基于 E2LSH 的轨迹 KNN 查询算法

邱 磊,吴志兵
(江南计算技术研究所,江苏 无锡 214083)

摘 要:目前海量时空轨迹数据近邻查询算法中存在计算时间复杂度较高的问题,因此提出了一种结合领域 POI 数据和 E2LSH 算法的轨迹 KNN 查询算法。首先利用 GeoHash 技术对地理空间进行编码,然后结合 POI 数据实现向量空间的初步降维,进而根据停留时间构建每条轨迹的向量,采用局部敏感哈希函数运算结果建立轨迹索引,最后对查询返回的相似轨迹集合分别进行距离计算,经过排序得到距离最近的 K 个查询结果。对于增量的轨迹数据,利用 E2LSH 算法计算哈希值,直接添加轨迹索引,从而避免了复杂的计算过程以及对现有轨迹索引的影响。基于合成数据及真实数据集的实验结果表明,该方法在海量时空轨迹数据的近邻查询中,虽然牺牲了一定的准确率,但有效提升了算法效率,并能够高效简便地处理增量的时空轨迹数据。

关键词:海量轨迹大数据;近邻查询;地理空间编码;局部敏感哈希;轨迹索引
中图分类号:TP391 **文献标识码**:A **文章编号**:1673-629X(2020)03-0013-06
doi:10.3969/j.issn.1673-629X.2020.03.003

E2LSH-Based Algorithm for Trajectory KNN Query

QIU Lei, WU Zhi-bing
(Jiangnan Institute of Computing Technology, Wuxi 214083, China)

Abstract: At present, there is a problem of high computational time complexity in the nearest neighbor query algorithm of massive spatio-temporal trajectory data, so a trajectory KNN query algorithm combining domain POI data and E2LSH algorithm is proposed. Firstly, GeoHash technology is used to encode the geographic space, and then the initial dimensionality reduction of the vector space is realized by combining POI data, and then the vector of each trajectory is constructed according to the residence time. The locality sensitive hashing function is used to establish the track index. Finally, distance is calculated for the similar track set returned by the query, and the nearest K query results are obtained by sorting. For incremental trajectory data, the hash value is calculated by E2LSH algorithm, and the trajectory index is added directly, thus avoiding the complicated calculation process and the impact on the existing trajectory index. The experiment on synthetic data and real dataset shows that the proposed method can effectively improve the algorithm efficiency and process the incremental spatiotemporal trajectory data efficiently and easily, although some accuracy is sacrificed in the neighbor query of massive spatiotemporal trajectory data.

Key words: massive trajectory data; KNN-query; geospatial space encode; locality sensitive hashing; trajectory indexing

0 引 言

随着传感器技术和定位技术的发展,基于位置的服务应用越来越广泛。人类活动通过智能手机 APP 应用软件进行位置分享、签到等;车辆安装 GPS 或北斗定位系统,周期性地上报位置信息;动物学家记录动物的迁徙路径等等。在这些活动中产生了一系列轨迹数据,随着时间的推移,数据规模愈加庞大。

对于海量轨迹数据的挖掘,可以获得大量有价值信息。例如,对车辆轨迹数据分析研究获取驾驶行为与交通路网之间的相关性[1];对智能手机定位轨迹数据分析得到用户的移动特点和生活规律[2];对旅行轨迹数据分析向用户推荐满足不同需求的路径[3]。轨迹数据挖掘常常依赖于大量的轨迹查询行为。轨迹查询分为 KNN(k-nearest neighbor)查询和范围查询,而 KNN 查询又分为轨迹点查询和整条轨迹的查询。查询的第一步是定义两条轨迹的距离,也就是轨迹之间

收稿日期:2019-03-15 **修回日期**:2019-07-15 **网络出版时间**:2019-11-07
基金项目:核高基项目基金(2015zx01040)
作者简介:邱 磊(1986-),男,硕士研究生,工程师,通信作者,研究方向为轨迹大数据挖掘与应用;吴志兵,硕士,高级工程师,研究方向为云计算与大数据系统架构、高性能计算网络。
网络出版地址:http://kns.cnki.net/kcms/detail/61.1450.TP.20191107.0912.050.html

的相似度[4]。

传统的海量轨迹数据查询基于树结构进行索引,当轨迹较短,即维度较少时性能良好,但是随着维度的增加(比如 20 维以上),检索效率会急剧下降,在性能上甚至还不如对整个数据集合进行顺序检索,这就是高维空间数据处理都会遇到的“维数灾难”[5]问题,从而导致难以实现轨迹的快速查询。局部敏感哈希技术[6](locality sensitive hashing,LSH)是解决高维数据近邻查询的有效方法。传统的 LSH 采用汉明距离来实现近邻数据的快速查询,所以在实际应用中需要首先将数据嵌入到汉明空间,大大增加了计算的复杂度。目前在大规模数据高维近邻索引中,一般采用 LSH 在欧氏空间中的一种随机化方法,被称为精确局部敏感哈希(exact Euclidean locality sensitive hashing,E2LSH)[7]。LSH 技术在很多领域获得了成功应用,例如大规模网页的去重、海量图片搜索以及相似音视频的检索等等。

综上,文中提出了一种支持海量轨迹数据的 KNN 查询算法(trajectory E2LSH-based KNN query,TRA-EKNN)。首先研究轨迹大数据的向量空间建模问题,并结合特定领域的兴趣点(point of interest,POI)进行了初步降维,进而采用 E2LSH 算法实现轨迹数据的索引构建,并在合成数据以及实际数据集上进行 KNN 查询,最后对实验结果进行了分析和总结,并展望了后续工作。

1 相关工作

对于轨迹数据的 KNN 查询算法,根据查询结果的可靠性,可以分为两类:基于时空数据库索引的穷举算法和基于哈希的近似算法。

目前,对于轨迹数据的查询研究多基于穷举算法。时空数据库[8](spatio-temporal databases,STDB)将时间和空间有机结合,实现对时空对象的有效管理。由于 R 树以及改进树[9]在空间索引的巨大优势,被大多时空索引结构所采用。近似算法在允许一定误差的前提下,可以大大提高检索性能。在海量数据日益增多的情况下,近似算法受到研究者的广泛关注。在近似算法中,应用最为广泛的是基于哈希的检索技术。Athitsos 等[10]定义并实现了一种全新的基于距离的哈希算法(distance-based hashing,DBH)。Sanchez 等[11]提出了一种基于 LSH 算法的轨迹聚类算法。这两种算法都将轨迹视为连续的轨迹点,即一条曲线,因此计算过程中均没有考虑时间因素,不适应于严格定义的轨迹数据。

国内学者对 LSH 技术在轨迹数据分析中的应用也做了大量相关研究。赵家石等[12]采用局部敏感哈希技术寻找在一定时间内都相似的移动对象,从而避免传统方法的交集运算过程,实现轨迹相似度的快速估计。廖律超等[13]利用谱哈希实现对交通路网的快速聚类,挖掘出交通轨迹大数据的潜在特性,为交通规划提供了科学依据。印桂生等[14]将文本相似领域比较广泛应用的相似哈希(simhash)[15]技术应用到用户轨迹的相似性比较中,取得了较好效果。

2 TRA-EKNN 算法

2.1 相关定义

文中所研究的问题是在海量的轨迹大数据中,根据一条轨迹数据查询找出距离最近,也就是最相似的 k 条轨迹。

定义 1:轨迹是由移动对象在增量时间的一系列轨迹点组成,记为:

$$TR = \{P_1, P_2, \cdots, P_n\}$$

其中 P_i 是移动对象在 t 时刻的位置(用经纬度表示),即 $p_i = (lat, lng, t)$。

定义 2:网格空间是指对参与计算的地理空间进行划分编码形成的网格集合,记为:

$$G = \{gid_1, gid_2, \cdots, gid_m\}$$

其中 gid_i 表示编号为 i 的网格。地理空间可以是一个国家、一个城市,也可以是结合特定领域知识和含义进行筛选之后区域范围。

定义 3:网格序列[16-17]是指一条轨迹将经过的网格顺序排列之后的集合,记为:

$$GS = \{g_1, g_2, \cdots, g_m\}$$

其中 $g_i = (gid, t)$,即在网格 gid 停留时间为 t。

2.2 算法思想

TRA-EKNN 算法主要包括网格索引建立、轨迹向量建模和 KNN 查询实现等过程,具体流程如图 1 所示。

2.2.1 网格建立

网格索引建立是针对特定领域的 POI 集合计算每个地理位置的网格编号,然后合并形成一个网格空间 G。

文中采用 GeoHash 技术实现网格的编码。GeoHash 通过使用一个字符串同时表示经度和纬度两个坐标,是一种通用的地址编码方案,具有如下特点:将二维空间的经纬度编码成一个字符串,在传统的数据库中做到一维的高效索引;每一个 GeoHash 表示的不是一个点,而是一个矩形的区域;GeoHash 编码的前缀可以用来表示更大范围的区域。

GeoHash 的计算方法主要分为三步:

(1)经纬度转换为二进制。首先将全部的纬度范围(-90,90)平均分为区间(-90,0)和(0,90),如果目

标维度位于第一个区间,则编码为 0,否则为 1,例如 39.923 24 属于(0,90),取编码为 1。然后再将(0,90)分为区间(0,45)和(45,90),而 39.923 24 位于(0,45),编码为 0。以此类推,直到精度符合要求为止,最终得到纬度的编码 1011 1000 1100 0111 1001。经度也采用同样算法得到编码。

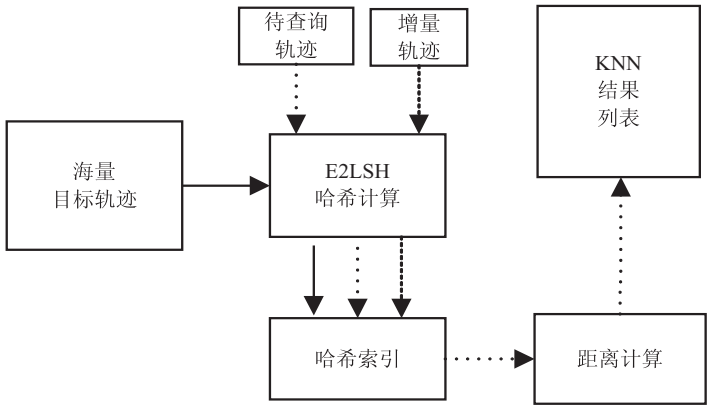

图 1　TRA-EKNN 算法框架

(2)按照经纬度的二进制合并。按照纬度在奇数位,经度在偶数位的规则进行合并。

(3)进行 Base32 编码。编码字符使用 0-9 和 b-z(其中去掉 a,i,l,o 四个字母)。

由以上的 GeoHash 计算过程,可以知道精度取决于编码长度,根据实际需求采用不同长度的编码,二者的对应关系如表 1 所示。

表 1　GeoHash 编码长度与精度

编码长度	Lat bits	Lng bits	精度/km
1	2	3	±2 500
2	5	5	±630
3	7	8	±78
4	10	10	±20
5	12	13	±2.4
6	15	15	±0.61
7	17	18	±0.076
8	20	20	±0.019

通过将地理位置编码,实现了对地理空间的网格索引。在此过程中,根据不同的应用场景采用不同的编码长度,实现效率与精度的统一。将所有特定领域的 POI 数据经过 GeoHash 编码,可以得到网格空间 G,相比于直接将所有的空间都进行编码,此方法利用了数据融合的思想,从而大大减少了网格数量,实现初步降维,在后续的计算过程中也更具实际意义。

2.2.2　轨迹向量建模

借鉴文本分类中向量空间模型(VSM)[18]的思想,将网格空间 G 视为 n 维向量的维度,本小节基于此,将轨迹向量化,然后再进行轨迹的距离计算。

轨迹网格序列化算法是向量建模的关键。将原始轨迹数据转换为网格序列,可以在保证一定精度的前提下,简化轨迹的处理过程。

算法 1:轨迹网格化。

输入:轨迹 TR = $\{P_1, P_2, \cdots, P_n\}$

网格空间 $G = \{\text{gid}_1, \text{gid}_2, \cdots, \text{gid}_m\}$

输出:网格序列 GS = $\{g_1, g_2, \cdots, g_m\}$

其中 $g_i = (\text{gid}, t)$

详细过程:

1. foreach P_i in TR
2. if($i == 1$)
3. last_time = P_i . t
4. last_geo = geohash(p_i . lat, p_i . lng)
5. continue
6. g. gid = geohash(P_i . lat, P_i . lng)
7. t_interval = P_i . t -last_time //时间差
8. if(g. gid in G)//只处理位于网格空间的数据
9. if(g. gid = = last_geo)
10. t_stay = t_stay+t_interval
11. else//进入另一网格
12. if(g. gid in GS)
13. Break// 截断处理
14. g. t = t_stay+t_interval/2
15. GS. append(g) //当前点加入序列
16. t_stay = t_interval/2
17. OUTPUT(GS)

算法 1 描述了轨迹转换为网格序列的全过程。最终的输出包含轨迹所经过的网格编码和相应的停留时间。由于采样时间的差异,导致在同一个区域可能出现多个轨迹点。停留时间的计算需要考虑两种情况:(1)两个轨迹点位于同一个网格时,间隔时间直接累加;(2)前后两个轨迹点位于不同网格,则停留时间按时间间隔的一半累加。从后续工作中可以看到,轨迹的向量化意味着轨迹网格不允许出现重复,所以算法 1 进行了简单处理即轨迹截断,另外一种方法是将后续轨迹点当作另一条轨迹继续处理。

经过算法 1 的处理,轨迹转换为在选定领域(网格空间)的网格序列。基于此,可以构建轨迹的向量矩阵[13],其中矩阵的行对应于全部的网格空间,列对应于每条轨迹。假设轨迹数据经过算法 1 的计算,提取出了 r 条轨迹,相应的网格空间大小为 n,则构成一个 $n \times r$ 的"网格-轨迹"矩阵 $\boldsymbol{T}$,每个元素的值定义为轨迹点在对应网格停留时间。

$$\boldsymbol{T} = \begin{bmatrix} & \mathrm{GS}_1 & \mathrm{GS}_2 & \cdots & \mathrm{GS}_r \\ \mathrm{gid}_1 & t_{11} & t_{12} & \cdots & t_{1r} \\ \mathrm{gid}_2 & t_{21} & t_{22} & \cdots & t_{2r} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \mathrm{gid}_n & t_{n1} & t_{n2} & \cdots & t_{nr} \end{bmatrix}$$

将轨迹进行以上处理之后,可以据此定义两条轨迹向量的距离:

$$d(i,j) = \sqrt{\sum_{k=1}^{N} (t_{ik} - t_{jk})^2} \tag{1}$$

轨迹的网格空间可以理解为针对特定领域所关心的所有位置,也就是在轨迹的距离(相似)公式中参与计算的位置,其他非必要的位置不参与计算。相比于通过聚类方法提取 POI(兴趣点)的方法,大大降低了数据预处理过程的难度和时间消耗。

2.2.3 E2LSH 算法索引构建

E2LSH 是 LSH 在欧氏空间的一种随机化实现方案。基本的思路是:采用基于 p 稳定分布的局部敏感函数对原始的高维数据进行降维映射。可以看出,E2LSH 继承了 LSH 的特点,即在原始空间相邻的数据,经过哈希之后,在新的空间上存在很大的概率也相近,具体哈希函数如下:

$$h_{a,b}(\vec{v}) = \left\lfloor \frac{\vec{a} \cdot \vec{v} + b}{w} \right\rfloor \tag{2}$$

其中,$\boldsymbol{a}$ 是独立随机选择的 d 维向量,b 为范围 $[0,w]$ 的一个随机数。函数 h 将一个 d 维向量映射到一个整数集,然后以 w 为宽度进行等距离划分。E2LSH 将 k 个哈希函数联合使用,称之为函数族:

$$g_i(\vec{v}) = (h_1(\vec{v}), h_2(\vec{v}), \cdots, h_k(\vec{v})), 1 \leqslant i \leqslant L \tag{3}$$

函数族 g 对应于 L 个哈希表,随机独立选取的 k 个 h 函数又构成了每个 g 函数,函数的值对应具体的哈希桶。

从算法思想的本质上看,LSH 是不确定的,概率性的,可以归结为近似算法。随着数据规模越来越大,数据处理的时效性要求面临很大挑战,这时通过牺牲一定精度以获得更快的处理速度,在一些领域是可行而有效的。本节将此算法应用于轨迹数据索引的构建,具体过程如下:

(1)对于数据库中的轨迹,经过算法 1 得到一条包含停留时间的网格序列;

(2)对于每条网格序列,由式(3)得到长度为 k 的哈希桶,计算 L 次,组成 L 个哈希表;

(3)对哈希表中的每个桶生成一个索引文件。

2.2.4 KNN 查询实现

KNN 查询的目标是在轨迹数据中找到与查询轨迹 TR 距离最近的 k 条轨迹。传统的线性算法通过计算待查询 TR 与所有的轨迹的距离,然后从中选出最近的 k 条轨迹,显然在这过程中计算量十分的巨大。虽然可以采用 R-Tree、KD-Tree 等树结构进行优化,但是"维数灾难"的问题,使得轨迹点数较多时,存在着大量的回溯计算,导致树结构不能发挥作用,这时退化成线性运算。

本算法在上一节构建好轨迹数据的 E2LSH 索引的基础上进行查询,具体流程如下:

(1)运用算法 1 得到所要查询的轨迹对应的网格序列;

(2)计算待查询网格序列的 L 个哈希表索引;

(3)遍历哈希表,找到对应的哈希桶,对桶内的轨迹,利用式(1)计算对应的距离,返回最近的 k 条轨迹。

本算法使用 E2LSH 技术,首先将所有的轨迹数据按照距离划分到不同的哈希桶,同一个哈希桶的轨迹距离较近,所以 KNN 查询只需要根据不同哈希值比较同一个或者相近桶内的轨迹,从而极大地减少了需要计算比较的轨迹数量,使得海量轨迹的 KNN 查询效率得到有效提升。

3 实验结果与分析

实验采用两个数据集:合成数据用来检测算法的时间效率,并与传统查询算法进行对比;上海出租车轨迹数据用来证明算法在实际应用中的有效性。实验环境为单机,处理器为 Intel(R) Core(TM) i3 M 380 @ 2.53 GHz,内存 6 G,操作系统为 Windows7 64 位 SP1,采用 python 语言编程实现。

3.1 合成数据实验

本实验的合成轨迹数据,根据网格数量和轨迹停留时间随机生成。假设网格数量为 m,轨迹数量为 n,根据 m 和 n 的不同取值,查询某条轨迹的最近 10 条轨迹,消耗时间如图 2 所示。

其中 10 000 * 10 000 表示 10 000 条轨迹对应于 10 000 个网格空间。可以看出时间消耗与二者的乘积大致是一个线性关系,随着网格空间的增大和轨迹数量的增多,计算所消耗的时间成线性增长。

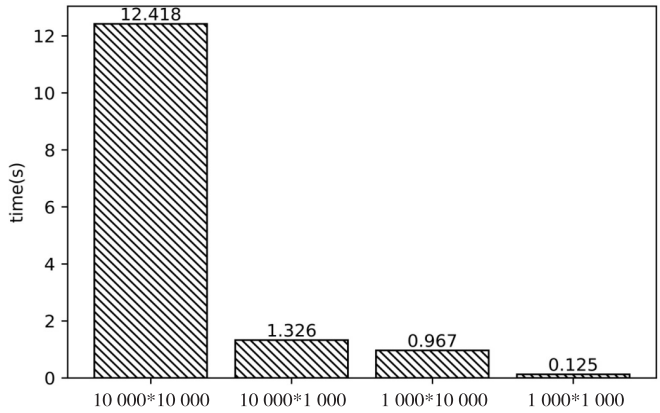

图 2　穷举消耗时间对比

采用 TRA-EKNN 算法对上述数据进行 KNN 查询,根据文献[6]中的建议值,算法中 w 取值为 4, k 取 80, L 取 10,此时获得的准确率大于 95%,时间消耗如图 3 所示。

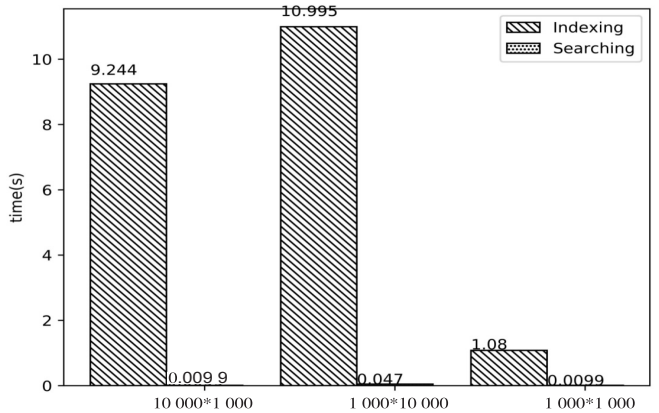

图 3　TRA-EKNN 消耗时间对比

从图 3 可知,TRA-EKNN 算法中,时间消耗主要分索引建立和轨迹查询两部分,而且索引的建立时间远大于查询时间,而且当网格空间固定即轨迹的维度相同时,轨迹的数量只影响索引的建立时间,而查询时间基本相同。

在实际的应用中,一般要先将海量的轨迹建立索引,图 1 所示的查询算法框架也体现了这一点。在后续的 KNN 查询时间一般指的就是图 3 中的查询时间,也就是说在进行轨迹查询之前索引已经建立完毕,查询过程中无需关心 E2LSH 索引的建立时间。对比图 2 可以看出,本算法的查询时间消耗明显小于穷举算法。

3.2　上海出租车轨迹数据实验

上海市出租车数据集,采集了 2007 年 2 月 20 日当天 4 316 台出租车的 GPS 记录信息,包括出租车标识、时间戳、经纬度、瞬时速度、顺时针角度以及载客情况等。本实验只取标识、时间戳和经纬度三个信息。

根据采集到的上海市交通相关的 POI 数据共计 242 556条,考虑到实验数据是出租车 GPS 轨迹结合表 1 的 GeoHash 长度与精度关系,哈希长度设定为 7,误差约为 76 米,将 POI 数据中的经纬度计算 GeoHash,去重后,共计有 71 495 个网格。上海面积约为 6 340 平方公里,如果全部进行 7 位的 GeoHash 编码,大约一百万个网格(由于行政区划的不规则,数据不准确,在此仅进行估算)。可以看出由于结合了交通领域的 POI 数据,使得网格空间大幅压缩。

对出租车轨迹采用算法 1 进行处理,共提取有效轨迹对应的网格序列 1 896 条,由于出租车行驶过程中会出现司机休息和等待乘客等情况,停留时间会比较长,为了体现轨迹比较的实际意义,将停留时间大于二十分钟的轨迹截断,然后将其余轨迹的停留时间作归一化处理。

采用 TRA-EKNN 算法,对所有的轨迹数据建立 E2LSH 索引,经过多次试验,此算法可以在较短的时间内查询出 KNN 结果,并有较高的准确率。

3.3　算法分析

正如前文所述,时空轨迹数据的 KNN 算法主要是基于传统的穷举算法,即采用一一匹配的暴力搜索的方式进行,此时时间复杂度是线性的,即 O(N)。随着数据规模越来越大,这种方式会消耗大量的时间。

文中提出的 TRA-EKNN 算法,首先建立了 LSH 索引,虽然带来了空间消耗增加的问题,但检索的时间

复杂度降为了O(logN),这对于一些时效性要求较高的场景是完全适用的。

4 结束语

将在海量文本、音频、视频数据处理广泛应用的E2LSH算法引入到轨迹数据的KNN查询中。首先针对不同的应用场景,结合特定领域POI数据建立网格空间;然后对轨迹数据进行预处理,构造出轨迹的网格向量,利用E2LSH算法实现索引。从图1算法的实现框架来看,对于增量的轨迹只需要计算哈希值,然后插入索引即可,计算简单。实验结果表明,在保证一定的准确率的情况下,该算法有效降低了轨迹KNN查询时间。下一步将继续对轨迹的特征提取方法进行探索,找出更适合轨迹数据处理的向量空间构建方案,使此方法能够在海量的轨迹数据挖掘中得到更广泛的应用。

参考文献:

[1] FANG Z,SHAW S L,TU W,et al. Spatiotemporal analysis of critical transportation links based on time geographic concepts:a case study of critical bridges in Wuhan,China[J]. Journal of Transport Geography,2012,23:44-59.

[2] 马宇驰,杨 宁,谢 琳,等. 基于轨迹时空关联语义和时态熵的移动对象社会角色发现[J]. 计算机研究与发展,2012,49(10):2153-2160.

[3] ZHENG Y,ZHANG L,XIE X,et al. Mining interesting locations and travel sequences from GPS trajectories[C]//Proceedings of the 18th international conference on world wide web. Madrid,Spain:ACM,2009:791-800.

[4] ZHENG Y. Trajectory data mining:an overview[J]. ACM Transactions on Intelligent Systems and Technology,2015,6(3):1-41.

[5] BELLMAN R E. Dynamic programming[M]. Princeton:Princeton University Press,1957.

[6] DATAR M,IMMORLICA N,INDYK P,et al. Locality-sensitive hashing scheme based on p-stable distributions[C]//Proceedings of the twentieth annual symposium on Computational geometry. [s. l.]:ACM,2004:253-262.

[7] ANDONI A,INDYK P. E2LSH:exact Euclidean locality-sensitive hashing[DB/OL]. 2016. http://web. mit. edu/andoni/www/LSH/index. html.

[8] SHEKHAR S,CHAWLA S. 空间数据库[M]. 谢昆青,译. 北京:机械工业出版社,2004:118-123.

[9] NASCIMENTO M A,SILVA J R O. Towards historical R-trees[M]//Proceedings of the ACM symposium on applied computing. Atlanta,Georgia,USA:ACM,1998:235-240.

[10] ATHITSOS V,POTAMIAS M,PAPAPETROU P,et al. Nearest neighbor retrieval using distance-based hashing[C]//2008 IEEE 24th international conference on data engineering. Cancun:IEEE,2008:327-336.

[11] SANCHEZ I,AYE Z M M,RUBINSTEIN B I P,et al. Fast trajectory clustering using hashing methods[C]//2016 international joint conference on neural networks (IJCNN). Vancouver,BC:IEEE,2016:3689-3696.

[12] 赵家石,杨 静,张健沛. 一种隐私保护的在线相似轨迹挖掘方法[J]. 哈尔滨工业大学学报,2013,45(11):101-105.

[13] 廖律超,蒋新华,邹复民,等. 一种支持轨迹大数据潜在语义相关性挖掘的谱聚类方法[J]. 电子学报,2015,43(5):956-964.

[14] 印桂生,程伟杰,董宇欣,等. 使用轨迹指纹和地点相似性的地点推荐[J]. 哈尔滨工程大学学报,2016,37(3):414-419.

[15] SADOWSKI C,LEVIN G. Simhash:hash-based similarity detection[R]. [s. l.]:[s. n.],2007.

[16] 杨 阳,吉根林,鲍培明. 基于网格索引的时空轨迹伴随模式挖掘算法[J]. 计算机科学,2016,43(1):107-110.

[17] 司 阳,肖秦琨. 基于长短时记忆和动态贝叶斯网络的序列预测[J]. 计算机技术与发展,2018, 28(9):59-63

[18] SALTON G,WONG A,YANG C S. A vector space model for automatic indexing[J]. Communications of the ACM,1975,18(11):613-620.