第 8 卷(A 版) 第 7 期
2003 年 7 月
中国图象图形学报
Journal of Image and Graphics
Vol. 8(A),No. 7
Jul. 2003

空间数据库中连接运算的处理与优化

李立言　秦小麟
(南京航空航天大学计算机科学与工程系,南京　210016)

摘　要　空间数据库的性能问题严重制约了它的应用与发展.由于空间连接运算是空间数据库中最复杂、最耗时的基本操作,因此其处理效率在很大程度上决定了空间数据库的整体性能.尽管目前已经有许多空间连接算法,但空间连接运算的代价估计和查询优化仍然有待进一步研究.众所周知,大部分空间连接算法都是基于 R 树索引实现的,如果参与空间连接运算的关系上没有索引或只有部分索引,那么就需要使用特殊的算法来处理.另外,各种算法的代价评估模型需要一个相对统一的计算方法,实践证明,根据空间数据库的实际情况,使用 I/O 代价来估计算法的复杂性较为合理.在此基础上,针对复杂的空间查询中可能出现多个关系参与空间连接运算的情况,故还需要合理地应用动态编程算法来找出代价最优的连接顺序,以便最终形成一个通用的算法框架.通过对该算法框架的复杂性分析可以看出,在此基础上实现的空间数据库查询优化系统将具有较高的时空效率,并且能够处理非常复杂的空间查询.

关键词　数据库(520・4050)　空间数据库　空间连接运算　R 树索引　动态编程　查询优化
中图法分类号:TP311.13　**文献标识码**:A　**文章编号**:1006-8961(2003)07-0732-06

Processing and Optimization of Join Operation in Spatial Database

LI Li-yan, QIN Xiao-lin
(*Dept. of Computer Science and Engineering, Nanjing University of Aeronautics and Astronautics, Nanjing* 210016)

Abstract　The performance problem of spatial database limits its application and development seriously. Spatial join is the most complex and time-consuming operation in spatial database system. Its efficiency determines the performance of the whole spatial database system to a great extent. Although there are many spatial join algorithms already, cost estimation and query optimization of spatial join operation need further study. Most spatial join algorithms are implemented based on R-tree index, but if the corresponding relations have no indices, or only have partly indices, special algorithms should be used to handle the situation. Cost estimation models of each algorithm need a relatively uniform calculation method. Considering the characteristic of spatial database, it's reasonable to use I/O cost to estimate the complexity of each algorithm. Based on the above approaches, and because complex spatial queries may include multiple relations for spatial join, dynamic programming algorithm should be used to choose the proper join order which has minimum cost. It becomes a universal algorithm framework. Through the complexity analysis of the algorithm framework, the spatial database query optimization system implemented base on this approach will have better spatial and temporal efficiency, and can handle very complex spatial queries.

Keywords　Spatial database, Spatial join operation, R-trees index, Dynamic programming, Query optimization

0　引　言

空间连接运算是空间数据库最重要的基本操作之一.它是根据某个空间谓词(例如“相交”、“包含”)把来自两个关系的数据合并起来的一种操作,例如,“找出长江流经的所有省份的名称”,这就是一个以“相交”为空间谓词的空间查询.

目前,空间连接处理一般分为如下两个步骤:(1)筛选(filter),该步是首先使用简单的数据结构

基金项目:国家自然科学基金资助项目(49971063);江苏省自然科学基金资助项目(BK2001045)
收稿日期:2002-08-06;改回日期:2003-02-27

来近似表示空间对象,其比较常用的数据结构是 MBR(Minimum bounding rectangle),然后对这种简化的对象进行空间连接运算,其所得的结果集称为候选集;(2)细化(refinement),它是以筛选出的结果集为输入来逐一对真实的空间对象进行空间比较操作,看看是否真正符合原来的空间谓词.

由于细化步骤一般都具有很大的时间开销,因此在做查询优化时,应该尽量推迟这一步骤,而是先利用其他的限制条件来减小候选集,以减少时间开销.例如,“找出长江流经的所有人口多于4 000万的省份的名称”,就可以先计算出各个省份与长江的 MBR 相交的候选集,然后从中过滤掉不符合“人口多于4 000万”这一条件的元组,最后再对真正的空间对象进行相交运算即可得到所需的结果.在这个例子中,由于前两个步骤是可以交换的,因此需要对筛选步骤的代价进行评估,以确定合适的执行顺序.

基于 MBR 的 R 树索引是筛选步骤中最常用的数据结构.本文将分别针对参与连接运算的两个关系上有两个、一个或者没有 R 树索引的情况给出相应的空间连接算法及其代价评估模型.

1　研究背景

R 树是最常用的空间存取方法,它可以看成是 B 树在多维空间上的扩展[1~3].同 B 树一样,R 树是一种可动态调整的平衡树,且每个 R 树结点包含若干个〈*ptr*,*mbr*〉对.对于叶结点,*ptr* 是指向某个空间对象的指针,*mbr* 则是对应于该对象的最小外框;对于中间结点,*ptr* 指向某个子结点,而 *mbr* 则是该子结点所包含的所有空间对象的最小外框.

基于 R 树的空间连接算法[1]最早由 Brinkhoff 等人提出.如果参与连接运算的两个关系上都有 R 树索引,那么该算法将同步进行深度优先遍历,同时排除与 MBR 不相交的子树.由于该算法具有很高的效率,因此至今仍是最重要的空间连接算法之一.Huang 等人提出了一种基于广度优先搜索的改进型 R 树连接算法[4],该算法当缓存空间足够大时,具有更高的效率.

当仅有一个关系上有 R 树索引时,可以使用传统的嵌套索引循环(indexed nested loop)连接算法,即通过循环搜索 R 树索引来逐个匹配无索引关系中的每个元组,但这种方法只适用于关系非常小的时候,在通常情况下效率极低.为此,Lo 和 Ravishankar 提出了种子树连接算法(seeded tree join)[5],该算法首先以已有的 R 树为种子来对没有索引的关系建立一棵类 R 树索引,然后调用 R 树连接算法来完成连接运算.

如果参与连接运算的两个关系上都没有索引,则可以使用空间哈希连接算法(spatial hash-join)[6].该算法是使用采样信息来划分第 1 个关系,同时创建若干个“桶(buckets)”;然后把第 2 个关系按照同样的桶来划分.这样划分完成后,就可以在范围相同的桶上执行连接操作了.

Patel 和 DeWitt 提出了另一种基于哈希算法的空间连接算法,称为“基于划分的空间归并连接(partition based spatial merge join)”算法[7],该算法首先将空间进行正规划分,然后将参与连接的两组空间对象映射到分区上,再使用空间扫描算法把覆盖了同样区域的分区组连接起来.由于某些对象可能会占用多个分区,所以需要把连接运算的输出结果进行排序,以便删除重复的元组.而这种排序操作可能会带来很大的 I/O 开销.

把以上这些空间连接算法组合起来,再针对不同的情况使用相应的算法,就可以处理空间数据库中的各种连接操作了,然而,如果参与连接运算的关系数目多于两个,则连接顺序的选择就变得十分重要.这是因为不同的连接顺序所产生的时间和空间代价相差很大,而连接顺序的组合会随关系数目的增加成指数增长的缘故.为了确定合适的连接顺序,首先需要估计各个算法的查询代价,尽管实际的查询代价往往会受到数据分布情况的影响,但是代价估计的目的不是准确的计算代价大小,而是帮助选择一个物理查询计划.由此可见,只要估计方法能够反映各个查询计划的相对代价大小就可以达到目的.有了算法的查询代价后,接下来还需要找到总代价最小的连接顺序.为了避免穷举法的低效和爬山法的盲目性,本文使用了动态编程算法,该算法是采用“自底向上”的策略,由于在计算较大子表达式的计划时,只考虑每个子表达式的最佳计划,因此具有较高的效率,而且一般都能找到最优计划.

2　典型算法及代价评估

2.1　R 树连接算法

R 树连接算法的核心是对两棵 R 树同步进行深度优先遍历,当两个结点的 *mbr* 相交时,则递归

地进入下一层扫描，直到叶结点相交，输出结果为止[1]，其算法的伪代码如下：

算法 1 R 树连接算法

```
r_tree_join(Node I1, Node I2)/* I1、I2—R 树的结点 */
{
  for(I1 中的每个元素 E1)
    for(I2 中的每个元素 E2)
      if (overlap(E1.mbr, E2.mbr))
        if (I1,I2 是叶结点)
          /* 输出到结果集 */
          output(E1.ptr, E2.ptr);
        else if (I1 是叶结点) {
          /* 从磁盘或缓冲区读入页面 */
          read_page(E2.ptr);
          r_tree_join(E1.ptr, E2.ptr);
        } else if (I2 是叶结点) {
          read_page(E1.ptr);
          r_tree_join(E1.ptr, E2.ptr);
        } else {
          read_page(E1.ptr);
          read_page(E2.ptr);
          r_tree_join(E1.ptr, E2.ptr);
        }
}
```

换个角度看，R 树连接算法实际上是一系列的窗口查询(window queries)，其分别由 $\boldsymbol{I}_1$ 和 $\boldsymbol{I}_2$ 结点的元素作为数据和查询窗口.

在 R 树连接算法中，由于每个结点都可能会被多次存取，因此内存缓冲区管理策略对算法的代价影响很大. 假设内存页面缓冲采用最近最少使用法 LRU(Least recently Used)策略，则 R 树连接算法的代价可表示为[2]：

$$C_{RJ}=T_1+T_2+(N_A(\boldsymbol{I}_1,\boldsymbol{I}_2)-T_1-T_2)P(M) \quad (1)$$

其中，T_1、T_2 表示这两棵 R 树所占用的内存页面数；$N_A(\boldsymbol{I}_1,\boldsymbol{I}_2)$表示算法所存取的 R 树结点的总数；$P(M)$表示所要存取的结点不在缓冲区的概率，$M$ 是缓冲区的大小. 关于 $N_A(\boldsymbol{I}_1,\boldsymbol{I}_2)$和 $P(M)$的详细计算方法请参见文献[2].

2.2 种子树连接算法

设空间对象集 $\boldsymbol{A}$、$\boldsymbol{B}$ 参与连接运算，其中，$\boldsymbol{A}$ 上有 R 树索引 $\boldsymbol{R}_A$，$\boldsymbol{B}$ 上无索引. 种子树连接算法可分为如下 4 个步骤：

(1) 采种(seeding)过程，根据系统参数(主要考虑对象数目和内存大小)计算采种的高度 k，把 $\boldsymbol{R}_A$ 顶部的 k 层拷贝作为种子，其中最底层结点中的项称为槽(slots). 拷贝完成后，再把槽中的指针清空.

(2) 生长过程，把 $\boldsymbol{B}$ 中的每个对象插入到种子树中，从根节点开始，先找到合适的槽，如果是第 1 次插入此槽，则需要分配一个新的结点；然后通过调整槽指针指向该结点来把对象插入到该新结点中；如果槽已经生成了子结点，则视此结点为 R 树根结点；最后按照 R 树的插入算法把对象插入 R 树中合适的位置. 槽下面的这种 R 树称为生成子树(grown subtree).

(3) 清理过程，按照槽的子结点的 mbr，逐层向上调整种子中结点的 mbr，如果槽中没有子结点，则删去此槽，同时相应调整上层的 mbr.

(4) 树匹配(tree matching)过程，此过程相当于用 R 树连接算法来处理 $\boldsymbol{R}_A$ 和生成的种子树.

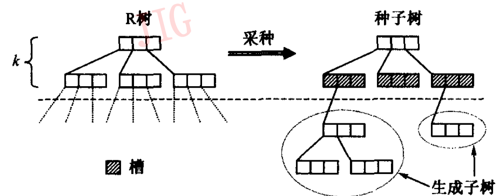


图 1 种子树的构造

种子树并不是一棵严格的 R 树，由于插入各个槽的对象数目可能相差很大，很可能导致生成子树的高度不相等，所以种子树不是一棵平衡树，但是由于基于深度优先的树匹配算法并不要求参与运算的树是平衡树，所以仍然适用. 种子树的优点是提高了树匹配的效率，由于种子树与提供种子的 R 树相应的各个结点之间有相同或相似的范围，所以匹配时的命中率很高，而与结点相交的比例则很低. 由于种子树连接算法在生长过程中占用内存很大，因此如果树的大小超出了内存缓冲区的大小，那么就只有把一部分写入磁盘，需要时再读出. 为了避免插入对象时频繁地读写磁盘，可以把将要插入某个槽的对象先存储到对应的临时文件中，这样当所有的对象都插入后，再把各个槽对应的临时文件读出来生成 R 树，以便成为该槽的生成子树. 这种方法由于仍然要求每个槽至少分配到一个内存页面，才能有效避免缓冲区摆动(buffer thrashing)，因此应该通过修正系统参数来影响槽的数目，以满足需要.

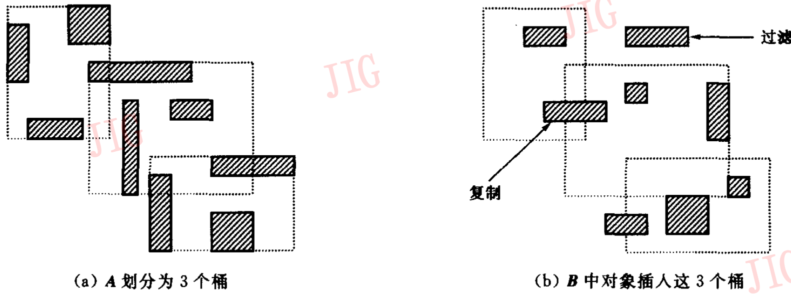
为了估计种子树算法的代价，可首先假设系统内存足够大，这样虽能够避免缓冲区摆动，但是不一定能容纳整个种子树，因此需要在生长阶段使用临

时文件. 在采种阶段,算法的代价为 $s_A \cdot T_A$,即从 $\boldsymbol{R}_A$ 树上拷贝 k 层所存取的页面数,其中 s_A 表示 $\boldsymbol{R}_A$ 树中前 k 层结点所占的比例;在生长阶段,$\boldsymbol{B}$ 中的每个空间对象需要被读出并写到某个槽对应的临时文件中,最后还要读出来创建生成子树,其存取代价为 $3P_B$;清理阶段本来没有存取操作,不过清理后的整个种子树还要写回磁盘,其代价为 T_B;树匹配阶段的代价约为 T_A+T_B,即两棵树的所有页面都被读入的代价. 这样,种子树连接算法的总代价[3]可表示为

$$C_{\mathrm{STJ}} = (1 + s_A)T_A + 3P_B + 2T_B \tag{2}$$

2.3　空间哈希连接算法

空间哈希连接算法是由基于传统关系模型上的哈希连接算法扩展而来的[6]. 设两个空间对象集 $\boldsymbol{A}$、$\boldsymbol{B}$ 参与连接运算,运算时首先根据系统参数(主要考虑对象数目和内存大小)来计算桶的数目,然后对 $\boldsymbol{A}$ 进行采样,以确定各个桶的初始范围,再把 $\boldsymbol{A}$ 中的对象插入到各个桶中,同时根据需要扩充桶的范围;最后把 $\boldsymbol{B}$ 中的对象插入到同样的桶中,如果某个空间对象覆盖了两个以上的桶区域,则要再复制相应份数的该对象,并将其插入到每个相交的桶中,这样不与任何桶相交的对象就被过滤掉了,而且这种桶划分的质量直接影响空间哈希连接算法的效率. 由于采样的不确定性,无法保证 $\boldsymbol{A}$ 被均匀划分(即各个桶中所包含的对象数目接近相等),且由于 $\boldsymbol{A}$ 和 $\boldsymbol{B}$ 中对象的空间分布情况不同,更无法保证 $\boldsymbol{B}$ 被均匀划分,因此改进采样和桶划分的质量是提高算法效率的关键.



(a) $\boldsymbol{A}$ 划分为 3 个桶　　(b) $\boldsymbol{B}$ 中对象插入这 3 个桶

图 2　空间哈希连接的划分过程

划分完成以后,$\boldsymbol{A}$ 和 $\boldsymbol{B}$ 就可以在相同的桶区域上直接进行连接运算. 当某个桶中的 $\boldsymbol{A}$ 或 $\boldsymbol{B}$ 的对象子集能够全部装入内存时,则只需对这两个集合进行单次扫描,即可完成这个桶中的连接运算,但是如果假设不能成立,那么就只好先对桶中的 $\boldsymbol{A}$ 或 $\boldsymbol{B}$ 的对象子集建立 R 树索引,再使用嵌套索引循环连接算法进行连接运算,但由于这样做的代价很高,因此应该通过修正系统参数来增加桶的数目,以尽量避免这种情况.

如果要估计空间哈希连接算法的代价,则需假设内存足够大,即能够容纳单个桶中的 $\boldsymbol{A}$ 或 $\boldsymbol{B}$ 的对象子集. 在空间对象集 $\boldsymbol{A}$ 的划分阶段,采样代价为 cD,其中,D 为桶的数目,c 是一个常数,表示随机 I/O 的代价,采样后对 $\boldsymbol{A}$ 的划分代价为 $2P_A$,即 $\boldsymbol{A}$ 中每个对象读写一次的代价;在空间对象集 $\boldsymbol{B}$ 的划分阶段,读取代价为 P_B,写入代价为 $(r_B-f_B)P_B$,其中 r_B 表示重复对象数,f_B 表示过滤对象数;连接阶段代价为 $P_A+(r_B-f_B)P_B$,即写回磁盘的空间对象全部读取一次的代价. 空间哈希连接算法的总代价[3,8]为

$$C_{\mathrm{HJ}} = cD + 3P_A + (1 + r_B - f_B)P_B \tag{3}$$

3　基于动态编程的查询优化

空间数据库常常要处理多个空间关系的连接,其最关键的问题是确定空间连接的顺序. 动态编程算法提供了一种自底向上的优化策略,它能够以较高的效率从可能的查询计划中找出代价较小的计划. 由于动态编程算法没有考虑全局优化,因此可能在某些情况下找不到最优计划,但是通过适当的改进(例如 Selinger 风格优化[9]),就可以在大多数情况下找到代价最优的计划.

动态编程的基本思想是对一个代价表进行填充,尔后只保留推出最终结论所需的最少信息. 假设有 n 个关系 $\boldsymbol{R}_1,\boldsymbol{R}_2,\cdots,\boldsymbol{R}_n$ 参与空间连接,则需要为包含这些关系中的两个到 n 个关系的每一个“可能”的子集构建一个表项,“可能”是指这个子集中的关系能够通过空间谓词连接起来,换句话说,若以关系

为顶点,以空间连接谓词为边来构造一个查询图,则此图的每个含有两个及两个以上顶点的连通子图应占一个表项.每个表项 $\boldsymbol{Q}$ 代表一种查询策略,它包含如下3种信息:

(1) 这个子集中的全部关系连接的最小代价($\boldsymbol{Q}$.cost);

(2) 达到最小代价的连接计划($\boldsymbol{Q}$.plan);

(3) 这些关系连接结果的估计大小($\boldsymbol{Q}$.size).

最小代价可以按照上一节给出的公式来估计,而连接结果的估计大小则与连接算法、连接次序无关,只与数据的分布情况有关.假设数据为正态分布,则连接结果的大小 $S(\boldsymbol{A},\boldsymbol{B})$[2,3]可表示为

$$S(\boldsymbol{A},\boldsymbol{B}) = S_B \cdot S_A \cdot (r_A + r_B)^2 \qquad (4)$$

其中,r_A 表示集合 $\boldsymbol{A}$ 中空间对象的 *mbr* 的平均边长相对值,其取值范围被标准化到[0,1];S_B,S_A 分别为集合 $\boldsymbol{A}$、$\boldsymbol{B}$ 中元素个数.

综合以上结论,即可实现基于动态编程的空间连接查询优化算法,其框架如下:

算法2 空间连接查询优化算法

```
optimize(Query Q, int n)
/*Q—逻辑查询计划,其中包括关系集{R1,…,Rn};n—关系数 */
{
  for (each {A,B} in {R1,…,Rn} and {A,B}可连接) {
  /* 所有具有连接谓词的关系对 */
    if (A,B 有索引) {
      Q{A,B}.cost=C_RJ(A,B);/* 使用式(1) */
      Q{A,B}.plan=[RJ,A,B];/* 使用广义表保存查询计划 */
    } else if (A 有索引) {
      Q{A,B}.cost=C_STJ(A,B);/* 使用式(2) */
      Q{A,B}.plan=[STJ,A,B];
    } else if (B 有索引) {
      Q{A,B}.cost=C_STJ(B,A);
      Q{A,B}.plan=[STJ,B,A];
    } else {
      Q{A,B}.cost=C_HJ(A,B);/* 使用式(3) */
      Q{A,B}.plan=[HJ,A,B];
    }
    Q{A,B}.size=S(A,B); /* 使用式(4) */
  }
  for (i=3 to n)
    for (each {T1,…,Ti} in {R1,…,Rn} and {T1,…,Ti}可连接) {
    /* 所有具有连接谓词的关系集 */
      Q{T1,…,Ti}.cost=MAX; Q{T1,…,Ti}.plan=NULL;
      for (each 分解 Q{T1,…,Ti}→{Q1,Q2} and {Q1,Q2}可连接) {
      /* 把关系集分解为两个具有连接谓词的关系子集 */
        if (|Q1|=1 and Q1 中的关系 R 有索引) {
        /* 单关系子集,且该关系上有索引 */
          {Q1,Q2}.cost=Q2.cost+C_STJ(Q1,Q2);
          {Q1,Q2}.plan=[STJ,Q1.plan,Q2.plan];
        } else {
          {Q1,Q2}.cost=Q1.cost+Q2.cost+C_HJ(Q1,Q2);
          {Q1,Q2}.plan=[HJ,Q1.plan,Q2.plan];
        }
        if ({Q1,Q2}.cost<Q{T1,…,Ti}.cost) {
        /* 只保留已知的代价最小的查询计划 */
          Q{T1,…,Ti}.cost={Q1,Q2}.cost;
          Q{T1,…,Ti}.plan={Q1,Q2}.plan;
          Q{T1,…,Ti}.size=S(Q1,Q2);
        }
      }
    }
}
```

该算法步骤为:先计算所有两个关系连接的代价,然后逐渐增加关系数,当计算 i 个关系的连接代价时,由于从两个到 $i-1$ 个关系的连接代价都已经算出,因此只需将其分解即可;同时由于中间关系都没有索引,所以只有当分解出的某项只含有一个关系,并且该关系上有索引时,才能使用种子树连接算法,否则就只能使用哈希连接算法.

该算法中用到了 C_{RJ}、C_{STJ}、C_{HJ} 和 S 这4个估计函数,分别对应于前面所描述的4个公式.这4个函数均以两个关系作为输入参数,通过对这两个关系进行统计信息的分析,即可计算出连接的代价或大小.其中,原始关系的统计信息由空间数据库直接提供,而中间关系的统计信息则需要通过估计函数来间接计算得到.

例如,设有3个关系 $\boldsymbol{R}_1$、$\boldsymbol{R}_2$、$\boldsymbol{R}_3$ 进行空间连接,3个关系之间都有连接谓词,其中,$\boldsymbol{T}_1$ 上没有索引,$\boldsymbol{R}_2$、$\boldsymbol{R}_3$ 上有R树索引.在寻找最佳连接顺序时,首先计算这3个关系两两连接的代价,即得到中间关系 $\boldsymbol{R}_{1,2}$、$\boldsymbol{R}_{2,3}$ 和 $\boldsymbol{R}_{1,3}$ 的估计信息,其中,计算 $\boldsymbol{R}_{1,2}$ 和 $\boldsymbol{R}_{1,3}$ 时,使用种子树连接算法;而计算 $\boldsymbol{R}_{2,3}$ 时,则使用R树连接算法,同时每个中间关系的大小也要估算出来;接下来即可利用这3个中间关系来计算 $\boldsymbol{R}_1$、$\boldsymbol{R}_2$、$\boldsymbol{R}_3$ 连接的总代价,其中,计算 $\boldsymbol{R}_{2,3}$ 与 $\boldsymbol{R}_1$ 连接时,使用

空间哈希连接算法;而计算 $R_{1,2}$ 与 R_3、$R_{1,3}$ 与 R_2 连接时,则使用种子树连接算法;最后根据总代价的大小,就可以找到最佳的连接策略.

动态编程算法的时空复杂度与输入数据之间的关系有关,对于算法 optimize 而言,时空开销最大的情况就是任意两个输入关系之间都有连接谓词,即连接关系图是一个完全图.在这种情况下,算法的第 i 层需要在代价表中存储 $C(i,n)$ 个表项,其总的空间开销为

$$C_{space}=C(2,n)+C(3,n)+\cdots+C(n,n)=2^n-n-1$$

同样,在算法的第 i 层需要进行分解,其时间开销为

$$C(i,n)\cdot(C(1,i)+C(2,i)+\cdots+C(i-1,i))/2$$
$$=C(i,n)\cdot(2^{i-1}-1)$$

而总的时间开销为

$$C_{time}=C(2,n)+\sum_{i=3}^{n}(C(i,n)\cdot(2^{i-1}-1))$$
$$=\sum_{i=2}^{n}(C(i,n)\cdot(2^{i-1}-1))$$

4 结　论

本文讨论了空间连接运算的处理和优化过程中的关键技术问题,总结了基于 R 树索引的几种空间连接算法及相应的代价模型,并使用动态编程的思想把这些算法整合起来,给出了实现空间查询优化的算法框架.

通过设计与实现一个基于 Realms[10]的空间分析模块,并集成进 PostgreSQL 中,从而实现了一个空间分析数据库原型系统 SADBS(Spatial analysis database systems).在此基础上,笔者等正在研究和实现本文所描述的空间查询优化技术,目前已经完成了 R 树索引模块与空间分析模块的集成,而且各种空间连接算法和动态编程算法也有了初步实现,正在进一步完善之中.

参 考 文 献

1 Brinkhoff T, Kriegel H P, Seeger B. Efficient processing of spatial joins using R-trees[A]. In: Proceedings of the 1993 Association for Computing Machinery Special Interest Group International Conference on Management of Data [C]. Washington,D.C.USA,1993: 237～246.

2 Huang Y W, Jing N, Rundensteiner E A. A cost model for estimating the performance of spatial joins using R-trees[A]. In: Proceedings of Ninth International Conference on Scientific and Statistical Database Management[C]. Olympia, Washington USA,1997: 30～38.

3 Mamoulis N, Papadias D. Integration of spatial join algorithms for processing multiple inputs[A]. In: Proceedings of the 1999 Association for Computing Machinery Special Interest Group International Conference on Management of Data [C]. Philadelphia,Pennsylvania USA,1999: 1～12.

4 Huang Y W, Jing N, Rundensteiner E A. Spatial joins using R-trees: Breadth first traversal with global optimizations[A]. In: Proceedings of 23rd International Conference on Very Large Data Bases[C]. Athens,Greece,1997:396～405.

5 Lo M L, Ravishankar C V. The design and implementation of seeded trees: an efficent method for spatial joins [J]. IEEE Transactions on Knowledge and Data Engineering, 1998,10(1): 136～152.

6 Lo M L, Ravishankar C V. Spatial hash-joins [A]. In: Proceedings of the 1996 Assciation for Computing Machinery Special Interest Group International Conference on Management of Data[C]. Montreal,Cananda,1996: 247～258.

7 Patel J M, DeWitt D J. Partition based spatial merge join[A]. In: Proceedings of the 1996 Association for Computing Machinery Special Interest Group International Conference on Management of Data[C]. Montreal,Cananda, 1996: 259～270.

8 Koudas N, Sevcik K C. Size separation spatial join[A]. In: Proceedings of the 1997 Assciation for Computing Machinery Special Interest Group International Conference on Management of Data[C]. Tucson, Arizona USA, 1997: 324～335.

9 Garcia-Molina H, Ullman J D, Widom J. Database system implementation[M]. Upper Saddle River, New Jersey USA: Prentice Hall,2000:Chapter 7,Section 6.

10 Güting R H, Schneider M. Realms: A foundation for spatial data types in database systems[A]. In: Proceedings of the 3rd International Symposium on Large Spatial Databases [C]. Singapore,1993: 14～35.

李立言　1978 年生,南京航空航天大学硕士研究生.研究方向为空间数据库、主存数据库.

秦小麟　1953 年生,硕士学位,现任南京航空航天大学教授,博士生导师.当前的研究方向为空间分析 DBMS、时空数据库、GIS、安全数据库等.