基于时空标签轨迹的 k 近邻模式匹配查询

许建秋，梁珺秀，秦小麟
（南京航空航天大学计算机科学与技术学院，江苏 南京 211106）

摘 要：时空标签轨迹在传统的时空轨迹数据基础之上融入了具有语义含义的标签信息，丰富了移动对象数据。针对该数据提出 k 近邻模式匹配查询，即在给定时间区间内匹配相应的模式且距离查询轨迹最近的 k 条轨迹。设计并实现标签 R 树（LR-Tree），即增加标签表并在 R 树每项中添加标签位图，及基于 LR-Tree 的 k 近邻模式匹配查询算法。通过真实数据和合成数据将 LR-Tree 与 3DR-Tree、SETI 及 TB-Tree 进行对比，实验表明 LR-Tree 具有更好的剪枝能力，从而验证了所提算法及索引的有效性。

关键词：时空标签轨迹；k 近邻算法；模式匹配；索引
中图分类号：TP311
文献标识码：A
doi: 10.11959/j.issn.1000-436x.2018063

k nearest neighbor pattern match queries over spatio-temporal label trajectories

XU Jianqiu, LIANG Junxiu, QIN Xiaolin
College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing 211106, China

Abstract: Spatio-temporal label trajectories extended traditional spatio-temporal trajectories with semantic labels. k nearest neighbor pattern match was proposed to return the k nearest trajectories that fulfilled the temporal pattern condition. The Label R-Tree (LR-Tree for short) was proposed, which appending a label table and adding label bitmap in each entry, and k nearest neighbor pattern match query algorithm based on LR-Tree was designed. Using both real and synthetic datasets, the LR-Tree was extensively evaluated in comparison with 3DR-Tree, SETI and TB-Tree. The experimental results demonstrate that LR-Tree showing better pruning ability, and verify the effectiveness of proposed algorithm and index.

Key words: spatio-temporal label trajectories, k nearest neighbor algorithm, pattern match, index

1 引言

随着智能电话及汽车导航系统等含 GPS 技术的设备广泛普及，每天都有大量的移动对象位置数据采集，目前，针对移动对象的查询主要包含 k 近邻查询、范围查询、反向 k 近邻查询和轨迹相似查询[1~4]等。

越来越多的应用把移动对象轨迹数据与对象的文本属性关联起来，得到时空文本属性随时间变化的移动对象。目前，针对包含文本属性的移动对象提出了语义轨迹[5]、活动轨迹[6]、符号轨迹[7]等，衍生出应用广泛的关于时空文本的移动对象查询[8~10]，即结合时空查询和文本查询以寻求最优结果的混合查询，如可以作为路径推荐的活动轨迹相似查询（ATSQ, activity trajectory similarity query）[6]、k 近邻关键字查询[11]、符号空间轨迹混合查询[12]等。

在某些应用场景下，人们更倾向于提出给定模式的查询，如“查找在 15:00~18:00，距离查询轨迹最近的前 k 个从 P_4 出发经过若干景点后先经过 P_1 后经过 P_2 的移动对象”。其中，P 表示景点，查询

收稿日期：2017-09-05；**修回日期**：2018-03-16
通信作者：许建秋，jianqiu@nuaa.edu.cn
基金项目：中央高校基本科研业务费专项基金资助项目（No.NS2017073）
Foundation Item: The Central University Basic Business Expenses Special Funding for Scientific Research Projects (No. NS2017073)

中提出的由 P_4 出发先经过 P_1 和 P_2 是有顺序要求的，且时间在给定的查询范围内，这种含有顺序的文本属性请求被称为给定模式，目前还未有上述查询请求的研究成果发表。将这种位置随时间变化且包含文本信息到时间区间映射的轨迹称为时空标签轨迹，将这类查询定义为基于时空标签轨迹的 k 近邻模式匹配查询。与现有查询相比，k 近邻模式匹配查询存在以下特点：1) 模式匹配能处理较复杂的文本属性查询请求；2) 将模式匹配与时空属性查询相结合，同时解决不同维度的查询。

如图 1 所示，查找$[t_1, t_2]$时间范围内由 P_4 出发，先经过 P_1，后经过 P_2 且距离 *Qtraj* 最近的轨迹，可以看出在给定查询时间区间内，距离 *Qtraj* 最近的轨迹为 $traj_2$，而在时间区间$[t_1, t_2]$内，$traj_2$ 不存在匹配给定模式的子轨迹段，在 $traj_1$ 和 $traj_3$ 中都存在匹配模式的轨迹段，而 $traj_1$ 距离 *Qtraj* 更近，因此，返回 $traj_1$ 为查询结果。

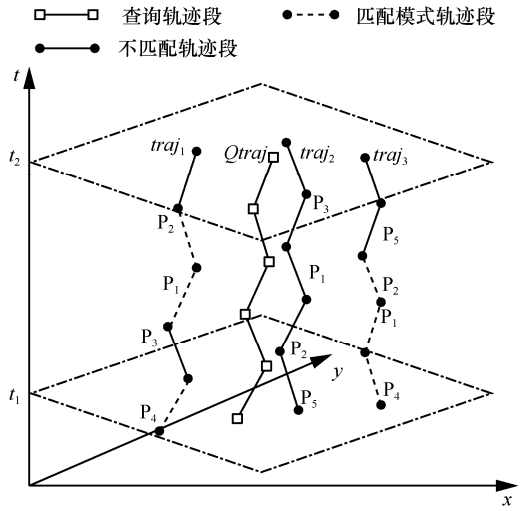


图 1　k 近邻模式匹配查询

为了解决 k 近邻模式匹配查询请求，本文主要贡献如下。

1) 将时空属性与随时间变化的文本属性结合，提出时空标签轨迹。

2) 提出基于时空标签轨迹的 k 近邻模式匹配查询（KPMQ, k nearest neighbor pattern match query），并给出了 KPMQ 的定义。

3) 提出标签 R 树索引，并设计基于标签 R 树的深度优先遍历算法处理基于模式匹配的 k 近邻查询，提高上述查询效率。

2　相关工作

本文主要研究时空标签轨迹的查询及索引问题，相关工作包括 2 个部分：文本信息轨迹以及时空文本索引。

2.1　文本信息轨迹

目前，针对含文本属性移动对象已有大量研究，主要包含活动轨迹、语义轨迹、符号轨迹等。活动轨迹是由 Zheng 等[6]提出的表示包含关于特定地点处的用户活动信息的新类型的轨迹数据。针对这种轨迹提出了基于等级的活动轨迹搜索及基于顺序的活动轨迹搜索等查询。但是这种轨迹数据仅适用于描述用户活动，并不能描述其他文本类型的轨迹数据。语义轨迹是由 Alvares 等[5]提出来的表示用注释标记整个轨迹或其部分轨迹段的轨迹，每个注释表示在其对应点处用户的状态或行为，用这些状态或行为来丰富几何轨迹。语义轨迹的概念已经吸引了在不同领域工作的许多研究者的兴趣，如数据分析、概念建模、语义网、隐私等，研究问题主要包括语义[13]、移动知识发现[14]、隐私保护[15] 3 个方面。很多学者都考虑研究轨迹的停留点处的语义信息，在文献[5]中，采用 SmoT 算法提取轨迹中的停留点，在文献[16]中用 CBSMOT 算法提取。语义轨迹关注点在轨迹的拐点处，而忽略了在轨迹的运动过程中可能产生的有用信息。符号轨迹是由 Güting 等[7]提出的不包含空间位置属性，轨迹表现为随时间变化的标签。不同于上述 2 种轨迹，符号轨迹可以用于表示个人在时间区间内的活动等，例如，表示从家到工作地点的交通工具，在旅行期间访问的景点等，并针对符号轨迹提出了模式匹配查询和符号轨迹的重写操作[17]。在文献[12]中提出了一种结合符号轨迹和空间轨迹的混合查询，从符号的维度提取满足给定模式的符号子轨迹，由得到的子轨迹的时间范围来限制空间维度，当子轨迹空间维度与几何条件匹配时，则将其作为结果返回，最终得到同时满足符号和时空条件的轨迹集合，但这种混合查询仅考虑轨迹本身的属性，不考虑轨迹间的相互关系。

2.2　时空文本索引

当查询记录过大时，为提高移动对象的查询效率，索引是其中不可或缺的环节。常见的索引包括 R 树[18]、网格索引及其变体等。3DR-Tree 是 R 树在时间空间上的扩展，由时间和空间 3 个维度组成，

它将时间作为额外的维度，处理时间范围的查询，3DR-tree 中每个节点中的项包含指向所有子节点的指针及其最小边框矩形（MBR, minimal bounding rectangle），MBR 为覆盖其所有子节点的最小边框矩形。SETI (scalable and efficient trajectory index)由 Prasad 等[19]提出，索引使用简单的 2 层索引结构，将时间和空间属性分开，分别对时间维度和空间维度进行索引，将空间维度按固定大小的网格划分为不重叠的单元，分布在同一单元格的轨迹段保存在同一数据文件中，对每个单元格中的所有轨迹段的时间属性建立一维 R-Tree 索引，因此，在同一个数据文件中的轨迹段在空间上是相近的。TB-Tree（trajectory bundle Tree）由 Dieter[20]提出，每个叶子节点只保存相同轨迹的轨迹段，这种结构保存了移动对象轨迹，忽略了轨迹空间属性，导致同一个移动对象分布在不同的叶子节点中，导致节点重叠，空间分辨能力减弱。IR-Tree 由 Cong[21]提出，本质上是由倒排文件扩展的 R 树，在树的每个节点中含有指向倒排文件的指针，文件描述了存储在节点中对象的活动。GAT（grid index for activity trajectories）由 Zheng 等[6]提出，建立分层网格，将整个空间划分为 d-Grid（$2^d \times 2^d$ 个网格），并进一步构建(d-1)- Grid, …, 1-Grid，由 d-Grid 往上层构建倒排索引表示各个活动出现的网格，同时，在 d-Grid 层每个网格中，为每个出现的活动建立倒排索引表示哪些轨迹包含当前活动，最后建立一个数据结构表示每条轨迹中包含的所有活动。在这些包含文本属性的索引中，文本属性对应的都是轨迹的点，而不是整个时间区间，因此，不适用于时空标签轨迹。

综上所述，符号轨迹仅考虑语义信息忽略了位置，不支持同时具有语义和时空信息的查询。语义轨迹和活动轨迹解决了时间点处的位置语义描述，但没有考虑且不支持基于时间区间的语义属性。针对该问题提出时空标签轨迹模型，为有效支持基于时空标签轨迹的 k 近邻模式匹配查询，设计标签 R 树索引以解决传统移动对象索引结构不能对映射至时间区间上的语义属性进行很好的描述的问题。

3 问题定义

KPMQ 返回给定时间区间内满足给定模式且与查询轨迹 *Qtraj* 距离最近的前 k 条轨迹。移动对象包含的文本信息以标签（label）的形式保存，为了明确所解决的问题，下面给出相关定义。

定义 1 时空标签轨迹。$traj=<[I_1, l_1, loc1_1, loc2_1], \cdots, [I_n, l_n, loc1_n, loc2_n]>$，其中，$I_j$ 表示时间区间，l_j 表示 I_j 对应的标签，$loc1_j$ 表示时间区间开始时刻移动对象的位置，$loc2_j$ 表示时间区间结束时刻移动对象的位置，其中，每个$[I_j, l_j, loc1_j, loc2_j]$为轨迹单元。

定义 2 模式。$P=<p_1,\cdots, p_n>$，其中，p_i 为以下 2 种形式之一。

1) p_i 为(l)或()，其中，l 为标签，称这种为单元模式。

2) p_i 为*、+、[p]、[$p_i|p_j$]、[p]+、[p]*或[p]?，称这种为简单模式，简单模式中的 p 为单元模式或简单模式，简单模式中符号与正则表达式中符号表示意思相同。

每个单元模式表示一个事件，例如，将图 1 查询模式表示为<(P_4), *, (P_1) , (P_2),*>，为了表示事件的顺序性，引入正则表达式来描述完整的事件。

定义 3 模式匹配（PMatch）。模式 $P=<p_1,\cdots, p_n>$ 与包含模式 $P'=<p'_1, \cdots, p'_m>$ 的轨迹 *traj* 模式匹配。

1) $\forall i \in [1, n]$，$\exists j \in [1, m]$：$p_i = p'_j$。

2) $\forall i_1, i_2 \in [1, n]$，$i_1 < i_2$，$\exists j_1, j_2 \in [1, m]$：$p_{i_1} = p'_{j_1}$，$p_{i_2} = p'_{j_2}$，且 $j_1 < j_2$。

综上，当 *traj* 中存在轨迹段的标签信息与 P 中标签的内容相同且顺序一致时，称轨迹与 P 模式匹配。如图 1 所示，轨迹 $traj_1$ 中的<(P_4), (P_3), (P_1), (P_2)>子轨迹段模式与模式<(P_4), *, (P_1), (P_2), *>匹配。

定义 4 k 近邻模式匹配查询(KPMQ)。给定时间区间 I，查询轨迹 *Qtraj* 以及模式 P，返回轨迹集合 D 中大小为 k 的子集 D'。

1) $\forall traj \in D'$，PMatch(Interpolate(I, *traj*), P)。

2) 对于轨迹集合 B，$\forall t \in B$, PMatch (Interpolate(I, t), P)，$\forall traj' \in B-D'$，有 Distance (*Qtraj*, *traj'*) ≥Distance(*Qtraj*, *traj*)。

Interpolate (I, *traj*)表示在时间区间 I 内 *traj* 的子轨迹段，B 为 D 中匹配模式的轨迹集合，D'轨迹集合为 B 中距离 *Qtraj* 最近的前 k 条轨迹组成的子集，则 D' 即为 k 近邻模式匹配查询的结果。

4 标签 R 树

标签 R 树（LR-Tree）形式上由一个 3DR-Tree

和一个标签表组成，与 3DR-Tree 的不同点如下。1) 每个 R 树节点的项（entry）中增加一个固定大小的位图，位图中的信息“0/1”代表了当前项指向的子节点的标签存在性，当位图的位为 1 时，表示位图中的位对应的标签表中的标签存在，为 0 则不存在。2) 位图的每个位通过散列函数对应到标签表中的一个或多个位置，表中每位保存一个标签。叶子节点项位图中的每一位表示对应的移动对象的标签存在性，非叶子节点项中的位图通过其子节点的位图执行按位或操作得出。

4.1 LR-Tree 结构

4.1.1 LR-Tree 节点项

LR-Tree 叶子节点项表示为(rid, MBR, bitset)，其中，rid 表示项所指向的下层节点，MBR 即为将所有子节点项中的 MBR 包围的最小矩形框，即为当前项的 MBR，bitset 由节点项所指向的子节点的所有位图执行按位或操作得到。叶节点项表示为(tid, MBR, bitset)，其中，tid 表示项所指向的移动对象，MBR 为将移动对象包围的最小矩形，bitset 为指向的移动对象包含的标签到标签表的映射计算所得的位图。

如图 2 所示，N 表示节点，E 表示节点中的项，节点 N_2 中 2 个项 E_3、E_4 的位图即为对应 N_4、N_5 中所有位图进行按位或操作所得，每个项包含指针指向对应的下层子节点。

LR-Tree 在 R-Tree 的基础上增加标签信息，使在原有索引结构基础上，不仅能表示地理位置，还能包含轨迹标签信息存在性，因此，在查询过程中可以同时考虑时空和文本约束条件，来提高查询效率。增加的标签信息在节点项中以位图为单位存储，占用空间大小以 bit 为单位，位图大小的设定可以根据标签表中标签数量大小改变，使索引所占空间较 R 树空间增长较小。

4.1.2 LR-Tree 位图

移动对象的全部标签保存在 LR-Tree 标签层的标签表中，标签表与项中的位图通过散列函数相关联，标签通过散列函数映射到位图中，位图中每位对应标签表中的一个或多个标签，当标签出现在移动对象中时，将通过散列函数映射到位图的比特置 1。

LR-Tree 中位图长度在初始化后不能改变，当标签表长度大于位图长度时，可以通过散列函数将表中不同的标签映射到位图中的同一位。如图 2 所示，LR-Tree 固定位图长度为 5，而标签表长度为 6 时，E_7 中的 1 号位指向表中 1 号标签，而 E_{13} 中 1 号位指向标签表中 6 号标签，可以看出 1 号和 6 号标签同时映射到位图中的 1 号位，因此，当位图 1 号位为 1 时，表示 1 号标签和 6 号标签中至少有一个存在。

4.2 LR-Tree 构建

在标签 R 树索引构造过程中，采用批量更新（bulkload）[22]构造，首先将所有轨迹的 MBR 按空

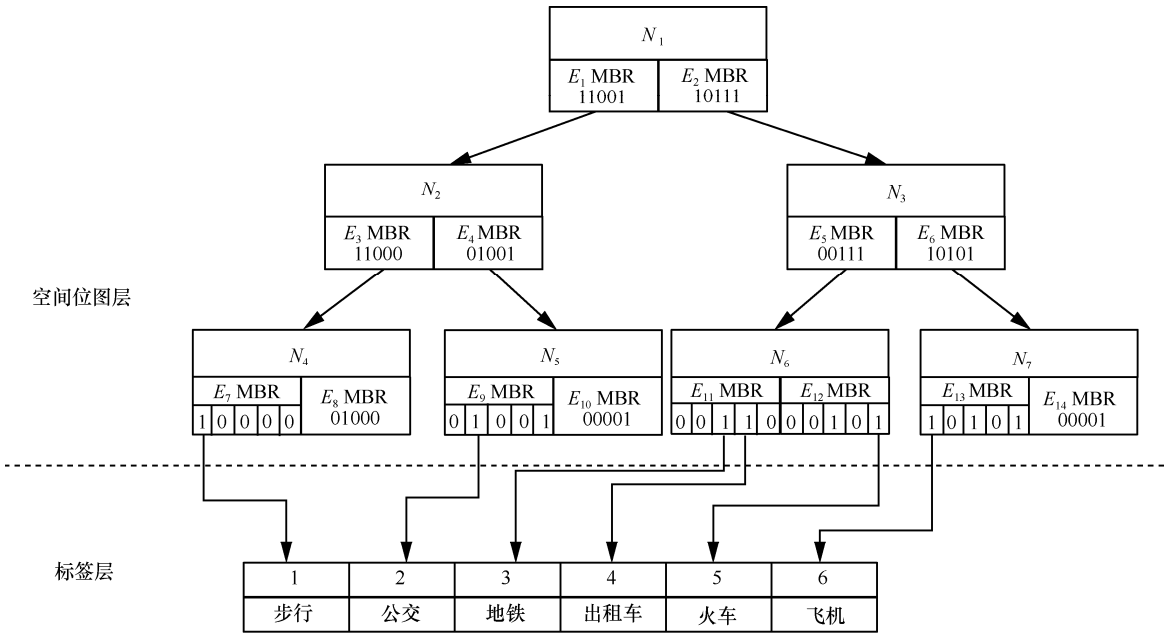


图 2　标签 R 树

间位置顺序进行排序，将排序后的项依次插入到空节点 N 中，当 N 中项的数目达到上限或当前插入的项距离 N 太远时，将 N 插入至 LR-Tree 中，并新建一个节点 M，将该项插入 M 中。在将节点插入 LR-Tree 时，对当前节点项中所有位图进行或操作，得到的结果作为插入项的位图。

对于直接插入算法（directly insert），将待插入轨迹直接插入 LR-Tree 中，在建树过程中先保留叶子节点处位图，只考虑空间属性自底向上建树，将内部节点位图均置为 0。在建树完成后，自底向上更新 LR-Tree 中所有位图。相较于直接插入，批量更新方法避免了对每一个插入项都产生一次 I/O，提高了标签 R 树索引构建效率。

5 k 近邻模式匹配查询

5.1 算法基本思想

k 近邻模式匹配查询（KPMQ）返回在时间区间 I 内，匹配给定模式 P，且距离查询轨迹 $Qtraj$ 最近的前 k 条轨迹。利用 LR-Tree 处理该查询，采用深度优先方法由根节点开始遍历 LR-Tree。设置一个长度为 k 的优先队列 Q，存储当前找到的 k 个结果。在访问 LR-Tree 中的每个节点时，以 I、P 中标签对应位图、Q 中移动对象的 MBR 与 $Qtraj$ 的 MBR 间的最大距离值，作为 LR-Tree 的剪枝条件。在遍历 LR-Tree 后，保存在优先队列 Q 中的 k 条轨迹匹配模式 P 且距离查询轨迹 $Qtraj$ 最近，即为 k 近邻模式匹配查询的结果。在查询过程中充分利用已知限制条件，遍历 LR-Tree 从候选节点裁剪掉不可能的部分，提高查询效率，具体步骤如图 3 所示。

定义 5 位图匹配（BMatch）。对于给定模式 P 的位图 $Tbit$ 和 LR-Tree 项中位图 $Ebit$，$\forall\, Tbit[i]=1$，若 $Ebit$ 中对应的位 $Ebit[i]=1$，则位图 $Ebit$ 与给定模式 P 位图匹配，反之若 $\exists Tbit[i]=1$ 而 $Ebit[i]\neq 1$，则不匹配。

基于 k 近邻模式匹配查询如算法 1 所示，主要包括以下步骤。

1) 设定长度 k 的优先队列 Q 保存当前查询结果，栈 S 保存查询节点，将 LR-Tree 根节点入栈。

2) 对于每个出栈的节点 N，判断 N 中的项是否与查询时间区间相交且位图匹配。$Q.top()$处保存当前队列中距离查询轨迹最远的移动对象，若当前队列 Q 长度为 k，将最大距离值定为 $roof$，进一步计算项的 MBR 与查询轨迹的 MBR 间距离，将距离值大于 $roof$ 的项过滤，若 Q 长度小于 k 则转入步骤 3)。

3) 对于步骤 2)中满足条件的项，如算法 2 所示，若当前节点为非叶节点，则将项按与查询轨迹 $Qtraj$ 的 MBR 间的距离值顺序入栈；若节点为叶子节点，判断项指向移动对象进行轨迹插值后的子轨迹段 segtra 是否匹配模式 P，对于匹配项进一步计算插值后的 $Qtraj$ 与 segtra 间的距离，若距离值小于 $roof$ 则将其加入优先队列中，并更新 $roof$ 值作为之后的判断条件。

4) 在遍历 LR-Tree 后，Q 中得到的所有轨迹则为匹配模式且距离查询轨迹最近的前 k 条轨迹。

算法 1 K 近邻模式匹配查询（KPMQ）

输入 查询模式 P
查询时间区间 I
查询轨迹 $Qtraj$
查询结果个数 k
移动对象集合 D
LR-Tree

输出 优先队列 Q

1) $Q\leftarrow\varnothing$； $S\leftarrow\varnothing$；
2) S.push(LR-Tree.RootNode);
3) QMBR$\leftarrow Qtraj$.MBR();
4) while (S 不为空)
5) $Node\leftarrow S.top()$; $S.pop()$;
6) for each $entry\in Node$
7) EMBR$\leftarrow entry$.MBR();
8) if(($I\cap Qtraj.Interval$)与 EMBR.$Interval$ 相交)
9) if(BMatch($P.Tbit$, $E.Ebit$))

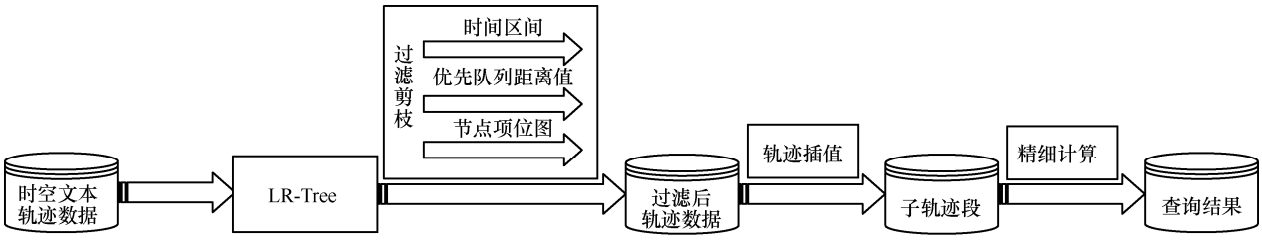


图 3 k 近邻模式匹配查询过程

10) if(|*Q*| = *k*)
11) *roof* = Distance(*Q.top*(), *Qtraj*)
12) if(Distance(QMBR, EMBR)<*roof*)
13) Update(*S*, *Qtraj*, *Q*, entry, *P*);
14) end if
15) else
16) Update(*S*, *Qtraj*, *Q*, *entry*, *P*);
17) end if
18) end if
19) end if
20) end for
21) end while
22) return *Q*

5.2　过滤过程

由 *k* 近邻查询过程得到如下几个重要引理。

引理 1　给定时间区间 *I* 和查询轨迹 *Qtraj*，若节点项时间区间 *E.Interval*∩(*I*∩*Qtraj*. *Interval*)=∅，将以项指向的节点为根节点的子树裁剪。

引理 2　给定模式的位图 *Tbit* 与节点项中位图 *Ebit*，若 BMatch(*Tbit*, *Ebit*)=false，则将项指向的节点从树中裁剪。

考虑给定模式 *P*，提取 *P* 中所有标签，并通过标签表及对应的散列函数得到模式位图 *Tbit*，对于与 *Tbit* 位图不匹配的节点项，则表示当前节点项的所有叶子节点中的位图，必然存在一个或一个以上对应的位不为 1，则所指向的时空标签轨迹中必然不存在查询模式中的所有标签，因此将以该项指向的子节点为根的子树裁剪。

引理 3　节点项最小边框矩形 EMBR、查询轨迹 QMBR 及优先队列的最大距离值 *roof*，若 Distance (QMBR, EMBR)≥*roof*，则将项指向的节点从树中裁剪。

在遍历标签 R 树的过程中，将符合查询条件的时空标签轨迹保存在优先队列 *Q* 中，*Q* 存储当前查询中找到的满足条件且距离查询轨迹 *Qtraj* 最近的前 *k* 条轨迹。因此，若节点项 EMBR 与查询轨迹 QMBR 间的距离值大于或等于 *roof*，则查询轨迹与节点项 EMBR 中包含的所有轨迹的距离必然不小于 *roof* 值，不可能作为距离最近的前 *k* 条轨迹保存在 *Q* 中，因此，将项所指向的节点从树中裁剪。

算法 2　Update
输入　栈 *S*
　　查询轨迹 *Qtraj*
　　优先队列 *Q*
　　项 *entry*
　　查询模式 *P*
输出　更新 *entry* 后的 *S*、*Q*

1) if (*Node* 为内部节点)
2) *S*.push(*entry.ptr*);//插入后的栈中的 *entry* 按距离值排序
3) else
4) if(PMatch(*P*, Interpolate(*I*,*Qtraj*))
5) if(Distance(Interpolate(*I*,*Qtraj*), Interpolate(*I*, *entry.traj*))<*roof*)
　//Interpolate 表示取查询时间内的轨迹段
6) *Q*.push_back(*entry.tid*);
7) 更新 *Q* 中最大距离值 *roof*;
8) end if
9) end if
10) end if

采用深度优先方法遍历 LR-Tree，对于每个出栈的节点，在将其所有子节点入栈前，在步骤 2)中先将其中不符合条件的项过滤。对于筛选后的子节点按照它与 *Qtraj* 的 MBR 间的距离排序后从大到小入栈，确保下回出栈的节点是当前栈中距离 *Qtraj* 最近的矩形框，尽早利用优先队列的 *roof* 值剪枝，提高 LR-Tree 空间剪枝能力。

5.3　精细计算过程

对于筛选过后要进行精细计算的每条轨迹，进行模式匹配与距离计算查询前，先进行轨迹插值计算得到子轨迹段。如图 4 所示，查询时间区间箭头宽度表示查询区间范围，利用查询区间范围对每条轨迹进行约束，取轨迹定义时间区间与查询时间区间交集内的子轨迹段中的信息，包含时空信息及标签信息，图 4 中虚线部分即为插值后的子轨迹段。

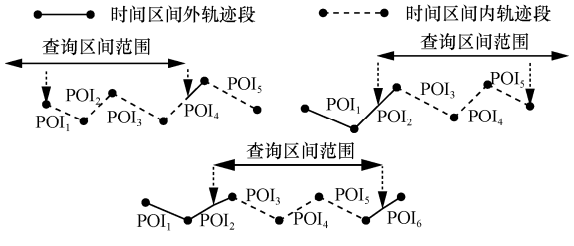


图 4　轨迹插值

在插值得到子轨迹段后，先对子轨迹段进行模式匹配，将由正则表达式表示的查询模式 *P* 转化为

NFA，并判断每个子轨迹段能否到达 NFA 中的终态，将可到达终态的轨迹段取出，再进行距离计算，由优先队列的 *roof* 值对轨迹段进行筛选，最终优先队列 Q 中保存的轨迹即为查询结果。

6 实验及分析

为了验证提出的标签 R 树处理 k 近邻模式匹配查询算法的有效性，采用 C++语言在 Linux 环境下，扩展可扩充数据库 SECONDO[23]，向数据库中添加操作代码，实现基于时空标签轨迹的 k 近邻模式匹配查询。实验环境为：Intel(R) Core(TM) I3-2120 CPU@ 3.30 GHz，4 GB 内存，Ubuntu14.04 64bit 操作系统。

6.1 实验数据

实验中使用真实数据集和合成数据集做对比实验。2 种数据均表示语义属性对应至时间区间上的时空轨迹。真实数据集是微软亚洲研究院 Geolife[24]。项目组收集 182 个用户 3 年内的 GPS 数据，其中，部分用户用交通方式标记了他们的运动数据（如步行、火车等）。数据记录真实世界中不同出现频率的标签，因此，能准确地表示不同标签在查询中影响。合成数据集是 SECONDO 系统中人工合成地铁轨迹的数据 *Train*，包含 562 辆地铁的运动轨迹，为每条轨迹贴上不同的合成标签，其中，标签为 1~50 中的任意数字，每个数字表示不同的标签，每种标签出现的概率相同，每条轨迹包含 5~10 个标签。地铁数据模拟带标签的数据集合，从大规模数据及多种标签角度考虑对查询效率的影响。采用数据加倍方法对轨迹进行 x、y 方向上的偏移，得到平移的轨迹数据。数据集信息如表 1 所示，*Train*(n)是在 *Train* 的基础上进行 x、y 方向上平移得到的 n^2 倍数据。

表 1 数据集的数据统计

名称	轨迹数量	轨迹单元数
GeoLife	4 300	4 124 326
Train	562	51 544
Train(10)	56 200	5 154 400
Train(15)	126 450	11 597 400
Train(20)	224 800	20 617 600
Train(30)	505 800	46 389 600
Train(50)	1 405 000	128 860 000

6.2 索引构建

6.2.1 索引构建性能比较

本节实验研究比较采用批量更新算法和直接插入算法在构建 LR-Tree 上的性能比较，使用合成数据集构建 LR-Tree，实验结果如图 5 所示。可以看出直接插入算法的索引构建开销远大于采用批量更新算法。随着数据量的增加，直接插入算法的构建时间呈显著增长趋势，而批量更新算法增长更为缓慢，相较于直接插入方法，批量更新方法的构建时间降低了约 90%。

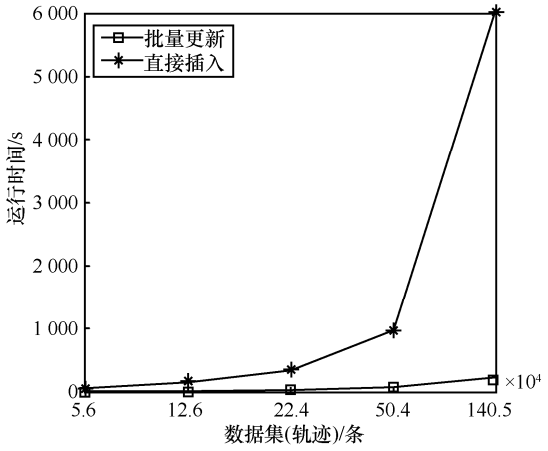


图 5 批量更新直接插入性能比较

6.2.2 位图长度对空间影响

本部分实验比较 LR-Tree 中设置不同长度的位图对索引空间大小的影响。LR-Tree 保存在以 1K 为单位的数据块中。图 6 中 st-status 表示时空属性占用空间大小，bitmap 表示位图占用空间大小。由图 6(a)可以看出位图长度保持在 10~30 时，LR-Tree 大小保持不变，长度为 40~50 时，位图空间增大。由于位图占用空间较小，导致位图长度在一定范围内增长时，总体增长的空间大小不超过数据块剩余空间大小，位图占用空间无明显增加，因此实验设置位图长度默认值为 30。

由图 6(b)可以看出随着数据量增加，树的节点增加，位图所占空间增加，但位图所占大小始终保持在整体大小的 4%~7%之间。

6.3 查询性能实验

实验测试 LR-Tree 的查询处理性能，包括数据量、模式中不同频率的标签、标签数量、给定时间区间长度和返回结果数 k。对不同的参数进行测试，比较 k 近邻模式匹配查询的平均 I/O 次数及 CPU 时间，测试 LR-Tree 的剪枝能力，从而比较其处理查询性能，实验参数设置如表 2 所示。

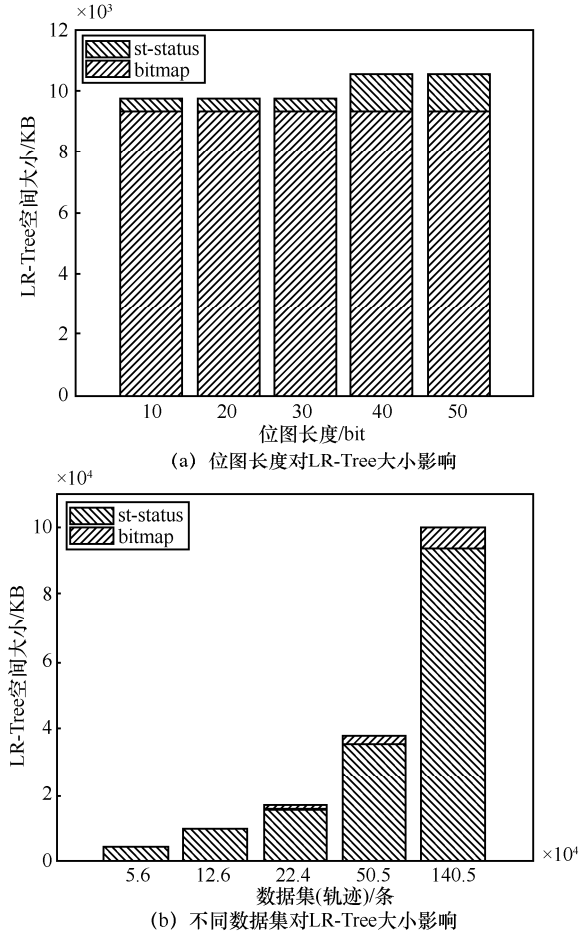


图 6　LR-Tree 位图—时空占用比

表 2　　**实验参数**

参数名	参数值
标签	飞机, 自行车, 火车, 步行, 出租车, 汽车, 地铁, 公交
标签数量	1, 3, 5, 7, 9
时间区间长度	1 h, 2 h, 3 h, 1d, 1m
返回结果数	1, 100, 300, 600, 1 000
数据集(轨迹)/条	56 000, 126 000, 224 000, 505 000, 1 405 000

6.3.1　数据量对算法性能影响

本部分实验以不同数据集为实验数据，在相同查询时间、返回结果及查询模式下，比较 4 种索引在不同数量级的数据集上的性能，结果如图 7 所示。可以看出随着数据量增长，相较于另外 3 种索引，LR-Tree 的 I/O 代价及 CPU 时间增长更为缓慢，相较 3DR-Tree 降低了 50%～70%的 I/O 及 CPU 代价。

6.3.2　查询标签数量对算法性能影响

图 8(a)和图 8(b)显示了标签数量对查询算法的影响，随着标签数量增加，LR-Tree 的 I/O 代价增加不明显并逐渐趋于稳定，而另外 3 种索引空间剪枝能力大幅度下降，I/O 代价急剧增加。由于 LR-Tree 的剪枝同时包含标签和时空，随着标签数量增加，导致匹配给定模式的移动对象数量急剧减少，使空间剪枝能力降低，I/O 代价略微增加。因此，当标签数量较多时，LR-Tree 较另外 3 种索引具有明显的剪枝能力，当查询模式中标签数大于 3 时，查询效率提高了 90%以上。

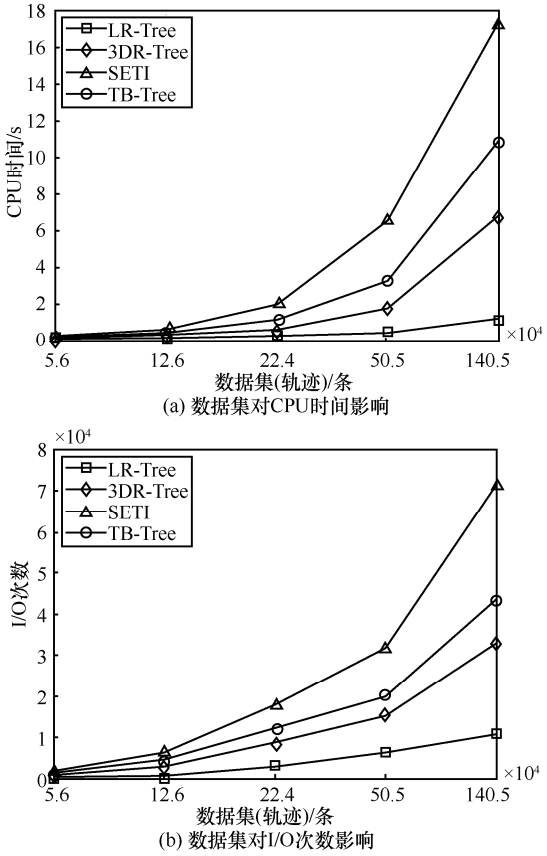
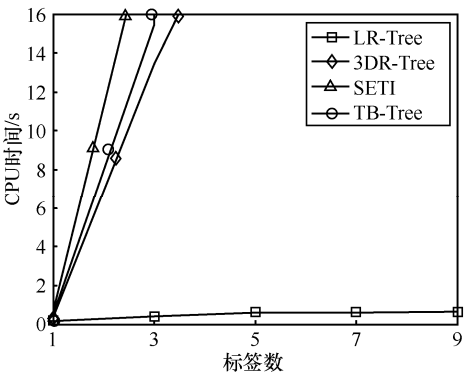


图 7　数据集对查询性能影响

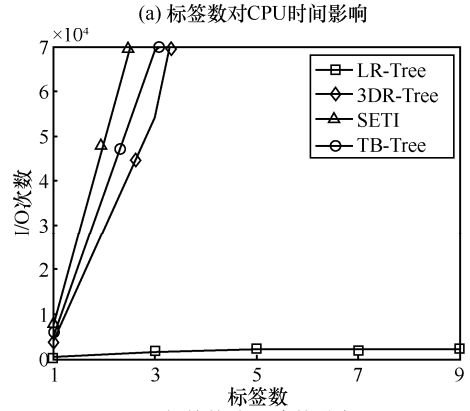
在本节实验中，同时对比基于 LR-Tree 的深度优先算法（DF）和广度优先算法(BF)受标签数量变化的影响，结果如图 8(c)所示。可以看出相较于 BF 算法，采用 DF 算法的 I/O 次数降低了 50%以上，体现了基于 LR-Tree 的 DF 算法的优越性。

6.3.3　查询模式对算法性能影响

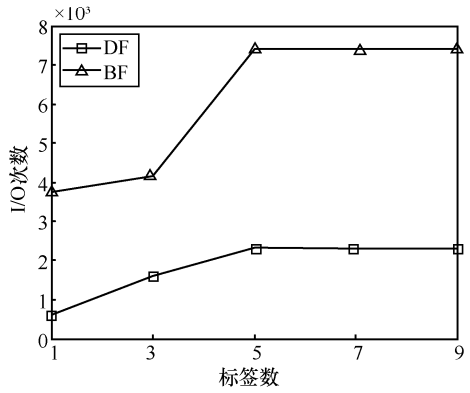
本节实验以 Geolife 为实验数据，统计不同标签出现次数，得到的结果如表 3 所示。实验得到的结果如图 9(a)和图 9(b)所示，对比表 3 中标签出现次数，可以看出相较于另外 3 种索引，对于出现频率较低的标签，如飞机、自行车及火车、LR-Tree 能有效地使用关键字剪枝，减少 I/O 代价和 CPU 时间，相较于 3DR-Tree 提高 30%及以上的查询效率。



(a) 标签数对CPU时间影响



(b) 标签数对I/O次数影响



(c) DF/BF算法受标签数影响下I/O次数的变化

图 8　标签数对查询算法影响

表 3　　**标签频数**

标签名称	标签数量	标签百分比
飞机	18	0.19%
出租车	813	9.10%
火车	147	1.65%
步行	574	6.43%
自行车	477	5.34%
汽车	1594	17.86%
地铁	1564	17.52%
公交	3740	41.90%

6.3.4　给定时间区间长度对算法性能影响

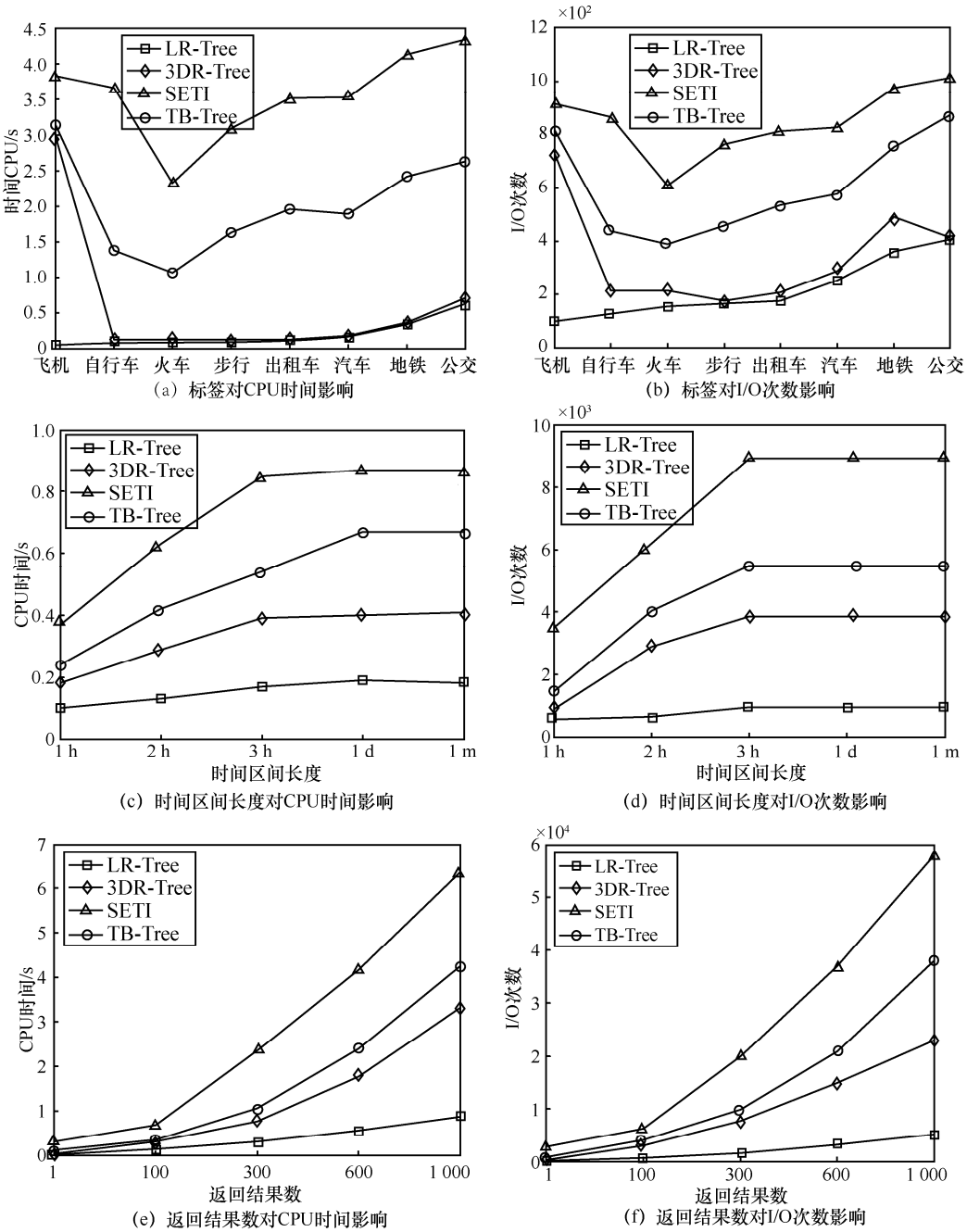
合成数据中轨迹的定义时间区间长度在 2~3h 内，在本次实验中将给定时间区间的开始时刻定义为查询轨迹的开始时刻，得到的结果如图 9(c)和图 9(d)所示。可以看出当给定时间区间的长度小于查询轨迹的长度时，随着时间区间长度增加 I/O 代价增加，当大于查询轨迹长度时，则保持不变。由于过滤条件中包含查询时间区间，随着时间区间增大，过滤的节点减少导致 I/O 代价增加，当给定时间区间大于查询轨迹定义时间区间长度时，过滤的时间区间固定为查询轨迹的定义时间区间，因此过滤时间区间条件不变，对 I/O 次数不产生影响。可以看出相较于另外 3 种索引，LR-Tree 产生了更少的 I/O 次数及 CPU 时间，表现出更好的剪枝能力，相较于 3DR-Tree 提高 40%~50%的查询效率。

6.3.5　返回结果数 k 对算法性能影响

本部分实验设置不同 k 值，得到基于 4 种不同索引的查询算法性能对比如图 9(e)和图 9(f)所示，可以看出随着返回结果数量增加，I/O 次数增加。由于 4 种索引都包含通过优先队列的阈值剪枝的方法，k 值影响空间剪枝能力，因此，当 k 值较小时，能尽早地利用优先队列中的最大距离值对大于最大距离值的节点进行剪枝，当 k 值增加时，空间剪枝能力降低，I/O 次数及 CPU 代价增加，相较于 3DR-Tree 降低了 60%～80%的 I/O 次数及 CPU 代价。

7　结束语

本文设计了一种基于标签 R 树的 k 近邻模式匹配查询算法，增加一个标签表并在已有 R 树节点项中加入判断标签是否存在的位图，使 R 树索引仅在已有空间剪枝能力基础上，增加标签属性的剪枝能力，给出 LR-Tree 的批量更新方法并与直接插入建树方法进行比较，体现了批量更新方法的高效性。对于 k 近邻模式匹配查询，用真实数据和合成数据比较 LR-Tree 与 3DR-Tree、SETI 及 TB-Tree 索引在不同参数下的剪枝能力。实验结果表明 LR-Tree 在位图空间仅占据 4%~7%空间的基础上，能够更有效地支持 k 近邻模式匹配查询，并在数据量增长较大时有良好的可扩展性。本文算法考虑的是模式完全匹配，在接下来的工作中可以考虑当模式部分匹配时，轨迹间相似程度的算法。



(a) 标签对CPU时间影响　(b) 标签对I/O次数影响

(c) 时间区间长度对CPU时间影响　(d) 时间区间长度对I/O次数影响

(e) 返回结果数对CPU时间影响　(f) 返回结果数对I/O次数影响

图 9　不同参数对 CPU 时间及 I/O 次数影响

参考文献：

[1] XUAN K, ZHAO G, TANIAR D, et al. Continuous range search query processing in mobile navigation[C]//The IEEE International Conference on Parallel and Distributed Systems. 2008: 361-368.

[2] GAO Y J, LI C, CHEN G C, et al. Efficient *k*-nearest- neighbor search algorithms for historical moving object trajectories[J]. Journal of Computer Science and Technology, 2007, 22(2): 232-244.

[3] BENETIS R, JENSEN C S, KARČIAUSKAS G, et al. Nearest neighbor and reverse nearest neighbor queries for moving objects[C]//The IEEE International Symposium on Database Engineering & Applications. 2002:44-53.

[4] TIAKAS E, PAPADOPOULOS A N, NANOPOULOS A, et al. Searching for similar trajectories in spatial networks[J]. Journal of Systems & Software, 2009, 82(5): 772-788.

[5] ALVARES L O, BOGORNY V, KUIJPERS B, et al. A model for enriching trajectories with semantic geographical information[C]//The ACM International Symposium on Advances in Geographic Information Systems. 2007: 22.

[6] ZHENG K, SHANG S, YUAN N J, et al. Towards efficient search for activity trajectories[C]//The IEEE International Conference on Data Engineering. 2013: 230-241.

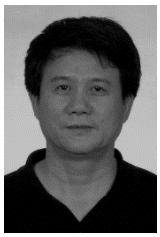
[7] GÜTING R H, VALDÉS F, DAMIANI M L. Symbolic trajectories[J].

ACM Transactions on Spatial Algorithms & Systems, 2015, 1(2): 1-51.
[8] WU D, YIU M L, JENSEN C S, et al. Efficient continuously moving top-*k* spatial keyword query processing[C]//The IEEE International Conference on Data Engineering. 2011: 541-552.
[9] ZHENG B, YUAN N J, ZHENG K, et al. Approximate keyword search in semantic trajectory database[C]//The IEEE International Conference on Data Engineering. 2015: 975-986.
[10] HAN Y, WANG L, ZHANG Y, et al. Spatial keyword range search on trajectories[M]//Database Systems for Advanced Applications. Springer International Publishing. 2015: 223-240.
[11] FELIPE I D, HRISTIDIS V, RISHE N. Keyword search on spatial databases[C]//The IEEE International Conference on Data Engineering. 2008: 656-665.
[12] DAMIANI M L, ISSA H, GÜTING R H, et al. Hybrid queries over symbolic and spatial trajectories: a usage scenario[C]//The IEEE International Conference on Mobile Data Management. 2014: 341-344.
[13] SPACCAPIETRA S, PARENT C, DAMIANI M L, et al. A conceptual view on trajectories[J]. Data & Knowledge Engineering, 2008, 65(1): 126-146.
[14] ZHANG C, HAN J, SHOU L, et al. Splitter: mining fine-grained sequential patterns in semantic trajectories[J]. Proceedings of the Very Large Data Bases Endowment, 2014, 7(9): 769-780.
[15] PARENT C, SPACCAPIETRA S, RENSO C, et al. Semantic trajectories modeling and analysis[J]. ACM Computing Surveys, 2013, 45(4): 1-32.
[16] PALMA A T, BOGORNY V, KUIJPERS B, et al. A clustering-based approach for discovering interesting places in trajectories[C]//The ACM Symposium on Applied Computing. 2008: 863-868.
[17] VALDÉS F, DAMIANI M L, GÜTING R H. Symbolic trajectories in SECONDO: Pattern Matching and Rewriting[J]. Lecture Notes in Computer Science, 2013, 7826(2): 450-453.
[18] GUTTMAN A. R-trees: a dynamic index structure for spatial searching[C]//The ACM SIGMOD International Conference on Management of Data. 1984: 47-57.
[19] PRASAD V, ADAM C, EVERSPAUGH C, et al. Indexing large trajectory data sets with SETI[C]// Conference on Innovative Data Systems Research. 2003.
[20] PFOSER D, JENSEN C S, THEODORIDIS Y. Novel approaches in query processing for moving object trajectories[C]//International Conference on Very Large Data Bases. 2000: 395-406.
[21] CONG G, JENSEN C S, WU D. Efficient retrieval of the top-*k* most relevant spatial web objects[J]. Proceedings of the Very Large Data Bases Endowment, 2009, 2(1): 337-348.
[22] BERCHTOLD S, BÖHM C, KRIEGEL H P. Improving the query performance of high-dimensional index structures by bulk load operations[J]. Lecture Notes in Computer Science, 1998, 1377(2): 216-230.
[23] GUTING R H, ALMEIDA V, ANSORGE D, et al. SECONDO: an extensible DBMS platform for research prototyping and teaching[C]// International Conference on Data Engineering. 2005: 1115-1116.
[24] ZHENG Y, XIE X, MA W Y. GeoLife: a collaborative social networking service among user, location and trajectory[J]. Bulletin of the Technical Committee on Data Engineering, 2010, 33(2): 32-39.

[作者简介]

许建秋（1982-），男，江苏南京人，博士，南京航空航天大学副教授，主要研究方向为移动对象数据库。

梁珺秀（1994-），女，江西抚州人，南京航空航天大学硕士生，主要研究方向为移动对象数据库。



秦小麟（1953-），男，江苏南京人，博士，南京航空航天大学教授，主要研究方向为安全数据库、时空数据库、分布式环境数据管理与安全等。