文章编号: 1001- 2486(2009) 03- 0104- 06

一种适用于点和区间混合型维度数据集的多维索引*

张 , 唐九阳, 戴长华, 肖 卫东
(国防科技大学 信息系统与管理学院, 湖南 长沙 410073)

摘 要: 点和区间混合型维度数据集是空间数据库系统和 GIS 中重要的数据对象。在分析研究 R* 树和 SS 树的基础上, 提出了一种适用于索引点和区间混合型维度数据集的索引结构——PI 树。PI 树利用超球划分数据集的多维空间, 以提高结点存储利用率, 从而降低数据插入时的 I/O 次数。文章给出了 PI 树插入、删除和查询算法的形式化描述。理论分析和实验结果表明, 所提的 PI 树性能上总体优于 R* 树。

关键词: 点和区间混合型维度; 多维索引; PI 树
中图分类号: TP391 文献标识码: A

A Multidimensional Indexing for Data Sets of Point and Interval Dimensions

ZHANG Chong, TANG Jia-yang, DAI Chang-hua, XIAO Wei-dong
(College of Information System and Management, National Univ. of Defense Technology, Changsha 410073, China)

Abstract: Data sets of point and interval dimensions are important in spatial database system and GIS. This paper proposes an index structure PI-tree for the data sets of point and interval dimensions, based on an analysis of R*-tree and SS-tree, of which the former is used to demarcade multidimensional space of data set using hyper-sphere with the aim of improving accessing and reducing the I/O times of inserting data. Furthermore, the algorithm of insertion, deletion and retrieval of PI-tree is also presented. Finally, the analysis and results of experiment show that PI-tree outperforms R*-tree.

Key words: point and interval dimensions; multidimensional indexing; PI-tree

随着空间数据库、地理信息系统和多媒体等领域技术的发展, 多维数据集的应用日益普遍[1]。为了提高多维空间对象的搜索效率, 多维索引机制的研究得到了广泛关注[2]。

传统的多维索引结构, 如 R 树类, 能够很好地解决点类型或者区间类型多维数据集的空间划分、索引构建和查询等问题。然而某些应用中存在这样一种多维数据集: 多维数据的某一些维度上是点类型数据, 而另外一些维度上是区间类型数据, 以往研究中很少考虑这种维度中点类型和区间类型并存的情况。虽然可以利用 R 树[3] (或者其变体, 如 R* 树[4]) 构建最小限定矩形(Minimal Bounding Rectangle, MBR) 包围不规则形状, 再划分子空间建立多维索引, 但这种方法需要保存各个维度的区间范围, 导致结点的每个索引项所占用的空间较大, 从而增加了磁盘 I/O, 并且其插入结点时 CPU 开销也很大。针对上述不足, 本文借鉴 SS 树[5] 中超球的思想, 提出了一种针对点和区间混合型维度数据集的多维索引结构——PI 树, 并详细介绍了 PI 树的结构、相应算法。实验结果表明本文提出的 PI 树建树效率高于 R* 树, 存取数据时 I/O 次数相对减少, 检索效率也相应提高。

1 相关工作

多维索引的历史可以追溯到 20 世纪 70 年代中期。举几种典型的索引结构: R* 树是 R 树的一种变体, 它对 R 树进行了算法优化, 是一种既适用于点类型数据也适用于区间类型数据的多维索引结构。R* 树通过改进了插入和分裂算法并且引入强制重插机制来提高 R 树的性能。SS 树是一种用于多维点

* 收稿日期: 2008- 10- 29
基金项目: 国家自然科学基金资助项目(60172012); 湖南省自然科学基金重点资助项目(03JJY3110)
作者简介: 张 (1982—), 男, 博士生。

数据相似搜索的索引结构。它使用了限定球来划分区域, 提高了最近邻(Nearest Neighbor, NN) 查询的性能。SS 树采用的超球体积很大, 导致超球之间重叠严重, 所以增加了多路搜索的开销[6] 。SR 树[6] 是一种用于高维 NN 查询的多维索引结构。SR 树的特点是将限定球与限定矩形相结合。性能测试表明, SR 树优于SS 树和 R^* 树。

2　问题描述与分析

传统多维索引结构要么针对点类型数据(如图像的特征向量($x_0, x_1, x_2, \cdots, x_n$) , 无论多少维, 每一维上提供的都是一个点数据) , 要么针对区间类型数据(如 GIS 中的三维坐标中描述一个物体需要每一维度提供它的区间范围) 而构建, 很少综合考虑点和区间维度并存的数据集。

考虑一个既有点类型维度又有区间类型维度的地理信息元数据集, 假设为二维的情况(x, y) , x 代表地域的经度维, y 代表时间维。在 x 坐标轴上提供的是区间型数据, 如东经 30° —西经 40°, 在 y 坐标上提供的是点类型数据, 如 2007- 8- 27, 那么这个数据集的直观表示如图 1(a) 。

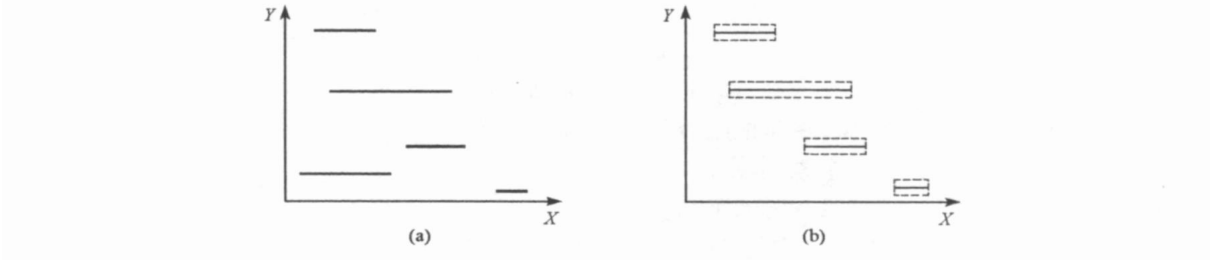

图 1　二维混合型维度

Fig. 1　Hybrid 2-dimensions

由图 1(a) 可见该数据集在二维中的表现形式是线段, 按照 R 树的经典方法, 需要对“ 不规则” 的形状做MBR, 也就是要将 y 坐标上的点数据进行扩展, 如图 1(b) 所示。

扩展之后就可以在这些“叶结点” 上建立“ 中间结点” 的MBR, 从而构建基于 R 树的多维索引(或者其变体) 。由于该方法进行维度扩展时增加了索引项的大小, 不利于索引的存储和检索。

因此, 本文提出采用限定圆的方法, 取每个线段的中点作为圆的圆心, 每个线段的一半长度作为圆的半径, 如图 2(a) 所示。这种空间划分办法只需要存储一个中心点和圆的半径即可。

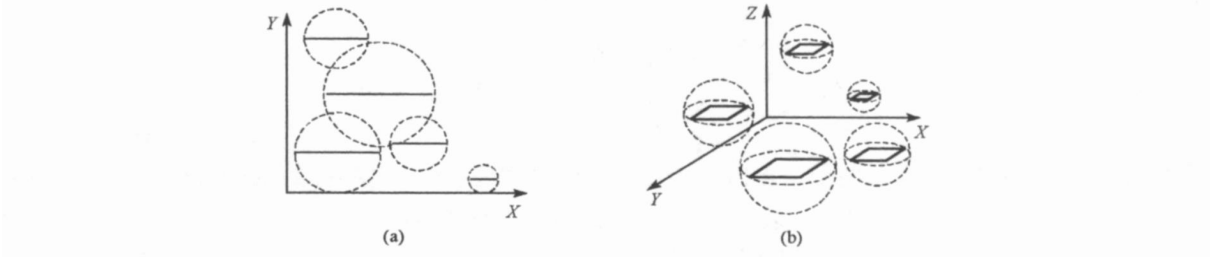

图 2　二维和三维限定圆(球)

Fig. 2　Bounding round(sphere) of 2- dimension and 3- dimension

下面考虑 3 维空间(x, y, z) 的情况, 假设在 x 维度上和 y 维度上提供的是区间型数据, 在 z 维度上是点类型的数据, 那么直观的空间结构是一些与 z 轴垂直的不同的面。利用上述思想, 取各个面在 xy 平面上的投影的中心(x_0, y_0) , 再加上平面本身与 z 轴的交点 z_0, 组成了球的中心点(x_0, y_0, z_0) ; 取(x_0, y_0) 到任意一个顶点的长度作为半径, 则可以做出一个三维限定球, 如图 2(b) 所示。上述限定圆和限定球就是本文要构建索引树的叶结点的索引项。

3 PI(Point and Interval)树结构与相关算法

3.1 PI树结构

本文提出的 PI 树是一棵类似于 B^+ 树的动态平衡树, 在二维中的空间划分如图 3(a) 所示, 树的结构如图 3(b) 所示, 它满足以下特征:

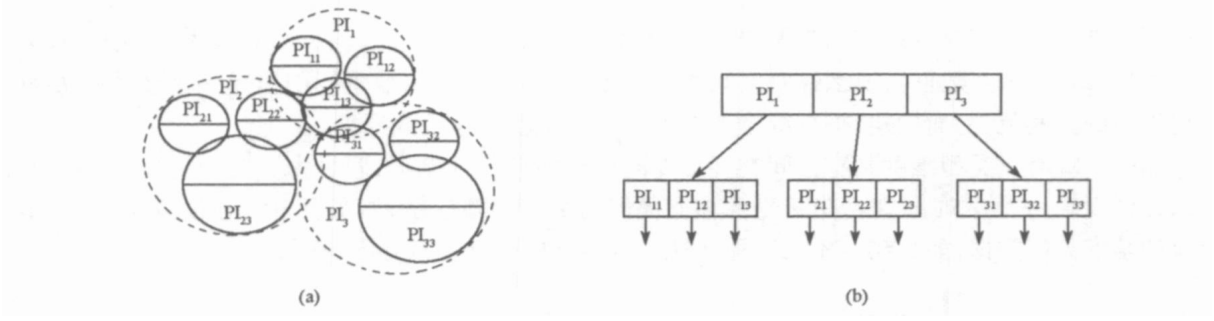

图 3 PI 树结构
Fig. 3 Structure of PI-tree

(1) 根结点至少有两个索引项, 除非根结点是树的唯一结点;

(2) 每个非根结点有 m 到 M 个索引项;

(3) 在上述两个约束条件下, 保持树的平衡。

(4) 叶结点索引项(例如, 图 3(b) 中的 PI_{11}) 的中心坐标 $C(C_1, C_2, C_3, \cdots, C_i, \cdots, C_n)$ 的计算方法如式(1), 设维度为$(X_1, X_2, X_3, \cdots, X_i, \cdots, X_n)$, 其中 X_1 到 X_k 为点数据类型维度, X_{k+1} 到 X_n 为区间数据类型维度, 且 X_{k+1} 到 X_n 的区间维度组成一个超立方体:

$$\begin{cases} C_i = X_i, & 1 \leqslant i \leqslant k \\ C_i = \dfrac{X'_i + X''_i}{2}, & k+1 \leqslant i \leqslant n \end{cases} \tag{1}$$

其中, X'_i 和 X''_i 分别代表第 i 维区间的起始点坐标和终点坐标。

(5) 内部结点索引项(如图 3(b) 中的 PI_1) 的中心坐标 $C(C_1, C_2, C_3, \cdots, C_i, \cdots, C_n)$ 的计算方法如下:

$$C_i = \frac{\sum_{k=1}^{m}(D_k.C_i \times D_k.w)}{\sum_{k=1}^{m} D_k.w} \quad (1 \leqslant i \leqslant n) \tag{2}$$

其中, k 是对内部结点索引项的子结点$(D_1, D_2, \cdots, D_m)$ 的索引, i 是对维度的索引, $D_k.C_i$ 表示子结点 D_k 的中心坐标的第 i 个分量, $D_k.w$ 表示子结点 D_k 的所有数据项的数量。

(6) 叶结点索引项的最小限定超球的半径 r 的计算方法如式(3), 设维度为$(X_1, X_2, X_3, \cdots, X_i, \cdots, X_n)$, 其中 X_1 到 X_k 为点类型数据维度, X_{k+1} 到 X_n 为区间类型数据维度, 且 X_{k+1} 到 X_n 的区间组成一个超立方体 R, 设叶结点索引项的中心坐标为 $C(C_1, C_2, C_3, \cdots, C_i, \cdots, C_n)$:

$$r = MAXDICT(C, R) \tag{3}$$

其中函数 $MAXDICT(C, R)$ 计算的是点 p 到超立方体 R 的最大距离:

$$MAXDICT(p, R) \equiv \max_{q \in R}(||p - q||) \tag{4}$$

其中, q 是超立方体 R 的顶点。

(7) 内部结点索引项的最小限定超球的半径 r 的计算方法如下:

$$r = \max_{1 \leqslant k \leqslant m}(||C - D_k.C|| + D_k.r) \tag{5}$$

其中, k 是对内部结点索引项对应的子结点$(D_1, D_2, \cdots, D_m)$ 的索引, C 是内部结点索引项的中心坐标,

$D_k.C$ 表示子结点 D_k 的中心坐标, $D_k.r$ 表示子结点 D_k 的最小限定超球的半径。

3.2 PI 树的插入操作

PI 树的插入操作涉及选择最优叶结点, 以及当结点溢出时的强制重插算法和结点分裂算法。

(1) 选择最优叶结点

在 R^* 树的叶结点插入优化准则中, 将目录 MBR 的面积以及目录 MBR 间的重叠面积作为选择插入叶结点的根据。SS 树选择最优叶结点的策略是选择与新索引项的中心距离最短的叶结点。在文献[7]的基础上, 本文提出以下选优算法:

①Set N = Root;

②IF N 是叶子结点, THEN Return N;

ELSE

遍历 N 的索引项 $I_i(0 \leqslant i \leqslant p)$, p 为 N 的实际索引项数量, 如果插入新增数据后不增加某索引项指向的结点的覆盖半径, 则选择这个索引项; 如果这样的索引项不唯一, 选择中心距新增数据最近的结点对应的索引项; 如果对所有的索引项指向的结点都会增加覆盖半径, 则选择覆盖半径增加最小的结点对应的索引项;

③Set N = ②中选定索引项指向的结点, 重复执行②。

(2) 强制重插

强制重插主要使用在分裂结点不是根结点, 且在插入新增数据过程中, 对分裂结点所在的层而言是第一次发生溢出时才调用, 主要算法是:

①对分裂结点 N 的 $M+1$ 项(M 为 N 的最大索引项数量), 计算它们的中心到 N 的最小限定超球中心的距离;

②按距离降序排列 N 的 $M+1$ 项;

③选取前 ρM 个索引项(ρ 一般取 30%) 重新插入到其他同级结点中。

(3) 结点分裂

R^* 树的结点分裂算法比较复杂, 其中又涉及面积重叠的计算, 所以在 PI 树中, 借鉴 SS 树, 本文提出以下结点分裂算法:

①遍历分裂结点 N 的 $M+1$ 项(M 为 N 的最大索引项数量), 计算任意两个索引项覆盖中心的距离;

②选择两个距离最大的索引项的覆盖中心作为两个新结点的中心;

③将剩余点分配到距离最近的中心的结点中。

3.3 PI 树的查询算法

设给定超球 $S_q(C_q, r_q)$, 其中 C_q 是超球球心, r_q 是半径。要查询其中所包含的空间对象, 在查询时首先从 PI 树的根结点开始。

①IF 根结点中直接包含的是数据索引项, THEN

逐个检查每个数据索引项 $S_0(C_0, r_0)$ 是否满足 $||C_q - C_0|| \leqslant r_q - r_0 (r_0 \leqslant r_q)$, 如果满足, 则此数据就是检索的对象之一。

②ELSE IF 根结点中是内部结点的索引项 $S_I(C_I, r_I)$, THEN

逐个检查各个内部结点索引项是否满足 $||C_q - C_I|| < r_q + r_I$, 如果满足, 则此索引项中的对象就有可能包含在 S_q 中, 需要沿此内部结点索引项向下继续搜索; 如果不满足, 则就不必沿此内部结点索引项向下搜索。

3.4 PI 树的删除算法

PI 树的删除算法如下:

①从根搜索到待删除数据 P 所在的叶结点 N;

②从 N 中删除 P, N 的索引项数量= N 的索引项数量- 1;

③IF N 的索引项数量< m, THEN

删除 N, 并将 N 的剩余索引项重新插入 PI 树中

重新插入可能导致其他结点的分裂,整个过程进行了对 PI 树的一次重整。

4 性能评价

本文通过实验验证 PI 树的有效性,对算法的查询效果和搜索效率进行实验分析。仿真实验的软硬件环境为 Intel Pentium© 4, CPU 主频 3.06GHz, 1GB 内存, Windows XP SP2 操作系统。

4.1 性能分析与比较

R^* 树与本文提出的 PI 树都可以对点和区间混合型维度数据集建立索引,因此本文首先在理论上从数据项插入性能和结点存储量两个方面比较 PI 树和 R^* 树的性能。

在插入新数据项时, PI 树与 R^* 树的主要差别在于选取最佳叶结点时, PI 树只需循环遍历所有叶结点一次,而 R^* 树则要两层嵌套循环遍历结点,因此,对于 n 个叶结点, PI 树插入新数据项的时间复杂度为 $O(n)$,而 R^* 树插入新数据项的时间复杂度为 $O(n^2)$。

下面再考虑每个结点的存储量,在保存索引文件时,对于 PI 树的边界超球只需保存一个 N 维的点和半径长度即可,而对于 R^* 树的边界超矩形,要保存 N 维的起始坐标和终止坐标,可见利用边界超球来替代边界超矩形,可以降低每个数据项的存储空间,从而使得每个结点中索引项的数量得到增加。

4.2 实验结果及分析

全部实验程序在 Visual C++ 6.0 中实现,实验中树的结点的大小设置为 4096B,这与操作系统的磁盘块大小相一致,实验中的数据来自地理信息应用中描述资源的元数据文件。共采集了 100 000 个元数据样本,经过提取、归一化处理后产生了 100 000 个在[0, 1) 区间上点和区间混合型维度的多维数据。为了对比 R^* 树与 PI 树的性能,本文采用叶结点索引项数量、插入和搜索时 CPU 耗时以及插入和搜索时磁盘 I/O 次数作为性能评价指标。

为了快速比较两种结构的叶结点索引项数量,程序简化了结点的数据结构,对比如表 4 所示。

表 1 R^* 树与 PI 树叶节点索引项数量比较

Fig. 1 Comparison of R^*-tree and PI-tree on leaf node indexes

维数量	R^* 树	PI 树
6 维	94	179
7 维	81	158
8 维	71	142
9 维	64	129
10 维	57	118
16 维	36	78
18 维	33	70
24 维	25	54
30 维	20	44

由表 1 可见,在每种情况下, PI 树叶结点的索引项的数量大约是 R^* 树的 2 倍。这样,在数据项的数量是一定的条件下,每次 PI 树交换至主存中的数据项数量要多于 R^* 树,降低了磁盘 I/O 的次数,并且每个结点的索引项数量大小决定了整体树的高度, PI 树的高度要比 R^* 树的高度低,这样对提高查找效率是有利的。

分别在大小不同(从 10 000~ 100 000)的数据集上测试插入一个新结点时 PI 树和 R^* 树平均消耗的 CPU 时间和磁盘访问的次数,实验中,数据维数是 16 维,且前 6 维是点类型数据,后 10 维是区间类型数据,实验结果如图 4 所示。从图 4(a) 可以看出当插入一个新结点时, PI 树所耗的 CPU 时间明显比 R^* 树少,这和 4.1 节的性能分析是一致的;在磁盘访问方面,如图 4(b) 所示,在数据集数量小于 83 000 的情况下, PI 树的磁盘访问次数要比 R^* 树低,这与 4.1 节中的性能分析是相一致的,但在数据集增大时, PI 树的磁盘访问次数要多于 R^* 树,这是因为,在叶结点选优时, R^* 树采取的策略是选择与其他结点面积重叠之和最小的叶结点,它避免了过多的结点重叠,减少了多路搜索,即 R^* 树以消耗更多 CPU 时间的代价换取了较少的磁盘 I/O 次数。

本文又分别在相同的数据集上做了搜索性能的实验,实验结果如图 5 所示。从图 5(a) 可以看出 PI 树在搜索耗时上要优于 R^* 树;图 5(b) 反映出 PI 树搜索时的磁盘访问次数也比 R^* 树的磁盘访问次数要少。这是因为超球的定位特性 ——只需要圆心坐标和半径即可定位,这在比较范围时要优于 R^* 树的逐维比较算法,提高了查询性能。整体来讲,对于点和区间混合类型的多维数据集的索引达到了预期的

目标。

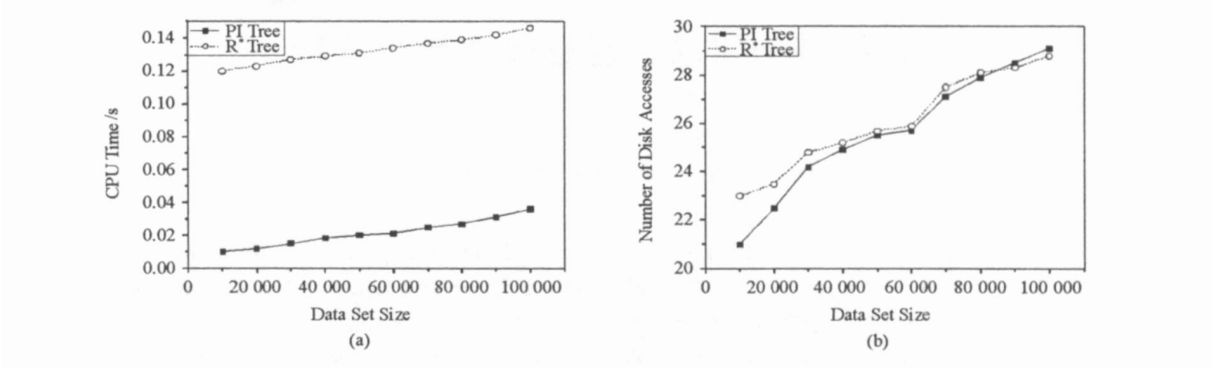

图 4　插入数据时 PI 树与 R* 树的对比

Fig. 4　Comparison of PI-tree and R*-tree on inserting data

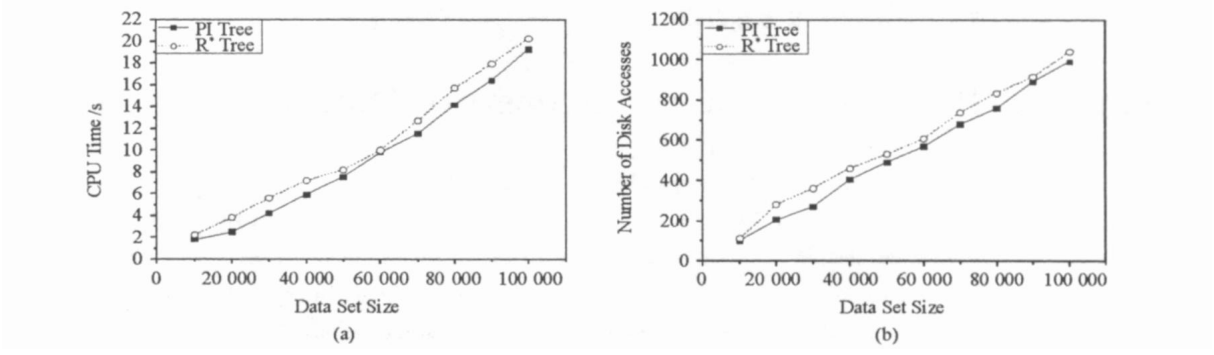

图 5　检索时 PI 树与 R* 树的对比

Fig. 5　Comparison of PI-tree and R*-tree on retrieving

5　结 论

本文针对点和区间混合型维度多维数据集, 提出了多维索引结构 PI 树, 并给出了相应的构建、查找和删除算法, 与经典的 R* 树索引做了性能对比实验, 结果表明本文提出的 PI 树性能要优于 R* 树。下一步中将会进一步优化该算法, 使之能对大规模数据也具有较好的性能并在实际中进行应用。

参 考 文 献:

[1]　Gaede V, Gunther O. Multidimensional Access Methods[J]. ACM Computing Surveys, 1998, 30(2): 170~ 231.

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

[3]　Guttman A. R-tree: A Dynamic Index Structure for Spatial Searching[C]// Proceedings of the ACM SIGMOD International Conference on the Management of Data, 1984: 47– 54.

[4]　Beckmann N, Kriegel H, Schneider R, et al. The R*-tree: An Efficient and Robust Access Methods for Points and Rectangles[C]// Proceedings of the ACM SIGMOD International Conference on the Management of Data, 1990: 322– 331.

[5]　White D A, Jain R. Similarity Indexing with the SS-tree[C]// Proceedings of the 12th IEEE International Conference on Data Engineering, 1996: 516 – 523.

[6]　Katayama N, Satoh S. The SR-tree: An Index Structure for High-dimensional Nearest Neighbor Queries[C]// Proceedings of the ACM SIGMOD International Conference on the Management of Data, 1997: 369– 380.

[7]　杨建武, 陈晓鸥. 半结构化数据相似搜索的索引技术研究[J]. 计算机学报, 2002, 25(11): 1219– 1226.

[8]　陈冬霞, 吉根林, 方昭辉. 基于内容的图像检索中 SS– 树索引的 Java 实现[J]. 南京师范大学学报(工程技术版), 2005, 5(4): 53– 56, 81.

[9]　张明波, 陆锋, 申排伟, 等. R 树家族的演变和发展[J]. 计算机学报, 2005, 28(3): 290– 300.

[10]　古毅, 吴中福, 魏丽, 等. 高维空间数据索引结构分析研究[J]. 计算机科学, 2006, 33(5): 142– 145.