第 35 卷 第 4 期 计 算 机 工 程 2009 年 2 月
Vol.35 No.4 Computer Engineering February 2009
·博士论文· 文章编号：1000—3428(2009)04—0023—02 文献标识码：A 中图分类号：TP311.12

TPR*树索引构建及其动态维护方法

廖 巍[1]，吴晓平[1]，严承华[1]，钟志农[2]
（1. 海军工程大学电子工程学院，武汉 430033；2. 国防科技大学电子科学与工程学院，长沙 410073）

摘 要：提出一种新的 TPR*树索引构建方法，在根节点层利用速度矢量对移动对象集进行划分，根据速度矢量的大小将移动对象聚集到不同子节点中，并逐层构建 TPR*树。在根节点层用溢出桶存储插入的移动对象记录，同时对 TPR*树索引进行批量插入更新，以减少其插入更新维护的代价。实验结果表明，该方法是可行的。

关键词：TPR*树；构建算法；溢出桶；插入延迟更新

Method for Index-building and Dynamic Maintenance of TPR*-tree

LIAO Wei[1], WU Xiao-ping[1], YAN Cheng-hua[1], ZHONG Zhi-nong[2]
(1. College of Electronic Engineering, Naval Engineering University, Wuhan 430033;
2. College of Electronic Science & Engineering, National University of Defense Technology, Changsha 410073)

【Abstract】 A novel method for index-building of TPR*-tree is proposed, which splits the moving object set at root node level by using velocity vector. According to the value of velocity vector, the moving objects are clustered into different sub-nodes, which then constructs the TPR*-tree. The records of moving objects are stored by using overflow bucket at root node level, and the index of TPR*-tree is inserted and updated with a batch, which decreases the cost of insertion maintenance. Experimental results show this method is feasible.

【Key words】 TPR*-tree; construction algorithm; overflow bucket; insertion delay update

1 概述

随着无线通信与 GPS 定位技术的发展，在许多领域需要对移动终端的位置进行追踪管理以提供相关的查询服务。由于移动终端的位置随时间不断变化，因此目前国内外大部分研究工作均采用移动对象数据库技术，即在数据库中记录移动对象的位置和运动特性信息，对其进行管理并处理用户提交的各种查询[1]。

近年来研究者提出的具有代表性的移动对象当前及未来位置索引方法主要包括：TPR-tree[2]及其变种 TPR*-tree[3]和 LUGrid 索引[4]等。其中，TPR*-tree 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。但由于 TPR*-tree 索引构建算法会将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页面中，因此造成同一页面中的移动对象速度分布不均匀，导致节点 MBR 随时间变化而急剧扩展，索引性能随时间变化而迅速下降。

本文综合考虑移动对象在速度域和空间域中的分布，提出一种新的 TPR*-tree 索引构建算法，并在 TPR*-tree 索引根节点层引入溢出桶以缓存插入的移动对象记录，提出插入延迟更新的动态维护算法，有效提高了 TPR*-tree 索引的查询和插入更新性能。

2 改进的 TPR*-tree 索引构建算法

文献[5]提出利用时空直方图由底向上构建 TPR-tree 索引的 HBU 批装载(bulk loading)技术。借鉴 HBU 算法思想，本文提出利用速度直方图对速度域进行划分的 TPR*-tree 索引构建算法，其主要步骤为：首先根据移动对象集合的大小和 TPR*-tree 节点扇出计算预期的根子节点项数目和各子节点速度矢量窗口范围，并对速度域进行划分；然后对每个根子节点中的移动对象集合利用 HBU 算法逐层构建 TPR*-tree 索引。

令 f 为 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 $M_i < n < M_{i+1}$，同时每个根子节点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 $K = \left\lceil n / M_i \right\rceil$，由于移动对象速度分布的不均匀性，在确定了根子节点数目之后，还必须确定每个根子节点的速度域范围。在此引入一种度量准则，即根子节点矩形包围框在索引生命期 L 之内所覆盖的区域面积最小，以选取最优的根子节点在速度域上的划分方案。

令 Vx, Vy 分别表示速度域在 2 个速度维上的划分数目，则有 $Vx = \left\lceil K / Vy \right\rceil$，给定移动对象集合在 2 个速度维上的直方图 Hx, Hy，期望的根子节点数目 K，算法 DeterminePartition 描述了速度域划分算法的具体步骤。

算法 1(DeterminePartition 算法)
输入 Hx, Hy, K
输出 Lx, Ly
BEGIN
For any integer combination (Vx, Vy) satisfyingVx=K/Vy
do

基金项目：国家“863”计划基金资助项目(2007AA12Z208)
作者简介：廖 巍(1980－)，男，讲师、博士，主研方向：移动对象数据库，数据库与网络安全；吴晓平，教授、博士；严承华、钟志农，副教授、博士
收稿日期：2008-08-20 **E-mail：**liaowei_2000@163.com

Lx = CalculateExtent(Vx, Hx)

Ly= CalculateExtent(Vy, Hy)

$cost \leftarrow \sum_{i=1}^{Vx}\int_0^L Lx[i]tdt + \sum_{i=1}^{Vy}\int_0^L Ly[i]tdt$

If cost is better valued,

return current Lx, Ly values

End If

End For

End

令 *Num* 为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*，中间变量 *NumPerPartition* 表示每个根子节点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)

输入 *Num*, *H*

输出 *L*

BEGIN

$NumPerPartition \leftarrow N/Num$, $j \leftarrow 0$;

For i from 0 to Num do

Starting from the jth bucket, count the H[j].num into the partition until their sum = NumPerPartition;

L[i].extent ← sum of the H[j].extent

Adjust j to corresponding value for next

End For

return L

END

3 插入算法

目前的 TPR*-tree 索引插入算法均采用独立插入更新机制，当插入新的移动对象位置记录时，插入算法立即扫描 TPR*-tree 索引，在合适的叶节点中插入记录。为提高 TPR*-tree 索引的插入更新性能，本文引入溢出桶内存结构，并采用插入延迟更新策略，即把插入的移动对象记录缓存在溢出桶中，只有当溢出桶满时才将缓存的移动对象记录批处理地插入到合适的 TPR*-tree 索引叶节点中。

当用户向服务器提交插入新的移动对象记录时，插入算法先在溢出桶中插入新的记录，而并不立即更新 TPR*-tree 索引，只有当溢出桶超出内存阈值时，才将溢出桶中的所有移动对象记录更新到磁盘上去。溢出桶中的移动对象记录依照 *OID* 进行排序，在处理用户提交的查询时，查询算法先在基于内存的溢出桶中进行搜索，然后在 TPR*-tree 索引中利用现有的查询处理算法，即可保证查询结果的正确性。

下面给出插入算法 Insert 的具体步骤：

算法 3(Insert 算法)

输入 *bucket*

输出 TPR*-tree

BEGIN

For all entries in *bucket*, search the TPR*-tree from root and decide the child nodes that contain the moving objects in entries;

For entries that don't lie in any child node, choose the nodes that have the least expansion;

For each node do

Insert the entries into corresponding nodes until the leaf level;

End For

END

4 实验

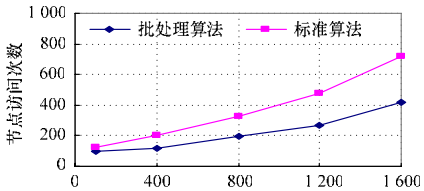
4.1 实验内容与设置

本文利用基于道路网络的移动对象产生器[6]生成移动对象数据集和查询集合。产生器的输入为德国城市 Oldenburg 道路图，输出为在道路网络上运动的移动对象集合。数据集大小为 100 KB，移动对象用二维空间点坐标表示。

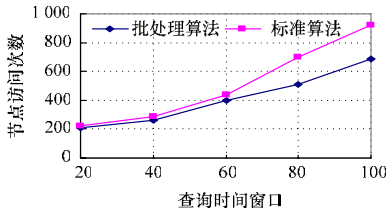
实验数据集利用 TPR*-tree 进行索引，TPR*-tree 页面大小设置为 1 KB，中间节点扇出为 31，叶节点中可以保存大约 54 个移动点对象，索引树高度为 4 层。使用的页面缓存大小为 50 KB，即 50 个缓存页面，并使用最近使用(LRU)缓存替代策略。查询性能用处理 100 个查询所需要的平均节点访问次数来衡量。实验运行在 P4 3.0 GHz 处理器，256 MB 内存和 7 200 RPM 硬盘的装有 Windows 系统的计算环境下。

4.2 实验结果分析

图 1(a)和图 1(b)分别为固定预测范围窗口且查询时间窗口为 50、查询空间窗口为 1 000 时，TPR*-tree 采用标准构建和插入算法与采用本文构建和插入延迟更新算法在处理查询时所需要的索引节点访问代价比较。



(a)查询时间窗口为 50

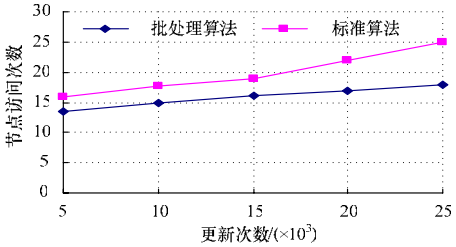


(b)查询空间窗口为 1 000

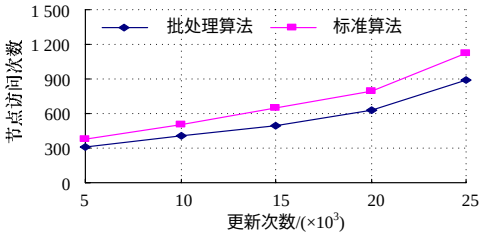
图 1　预测范围窗口查询性能比较

可以看出，本文构建和插入延迟算法的 TPR*-tree 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 区域之间重叠较少，因此具有较好的查询性能。

图 2 比较了 TPR*-tree 采用标准构建和插入算法与采用本文构建和插入延迟更新算法在随着时间(移动对象更新次数)变化的更新和查询性能。



(a)更新代价



(b)查询代价

图 2　索引更新及查询性能比较　(下转第 27 页)