空间索引的研究*)

Research on Spatial Indexes

陈 菲 秦小麟

(南京航空航天大学计算机科学与工程系 南京210016)

Abstract Search operations in database require special support at physical level in conventional databse as well as in spatial database. However, index mechnism in conventional databases cannot efficiently access spatial data because of their specialty. This apaer gives a survey and analysis of spatial access methods from various points of view. Finally, the spatial access method used in SADBS, a spatial anaysis DBMS developed by us, is introduced.

Keywords Spatial data, Spatial access mechanism, Spatial Index, R-tree

1. 引言

空间数据是指带有空间坐标的数据,它不仅能表示实体本身的空间位置及形态,而且还包含实体属性和空间关系的信息。在地理信息系统(GIS)、计算机辅助设计(CAD)、多媒体信息系统(MMIS)等诸多应用领域都涉及到空间数据的存取和管理。特别是这两年来备受关注的数字地球,其绝大部分功能将以空间数据为基础信息。

在这些应用中,海量、复杂的空间数据使得索引机制显得尤为重要。例如对于一个普通的空间查询不仅需要多次重复耗时、复杂的几何操作,还需要多次读取磁盘,如果没有一个高效的空间索引支持查询,系统的效率会受到严重影响。

索引是用来提供快速、有选择性的存取数据库的一种机制,它相当于一个映射机构,将属性的值转换为相应的记录的地址或地址集。对于空间数据,其存取主要依赖于空间对象之间的位置关系,而不是象传统数据那样,依赖于某个(些)属性的值。这就使得空间索引机制与传统索引机制不同,并且很多对传统数据有效的索引机制却不能很有效地应用于空间数据。

本文对空间索引所取得的成就与问题进行了分析与回顾,揭示了这些问题所产生的根源。并介绍了我们所研究的空间分析 DBMS-SADBS 中的索引方法。

2. 空间数据的特点

如果要建立高效的空间索引,就要了解空间数据与传统数据的区别。

(1)不可排序性:前面已经说过空间数据主要依赖于空间位置进行存取,所以空间索引是根据空间数据间的邻近性来建立的。然而对于多维的空间数据却无法建立一个可以反映其邻近性的排序,即无法找到一个从多维空间到一维空间的映射 f,使得多维空间的任意两点 V_1、V_2,$f(V_1)$,$f(V_2)$相邻当且仅当 V_1和 V_2在空间相邻。

(2)相关性:一个空间对象可能是点(集)、线(集)或区域。除了单独的一个点以外,一个 n 维空间对象至少在其中的一维上覆盖一个区间,而且一个空间数据库的数据量通常是很庞大的,这使得空间对象间极有可能相互重叠,即空间数据间的相关性很大,建立在空间位置关系上的空间索引的效率也随之降低。

(3)数据复杂性:在现实世界中,空间对象的大小、边界的几何构造可以是多种多样的,空间对象的分布也是不均匀的。所以空间对象在计算机中的表示也很复杂,相应的空间位置的判断也较复杂。很多索引机制先用一种较为规则、简单的几何图形去近似空间对象的几何构造,然后再根据这些几何图形的空间位置关系建立索引。

空间数据的这些特征使得迄今为止还无法找到一种高效的空间索引机制,虽然人们不断提出有所改进的空间索引机制,却无法打破多维空间所带来的束缚。

鉴于空间数据的特性,一个理想的空间存取机制必须满足的最基本的要求有:

(1)可扩展性:存取机制的效率不会随数据量的变大而降低,更重要的是不会随着数据的维数增大而降低。对于当今存在的很多存取机制其可扩展性都不是很好,特别是随着空间数据维数的增加其效率大大降低。

(2)时间复杂度。空间存取方法的查询复杂度应低于线形复杂度。

(3)空间复杂度。索引所占空间应小于数据所占空间。

*)本文受国家自然科学基金资助(批准号49971063、69973022)。

(4)与输入的数据和插入的顺序无关。无论数据是如何分布的以及插入的顺序是如何的，存取机制应保持其效率。当数据在不同维上分布不同时，这点就显得尤为重要。

3 空间存取方法

多维存取方法包括点存取方法(point access methods)又称 PAMS 和空间存取方法(spatial access methods)又称 SAMS。前者是对空间的点进行存取的方法，后者是对包括点(集)、线(集)或区域等空间对象进行存取的方法。本文将以后者即空间存取方法为主进行讨论。

在已有的空间存取方法中，基本上可以从以下几个方面分类讨论索引机制：(1)对空间对象近似的方法；(2)是否进行转化；(3)对空间对象还是对空间进行分割；(4)是否考虑辅存；(5)静态还是动态。

3.1 对空间对象几何构造近似的方法

前面已经提过空间对象的几何构造是复杂多样的，因此为了简化问题，在建立索引之前要将空间对象的几何构造近似成较为简单的几何图形。近似的方法可分为保守、进步和一般三种[1]。若原空间对象的边界完全包含于近似形之中，则该方法是保守的。如果所有被近似的几何构造包含的点都是属于原空间对象的，则称该方法是进步的。而一般近似法主要是利用诸如减少原空间对象边界点的个数等方法以简化空间对象边界的几何构造，其近似形和空间对象之间不存在包含或被包含的拓扑关系(如图1)。

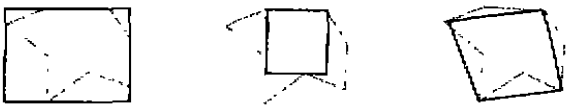

图1 近似的方法

现在绝大多数的存取方法是建立在保守近似基础之上的。根据近似的几何构造不同，又可分为矩形、球形、还有网格，网格近似方法用于下面将要介绍的转化法中，而在其他情况下最常用的还是矩形，又称为最小边界矩形 MBR(minimum bounding rectangles)。它是一个 n 维矩形，表示为$(I_0, I_1, \cdots, I_{n-1})$，其中 I_i 是空间对象在该维上所占区间。而利用球性进行近似时，其边界扩大程度要大于矩形，增加了空间对象之间相互重叠的可能性，从而降低了索引的性能。

3.2 是否转化

在对空间对象建立索引时有两种主要的方法，一种是使用转化法，即将 d-维空间的空间对象转化为更高维或一维空间的空间对象进行索引；另一种则是直接对空间对象进行索引。

1)转化到更高维空间：将 d-维空间对象表示成更高维空间的点，然后再利用点存取方法对其进行存取。例如，二维空间的矩形可以用其在一条对角线上的两个顶点来表示，其坐标形式为(x_1, y_1, x_2, y_2)，而这可以看作四维空间的一个点。这种方法的优点是可以直接利用空间点的存取方法实现对空间对象的存取，但这种方法实际上是将一个区域映射到一个点，在这个过程中可能会丢失原有数据所包含的一些信息，如不能保持原有空间对象的邻近性。而且区域查询、相交等一些操作也必须通过相应的转化来实现，增加了空间操作的实现难度。

2)转化到一维空间：这也是后面将要介绍的划分空间法的一种。将数据空间划分成大小相同的网格，再根据一定的方法将这些网格编码，使得每一个网格的标号是唯一识别的，并在一定程度上保持空间邻近性，即相邻的网格的标号也相邻。一个空间对象由一组网格组成(如图2)。常用的网络编码方法有行排序、Z 排序和 Hilbert 值排序(如图3)。其中 Hilbert 值排序最能反映空间邻近性。

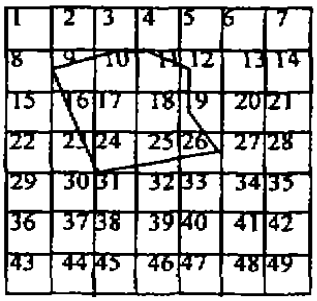

图2 该空间对象可表示为由9,10,11,12,16,17,18,19,23,24,25,26号网格组成

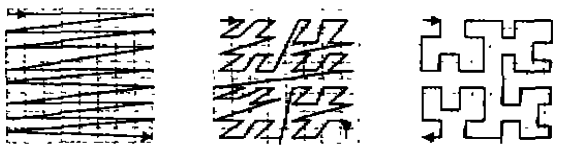

图3 网格的编码

这种转化法将多维空间的网格映射到一维空间，再用 B 树等一维索引机制对这些网格进行存取，效率非常高。其缺点由于空间对象是由一组网格来近似的，其近似的精度取决于网格划分的密度，网格划分越密，近似程度越高，但同时一个空间对象所包含的网格也越多，对空间对象进行存取、更新的代价也就越高，这是一对矛盾。同时这种转化法也是后面要讨论的划分空间法的一种，因此也具有划分空间法一样的问题，即

删除和插入操作较复杂。

3.3 划分空间或划分空间对象

3.3.1 划分空间 空间索引机制研究的一个关键问题在于如何划分数据空间，以及索引如何根据划分方法将数据组织起来。一种方法是对数据所在的空间进行划分，由于子空间两两互不相交，所以同一个空间对象有可能被分割到好几个子空间，被存放在不同的页面中，这种数据冗余使得空间对象在插入和删除时操作复杂，不但要增加查询的时间，还有可能增加索引节点溢出的频率。

划分空间法的优点在于，因为子空间两两互不相交，所以对于点查询，索引中只存在一条搜索路径，但对于区域查询仍然存在多条路径。

利用划分空间法的存取机制主要有 K-D 树[2]以及 K-D-B 树[3]，R^+ 树[4]，CELL 树[5]等。

3.3.2 划分空间对象 另一种方法是划分空间对象，由 Guttman 于1984年提出的 R 树[6]是采用该方法的代表。R 树家族在空间存取机制中占有非常重要的地位。一些商用系统如 Informix 都是采用 R 树来作为空间存取机制的。

R 树的结构类似于 B^+ 树的平衡树。其叶子节点包含多个形式为(OID，Rectangle)的实体，其中 OID 指向数据库中的空间对象，Rectangle 是该空间对象的最小边界矩形；其非叶子节点包含多个形式为(child-point，Rectangle)的实体，其中 child-point 指向其孩子节点，而 Rectangle 是包含其孩子节点中所有 Rectangle 的最小边界矩形。R 树的所有叶子都在同一层，且所有非树根的节点有 m 到 M($2 \leqslant m \leqslant M$)个孩子，非叶子树根至少含有2个孩子。

可见这种方法中一个空间对象完全包含于一个子空间中，因此不可避免各个子空间有可能产生重叠。无论对于点查询还是对于区域查询都可能存在一条以上的搜索路径，遍历多个子树，使得查询效率降低。所以 R 树优化的一个重要途径就是尽可能减少各子空间之间的重叠，即增加各子空间内的数据聚集性，减少各子空间间的数据相关性。对于 R 树这样一种动态建立的树，索引的性能主要决定于节点插入算法、特别是插入后有溢出时的节点分裂算法。插入和分裂算法的不同是 R 树家族中各种索引机制之间的主要区别。

要提高 R 树的查询效率主要从两方面着手，一方面尽可能地减少所需要搜索的路径；一方面尽可能降低索引树的高度。当插入一个新的空间对象时，当一个节点发生溢出要分裂时，该如何做才能保证以上两点呢？这实际上涉及到两个问题：一是优化的标准；一是局部优化与整体优化之间的关系。

R*树[7]一文中讨论了四个优化的标准以及它们之间的关系。(1)中间节点的 MBR 所覆盖区域的面积应最小，即插入一个空间对象时，应将它插入到 MBR 覆盖面积扩大程度最小的节点中，这样一来，被中间节点覆盖到的"无用区域"(没有空间对象的区域)就会减少，因此可以在树的较高层决定从哪条路径进行搜索。(2)中间节点 MBR 所覆盖的区域间重叠最小，这样同样可以减少所需搜索的路径。(3)中间节点所覆盖的区域的几何形状尽量接近正方形。(4)增大节点的存贮利用率，以降低索引树的高度。

所谓局部优化即仅仅考虑要分裂节点局部的数据组织来优化索引；而整体优化则从总体考虑，使得索引对整个数据空间的数据组织达到最优。显然局部最优不一定能保证全局最优。而一般的节点分裂算法都是局部优化。因此要优化 R 树，就应采取一定措施扩大节点分裂时索引优化的范围。

R*采取了一种"强制重新插入"[7]的技术。当节点发生溢出时，保留节点中最相邻的一部分 MBR，而其余的则依据插入算法从新插入。"强制重新插入"技术从一定程度上扩大了重新组织数据时所应考虑的数据空间的范围，因此 R 树的性能得到了很大的改善。文[7]中的实验也证明了此点。

但 R*树构造起来太复杂，有可能花费更多的 CPU 时间(虽然文[7]中实验证明并没有花费更多的 I/O 时间)。Hilbert R 树[8]利用 Hilbert 值建立空间对象在一维空间上的一种排序，其索引构造过程类似于 B^+ 树，非常简单。更重要的是，当节点发生溢出时，Hilbert R 树采取了一种 S—S+1的分裂算法，即将发生溢出的节点中的 MBR 与该节点的 S-1个兄弟中的 MBR 放在一起考虑数据的重新组织。只有当这 S 个节点都无法容纳多余的实体时，才发生节点分裂，并将 S 个节点中的实体平均划分到这 S+1个节点中。Hilbert R 树构造过程简单，组织数据时利用 Hilbert 值进行全局性排序、S—S+1的分裂算法达到了一种较为全面的优化。

R 树与划分空间法比起来减少了空间对象存储冗余，使得插入和删除较简单。而且它的另一个重要的特点就是对辅存的考虑。对支持辅存的索引结构将在后面讨论。

3.4 是否考虑辅存

对于早期的空间索引结构，如 K-D 树都是二叉树，这种结构不支持辅存，因此不适用于大型的空间数据库。不过这些索引机制的基本思想在后来的索引机制中仍得到了继承，不同的是索引结构由二叉树变为了支持辅存的多元树。以 R-树为例，R 树的每一个节点对应于一个物理页面，M 是该页可存放实体的最大个数，m 限定了最小存储利用率。对于支持辅存的结构可以做到逻辑上相邻的空间对象物理上也相邻，这样可以提高查询效率。

3.5 静态或动态

索引的要求是要支持动态的更新以使得索引和数据库保持一致。前面所讨论的那些索引机制都是动态更新的。但是这些索引会随着更新操作的增多，性能下降，甚至会下降到无法容忍的地步。对于 R 树家族的索引机制，插入新的空间对象会导致节点的分裂，节点间的相关性就有可能增加，查询效率就会降低。对于 K-D 树等非平衡的索引结构，当插入的新的空间对象都处于某个邻近的区域，会使得树的某个枝的深度远远大于别的枝，查询的时间复杂度退化为线形。

对于那些更新操作较少的空间数据数据库而言，可以考虑先将这些数据进行一个预处理，然后对这一批数据一次建立索引，而不是象动态索引一样逐个插入。这种算法称为压缩(packing)算法。R 树的压缩算法一般根据一定规则将所有空间对象的最小边界矩形(假设有 r 个)进行排序，并依次将它们分成[r/n]组(每个节点可放 n 个)，每组矩形存放于一个节点。计算出包含该组所有矩形的最小边界矩形，再对这些矩形重复上述过程，直至生成整棵树。

压缩算法所用的排序方法有：文[9]中提出根据矩形中心的 x 轴坐标的大小排序，文[10]中提出根据矩形中心的 Hilbert 值排序，文[11]中依次根据各个坐标轴的坐标进行排序。

压缩算法可以提高空间利用率，并且由于索引是利用全局性的优化建立起来的，所以查询的效率也很高。

4 SADBS 的索引方法

我们所研制的 SADBS 是基于 Realms[15,16]和主存、具有空间分析能力的 DBMS。Realms 中的空间对象不是建立在欧几里德空间，而是建立在离散网格上的。Realms 概念约束了构成空间对象的点和线的取值和顺序关系，保证各种空间分析算法的实现。Realms 的离散性极大地简化了 Hilbert 值的计算，而且 Hilbert R 树构造和维护都比较简单，查询和更新的效率都很高，所以在该系统中采用了 Hilbert R 树为主体的索引方法。

Hilbert R 树的缺点是，在节点分裂时仅仅考虑空间对象中心的 Hilbert 值，所以空间对象较大时，索引的性能就会下降。为了克服这个缺点，本系统在建立和维护索引时，将较大的空间对象提升到树中较高层，使得索引树同一层中所有的 MBR 大小相当，同时中间节点所覆盖区域间的重叠也大大降低。

结论 对于现有的各种各样的空间存取机制，很难说哪一种明显优于其他的存取机制。这是由于空间数据本身不存在一种排序可以完全反应其空间邻近性，因此有很多因素会影响到一个空间存取机制的效率。在 SADBS 系统原有的存储管理机制基础上，结合 Realms 的特点建立上述的索引机制，有效保障了该系统强大的空间分析功能的效率。

对于当代大型数据库而言，并行控制是必不可少的功能。R-Link 树[12]是一种支持并行存取数据的索引机制，近几年有很多人开始研究建立在各种空间索引基础上的并行存取机制[13,14]。在这个方面仍需进一步的研究工作。

参 考 文 献

1 Brinkhoff T, Kriegel H P, Schneider R. Comparison of Approximations of Complex Objects Used for Approximation-based Query Processing in Spatial Database System. In: Proc. of 9th Int'l Conf. on Data Engineering, 1993

2 Bentley J L. Multidimension Binary Search Trees Used for Associative Searching. Comm ACM, 1975, 19(9): 509～517

3 Robinson J T. The K-D-B Tree: A Search Structure for Large Multidimensional Dynamic Indexes. In: Proc. ACM Conf. on Management of Data, 1981. 10～18

4 Sellis T, Roussopoulos N, Faloursos C. The R+-Tree: A Dynamic Index for Multidimensional Objects. In: Proc. 13th Int'l Conf. on VLDB, 1987. 507～518

5 Günther O. The Cell Tree: An Object-Oriented Index Structure for Geometric Databases. In: Proc. of the IEEE Int'l Conf. on Data Engineering, 1989

6 Guttman A. R-trees: A Dynamic Index Structure for Spatial Searching. In: Proc. ACM SIGMOD Int'l Conf. on Management of Data, 1984. 47～57

7 Beckmann N., Kriegel H. -P., Schneider R, Seeger B. The R*-Tree: An Efficient and Robust Access Method for Points and Rectangles. In: Proc. ACM Conf. on Management of Data, 1990. 322～331

8 Kamel I, Faloutos C. Hilbert R-tree: An Improved R-tree Using Fractals. In: Proc. 20th Int. Conf. on VLDB, 1994. 500～509

9 Rossopolous N, Leifker D. Direct Spatial Search on Pictorial Database Using Packed R-Trees. In: Proc. SIGMOD, 1985. 17～31

10 Kamel I, Faloutos C. On Packing R-trees. In: Proc. 2nd Int. Conf. on Information and Knowledge Management (CIKM), 1993. 490～499

11 Leutenegger S T, Edgiyton J M, Lopez M A. STR: A Simple and Efficient Algorithm for R-tree Packing. In: Proc. of the IEEE Int'l Conf. on Data Engineering, 1997

12 Kornacker M, Banks D. High-Concurrency Locking in R-trees. In: Proc. 21st Int'l Conf. VLDB, 1995. 134～145

13 Chaknabarti K, Mehrotra S. Dynamic Granular Locking Approach to Phantom Protection in R-trees. In: Proc. of the IEEE Int'l. Conf. on Data Engineering, 1998

14 Chaknabarti K, Mehrotra S. Efficient Concurrency Control in Multidimensional Access Methods: [Technical Report TR-MARS-97-12]. Department of Computer Science, University of Illinois, October 1998

15 Güting R H, Schneider M. Realms: A Foundation for Spatial Data Types in Database System. In: Proc. 3rd Intl. Symposium on Large Spatial Databases, 1993. 14～35

16 秦小麟. 空间分析数据库的研究方法及技术. 中国图像图形学报, 2000, 5(9): 711～715