp_3 的长度大于 p_1p_3，那么 R2NN(p)将成为空集。然而，只要数据集中除 p 外尚有其它数据点存在，2NN(p)就永远不会成为空集。这就说明了对于任意 $p_i\in kNN(p)$，并不意味着 $p_i\in RkNN(p)$，反之亦然。多维空间情况与之相似。

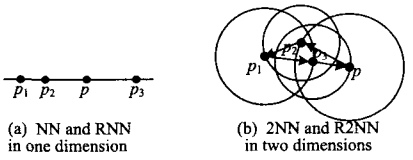

图 1 kNN 与 RkNN 的实例

定义 3(Voronoi cell)[7] 给定一组生成点 $P=\{p_1,\cdots,p_n\}\subset R^2$，其中 $2<n<\infty$，且当 $i\neq j$ 时，$p_i\neq p_j$。$i,j\in\{1,\cdots,n\}$。Voronoi cell 由以下公式给出：$VC(p_i)=\{p \mid dist(p,p_i)\leqslant dist(p,p_j)\}$。其中 $dist(p,p_i)$为 p 与 p_i 之间的最小距离。由 p_i 所决定的区域称为 Voronoi cell 区域，它们的生成点被称为邻接生成点。Voronoi cell 简记为 VC。

在 VC 生成算法中，可利用 $n-1$ 个二等分的半平面相交来计算 $VC(p_i)$，其时间复杂性为 $O(n^2\log n)$[7]。如无特殊说明，所有的数据查询点和生成点均在同一数据集中。

2 基于 Rav-tree 的索引与查询

2.1 VC 区域估计

由于 VC 生成具有唯一性，故充分利用 VC 的每个单元就是解决查询的关键。空间中数据点的最小外包区域可以最大限度提高 RNN 的查询效率。减少对 RNN(RkNN)候选集的访问次数。因此，要做到最小化访问次数，每个索引分支的外包区域就必须尽可能地满足以下 3 个条件：

1)在索引结构中，任意的两个叶结点中的 MBR 互不相交[2)]；

2)在索引树的每一层，同层结点的所有 MBR 必须覆盖整个数据空间；

3)数据点 p_i 包含在一子树之中当且仅当 $V(p_i)$与子树的根结点的 MBR 相交。

当 $VC(p_i)$与多个 MBR 相交时，p_i 可能存在于多个叶结点之中，条件 1)和条件 2)由 R*-树的性质可知。条件 3)的证明参见文献[6]。

由表 1 可知，满足条件 3 是相对困难的，尤其是随着维度的增加，VC 构建代价是很大的。为了减少构建 VC 的开销，并且要保证近似的准确性，因此需要对 VC 区域重新进行近似估计，这样既可以支持多种查询，又能保证较高的准确性。

表 1 不同索引结构所满足的条件与所支持的查询

	条件 1	条件 2	条件 3	反向	近似
R*	不支持	支持	不支持	不支持	不支持
RNN	支持	支持	不支持	支持	不支持
ANN	支持	支持	扩展支持	不支持	支持
Rav	支持	支持	扩展支持	支持	支持

如图 2 所示，p_1 是查询点，$VC(p_1)$是 p_1 所在的 VC，假定 p_3 是 p_1 的 NN。根据 VC 定义可知，以 p_1m 为半径的圆完全包含在 $VC(p_1)$中，这是对 $VC(p_1)$的一种近似估计。

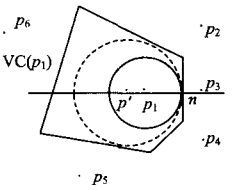

图 2 VC 区域估计

进一步假设，如果 p_3 也是 p_1 的 RNN，那么将 p_1 向 p_1p_3 的反方向移动到位置 p'，并且以 $p'm$ 为半径做圆(图中虚线圆)，显然虚线圆对于 $VC(p_1)$具有更好的近似性。

设 $f=|p'm|/|p_1m|$，为近似估计系数。当有新的数据点在 VC 区域内进行更新(包含数据点的插入与删除)时，则需要对 VC 进行重估。这里重估所需考虑的点是以 p_1 为 NN 的点，即 p_1 的 RNN。设新的估计圆的半径为 $dnn(p)$，假设 $p_k=RNN(p_1)$，那么有 $|p_1p_k|\leqslant 2dnn(p)/f$，即有 $|p_1p_k|\leqslant 2dnn(p)|p_1m|/|p'm|$，所以 $dnn(p)\geqslant|p'm||p_1p_k|/2|p_1m|$。

定义 4(VC 估计圆) 以各区域内的最近邻 p_i 为圆心、以 $dnn(p)$为半径所做的圆称为 VC 估计圆(VC circle)。

定义 5(ANN 查询) d 维数据集 P 和查询点 p，ANN 查询就是找出 P 的子集 $ANN(p)$，满足 $ANN(p)=\{r\in P \mid \forall p_i\in P: dnn(p_i,r)\leqslant dnn(p,p_i)$，$p_i$ 为 p 的邻接生成点$\}$，其中 $dnn(p_i,r)$的最大值为 VC 估计圆中最大圆的半径。

定义 6(RANN 查询) d 维数据集 P 和查询点 p，RANN 查询检索所有将 p 作为其 ANN 的数据点，满足 $RANN(p)=\{p_i\in P \mid \forall dnn(p,p_i)\leqslant dnn(p_i,p_j)$，$p_j$ 也是 p 的邻接生成点$\}$。

在距离度量上，由于 VC 区域的重新估计，新的 VC 估计圆半径采用 dnn 度量。由于 RANN 与 ANN 之间不一定存在 RNN 与 NN 之间的关系，故不能找到类似于 RkNN 与 kNN 之间精确的集合关系。因此，对于 ANN 与 RANN 的查询分别给出各自查询算法及中间算法。

2)本文以 R 树系为原型，故此描述仍采用最小外包矩形区域，即 MBR。

定义 7(六分区域) 假设查询点 $p=(x_p,y_p)$,其中 x_p 和 y_p 分别是其横坐标值和纵坐标值,3 条直线相交于点 p,且其中的一条直线平行于横坐标,它们相交的角度均为 60°,则此 3 条直线将 p 点周围的空间分割成 6 个区域 $S_1,S_2,\cdots,S_6$。这 3 条直线称为空间分割线,将 6 个分割区域称为六分区域。

定义 8(分域查询候选集) 在查询点 p_i 的若干个连续六分区域内的查询称为分域查询。进行分域查询所得的 p_i 的 ANN 和 RANN 分别为分域 ANN 和分域 RANN。由所有分域查询所得的查询结果组成的数据集称为分域查询候选集。

分域查询可以充分利用 VC 的过滤功能,使得大量的数据点可以在查询中被过滤掉,并且在索引结构中可以直接通过分域的分支索引策略进行剪枝,省略大量的距离比较和计算等操作。故根据 VC 和 RANN 查询的定义可知,在六分区域得到 RANN 的分域查询候选集查询,其效率明显高于直接查找全局的 RANN 查询的效率。

如图 3 所示,在查询点 p_i 的六分区域所划分的 VC 局部图中,RNN 查询结果由于新的评估方式的改变而对查询结果产生影响。例如由图可知 p_6 位于 p_1 的估计圆外,却在以 p_1p_i 为半径的圆内,因此可知 p_6 是 p_1 的 ANN。如果 p_i 的 VC 估计圆与 p_1 的 VC 估计圆相切或相交,那么 p_i 加入 p_1 的 ANN 集,p_1 对应地也加入了 p_i 的分域查询候选集(RANN)。继续考察 S_4 分域,p_1 和 p_6 位于同一分域内,RNN(p_i)的查询结果应该只是 p_1。但是 p_1 的 VC 估计圆内不含其它 VC 的生成点。因此将 p_6 也加入到分域查询候选集(RANN)。

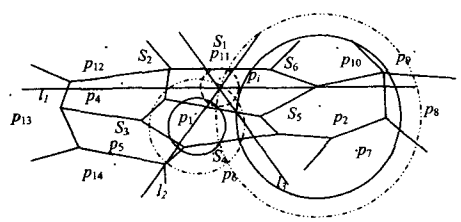

图 3 利用 VC 区域与六分区域的分域查询实例

2.2 基于 Rav-tree 的查询算法

Rav-tree 满足良好树形索引结构的高度平衡性要求,在结构上更接近 R*-tree。对任意一点 p,叶结点存储了以 p 为圆心、以 $dnn(p,p_i)$ 为半径的 VC 估计圆。只要确定那些包含点 p_i 的 VC 估计圆,并返回其圆心 p,即得到满足条件的查询结果。由于 Rav-tree 满足条件 1)和条件 2),保证了任意一条分支路径必然能够得到一个结果,并且在每一层都可以覆盖整个查询空间,利用树型索引结构的分支策略就可以实现 RANN 查询。因此,Rav-tree 将 VC 估计圆和树型索引有机地融合在索引结构之中,可实现如 ANN 和 RANN 等多种近似查询。

假设 p_i 是查询点,如果当前结点是叶结点,则对叶结点中每个记录($appr,dnn,*cp$)计算该点 $appr$ 和查询点 p_i 的距离下界 $d_l(p_i,appr)$,而下界范围参照 dnn 的计算结果。其中,$appr$ 为区域评估的近似距离,dnn 为点与 NN 的距离,$*cp$ 为指针。若当前结点为索引结点,则计算 p_i 和每个记录($circle$,max_dnn,$*cp$)中的 $dnn(p_i,circle)$。如果 $dnn(p_i,circle)>$max_dnn,那么该记录中 $*cp$ 所指向的子树就被剪除,其中的任何点和 p_i 的距离都不会小于该点的 dnn,即不可能成为 p_i 的 ANN;否则,递归调用 ANNQuery 查询。基于 Rav-tree 的 ANN 查询算法如下。

```
ANNQuery(P, p_i)
输入:数据集 P,查询点 p_i;
输出:p_i 的 ANN 候选集;
begin
    初始化 VC circle 为根结点所指的所有 circle( );
    初始化 d_l 为结点中 circle( )的半径下界;
    while (circle( ) is not empty)
      if (currentnode is a leaf node) then
        for (every entry(appr,dnn,*cp) in currentnode)
          计算距离下界 dist=d_l(p_i,appr);
            if (dist<dist(p_i,p_j)) then //dist(p_i,p_j)为 p_i 到 p_j
的距离
              dist←dist(p_i,p_j);
              else if (dnn<dist(p_i,p_j)) then
                  得到 *cp 所指向的点 p_j;
                  p_j←p,dist(p_i,p_j)←dnn;//p_j 加入 ANN
    else if (currentnode is a internal node) then
      for (all entry in currentnode)
        for (every currentnode in the sorted order)
          if (d_l<dnn(p,p_j)) then //d_l 为 p_j 点的 circle 的半径下
界
            circle( )←circle( )-circle(p_j);
          else ANN←p_j;
    ANNQuery(*cp,p_i) //*cp 为索引结点指针,递归调用
    end
```

证明:(正确性)基于 Rav-tree 的 ANNQuery( )类似于基于 R*-tree 的 NN 查询,故正确性可由 R*-tree 的 NN 查询算法的正确性保证。

(可终止性)算法 ANNQuery( )是递归的,根结点的子树中的每个内部结点都可以调用算法 ANNQuery( ),Rav-tree 中内部结点的数目是有限的,所以该算法被递归的次数也是有限的;算法中的 while 循环和两个单重 for 循环均可自动终止,故该算法可终止。

(时间复杂性分析)Rav-tree 中每个结点最少有 m 个点,包含 n 个索引记录的 Rav-tree 树高度至多是 $\log_m n-1$,结点的最大数目是 $n/m+n/m^2$。计算时间复杂性主要在分支限界上,由于每次循环排除一个分支,所访问的结点个数为($n/m+n/m^2$)/2,故算法的时间复杂性为 $O(\log_m(n/m+n/m^2))$。

在 ANN 查询中,R*-tree 索引结构的索引结点在查询区域中存在大量重叠[8]。当维数较高时,重叠相交的数量会更多。一般情况下,索引结构必须存储所有在搜索区域重叠的结点。Rav-tree 的存储优势在于即使多个结点在查询区域中相交,也无需存储所有的 VC 估计圆。

Rav-tree 支持 ANN 查询达到最小页面访问数量需要两个过程:

(1)从根结点开始,自顶向下为当前结点计算每一个分支的分域查询;

(2)叶结点中数据点均为查询点 p_i 的 ANN。

如果在 VC($currentnode,p_i,d_l$)中找到一个数据点,那么不需要寻找其它点,就可以正确地得到 RANN。对多个相交查询区域情况,分域查询策略使得在分域中得到候选集的几

率最大，故能够以最小数量的访问次数得到查询结果。

由以上论述得出基于 Rav-tree 的分域查询算法如下。

输入：数据集 P，查询点 p_i，分域查询$[Si]$，ANN 数目 k；

输出：具有($[S_i]$，p_1，p_2，…，p_k)形式的一列数据；

DIV($[S_i]$，P，p_i)

begin

$K=\emptyset$，$C=\emptyset$，$R=\emptyset$，$[S_i]=\emptyset$；// K 为包含当前 p_i 的 k 个 ANN，C 包含在分域$[S_i]$里有可能是结果的候选点，R 包含在分段$[S_i]$里，但不可能是结果的数据点，$[S_i]$初始化为空。

call K←ANNQuery(P，p_i)；//将点 p_i 的 k 个 ANN 存入 K 集

call Vor$_s$(K)←gen Vor(P，s)；//生成局部 s 阶 VC[7]且 $s<k$

for Vor(p_s) edge∈Vor(K) do

if(Vor(p_s) edge∩$[S_i]\neq\emptyset$) then

$[S_i]$←[Vor edge∩$[S_i]$]

if $p_ip_s\leqslant$max_$dnn[S_i]$ then

C←p_s

else K←$K-p_s$

call R←VC(*currentnode*，p_i，d_l) //将 p_i 存入 R 集

return ($[S_i]$，$K\cap C$)

end

证明：(正确性)算法首先调用 ANNQuery(P，p_i)来求得单个查询点的 ANN。ANNQuery( )的正确性已经证明。由 VC 定义和性质确定了预生成的局部 s 阶 VC 的生成点集合。由估计圆的假设可知 VC(*currentnode*，p_i，d_l)过滤掉了不属于 K 集和属于 R 集的点。故该算法是正确的。

(可终止性)算法中所调用的算法均可证明是可终止的。算法中含有一个 for 循环，可知 VC 所包含的棱的数量与生成点有关，而生成点与 K 集都是有限的，因此循环可自动终止。故该算法是可终止的。

(时间复杂性分析)对于 n 个点的数据空间生成全局 VC 的时间复杂性为 $O(n^2\log n)$。当$[S_i]$穿过 s 阶 VC 单元的数目较少时得到 ANN 的时间复杂性可忽略。若落在以 p_i 为圆心，d_l 为半径的圆内的点的个数为 h，k 为要初始 ANN 的个数，那么执行 gen Vor(P，s)过程的时间复杂性为 $O(hk^2+h\log h)$。故在查询段穿过的空间对于整个空间较小的情况下，该算法的时间复杂性为 $O(n^2\log n+hk^2+h\log h)$。

算法利用分支限界得到分域查询上所有点的 k 个 ANN 所在范围的候选集。通过 DIV($[S_i]$，P，p_i)算法可以在基于 Rav-tree 上实现有效的 RANN 查询，同时也就构造出从局部到全局的 VC。基于 Rav-tree 的 RANN 查询算法如下。

RANNQuery(*currentnode*，p_i) // *currentnode* 为 Rav-tree 当前结点数据，p_i 为查询点

输入：当前结点 *currentnode*，查询点 p_i；

输出：p_i 的 RANN 集；

begin

if (*currentnode* is a leaf node) then

for (every entry(*appr*，*dnn*，* *cp*) in *currentnode*)

$dist_l$←d_l(p_i，*appr*)；//计算当前距离下界

if ($dist_l<dnn$) then

取得* *cp* 所指的 p_j；

call DIV($[S_i]$，P，p_i)；

$dist_2$←d_l(p_i，p_j) //$i\neq j$，$dist_2$ 为 VC 估计圆半径距离

else if ($dist_2<dnn$) then

RANN←p_j；

else if (*currntnode* is a internal node) then

for (every entry(*circle*，max_*dnn*，* *cp*) in currentnode)

$dist_2$←d_l(p_j，*circle*)；//计算当前距离下界

if ($dist_2\leqslant$max_*dnn*) then

R←p_j；//将 p_j 加入 R 集

else return；

return RANN；

end.

证明：RANN 查询的算法分析与前述算法类似，故省略。

3 实验设置与结果

平台配置：CPU Pentium D 3.0GHz，1G 内存，Windows XP sp2，Borland C++ builder 6.0 编程实现。其平台的框架结构如图 4 所示。测试数据分为两类：①随机函数产生的满足 uniform 分布的合成数据；②California 道路网络结点的真实数据。其中合成数据量最大值为 10^5，真实数据的数量最大值为 21047。如图 5 所示，以经纬坐标表示点。

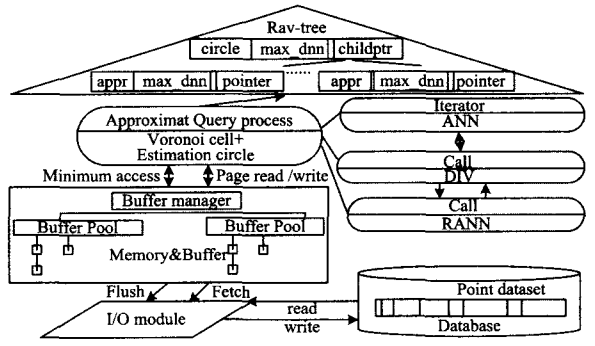

图 4 Rav-tree 索引的 RANN 查询框架结构

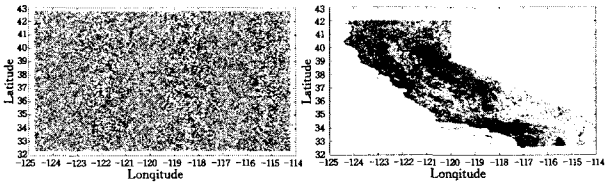

图 5 标准数据与真实数据分布

为了避免近似估计随机性和查询可能存在的偶然性，取 100 次运行的平均值作为实际指标。Rav-tree 的索引结点和叶结点大小均设为 4kB，索引结点最多可存储 100 个单元。采用 LRU 缓存策略，初始时仅保存根结点所在页面，缓存最多可加载 128 个页面。

表 2 中查询时间为由 C++中的 GetTickCount( )计算的 CPU 执行时间，单位为秒。由于 R*-tree，RNN-tree 仅支持 NN 查询，并没有考虑到 ANN 查询，因此对这 3 种索引的查询测试的时间为 NN 查询时间，该结果对近邻查询时间对比没有实质性影响。结果表明，Rav-tree 通过有效的非启发式的过滤方法大量地减少了查询时间。其中，d 为维数，M 为结点最多存储点数量。

表 2 标准数据下不同索引结构近邻查询对比

t(s) \ n	10000	20000	40000	80000	100000
R*-tree	0.329	0.668	1.357	2.628	5.367
RNN-tree	0.293	0.612	1.239	2.318	4.568
ANN-tree	0.075	0.144	0.285	0.532	0.974
Rav-tree	0.072	0.136	0.251	0.488	0.876

$d=2$，$M=100$

为了进一步验证 Rav-tree 的 ANN 查询效率，将两种查

询时间最短的索引结构的磁盘存取次数(PA)进行比值计算，曲线分别为访问叶结点和所有结点的 PA 次数比值。对比结果如图 6 和图 7 所示。随着数据量的增长，Rav-tree 的 PA 比值优势就越明显，效率的提高主要得益于区域评估方法使得 Rav-tree 存储开销减小，以及非启发式规则的查询算法所带来的性能提升。

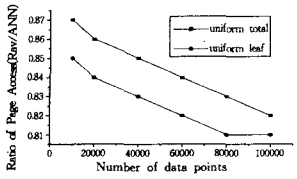

图 6 Rav-tree 与 ANN-tree 在标准数据集的磁盘存取次数比值

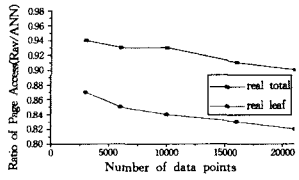

图 7 Rav-tree 与 ANN-tree 在真实数据集的磁盘存取次数比值

RANN 查询是近似查询而非精确查询，是精确性与效率的一种折中。以上实验验证了基于 Rav-tree 的典型查询相比其它索引的查询效率有明显提高。RANN 查询的查全率(Recall)直接影响查询的准确性，特别是在距离度量计算时，查全率是影响计算准确性最重要的因素。因此，实验对 RANN 查询的查全率进行了测试，结果如图 8 所示。k 为类似 RkNN 查询中预先定义的点数量(范围为 5～50)。由结果可知，即使在 k 值较小情况下，83%以上的查全率完全可以满足查询的准确性要求。

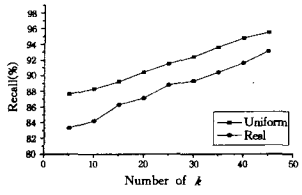

图 8 Rav-tree 在两种数据上的查全率

结束语 不同于启发式规则的索引结构，Rav-tree 利用局部近似得到 ANN 查询结果，进一步结合分域查询实现了 RANN 查询。此方法利用 VC 区域估计和六分区域通过过滤方式快速得到 ANN 查询候选集。Rav-tree 完善了现有索引结构对 RANN 查询的支持。实验结果证明了基于 Rav-tree 的 ANN 查询和 RANN 查询的有效性。

参 考 文 献

[1] Clarkson K L. Nearest Neighbor Queries in Metric Spaces[J]. Discrete & Computational Geometry, 1999, 22(1): 63-93

[2] 刘永山，郝忠孝. 空间对象的反最近邻查询[J]. 计算机科学，2005, 32(11): 115-118

[3] Achtert E, Böhm C, Kröger P. Efficient reverse k-nearest neighbor search in arbitrary metric spaces[C]// Proc. ACM SIGMOD ICMD. Chicago, Illinois, USA, June, 2006: 27-29

[4] Corral A, Manolopoulos Y, Theodoridis Y, et al. Algorithms for processing K-closest-pair queries in spatial databases[J]. Data Knowl. Eng, 2004, 49(1): 67-104

[5] 徐红波，郝忠孝. 一种基于 Z 曲线近似 k-最近对查询算法[J]. 计算机研究与发展，2008, 45(2): 310-317

[6] Lin King-Ip, Yang Congjun. The ANN-tree: An Index for efficient approximate nearest neighbor search[C]// Proc. the 7th International DASFAA. Hong Kong, China, April 2001: 174-181

[7] Sack J R, Urrutia J. Voronoi Diagrams[M]. Handbook on Computational Geometry. Ottawa: Elsevier Science, 2000: 201-290

[8] 李博涵，郝忠孝. 一种基于聚类分析的 R* 树结点重叠判定算法[J]. 计算机研究与发展，2008, 45(12): 2154-2161

(上接第 152 页)

高 SA 元素的重用性，又能灵活地规约方面组件与组件、连接件各种复杂的组合关系，从而使得所设计出的 SA 模型更加易于理解、易于重用、易于演化。我们的下一步工作方向是继续修改和完善 AC2-ADL 的语法和语义，并在此基础上设计出一种显示的编织机制，通过它将 AC2-ADL 建立的 SA 模型编织为仍用 AC2-ADL 描述的仅含组件、连接件的 SA 模型，使得 SA 设计人员更加易于分析并验证 SA 整体行为和质量特征。

参 考 文 献

[1] Medvidovic N, Taylor RN. A classification and comparison framework for software architecture description language[J]. IEEE Transactions on Software Engineering, 2000, 26(1): 70-93

[2] 唐稚松. 时序逻辑程序设计与软件工程[M]. 北京：科学出版社，2002, 5: 47-49

[3] Pinto M, Fuentes L, Troya J. DAOP-ADL: An Architecture Description Language for Dynamic Component and Aspect-Based Development[C]// Proceedings 2nd International Conference on Generative Programming and Component Engineering, GPCE 2003. Erfurt, Germany, September 2003

[4] Pessemier N, Seinturier L, Coupaye T. A Model for Developing Component-Based and Aspect-Oriented Systems[C]// Proceedings of the 5th International Symposium on Software Composition, SC 2006. Vienna, Austria, March 2006

[5] Perez J, Ramos I, et al. PRISMA: Towards Quality, Aspect Oriented and Dynamic Software Architectures[C]// Proceedings of the 3rd IEEE International Conference on Quality Software, QSIC 2003. Texas, USA, November 2003

[6] Navasa A, Pérez M A, Murillo J M. Aspect Modeling at Architecture Design[C]// Proceedings of the 2nd European Workshop on Software Architecture, EWSA 2005. Pisa, Italy, June 2005

[7] Batista T, Chavez C, Garcia A, et al. Aspectual Connectors: Supporting the Seamless Integration of Aspects and ADLs[C]// Proceedings of the 20th Brazilian Symposium on Software Engineering, SBES'06. Florianópolis, Brazil, October 2006

[8] 朱雪阳，唐稚松. 基于时序逻辑的软件体系结构描述语言 XYZ/ADL[J]. 软件学报，2003, 14 (4): 714-720