

# 城市路网上动态迁移的移动对象索引结构

张郁彬<sup>1,2</sup>, 张深深<sup>3</sup>, 孟旭东<sup>1,2</sup>

- (1. 宽带无线通信与传感网技术教育部重点实验室, 江苏 南京 210003;  
2. 南京邮电大学 江苏省电信网络融合实验室, 江苏 南京 210003;  
3. 南京邮电大学 计算机学院, 江苏 南京 210003)

**摘要:**移动对象索引技术是有效管理海量移动对象数据的支撑。目前的移动对象索引方法如 FNR-tree、NDTR-tree 等均采用基于磁盘的索引结构,忽略了移动对象在城市道路上密度分布不均衡的情况,因此在移动对象位置更新频繁时,该类方法的性能会严重下降。针对以上不足,提出一个针对城市路网上热点区域变化进行内外存索引迁移的结构(hot-spots dynamic migration index, HDMI)。HDMI 是双层索引结构,上层采用 R\*-tree 对路网数据进行管理,下层采用 R-tree 群对实时更新的移动对象运动信息进行索引。HDMI 采用基于内存的索引结构管理热点区域和该区域中的移动对象,针对非热点区域和其中的移动对象则采用基于外存的索引结构来进行管理。HDMI 能够根据道路上车辆密度的变化进行内外存索引的迁移,从而在有限的内存条件下,保障索引更新和查询的性能。实验结果表明, HDMI 于 NDTR-tree 相比,不仅减少了索引建立和维护的 I/O 代价,而且提高了时空窗口查询处理性能。

**关键词:**城市路网;热点区域;移动对象;动态迁移;索引技术

中图分类号: TP311

文献标识码: A

文章编号: 1673-629X(2018)03-0047-07

doi: 10.3969/j.issn.1673-629X.2018.03.010

## A Moving Object Index Structure of Dynamic Migration on Urban Road Network

ZHANG Yu-bin<sup>1,2</sup>, ZHANG Shen-shen<sup>3</sup>, MENG Xu-dong<sup>1,2</sup>

- (1. Key Lab of Broadband Wireless Communication and Sensor Network of Ministry of Education, Nanjing 210003, China;  
2. Jiangsu Telecom Network Fusion Laboratory, Nanjing University of Posts and Telecommunications, Nanjing 210003, China;  
3. School of Computer, Nanjing University of Posts and Telecommunications, Nanjing 210003, China)

**Abstract:** Moving object indexing technology is the support for managing the massive moving object data effectively. The current moving object indexing methods such as FNR-tree and NDTR-tree adopt the disk-based index structure, and ignore the situation that the density distribution of the moving objects is reduced on the urban road, so when the moving object location updates frequently, the performance of the index method will be a serious decline. Aiming at the shortcomings of the above methods, we propose a structure for the internal and external index migration of urban hotspots (hot-spots dynamic migration index, HDMI). It is a double-layer index structure, the upper layer uses R\*-tree to manage the network data, and the lower layer uses the R-tree group to index the moving object motion information updated in real time. HDMI uses a memory-based index structure to manage hotspots and moving objects in that area, and for non-hotspot areas and moving objects in them, they are managed using an external-based indexing structure. It can carry on the migration of the internal and external index based on the vehicle density changes on the road, which in limited memory conditions, to protect the index update and query performance. Experiments show that HDMI reduces the I/O cost of index creation and maintenance compared to NDTR-tree, and improves the performance of space-time window query processing.

**Key words:** urban road network; hotspot area; moving object; dynamic migration; indexing technology

收稿日期: 2017-04-13

修回日期: 2017-08-15

网络出版时间: 2017-12-05

基金项目: 国家科技重大专项(2012ZX03001008-003, 2011ZX03005-004-03); 国家“973”重点基础研究发展计划项目(2013CB329005)

作者简介: 张郁彬(1993-), 女, 硕士研究生, CCF 会员(74427G), 研究方向为通信监管与优化、数据管理; 孟旭东, 教授, 研究方向为电信网络和 IP 网络的交换、异构网络集成及业务融合、未来互联网体系结构、网络计算与分布式处理。

网络出版地址: <http://cnki.net/kcms/detail/61.1450.TP.20171205.0904.034.html>

## 0 引言

近年来 WIFI、4G、5G 等无线通信技术得到了快速发展,智能手机、PDA 等专业手持、车载设备也渐渐普及。与此同时,基于卫星的无线定位系统也在迅速发展升级,如美国的 GPS 系统、欧洲的伽利略系统以及国内自主设计的北斗系统等都在不断提高着定位精度。移动通信的发展、智能终端的普及以及定位精度的提高使得一些与位置相关的服务成为可能<sup>[1]</sup>。

基于位置<sup>[2-3]</sup>的应用都需要管理海量的移动对象位置信息,如何有效地对这些位置数据进行管理成为急需解决的问题。传统的外存索引无法处理如此海量的位置信息数据,是目前大多数移动对象索引结构急需解决的理论技术难题。目前移动对象索引策略分为两大类:一类是磁盘索引,一类是内存索引。前者将索引存储在磁盘上,无法支持频繁、巨量的更新和查询。后一种策略将索引放在内存中,以加快处理速度,降低查询响应时间,但目前这种索引受到内存容量的限制。

因此针对城市路网上移动对象在道路分布上的不均衡,设计了内外存迁移的索引结构(hot-spots dynamic migration index, HDMI)。该索引结构根据道路车辆密度的动态变化自适应地实现内外存索引迁移,以适应负载增长,降低索引维护代价。

## 1 相关工作

索引技术<sup>[4]</sup>能支持对移动对象数据的高效检索和管理,通过多年的实验和研究,国内外相关人员在移动对象索引结构研究领域取得了一定成果。大致可以将移动对象索引分为两类,一类是根据数据的对象分布对索引节点进行划分,以 R-tree 及其变体 R\*-tree、扩展树构建的索引结构为代表。MV3R-tree 用 R-tree 及其变体来对历史轨迹进行索引。TPR-tree<sup>[5]</sup>可以索引移动对象的当前和未来位置,其每个节点用一个时间相关的速度矢量对空间矩形进行界定,如果移动对象长时间不运动,则会降低索引很大的性能。REXP-tree<sup>[6]</sup>用来索引移动对象的现在和将来位置,其索引节点中明确标识运动矢量失效时间,失效后按照一定策略进行删除,保证索引的紧凑和有效。TPR-tree 的变体还有 TPR\*-tree<sup>[7]</sup>、HTPR-tree<sup>[8]</sup>等,TPR\*-tree 采用了不同的索引节点维护算法,从而更为紧凑。另一类索引根据空间划分对索引节点进行组织,大多是以格栅及其变体为基础构建的索引结构。ST2B-tree 是一种格栅和 B-tree 相结合的索引结构,它将空间中移动对象根据密度分布、采用不同粒度的格栅进行管理,以适应负载变化迅速的情况。DSTR-tree<sup>[9]</sup>是一种网络受限移动对象的动态概略化轨迹索引。该索引结构将索引空间划分成等距格栅,通过格栅单元对每

一条移动对象轨迹进行概略化,由于概略化轨迹的粒度大大粗于原始轨迹,从而显著地降低索引更新代价。以上索引结构大多是在移动对象自由运动的情况下构建的,针对被限制在城市路网中的移动对象的研究则不多。

目前,国内外研究学者提出了一些基于受限网络的移动对象索引结构,例如 Frentzos 等提出的双层索引结构 FNR-tree<sup>[10]</sup>,其在一定程度上弥补了基于路网的移动对象索引技术的不足。FNR-tree 的上层索引结构 2DR-tree 用来索引路段信息,下层索引结构 1DR-tree 用来管理移动对象的轨迹,2DR-tree 中每个叶子节点都对应着一个下层 1DR-tree。但 FNR-tree 存在几点不足:

(1) FNR-tree 难以高效地支持移动对象历史轨迹查询和时间道路查询。

(2) FNR-tree 沿用了 R-tree 的自顶向下搜索策略,在精确查询时,可能要从顶部同时向多个叶子方向遍历,造成大量重复、冗余的查找路径,降低了索引结构的性能。

(3) FNR-tree 将移动对象出现在路网每个位置的机率视为均等。但是现实路网中,不同道路的交通繁忙程度是不同的。

针对 FNR-tree 的不足,Almeida 等提出了索引结构 MON-tree<sup>[11]</sup>,其对道路网络的索引不再基于直线边,而是基于折线道路,降低了移动对象跨越不同边时的代价。MON-tree 的缺陷在于对交通网络的索引基于道路,会造成节点的重合。

NDTR-tree<sup>[12]</sup>是丁治明教授提出的基于路网的移动对象索引结构,其为 UTR-tree 的改进索引结构<sup>[13]</sup>,是目前最具有代表性的路网移动对象索引结构。NDTR-tree 是双层的 R-tree 索引结构,上层索引为一个 R-tree,对路网的原子路段建立索引;下层为一组独立的 R-tree,每个 R-tree 与一条道路相对应,下层 R-tree 用于对在该条道路上提交的移动对象轨迹片段进行索引。NDTR-tree 存在如下不足:

(1) 不支持对移动对象轨迹的高效查询,因为需要遍历整个下层 R-tree 群,造成极大的开销。

(2) 索引维护的代价较高,每当移动对象发生位置更新请求时,索引进行一次维护,需要一次插入和删除操作。

(3) 在实际路网中,道路相交部分较多,采用结构组织索引,由于算法在节点插入时只考虑边际面积增加的情况,会产生很多索引空间的重叠,不利于查询。

上述集中式磁盘时空索引<sup>[11]</sup>在移动对象数目巨大、更新频繁的情况下难以保证查询的实时响应和精度;基于城市路网的移动对象索引技术大多没有考虑

移动对象在路网上分布不均衡的情况,当路网道路中车辆变化差异很大时,很容易降低整个索引结构的性能。

## 2 基于相对位置的移动对象模型

本节将介绍 HDMI 采用的城市路网和移动对象模型和主要概念,并给出形式化定义。

定义 1:路网  $Network = (Edge, Node)$ , 其中  $Edge$  为路段的集合,每一条路段对应一条封闭的线段; $Node$  为道路节点的集合,每个节点对应地图上的一个经纬度  $(x, y)$ 。

定义 2:道路节点  $Node = (nodeId, x, y)$ , 其中  $nodeId$  为道路节点的标识,  $x$  和  $y$  为节点在地图空间平面上的经纬度。

定义 3:道路  $Road = (roadId, Edge, Node, len)$ , 其中  $roadId$  为该条道路的标识,  $Edge$  为此道路中所有路段组成的集合,  $Node$  为此道路上所有节点组成的集合,  $len$  为此道路的长度。

定义 4:路段  $Edge = (edgeId, roadId, node_s, node_e, len, weight, position_{edge})$ , 其中路段  $Edge$  就是从路段起点到终点不含有任何道路交点的道路片段,  $edgeId$  为该路段在道路中的标识,  $roadId$  为该路段所在道路的标识,  $node_s$  和  $node_e$  分别表示路段的起点和终点在二维空间的经纬度,  $len$  为此路段的长度,  $weight$  为该路段占其所属道路的比例, 计算方法为  $weight = Edge.len / Road.len$  ( $Road$  为路段  $Edge$  所属的道路),  $position_{edge}$  ( $0 \sim 1$ ) 为该路段的起点在道路的相对位置。

文献[14]提到的移动对象数据模型是文中移动对象模型的基础,该模型将移动对象的位置更新信息  $(x, y, t)$ , 通过式(1)转化成可以用  $R$ -tree 存储的二维信息  $(position, t)$ 。该模型通过将二维空间降低到一维空间来满足构建  $R$ -tree 的条件, 主要概念及形式化定义如下:

已知移动对象更新位置坐标  $(x, y)$  到路段起点  $node_s$  的距离为  $s$ , 该路段长度为  $len$ , 那么  $position$  是移动对象当前位置整条道路的比例, 计算方法如下:

$$position = position_s + weight \times (s / len) \quad (1)$$

其中,  $position_s$  为移动对象在道路中的初始位置;  $weight$  为所属路段占其当前道路的比例。

定义 5:移动对象位置更新信息  $Move = (moveId, t, x, y)$ , 其中  $moveId$  为移动对象的标识,  $t$  为移动对象位置更新时刻,  $x$  和  $y$  为移动对象所处经度和纬度。

定义 6:移动对象运行矢量  $Moving = (moveId, t, roadId, position)$ , 其中  $moveId$  为移动对象的标识,  $t$  为提交该运行矢量的时刻,  $roadId$  为该移动对象所处道路的唯一标识,  $position \in (0, 1)$  是该移动对象所处道

路的相对比例位置。  $roadId$  和  $position$  通过定义 5 中的  $Move$  和路段信息  $Edge$  计算得出。

定义 7:移动对象链表轨迹单元  $MU = (U, uid)$ , 多个连续的移动对象轨迹单元组成了移动对象轨迹。轨迹单元  $U$  是两个运行矢量之间的线段,  $U$  表示为  $U(Moving_s, Moving_e)$  或者  $U(Moving_a)$ ,  $Moving_s$  和  $Moving_e$  分别表示运行矢量的起点和终点, 活动运行矢量用  $Moving_a$  表示, 其对应一条射线,  $uid$  则用来记录轨迹单元在下层  $R$ -tree 中的位置。

## 3 HDMI 的数据结构

城市路网上的移动对象管理呈现三个特征: (1) 移动对象在路网上分布不平衡, 导致不同路段上移动对象位置更新和查询覆盖的不平衡; (2) 随着时间的变换, 热点区域也在不断变化; (3) 相比于规模不断增长的移动对象, 可用的计算机内存总是有限的。基于这种认识, 文中提出了移动对象索引结构 HDMI, 该结构根据道路车辆密度的动态变化自适应地实现内外存索引迁移, 大幅度降低 I/O 操作, 提高查询效率。

### 3.1 热点区域

在实际路网中, 不同道路上的移动对象分布的不均匀导致相应道路上的位置更新频率差异很大; 针对不同道路轨迹的查询频率差异明显。如果将这些被频繁访问的道路和移动对象存储在内存中, 则能大幅度提升移动对象索引结构的性能。正是基于此, HDMI 索引结构针对动态变化的热点区域引入基于内存的索引结构, 以减少索引结构的磁盘访问次数, 最大限度地提升索引结构的性能。

定义 8:假设一条道路的长度为  $Road.len$ , 该条道路上的车辆数量为  $Num$ , 如果道路内车辆的密度(单位道路内的车辆数)超过阈值, 则该路段为热点道路, 热点道路组成的区域称为热点区域。

$$\rho = \frac{Num}{Road.len} \quad (2)$$

随着移动对象在路网分布的不断变化, 组成热点区域的热点道路也在动态变化, 内外存索引根据热点区域的变化进行迁移更新, 保证新增的热点区域和移动对象能被内存索引管理, 将由于车辆密度降低导致变为非热点的区域和移动对象迁移到外存, 以此来保证繁忙道路索引常驻系统内存中, 以响应该区域内频繁的移动对象位置更新。

### 3.2 上层城市路网索引

HDMI 的上层索引结构是针对城市路网构建的  $R^*$ -tree 索引, 用于索引城市路网中的路段, 针对热点区域构建基于内存的  $R^*$ -tree 索引, 非热点区域构建磁盘  $R^*$ -tree 索引文件放入基于外存的磁盘文件。

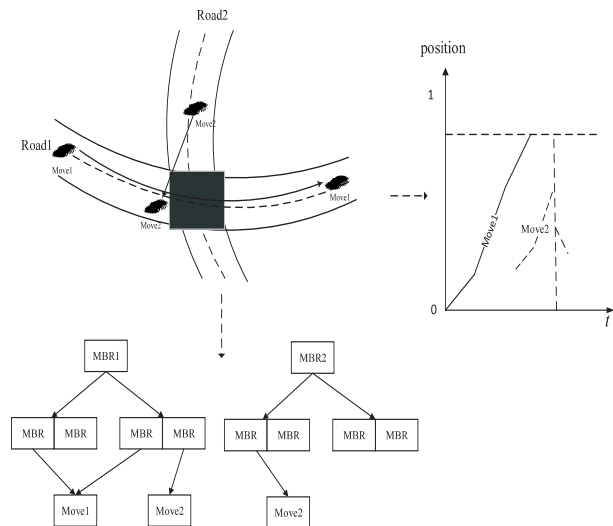


其叶子节点的数据结构为  $(MBR, nodeId, Edge)$ , 其中  $MBR$  表示包含该路段的最小边界矩形,  $nodeId$  为该叶子节点在索引中的标识,  $Edge$  为该路段信息; 中间节点为  $(MBR', nodeId', tree)$ , 其中  $MBR'$  表示包含下层节点的  $MBR$ ,  $tree$  表示指向下层节点的指针,  $nodeId'$  表示该中间节点在索引中的标识。最小矩形框包含着城市路网的每个路段, 所有最小边界矩形一步一步迭代后, 形成最大的  $MBR$ , 最终城市路网被两个最大的  $MBR$  包含, 分别包含热点区域和非热点区域。

### 3.3 下层移动对象轨迹索引

HDMI 下层索引是对应于道路的  $R$ -tree 索引群, 热点区域的道路对应内存  $R$ -tree 群, 非热点区域道路对应外存  $R$ -tree 群, 它们基于位置和时间 ( $position \times t$ ) 平面。通过下层索引管理移动对象轨迹单元, 动态地维护移动对象位置更新信息。下层叶子节点的数据结构为  $(MBR, id, Moving_{start}, Moving_{end}, moveId)$ , 其中  $MBR$  为包含该轨迹的最小边界矩形,  $Moving_{start}$  表示轨迹单元的开始运行矢量,  $Moving_{end}$  表示轨迹单元的结束运行矢量, 若为活动轨迹单元, 则  $Moving_{end}$  为空,  $id$  表示该叶子节点在索引中的标识,  $moveId$  表示移动对象的唯一标识。

中间节点的数据结构为  $(MBR', id', tree)$ , 其中  $MBR'$  为包含下层节点的  $MBR$ ,  $id'$  表示中间节点在索引中的标识,  $tree$  指向下层节点。下层结构如图 1 所示, 下层  $R$ -tree 群管理基于  $position-t$  平面的移动对象轨迹片段, 例如移动对象  $Move1$  只在道路  $Road1$  上运动, 那么它的所有轨迹单元被道路  $Road1$  的  $R$ -tree1 ( $MBR1$ ) 管理。图中的方框表示移动对象  $Move2$  的活动轨迹单元, 其从道路  $Road2$  运动到道路  $Road1$ , 那么它的轨迹单元将被两个道路树  $R$ -tree1 ( $MBR1$ ) 和  $R$ -tree2 ( $MBR2$ ) 管理。



2:  $n, 2$  代表上层索引由 2 个  $R^*$ -tree 索引构成,  $n$  代表下层索引由对应  $n$  条道路的  $R$ -tree 群构成。每个上层索引叶子节点都指向下层索引群中的某个树, 当然不同叶子节点也可能指向同一条道路。比如新模范马路包含四个路段, 则上层叶子节点中有四个记录均指向下层索引中属于新模范马路的  $R$ -tree (记录中  $Edge$  信息指向下层道路索引)。

## 4 基于 HDMI 索引结构的操作算法

### 4.1 上层索引结构的更新算法

算法 1: HDMI 上层索引的更新算法。

输入: 批量更新的移动对象更新信息  $Move_{i=1}^n$ ;

输出: 自适应的上层路网索引。

```

1  memoryRoad ← the Road in the memory
2  for each  $Move_{i=1}^n = (moveId, t, x, y)$ 
3  (roadId, Num++) ← search the roadId of movei from the
    $R^*$ -tree;
4  end
5  foreach  $Road_i = (roadId, Edge, Node, len, Num)$ 
6  density ←  $\frac{Num}{Road.len}$ ;
7  if (density ≥ P) then
8  if (roadId not in the memoryRoad) then
9  insert theRoadi into memoryR $^*$ -tree;
10 insert the roadId into ArraySet<integer> memoryRoad;
11 else
12 if (roadId in the memoryRoad) then
13 insert theRoadi into diskR $^*$ -tree;
14 remove the roadId from the memoryRoad;
15 end

```

算法 1 的主要步骤如下:

(1) 根据每个移动对象位置更新信息查询上层  $R^*$ -tree 索引, 计算每条道路中移动对象的个数 (第 2~4 行), 计算每条道路的车辆密度 (第 6 行);

(2) 道路车辆密度大于等于阈值  $P$ , 并且该道路不在内存中, 则将该道路更新到内存索引中, 并将这条道路的  $roadId$  插入到存储热点道路的数组  $memoryRoad$  中 (第 8~10 行);

(3) 道路车辆密度小于规定的阈值  $P$  并且该道路在内存中, 则将该道路更新到外存索引, 并将这条道路的  $roadId$  从存储热点道路的数组  $memoryRoad$  移除 (第 12~14 行)。

时间复杂度参数设置: 该上层  $R^*$ -tree 索引有  $N$  个索引记录,  $M$  为  $R^*$ -tree 一个节点中的最大条目数, 热点区域与整个地图的比例是  $1:\lambda$ ,  $\lambda \in 1+\infty$  根据车辆密度动态变化。算法 1 中 HDMI 上层路网索引的更新需要内外存索引的迁移来完成, 有  $p$  条热点道路由于移动对象密度的变化变成非热点道路, 该过程

图 1 HDMI 下层索引结构以及移动对象轨迹

HDMI 上下层索引之间的  $R$ -tree 的数量关系是

的时间复杂度为  $O(p \times \log_M \lambda - 1N/\lambda)$ 。

## 4.2 下层 R-tree 群索引的建立和维护算法

下层动态索引的建立和维护的整个过程如算法 2 所示,其主要步骤为:

(1) 根据移动对象位置更新信息 Move 查找上层索引,确定移动对象所在道路 roadId 和相对位置 position(第 2 行)。

(2) 根据 roadId 和 position 生成移动对象运行矢量 Moving(moveId, t, roadId, position)(第 3 行)。

(3) 下层索引是基于 position $\times t$  平面的 R-tree(第 4 行),完成了下层 R-tree 索引构建所需的数据结构 rectangle(position1, position2,  $t_1, t_2$ ),运用基于  $x \times y$  平面构建的 R-tree 的思想对其进行重构。通过 searchRoadHash() 查询到道路 roadId 所对应的 R-tree。

(4) 通过 TrajectoryMap 表查询 moveId 所对应的双向链表 MU(第 5 行);由 Moving 和 uid 生成新的移动轨迹单元 MUnew(第 6 行),并将新生成的轨迹单元插入到 searchTrajectoryMap 中(第 7 行)。

(5) 如果链表为空,并且移动对象所处道路为热点区域道路,将 Moving 插入到热点区域道路对应的道路的内存 R-tree 中(第 9~10 行);如果移动对象所处道路为非热点区域道路,将 Moving 插入到非热点区域道路对应的道路的外存 R-tree 中(第 12 行)。

(6) 如果链表不为空,删除该移动对象上一时刻在 R-tree 对应的数据(第 14~15 行),并根据移动对象是否处在热点区域分别插入到内外存 R-tree 索引中(第 16~20 行)。

算法 2: HDMI 下层索引的建立和维护算法。

输入: 批量更新的移动对象更新信息 Move $_{i=1}^n$ , 热点区域道路表 memoryRoad;

输出: 更新后的 HDMI。

```

1  foreach Move $_{i=1}^n$  = (moveId, t, x, y)
2  (roadId, position) ← traverse the upper R* -tree to find the roadId and position;
3  Moving ← (roadId, x, y, t, position);
4  Rtree ← searchRoadHash(roadId);
5  MU ← searchTrajectoryMap(moveId);
6  MU $_{new}$  ← (Moving, Rtree. uid);
7  insert the MU $_{new}$  into the TrajectoryMap;
8  if (MU = null) then
9  if (roadId in the memoryRoad) then
10 insert the Moving into memory_RoadId_R-tree;
11 else
12 insert the Moving into disk_RoadId_R-tree;
13 else
14 U(Moving, uid) ← MU. last, last_R-tree ← search(uid);
15 delete U(Moving, uid) from last_R-tree;
```

```

16 if (roadId in the memoryRoad) then
17 insert the Moving into memory_RoadId_R-tree;
18 else
19 insert the Moving into disk_RoadId_R-tree;
20 end
```

HDMI 下层 R-tree 索引有  $N$  个索引记录,  $M$  为 R-tree 一个节点中的最大条目数。有  $n$  个移动对象进行位置更新, 其中有  $m$  个位于热点区域的道路, 由于热点道路中移动对象的更新在内存完成, 不考虑更新代价, 所以算法 2 的时间复杂度为  $O(n - m \times \log_M N)$ 。

## 4.3 时空窗口查询

时空窗口查询是指查询一段时间  $(t_1, t_2)$  内经过地图上指定二维空间  $(x_1, x_2, y_1, y_2)$  的移动对象, 时空窗口查询也表示为  $(\Delta x, \Delta y, \Delta t)$  的形式。如查询 2008-02-02 15:15:04 到 2008-02-02 17:40:02 间, 经过南京市三牌楼区域的车辆。具体查询过程为:

(1) 将窗口  $(x_1, x_2, y_1, y_2)$  作为上层范围查询条件, 搜索上层内外存路网 R\* -tree, 求得查询窗口与路段的交点集合  $(Edge_i, x_i, y_i)_{i=1}^n$ 。每条路段可能不止一个交点(第 1 行);

(2) 遍历每个交点  $(Edge_i, x_i, y_i)$ , 计算交点在道路的唯一标识 roadId $_i$  和相对位置 position(第 3 行);

(3) 针对每个  $(roadId_i, position_i)$ , 根据 Road\_Hash 表定位下层所对应的内外存 R-tree(第 4 行);

(4) 求出  $(position, \Delta t)$  范围内移动对象节点 MU $_i$ (第 5 行), 遍历 MU $_i$  得到其中的 moveId(第 6~9 行)。

算法 3: 时空窗口查询。

输入: 查询区域, 时间范围  $(t_1, t_2)$ , 道路索引 Road\_Hash;

输出: 移动对象标识集合 result。

```

1 (Edge $_i, x_i, y_i$ ) $_{i=1}^n$  ← 查询上层 R* -tree 在  $(\Delta x, \Delta y)$  内或者与该查询区域相交的路段;
2 foreach (Edge $_i, x_i, y_i$ ) $_{i=1}^n$ 
3 roadId $_i, position_i$  ← 根据 (Edge $_i, x_i, y_i$ ) 求得交点的位置和道路标识;
4 roadR-tree ← searchRoadHash(roadId $_i$ );
5 MU $_i$  ← 查询 roadR-tree 中 (position $_i, \Delta t$ ) 的节点;
6 foreach MU $_i$  $_{i=1}^n$ 
7 moveId ← 从 MU 取得移动对象的标识;
8 result ← 添加 moveId 到结果集;
9 end
10 end
```

通过上层 R\* -tree 索引查询到区域与路段有  $n$  个交点, 其中有  $m$  个位于热点道路, 相关该阶段的时间复杂度为  $O(n \times \log_M \lambda - 1N/\lambda)$ , 参数请参照算法 1; 通过下层 R-tree 群索引查询到符合条件移动对象, 该阶段的时间复杂度为  $O((n - m) \times \log_M N)$ , 所以算法 2 的时间复杂度分析为  $O((n - m) \times \log_M \lambda - 1N/\lambda + (n - m) \log_M N)$ 。

5 实验与性能分析

5.1 实验数据集

实验采用的城市路网数据集是从网站 <http://www.openstreetmap.org/>下载的北京真实 OpenStreet-Map(OSM)地图,如表 1 所示,其中包含北京城区内 57 036 条道路和 317 551 个交点。

表 1 北京城市路网数据

经度	纬度	道路数	节点数
116.080 74,116.766 48	39.681 25,40.205 05	57 036	317 551

移动对象数据集是北京市 2008-02-02 12:00:00 到 2008-02-08 19:00:00 时间段内 10 000 台出租车行驶轨迹。在这个数据集中一共有 15 000 000 个点,轨迹行驶的距离有 9 000 000 km。

5.2 索引建立和维护代价比较分析

索引建立和维护时的 I/O 代价是评判一个索引结构性能的重要指标,所以分别计算在 1 000、2 000、3 000、5 000 和 10 000 个北京出租车数量的情况下 NDTR-tree 和 HDMI 建立和维护索引过程中 I/O 的次数,结果如图 2 所示。

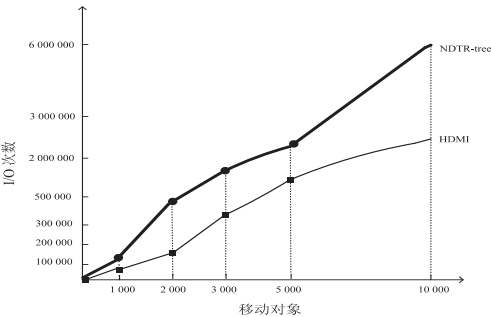


图 2 I/O 次数比较

从图 2 可以看到,随着移动对象数量的不断递增,NDTR-tree 磁盘节点的存取 I/O 次数迅速增加,HD-MI 索引结构随着移动对象数据的增加磁盘节点存取次数变化相对平稳,所以 HDMI 索引的更新性能优于 NDTR-tree。

以下三个方面导致 HDMI 索引与 NDTR-tree 更新性能存在如此大的差异:

(1) 针对热点区域行驶的移动对象,NDTR-tree 每次更新都要访问磁盘的路网 R\*-tree 和移动对象 R-tree,这就会引起很高的索引更新 I/O,同时导致查询响应速度缓慢,影响了系统的整体性能。而 HDMI 将热点区域的 R\*-tree 放入内存,在更新和查询时可以直接从内存中取得,可以快速响应该区域车辆的更新操作,很好解决了大批量移动对象在繁忙道路定位的瓶颈,减少了大量的 I/O 操作。

(2) 随着移动对象的逐渐增加,NDTR-tree 的下层移动对象索引结构效率会迅速降低。因为,其下层 R-tree 针对每个移动对象的位置更新都需要进行索引

的更新,同时很多长时间不运动的移动对象占据了索引数据项。而 HDMI 采用了哈希表和双向链表保存移动对象轨迹信息,当需要跟踪移动对象时仅需遍历链表尾节点来获取其历史轨迹信息,不用再遍历该移动对象所在的 R-tree,提高了查询效率,减少了 I/O 开销。

(3) HDMI 将需要插入、更新和删除的移动对象保存在缓冲区中,当缓冲区中移动对象数目超过规定阈值时或按照一定的时间间隔,将缓冲区中的移动对象批量更新到索引结构中,减少了索引结构频繁的 I/O 操作。

5.3 时空窗口查询性能分析

(1) 热点区域变化。

在移动对象数量分别为 1 000、2 000、3 000、5 000、10 000,占据地图 15% 查询窗口的情况下并且分别以地图的 5%、10%、15%、20% 作为热点区域进行范围查询实验,对比查询时所耗费时间。当索引结构 HDMI 自适应到热点区域占路网 5%、10%、15%、20% 的时候,暂停索引的更新,同时保证查询窗口经过热点区域。图 3 为 HDMI 索引结构的时空范围查询基于不同的热点区域时间开销。

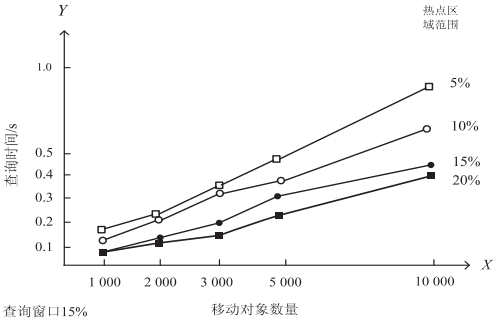


图 3 时空窗口查询

两次实验证明 HDMI 索引的窗口查询性能在引入热点区域后得到了很大提升,不过受热点区域的影响较大,热点区域越大查询性能越高,因为热点区域索引结构是内存索引,比外存索引查询速度快很多。但热点区域太大会占用大量内存,并且内外存索引迁移更新时有一定的代价,所以选择合适范围的热点区域对于系统的整体性能至关重要。

(2) 查询窗口变化。

在移动对象数量分别为 1 000、2 000、3 000、5 000、10 000 并且分别以地图的 10%、15%、20%、30% 的面积作为查询窗口进行范围查询实验,对比查询时所耗费时间。当索引结构 HDMI 自适应到热点区域占路网 10% 的时候,暂停索引的更新,同时根据热点区域的范围保证查询窗口经过热点区域。图 4 显示了 NDTR-tree、HDMI 索引结构基于不同的查询窗口的时空窗口查询的时间开销代价。

从图 4 中可以看出,NDTR-tree 和 HDMI 的窗口

查询性能基本不受移动对象数量的影响,主要还是取决于上层路网索引结构。 $R^*$ -tree 的覆盖度和重叠度较低,所以查询效率相比  $R$ -tree 高。由于 HDMI 采用

了  $R^*$ -tree 作为上层路网索引结构,而 NDTR-tree 的上层路网索引采用传统的  $R$ -tree,因此 HDMI 在窗口查询性能方面比 NDTR-tree 高出许多。

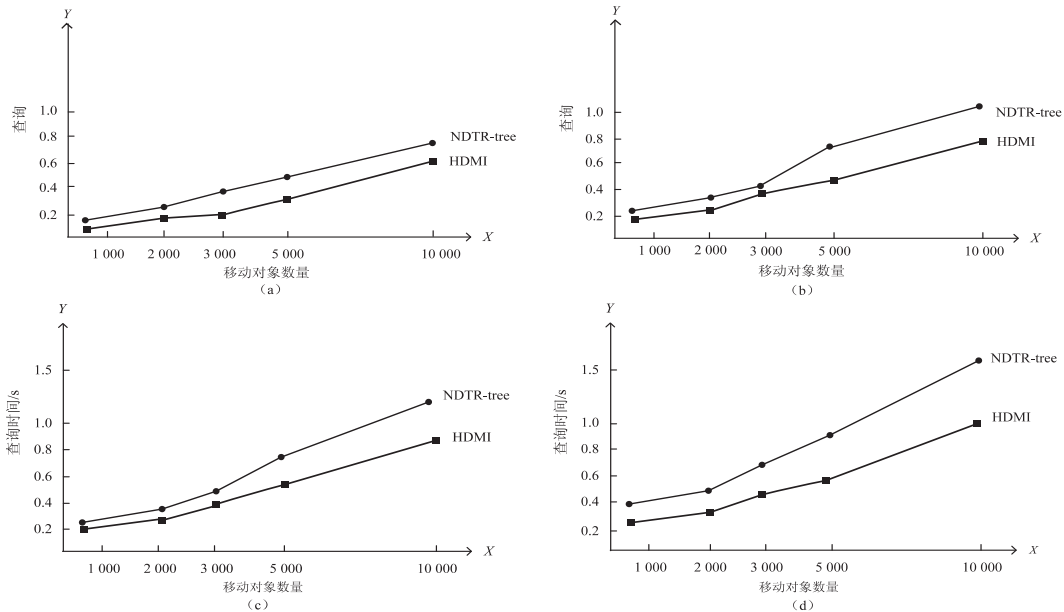


图4 时空窗口查询

6 结束语

针对 NDTR-tree 等索引结构在索引维护和轨迹查询方面的不足,提出了一种城市路网上动态迁移的索引结构 HDMI。内存索引管理热点区域及其中的移动对象,非热点区域及其中的移动对象则由外存索引管理,内外存索引根据热点区域的变化进行迁移更新,实现了快速查询。同时,HDMI 实现了基于缓存的更新,减少了系统 I/O 访问次数和时间。实验结果证明 HDMI 在索引维护和查询性能上优于 NDTR-tree,但 HDMI 只能索引城市路网上移动对象的历史和当前轨迹,对未来轨迹的索引将是今后研究的重点。

参考文献:

[1] 丁治明,孟小峰,白芸,等. 基于关系数据库的位置相关查询处理[J]. 计算机研究与发展,2004,41(3):492-499.

[2] 孟小峰,丁治明. 移动数据管理:概念与技术[M]. 北京:清华大学出版社,2009.

[3] GÜTING R H,BOHLEN M H,ERWIG M,et al. A foundation for representing and querying moving objects[J]. ACM Transactions on Database Systems,2000,25(1):1-42.

[4] WOLFSON O,XU B,CHAMBERLAIN S,et al. Moving object databases:issues and solutions[C]//Proceedings of the 10th SSDBM. [s.l.]:[s.n.],1998:111-122.

[5] SALTENIS S,JENSEN C S,LEUTENEGGER S T,et al. Indexing the positions of continuously moving objects[C]//Proceedings of the ACM SIGMOD international conference

on management of data. New York, NY, USA:ACM,2000:331-342.

[6] SALTENIS S,JENSEN C S. Indexing of moving objects for location-based services[C]//Proceedings of the 18th ICDE. [s.l.]:[s.n.],2002:463-472.

[7] TAO Y,PAPSDIAS D,SUN J. The TPR-tree:an optimized spatio-temporal access method for predictive queries[C]//Proceedings of the 29th international conference on very large data bases. Berlin,Germany:VLDB Endowment,2003:790-801.

[8] TUNG H D T,JUNG Y J,LEE E J,et al. Moving point indexing for future location query[C]//Proceedings of the ER workshops. [s.l.]:[s.n.],2004:79-90.

[9] 丁治明. 一种适合于频繁位置更新的网络受限移动对象轨迹索引[J]. 计算机学报,2012,35(7):1448-1461.

[10] FRENTOS E. Indexing objects moving on fixed networks[C]//Proceedings of the international symposium on advances in spatial and temporal databases. [s.l.]:[s.n.],2003:289-305.

[11] ALMEIDA V T,GÜTING R H. Indexing the trajectories of moving objects in networks[J]. GeoInformatica,2005,9(1):33-60.

[12] 丁治明,李肖南,余波. 网络受限移动对象过去、现在及将来位置的索引[J]. 软件学报,2009,20(12):3193-3204.

[13] 丁治明,余波,李曼,等. 网络受限移动对象不确定性轨迹的索引[J]. 计算机科学,2008,35(3):79-83.

[14] 乔少杰,韩楠,王超,等. 基于路网的移动对象动态双层索引结构[J]. 计算机学报,2014,37(9):1947-1958.