第35卷 第23期 计 算 机 工 程 2009年12月
Vol.35 No.23 Computer Engineering December 2009
·软件技术与数据库· 文章编号：1000—3428(2009)23—0032—03 文献标识码：A 中图分类号：TP391

一种基于R-树的空间索引结构

刘润涛[1]，安晓华[2]，高晓爽[2,3]
(1. 哈尔滨理工大学信息与科学计算技术研究所，哈尔滨 150080；2. 哈尔滨理工大学应用科学学院，哈尔滨 150080；
3. 哈尔滨师范大学数学系，哈尔滨 150500)

摘 要：为了有效构建R-树，通过分析数据矩形的性质，结合改进的K-均值算法，提出一种用于构建R-树的数据矩形聚类新方法，给出基于R-树和四叉树的空间索引结构以及该空间索引结构的构造算法和节点插入算法。研究结果表明，该索引结构具有更紧凑的结构和更高的空间查询效率。

关键词：空间索引；聚类算法；R-树

Spatial Index Structure Based on R-tree

LIU Run-tao[1], AN Xiao-hua[2], GAO Xiao-shuang[2,3]
(1. Institute of Information and Scientific Computing Technology, Harbin University of Science and Technology, Harbin 150080;
2. College of Applied Science, Harbin University of Science and Technology, Harbin 150080;
3. Department of Mathematics, Harbin Normal University, Harbin 150500)

【Abstract】 In order to construct R-tree effectively, this paper proposes a new method of clustering data rectangles used to construct R-tree by analyzing the characteristics of data rectangles and combining the improved K-means algorithm. A spatial index structure based on R-tree and quadtree is proposed. Constructing algorithm and node inserting algorithm for new index structure are given. Research results show that the structure has more compact structure and higher query efficiency.

【Key words】 spatial index; clustering algorithm; R-tree

空间数据的快速索引是实现海量数据管理的关键技术之一，直接影响空间数据库的整体性能。R-树[1]是目前应用最广泛的一种数据结构，由于采用最小外接矩形(MBR)界定空间实体，因此不可避免地导致约束矩形区域重叠并出现空白空间。随着其重叠区域和空白区域的增大，查询效率会大大降低。本文对R-树的算法进行了较为深入的研究，将改进的K-均值聚类算法引入算法中，以减少R-树空间矩形的重叠区域和空白区域，从而大大提高空间查询的效率。

1 R-树

R-树是一种类似于B树的高度平衡树。对于一棵M阶的R-树，其节点结构可描述如下：

叶子节点：($COUNT$,$LEVEL$,<OI_1,MBR_1>,<OI_2,MBR_2>,…,<OI_M, MBR_M>)；

中间节点：($COUNT$,$LEVEL$,<CP_1,MBR_1>,<CP_2,MBR_2>,…,<CP_M, MBR_M>)。

其中，<OI_i, MBR_i>称为数据项，OI_i为空间目标的标识，MBR_i为该目标在k维空间中的最小包围矩形(简称为数据矩形)；<CP_i,MBR_i>称为索引项，CP_i为指向子树根节点的指针，MBR_i代表其子树索引空间，为包围其子树根节点中所有目录矩形或数据矩形的最小包围矩形(简称目录矩形)；$COUNT \leqslant M$指示节点中用的索引项或数据项个数(即该节点的孩子个数)；$LEVEL \geqslant O$指示该节点在树中的层数(O表示叶节点)。

图1中的实线框(1,2,3,4)表示空间目标的“MBR”，即数据矩形，R_1, R_2, R_3表示中间节点(包括根节点)索引项对应的索引空间。图2是对应图1中空间对象集合的R-树。易见空间位置上距离较近的对象在R-树中基本落在同一个节点上，即R-树的数据组织与空间划分问题是一个典型的空间聚类问题。由于已有的聚类算法大多是基于模式识别设计的，在空间应用领域多半只能处理点状目标，因此对于具有点状、线状、面状等多种复杂数据类型的空间对象，算法的应用受到限制[2]。为此，本文采用改进后的K-均值聚类算法实现R-树的构建。

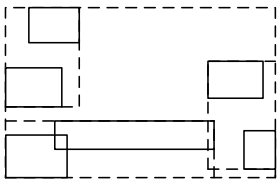

图1 一个空间对象集合

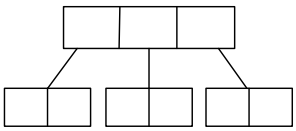

图2 R-树示意图

2 K-均值算法的基本思想

K-均值算法[3-4]是先随机选取k个点作为初始聚类中心，再计算各数据到各聚类中心距离，把数据对象归到离它最近

基金项目：国家自然科学基金资助项目(10571037)；黑龙江省教育厅基金资助项目(11511027)

作者简介：刘润涛(1961－)，男，教授、博士研究生，主研方向：空间数据库；安晓华、高晓爽，硕士研究生

收稿日期：2009-03-22 **E-mail**：liurthar@sina.com

的那个聚类中心所在的类；计算调整后新类的聚类中心，若相邻 2 次聚类中心无变化，数据调整结束。

本文对 K-均值聚类算法的改进和扩充如下：

(1)对 R-树中的面要素，采取 2 个实体之间最近距离原则。试验发现，2 个矩形的距离不能简单地用矩形的中心点距离表示，而应采用 2 个矩形间最近点的距离，如图 3 所示，当 2 个矩形有重叠时，距离为 0[5]。

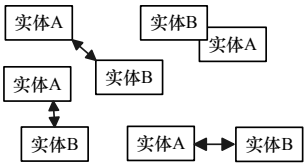

图 3　空间实体间距离的计算

(2)初始聚类中心的选取对聚类结果有较大影响。本文选取空间数据集 *MBR* 的 4 个顶点坐标作为初始聚类中心，试验发现这样可有效避免聚类中心陷入局部。

(3)R-树用空间对象的 *MBR* 表示实体，在计算每个聚类的均值 $M_i(x, y)$, i=1,2,⋯,k 时，用 *MBR* 的 4 个顶点计算每个聚类的均值：$M_i x = N^{-1}\sum_{i=1}^{2N} x_i$, $M_i y = N^{-1}\sum_{i=1}^{2N} y_i$ 。

3　新索引结构

新索引结构的基本思想：计算空间对象集合的 *MBR* 作为根节点，用数据集 *MBR* 的 4 个顶点作为初始聚类中心，其他对象则根据它们与这 4 个初始聚类中心的最远距离分别分配到最相似的聚类中，形成初始聚类。计算每个聚类中对象的均值以确定新聚类中心，重新分配对象并重复这一过程，直到聚类不发生变化，输出的 4 个聚类就是根节点的 4 个孩子节点，再以每个孩子节点作为子空间对象集合，递归进行上述步骤。

R-树的节点结构描述如下：

```
typedef struct Rect{/*描述目标 MBR(数据矩形)和索引空间(目录矩形)的结构*/
float xl, xr, yt, yb;
                } Rect
        typedef struct Node{/*R-树节点项的数据结构*/
                struct Rect rect;
struct Node *child; /*对于叶节点，存储的是目标标识；对于中间节点，存储的是孩子指针*/
                } Node
```

树的生成算法 Tree-Creattion(I,n,head)描述如下：

输入　n 个数据矩形 $r_0, r_1, \cdots, r_{n-1}$

输出　头节点指针 *head*

(1)计算数据集合的最小外包矩形 P，*head->rect=P*(根节点的 *MBR*)。

(2)确定根节点的 4 个孩子节点 *child*(1), *child*(2), *child*(3), *child*(4)。

1)选取 P 的 4 个顶点坐标作为初始聚类中心 $Z_1(1)$, $Z_2(1)$, $Z_3(1)$, $Z_4(1)$。括号内的序号为寻找聚类中迭代运算的序列号：

```
Z1(1)=(rect->xl, rect->yt);
Z2(1)=(rect->xr, rect->yt);
Z3(1)=(rect->xr, rect->yb);
Z4(1)=(rect->xl, rect->yb);
```

2)计算聚类统计量，即 $r_0, r_1, \cdots, r_{n-1}$ 到 4 个聚类中心 $Z_1(1)$, $Z_2(1)$, $Z_3(1)$, $Z_4(1)$的最近距离平方：

```
/*for i=0 to i=n-1 do
d (ri, Z1)= (rix - rect->xl )2+ (riy–rect->yt)2;
d (ri, Z2)= (rix - rect->xr )2+ (riy–rect->yt)2;
d (ri, Z3)= (rix - rect->xr)2+ (riy–rect->yb)2;
d (ri, Z4)= (rix - rect->xl )2+ (riy–rect->yb)2;*/
```

3)$r_0, r_1, \cdots, r_{n-1}$ 按最近距离原则分配给 4 个聚类 $Z_i(1)$, i=1, 2, 3, 4，形成初始聚类，如图 4 所示：

```
/*for i=0 to i=n-1 do
    找到 Zm: d (ri, Zm)= min 1≤j≤4 d(ri, Zj)，把 ri 分配到类 Zm 中;
    Nm=Nm+1;
    记录聚类中数据矩形的个数，初始值为 0*/
```

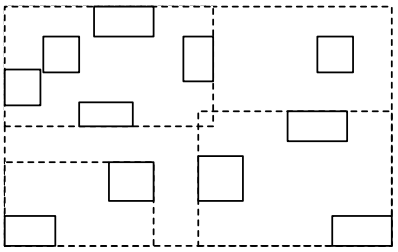

图 4　初始聚类

4)对初始聚类进行调整，通过对聚类中多余节点再分配，形成调整后的聚类，如图 5 所示。

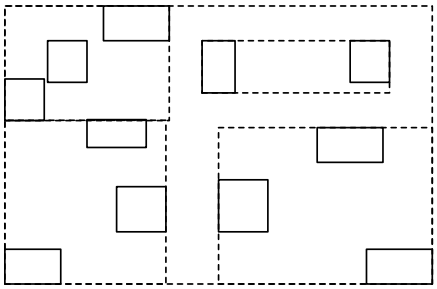

图 5　调整后的聚类

①检查 $Z_i(1)$, i=1,2,3,4 中的数据个数 N_i，如果 $\lfloor n/4 \rfloor \leqslant N_i \leqslant \lceil n/4 \rceil$，则退出节点的分配，如果 4 个聚类中 N_i 都满足条件 $\lfloor n/4 \rfloor \leqslant N_i \leqslant \lceil n/4 \rceil$，则执行步骤(3)。

②假定 $N_i > \lceil n/4 \rceil$，则删除聚类 $Z_i(1)$中距离聚类中心比较远的 $N_i - \lceil n/4 \rceil$ 个数据矩形，设 $p_1, p_2, \cdots, p_m$, $m= N_i - \lceil n/4 \rceil$，按步骤 3)将 $p_1, p_2, \cdots, p_m$ 重新分配到 $Z_j(1)$中，其中，j=1,2,⋯,4, $j \neq i$。

③重复步骤 3)、步骤 4)，直至 n 个数据矩形被均匀地分配到 4 个聚类中。

5)计算聚类中数据矩形的均值 $M_i(x, y)$，将 $M_i(x, y)$作为新的聚类中心，重复步骤 2)~步骤 4)，直到 4 个聚类不再发生变化，输出聚类 $Z_1(t)$, $Z_2(t)$, $Z_3(t)$, $Z_4(t)$。

(3)执行以下步骤：

```
计算 Z1(t)中所有数据的 MBR，并将其赋值给 child(1)->rect;
call Tree-Creattion(child(1)->rect,n1, child1);    /*递归构造中间节点*/
计算 Z2(t)中所有数据的 MBR，并将其赋值给 child(2)-> rect;
call Tree-Creattion(child(2)->rect,n2, child(2));  /*递归构造中间节点*/
计算 Z3(t)中所有数据的 MBR，并将其赋值给 child(3)->rect;
call Tree-Creattion(child(3)->rect,n3, child(3));  /*递归构造中间节点*/
计算 Z4(t)中所有数据的 MBR，并将其赋值给 child(4)-> rect;
```

```
call  Tree-Creation(child(4)->rect,n4,child(4));   /*递归构造中间节点*/
return head;
```

4 R-树的节点插入

R-树的节点插入算法 Node-Insert(head, p)描述如下：

输入 一棵 R-树的头节点 *head*；待插入节点的 *MBR* 数据 *p*

输出 树的头节点指针 *head*

```
{   if(head==nulll) /*给定的树是一棵空树*/
node=(node)malloc(sizeofnode)/*为节点 node 申请空间*/
node->rect =p;
  for j=1 to 4 do
        node->child(j) =null;
              head=node;
  for j=1 to 4 do
         head->child(j) =null;
else if head->child(1)=null then/*给定的树根节点是叶节点*/
  head->child(1)->rect=node->rect;
node->child(2)->rect =p;
child(3)->rect =null;
child(4)->rect =null;
else if child(3)->rect==null then/*给定的树只有 2 个叶节点*/
  child(3)->rect =p;
else if child(4)->rect ==null then /*给定的树只有 3 个叶节点*/
child(4)->rect =p;
else /*给定的树的根节点有 4 个孩子节点*/
a1=area(MBR(p ∪ head->child(1)->rect) ∩ child(2)->rect ∩ child(3)->rect ∩ child(4)->rect);
a2=area(MBR(p ∪ I.child(2) ->rect) ∩ child(1) ->rect ∩ child(3) ->rect ∩ child(4)->rect);
a3=area(MBR(p ∪ child(3) ->rect) ∩ child(2)->rect ∩ child(1)->rect ∩ child(4)->rect);
a4=area(MBR(p ∪ child(4)->rect) ∩ child(2)->rect ∩ child(3)->rect ∩ I.child(1)->rect);
求 i 使得 ai=min{aj}/*寻找 p 应插入的节点*/;
    Node-Insert(child(i), p);
        /*到节点 node 的第 i 个孩子节点中为 p 查找插入位置*/
        I 是包含 I 和 p 的 MBR /*修改节点 node 的 MBR*/
        return head;}
```

5 结束语

本文利用 R-树的聚类特性，将空间聚类引入 R-树的生成算法中，针对空间数据的特点和 R-树的性能约束对现有的 K-均值聚类算法进行了改进和扩充，提出了算法 Tree-Creattion(I,n,head)，该算法对均匀分布和非均匀分布的空间对象均适用。最后给出 R-树节点插入算法 Node-Insert(node,p)。研究表明，新算法构建了一种基于 R-树和四叉树的空间索引结构，较好地提高了 R-树的检索性能。

参考文献

[1] 张明波, 陆　锋. R-树家族的演变和发展[J]. 计算机学报, 2005, 28(3): 289-300.

[2] 张　芩, 王振民. QR-树: 一种基于 R-树和四叉树的空间索引结构[J]. 计算机工程与应用, 2004, 40(9): 100-103.

[3] 周文勇. 改进的 K 均值聚类算法[J]. 计算技术与自动化, 2007, 26(2): 56-58.

[4] 黄继先, 鲍光淑. 基于混合聚类算法的动态 R-树[J]. 中南大学学报, 2006, 37(2): 366-370.

编辑　张　帆

~~~~~~

（上接第 31 页）

不同的缺陷属性按多种方式进行统计和分析。

(3)比较实际特征分布与期望特征分布

把当前过程的缺陷分类结果与期望的缺陷特征分布进行比较，如果存在一定的偏差，则说明阶段性产品某部分处于失控状态，须采取行动解决过程问题。

5.2 实施 ODC 的注意问题

在采用新方法的初期，可以采用讨论会的形式对难以确定的分类进行分析，这样可以有效地保证缺陷属性分类的可靠性和精确性。另外对缺陷分类数据随机抽样检查也是保证分类准确的有效手段。实施 ODC 进行过程改进需要组织、资金和人员多方面的支持，否则会陷入“只进行分析，不进行改进”的误区，同时还应保证对过程改进活动进行监控和跟踪。ODC 是一个需要实践的方法，需要项目成员间的共同合作。此外，软件组织重视质量的企业文化是引入新技术不可缺少的保证。

6 结束语

软件过程的度量和分析能体现过程的当前状态，发现过程中潜在的问题，为过程改进提供依据。ODC 提供了一种过程度量的新方法，极大地改进了开发过程中的度量，提供了合理的量化标准。ODC 作为一项从软件缺陷中提取语义信息的技术，通过快速捕捉软件缺陷语义信息，对缺陷进行分类，建立缺陷类型分布与开发过程的对应关系，从而推断开发过程的状态是否良好。作为一种非常有效的缺陷管理方法，ODC 不仅实施成本低，而且收效显著。

参考文献

[1] Chillarege R, Bhandari I S, Chaar J K, et al. Orthogonal Defect Classification——A Concept for In-process Measurements[J]. IEEE Transactions on Software Engineering, 1992, 18(11): 943-956.

[2] Chillarege R. ODC for Process Measurement, Analysis and Control[C]//Proceedings of the 4th International Conference on Software Quality. McLean, VA, USA: [s. n.], 1994.

[3] IBM Research Center for Software Engineering. Orthogonal Defect Classification[EB/OL]. (2002-02-02). http://www.research.ibm.com/softeng/odc/detodc.htm.

[4] Chillarege R, Biyani S. Identifying Risk Using ODC Based Growth Models[C]//Proceedings of the 5th International Symposium on Software Reliability Engineering. Monterey, Canada: IEEE Press, 1994: 282-288.

[5] Chillarege R, Bassin K A. Software Triggers as a Function of Time——ODC on Field Faults[C]//Proc. of the 5th IFIP Working Conference on Dependable Computing for Critical Applications. Illinois, USA: IEEE Press, 1995: 188-197.

编辑　张正兴
~~~~~~