一种适合于频繁位置更新的网络受限移动对象轨迹索引

丁治明
(中国科学院软件研究所基础软件国家工程研究中心 北京 100190)

摘 要 移动对象索引是支持海量移动对象管理的一项关键技术.目前的移动对象时空轨迹索引方法如STR-Tree、TB-Tree、FNR-Tree、MON-Tree等均直接以轨迹单元作为基本的索引记录单位,在位置更新时需要频繁地在索引中插入新的记录,从而严重地影响了数据库的总体性能.为了解决上述问题,文中提出一种网络受限移动对象的动态概略化轨迹R树索引(DSTR-Tree).DSTR-Tree将索引空间划分成等距格栅,并通过格栅单元对每一条移动对象轨迹进行概略化,然后以概略化轨迹单元为基本索引记录单位建立R树索引.由于概略化轨迹的粒度大大粗于原始轨迹,因此移动对象不需要在每次位置更新的同时触发索引更新,而仅需要在轨迹跨越当前格栅单元时才进行索引更新,从而显著地降低了索引更新的代价.实验结果表明,DSTR-Tree在移动对象数据库频繁位置更新的实际运行条件下,提供了良好的索引维护及总体查询处理性能.

关键词 移动对象;数据库;时空轨迹;概略化;索引
中图法分类号 TP309 **DOI号**: 10.3724/SP.J.1016.2012.01448

An Index Structure for Frequently Updated Network-Constrained Moving Object Trajectories

DING Zhi-Ming
(*National Fundamental Software Research Center, Institute of Software, Chinese Academy of Sciences, Beijing* 100190)

Abstract Index is a key technology to improve the query processing performance of moving objects databases. However, current index methods for moving object trajectories, such as STR-Tree, TB-Tree, FNR-Tree, and MON-Tree, take trajectory units as the basic index records, and therefore, frequent insertions are needed when location updates occur in order to keep the consistency between the index and the trajectories in database, which greatly affects the overall performance of moving objects databases. To solve this problem, we propose a new index method, Dynamic Sketched-Trajectory R-Tree for Network-constrained Moving Objects (DSTR-Tree) in this paper. The DSTR-Tree divides the spatial-temporal space into equal-sized grid cells, transforms every trajectory into a sketched trajectory with each unit connecting two centers of the grid cells that the moving object travels through, and indices the sketched trajectory units as an R-Tree. Since the sketched trajectory has much coarser granularity than the original trajectory, the index updating cost can be greatly reduced — when a location update occurs, even though the original trajectory needs to be changed, the sketched trajectory may remain unchanged so that the DSTR-Tree does not need to be changed either. The experimental results show that the DSTR-Tree outperforms the previously proposed trajectory index methods in running moving objects databases with frequent location updates.

Keywords moving objects; database; spatial-temporal trajectory; sketched; index

收稿日期:2011-02-17;最终修改稿收到日期:2011-06-07.本课题得到国家自然科学基金项目(91124001,60970030)资助.**丁治明**,男,1966年生,博士,研究员,博士生导师,中国计算机学会(CCF)高级会员,主要研究领域为数据库与知识库系统、移动计算、时空数据库与数据挖掘、信息检索、物联网与云计算等. E-mail: zhiming@iscas.ac.cn.

1 引　言

近年来，移动对象数据库（Moving Objects Databases，MOD）成为了一个热点研究领域并得到了国内外研究人员的广泛关注. 移动对象数据库属于时空数据库（Spatio-Temporal Database）的范畴，是指对位置不断移动的物体或目标（如汽车、飞机、轮船、行人等）的动态位置及其它相关属性进行表示与管理的数据库[1]. 越来越多的应用要求对移动对象进行管理，而定位技术和无线通信技术的发展使得跟踪和记录移动对象的位置成为可能.

在典型的移动对象数据库系统中，通常存放着海量移动对象的时空数据. 例如，一个大中型城市的移动对象数目可以达到数百万甚至更多. 为了支持对这些移动对象过去及当前位置的查询，有效的索引手段是需要解决的关键问题.

在移动对象索引方面，人们已经进行了大量的工作，这些工作可以分为两大类：针对移动对象当前位置的索引和针对移动对象完整时空轨迹的索引. 移动对象当前位置索引的典型代表是 TPR-Tree[2]，其基本思想是在 R* 树索引的基础上，允许最小包容矩形（MBR）包含速度和方向等参数信息，从而使 MBR 能随时间参数进行变化，而不需要随着移动对象位置的变化频繁地修改索引记录. TPR-Tree 提出后，人们在此基础上进行了大量的改进工作，提出了许多 TPR-Tree 的变种树如 R^{EXP}-Tree[3]、TPR*-Tree[4]、Bulk-loading TPR-Tree[5]、HTPR-Tree[6]等. 此外，文献[7-9]还针对移动对象当前位置索引中由于频繁位置更新所导致的写代价过高问题进行了优化. 但是，所有上述索引均只能支持对移动对象当前时刻位置的查询，而不能支持对过去位置的查询.

移动对象完整时空轨迹的索引包含了移动对象过去及当前的位置信息，因此比移动对象当前位置索引具有更为广泛的用途. 这方面的研究工作又可以分为两大类：基于 Euclidean 空间的轨迹索引和基于交通网络的轨迹索引.

基于 Euclidean 空间的轨迹索引以 Euclidean 轨迹单元为索引记录的基本单位进行索引的组织[10-13]，其中每个 Euclidean 轨迹单元是 $X\times Y\times T$ 空间中的一个直线段，这些直线段可以组织成 R 树、Quad 树、Grid File 等形式，从而支持对移动对象的快速搜索. 上述方法的缺点在于 Euclidean 轨迹单元是直线，因此需要大量的轨迹单元来刻画复杂的移动对象轨迹曲线，效率较低.

为了进一步提高效率，人们越来越多地转向基于交通网络的移动对象轨迹索引，并提出了多种方法[14-17]. 基于交通网络的移动对象轨迹索引一般采用双层结构，其中上层是一个 2 维的 R 树，用于对固定的道路网络进行索引；下层是一系列的 R 树，每个下层 R 树与一条道路相对应，用于对移动对象在该条道路中提交的轨迹单元进行索引. 其中，每个轨迹单元对应于移动对象在曲线道路上的一段匀速行驶过程，可以刻画 $X\times Y\times T$ 空间中的一个曲线段. 与基于 Euclidean 空间的轨迹索引相比，基于交通网络的轨迹索引减少了轨迹单元的数目并降低了存储开销.

尽管在移动对象轨迹索引方面人们已经取得了一些重要的进展，但是目前的索引方法还存在着诸多缺陷：

（1）几乎所有的移动对象轨迹索引均以轨迹单元作为索引记录的基本单位，由于索引记录的粒度太细，每次当移动对象发生位置更新并产生新的轨迹单元时，均需要在索引中插入相应的索引记录，因此索引更新的频率等同于位置更新的频率，从而造成极大的索引更新开销；

（2）目前已经提出的基于网络的移动对象轨迹索引方法均采用了双层索引结构，这种结构缺乏通用性，仅适合于在专用系统中实现，很难在通用的可扩充数据库系统（如 PostgreSQL）中实现；

（3）基于网络的移动对象轨迹索引仅能处理移动对象与路网匹配的情况，不能表示移动对象与路网匹配不上（如移动对象在电子地图之外的小路上行驶）的情况，缺乏灵活性.

为了解决上述问题，本文提出一种新的移动对象轨迹索引方法：网络受限移动对象的动态概略化轨迹 R 树索引（Dynamic Sketched-Trajectory R-Tree for Network-constrained Moving Objects，简称 DSTR-Tree）. DSTR-Tree 将索引空间 $X\times Y\times T$ 划分成等距格栅，通过格栅单元对每一条时空轨迹进行概略化，并以概略化轨迹单元为基本索引记录单位建立 R 树索引. 这种方法实际上对移动对象时空轨迹进行了粒度粗化. 当发生位置更新时，如果新产生的轨迹单元没有跨越移动对象上次位置更新时所对应的格栅单元（该格栅单元称为移动对象的“活动格栅单元”），则不需要对索引进行任何修改；仅当新产生的轨迹单元跨越了活动格栅单元时才需要在

索引中插入新的记录,因此极大地降低了索引更新的代价,符合移动对象频繁位置更新的现实情况.此外,DSTR-Tree 采用了一种典型的单层树型结构,可以轻易地在通用数据库系统如 PostgreSQL 中实现.最后,DSTR-Tree 可以兼容移动对象在路网中和路网之外行驶的情况,具有充分的灵活性和实用性.

本文第 2 节给出网络受限移动对象通用时空轨迹数据模型;第 3 节描述 DSTR-Tree 的基本结构及相关算法;第 4 节对实验结果进行分析;第 5 节给出相关结论.

2 网络受限移动对象的通用时空轨迹模型

从抽象模型的角度来看,移动对象的时空轨迹对应于 $X\times Y\times T$ 空间中的一条曲线.在移动对象数据库中,需要对上述曲线进行离散化才能被计算机处理.本节讨论时空轨迹的离散化表示方法.

在移动对象轨迹的模型化方面,早期的工作采用的是直接基于 Euclidean 空间的表示方法[10,18],即通过 $X\times Y\times T$ 空间中的一系列直线线段来表示轨迹.为了刻画复杂的轨迹曲线,通常需要大量的直线线段,这意味着移动对象数据库需要更多的位置更新和更多的存储空间来生成和管理这些轨迹.近几年来,越来越多的工作转向了基于路网的轨迹表示方法[15,19-20],即通过一系列基于路网的轨迹单元来表示轨迹,每个基于路网的轨迹单元刻画移动对象在曲线道路上的一段匀速行使过程.由于基于路网的轨迹单元对应于 $X\times Y\times T$ 空间中的一段曲线,因此与基于 Euclidean 的表示方法相比,上述方法极大地降低了轨迹单元的数量和位置更新的代价.其缺点是无法表示移动对象与路网匹配不上的情况,如当移动对象在电子地图之外的小路或广场上行驶,或者当电子地图没有及时更新导致与实际道路不符时,均有可能出现移动对象与路网匹配不上的情况.

为了克服目前移动对象轨迹模型化方法的缺陷,增加轨迹表示的通用性和灵活性,我们在本节给出一种通用的移动对象轨迹模型,作为 DSTR-Tree 索引的基础.该模型在主要考虑移动对象路网受限运动的前提下,兼容与路网匹配不上的特殊情况.

为了讨论方便,我们假设移动对象在与路网匹配时采用文献[21]提出的基于路网的位置更新策略,即当移动对象跨越不同的道路时触发 IDTLU 位置更新;当移动对象的计算位置与 GPS 位置的差值达到规定的距离阈值 ξ_d 时触发 DTTLU 位置更新;当移动对象的行驶速度与上次位置更新时的速度差值达到规定的速度阈值 ξ_s 时触发 STTLU 位置更新;当移动对象与路网不能匹配时,采用基于固定时间间隔的位置更新策略(FTLU),即每隔规定的时间阈值 ξ_t 触发一次位置更新.此外,当移动对象的路网匹配状态发生改变(由匹配状态变成不匹配状态,或反之)时需要立刻进行位置更新.

定义 1(交通网络). 交通网络 N 定义为

$$N=(R,J),$$

其中,R 是道路的集合,J 是交叉路口的集合.

定义 2(道路). 道路 r 定义为如下形式:

$$r=(rid,geo,len,(jid_j,pos_j)_{j=1}^{m}),$$

其中,rid 是道路标识;geo 是该道路的地理几何形状;len 是道路的长度;$(jid_j,pos_j)_{j=1}^{m}$ 描述该条道路所包含的各个交叉路口(见定义 3)的标识以及它们在道路中的相对位置(设每条道路的总长度为 1,则该条道路中的任一相对位置 pos 可以用[0,1]之间的一个实数表示).

在上述定义中,道路的几何形状 geo 用一条折线 pl 表示,pl 是由 $X\times Y$ 平面中的一组点所组成的序列,即

$$pl=(x_i,y_i)_{i=1}^{n}(n\geq 2),$$

其中,(x_1,y_1) 为 pl 的起点,称为 r 的"0-端点",(x_n,y_n) 为 pl 的终点,称为 r 的"1-端点".

定义 3(交叉路口). 交叉路口 j 与实际交通网络中的交叉路口、道路出入口或道路端点相对应,定义为如下形式:

$$j=(jid,loc,((rid_i,pos_i))_{i=1}^{n},m),$$

其中 jid 是交叉路口的标识;loc 是 j 的地理位置,用其(x,y)坐标表示;$(rid_i,pos_i)(1\leq i\leq n)$描述 j 所连接的第 i 条道路,其中 rid_i 是道路标识,$pos_i\in[0,1]$是 j 在该条道路中的相对位置;m 是 j 的连接矩阵,用以描述该交叉路口的各交通流之间的连通关系[19].

定义 4(网络位置). 交通网络 N 中的任意一个网络位置 $npos$ 可以包括两种情况:① 位于某个交叉路口,此时可以直接以交叉路口的标识 jid 表示;② 位于道路中,此时可以用其所在的道路标识 rid 及在该道路中的相对位置 pos 表示.因此网络位置 $npos$ 定义为

$$npos=\begin{cases}jid, & npos\ 位于交叉路口\\ (rid,pos), & npos\ 位于道路中\end{cases}.$$

定义 5(移动对象的运行矢量). 移动对象 mo 在 t 时刻的运行矢量 $\boldsymbol{mv}$ 定义为

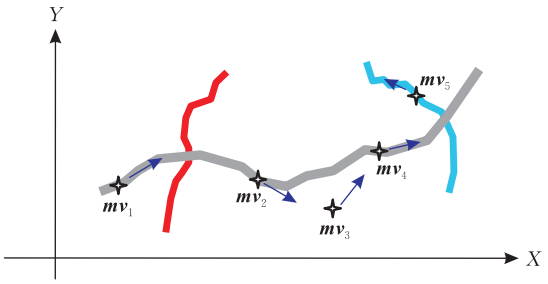
$$\boldsymbol{mv}=(t,(x,y),v,d,npos),$$

其中,t 是采集该运行矢量的时刻;(x,y)、v、d 分别是移动对象在 t 时刻的位置、速度、方向;$npos$ 是 $\boldsymbol{mv}$ 对应的网络位置(当移动对象的位置与道路网络匹配不上时,$npos$ 取空值). 在 $\boldsymbol{mv}$ 的各个参数中,t,(x,y),v,d 是由 GPS 采样得到的,而 $npos$ 是通过路网匹配得到的. 如果 $npos$ 为空,则 $\boldsymbol{mv}$ 称为 Eulidean 运行矢量;如果 $npos$ 有具体的网络位置匹配值,则 $\boldsymbol{mv}$ 称为路网匹配的运行矢量.

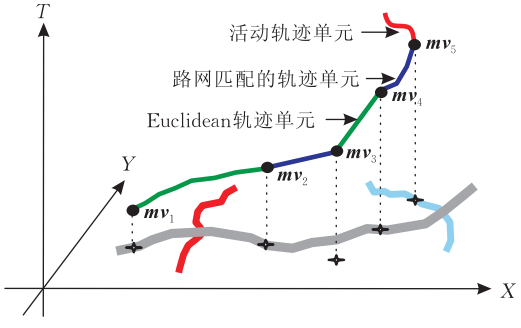
定义 6(移动对象的时空轨迹). 移动对象 mo 的时空轨迹 $traj$ 是 mo 在行驶过程中通过位置更新操作所生成的一组运行矢量的序列,用以描述 mo 的位置随着时间变化的过程,定义为如下形式

$$traj=(\boldsymbol{mv}_i)_{i=1}^{n}=((t_i,(x_i,y_i),v_i,d_i,npos_i))_{i=1}^{n}.$$

一条时空轨迹可以看成是若干个轨迹单元(见定义 7)所组成的序列,对应于 $X\times Y\times T$ 空间中的一条曲线. 图 1 给出了一个移动对象在路网中的行驶过程及对应得时空轨迹曲线.



(a) 移动对象在路网中的行驶过程



(b) 对应的移动对象时空轨迹曲线

图 1　移动对象的行驶过程以及对应的时空轨迹

移动对象的轨迹是通过位置更新操作得到的. 目前已经提出了多种不同的位置更新方法[21-24]. 在位置更新中,移动对象原始提交的参数一般包含运行矢量中的 t、(x,y)、v、d,而 $npos$ 的匹配可以在移动对象端或者在服务器端完成. 移动对象在行驶的过程中不断采集新的运行矢量并将之发送给服务器,因此服务器上保存的轨迹是不断增长的. 每隔一段时间(如两个星期),数据库需要对轨迹数据进行转储,并生成新的移动对象关系表. 一个理想的位置更新方法需要同时保证如下两个条件:

(1) 所生成的时空轨迹应精确地刻画移动对象实际的当前位置及历史行驶过程;

(2) 所生成的时空轨迹应包含尽可能少的运行矢量(或轨迹单元),并在其生成过程中采用尽可能少的位置更新操作.

定义 7(移动对象的轨迹单元). 时空轨迹 $traj=(\boldsymbol{mv}_i)_{i=1}^{n}=((t_i,(x_i,y_i),v_i,d_i,npos_i))_{i=1}^{n}$ 中的任意两个相邻的运行矢量 $\boldsymbol{mv}_i$ 和 $\boldsymbol{mv}_{i+1}$ $(1\leq i<n)$ 构成一个轨迹单元,记为 $\mu(\boldsymbol{mv}_i,\boldsymbol{mv}_{i+1})$. 此外,$traj$ 的最后一个运行矢量 $\boldsymbol{mv}_n$ 也对应着一个轨迹单元,记为 $\mu(\boldsymbol{mv}_n)$(我们称 $\mu(\boldsymbol{mv}_n)$ 为该移动对象的活动轨迹单元).

根据不同的路网匹配情况,轨迹单元 $\mu(\boldsymbol{mv}_i,\boldsymbol{mv}_{i+1})$ $(1\leq i<n)$ 可以被解析成不同的几何形态:

(1) 如果 $\boldsymbol{mv}_i$ 和 $\boldsymbol{mv}_{i+1}$ 均为路网匹配的运行矢量,则 $\mu(\boldsymbol{mv}_i,\boldsymbol{mv}_{i+1})$ 称为路网匹配的轨迹单元,对应于 $X\times Y\times T$ 空间中的一条曲线线段,该曲线线段反映移动对象沿着路网从 $\boldsymbol{mv}_i$ 到 $\boldsymbol{mv}_{i+1}$ 匀速行驶的时空过程(如图 1(b)中的 $\mu(\boldsymbol{mv}_1,\boldsymbol{mv}_2)$、$\mu(\boldsymbol{mv}_4,\boldsymbol{mv}_5)$);

(2) 如果 $\boldsymbol{mv}_i$ 和 $\boldsymbol{mv}_{i+1}$ 之一或二者皆为 Eulidean 运行矢量,则 $\mu(\boldsymbol{mv}_i,\boldsymbol{mv}_{i+1})$ 称为 Euclidean 轨迹单元,对应于 $X\times Y\times T$ 空间中的一条直线线段,该直线线段反映移动对象从 $\boldsymbol{mv}_i$ 到 $\boldsymbol{mv}_{i+1}$ 匀速直线行驶的时空过程(如图 1(b)中的 $\mu(\boldsymbol{mv}_2,\boldsymbol{mv}_3)$、$\mu(\boldsymbol{mv}_3,\boldsymbol{mv}_4)$).

对于活动轨迹单元 $\mu(\boldsymbol{mv}_n)$,其几何形状可以通过如下方式求得(设每个移动对象有一个活动状态标记 $activeFlag$,表示其是否处于联机行驶状态):

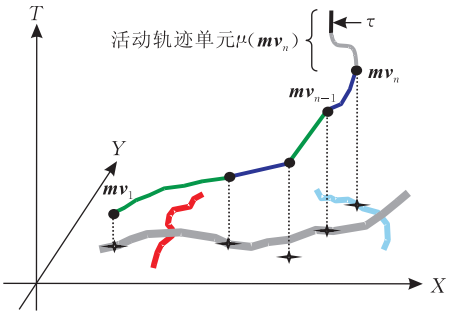
(1) 如果 $activeFlag=$false,则移动对象处于离线关闭状态,此时 $\mu(\boldsymbol{mv}_n)$ 为空值;

(2) 如果 $activeFlag=$true,则移动对象处于活动行驶状态,此时可以进一步细分为两种情况:

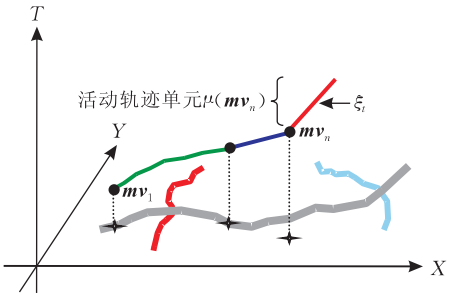
(2.1) 如果 $\boldsymbol{mv}_n$ 是路网匹配的运行矢量,则 $\mu(\boldsymbol{mv}_n)$ 对应于 $X\times Y\times T$ 空间中的一条曲线线段,该曲线线段反映移动对象沿当前道路按照 $\boldsymbol{mv}_n$ 中描述的速度、方向匀速行驶至道路终点(见图 2(a)中 $\mu(\boldsymbol{mv}_n)$ 的灰色曲线部分),并继续停留在道路终点 τ

时间(见图 2(a)中 $\mu(\boldsymbol{mv}_n)$的黑色垂直线部分. τ 的计算方法将在下面进一步分析)的行驶过程. 上述 $\mu(\boldsymbol{mv}_n)$的几何曲线反映的是移动对象在 $\boldsymbol{mv}_n$ 之后的计算位置. 根据文献[21-22]的分析,移动对象的计算位置在按照 $\boldsymbol{mv}_n$ 中的行驶参数到达当前道路终点之后,将继续停留在道路终点.

(2.2)如果 $\boldsymbol{mv}_n$ 是Euclidean运行矢量,则 $\mu(\boldsymbol{mv}_n)$ 对应于 $X\times Y\times T$ 空间中的一条直线线段,该直线线段反映移动对象从 $\boldsymbol{mv}_n$ 开始,按照 $\boldsymbol{mv}_n$ 中的速度、方向等运行参数匀速直线行驶 ξ_t 时间的行驶过程(见图 2(b)).



(a) $\boldsymbol{mv}_n$为路网匹配的运行矢量



(b) $\boldsymbol{mv}_n$为Euclidean运行矢量

图 2 活动轨迹单元 $\mu(\boldsymbol{mv}_n)$对应的几何曲线

通过分析可知,$\mu(\boldsymbol{mv}_n)$所对应的上述几何曲线的时间跨度能够保证下一次位置更新之前的所有计算位置都包含在该曲线中了. 如果 $\boldsymbol{mv}_n$ 为路网匹配的运行矢量(见上述(2.1)),则 $\mu(\boldsymbol{mv}_n)$几何曲线的时间跨度为按照最慢可能的速度到达道路终点所需要的时间;如果 $\boldsymbol{mv}_n$ 为 Euclidean 运行矢量(见(2.2)),则 $\mu(\boldsymbol{mv}_n)$几何曲线的时间跨度为 FTLU 位置更新的时间间隔. 因此可以保证,在上述时间跨度用完之前,必然会有新的位置更新产生.

下面讨论上述(2.1)中 τ 的计算方法. 如图 2(a)所示,活动运行矢量 $\boldsymbol{mv}_n$ 为路网匹配的运行矢量. 设 $\boldsymbol{mv}_n=(t_n,(x_n,y_n),v_n,d_n,npos_n)$,且 $npos_n=(rid_n,pos_n)$. 根据 $\boldsymbol{mv}_n$ 中的参数,可以计算出移动对象在 t_n 时刻离道路终点的距离为 $(1-pos_n)\times length(rid_n)$,其中 $length(rid_n)$ 为道路 rid_n 的长度. 移动对象按照 v_n 的速度行驶至道路终点所需要的时间为

$$t_{\text{norm}}=((1-pos_n)\times length(rid_n))/v_n,$$

而移动对象在不触发 STTLU 的情况下的最慢可能速度为 $(v_n-\xi_s)$,因此在不触发位置更新的前提下最晚到达道路终点的时间为

$$t_{\text{slow}}=((1-pos_n)\times length(rid_n))/(v_n-\xi_s),$$

所以有

$$\tau=t_{\text{slow}}-t_{\text{norm}}.$$

3 DSTR-Tree 索引的结构及相关算法

本节首先讨论 DSTR-Tree 的基本结构,然后给出 DSTR-Tree 索引的初始建立、动态维护及查询处理算法.

3.1 移动对象轨迹的概略化及 DSTR-Tree 索引的结构

为了对轨迹进行概略化,首先需要对 $X\times Y\times T$ 空间进行格栅化. 设移动对象数据库的应用时空空间为 $I_x\times I_y\times I_t$,其中,$I_x=[x_0,x_1]$,$I_y=[y_0,y_1]$,$I_t=[t_0,\perp]$(I_t的终点为$\perp$(未定义),因为当前时刻是不断增长的). 在进行格栅化时,可以将 I_x 划分成 n 个等大小的子区域,每个子区域的大小为

$$\zeta_x=\frac{x_1-x_0}{n}.$$

同理可以将 I_y 划分成 m 个等大小的子区域,每个子区域的大小为

$$\zeta_y=\frac{y_1-y_0}{m}.$$

对于 I_t,由于其终点未定义,可以将之划分成大小为 Δt 的等长时间段,即

$$\zeta_t=\Delta t.$$

通过上述方法,$I_x\times I_y\times I_t$ 空间被划分成了等距格栅. 格栅单元用其编号 (N_x,N_y,N_t)来表示,其中 N_x,N_y,N_t 分别为该单元在 X,Y,T 轴所对应的子区域编号,例如图 3 中右上角标记灰色的单元编号为(4,3,2).

将 $I_x\times I_y\times I_t$ 空间进行格栅化之后,接下来需要对轨迹进行概略化,从而形成概略化的轨迹(Sketched Trajectory). 为了方便叙述,称进行概略化之前的轨迹为"原始轨迹".

定义 8(移动对象的概略化轨迹). 设 $traj=((t_i,(x_i,y_i),v_i,d_i,npos_i))_{i=1}^{n}$ 是移动对象的轨迹,其概略化轨迹 $sketch(traj)$ 是将 $traj$ 轨迹曲线所穿

行的格栅单元的中心点通过直线线段连接起来所形成的轨迹，定义为如下形式：

$$sketch(traj)=(c_j)_{j=1}^{k}=((t_j,x_j,y_j))_{j=1}^{k},$$

其中 $c_j=(t_j,x_j,y_j)(1\leq j\leq k)$ 是 $traj$ 所穿行的第 j 个格栅单元的中心点的坐标. $sketch(traj)$ 中两个相邻的中心点 c_j 和 c_{j+1} 构成一个概略化轨迹单元(Sketched-Trajectory Unit)，记为 $\hat{\mu}(c_j,c_{j+1})$，它对应于 $X\times Y\times T$ 空间中的一条连接 c_j 和 c_{j+1} 的直线线段. 注意，概略化轨迹 $sketch(traj)$ 中已经包含了 $traj$ 的活动轨迹单元 $\mu(\boldsymbol{mv}_n)$ 的概略化信息. 图 3 给出了对 $I_x\times I_y\times I_t$ 空间进行格栅化及对原始轨迹进行概略化的例子.

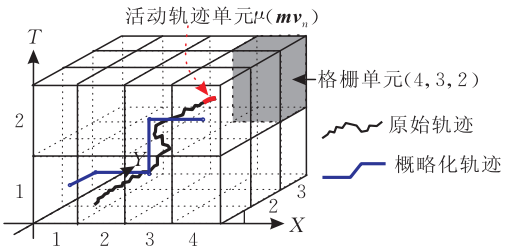


图 3　时空应用区域 $I_x\times I_y\times I_t$ 的格栅化及原始轨迹的概略化

如图 3 所示，概略化轨迹用大为减少的线段近似表示了原始轨迹的几何形状. 同一条原始时空轨迹所对应的概略化轨迹的单元数目取决于格栅单元的大小：格栅单元越大，则概略化轨迹的单元数目越少，反之亦然.

算法 1 描述了根据移动对象的原始时空轨迹生成概略化轨迹的过程. 其中，函数 $getCellLocated(\boldsymbol{mv})$ 计算运行矢量 $\boldsymbol{mv}$ 所位于的格栅单元，$getCellsTravelled(\mu)$ 计算轨迹单元 μ 所穿行的格栅单元序列，$extractCell(cellseq,i)$ 从格栅单元序列 $cellseq$ 中提取第 i 个格栅单元，$getCenter(cell)$ 计算格栅单元 $cell$ 的中心点坐标，$|cellseq|$ 返回格栅单元序列 $cellseq$ 中所包含的格栅单元数目，apppend($sketchTraj$, $center$) 将中心点坐标 $center$ 添加到 $sketchTraj$ 的最后，$doNothing()$ 不做任何事情而直接返回到主程序.

算法 1.　移动对象概略化轨迹生成算法.

全局变量：$I_x\times I_y\times I_t$　//应用空间区域

　　ζ_x,ζ_y,ζ_t　//格栅单元的划分粒度

输入：$traj=(\boldsymbol{mv}_i)_{i=1}^{n}=((t_i,(x_i,y_i),v_i,d_i,npos_i))_{i=1}^{n}$

　　$activeflag$　//移动对象的活动标记

输出：$sketchTraj=((t_j,x_j,y_j))_{j=1}^{k}$

```
1.  sketchTraj=NULL;
2.  startingCell=getCellLocated(mv_1);
3.  apppend(sketchTraj,getCenter(startingCell));
4.  IF n=1 THEN
5.   IF activeflag=FALSE THEN
6.    Return(sketchTraj);
7.   ELSE
8.    cellsTravelled=getCellsTravelled(μ(mv_1));
9.    IF (|cellsTravelled|=1) AND
        (extractCell(cellsTravelled,1)=
         startingCell) THEN
10.    Return(sketchTraj);
11.   ELSE
12.    FOR j=2 to |cellsTravelled| DO
13.     apppend(sketchTraj,getCenter
              (extractCell(cellsTravelled,j)));
14.    ENDFOR
15.   ENDIF
16.  ENDIF
17. ELSE  //n>1
18.  currentCell=startingCell;
19.  FOR i=1 to n-1 DO
     //处理非活动轨迹单元 μ(mv_i,mv_{i+1})(1≤i≤n-1)
20.   cellsTravelled=getCellsTravelled(μ(mv_i,mv_{i+1}));
21.   IF (|cellsTravelled|=1) AND
        (extractCell(cellsTravelled,1)=currentCell)
        THEN
22.    doNothing();
23.   ELSE //μ(mv_i,mv_{i+1})穿越了多个格栅单元
24.    FOR j=2 to |CellsTravelled| DO
25.     apppend(sketchTraj,getCenter
              (extractCell(cellsTravelled,j)));
26.    ENDFOR;
27.    currentCell=extactCell(cellsTravelled,
                              |cellsTravelled|);
28.   ENDIF
29.  ENDFOR
     //处理活动轨迹单元 μ(mv_n)
30.  IF activeflag=TRUE THEN
31.   cellsTravelled=getCellsTravelled(μ(mv_n));
32.   IF (|cellsTravelled|=1) AND
        (extractCell(cellsTravelled,1)=currentCell)
        THEN
33.    doNothing();
34.   ELSE
35.    FOR j=2 to |CellsTravelled| DO
36.     apppend(sketchTraj,getCenter
              (extractCell(cellsTravelled,j)));
37.    ENDFOR
38.   ENDIF
39.  ENDIF
40.  Return(sketchTraj);
41. ENDIF
```

在算法 1 中，函数 $getCellsTravelled(\mu)$ 的返回值可以包含一个或多个格栅单元，这些格栅单元的中心点被依次添加到 $sketchTraj$ 中.

通过算法 1 可以看出，当移动对象的原始轨迹单元不跨越格栅单元时，可以直接跳过该单元而不需要做任何处理；仅当原始轨迹单元跨越一个或多个格栅单元时，才需要生成相应的概略化轨迹单元. 在特殊情况下（如移动对象沿某条道路长时间匀速行驶时），原始轨迹单元可能跨越多个格栅单元，此时需要将这些格栅单元的中心连线依次加入到概略化轨迹中（见图 4(b)中的情况(3)）. 由于算法对活动轨迹单元也进行了相同的处理（见算法 1 第 30～39 行，其中活动轨迹单元 $\mu(\boldsymbol{mv}_n)$ 是根据移动对象最后一次位置更新时提交的行驶参数进行预测并实体化所得到的单元，如图 2 所示），从而使得移动对象即使长时间不进行位置更新时，DSTR-Tree 中也预先包含了相应的概略化轨迹单元. 这些预先计算的、与 $\mu(\boldsymbol{mv}_n)$ 相对应的概略化轨迹单元需要在下一次位置更新时进行调整和重新预测（见算法 3）.

图 4 分析了在处理一个新的轨迹单元 μ 时会出现的 3 种典型情况，其中标记灰色的单元是 $currentCell$ 所对应的格栅单元. 在 3 种典型情况中，(1)和(2)较为常见，而(3)仅在偶然的情况下才会出现，这是因为格栅单元的粒度通常远大于轨迹单元的粒度，所以不会经常出现 μ 跨越 3 个或更多格栅单元的情况.

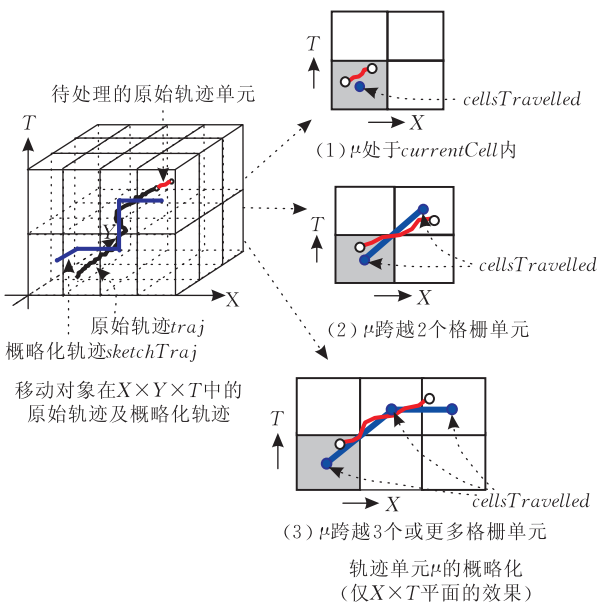


图 4 对原始轨迹单元 μ 的概略化处理

通过图 4 可以看出，在 DSTR-Tree 中，选取合适的格栅单元大小是至关重要的. 如果格栅单元过小，则会频繁地出现一个原始轨迹单元跨越多个格栅单元的情况，从而使得格栅化得不偿失；反之，如果格栅单元过大，又会导致索引块中无用记录增多，并进一步导致元组求精时间的增加. 为了获得良好的总体性能，在确定格栅单元大小时，可以参照原始轨迹单元的大小按比例放大，使得一个格栅单元能够平均容纳 φ 个原始轨迹单元.

根据具体的系统情况，φ 可以取不同的值（φ 还可以取小数，如 1.5），见本文的实验部分. 例如，设系统中移动对象的平均位置更新时间间隔为 30 s，平均速度为 60 km/h，30 s 对应的行驶距离为 500 m，所以原始轨迹单元的平均大小为 500 m×500 m×30 s. 如果取 $\varphi=5$，则格栅单元的大小可以确定为 2500 m×2500 m×150 s.

在完成 $I_x\times I_y\times I_t$ 空间的格栅化和原始时空轨迹的概略化之后，以每个概略化轨迹的各个轨迹单元为基本索引记录单位建立 R 树索引，即可得到 DSTR-Tree. 图 5 给出了 DSTR-Tree 的结构.

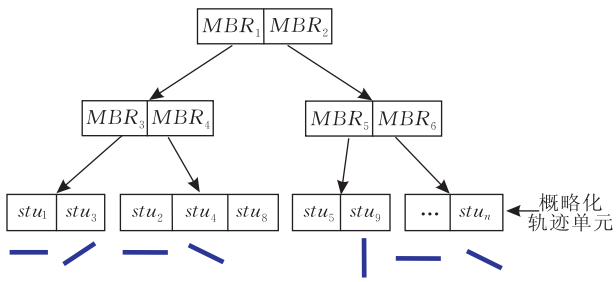


图 5 DSTR-Tree 索引的结构

在图 5 中，DSTR-Tree 叶子结点中的记录项结构为〈MBR，PT_{mo}，stu〉，其中 stu 为概略化轨迹单元，MBR 为该概略化轨迹单元的最小包容矩形，PT_{mo} 为指向实际移动对象记录的指针或记录标识. 中间结点的记录项为〈MBR，PT_{node}〉，其中 MBR 包含下层结点所有记录的最小包容矩形，PT_{node} 是指向下层结点的指针.

3.2 DSTR-Tree 索引的初始建立算法

在移动对象数据库中初始建立 DSTR-Tree 索引时，需要对数据库中的每条时空轨迹进行概略化，并将概略化轨迹中的每个轨迹单元逐一插入到 DSTR-Tree 中. 算法 2 给出了 DSTR-Tree 的初始建立算法. 在 DSTR-Tree 对数据库中的轨迹数据进行处理的同时，移动对象数据库还会继续接收到新的位置更新消息. 这些新位置更新消息被暂时缓存在 $buffer$ 中，等数据库中的轨迹数据被处理完毕之后，再调用动态维护算法（算法 3）处理这些新收到

的位置更新消息.

算法 2.　DSTR-Tree 的初始建立算法.

输入：*MObjs*　　//移动对象的集合

输出：*dstrTree*　　//DSTR-Tree

1. *dstrTtree*=NULL;
2. *buffer*=∅;
3. asynExec(*receiveLUMsg*(*buffer*));
   //启动另一线程接受新的位置更新并存入 *buffer*;
4. FOR EACH *mo*∈*MObjs* DO
5. *sketchTraj*=*skecth*(*mo.traj*,*mo.actviveflag*);
   //调用算法 1 计算概略化轨迹;
6. FOR EACH *skecthTrajUnit*∈*getUnits*(*sketchTraj*) DO
7. insert(*dstrTree*,*indexRec*(*mo.id*, *skecthTrajUnit*));
8. ENDFOR
9. ENDFOR
10. WHILE *buffer*≠∅ DO
11. *lumsg*=*getLUMsg*(*buffer*);
12. IF notIndexed(*lumsg*) THEN
13. 调用算法 3 根据 *lumsg* 对 *dstrTree* 进行维护;
14. ENDIF
15. ENDWHILE
16. Return(*dstrTree*).

在算法 2 中，函数 asynExec(*func*)通过启动另一线程异步地执行函数 *func*，*receiveLUMsg*(*buffer*)接收位置更新消息并存入到 *buffer* 中，*getUnits*(*sketchTraj*)返回 *sketchTraj* 中的所有轨迹单元所组成的集合，*indexRec*(*moid*,*stu*)根据移动对象标识 *moid* 和概略化轨迹单元 *stu* 生成相应的索引记录，*getLUMsg*(*buffer*)从 *buffer* 中取出(同时从 *buffer* 中删除)一个位置更新消息，notIndexed(*lumsg*)判断 *lumsg* 是否在相应的移动对象被处理之前已经写入其原始轨迹中了，这种情况下 *lumsg* 的信息已经包含在 *dstrTree* 中了，不需要再进行处理.

需要注意的是，概略化轨迹及概略化轨迹单元仅在建立索引和维护索引的过程中使用，并不需要永久地保存在数据库中.

3.3　DSTR-Tree 索引在位置更新时的动态维护算法

在 DSTR-Tree 索引初始建立之后，当发生位置更新时，不仅需要在数据库中将新的运行矢量附加到轨迹中，而且还需要对 DSTR-Tree 进行动态维护. 由于移动对象在行驶过程中不断地通过位置更新操作向数据库服务器发送新的运行矢量，因此其时空轨迹是随着新的运行矢量的加入而不断增长的. 当原始轨迹的增长导致概略化轨迹的变化时，就需要同时对 DSTR-Tree 索引进行相应的修改.

更具体地说，对于一个活动移动对象 *mo*，设其原始时空轨迹为 $traj=(\boldsymbol{mv}_i)_{i=1}^{n}=((t_i,(x_i,y_i),v_i,d_i,npos_i))_{i=1}^{n}$，对应的概略化轨迹为 $sketch(traj)=((t_j,x_j,y_j))_{j=1}^{k}$. 当服务器新接收到 *mo* 发送来的运行矢量 $\boldsymbol{mv}_u$ 时，需要将 $\boldsymbol{mv}_u$ 附加到 *traj* 中. 上述过程对应着原始轨迹的几何曲线的变化(见图 6)，部分情况下原始轨迹的变化也导致概略化轨迹的变化，此时需要对 DSTR-Tree 进行相应的修改. 但由于概略化轨迹单元的粒度远大于原始轨迹移动单元的粒度，大部分情况下 $\boldsymbol{mv}_u$ 加入到 *traj* 中不会对 *sketch*(*traj*)造成任何改变，此时就不需要对索引进行任何修改.

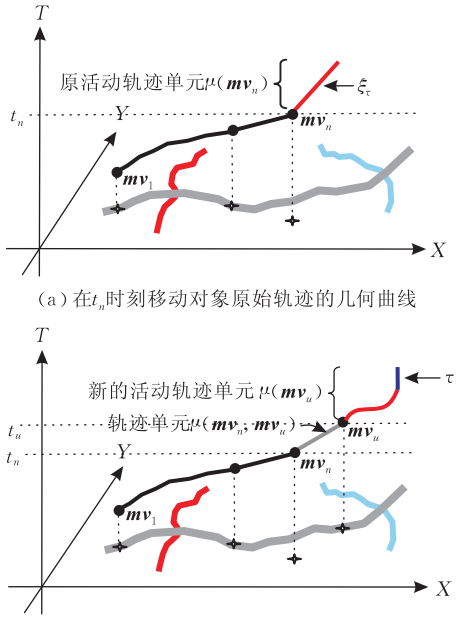


图 6　在发生位置更新时移动对象原始轨迹几何曲线的改变

数据库服务器在接收到一个位置更新消息时对 DSTR-Tree 索引进行维护的过程如算法 3 所示. 在该算法中，函数 *getTrajectory*(*moid*)根据移动对象的标识在数据库中检索指定移动对象的轨迹(为了加快检索速度，可以基于轨迹数据建立以 *moid* 为关键字的 B^+ 树索引)，*final*(*traj*)提取轨迹 *traj* 的最后一个运行矢量，*getCellsTravelledExt*($\mu_1,\mu_2,\cdots$)计算轨迹单元序列 $\mu_1,\mu_2,\cdots$ 所穿行的格栅单元序列，*getSTUnits*(*cellseq*)返回格栅单元序列 *cellseq* 对应的所有轨迹单元所组成的集合.

在算法 3 中，$cellsTravelled_{\text{old}}$ 是原活动轨迹单元 $\mu(\boldsymbol{mv}_n)$ 所穿过的格栅单元的序列，如〈*cell*3,

$cell4\rangle.cellsTravelled_{new}$是 $\mu(\boldsymbol{mv}_n,\boldsymbol{mv}_u)$和 $\mu(\boldsymbol{mv}_u)$所穿行的格栅单元序列，如$\langle cell3,cell5,cell4\rangle$如果二者不相等，则需要对 DSTR-Tree 进行调整，通过插入 *unitsInsert*，并删除 *unitsDelete*，可以将 DSTR-Tree 调整为与新概略化轨迹相对应的状态.

算法 3. 位置更新时 DSTR-Tree 索引的维护算法.

输入：$LUMsg=(moid,t,(x,y),v,d,npos)$；
//收到的位置更新消息
dstrTree //DSTR-Tree 索引

1. $\boldsymbol{mv}_u=(t,(x,y),v,d,npos)$；
2. $\boldsymbol{mv}_n=final(getTrajectory(moid))$；
3. $cellsTravelled_{old}=getCellsTravelled(\mu(\boldsymbol{mv}_n))$；
4. $cellsTravelled_{new}=getCellsTravelledExt(\mu(\boldsymbol{mv}_n,\boldsymbol{mv}_u),\mu(\boldsymbol{mv}_u))$；
5. IF $cellsTravelled_{old}=cellsTravelled_{new}$ THEN
//概略化轨迹没有发生变化
6. doNothing()；
7. ELSE //概略化轨迹发生了变化
8. $unitsInsert=getSTUnits(cellsTravelled_{new})-getSTUnits(cellsTravelled_{old})$；
9. $unitsDelete=getSTUnits(cellsTravelled_{old})-getSTUnits(cellsTravelled_{new})$；
10. FOR $skecthUnit\in unitsDelete$ DO
11. delete($dstrTree$,$indexRec(moid,skecthUnit)$)；
12. ENDFOR
13. FOR $skecthUnit\in unitsInsert$ DO
14. insert($dstrTree$,$indexRec(moid,skecthUnit)$)；
15. ENDFOR
16. ENDIF

在移动对象数据库中，每次收到一个新的位置更新消息 *LUMsg* 时，均需要将 *LUMsg* 插入到相应移动对象的轨迹中，同时调用算法 3 对索引进行维护. 由于概略化轨迹的粒度远大于原始轨迹的粒度，大部分情况下算法 3 并不需要对索引作任何修改(见第 5～6 行)，从而有效地降低了索引更新的代价.

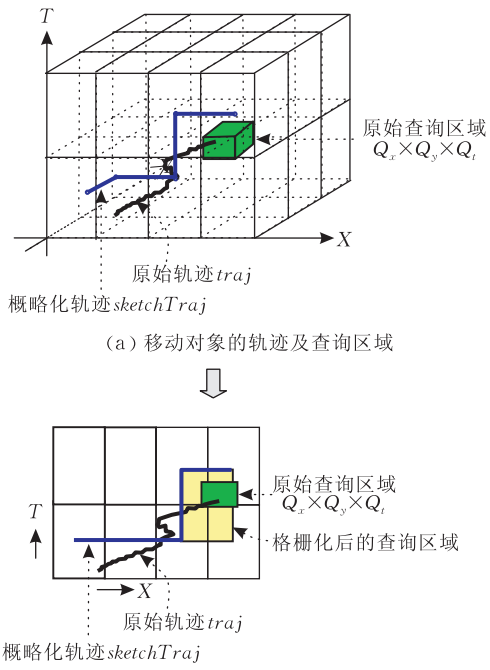
3.4 基于 DSTR-Tree 的移动对象查询处理算法

本节分析基于 DSTR-Tree 的移动对象查询处理方法. 设有任一查询 Q，其查询区域为

$$\text{range}(Q)=Q_x\times Q_y\times Q_t,$$

其中 $Q_x=[q_x^0,q_x^1]$，$Q_y=[q_y^0,q_y^1]$，$Q_t=[q_t^0,q_t^1]$.

在处理上述查询时，需要首先对查询区域 $range(Q)$进行格栅化，即要将查询区域修正为以格栅单元的中心作为区域的起止点，如图 7 所示(为了方便叙述，称 $range(Q)$为原始查询区域).



(a) 移动对象的轨迹及查询区域

(b) 对查询区域的格栅化(仅$X\times T$平面的效果)

图 7 对查询区域的格栅化

在图 7 中，移动对象的原始轨迹和原始查询区域 $range(Q)$相交，因此该移动对象理应作为索引的查询结果输出. 但是，由于在 DSTR-Tree 中存放的是概略化轨迹，且该概略化轨迹与查询区域 $range(Q)$并不相交，因此如果不对查询区域进行格栅化，则会漏检该移动对象. 通过分析可知，由于所有经过相应格栅单元的原始轨迹都被映射到格栅单元的中心了，因此只要查询区域包含了格栅单元的中心即不会产生漏检. 对查询区域格栅化的目标即是保证所有其原始轨迹与原始查询区域相交的移动对象，必然有其概略轨迹与格栅化查询区域相交.

下面以 $Q_x=[q_x^0,q_x^1]$的变换为例，分析查询区域格栅化的过程. 对 q_x^0和 q_x^1分别进行如下变换，可以将其起始点分别对应到相应格栅单元的中心：

$$q_x^0=x_0+\left(\left\lfloor\frac{q_x^0-x_0}{\xi_x}\right\rfloor+0.5\right)\times\xi_x,$$

$$q_x^1=x_0+\left(\left\lfloor\frac{q_x^1-x_0}{\xi_x}\right\rfloor+0.5\right)\times\xi_x.$$

类似地，可以对 $Q_y=[q_y^0,q_y^1]$，$Q_t=[q_t^0,q_t^1]$进行相应的变换，从而得到格栅化之后的查询区域.

注意，格栅化之后的查询区域尽管在形式上表示为一个矩形区域，但由于区域的起始点可以相等，因此实际上可以对应于 $X\times Y\times T$ 空间中的一条直线或者一个点.

在完成上述变换之后，可以根据格栅化后的查询区域对 DSTR-Tree 进行查询，得到一组候选移动对象的集合(见算法 4 中的 *filterResult*)．然后系统对 *filterResult* 中的移动对象按照查询条件逐个进行计算，并输出满足查询条件的移动对象作为最终查询结果．

基于 DSTR-Tree 的查询处理算法如算法 4 所示．在该算法中，函数 *gridCenterAlign*(*range*)对查询区域 *range* 进行格栅化，*evaluate*(*moTuple*, *Q*)根据查询 *Q* 对元组 *moTuple* 进行计算，如果元组不满足查询条件则返回 NULL；否则返回查询计算的结果．

算法 4.　基于 DSTR-Tree 的移动对象查询处理算法．

输入：查询 *Q*，其查询区域为 $range(Q)=Q_x\times Q_y\times Q_t$；*dstrTree*

输出：*refineResult*　//*Q* 的查询处理结果

1. $filterResult \leftarrow$ Search(*dstrTree*, $gridCenterAlign(Q_x\times Q_y\times Q_t)$)；
2. $refineResult=\varnothing$；
3. FOR EACH $PT_{mo}\in filterResult$ DO
4. $moTuple=getTuple(PT_{mo})$；
5. IF $evaluate(moTuple,Q)\neq$NULL THEN
6. $refineResult=refineResult\cup evaluate(moTuple,Q)$；
7. ENDIF
8. ENDFOR
9. Return(*refineResult*).

如算法 4 所示，基于 DSTR-Tree 的查询分为两个阶段：

(1)索引筛选(Filtering)阶段，通过 DSTR-Tree 索引筛选出查询区域所对应的移动对象集合 *filterResult*(该集合大于实际满足查询条件的移动对象集合)；

(2)元组求精(Refinement)阶段，对 *filterResult* 集合中包含的移动对象的数据库元组进行查询计算，并将最终结果 *refineResult* 返回给查询用户．

通过算法 4 可以看出，数据库系统查询处理的总时间却取决于索引筛选时间和元组求精时间．其中，索引筛选时间又受到索引更新代价及索引查询代价的双重影响．通过分析可知，系统中格栅单元的粒度越大，则索引更新的代价越低，而 *filterResult* 中包含的无用记录越多，导致元组求精的代价越大；反之，系统中格栅单元的粒度越小，则 *filterResult* 中包含的无用记录越少，元组求精的代价也越小，但是索引更新的代价却越高，也会间接影响总体查询性能．由此可见，为了达到最佳的总体查询性能，需要选择合适的格栅单元粒度．

4　实验比较与分析

在“基于路网的移动对象数据库及交通流统计分析系统(NMOD-TFSA)”[25]中，我们实现了本文提出的 DSTR-Tree 索引模块．NMOD-TFSA 是在 PostgreSQL 8.2.4 数据库内核及其空间数据扩展 PostGIS 的基础上实现的一个移动对象数据库系统，支持完整的空间数据类型、交通网络数据类型、移动对象时空轨迹数据类型以及丰富的时空查询与位置更新操作．NMOD-TFSA 系统的结构如图 8 所示．

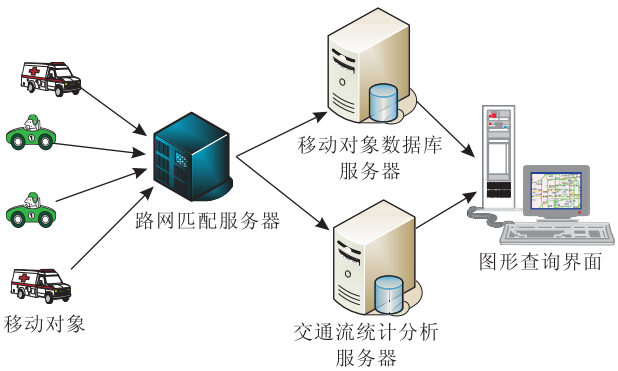


图 8　NMOD-TFSA 移动对象数据库系统的结构

DSTR-Tree 是在 PostgreSQL 所提供的通用索引框架 GIST 的基础上实现的．GIST 是一个可扩充的通用树形索引框架，通过开放相应的索引记录结构定义以及排序、插入、删除、查询等函数，可以实现用户自定义的单层树形索引结构，如 R 树、R* 树等．由于 DSTR-Tree 是一种典型的单层树形结构，因此可以通过 GIST 无缝地在 PostgreSQL 中实现，这是与其它基于路网的移动对象索引相比所具有的优点之一．MON-Tree[15]、FNR-Tree[14]、NDTR-Tree[17]等均采用了双层结构，因此无法直接在 GIST 中实现．

为了分析相关索引的性能，我们将 DSTR-Tree 与目前主流的移动对象轨迹索引方法进行了比较．由于已经提出的移动对象轨迹索引方法如 STR-Tree、TB-Tree、MON-Tree、NDTR-Tree 等均以原始轨迹单元作为索引记录的基本单位，因此它们在位置更新时具有相同量级的索引更新频率；此外，由于 MON-Tree 和 NDTR-Tree 采用双层结构，不能

在 PostgreSQL 中基于 GIST 直接实现，而 DSTR-Tree 的主要设计目的之一即通过单层通用的索引数据结构(详见第 1 节)，来克服双层索引结构不易在通用 DBMS 中实现且无法兼容路网之外移动对象的局限，因此本文选择上述方法中最具有代表性的单层索引 TB-Tree 作为比较对象.

由于 TB-Tree 是基于 Euclidean 空间的，而 DSTR-Tree 可以同时兼容 Euclidean 运行矢量和基于路网的运行矢量两种情况，为了保证实验结果的公正性，在实验中强制规定 DSTR-Tree 和 TB-Tree 均处理完全相同的基于 Euclidean 的轨迹，从而使得二者在相同的位置更新频率下进行比较和分析. 此外，TB-Tree 只能处理移动对象的历史轨迹信息，不能依据活动轨迹单元信息对当前位置进行估算. 为了使之能对当前位置进行预测计算处理，将活动轨迹单元的插入与修正过程(见第 3.3 节)也加入到 TB-Tree 中.

实验中的交通网络数据采用的是北京市的真实 GIS 地图数据(为了实验方便，我们仅选择了三级以上的道路)，并通过一个数据转换程序，将 GIS 数据转换成了本文所定义的格式. 移动对象的轨迹数据则采用了北京星通联华交通科技有限公司提供的北京市出租汽车 GPS 历史数据，通过一个回放程序重新再现移动对象的位置更新及轨迹生成过程.

实验的硬件平台是 Genuine Intel(R) CPU 2140 处理器，1.6 GHz 主频，1 GB 内存，运行 Linux 操作系统. 表 1 列出了实验中的主要参数.

表 1 模拟实验的主要参数

参数名称	参数值(单位)	参数含义
N_{mo}	1000～9000	移动对象的个数
N_{routes}	38577	路网中道路的数量
N_{juncts}	15584	路网中交叉路口的数量
Lon-range	116.125～116.625	地图的经度范围
Lat-range	39.75～40.083	地图的纬度范围
Duration	10800(s)	每一轮模拟持续的时间
ξ	65(s)	移动对象触发位置更新的时间间隔
tuSize	1000 m×1000 m×65 s	原始轨迹单元平均大小
φ	2～10	格栅单元大小与原始单元大小的倍数关系
cellSize	*tuSize*×φ(φ=2～10)	DSTR-Tree 进行空间划分时的格栅单元大小，为 *tuSize* 的 φ 倍

在实验数据集中，原始轨迹单元的平均大小大约为 *tuSize*=1 km×1 km×65 s. DSTR-Tree 格栅单元的大小取其 φ 倍，如当 φ=5 时，格栅单元的大小为 *cellSize*=5 km×5 km×325 s. 为了实验方便，统一取 φ 为 2、4、6，即比较 *cellSize* 分别取值为 2 km×2 km×130 s、4 km×4 km×260 s、6 km×6 km×390 s 时的实验结果.

在实验中比较的性能指标包括：

(1) 索引更新代价(以索引记录插入和删除的总操作数来表示)；

(2) 索引所占用的存储空间大小；

(3) 数据库的查询处理时间；

(4) 数据库的查询处理时间与格栅单元大小之间的关系.

索引更新代价的实验结果如图 9 所示. 从图中可以看出，DSTR-Tree 在各种参数条件下的索引更新代价都低于 TR-Tree. 由于 DSTR-Tree 加大了索引记录的粒度，不需要像 TB-Tree 那样在每次位置更新时都要往索引中写入新记录，因此减少了索引更新的次数. 另外可以看出，DSTR-Tree 的格栅粒度越大其索引更新代价越低，因为移动对象仅在其时空轨迹跨越格栅单元时才需要进行索引更新.

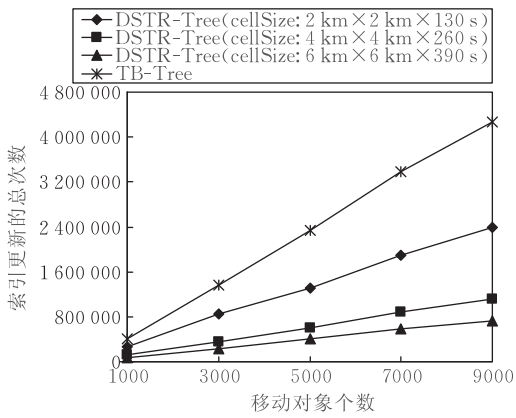


图 9 索引更新代价

图 10 给出了索引存储空间消耗的实验结果. 从图中可以看出，DSTR-Tree 在各种参数条件下的索引存储空间普遍低于 TB-Tree. 这是因为 DSTR-Tree 索引以概略化轨迹单元为索引记录的基本单位，而 TB-Tree 以原始轨迹单元为索引的记录单位. 对于同一个移动对象，其概略化轨迹单元的数目远小于原始轨迹单元的数目，因此 DSTR-Tree 的记录数比 TB-Tree 要少，索引存储空间也小于 TB-Tree.

图 11 是移动对象数据库在频繁位置更新的动态运行条件下进行查询处理的时间. 如前所述，查询处理时间由索引查找时间和元组求精时间组成. 为了显示查询处理各阶段时间的细节，在图 11(a)和图 11(b)分别给出索引查询时间和元组求精时间的实验结果，在图 11(c)中给出总体查询处理时间.

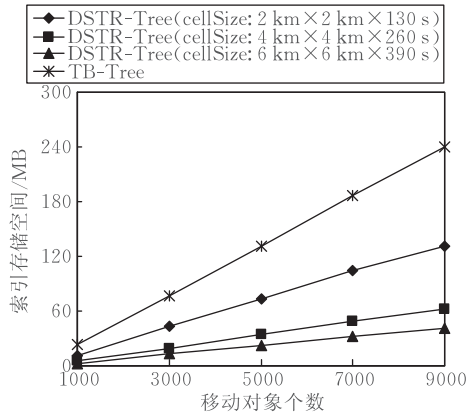
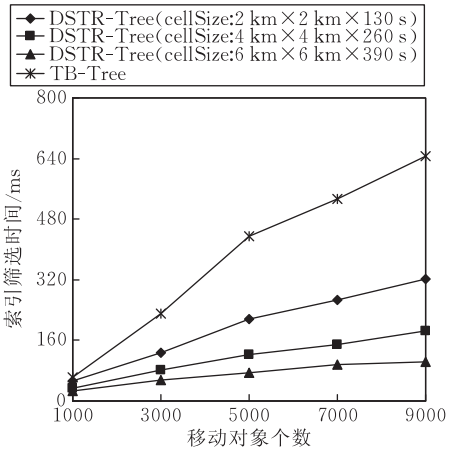


图 10　索引存储空间

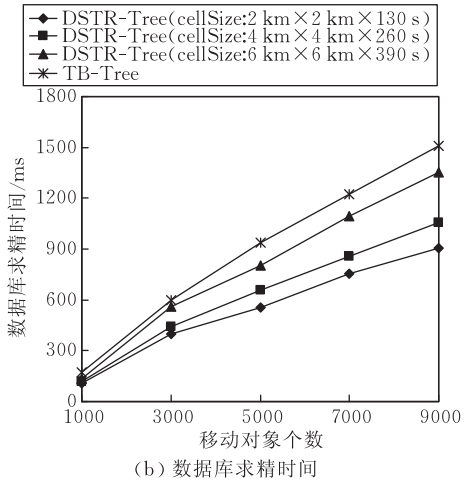
从图 11(a)可以看出，DSTR-Tree 的索引查找时间总体上低于 TB-Tree，这是因为一方面 TB-Tree 比DSTR-Tree要大，因此索引查找需要扫描更多的记录；另一方面，TB-Tree 在索引查找的同时需要处理频繁的索引更新操作，而 DSTR-Tree 的索引更新代价要小得多.

如图 11(b)所示，总体上讲，DSTR-Tree 的元组求精时间也少于 TB-Tree. 尽管 TB-Tree 比 DSTR-Tree 具有更高的准确率(即 filterResult 中包含的无用记录少)，但是这一优势被频繁的索引更新操作抵消掉了.

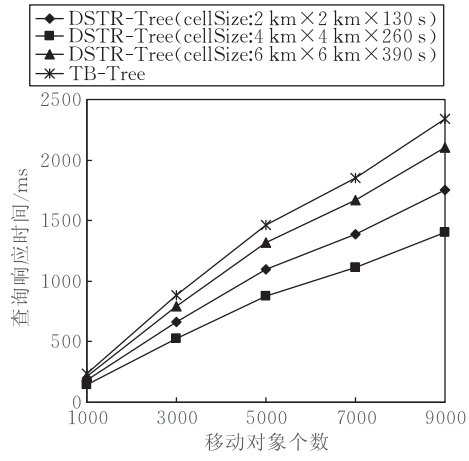
从图 11(c)可以看出，在频繁位置更新的条件下，DSTR-Tree 比 TB-Tree 具有更快的动态查询处理时间. 查询处理时间是索引查找时间与元组求精时间之和，再加上一些额外的开销，它同样极大地受到索引更新代价的影响. 此外，从图中还可以看出，DSTR-Tree 的查询响应时间与格栅的大小不呈单调关系.



(a) 索引筛选时间



(b) 数据库求精时间



(c) 查询响应时间

图 11　DSTR-Tree 与 TB-Tree 的性能比较

为了更好地分析查询响应时间与格栅单元大小的关系，变化 φ 的取值，从而得到格栅大小与查询响应时间的关系如图 12 所示(实验中移动对象的个数固定为 5000 个).

从图中可以出看，在本文的实验环境下，当格栅大小是原始轨迹单元大小的 4 倍(即 $\varphi=4$)时，系统能达到最佳的总体查询性能. 格栅单元大小的选择与实验环境密切相关，通常的影响因素包括数据库

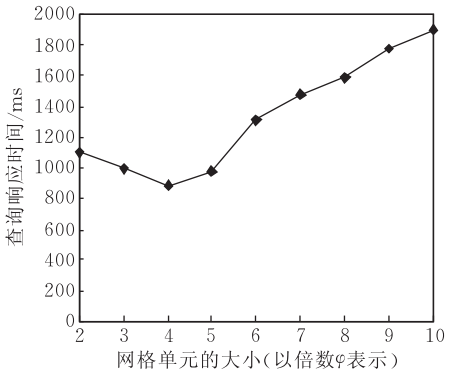


图 12 查询响应时间与格栅单元大小的关系

服务器的读写负荷比例、元组求精计算速度、数据库写速度等.

综合以上分析,与 TB-Tree 等以移动对象原始轨迹单元为基本索引单位的索引方法相比,DSTR-Tree 在移动对象数据库频繁位置更新的现实动态运行条件下具有更好的总体性能.

5 结 论

移动对象索引是支持海量移动对象管理的一项关键技术.然而,目前的移动对象轨迹索引方法如 STR-Tree、TB-Tree、FNR-Tree、MON-Tree、NDTR-Tree 等均以轨迹单元作为索引的基本记录单位,每次位置更新的同时需要进行一次索引更新,从而极大地增加了索引更新的代价,限制了数据库系统所能支持的移动对象的规模.此外,现有的网络受限移动对象的轨迹索引只能支持移动对象与路网完全匹配的情况,且采用双层结构,缺乏必要的灵活性,影响了它们在实际应用系统中的广泛使用.

为了解决上述问题,本文提出了一种网络受限移动对象的动态概略化轨迹 R 树索引(DSTR-Tree).DSTR-Tree 以移动对象的概略化轨迹单元作为索引记录的基本单位,当移动对象不跨越格栅单元时不需要对索引进行更新,因此降低了索引更新的代价,有效地提高了移动对象数据库的动态查询性能.此外,DSTR-Tree 采用典型的单层树形结构,可以无缝地在通用数据库框架如 PostgreSQL 中实现.最后,DSTR-Tree 能够同时支持移动对象在路网内和在路网之外移动的情况,具有充分的灵活性和实用性.实验结果表明,DSTR-Tree 在实际移动对象数据库频繁位置更新的现实条件下,提供了优秀的索引维护及总体查询处理性能.

致 谢 感谢中国科学院软件研究所的郭黎敏博士在 NMOD-TFSA 系统实现及 DSTR-Tree 索引实验方面所作的大量工作!

参 考 文 献

[1] Wolfson O, Xu B, Chamberlain S et al. Moving object databases: Issues and solutions//Proceedings of the 10th SSDBM. Capri, Italy, 1998: 111-122

[2] Saltenis S, Jensen C S, Leutenegger S T et al. Indexing the position of continuously moving objects//Proceedings of the ACM SIGMOD 2000. Texas, USA, 2000: 331-342

[3] Saltenis S, Jensen C S. Indexing of moving objects for location-based services//Proceedings of the 18th ICDE. San Jose, CA, 2002: 463-472

[4] Tao Y, Papadias D, Sun J. The TPR*-tree: An optimized spatio-temporal access method for predictive queries//Proceedings of the 29th International Conference on VLDB. Berlin, Germany, 2003: 790-801

[5] Lin B, Su J. On bulk loading TPR-tree//Proceeding of the 5th International Conference on Mobile Data Management (MDM'2004). Berkeley, USA, 2004: 114-124

[6] Tung H D T, Jung Y J, Lee E J et al. Moving point indexing for future location query//Proceedings of the ER Workshops'2004. Shanghai, China, 2004: 79-90

[7] Chen N, Shou L, Chen G et al. Adaptive indexing of moving objects with highly variable update frequencies. Journal of Computer Science and Technology, 2008, 23(6): 998-1014

[8] Chen J, Meng X. Update-efficient indexing of moving objects in road networks. Geoinformatica, 2009, 13(4): 397-424

[9] Kwon D, Lee S, Lee S. Indexing the current positions of moving objects using the lazy update R-tree//Proceedings of the 3rd International Conference on Mobile Data Management (MDM'2002). Singapore, 2002: 113-120

[10] Pfoser D, Jensen C S, Theodoridis Y. Novel approach to the indexing of moving object trajectories//Proceedings of the 26th International Conference on VLDB. Cairo, Egypt, 2000: 395-406

[11] Pfoser D. Indexing the trajectories of moving objects. IEEE Data Engineering Bulletin, 2002, 25(2): 3-9

[12] Tao Y, Papadias D. MV3R-tree: A spatio-temporal access method for timestamp and interval queries//Proceeding of the 27th International Conference on VLDB. Roma, Italy, 2001: 431-440

[13] Ding R, Meng X, Bai X. Efficient index update for moving objects with future trajectories//Proceedings of the 8th International Conference on DASFAA, Kyoto, Japan, 2003: 183-194

[14] Frentzos E. Indexing objects moving on fixed networks//Proceedings of the 8th International Symposium SSTD. Santorini Island, Greece, 2003: 289-305

[15] Almeida V T, Güting R H. Indexing the trajectories of moving objects in networks. GeoInformatica, 2005, 9(1): 33-60

[16] Chen J, Meng X. Indexing future trajectories of moving objects in a constrained network. Journal of Computer Science and Technology, 2007, 22(2): 245-251

[17] Ding Zhi-Ming, Li Xiao-Nan, Yu Bo. Indexing the historical, current, and future locations of network-constrained moving objects. Journal of Software, 2009, 20(12): 3193-3204(in Chinese)
(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. 软件学报，2009，20(12)：3193-3204)

[18] Güting R H, Böhlen M H, Erwig M et al. A foundation for representing and querying moving objects. ACM Transactions on Database Systems, 2000, 25(1): 1-42

[19] Ding Z, Güting R H. Managing moving objects on dynamic transportation networks//Proceedings of the 16th International Conference on SSDBM. Santorini Island, Greece, 2004: 287-296

[20] Güting R H, Almeida V T, Ding Z. Modeling and querying moving objects in networks. VLDB Journal, 2006, 15(2): 165-190

[21] Ding Z, Zhou X. Location update strategies for network-constrained moving objects//Proceedings of the 13th International Conference on DASFAA. New Delhi, India, 2008: 644-652

[22] Wolfson O, Yin H. Accuracy and resource consumption in tracking and location prediction//Proceedings of the 8th International Symposium SSTD. Santorini Island, Greece, 2003: 325-343

[23] Civilis A, Jensen C S, Pakalnis S. Techniques for efficient road-network-based tracking of moving objects. IEEE Transactions Knowledge and Data Engineering, 2005, 17(5): 698-712

[24] Tiesyte D, Jensen C S. Efficient cost-based tracking of scheduled vehicle journeys//Proceedings of the 9th International Conference on MDM. Beijing, China, 2008: 9-16

[25] Ding Z, Huang G. Real-time traffic flow statistical analysis based on network-constrained moving object trajectories//Proceedings of the 20th International Conference on DEXA. Linz, Austria, 2009: 173-183

DING Zhi-Ming, born in 1966, Ph. D., professor, Ph. D. supervisor. His main research interests include database systems, mobile computing, embedded systems, spatial-temporal database/data mining, and information retrieval.

Background

The focus of this paper is on the trajectory index problem in Moving Objects Databases (MOD). MOD is a database which can track and manage the dynamically changing locations of moving objects such as cars, ships, flights, and pedestrians. An MOD system can manage huge numbers of moving objects so that index is crucial to query them efficiently.

In recent years, the moving object trajectory index problem has been intensely studied with a lot of methods proposed, such as STR-Tree, TB-Tree, FNR-Tree, and MON-Tree. However, nearly all existing trajectory indices take trajectory units as the basic index records, whose granularity is too fine. As a result, frequent index updates are needed when location updates occur so that the efficiency can be greatly affected.

To solve this problem, we propose a new index method, Dynamic Sketched-Trajectory R-Tree for Network-constrained Moving Objects (DSTR-Tree) in this paper. The DSTR-Tree divides the spatial-temporal space into equal-sized grid cells, transforms every trajectory into a sketched trajectory with each unit connecting two centers of the grid cells that the moving object travels through, and indices the sketched trajectory units as an R-Tree. Since the sketched trajectory has much coarser granularity than the original trajectory, the index updating cost can be greatly reduced — when a location update occurs, even though the original trajectory needs to be changed, the sketched trajectory may remain unchanged so that the DSTR-Tree does not need to be changed either.

We have implemented the DSTR-Tree in an object-relational database, PostgreSQL, and compared it with other trajectory indices in terms of index update costs, storage consumption, query response time, and selectivity. The experimental results show that the DSTR-Tree outperforms the previously proposed trajectory index methods in running moving objects databases with frequent location updates.

The paper was partially supported by National Natural Science Foundation of China under Grant Nos. 91124001 and 60970030.

The research team has been conducting research on Moving Objects Databases for about 10 years, with a series of results achieved such as the MODTN database model for network-constrained moving objects, the UTR-Tree and NDTR-Tree index methods for moving object trajectories, the Net-LUM and the ANLUM location update mechanisms for moving object tracking. We have also implemented a moving objects database system called NMOD-TFSA, which can manage trajectories of moving objects and extract traffic conditions through statistical analysis.

The work of the paper is focusing on the management of huge moving objects in databases, and can be used to support quick query processing of trajectories.