

# 基于网格索引的连续 Skyline 计算方法

田 李 邹 鹏 李爱平 贾 焰

(国防科技大学计算机学院 长沙 410073)

**摘 要** 考虑按任意顺序随机增删的数据流场景下连续 Skyline 计算问题,首先基于已有工作提出了一个基本算法 BCSC;然后基于“影响区域”的观察,提出了一个基于网格索引数据结构的算法 GICSC,其基本思想为:(1)将数据空间划分为若干大小相等的网格,采用网格索引方法对数据点进行组织和管理;(2)用网格将数据空间表示为自由区域和影响区域两部分,发生在自由区域中的数据变化可以从理论上保证不影响计算结果,因此仅需对落于影响区域的数据增删进行运算,从而降低数据规模;(3)算法的计算模块通过逐步扩展的方法,无需遍历全部数据便可获得初始的 Skyline 集合及影响区域,维护模块通过类似方法计算数据变化对 Skyline 集合的影响,同时动态更新影响区域的大小.由于没有对数据流特性进行假设限制,因此 BCSC 和 GICSC 算法具有更广泛的适应性.理论分析和实验结果均验证了上述方法的有效性.

**关键词** 连续 Skyline 计算;数据流;网格索引数据结构

中图法分类号 TP311

## Grid Index Based Algorithm for Continuous Skyline Computation

TIAN Li ZOU Peng LI Ai-Ping JIA Yan

(School of Computer, National University of Defense Technology, Changsha 410073)

**Abstract** This paper addresses the problem of continuous Skyline computation on streams with random additions and deletions. A straightforward method called BCSC (Basic Continuous Skyline Computation algorithm) is firstly raised, then a Grid Index based Continuous Skyline Computation algorithm (GICSC) is presented based on the observation of influence region. The main idea of GICSC is as follows: (1) The work space is divided into lots of regular grids, and the valid data points are indexed and managed by this grid structure; (2) Some grids are organized as the influence region, while the rest compose of the free region. GICSC achieves low running time by handling data additions/deletions only from points that fall in the influence region, while data changes in the free region are omitted with correctness guarantee. (3) The computation module adopts a smart method to obtain the initial Skyline set and influence region without having to process all the data points; after that the maintenance module computes the change of Skyline and maintains the influence region dynamically when data changes. Since there is no assumption limitation of stream characters, the BCSC and GICSC algorithms are more adaptive. Analytical and experimental evidences show the efficiency of proposed approaches.

**Keywords** continuous Skyline computation; data stream; grid indexed data structure

收稿日期:2007-03-10;最终修改稿收到日期:2008-02-23. 本课题得到国家“八六三”高技术研究发展计划项目基金(2006AA01Z451, 2007AA01Z474)资助. 田 李,男,1980年生,博士研究生,主要研究方向为分布计算、数据库等. E-mail: Tian.L.cn@163.com. 邹 鹏,男,1957年生,教授,博士生导师,主要研究领域为分布计算和高级操作系统等. 李爱平,男,1974年生,博士,助理研究员,主要研究方向为人工智能、数据库和分布计算等. 贾 焰,女,1960年生,教授,博士生导师,主要研究领域为数据库、分布计算和人工智能等.

## 1 引言

假设数据集  $P$  中每条记录  $p$  都有  $d$  个属性  $p.x_1, \dots, p.x_d$ . 对于记录  $p_1$  和  $p_2$ , 如果  $p_1$  在任一维上的评价价值都不比  $p_2$  差, 且至少在某维上的评价价值优于  $p_2$ , 则称  $p_1$  “支配”  $p_2$ . Skyline 查询得到所有不被任何其它点所支配的点. 如图 1 在二维坐标上表示酒店信息, 每个点代表一个酒店, 横坐标和纵坐标分别表示该酒店的房价及其到某旅游景点的距离. 显然如果酒店  $a$  与  $b$  相比既近又便宜, 则  $a$  比  $b$  好 ( $a$  支配  $b$ ). 但通常情况下距离景点越近的酒店房价越高, Skyline 操作会返回所有不比其它酒店差的酒店 (图 1 加黑点), 供旅游者决策参考. 本质上, 对任意单调评价函数  $H$ , 使其达到最优值的点必定属于 Skyline 集合, 因此 Skyline 操作在多标准决策、偏好查询等领域具有非常重要的作用.

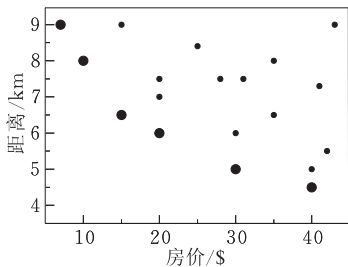


图 1 酒店 Skyline 示例

Skyline 计算在传统数据库领域<sup>[1-7]</sup>及分布环境下<sup>[8-11]</sup>均得到了广泛深入的研究. 近几年数据流处理和移动传感器网络成为数据库领域的一个研究热点, 基于滑动窗口数据流<sup>[12]</sup>和移动对象环境下的 Skyline 查询也得到了关注<sup>[13-15]</sup>, 已有一些连续跟踪计算 Skyline 的方法.

本文考虑按任意顺序随机增删的数据流场景下连续 Skyline 计算问题. 一个典型的例子是股票交易市场对股票信息的监控, 每支股票可以用如风险、交易量、每股平均价格等统计信息来描述, 投资者要对感兴趣的股票进行 Skyline 查询以供投资参考. 每笔股票交易都会导致对应股票的属性在原来值的基础上发生变化, 因此需要基于更新消息流进行连续 Skyline 计算. 该场景下, 由于无法预知下一笔交易会对应哪个股票, 而且用户关注的股票数也会随时变化, 因此代表股票信息数据点的增删及位置移动是以随机的形式发生, 使得该场景下的连续 Skyline 计算成为一个挑战性问题.

按任意顺序随机增删的数据流不再满足“先进先出”特性, 因此与文献[13-14]基于的滑动窗口数据流完全不同. 随机增删数据流场景与文献[15]考虑的移动对象环境也有很大差别: (1) CSQ 方法<sup>[15]</sup>假设对象以平稳的方式移动, 并强调以对象的移动速率来预测其未来位置. 这种假设在很多应用场景下并不成立, 而随机增删数据流场景认为对象下一时刻的位置是不可预测的. (2) 文献[15]中认为每个对象由不断变化的位置属性值和一组不变的静态属性值描述, “支配”关系是基于静态属性值大小和点之间“距离”来定义的, 也就是说, 所有动态属性均集中体现在“距离”特性中. 而在本文考虑的场景中, 对象的所有属性值均动态变化, 而且对每个属性均独立考虑其对“支配”关系的贡献.

首先基于已有工作提出一个直观的 BCSC 算法 (Basic Continuous Skyline Computation algorithm), 然后提出一个更为高效的 GICSC 算法 (Grid Index based Continuous Skyline Computation algorithm), 其主要设计思想如下: 将工作空间划分为若干大小相等的网格, 采用网格索引方法组织数据点; 工作空间分为影响区域和自由区域两部分, 当增删的数据点位于影响区域时可能会引起 Skyline 集合的变化, 而自由区域中的数据变化则可从理论上保证对结果无影响. 算法的计算模块负责计算初始 Skyline 结果, 并用最少的网格表示初始影响区域; 维护模块负责计算数据变化对 Skyline 集合的影响, 同时动态更新影响区域的大小. 理论分析和实验结果证明, 由于只需对落入影响区域中的部分数据进行计算, 因此该算法具有较高的运算效率.

本文第 2 节总结 Skyline 计算的相关工作; 第 3 节对问题进行形式化描述, 基于已有工作提出直观的 BCSC 算法, 并描述启发我们工作的一些观察特性; 第 4 节给出一种基于网格索引的数据结构, 基于该结构提出 GICSC 算法; 阐述了初始计算和动态维护功能模块的详细运算过程, 并从理论上证明了算法的正确性; 第 5 节对算法复杂度及实现相关的关键内容进行分析; 第 6 节通过实验验证上述方法的有效性; 第 7 节对全文进行总结和展望.

## 2 相关工作

Borzsonyi 等<sup>[1]</sup>首次将 Skyline 操作引入数据库系统, 提出了 BNL 和 D&C 两种计算方法. BNL 的优点是无需建立索引或把数据文件排序, 可以应用

到任意维空间;缺点是依赖内存,当候选列表比内存还大时,需要将列表中的部分数据保存到临时文件中,导致 BNL 算法的多遍执行. D&C 方法在数据规模较小时具有较高的效率,对于大数据集,划分合并过程会产生大量的 I/O 代价. SFS 方法<sup>[2]</sup>是 BNL 方法的变种,将数据集按照某个单调函数进行预排序后,可以通过单遍扫描有效计算 Skyline 集合,但增加了预排序的开销. 上述 3 种方法均需处理完所有数据后才开始输出结果,因此不适合在线计算. 文献[3]将原始数据集编码成位图(bitmap),提出了一种基于比特位运算的改进方法,同时还提出了一种基于索引(index)的方法,均能够在扫描全部数据集之前开始输出 Skyline 结果,因此适合在线计算. 但是这两种方法需要预先对数据集进行编码和创建索引,预处理需要较大的计算和存储开销,且不适合频繁发生增删变化的数据集;此外,这两种方法产生 Skyline 点的顺序是固定的,用户无法根据自己的喜好进行干预. Kossmann 等<sup>[4]</sup>采用 R-tree 结构对数据库进行索引,提出了 NN 算法. NN 方法随着运算时间增长逐渐得到越来越多的 Skyline 点,并且不会因为数据点在某一维上的明显优势而优先返回它,因此是渐进、公平的. 此外 NN 方法能够接受外界交互信息,用户可以干预结果的生成顺序,因此具有很好的灵活性. Papadias<sup>[5]</sup>分析了 NN 方法在 I/O 和存储方面存在的不足并对之进行了改进,提出了 BBS 算法,通过为每个项增加两次支配判断操作,能够将无需进一步考虑的数据进行裁减和丢弃,从而降低运算和存储规模,并在理论上证明了 BBS 算法是 I/O 最优的.

Skyline 查询也得到了国内学者的广泛关注. 文献[6]提出了一个基于查询窗口的算法:每个 Skyline 点对应一个有效区,描述了其它 Skyline 点可能存在的区域;系统维护一个查询窗口,只对有可能存在新 Skyline 点的区域进行查询;查询窗口的大小根据当前查找到的 Skyline 点及其有效区进行动态变化. 系统不断改变查询窗口缩减查询空间,因此不用访问全部数据集就能得到最终结果. 最近,高云君等<sup>[7]</sup>在存储开销方面对 BBS 算法进行了改进,定义了两种裁减策略,改进后的 IBBS 方法既有最佳 I/O 代价和较低的 CPU 开销,又有最少的存储消耗.

上述工作主要考虑集中环境下的 Skyline 计算,其目标是利用空间索引或编码技术快速得到 Skyline 结果,减少查询的 CPU、内存和 I/O 消耗. 与集中式环境不同,分布环境下的 Skyline 计算主

要考虑如何减少网络通信量. Balke 等<sup>[8]</sup>首次研究了 WEB 数据各维值分别存放在不同远程节点的分布环境下 Skyline 查询处理技术;文献[9]研究了数据子集远程存放的分布环境中全局 Skyline 的处理方法;文献[10]研究了 P2P 环境下的 Skyline 查询问题. 在国内,邓波等<sup>[11]</sup>在星型网络拓扑结构下,从提高并行度和支持渐进式计算方面对文献[8]方法进行了改进.

近几年数据流处理<sup>[12]</sup>成为数据库领域的一个研究热点,有学者研究了滑动窗口数据流上的连续 Skyline 计算. Tao 等<sup>[13]</sup>提出了一个系统结构,并给出了 Lazy 和 Eager 两个算法框架,讨论了算法的具体实现,并通过理论分析和实验结果证明了其算法能够满足较快速度数据流上的 Skyline 连续计算需求. Lin 等<sup>[14]</sup>考虑了连续数据流上的  $n$ -of- $N$  查询. 该研究充分发掘了滑动窗口数据流的特性,定义了“关键支配”关系,使系统需要保存的数据达到最小;采用新颖的编码方法将 Skyline 计算映射为刺探查问题(stabbing query),采用触发机制处理连续  $n$ -of- $N$  Skyline 查询,并通过实验证明其方法能够处理速度很快的数据流.

Huang 等研究了移动对象环境下的连续 Skyline 计算问题<sup>[15]</sup>. 该文假设包括查询点在内的所有数据点均以平滑可预测的方式进行移动,导致 Skyline 集合持续发生变化,提出了一个连续跟踪算法 CSQ;首先根据对象的静态属性值求出一定在 Skyline 集合中的点;然后根据这些点得出其它 Skyline 点可能存在的区域. 该文深入分析了点的空间位置对支配关系的贡献,限定了导致 Skyline 集合发生变化点的存在区域,并提供了一种高效的动态维护方法.

最近,关于 Skyline 的研究又掀起了一个高潮,主要体现在 Skyline 的一些变种操作和在特定应用场景中的处理方法上. 如:子空间上的 Subspace Skyline 和 Skyline Cube 计算<sup>[16-19]</sup>、以降低通信开销为目的的 P2P 环境下的(Subspace) Skyline 计算方法<sup>[20-21]</sup>、基于欧式距离<sup>[22]</sup>和城市道路网络中<sup>[23]</sup>的多源 Skyline 查询问题、反转 Skyline 查询(Reverse Skyline Query)处理<sup>[24]</sup>、不确定数据集上的概率 Skyline<sup>[25]</sup>等. 为了减少高维情况下的 Skyline 点数目,有学者研究了  $k$ -Dominant Skyline<sup>[26]</sup>、Skyline Frequency<sup>[27]</sup>和 Strong Skyline Point<sup>[28]</sup>等问题. 为了减小输出结果集, Koltun 等提出了近似支配代表(approximately dominating representatives)的概

念<sup>[29]</sup>, Lin 等将 Skyline 问题与 Top- $k$  问题结合, 研究了“ $k$  个最具代表性的 Skyline 点”(Top- $k$  RSP) 的问题<sup>[30]</sup>. 当数据属性维度上的可能值很少时 (low-cardinality domains), Morse 等提出了一个基于栅格的具有线性速度且与数据分布无关的快速 Skyline 计算方法<sup>[31]</sup>; Lee 等分析 Skyline 查询和 Z-order curve 之间的关联, 设计了一种新的称为 ZBtree 的索引结构, 基于该结构提出了 Skyline 计算、更新方法, 可以高效解决 Skyline 计算、增量式维护 Skyline 结果以及  $k$ -Dominate Skyline 问题<sup>[32]</sup>.

总之, 自 2001 年提出以来至今, Skyline 问题得到了广大研究者的广泛关注. 与上述已有工作不同, 本文考虑的是一种特殊数据流场景 (称为更新数据流或多流场景) 下的连续 Skyline 增量式计算问题.

### 3 预备知识

问题形式化描述如下:  $D = \{O_1, O_2, \dots, O_N\}$  是  $N$  个被监控对象的集合, 每个对象当前状态表示为  $\langle id, x_1, x_2, \dots, x_d \rangle$ , 其中  $id$  为对象的唯一标识,  $x_i$  ( $i=1, 2, \dots, d$ ) 是描述对象状态的  $d$  个属性, 组成了一个  $d$  维空间, 每个对象的当前状态都对应该空间中的一个点<sup>①</sup>. 数据元组  $tp = \langle flag, id, y_1, y_2, \dots, y_d \rangle$  持续到达, 表示新增监控对象  $O_{id}$  ( $flag=1$ )、取消对  $O_{id}$  对象的监控 ( $flag=-1$ ) 或  $O_{id}$  对应对象的当前最新属性值 ( $flag=0$ ), 导致数据集中点的增加、删除或者位置发生移动. 属性值变化导致的位置移动可以看作旧点失效同时增加了新的数据点, 因此后续讨论只考虑数据点增加和删除两个基本操作. 由于无法预知下一数据元组的类型及对应  $id$ , 因此数据集中点的变化是随机的. 不失一般性, 假设所有属性值均为  $(0, 1)$  范围内的浮点数, 且其值越小越好, 相关定义如下.

**定义 1.** 支配. 设两个对象  $O_1, O_2$  的属性分别为  $(x_1, x_2, \dots, x_d)$  和  $(y_1, y_2, \dots, y_d)$ , 若  $\forall i, x_i \leq y_i$  都成立, 且  $\exists j$  满足  $x_j < y_j$  ( $1 \leq i, j \leq d$ ), 则称  $O_1$  支配  $O_2$ , 表示为  $Dominate(O_1, O_2)$ . 显然, 支配关系存在传递性.

**定义 2.** Skyline 集合. 所有不被任何其它点所支配的点的集合称为 Skyline 集合, 其中每个点称为 Skyline 点.

用  $SK$  表示 Skyline 集合,  $RE = D - SK$  表示所有非 Skyline 点组成的集合, 我们的目标是实时跟踪最新  $SK$ , 以流的形式输出其变化情况  $\langle t, \pm, e \rangle$ ,

表示在  $t$  时刻点  $e$  成为 Skyline 点 (+) 或者从 Skyline 集合中移除 (-).

#### 3.1 基本方法 BCSC

考虑到数据集发生变化时只会影响到有限范围内的数据, 而绝大部分数据点及其位置关系并没有发生变化, 因此提高数据流环境下连续 Skyline 计算效率的有效方法是根据数据集发生的变化, 在原有结果的基础上求得更新的 Skyline 集合. 总结已有工作, 分析数据集发生变化时对 Skyline 集合的影响<sup>[1, 13-14]</sup>, 可以得到如下定理.

**定理 1.** 当新增某个数据点  $r$  时, Skyline 集合的变化遵循以下 3 个原则:

(1) 若  $\exists e \in SK$  满足  $Dominate(e, r)$ , 则  $SK$  不发生任何变化; (2) 若  $\forall e \in SK, Dominate(e, r)$  均不成立, 则  $SK = SK + \{r\}$ ; (3) 若  $\exists e \in SK$  满足  $Dominate(r, e)$ , 则  $SK = SK - S$ , 其中  $S = \{e | e \in SK \text{ 且 } Dominate(r, e)\}$ .

**定理 2.** 当某个数据点  $r$  失效时, Skyline 集合的变化遵循以下 3 个原则:

(1) 对  $\forall e \neq r$ , 若  $r$  失效前  $e \in SK$  成立, 则  $r$  失效后  $e \in SK$  仍成立; (2) 若  $r \in RE$ , 则  $SK$  不发生任何变化; (3) 若  $r \in SK$ , 则  $r$  失效后  $SK = SK - \{r\} + S$ , 其中  $S$  为仅被  $r$  唯一支配的数据集的 Skyline.

根据定义 1、定义 2 不难证明上述定理的正确性. 根据上述分析, 下面提出基本方法 BCSC. 它可以在原有计算结果的基础上根据数据集变化情况得到更新后的 Skyline 集合, 其中模块 AP (Addition Processing) 依据定理 1 处理数据点的增加; 模块 DP (Deletion Processing) 依据定理 2 处理数据点的失效, 伪代码描述如图 2 所示. BCSC 的主要思想与文献[13]的 Lazy 算法完全一致, 但由于其不再限于更新数据流场景, 因此无法像 Lazy 那样根据到达顺序对系统保存的数据进行裁减和丢弃.

BCSC 算法对每个新增的数据点都要计算其是否被当前的 Skyline 点支配, 以判断该新增点是否属于更新后的 Skyline 集合 (图 2 BCSC\_AP 第 2 行); 当某个 Skyline 点  $r'$  失效时, 需要对整个 RE 集合中的点进行计算, 求得被  $r'$  唯一支配点集的 Skyline 集合 (图 2 BCSC\_DP 第 4, 5 行). 以上两项操作运算量大, 是决定 BCSC 效率的关键因素.

① 文中交替使用的属性元组、点、位置等名词具有相似的含义.

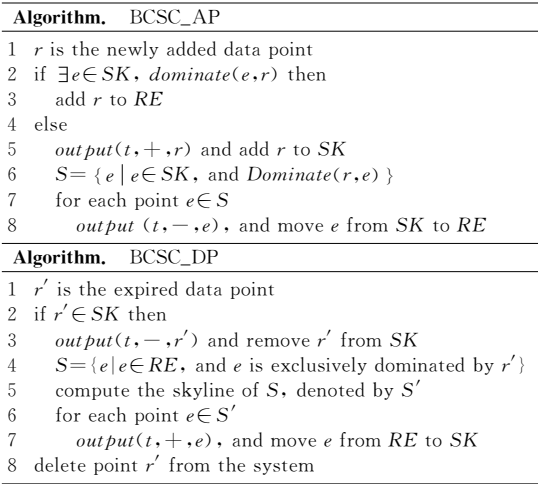


图 2 BCSC 算法关键模块伪代码描述

3.2 影响区域

通过对 Skyline 问题进行的深入分析,下面提出“影响区域”(Influence Region, IR)的概念,图 3 表示了二维空间中的一个例子。

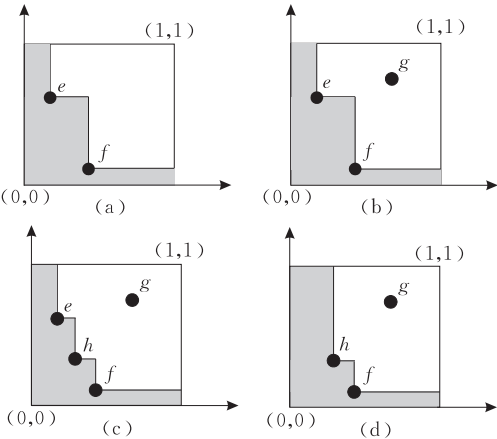


图 3 影响区域示意

如图 3(a)所示,连接 Skyline 点  $e, f$  形成的折线将工作空间划分为两部分:曲线左下部分(图中阴影部分)称为“影响区域”,其余部分称为“自由区域”。若数据点的添加删除操作发生在自由区域(如图 3(b)中的  $g$  点),则不会对当前 Skyline 集合产生任何影响;若数据的增删落在影响区域,则会导致 Skyline 集合发生变化,上述两个区域的大小同时也会受到影响。具体来说,假设新增加的  $h$  点落在影响区域,会导致影响区域缩小(如图 3(c)所示);若某一 Skyline 点失效,影响区域则会扩大(图 3(d)表示了在某时刻  $e$  点失效后的区域划分情况)。

基于上述观察,本文提出另外一种连续 Skyline 计算的方法。4.1 节首先提出一种网格索引数据结构,该结构将距离相近的点用规则网格统一组织,同

时允许用网格近似表示影响区域;4.2 节提出了一种无需遍历所有数据即可计算初始 Skyline 和影响区域的方法;连续 Skyline 跟踪计算及影响区域的动态维护过程在 4.3 节进行了详细描述。

4 基于网格索引的 GICSC 算法

4.1 网格索引数据结构

受已有工作启发<sup>[33-34]</sup>,设计了一种基于网格索引的数据结构,如图 4 所示。

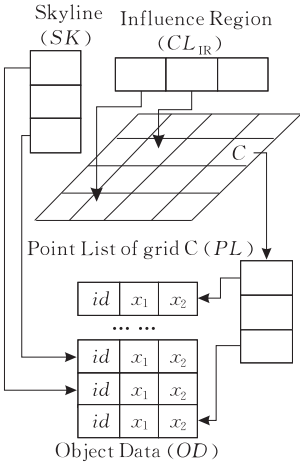


图 4 网格索引数据结构

以二维空间为例,每个对象  $O$  的当前状态可以表示为  $\langle id, x_1, x_2 \rangle$ ;整个空间被划分成多个等宽的网格,网格在任一维上的宽度均为  $\delta$ ,网格  $C_{i,j}$  包含了同时满足  $i \cdot \delta < O.x_1 \leq (i+1) \cdot \delta$  和  $j \cdot \delta < O.x_2 \leq (j+1) \cdot \delta$  的所有对象  $O$ ,用  $IN(O, C_{i,j})$  表示这种包含关系;反过来,对于任意对象  $O$ ,可以很容易计算出其所在的网格  $C_{i,j}$ ,其中  $i = \lfloor O.x_1 / \delta \rfloor, j = \lfloor O.x_2 / \delta \rfloor$ . 用  $C_{i,j}.LB$  和  $C_{i,j}.RT$  分别表示网格  $C_{i,j}$  的左下角和右上角顶点,显然对包含于网格中的任意数据点  $e, \text{Dominate}(C_{i,j}.LB, e)$  和  $\text{Dominate}(e, C_{i,j}.RT)$  均成立。

所有对象数据按  $id$  以 Hash 表形式进行组织,以方便对象的查找;每个网格包含一个指针列表  $PL$ ,指向该网格包含的数据点;系统包括一个 Skyline 集合的指针列表  $SK$ ,记录了全部 Skyline 点信息;系统还包括一个网格指针的列表  $CL_{IR}$ ,是用网格近似表示的影响区域。

**定义 3.** 点与网格的支配. 对于数据点  $e$  和网格  $C_{i,j}$ ,若  $\text{Dominate}(e, C_{i,j}.LB)$  成立,则称  $e$  严格支配  $C_{i,j}$ ,用  $\text{StrictDominate}(e, C_{i,j})$  表示;若  $\text{Dominate}(e, C_{i,j}.RT)$  成立,则称  $e$  松支配  $C_{i,j}$ ,用



$LaxDominate(e, C_{i,j})$  表示。

**定义 4.** 影响区域, 不被任何 Skyline 点严格支配的网格组成的集合称为用网格近似表示的影响区域, 即  $CL_{IR} = \{ \cup C_{i,j} \mid \forall e \in SK, StrictDominate(e, C_{i,j}) \text{ 均不成立} \}$ . 工作空间的其余部分称为近似自由区域。

根据定义 4, 图 3(a) 所示的影响区域在  $4 \times 4$  网格中的近似表示如图 5 斜线部分所示。可见,  $CL_{IR}$  是包含精确影响区域 IR (图 5 阴影部分) 的最小网格集合。

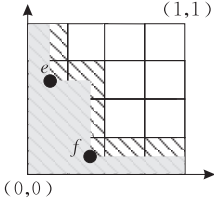


图 5 用网格近似表示的影响区域

由定义 3、定义 4 及支配关系的传递性可以得到如下引理。

**引理 1.** 对于数据点  $p$ 、网格  $C_{i,j}$  以及满足  $IN(e, C_{i,j})$  的任意点  $e$ , 若  $StrictDominate(p, C_{i,j})$  成立, 则  $Dominate(p, e)$  成立; 若  $LaxDominate(p, C_{i,j})$  不成立,  $Dominate(p, e)$  均不成立。

**定理 3.** 对于满足  $IN(e, C_{i,j})$  的数据点  $e$  和网格  $C_{i,j}$ , 若  $C_{i,j} \notin CL_{IR}$ , 则 Skyline 集合不会因  $e$  的增删而变化。

证明.  $C_{i,j} \notin CL_{IR}$ , 由定义 4 得  $\exists p \in SK, StrictDominate(p, C_{i,j})$ , 又  $IN(e, C_{i,j})$ , 由引理 1 得  $Dominate(p, e)$ . (1) 若  $e$  为新增数据点, 由定理 1 得, Skyline 集合不因为  $e$  的增加而变化; (2) 若  $e$  为已存在的数据点, 由  $Dominate(p, e)$  及 Skyline 集合的定义知,  $e \in RE$ . 由定理 2 得, Skyline 集合不因为  $e$  的失效而变化。综上所述,  $e$  的增删对 Skyline 集合没有影响。证毕。

定理 3 说明, 只有当增删的数据点落在  $CL_{IR}$  表示的区域中时才可能引起 Skyline 集合的变化, 而对于发生在其它区域中的数据变化则无需处理, 从而可以降低数据规模, 提高运算效率。基于上述数据结构和理论分析, 下面提出 GICSC 算法。剩余的问题主要包括如何计算初始 Skyline 集合和影响区域, 以及当数据变化发生在影响区域中时, 如何动态维护 Skyline 集合和影响区域的变化。

## 4.2 初始计算模块

初始计算模块 CM (Computation Module) 负责计算初始 Skyline 集合和影响区域。一个直接的方法

是首先采用现成算法求得初始 Skyline 集合, 然后遍历所有网格并根据定义 4 逐个判断得到初始的  $CL_{IR}$ 。导致该方法低效的原因主要有两个: 计算 Skyline 集合和影响区域的过程相互独立, 没有发掘可共享的计算操作; 需要遍历所有网格以求得正确的影响区域。受 NN<sup>[4]</sup> 方法和 BNL<sup>[1]</sup> 算法启发, 提出一个无需遍历全部空间的初始计算方法 GICSC\_CM, 如图 6 所示。其中  $Num$  表示某维上网格划分的个数, 假设任一维上属性的值域均为  $U$ , 网格在任一维上的宽度均为  $\delta$ , 则  $Num = U/\delta$ 。由于本文假设任一维的属性值域均为  $(0, 1)$ , 因此  $Num = 1/\delta$ 。

系统首先将 Skyline 集合  $SK$  和影响区域  $CL_{IR}$  初始化为空, 并创建一个空的遵循先进先出规则的队列  $Q$  (第 1, 2 行); 然后从工作空间的最左下角网格  $C_{0,0}$  开始, 将之加入到  $Q$  中并开始处理。在每次循环过程中, 从  $Q$  队列头取下一个网格  $C_{i,j}$ , 对与  $C_{i,j}$  相邻的网格  $C_{i+1,j}$  和  $C_{i,j+1}$  进行判断, 若相邻网格尚未扩展过, 且当前  $SK$  中的任意数据点  $e$  均不严格支配该网格 (第 7, 9 行), 则将之扩展增添至  $Q$  队列尾; 第 11~12 行表示根据当前  $SK$  中的数据, 对  $C_{i,j}$  网格中的每一个点  $p$  都依据 BNL 算法<sup>①</sup> 进行处理, 同时更新  $SK$ ; 最后将  $C_{i,j}$  添加至影响区域  $CL_{IR}$  中; 依次重复上述运算, 直至  $Q$  为空。

### Algorithm. GICSC\_CM

Input: Data set  $P$

Output:  $SK$  (Skyline of  $P$ ) and  $CL_{IR}$  (Initial influence region)

- 1  $SK = NIL, CL_{IR} = NIL$
- 2 Initialize an empty first-in first-out queue  $Q$
- 3 Let  $C$  be the grid in left-bottom corner of the workspace  $(C_{0,0})$
- 4 Insert  $C$  into  $Q$
- 5 Repeat
- 6   Get the next entry  $C_{i,j}$  of  $Q$
- 7   If  $C_{i+1,j}$  is not extended, and  $i+1 < Num$ , and  $\forall e \in SK, StrictDominate(e, C_{i+1,j})$  is not hold then
- 8     Extend  $C_{i+1,j}$  and append it to  $Q$
- 9   If  $C_{i,j+1}$  is not extended, and  $j+1 < Num$ , and  $\forall e \in SK, StrictDominate(e, C_{i,j+1})$  is not hold then
- 10    Extend  $C_{i,j+1}$  and append it to  $Q$
- 11   For each point  $p$  in  $C_{i,j}$
- 12      $BNL(p, SK)$ , and output skyline change
- 13   Add  $C_{i,j}$  into  $CL_{IR}$
- 14 Until  $Q$  is empty

图 6 GICSC 算法初始计算模块

图 3(a) 所示例子在  $4 \times 4$  网格结构中的计算过程如图 7 所示, 其中阴影部分表示当前影响区域, 斜

① 由于数据流环境下对处理速度要求较高, 故此处是指在内存中实现的 BNL 算法, 即候选 Skyline 点列表  $SK$  全部放在内存中。

线部分表示队列  $Q$  中包含的网格, 网格中的数字表示其扩展加入到  $Q$  中的顺序. 系统首先从 0 号网格  $C_{0,0}$  开始(如图 7(a)所示); 图 7(b)显示了处理完 0 号网格并扩展相邻网格后的情形; 从  $Q$  中取出 1 号网格( $C_{1,0}$ )继续进行处理(如图 7(c)所示), 由于该网格中包含 Skyline 点  $f$ , 因此 BNL 方法导致  $SK$  中增加了该点; 继续处理 2 号网格( $C_{0,1}$ )(如图 7(d)所示), 由于网格 4 已经在处理网格 1 时被扩展过, 故此只需扩展网格 5( $C_{0,2}$ ); 处理 3 号网格和 4 号网格时, 由于此时  $SK$  中包含  $f$  点, 且  $StrictDominate(f, C_{2,1})$ , 因此网格  $C_{2,1}$  不满足扩展条件(如图 7(e)所示); 处理 5 号网格的过程与前面类似, 增加了新的 Skyline 点  $e$ , 且只对尚未扩展过的 8 号网格( $C_{0,3}$ )进行扩展(如图 7(f)所示). 对 6~8 号网格的处理将不会再引起网格扩展, 因为其相邻的网格均被  $SK$  中数据点( $e$  或  $f$ )严格支配. 依次处理完这 3 个网格后程序会正常结束, 得到与图 5 一致的 Skyline 集合和影响区域.

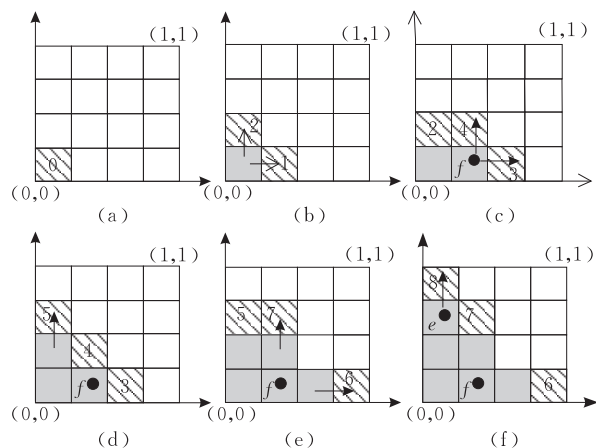


图 7 GICSC\_CM 运算过程示意

下面对 GICSC\_CM 算法的正确性进行证明. 由算法 6~10 行网格扩展的顺序, 容易得到如下引理.

**引理 2.** GICSC\_CM 算法中, 扩展网格  $C_{m,n}$  进入队列  $Q$  时, 所有满足  $i < m$  且  $j < n$  的网格  $C_{i,j}$  均已处理完毕.

**定理 4.** GICSC\_CM 算法能够输出正确的 Skyline 集合和符合定义 4 的影响区域.

**证明.** 定理的证明分为两部分: (1) 算法结束后的  $SK$  即为全局 Skyline 集合; (2) 算法结束后的  $CL_{IR}$  即为定义 4 规定的影响区域.

(1) 由于  $SK$  的变化是通过内存中的 BNL 算法维护的. 若对数据集中的所有数据均进行处理, 则运

算结束后  $SK$  必为全局 Skyline 集合. GICSC\_CM 算法只对部分网格中的数据进行了处理, 因此只需证明未处理网格中不包含 Skyline 点.

用  $SK_i$  表示第  $i$  次循环过程中的 Skyline 点候选列表, 对  $i < j$ , 若  $e \in SK_i$  成立, 则  $e \in SK_j$  或  $\exists p \in SK_j, Dominate(p, e)$  必有一个成立; 由 GICSC\_CM 算法第 9, 11 行可知, 若网格  $C$  未被处理, 则必然  $\exists i, \exists e \in SK_i$  满足  $StrictDominate(e, C)$ , 所以当 GICSC\_CM 算法终止时, 必  $\exists p \in SK, StrictDominate(p, C)$ . 即当算法结束时, 对  $\forall C \notin CL_{IR}$ , 必  $\exists p \in SK, StrictDominate(p, C)$ . 由引理 1 和定义 1 知, 未处理网格中不包含 Skyline 点.

(2) 算法第 13 行说明, 任何进入队列  $Q$  的网格都会加入到  $CL_{IR}$  中. 假设  $C_{i,j}$  在第  $i$  次循环过程中被扩展并加入到  $Q$ , 算法第 7~10 行仅判断其不被  $SK_i$  中的任何点严格支配. 因此若算法返回  $CL_{IR}$  与定义 4 相符, 需要证明网格  $C_{i,j}$  进入  $Q$  后, 它也不会被后续  $SK_j (j > i)$  中的任何点严格支配.

对网格  $C_{i,j}$  中的点  $e$  和网格  $C_{m,n}$ , 依据 4.1 节相关定义可知, 若  $StrictDominate(e, C_{m,n})$  成立, 则  $i < m$  和  $j < n$  一定成立. 而引理 2 说明, 扩展网格  $C_{m,n}$  进入队列  $Q$  时, 所有满足  $i < m$  且  $j < n$  的网格  $C_{i,j}$  均已处理完, 即所有可能严格支配该网格的点在扩展网格时均已考虑完毕, 因此它不会被后续循环处理中的任何点严格支配.

综上所述, 算法 GICSC\_CM 能够输出正确的 Skyline 集合和符合定义 4 的影响区域. 证毕.

最后, 虽然上述分析基于二维空间, 但是网格索引数据结构及 GICSC\_CM 算法可以很容易应用到任意  $d$  维 ( $d > 2$ ) 空间中. 例如在 3D 空间中, 算法的唯一变化就是处理完网格  $C_{i,j,w}$  后, 考虑扩展的相邻网格变为  $C_{i+1,j,w}, C_{i,j+1,w}$  和  $C_{i,j,w+1}$ . 另外, 本文假设属性值越小越好, 算法简单修改后也可适应其它情况. 例如当属性值越大越好时, 算法应该从最右上角的网格开始处理, 扩展的相邻网格也相应变为  $C_{i-1,j}$  和  $C_{i,j-1}$ .

### 4.3 维护模块

计算出初始结果后, 维护模块 (maintenance module) 负责接收数据流, 计算 Skyline 集合发生的变化并动态更新影响区域大小. 数据集的基本操作只有增加和删除两种, 与 BCSC 算法类似, 维护模块主要包括 GICSC\_AP 和 GICSC\_DP 两部分, 分别处理数据点的增加和失效, 伪代码描述如图 8 所示.

**Algorithm.** GICSC\_APInput: An incoming data tuple  $r$ Output: Change of  $SK$  and  $CL_{IR}$ , if needed

```

1  $r$  is the newly added data point, and  $C_{i,j}$  is the corresponding
  grid
2 Add  $r$  to  $C_{i,j}.PL$ 
3 If  $C_{i,j} \in CL_{IR}$  then
4   If  $\forall e \in SK, Dominate(e, r)$  is not hold then
5     output( $t, +, r$ ) and add  $r$  to  $SK$ 
6   For every  $e \in SK$ 
7     If  $Dominate(r, e)$  then
8       output( $t, -, e$ ) and remove  $e$  from  $SK$ 
9   If  $StrictDominate(r, C_{i+1,j+1})$  and  $C_{i+1,j+1} \in CL_{IR}$  then
10    Initialize an empty first-in first-out queue  $Q$ 
11    Insert  $C_{i+1,j+1}$  into  $Q$ 
12    Repeat
13      Get the next entry  $C_{m,n}$  of  $Q$ 
14      If  $C_{m+1,n} \in CL_{IR}$  and  $m+1 < Num$  then
15        Extend  $C_{m+1,n}$  and append it to  $Q$ 
16      If  $C_{m,n+1} \in CL_{IR}$  and  $n+1 < Num$  then
17        Extend  $C_{m,n+1}$  and append it to  $Q$ 
18      Remove  $C_{m,n}$  from  $CL_{IR}$ 
19    Until  $Q$  is empty

```

**Algorithm.** GICSC\_DPInput: Deletion of data tuple  $r'$ Output: Change of  $SK$  and  $CL_{IR}$ , if needed

```

1  $r'$  is the expired data point, and  $C'$  is the corresponding grid
2 If  $C' \in CL_{IR}$  and  $r' \in SK$  then
3   output( $t, -, r'$ ) and remove  $r'$  from  $SK$ 
4   Initialize an empty first-in first-out queue  $Q$ 
5   Insert  $C'$  into  $Q$ 
6   (Same as line 5~14 of algorithm GICSC_CM)
7 Remove  $r'$  from  $C'.PL$ , and delete point  $r'$  from the
  system

```

图 8 GICSC 算法维护模块伪代码描述

当新点到达时,首先将之加入到对应网格的  $PL$  列表中(GICSC\_AP 算法第 2 行);若该点落在自由区域内,则算法结束;否则,若新点成为 Skyline 点则需更新  $SK$  和  $CL_{IR}$ . 依据定理 1,第 4 行判断新点是否为 Skyline 点,若是则在第 5~8 行对  $SK$  进行修改;第 9~19 行将  $CL_{IR}$  中被新点严格支配的网格剔除,缩减影响区域的大小. 影响区域缩减采取了与初始计算类似的方法,无需对  $CL_{IR}$  中的全部网格进行扫描判断,而是从新加点所在网格的右上角相邻网格( $C_{i+1,j+1}$ )开始,通过队列  $Q$  和相邻网格的逐步扩展判断完成.

图 9 示意了上述过程,其中图 9(a)表示新点到达前的状态;某时刻点  $g$  到达,由于其支配已有 Skyline 点  $f$ ,导致点  $f$  从 Skyline 集合中移出,同时

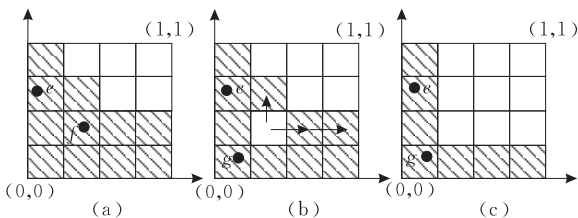


图 9 GICSC\_AP 运算过程示意

从  $g$  所在网格的右上角相邻网格( $C_{1,1}$ )开始依次扩展删除  $CL_{IR}$  中被  $g$  严格支配的网格(图 9(b));最后得到图 9(c)所示的更新后的 Skyline 集合和影响区域.

当旧点失效时,若失效点为非 Skyline 点,则只需将该点从系统中删除(GICSC\_DP 算法第 7 行),否则需要计算  $SK$  和  $CL_{IR}$  的更新情况. 更新从失效点所在的网格开始,通过与 GICSC\_CM 相同的方法完成(第 4~6 行). 限于篇幅,过程示例及维护模块的正确性证明略.

## 5 算法分析

BCSC 和 GICSC 是两个算法框架,具体实现时可以考虑运算效率、存储开销等因素,选择不同的数据结构来实现. BCSC 算法分析与 Lazy 算法分析<sup>[13]</sup>非常类似,因此本节只对 GICSC 算法进行分析.

GICSC 算法中频繁的操作是根据点确定其所属网格以及查找某网格是否属于影响区域,因此网格的存储组织是影响性能的一个重要因素. 最直接的方法是采用  $d$  维数组进行存储,网格在各维的编号作为数组的下标. 该方法具有操作简单存取快捷的特点,但是会导致大量存储开销,且随着维度的增加呈指数级增长;另外当数据分布不均匀时,许多网格内没有落入任何点,导致大量存储浪费. 为了节省存储开销,可以只为非空网格动态分配空间并采用链表结构进行组织. 但该方法使得网格定位查找时间随链表长度线性增长,严重影响运算效率. 第三种方法是采用动态索引结构如 R-Tree 等,动态分配内存并具有较高的查找效率,是上述两种方法的折衷. 本文结合多维数据线性化技术 Z-Curve<sup>[35]</sup>和现成的索引结构对网格进行组织存储.

依据文献[35]提出的方法,每个网格可以快速计算获得唯一的 Z-Value:假设空间中每维上划分的网格数均为  $2^n$  个,则网格  $C_{x_1,x_2,\dots,x_d}$  可以用  $d \times n$  个 bit 位表示为  $(b_{11} b_{12} \dots b_{1n}, b_{21} b_{22} \dots b_{2n}, \dots, b_{d1} b_{d2} \dots b_{dn})$ ,其中  $b_{ij} \in \{0,1\}$  为网格第  $i$  维坐标二进制表示的第  $j$  个 bit 位;而 Z-Value 定义为二进制串  $b_{11} b_{21} \dots b_{d1} b_{12} b_{22} \dots b_{d2} \dots b_{1n} b_{2n} \dots b_{dn}$  代表的整数值. 本文采用动态创建非空网格的方法,并将之按照 Z-Value 值以 Hash 表形式进行组织; $CL_{IR}$  以 Z-Value 作为关键字,并选择合适的索引结构以提高搜索效率. 因为在查找过程中只需判断网格的 Z-Value 值,因此与



其它方法需要判断  $d$  个值相比, 搜索效率可以得到较大提高; 而动态创建非空网格的方法则会最大限度地节省存储开销. 例如在图 9(a) 所示的情形下,  $CL_{IR}$  包含 11 个网格, 但只有  $C_{0,2}$  和  $C_{1,1}$  中包含数据点, 其它都是空网格, 因此系统只为这两个网格开辟空间以保存对应的 Z-Value、PL、扩展标识等, 而其它 9 个网格仅在  $CL_{IR}$  中开辟了存放其 Z-Value 的空间以供查询判断使用.

GICSC 算法还需要频繁判断某点是否为 Skyline 点(图 8 GICSC\_DP 算法第 2 行), 因此  $SK$  也应该以对象  $id$  作为关键字并选择合适的索引结构以提高搜索效率; 而算法对具体网格内的点总是依次进行处理, 因此 PL 可以采用链表结构以减少存储开销.

下面对算法的复杂度进行分析. 首先分析算法的空间复杂度, 为简便起见, 假设内存中的最小单元可以存放一个浮点数或一个指针类型数据, 考查算法所需内存单元数. 假设数据集中平均包含  $N$  个数据点, 存储共需  $N(d+1)$  个内存单元. 用  $\overline{SK}$  表示  $SK$  集合的势的平均大小, 则保存  $SK$  列表需要  $\overline{SK} \cdot (1+st_{SK})$  个内存单元, 其中一个单元存放数据点的  $id$ ,  $st_{SK}$  根据  $SK$  实现采用数据结构的不同而不同, 是每个元组维持结构需要的平均额外开销: 若  $SK$  采用链表结构, 则  $st_{SK}=1$ , 保存链表结构所需的指向下一元组的指针信息; 若  $SK$  采用二叉树结构, 则  $st_{SK}=2$ , 分别保存指向左右子元组的指针. 类似的, 用  $\overline{CL}$  表示  $CL_{IR}$  集合的势的平均值, 则影响区域平均需要  $\overline{CL} \cdot (1+st_{CL})$  个内存单元, 分别保存网格的 Z-Value 值和维护数据结构. 由于采用链表结构, 网格中 PL 的每个元组需要 2 个内存单元, 分别存放数据点  $id$  和下一元组的指针, 所有网格的 PL 共需  $2N$  个内存单元. 最后, 每个非空网格需要 3 个单元, 分别保存对应的 Z-Value、PL 头指针和是否扩展的状态标识. 每维上的值域均为  $(0,1)$ , 网格宽度为  $\delta$ , 系统中包含的总网格数为  $CN=(1/\delta)^d$ , 假设非空网格的比例为  $P$ , 因此共需  $3P(1/\delta)^d$  个单元. 算法运行过程中的队列  $Q$  长度不会超过  $\overline{CL}$ , 因此最多需要  $\overline{CL}$  个内存单元. 综上可以得到 GICSC 算法的空间复杂度为

$$O(N(d+3)+\overline{SK} \cdot (1+st_{SK}) + \overline{CL} \cdot (2+st_{CL}) + 3P(1/\delta)^d) \quad (1)$$

可见, 所需的空间与数据规模、维度、数据分布、网格宽度以及具体实现选择的数据结构相关.

下面分析 GICSC 算法的时间复杂度. 与已有工作类似<sup>[33-34]</sup>, 假设数据在  $d$  维空间中服从均匀分布<sup>①</sup>, 则每个网格平均包含数据点  $N\delta^d$ . 由于最左下角网格中包含数据点, 因此  $CL_{IR} = \{C_{x1,x2,\dots,xd} \mid \text{至少有一个 } x_i \text{ 等于 } 0\}$ . 对于新到达点来说, 落在影响区域中的概率可以表示为  $P_{arriv} = \overline{CL}/CN$ , 即新到点有  $1-P_{arriv}$  的概率在  $O(1)$  时间内处理完毕(GICSC\_AP 算法第 1~2 行). GICSC\_AP 算法第 4 行可以通过逐个比较  $SK$  中点的方法在  $O(\overline{SK})$  时间内完成. 当新点为 Skyline 点时, 需要执行第 5~19 行的操作. 由于数据服从均匀分布, 因此新到点或失效点为 Skyline 点的概率可以近似表示为  $P_{sky} = \overline{SK}/N$ . 用  $c_{SK}$  表示更新  $SK$  数据结构所用的平均代价, 则第 5~8 行最多可在  $O(c_{SK} \cdot (\overline{SK}+1))$  时间内完成; 假设判断某网格是否在  $CL_{IR}$  中(第 9 行)所需时间为  $c_{CL\_search}$ , 更新  $CL_{IR}$  结构(第 18 行)的代价为  $c_{CL}$ , 由于队列  $Q$  中的网格个数最多不会超过  $\overline{CL}$ , 且每个网格出入队列  $Q$  的操作可在常数时间内完成, 因此 GICSC\_AP 第算法第 9~19 行所需的时间最多不会超过  $c_{CL\_search} + \overline{CL} \cdot (c_{CL} + d \cdot c_{CL\_search})$ . 综上分析得到 GICSC 算法在数据均匀分布假设下处理一个新到点的平均时间为

$$O(P_{arriv} \cdot (\overline{SK} + P_{sky} \cdot (c_{SK}(\overline{SK}+1) + c_{CL\_search} + \overline{CL} \cdot (c_{CL} + d \cdot c_{CL\_search}))) + (1-P_{arriv})) \quad (2)$$

类似地, 可以分析得到 GICSC\_DP 算法处理旧点失效的平均时间复杂度. 第 7 行查找并删除数据点所需的时间最多为  $O(N\delta^d)$ . 第 2 行判断失效点是否为 Skyline 点用  $c_{SK\_search}$  表示, 失效点以  $P_{sky}$  的概率为 Skyline 点, 需要执行第 3~6 行的操作. 第 3 行的代价为  $c_{SK}$ , 第 6 行(GICSC\_CM 算法的第 5~14 行)循环次数不会超过  $\overline{CL}$ . 在每次循环中, GICSC\_CM 算法的第 6~10 行需要的代价为  $O(d)$ , 第 13 行需要的代价为  $c_{CL}$ , 而第 11~12 行可通过将网格内的每个数据点与  $SK$  中数据点逐个比较的方法在  $O(\overline{SK} \cdot N\delta^d)$  内完成. 综上得到 GICSC 算法在数据均匀分布假设下处理一个旧点失效的平均时间为

$$O(c_{SK\_search} + N\delta^d + P_{sky} \cdot \overline{CL} \cdot (d + \overline{SK} \cdot N\delta^d + c_{CL})) \quad (3)$$

① 通过简化假设, 大体观察参数对性能的影响.

## 6 实验评测

### 6.1 实验设置

在一台配置为 PIV 2.8GHz、1GB 内存、Windows XP 的 PC 机上,选择 Delphi 实现了一个模拟仿真系统,对以下算法进行了对比验证:(1) BBS<sup>[5]</sup>:在每次更新发生后重新计算全部数据集的 Skyline;(2) L\_BCSC:BCSC 算法的具体实现,其中 *SK*、*RE* 等均采用链表结构;(3) L\_GICSC:GICSC 算法的具体实现,其中 *SK*、*CL<sub>IR</sub>*、*PL* 等均采用链表结构;(4) L\_Lazy 和(5) L\_Eager;文献[13]两个算法框架采用链表数据结构的具体实现<sup>①</sup>.后三种实现中均采用 BNL 方法计算某些数据子集的 Skyline(如 BCSC\_DP 算法第 5 行).由于文献[14]主要考虑 *n*-of-*N* 查询,即侧重于考虑不同查询间的计算共享,以便能够同时跟踪多个查询或在 *n* 发生变化时也可以快速得到结果,与本文及文献[13]侧重点不同,因此没有对其方法进行比较;而 CSQ 方法<sup>[15]</sup>将所有动态属性综合为“距离”参数,在本文场景下蜕变为 NN 问题,因此也没有对之进行比较.

实验在广泛使用的 3 种模拟数据上进行:(1) 正相关数据 *corr*:若对象某属性值大/小,则其它属性

值也很可能大/小;(2) 独立数据 *indep*:对象各属性值相互独立;(3) 反相关数据 *anti*:若对象某属性值大/小,则其它属性值很可能小/大.数据分布情况如图 10 所示;表 1 显示了实验数据 Skyline 集合的平均大小情况.每种类型的数据均产生 1000000 个数据点,持续到达用来模拟数据流场景;系统保留 *N* 个数据点,模拟 *N* 个被监控对象;实验中不考虑热点情况,即新到达数据随机替换 *N* 个数据中的一个,因此增删操作是成对出现的.

表 1 实验数据集 Skyline 集合平均大小

(a) 随维度 <i>d</i> 变化的情况 ( <i>N</i> =100000)			
<i>d</i>	正相关数据	独立数据	反相关数据
2	4	12	31
3	9	74	447
4	24	289	3602
5	38	964	17849
6	66	2348	49665
7	81	5261	82358

(b) 随 <i>N</i> 变化的情况 ( <i>d</i> =3)			
<i>N</i>	正相关数据	独立数据	反相关数据
10000	7	55	242
50000	10	67	356
100000	12	76	336
150000	13	82	403
200000	13	73	395
250000	20	72	451

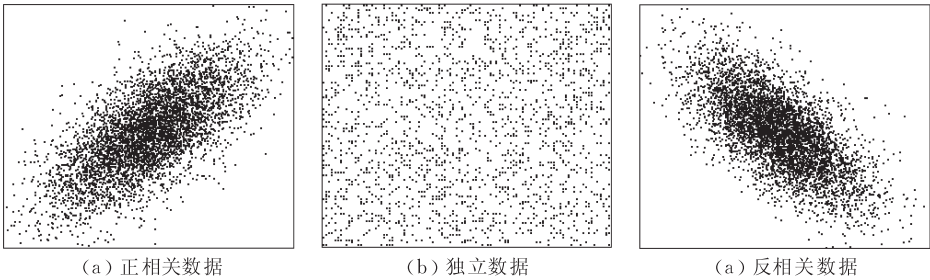


图 10 实验所用 3 种数据分布示意

### 6.2 实验结果

图 11 是实验中 BBS 算法的平均性能,其中图 11(a)示意了 *N*=10000 时平均处理时间随维度的变化情况,图 11(b)为 *d*=3 时平均处理时间随 *N* 的变化情况.可见对正相关数据处理效率最高,反相关数据效果最差,独立数据介于二者之间.具体来说,对正相关数据,在 *d*<7 时每秒可以处理的数据更新达到  $10^2 \sim 10^3$  级别;但对独立和反相关数据,算法的效率随着 *d* 的增加急剧下降,尤其当 *d*>3 时每秒处理数据变化不到  $10^2$  个;算法对 *N* 增大也很敏感,尤其是对反相关数据,当 *N*>100000 时,每秒处理数据仅为  $10^1$  数量级甚至以下.因此总体来说,BBS 算法无法满足数据流环境下的速度要求.

图 12 为实验中 L\_BCSC 算法平均性能示意图,与 BBS 算法类似,对正相关数据处理效率最高,反相关数据效果最差.与 BBS 算法相比,L\_BCSC 算法总体性能提高了一个数量级,而且随 *d* 和 *N* 变化的趋势也变得更为缓和.性能提高主要是因为数据集发生变化时,L\_BCSC 算法无需对所有点重新计算,而只需计算数据变化引起的原有 Skyline 集的变化情况.

网格宽度  $\delta$  是影响 GICSC 算法效率的重要参数. $\delta$  越小,用网格表示的近似影响区域就越准确,

① 虽然采用索引结构可以提高算法效率,但考虑到实现难易程度,所有算法均采用简单链表数据结构实现,以进行公平比较.

从而导致更多的数据点落入自由区域,降低运算规模效果更显著;但 $\delta$ 变小会导致网格数目增多,增大网格处理开销和存储开销.反之, $\delta$ 越大,由网格引起的额外开销会变少,但其对降低运算规模的效果也变差,而且网格内包含的点增多导致 $PL$ 变长.极端情况下若整个空间只包括一个网格,则GICSC算

法蜕变为BNL算法.图13为 $d=2,N=10000$ 情景下L\_GICSC算法性能受网格宽度影响的情况,其中横坐标为每维划分的网格数.可见总体来说,随着每维网格数目的增多,L\_GICSC对3种数据的处理能力均先变好后又变差,与上述分析一致.

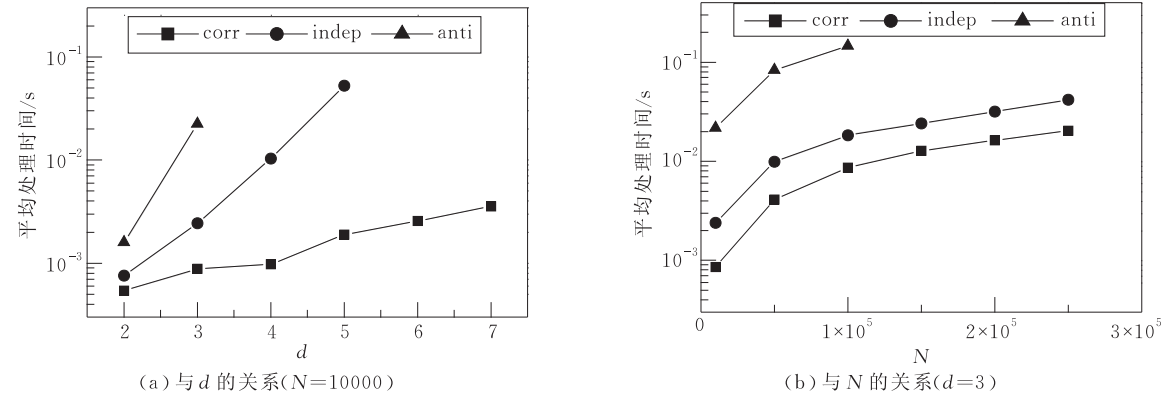


图 11 BBS 算法性能

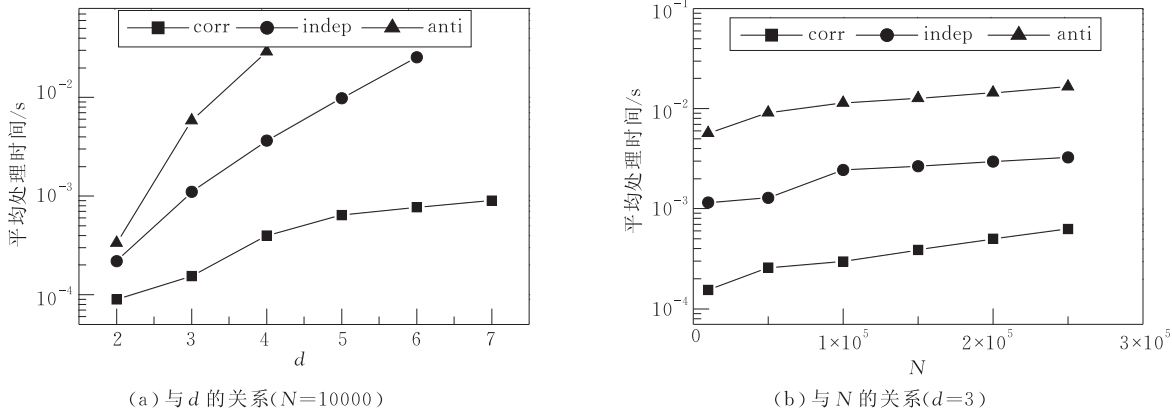


图 12 L\_BCSC 算法性能

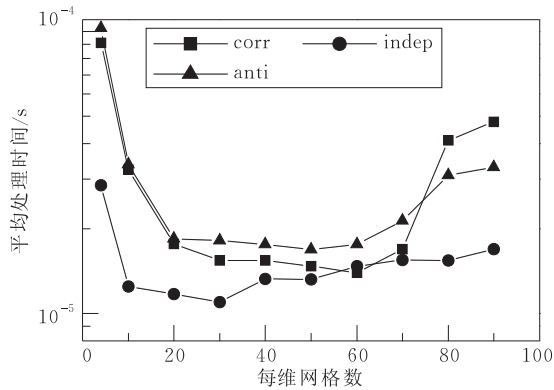


图 13 L\_GICSC 算法性能与每维网格数关系( $d=2,N=10000$ )

值得注意的是,L\_GICSC 算法对 3 种不同分布数据的处理能力差距变小.分析原因,GICSC 算法性能提高主要体现在发生在自由区域中的数据变

化;而数据分布不同导致 $CL_{IR}$ 中包含的网格数存在差异,独立数据最小,因此处理效率提高多一些;另一方面,当数据正相关或反相关分布时,空间中的许多网格没有包含任何数据点,降低了网格处理的规模.表 2 列出了 $d=2,N=10000$ 时不同数据分布情况下非空网格和 $CL_{IR}$ 包含网格所占比例情况.

表 2 数据分布对 GICSC 算法参数的影响( $d=2,N=10000$ )

每维网格	非空网格/%			$CL_{IR}$ 包含网格/%		
	corr	indep	anti	corr	indep	anti
10	63.00	100	66.00	19.00	19.00	34.00
20	54.75	100	53.75	9.75	9.75	34.25
30	40.44	100	42.78	14.56	6.56	30.22
40	41.94	100	42.13	11.00	4.94	28.75
50	36.56	100	36.52	10.32	3.96	31.56
60	41.31	100	32.39	4.78	3.31	33.25
70	36.29	100	31.33	9.31	2.84	32.37
80	32.20	100	29.70	8.17	2.48	33.56
90	33.64	100	32.83	5.88	2.21	26.88

图 14(a)为 L\_GICSC 算法性能随  $d$  的变化情况<sup>①</sup>. 可见, 当  $d$  较小时 L\_GICSC 算法具有非常好的运算效率, 对 3 种类型数据都能达到每秒  $10^4 \sim 10^5$  数量级的处理速度, 与其它方法相比具有明显优势; 但当  $d$  较大时 ( $d > 4$ ), 优势不再明显. 这是因为网格数目随维度增加呈指数级增长, 处理网格所需的额外开销随着  $d$  的增大迅速增长; 而减少每维网格数则会导致自由区域相对变小, 降低数据规模

效果变差, 因此整个算法性能相对下降. 图 14(b)说明, 随着  $N$  的增大, 算法的平均处理时间并没有呈现明显的增长趋势. 这是因为 L\_GICSC 算法所需时间主要取决于发生在影响区域中的数据变化. 由表 2 可以看出, 影响区域一般小于自由区域, 因此当  $N$  增大时, 大量数据变化发生在自由区域中, 导致算法平均处理时间并没有随着  $N$  的增大而明显增长.

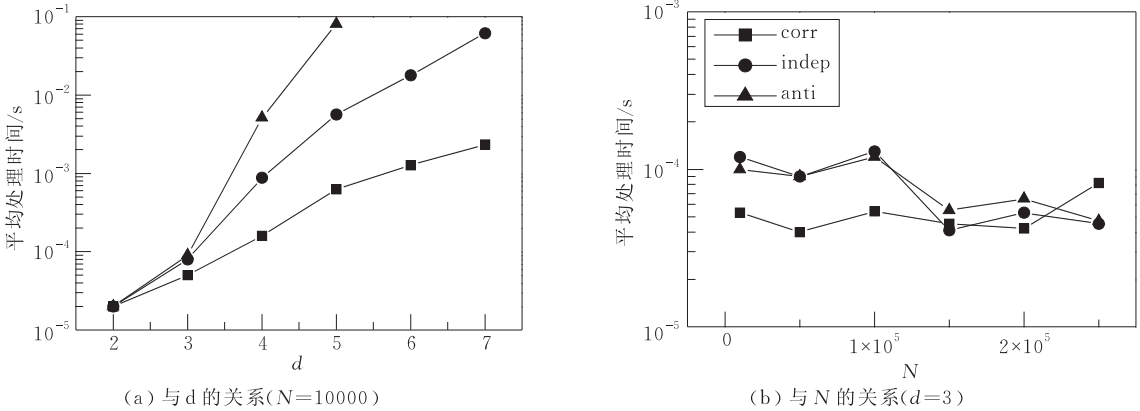


图 14 L\_GICSC 算法性能( $\delta$ 分别取最优值)

以独立分布数据为例, 不同算法所需的存储开销情况比较如图 15 所示. 总体来说 L\_BCSC 仅需维护有限的  $SK, RE$  等数据结构, 因此所需空间最小; L\_GICSC 由于需要维护网格索引的数据结构, 所需空间最大; BBS 所需空间介于两者之间.

裁减, 因此其性能要低于 L\_Lazy, 与 L\_Eager 相当. 但 L\_GICSC 算法的性能与其它方法相比有较为明显的优势. 因此, GICSC 也可有效用于滑动窗口数据流模型.

6.3 实验结论

在不满足滑动窗口模型的数据流场景中连续计算 Skyline 集合, 可以在每次数据更新后采用现成算法重新计算最新结果, 该方法非常低效, 且对维度和数据规模的增大非常敏感, 难以满足数据流场景下对运算效率的要求; 本文提出的 BCSC 方法能够满足流速较快场景下连续 Skyline 计算需求, 根据数据分布以及维度和数据规模不同, 平均每秒可以处理的数据更新达到  $10^2 \sim 10^4$  数量级; GICSC 算法的性能主要取决于网格宽度的选择, 在网格划分合适的情况下, GICSC 算法的性能比 BCSC 算法还要高出 1 个数量级, 尤其在维度较低、数据规模较大场景下 GICSC 算法优势更为明显.

BCSC 和 GICSC 算法同样可以用于滑动窗口数据流场景. 由于没有对数据进行丢弃裁减, 因此 BCSC 算法性能低于文献[13]提出的 Eager 和 Lazy 算法; 但实验表明 GICSC 算法性能与已有算法相比仍有一定优势.

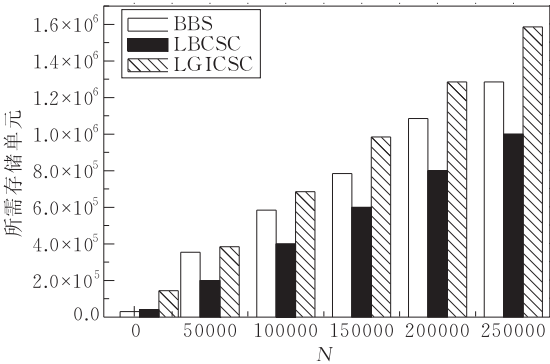


图 15 存储开销比较(indep 数据,  $d=3$ ,  $\delta=1/30$ )

BCSC 和 GICSC 算法适应于按任意顺序增删的数据流, 当然也能够用于滑动窗口数据流上的连续 Skyline 计算. 本节最后在独立分布的数据上, 以基于时间的滑动窗口数据流为场景, 对本文提出的方法和 L\_Lazy 及 L\_Eager 进行比较, 如附图 1 所示.

可以看出, 总体来说 L\_Lazy 要优于 L\_Eager, 与文献[13]结论一致. L\_BCSC 算法实际上是 L\_Lazy 的变种, 但是它没有根据滑动窗口特性进行

① 其中  $\delta$  随  $d$  及数据分布不同, 分别取不同的值, 以取得较好的计算性能.



## 7 结束语

为了在不满足滑动窗口模型的数据流上进行连续 Skyline 计算,首先基于已有工作提出了基本算法 BCSC,然后基于影响区域的观察,根据网格索引数据结构提出了 GICSC 算法.理论分析和实验结果证明了所提方法在随机增删数据流上连续跟踪 Skyline 集合的有效性.与已有方法相比,GICSC 算法在滑动窗口数据流场景下仍有一定优势.

本文工作还可以从以下两个方面继续深入:(1)深入分析网格宽度对 GICSC 算法性能的影响,并对最优宽度选择给出理论上的证明和指导;(2)在非均匀分布的数据中,空间中包含大量的空网格,如何选择合适的数据结构,避免对大量空网格的频繁查询处理,是 GICSC 算法改进提高的一个重要方面.

## 参 考 文 献

- [1] Borzsonyi S, Kossmann D, Stocker K. The skyline operator//Proceedings of the International Conference on Data Engineering (ICDE). Heidelberg, Germany, 2001: 421-430
- [2] Chomicki J, Godfrey P, Gryz J, Liang D. Skyline with pre-sorting//Proceedings of the International Conference on Data Engineering (ICDE). Bangalore, India, 2003: 717-719
- [3] Tan K L, Eng P K, Ooi B C. Efficient progressive skyline computation//Proceedings of the International Conference on Very Large Data Bases (VLDB). Roma, Italy, 2001: 301-310
- [4] Kossmann D, Ramsak F, Rost S. Shooting stars in the sky: An online algorithm for skyline queries//Proceedings of the International Conference on Very Large Data Bases (VLDB). Hong Kong, China, 2002: 275-286
- [5] Papadias D, Tao Y. Progressive skyline computation in database systems. ACM Transactions on Database Systems (TODS), 2005, 30(1): 41-82
- [6] Liu Xin, Yu Jing, Liu Guo-Hua. Algorithm for skyline queries based on window query. Journal of Yanshan University, 2005, 29(5): 398-402(in Chinese)  
(刘欣,余靖,刘国华.基于窗口查询的轮廓查询算法.燕山大学学报,2005,29(5):398-402)
- [7] Gao Yun-Jun, Chen Gen-Cai, Chen Ling, Chen Chun. A memory-optimal branch-and-bound algorithm for skyline queries//Proceedings of the NDBC 2006. Guangzhou, 2006: 526-533(in Chinese)  
(高云君,陈根才,陈岭,陈纯.一种存储最佳的分支界限轮廓查询算法//全国数据库学术会议(NDBC).广州,2006:526-533)

- [8] Balke W T, Gützer U, Zheng J X. Efficient distributed skyline for web information systems//Proceedings of the International Conference on Extending Database Technology (EDBT). Heraklio,Greece, 2004: 256-273
- [9] Wu P, Zhang C, Feng Y. Skyline queries for scalable distribution//Proceedings of the International Conference on Extending Database Technology (EDBT). Munich, Germany, 2006: 112-130
- [10] Hose K. Processing skyline queries in P2P systems//Proceedings of the VLDB 2005 PhD Workshop. Trondheim, Norway, 2005: 36-40
- [11] Deng Bo, Jia Yan, Yang Shu-Qiang. An efficient algorithm for distributed Skyline queries. Computer Engineering and Science, 2007, 29(9): 97-100(in Chinese)  
(邓波,贾焰,杨树强.一种高效的分布式 Skyline 查询算法.计算机工程与科学,2007,29(9):97-100)
- [12] Babcock B, Babu S, Datar M, Motwani R, Widom J. Models and issues in data stream systems//Proceedings of the ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS). Madison, Wisconsin, 2002: 1-16
- [13] Tao Y, Papadias D. Maintaining sliding window skylines on data streams. IEEE Transactions on Knowledge and Data Engineering, 2006, 18(3): 377-391
- [14] Lin X, Yuan Y, Wang W, Lu H. Stabbing the sky: Efficient skyline computation over sliding windows//Proceedings of the International Conference on Data Engineering (ICDE). Tokyo, Japan, 2005: 502-513
- [15] Huang Z, Lu H, Ooi B C, Tung A K H. Continuous skyline queries for moving objects. IEEE Transactions on Knowledge and Data Engineering, 2006, 18(12): 1645-1658
- [16] Pei J, Jin W, Ester M, Tao Y. Catching the best views of skyline: A semantic approach based on decisive subspaces//Proceedings of the International Conference on Very Large Data Bases (VLDB). Trondheim, Norway, 2005: 253-264
- [17] Yuan Y, Lin X, Liu Q, Wang W, Yu J X, Zhang Q. Efficient computation of the skyline cube//Proceedings of the International Conference on Very Large Data Bases (VLDB). Trondheim, Norway, 2005: 241-252
- [18] Tao Y, Xiao X, Pei J. SUBSKY: Efficient computation of Skylines in subspaces//Proceedings of the International Conference on Data Engineering (ICDE). Atlanta, USA, 2006: 65-75
- [19] Pei J, Fu A W-C, Lin X, Wang H. Computing compressed multidimensional skyline cubes efficiently//Proceedings of the International Conference on Data Engineering (ICDE). Istanbul, Turkey, 2007: 96-105
- [20] Vlachou A, Doukeridis C, Vazirgiannis M, Kotidis Y. SKYPEER: Efficient subspace skyline computation over distributed data//Proceedings of the International Conference on Data Engineering (ICDE). Istanbul, Turkey, 2007: 416-425

[21] Wang S, Ooi B C, Tung A K H, Xu L. Efficient skyline query processing on peer-to-peer networks//Proceedings of the International Conference on Data Engineering (ICDE). Istanbul, Turkey, 2007: 1126-1135

[22] Sharifzadeh M, Shahabi C. The spatial skyline queries//Proceedings of the International Conference on Very Large Data Bases (VLDB). Seoul, Korea, 2006: 751-762

[23] Deng K, Zhou X, Shen H T. Multi-source skyline query processing in road networks//Proceedings of the International Conference on Data Engineering (ICDE). Istanbul, Turkey, 2007: 796-805

[24] Dellis E, Seeger B. Efficient computation of reverse skyline queries//Proceedings of the International Conference on Very Large Data Bases (VLDB). Vienna, Austria, 2007: 291-302

[25] Pei J, Jiang B, Lin X, Yuan Y. Probabilistic skylines on uncertain data//Proceedings of the International Conference on Very Large Data Bases (VLDB). Vienna, Austria, 2007: 15-26

[26] Chan C Y, Jagadish H V, Tan K L. Finding  $k$ -dominant skylines in high dimensional space//Proceedings of the ACM SIGMOD International Conference on Management of data. Chicago, Illinois, 2006: 503-514

[27] Chan C Y, Jagadish H V, Tan K L, Tung A K H, Zhang Z. On high dimensional skylines//Proceedings of the International Conference on Extending Database Technology (EDBT). Munich, Germany, 2006: 478-495

[28] Zhang Z, Guo X, Lu H, Tung A K H, Wang N. Discovering strong skyline points in high dimensional spaces//Proceedings of the International Conference on Information and Knowledge Management (CIKM). Bremen, Germany, 2005: 247-248

[29] Koltun V, Papadimitriou C H. Approximately dominating representatives. Theoretical Computer Science, 2007, 371 (3): 148-154

[30] Lin X, Yuan Y, Zhang Q, Zhang Y. Selecting stars: The  $k$  most representative skyline operator//Proceedings of the International Conference on Data Engineering (ICDE). Istanbul, Turkey, 2007: 86-95

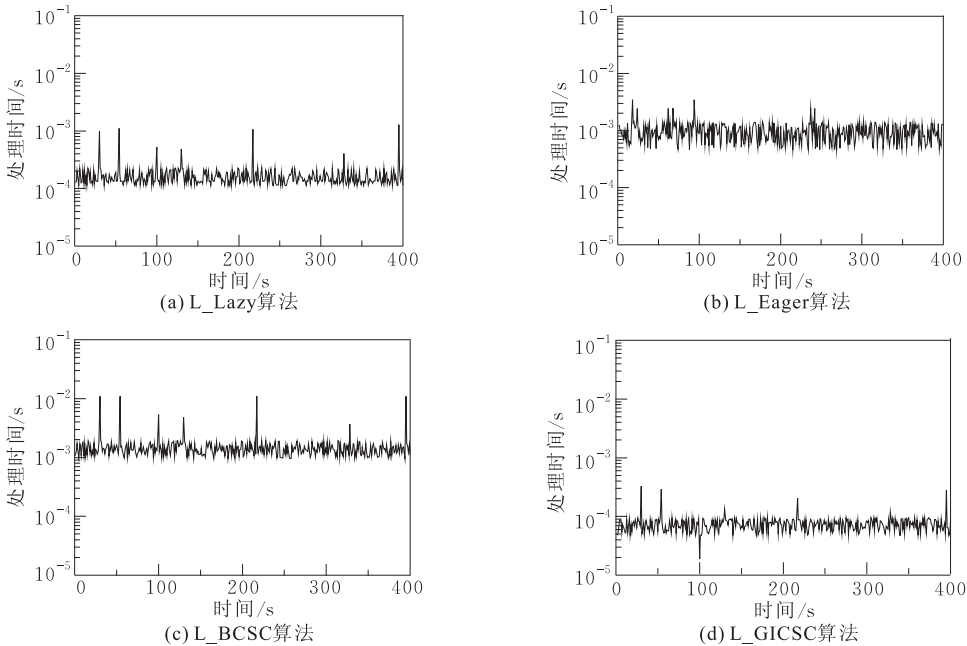
[31] Morse M, Patel I, Jagadi H V. Efficient skyline computation over low-cardinality domains//Proceedings of the International Conference on Very Large Data Bases (VLDB). Vienna, Austria, 2007: 267-278

[32] Lee K, Zheng B, Li H, Lee W C. Approaching the skyline in  $Z$  order//Proceedings of the International Conference on Very Large Data Bases (VLDB). Vienna, Austria, 2007: 279-290

[33] Yu X, Pu K Q, Koudas N. Monitoring  $k$ -nearest neighbor queries over moving objects//Proceedings of the International Conference on Data Engineering (ICDE). Tokyo, Japan, 2005: 631-642

[34] Mouratidis K, Bakiras S, Papadias D. Continuous monitoring of top- $k$  queries over sliding windows//Proceedings of the ACM SIGMOD International Conference on Management of Data. Chicago, Illinois, 2006: 635-646

[35] Orenstein J A, Merrett T H. A class of data structures for associative searching//Proceedings of the ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS). Waterloo, Ontario, Canada, 1984: 181-190



附图 1 滑动窗口数据流上的性能比较(indep 数据,  $d=3$ ,  $N=10000$ ,  $\delta=1/15$ )



**TIAN Li**, born in 1980, Ph.D. candidate. His current research interests include distributed computing, data stream processing and data mining.

**ZOU Peng**, born in 1957, professor. His main research

interests include distributed computing and advanced operation systems.

**LI Ai-Ping**, born in 1974, Ph.D.. His current research interests include artificial intelligence, database and distributed computing.

**JIA Yan**, born in 1960, professor. Her current research interests include database and distributed computing.

**Background**

Skyline and other directly related problems, such as multi-objective optimization and maximum vectors, can be found in a wide spectrum of optimization applications and have been extensively studied. Recently a new class of data-intensive applications which is modeled as data stream has been widely researched. We consider continuous-query based applications and address the problem of continuous Skyline computation on streams with random additions and deletions,

which is quite different from the sliding window streams scene and moving objects environment. A straightforward method called BCSC and a novel grid indexed based algorithm called GICSC are presented in this paper, which are designed specially for the high dynamic scenario and proved to be efficient and powerful. This work is supported by the National High-Tech Research and Development Plan of China ("863" plan) under Grant Nos. 2006AA01Z451, 2007AA01Z474.