移动环境中的位置依赖连续轮廓查询

黄伯虎[1,2]，张海宾[1]，王小兵[1]，刘旭东[3]
(1. 西安电子科技大学计算理论与技术研究所，710071，西安；2. 西安电子科技大学综合业务网理论及关键技术国家重点实验室，710071，西安；3. 中国人民解放军69023部队，830063，乌鲁木齐)

摘要：针对移动环境中查询点快速移动时连续、高效输出给定搜索区域数据轮廓的问题，提出一种位置依赖连续轮廓查询算法(LDCS)．该算法结合数据流技术，首先使用R树快速更新查询数据，然后利用两次连续计算时搜索区域的重叠性构造被动数据流，并对新增和失效数据分别进行处理，从而连续输出轮廓．由于充分利用了已有结果，LDCS的计算量较传统算法有大幅下降．实验结果表明，LDCS特别适合计算频度要求较高的场合，与基于网格索引的算法相比，时间效率随着数据集规模的增大显著提升．

关键词：数据流；位置服务；轮廓；查询处理；移动计算
中图分类号：TP301 **文献标志码**：A **文章编号**：0253-987X(2012)06-0079-08

Location-Dependent Continuous Skyline Query in Mobile Environment

HUANG Bohu[1,2], ZHANG Haibin[1], WANG Xiaobing[1], LIU Xudong[3]
(1. Institute of Computing Theory & Technology, Xidian University, Xi'an 710071, China; 2. State Key Laboratory of Integrated Service Networks, Xidian University, Xi'an 710071, China; 3. Unit 69023 of PLA, Urumqi 830063, China)

Abstract: For the issue of continuous skyline query in mobile environment where the query point is moving fast toward unpredictable directions, a novel algorithm named LDCS (location dependent continuous skyline query) is proposed. Along with data stream introduction, R-tree is used to quickly update the data set in search area. Then a "passive" data stream is constructed by overlap of two adjacent search areas and the "new" and "fail" data sets are dealt with separately. Compared with traditional algorithms, LDCS is set lighter calculation tasks due to its usage of historical results. The experimental results show that LDCS is particularly suitable for the high frequency calculation, and with the increasing size of data set, it gets significant improvement in efficiency over grid-based algorithms.

Keywords: data streams; location-based service; skyline; query processing; mobile computing

轮廓查询也称Skyline计算，最早由Borzsonyi等人在2001年作为一种新型数据库操作算子提出[1]，目的旨在发现数据集中所有用户可能感兴趣的记录，从而为决策提供依据．作为一种基础的数据操作方法，轮廓查询在许多领域有着广泛的应用，如多目标决策、导航系统、数据挖掘和用户偏好查询等．

当前，基于传统静态数据集和固定查询点的轮廓查询算法已经研究得比较充分，在一些重要国际会议和期刊上出现了大批研究成果[2-5]．近年来，随着技术的快速发展，新的应用需求不断涌现，数据集呈现出动态、连续的特点，于是新环境下高效轮廓查询算法的研究成为一个热点．

位置服务(LBS)[6]是指通过移动终端和无线或

收稿日期：2011-10-26． 作者简介：黄伯虎(1979—)，男，讲师． 基金项目：国家"973计划"资助项目(2010CB328102)；国家自然科学基金资助项目(61003079)；教育部高等学校博士学科点专项科研基金资助项目(20100203120012)；中央高校基本科研业务费专项资金资助项目(K5051203003)．
网络出版时间：2012-02-25 网络出版地址：http://www.cnki.net/kcms/detail/61.1069.T.20120225.0850.005.html

卫星通信网络的配合，确定出移动用户的实际地理位置，并为用户提供与位置相关信息的服务模式. 在民用和军事领域，位置服务均有着重要的应用，如GIS(Geographic Information System)、数字战场环境下移动目标的监控和管理等.

轮廓查询和位置服务相结合，可以为用户在移动环境下提供决策支持. 图1是一类典型的应用场景，旅行者开车经过一个陌生城市，当前位置在 q，沿着图中虚箭头线行进，此刻他希望寻找一个合适的酒店休息. 为了获取帮助，打开导航设备，输入要求：离当前位置距离不超过2 km；酒店星级尽可能高；价格尽可能低. 希望导航设备能够提供符合要求的酒店供自己选择. 显然，这是一个轮廓查询问题. 不过，相对于传统的计算环境，有其新特点：①查询点 q 不是固定的，而是在不断的移动，图1中 q 沿着虚箭头线运动；②用户关注的不是整个区域(包含所有酒店)，而是被约束在一定范围(半径 r)内的子区域 R，并且随着 q 的移动，R 在不断变化；③查询并非只进行一次，而是要随着 q 的不断移动和 R 的不断变化连续输出结果，计算的频度可以以时间或距离为单位，例如每5 s或每移动50 m进行一次.

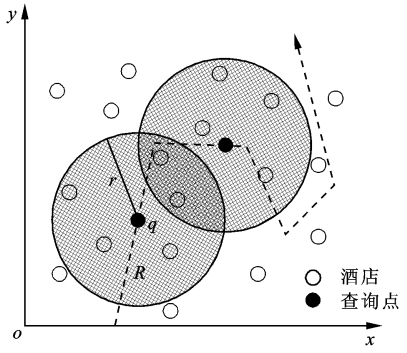


图1 应用场景示例

在图1的场景中，虽然整个数据集是静态的，但子区域 R 中的数据集在不断的变化. 在某种程度上，这种不断变化的趋势可以看作是一种被动的数据流(由 q 的移动引起). 然而，与常规的数据流[7-8]相比，其差异在于：由于 q 移动方向的不确定性，R 中数据对象的更新并不遵守先进先出(FIFO)原则，也就是说当前在数据集中保持时间最长的对象不一定在下次更新时会被淘汰.

目前，数据流上的动态轮廓查询问题已有不少研究成果[9-13]，但这些工作均考虑常规数据流，与本文定义的数据流不同. 基于位置服务的轮廓查询也有许多研究进展[14-17]，不过由于应用场合的多变性，面对类似图1的场景，现有算法尚不能完全适用.

本文借鉴已有成果，针对移动环境中查询点快速移动时连续、高效输出给定搜索区域数据轮廓的问题，提出了一种位置依赖的连续轮廓查询算法(LDCS). 算法分3步：更新查询数据，构造数据流，计算轮廓集. 首先使用R树快速更新查询数据，然后利用两次连续计算时搜索区域的重叠性构造被动数据流，最后对新增和失效数据分别进行处理得到轮廓集. 由于充分利用了已有结果，LDCS算法的计算量较传统算法有很大的下降. 实验结果表明，LDCS算法拥有较好的时间效率，特别适合计算频度要求较高的场合，可以近似实时形式输出结果.

1 问题及概念

1.1 问题的形式化描述

为方便理解，结合图1给出本文问题的形式化定义. 设数据集 $D=\{h_1,h_2,\cdots,h_n\}$，$|D|=n$(图1中 D 相当于所有酒店组成的集合). 元素 $h_i(i=1,2,\cdots,n)$ 是位于2维欧式空间中的数据点(相当于酒店)，x、y 分别为其2维的坐标值. h_i 的属性定义在一个 d 维空间中，$h_i[k](k\in[1,d])$ 表示 h_i 第 k 维的属性值. q 为查询点(旅行者位置)，R 为以 q 为圆心、半径为 r 的搜索区域，D_R 为 R 内的数据集，显然 $D_R\subseteq D$. 符号“$\trianglelefteq$”的含义为“不差于”，“$\triangleleft$”的含义为“好于”.

定义1 支配. 设 $h_i,h_j\in D$，当且仅当 $\forall k\in[1,d]$，$h_i[k]\trianglelefteq h_j[k]\wedge\exists t\in[1,d],h_i[t]\triangleleft h_j[t]$ 时，称 h_i 在 D 中支配 h_j，记作 $h_i\prec_D h_j$.

当上述条件不满足时，称 h_i 在 D 中不支配 h_j，记做 $h_i\not\prec_D h_j$.

若 $h_i\not\prec_D h_j\wedge h_j\not\prec_D h_i$，则称 h_i 和 h_j 在 D 中互不支配，记作 $h_i<>_D h_j$.

定义2 仅被支配. 设 $h_i,h_j\in D$，h_j 在 D 中仅被 h_i 支配，记作 $h_i\not\prec_D h_j$，当且仅当 $h_i\prec_D h_j\wedge\forall h_c\in D(c\neq i),h_c\not\prec_D h_j$.

定义3 轮廓. D 的轮廓，记作 $S_L(D)$，由 D 中不被支配的元素构成，即 $S_L(D)=\{h_i\in D|h_j\not\prec_D h_i if h_j\in D\}$.

与此相对，D 中非轮廓点集定义为 $\overline{S_L}(D)=D-S_L(D)$.

本文的目标是：当 q 不断移动、R 不断变化时，高效、连续地输出 $S_L(D_R)$.

1.2 相关工作

轮廓计算理论研究工作始于20世纪70年

代[18],2001 年引入数据库领域[1]. 早期的研究主要集中在静态数据集上,出现了一批经典算法[1-3,5]. 虽然对于高维、海量、动态数据集,这些算法已经不能很好地适应,但在局部小规模情况下,它们依然能发挥简洁高效的特点,本文部分环节便使用了这些算法.

当前,轮廓计算正逐步趋向于在具体环境中应用,如 Web 信息系统、分布式系统、数据流和移动公路网络等[19]. 本文研究工作主要基于数据流技术和移动环境,这两方面已有许多研究成果.

Lin 等[9]关注 n-of-N 轮廓查询问题,定义了关键支配关系,使系统需要保存的数据量达到最小,并采用新颖的编码方法将轮廓计算映射为刺探查询问题,实验证明该方法能够处理速度较快的数据流. Tao 等[10]提出了基于滑动窗口模型的连续轮廓计算方法,给出了 Lazy 和 Eager 两个计算框架,通过理论分析和实验结果证明了它们能够满足快速数据流上的连续轮廓查询需求. Lu 等[11]研究了分布式数据流上的连续轮廓计算问题. 张丽等[12]提出了基于网格索引的数据流连续轮廓处理方法. Su 等[13]探索了不确定性数据流上的持续概率轮廓查询问题.

在移动环境下,Huang 等[14]假设包括查询点在内的所有数据点以平滑可预测的方式移动,提出了连续跟踪算法(CSQ),设计了 kinetic-based 数据结构,深入分析了点的空间位置与支配关系之间的联系,限定了导致轮廓集合发生变化的点的存在区域,并给出了一种高效动态数据集上的连续轮廓查询方法. Lee 等[15]研究了位置和速度等多维动态属性情况下的连续轮廓计算,通过对候选对象集空间支配关系的分析,确定任意时刻的轮廓集. Zheng 等[16]提出位置依赖轮廓查询问题,并利用 δ-scanning 算法查找固定空间对象集合中的位置依赖轮廓集. Lin 等[17]关注用户位置不精确情况下的轮廓查询问题.

然而,由于应用环境的多变性,现有算法依然不能满足所有需求,仍需不断地探索和改进.

2　LDCS

q 移动时,$S_L(D_R)$ 的计算可分为 3 步:①更新 D_R;②计算 $S_L(D_R)$;③输出结果. 这一过程不断重复,从而得到连续的轮廓集. LDCS 主要以提升计算效率为目标,下面具体讲述各关键步骤的实现.

2.1　D_R 的确定

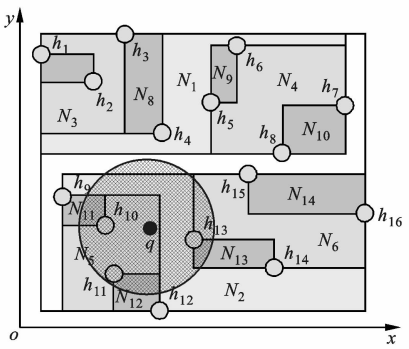
最为简洁地确定 D_R 的算法如图 2 所示(这里称为 NIndex_R 算法):计算 D 中每一个数据点到 q 的距离,然后与 r 比较,如果小于等于 r,则该点属于 D_R.

```
算法:NIndex_R
功能:更新D_R
输入:查询点q,搜索半径r,数据集D
输出:D_R
1    D_R={};
2    for (D中的每一个数据点e)
3    {
4        g←计算e到q的距离;
5        if (g≤r)
6            将e插入D_R;
7    }
8    返回结果D_R;
```

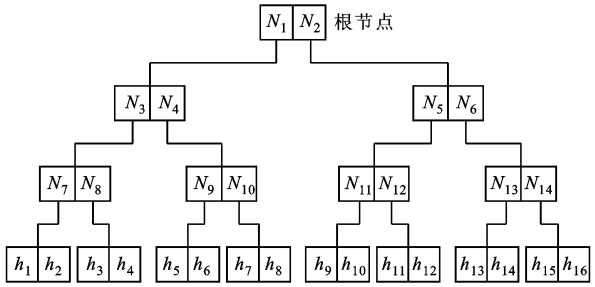
图 2　NIndex_R 算法

不过由于 D 的规模通常较大,且 D_R 更新较为频繁,这种遍历计算的方法非常低效. 为此,引入索引结构以降低计算量,给出一种基于 R 树[20]的算法,步骤如下.

(1)对 D 构造 R 树. 构造方法见文献[21],图 3 是一个示例,R 树中每个节点对应着一个最小边界矩形(MBR),中间节点的项(entry)指向下层的 MBR,叶子节点存放数据点. 由于 D 中数据变化性相对较小,R 树建成后不需要频繁的动态更新,可保证较高的计算效率.



(a)MBR



(b)R 树

图 3　R 树结构

(2)计算 D_R. 算法伪代码描述见图 4. 首先将 R 树根节点 root 中的所有项加入空队列 Q,然后取出 Q 中的第一个元素(e),判断其对应的 MBR 是否与 R 相交,如果相交且 e 为中间节点,则将其子节点中的所有项加入 Q,否则将 e 加入 D_R,循环往复,直到 Q 为空. 计算过程有点类似于树的广度优先搜索,只是当发现某个 MBR 与 R 不相交时,便不再访问其子树. 当 R 范围较小时,这一剪枝操作的效果非常明显,可大幅提高计算效率. 表 1 以图 3 为例显示了计算过程,左列表示执行的动作,中间为 Q 中的元素,右列显示 D_R 的变化情况.

```
算法: Updata_R
功能: 更新D_R
输入: R树根节点root, 查询点q, 搜索半径r
输出: D_R
1    D_R={};
2    将root中所有项插入队列Q;
3    while (Q不空){
4        e←队列中Q第一个元素;
5        g←计算q到e的距离;
6        if (g≤r)
7            if (e是中间节点)
8                将e孩子节点中的项插入队列Q;
9            else
10               将e插入D_R;
11   }
12   返回结果D_R;
```

图 4 D_R 更新算法

表 1 D_R 计算过程示例

动作	队列 Q 中元素	D_R
访问 root	~~N_1~~, N_2	{ }
扩展 N_2	N_5, N_6	{ }
扩展 N_5	N_6, N_{11}, N_{12}	{ }
扩展 N_6	N_{11}, N_{12}, N_{13}, N_{14}	{ }
扩展 N_{11}	N_{12}, N_{13}, N_{14}, h_9, h_{10}	{ }
扩展 N_{12}	N_{13}, N_{14}, h_9, h_{10}, h_{11}, h_{12}	{ }
扩展 N_{13}	~~N_{14}~~, ~~h_9~~, h_{10}, h_{11}, h_{12}, h_{13}, h_{14}	{ }
插入 h_{10}	h_{11}, h_{12}, h_{13}, h_{14}	$\{h_{10}\}$
插入 h_{11}	~~h_{12}~~, h_{13}, h_{14}	$\{h_{10}, h_{11}\}$
插入 h_{13}	~~h_{14}~~	$\{h_{10}, h_{11}, h_{13}\}$

2.2 数据流的形成

D_R 更新后,可以将其看做一个静态数据集并使用块嵌套环算法(BNL)、分治(D&C)等常规算法计算轮廓,这样做简单,但效率较差,尤其不适合 D_R 更新较快且需要以近似实时形式输出结果的场合. 事实上,R 在变化前后,有很多区域是重叠的,见图 5. 设 t 时刻 R 对应的数据集为 D_R^t,下一计算时刻 $t+1$ 对应的数据集为 D_R^{t+1}. 显然,在计算 $S_L(D_R^{t+1})$ 时,如果能够利用已求得的 $S_L(D_R^t)$,则可提高计算效率. 为此构造如下数据流:令 $S^+=D_R^{t+1}-D_R^t$,$S^-=D_R^t-D_R^{t+1}$,则有 $D_R^{t+1}=D_R^t-S^-+S^+$,即 D_R^{t+1} 可看做是在 D_R^t 的基础上通过增加 S^+ 和删除 S^- 得到,这样在 q 的不断移动过程中,便形成了一个数据流.

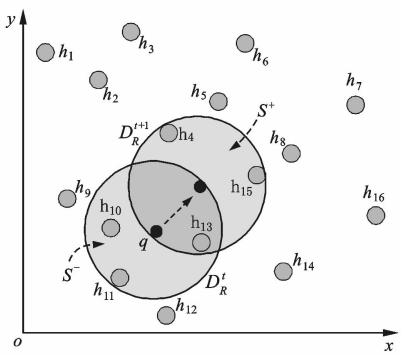


图 5 数据流的形成

2.3 轮廓计算

2.3.1 计算框架 在 t 时刻,经过计算 D_R^t 被分为两部分:$S_L(D_R^t)$ 和 $\overline{S_L}(D_R^t)$,如图 6 所示. 在 $t+1$ 时刻,S^+ 和 S^- 被分别存放在新增数据缓存和失效数据缓存中. 计算时,首先由失效数据处理模块对 S^- 进行处理,然后由新增数据处理模块处理 S^+,最后输出结果 $S_L(D_R^{t+1})$.

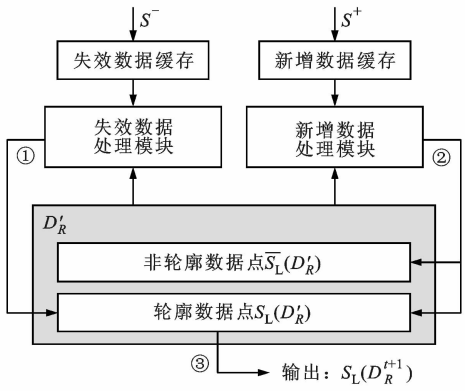


图 6 轮廓计算框架

2.3.2 失效数据处理 S^- 中的任何一个数据点 e,存在 $e\in\overline{S_L}(D_R^t)$ 或 $e\in S_L(D_R^t)$ 两种可能. 若 $e\in\overline{S_L}(D_R^t)$,则 e 失效后不会对轮廓产生任何影响,但是若 $e\in S_L(D_R^t)$,则 e 失效后必然会引起轮廓发生变化. 基于此,我们给出图 7 所示失效数据处理算

法,其过程为:依次扫描 S^- 中的各数据点 e,如果 $e \in \overline{S_L}(D_R^t)$,直接将其删除;否则,从 $S_L(D_R^t)$ 中删除 e,同时计算 D_R^t 中所有仅被 e 支配的数据点,并将其插入 $S_L(D_R^t)$.

```
算法:失效数据处理
输入:S⁻, D_R^t
输出:更新后的D_R^t
1    for (S⁻ 中的数据点e)
2        if  e∈S̄_L(D_R^t)
3            从S̄_L(D_R^t)中删除e;
4        else {
5            从S_L(D_R^t)中删除e;
6            U←计算仅被e支配的点;
7            合并U到S_L(D_R^t);
8        }
```

图 7　失效数据处理算法

在图 7 算法中,最为耗时的操作为第 6 行(仅被支配数据点)的计算,因此如何快速计算出 D_R^t 中仅被 e 支配的数据点集 U 成为决定算法效率的关键.下面给出解决方案,不失一般性,以 2 维空间数据集为例.

定义 4　支配区域.设 $e \in D$,e 在 D 中的支配区域记作 $V_D(e)$,指 D 中被 e 所支配的空间.

定义 5　反支配区域.设 $e \in D$,e 在 D 中的反支配区域记作 $V_D^A(e)$,指 D 中能支配 e 的空间.

令 $W = D_R^t$,图 8a 给出了数据点 e 的支配区域和反支配区域.图 8b 中,$a,e,b \in S_L(W)$,且 a,e,b 3 点在轮廓线上相邻,则所有只被 e 支配的点只可能存在于区域 $G = V_W(e) - V_W(a) - V_W(b)$ 中,因为 $V_W(e)$ 中其他区域的点,如 f、g、h,都至少被两个或两个以上轮廓点支配.计算 U 时,只需首先获得 G 中的数据集 D_G,然后采用常规算法(BNL、D&C 等)计算 $S_L(D_G)$,则 $U = S_L(D_G)$.

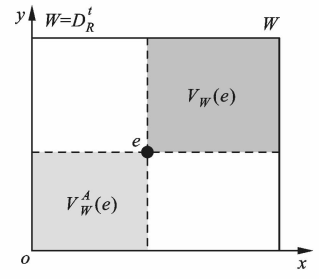
2.3.3　*新增数据处理*　新增数据处理模块主要负责对 S^+ 进行处理,分两个步骤.

(1)预处理.轮廓计算过程中数据点之间的比较是最为基本的操作,因此减少比较次数是提高效率的有效手段之一.

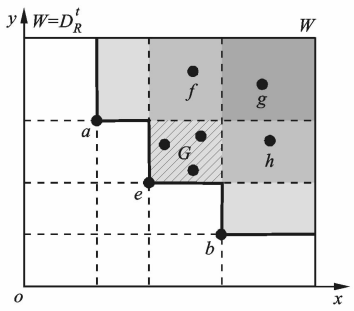
定理 1　设 $S^+ \subseteq D$,$e \in S^+$,若 $e \in \overline{S_L}(S^+)$,则 $e \notin S_L(D)$.

证明　$e \in \overline{S_L}(S^+) \Rightarrow \exists c \in S^+, c \prec_{S^+} e$.又 $S^+ \subseteq D$,则 $\exists c \in D, c \prec_D e$,故 $e \notin S_L(D)$.

基于定理 1,预处理过程为:首先通过传统算法



(a)支配区域和反支配区域



(b)仅被支配区域

图 8　仅被支配区域计算

将 S^+ 分为两个子集 $S_L(S^+)$ 和 $\overline{S_L}(S^+)$,然后将 $\overline{S_L}(S^+)$ 直接并入 $\overline{S_L}(D_R^t)$.经过这样的处理后,S^+ 中仅剩下 $S_L(S^+)$,从而减少了后续处理中需要比较的数据点.

(2)轮廓计算.令 $W = D_R^t$,对 S^+ 中每一个数据点 e,首先计算 $V_{S_L(W)}^A(e)$,如果 $V_{S_L(W)}^A(e)$ 不为空,说明 e 被支配,直接插入 $\overline{S_L}(W)$;否则,说明 e 不被 $S_L(W)$ 中任何数据点支配,e 为轮廓点,插入 $S_L(W)$.同时,新轮廓点的加入必然会改变原来的轮廓结构,因此还需进行调整:计算 $V_{S_L(W)}(e)$,如果 $V_{S_L(W)}(e)$ 不为空,说明有其他轮廓点被 e 支配,则将 $V_{S_L(W)}(e)$ 中的数据点从 $S_L(W)$ 转移到 $\overline{S_L}(W)$.最后,输出 $S_L(W)$,即为 $S_L(D_R^{t+1})$.算法伪代码描述如图 9 所示.

3　实验及性能分析

实验环境为:Intel 酷睿 2 四核 Q950 处理器,4GB 内存,Windows XP SP3 操作系统,所有代码在 VC++6.0 下编译运行.实验参数:数据集规模 $n = (2\times10^5, 4\times10^5, 6\times10^5, 8\times10^5, 10\times10^5, 12\times10^5)$;数据点空间位置服从均匀分布,坐标区域 10×10;数据点非空间属性维数为 5,独立均匀分布,范围为[0,1],取单精度浮点数(4 B),值越小越优.

```
算法：新增数据处理
输入：S+, D_R^t
输出：S_L(D_R^t)
1    S+预处理；
2    W=D_R^t；
3    for (S+中剩余数据点e) {
4        计算e在S_L(W)中的反支配区域V^A_{S_L(W)}(e)；
5        if (V^A_{S_L(W)}(e) 不空)
6            插入e到S_L(W)；
7        else{
8            插入e到S_L(W)；
9            计算e在S_L(W)中的反支配区域V_{S_L(W)}(e)；
10           if (V_{S_L(W)}(e)不空)
11           for (V_{S_L(W)}(e)中数据点c)
12               移动c到S_L(W)；
13       }
14   }
15   返回结果S_L(W)；
```

图 9 新增数据处理算法

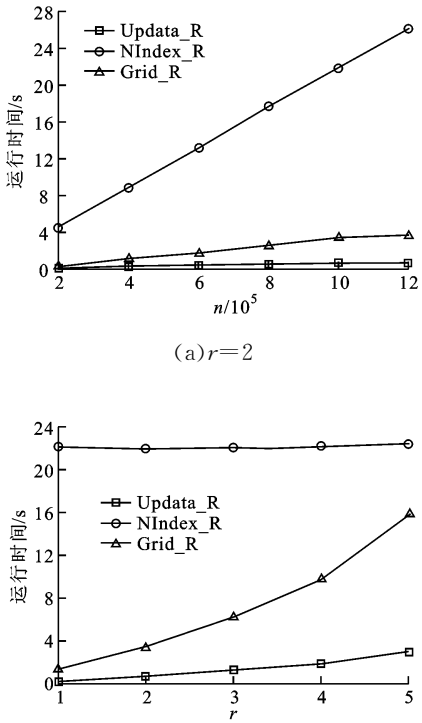
由于尚未发现可完全匹配本文所述环境的其他算法，采用分段测试的方法进行比较. 每组实验进行10次独立测试，结果取均值.

3.1 D_R 更新速度测试

实验测试 D_R 在不同情况下的更新速度. 将Update_R算法(图4)、NIndex_R(图2)及使用网格索引[22]的算法(这里称为Grid_R算法)进行比较，分两种场景测试：①搜索半径确定($r=2$)，数据集规模变化($n=(2\times10^5,\ 4\times10^5,\ 6\times10^5,\ 8\times10^5,\ 10\times10^5,\ 12\times10^5)$)；②数据集规模确定($n=10\times10^5$)，搜索半径变化($r$ 为1，2，3，4，5). 图10为测试数据，从中可以看出，无论在那种场景下，Update_R算法和Grid_R算法的效率都明显优于NIndex_R算法(相差两个数量级)，在本文环境下，使用R树索引比使用网格索引效果好. 图10a中，当数据规模 n 增大时，R树中节点数会相应增加，但因树的深度几乎没有变化，故Update_R算法的计算时间增加趋势并不明显. 图10b中，随着 r 的增大，R树剪枝效果变差，访问的数据点数量增多，因此Update_R算法计算时间随之增大. 设R树节点最大容量为 M，最小容量为 $m\leqslant M/2$，一个规模为 n 的数据集，均匀分布在 $x\times y$ 的矩形区域中，其R树的最大深度为 $\lfloor \log_m n \rfloor -1$，数据点落在半径为 r 的区域中的概率为 $\pi r^2/(xy)$，故半径为 r 的区域中数据点的数量为 $n[\pi r^2/(xy)]$. 因此，在最坏的情况下，Update_R算法的效率为 $O(n\log_m n)$.

3.2 区域重叠度与算法执行效率测试

2.2节中曾提到，t 时刻搜索区域可能会和 $t+1$ 时刻搜索区域存在重叠(图5中深色阴影区域)，我们定义区域重叠度 σ 为重叠区域面积与搜索区域面积之比，这里重叠区域的面积与 t 至 $t+1$ 时刻 q 点的移动距离(取决于 q 点的移动速度和计算间隔时间)相关. 显然，σ 越大，D_R^t 与 D_R^{t+1} 中重叠的数据越多，计算时更新的数据越少；反之，σ 越小，D_R^t 与 D_R^{t+1} 中重叠的数据越少，计算时更新的数据越多. 本实验主要测试 σ 对于算法效率的影响，可反映出LDCS是否适合高频度的持续计算. 对比算法为未考虑重叠情况的传统算法(这里称为T_Sky算法)，其特点为：采用BNL算法搜索区域轮廓，每次独立计算. 测试场景：搜索半径 $r=2$，区域重叠度变化(σ 为0%，20%，40%，60%，80%，100%)，数据集规模变化($n=(2\times10^5,4\times10^5,\ 6\times10^5,\ 8\times10^5,\ 10\times10^5,\ 12\times10^5)$).



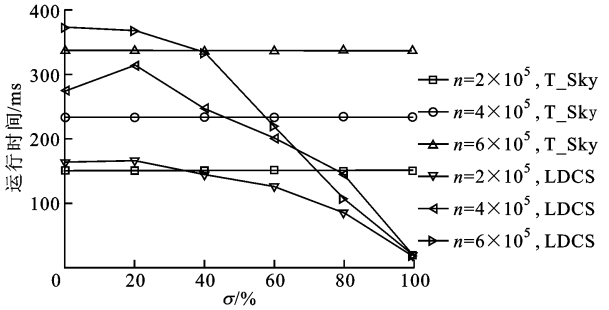
(a) $r=2$

(b) $n=10\times10^5$

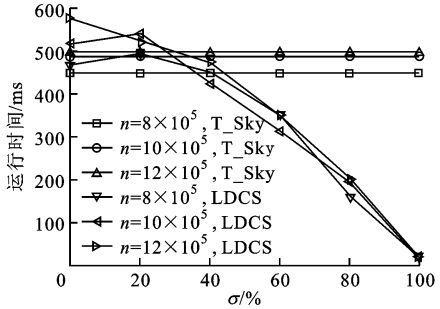
图10 D_R 更新速度测试

测试数据见图11，可以看出，无论哪种规模，随着重叠度 σ 的增大，LDCS算法效率逐步提升，计算时间差异减小. 这是因为LDCS算法以对更新数据流(S^+，S^-)的处理为基础，当 σ 增大时，S^+、S^- 中数据量在变小，$\sigma=100\%$，意味着 $|S^+|=|S^-|=0$，即无数据更新，计算量自然很小. 不过在 σ 较小(如0%，20%)时，LDCS性能要比T_Sky算法差，因为此时两种算法处理的数据量差别不大，但LDCS还

有一些额外的开销(如计算 S^+、S^- 等)，从而增加了计算时间. 由此可以看出，LDCS 在重叠度较大时可以体现良好的性能，因此特别适合于计算频度要求很高的场合，其时间效率随着数据集规模的增大显著提升.



(a)$n=2\times10^5$，4×10^5，6×10^5



(b)$n=8\times10^5$，10×10^5，12×10^5

图 11　区域重叠度与算法效率

4　结　论

本文研究了在移动环境中，当查询点快速移动时，如何连续、高效地输出关注区域数据轮廓的问题，给出了一种有效的算法 LDCS. 算法的最大特点是引入数据流查询技术，通过构造被动数据流，将前后两次计算相关联，从而利用已有结果降低计算量. 同时，也通过建立 R 树索引，进一步降低数据更新代价. 实验结果表明，本文算法计算效率高，特别适合于计算频度要求很高的场合，可以近似实时形式输出结果.

参考文献：

[1] BORZSONYI S, KOSSMANN D, STOCKER K. The skyline operator[C]// Proceedings of the 17th International Conference on Data Engineering. New York, USA: IEEE, 2001: 421-430.

[2] CHOMICKI J, GODFREY P, GRYZ J, et al. Skyline with presorting[C] // Proceedings of ICDE. New York, USA: IEEE, 2003: 717-816.

[3] KOSSMANN D, RAMSAK F, ROST S. Shooting stars in the sky: an online algorithm for skyline queries [C] // Proceedings of VLDB. San Francisco, USA: Morgan Kaufmann, 2002: 275-286.

[4] PAPADIAS D, TAO Y, FU G, et al. An optimal and progressive algorithm for skyline queries[C] // Proceedings of the ACM SIGMOD International Conference on Management of Data. New York, USA: ACM, 2003: 467-478.

[5] PAPADIAS D, TAO Y, FU G, et al. Progressive skyline computation in database systems[J]. ACM Transactions on Database Systems, 2005, 30(1): 41-82.

[6] JENSEN C S, FRIIS-CHRRISTENSEN A, PEDERSEN T B, et al. Location-based services: a database perspective[C] // Proceeding of 8th Scandinavian Research Conference on Geographical Information Science. Aas, Norway: Agricultural University of Norway, 2001: 59-68.

[7] BABCOCK B, BABU S, DATAR M, et al. Models and issues in data streams[C] // Proceeding of the 21st ACM Symposium on Principles of Database Systems. New York, USA: ACM, 2002: 1-16.

[8] GABER M M, ZASLAVSKY A, KRISHNASWAMY S. Mining data streams: a review[J]. ACM SIGMOD Record, 2005, 34(2): 18-26.

[9] LIN X, YUAN Y, WANG W, et al. Stabbing the sky: efficient skyline computation over sliding Windows[C] // Proceeding of ICDE. New York, USA: IEEE, 2005: 502-513.

[10] TAO Y, PAPADIAS D. Maintaining sliding window skylines on data streams[J]. IEEE Transactions on Knowledge and Data Engineering, 2006, 18(3): 377-391.

[11] LU Hua, ZHOU Yongluan, HAUSTAD J. Continuous skyline monitoring over distributed data streams [C] // Proceedings of the 22nd International Conference on Scientific and Statistical Database Management. Heidelberg, Germany: Springer, 2010: 565-583.

[12] 张丽，邹鹏，贾焰，等. 数据流上连续动态 skyline 查询研究[J]. 计算机研究与发展，2011，48(1)：77-85. ZHANG Li, ZOU Peng, JIA Yan, et al. Continuous dynamic skyline queries over data stream[J]. Journal of Computer Research and Development, 2011, 48(1): 77-85.

[13] SU Hui Zhu, WANG En Tzu, CHEN A L P. Contin-

uous probabilistic skyline queries over uncertain data streams[C]// Proceedings of the 21st International Conference on Database and Expert Systems Applications. Heidebery, Germany: Springer, 2010: 105-121.

[14] HUANG Z, LU H, OOI B, et al. Continuous skyline queries for moving objects[J]. IEEE Transactions on Knowledge and Data Engineering, 2006, 18(12): 1645-1658.

[15] LEE M W, HWANG S W. Continuous skylining on volatile moving data[C]// Proceedings of the IEEE International Conference on Data Engineering. New York, USA: IEEE, 2009: 1568-1575.

[16] ZHENG B, LEE K C K, LEE W C. Location-dependent skyline query[C]// Proceedings of the 9th International Conference on Mobile Data Management. New York, USA: IEEE, 2008: 148-155.

[17] LIN Xin, XU Jianliang, HU Haibo. Range-based skyline queries in mobile environments[J]. IEEE Transactions on Knowledge and Data Engineering, 2011, 23(11): 1059-1072.

[18] KUNG H T, LUCCIO F, PREPARATA F P. On finding the maxima of a set of vectors[J]. Journal of the ACM, 1975, 22(4): 469-476.

[19] 魏小娟, 杨婧, 李翠平, 等. Skyline 查询处理[J]. 软件学报, 2008, 19(6): 1386-1400.
WEI Xiaojuan, YANG Jing, LI Cuiping, et al. Skyline query processing[J]. Journal of Software, 2008, 19(6): 1386-1400.

[20] GUTTMAN A. R-trees: a dynamic index structure for spatial searching[C]// Proceedings of SIGMOD. New York, USA: ACM, 1984: 47-57.

[21] LEUTENEGGER S, LOPEZ M, EDGINGTON J. STR: an efficient and simple algorithm for R-tree packing[C]// Proceedings of the 13th IEEE International Conference on Data Engineering. New York, USA: IEEE, 1997: 497-506.

[22] DONGSEOP K, SANG J L, WONIK C, et al. An adaptive hashing technique for indexing moving object [J]. Data & Knowledge Engineering, 2006, 56(3): 287-303.

（编辑 杜秀杰）