动态环境下移动对象索引技术研究

何凯涛　唐 宇　廖 巍　郁文贤
(国防科技大学电子科学与工程学院　长沙　410073)

摘　要：TPR-tree 是目前广泛使用的移动对象当前及未来位置索引技术。该文综合考虑移动对象在速度域和空间域中的分布，提出了 ETPR 树索引结构，首先在速度域中对移动对象集进行划分，根据速度矢量大小将移动对象映射到不同的速度桶，每个速度桶中移动对象具有相近的速度矢量；对每个速度桶中的移动对象，则利用 TPR 树进行索引。性能分析和实验表明， ETPR 树索引的动态更新性能和查询性能均优于 TPR*-tree 等索引。

关键词：移动对象索引；ETPR 树；构建算法
中图分类号：TP392　　**文献标识码**：A　　**文章编号**：1009-5896(2008)10-2507-05

Research on Moving Objects Indexing Method in Dynamic Environment

He Kai-tao　Tang Yu　Liao Wei　Yu Wen-xian
(*College of Electronic Science and Engineering, National University of Defense Technology, Changsha* 410073, *China*)

Abstract: TPR-tree is the most popular indexing method for the current and future position of moving objects. In this paper, a new indexing method, ETPR-tree, which takes into account the distribution of both velocity domain and space domain is presented. First the velocity domain is split, and moving objects are classified into different velocity buckets by their velocities, thus the moving objects in one bucket have similar velocities. Then TPR-tree is used to index the moving objects in each bucket. Experimental results show that ETPR-tree update and query performance outperform any other indexing method including TPR*-tree.

Key words: Moving objects indexing method; ETPR-tree; Construction algorithm

1　引言

随着无线通信与 GPS 等空间定位技术的发展，在许多应用如交通调度控制及移动计算等领域往往需要对移动终端的位置进行追踪管理以提供相关查询服务。但是由于移动终端的位置随着时间变化，而在传统空间索引结构中存储空间对象的具体位置，无法适应大量的空间对象位置更新操作，因而不适合于移动对象的存储与检索[1]。目前大部分研究工作通常采用时空访问方法，即在索引结构中存储移动对象的运动特性而非位置信息以对移动对象位置进行存储和检索[2]。

近年来研究者提出了许多基于参数化和二元变换的索引方法来对移动对象当前及未来位置进行存储和检索，如 TPR-tree[1]及其变种 TPR*-tree[3]，VCI R-tree[4]，STRIPE 索引[5]等。在上述索引方法中，VCI R-tree 在处理大量并发简单时空查询时表现出较好的性能；STRIPE 索引在变换域对移动对象位置进行索引和查询，处理简单时空查询具有较好性能，但对于连续邻近查询等算法性能并非有效。TPR-tree 及 TPR*-tree 由于能够沿用传统 R-tree 空间索引查询及插入、删除等动态更新算法而成为目前最实用的移动对象当前及未来位置索引方法。

在实际应用中，移动对象数据库中往往管理着速度分布极不均匀的移动对象，而现有的移动对象索引包括 TPR-tree 构建算法将空间域与速度域等同考虑，忽略了移动对象在速度域中分布的特殊性，索引构建算法可能将在空间域邻近但速度域中不相邻的移动对象聚簇在同一个索引节点 MBR 中，造成同一页面中的移动对象速度分布不均匀，导致节点 MBR 随着时间变化而急剧扩展，索引性能随着时间变化而迅速下降。

基于上述考虑，本文综合考虑移动对象在速度域和空间域中的分布，提出了一种改进的移动对象索引 ETPR 树 (Enhanced Time-Parameterized R-tree, ETPR-tree)，并给出了 ETPR 树索引的构建、动态维护及查询算法。本文内容组织如下：第 2 节介绍 TPR-tree 和 TPR*-tree 索引等相关工作；第 3 节则详细讨论了 ETPR-tree 及其构建、动态维护及查询算法；第 4 节是实验结果与性能评价；最后是结束语。

2　相关工作

Saltenis 等人[1]提出的 TPR-tree 是最早用于检索移动对象当前及未来位置的参数化访问方法。TPR-tree 使用时间参数化的 MBR 来包围移动对象，其索引结构和 R-tree 结构非常类似，区别在于 TPR-tree 索引结构中每个叶节点记录中不仅存储了移动对象的位置信息，而且包括了移动对象的速

2007-03-14 收到，2007-09-17 改回
国家自然科学基金(60472031)资助课题

度矢量。相应地，其中间节点记录同样包含子节点 MBR 及速度矢量 VBR。TPR-tree 节点 MBR 是关于时间的函数(利用 VBR 来描述 MBR 在不同维方向上的速度矢量)。具体来讲，在每一维上，MBR 上/下界是其所包含的所有对象或子节点 MBR 速度的最大/小速度，即不论移动对象位置如何变化，TPR-tree 节点 MBR 始终包含其子节点或移动对象。

图 1 所示为二维空间 TPR-tree 索引在构建时刻 $t=0$ 及未来时刻 $t=1$ 时最小矩形包围框 MBR 和速度包围框 VBR 表示。在 TPR-tree 中，预测窗口查询与 R*-tree 同样方式执行，不同的是在任意未来查询时刻 t，移动对象及节点的 MBR 必须利用 VBR 在线计算出然后与查询窗口进行比较。TPR-tree 的插入算法与标准 R*-tree 插入算法类似，区别在于插入算法中选择路径的度量准则由静态参量变为时间参数化的积分度量。具体来讲，在 R*-tree 中插入新的对象记录时，插入算法使用一些静态参量诸如 MBR 面积、周长、中心距等度量来作为插入路径选择时衡量准则。相应地，TPR-tree 中使用时间参数化的度量来作为插入算法中路径选择时的衡量准则。在 TPR-tree 中发生插入操作时，插入算法每次递归操作都会选择一个节点，其扩张体积是所有非叶节点中最小的，如周长积分在叶节点层具有最小的扩张。当插入操作中发生节点上溢时，分裂算法则类似于 R*-tree。但是在分裂方案选优排序时不仅仅考虑节点最小包围框分布，而且还必须考虑其速度矢量以达到节点的最优聚簇。

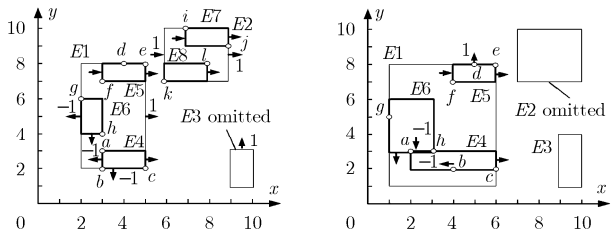

图 1 TPR 树矩形包围框

3 ETPR 树索引

3.1 ETPR 树索引构建算法

具体的说，ETPR 树除保留了基本的 TPR 树结构，另外还增加了一个基于内存的线性链表结构，存放速度桶的描述信息，记录形式为三元组 < MBR,VBR,tpr > ，其中 MBR,VBR,tpr 分别表示构建时刻速度桶的空间域范围、速度域范围、及指向构建于速度桶所包含的移动对象之上的 TPR 树索引的指针。如图 2 所示为 ETPR-tree 树索引结构，图中右上角是一个速度桶哈希表结构，每条记录描述了所指向的 TPR 树索引 MBR，VBR，图中左边所示为每个速度桶对应的 TPR 树索引。

Lin 等人在文献[6]中提出了利用时空直方图由底向上构

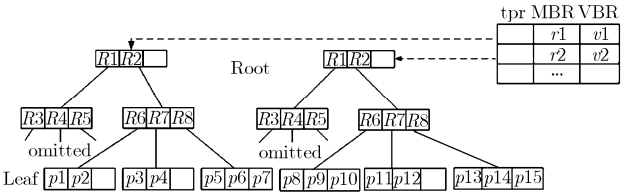

图 2 ETPR 树索引结构

建 TPR-tree 索引的 HBU 批装载(bulk loading)技术，HBU 算法思想是从叶节点层开始，根据 TPR-tree 索引每层的预期节点数目和当前层矩形包围框数目，由底向上地逐层构建 TPR-tree 索引。借鉴了 HBU 算法的思想，本文提出了基于速度直方图对速度域进行划分的 ETPR-tree 索引构建算法，其主要步骤如下：算法首先根据移动对象集合大小和 TPR-tree 节点扇出计算出预期的速度桶数目和速度桶窗口范围，并对速度域进行划分；然后对每个速度桶中的移动对象利用 HBU 算法逐层构建 TPR-tree 索引。

令 f 表示 TPR-tree 索引扇出，$M_i=f^i$ 表示高度为 i 的 TPR-tree 索引所包含的移动对象数目，假定移动对象集合数目为 n，其对应的预期 TPR-tree 索引高度为 $i+1$，则有 $M_i<n<M_{i+1}$，对 TPR-tree 索引而言，我们希望其所包含的 TPR-tree 高度为 i，即每个速度桶所包含的移动对象集合数目不能超出 M_i，因此期望的速度桶数目定为 $K=[n/M_i]$，由于移动对象速度分布的不均匀性，在确定了速度桶数目之后，还必须确定每个速度桶的速度范围。在此，引入一种新的度量准则，矩形包围框在索引生命期之内所覆盖的区域面积最小，以选取最优的速度桶划分方案。

令 V_x,V_y 分别表示速度域在两个速度维上的划分数目，则有 $V_x=[K/V_y]$，给定移动对象集合在两个速度维上的直方图 H_x,H_y，期望的速度桶数目 K，算法 DeterminePartition()描述了速度域划分算法具体步骤。

算法 1 DeterminePartition 算法

Algorithm DeterminePartition

Input: H_x, H_y, K; Output: L_x,L_y

BEGIN

(1)For any integer combination (V_x , V_y) satisfying $V_x=[K/V_y]$ do

(2) $L_x=\text{CalculateExtent}(V_x,H_x)$

(3) $L_y=\text{CalculateExtent}(V_y,H_y)$

(4) $\text{cost}\leftarrow\sum_{i=1}^{V_x}\int_0^h L_x[i]\text{d}t+\sum_{i=1}^{V_y}\int_0^h L_y[i]\text{d}t$

(5)If cost is better valued,

(6)return current L_x, L_y values

(7)End If

(8)End For

End

其中算法 1 的步骤(1)遍历所有可能的速度域划分方案；步骤

(2)到步骤(3)计算出当前划分方案下的速度桶范围分布；步骤(4)计算出当前划分方案的代价；步骤(5)至步骤(6)从所有划分方案中选择具有最小代价的方案。

令 Num 表示该速度维上给定的划分数目，给定移动对象集合在该速度维上的直方图 H ，引入中间变量 NumPer Partition 表示每个速度桶中预期的移动对象数目，则算法 CalculateExtent()描述了每个速度维上划分范围的具体计算步骤。

算法 2　CalculateExtent 算法

Algorithm CalculateExtent

Input: Num,H ;　Output: L .

BEGIN

(1) NumPerPartition $\leftarrow N/\text{Num}$, $j \leftarrow 0$;

(2)For i from 0 to Num do

(3)Starting from the jth bucket, count the $H[j]$. num into the partition until their sum = NumPer Partition

(4) $L[i]$. extent $\leftarrow$ sum of the $H[j]$. extent

(5)Adjust j to corresponding value for next

(6)End For

(7)return L

END

其中算法 2 的步骤(1)计算出每个速度桶划分预期的移动对象数目；步骤(2)至步骤(5)分别计算出在每个速度桶划分的速度范围；最终每个划分具有相近的移动对象数目，划分范围为其所包含的速度直方图单元集合的速度范围。

速度域划分算法目的是将移动对象集合根据速度矢量大小划分成不同具有相近移动对象数目的子集合，以保证每个速度桶对应的 TPR-tree 索引具有相同的高度，从而使得 TPR-tree 索引具有良好的结构。算法 2 给出了 TPR-tree 索引的速度桶构造算法，每个速度桶对应的 TPR 树索引构造算法参见文献[6]，在此不再赘述。

3.2　ETPR 树索引插入、删除算法

在 ETPR-tree 索引中插入新的移动对象记录时，插入算法首先依次扫描内存中的速度桶链表，从中找到移动对象速度矢量所落在的速度桶，根据速度桶记录中的指向 TPR-tree 索引的指针获得所插入的 TPR-tree 索引；然后在此 TPR-tree 索引中利用标准的 TPR*-tree 插入算法将新的移动对象记录插入到合适的位置；最后生成新的哈希表索引记录并插入到哈希表中合适的位置。若在插入过程中发生节点上溢，对上溢节点的分裂方法与 TPR*树类似，具体算法可参考文献[3]。若速度桶指记录指向的 TPR-tree 索引根节点发生上溢，则算法必须对此速度桶所包含的移动对象集合利用 3.1 节提出的构建算法重新构建两个 TPR-tree 索引，并在速度桶链表中删除原有记录并插入两条新的速度桶记录。

从 ETPR-tree 索引删除移动对象记录时，删除算法首先利用基于磁盘的移动对象标识哈希索引直接定位移动对象记录所在的 TPR-tree 叶节点页面，从中删除此记录即可，若叶节点页面的矩形包围框 MBR 或速度包围框 VBR 发生变化，则利用标准的 TPR-tree 索引删除算法对删除路径中的节点依次进行 MBR 和 VBR 的修改以紧致节点包围框，同时必须从哈希辅助索引表中删除此移动对象对应的记录。

3.3　ETPR 树索引窗口范围查询算法

给定窗口范围查询 $q(w,t)$ ，其中 w 表示查询窗口的空间范围， t 表示查询窗口的时间范围。查询算法首先扫描速度桶链表，对每个速度桶记录，根据索引构建时刻的速度桶 MBR 和 VBR 大小，可以判断出速度桶内的移动对象是否可能会落在查询 q 窗口范围内，若查询 q 在查询时间窗口 t 范围内与此速度桶所覆盖的空间域不相交，则查询算法无需对此速度桶所指向的 TPR-tree 索引进行访问；否则必须对速度桶指向的 TPR-tree 索引采用深度优先搜索算法进行查询。

若矩形包围框 E 在查询时间窗口 t 范围内与查询 q 的空间区域 w 不相交，那么矩形包围框 E 内的移动对象不会落在查询 q 中。显然，只有当矩形包围框 E 的 4 个顶点在查询窗口 t 范围内不落在空间区域 w 中，且查询 q 的空间区域 w 的 4 个顶点在时间窗口 t 范围内不落在矩形包围框 E 中，可以保证矩形包围框 E 在查询时间窗口 t 范围内与查询 q 的空间区域 w 不相交。

矩形包围框 E 在查询时间窗口 t 范围内与查询 q 的空间区域 w 相交可以分为以下 3 种情况：(1)在查询时刻矩形包围框 E 与查询 q 的空间区域 w 相交，显然必须搜索节点 E ；(2)在查询时刻矩形包围框 E 与查询 q 的空间区域 w 不相交，但在查询时间窗口 t 范围内某时刻查询 q 的空间区域 w 的某个顶点落在矩形包围框 E 中，此时也必须搜索节点 E ；(3)在查询时刻矩形包围框 E 与查询 q 的空间区域 w 不相交，但在查询时间窗口 t 范围内某时刻矩形包围框 E 的某个顶点落在查询 q 的空间区域 w 中，此时也必须搜索节点 E 。

算法 3 描述了预测窗口查询处理算法的主要步骤。

算法 3　WindowQuery

Algorithm WindowQuery

Input: $q(w,t)$, TPR-tree;　Output: Set(q)

BEGIN

(1)For all entries in velocity buckets Do

(2)If any vertex of bucket lies in w or any vertex of w lies in bucket during t

(3)Then do depth-first search in the TPR-tree of this bucket;

(4)Else skip this bucket;

(5)End If

(6)From root point, for each entry E in TPR-tree Do

(7)If any vertex of E lies in w or any vertex of w lies in E during t

(8)Then recursive search this subtree in depth-first

manner until the leaf level and add results to Set(q);

(9)Else skip the subtree;

(10)End If;

(11)End For

(12)End For;

END

其中算法 3 的步骤(1)到步骤(5)对速度桶链表进行访问，裁剪不需要进行搜索的速度桶；步骤(6)到步骤(8)则对 TPR-tree 中间节点按照上述 3 种情况进行判断，若存在某时刻使得矩形包围框 E 的顶点落在查询 q 的空间区域 w 中，或查询 q 的空间区域 w 的顶点落在矩形包围框 E 中，则对此节点进行深度搜索；否则裁剪此节点。

4 实验结果与分析

4.1 实验内容与设置

为了评估 ETPR-tree 索引的查询与动态更新性能，我们进行了如下的实验。实验模拟数据与文献[3]类似，采用随机生成，数据集大小为 100k 个移动对象，移动对象用点坐标来表示，均匀分布在 10000 距离单元×10000 距离单元的空间域内(距离单元可以取为 1m、10m、1 公里等，以下表示空间域的数值单位均为距离单元)。移动对象运动用如下方式模拟，从 LA[Tiger](包含 128k 个二维矩形框)实际数据集中抽取 5000 个 MBR，其中心作为移动对象运动目的地，在参考时刻每个移动对象随机选择一个目的地运动，速度大小在 $[0, 20]$距离单元/S 之间均匀分布，移动对象到达目的地后则会重新随机选择一个新目的地和速度大小进行运动。若移动对象速度更新超过 5 次后，则从数据集中删除此对象，并随机生成新的移动对象加入数据集中。每个移动对象平均每隔 150 个时间单元更新速度矢量(我们令时间单元 unit=1s)，即在每个时间单元数据集中有大约 534 个移动对象发出更新请求，133 个移动对象删除请求和 133 个移动对象插入请求。

预测窗口查询产生方式如下：(1)查询 q 空间窗口大小 qRlen 分别固定为 100×100, 400×400, 800×800, 1200×1200, 1600×1600，在查询时间范围内窗口大小保持不变；(2)窗口中心点位置在 $(10000-\text{qRlen})\times(10000-\text{qRlen})$ 空间域内随机分布；(3)查询时间窗口范围为 [Tisu,Tisu + TL] (Tisu 为查询提交时刻)，其中 TL 为查询时间窗口大小 qTlen，令 TL = 20,40,60,80,100(s)。查询性能估计采用回答 10 个预测窗口查询所需要的平均节点访问次数来衡量。动态更新性能则用每个时间单元内移动对象更新所需要的平均节点访问次数来衡量。

表 1 ETPR-tree 索引参数

参数	参数默认值	参数描述
速度桶数目	12/16/20/25/30	速度域格网大小分别为 3×4,4×4,4×5,5×5,5×6
页面大小	1kB	TPR-tree 的页面大小
中间节点扇出	31 个	ETPR-tree 所指向的 TPR-tree 中间节点扇出
叶节点扇出	62 个	ETPR-tree 所指向的 TPR-tree 叶节点所包含的移动对象记录数目
平均高度	3	ETPR-tree 所指向的 TPR-tree 大约平均高度

4.2 实验结果与分析

本文比较了 TPR-tree 与 TPR*-tree，ETPR-tree 在移动对象速度更新及回答预测窗口查询时所需要的平均节点访问次数。由于索引性能随着时间变化而下降，因此我们比较了不同索引方法在每隔 5k 次更新后的预测窗口查询和动态更新性能。

图 3(a)和 3(b)所示为分别固定预测范围窗口查询时间窗口 qTlen = 50 及查询空间窗口 qRlen = 1000 时，TPR*-tree、TPR-tree 与 ETPR-tree 回答查询所需要的索引节点访问代价。从中可以看出，基于 ETPR-tree 索引的预测范围查询算法具有最好的查询性能。基于 TPR*-tree 索引的查询算法次之，基于 TPR-tree 索引的查询算法最差。这是由于 TPR 索引节点矩形框由于考虑了移动对象速度分布，随着时间变化，MBR 区域之间重叠与 TPR*-tree 和 TPR-tree 相比非常少，因而具有最好的查询性能。TPR*-tree 和 TPR-tree 则会由于 MBR 之间的大量重叠导致查询搜索许多不必要的节点，从而影响了查询性能。

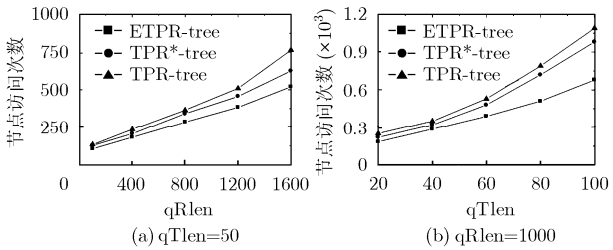

图 3 预测范围窗口查询性能比较

图 4(a)和 4(b)比较了 TPR*-tree、TPR-tree 与 ETPR-tree 索引随着时间(移动对象更新次数)变化更新和查询性能的比较。从图 4(a)可以看出，ETPR-tree 索引更新代价小于 TPR*-tree 和 TPR-tree 索引。这是由于 ETPR-tree 索引综合考虑了移动对象速度域和空间域分布，具有良好的紧致性，从而减少了标准 TPR*-tree 更新算法额外的搜索代价，另外由于 ETPR-tree 索引将速度桶指向的 TPR-tree 根节点存放在内存中，因此与 TPR-tree 和 TPR*-tree 相比减少了一些节点访问代价，其代价是增加了额外的内存空间。图 4(b)所示为固定查询时间窗口 qTlen = 50 及查询空间窗口 qRlen = 1000 时，在每隔 5k 次移动对象速度更新时回答同样一个预测范围窗口查询所需要的节点访问次数。可以看出，随着时间(移动对象更新次数)变化，TPR*-tree，TPR-tree 与 ETPR-tree 回答查询所需要的索引节点访问代价随

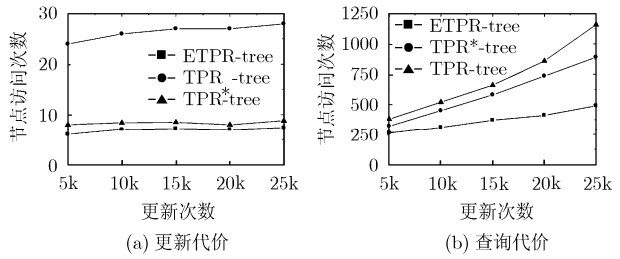

图 4 索引更新及查询性能比较

之增加，但是相比之下，由于 ETPR-tree 索引节点 MBR 随着时间变化扩张程度远小于 TPR*-tree、TPR-tree，因此随时间变化性能下降并不是非常明显，具有很好的稳定性。

5 结束语

本文提出了一种动态环境下新的移动对象混合索引(ETPR-tree)方法。ETPR-tree 索引综合考虑移动对象在速度域和空间域中的分布，首先在速度域中对移动对象集进行划分，根据速度矢量大小将移动对象映射到不同的速度桶中，每个速度桶中移动对象具有相近的速度矢量；对每个速度桶中的移动对象，则利用 TPR-tree 进行索引。实验表明，ETPR-tree 查询性能及动态更新性能均优于 TPR*-tree 等通用移动对象索引。

参 考 文 献

[1] Saltenis S, Jensen C S, and Leutenegger S, *et al.*. Indexing the positions of continuously moving objects. In: Proc. of the SIGMOD, New York, USA, 2000: 331–342.

[2] Mokbel M F, Ghanem T M, and Aref W G. Spatio-temporal access methods. *IEEE Data Engineering Bulletin*, 2003, 26(2): 40–49.

[3] Tao Yufei, Papadias D, and Sun Jimeng. The TPR*-Tree: An optimized spatio-temporal access method for predictive queries. In: Proc. of VLDB, Berlin, Germany, 2003: 790–801.

[4] Prabhakar S, Xia Y, and Kalashnikov D V, *et al.*. Query indexing and velocity constrained indexing: scalable techniques for continuous queries on moving objects. *IEEE Trans. on Computers*, 2002, 51(10): 1124–1140.

[5] Patel J M, Chen Yun, and Chakka V P. STRIPES: An efficient index for predicted trajectories. In: Proc. of SIGMOD, Paris, France, 2004: 637–646.

[6] Lin Bin and Su Jianwen. On bulk loading TPR-Tree. In: Proc. of the International Conference on Mobile Data Management (MDM), Berkeley, California, 2004: 114–124.

何凯涛： 男，1970 年生，博士生，研究领域为空间信息系统、空间数据库、空间信息服务.

唐　宇： 男，1977 年生，博士，副教授，主要研究领域为空间信息服务、网格技术、数据库技术.

廖　巍： 男，1980 年生，博士，研究领域为空间数据库、时空数据库.