第34卷 第24期 计 算 机 工 程 2008年12月
Vol.34 No.24 Computer Engineering December 2008
·软件技术与数据库· 文章编号：1000—3428(2008)24—0046—03 文献标识码：A 中图分类号：TP311

基于R*-tree的时空数据库索引VC-tree

张桂杰，岳丽华，金培权
(中国科学技术大学计算机系，合肥 230027)

摘 要：在时空数据的索引结构中，HR-tree可以高效处理时间片查询，但对时间段查询效率低下，同时存在存储冗余。3D-tree索引的效率较低，双树结构使索引维护较为困难，且磁盘访问开销大。该文提出一种新的基于R*-tree的索引结构VC-tree，便于管理维护，可以高效满足时空查询，并满足有效时间内的未来查询。

关键词：时空数据库索引；R-tree索引；HR-tree索引；VC-tree索引

VC-tree: Spatial-temporal Database Index Based on R*-tree

ZHANG Gui-jie, YUE Li-hua, JIN Pei-quan
(Department of Computer, University of Science and Technology of China, Hefei 230027)

【Abstract】 The Historical R-tree(HR-tree) is a spatial-temporal access method. Since all objects are indexed by a single tree, the size and height of the tree is expected to be larger than that of the corresponding HR-tree at the query timestamp. However, the space requirements of HR-trees are still prohibitive in practice, because for most typical datasets HR-trees almost degenerate to independent R-tree, one for each timestamp. Their performance deteriorates very fast for interval queries as the interval length increases. And 3D-tree is another spatial-temporal access method. The retrieval of objects according to their spatial-temporal relationships with others, demands access to both indexes and, in a second phase the computation of the intersection set between the two answer sets. Access to both indexes is usually costly and, in many cases, most elements of the two answer sets are not found in the intersection set.

【Key words】 spatial-temporal database index; R-tree; HR-tree; VC-tree

1 概述

R-tree[1]提出了用最小包围矩形来表示空间对象，并给出了一种动态建立、索引、查询空间数据的算法。它将空间上相近的时空对象放在一起，便于索引和高效查询。R*-tree[2]在R-tree的基础上对部分算法进行了优化改进。3DR-tree[3]把时间作为时空对象的一维，它将时间和空间分开来索引，为时空对象建立2个索引结构，分别索引对象空间属性和时间属性。它可以满足时间点和时间片的查询，但是这种索引结构从根本上决定了存储冗余和索引低效，因为同一对象的信息会同时存储在2棵索引树中。进行时空数据查询时会产生一个空间对象集和一个时间对象集，最坏情况下2个集合的交集为空。HR-tree[4]以时间戳为基准构造当前时刻的R- Tree，它保存没有改变的前一刻的节点的指针，生成新的节点保存变化的节点的新值。它可以高效地满足时间点查询，然而用该种结构存储冗余是一大突出问题，同时时间片查询效率低下是该结构无法解决的根本问题。STR-tree[5]是对时空数据库中运动物体的一种索引方式，它用点来代表时空对象，把时空坐标系中运动对象的轨迹描述成一条折线，根据点的运动方向的改变将物体不同时间间隔的轨迹分开存储在不同的节点中。该方法在空间查询上难以处理，并且同一物体根据运动轨迹的变化存放在不同节点导致访问某个时间段内该物体要耗费较大的时间和空间代价。MV3R-tree[6]是基于MVR-tree, HR-tree, 3DR-tree改造的索引结构，在集成了上述3种树优点的同时也继承了它们的缺点。本文权衡存储和查询效率等综合因素提出了一种新的索引结构，即基于R*-tree的Virtual Column Tree(VC-tree)。

2 VC-tree

2.1 VC-tree简介

在R*-tree方法中，用包含空间对象的最小外包矩形来描述一个空间对象。如图1所示，矩形*S*在R*-tree索引中可以代表一个空间对象。在索引树中，根据节点外包矩形框的信息可以得到与其他节点的空间相关性(相交、包含、相离等)。

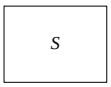

图1 空间对象

如果*S*包含的空间对象同时具备时间特性，那么可以采用图2的方式来描述该物体。在空间和时间坐标系下*S*变成柱体*SS*，*S*是空间维的描述，柱体的高是时间维的表示。

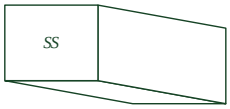

图2 时空对象

针对图2，将空间与时间统一构造VC-tree索引。将空间物体用生成时刻的最小外包围矩形表示，同时节点中保存生成时间的值和矩形框扩展的速度。时间是物体的一维，随着时间的增长，存活的物体在时间维上增长，基于矩形框描述的物体从外观上看类似柱体，或者是柱台体。这也就是用最

作者简介：张桂杰(1982－)，男，硕士研究生，主研方向：时空数据库；岳丽华，教授、博士生导师；金培权，副教授

收稿日期：2008-04-24 **E-mail：**llyue@ustc.edu.cn

小外包柱体描述时空对象，因此给出基于最小外包柱体的索引 VC-tree。柱体的下底是生成时刻包围物体空间属性的最小外包矩形，上底是对象消亡时刻包围物体空间属性的最小外包矩形，柱体的纵向高度为物体的生成时间到消亡时间的时间差长度，本文不存储真正的包围柱体而是存储时空对象的生成和消亡时刻和对象的扩展速度(速度大于 0 是扩展，小于 0 是负扩展，等于 0 是 0 扩展)根据相关速度和时间快速计算相关的柱体。因此该树节点采用虚拟的包含物体的最小柱体表示。VC-tree 索引的结构类似 R-tree，图 3 所示为空间对象的位置分布状况，相应的 R-tree 索引结构如图 4 所示。

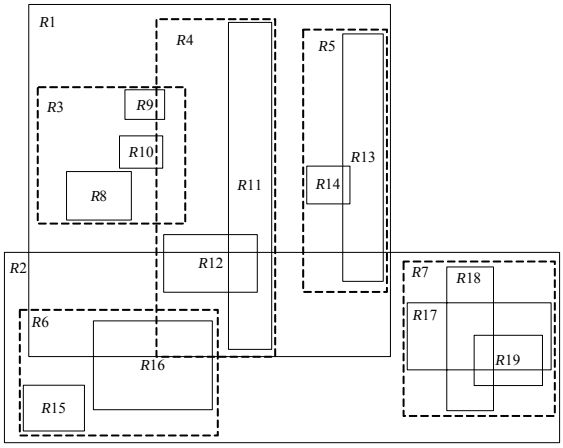

图 3　空间对象的位置分布

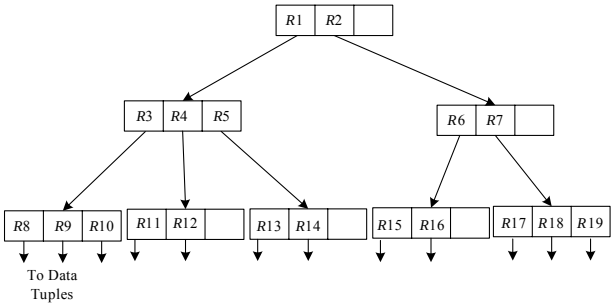

图 4　R-tree 索引结构

VC-tree 是一棵高平衡树，每个非叶子节点包含 $n(m \le n \le M)$个元素。其中，$m=M/2$, $M \in N$；根节点元素个数$\le m$。在 VC-tree 索引树中，叶子节点 *pointer* 指向数据库中的时空对象记录，叶子节点记录指向对象的生成和消亡时刻以及对象扩展的平均速度。非叶子节点包含若干个下层节点，同一个非叶子节点包含的各个子节点在空间上相近，时间上相邻。VC-tree 满足时间片和时间段的高效查询，同时因为引入了扩展速度，可以查询预测节点的变化，达到预警参考的效能。

2.2　VC-tree 节点结构

叶子节点的数据结构如下：

<*S*,*Tstart*,*Tend*,*V*,*pointer*>

其中，*S* 为包含物体的柱体/锥体的 *Tstart* 时刻的最小矩形；*V* 为物体的平均扩展速度，扩展为正数，收缩为负数，静止的物体为 0(假定物体的移动方向和速度不变)；*pointer* 指向实体；*Tstart*,*Tend* 分别为物体的生成和消亡时间。

非叶子节点的数据结构如下：

<*S*, *Tstartmin*, *Tendmin*, *Tstartmax*, *Tendmax*, *Vmin*, *Vmax*, *pointer*>

其中，*S*, *V*, *pointer* 同叶子节点；*Tstartmin* 为该节点包含的子节点中最早的开始时间；*Tendmin* 为该节点包含的子节点中最早的消亡时间；*Tstartmax* 为该节点包含的子节点中最晚的开始时间；*Tendmax* 为该节点包含的子节点中最晚的消亡时间。时间可以采用时间戳的方式，未消亡的为 now，表示为* 。

该树遵循以下规则：

(1)树的节点包含的元素个数不大于 $M(M \in N)$。

(2)树的节点包含的元素个数不小于 $m(M/2<m<M)$，根节点除外。

(3)该树是一棵平衡树，所有叶子节点都在同一层上。

3　VC-tree 相关操作

VC-tree 的相关操作是建立在 R-tree 和 R*-tree 基础上的。关于空间的相关概念定义如下：

Margin-value: margin area(first group)+margin area(second group)

Area-value: Area(first group)+Area(second group)

Overlap-value: Area(first group)∩Area(second group)

3.1　插入操作

在 VC-tree 树中插入新实体的策略是基于 R*-tree 的插入操作。插入一个新的元素时，首先定位要插入的子树，然后找到要插入的叶子节点 *L*，如果 *L* 有空间，那么直接插入新元素；如果 *L* 已经存在 *M* 个元素，那么插入新节点时，将引起节点分裂；新元素插入完成后调整树，使其重新达到高平衡树。

Insert 算法：将实体 *E* 插入到 VC-tree 中。算法如下：

Insert()

{(1)设叶子节点 L=ChooseSubtree();

(2)若 L 有空间容纳 E 将 E 插入 L 中，根据新节点的 Tstartmin, Tendmin, Tstartmax, Tendmax, Vmin, Vmax 调整 L 的相关值，设置 pointer 指针指向 E。否则调用 SplitNode 分裂 L 成 L'和 LL，分裂后的节点包含 L 中的实体和 E 调用 AdjustTree 调整树；

(3)若节点分裂向上传递导致根节点分裂生成新的根节点，子树为分裂的 L', LL； }

ChooseSubtree 算法：查找相应插入数据的路径，返回满足条件的节点。算法如下：

ChooseSubtree()

{(1)设 N 是树的根节点；

(2)若 N 是叶节点，若 N 是根节点则返回 N，若 N 不是根节点返回 N 的父节点。否则在 N 中查找节点 L，L 满足该节点以最大速度 V 扩展到当前时间时矩形 S 可以包含新节点的 S，在满足条件的 L 集合中查找节点 L'起始 S 包含新节点。如果有多个满足条件的 L，选择最小时间戳最大的；

(3)N=L goto (2); }

节点分裂的基本原则是在空间维上采用 R*-tree 的分裂策略，在时间维上按照生成时间分裂。分裂时优先考虑空间特性，然后再参考时间特性给出最优分裂结果。

SplitNode(), ChooseSplitAxis(), ChooseSplitIndex()同 R*-tree 中算法。

AdjustTree 算法：从叶子节点 *L* 向上调整树的结构，在需要分裂的节点处进行分裂。算法如下：

AdjustTree()

{(1)如果 L 是将要分裂的叶子节点，令 N=L 且 NN 是分裂后的第 2 个节点；

(2)若 N 是根节点，则停止；

(3)设 P 是 N 的父节点，设 En 是 N 在 P 中的链接点，调整 En 的矩形框的值使其可以紧紧包含 N 的值；

(4)若分裂过程中产生 NN，如果 P 中存在空间则在 P 中创建紧紧包含 NN 的链接点，并链接 NN。如果 P 中元素个数已经是 M，那么 P 分裂产生 P，PP 包含 N, NN 中的元素。

(5)若 P 分裂，令 N＝P, NN＝PP；跳转到(2)继续。 }

3.2 删除操作

在 VC-tree 中删除一个节点时，首先要定位要删除的节点所在的子树，找到要删除的叶子节点 L。如果删除元素后 L 中的元素个数少于 m，那么 L 将和相关兄弟节点合并然后删除 L 节点，如果 L 节点被删除，将调用树的压缩算法来调整树使其达到新的平衡。

Delete()算法：从数据库中删除元素 E，算法如下：

Delete()

{(1)调用 ChooseSubtree()定位 E 所在的叶节点 L；

(2)若 L 不存在停止，否则根据 E 的 Tstartmin, Tendmin, Tstartmax, Tendmax,Vmin, Vmax 查看 L 中的相关值是否与其相等，如果是，查找 L 的所有节点相关值并调整。从 L 中删除 E，调用 CondenseTree 算法调整树；

(3)若调整后的树的根节点只有一个孩子节点 N，令 N 为树的根节点，删除旧的根节点； }

Condense()算法：叶子节点 L 因元素删除导致包含的元素数少于 m，则 L 中的元素需要分配到 L 的兄弟节点中，然后删除 L；同时 L 的上层节点要调整外接柱体包的大小，使其达到最小。算法如下：

Condense Tree()

{(1)令 N=L，设 Q 为包含擦除节点的数组，初始为空；

(2)若 N 是根节点，跳转(5)；否则设 P 为 N 的父节点，则 En 是 N 在 P 中的链接点；

(3)若 N 包含的元素数≤m，则删除 P 中 N 的链接点 En，将 N 添加到 Q 中，否则调整 EnI 的大小紧紧包含 N 中的所有元素；

(4)N=P, repeat from 2;

(5)将 Q 中的元素重新插入到树中，叶子节点插入到叶子节点层； }

3.3 查询处理

空间数据库查询存在的几种可能是：空间，时间片，时间段，时间点上的空间，时间段内的空间查询等。时间的查询可以根据时间戳的值来查询，时间空间的查询相当于柱体窗口查询，可以查找满足包含该查询柱体窗的柱体。同时在节点中保存了柱体扩展的速度，可以预测未来，各个柱体之间的位置关系和何时可否达到某种空间关系(如相交、相离、穿过等)。在现实应用中，例如海上石油泄露事故中油区的扩散问题可以及时地跟踪和预测，为决策提供预测。

4 实验结果

实验中该结构和 3DRTree索引结构在性能方面作了比较。实验数据遵循测试关于移动点和矩形对象运动基准 GSTD[7]方法人工生成。实现中将初始密度定为 0.5，起始时刻时空对象的初始位置和大小按照正态分布，在随后的某个随机时刻后采用随机方法生成时空对象的位置和大小。查询中采用在不同数据量的前提下，随机查询 1 000 次求得页面的平均置换次数，每次的查询窗口和时间点采用随机的方式生成。

实验结果比较如下：

图 5 中横坐标代表一次插入的数据的条数，纵坐标代表插入一条数据所用的平均时间。实验过程中测试插入不同量级的数据时，2 种索引结构所用的插入时间。实验过程中分别生成 1 000, 5 000, 10 000, 50 000, 100 000 等量级的数据，插入到现有的树结构中，得出各个量级的总的插入时间。根据数据量的不同求出不同量级下的每条的插入时间，多次实验获得平均值。

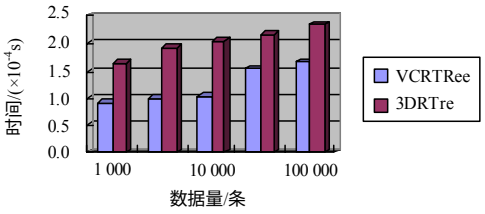

图 5 数据插入时间比较

图 6 中横坐标代表现有索引结构中存在的数据条数，纵坐标表示每个随机查询窗口引起的比较次数。查询过程中随机生成查询窗口，分别在 1 000, 5 000, 10 000, 50 000, 100 000 数量级下展开查询，每次查询采用 1 000 次随机查询求取总的查询时间，求出平均值；多次实验最终获得图 6 中的平均数值。

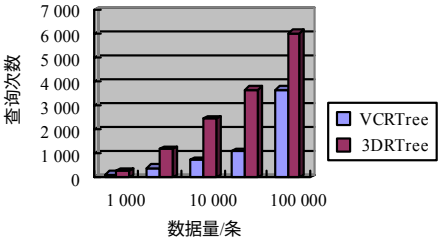

图 6 查询次数比较

5 结束语

VC-tree 基本思想基于 R*-tree 的扩展。VC-tree 较之 R-tree 有了处理时间数据的能力，解决了 H-tree 时间段查询效率低下的问题，部分弥补了 3D-tree 磁盘访问量大和有效查询低效的缺陷。

参考文献

[1] Guttman A. R-trees: A Dynamic Index Structure for Spatial Searching[C]//Proc. of International Conference on Management of Data. Boston, USA: ACM Press, 1984: 47-54.

[2] Beckmann N, Kriegel H P, Schneider R, et al. The R* Tree: An Efficient and Robust Access Method for Points and Rectangles[C]// Proc. of International Conference on Management of Data. Boston, USA: ACM Press, 1990: 322-331.

[3] Yannis T, Michael V, Timos S. Spatio-temporal Indexing for Large Multimedia Applications[C]//Proc. of International Conference on Multimedia Computing and Systems. [S. l.]: IEEE Press, 1996.

[4] Nascimento M A, Silva J R O. Towards Historical R-trees[C]//Proc. of the 13th ACM Symposium on Applied Computing. [S. l.]: ACM Press, 1998: 235-240.

[5] Pfoser D. Indexing the Trajectories of Moving Objects[J]. IEEE Data Engineering Bulletin, 2002, 25(2): 2-9.

[6] Tao Yufei, Papadias D. MV3R-tree: A Spatiotemporal Access Method for Timestamp and Interval Queries[C]//Proc. of the 27th International Conference on Very Large Databases. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2001: 431-440.

[7] Theodoridis Y, Silva J, Naseimento M. On the Generation of Spatiotemporal DataSet[C]//Proc. of the 6th Int'l Symposium on Spatial Databases. London, UK: Springer-Verlag, 1999.