第33卷 第11期 计 算 机 工 程 2007年6月
Vol.33 No.11 Computer Engineering June 2007
·软件技术与数据库· 文章编号：1000—3428(2007)11—0076—03 文献标识码：A 中图分类号：TP311

TPR+-tree：一种面向预言查询的有效时空索引

张 驭，岳丽华，金培权
(中国科学技术大学计算机科学技术系，合肥 230026)

摘 要：提出了一种面向预言查询的时空索引技术：TPR+-tree，给出了TPR+-tree的数据结构和关键算法，并引入了双极值子结点的概念，通过对双极值子结点进行检测和排除，减小了结点面积，改善了结点间的重叠。试验结果表明，TPR+-tree具有更高的查询性能，是一种有效的面向预言查询的时空索引。

关键词：时空索引；预言窗口查询；双极值子结点

TPR+-tree: An Efficient Spatial-temporal Index for Predictive Window Query

ZHANG Yu, YUE Lihua, JIN Peiquan
(Department of Computer Science & Technology, University of Science &Technology of China, Hefei 230026)

【Abstract】 This paper proposes a spatial-temporal indexing technique named TPR+-tree that supports the predictive window query, presents the data structure and the key algorithms and introduces the concept of double-extremum childnode. TPR+-tree reduces the area and overlap of nodes by checking and eliminating the double-extremum childnodes. The experimental results indicate that TPR+-tree promotes the query performance. It is an efficient spatial-temporal index for predictive window query.

【Key words】 Spatial-temporal index; Predictive window query; Double-extremum childnode

1 概述

时空数据库(Spatio-temporal Database)是存储着大量动态对象的数据库系统，这里的动态包括对象的位置和/或范围的变化。近年来，时空数据库在日益增长的各项应用(如交通管理、气象检测、地籍管理、移动计算)中发挥着越来越大的作用。时空索引研究在近年也十分活跃，并已经针对不同类型的时空数据提出了一些索引算法。按照查询时间与当前时间的关系划分，时空索引主要分为面向历史查询的索引和面向预言查询的索引。本文主要研究后一种索引。

预言窗口查询(predictive window query)是指给定一个查询区域 q_R 和一个未来时间间隔 q_T，检索在任意时间戳 $t \in q_T$ 与 q_R 相交的对象集合(例如：查询在接下来的 10min 内出现在北京上空的所有飞机)。如果 q_T 是一个时间戳，则该查询称为预言时间戳查询，否则称为预言时间段查询。

时空数据库通常使用移动函数(motion function)来表示对象的移动，通常为线性函数，数据更新只发生在函数参数改变的时候。与此相应，一个对象 o 的记录包括(i)在基准时刻 t_{ref} 对象的范围 o_R (ii)对象的当前速度 o_V。因此，对象 o 在任意时刻 t 的位置和范围可表示为 $o_R+(t-t_{ref})o_V$。

目前较为流行的面向预言查询的时空索引主要有TPR-tree(time-parameterized R-tree)[2]及其扩展，如TPR*-tree[3]，它们都是在R*-tree[1]的原有结构和算法上进行了扩展和优化。TPR-tree将对象表示为MBR/VBR对，其中MBR表示在基准时刻 t_{ref} 对象的范围，VBR表示对象的当前速度。TPR-tree在时间段 $[t_C, t_C+H]$ 内优化了查询，其中 t_C 为当前更新时间，H是一个称为horizon的参数，表示未来查询的时间极限。它的优化算法只是简单地将R*-tree中的 4 大优化参数(面积、重叠、边缘和质心间距离)替换为它们的积分版本。以面积为例，即将结点N的面积增加值 $A(N)$ 替换为面积增加值的积分 $\int_{tc}^{tc+H} A(N,t)dt$。TPR-tree的算法过于简单，没有过多地考虑树的结点随时间迅速增大的特性。随着时间的推移，结点的面积不断增大，重叠越发严重，会导致查询效率的降低。TPR*-tree与 TPR-tree的结构一致，并在其上优化了算法。它使用sweep region替换了TPR-tree中的积分，sweep region虽然易于计算，却并不能很好地体现对象在时间段中的变化。

本文提出了一种新的面向预言查询的时空索引TPR+-tree，它改进了 TPR-tree 的结构和算法。其中，双极值子结点的概念，通过对双极值子结点的检测和排除，减小了结点面积，减少了结点间的重叠，并使结点 MBR 趋于正方(quadratic)，提高了查询性能。试验结果证明了这一点。

2 背景及问题描述

2.1 保守边界矩形及收紧算法

由于结点中只记录速度和 t_{ref} 时刻(通常为索引建立时刻)的范围，结点随着时间的推移其 MBR 是变化的，需要查询或更新的时候使用公式 $o_R+(t-t_{ref})o_V$ 动态计算，因此保证结点 N 在 t 时刻完全包含其所有子结点变得格外重要。文献[2]提出了保守边界矩形的概念，即任意时刻保证结点边界矩形

基金项目：国家自然科学基金资助项目(6040302)
作者简介：张 驭(1979－)，男，硕士生，主研方向：时空数据库索引；岳丽华，教授、博导；金培权，博士
收稿日期：2006-06-30 **E-mail：**zyjacky@mail.ustc.edu.cn

任意维的左边界以结点下所有子结点在该维上的最小速度移动，右边界以结点下所有子结点在该维上的最大速度移动。具体的，结点在 t_{ref} 时刻的 MBR/VBR 可表示为：

$$x^{\vdash} = \min\{n_i.x^{\vdash}(t_{ref})\}$$

$$x^{\dashv} = \max\{n_i.x^{\dashv}(t_{ref})\}$$

$$v^{\vdash} = \min\{n_i.v^{\vdash}\}$$

$$v^{\dashv} = \max\{n_i.v^{\dashv}\}$$

以上被称为 load-time 边界矩形，可以保证在 t_{ref} 后的任意时刻完全包含其所有子结点，但却不能保证最小，且增长迅速。因此文献[2]提出在更新时收紧边界矩形，具体的，结点在 t_{upt} 时刻的 MBR/VBR 可表示为：

$$x^{\vdash} = \min\{n_i.x^{\vdash}(t_{upt})\} - v^{\vdash}(t_{upt} - t_{ref})$$

$$x^{\dashv} = \max\{n_i.x^{\dashv}(t_{upt})\} - v^{\dashv}(t_{upt} - t_{ref})$$

以上被称为 update-time 边界矩形。

2.2 3 个时间参数

在当前时刻对未来查询，并不确定移动对象的速度何时更新，查询的时间越远，产生的误差可能越大，因此现实应用中应限制时间窗口相对现在时刻的延伸。同时随着时间的推移，结点边界矩形不断增大，会导致重叠增加，索引效率降低，因此索引需要使用一段时间后重建。文献[2]提出了 3 个影响 TPR-tree 性能的时间参数：

(1)Querying window(W)：查询窗口距离当前时刻最长的时间间隔；

(2)Index usage time(U)：索引可用于查询的最长时间；

(3)Time horizon(H)：W 与 U 之和。索引在建立了 H 时长后需要重建。

2.3 双极值子结点

设 n 为结点 N 在第 d 维上的速度最小/大的子结点，同时在 t 时刻 n 成为 N 中位置最左/右的子结点，那么称 n 为 N 在第 d 维上的双极小/大值子结点。当 N 为叶结点时，则称 n 为双极值对象。由于采用保守边界矩形，双极值子结点同时决定了父结点的 MBR 和 VBR 的范围，而且是最快移离父结点 MBR 质心的子结点，最为糟糕的是，它使得收紧算法无效，父结点 MBR 的增长速度达到最快，重叠越发严重。

不失一般性，以双极小值子结点为例。设 n 为结点 N 在第 d 维上的双极小值子结点，N 在 t_1 和 t_2 时刻两次更新，因此：

$$\begin{aligned} x_2^{\vdash} &= \min\{n_i.x^{\vdash}(t_2)\} - v^{\vdash}(t_2 - t_{ref}) \\ &= n.x^{\vdash}(t_2) - n.v^{\vdash}(t_2 - t_{ref}) \\ &= n.x^{\vdash}(t_1) + n.v^{\vdash}(t_2 - t_1) - n.v^{\vdash}(t_2 - t_{ref}) \\ &= n.x^{\vdash}(t_1) - n.v^{\vdash}(t_1 - t_{ref}) \\ &= \min\{n_i.x^{\vdash}(t_1)\} - v^{\vdash}(t_1 - t_{ref}) = x_1^{\vdash} \end{aligned}$$

t_2 时刻的更新并没有缩小 N 的 MBR。

3 TPR+-tree 数据结构和算法

3.1 数据结构

TPR+-tree 是一棵平衡多叉树。结点结构为 $(DE, e_1, e_2, ..., e_n)$，其中 DE 标识双极值结点的入口位置，结构为 $(DMin_1, DMax_1, DMin_2, DMax_2, ..., DMin_d, DMax_d)$，$DMin_i$ 和 $DMax_i$ 分别表示结点在第 i 维上的双极小/大值子结点所在的入口；e_i 表示入口，其中中间结点的入口结构为(MBR,VBR,cnptr)，叶结点的入口结构为(MBR,VBR,optr)，cnptr 和 optr 分别表示指向子结点和对象的指针。

3.2 插入和删除

TPR+-tree 沿用了 TPR-tree 的查询算法，改进了插入和删除算法，其中优化了 ChoosePath 和 ShrinkBounding Rectangle 算法，同时提出了新的 EliminateOverloap 和 PickDoubleExtremum 算法。

3.2.1 PickDoubleExtremum

如前所述，双极值子结点会使收紧算法无效，从而加重重叠。假设数据空间中有 6 个对象，如图 1(a)。在 t_{ref} 时刻结点的划分如图 1(b)，在 t_{acc} 时刻结点的划分如图 1(c)，此时对象 2 和 4 成为了各自父结点的双极值对象。正因为这两个对象的不合理分派造成了父结点重叠，且随着时间的推移其所处父结点的边界不会收缩，重叠只会越发严重。图 1(d)为较好的结点划分。

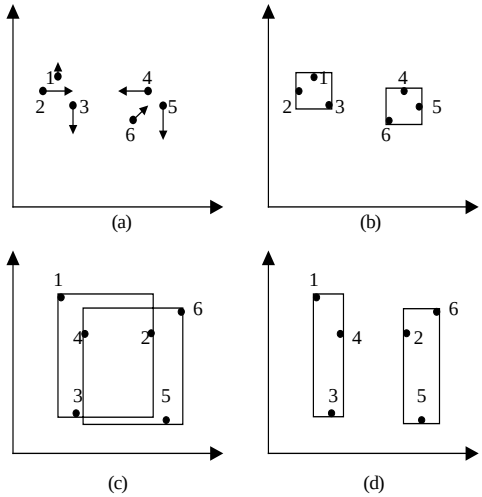


图 1 移动对象及叶结点 MBR 变化

PickDoubleExtremum 算法就是将双极值子结点的入口从父结点中提出，然后重新插入。为了优化算法，TPR+-tree 在结点中增加了数据结构 DE。

算法 PickDoubleExtremum

(1)访问 DE 结构，找出没有记录双极小或极大值子结点的维，然后遍历所有入口，查找相应维上的双极小或极大值子结点，并记入 DE 中相应的域；

(2)按照双极值所在维上的速度绝对值的顺序(假设数据空间每一维长度相等)，将 DE 中的双极值子结点逐一提出，删除父结点中相应入口，直到 DE 中双极值子结点全部被提出或者父结点的入口数=m(结点最小入口数)。

提出双极值子结点可以减小父结点面积，消除重叠，同时提出双极值所在维上的速度绝对值较大的子结点可以使父结点趋于正方，进一步减少重叠的可能。在提出双极值子结点后，可能另一个子结点会成为新的双极值子结点，该子结点不需要在本次 PickDoubleExtremum 中被提出，因为很快会有速度绝对值更大的结点插入父结点或者该子结点会在下一次的 PickDouble-Extremum 中被提出。

3.2.2 ChoosePath

TPR-tree 的 ChoosePath 算法继承了 R*-tree 的思想，只是将面积、重叠等参数换成了它们的积分版本。但算法并没有说明，当对象插入数个结点均不退化，对数个结点面积和重叠增加值的积分均为 0 时该选择哪个结点插入，这种情况在索引使用一段时间，重叠现象增多时会频繁出现。假设当

对象 o 插入结点 N_1 和 N_2 后，两结点面积增加值的积分 $\int_{tc}^{tc+H} A(N_1,t)dt = \int_{tc}^{tc+H} A(N_2,t)dt = 0$，那么必然 N_1 和 N_2 在 t_C 时间后一直重叠，且 o 一直位于两结点的重叠区域中。TPR+-tree 的 ChoosePath 算法在对象插入数个结点不退化的情况下，会首先尝试消除重叠，调用 EliminateOverloap 算法，在 Eliminate-Overloap 算法结束后如果还存在不退化的多个结点，则随机地插入任意一个。

算法 EliminateOverloap

(1)从不退化的结点开始，向叶结点方向一层层查找其插入同一对象后不退化的子结点，直到到达叶结点或者该结点的所有子结点在插入对象后面积增加值的积分均>0。将每条查找路径上的末端结点简称为末端结点。

(2)对每个末端结点调用 PickDouble-Extremum 算法，提出双极值子结点，同时调用 ShrinkBoundingRectangle 算法。

(3)把提出的双极值子结点重新插入到树中。

3.2.3 ShrinkBoundingRectangle

TPR-tree 的 ShrinkBoundingRectangle 算法只是简单地将更新的叶结点到根的唯一路径上所有结点 MBR 收紧为 update-time 边界矩形。前文说过，双极值子结点的存在会使收紧算法失效，因此 TPR+-tree 的 ShrinkBoundingRectangle 算法在收紧结点 MBR 前会先提出结点中的双极值子结点。

算法 ShrinkBoundingRectangle

(1)调用 PickDoubleExtremum 算法，提出双极值子结点。

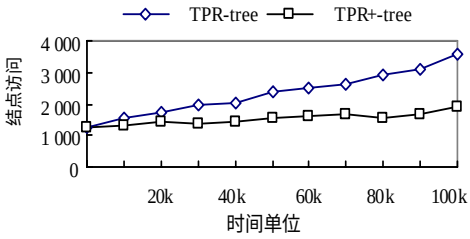
(2)把提出的双极值子结点重新插入到树中。

(3)将更新的结点到根的唯一路径上所有结点 MBR 收紧为 update-time 边界矩形。

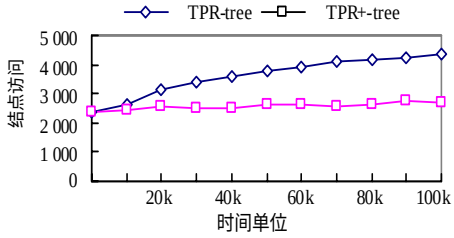
PickDoubleExtremum 算法由 ChoosePath 算法和 ShrinkBoundingRectangle 算法调用，接着，在将提出的双极值子结点重新插入树的过程中有可能触发 ChoosePath 算法和 ShrinkBoundingRectangle 算法，从而产生多米诺骨牌效应，使整个对象空间中的重叠现象得到一定程度的改善。值得注意的是，假设双极值子结点 n 从父结点 N 中提出，又被重新插入到 N 中，则在新一轮的 PickDoubleExtremum 算法中该子结点不被提出，否则算法将不会结束。

4 试验结果

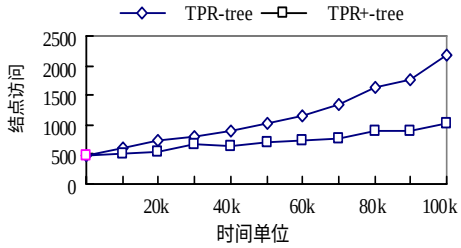
本文使用了文献[5]提供的时空索引库中的 TPR-tree，并在其上开发了 TPR+-tree，据此对两种树进行了查询性能上的比对。数据空间为 2 维，每一维上的区间为[0,10 000]，共有 100 000 个点对象，结点最大入口数为 27，树高 4 层。在 0 时刻插入 100 000 个点对象，对象每一维位置值在[0,10 000]间随机产生，每一维速度值在[-50,50]间随机产生，其后每一个时间单位随机更新一个对象，即对该对象删除和插入一次。W 固定为 5 000 个时间单位，H 固定为 110 000 个时间单位。查询窗口有 3 个主要参数：$q_R len$，$q_V len$，$q_T len$，分别表示查询窗口范围长度、速度长度和时间长度。查询窗口每一维的起始位置在[0,10 000-$q_R len$]间随机产生，起始速度在[-10,10-$q_V len$]间随机产生，起始时间在[$t_C, t_C + 5000 - q_T len$]间随机产生。每隔 10 000 个时间单位执行一次查询。图 2(a)/ 2(b)、图 2(c)/ 2(d)、图 2(e)/ 2(f)分别使用不同的 $q_R len$，$q_V len$，$q_T len$值对 TPR-tree 和 TPR+-tree 的结点访问量进行了比对，可以看出随着时间的增加，由于减小了结点面积和改善了重叠，因此 TPR+-tree 的结点访问量较 TPR-tree 有明显减少，性能有明显提升。



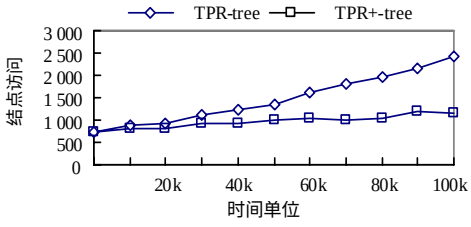
(a) $q_R len$=100，$q_V len$=5，$q_T len$=2 500



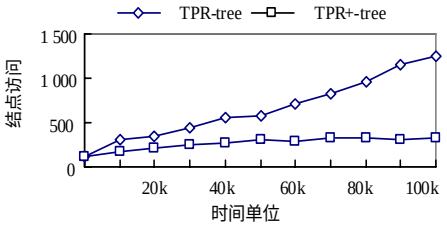
(b) $q_R len$=1 600，$q_V len$=5，$q_T len$=2 500



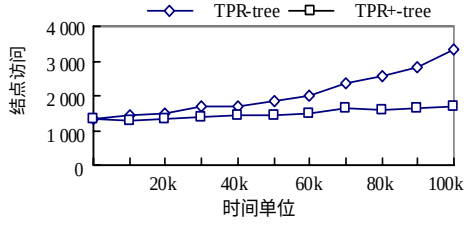
(c) $q_R len$=400，$q_V len$=0，$q_T len$=2500



(d) $q_R len$=400，$q_V len$=10，$q_T len$=2500



(e) $q_R len$=400，$q_V len$=5，$q_T len$=1



(f) $q_R len$=400，$q_V len$=5，$q_T len$=5000

图 2 TPR-tree 和 TPR+-tree 试验比对

5 结论

本文提出了一种有效的面向预言查询的时空索引 TPR+-tree，它通过对双极值子结点的检测和排除，较 TPR-tree 有更高的查询性能。试验结果证明了这一点。

（下转第 81 页）