高技术通讯 2010 年 第 20 卷 第 8 期:833 ~ 839

doi:10.3772/j.issn.1002-0470.2010.08.011

基于 B^Z 树深度优先高维空间范围查询算法①

徐红波②* 郝忠孝* **

(*哈尔滨理工大学计算机科学与技术学院 哈尔滨 150080)

(**哈尔滨工业大学计算机科学与技术学院 哈尔滨 150001)

摘 要 考虑到在低维空间中基于线性扫描、R 树、VA 文件和 NB 树的空间范围查询算法的查询效率较高,而在高维空间中这些算法均出现不同程度的性能恶化现象,将降低空间维度作为解决高维空间范围查询问题的关键,并利用基于 Z 曲线的网格划分方法降低空间维度,使用 Z 区域聚类相似数据给出了一种改进的索引结构 B^Z 树,提出了一种深度优先高维空间范围查询算法 ZRRQ。该算法采用高效剪枝策略,能够快速遍历 B^Z 树。实验结果表明,在高维空间中该算法优于基于线性扫描、R 树、VA 文件和 NB 树的空间范围查询算法。

关键词 高维空间,范围查询,降维,Z 曲线,Z 区域,B^Z 树

0 引 言

近年来,众多领域出现了高维数据应用,例如地理信息系统、数据仓库和基于内容的多媒体信息检索[1]。在这些系统中数据常以高维向量形式存在,且数据规模较大,范围查询是常用的检索方式。范围查询,又称窗体查询或矩形查询,在高维空间中,范围查询查找落在查询区域中的点,或者给定查询点和查询半径,查找到查询点的距离小于等于查询半径的点[2]。在实际查询过程中,例如在电子商务,医疗诊断等领域,用户不仅对检索精度有一定的要求,对检索响应时间也会提出很高的要求,这样必须保证系统有较高的检索效率,因此有必要研究大规模高维向量空间范围的查询算法。本文提出了一种基于 B^Z 树的深度优先高维空间范围查询算法——ZRRQ,它从降低空间维度、数据聚类和剪枝策略三方面解决了高维空间范围查询问题。

1 高维空间范围查询算法分析

目前有线性扫描等多种算法。

线性扫描算法的基本操作是依次遍历点集中的每个点。在访问每个点时,判断当前点是否落在查询区域中,若落在,则该点在结果集中,否则继续判断下一个点,直到遍历点集中的所有点。显然,线性扫描算法的计算代价太大。

为了降低计算复杂性,目前已有许多文献对此提出了不同的解决方法[3,4],其中最经典的空间索引结构应该是 R 树及其变种树。基于 R 树的空间范围查询算法采用递归策略,算法从根结点开始从上向下、深度优先地遍历整棵树。若当前结点是索引结点,则判断结点的分支对应的最小外包矩形是否与查询区域相交,若相交,递归遍历该分支。若当前结点是叶子结点,则判断叶子结点中的数据点是否落在查询区域中。R 树家族在建树过程中存在最小外包矩形之间重叠现象。在高维空间中,重叠现象尤为严重。算法的执行时间指数依赖于空间维度,几乎线性扫描整个数据空间。

近年来,学者们从两方面出发提出解决"维度灾难"的方法:(1) 简化高维向量表示形式,利用近似向量代替高维向量,例如 VA 文件[5];(2) 将高维向量通过一定转换方式变成一维表示形式,利用经典线性索引技术 B^+ 树存储转换后的数据,例如 NB 树[6]、金字塔技术[7]。VA 文件使用位串(点所在网格的坐标)近似表示点,其文件大小明显小于源文件大小。算法顺序扫描位串,若查询区域包含位串,则位串对应的点在结果中,否则,判断查询区域与位串

① 黑龙江省自然科学基金(F200601)资助项目。

② 男,1980 年生,博士生,讲师;研究方向:空间数据库索引结构及查询算法;联系人,E-mail: x_h_b@tom.com

(收稿日期:2009-04-10)

是否相交,若相交,则访问在源文件中对应点的坐标,判断该点是否落在查询区域内。算法明显减少了磁盘IO次数。当判断查询区域与位串之间位置关系时,需要将位串分解成 d 个网格坐标,这导致算法的CPU运算代价较高。

NB树首先计算点到坐标原点的距离,以该距离作为关键字将点存储在 B^+ 树中。算法在 B^+ 树中定位查询点,向前和向后扫描 B^+ 树中的双向链表。当前驱点的距离值小于查询点的距离值减去查询半径时,退出向前遍历,否则,访问前驱点的坐标,判断该点与查询点之间的距离是否小于查询半径,若小于,则该点在结果中,直到 B^+ 树中双向链表的表头为止。依据点到坐标原点的距离值,算法只需要访问很少的点,其查询效率较高。但是,算法只适用于欧几里德距离。

金字塔技术通过转换函数将高维点向量转换成一维数值表示形式,并将转换的数值利用 B^+ 树建立索引,达到缩小搜索空间的目的。由于检索过程引入了不必要的距离计算,并且随着数据量的增加这类计算也将增多,因此检索效率下降较快。金字塔优化技术可提高检索剪枝效率,在不损失精度的前提下,尽可能减少高维向量距离计算次数,使得改进后的金字塔技术能实现快速范围查询。

本文从降低空间维度、数据聚类和剪枝策略三个方向上解决高维空间范围查询算法。在降低空间维度的同时空间填充曲线具有较优数据聚类性质,适用于众多空间查询问题,如最近邻、最近对[8]等。本文采用基于空间填充曲线的网格划分方法降低空间维度,使用Z区域聚类数据给出了一种基于Z区域的索引结构 B^Z 树,提出了一种基于 B^Z 树的深度优先空间范围查询算法。该算法采用高效剪枝策略,使得其查询效率较高。

2 基础知识

2.1 空间填充曲线

空间填充曲线是一种降低空间维度的方法。它像线一样穿过空间中每个离散网格,且只穿过一次,按照线性顺序对网格进行编号。空间填充曲线主要有Hilbert曲线、Z曲线和Gray曲线。Hilbert曲线的数据聚类特性最优,Z曲线的数据聚类特性最差;Hilbert曲线的映射过程最复杂,Z曲线的映射过程最简单[9]。

基本Z曲线是大小为 2×2 且阶为1的曲线。为了获得 i 阶Z曲线,需将基本Z曲线的网格 G_0、G_1、G_2 和 G_3 由 $i-1$ 阶Z曲线进行填充。1阶Z曲线(图1(a))的4个网格由1阶Z曲线进行填充得到2阶Z曲线(图1(b));1阶Z曲线的4个网格由2阶Z曲线进行填充将得到3阶Z曲线。

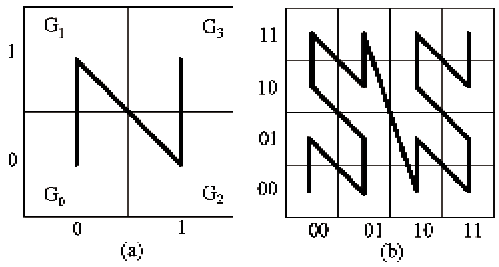

图1 Z曲线

当空间维度 d 为定值时,曲线的阶 m 决定空间分割的粒度,m 越大网格越小。m 阶Z曲线将二维空间分割成大小相等的 2^{2m} 个网格,且依次通过每个网格。同理,m 阶Z曲线将 d 维空间分割成大小相等的 2^{dm} 个网格,且依次通过每个网格。

2.2 空间范围查询

定义1 查询区域定义为位于 d 维空间中的超矩形,记作 $QR = < QR_l(a_1,\cdots,a_d), QR_u(b_1,\cdots,b_d) >$,其中 QR_l 为左下角顶点坐标,QR_u 为右上角顶点坐标。

定义2 设点集 P,查询区域 $QR = < QR_l(a_1,\cdots,a_d), QR_u(b_1,\cdots,b_d) >$,空间范围查询定义为 $RQ(P,QR) = \{p(p_1,\cdots,p_d) \mid p \in P \wedge p \in QR\}$,其中 $p \in QR$ 定义为 $a_i \leqslant p_i \leqslant b_i (i = 1,\cdots,d)$。

例1: 由图2可知整个空间 $\Omega = [0,100] \times [0,100]$,$QR = < QR_l(9,52), QR_u(35,82) >$。点 QR_l

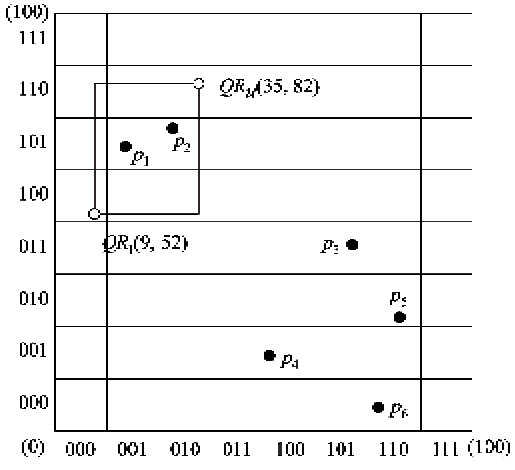

图2 二维空间范围查询

所在网格的坐标为$(000_2,100_2)$,点 QR_u 所在网格的坐标为$(010_2,110_2)$。由点的分布可知空间范围查询 $RQ(P,QR)=\{p_1,p_2\}$。

3 网格划分方法

网格划分将区间分割成长度相等的子区间。每维的划分彼此独立,即每维的分区数目可以相等,也可不等。为了简化问题,假设每维的分区数目均相等,且将区间从左向右依次进行编号,编号作为区间的坐标。在一维空间中网格划分的结果是线段;在二维空间中网格划分的结果是矩形;在三维空间中网格划分的结果是长方体;在 d 维空间中网格划分的结果是超矩形。

将空间进行网格划分之后,如何根据点的坐标计算点的网格坐标?

定义 3 设点 $p(p_1,\cdots,p_d)\in\Omega=D_1\times\cdots\times D_d(D_i=[l_i,u_i](1\leqslant i\leqslant d)$,每维的分区数目均为 $2^m(m\in N)$,点 p 的网格坐标记作 $g(p)=(g_1,\cdots,g_d)$,其中 $g_i=\lfloor p_i/((u_i-l_i)/2^m)\rfloor(1\leqslant i\leqslant d)$。

从每维的分区数目均为 2^m 可知网格坐标的分量 g_i 可用长度为 m 的二进制位串表示,即 $g_i=g_{i1}\cdots g_{ij}\cdots g_{im}((1\leqslant j\leqslant m)\wedge(g_{ij}(\{0,1\}))$。

高维空间范围查询的查询区域是超矩形。超矩形可由主对角线上的两个顶点确定。

定义 4 设 m 阶 Z 曲线填充 d 维空间 Ω,任意网格的坐标为 $g(g_1,\cdots,g_d)(g_i=g_{i1}\cdots g_{im}(1\leqslant i\leqslant d))$,网格在 Z 曲线上的位序称为网格的 Z 值,网格的 Z 值定义为 $Z(g)=\sum_{j=1}^{m}\sum_{i=1}^{d}g_{ij}\cdot 2^{d(m-j)+(d-i)}$。

例 2:图 3 给出了网格 $g(011_2,110_2)$ 的 Z 值计算过程,将计算 Z 值的过程称为位交插操作。

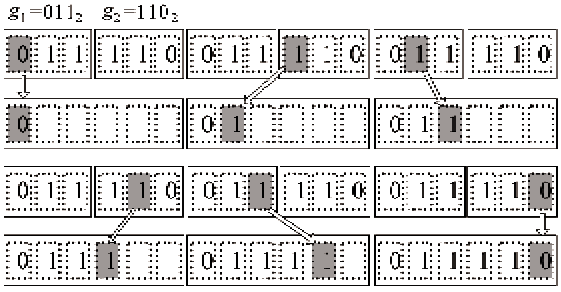

图 3 网格 $g(011_2,110_2)$ 的位交插操作

引理 1 集合 $G=\{g(g_1,\cdots,g_d)\mid g_i=g_{i1}\cdots g_{ij}\cdots g_{im}((1\leqslant i\leqslant d)\wedge(1\leqslant j\leqslant m)\wedge(g_{ij}\in\{0,1\}))\}$ 到集合 $L=\{0,\cdots,2^{md}-1\}$ 的关系 $Z(g)$ 是一个函数。

证明:当网格坐标取最小值 0 时 $Z(g(0,\cdots,0))=0$;当网格坐标取最大值 2^m-1 时 $Z(g(2^m-1,\cdots,2^m-1))=2^{md}-1$。根据位交插操作可知,0 是 $Z(g)$ 的最小值,$2^{md}-1$ 是 $Z(g)$ 的最大值,即命题 $(\forall g\in G)Z(g)\in L$ 成立,故 $Z(g)$ 是一个函数。证毕。

引理 2 函数 $Z(g)$ 为从 G 到 L 上的入射。

证明:根据入射定义可知,只需证明命题 $(\forall g^1,g^2\in G)g^1\neq g^2\Rightarrow Z(g^1)\neq Z(g^2)$ 成立。假设 g^1 和 g^2 之间至少在第 i 维坐标的第 j 位上不相等,即 $g^1=(g_1,\cdots,g_{i1}\cdots g_{ij}=0\cdots g_{im},\cdots,g_d)$ 和 $g^2=(g_1,\cdots,g_{i1}\cdots g_{ij}=1\cdots g_{im},\cdots,g_d)$。根据定义 4 可知 g_{ij} 位于 $Z(g^1)$ 和 $Z(g^2)$ 的第 $d(m-j)+(d-i)+1$ 位,至少在这个位置上 $Z(g^1)$ 和 $Z(g^2)$ 不相等,即 $Z(g^1)(Z(g^2)$。故函数 $Z(g)$ 为入射。证毕。

引理 3 令 X 和 Y 为有限集,若 X 和 Y 的元素个数相同,即 $|X|=|Y|$,则 $f:X\to Y$ 是一个入射,当且仅当它是一个满射。

证明:(1)若 f 是一个入射,则 $|X|=|f(X)|$,因此 $|f(X)|=|Y|$,由 f 的定义可知 $f(X)\subseteq Y$,又因为 $|Y|$ 是有限的,故 $f(X)=Y$,因此 f 是一个入射推出 f 是一个满射。

(2)若 f 是一个满射,由满射定义可知 $f(X)=Y$,于是 $|X|=|Y|=|f(X)|$。因为 $|X|=|f(X)|$ 和 $|X|$ 是有限的,故 f 是一个入射,因此 f 是一个满射推出 f 是一个入射。

定理 1 函数 $Z(g)$ 是从 G 到 L 上的双射。

证明:集合 G 和 L 是有限集,且 $|G|=|L|=2^{md}$,根据引理 2 可知函数 $Z(g)$ 是入射的,又根据引理 3 可知函数 $Z(g)$ 是满射的。故函数 $Z(g)$ 是双射的。证毕。

定义 5 设 $z=z_{md}\cdots z_1(\{0,\cdots,2^{md}-1\}(z_i(i0,1:\wedge(1\leqslant i\leqslant md))$,$Z(g)$ 的反函数定义为 $Z^{-1}(z)=g(g_1,\cdots,g_d)$,其中 $g_i=\sum_{j=1}^{m}z_{d(m-j)+(d-i)+1}\cdot 2^{j-1}(1\leqslant i\leqslant d)$。

定义 6 Z 地址是一个用竖线分割的整数序列,Z 地址 α 定义为 $\alpha=\alpha_1\mid\cdots\mid\alpha_i\mid\cdots\mid\alpha_l(i\leqslant m)$,其中 $\alpha_i=0,1,\cdots,2^d-1$。

定义 7 设 Z 地址 $\alpha=\alpha_1\mid\cdots\mid\alpha_l,\beta=\beta_1\mid\cdots\mid\beta_l$,若命题 $(\exists i,j)(1\leqslant i\leqslant l)\wedge(1\leqslant j\leqslant i-1)\wedge(\alpha_i<\beta_i)\wedge(\alpha_j=\beta_j)$ 成立,则称 Z 地址 α 小于 Z

地址 β,记作 $\alpha < \beta$。

定义 8 d 维 1 阶 Z 曲线将 d 维超矩形 R 分割成 2^d 个网格,沿着 Z 曲线方向将每个网格记为 $g(i)(i = 0,1,\cdots,2^d - 1)$,Z 地址 k 对应的 Z 空间定义为 $ZS_R(k) = \bigcup_{i=0}^{k} g(i)(k = 0,1,\cdots,2^d - 1)$。

Z 曲线仍然可以继续填充每个网格 $g(k)(k = 0,1,\cdots,2^d - 1)$,Z 地址 k,j 对应的 Z 空间 $ZS_R(k,j) = ZS_R(k) \cup ZS_{g(k)}(j)(k = 0,1,\cdots,2^d - 1)$,其中 $ZS_{g(k)}(j)$ 是 $g(k)$ 的子网格。

例 3:图 4 给出 Z 空间及其 Z 地址示例:图(a)中网格 $g(000_2, 001_2)$ 的 Z 值为 000001_2,该 Z 空间的 Z 地址为 0|0|1;图(b)中网格 $g(000_2,010_2)$ 的 Z 值为 000100_2,该 Z 空间的 Z 地址为 0|1|0;图(c)中网格 $g(0_2, 0_2)$ 的 Z 值为 00_2,Z 空间的 Z 地址为 0;图(d)中网格 $g(10_2, 10_2)$ 的 Z 值为 1100_2,该 Z 空间的 Z 地址为 3|0。

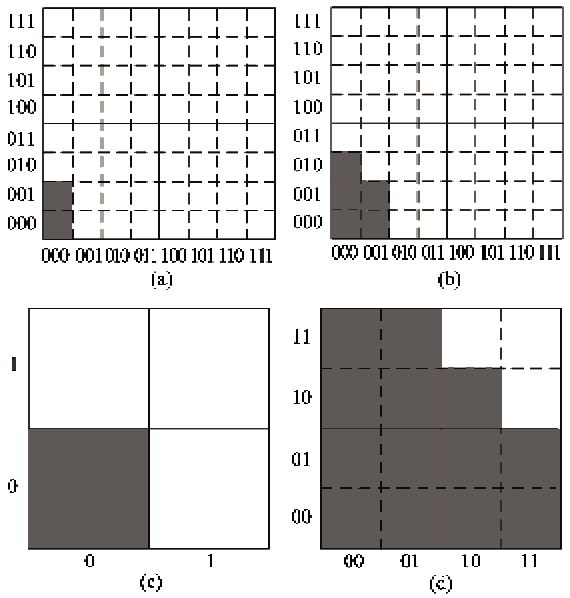

图 4 Z 空间及其 Z 地址

定义 9 若 Z 地址 α 小于 Z 地址 β,则 Z 区域 $ZR(\alpha:\beta] = ZS(\beta) - ZS(\alpha)$。

例 4:图 5 给出 6 个 Z 区域 ZR_1(0|0|−1:0|0|3]、ZR_2(0|0|3:0|2|1]、ZR_3(0|2|1:1|3|3]、ZR_4(1|3|3:2|0|3]、ZR_5(2|0|3:2|3|3] 和 ZR_6(2|3|3:3|3|3]。$ZR_1 \cup ZR_2 \cup ZR_3 \cup ZR_4 \cup ZR_5 \cup ZR_6 = ZR$(0|0|−1:3|3|3],$ZR_i \cap ZR_j = \varnothing 1 \leqslant i,j \leqslant 6 \wedge i \neq j$),故 Z 区域覆盖整个空间,且 Z 区域之间互不相交。

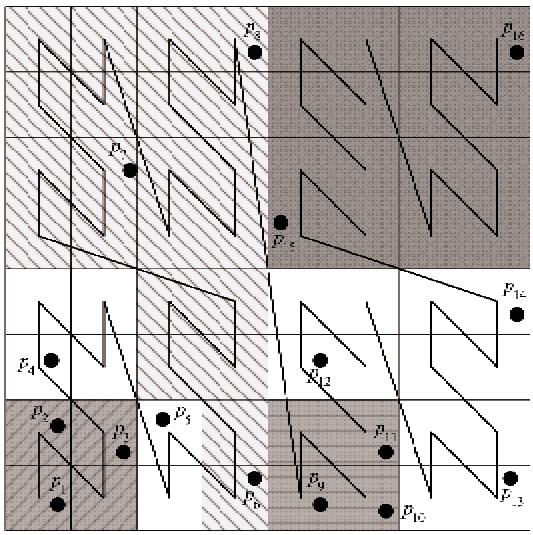

图 5 二维空间中 Z 区域划分

4 B^Z 树索引结构

B^Z 树是一棵平衡多叉树。B^Z 树吸取 B^+ 树和 R 树建树的优点。B^Z 树将点所在网格的 Z 值作为关键字,所有点均存储在叶子结点中,且按关键字进行排序,从而叶子结点对应特定的 Z 区域。

索引结点和叶子结点的结构如图 6 所示。count 表示索引结点的分支数或叶子结点包含的点数;level 表示结点的层数,假定叶子结点的层数为 0,叶子结点的上一层的层数为 1,依次往上递增,根结点的层数最大;pointer 表示指向分支的指针,region 表示该分支对应的 Z 区域;ID 为点的指针,key 为该点所在网格的 Z 值。索引结点的 Z 区域是所有分支 Z 区域的并集,即 $\text{Region} = \text{region}_1 \cup \cdots \cup \text{region}_n$。叶子结点中所有点的 Z 值落在其 Z 区域内,即 $\text{key}_i \in \text{region}\ (1 \leqslant i \leqslant m)$。

Struct INDEXNODE

count	level	Region
region_1	……	region_n
pointer_1	……	pointer_n

Struct LEAFNODE

count	level	region
key_1	……	key_m
ID_1	……	ID_m

图 6 索引结点和叶子结点的结构

例 5:图 5 中点集对应的 B^z 树索引结构如图 7 所示。

4.1 插入操作

首先,选择合适的叶子结点来存放插入点。若叶子结点中尚有空位(点数小于 M),则在该结点中记录点的 ID。若没有空位,则将叶子结点中的点(包括插入点)分成两部分,分别记录在叶子结点和

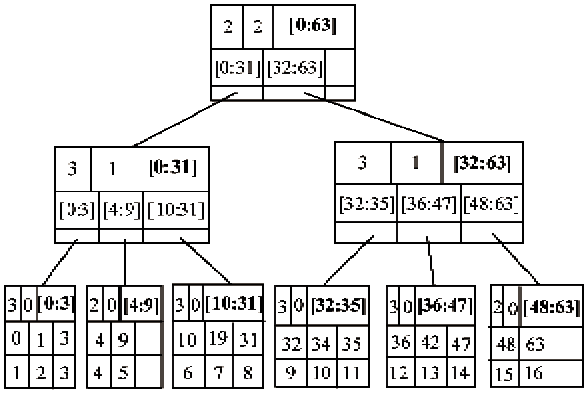

图 7　B^Z 树索引结构

新生成的叶子结点中,然后从叶子结点开始,对树的结构进行调整。若由于该子结点的分裂导致根结点中分支个数超过 M,则需要将根结点分裂成两个结点,再生成一个新结点作为根结点,而原根结点分裂成的两个结点作为其子结点,此时树增加一层。

4.2　分裂操作

假设叶子结点最多存储 M 个点,叶子结点当存储 $M+1$ 个点时发生上溢问题,此时应该将叶子结点分裂成两个叶子结点。若叶子结点的 Z 区域为 $(\alpha,\gamma]$,Z 区域的分裂需通过引入一个 Z 地址 $\beta(\alpha<\beta<\gamma)$。Z 区域被分割成 $(\alpha,\beta]$ 和 $(\beta,\gamma]$ 两个 Z 区域,同时叶子结点被分割为两个叶子结点。

例 6:图 8 给出 Z 区域的分裂操作的示例:$M=5$,当插入点$(001_2,101_2)$后,Z 区域(0|0|3,1|3|1]溢出,被分裂为两个 Z 区域,即 Z_1 区域(0|0|3,1|0|2]和 Z_2 区域(1|0|2,1|3|1]。

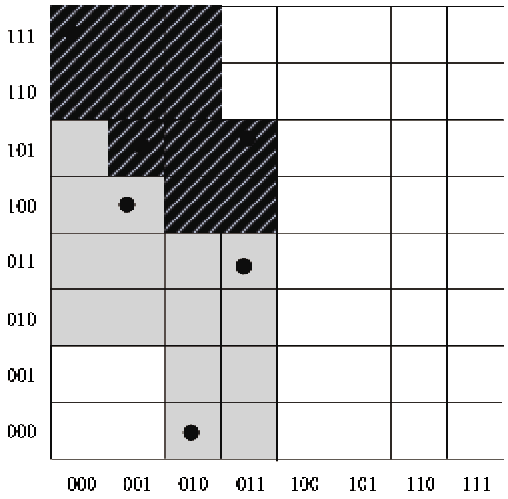

图 8　Z 区域的分裂操作

4.3　删除操作

通过精确查询操作,确定删除点所在叶子结点。将点从该叶子结点中删除,当叶子结点存储的点数小于 $M/2$ 时叶子结点发生下溢问题,此时应该从下至上调整树的结构。如果调整操作过后,根结点只有一个分支,则将根结点删除,让其分支对应的结点成为根结点,此时树减少 1 层。

5　空间范围查询算法

基于 B^Z 树的高维空间范围查询算法 ZRRQ 的基本思想是:若当前访问的结点是叶子结点,则判断其包含的点是否落在查询区域中;若当前访问的结点是索引结点,则判断其 Z 区域是否与查询区域的 Z 区域相交,若相交,则依次判断其分支的 Z 区域是否与查询区域的 Z 区域相交,若相交,递归遍历该分支对应的子树。

算法 ZRRQ(QR, *region*, *root*)

输入:查询区域 QR,QR 对应的 Z 区域 *region*,B^Z 树的根结点指针 *root*;

输出:落在查询区域 QR 中的点集 *result*;

```
begin
    result: = Φ;
    node: = iread(root);
  if ( node. level = 0)
    for i: = 0 to i < node. count do
        point: = dread(node. ID[i]);
      if (inside(point, QR)) then
        point ∈ result;
else
  if (intersect(node. Region, region))
    for i: = 0 to i < node. count do
      if (intersect(node. region[i], region))
        ZRRQ(QR, region, node. pointer_i_);
return result;
end.
```

函数 *iread*(*root*)从索引文件中读取指针 *root* 指向的结点数据。函数 *dread*(*node*.ID[i])从数据文件中读取指针 *node*.ID[i]指向的点数据。函数 *inside*(*point*, QR)判断点 *point* 是否落在查询区域 QR 中。函数 *intersect*(*node*.region[i], *region*)判断索引结点 *node* 的第 i 个分支对应的 Z 区域 *node*.region[i]是否与查询区域对应的 Z 区域 *region* 相交。

例 7:查询区域如图 2 所示,点集的分布如图 5 所示,索引结构如图 7 所示。算法 ZRRQ 的执行过程:从索引文件中取出根结点,判断查询区域的 Z 区域[16:28]与根结点的第一个分支的 Z 区域[0:31]

相交,在该分支上递归调用算法,在递归调用的算法中判断出索引结点的第三个分支的Z区域[10:31]与Z区域[16:28]相交,在该分支上继续递归调用算法,在递归调用的算法中当前结点已经是叶子节点,判断出点 p_7 落在查询范围内。接下来,递归算法依次结束,直至算法结束。

定理 2 算法 ZRRQ 的时间复杂度为 $O(hs)$,空间复杂度为 $O(h)$。

证明:(终止性)算法属于递归算法,递归的层次最多为 B^Z 树的高度 h,最坏的情况是算法深度优先遍历树中所有结点,故算法是可结束的。(正确性)根结点的Z区域是整个数据空间,其分支的Z区域是根结点的Z区域的完全分割,另一方面,查询区域的Z区域完全包含查询区域,故算法是正确的。(复杂度)算法的基本操作是判断叶子结点中点是否落在查询区域中,设查询区域的Z区域与 s 个叶子结点的Z区域相交,每个叶子结点需要 h 次外存访问,故时间复杂度为 $O(hs)$。算法在递归调用时需要分配一定堆栈空间,递归调用的层数最多为 h,故空间复杂度为 $O(h)$。

6 实验结果

为了验证算法的查询效率,实验比较了算法ZRRQ与基于线性扫描、R树、VA文件、NB树的空间范围查询算法之间的性能差异。

实验环境:3.0GHz 奔腾 D 处理器、512M 内存、160G 硬盘、Windows XP、Visual C++ .net 2003。

测试数据:真实数据为旧金山道路网络结点数据(http://cs-people.bu.edu/lifeifei/tpq.html),数据量为174955;人工合成数据为由随机函数产生的满足均匀分布的数据,数据量从10万条到500万条。图9显示真实数据的分布情况。

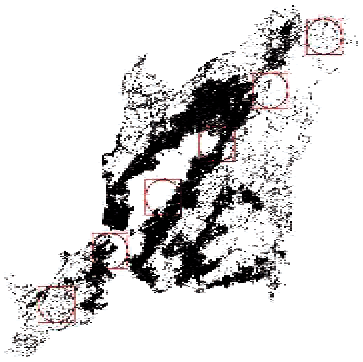

图9 旧金山道路网络结点的分布

6.1 合成数据实验

实验1考查数据量对查询时间的影响,实验中数据均是8维的,数据量从10万条至500万条,查询区域在每维上的长度是该维长度的1/3。因为点是随机分布的,所以理论上来说落在查询区域中的点数是数据文件中总数的 3^8 之一。实验结果如表1所示,其中 d 表示空间维度,M 表示R树中结点的分支数,m 表示Z曲线的阶,b_j 表示VA文件中位串的二进制位数,NODESIZE为 B^Z 树中存储索引结点和叶子结点的页面大小。

表1 数据量与查询时间之间的关系

t(s)	n=10万	20万	60万	80万	100万	200万	300万	500万
Brute-force	0.89	1.85	5.48	7.4	9.23	18.31	31.12	53.7
R-tree	0.06	0.12	0.32	0.43	0.59	1.15	1.59	16.34
VA-file	0.98	1.81	5.43	7.31	9.06	18.18	27.31	48.62
NB-tree	0.23	0.46	1.37	1.82	2.29	4.81	7.18	12.18
ZRRQ	0.01	0.03	0.07	0.1	0.14	0.29	0.43	0.67

$d=8, M=50, m=2, b_j=2$, NODESIZE=512

算法ZRRQ的执行时间随数据量的递增缓慢增长,而线性扫描和VA文件的执行时间增长较快。从表1可知算法ZRRQ的执行时间小于其他算法。

实验2考查维度对查询时间的影响。实验中数据的维度分别为8、16、32、64、128、256,384、512,数据量均是50万条,查询区域在每维上的长度是该维长度的1/3。实验结果如表2所示。

随着维度的增加,维度对查询时间影响比较大。当 $d \geqslant 128$ 时顺序扫描的查询时间小于R树的查询时间。当 $d \geqslant 16$ 时VA文件的查询时间小于顺序扫描的查询时间。算法ZRRQ和NB树的性能没有明显恶化,优于其他三个算法。

6.2 真实数据实验

在旧金山道路网络上6个查询范围如图9所示,圆形区域是NB树的查询区域,正方形是其他算

表 2 维度与查询时间之间关系

t(s)	$d=8$	16	32	64	128	256	384	512
Brute-force	4.6	4.59	4.68	9.75	16.78	25.64	34.75	43.48
R-tree	0.26	0.31	0.51	8.37	18.71	37.75	40.79	95.48
VA-file	4.65	4.51	4.5	4.62	5.18	12.32	7.9	9.17
NB-tree	0.31	0.32	0.39	0.5	0.75	0.79	0.25	0.93
ZRRQ	0.06	0.04	0.06	0.04	0.06	0.06	0.04	0.04

$n=500000, M=50, m=1, b_j=2$, NODESIZE = 512

法的查询区域。查询结果如表 3 所示,其中,n_1 是落在正方形查询区域内的点数,n_2 是落在圆形查询区域内的点数。

表 3 查询区域与查询时间之间关系

t(s)	QR_1	QR_2	QR_3	QR_4	QR_5	QR_6
Brute-force	1.797	2.547	2.297	2.657	1.688	1.641
R-tree	0.109	1.031	0.875	1.094	0.094	0.032
VA-file	1.829	3.172	2.781	3.688	1.875	1.844
NB-tree	0.11	0.789	0.985	1.172	0.078	0.047
ZRRQ	0.125	0.75	1.297	1.125	0.188	0.063
n_1	692	4337	4319	6631	519	190
n_2	512	3627	3425	4705	265	158

$d=2, n=174955, M=50, m=6, b_j=5$, NODESIZE = 512

从表 3 可知在二维空间上算法 ZRRQ 的用时略大于 R 树和 NB 树的用时。

7 结 论

由实验结果可知在二维空间中算法 ZRRQ 的性能与基于 R 树的空间范围查询算法相当,当维度增加时基于 R 树的空间范围查询算法的性能急剧下降,而算法 ZRRQ 的性能仍然高效,且优于基于 VA 文件和基于 NB 树的空间范围查询算法。下一步工作是利用 B^Z 树索引结构解决其他空间查询问题,使之成为一种通用高效的高维空间索引结构。

参考文献

[1] Ashwin R B, Mukesh A, Srinivasan S. Mercury: supporting scalable multi-attribute range queries. *ACM SIGMOD Computer Communication Review*, 2004, 34(4): 353-366

[2] Gisli R H, Hanan S. Index-driven similarity search in metric spaces. *ACM Transactions on Database Systems*, 2003, 28(4): 517-580

[3] Bohm C, Berch S, Keim D A. Searching in high-dimensional spaces: index structures for improving the performance of multimedia databases. *ACM Computer Survey*, 2001, 33(3): 322-373

[4] 郭薇, 郭菁, 胡志勇. 空间数据库索引技术. 上海: 上海交通大学出版社, 2006. 20-23

[5] Weber R, Blott S. An approximation-based data structure for similarity search. ESPRIT project HERMES patent: 9141, 1997

[6] Fonseca M, Joaquim A J. Indexing high dimensional data for content based retrieval in large databases. In: Proceedings of the 8th International Conference on Database Systems for Advanced Applications, Kyoto, Japan, 2003. 223-245

[7] 梁俊杰, 杨泽新, 冯玉才. 大规模高维向量空间的快速范围查询. 小型微型计算机系统, 2007, 28(7): 1225-1229

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

[9] Mokbel M, Aref W. Analysis of multi-dimensional space-filling curves. *GeoInformatica*, 2003, 7(3): 179-209

Depth first and high-dimensional space range query algorithm based on B^Z-tree

Xu Hongbo*, Hao Zhongxiao* **

(* College of Computer Science and Technology, Harbin University of Science and Technology, Harbin 150080)

(** College of Computer Science and Technology, Harbin Institute of Technology, Harbin 150001)

Abstract

In consideration of the fact that the spatial range query algorithms based on linear scan, R-tree, VA-file and NB-tree can achieve better performances in low-dimensional space, but in high-dimensional space their performances suffer a great loss, the paper regards that the reduction of the dimensionality is the key to the spatial range query in high-dimensional space, and uses the Z curve-based grid partition method to reduce the dimensionality, giving an index structure of B^Z-tree and presenting a depth first high-dimensional spatial range query algorithm ZRRQ. It adapts an effective cut-branch strategy, can traverse B^Z-tree fast. The experimental results indicate that its performance is better than that of spatial range query algorithms based on linear scan, R-tree, VA-file and NB-tree.

Key words: high-dimensional spatial, range query, reduction of dimensionality, Z curve, Z-region, B^Z-tree