应用于流数据的连续多维度广义轮廓查询

杨洋,李艳红*,彭亚威,肖梦
(中南民族大学 计算机科学学院,武汉 430074)

摘 要 轮廓运算符自提出以来引起了研究人员的极大兴趣,随后各种轮廓查询的变体不断涌现,其中包括流数据上的子空间轮廓查询.为研究针对实际应用中复杂数据维度的需求,提出了广义轮廓(Genl-Skyline)的概念,并结合现有变体进一步提出了连续多维度广义轮廓(CMGS)问题.为解决该问题,提出了倒排轮廓支配表(ISDT),引入了嵌套轮廓方案以最小化ISDT结构,以及提出了基于连续数据属性的强弱修剪策略用于数据集剪枝,同时还设计了伴生索引ISDT-BM以支持在ISDT上高效搜索CMGS结果.最后,广泛的对比实验验证了ISDT结构及相关算法在解决CMGS查询问题上的可行性和高效性.

关键词 轮廓查询;多维度轮廓;流数据;动态维护
中图分类号 TP399 **文献标志码** **文章编号** 1672-4321(2025)04-0546-14
doi:10.20056/j.cnki.ZNMDZK.20250414

Continuous Multi-dimensional Genl-Skyline query for streaming data

YANG Yang, LI Yanhong*, PENG Yawei, XIAO Meng
(College of Computer Science, South-Central Minzu University, Wuhan 430074, China)

Abstract The skyline operator has sparked great interest among researchers since its proposal, and subsequently various variants of skyline queries have emerged, including subspace skyline queries on streaming data. The concept of generalized skyline (Genl-Skyline) is proposed for the needs of complex data dimensions in practical applications, the Continuous Multi-dimensional Genl-Skyline (CMGS) problem is also proposed by combining existing variants. To address the problem, the study proposes the Inverted Skyline Dominance Table (ISDT), introduces a nested skyline scheme to minimize the ISDT structure, and proposes a strong-weak pruning strategy based on continuous data attributes for dataset pruning. It also designs an associated index ISDT-BM to support efficient search for CMGS results on ISDT. Finally, extensive experiments validate the feasibility and efficiency of the ISDT structure and related algorithms in solving CMGS queries.

Keywords skyline query; multi-dimensional skyline; streaming data; dynamic maintenance

在现有的轮廓查询研究中,查询数据集中元组的各个维度通常是实数,并且用户偏好被定义为越小越好.然而元组也可能包含字符串、枚举、日期或其他对象的嵌套.要对这些类型的值进行轮廓查询,则需要进行一些预处理,例如应用单调映射函数将它们映射为实数.为了使轮廓查询更加灵活、实用和通用,本文提出了广义轮廓(Genl-Skyline)的概念.除此之外,轮廓查询还有许多变体和应用,连续和多维度轮廓问题则是本文所关注的,其中:(1)多维轮廓允许用户自由选择他们需要参考的维度,而不是标准轮廓问题中的默认维度;(2)连续轮廓则将标准轮廓问题应用于动态数据集(如流数据集)而不是静态数据.这两个变体在一定程度上增加了轮廓问题的规模和解决难度.本文将广义轮廓与现有的多维度轮廓和连续轮廓相结合,提出了连续多维度广义轮廓,简称为CMGS(Continuous Multi-dimensional Genl-Skyline)问题.

设$\boldsymbol{R}$为一个实时餐厅推荐系统的关系表,其中每

收稿日期 2024-09-04
* **通信作者** 李艳红(1973-),女,教授,博士,研究方向:时空数据处理、网络大数据,E-mail:liyanhong@mail.scuec.edu.cn
基金项目 湖北省自然科学基金资助项目(2017CFB135);中央高校基本科研业务费专项资金资助项目(CZY23019)

一个元组包含3个维度:价格(p)、客容量(c)、评分(r),其中客容量为枚举常量IDLE(空闲)、MODERATE(一般)和BUSY(繁忙),评分的值为一个长度为3的数组,该数组的各个分量分别表示餐厅在味道、环境和服务这3个指标上的评分.表1展示了每个维度定义域的具体符号形式.除此之外,t_α和t_ω用于表示元组到达和过期的时间戳.表2展示了$\boldsymbol{R}$的实例,图1为该实例中元组到达和过期的时间表,其中绿色表示元组到达,红色表示元组过期.

表1　$\boldsymbol{R}$实例各个维度的定义域

Tab. 1　The domains of each dimension in $\boldsymbol{R}$ instance

维度	定义域
p	$\{x \mid x > 0\}$
c	{IDLE, MODERATE, BUSY}
r	$\{[x_0, x_1, x_2] \mid x_0, x_1, x_2 \in [0, 5]\}$

表2　$\boldsymbol{R}$实例

Tab. 2　$\boldsymbol{R}$ instance

元组	价格(p)	客容量(c)	评分(r)	到达时间(t_α)	过期时间(t_ω)
r_0	60	IDLE	[4.0, 3.0, 2.0]	1	5
r_1	70	BUSY	[4.7, 4.3, 4.5]	2	4
r_2	180	IDLE	[5.0, 5.0, 5.0]	3	9
r_3	190	MODERATE	[2.0, 5.0, 5.0]	5	6
r_4	110	MODERATE	[3.0, 3.0, 4.5]	6	8
r_5	75	IDLE	[4.0, 4.0, 4.0]	8	9

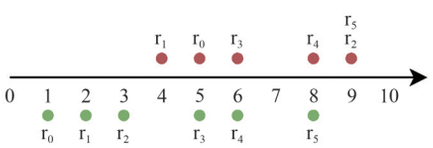

图1　$\boldsymbol{R}$实例的元组时间表

Fig. 1　The schedule of tuples in $\boldsymbol{R}$ instance

不失一般性地假设,对于维度p,用户更倾向于较低的餐厅.对于维度c,用户倾向于选择IDEL的餐厅,其次是MODERATE和BUSY.对于维度r,口味是餐厅最重要的因素,其次是环境,最后是服务,即对3个分量依次比较,直到出现两者不相同,取较大值的一方.假设当前时间戳为2,用户认为评分(r)没有参考价值,其仅考虑价格(p)和客容量(c)并希望在便宜且客容量为IDLE的餐厅就餐.此时,该系统推荐给用户的是餐厅r_0,因为在当前时间戳下,只有两家有效餐厅r_0和r_1,且r_0在维度p和c上支配r_1.因此,r_0是在时间戳2时,$\boldsymbol{R}$实例关于子空间$\boldsymbol{\Phi} = \{p, c\}$的CMGS结果,即$\mathrm{Sky}_2(\boldsymbol{\Phi}) = \{r_0\}$.

CMGS问题还可以被应用于更广泛的场景.设想一个数据分析系统,无论是用于股票数据、车辆数据还是社交媒体数据,这些数据集通常都很复杂且是连续的,每个数据都有不同的到达时间和过期时间,用户需要从数据分析系统中获得数据集的实时轮廓集合.CMGS正是为以下挑战而设计的:1)每个维度的值从实数扩展到任意数据类型,处理这些数据的轮廓需要重新定义;2)系统中的元组是连续且庞大的,需要建立一种新的索引结构来高效组织元组信息,并以最低成本维护动态更新;3)每个元组具有不同的生命周期,先到达的元组可能会比后到达的元组更晚过期.因此,传统基于队列的模式无法处理元组的到达和过期,大大增加了查询处理的复杂性;4)在确保结果准确性的同时,还需要考虑效率,这对于流数据的查询处理至关重要.本文的主要贡献如下:

(1)本文提出了广义轮廓的思想,并将连续多维度广义轮廓(CMGS)问题公理化,使其可以应用于更广泛的场景;

(2)设计了一个名为倒排轮廓支配表(ISDT, Inverted Skyline Dominance Table)的数据结构,其是解决CMGS问题的核心组件.ISDT索引能够动态管理带有时间戳的元组之间复杂的子空间支配关系.通过提出和应用多种最小化和剪枝策略,确保了ISDT索引的效率.同步维护的基于位图的结构ISDT-BM(BitmapforISDT)可以通过位运算快速从ISDT中搜索与给定子空间对应的CMGS结果;

(3)在3种合成数据集和多个真实数据集上进行了广泛的对比实验,证明了ISDT索引以及相关算法在解决CMGS问题中的可行性、完备性、兼容性和有效性.

1　相关工作

BORZSONY等[1]首次引入了轮廓算子以扩展数据库系统,并提出了BNL算法,该算法采用了一种直观的处理方法,但使用了内存块来维护有限数量的数据对象.此后,一系列基础的轮廓查询算法被提出,如SFS算法[2]、SaLSa算法[3]、DC轮廓算法[4]、位图算法[5]、NN算法[6]以及BBS算法[7].随后,LEE等提出了BSkyTree算法[8-9],该算法通过基于枢轴点划分数据来提高评估支配性或不可比性的效率.最近,一些前沿的轮廓问题的相关工作被相继提出,如不完整数据流上的轮廓查询[10]、基于位置的轮廓[11-12]、大数据中的轮廓处理[13-14]、空间文本轮廓查

询[15-16]、数据联合体上的轮廓查询[17]等等. HAN等[18]提出了一种新颖的GPR算法,该算法基于预排序和重用原则,能够在大规模数据上高效计算G-Skyline群体. 李光辉等[19]基于正交查询范围来回答G-Skyline查询中的why-not问题,讨论了G-Skyline查询中产生why-not问题的原因,概述了如何修改why-not点和正交查询范围,使基于正交范围的G-Skyline查询的候选点集中包含why-not点.

标准轮廓算法专门用于计算静态数据集上的轮廓,而不是处理动态数据. LIN等[20]探讨了在数据流中面临快速数据更新时进行在线计算的领域,并使用了滑动窗口模型[21]. TAO和PAPADIAS[22]提出了数据流环境中的轮廓计算,他们还设计了Lazy和Eager算法来持续监控新到的数据对象并增量维护轮廓. LIN等[23]研究了n-of-N模型中的在线轮廓计算. SUN等[24]提出了BOCS算法,以解决分布式数据流上的轮廓相关问题. MORSE等[25]提出了一个名为连续时间区间轮廓的新轮廓操作符用于持续维护子空间轮廓,并提出了LookOut算法来解决该轮廓问题. 受数据流上子空间轮廓查询问题的启发,ALAMI和MAABOUT提出了一个MSSD(流数据上的多维度轮廓)框架[26],该框架包含3个数据结构,即缓冲区B、数据集R和一个名为NSCt的索引结构,用于存储每个对象在其生命周期内的支配子空间.MSSD结构还采用了缓冲策略. 数据流的源头每单位时间发出一个对象,每当B收集到k个对象时,它们会被插入到R中,并且过期的对象会被丢弃. 同时,如果有查询请求,NSCt会被调用以获取轮廓结果.

2 问题定义

2.1 偏好函数

定义1 (偏好函数) 设$R=\{r_0,\cdots,r_{n-1}\}$为一个在m-维空间上包含n个元组的数据集,其中r_i的第j个维度上的分量被记作$r_i[d_j]$,d_j的定义域被记作$\Gamma(d_j)$,即有$r_i=\{r_i[d_0],\cdots,r_i[d_{m-1}]\}$,其中$0\leqslant i<n$,$0\leqslant j<m$. 全空间$\Phi^+=\{d_0,\cdots,d_{m-1}\}$表示关于数据集的整个$m$-维空间中的全部维度集合;子空间$\Phi\subseteq\Phi^+$表示全空间的一个非空子集. 设$r_x$和$r_y$为两个$R$中的元组,定义偏好函数$f_d(r_x,r_y)$是一种用于判断$r_x$和$r_y$的序关系的符号函数(返回值仅有$-1$,0和1),返回值1表示$r_x$在关于$\Gamma(d)$的序关系上前驱$r_y$,$-1$表示$r_y$在关于$\Gamma(d)$的序关系上前驱$r_x$. 为了满足轮廓问题中的支配定义,偏好函数$f_d$需满足以下特性:

(1)反自反性. $(\forall\alpha\in\Gamma(d))\neg(f_d(\alpha,\alpha)\neq 0)$;

(2)非对称性. $(\forall\alpha,\beta\in\Gamma(d))f_d(\alpha,\beta)=x\Rightarrow f_d(\beta,\alpha)=-x$;

(3)传递性. $(\forall\alpha,\beta,\gamma\in\Gamma(d))(f_d(\alpha,\beta)=x\wedge f_d(\beta,\gamma)=x)\Rightarrow f_d(\alpha,\gamma)=x$;

(4)连接性. $(\forall\alpha,\beta\in\Gamma(d))f_d(\alpha,\beta)\neq 0\Rightarrow(f_d(\alpha,\beta)=1\vee f_d(\alpha,\beta)=-1)$.

偏好函数f_d可以由系统(如餐厅推荐系统)自由实现其具体计算过程,被考虑最多的"越小越好"策略以及前文示例中的维度c和维度r的比较策略均是其中一种.

说明1 连接性指定偏好函数应该作用于整个域,这意味着$\Gamma(d)$应该关于偏好函数构成严格全序集. 这个条件的提出是为了遵循传统的支配定义,因为其通常仅考虑实数域上的"小于"(<)或"大于"(>)关系.

2.2 CMGS问题

定义2 (广义支配) 给定m-维空间上的两个元组$r_i,r_j\in R$. 当且仅当r_i和r_j在任意一个维度d上都有$f_d\geqslant 0$且r_i和r_j在至少一个维度d'上有$f_{d'}>0$满足时,则称r_i支配r_j,记作$r_i\prec r_j$.

定义3 (连续多维度广义轮廓(CMGS)) 设$\mathcal{R}=\{r_0,\cdots\}$为一个连续数据集,$t_c$为当前时间戳. $\mathcal{R}$中的每个元组都有两个时间维度t_α和t_ω,分别表示元组的到达时间和过期时间. $\mathcal{R}$的CMGS结果记作$\mathrm{Sky}_{t_c}(\mathcal{R},\Phi)$,包括了所有在子空间$\Phi$上不被$\mathcal{R}_c$中其他元组支配的元组,即$\mathrm{Sky}_{t_c}(\mathcal{R},\Phi)=\{r_c|r_c\in\mathcal{R}_c\wedge\neg\exists r'_c\in\mathcal{R}_c\wedge r'_c\prec_\Phi r_c\}$,其中$\mathcal{R}_c$是$\mathcal{R}$中的有效元组集合,即$t_\alpha$小于或等于$t_c$且$t_\omega$大于$t_c$的元组集合,即$R_c=\{r|r\in\mathcal{R}\wedge t_\alpha(r)\leqslant t_c\wedge t_\omega(r)>t_c\}$, $\mathrm{Sky}_{t_c}(\mathcal{R},\Phi^+)$被缩写为$\mathrm{Sky}_{t_c}(\mathcal{R})$.

继$\boldsymbol{R}$实例,设当前时间戳为3,此时的有效元组为r_0,r_1和r_2,其关于每个维度的CMGS结果如表3所示.

表3 $\boldsymbol{R}$实例的CMGS结果

Tab. 3 The CMGS results of $\boldsymbol{R}$ instance

子空间	CMGS结果	子空间	CMGS结果
$\{p\}$	$\{r_0\}$	$\{p,r\}$	$\{r_0,r_1,r_2\}$
$\{c\}$	$\{r_0,r_2\}$	$\{c,r\}$	$\{r_2\}$
$\{r\}$	$\{r_2\}$	$\{p,c,r\}$	$\{r_0,r_1,r_2\}$
$\{p,c\}$	$\{r_0\}$		

2.3 问题声明

设$\mathcal{R}$为一个由无穷序列的m-维元组r组成的连续数据，t_c为当前时间戳，设q_Φ为用户发起的关于子空间Φ的CMGS查询.本文需要解决的问题是如何高效地返回与q_Φ对应的CMGS结果$\mathrm{Sky}_{t_c}(\Phi)$.

3 ISDT结构

本节将详细介绍倒排轮廓支配表(ISDT)，其用于维护元组的多维度支配关系.具体来说，对于连续数据集$\mathcal{R}$中的每个m-维元组r，r可能在总共2^m-1个子空间中被支配.由于连续数据中每个元组的生命周期可能不同，还需要记录这些被支配的维度何时不再被支配.在介绍ISDT结构之前，先引入几个定义.

定义4 (支配子空间和支配子空间对(DSP))　设$r\in R$为一个元组，Φ为一个子空间.当且仅当$r\notin \mathrm{Sky}(R,\Phi)$时，$\Phi$是一个$r$的支配子空间.设$t$为$r$的过期时间.设$R_c$为$R$的子集，且$R_c$中的每个元组$r_c$有相同的过期时间$t'$且其在一个子空间$\Phi_c$上支配$r$.假设$S_\Phi$由全部的$\Phi_c$构成，称数据对$P=\langle t_\varepsilon,S_\Phi\rangle$是一个在$t_\varepsilon$时$r$的支配子空间对，其中$t_\varepsilon=\min(t',t)$.

定义5 (倒排轮廓支配表(ISDT))　设$\mathcal{R}_c$为$\mathcal{R}$中的有效元组.对于$\mathcal{R}_c$中的每个元组r，有一个称为支配哈希表(DHT)的结构$\mathrm{HT}_r=\{\langle t_\varepsilon,S_\Phi\rangle,\cdots\}$，其由$r$的全部DSP构成(DSP的时间戳作为键，子空间集合作为值).将集合$\{\mathrm{HT}_{r_c}|r_c\in\mathcal{R}_c\}$称为$\mathcal{R}$的ISDT.

$\boldsymbol{R}$实例在时间戳3时的ISDT结构如图2所示.其中r_1的过期时间在$\mathcal{R}_c$中最小，r_1的全部支配子空间中都被插入了一个时间戳为4的数据对，所以r_1的DSP结构与其他两个不同.

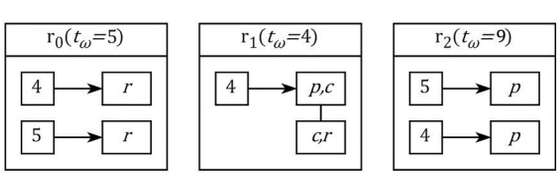

图2　$\boldsymbol{R}$实例在时间戳3时的ISDT结构

Fig. 2　The ISDT structure of $\boldsymbol{R}$ instance attimestamp 3

定理1　设$r\in\mathcal{R}_c$.当且仅当Φ不是HT_r中任何数据对中子空间集合中任何元组Φ'的子集，则r是关于子空间Φ的轮廓元组，即$r\in\mathrm{Sky}(\mathcal{R},\Phi)\Leftrightarrow\neg\exists\langle t,S_\Phi\rangle\in\mathrm{HT}_r(\exists\Phi'\in S_\Phi(\Phi\subset\Phi'))$.此外，如果$\exists\langle t,S_\Phi\rangle\in\mathrm{HT}_r(\exists\Phi'\in S_\Phi(\Phi\subset\Phi'))$，则子空间$\Phi$被称为$\mathrm{HT}_r$的一个项.

证明　分别证明其充分性和必要性.

必要性.假设有一个元组$r\in\mathrm{Sky}(\mathcal{R},\Phi)$，并且在$r$的$\mathrm{HT}_r$中有一个数据对$\langle t,S_\Phi\rangle$，其中包括$t$和$S_\Phi$，且$\Phi\in S_\Phi$.那么，必定存在另一个元组$r'$在子空间$\Phi$上支配$r$，即$r'\prec_\Phi r$.因此$r$不可能是关于$\Phi$的轮廓元组，即$r\notin\mathrm{Sky}(\mathcal{R},\Phi)$，这与之前的假设$r\in\mathrm{Sky}(\mathcal{R},\Phi)$矛盾.

充分性.假设在r的HT_r中不存在这样的数据对$\langle t,S_\Phi\rangle$，其包含t和S_Φ，且S_Φ中至少有一个子空间Φ'是Φ的超集.这个假设等价于假设在HT_r中的每一个$\langle t,S_\Phi\rangle$的子空间集合S_Φ中都不包含任何Φ'，且Φ'是Φ的超集.于是，没有其他元组r'在子空间Φ上支配r，即$\neg\exists r'(r'\in\mathcal{R}\wedge r'\prec_\Phi r)$.因此，$r$是$\mathcal{R}$的轮廓元组，即$r\in\mathrm{Sky}(\mathcal{R},\Phi)$.

4 CMGS处理框架

与大部分渐进维护结构类似，ISDT结构在解决CMGS问题时分为静态构建和动态维护，下文将详细介绍这两个部分，以及针对ISDT处理CMGS问题的进一步优化策略.

4.1 ISDT的构建

首先引入一个名为支配关系对的概念.

定义6 (支配关系对(DRP))　设r_i和r_j为两个m-维元组.$\langle\Phi_j,\Phi_i\rangle$被称为关于$r_i$和$r_j$的支配关系对，其中$r_i$仅在$\Phi_i$的每个维度上支配$r_j$，而$r_j$仅在$\Phi_j$的每个维度上支配$r_i$，即$\Phi_i=\{d|d\in\Phi^+,f_d(r_i,r_j)=1\}$，$\Phi_j=\{d|d\in\Phi^+,f_d(r_j,r_i)=1\}$.

ISDT-C算法.算法的伪代码如算法1所示，该算法输入一个元组集合R并初始化ISDT.首先，通过一个双层循环对集合中的元组不重复地成对进行处理(第2和第4行).算法获取一对元组r_i和r_j及其对应的HT_{r_i}和HT_{r_j}(第3和第5行).随后，r_i和r_j的DRP$\langle\Phi_j,\Phi_i\rangle$以及它们各自的过期时间$t_\omega(r_i)$和$t_\omega(r_j)$的最小值被计算并将DRP插入到相应的DHT中(第6-8行).最后，算法将依次构建好的成对DHT插入到ISDT结构并将其返回(第10和12行).

复杂度分析.设R中元组的数量为n，维度数量为m.该算法时间复杂度为$O(n^2)$.ISDT-C算法在构建ISDT的过程中，会始终在存储堆栈上占用空间以存储ISDT结构，DPR$\langle\Phi_j,\Phi_i\rangle$为两个子空间组成的

算法1 ISDT-C(R)

输入：数据集R

输出：构建后的ISDT

```
ISDT← ∅
for each r_i ∈ R
  HT_{r_i} ← r_i的DHT
  for eachr_j ∈ R, j > i
    HT_{r_j} ← r_j的DHT
    ⟨Φ_j, Φ_i⟩ ←关于r_i和r_j的DRP
    t_ε ← min(t_ω(r_i), t_ω(r_j))
    插入t_ε和Φ_j到HT_{r_i},插入t_ε和Φ_i到HT_{r_j}
  end for
  插入HT_{r_i}和HT_{r_j}到ISDT
end for
return ISDT
```

对,其中子空间$\boldsymbol{\Phi}_i$和$\boldsymbol{\Phi}_j$实际上为维度数组,因此在一般情况(不考虑r_i和r_j同一维度上的数据相同)下,有$\left|\boldsymbol{\Phi}_i\right|+\left|\boldsymbol{\Phi}_j\right|=m$. 该算法的空间复杂度为$O\left(n^2\times m\right)$.

4.2 ISDT的最小化

ISDT在构建后会存储重复信息,例如,在$\boldsymbol{R}$实例中,r_0有两个DSP,$P_0=\left\langle 4,\left\{\{r\}\right\}\right\rangle$和$P_1=\left\langle 5,\left\{\{r\}\right\}\right\rangle$,其中$P_0$不需要存储,因为$P_0$表示在时间戳4之前$r_0$不会成为关于$\{r\}$的任何子集的轮廓元组,而$P_1$表示在时间戳5之前,$r_0$不会成为关于$\{r\}$的任何子集的轮廓元组,$P_1$表达的信息能够覆盖$P_0$. 因此ISDT结构需要进一步精简,以存储尽可能少的信息,从而提高后续轮廓搜索的效率并减少内存使用.

4.2.1 最小ISDT

定义7 (最小ISDT) 设HT为一个DHT. 当且仅当满足以下条件时,则称HT是最小:(1)HT中任意一个DSP的子空间集合中的任意两个元素(子空间)互不覆盖;(2)HT中任意两个DSP各自的子空间集合中的任意两个子空间$\boldsymbol{\Phi}$和$\boldsymbol{\Phi}'$,若有$\boldsymbol{\Phi}$覆盖$\boldsymbol{\Phi}'$,则$\boldsymbol{\Phi}$对应DSP的时间戳小于$\boldsymbol{\Phi}'$对应DSP的时间戳. 当且仅当ISDT每个DHT都是最小的,则此时的ISDT为最小ISDT.

关于ISDT的最小化,本文将提出两种DHT的最小化策略,分别是贪心策略和嵌套轮廓策略. 这两种策略区别在于,贪心策略在每次插入DSP时都会最小化DHT,而嵌套轮廓策略则是在完成所有可能的新DSP插入后才对DHT进行最小化.

4.2.2 贪心策略

MinimizeISDT算法. 使用贪心策略的MinimizeISDT算法的伪代码见算法2. 该算法在DSP要放入DHT时调用,其输入为DHT HT和DSP$\left\langle\hat{t},\left\{\hat{\boldsymbol{\Phi}}\right\}\right\rangle$. 首先,算法访问HT中的每个DSP,以及每个DSP的子空间集S_{Φ}中的每个子空间$\boldsymbol{\Phi}$(第1-2行). 如果输入的$\hat{t}$小于或等于t,且输入的子空间$\hat{\boldsymbol{\Phi}}$是$\boldsymbol{\Phi}$的子集,则说明输入的DSP被HT中的现有DSP覆盖,因此算法直接返回(第3-5行). 如果输入的$\hat{t}$大于或等于t,且输入的子空间$\hat{\boldsymbol{\Phi}}$是$\boldsymbol{\Phi}$的超集,则现有子空间$\boldsymbol{\Phi}$被输入子空间$\hat{\boldsymbol{\Phi}}$覆盖,因此算法从S_{Φ}中移除$\boldsymbol{\Phi}$(第6-9行). 最后,算法从HT中移除子空间集为空的DSP(第9-11行)并返回最小化后的ISDT(第14行).

算法2 MinimizeISDT(HT, $\left\langle\hat{t},\left\{\hat{\boldsymbol{\Phi}}\right\}\right\rangle$)

输入：DHT HT, DSP$\left\langle\hat{t},\left\{\hat{\boldsymbol{\Phi}}\right\}\right\rangle$

输出：最小化后的HT

```
1  for each < t, S_Φ > ∈ HT
2    for each Φ ∈ S_Φ
3      if  t̂ ⩽ t && Φ̂ ⊂ Φ
4        return
5      end if
6      if  t̂ ⩾ t && Φ̂ ⊃ Φ
7        从S_Φ移除Φ
8      end if
9      if S_Φ == ∅
10       从HT从移除t
11     end if
12   end for
13 end for
14 return HT
```

复杂度分析. 该算法的时间复杂度与DHT中的子空间数量有关,假设其为p. 在最佳的情况下,算法将直接返回,时间复杂度为$O(1)$. 在最坏的情况下,算法将遍历DHT中的所有子空间,时间复杂度为$O\left(p\right)$. 该算法在执行过程中并不额外开辟堆栈,故其空间复杂度为$O(1)$.

4.2.3 嵌套轮廓策略

本小节将讨论ISDT最小化问题到广义轮廓问题的转换.

定义8 (嵌套轮廓问题) 设DHT HT=$\left\{P_0,\cdots,P_k\right\}$,其中$P_i=\left\{t_i,S_i=\left\{\boldsymbol{\Phi}_{i0},\cdots,\boldsymbol{\Phi}_{il_i}\right\}\right\}$. 设$R^N=\left\{N_0,\cdots,N_k\right\}$为一个关系表,其中$N_i=\left\{\left\{t_i,\boldsymbol{\Phi}_{i0}\right\},\cdots,\left\{t_i,\boldsymbol{\Phi}_{il_i}\right\}\right\}$,注意到$N_i$中的元组由$P_i$映射,

且有$|N_i| = l_i$以及$|R^N| = \sum_{i=0}^{k} l_i$.$R^N$仅包含两个维度$d_t$和$d_s$,$d_t$和$d_s$的偏好函数如公式(1)和公式(2)所示(仅给出$f_d = 1$的情况).$R^N$关于全维度$\Phi^+ = \{d_t, d_s\}$的广义轮廓问题被称为嵌套轮廓问题.

$$f_{d_t}(a,b) = 1 \Leftrightarrow a > b, \tag{1}$$

$$f_{d_s}(A,B) = 1 \Leftrightarrow A \supset B. \tag{2}$$

定理 2　ISDT 的最小化问题等价于嵌套轮廓问题.

证明　ISDT的最小化问题等价于其中全部DHT的最小化问题的总和,因此下面将讨论DHT的最小化问题(P1)和嵌套轮廓问题(P2)的等价关系.从P1归约到P2:设HT中至少存在$P_i = \{t_i, \{\Phi_{i1}, \Phi_{i2}\}\}$以及$P_j = \{t_j, \{\Phi_j\}\}$,其中$\Phi_j \subseteq \Phi_{i2} \subset \Phi_{i1}$且$t_i \geqslant t_j$.根据定义7中的条件1)和2),问题P1即是移除$P_i$中的$\Phi_{i2}$以及整个$P_j$.将HT映射为$R^N$,则$R^N$中至少包含元组集合$\{r_0 = \{t_i, \Phi_{i1}\}, r_1 = \{t_i, \Phi_{i2}\}, r_2 = \{t_j, \Phi_j\}\}$,应用公式(1)和公式(2)提供的偏好函数,$R^N$关于全空间的广义轮廓集合为$\{r_0\}$,其映射为DSP形式,即$P_i = \{t_i, \{\Phi_{i1}\}\}$.从P2归约到P1:$R^N$关于全空间的广义轮廓集合即是由$R^N$中全部不被其他元组支配的元组构成的,设$r_0 = \{t_i, \Phi_i\} <_{\Phi^+} r_1 = \{t_j, \Phi_j\}$,则有以下情况:1)$t_i > t_j$且$\Phi_i \supseteq \Phi_j$;2)$t_i = t_j$且$\Phi_i \supset \Phi_j$.现将$r_0$和$r_1$映射为DSP形式.考虑情况1),有$P_i = \{t_i, \{\Phi_i\}\}$以及$P_j = \{t_j, \{\Phi_j\}\}$,根据定义7的条件2),$P_j$将被移除.考虑条件2),有$P_i = \{t_i, \{\Phi_i, \Phi_j\}\}$,根据定义7的情况1),$P_i$中的$\Phi_j$将被移除.最后在上述两种情况下,均有$P_i = \{t_i, \{\Phi_i\}\}$,其映射为$R^N$的形式,则有$r = \{t_i, \Phi_i\}$,其即为$r_0$.

MinimizeISDT+算法.使用嵌套轮廓策略的MinimizeISDT+算法的伪代码如算法3所示.首先,算法将HT映射为数据集R^N(第1行).利用标准的轮廓算法计算得到R^N中的非轮廓元组R_η^N(第2行),最后R_η^N被映射为HT_η并被返回(第3、4行).

复杂度分析.设n为HT中的子空间数量,$m = 2$为数据集R^N维度数量,则有$|R^N| = n$.HT映射为R^N的时间复杂度为$O(n)$,使用SFS算法来计算R^N的轮廓集合,其平均时间复杂度为$O(n \times \ln n)$,R_η^N映射为HT_η时间复杂度为$O(|R_\eta^N|)$.在n个元组中轮廓元组的期望数量如公式(3)[25-26]所示:

$$E(|\mathrm{Sky}(R)|) = \frac{(\ln(n+\gamma))^{m-1}}{(m-1)!}, \gamma = \lim_{i \to +\infty} \sum_{j=1}^{i} \frac{1}{j} - \ln i \approx 0.577, \tag{3}$$

其中γ是欧拉-马歇罗尼常数,且通常远小于n,则$|R_\eta^N| = \frac{(\ln(n+\gamma))^{m-1}}{(m-1)!} = \ln(n+\gamma)$.综上所述,MinimizeISDT+算法的时间复杂度为$O(n + n + \gamma + n \times \ln n)$,即$O(n \times \ln n)$.MinimizeISDT+算法的空间复杂度为$O(n)$,这主要来自数据集$R^N$和其轮廓集合的占用.

算法3 MinimizeISDT+(HT)

输入: DHT HT
输出: 最小化之后的DHT HT
1 HT映射为R^N
2 $R_\eta^N \leftarrow R^N - \mathrm{Sky}(R^N)$
3 R_η^N映射为HT_η
4 return HT_η

4.3　ISDT的动态维护

4.3.1　ISDT的剪枝

由于每个元组的生命周期不同,许多元组可以在其过期之前就能够被断言不会成为关于某特定子空间的轮廓元组.基于这一特性,提出以下两种剪枝策略:

(1)(强剪枝)如果在元组r的HT_r中存在一个DSP $p = \langle t, S_\Phi \rangle$,其中$t$等于$r$的过期时间戳$t_\omega(r)$,并且$S_\Phi$包含全空间$\Phi^+$,即$\langle t, S_\Phi \rangle = \langle t_\omega(r), \{\Phi^+\} \rangle$(如果$S_\Phi$包含$\Phi^+$,则在DSP最小化后只会存在$\Phi^+$),因此可以直接从系统中移除$r$;

(2)(弱剪枝)如果在元组r的HT_r中存在一个DSP $p = \langle t, S_\Phi \rangle$,其中$t$小于$r$的过期时间戳$t_\omega(r)$,并且$S_\Phi$包含全空间$\Phi^+$,即$\langle t, S_\Phi \rangle = \langle t, \{\Phi^+\} \rangle$,其中$t < t_\omega(r)$,则记录$t_p$作为$r$的标签,称为剪枝时间戳.$r$在后续计算中被忽略,直到达到$t_p$所表示的时间戳.

定理 3　设元组r被弱剪枝,剪枝时间戳为t_p.设$\tilde{r}$为另一个元组,如果$t_\omega(\tilde{r}) \leqslant t_p$,则ISDT不需要存储$r$关于$\tilde{r}$的DSP.

证明　如果r被弱剪枝,则存在另一个元组$\hat{r}$(不同于$\tilde{r}$),满足$\hat{r} <_{\Phi^+} r \wedge t_\omega(\hat{r}) = t_p(r) < t_\omega(r)$.假设$\tilde{r} <_\Phi r$,由于$t_\omega(\tilde{r}) \leqslant t_p(r)$,有$t_\omega(\tilde{r}) \leqslant t_\omega(\hat{r}) < t_\omega(r)$.则$r$的DHT为$\{\langle t_p, \{\Phi^+\} \rangle, \langle t_\omega(\tilde{r}), \{\Phi\} \rangle\}$,且其没有被最小化.在$HT_r$的最小化过程中,$\langle t_\omega(\tilde{r}), \{\Phi\} \rangle$将被移除.综上所述,ISDT不需要存储$r$关于$\tilde{r}$的DSP.

4.3.2 ISDT维护算法

ISDT-M算法.应用剪枝策略的ISDT-M算法的伪代码如算法4所示.算法的前半部分主要进行结构的初始化和DRP的计算(第1-6行).根据定理3,首先判断$t_\omega(r) \geqslant t_\omega(\hat{r})$是否成立(第7行).随后算法检查$\boldsymbol{\Phi}$是否为全空间$\boldsymbol{\Phi}^+$,以便进行后续的剪枝.如果$t_\omega(r) \geqslant t_\omega(\hat{r})$,则$\hat{r}$被强剪枝,其直接从数据集中移除;否则,$\hat{r}$被弱剪枝,算法将$\hat{r}$的剪枝时间戳$t_p$设置为$t_\omega(r)$(第8-13行).如果上述条件不满足,则将$t_\varepsilon$和$\boldsymbol{\Phi}$放入DHT中(第16行).最后,旧的$\mathrm{HT}_r$被更新,新的$\widehat{\mathrm{HT}}_r$被插入到ISDT(第18和20行),维护后的ISDT被返回.

算法4 ISDT-M$(R, \hat{R})$

输入: 旧数据集R和新数据集$\hat{R}$
输出: 维护后的ISDT

```
1  ISDT←当前的ISDT
2  for each r ∈ R
3    HT_r ← r的DHT,移除HT_r[t]
4    for each  r̂ ∈ R̂
5      ĤT_r,r̂ ← r̂的DHT,⟨Φ, Φ̂⟩ ←关于r̂和r的DRP
6      t_ε ← min(t_ω(r), t_ω(r̂))
7      if t_ω(r) > t_p(r̂)&&Φ ≠ ∅
8        if Φ == Φ⁺
9          if t_ω(r) ⩾ t_ω(r̂)
10           移除r̂
11         else
12           t_p(r̂) ← t_ω(r)
13         end if
14         break
15       end if
16       将t_ε和Φ插入ĤT_r
17     end if
18     类似7-17行更新HT_r
19   end for
20   插入ĤT_r到ISDT
21 end for
22 return ISDT
```

图3显示了$\boldsymbol{R}$实例在时间戳3到5时的ISDT结构的维护过程,红色部分表示被移除的DSP,绿色部分表示新增的DSP.由于r_0和r_1的过期,它们对应的DHT被完全移除.元组r_3是新到达的,其与现有元组(仅r_2)进行比较,并将对应的$\left\langle 6, \{\{p, c, r\}\}\right\rangle$放入$\mathrm{HT}_{r_3}$中.由于$\mathrm{HT}_{r_2}$中的所有DSP因过期而被移除,所以$\mathrm{HT}_{r_2}$为空.

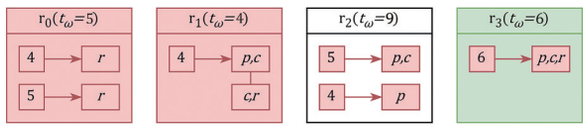

图3 $\boldsymbol{R}$实例在时间戳3到5时的ISDT结构的维护过程

Fig. 3 The maintenance process of ISDT structure of $\boldsymbol{R}$ instance from timestamp 3 to 5

4.4 ISDT的搜索

4.4.1 朴素方法

根据定理1,可以通过枚举数据集中每个元组对应的DHT来获得当前时间戳下关于某个子空间的轮廓结构.假设$|R| = n$,维度数量为m,R中的轮廓元组的期望数量为ϵ.在每次ISDT完全维护后,R中仍然有$n - \epsilon$个元组.对于一个未最小化ISDT,ISDT中仍然有$n - \epsilon$个DHT,但每个DHT中有$2^m - 2$个子空间,因此若进行朴素搜索,则其时间复杂度为$O(n \times 2^m)$.搜索过程中并不需要开辟额外堆栈空间,因此其空间复杂度为$O(1)$.

4.4.2 基于位图的优化方法

在ISDT上搜索轮廓元组与每个元组的DSP中的时间戳无关.实际上,DHT中存储的部分信息对于获取当前轮廓元组是冗余的,而遍历访问会导致显著的时间消耗.因此,考虑使用基于位图的结构,仅存储当前时间戳的有效信息,并利用位运算来大幅提高获取轮廓元组的效率.本文将提出用于ISDT的位图索引(ISDT-BM),以加速在ISDT结构上查询CMGS结果的速度.ISDT-BM由一个公共的未剪枝元组有序列表L和拥有$2^m - 2$个位图的数组$\mathrm{BM} = \{\mathrm{bm}_0, \cdots, \mathrm{bm}_{2^m-3}\}$组成.每个位图$\mathrm{bm}_i$对应一个子空间$\boldsymbol{\Phi}_i$(不包括空集和全空间),其中$i$是$\boldsymbol{\Phi}_i$的子空间编码减1.列表$L$中元组的位置$j$对应于每个位图$\mathrm{bm}_i$中的位$\mathrm{bm}_i[j]$的位置,这个位置记录了元组是否关于子空间$\boldsymbol{\Phi}_i$上支配.子空间编码的定义如下.

定义9 (子空间编码) 设$\boldsymbol{\Phi}^+ = \{d_0, \cdots, d_{m-1}\}$为一个有序的$m$-维全空间集合,$\boldsymbol{\Phi}$为$\boldsymbol{\Phi}^+$中$k$个元素构成的$k$-维子空间.对于$\boldsymbol{\Phi}$的第$i$个维度$\boldsymbol{\Phi}[i]$,设其在$\boldsymbol{\Phi}^+$中的索引为$\chi_i$,其中$0 \leqslant \chi_i < m$.基于全空间$\boldsymbol{\Phi}^+$的子空间$\boldsymbol{\Phi}$的编码是一个整数$\sum_{i=0}^{|\boldsymbol{\Phi}|-1} 2^{|\boldsymbol{\Phi}^+|-1-\chi_i}$,记作$\Xi(\boldsymbol{\Phi})$.

ISDT-BM中的核心计算方法如公式(4)和(5)所示:

$$b_{\lambda-1} \leftarrow b_{\lambda-1} \vee \left(1 \ll \left(|b_\lambda| - p - 1\right)\right), \tag{4}$$

$$b_{\lambda-1} \leftarrow b_{\lambda-1} \wedge \left(1 \ll \left(|b_\lambda| - p - 1\right)\right), \tag{5}$$

其中$\lambda = \Xi(\Phi)$为Φ的子空间编码,p为元组在L中的索引. 公式(4)表示将λ对应的位图$b_{\lambda-1}$中表示p的位置设为1,即插入操作;公式(5)表示将上述位置设为0,即删除操作.

ISDT-S算法. 算法的伪代码如算法5所示. 它以子空间Φ作为输入,输出对应的CMGS结果$\mathrm{Sky}(\Phi)$. 首先,算法将轮廓集Sky初始化为空集,并获取有序列表L和位图BM(第1行). 如果Φ是全空间,则直接返回L作为轮廓结果(第2-4行). 否则,初始化一个临时位图变量$\widehat{\mathrm{bm}}$,将每一位设置为0(第5行). 接下来,算法遍历所有位图,如果Φ是位图对应子空间的子集,则将该位图与$\widehat{\mathrm{bm}}$进行按位或操作,并将结果赋值给$\widehat{\mathrm{bm}}$(第6-11行). 然后,算法将L中特定位置的元组添加到Sky中,其中$\widehat{\mathrm{bm}}$中对应位置的位为0(第13-16行). 最后,算法返回结果集Sky(第17行).

算法5 ISDT-S(Φ)

输入:一个子空间Φ

输出:关于Φ的CMGS结果

```
1  Sky ← ∅, L ← 有序列表, BM = {bm_0, ···, bm_{2^m − 3}} ← 位图集合
2  if Φ == Φ+
3     return L
4  end if
5  bm̂ ← {0, ···}
6  for each bm_λ ∈ BM
7     λ̂ ← Ξ(Φ)
8     if λ̂ ∧ λ == λ̂
9        bm̂ ← bm ∨ bm̂
10    end if
11 end for
12 for i ∈ |L|
13    if bm̂ ∧ 1 ≪ (|L| − λ − 1) == 0
14       加入L[λ]到Sky
15    end if
16 end for
17 return Sky
```

复杂度分析. 算法ISDT-S的时间复杂度与BM和L的规模有关.BM的大小为$2^m - 2$,其中m是元组的维度,L的大小是关于全空间Φ^+的轮廓元组数量,即$|L| = |\mathrm{Sky}(R)|$. 轮廓元组的期望数量如公式(3)所示. 算法ISDT-S的时间复杂度为$O(2^m + |L|)$. 算法ISDT-S需要开辟$2^m - 2$个位图,每个位图有L位,所以其空间复杂度为$O(|L| \times 2^m)$.

考虑$\boldsymbol{R}$实例,其在时间戳3时的ISDT-BM索引如图4所示. 假设要查询的子空间是维度c,首先算法对所有与c的超集对应的位图进行按位或操作,即$\widehat{\mathrm{bm}} = 0\mathrm{b} \vee 10\mathrm{b} \vee 10\mathrm{b} = 10\mathrm{b}$. 在$L$中与$\widehat{\mathrm{bm}}$中值为0的位对应的元组是$r_0$和$r_2$,因此它们被添加到Sky集合中. 综上所述,有$\mathrm{Sky}_3(\{c\}) = \{r_0, r_2\}$.

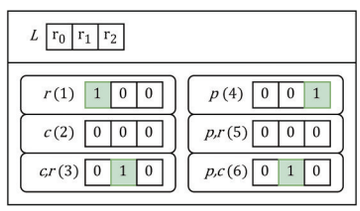

图4　$\boldsymbol{R}$实例在时间戳3时的ISDT-BM索引

Fig. 4　The ISDT-BMindex of $\boldsymbol{R}$ instance at timestamp 3

5　实验

为了评估ISDT的性能,使用了合成数据集和真实数据集. 实验中使用的真实数据集是其他轮廓研究中常用的数据集,如表4所示,涉及维度数量m、元组数量n和数据集是否经过泛化. 对于合成数据集,基于BORZSONY等提出的模式生成随机数据[1],使用不同的分布(独立分布(IND)、相关分布(COR)、反相关分布(ACOR)). 对于每个合成数据集,在$\{10^4, 10^5, 10^6\}$范围内变化元组数量n,在$\{4, 8, 12, 16, 20\}$内变化维度数量m,这些指标与相关研究中的取值类似. 实验中的主要数据结构ISDT及相关算法使用C++实现,运行在一台配备了Intel Core i7-12700H处理器(4.70 GHz)和32 GB内存的Windows 11计算机上.

表4　真实数据集以及相关参数

Tab. 4　Real datasets and related parameters

数据集	元组数量	维度数量	广义化
NBA	20943	17	否
MBL	92797	18	否
INSEE	2628433	22	否
FIFA23	17660	29	是

5.1　结构构建评估

在本节中,将比较ISDT与其他类似结构(NSC、CSC和HashCube)在合成数据集和真实数据集上的构建时间和内存使用情况,这些数据结构可以有效地处理静态多维轮廓问题. 由于仅考虑构建时间,数据相对静态. 将所有元组的到达时间设置为0,过期时间设置为1,旨在确保每个元组在时间戳内是静态的,以便与其他数据结构进行更好地比较和评估. 关于内存使用,对于ISDT,统计DSP的总数量和ISBM

中位图的数量(使用64位整数表示64位的位图,而不是位数组),对于NSC,统计总对数,对于CSC和HashCube,统计需要存储的子空间总数.综上所述,评估的是局部性能,而不是完整全运行框架的性能.

5.1.1 合成数据集

图5-7分别显示了这些数据结构在3个合成数据集(IND、COR和ACOR)上的构建时间(随着维度数量m和元组数量n的变化).对于数据集IND和ACOR,每种结构的构建时间在m和n变化时表现出类似的趋势.由于IND和ACOR有更多的轮廓元组和较少的非轮廓元组,这使得早期剪枝变得不容易,因此它们的构建时间通常较长.而COR的构建时间通常较短,因为它轮廓元组较少,元组早期被支配和剪枝概率更高.HashCube和CSC的趋势相似,而ISDT和NSC的趋势非常相似.后两者在$m = 16$和$m = 20$时的时间远低于前两者.在构建ISDT时,处理的数据通常在同一时间内到达,因此无需管理元组,ISDT的构建时间可以达到与NSC相同的数量级.

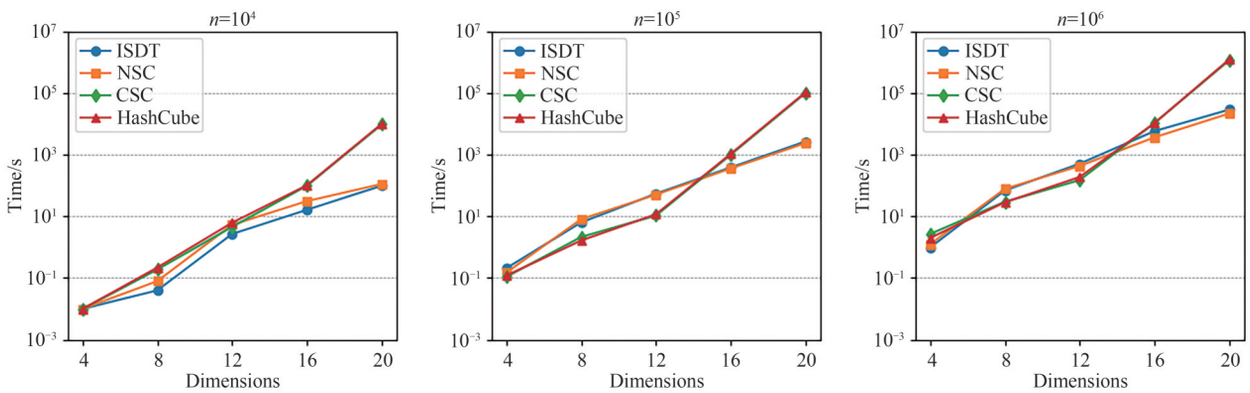

图5 IND数据集的构建时间

Fig. 5 The construction time of IND dataset

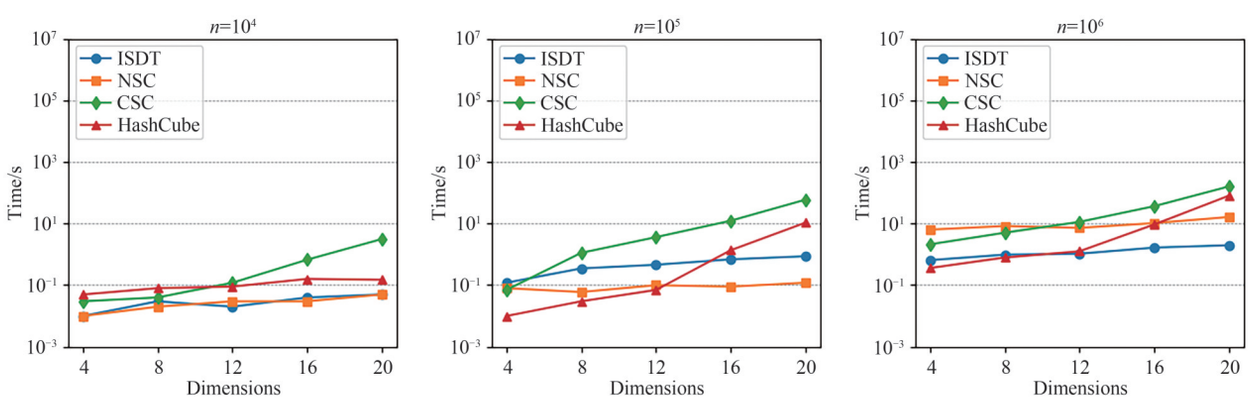

图6 COR数据集的构建时间

Fig. 6 The construction time of COR dataset

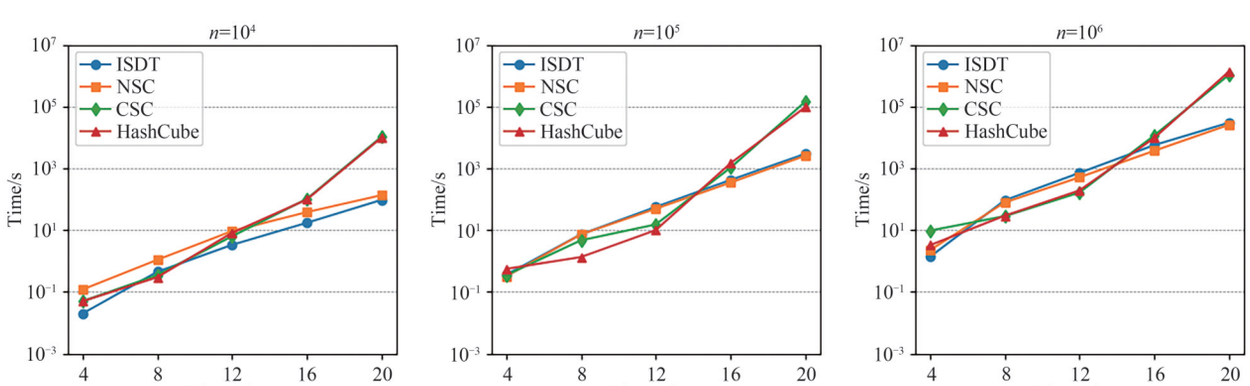

图7 ACOR数据集的构建时间

Fig. 7 The construction time of ACOR dataset

图 8-10 分别显示了 3 种合成数据集(IND、COR 和 ACOR)上的数据结构内存使用情况.可以观察到,当 m 和 n 变化时,HashCube 通常占用最高的内存,并且比其他结构高出显著的量(当 m = 20 时高出 1000 倍).ISDT、NSC 和 CSC 的内存使用情况相当,随着 n 的变化呈线性增长.ISDT 的内存使用量略高于 NSC,因为 ISDT 存储了额外的时间戳信息,这些信息在其他技术中没有涉及.值得一提的是,COR 数据集的内存使用量在 3 个合成数据集中也是最低的,而 IND 和 ACOR 的内存使用量相当.

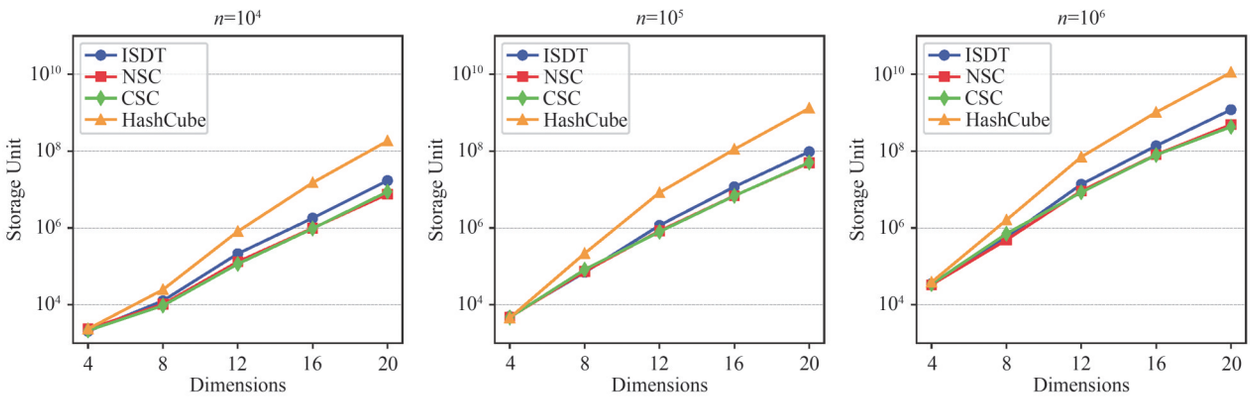

图 8　IND 数据集的内存使用

Fig. 8　The memory usage of IND dataset

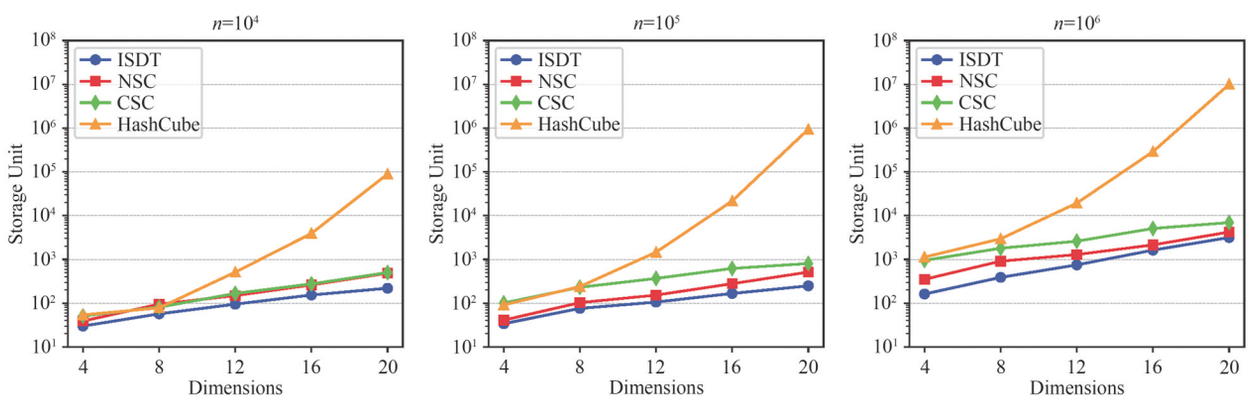

图 9　COR 数据集的内存使用

Fig. 9　The memory usage of COR dataset

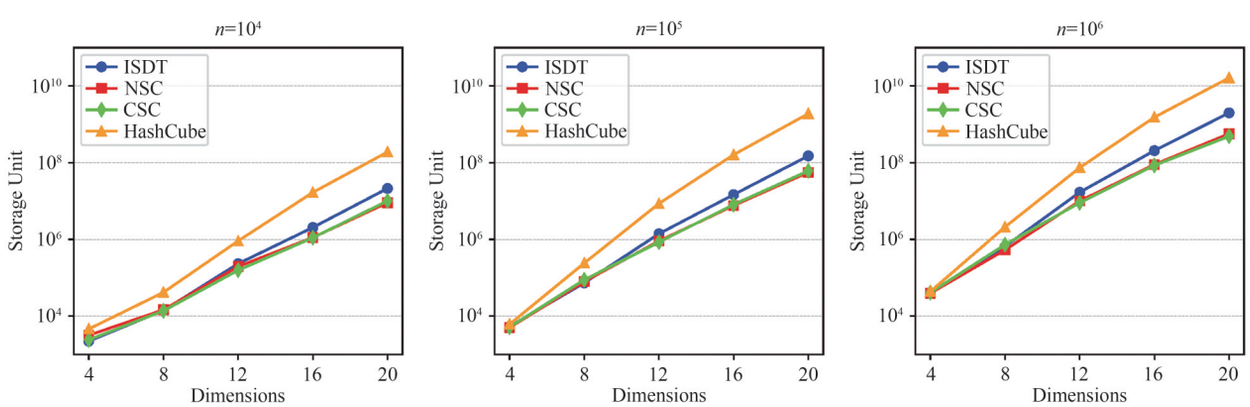

图 10　ACOR 数据集的内存使用

Fig. 10　The memory usage of ACOR dataset

5.1.2　真实数据集

图 11 显示了在真实数据集上数据结构的性能表现.可以看出,对于真实数据集,HashCube 仍然具有最高的时间消耗和内存使用.对于 INSEE 数据集,CSC 和 HashCube 无法在合理的时间内处理数据.尽管 INSEE 数据集的元组数量庞大(2628433),但其元组相关性较高,ISDT 和 NSC 仍能够轻松处理.ISDT 与 NSC 结构的性能相当,但由于需要存储

额外的时间戳信息,ISDT的内存消耗仍然较高.

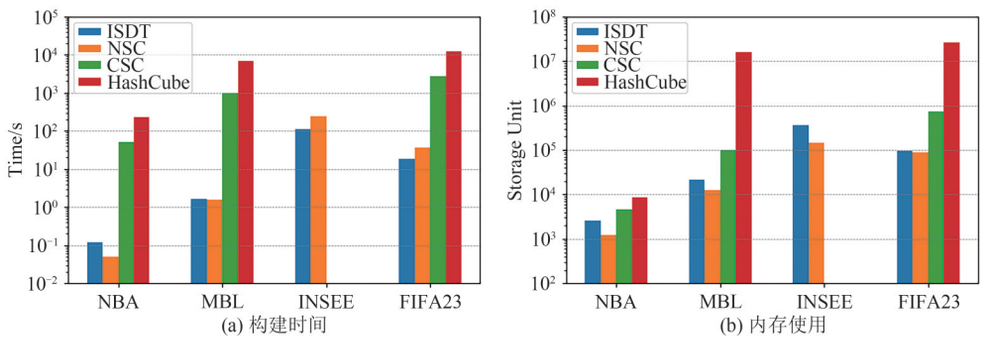

图11 真实数据集的性能评估

Fig. 11 The performance evaluation on real datasets

5.2 结构搜索评估

本节将评估在各种结构上搜索轮廓答案的性能,每个结构的评估将在数据集完全构建之后进行.对于ISDT结构,默认使用ISDT-BM来提供轮廓结果.

5.2.1 合成数据集

图12-14分别显示了在各种合成数据集上的轮廓搜索时间.可以观察到,除了ISDT外,其他结构在IND和ACOR数据集上的轮廓搜索时间在m和n变化时类似,而在COR数据集上的时间消耗是IND和ACOR数据集的百分之一,这与构建时间的整体性能评估一致.对于ISDT,其在COR数据集上的性能一般,但在IND和ACOR数据集上表现良好,因为ISDT-BM的时间消耗主要与维度和元组数量有关.同时,ISDT的时间消耗在维度增长时增长率最低,大约为$16 = 2^4$,其性能随着n和2^m线性增长.

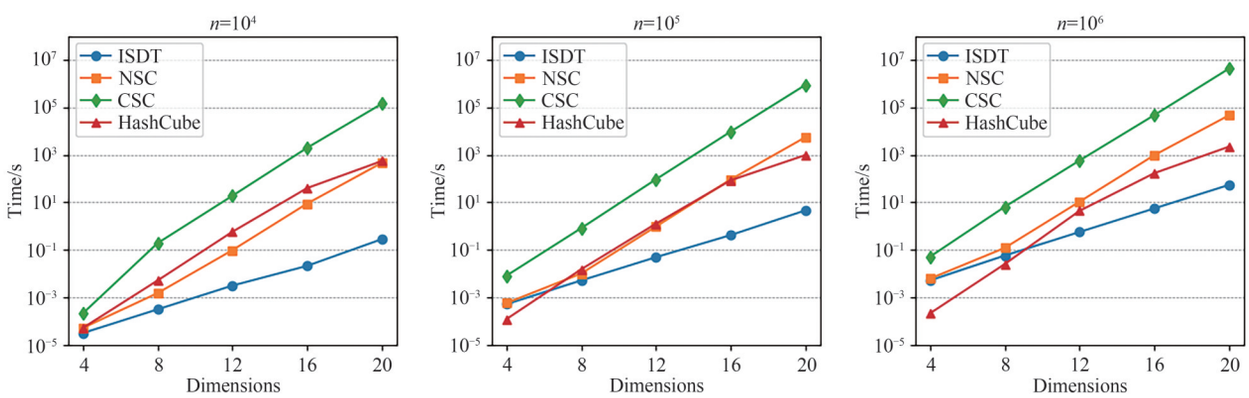

图12 IND数据集的搜索时间

Fig. 12 The search time of IND dataset

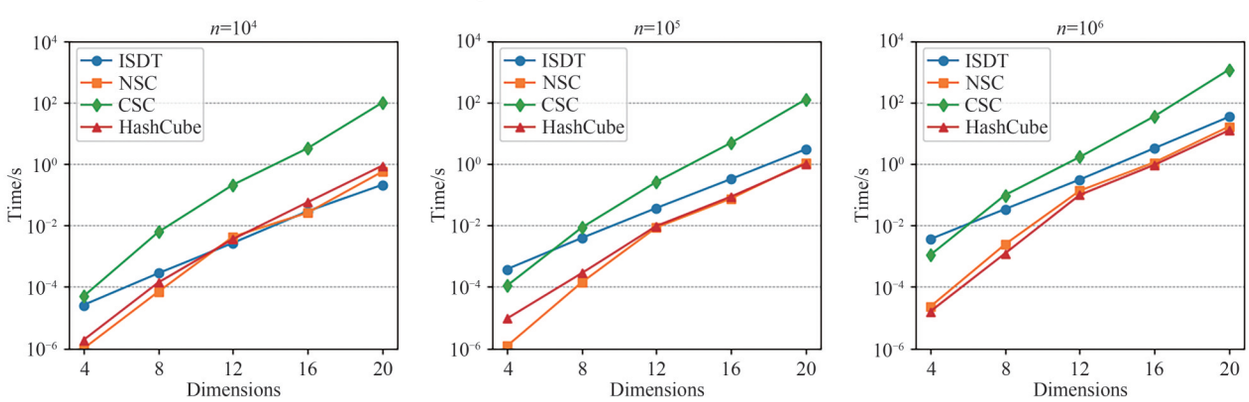

图13 COR数据集的搜索时间

Fig. 13 The search time of COR dataset

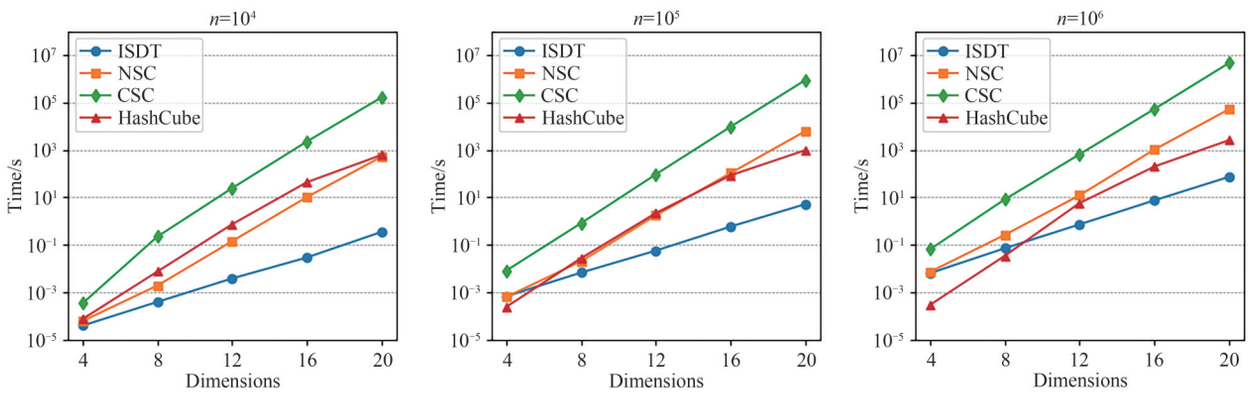

图 14　ACOR 数据集的搜索时间

Fig. 14　The search time of ACOR dataset

5.2.2　真实数据集

图 15 显示了在处理真实数据集时,各结构在搜索轮廓答案上的性能. 需要注意的是,对于 INSEE 数据集,CSC 和 HashCube 未能在合理的时间内构建完成,因此没有它们在轮廓搜索上的实验数据. 可以观察到,ISDT 在这些结构中仍然表现最佳,在 INSEE 数据集上,ISDT 的轮廓答案搜索效率约为 NSC 的 30 倍. ISDT 在处理大量元组的数据集时,搜索效率仍然表现良好.

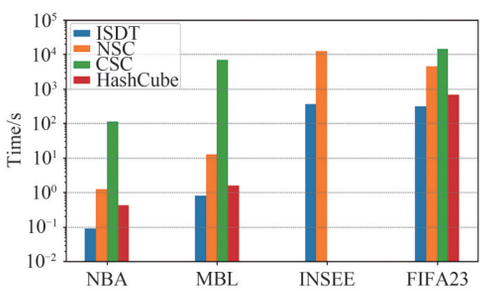

图 15　真实数据集的搜索时间

Fig. 15　The search time of real datasets

5.3　结构维护评估

以下对 ISDT 结构在频繁数据更新中的效率也进行了评估. 前文所对比的数据结构不再适用,因为它们不特别支持处理带有时间戳的元组. 尽管一些数据结构支持逐步维护或重建,但它们的时间消耗无法与专门设计的用于处理连续数据的结构相匹敌. 因此本小节主要将 ISDT 与 NSCt 和 DBSky 进行比较.

流数据配置的相关参数、含义和取值如表 5 所示. μ 和 ν 间接决定了系统中元组的最小数量,即 ν/μ,而 υ 决定了评估结构将同时应对的元组数量,即 υ/μ. 例如,当 μ = 0.1 s 和 ν = 24 h 的时候,系统中至少有 864000 个元组,而当 υ = 1200 s 时,结构需要处理新增的 12000 个元组.

表 5　流数据参数符号及定义

Tab. 5　The notation and meaning of streaming data parameters

参数	意义	取值
μ	两个连续元组的时间间隔	1 s
ν	元组最小的生命周期	{6 h, 12 h, 24 h}
υ	维护频率	{1200 s}

5.3.1　合成数据集

图 16 显示了当 μ = 1 s 时,IND 数据集的维护时间,图中的红线表示 1200 s,其为维护数据集的间隔. 在合成数据集中,仅给出了 IND 数据集上的维护时间,因为对于 COR 数据集,时间消耗始终较低(通常少于 10 s),而 ACOR 数据集的数量级与 IND 相同. 可以观察到,DBSky 的维护时间始终最长,当维度数量大于或等于 12 时,其维护时间不是合理的值,而 ISDT 的维护时间在 3 种结构中通常是最少的. 总体而言,ISDT 的性能在三者中最佳,ISDT 和 NSCt 都能在非极端的情况下完成数据集的维护.

5.3.2　真实数据集

对于真实数据集,本文随机选取每个数据集 5% 的元组,并提前对剩下 95% 的元组完成构建,随后评估插入该 5% 的元组所需要的时间. 图 17 显示了真实数据集下各个结构完成维护的时间. 当数据量较小时,不同结构之间的差异并不显著,而当数据集的元组数量增大时,ISDT 和 NSCt 的表现较好. 对比合成数据集,真实数据集的效果往往更好,因为现实中的数据往往有着更高的正相关性,无论是 ISDT 还是 NSCt 都可以利用各自的剪枝策略来移除绝大部分元组.

6　结论

本文提出了连续多维度广义轮廓(简称 CMGS),设计并实现了一个高效的数据结构 ISDT,用于存储

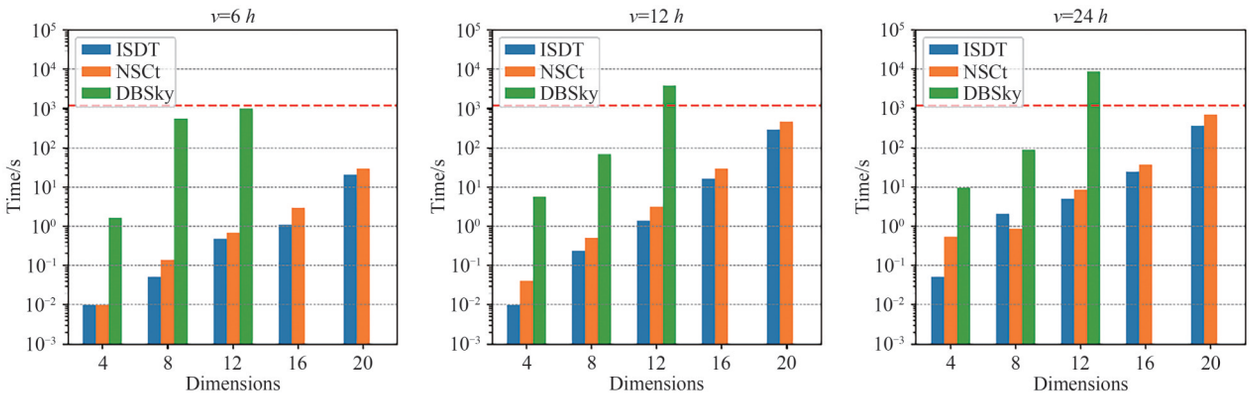

图 16 μ = 1 s时的IND数据集维护时间

Fig. 16 The maintenance time of IND dataset when μ = 1 s

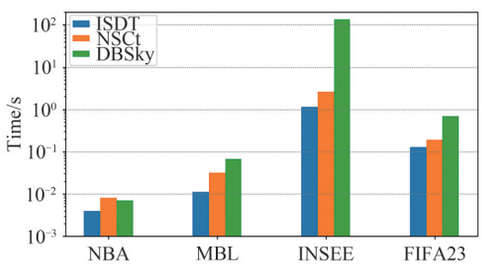

图 17 真实数据集的维护时间

Fig. 17 The maintenance time of real datasets

元组之间具有时间戳的子空间支配关系. 基于ISDT, 提出了若干相关的高效算法、两种最小化策略和两种剪枝策略. 通过实验表明:ISDT结构在静态数据场景下表现出与NSC和CSC等结构相当的处理性能,在动态数据场景下则比NSCt更高效. 因此,这些实验也验证了ISDT结构在处理CMGS问题上的可行性和效率. 未来,将对本文提出的ISDT框架做进一步优化,以适应超大数据集,并对其做进一步扩展以解决更多变体的天际线问题.

参 考 文 献

[1] BORZSONY S, KOSSMANN D, STOCKER K. The skyline operator [C]//Proceedings 17th International Conference on Data Engineering. Heidelberg:IEEE, 2001: 421-430.

[2] CHOMICKI J, GODFREY P, GRYZ J, et al. Skyline with presorting[C]//Proceedings 19th International Conference on Data Engineering. Bangalore:IEEE, 2003: 717-719.

[3] BARTOLINI I, CIACCIA P, PATELLA M. *SaLSa*: Computing the skyline without scanning the whole sky[C]// Proceedings of the 15th ACM International Conference on Information and Knowledge Management-CIKM '06. Arlington:ACM, 2006: 405-414.

[4] LU H X, LUO Y, LIN X. An optimal divide-conquer algorithm for 2D skyline queries[C]// Advances in Databases and Information Systems. Berlin: Springer Berlin Heidelberg, 2003: 46-60.

[5] TAN K, ENG P, OOI B. Efficient progressive skyline computation[C]//International Conference on Very Large Data Bases. Roma: VLDB Endowment:2001:301-310.

[6] KOSSMANN D, RAMSAK F, ROST S. Shooting stars in the sky: An online algorithm for skyline queries [C]// International Conference on Very Large Data Bases. Hong Kong: VLDB Endowment, 2002:275-286.

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

[8] LEE J, HWANG S. Bskytree: Scalable skyline computation using a balanced pivot selection[C]//International Conference on Extending Database Technology. Lausanne: ACM, 2010:195-206.

[9] LEE J, HWANG S W. Scalable skyline computation using a balanced pivot selection technique [J]. Information Systems, 2014, 39: 1-21.

[10] BAI M, HAN Y, YIN P, et al. S_IDS: An efficient skyline query algorithm over incomplete data streams[J]. Data & Knowledge Engineering, 2024, 149: 102258.

[11] WANG Z, ZHANG L, DING X, et al. A dynamic-efficient structure for secure and verifiable location-based skyline queries[J]. IEEE Transactions on Information Forensics and Security, 2022, 18: 920-935.

[12] BAI M, WANG Q, CHANG S, et al. Location-based skyline query processing technology in road networks[J]. The Journal of Supercomputing, 2024, 80(3): 3183-3211.

[13] BOURAHLA C, MAAMRI R, BRAHIMI S. Skyline recomputation in big data [J]. Information Systems, 2023, 114: 102164.

[14] WU J M, ZHOU H, LIN J C, et al. Mining skyline patterns from big data environments based on a spark framework [J]. Journal of Grid Computing, 2023,

21(2): 22.

[15] SUN M, TENG Y, ZHAO F, et al. Spatio-textual group skyline query[C]//DASFAA 2023 International Workshops. Cham: Springer Nature Switzerland, 2023: 34-50.

[16] LI C, DONG L. Boolean spatial temporal text keyword skyline query[C]//Advanced Data Mining and Applications. Cham: Springer Nature Switzerland, 2023: 210-224.

[17] ZHANG Y, SHI Y, ZHOU Z, et al. Efficient and secure skyline queries over vertical data federation[J]. IEEE Transactions on Knowledge and Data Engineering, 2023, 35(9): 9269-9280.

[18] HAN X, WANG J, LI J, et al. Efficient computation of G-Skyline groups on massive data [J]. Information Sciences, 2022, 587: 300-322.

[19] 李光辉, 李艳红, 杨洋, 等. 在正交查询范围内解决G-Skyline查询中的why-not问题[J]. 中南民族大学学报(自然科学版), 2023,42(5): 678-688.

[20] LIN X, YUAN Y, WANG W, et al. Stabbing the sky: Efficient skyline computation over sliding windows[C]//21st International Conference on Data Engineering (ICDE'05). Tokoyo:IEEE, 2005: 502-513.

[21] BABCOCK B, BABU S, DATAR M, et al. Models and issues in data stream systems[C]//Proceedings of the Twenty-First ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems. Madison: ACM, 2002: 1-16.

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

[23] LIN X, LU H, XU J, et al. Continuously maintaining quantile summaries of the most recent N elements over a data stream [C]//Proceedings of 20th International Conference on Data Engineering. Boston: IEEE, 2004: 362-373.

[24] SUN S, HUANG Z, ZHONG H, et al. Efficient monitoring of skyline queries over distributed data streams[J]. Knowledge and Information Systems, 2010, 25(3): 575-606.

[25] MORSE M, PATEL J M, GROSKY W I. Efficient continuous skyline computation[C]//22nd International Conference on Data Engineering (ICDE'06). Atlanta: IEEE, 2006: 108.

[26] ALAMI K, MAABOUT S. A framework for multidimensional skyline queries over streaming data[J]. Data & Knowledge Engineering, 2020, 127:101792.

(责编　曹东,校对　雷建云)