软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software, 2023,34(3):1425−1450 [doi: 10.13328/j.cnki.jos.006718]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

高速流环境下近似连续 *k* 代表轮廓查询算法*

朱 睿[1], 宋枺尧[1], 王 斌[2], 杨晓春[2], 张安珍[1], 夏秀峰[1]

[1](沈阳航空航天大学 计算机学院, 辽宁 沈阳 110136)

[2](东北大学 计算机科学与工程学院, 辽宁 沈阳 110169)

通信作者: 王斌, E-mail: wangbin@neu.edu.cn

摘 要: *k* 代表轮廓查询是从传统轮廓查询中衍生出来的一类查询. 给定多维数据集合 *D*, 轮廓查询从 *D* 中找到所有不被其他对象支配的对象, 将其返回给用户, 便于用户结合自身偏好选择高质量对象. 然而, 轮廓对象规模通常较大, 用户需要从大量数据中进行选择, 导致选择速度和质量无法得到保证. 与传统轮廓查询相比, *k* 代表轮廓查询从所有轮廓对象中选择"代表性"最强的 *k* 个对象返回给用户, 有效地解决了传统轮廓查询存在的这一问题. 给定滑动窗口 *W* 和连续查询 *q*, *q* 监听窗口中的数据. 当窗口滑动时, 查询 *q* 返回窗口中, 组合支配面积最大的 *k* 个对象. 现有算法的核心思想是: 实时监测当前窗口中的轮廓对象集合, 当轮廓对象集合更新时, 算法更新 *k* 代表轮廓. 然而, 实时监测窗口中, 轮廓集合的计算代价通常较大. 此外, 当轮廓集合规模较大时, 从中选择 *k* 代表轮廓的计算代价是同样巨大的, 导致已有算法无法在高速流环境下使用. 针对上述问题, 提出了 ρ-近似 *k* 代表轮廓查询. 为了支持该查询, 提出了查询处理框架 PAKRS (predict-based approximate *k* representative skyline). 首先, PAKRS 利用高速流的特性对当前窗口进行划分, 根据划分结果构建未来窗口预测结果集, 用其预测新流入窗口数据成为轮廓对象的最早时间. 其次, 提出了索引 ρ-GRID. 它帮助 PAKRS 在 2 维和 *d* 维(d>2)环境下, 分别以 $O(k/s+k/m)$和 $O(2^L d/m+2^L d/s)$的增量维护代价下筛选近似 *k* 代表轮廓, *L* 是一个小于 *k* 的正整数. 由理论分析证明可知, PAKRS 的计算复杂度小于前人所提的算法计算复杂度. 最后, 通过大量实验对所提算法性能进行评估. 结果表明, PAKRS 的运行时间是 PBA (prefix-based algorithm)算法的 1/4、GA (greedy algorithm)算法的 1/6、ε-GA (ε-constraint greedy algorithm)算法的 1/3.

关键词: 轮廓查询; *k* 代表轮廓查询; 滑动窗口; 分片; 高速流

中图法分类号: TP311

中文引用格式: 朱睿, 宋枺尧, 王斌, 杨晓春, 张安珍, 夏秀峰. 高速流环境下近似连续 *k* 代表轮廓查询算法. 软件学报, 2023, 34(3): 1425–1450. http://www.jos.org.cn/1000-9825/6718.htm

英文引用格式: Zhu R, Song FY, Wang B, Yang XC, Zhang AZ, Xia XF. Approximate Continuous *k* Representative Skyline Query Algorithm over High-Speed Streaming Data Environment. Ruan Jian Xue Bao/Journal of Software, 2023, 34(3): 1425−1450 (in Chinese). http://www.jos.org.cn/1000-9825/6718.htm

Approximate Continuous *k* Representative Skyline Query Algorithm over High-Speed Streaming Data Environment

ZHU Rui[1], SONG Fu-Yao[1], WANG Bin[2], YANG Xiao-Chun[2], ZHANG An-Zhen[1], XIA Xiu-Feng[1]

[1](School of Computer Science, Shenyang Aerospace University, Shenyang 110136, China)

[2](School of Computer Science and Engineering, Northeastern University, Shenyang 110169, China)

Abstract: *k* representative skyline query is a type of query derived from traditional skyline query. Given a set of *d*-dimensional dataset *D*, a skyline query finds all objects in *D* that are not dominated by other ones, which helps users to select high-quality objects based on their

* 基金项目: 国家自然科学基金(62102271, 62072088, 61991404); 国家重点研发计划(2020YFB1707901); 沈阳市创新人才项目(RC200439)

收稿时间: 2021-11-25; 修改时间: 2022-02-07, 2022-04-27; 采用时间: 2022-06-05

preference. However, the scale of skyline objects may be large in many cases, users have to choose target objects from a large number of objects, leading that both the selection speed and quality cannot be guaranteed. Compared with traditional skyline query, k representative skyline query chooses the most "representative" k objects from all skyline objects, which effectively solves such problem causes by traditional skyline query. Given the sliding window W and a continuous query q, q monitors objects in the window. When the window slides, q returns k skyline objects with the largest group dominance size in the window. The key behind existing algorithms is to monitor skyline objects in the current window. When the skyline set is updated, the algorithm updates k representative skyline set. However, the cost of monitoring skyline set is usually high. When the skyline set scale is large, the computational cost of choosing k representative skyline objects is also high. Thus, existing algorithms cannot efficiently work under high-speed stream environment. This study proposes a query named ρ-approximate k representative skyline query. In order to support this type of queries, a novel framework is proposed named PAKRS (predict-based approximate k representative skyline). Firstly, PAKRS partitions the current window into a group of sub-windows. Next, the predicted result sets of a few future windows are constructed according to the partition result. In this way, the earliest moment can be predicted when new arrived objects may become skyline objects. Secondly, an index is proposed named ρ-GRID, which can help PAKRS to select ρ-approximate k representative skyline with $O(k/s+k/m)$ computational cost under 2-dimensional space, and $O(2^L d/m+2^L d/s)$ computational cost under d-dimensional space (d>2), where L is a little integer smaller than k. Theoretical analysis shows that the computational complexity of PAKRS is lower than the state-of-the-art efforts. Extensive experiments have been conducted to confirm the efficiency and effectiveness of the proposed algorithms. Experimental results show that the running time of PAKRS is about 1/4 times of PBA (prefix-based algorithm), algorithm 1/6 times of GA (greedy algorithm) and about 1/3 times of ε-GA (ε-constraint greedy algorithm).

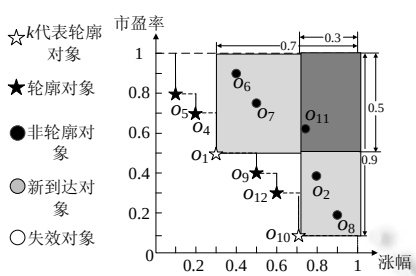
Key words: skyline query; k representative skyline query; sliding window; partition; high-speed streaming

k 代表轮廓查询[1−5]是从传统轮廓查询[6,7]中衍生出来的一类查询, 它从整体轮廓集合中选择 k 个具有代表性的对象返回给用户, 便于用户做出决策, 具有重要的研究意义. 给定一组 d 维对象集合 D, $\forall o_1,o_2\in D$, 如果 o_1 在所有维度上的值都不大于 o_2, 且至少在一个维度上的值小于 o_2, 则称 o_1 支配 o_2(记为 $o_1\prec o_2$). $\forall o\in D$, 如果 o 不被 D 中其他对象支配, 则称 o 为 D 中的轮廓对象. D 中所有轮廓对象组成了轮廓集合. 轮廓查询的作用是从数据集中找到具有代表性的对象, 将其推荐给用户, 方便用户进行决策. 当轮廓集合规模较小时, 用户可根据推荐结果进行快速决策. 然而, 轮廓集合规模通常对数据分布、维度、规模均较为敏感. 这导致轮廓集合规模通常较大, 最坏情况下, 与数据规模呈线性关系, 用户很难在轮廓集合中进行快速且高质量的决策. 鉴于此, k 代表轮廓查询被提出. 与传统轮廓查询相比, 它从轮廓集合筛选出少量更具"代表性"的对象推荐给用户. 由于参数 k 往往较小, 这使得用户可基于小规模数据进行决策, 从而保证了决策效率和质量. k 代表轮廓的定义方法有很多种, 本文采用 Bai 等人[5]提出的定义展开研究, 目标是在轮廓集合中找到一个包含 k 个对象的子集. 在所有包含 k 个对象的子集中, 该子集组合支配区域面积/体积最大(即对象支配区域的并集最大). 研究表明, 该定义下选择的 k 个对象更具有代表性.

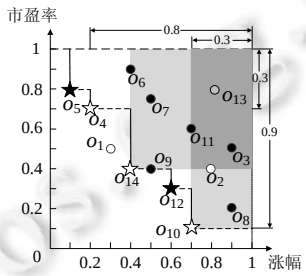
本文研究 k 代表轮廓查询在数据流环境下的一类重要应用: 面向高速流的 k 代表连续查询问题. 它在流数据管理领域被广泛使用. 例如, 在股票推荐系统中, 股票交易市场实时产生大量数据, 而人们在分析时通常只关注最近一段时间内产生的股票信息. 因此, 系统可实时监控最近一段时间内产生的股票交易数据, 每当新的交易数据产生/旧的交易数据过期时, 系统将"代表性"最强的 k 只股票组合推荐给用户, 帮助用户进行决策. 其中,系统将近期产生的数据视为流数据, 每秒新产生(或过期)的数据个数视为流速. 此外, 在森林火灾监控系统中, 海量传感器每时每刻产生大量的数据, 系统根据传感器采集数据进行实时分析, 每当新数据产生/旧数据过期时, 系统根据各个监控地点的温度、湿度、风速, 实时监测火灾隐患较高的 k 个地点组合, 从而达到实时重点防控的目的.

不失一般性地, 本文利用滑动窗口模型刻画流数据模型. 滑动窗口可分为基于计数的窗口和基于时间的窗口. 其中, 基于计数的窗口 $W\langle N,s\rangle$包含最近 N 个到达窗口的对象, s 表示窗口滑动时流入/流出窗口的对象数目. 它可用于刻画流速, s 越大, 窗口滑动时流入/流出窗口的对象越多, 流速越快. s 的另一个作用是将长度为 N 的窗口 W 划分为一组长度为 s 的滑片$\{s_0,s_1,\ldots,s_{m-1}\}$. 任意滑片 s_i 中包含 s 个第 i 批次流入窗口的数据. 基于时间的窗口 $W\langle N,s\rangle$包含最近 N 个时刻内到达窗口的对象, s 指窗口滑动的时间间隔(即每隔 s 时刻滑动一次). 在接下来的内容中, 本文以第 1 类窗口为例描述算法, 但本文所提算法同样适用于第 2 类窗口.

例 1: 用户在购买股票时, 通常喜欢选择低风险的股票进行投资. 在众多股票属性中, 市盈率和近 5 分钟涨幅是两个能够反映股票风险高低的重要指标(用户通常喜欢选择市盈率低且近期涨幅小的股票). 如图 1(a)所示, 滑动窗口 W_0 包含了对象 o_1–o_{12}, 它们表示 12 只股票数据. 其中, o_1, o_4, o_5, o_9, o_{10} 和 o_{12} 不被其他对象支配, 它们构成了轮廓对象集合. 当 k=2 时, 可通过计算轮廓集合中任意两对象的组合支配面积找到 2 代表轮廓. 以集合$\{o_1,o_{10}\}$为例, 对象 o_1 的支配区域是一个边长为 0.7×0.5 的矩形, 其面积为 0.35; 对象 o_{10} 支配区域的面积为 0.27; 而 o_1 和 o_{10} 的公共支配区域(深色阴影区域)的面积为 0.15. 因此, 集合$\{o_1,o_{10}\}$组合支配区域(即支配区域并集)的面积为 0.35+0.27–0.15=0.47. 在计算出两两对象组合对应的支配面积后, 可发现对象 o_1 和 o_{10} 的组合支配面积最大(详见图 1(c)). 因此, 滑动窗口 W_0 下 2 代表轮廓集合为$\{o_1,o_{10}\}$. 当窗口从 W_0 滑动到 W_1 后, o_1, o_2 流出窗口, o_{13}, o_{14} 流入窗口(见图 1(b)). W_1 中的轮廓集合更新为$\{o_4,o_5,o_{10},o_{12},o_{14}\}$, 2 代表轮廓集合更新为$\{o_{10},o_{14}\}$(详见图 1(c)).



(a) 基于 W_0 的 2 代表轮廓



(b) 基于 W_1 的 2 代表轮廓

	o_1	o_4	o_5	o_9	o_{10}	o_{12}
o_1	—	0.38	0.39	0.4	0.47	0.42
o_4	—	—	0.26	0.39	0.42	0.4
o_5	—	—	—	0.38	0.39	0.38
o_9	—	—	—	—	0.39	0.34
o_{10}	—	—	—	—	—	0.34
o_{12}	—	—	—	—	—	—

	o_{14}
o_4	0.3
o_5	0.3
o_{10}	0.45
o_{12}	0.36

(c) 组合支配面积

图 1 滑动窗口下, k 代表轮廓连续查询(N=12, s=2, k=2)

Bai 等人首次研究了数据流环境下的 k 代表轮廓查询问题, 并提出了算法框架 k-LDS (k-largest dominance skyline). 其核心思想是, 利用连续轮廓查询算法实时维护当前窗口中的轮廓集合. 当轮廓集合发生改变时, 该工作分别提出了算法 PBA (prefix-based algorithm)和近似算法ε-GA (ε-constraint greedy algorithm)支持 2 维和 d 维(d>2)环境下的 k 代表轮廓查询. 该工作需要维护窗口中所有“潜在”轮廓对象, 从而引发了高昂的计算代价. 给定对象 o, 如果它不被晚于其到达窗口的对象支配, 则称为“潜在”轮廓对象. 反之, 因为 o 在流出窗口前无法成为轮廓对象, 所以称为无效对象. 分析可知, “潜在”轮廓对象集合的维护代价对数据分布、数据维度较为敏感, 其维护代价通常较高. 此外, PBA 算法和ε-GA 算法需基于全体轮廓对象选择 k 代表轮廓, 其计算代价与轮廓对象规模呈线性关系. 当轮廓集合规模较大或流数据更新频繁时, 这些算法同样需要消耗高昂的计算代价. 由于高速流具有数据规模大、更新频繁、流速快等特点, k-LDS 通常无法在高速流环境下使用.

鉴于上述问题, 本文提出了ρ-近似连续 k 代表轮廓查询. 给定滑动窗口 $W\langle N,s\rangle$, ρ-近似连续 k 代表轮廓查询 q 监听窗口中的数据, 窗口滑动时, s 个对象流入窗口, s 个过期对象流出窗口, 查询 q 返回当前窗口中近似 k 代表轮廓集合 ARS (approximate representative skyline). 和精确 k 代表轮廓集合 RS (representative skyline)相比, 二者支配区域面积(或体积)的比值小于参数ρ. 换言之, 令 $DS|(RS)|$和 $DS|(ARS)$分别表示 RS 和 ARS 中对象对应的组合支配面积(或体积), 则 $\frac{|DS(RS)|}{|DS(ARS)|}\leqslant\rho$. 本文致力于提出一组算法, 一方面可利用高速流的性质降低“潜在”轮廓对象维护代价; 另一方面, 可通过访问少量轮廓对象计算近似 k 代表轮廓. 在保证近似比小于ρ的前提下, 降低轮廓对象规模对算法性能带来的冲击,提高运行效率.

为了支持近似 k 代表轮廓的高效查询, 本文需克服以下挑战.

(i) “潜在”轮廓对象的规模较大, 高效维护这些对象间的支配关系是十分困难的. 如何利用高速流的特性提高维护效率, 是本文面临的第 1 个挑战.

(ii) 轮廓集合通常规模较大, 如何快速地从中选择少量对象参与近似计算是同样困难的. 如何高效地从轮廓集合中选择少量对象参与近似 k 代表轮廓对象计算, 是本文遇到的第 2 个挑战.

鉴于此, 本文提出查询处理框架 PAKRS (predict-based approximate k representative skyline). 首先, PAKRS 利

用高速流的特性将当前窗口 W_0 中的对象按照其到达顺序划分到一组分片 $\{P_0,P_1,P_2,\ldots,P_t\}$($t$=⌊$\log_2 m$⌋,$m$=$N$/$s$); 随后, PAKRS 根据划分结果删除各分片内的无效对象, 预测各未来窗口中的轮廓对象. 以此为基础, PAKRS 构建近似 k 代表轮廓集合支持当前窗口下的查询, 构建少量未来窗口下的预测近似 k 代表轮廓集支持后续增量维护. 为实现近似 k 代表轮廓集的高效构建, 本文提出了索引ρ-GRID. 它根据参数ρ设置网格的划分粒度, 利用网格单元格之间的位置关系, 选择少量轮廓对象参与近似 k 代表轮廓的计算, 从而降低轮廓对象规模对算法性能带来的影响. 当窗口滑动时, PAKRS 扫描新流入窗口中的对象, 预测它们成为轮廓对象的最早时间, 更新预测结果集, 周期性地对窗口进行划分, 利用ρ-GRID 更新近似k代表轮廓集. 与传统连续轮廓查询算法相比, PAKRS 只需利用少量未来窗口下的预测近似 k 代表轮廓对象即可实现增量维护, 从而降低了计算代价. 与此同时, ρ-GRID 可进一步帮助 PAKRS 降低近似 k 代表轮廓集更新代价. 本文的主要贡献如下.

(1) 提出了ρ-近似 k 代表轮廓查询. ρ是一个阈值, 用于限制精确结果与近似查询结果所对应组合支配面积(或区域)的比值.

(2) 提出了带误差保证的近似 k 代表轮廓查询算法. 它在 2 维和 d 维(d>2)环境下, 分别以 O(k/s+k/m), O($2^L d$/m+$2^L d$/s)的计算代价, 实现近似 k 代表轮廓查询(L 是一个小常数, m=N/s), 计算代价与轮廓集合规模无关.

(3) 提出了基于预测结果集的增量维护算法. 该算法通过对窗口进行划分, 在最坏情况下, 使用 O($k\log_2 m$)个对象实现预测结果集的增量维护. 结合本文提出的ρ-GRID 相关算法, PAKRS 的均摊增量维护代价在 2 维和 d(d>2)维环境下分别为 O($\log_B 2(k\log_2 m)$+$\log_2 2s$+k/m+k/s) 和 O($2^L d$/m+$2^L d$/s+$\log_2 ds$+$\log_B d(k\log_2 m)$). B 是一个大于 2 的常数. 如第 3.3 节所述, PAKRS 利用“流速”s 这一参数降低了增量维护代价, 其时间复杂度低于 k-LDS.

本文第 1 节介绍相关工作. 第 2 节给出问题定义. 第 3 节介绍 PAKRS 查询处理框架. 第 4 节通过大量的实验对 PAKRS 性能进行评估. 第 5 节是结论.

1 相关工作

轮廓查询是多维数据管理[8−18]领域中的经典问题, 轮廓查询及其变种已被大批学者深入研究. 和本文相关的研究包含 4 类: 滑动窗口模型下连续轮廓查询[10−18]、k 支配查询、基于静态数据的 k 代表轮廓查询以及基于滑动窗口的 k 代表轮廓查询.

Lin 等人[10]研究了基于滑动窗口的连续轮廓查询问题, 提出了 n-of-N 轮廓查询算法. 该算法利用间隔树保存候选对象成为轮廓查询结果的时间段, 实现轮廓集合的快速计算. Tao 等人[11]提出了算法 Lazy 和 Eager 支持查询. 算法利用数据间的时序关系, 删除窗口中的无效对象. Wang 等人[12]对 Tao 等人提出的 Lazy 算法加以改进, 提出了 NNSC (nearest neighbor-based skyline computation)算法. 该算法利用最近邻过滤的思想, 对数据集中每个对象设置一个阈值, 并对其进行预处理, 从而降低了计算代价. Tang 等人[13]提出了 KDStreamSky (KD-tree-based stream skyline)算法. 它采用事件链机制处理数据点的状态变化, 快速响应用户查询要求. Ren 等人[14]针对不完整数据流展开研究, 提出了算法 Sky-iDS (skyline over incomplete data stream). 它从不完整数据流中检索具有高可信度的轮廓对象, 实现不完整数据流中对象缺失属性的填充. Karim 等人[15]提出了面向多维数据流的子空间轮廓查询处理框架. 为支持增量维护, 他们进一步引入了一个可定期维护的索引结构 NSCT (negative skycube with timestamps)存储被支配的子空间集合. 这些工作的核心是维护所有“潜在”轮廓对象, 当有数据新流入窗口时删除无效对象, 从而降低空间代价. 其中, 给定对象 o, 如果它不被所有晚于其到达的对象支配, 则称其为“潜在”轮廓对象; 反之, 因为 o 在其生命周期内无法成为轮廓对象, 所以称其为无效对象. 然而, 一个对象是否为“潜在”轮廓对象不仅取决于它在各维度上的值, 更取决于其流入窗口的时间. 许多在各维度上值较大但流入窗口时间较晚的对象也是“潜在”轮廓对象. 在最坏情况下, “潜在”轮廓对象与窗口中对象的规模相同, 这导致“潜在”轮廓对象的维护代价较高.

k 支配查询是一类与 k 代表轮廓查询相似的查询. 如 Tao 等人[19]所述, 这类查询将 k 个支配对象数量最多

的对象返回给用户. 该查询控制了返回结果数量, 但不保证查询结果全部为轮廓对象. Chan 等人[20]提出了一种新的定义, 他们将其定义为返回在指定 k 个维度上不被其他对象支配的对象. Li 等人[21]研究了路网环境下的 k 支配查询问题, 将 k 支配查询应用到路网轮廓查询中, 解决了道路网环境下的多目标查询和决策问题. Cui 等人[22]针对移动过程中的 k 支配最近邻查询展开研究, 将 k 近邻与轮廓查询相结合, 返回了在空间属性和非空间属性上均不被支配且与查询点 q 距离最小的 k 个对象. 与 k 支配查询相似, 许多学者研究了 k-GSky 查询. 此类查询返回 k 组支配能力最强的对象. 查询结果中的对象也可包含非轮廓对象. 在众多研究成果中, Zhu 等人[23]提出了一种基于位图索引的方法支持查询, Wang 等人[24]提出了一种基于最小支配图的结构支持查询, Zhou 等人[4]提出了分层优化等方法加速查询.

与 k 支配查询不同, k 代表轮廓查询返回的所有对象必须是轮廓对象. 在静态数据环境下, Lin 等人[1]最早提出了 k 代表轮廓查询的概念, 并提出了 RSP (representative skyline points)算法. 他们利用对象的支配个数衡量其代表能力, 利用动态规划返回 k 个代表能力最强的轮廓对象. Tao 等人[2]提出了 DRS (distance-based representative skyline)算法. 它是一种基于距离的 k 代表轮廓查询算法. 该算法用轮廓对象间的距离衡量查询结果的代表能力, 返回 k 个与其他轮廓对象距离差值最大的对象作为最终查询结果. Sarma 等人[3]引入了点击概率的概念, 提出了一种基于点击概率的 k 代表轮廓查询算法, 期望返回 k 个具有最大点击概率的轮廓对象作为查询结果. Bai 等人[4]研究了全序域和偏序域上基于距离的 k 代表轮廓查询问题, 使用改进极限学习机 ELM (extreme learning machine)将整个轮廓集合划分为 k 个聚类, 针对每个聚类计算轮廓对象, 从而计算最终结果. 此外, Bai 等人[25]研究了基于最大覆盖的代表轮廓优化算法. 该算法利用前缀支配表加速查询, 其作用是使用少量加减运算实现代表轮廓的筛选. Malene 等人[26]给出了该问题的解决方案. 然而, 该算法存在的问题是计算的复用率较低, 这导致算法的整体效率相对较低. Huang 等人[27]针对该问题的鲁棒性展开研究, 其核心思想是: 采用贪心算法求出一个 coreset 集合, 随后, 针对该集合进行计算. Zhou 等人[28]提出一种新的查询 TFPP (top k favority probabilistic products), 其作用是为不同用户找到满足不同偏好的 k 个产品. Zhou 等人[29]利用 k 代表轮廓查询技术, 进一步研究了约束条件下的最优产品组合价格促销问题.

在数据流环境下, Bai 等人[5]首次研究了数据流环境下的 k 代表轮廓查询问题. 该工作利用组合支配区域评价查询结果的代表性. 如前文所述, 他们提出了算法框架 k-LDS, 并分别提出了精确算法 PBA (prefix-based algorithm)支持 2 维环境下的 k 代表轮廓查询和近似算法 ε-GA 支持 d 维(d>2)环境下的 k 代表轮廓查询. 分析可知, 算法在 2 维和 d 维(d>2)环境下的增量维护代价分别为 $O(\log_B 2|C|+(|SK|-k)k)$和 $O(\log_B d|C|+d+(|SK|-L)2^L d)$. $|SK|$表示窗口中轮廓对象的数量, L 为一个比 k 小的正整数, $|C|$为潜在轮廓集合的规模. B 是 R-Tree 中各结点最大容量.

Lin、Tao、Sarma、Zhou、Huang 等人均基于静态数据展开研究, 这些算法无法在流环境下高效工作. Bai 等人所提算法虽可在流环境下工作, 但需要利用连续轮廓查询算法维护当前窗口中的轮廓集合, 引发了高昂的计算代价. 此外, 当$|SK|$较大或窗口中轮廓集合更新频繁时, 算法计算代价也会相应升高, 同样无法在高速流环境下使用. 因此, 找到一个可在高速流环境下高效工作且返回高质量查询结果的算法, 具有非常重要的理论意义和应用价值.

2 问题分析及定义

本节首先介绍滑动窗口、支配区域、组合支配区域以及 k 轮廓查询的基本概念, 随后正式介绍问题定义. 为便于描述, 给定 d 维空间中的对象 o, 它可表示为元组$\langle o[1],o[2],\ldots,o[d]\rangle$, SK(skyline 的简写)表示轮廓对象集合. 表 1 为本文所用符号.

给定滑动窗口 $W\langle N,s\rangle$, 可分为基于计数的窗口和基于时间的窗口. 基于计数的窗口包含最近 N 个到达窗口的对象, s 表示窗口滑动时流入/流出窗口的对象数目. 基于时间的窗口包含最近 N 个时刻内到达窗口的对象, s 指窗口滑动的时间间隔(即每隔 s 时刻滑动一次). 以第 1 类窗口为例, 给定长度为 N 的窗口 W, 每当窗口滑动时, s 个新到达的对象流入窗口, 窗口中最早流入的 s 个对象流出窗口. s 可用于刻画流速, s 越大, 流速越快. 参

数 s 的另一作用是将窗口 W 划分为一组长度为 s 的滑片$\{s_0,s_1,\ldots,s_{m-1}\}$. 滑片 s_i 中包含 s 个第 i 批次流入窗口的数据, $m(=N/s)$是滑动窗口的一个自然属性,表示滑片的个数.

表 1 符号表

符号	含义
$DS(o_i)$	对象 o_i 的支配区域
$DS(O)$	对象集合 O 中对象对应的组合支配区域
$IS(O)$	对象集合 O 中对象支配面积的交集
C	候选轮廓集合
N	滑动窗口长度
ARS	近似 k 代表轮廓集合,
RS	精确 k 代表轮廓集合,
SK	轮廓集合
ρ	近似 k 代表轮廓近似比
n	网格划分粒度
S	精确 k 代表轮廓集合的组合支配面积
c_{FD}, c_{SD}, c_S	全支配单元格、半支配单元格、轮廓单元格
TSK	临时轮廓集合

接下来, 本节介绍对象支配区域和组合支配区域的概念.

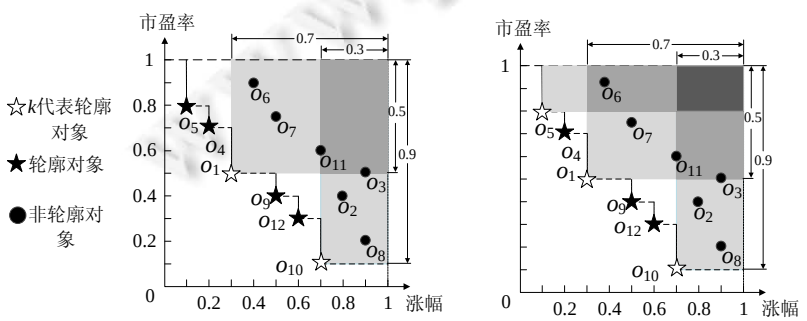
给定 d 维空间$[0,1]^d$中的对象集合 D, $\forall o\in D$, 它可以支配位于其右上方矩形(或超矩形)中的对象. 该矩形(或超矩形)被称为对象 o 的支配区域, 记为 $DS(o)$. $DS(o)$左下方和右上方坐标分别由元组$\langle o[1],o[2],\ldots,o[d]\rangle$和$\langle 1,1,\ldots,1\rangle$表示. 它的面积$|DS(o)|$(或体积)等于 $\prod_{i=1}^{i=d}1-o[i]$.

给定 d 维对象集合 D 的子集 $O\{o_1,o_2,\ldots,o_s\}$, O 的组合支配区域(记 $DS(O)$)为 O 中各对象支配区域的并集, 它的值可由公式(1)计算得到.

$$|DS(O)|=\sum_{i=1}^{|O|}(-1)^{i+1}\sum_{O_i}|IS(O_i)| \tag{1}$$

O_i 指 O 中由任意 i 个对象组成的集合. $IS(O_i)$指集合 O_i 中对象支配区域的交集. 当 i=1 时, $IS(O_1)$指单个对象的支配区域. k 代表轮廓集合表示一个规模为 k 的轮廓对象子集. 在所有包含 k 个对象的子集中, 该子集对象组合支配区域面积(或体积)最大.

如图 2(a)所示, 给定集合 $O=\{o_1,o_4\}$, O 的支配区域 $DS(O)$由图 2(a)中灰色部分所标记, 其支配面积 $|DS(O)|=0.35+0.27-0.15=0.47$. 在所有规模为 2 的轮廓对象子集中, $O=\{o_1,o_{10}\}$的组合支配面积最大, 是对象集合 D 的 2 代表轮廓. 如图 2(b)所示, 给定集合 $O'=\{o_1,o_5,o_{10}\}$, O'的支配区域 $DS(O')$由图 2(a)中灰色部分所标记, 其支配面积$|DS(O)|=0.35+0.18+0.27-0.14-0.15-0.06+0.06=0.51$. 图 2(b)中, 深灰色部分标记了这 3 个对象支配区域的交集. 在所有规模为 3 的轮廓对象子集中, $O=\{o_1,o_5,o_{10}\}$的组合支配面积最大, 是对象集合 D 的 3 代表轮廓.



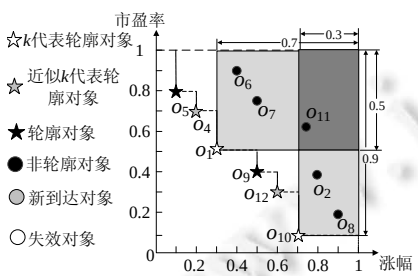
(a) 2 代表轮廓 (b) 3 代表轮廓

图 2 组合支配面积

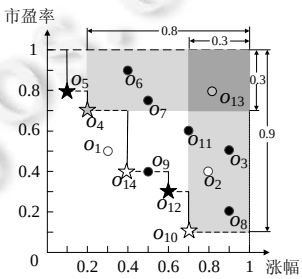
接下来, 本节正式介绍问题定义.

定义 1(近似连续 k 代表轮廓查询). 给定滑动窗口 W, 近似连续 k 代表轮廓查询 $q\langle N,s,k,\rho\rangle$监视窗口中的数据. 当窗口滑动时, s 条数据流入窗口, s 条最早到达的数据流出窗口. q 返回当前窗口中一个近似 k 代表轮廓集合 ARS. 和当前窗口下的精确 k 代表轮廓集合 RS 相比, ARS 满足 $\frac{|DS(RS)|}{|DS(ARS)|}\leqslant\rho$.

如图 3 所示, 假设 s=2, k=2, ρ=1.12, 当前窗口 W_0 的轮廓集合 SK_0={o_1,o_4,o_5,o_9,o_{10},o_{12}}. 组合支配面积最大的集合 RS_0={o_1,o_{10}}, 其支配面积|$DS(RS_0)$|=0.47. 因为|DS({o_1,o_{12}})|=0.42 且 $\frac{|DS(RS_0)|}{|DS(\{o_1,o_{12}\})|}\leqslant 1.12$, {$o_1$,$o_{12}$}可作为近似 k 代表轮廓输出. 同理, {o_4,o_{10}}也可作为近似 k 代表轮廓输出. 由此可见, 近似结果不唯一, {o_1, o_{12}}, {o_4,o_{10}}, {o_1,o_{10}}均可被作为查询结果输出. 当窗口从 W_0 滑动到 W_1 后, 对象 o_1 和 o_2 离开窗口, o_{13} 和 o_{14} 进入窗口. 当前窗口 W_1 中的轮廓集合更新为 SK_1={o_4,o_5,o_{10},o_{12},o_{14}}, RS_1={o_{10},o_{14}}, 其组合支配面积|$DS(RS_1)$|=0.45. 经满足误差约束的集合 $ARS_1=\{o_4,o_0\}\left(\frac{|DS(RS_1)|}{|DS(ARS_1)|}=\frac{0.45}{0.42}=1.07\leqslant\rho\right)$. 因此, 除精确结果外, {$o_4$,$o_{10}$}也可作为近似 k 代表轮廓集.



(a) 基于 W_0 的近似 2 代表轮廓



(b) 基于 W_1 的近似 2 代表轮廓

	o_1	o_4	o_5	o_9	o_{10}	o_{12}
o_1	—	0.38	0.39	0.4	0.47	0.42
o_4	—	—	0.26	0.39	0.42	0.4
o_5	—	—	—	0.38	0.39	0.38
o_9	—	—	—	—	0.39	0.34
o_{10}	—	—	—	—	—	0.34
o_{12}	—	—	—	—	—	—

	o_{14}
o_4	0.3
o_5	0.3
o_{10}	0.45
o_{12}	0.36

(c) 组合支配面积

图 3 滑动窗口下, 近似 k 代表轮廓连续查询(N=12, s=2, k=2, ρ=1.12)

3 查询处理框架 PAKRS

本节提出一种基于预测结果集的查询处理框架 PAKRS (predict-based approximate k representative skyline), 支持连续近似 k 代表轮廓查询. 它利用高速流的特性对窗口进行划分, 根据划分结果预测未来各窗口的轮廓对象, 从而预测新流入对象成为轮廓对象的最早时间. 为了帮助 PAKRS 高效筛选近似 k 代表轮廓, 本节进一步提出了数据结构ρ-GRID, 支持近似 k 代表轮廓查询.

3.1 PAKRS概述

定义 2(预测轮廓集). 给定当前滑动窗口 W_0={s_0,s_1,…,s_{m-1}}, 所有满足如下条件的对象构成了未来窗口 W_i 的预测轮廓集 SK_i: (i) $o\in\bigcup_{i=j}^{m-1}s_i$; (ii) o 不被 $\bigcup_{j=i}^{m-1}s_j$ 中对象支配, 但被 $\bigcup_{j=i-1}^{m-1}s_j$ 中对象支配. 其中, m=N/s, N 为 W_0 中对象的数量; s_i 表示第 i 批流入窗口的数据, 其规模等于 s.

W_0的子窗口 SW_i 包含滑片 $\bigcup_{j=i}^{m-1}s_j$ 中对象, 该子窗口既是当前窗口 W_0 的子窗口, 也是未来窗口 W_i={s_i,s_{1+i},…, s_{m-1+i}}的子窗口. 如果 $o\in\bigcup_{i=j}^{m-1}s_i$ 并且 o 不被 $\bigcup_{j=i}^{m-1}s_j$ 中对象支配, 则它有可能成为未来窗口 W_i 的轮廓对象. 因此, PAKRS 可利用预测轮廓集 SK_i 预测未来窗口 W_i 的轮廓集合. 具体如下.

① $\forall o\in s_u$, 如果 $u\leqslant i$ 且 o 被 SK_i 中对象支配, 那么 o 在流出窗口前不会成为轮廓对象.

② $\forall o'\in s_v$, 如果 $v>i$ 且 o 被 SK_i 中对象支配, 那么 o 在窗口滑动到 W_i 前不会成为轮廓对象.

显然, PAKRS 可利用未来窗口的预测轮廓集评估新增对象成为轮廓对象最早时间. 以此为基础, 算法根

据评估结果预测新增对象成为近似 k 代表轮廓的最早时间, 进而实现近似 k 代表轮廓的增量维护. 然而, 未来窗口预测轮廓集规模通常较大, 基于这部分数据进行增量维护需要消耗较高的计算代价. 鉴于此, 本文进一步提出了 PRSSET (partition-based representative skyline prediction SET)的概念.

定义 3(PRSSET). 给定当前滑动窗口 $W_0=\{s_0,s_1,\dots,s_{m-1}\}$, PAKRS 将 W_0 划分为一组分片 $\{P_0,P_1,\dots,P_t\}$. 其中, P_t 包含了滑片 $\bigcup_{i=m/2}^{m-1} s_i$ 中的对象, P_{t-1} 中包含了 $\bigcup_{i=m/4}^{i=m/2-1} s_i$ 中的对象, P_1 和 P_0 分别包含位于 s_1 和 s_0 中的对象. PRSSET $\boldsymbol{P}$ 包含了当前窗口 W_0 和未来窗口 $\{W_1,W_2,W_4,W_8,\dots,W_{m/2}\}$ 中近似 k 代表预测轮廓集 $\{ARS_0,ARS_1,ARS_2,ARS_4,ARS_8,\dots,ARS_{m/2}\}$.

ARS_i 是基于预测结果集 SK_i 计算得到的. 本文假设分片的数目 $t=\log_2 m$, m 的取值使 t 为正整数(m=2,4,8,16,…). 当 m 的值不符合上述假设时, 本文所提算法仍然适用. 具体原因如下: 假设 $m=m'+\xi$, m'的取值使 t 为正整数(m'=2,4,8,16…), $0<\xi<m'$, 分片 $\{P_0,P_1,\dots,P_t\}$ 分别包含 $\{s_0\},\{s_0\},\dots,\{s_{m'/2},s_{m'/2+1},\dots,s_{m'+\xi-1}\}$ 中的对象, 它们分别包含了 $\{1,1,2,4,\dots,m'/2+\xi\}$ 个滑片中的对象. 由于分片 $\{P_0,P_1,\dots,P_{t-1}\}$ 符合本节所提假设, 可按本节所提策略执行增量维护. 对于 P_t, 当 $\{P_0,P_1,\dots,P_{t-1}\}$ 中的对象全部流出窗口后, 算法需对该分片进行分裂 P_t. 假设 P_t 被分裂为 $\{P_t^1,P_t^2,...,P_t^t\}$, 它们分别包含 $\{s_{m'/2}\},\{s_{m'/2+1}\},\dots,\{s_{3m'/4+1,\dots,s_{m'-1}}\},\{s_{m',\dots,s_{m'+\xi-1}}\}$ 中的对象. $\{P_t^1,P_t^2,...,P_t^{t-1}\}$ 同样满足前文假设, 而 P_t^t 包含了 $\{s_{m',\dots,s_{m'+\xi-1}}\}$ 中的对象, 不满足前文假设. 当有新数据到达时, 这部分数据被并入 P_t^t, 当 P_t^t 中滑片的规模达到 $m'/2+\xi$ 后, $\{P_t^1,P_t^2,...,P_t^{t-1}\}$ 中的对象全部流出窗口. 此时, PAKRS 可重复上述操作对 P_t^t 进行分裂. 综上所述, 当 m 的取值无法使 t 为正整数时, 本文所提算法仍然适用.

如图 4 所示, 假设 m=32, PAKRS 将整个窗口划分为分片 $\{P_0,P_1,\dots,P_5\}$. 其中, $P_5=\{s_{16},\dots,s_{31}\}$, $P_4=\{s_8,\dots,s_{15}\}$,…, $P_1=s_1$, $P_0=s_0$. PRSSET $\boldsymbol{P}$ 包含了未来窗口窗口 $\{W_0,W_1,W_2,W_4,\dots,W_{16}\}$ 对应的预测近似 k 代表轮廓集 $\{ARS_0,ARS_1,ARS_2,ARS_4,ARS_8,\dots,ARS_{16}\}$. 这样一来, PAKRS 可利用 $\boldsymbol{P}$ 预测新流入对象成为轮廓对象的最早时间, 进而评估其成为近似 k 代表轮廓的最早时间. 当窗口从 W_0 滑动 W_1 后, s_{32} 进入窗口. $\forall o\in s_{32}$, 如果 o 被 ARS_4 中的对象支配但不被 $\{ARS_8,ARS_{16}\}$ 中的对象支配, 因为 ARS_4 是未来窗口 W_4 的预测近似 k 代表预测结果集, o 最早在 W_5 成为查询结果. 与此同时, 如果 o 能够支配 ARS_8 中的对象 O', 那么 O'和所有被 O'支配的对象均可被删除. 当窗口从 W_0 滑动到 W_5 时, 如果 o 仍是 W_5 的轮廓对象, PAKRS 评估其是否为近似 k 代表轮廓对象, 如果是, 则将其加入 PRSSET $\boldsymbol{P}$.

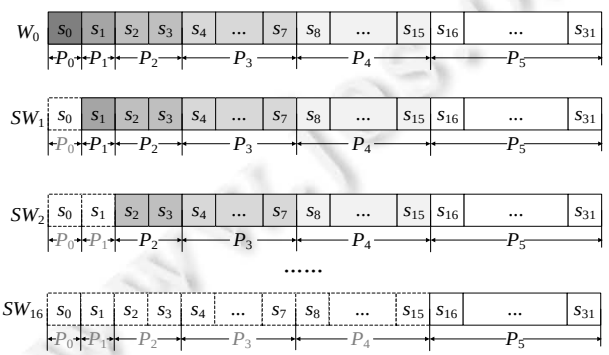


图 4 基于高速流的分片策略(m=32)

为了构造 PRSSET $\boldsymbol{P}$, 本文提出了算法 PRSI (prediction representive skyline initialization). 给定初始化窗口 W_0, PRSI 逆序扫描窗口中元素, 利用窗口中对象间的时序关系和支配关系删除无效对象, 保留未来窗口 $\{W_0,W_1,W_2,W_4,\dots,W_{m/2}\}$ 的预测轮廓集 $\{SK_0,SK_1,SK_2,SK_4,\dots,SK_{m/2}\}$. 为了计算 $\{ARS_0,ARS_1,ARS_2,ARS_4,\dots,ARS_{m/2}\}$, 本文进一步提出了 ρ-GRID 的概念, 并提出了基于 ρ-GRID 的近似 k 代表轮廓查询算法 RGRSS (RhoGrid-based representive skyline selection) (详见第 3.2 节). 与传统 k 代表轮廓查询算法相比, RGRSS 利用网格单元格之间的位置关系选取近似 k 代表轮廓. 分析可知, RGRSS 可返回带误差保证的 k 代表轮廓.

为了在流数据环境下对 PRSSET$\boldsymbol{P}$ 进行增量维护, 本文提出了算法 INCPRSS (incremental prediction representive skyline selection)(详见第 3.3 节). 该算法一方面预测新流入数据成为轮廓对象的最早时间, 另一方面对窗口进行动态划分, 并根据划分结果更新 $\boldsymbol{P}$. 例如, 当$\{s_0,s_1,\ldots,s_7\}$中的对象流出窗口时, P_4 中的对象成为窗口中最早流入窗口的对象, 它包含了$\{s_8,s_9,\ldots,s_{15}\}$中的对象. 此时, PAKRS 对 P_4 划分为一组新的分片 $\{P_4^0,P_4^1,P_4^2,P_4^3\}$, 它们分别包含了$\{s_8\}$,$\{s_9\}$,$\{s_{10},s_{11}\}$和$\{s_{12},\ldots,s_{15}\}$中的对象. 接下来, INCPRSS 通过构建与未来窗口$\{W_8,W_9,\ldots,W_{12}\}$对应的$\{ARS_8,ARS_9,\ldots,ARS_{12}\}$更新 $\boldsymbol{P}$. 为支持未来窗口$\{W_0,W_1,W_2,W_4,\ldots,W_{m/2}\}$下$\{ARS_0,ARS_1,ARS_2,ARS_4,\ldots,ARS_{m/2}\}$的增量维护, 本节提出了近似 k 代表轮廓增量查询算法 INCRGRSS (incremental RhoGrid-based representive skyline selection)(详见第 3.2.3 节). 该算法的均摊代价与轮廓集合规模无关, $\boldsymbol{P}$ 的更新代价也与 SK 规模无关(详见第 3.2.3 节). 由于基于ρ-GRID 的 RGRSS 算法和 INCRGRSS 分别是 PRSI 和 INCPRSS 的子算法, 本节在接下来的内容中首先介绍ρ-GRID 以及基于ρ-GRID 的算法 RGRSS, INCRGRSS.

3.2 基于ρ-GRID的近似k代表轮廓查询算法

3.2.1 ρ-GRID 定义

定义 4(ρ-GRID). 给定近似比ρ, ρ-GRID G 将整个空间划分为规模为 n^d 的网格. 其中, n 指网格划分粒度, 其值为:

$$n=\begin{cases}\left\lceil\dfrac{\rho d}{(\rho-\xi)S^*}\right\rceil, & |c_S|>k,\ d>2\\ \left\lceil\dfrac{\rho d}{(\rho-1)S^*}\right\rceil, & \text{其他}\end{cases}.$$

划分粒度 n 的设置和各变量的含义将在第 3.2.4 节说明. 对于划分之后的单元格, 本文根据它们之间的位置关系将其分为轮廓单元格、全支配单元格以及半支配单元格. 为便于描述, 本节以 2 维数据为例进行介绍.

如果不存非空单元格位于单元格 c 的左方和下方, 则 c 被称为轮廓单元格[30]; 对于任意非空单元格 c', 如果存在轮廓单元格位于其左下方, 由于 c'中的所有对象均被支配, 本节称 c'为全支配单元格; 对于其余非空单元格, 本节称其为半支配单元格. 区分这 3 类单元格的好处是, 可以准确地计算网格划分粒度 n 与近似比ρ之间的关系. 本文将在第 3.2.4 节中给出计算过程. 如图 5 所示, 图中不存在非空单元格位于单元格$\{c_3,c_7,c_{11},c_{15}\}$的左方和下方, 为轮廓单元格; 存在轮廓单元格位于$\{c_9,c_{13},c_{14},c_{17},c_{22}\}$的左下方, 为全支配单元格; 其余非空单元格$\{c_4,c_{16}\}$为半支配单元格.

引入ρ-GRID 的好处如下: (1) PAKRS 是以ρ-GRID 中轮廓单元格为基本单元遴选 k 代表轮廓, 这可以有效避免因大量轮廓对象聚集在小范围区域而产生高昂的计算代价, 也可帮助 PAKRS 在全局范围内选择代表性强的轮廓对象; (2) 同一单元格中不同对象支配面积差小于给定阈值, 算法在执行近似 k 代表轮廓查询时, 可根据轮廓单元格间的位置关系选取近似 k 代表轮廓对象. 因此, ρ-GRID 可在大幅度提高近似 k 代表轮廓查询效率的同时, 将近似结果与精确结果之间的误差控制在一定范围内.

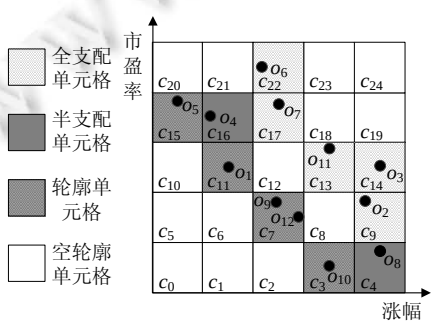


图 5 网格索引ρ-GRID

3.2.2 基于ρ-GRID 的近似 k 代表轮廓查询算法 RGRSS

ρ-GRID 的如下性质, 可帮助RGRSS实现非轮廓对象的快速过滤以及轮廓对象的快速定位: 给定ρ-GRID G、滑动窗口 W_0 以及 W_0 中的轮廓对象集合 SK_0, SK_0 中的对象一定包含在轮廓单元格和半支配单元格中. 原因在于: 对于任意非轮廓单元格中的对象 o, 一定存在位于某轮廓单元格中的对象 o'支配 o. 换言之, 非轮廓单元格中的对象一定不是轮廓对象. 与之相反, 半支配单元格和轮廓单元格中的对象均有可能成为轮廓对象. 因此, 轮廓对象一定位于轮廓单元格和半支配单元格. 基于上述性质, 本节提出算法 1 支持近似 k 代表轮廓查询.

算法 1. The RSK Construction Algorithm.

输入: Skyline Set SK_0.

输出: k-Representative Skyline Set RSK_0.

```
1  if |SK0|≤k then
2      return SK;
3  for i from 1 to |SK0| do
4      cId←getZaddress(SK0[i]); HG←map(SK0[i],cId);
5  VP←getvirtualcoordinate(HG);                    //得到虚拟对象集合
6      VGT←construction(VP);                       //构建 R-Tree
7      VSK0←getSkyline(VGT);                       //计算虚拟对象中的 skyline
8  cS←getskylinecell(·);
9  if |cS|<k then
10     if |OBJ(cS)|<k then
11         for i from 1 to |cS| do
12             ARS0←selectAll(cS[i]);
13         ARS0←ARS0+randomSelect(cSD);
14     if |OBJ(cS)|≥k then
15         ARS0←randomSelectOne(cS);
16 if |cS|≥k then
17     for i from 1 to |cS| do
18         TRS0←randomSelect(cS[i]);
19     ARS0←k-LDS(TRS0);
20 RS0←randomSelect(cS);
21 return RS0;
```

给定滑动窗口 W_0 中的轮廓对象集合 SK_0, 如果$|SK_0|\leqslant k$, 算法将 SK_0 中对象作为查询结果返回(第 1、2 行); 否则, 算法根据 SK_0 中对象坐标将它们映射到ρ-GRID G 对应的单元格(第 3、4 行). 对于所有非空单元格$\{c_1,c_2,\ldots,c_g\}$, 算法根据它们的左下角坐标构建一组虚拟对象$\{v_1,v_2,\ldots v_g\}$, 利用这组虚拟对象构建 R-Tree VGT. 随后, 算法根据构建结果, 从 VGT 中筛选虚拟轮廓对象集合 VSK_0, 进而找到 G 中的轮廓单元格集合 c_S (第 5– 8 行).

如果$|c_S|<k$且这些单元格包含的对象之和小于 k, 算法首先将 c_S 中对象全部输出至查询结果集 ARS_0; 随后, 算法从半支配单元格集合 c_{SD} 中任意选择 $k-|ARS_0|$个对象加入 ARS_0(第 10–13 行). 如果$|c_S|<k$ 且这些单元格包含的对象数目之和不小于 k, 算法首先在每个轮廓单元格中任选一个对象加入 ARS_0; 随后, 算法从 c_S 包含的剩余对象中任意选择$|c_S|-k$ 个轮廓对象加入 ARS_0(第 1、15 行). 如果轮廓单元格个数满足$|c_S|\geqslant k$, 算法从各轮廓单元格中任意选择一个轮廓对象$\{o_1,o_2,\ldots,o_{|c_S|}\}$加入临时构建的集合 TRS_0; 随后, 利用 k-LDS[5]查找 TRS_0 中的 k 代表轮廓对象(第 16–19 行). 在分析轮廓单元格的查询代价时, 由于计算代价只与轮廓单元格和半支配单

元格数量之和有关系, 与数据规模无关, 因此这部分的计算代价为 O(1).

例 2: 如图 6 所示, $\{o_1,o_4,o_5,o_9,o_{10},o_{12}\}$构成了轮廓对象集合 SK. 算法首先将对象(根据其坐标)映射到各自所在单元格. 如前文所述, 这些单元格一定是轮廓单元格或半支配单元格. 随后, 算法根据这些单元格的左下角坐标构建一组虚拟对象$\{v_1,v_2,v_3,v_4,v_5\}$, 计算这组虚拟对象中的轮廓对象$\{v_1,v_3,v_4,v_5\}$, 进而找到轮廓单元格. 因为轮廓单元格的数目大于 2(4>2), 所以算法从$\{c_{15},c_{11},c_7,c_3\}$中任意选择 4 个对象$\{o_5,o_1,o_9,o_{10}\}$执行 PBA 算法, 计算这组对象中的 2 代表轮廓. 它们是$\{o_1,o_{10}\}$, 它们可以作为近似结果支持查询.

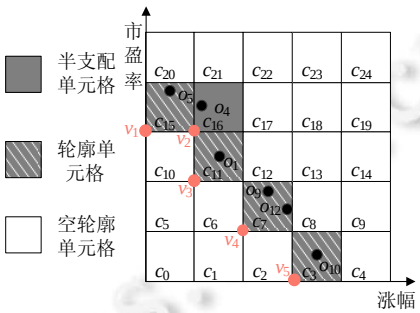


图 6 近似 k 代表轮廓初始化算法运行实例

- 复杂度分析

本文首先分析 k 代表轮廓初始化算法的时间复杂度. 算法 1 首先将当前窗口中的轮廓对象插入ρ-GRID, 这部分代价为 $O(|SK_0|)$. 随后, 算法执行轮廓单元格查询操作. 假设划分粒度为 n, 轮廓单元格和半支配单元格数量之和的上限为 $O(n^{d-1})$, 从这些单元格中找到轮廓单元格的计算代价为 $O(n^{d-1}\log_B dn^{d-1})$. 由于 n 是一个与ρ相关的常数, 与数据规模无关, 这一部分的代价可视为 O(1). 该分析方法与 Tao 等人[31]的分析方法相似. 当轮廓单元格数量$|c_S|>k$ 时, 算法还需以轮廓单元格为基本单元计算 k 代表轮廓. 由于轮廓单元格个数为 $O(n^{d-1})$, 根据文献[5], 该过程计算代价在 2 维环境下为 $O((n^{d-1}-k)k)(=O(k))$, 在 d 维(d>2)环境下为 $O((n^{d-1}-L)2^Ld)(=O(2^Ld))$. 这里, L 是一个小于 k 的正整数. 综上所述, 近似 k 代表轮廓算法的时间复杂度在 2 维和 d 维(d>2)条件下分别为 $O(|SK_0|+k)$和 $O(|SK_0|+2^Ld)$.

接下来, 本节分析算法的空间复杂度. 算法 1 利用虚拟网格实现ρ-GRID G 的构建, 而不是构建真实网格. 具体如下: 给定 SK_0 中任意对象 o, 算法首先根据 G 预先设定的分辨率 n 和 o 的位置找到包含 o 的单元格 c, c 的边长等于 $1/n$. 随后, 算法计算 c 的 Z-地址 $z(c)$. 以此为基础, 算法将 $z(c)$当成 o 的键值, 将 o 映射到哈希表 H_G 中(H_G 的规模设置为 $2|SK_0|$). 这样一来, 算法可根据轮廓对象的规模构建哈希表. 由于 H_G 空间代价与数据规模呈线性关系, 基于ρ-GRID 近似 k 代表轮廓算法的空间代价为 $O(|SK_0|)$.

3.2.3 基于ρ-GRID 的增量近似 k 代表轮廓查询算法 INCRGRSS

当窗口从 W_{i-1} 滑动到 W_i 时, PAKRS 执行近似 k 代表轮廓增量查询. 为提高计算效率, 本节进一步提出一种增量算法 INCRGRSS (incremental RhoGrid-based representive skyline selection). 算法描述如下: 假设窗口从 W_{i-1} 滑动到 W_i 后, 当前窗口中的轮廓对象集合更新为 SK_i, 算法将 SK_{i-1} 中的过期对象和无效对象从 H_G 中删除; 随后, 依次将 SK_i–SK_{i-1} 中的对象映射到哈希桶 H_G. 接下来, 算法根据新增的非空单元格和因对象流出(或变为无效)而变为空的单元格更新为 R-Tree VGT 和轮廓单元格集合 c_S. 如果轮廓单元格未发生变化, 算法则利用上一轮的计算结果支持查询; 否则, INCRGRSS 利用算法 1 中的第 9–20 行执行 k 代表轮廓查询. 最后,当窗口滑动 m 次后, 算法还需计算窗口中最后一个滑片的 k 代表轮廓. 原因将在第 3.2.4 节说明. 最后, 本节引入均摊代价分析算法的性能. 它指的是算法 u 次操作所构成的序列的总代价除以 u. 其基本理念在于: 当昂贵的操作出现频率很低时, 它们的成本可能会均摊到所有的操作上. 如果人工均摊的花销仍然便宜的话,对于整个序列的操作可有一个更加严格的约束. 本质上, 均摊分析就是在最坏的场景下, 对于一连串操作给出一个更加严格约束的一种策略.

- 复杂度分析

接下来, 本节分析增量近似 k 代表轮廓查询算法的时间复杂度. 当窗口从 W_{i-1} 滑动到 W_i 时, 算法首先将 SK_i–SK_{i-1} 中的对象插入ρ-GRID. 对于任意被插入ρ-GRID 的对象 o, 当其被晚于其流入的对象支配时, 它变为无效对象, 并且在流出窗口前均无法成为轮廓对象. 因此, 每个对象在其生命周期内至多被插入一次. 因为网格的插入代价为 O(1), 这部分均摊代价为 O(1). 当执行基于虚拟对象的轮廓查询时, 因为网格划分粒度 n 只跟ρ相关而与数据规模无关, 所以 $n^{d-1}\log_B dn^{d-1}$ 可以看成是一个常数. 因此, 这部分的计算代价为 $O(n^{d-1}\log_B dn^{d-1})=O(1)$. 在执行近似 k 代表轮廓查询时, 2 维和 $d(d>2)$维环境下的总体代价分别为 $O((n^{d-1}-k)k)$ $(=O(k))$和 $O((n^{d-1}-L)2^L d)$, 2 维和 d 维环境下, 每个对象的均摊代价分别为 $O(k/s)$和 $O(2^L d/s)$. 最后, 当窗口滑动 m 次后, 算法还需计算窗口中最后一个滑片的 k 代表轮廓, 这一部分的均摊代价在 2 维和 $d(d>2)$维环境下分别为 $O(|SK_m|k/sm)=O(k/m)$和 $O((|SK_m|-L)2^L d/sm)=O(2^L d/m)$. 因此, 算法的增量维护代价在 2 维和 $d(d>2)$维环境下分别为 $O(k/m+k/s)$和 $O(2^L d/s+2^L d/m)$.

- 讨论

本节需要强调如下 4 点.

(1) PAKRS 针对 W_0 执行初始化算法, 针对其他窗口均执行增量维护算法.

(2) ρ-GRID 可有效帮助 PAKRS 降低执行 k 代表轮廓查询的频率. 如上所述, 当窗口滑动后, 如果轮廓单元格集合未发生变化, 则无须执行近似 k 代表轮廓查询, 这可进一步降低 PAKRS 的计算代价.

(3) 算法的增量维护代价对数据规模不敏感. 如前文所述, 算法基于轮廓单元格展开计算, 这可有效避免轮廓对象规模对算法性能造成的冲击.

(4) 本节使用虚拟网格实现ρ-GRID 的维护, 这使得基于ρ-GRID 算法的空间代价只与数据规模相关, 与数据维度无关, 克服了数据维度对空间代价带来的冲击.

3.2.4 ρ-GRID 划分粒度的设置

接下来, 本节介绍划分粒度 n 的选取和算法的正确性证明. n 的选择取决于近似比ρ、维度 d. 当 $d>2$ 时, 粒度 n 的选取还与文献[5]中的近似比ξ有关. 为方便描述, $|c_{FD}|$, $|c_{SD}|$和$|c_S|$分别表示全支配单元格、半支配单元格和轮廓单元格的数量, GS 表示所有轮廓单元格中任取一个对象所构成的集合. $|DS_i|$表示半支配单元格 c_i 被 GS 支配的区域面积, $|INDS(GS)|$表示 GS 中对象在各自轮廓单元格内支配区域的面积之和.

引理 1. 给定轮廓单元格中任取一个对象所组成的集合 GS, 其组合支配面积满足不等式 $|DS(GS)|>\frac{|c_{FD}|}{n^d}$.

证明: 给定轮廓单元格 c 中任意对象 o, 其支配区域 $DS(o)$包括所有被 c 支配的全支配单元格所占区域、半支配单元格集合中被 o 支配的区域以及 c 中被 o 支配的区域. 给定 GS, 如等式(2)所示, 其组合支配面积等于各对象支配区域并集的面积:

$$|DS(GS)|=\frac{|c_{FD}|}{n^d}+\sum|DS_i|+|INDS(GS)| \tag{2}$$

由于$|DS_i|$和$|INDS(GS)|$均不小于 0, $|DS(GS)|$一定大于 $\frac{|c_{FD}|}{n^d}$. 引理 1 证毕. □

引理 2. 给定精确 k 代表轮廓集合 RS, 其组合支配面积满足不等式(3).

$$|DS(RS)|<\frac{|c_{FD}|}{n^d}+\frac{|c_{SD}|}{n^d}+\frac{|c_S|}{n^d}-\Delta S \tag{3}$$

这里, $|c_{SD}|$指半支配单元格的数量, ΔS 指 RS 与 SK 组合支配面积之差.

证明: 给定单元格 c 中任意对象 o, 其支配区域 $DS(o)$包括所有被 c 支配的全支配单元格所占区域、半支配单元格集合中被 o 支配的区域以及 c 中被 o 支配的区域. 因此, 给定 $SK=\{o_1,\ldots,o_s\}$, 其组合支配面积可以表示为各对象支配区域并集的面积. SK 的组合支配面积满足等式(4).

$$|DS(SK)|=\frac{|c_{FD}|}{n^d}+\sum|DS_i|+|INDS(SK)| \tag{4}$$

由于 $\sum|DS_i|<\frac{|c_{SD}|}{n^2}$ 且 $|INDS(SK)|<\frac{|c_S|}{n^2}$, $|DS(SK)|$满足不等式(5).

$$|DS(SK)|<\frac{|c_{FD}|}{n^d}+\frac{|c_{SD}|}{n^d}+\frac{|c_S|}{n^d} \tag{5}$$

因此, 当$|DS(SK)|$与$|DS(RS)|$的差值等于ΔS 时, $|DS(RS)|$满足不等式(6).

$$|DS(RS)|<\frac{|c_{FD}|}{n^d}+\frac{|c_{SD}|}{n^d}+\frac{|c_S|}{n^d}-\Delta S \tag{6}$$

其中, $|INDS(SK)|$表示集合 SK 中的对象在其各自单元格支配区域并集的面积, $|DS(RS)|$表示 k 代表轮廓对象的组合支配面积. 引理 2 证毕. □

本节根据引理 1、引理 2 推出引理 3. 它证明了精确 k 代表轮廓集合的组合支配面积$|DS(RS)|$和$|DS(GS)|$满足不等式(7).

引理 3. 精确 k 代表轮廓集合的组合支配面积$|DS(RS)|$和$|DS(GS)|$满足不等式(7).

$$||DS(RS)|-|DS(GS)||<\frac{d}{n} \tag{7}$$

证明: 首先, 根据引理 1 和引理 2, 当划分粒度为 n 时, 精确 k 代表轮廓集合 RS 与 GS 之间的组合支配面积之差满足不等式(8).

$$||DS(RS)|-|DS(GS)||<\frac{|c_{SD}|+|c_S|}{n^d}+\Delta S \tag{8}$$

因此, 不等式$||DS(RS)|-|DS(GS)||<\frac{|c_{SD}|+|c_S|}{n^d}+\Delta S$ 成立. 给定ρ-GRID G 中的单元格 c_1, 如果 c_1 中存在轮廓对象, 则其左下方及右上方单元格一定不存在轮廓对象. 如图 7 所示, 在 2 维情况下, G 中任意一条对角线 l_i 之间最多只存在一个单元格包含轮廓对象. 这里, i 的最大值等于 $2n-1$. 而轮廓对象只会存在于轮廓单元格和半支配单元格中, 因此有: $|c_{SD}|+|c_S|\leqslant 2n-1$.

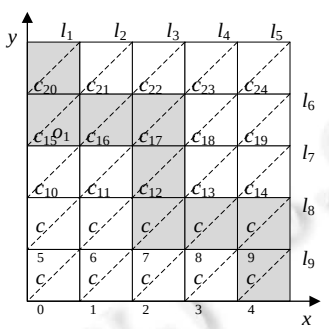


图 7 ρ-GRID G 中的各对角线

综上所述, 在 2 维条件下有: $||DS(RS)|-|DS(GS)||<\frac{2n-1}{n^2}<\frac{2}{n}$; 当扩展到 $d(d>2)$维时, 轮廓单元格和半支配单元格的最大值等于$\delta dn^{d-1}(0<\delta<1)$. 因此有: $||DS(RS)|-|DS(GS)||<\frac{\delta dn^{d-1}}{n^d}<\frac{d}{n}$. 引理 3 证毕. □

以引理 3 为基础, 定理 1 可证明: 精确 k 代表轮廓集合的组合支配面积$|DS(RS)|$和近似 k 代表轮廓集合的组合支配面积$|DS(ARS)|$在 d 维条件下满足$||DS(RS)|-|DS(ARS)||<\frac{d}{n}$. 该定理表明, 近似 k 代表轮廓算法可返回带误差保证的查询结果.

定理 1. 精确 k 代表轮廓集合的组合支配面积$|DS(RS)|$和近似 k 代表轮廓集合的组合支配面积$|DS(ARS)|$在 d 维条件下满足$||DS(RS)|-|DS(ARS)||<\frac{d}{n}$.

证明: 证明分为两种情况: (i) $|c_S|\leqslant k$; (ii) $|c_S|>k$. 当$|c_S|\leqslant k$时, 由于每个轮廓单元格中均有对象被存入近似 k 代表轮廓集合 ARS, 因此所有全支配单元格均包含在 ARS 的支配区域中. 此时, ARS 等价于 GS, 即有: $|DS(ARS)|>\frac{|c_{FD}|}{n^d}$, 结合引理 3 可以推出: $|DS(RS)|-|DS(ARS)|<\frac{d}{n}$. 当轮廓单元格数量满足$|c_S|>k$时, 给定 $DS(RS)$, 它满足不等式$|DS''(RS)|\leqslant|DS(RS)|\leqslant|DS'(RS)|$. 其中, $|DS'(RS)|$指所有轮廓对象都位于轮廓单元格左下角坐标时 k 代表轮廓的支配面积, $|DS''(RS)|$指所有轮廓对象都位于轮廓单元格右上角坐标时 k 代表轮廓的支配面积. 由于$|DS'(RS)|-|DS''(RS)|$同样小于$\delta dn^{d-1}(0<\delta<1)$, 且$|DS'(RS)|>|DS(RS)|$, $|DS''(RS)|<|DS(ARS)|$, 当执行近似 k 代表轮廓查询时, 算法误差满足不等式$|DS(RS)|-|DS(ARS)|<\frac{d}{n}$. 因此, 精确 k 代表轮廓集合的组合支配面积$|DS(RS)|$和近似 k 代表轮廓集合的组合支配面积$|DS(ARS)|$在 d 维条件下满足: $|DS(RS)|-|DS(ARS)|<\frac{d}{n}$. 其中, n 可由公式(9)计算得到.

$$n=\begin{cases}\left\lceil\dfrac{\rho d}{(\rho-\xi)|DS(RS)|}\right\rceil, & |c_S|>k,\ d>2\\ \left\lceil\dfrac{\rho d}{(\rho-1)|DS(RS)|}\right\rceil, & \text{其他}\end{cases}\tag{9}$$

定理 1 证毕. □

当 d>2 时, 本文所用ε-GA 算法是一个近似比为ξ的近似算法(ξ是一个与ε有关的常数, 等于$\frac{1}{1+\varepsilon}\left(1-\frac{1}{\sqrt{\mathrm{e}}}\right)$ $(0<\varepsilon<1)$. 因此, ξ的取值范围为(0.2,0.4), ε值的设定详见文献[5]). 假设基于轮廓单元格得到的精确解所对应支配面积为$|DS(ARS)|$, 那么基于ε-GA 得到的近似结果对应的支配面积$|DS'(ARS)|$满足$|DS'(ARS)|\geqslant\xi|DS(ARS)|$ $(0<\xi<1)$, 而$|DS(ARS)|$和$|DS(RS)|$之间需满足$\rho|DS(ARS)|\geqslant|DS(RS)|$, 所以有$|DS(RS)|\leqslant\rho/\xi|DS'(ARS)|$. 根据定理 1,给定近似比$\rho$、维度 d 以及阈值ξ, ρ-GRID G 对应的划分粒度可由公式(9)求得. 它保证精确 k 代表轮廓集合与近似 k 代表轮廓集合的组合支配面积的比值不会超过ρ.

然而, $|DS(RS)|$表示精确 k 代表轮廓对象的组合支配面积, 该值的变化是随窗口的滑动动态改变的, 这导致算法无法有效地设置ρ-GRID 的划分粒度. 本节利用高速流的特性解决这一问题. 鉴于窗口中的数据是批量流入窗口的, 给定窗口 $W_0=\{s_0,s_1,\ldots,s_{m-1}\}$, 本节在初始化过程中用 S^*作为$|DS(RS)|$的下界. 在 2 维环境下, 它是根据 s_m 中精确 k 代表轮廓所对应的组合支配面积计算得到的; 在 d(d>2)维环境下, 它是根据ε-GA 算法计算得到的. 由于计算结果小于真实的支配面积, 它能够表示 S^*的下界. 因为ρ-GRID 的划分粒度随 S 的降低而降低, 当用 S^*作为划分依据时, 该划分仍能保证计算结果满足误差约束条件. 因此, 最终的划分粒度为公式(10).

$$n=\begin{cases}\left\lceil\dfrac{\rho d}{(\rho-\xi)S^*}\right\rceil, & |c_S|>k,\ d>2\\ \left\lceil\dfrac{\rho d}{(\rho-1)S^*}\right\rceil, & \text{其他}\end{cases}\tag{10}$$

需要指出的是, 随着窗口的滑动, 当窗口滑动至 $W_{m-1}=\{s_{m-1},s_m,\ldots,s_{2m-2}\}$时, 算法根据 s_{2m-2} 中的 k 代表轮廓重新设置ρ-GRID 的划分粒度. 换句话说, 每当窗口滑动 m 次后, 算法都需要根据窗口中最后一个滑片中的 k 代表轮廓更新 S^*.

参数ρ的设置取决于用户需求. 随着近似比ρ的增加, 算法的准确性会有所降低, 但效率会随之提升; 反之, 随着近似比ρ的减小, 算法的准确性会随之提高, 但效率会降低. 因此, ρ的取值是一个权衡值, 它是根据用户对查询效率和查询精度的偏好综合得到的. 本文将在实验部分对不同数据分布下的近似比设置进行分析. 分析可知, 大部分情况下, 由ρ产生的计算代价是可以接受的. 例如, 当ρ=1.01, S^*=0.5, ξ=0.4, d=3 时, n 等于 10. 此时, 算法可通过不多于 100 个轮廓单元格计算近似 k 代表轮廓, 其计算代价远小于基于轮廓对象计算 k 代表轮廓.

3.3 基于GBRT的预测结果集初始化算法

- 索引 GBRT

为了支持PRSSET在流数据环境下的高效维护, 本节提出了索引GBRT (group bucket R-tree). 它是由一组R-Tree和一组桶构成的索引. R-Tree T_0维护PRSSET $\boldsymbol{P}$中的对象. 为了准确预测新增对象成为查询结果的最早时间, T_0中的对象在叶子节点内按到达顺序呈倒序排列. 这样一来, 当新到达的对象 o 访问 T_0时, 算法可从支配 o 的对象中快速找到最晚到达的一个, 根据该对象预测 o 成为轮廓对象的最早时间. 此外, GBRT 额外利用R-Tree $\{T_1,T_2,\ldots,T_t\}$管理分片$\{P_1,P_2,\ldots,P_t\}$中的预测轮廓对象. T_1 维护 SK_1–ARS_1 中的对象, T_2 维护 $SK_2\cup SK_3$–ARS_2 中的对象, T_3 维护$\bigcup_{i=4}^{i=7} SK_i - ARS_4$中的对象, T_t 维护$\bigcup_{i=m/2}^{i=m-1} SK_i - ARS_{m/2}$中的对象. 最后, 为支持增量维护, GBRT 利用一组桶$\{B_1,B_2,\ldots,B_t\}$维护新流入窗口且暂时无法被删除的数据. SK_i 和 ARS_i 分别表示未来窗口 W_i 的预测轮廓对象集合和预测近似 k 代表对象集合.

- 初始化算法 PRSI

如算法 2 所示, 给定当前窗口 W_0, 算法首先构造一组只包含根节点的 R-Tree $\{T_1,T_2,\ldots,T_t\}$. 随后, 算法利用轮廓查询经典算法 BBS[22]计算 s_{m-1} 中局部轮廓集合 LSK_{m-1} (第 3 行), 构建 R-Tree T 维护这部分对象. T 的作用是维护已扫描对象中的轮廓集合 TSK. 由于 s_{m-1}–SK_{m-1} 中的对象与 SK_{m-1} 中的对象同一批次流入窗口, s_{m-1}–SK_{m-1}中的对象无法在其生命周期内成为查询结果, 算法将其删除. 随后, 算法利用 BBS 计算 s_{m-2}中的局部轮廓集 LSK_{m-2}, 并进一步访问 T(第 4–10 行). 具体如下.

① $\forall o\in LSK_{m-2}$, 如果它不被 T 中的对象支配, 它被视为未来窗口 W_{m-2} 的预测轮廓对象, 因而被插入 T.

② $\forall o'\in LSK_{m-2}$, 如果它被 T 中的对象支配, 它无法在其生命周期内成为查询结果, 算法将其从窗口中删除.

③ $\forall o''\in T$, 如果它被 LSK_{m-2} 中的对象支配, 它最早在未来窗口 W_{m-1} 成为查询结果, 算法将其从 T 转移到 R-Tree T_t.

算法 2. The PRSI Algorithm.

输入: Stream Data S.

输出: R-Tree Set $\boldsymbol{T}$, Bucket Set $\boldsymbol{B}$.

```
1  Int u←log2m+1;
2  for i from m–1 to 2 do
3      SKi←BBS(si);
4      for j from 1 to |SKi| do
5          Dom(SKi[j])←Dominate(T);          //找到被 SKi[j]支配的对象
6          DomED(SKi[j])←Dominated(T);       //找到 T 中能够支配 SKi[j]的对象
7          if DomED(SKi[j])=∅ then
8              insert(T,SKi[j]);
9          if Dom(SKi[j])≠∅ then
10             remove(T,Tu,DomED(SKi[j]));
11     if log2i 是一个整数 then
12         SKi←T;
13         ARSi←appRSKY(SKi);
14         u←u–1;
15 T0←construction(ARSt,ARSt–1,…,ARS1,ARS0);
16 B←init(∅), update(T1,T2,…,Tt,ARSt,ARSt–1,…,ARS1,ARS0,T1,T2,…,Tt);
17 return K;
```

例 3: 如图 8 所示, T_0 的作用是维护 PRSSET $\boldsymbol{P}$, T_1–T_5 维护各未来窗口中的潜在轮廓对象, B_1–B_5 为各未来窗口所对应的桶, 它们在初始化时为空. 给定初始窗口 W_0={s_0,s_1,…,s_{31}}, 算法首先将窗口划分为分片{P_0, P_1,…,P_5}, 随后逆序扫描 P_5, 使用 BBS 算法计算 s_{31} 中的局部轮廓集 LSK_{31}, 构建 T 维护 LSK_{31} 中的对象. 接下来, 算法利用 BBS 计算 LSK_{30}. $\forall o\in LSK_{30}$, 如果 o 不被 T 中的对象支配, 算法将其插入 T; 如果它被 T 中的对象支配, 算法将其删除. 如果 T 中的对象被其支配, 则将这部分对象转移至 R-Tree T_5. 当扫描 P_5 后, 算法将 T 中的对象复制到 SK_{16}. 此后, 算法重复该过程, 依次逆序扫描 P_4–P_0, 构建{SK_8,…,SK_0}. 最后, 利用第 3.2 节中的算法计算{ARS_{16},…,ARS_0}, 利用它们构建 $\boldsymbol{P}$ 和 T_0, 利用 $\bigcup_{i=16}^{i=31} SK_i - ARS_{16}$, $\bigcup_{i=8}^{i=15} SK_i - ARS_8$, $\bigcup_{i=4}^{i=7} SK_i - ARS_4$, $SK_2\cup SK_3$–ARS_2, SK_1–ARS_1 更新 T_5–T_1. 最后, 算法初始化 B_1–B_5.

此后, 算法重复上述过程扫描其他滑片(第 4–14 行), 计算未来窗口{$W_{m/2}$,$W_{m/4}$,…,W_0}的预测轮廓集合{$SK_{m/2}$,$SK_{m/4}$,…,SK_0}, 构建 R-Tree {T_1,T_2,…,T_t}, 执行第 3.2 节中的算法构建近似 k 代表轮廓集合{$ARS_{m/2}$, $ARS_{m/4}$,…,ARS_0}. 扫描窗口后, 算法将 R-Tree {T_1,T_2,…,T_t}中的对象更新为 SK_1–ARS_1, $SK_2\cup SK_3$–ARS_2,…, $\bigcup_{i=m/2}^{i=m-1} SK_i - ARS_{m/2}$. 最后, 算法构建空桶{$B_1$,$B_2$,…,$B_t$}(第 15、16 行).

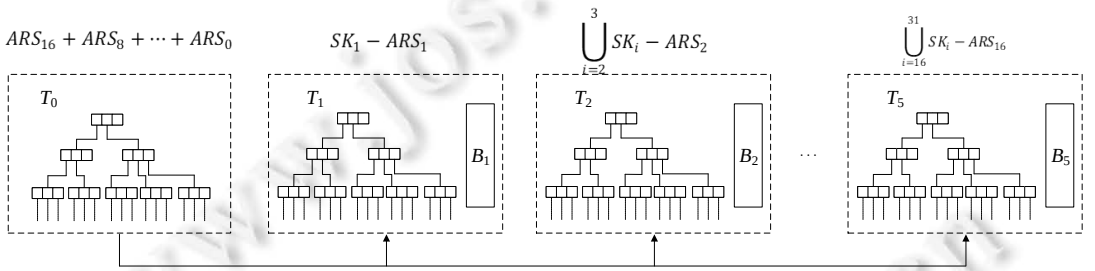


图 8 组合索引 GBRT (m=32)

- 讨论

本节所提算法优点如下.

(1) 高效删除当前窗口中无效对象. 首先, 算法可利用高速流的特性批量删除无效对象. 具体如下: 算法利用 BBS 批量删除同一批次流入窗口的无效对象. 由于这部分对象不被插入 T, 批处理算法可有效降低 T 的更新次数.

(2) 预测对象成为轮廓对象的最早时间. 给定当前扫描对象 o 和 TSK, 如果 o 支配 TSK 中对象 o', 则 o' 最早在 o 流出窗口时成为窗口中的轮廓对象. TSK 表示当前扫描区域内不被任何对象支配的对象.

(3) 刻画对象间支配关系的传递性. 给定对象{o_1,o_2,o_3}, 如果 o_1 支配 o_2, o_2 支配 o_3, 那么 o_1 一定支配 o_3. 假设 o_2 是所有支配 o_3 对象中最晚到达的, 算法在逆序扫描过程中能够保持对象间支配关系的传递性. 即算法在发现 o_1 支配 o_2 时, 能够推断出 o_1 支配 o_3. 为了维护对象间支配关系的传递性, 算法在初始化过程中, 为每一个对象 o 添加一个标记 $o.sky$, 它记录能够支配 o 的对象中最晚到达的对象. 这为算法在流环境下高效删除无效对象奠定了基础.

(4) 本节所提分片算法只是在逻辑上对窗口进行划分, 无须在各分片中保留窗口中的所有对象. 在对各分片进行维护时, 算法只需维护分片对应对象的最大/最小序号即可.

- 复杂度分析

接下来, 本节分析 PRSI 算法的计算复杂度. 首先, 算法需要扫描窗口进而找到所有轮廓对象. 对于同一分片内的对象, 计算代价为 O($s\log_2 ds$), 这部分总体代价为 O($N\log_2 ds$). 随后, 算法根据局部轮廓对象访问 T, 每个对象的计算代价为 O($\log_B d|TSK|+d|Dom(o)|$). 其中, $|Dom(o)|$为 TSK 中被 o 支配的对象个数, $\log_B d|TSK|$为访问 R-Tree 的均摊代价. 给定数据集 D 中的对象{o_1,o_2,…,o_N}, 有 $N\geqslant\sum_{i=1}^{N}|Dom(o_i)|$, 每个对象的平均计算代价

为 $O(\log_B d|TSK|+dN/N)=O(\log_B d|TSK|+\log_2 ds+d)$. $|TSK|$为逆序扫描过程中得到的临时轮廓对象集合规模. 另外, 由于初始化算法只需要针对窗口$\{W_0,W_2,W_4,\ldots,W_{m/2}\}$计算近似 k 代表轮廓(计算 ARS_0 时使用初始化算法, 针对其他窗口使用增量维护算法), 这使得每个对象至多被插入ρ-GRID 一次. 结合第 3.2.3 节中的结论, PAKRS 初始化算法时间复杂度在 2 维和 d 维(d>2)环境下的计算代价分别 $O(N(\log_B 2|TSK|+\log_2 2s)+\log_2 m(k/m+k/s))$和$O((2^L d/s+2^L d/m)\log_2 m+N(\log_B d|TSK|+\log_2 ds+d))$. B 表示 R-Tree 中结点的最大容量.

3.4 预测结果集增量维护算法INCPRSS

当一组新对象 s_m 到达窗口时, 算法对 GBRT 进行增量维护. 本节根据定理 2 预测新到达对象成为轮廓对象的最早时间.

定理 2. 给定对象 o 及未来窗口 W_i 和 W_j 的预测轮廓对象集合 SK_i 和 $SK_j(j>i)$, 如果 o 不被 SK_i 中的对象支配, 那么 o 也不被 SK_j 中的对象支配.

证明: 对于 SK_j 中的对象 o', 如果它不被 SK_i 中的对象支配, 那么它一定包含在 SK_i 中. 因此, SK_i 中不存在对象被 SK_j 中的对象支配. 综上: 如果 o 不被 SK_i 中的对象支配, 那么 o 也不被 SK_j 中的对象支配. 定理 2 证毕.

接下来, 本节正式介绍增量维护算法 INCPRSS. 算法首先针对 s_m 中的对象执行 BBS 算法得到 SK_m. 随后, $\forall o\in SK_m$, 算法访问 T, 根据 o 和 T 中的对象间的支配关系更新 GBRT. 具体如下.

① 存在 T_0 中的对象 o'被 o 支配: 这说明 o'变为无效对象, 算法将 o'删除. 此外, 由于算法维护了对象间支配关系的传递性, 当 o'被删除后, 窗口中已知被 o'支配的对象均可被删除. 最后, 假设 o'是所有被 o 支配的对象中成为预测结果最早的对象($o'\in ARS_i$), 算法将 o 插入 ARS_i 和 T_0.

② 假设 o''是 T_0 中所有支配 o 的对象中流入窗口最晚的对象, 算法根据 o''预测 o 成为轮廓对象的最早时间 $PT(o)$(该时间等于 o''流出窗口的时间). 如果 $T(P_i)\leqslant PT(o)\leqslant T(p_{i+1})$, 那么 o 被插入桶 B_i. $T(P_i)$表示 P_i 对象中流出窗口的最早时间. 特殊情况下, 如果没有任何对象支配 o, 算法将 o 放入 B_1. B_i 中的对象按照其到达窗口的顺序呈倒序排列. 和维护预测轮廓结果集 SK_i 中对象间的支配关系相比, 本节将 SK_i–ARS_i 中的对象放入桶 B_i, 只维护这部分对象间的时序关系. 这样一来, 算法不仅可以保证所有"潜在"的轮廓对象均被保留, 而且可以有效地降低维护对象间支配关系所引发高昂的计算代价. 更重要的是, 由于 B_i 内按其流入窗口的顺序呈降序排列, 如下文所述, 该性质便于 PAKRS 周期性地逆序扫描 B_i 中的对象, 更新预测结果集, 删除无效对象, 降低空间代价.

随着窗口的滑动, 如果窗口中第 1 个分片包含了多个滑片中的对象, 算法需重新划分该分片, 并更新 PRSSET $\boldsymbol{P}$. 假设 P_i, $\{s_u,s_{u+1},\ldots,s_{u+v-1}\}$需要被重新划分, 算法需将 P_i 分裂为 $\log_2 v+1$ 个新分片. 这些分片分别包含了$\{s_u\},\{s_{u+1}\},\{s_{u+2},s_{u+3}\},\ldots,\{s_{u+0.5v},s_{u+0.5v+1},\ldots,s_{u+v-1}\}$中的对象. 例如, 当 P_0, P_1, P_2, P_3 中的对象流出窗口后, 算法需将 $P_4\{s_8,s_9,\ldots,s_{15}\}$分裂为 4 个新分片. 这些新分片分别包含了$\{s_8\},\{s_9\},\{s_{10},s_{11}\},\{s_{12},\ldots,s_{15}\}$中的对象. 随后, 算法针对划分结果重新构建一组 R-Tree.

本节提出了算法 MSA(merge spilt algorithm)实现上述目的. 该算法包含两个阶段: 分裂阶段与合并阶段. 合并阶段又可分为局部合并和全局合并. 在局部合并阶段, 算法利用初始化算法访问 P_i 对应的桶 B_i, 计算 B_i 中的任意对象 o 成为轮廓对象的最早时间 $PT(o)$, 并根据预测结果将其转移至其他桶. 具体如下: 算法首先逆序扫描 B_i 中的对象. 在扫描过程中, $\forall o, o\in B_i(T(o)<T(o'))$, 如果 o'支配 o, 那么 o 可被直接删除. 如果 o 支配 o', 则根据 $T(o)$预测 o'成为查询结果的时间 $PT(o)$. 假设 $T(P_i)$表示 P_i 中第 1 个流入窗口对象的序号, 如果 $T(P_i)<PT(o)<T(P_{i+1})$, 算法将 o 保留在 B_i; 如果 $T(P_j)<PT(o)<T(P_{j+1})(j>i)$, 算法将 o 插入 B_j. 本节假设 o 是支配 o'的对象中流入窗口最晚的对象. 扫描 B_i 后, 算法进入全局合并阶段. 此时, 算法将 B_i 剩余对象依次插入 R-Tree T_i, 在插入过程中, 删除 T_i 中的无效对象. 最后, 算法进入分裂阶段. 假设 P_i 包含了$\{s_u,s_{u+1},\ldots,s_v\}$中的对象, 算法访问 T_i, 构造未来窗口$\{W_u,W_{u+1},\ldots,W_{u+0.25v},W_{u+0.5v}\}$的预测轮廓集$\{SK_u,SK_{u+1},\ldots,SK_{u+0.25v},SK_{u+0.5v}\}$, 利用第 3.4 节中的算法计算$\{ARS_u,ARS_{u+1},\ldots,ARS_{u+0.25v},ARS_{u+0.5v}\}$, 更新 PRSSET $\boldsymbol{P}$,构建与$\{W_u,W_{u+1},\ldots,W_{u+0.25v},W_{u+0.5v}\}$对应的 R-Tree $\{T_i^1,T_i^2,...,T_i^{t'}\}$. 接下来, 本节根据例 4 解释 $\{T_i^1,T_i^2,...,T_i^{t'}\}$ 的构建过程.

例 4: 如图 9 所示, 当 P_3 中最后一个滑片 s_7 中的对象流出窗口后, 算法需对 P_4 执行分裂操作. 根据分裂

算法, 它被分裂成 $\{P_4^0,P_4^1,P_4^2,P_4^3\}$. 随后, 算法逆序扫描 B_4 中对象, 删除无效对象, 计算 B_4 剩余对象成为轮廓对象的最早时间. 根据预测结果, 算法将这些对象重新分配到不同分片所对应的桶. 随后, 算法将保留的对象依次插入 P_4 对应的 R-Tree T_4. 在插入过程中, 算法一方面删除无效对象, 另一方面更新对象成为轮廓对象的最早时间. 在此之后, 算法计算$\{ARS_8,ARS_{8+1},\ldots,ARS_{8+4}\}$, 并更新 R-Tree T_0. 最后, 算法构建 R-Tree $\{T_4^1,T_4^2,T_4^3,T_4^4\}$. 首先, 算法复制 T_4 中所有非叶子节点到 T_4^4, 并将未来窗口$\{W_{8+4},W_{8+4+1},\ldots,W_{8+8-1}\}$对应的预测轮廓集转移至 T_4^4. 此时, T_4^4 成为与新分片 P_4^4 对应的 R-Tree. 随后, 算法更新 T_4, 删除不包含任何对象的叶子节点/非叶子节点. 接下来, 算法重复上述步骤, 依次构建 T_4^3, T_4^2, T_4^1.

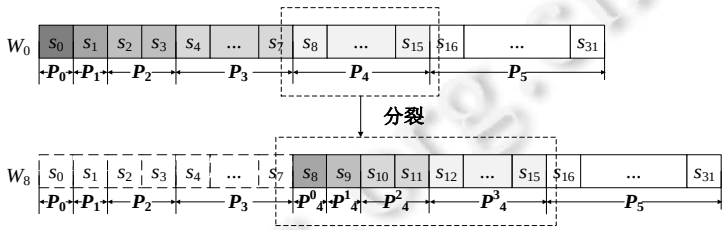


图 9 对分片的分裂操作

本节需要强调以下 3 点.

(1) 首先, 当对象流入窗口后, 算法至多为其执行 2 次 R-Tree 插入操作, 这部分代价为 $O(\log_2 d(|TSK|))$.

(2) 其次, 在对象流出窗口前, 它至多被分裂 $O(\log_2 m)$次, 每次的分裂代价为 O(1). 因此, 总体分裂代价为 $O(\log_2 m)$. 在分裂过程中, 算法并不对树的平衡性进行调整, 但 T_i 高度的上限为 $O(\log_2 N)$. 这种策略能够保证其具有较高的搜索效率.

(3) 最后, PAKRS 利用 3 种策略删除无效对象, 它们分别是: 在数据流入窗口时, 利用高速流的特性删除同一滑片中的无效对象; 利用对象间支配关系的传递性删除对象; 在分片分裂时, 调用初始化算法删除分片中的无效对象. 然而, PAKRS 只利用少数未来窗口中的预测轮廓对象实现无效对象的过滤和预测结果集的构建, 这导致各分片中部分无效对象无法被及时过滤, 引起了额外的空间代价. 为解决该问题, 算法在初始化时记录潜在轮廓对象的数目 π_{sk}. 随着窗口的滑动, 当算法维护潜在轮廓对象的数目达到 $2\pi_{sk}$ 时, 算法执行初始化操作, 更新各桶中的对象, 并重新赋值 π_{sk}.

- 复杂度分析

本节首先分析 PAKRS 框架的增量维护代价. 对于新到达的对象 o, 算法访问 R-Tree T, 找到能够支配 o 且到达时间最晚的对象. 由于 T 包含未来窗口$\{W_1,W_2,W_4,\ldots,W_{m/2}\}$中预测 k 代表轮廓对象, T 中的对象数量为 $O(k\log_2 m)$, 访问 T 的计算代价为 $O(\log_B d(k\log_2 m)+d|Dom(o)|)$. 其中, $|Dom(o)|$为被 o 支配的对象个数. $O(\log_B d(k\log_2 m))$是访问 R-Tree 的均摊代价. 当 N 个对象流入窗口后, 由于被删除对象的个数上限为 $O(N)$, 处理新流入窗口数据的均摊代价为 $O(\log_B d(k\log_2 m))$. 此外, 算法还需增量计算近似 k 代表轮廓集合. 根据第 3.2 节中的复杂度分析结果, 这部分代价在 2 维和 d 维(d>2)环境下为 $O(k/m+k/s)$和 $O(2^L d/s+2^L d/m)$. 最后, 算法还需在第 1 个分片滑出窗口时对后续分片重新划分操作. 如前文所述, 每个对象在流出窗口或变为无效对象前只会执行至多 2 次 R-Tree 插入操作, 这部分均摊代价为 $O(\log_B d|TSK|)$. 插入后, 每个对象至多被分裂 $O(\log_2 m)$次, 每次分裂代价为 O(1). 因此, 这部分代价为 $O(\log_2 m+\log_B d|TSK|)$. 综上, PAKRS 增量维护算法代价在 2 维和 d 维(d>2)环境下分别为 $O(\log_B 2(k\log_2 m)+\log_B 2|TSK|+\log_2 2s+k/m+k/s)$和 $O(2^L d/s+2^L d/m+\log_2 ds+\log_B d(k\log_2 m)+\log_B d|TSK|)$. B 表示 R-Tree 中结点的最大容量.

接下来, 本节分析 PAKRS 框架的空间代价. PASRS 的空间代价由两部分组成: GBRT 的空间代价以及 ρ-GRID 空间代价. GBRT 利用一组桶和 R-Tree 维护预测轮廓对象. 如算法 INCPRSS 所述, 算法不仅会在窗口滑动过程中利用对象间支配关系的传递性删除无效对象, 而且会在窗口初始化时记录潜在的轮廓对象数目 π_{sk}. 这样一来, 随着窗口的滑动, 当算法维护的潜在轮廓对象数目达到 $2\pi_{sk}$ 时, 算法执行初始化操作. 由于初

始化算法能够删除窗口中所有无效对象, GBRT 在最坏情况下保留的对象数目是|*C*|的 2 倍. 因此, GBRT 的空间代价为 O(|*C*|). 根据第 3.2 节的空间复杂度分析结果, ρ-GRID 空间代价为 O(|*SK*|). 因为|*SK*|<|*C*|, PAKRS 的空间代价为 O(|*C*|+|*SK*|)=O(|*C*|).

- 讨论

与 *k*-LDS 相比, PAKRS 具有以下优点:

(1) *k*-LDS 在计算轮廓集合时, 需要对候选轮廓集合 *C* 进行维护, 在最坏情况下, |*C*|=*N*, 此时, *k*-LDS 算法性能对数据规模较为敏感; 与之相反, PAKRS 至多利用 O($k\log_2 m$)个对象即可预测新流入对象成为轮廓对象的最早时间, 它对数据规模敏感性较低.

(2) *k*-LDS 在计算 *k* 代表轮廓集合时, 需要针对利用所有轮廓对象进行计算. 此时, *k*-LDS 算法性能对轮廓集合 *SK* 规模较为敏感, 在最坏情况下, *SK* 的规模与窗口内数据规模呈线性关系; 与之相反, PAKRS 可根据ρ选择常数个轮廓对象参与近似计算.

(3) *k*-LDS 在维护窗口中的 *k* 代表轮廓集合时, 没有考虑窗口流速 *s* 对查询效率的影响; 而 PAKRS 通过引入分片策略, 利用高速流的特性对当前窗口进行划分, 并以此为基础构建预测结果集, 因此, PAKRS 利用高速流特性提高了算法性能.

表 2 对比了 *k*-LDS 与 PAKRS 的时间复杂度. 其中, |*C*|和|*SK*|分别表示"潜在"轮廓对象规模和轮廓对象规模. 在最坏情况下, 它们的规模与窗口中数据规模呈线性关系. 与之不同的是, PAKRS 的代价对窗口中数据规模和轮廓对象规模均不敏感.

表 2 *k*-LDS 与 PAKRS 时间复杂度对比

维度	*k*-LDS	PAKRS
d=2	$O(\log_B 2\|C\|+(\|SK\|-k)k)$	$O(\log_B 2(k\log_2 m)+\log_2 2s+k/s+k/m)$
d>2	$O(\log_B d\|C\|+d+(\|SK\|-L)2^L d)$	$O(2^L d/m+2^L d/s+\log_2 ds+\log_B d(k\log_2 m))$

4 实验分析

4.1 实验准备

本节首先介绍了实验环境, 然后说明了数据集的获取, 接着讨论了参数的设置和实验方法. 算法采用 C++实现, 实验环境为 Intel(R) Xeon(R) Gold 6226R CPU@2.90 GHz, 2.89 GHz 的 16×2 核处理器; 1 TB 内存; 6 TB 硬盘的工作站, 操作系统为 Windows 10. 本文采用 5 组数据集验证算法的性能, 其中包括 3 个真实数据集 STOCK, TRIP, PLANET 和两组合成数据集.

第 1 个真实数据集 STOCK 为中国深沪两市 24 个月内约 2 300 只股票的交易记录. 实验数据规模为 1 GB 条. 每条记录表示为五元组⟨*stock ID*,*transaction time*,*volume*,*price*,*ratio*⟩, 分别表示股票编号、交易时间、成交量、交易金额和市盈率, 参与查询的属性为成交量、交易金额和市盈率. 为模拟流速, 算法对数据按照其交易时间进行排序, 根据 *s* 设置数据流入窗口的序号: 每 *s* 条数据赋为相同的流入序号. 第 2 个真实数据集 *TRIP* 为 72 个月美国纽约市出租车的出行记录, 数据规模为 1 GB 条, 每条记录表示为五元组⟨*taxi ID*,*pick-up time*, *drop-off time*,*travel distance*⟩, 分别表示出租车编号、上车时间、下车时间、出行距离, 参与查询的属性分别是乘车时间间隔(下车时间–上车时间)、出行距离. 为模拟流速, 算法对数据按照其上车时间进行排序, 根据 *s* 设置数据流入窗口的序号: 每 *s* 条数据赋为相同的流入序号. 第 3 个真实数据集为 MPCAT-OBS 卫星数据, 规模为 126 MB 条, 维度为 4 维. 每条数据包含一组观测坐标. 为模拟流速, 算法对数据按照其观测时间进行排序, 根据 *s* 设置数据流入窗口的序号: 每 *s* 条数据赋为相同的流入序号. 本实验所使用的真实数据集来自文献[32]. 合成数据集包括一组独立数据集和一组反相关数据集, 其中, 独立数据集中, 对象属性值之间服从独立分布; 反相关数据集中, 对象各属性之间呈反相关分布. 即若对象某维度性值较大, 则其他属性值会相对较小. 每个合成数据集包含 1 GB 条记录. 为模拟流速, 算法对数据按照其生成时间进行排序, 根据 *s* 设置数据流入窗口的序号: 每 *s* 条数据赋为相同的流入序号. 为测试数据维度对算法的性能的影响, 每组独立数据 集/

反相关数据集分别包含了 2–6 维数据. 数据集描述见表 3.

表 3 数据集描述

数据集	规模(条)	维度	说明
STOCK	1 GB	3	股票数据集
TRIP	1 GB	2	出租车出行数据集
PLANET	126 MB	4	卫星数据集
独立数据集	1 GB	2–6	数据间服从独立分布
反相关数据集	1 GB	2–6	数据间服从反相关分布

下面, 本节介绍参数的设置和默认值的选取. 实验测试 5 个指标, 分别是窗口长度 N、窗口流速 s、k 值、维度 d 以及阈值ρ. 如表 4 所示.

- 实验中, 窗口长度 N 从 5 MB 增加到 25 MB, 默认值为 10 MB.
- 由于本文研究面向高速流的近似 k 代表轮廓查询, 窗口流速 s 从窗口长度 N 的 5%变化到 20%, 即从 0.5 MB 条增加到 2 MB 条, 默认值为 10%, 即 1 MB 条. 在实际应用中, s 的规模能够模拟大部分的高速流. 以三维双精度类型数据为例, 单条数据占 24 字节. 当 s 取默认值 1 MB 时, 窗口滑动时加入窗口的数据(流出窗口的数据)占 24 MB 字节. 当 s 取 2 MB、数据维度为 6 时, 窗口滑动时加入窗口的数据(流出窗口的数据)占 96 MB 字节.
- k 从 5 变化到 25, 默认值为 10.
- 数据维度 d 从 2 变化到 6, 默认值为 3.
- 阈值ρ从 1.001 变化到 1.5, 默认值为 1.05.

表 4 参数设置

参数设置	默认值	变化范围
N	10 MB	[5 MB,25 MB]
s	10%×N	[5%×N,20%×N]
k	10	[5,25]
d	3	[2,6]
ρ	1.05	[1.001,1.5]

本文的实验方法是按照数据产生的顺序将数据一次性读入内存, 其好处是减少了 I/O 代价对算法评测产生的影响. 随后, 实验将前 N 条数据放入窗口执行初始化操作. 接下来, 实验在窗口滑动时执行增量维护和近似 k 代表轮廓查询. 实验过程中, 为了准确地反映算法的效率, 程序每处理 s 条数据后便启动对下一滑片中数据的处理. 当所有数据都处理完毕之后, 实验计算窗口每次滑动后的均摊处理时间. 在 2 维数据环境下, 本实验采用文献[5]中的 PBA 算法; 在 d 维(d>2)条件下, 与文献[5]中提出的 GA 算法和ε-GA 算法进行对比. 本实验中, PBA、GA 和ε-GA 均包含轮廓查询和 k 代表轮廓查询这两个步骤. 除此之外, 本节测试了参数ρ、数据维度 d 对算法性能的影响. 最后, 本节对比了不同算法之间的空间代价. 在这一环节, 由于本节提前将所有数据读入内存, 本节在展示的内存评测结果是总体消耗减去数据集中数据的内存消耗. 这样一来, 实验不仅可以模拟高速流, 而且可以准确地反映算法的计算代价和空间代价高低.

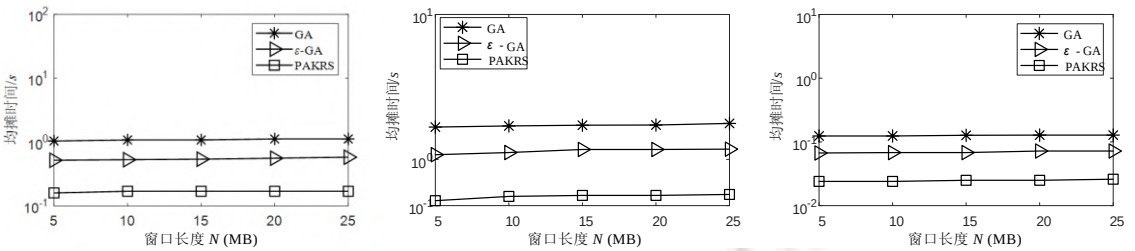
4.2 实验分析

第 1 组实验在真实数据集下测试窗口长度对算法性能的影响, 它从 N 从 5 MB 扩大到 25 MB, 其他参数使用默认参数, 实验记录处理每次滑动后的均摊时间. 如图 10 所示, 随着窗口长度的增加, 实验有以下发现.

(1) PAKRS 的性能最好, 它的运行时间约是ε-GA 算法的 0.25 倍、GA 算法的 0.17 倍.

(2) 随着窗口长度的增加, 3 种算法性能略均有下降, 而 PAKRS 的性能下降最缓慢. 原因在于, ε-GA 和 GA 在维护 k 代表轮廓的同时, 都需要利用索引维护所有潜在轮廓对象, 因而产生高昂的维护代价; 相反, PAKRS 可以通过维护少量未来窗口的预测结果集, 预测新流入对象成为查询结果的最早时间, 这有效地降低了计算代价. 此外, 随着窗口长度的增加, 轮廓对象规模也随之增加, 这导致了ε-GA 和 GA 算法的运行时间增加幅度大于 PAKRS; 与之相反, PAKRS 对窗口长度的变化并不敏感.

(3) 最后, PAKRS 利用高速流的特性在新流入窗口的数据中批量过滤无效数据, 这进一步提升了 PAKRS 的运行效率.



(a) STOCK (b) TRIP (c) PLANET

图 10 窗口长度 N 对算法效率的影响

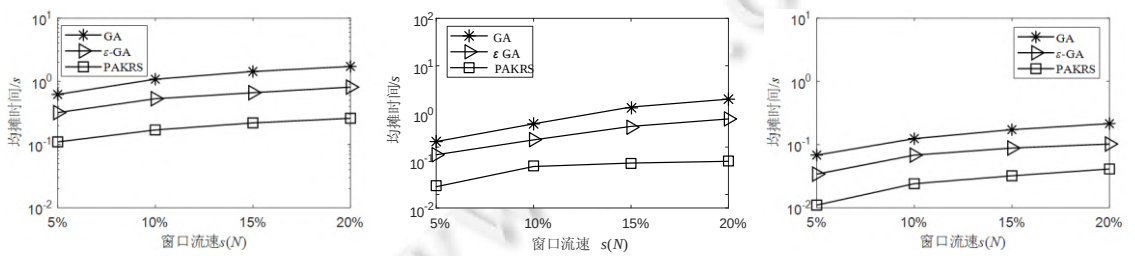
第 2 组实验在真实数据集下测试窗口流速 s 对算法性能的影响, 其他参数使用默认参数. 由于本文所提算法应用于高速流环境, 实验将流速 s 的最小值设置为窗口长度 N 的 5%, 并逐步增加到窗口长度的 20% (从 500 KB 条增加到 2 MB 条). 如图 11 所示, (1) PAKRS 算法的性能最好; (2) 随着窗口流速的增加, 3 种算法处理每个滑片的均摊时间均略有提高, 但升高幅度远小于 s 的增长速度. 这说明每个对象的均摊代价随 s 的增加而降低, 其原因如下.

(1) PAKRS 利用高速流的特性在新流入窗口的数据中批量过滤无效数据, 这会有效提升 PAKRS 的运行效率. 此外, s 越大, 被批量过滤的对象数目越多. 由此可见, PAKRS 充分利用高速流的特性降低轮廓对象的识别代价和候选轮廓对象的维护代价.

(2) 随着 s 的增加, PAKRS 需要维护的分片数量随之降低. 因此, PAKRS 针对预测结果集执行增量维护的总代价降低.

(3) 随着 s 的增加, 窗口中轮廓对象的改变频率会降低.

由于只有轮廓对象发生改变时才需要重新计算 k 代表轮廓对象, PAKRS 的总体计算代价也会随之降低. 随着 s 的增加, ε-GA 和 GA 的重算次数都略有降低, 这导致了 ε-GA 和 GA 的总运行时间也略有降低, 但降低幅度远小于 PAKRS. 因此, PAKRS 有效地利用了高速流的特性降低了算法的整体代价.



(a) STOCK (b) TRIP (c) PLANET

图 11 窗口流速 s 对算法效率的影响

第 3 组实验在真实数据集下测试参数 k 对算法性能的影响, 其他参数均使用默认参数, k 从 5 扩大到 25. 如图 12 所示.

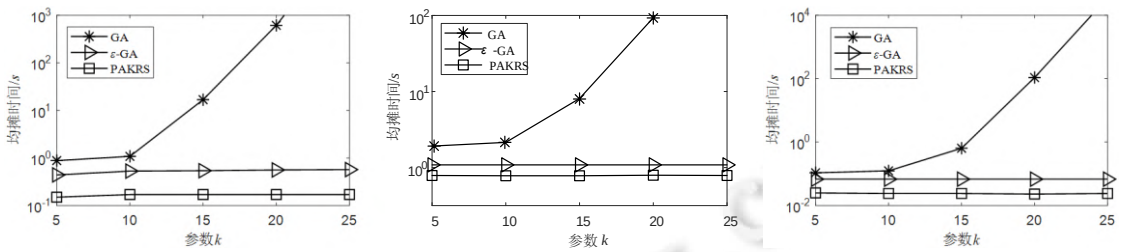
(1) PAKRS 的性能高于 GA 算法和 ε-GA 算法.

(2) PAKRS 在参数 k 较小时, 运行时间略高于 k 值变大时的运行时间. 当 k 到达 20 时, PAKRS 效率趋于稳定.

(3) GA 的效率随 k 的增加而大幅降低, 而 ε-GA 的效率几乎不受 k 的影响.

其原因在于, 当 k 值较小时, ρ-GRID 中的轮廓单元格数量一般大于 k, 此时需要以轮廓单元格为基本单元

计算近似 k 代表轮廓, 这会导致 PAKRS 查询的效率略微降低; GA 算法在计算 k 代表轮廓集合时, 需要枚举所有 k 个轮廓对象组合下的支配区域面积, 因此, 随着 k 的增加, GA 的查询效率会降低; 相反, ε-GA 算法根据 k 的增加调整阈值ε, 避免了枚举, 其效率对 k 不敏感.



(a) STOCK (b) TRIP (c) PLANET

图 12 参数 k 对算法效率的影响

第 4 组实验测试阈值ρ对算法性能的影响. 其他参数均使用默认值, ρ的取值从 1.001 扩大到 1.5. 如表 5 所示, PAKRS 的计算时间随ρ的升高而降低; 当ρ到达 1.1 后, 算法效率基本趋于稳定. 其原因在于: 当阈运行时间与ρ之间的关系. 实验发现: 随着ρ的增加, ρ-GRID 的划分粒度随之降低. 原因如下.

- 首先, 如第 3.4 节所述, 此时参与计算的轮廓单元格数目也会减少, PAKRS 整体计算代价也会降低. 当ρ值足够大时, 参与计算的轮廓单元格数目趋于稳定, 算法的整体性能也趋于稳定.
- 其次, PAKRS 的整体计算代价包括 GBRT 和近似 k 代表轮廓的维护代价, 而 GBRT 的维护代价与ρ取值无关. 因此, 当ρ发生变化时, PAKRS 的整体计算代价变化不大.
- 最后, 由于 GBRT 只在窗口滑动时重新计算近似 k 代表轮廓, 其计算频率相对较低, 这导致近似 k 代表轮廓的整体计算代价较低. 这从侧面说明了近似 k 代表轮廓算法的高效性和利用流速 s 降低整体计算代价的有效性.

表 5 运行时间 vs. ρ

数据集	维度	运行时间(s)							
		1.001	1.01	1.05	1.1	1.2	1.3	1.4	1.5
STOCK	3	2.11	1.96	**1.92**	1.92	1.92	1.92	1.92	1.92
TRIP	3	2.22	2.08	2.02	**1.98**	1.98	1.98	1.98	1.99
PLANET	4	1.12	1.05	1.03	**1.02**	1.02	1.02	1.02	1.02
独立数据集	2	0.32	0.36	0.41	**0.43**	0.44	0.44	0.44	0.45
	3	0.68	0.62	**0.58**	0.58	0.58	0.58	0.58	0.58
	4	2.52	2.31	**2.16**	2.16	2.16	2.16	2.16	2.16
	5	7.47	7.12	**6.79**	6.79	6.79	6.79	6.79	6.78
	6	67.23	66.42	**64.28**	64.07	63.92	63.92	63.92	63.92
反相关数据集	2	2.29	1.83	**1.66**	1.65	1.64	1.64	1.63	1.63
	3	6.52	6.35	6.22	**6.12**	6.12	6.12	6.12	6.12
	4	12.61	11.77	10.41	**10.27**	10.27	10.27	10.27	10.29
	5	41.62	38.32	36.11	**34.54**	33.76	33.76	33.76	33.76
	6	268.27	265.43	262.24	**260.78**	259.43	259.43	259.43	259.43

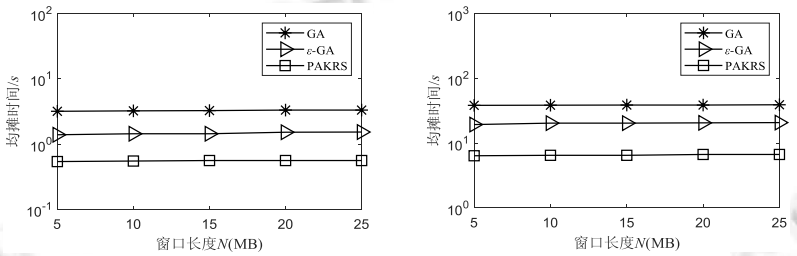
表 6 给出了随ρ增加, 精确 k 代表轮廓支配面积和近似 k 代表轮廓支配面积的比值. 实验中发现: 首先, 算法的精度随ρ增加而降低; 其次, 当ρ达到 1.1 后, 算法的误差趋于稳定. 原因在于, 算法是在不同轮廓单元格中选择近似查询结果, 这些轮廓单元格的组合可以覆盖大部分精确结果所对应的支配区域. 因此, ρ-GRID 可以帮助 PAKRS 在提高算法效率的同时返回高质量的查询结果. 本节分别在表 5、表 6 标记了各数据集下, 使得运行时间趋于稳定的ρ. 这说明, 当ρ达到 1.1 时, PAKRS 可在大部分情况下高效且高质量的返回结果. 这表明了 PAKRS 在实际应用中可代替精确算法支持高质量对象的筛选. 只要ρ值取值合理, PAKRS 就能在大多数情况下返回高质量的查询结果.

在对真实数据集进行测试后, 本节在合成数据集下, 分别测试了窗口长度 N、窗口流速 s 和参数 k 对算法

性能的影响. 参与ρ和 d 均使用默认值. 图 13–15 记录了实验的最终结果. 实验表明, PAKRS 算法的性能好于 GA 和ε-GA 算法. 这证明了 PAKRS 在不同数据分布下均能高效工作. 因此, PAKRS 对数据分布不敏感, 具有较好的稳定性.

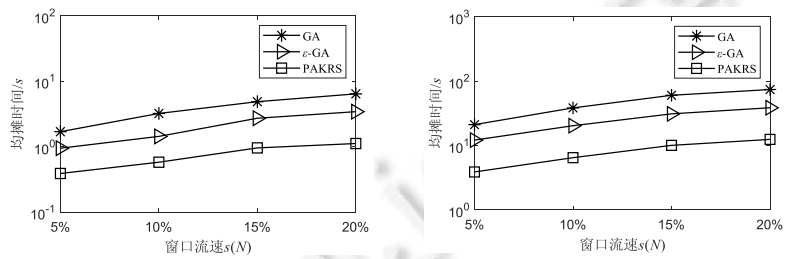
表 6 实际误差 vs. ρ

数据集	维度	实际误差							
		1.001	1.01	1.05	1.1	1.2	1.3	1.4	1.5
STOCK	4	1.000 7	1.002 4	**1.040 2**	1.047 0	1.056 8	1.103 6	1.224 1	1.281 4
TRIP	4	1.000 7	1.001 5	1.011 2	**1.025 4**	1.028 9	1.056 9	1.067 1	1.148 7
PLANET	3	1.000 8	1.008 6	1.038 4	**1.039 6**	1.057 0	1.058 1	1.146 2	1.242 1
独立数据集	2	1.000 3	1.002 6	1.009 6	**1.015 5**	1.025 8	1.027 9	1.048 5	1.052 5
	3	1.000 6	1.004 6	**1.012 6**	1.025 5	1.029 8	1.091 2	1.170 5	1.192 5
	4	1.000 6	1.004 3	**1.025 2**	1.056 4	1.081 0	1.112 4	1.115 5	1.128 5
	5	1.000 1	1.000 5	**1.002 7**	1.012 6	1.063 6	1.068 5	1.074 2	1.079 2
	6	1.000 2	1.000 2	**1.001 5**	1.001 9	1.001 9	1.003 7	1.006 8	1.010 6
反相关数据集	2	1.000 4	1.003 2	**1.003 4**	1.003 7	1.005 4	1.010 2	1.022 1	1.035 4
	3	1.000 6	1.004 2	1.004 4	**1.004 7**	1.007 4	1.014 2	1.054 2	1.095 4
	4	1.000 1	1.000 8	1.002 5	**1.006 5**	1.019 2	1.025 1	1.028 4	1.030 2
	5	1.000 1	1.000 3	1.000 7	**1.007 1**	1.010 4	1.011 0	1.027 4	1.042 5
	6	1.000 0	1.000 1	1.000 2	**1.006 9**	1.007 2	1.009 6	1.012 5	1.012 7



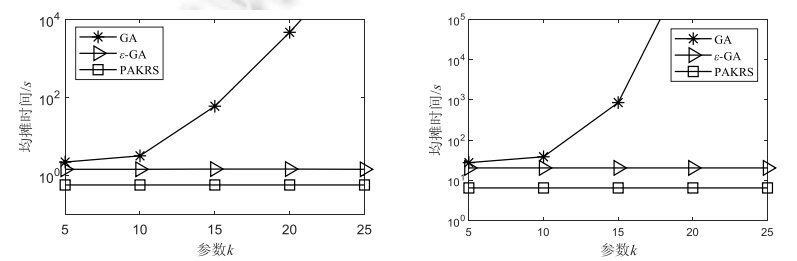
(a) 独立数据集 (b) 反相关数据集

图 13 运行时间 vs. N



(a) 独立数据集 (b) 反相关数据集

图 14 运行时间 vs. s



(a) 独立数据集 (b) 反相关数据集

图 15 运行时间 vs. k

接下来, 实验测试维度 d 对算法性能的影响. 测试数据集为独立数据集和反相关数据集, 其他参数均使用默认值, d 从 2 增加到 6. 表 7 展示了实验结果, 结果表明: (1) PAKRS 算法的性能最好; (2) 随着数据维度的增加, 3 种算法的处理时间均会增加, 而 PAKRS 增加幅度最低. 原因在于: 随着数据维度的增加, 轮廓对象的数量呈指数级增长. 因此, 这几种算法在进行轮廓查询时的效率均有所降低. 然而, PAKRS 由于使用了分片技术, 它保证算法在大部分时间内使用少量潜在轮廓对象支持轮廓查询. 因此, PAKRS 对数据维度的敏感度较低. 此外, 由于 PAKRS 引入了 ρ-GRID, 它在计算 k 代表轮廓集合时的效率受数据维度影响较小. 因此, PAKRS 对数据维度敏感程度最低.

表 7 运行时间 vs. d

数据集	算法	2	3	4	5	6
独立数据集	PAKRS	0.43	0.58	2.16	6.79	64
	PBA/GA	1.1	5.4	12.1	57.2	567
	ε-GA		2.62	12.5	27.9	288.2
反相关数据集	PAKRS	1.65	6.12	10.2	34.5	260.7
	PBA/GA	4.2	37.9	103	511	2109
	ε-GA		19.6	49.2	232	1492

最后, 实验测试对比了 k-LDS 和 PAKRS 的空间代价. 如表 8 所示, PAKRS 的空间代价小于 k-LDS 的空间代价. 原因在于, PAKRS 充分利用了高速流的特性. 当其处理一批到达窗口的数据时, 该算法只需保留这部分数据中的轮廓对象, 这一策略可帮助 PAKRS 删除窗口中大部分无效对象. 与之不同的是, k-LDS 没有利用高速流的特性优化无效对象的删除. 对于同一批次到达的数据, 即使某一对象 o 被另一对象 o' 支配, 如果算法先处理 o' 后处理 o, 那么 o 也无法被及时删除. 这导致 k-LDS 的空间代价较大. 此外, PAKRS 分别利用了对象支配关系的传递性、基于 π_{sk} 的方法删除无效对象, 这保证 PAKRS 可在大部分情况下只维护少量的无效对象.

表 8 内存消耗(MB)

数据集	算法		窗口长度 N (MB)					流速 s (N×%)			
			5	10	15	20	25	5	10	15	20
STOCK	ε-GA	d=3	0.23	0.29	0.37	0.46	0.53	0.29	0.29	0.29	0.29
	PAKRS	d=3	0.097	0.112	0.128	0.139	0.151	0.131	0.112	0.102	0.093
TRIP	PBA	d=2	0.05	0.057	0.064	0.071	0.076	0.057	0.057	0.057	0.057
	PAKRS	d=2	0.026	0.028	0.031	0.034	0.038	0.035	0.028	0.022	0.019
PLANET	ε-GA	d=4	0.81	1.03	1.24	1.51	1.76	1.03	1.03	1.03	1.03
	PAKRS	d=4	0.046	0.049	0.054	0.057	0.061	0.057	0.049	0.043	0.038
独立数据集	ε-GA	d=2	0.04	0.049	0.056	0.061	0.066	0.049	0.049	0.049	0.049
	PAKRS	d=2	0.019	0.021	0.024	0.029	0.035	0.028	0.021	0.017	0.015
	ε-GA	d=3	0.18	0.22	0.26	0.30	0.33	0.22	0.22	0.22	0.22
	PAKRS	d=3	0.081	0.086	0.111	0.126	0.141	1.04	0.086	0.073	0.068
	ε-GA	d=4	0.61	0.83	0.85	0.88	0.91	0.83	0.83	0.83	0.83
	PAKRS	d=4	0.38	0.46	0.49	0.53	0.57	0.54	0.46	0.43	0.39
	ε-GA	d=5	2.15	2.52	2.86	3.21	3.45	2.52	2.52	2.52	2.52
	PAKRS	d=5	1.09	1.22	1.51	1.73	1.98	1.45	1.22	1.18	1.04
	ε-GA	d=6	8.9	11.1	12.8	14.2	17	11.1	11.1	11.1	11.1
	PAKRS	d=6	4.8	6.2	7.5	8.3	9.1	7.4	6.2	5.5	5.7
反相关数据集	ε-GA	d=2	0.25	0.28	0.36	0.43	0.48	0.28	0.28	0.28	0.28
	PAKRS	d=2	0.089	0.091	0.092	0.103	0.120	0.122	0.091	0.086	0.074
	ε-GA	d=3	0.77	0.81	0.86	0.92	0.97	0.81	0.81	0.81	0.81
	PAKRS	d=3	0.34	0.42	0.51	0.49	0.52	0.55	0.42	0.41	0.36
	ε-GA	d=4	2.33	2.79	3.36	4.09	4.74	2.79	2.79	2.79	2.79
	PAKRS	d=4	1.28	1.37	1.57	1.98	2.32	1.55	1.37	1.17	1.08
	ε-GA	d=5	10.35	12.29	13.57	15.16	16.2	12.29	12.29	12.29	12.29
	PAKRS	d=5	6.19	6.23	7.36	8.27	8.87	7.49	6.23	5.88	5.46
	ε-GA	d=6	34.1	38.4	42.7	45.3	47.7	38.4	38.4	38.4	38.4
	PAKRS	d=6	19.8	21.6	22.4	23.7	23.9	25.4	21.6	18.7	16.3

本节的另一发现是, 随着流速的增加, PAKRS 的空间代价随之降低. 与之不同的是, k-LDS 对参数 s 不敏感. 其原因在于, k-LDS 没有利用流速这一参数降低空间代价. 最后, 如表 8 所示, PAKRS 在所有数据集下的空间代价均低于 k-LDS. 这表明 PAKRS 对数据分布不敏感, 具有较好的稳定性. 因此, PAKRS 利用了高速流

的特性, 有效地降低了空间代价.

综上所述, 与前人所提算法相比, PAKRS 能够在不同数据分布和参数下高效工作的同时, 返回较高质量的 k 代表轮廓. 因此, PAKRS 能够在高速流环境下支持近似 k 代表轮廓查询.

5 结束语

本文研究了高速流上的 k 代表轮廓查询问题, 并提出了ρ-近似 k 代表轮廓查询的概念. 为支持该查询, 本文提出了查询处理框架 PAKRS. 它通过对窗口进行有效的划分, 实现预测结果集的增量维护. 随后, 本文提出了索引ρ-GRID, 它根据参数ρ调整网格的划分粒度, 支持近似 k 代表轮廓查询. 实验结果表明, 本文所提算法在高速流上好于现有算法.

References:

[1] Lin X, Yuan Y, Zhang Q, *et al*. Selecting stars: The *k* most representative skyline operator. In: Proc. of the Int'l Conf. on Data Engineering. Istanbul, 2007. 86−95.

[2] Tao Y, Ding L, Lin X, *et al*. Distance-based representative skyline. In: Proc. of the Int'l Conf. on Data Engineering. Shanghai, 2009. 892−903.

[3] Sarma AD, Lall A, Nanongkai D, *et al*. Representative skylines using threshold-basedpreference distributions. In: Proc. of the Int'l Conf. on Data Engineering, Hannover, 2011. 387−398.

[4] Zhou X, Li K, Yang Z, *et al*. Efficient approaches to *k* representative G-skyline queries. ACM Trans. on Knowledge Discovery from Data, 2020, 14(5): 1−27.

[5] Bai M, Xin JC, Wang GR, *et al*. Discovering the *k* representative skyline over a sliding window. IEEE Trans. on Knowledge and Data Engineering, 2016, 28(8): 2041−2056.

[6] Borzsonyi S, Kossmann D, Stocker K. The skyline operator. In: Proc. of the IEEE Int'l Conf. Tucson, 2001. 421−430.

[7] Chomicki J, Godfrey P, Gryz J, *et al*. Skyline with presorting: Theory and optimization. In: Proc. of the Int'l Conf. on Intelligent Information Systems. Wroclaw, 2005. 216−225.

[8] Buchta C. On the average number of maxima in a set of vectors. Information Processing Letters, 1989, 33(2): 63−65.

[9] Godfrey P, Shipley R, Gryz J. Maximal vector computation in large data sets. In: Proc. of the Very Large Databases. Trondheim, 2005. 229−240.

[10] Lin X, Yuan Y, Wang W, *et al*. Stabbing the sky: Efficient skyline computation over sliding windows. In: Proc. of the 21st Int'l Conf. on Data Engineering. Tokyo, 2005. 502−513.

[11] Tao Y, Papadias D. Maintaining sliding window skylines on data streams. IEEE Trans. on Knowledge and Data Engineering, 2006, 18(3): 377−391.

[12] Zuo KZ, Hu P, Wang TC, *et al*. Privacy-preserving Skyline query protocol in two-tiered sensor networks. Ruan Jian Xue Bao/ Journal of Software, 2014, 25: 43−45 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/14013.htm

[13] Tang YF, Chen SP. Improved skyline query algorithm of data streams based on k-d tree index. Journal of Chinese Computer Systems, 2018, 39(3): 544−550 (in Chinese with English abstract).

[14] Ren W, Lian X, Ghazinour K. Skyline queries over incomplete data streams. The VLDB Journal, 2019, 28(6): 961−985.

[15] Alami K, Maabout S. A framework for multidimensional skyline queries over streaming data. Data & Knowledge Engineering, 2020, 127: Article No.101792.

[16] Tian L, Li AP, Zou P, *et al*. The continuous skyline computation on update data streams. Computer Engineering and Science, 2008, 30(5): 59−64 (in Chinese with English abstract).

[17] Zhang L, Zou P, Jia Y, *et al*. Continuous dynamic skyline queries over data stream. Journal of Computer Research and Development, 2011, 48(1): 77−85 (in Chinese with English abstract).

[18] Bai M, Xin JC, Wang GR, *et al*. Research on dynamic skyline query processing over data streams. Chinese Journal of Computers, 2016, 39(10): 2007−2030 (in Chinese with English abstract).

[19] Dimitris P, Tao YF, Greg F, *et al*. An optimal and progressive algorithm for skyline queries. In: Proc. of the ACM SIGMOD Int'l Conf. on Management of Data. San Diego, 2003. 467−478.

[20] Chan CY, Jagadish HV, Tan KL, *et al*. Finding *k*-dominant skylines in high dimensional space. In: Proc. of the ACM SIGMOD Int'l Conf. on Management of Data. Chicago, 2006. 27−29.

[21] Li S, Dou YN, Hao XH, *et al*. The method of the *k*-dominant space skyline query in road network. Journal of Computer Research & Development, 2020, 57(1): 227−239 (in Chinese with English abstract).

[22] Cui NN, Yang XC, Wang B, *et al*. Research on authentication of moving *k*-dominant NN queries. Chinese Journal of Computers, 2018, 41(8): 90−107 (in Chinese with English abstract).

[23] Zhu HY, Li XY, Liu Q, Xu ZC. Top-*k* dominating queries on skyline groups. IEEE Trans. on Knowledge and Data Engineering, 2020, 32(7): 1431−1444.
[24] Wang CP, Wang CK, Guo GY, Ye XJ, Yu P. Efficient computation of G-skyline groups. IEEE Trans. on Knowledge and Data Engineering, 2018, 30(4): 674−688.
[25] Bai M, Wang XT, Li GY, *et al*. Research on optimization algorithms of *k*-maximum coverage skyline queries. Chinese Journal of Computers, 2020, 43(12): 2276−2297 (in Chinese with English abstract).
[26] Malene SS, Sean C, Ira A. Maximum coverage representative skyline. In: Proc. of the Int'l Conf. on Extending Database Technology. Bordeaux, 2016. 702−703.
[27] Huang X, Zheng J. Deletion-robust *k*-coverage queries. In: Proc. of the Database Systems for Advanced Applications. Chiang Mai, 2019. 215−219.
[28] Zhou X, Li KL, Yang ZB, Li KQ. Finding optimal skyline product combinations under price promotion. IEEE Trans. on Knowledge and Data Engineering, 2019, 31(1): 138−151.
[29] Zhou X, Li KL, Xiao GQ, Zhou YT, Li KQ. Top *k* favorite probabilistic products queries. IEEE Trans. on Knowledge and Data Engineering, 2016, 28(10): 2808−2821.
[30] Gan J, Tao Y. DBSCAN revisited: Mis-claim, un-fixability, and approximation. In: Proc. of the ACM Conf. on Management of Data (SIGMOD). 2015. 519−530.
[31] Song FY, Zhu R, Zhang H, *et al*. Keyword skyline query algorithm over streaming data. Journal of Chinese Computer Systems, 2021, 42(9): 2004−2010 (in Chinese with English abstract).
[32] Zhu R, Wang B, Yang X, *et al*. SAP: Improving continuous top-*k* queries over streaming data. IEEE Trans. on Knowledge and Data Engineering, 2017, 29(6): 1310−1328.

附中文参考文献:

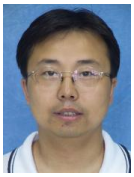
[12] 左开中, 胡鹏, 王涛春, 等. 两层传感器网络隐私保护 Skyline 查询协议. 软件学报, 2014, 25: 113−121. http://www.jos.org.cn/1000-9825/14013.htm
[13] 唐颖峰, 陈世平. 利用 k-d 树索引改进数据流 skyline 查询算法. 小型微型计算机系统, 2018, 39(3): 544−550.
[16] 田李, 李爱平, 邹鹏, 等. 更新数据流上的连续 Skyline 计算. 计算机工程与科学, 2008, 30(5): 59−64.
[17] 张丽, 邹鹏, 贾焰, 等. 数据流上连续动态 skyline 查询研究. 计算机研究与发展, 2011, 48(1): 77−85.
[18] 白梅, 信俊昌, 王国仁, 等. 数据流上动态轮廓查询处理技术的研究. 计算机学报, 2016, 39(10): 2007−2030.
[21] 李松, 窦雅男, 郝晓红, 等. 道路网环境下 *k*-支配区域 Skyline 查询方法. 计算机研究与发展, 2020, 57(1): 227−239.
[22] 崔宁宁, 杨晓春, 王斌, 等. 移动 *k*-支配最近邻查询验证研究. 计算机学报, 2018, 41(8): 90−107.
[25] 白梅, 王习特, 李冠宇, 等. 基于最大覆盖的代表 Skyline 问题的优化算法研究. 计算机学报, 2020, 43(12): 2276−2297.
[31] 宋楸尧, 朱睿, 张豪, 等. 数据流环境下的关键词轮廓查询算法. 小型微型计算机系统, 2021, 42(9): 2004−2010.

朱睿(1982−), 男, 博士, 副教授, CCF 高级会员, 主要研究领域为流数据管理, 查询处理与优化.

杨晓春(1973−), 女, 博士, 教授, 博士生导师, CCF 杰出会员, 主要研究领域为数据库理论与系统, 文本与时序大数据管理.

宋楸尧(1995−), 男, 硕士, 主要研究领域为流数据管理.

张安珍(1990−), 女, 博士,讲师, CCF 专业会员, 主要研究领域为大数据质量管理, 近似查询处理库.



王斌(1973−), 男, 博士, 教授, CCF 专业会员, 主要研究领域为数据质量管理, 文本数据管理.

夏秀峰(1965−), 男, 博士, 教授, CCF 高级会员, 主要研究领域为管理信息系统, 数据库.