基于最大覆盖的代表Skyline问题的优化算法研究

白 梅 王习特 李冠宇 宁 博 周 新
(大连海事大学信息科学技术学院 辽宁 大连 116000)

摘 要 Skyline查询作为多目标决策的重要手段之一,可以根据用户偏好,在大量的数据中挖掘出用户真正感兴趣的数据.然而,随着维度的增加以及数据分布的原因,会导致skyline结果数目过多,查询结果失去意义.目前,已有一些工作对代表skyline问题进行了研究,即在全部skyline结果中选取k个最具代表性的skyline元组.综合考虑代表skyline的代表性以及稳定性,本文选取基于最大覆盖的代表skyline问题(k-Maximum Coverage Skyline,k-MCS)问题进行研究.与之前的k-MCS计算方法相比,本文提出的算法具有更好的效率.针对k-MCS问题,首先,本文提出了2维上的基于前缀的优化算法OPA(Optimal Prefix Algorithm),OPA算法利用前缀支配表,可以通过少量的加减法运算完成最后的结果计算.接着,考虑到多维上k-MCS问题是一个NP-Hard问题,本文提出了优化贪心算法OGA和ε-OGA,OGA比基本贪心算法减少了50%以上的计算量.而ε-OGA通过引入参数ε,与OGA算法相比,仅牺牲$\varepsilon/(1+\varepsilon)$的精度,大大加快了计算效率.最后,通过大量的实验验证了本文所提算法OPA、OGA和ε-OGA的有效性和高效性.

关键词 代表skyline;k-MCS;前缀算法;贪心算法;优化算法
中图法分类号 TP301.6 **DOI号** 10.11897/SP.J.1016.2020.02276

Research on Optimization Algorithms of k-Maximum Coverage Skyline Queries

BAI Mei WANG Xi-Te LI Guan-Yu NING Bo ZHOU Xin
(*College of Information Science & Technology, Dalian Maritime University, Dalian, Liaoning* 116000)

Abstract As an important operator for multi-criteria decision making, skyline queries can find the data that users are really interested in from a large amount of data. The skyline consists of the points that are not dominated by other points. Given two points p_1 and p_2, p_1 dominates p_2 means: the values of p_1 are as good as or better than those of p_2 in all dimensions, and better in at least one dimension. In most cases, the full skyline set is a good recommendation set because the tuples which are dominated by others are filtered out. However, with the increase of dimensionality and distribution of the dataset, the number of skyline tuples may be too large. As the recommendation set, the full skyline set becomes meaningless. Because it is impractical for users to choose suitable skyline points after browsing all skyline points. Hence, recommending a few representative skyline points would be very helpful to users. There have been some works focused on representative skyline problems. Considering the representativeness and stability of representative skylines, we choose to study the k-Maximum Coverage Skyline (k-MCS for short) problem. Given a paremater k, the k-MCS is the set with k skyline points whose dominance size is the largest. Compared with the previous k-MCS algorithms, the proposed algorithms in this paper have better efficiency. Firstly, we propose an optimization prefix-based algorithm (OPA

收稿日期:2019-09-06;在线发布日期:2020-02-19.本课题得到国家自然科学基金项目(61702072,61602076,61976032)、博士后科学基金面上项目(2017M621122,2017M611211)、辽宁省自然科学基金(20180540003)、中央高校基本科研业务费专项资金(3132019202)资助.
白 梅,博士,副教授,主要研究方向为数据管理、云计算和数据查询优化.E-mail:baimei861221@163.com.王习特,博士,副教授,主要研究方向为大数据管理和并行数据管理.李冠宇,博士,教授,主要研究领域为智能信息处理和语义物联网.宁 博,博士,副教授,主要研究方向为数据管理和隐私保护.周 新,博士,讲师,主要研究方向为数据管理和机器学习.

for short) for the k-MCS problem in 2-dimensional datasets. Using the prefix-dominance-table, OPA can obtain the k-MCS result with $O(kM^2)$ times of addition and subtraction operations, where M is the number of full skyline points. Secondly, consider that the k-MCS problem in d-dimensional ($d\geq 3$) datasets is a NP-hard problem, two optimaization greegy algorithms OGA and ε-OGA are proposed. In OGA, an improved formula is proposed to quickly calculate the dominance size of a set. Also, a filtering strategy is proposed to avoid the calculations of some redundant tuples. Hence, compared with the basic greedy algorithm, OGA reduces more than 50% calculations and have the same accuracy. Next, by introducing a parameter ε, the algorithm ε-OGA is proposed. The computation of ε-OGA can be terminated in advance with a guaranteeing accuracy. Compared with OGA, ε-OGA can greatly speed up the calculation efficiency by sacrificing $\varepsilon/(1+\varepsilon)$ accuracy, where ε is a small value given by users. Finally, the effectiveness and efficiency of the proposed algorithms OPA, OGA and ε-OGA are verified by a large number of experiments. In conclusion, compared with the previous algorithm PBA, OPA has better efficiency and the same accuracy, because it can increase the reuse rate of the results in the prefix-dominance-tables and reduce unnessary calculations of some tuples. Compared with the basic algorithm GA, OGA has better efficiency and the same accuracy, because it can reuse the previous calculation results to compute the dominace size of a set. Also, it can avoid some unnessary calculations using the filtering strategies. Compared with ε-GA, ε-OGA has better accuracy by sacrificing a small amount of running time. Compared with the previous algorithm RT, ε-OGA has better accuracy and better time efficiency.

Keywords　representative skyline; k-MCS; prefix algorithm; greedy algorithm; optimization algorithms

1　引　言

随着“大数据”时代的到来，数据已经成为重要的生产因素. 对海量数据的挖掘和运用，已成为国内外广大学者的研究重点. 轮廓查询(skyline query)[1-2]作为多目标决策(Multi-Criteria Decision-Making, MCDM)手段，可以通过偏好函数帮助用户从海量信息中提取出有针对性的价值富集，同时，轮廓查询还可以快速地确定数据集的帕累托边界(pareto frontier)，这些都使得轮廓查询在许多实际应用中有着非常重要的作用.

在介绍轮廓集合的概念之前，需要先引入支配的概念. 具体地，给定两个元组 p_1 和 p_2，p_1 支配 p_2 指的是：在所有维度上，p_1 都好于或者等于 p_2；至少在一个维度上，p_1 要好于 p_2. 为了表述方便，在本文里，值越小被认为越“好”. 现实生活中，所有的数值都可以通过 0-1 标准化①落在[0,1]区间内. 因此，如果某个属性值越大越好，可以采用 $1-x$ 的方法进行转换，使得转换后的值越小越好. 轮廓集合包含了所有不被其它元组“支配”的元组. 如图 1 所示，一共有 16 条房产记录$\{p_1, p_2, \cdots, p_{16}\}$. 每一条房产信息包括 2 个维度信息：与最近的交通站点的距离和每平米单价. 其中，距离值和价格都通过 0-1 标准化映射到[0,1]区间内，并以“小”值为优. 图中 p_4 在两个维度都比 p_{10} 小，那么说明 p_4 的交通情况和价格都

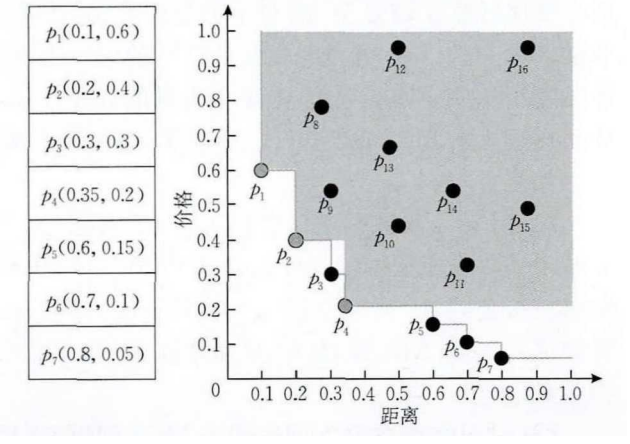


图 1　轮廓和 k 最大覆盖轮廓($k=3$)

① 0-1 标准化：也称离差标准化，它是对原始数据进行线性变换，使结果落到 [0,1] 区间.

比 p_{10} 好，则 p_4 支配 p_{10}. 图中的轮廓集合是 $\{p_1, p_2, \cdots, p_7\}$，其它的房产记录都可以被 $\{p_1, p_2, \cdots, p_7\}$ 中的 1 个或多个元组支配.

通过上述例子，可以发现尽管轮廓查询已经大大方便了人们的结果选择，但是当人们关注的维度较多，或者关注的数据集较大时，会造成轮廓元组的数目大大增加[2]. 这时，整体轮廓集合对于用户的意义将变小，因此，通过优中选优，推荐有限数目的具有代表性的轮廓元组给用户将变得十分必要. 本文主要针对代表轮廓问题进行了研究，即在整体轮廓中选出具有代表性的有限数目的轮廓元组. 代表轮廓在多目标决策问题上比轮廓集合更具有意义，尤其适合分布复杂和体量庞大的数据集合，更加适用在“大数据”时代中.

目前，关于代表轮廓问题[3-6]的研究已取得了很多成果. Bai 等人[3]和 Soholm 等人[4]提出了基于最大覆盖的代表轮廓问题——k-最大覆盖轮廓 k-MCS. 具体地，给定一个参数 k，从整体轮廓中选出 k 个轮廓元组，使得他们可以支配的面积（体积或超体积）达到最大. 如图 1 所示，图中全体轮廓为 $\{p_1, p_2, \cdots, p_7\}$，当 $k=3$ 时，$\{p_1, p_2, p_4\}$ 能够支配的区域如图中覆盖所示，它们的支配面积是 $0.04+0.09+0.52=0.65$. 其它包含 3 个轮廓元组的集合的支配面积都小于 0.65，所以，$\{p_1, p_2, p_4\}$ 就是 3-最大覆盖轮廓.

正如文献[3-4]中提到的，对比其它代表轮廓，k-最大覆盖轮廓更加稳定、计算速度更快、且具有很好的代表性. 但是，之前设计的 k-MCS 算法[3]在计算过程中，对于某些必要的中间结果没有进行保留，所以需要大量的重复性计算，导致算法效率偏低. 且造成每次计算比较复杂，浪费了大量的算力. 因此，本文设计一种方法，通过保留最为适合的中间结果，充分复用这些中间结果值，避免了大量的重复计算，从而加速了 k-MCS 的计算. 归结起来，本文的主要贡献如下：

(1) 针对 2 维数据空间中的 k-MCS 问题，提出了基于前缀支配表的前缀优化算法 OPA. 利用优化的求前缀公式，OPA 可以通过 $O(kM^2)$ 次加减法运算完成 k-MCS 的计算，其中 M 是整体 skyline 元组的数目.

(2) 针对多维数据空间中的 k-MCS 问题，提出了优化贪心算法 OGA，对比基础贪心算法，在保证相同精确度的前提下减少 50%以上的计算量；之后，在 OGA 算法的基础上，提出了提前截断算法 ε-OGA. 与 OGA 算法相比，ε-OGA 通过牺牲 $\varepsilon/(1+\varepsilon)$ 的精度，大大提高计算效率.

(3) 设计了详细的性能评价实验，实验结果表明本文所提出的算法 OPA、OGA 和 ε-OGA 能够分别高效地处理 2 维和多维空间中的 k-MCS 问题. 与之前算法相比较，本文提出的算法具有更好的计算效率.

本文第 2 节回顾相关工作；第 3 节介绍最大覆盖代表轮廓 k-MCS 的相关定义；第 4 节详细描述本文所提出的 k-MCS 的查询算法，针对 2 维空间，提出 OPA 算法；针对多维数据空间，提出优化贪心算法 OGA 和 ε-OGA；第 5 节给出实验结果与分析；第 6 节对全文进行总结.

2 相关工作

轮廓查询(skyline query)的概念最早由 Borzsonyi 等人[1]在 2001 年提出，轮廓查询的前身是最大向量问题[7]. 文献[8]对 skyline 及其变体查询进行了综述性概括，包括 skyline 查询及其变体查询的相关文献. 与本文关联较大的所有文献都在 2.2 节进行了介绍.

2.1 轮廓查询算法

文献[1]中最早提出了两个轮廓查询算法 BNL 和 D&C. BNL 依次扫描全体数据并把不被支配的元组加入到候选集合中，通过多次迭代求出最终的轮廓集合. D&C 算法把全体数据分成多个子集并求出每个子集的子集轮廓，然后合并所有的子集轮廓得到最终的轮廓集合. SFS 算法[9]首先按照单调函数把数据集排序，使得排在后面的数据不可能支配排在前面的数据，利用该性质来计算轮廓集合.

利用索引，轮廓查询的效率得到了大大提高. Bitmap 算法[10]首先把每个元组映射成一个 m 位的矢量，利用转换后的矢量求解最终的轮廓集合. NN 算法[11]和 BBS 算法[12]利用 R-tree 索引来管理全部的数据元组，NN 利用最近邻来进行过滤求得最终的轮廓集合. BBS 算法通过访问那些包含最终轮廓元组的 R-tree 节点来求得最终的轮廓集合. ZBtree 算法[13]采用 Z-order 索引来管理全体数据，并利用 Z-order 之间的顺序过滤，来计算最终的轮廓结果.

此外，还有很多研究是针对特定环境下的轮廓查询问题. 如数据流上的轮廓查询[14-15]算法，旨在解决那些数据频繁更新的多目标决策问题，适用于股票市场、传感器环境监控方面等. 分布式环境上的

轮廓查询算法[16-17]，旨在解决数据体量大的多目标决策问题，适用于电子商务环境、传感器网络等数据量大的环境中. 不确定环境上的轮廓查询[18]算法，旨在解决数据值不完全精确环境下的多目标决策问题，适用于数据范围观测环境中.

2.2　代表轮廓查询算法

随着数据时代的到来，轮廓查询处理的数据量越来越大，数据分布越来越复杂，从而导致了整体轮廓的大小越来越大. 当整体轮廓元组数目过多时，对整体轮廓的研究意义将变小. 因此，越来越多的学者致力于代表轮廓[3-6]的研究.

文献[19-24]中通过改变支配定义，来控制最终结果的数目，选出的结果元组有可能不是传统的轮廓元组. 文献[19]中提出了 k-支配的概念，元组 p_1 能够 k-支配 p_2 指的是：在选中的 k 个子维度上，p_1 支配 p_2. 通过选择合适的子维度就可以控制结果元组的数目. Peng 等人①针对高维上的 k-支配 skyline 问题提出了一种并行解决方法，利用 GPU 框架来快速计算 k-支配 skyline 结果. Xia 等人[20]提出了 ε-支配的概念，元组 p_1 能够 ε-支配 p_2 指的是：p_1 每个维度上的值增加 ε 后，转换后的 p_1 支配 p_2. 通过调整 ε 值，就能控制结果元组的数目. 信等人[21]提出了 ρ-支配的概念，元组 p_1 能够 ρ-支配 p_2 指的是：p_1 每个维度上的值增加 ρ 倍后，转换后的 p_1 支配 p_2. 通过调整 ρ 值，可以控制结果元组的数目. Zhang 等人[22]提出了圆锥支配的概念，p_1 能够圆锥支配 p_2 指的是：p_2 与 p_1 形成的斜率在圆锥角度范围内. 通过调整圆锥的角度，就可以调整结果元组的数目.

文献[23]提出了 top-k 轮廓，每个元组都根据该元组的支配能力（即该元组可以支配的其它元组的数目）进行排序，选出排在前 k 个的元组作为 top-k 轮廓结果返回. 之后，文献[24]研究了数据流上的 top-k 轮廓问题. 无疑地，该 top-k 轮廓只考虑了单个元组的支配能力，没有考虑返回结果的整体支配能力. 更有甚者，top-k 轮廓选出的代表元组有可能集中在一起，且不是轮廓元组，因此，top-k 轮廓的代表性并不够好.

Lin 等人[5]提出了基于整体支配数目的代表轮廓 RSP. RSP 希望选出 k 个轮廓点，使得选中的 k 个轮廓点可以支配的元组数目达到最大. 无疑地，RSP 能保证选中的轮廓元组具有良好的代表性. 但是，RSP 的稳定性较差，且计算复杂. 当数据集合变化时，非轮廓元组会影响 RSP 的结果.

Tao 等人[6]提出了基于距离的代表轮廓 DRS. DRS 采用距离来衡量选定集合的代表性. 给定一个有 k 个轮廓元组的子集 K，整体轮廓用 SKY 表示，子集 K 的代表因子为 $Er(K,SKY)=\max_{p\in SKY-K}\{\min_{p'\in K}|p,p'|\}$，其中 $|p,p'|$ 是元组 p 和 p' 的欧几里德距离. 简单说来，$Er(K,SKY)$ 的含义就是选中 k 个元组，记录每个未选中的元组距离它最近的选中元组的距离，其中的最大距离就是 K 的距离因子. DRS 是因子数 $Er(K,SKY)$ 最小的子集 K. 当某个维度进行缩放的时候，例如单位由千米变成米，DRS 的结果会发生变化. 同时，这种代表轮廓的定义也忽视了轮廓定义的核心，与支配能力完全无关.

另外，文献[25-26]中提出了后悔集合的概念. 文献[25]提出了最小后悔代表集合概念，文中首先定义了后悔的概念，即针对每个用户，全集的最高打分函数减去选中子集的最高打分函数就是该集合针对该用户的后悔值，而后悔率就是后悔值与选中子集打分函数的比值，最小后悔代表集合就是选中大小为 k 的子集，使得所有用户的后悔率的上确界值最小. 文献[26]提出了 k-后悔最小集合，文中定义了 k-后悔值为全集中第 k 高的打分函数值减去子集中最高打分函数值. 而 k-后悔率就是 k 后悔值与全集中第 k 高的打分函数值的比值. k-后悔最小集合就是选中大小为 r 的子集，对所有用户的 k-后悔率的上确界最小. 文献[25-26]都以后悔值为目标进行考虑，这样选中的代表集合与用户在各个维度的打分函数关系十分密切. 文献[27]中提出了基于意义和多样性的代表轮廓 SDRS，他们希望设定一个合理的参数，综合考虑多样性和意义. 其中，多样性采用距离来衡量，即选择的元组相距越远，多样性越好. 而选中元组的意义由 sigmoid 函数来衡量. 文献[28]中提出了基于点击的代表轮廓，即对每个轮廓元组都记录一个用户点击它的概率值，选出 k 个轮廓元组，使得用户点击其中一个元组的概率值达到最大. 上述的几种代表轮廓的定义都忽视了轮廓的核心定义——支配能力，且它们的计算都是非常复杂的.

另外还有很多文献[29]采用代表轮廓的概念用来过滤，这些文章中的代表元组的选择标准与本文选定的选择标准都是类似的，是基于支配面积（体积或者超体积）的，但是这些文章都没有给出详细的求解方法，导致选出的代表轮廓误差太大.

---

① Peng Y-W, Chen W-M. Parallel k-dominant skyline queries in high-dimensional datasets. Information Sciences. https://doi.org/10.1016/j.ins.2019.01.039

与本文最相近的文献是文献[3-4,30],它们研究的都是基于最大覆盖(即支配面积、体积或超体积)的代表轮廓查询 k-MCS 问题. 文献[4]中只论述了该代表轮廓选取标准的优越性,没有就 k-MCS 问题给出实际的解决办法. 文献[3]中给出了 k-MCS 问题的详细解决办法,但是它的处理办法中对计算的复用率不够,导致算法效率并不够高. 文献[30]主要针对 k-MCS 问题的删除鲁棒性问题进行了研究,但是只针对多维贪心算法给出了优化查询算法 RT. RT 采用了 Lazier 贪心加速算法[31]的思想,首先求出一个 corset 集合,而后只在 coreset 集合中进行计算,不用对全集进行计算.

本文针对 k-MCS 问题设计了一种更加灵活、高效的解决办法,有效提升了 k-MCS 问题的计算效率.

3 问题描述

本文针对基于最大覆盖的代表轮廓查询 k-MCS 问题进行了研究. 为了描述方便,表 1 中给出了本文的符号定义.

表 1 符号表示

符号	符号含义
p_1, p_2	数据元组
P	数据集合
$SKY(P)/SKY$	数据集合 P 的轮廓集合
d	数据集合 P 的维度
$p_1[i]$	元组 p_1 在维度 i 上的值
k	代表轮廓元组的数目
k-MCS(SKY)	基于最大覆盖的 k-代表轮廓
$DomSize(s_i)$/ $DomSize(S)$	元组 s_i/集合 S 的支配(超)体积/面积
$IntSize(S)$	集合 S 所有元组的相交支配(超)体积/面积
$IntSize(s_i, S)$	元组 s_i 和集合 S 支配面积/体积相交部分的大小
i-$Sets(S)$	S 中任意 i 个元组组成的所有集合
$PreSet(s_i)$	排在 s_i 之前的元组组成的集合
$PreSize(S, s_i)$	S 对 s_i 的前缀支配面积
$IncreSize(s_i, S)$	元组 s_i 相对于 S 的增量支配体积/面积

给定 1 个 d 维的数据集合 P,每个数据元组 p 可以表示为 $p=\langle p[1], p[2], \cdots, p[d]\rangle$,且每个维度上的值都以小值为优,并映射到[0,1]区间内. 通过映射函数,可以把每个数据在各维度上的值都转化到[0,1]范围内. 下面,本文回顾一下轮廓的基本概念.

定义 1(支配[1]). 给出 d 维数据集合 P, p_1, $p_2 \in P$, p_1 支配 p_2(记作 $p_1 \prec p_2$)需要满足以下两个条件:(1) $\forall i \in \{1,2,\cdots,d\}$, $p_1[i] \leq p_2[i]$;(2) $\exists j \in \{1,2,\cdots,d\}$, $p_1[j] < p_2[j]$.

集合 P 中所有不被其它元组支配的元组就组成了 P 的轮廓集合,记作 $SKY(P)=\{p_i \mid p_i, p_j \in P, \nexists p_j \prec p_i\}$,简写为 SKY.

本文研究的是基于最大覆盖(支配面积或体积)的代表轮廓问题 k-MCS. 在介绍 k-MCS 的标准定义之前,先介绍如何求解一个集合的支配面积(体积). 由于每个元组在所有维度上的值都可以标准化到 0-1 范围内,基于此标准,下面给出一个 d 维元组 p 的支配大小(支配体积或支配面积),记作 $DomSize(p)=\prod_{i=1}^{d}(1-p[i])$. 给定一个含有 n 个元组的 d 维集合 $S=\{p_1, p_2, \cdots, p_n\}$, S 的相交支配大小指的是集合中所有元组共同支配区域的大小,记作 $IntSize(S)=\prod_{i=1}^{d}(1-\max_{p_j \in S}(p_j[i]))$. 给定元组 p 和集合 S, p 和 S 的相交支配大小为能够被 p 和 S 共同支配区域的大小,记作 $IntSize(p, S)$.

如图 2 中所示,元组 p_2 的支配大小为 $DomSize(p_2)=(1-0.2)\times(1-0.4)=0.48$. 集合 $S_1=\{p_1, p_2, p_4\}$ 的相交支配区域如图中深色部分所示,相交支配大小为 $IntSize(S_1)=(1-0.35)\times(1-0.6)=0.26$. 给定集合 $S_2=\{p_1, p_2\}$ 和元组 p_4, $IntSize(p_4, S_2)$ 的大小如图中所有填色部分所示 $(1-0.4)\times(1-0.35)=0.39$.

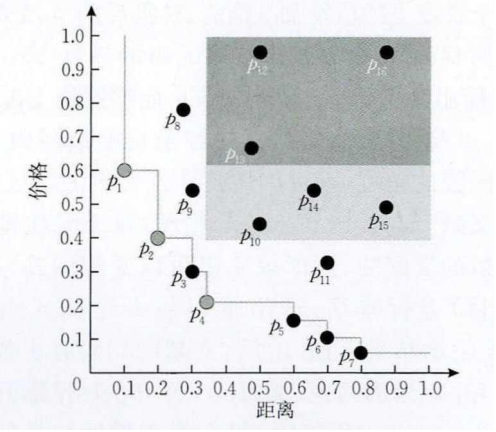


图 2 集合的支配大小举例

根据文献[3]中的定义 2,可以得到一个集合的含有 n 个元组的 d 维数据集合 $S=\{p_1, p_2, \cdots, p_n\}$, S 的支配大小如式(1)所示:

$$\begin{aligned} DomSize(S)=&\sum_{i=1}^{n} DomSize(p_i)+\\ &(-1)^1\sum_{S_j \in 2\text{-}Sets(S)} IntSize(S_j)+\\ &(-1)^2\sum_{S_j \in 3\text{-}Sets(S)} IntSize(S_j)+\cdots+\\ &(-1)^{i-1} IntSize(S) \end{aligned} \tag{1}$$

其中，i-$Sets(S)$指的是 S 中含有 i 个元组的所有集合. 例如，给定集合 $S=\{p_1,p_2,p_3,p_4\}$，3-$Sets(S)=\{\{p_1,p_2,p_3\},\{p_1,p_2,p_4\},\{p_1,p_3,p_4\},\{p_2,p_3,p_4\}\}$. 式(1)利用集合的容斥原理来求解任意集合的支配面积.

定义 2(k-最大覆盖轮廓 k-MCS[3]). 给出 d 维数据集合 P 的轮廓集合 SKY 和参数 k，如果 SKY 中的元组数目少于 k 个，那么最大覆盖轮廓 k-MCS 就是 SKY；反之，k-MCS就是SKY中由k个元组组成的支配面积最大的那个集合，记作 k-$MCS(SKY)=\{S_i \mid S_i,S_j \in k\text{-}Sets(SKY),\nexists S_j, DomSize(S_j)>DomSize(S_i)\}$，简写为 k-$MCS(SKY)$.

如图 2 所示，给定参数 $k=3$，轮廓集合 $SKY=\{p_1,p_2,\cdots,p_7\}$，它的 3-最大覆盖轮廓是 3-$MCS(SKY)=\{p_1,p_2,p_4\}$，它的支配面积是 0.65. 任意其它 3 个轮廓元组的支配面积一定小于 0.65，所以该轮廓集合的 3-最大覆盖轮廓是$\{p_1,p_2,p_4\}$.

4　最大覆盖代表轮廓 k-MCS 查询处理

本节中，针对最大覆盖轮廓问题 MCS 的解决方案给出了详细的描述. 本文就 MCS 问题分别给出了 2 维和多维(3 维及 3 维以上)的解决方案.

4.1　2 维 k-MCS 的计算

本节关注的是在 2 维环境下，在轮廓集合中求解 MCS 问题. 考虑到 2 维环境下轮廓元组的分布特征：如果所有的轮廓元组在一个维度上呈升序排列，在另一个维度上一定是降序排列. 这里规定轮廓集合 SKY 以及 SKY 的任意子集中的所有元组都按照第一维属性值由小到大进行排列. 本小节首先描述了 k-MCS 问题的基本解决方法——动态规划解决方法；接着，描述了本文提出的前缀优化算法.

4.1.1　k-MCS 的动态规划解决方案

给定参数 k 和轮廓集合 $SKY=\{s_1,s_2,\cdots,s_M\}$，$|SKY|=M$，对于 SKY 中任意含有 k 个元组的子集，所有能够排在第 i 个位置的元组组成了集合 i-$order=\{s_i,s_{i+1},\cdots,s_{M-k+i}\}$. 如图 2 所示，共有 7 个轮廓元组，当 $k=3$ 时，2-$order=\{p_2,p_3,p_4,p_5,p_6\}$. 对于任意包含 3 个元组的集合，$p_1$ 和 p_7 永远不可能出现在第 2 个位置上.

给定轮廓集合 $SKY=\{s_1,s_2,\cdots,s_M\}$，对于任意轮廓元组 $s_i \in SKY$，集合 $\{s_1,s_2,\cdots,s_{i-1}\}$ 中的元组都排在 s_i 之前，称 $\{s_1,s_2,\cdots,s_{i-1}\}$ 是 s_i 的前缀集合，记作 $PreSet(s_i)=\{s_1,s_2,\cdots,s_{i-1}\}$. 给定一个元组 s_i 和 $PreSet(s_i)$ 的任意子集 $S=\{s'_1,s'_2,\cdots,s'_l\}\subset PreSet(s_i)$，那么 S 对 s_i 的前缀支配面积是 $S\cup\{s_i\}$ 的支配面积减去 s_i 的支配面积，记作 $PreSize(S,s_i)$.

$$\begin{aligned}PreSize(S,s_i)&=DomSize(S\cup\{s_i\})-DomSize(s_i)\\&=\sum_{j\in[1,l-1]}(1-s'_j[2])(s'_{j+1}[1]-s'_j[1])+\\&\quad(1-s'_l[2])(s_i[1]-s'_l[1])\end{aligned}\tag{2}$$

如图 2 所示，给定集合 $S=\{p_1,p_2,p_4\}$ 和元组 p_5，S 对 p_5 的前缀支配面积可以记作 $PreSize(S,p_5)=(1-0.6)\times(0.2-0.1)+(1-0.4)\times(0.35-0.2)+(1-0.2)\times(0.6-0.35)=0.33$.

下面，借助前缀集合的概念，定义了s_i的最大组合.

定义 3(l-最大组合[3]). 给定任意轮廓元组 $s_i \in SKY$，在 $PreSet(s_i)$中任选 $l-1$ 个元组，使得这 $l-1$ 个元组与 s_i 组合在一起的集合支配面积(或体积)最大，选定的 $l-1$ 个元组就组成了 s_i 的$(l-1)$-最大前缀，记作$(l-1)$-$LarPre(s_i)$. $(l-1)$-$LarPre(s_i)$与 s_i 的并集，构成了 s_i 的 l-最大组合，记作 l-$LarCom(s_i)$：

$$\begin{aligned}(l-1)\text{-}LarPre(s_i)=\{&S_m \mid S_m\subset PreSet(s_i),\\&|S_m|=l-1,\nexists S_n\subset Preset(s_i),|S_n|=l-1,\\&DomSize(S_n\cup\{s_i\})>DomSize(S_m\cup\{s_i\})\}\end{aligned}\tag{3}$$

$$l\text{-}LarCom(s_i)=(l-1)\text{-}LarPre(s_i)\cup\{s_i\}\tag{4}$$

如图 2 所示，给定轮廓元组 p_4，$PreSet(p_4)=\{p_1,p_2,p_3\}$，$p_4$ 的 2-最大前缀为 2-$LarPre(p_4)=\{p_1,p_2\}$，p_4 的 3-最大组合为 3-$LarCom(p_4)=\{p_1,p_2,p_4\}$. 其它前缀集合$\{p_1,p_3\}$、$\{p_2,p_3\}$与 p_4 组合的支配面积都小于$\{p_1,p_2,p_4\}$的支配面积.

下面，通过引入文献[3]的定理，就可以用动态规划思想来解决 k-MCS 问题.

引理 1. 给出 2 维轮廓集合 $SKY=\{s_1,s_2,\cdots,s_M\}$和参数 k，以及集合 k-$order=\{s_k,s_{k+1},\cdots,s_M\}$中所有元组的 k-最大组合，那么 SKY 的 k-$MCS(SKY)$一定是这些 k-最大组合中支配面积最大的那个集合.

证明. 参见文献[3]中的定理 1.

引理 2. 给定2维轮廓集合SKY和$s_i \in l$-$order=\{s_l,s_{l+1},\cdots,s_{M-k+l}\}$，那么 s_i 的$(l-1)$-最大前缀一定属于$\{s_{l-1},s_l,\cdots,s_{i-1}\}$($(l-1)$-$order$ 中排在 s_i 之前的所有元组)中某元组的$(l-1)$-最大组合.

证明. 参见文献[3]中的定理 2.

根据引理 2 和 3 可以发现，给定 2 维环境下的轮廓集合和参数 k 时，能够利用动态规划方法解决 k-MCS 问题. 如图 3 所示，给出 7 个轮廓元组和参数 $k=3$，首先，得到 1-$order$ 中每个元组的 1-最大

组合，就是该元组本身，得到$\{p_1\}$、$\{p_2\}$、$\{p_3\}$、$\{p_4\}$和$\{p_5\}$. 接着，对 2-*order* 中每个元组，从相应的 1-最大组合中，求出 1-最大前缀. 如元组 p_4 的 1-最大前缀，一定是$\{p_1,p_2,p_3\}$中某个元组的 1-最大组合(由引理 2 得到)，$PreSize(\{p_1\},p_4)=(1-0.6)\times(0.35-0.1)=0.1$ 是最大的，所以 p_4 的 1-最大前缀是$\{p_1\}$. 求出 2-*order* 中每个元组 1-最大前缀后，得到每个元组的 2-最大组合$\{p_1,p_2\}$、$\{p_1,p_3\}$、$\{p_1,p_4\}$、$\{p_2,p_5\}$和$\{p_2,p_6\}$. 最后，求出 3-*order* 中每个元组的最大组合，得到$\{p_1,p_2,p_3\}$、$\{p_1,p_2,p_4\}$、$\{p_1,p_4,p_5\}$、$\{p_1,p_4,p_6\}$和$\{p_1,p_4,p_7\}$.

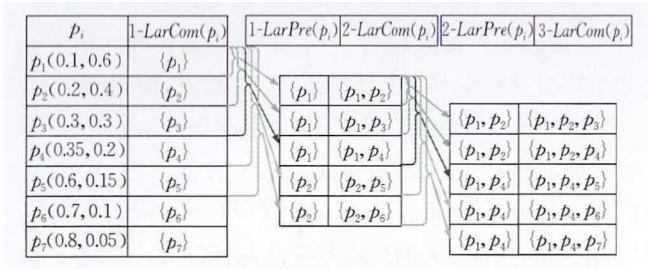


图 3 k-MCS 问题的动态规划解决方法实例

4.1.2 k-MCS 的前缀优化算法 OPA

在动态规划方法中，对于 *l-order* 中的任意元组 s_i，想要求得 s_i 的$(l-1)$-最大前缀，需要计算$\{s_{l-1},s_l,\cdots,s_{i-1}\}$中每个元组的$(l-1)$-最大组合对 s_i 的前缀支配面积. 在本文提出的优化算法中，可以减少前缀支配面积的计算代价和计算次数，从而提高查询效率.

定义 4(l-前缀支配表). 给定轮廓集合 $SKY=\{s_1,s_2,\cdots,s_M\}$和参数 k，对于 *l-order* 中的所有元组 s_i，都求得 *l-LarCom*(s_i)的支配面积，以及 *l-LarCom*(s_i)对$\{s_{i+1},s_{i+2},\cdots,s_{M-k+l+1}\}$(($l+1$)-*order* 中排在 s_i 之后的元组)中每个元组的前缀支配面积，并记录相应前缀支配面积，形成 l-阶前缀支配表.

如图 4 所示，共有 7 个轮廓元组以及 $k=3$，1-*order*$=\{p_1,p_2,p_3,p_4,p_5\}$，图 4 是一个 1-阶前缀支配表. 记录了 1-*order* 中每个元组的支配面积，以及它们的 1-最大前组合对 2-*order*$=\{p_2,p_3,p_4,p_5,p_6\}$中对应元组的前缀支配面积. 图中计算了 1-*LarCom*(p_1)对 p_2 到 p_6 的前缀支配面积，其中 1-*LarCom*(p_1)对 p_2、p_3和 p_4的前缀支配面积大于其它 1-最大前组合对 p_2、p_3和 p_4的前缀支配面积，由此可以得到 p_2、p_3和 p_4的 2-最大组合是$\{p_1,p_2\}$、$\{p_1,p_3\}$和$\{p_1,p_4\}$.

	2-*LarCom*(p_i)	$\{p_1,p_2\}$ 0.52	$\{p_1,p_3\}$ 0.57	$\{p_1,p_4\}$ **0.62**	$\{p_2,p_5\}$ 0.58	$\{p_2,p_6\}$ 0.57
	1-*LarCom*(p_i)	p_2(0.48)	p_3(0.49)	p_4(0.52)	p_5(0.34)	p_6(0.27)
0.36	p_1(0.1, 0.6)	0.04	0.08	0.1	0.2	0.24
0.48	p_2(0.2, 0.4)		0.06	0.09	0.24	0.3
0.49	p_3(0.3, 0.3)			0.035	0.21	0.28
0.52	p_4(0.35, 0.2)				0.2	0.28
0.34	p_5(0.6, 0.15)					0.085

图 4 1-阶前缀支配表

在建立$(l+1)$-阶前缀支配表时，之前的 1 到 l-阶前缀支配表已经建立. 因此，可以用如下方法通过一次加法运算来求得$(l+1)$-阶前缀支配表中的前缀支配面积. 给定元组 $p_i\in l\text{-}order$，p_i 的 l-最大组合为 *l-LarCom*$(p_i)=\{p_1,p_2,\cdots,p_{i-1},p_i\}$，*l-LarCom*($p_i$)对 $p_j(i<j)$的前缀支配面积可以采用如下公式求得：

$$PreSize(l\text{-}LarCom(p_i),p_j)=PreSize(\{p_1,\cdots,p_{i-1}\},p_i)+PreSize(p_i,p_j) \quad (5)$$

由于 $p_i\in l\text{-}order$，因此 p_{i-1} 一定属于$(l-1)$-*order*，则 $PreSize(\{p_1,p_2,\cdots,p_{i-1}\},p_i)$一定在$(l-1)$-阶前缀支配表中被求出. 通过对比式(2)和(5)，可以发现，在求解前缀支配面积时，式(5)的计算量为 1 次加法，式(2)的计算量为 l 次乘法加 $3l$ 次加减法，式(5)的计算量远小于式(2).

需要注意的是，为了利用式(5)求解所有的前缀支配面积，对于任意 $i<j$，需要记录元组 p_i 对 p_j 的前缀支配面积，形成 1-阶全支配表. 如图 5 所示.

		p_2(0.48)	p_3(0.49)	p_4(0.52)	p_5(0.34)	p_6(0.27)	p_7(0.19)
0.36	p_1(0.1, 0.6)	0.04	0.08	0.1	0.2	0.24	0.28
0.48	p_2(0.2, 0.4)		0.06	0.09	0.24	0.3	0.36
0.49	p_3(0.3, 0.3)			0.035	0.21	0.28	0.35
0.52	p_4(0.35, 0.2)				0.2	0.28	0.36
0.34	p_5(0.6, 0.15)					0.085	0.17
0.27	p_6(0.7, 0.1)						0.09
0.19	p_7(0.8, 0.05)						

图 5 1-阶全支配表

前缀优化算法的基本框架：根据引理 1 和引理 2，前缀优化算法可采用如下算法框架来求 k-MCS，如算法 1 所示.

算法 1. 前缀优化算法——基本框架.

输入：skyline 集合 SKY，参数 k

输出：k-MCS

1. 填写 1-阶前缀支配表和 1-阶全支配表；
2. 对于任意 $p_j\in 2\text{-}order$，利用 1-阶前缀支配表，求出 p_j 的 2-最大组合；
3. FOR(i 从 2 到 $k-1$)
4. 对于 *i-order* 中的所有元组的 i-最大组合，求出对应的 i-阶前缀支配表，利用式(5)；
5. 利用 i-阶前缀支配表和式(5)，可以对于任意的 $p_j\in(i+1)\text{-}order$，求出 p_j 的$(i+1)$-最大组合，并

记录 p_j 的 $(i+1)$-最大组合的支配面积；//引理 2

6. ENDFOR

7. 对 k-order 中所有元组的 k-最大组合，选出其中支配面积最大的集合作为 k-MCS；//引理 1

可以发现，算法 1 和动态规划方法的区别在于：计算前缀支配面积采用的公式不同. 因此，在求解 k-MCS 问题上，算法 1 的计算量要远小于动态规划方法[3]的计算量.

在算法 1 的 3～6 行中，给定 i-order 中的所有元组的 i-最大组合，来求解 $(i+1)$-order 中所有元组的 $(i+1)$-最大组合. 在该过程中，并不需要填写 i-阶前缀支配表中的所有值. 下面，给出前缀优化策略，可以有效缩减 i-阶前缀支配表中的非必要值，从而形成 i-阶前缀支配优化表.

下面采用优化策略 1 和优化策略 2 来优化前缀支配表的填写.

优化策略 1. 采用 l-最大组合 $(l\geq 1)$ 的支配面积来缩减 l-前缀支配表的计算，可以直接省略某个 l-最大组合对其它所有元组的前缀支配面积的计算，如定理 1 所示.

定理 1. 给定 l-order 中所有元组的 l-最大组合，如果 $p_i \in l\text{-order}$，$\forall p_j \in l\text{-order}$ 且 $j>i$，当 $l\text{-LarCom}(p_i)$ 的支配面积大于等于 $l\text{-LarCom}(p_j)$ 的支配面积时，那么 $l\text{-LarCom}(p_j)$ 不可能成为 $(l+1)$-order 中任何元组的 l-最大前缀，所以在 l-阶前缀支配表中，不需要求 $l\text{-LarCom}(p_j)$ 的前缀支配面积.

证明. 已知 $DomSize(l\text{-LarCom}(p_i)) > DomSize(l\text{-LarCom}(p_j))$，对于任意 p_m 排在 p_j 之后，可以得到 $PreSize(l\text{-LarCom}(p_j),p_m)=DomSize(l\text{-LarCom}(p_j))-(1-p_m[1])\times(1-p_j[2])$ 以及 $PreSize(l\text{-LarCom}(p_i),p_m)=DomSize(l\text{-LarCom}(p_i))-(1-p_m[1])\times(1-p_i[2])$. 由于 p_i 排在 p_j 之前，可知 $1-p_i[2]>1-p_j[2]$，所以 $(1-p_m[1])\times(1-p_i[2])>(1-p_m[1])\times(1-p_j[2])$. 可以得到 $PreSize(l\text{-LarCom}(p_i),p_m)>PreSize(l\text{-LarCom}(p_j),p_m)$. 证毕.

如图 6 所示，根据图 4 的 1-阶前缀支配表，可以得到 $2\text{-order}=\{p_2,p_3,p_4,p_5,p_6\}$ 中每个元组的 2-最大组合. 根据 1-阶前缀支配表，可以计算每个 2-最大组合的支配面积，其中 $2\text{-LarCom}(p_4)=\{p_1,p_4\}$ 的支配面积大于其它 2-最大组合的支配面积. 根据定理 1，不需要计算 $2\text{-LarCom}(p_5)$ 和 $2\text{-LarCom}(p_6)$ 对相应元组的前缀支配面积.

3-LarCom(p_i)		$\{p_1,p_2,p_3\}$ 0.59	$\{p_1,p_2,p_4\}$ **0.65**	$\{p_1,p_4,p_5\}$ 0.64	$\{p_1,p_4,p_6\}$ 0.65	$\{p_1,p_4,p_7\}$ 0.65
		$p_3(0.3)$	$p_4(0.35)$	$p_5(0.6)$	$p_6(0.7)$	$p_7(0.8)$
0.52	$\{p_1,p_2\}$	0.1	0.13	0.28	0.34	0.4
0.57	$\{p_1,p_3\}$		0.115	0.29	0.36	0.43
0.62	$\{p_1,p_4\}$			0.3	0.38	0.46
0.58	$\{p_2,p_5\}$				–	–
0.57	$\{p_2,p_6\}$					–

图 6　2-阶前缀支配表——优化策略 1

优化策略 2. 根据不同 l-最大组合对相同元组的前缀支配面积，可以省略某个 l-最大组合对某些元组的前缀支配面积的计算，如定理 2 所示.

定理 2. 给定 $p_i,p_j \in l\text{-order}$ 且 $i<j$，$p_m \in (l+1)\text{-order}$ 且 $j<m$，如果 $PreSize(l\text{-LarCom}(p_i),p_m)\leq PreSize(l\text{-LarCom}(p_j),p_m)$，那么 $\forall n>m$ 且 $p_n \in (l+1)\text{-order}$，$PreSize(l\text{-LarCom}(p_i),p_n)\leq PreSize(l\text{-LarCom}(p_j),p_n)$，不需要计算 $l\text{-LarCom}(p_i)$ 对 p_n 的前缀支配面积.

证明. 根据前缀支配面积的概念，可以很容易得到 $PreSize(l\text{-LarCom}(p_i),p_n)=PreSize(l\text{-LarCom}(p_i),p_m)+(1-p_i[2])\times(p_n[1]-p_m[1])$，且 $PreSize(l\text{-LarCom}(p_j),p_n)=PreSize(l\text{-LarCom}(p_j),p_m)+(1-p_j[2])\times(p_n[1]-p_m[1])$. 根据已知条件，$PreSize(l\text{-LarCom}(p_i),p_m)<PreSize(l\text{-LarCom}(p_j),p_m)$ 且 $1-p_i[2]<1-p_j[2]$，所以可以得到 $PreSize(l\text{-LarCom}(p_i),p_n)$ 一定小于 $PreSize(l\text{-LarCom}(p_j),p_n)$. 证毕.

如图 7 所示，对于元组 p_5，$PreSize(\{p_1,p_3\},p_5)=0.29<PreSize(\{p_1,p_4\},p_5)=0.3$，根据定理 2 可知，所有排在 p_5 之后的元组 p_6 和 p_7，$\{p_1,p_4\}$ 对 p_6 和 p_7 的前缀支配面积一定大于 $\{p_1,p_3\}$ 对 p_6 和 p_7 的前缀支配面积. 因此，不需要计算 $\{p_1,p_2\}$ 以及 $\{p_1,p_3\}$ 对 p_6 和 p_7 的前缀支配面积.

3-LarCom(p_i)		$\{p_1,p_2,p_3\}$ 0.59	$\{p_1,p_2,p_4\}$ **0.65**	$\{p_1,p_4,p_5\}$ 0.64	$\{p_1,p_4,p_6\}$ 0.65	$\{p_1,p_4,p_7\}$ 0.65
		$p_3(0.49)$	$p_4(0.52)$	$p_5(0.34)$	$p_6(0.27)$	$p_7(0.19)$
0.52	$\{p_1,p_2\}$	0.1	0.13	0.28	–	–
0.57	$\{p_1,p_3\}$		0.115	0.29	–	–
0.62	$\{p_1,p_4\}$			0.3	0.38	0.46
0.58	$\{p_2,p_5\}$				–	–
0.57	$\{p_2,p_6\}$					–

图 7　2-阶前缀支配表——优化策略 2

根据定理 1 和定理 2 的描述可知，在填写任意 l-阶前缀支配表时，不需要填写所有的值，那些必要填写的值组成了 l-阶前缀支配优化表. 需要注意的是，利用式(5)填写所有的 $(l+1)$-阶前缀支配优化表的值，所有的值都可以在 l-阶前缀支配优化表和

2-阶全支配表中找到.

下面,算法 2 给出了如何利用优化策略来填写 l-阶前缀支配表.

算法 2. 前缀优化算法——l-阶前缀支配优化表的生成.

输入:1-阶全支配表,$(l-1)$-阶前缀支配优化表

输出:l-阶前缀支配优化表

1. 根据$(l-1)$-阶前缀支配优化表,记录每个 l-LarCom(p_i)的支配面积,并根据定理 1 省略计算相应的前缀支配面积; //优化策略 1
2. FOR(每个 l-$LarCom(p_j)$,j--)
3. FOR(l-order 中的每个元组 p_i,i++)
4. 利用$(l-1)$-阶前缀支配表和 1-阶全支配表,用式(5)求出 l-$LarCom(p_j)$对 p_i的前缀支配面积 $PreSize(l\text{-}LarCom(p_j),p_i)$;
5. $\forall p_m \in l\text{-}order$ 且 $m \geq j$;
6. IF($PreSize(l\text{-}LarCom(p_j),p_i) < PreSize(l\text{-}LarCom(p_m),p_i)$)//优化策略 2
7. 省略计算 l-$LarCom(p_j)$对排在 p_i之后的元组的前缀支配面积;
8. ENDIF
9. ENDFOR
10. ENDFOR
11. 完成 l-阶前缀支配优化表的填写;

算法 2 描述了如何利用$(l-1)$-阶前缀支配优化表和 1-阶全支配表完成 l-阶前缀支配优化表的填写.下面以举例的形式来介绍算法 2.例如,当已知 2 阶前缀支配优化表(如图 7 所示)和 1-阶全支配表(图 5 所示),就可以完成 3-阶前缀支配优化表的填写(图 8 所示).首先,已知每个 3-阶最大组合的支配面积,发现$\{p_1,p_2,p_4\}$支配面积的前缀大于等于其后续组合的支配面积,因此省略计算$\{p_1,p_4,p_5\}$和$\{p_1,p_4,p_6\}$的前缀支配面积.而后,用式(5)依次计算 3-$LarCom(p_4)$对 p_5、p_6、p_7的前缀支配面积为 0.33、0.41 和 0.49.再计算 3-$LarCom(p_3)$对 p_4、p_5 的前缀支配面积为 0.135 和 0.31.由于 0.31<0.33,省略后续的计算,从而完成 3-阶前缀支配优化表的填写.

4-$LarCom(p_i)$		$\{p_1,p_2,p_3,p_4\}$ **0.655**	$\{p_1,p_2,p_4,p_5\}$ 0.67	$\{p_1,p_2,p_4,p_6\}$ 0.68	$\{p_1,p_2,p_4,p_7\}$ 0.68
		p_4(0.52)	p_5(0.34)	p_6(0.27)	p_7(0.19)
0.59	$\{p_1,p_2,p_3\}$	0.135	0.31	–	–
0.65	$\{p_1,p_2,p_4\}$		0.33	0.41	0.49
0.64	$\{p_1,p_4,p_5\}$			–	–
0.65	$\{p_1,p_4,p_6\}$				–

图 8 3-阶前缀支配优化表

根据算法 1 和 2 可知,通过完成 1~$(k-1)$-阶前缀支配优化表的填写就可以得到 k-MCS 结果.对每个 l-阶前缀支配表,不考虑优化策略时,需要填写最多 $\frac{(M-l+1)\times(M-l)}{2}$个值,其中 M 是整体 skyline 元组的数目.以此类推,因为前缀优化算法需要填写 1-阶到$(k-1)$-阶的前缀支配表,所以,在不考虑优化策略 1 和 2 时,前缀优化算法需要填 $\sum_{i=1}^{k-1}\frac{(M-i+1)\times(M-i)}{2}=\frac{3kM^2+(6-3k^2)M+k^3-3k^2+2k}{6}$ 个值,优化策略只减少计算量,并不降低时间复杂度,因此前缀优化算法的时间和空间复杂度为 $O(kM^2)$.

考虑到优化策略 1 和优化策略 2,在实际进行计算时,并不需要填写支配优化表中所有的值.在计算 l 阶前缀支配表时,最优情况仅需要填写 $O(M-l)$个值,而最坏情况才需要填写 $O((M-l)^2)$个值.而即便是最坏情况,算法需要的时间复杂度和空间复杂度仍然符合标准,因为通常情况下,整体 skyline 数目 M 的取值要远小于数据集大小,否则 skyline 计算将失去意义.

4.2 多维 k-MCS 的计算($d \geq 3$)

在文献[3]中,已经阐明了 k-MCS 问题在多维环境下的求解是十分复杂的.因此文献[3]采用了贪心方法来解决多维上的 k-MCS 问题.本章提出了一种改进的贪心算法来求解多维的 k-MCS 问题,借鉴了文献[32]的贪心算法的优化思想,但是应用了更高效的增量支配体积计算公式,所以本文算法拥有更高的时间效率.

4.2.1 基础贪心算法的描述

在介绍贪心算法之前,本文首先介绍文献[3]中提出的概念——增量支配体积.

定义 5(增量支配体积)[3]. 给出轮廓集合 SKY 中一个大小为 n 的子集 $N=\{s_1,s_2,\cdots,s_N\}\subseteq SKY$,如果一个元组 $s_i \in SKY$ 且 $s_i \notin N$,那么 s_i 相对于 N 的增量支配体积为 $DomSize(s_i)$-$IntSize(s_i,N)$,根据集合的容斥原理可以得到式(6):

$$\begin{aligned} IncreSize(s_i,N)=&DomSize(s_i)-\Big[\sum_{\forall s_j\in N} IntSize(\{s_i,s_j\})+\\ &(-1)^1\sum_{\forall S_j\in 2\text{-}Set(N)} IntSize(\{s_i\}\cup S_j)+\cdots+\\ &(-1)^{|N|-1} IntSize(\{s_i\}\cup N)\Big] \end{aligned} \tag{6}$$

根据定义 5,可以采用贪心算法来解决 k-MCS

问题，算法描述如算法 3 所示.

算法 3.　多维 k-MCS 问题的贪心解决策略

输入：多维 skyline 集合 SKY，参数 k

输出：近似的 k-MCS 结果集 G

1. 初始化结果集 G 为空；
2. 计算集合 SKY 中所有元组的支配体积；
3. 选择支配体积最大的元组加入到结果集 G 中；
4. WHILE(集合 G 的大小<=k)
5. 　FOR(SKY-G 中的每个轮廓元组 s_i)
6. 　　计算 s_i 相对于 G 的增量支配面积；
7. 　ENDFOR
8. 　把增量支配面积最大的元组加入到 G 中；
9. ENDWHILE

如图 9 所示，给定 7 个轮廓元组组成的集合 $\{s_1,s_2,s_3,s_4,s_5,s_6,s_7\}$，以及参数 $k=3$. 根据算法 3 所示，首先选出支配体积最大的元组 s_1 加入到结果集中；而后利用式(6)求出相对 $\{s_1\}$ 集合增量支配体积最大的元组 s_3，并把 s_3 加入到结果集中；最后，选出相对 $\{s_1,s_3\}$ 集合增量支配体积最大的元组 s_2，并把 s_2 加入到结果集中. 这时，结果集中含有 3 个元组，算法结束，贪心策略求得的结果集为 $\{s_1,s_3,s_2\}$.

	$DomSize(s_i)$	$IncreSize(s_i,\{s_1\})$	$IncreSize(s_i,\{s_1,s_3\})$
$s_1(0.5,0.5,0.45)$	**0.1375**	–	–
$s_2(0.8,0.2,0.2)$	0.128	0.073	**0.061**
$s_3(0.2,0.2,0.8)$	0.128	**0.078**	–
$s_4(0.3,0.8,0.2)$	0.112	0.057	0.049
$s_5(0.7,0.2,0.7)$	0.072	0.027	0.009
$s_6(0.4,0.7,0.6)$	0.072	0.012	0.006
$s_7(0.8,0.9,0.1)$	0.018	0.007	0.007

图 9　贪心策略解决 3 维 k-MCS 问题($k=3$)

然而，在计算过程中，我们发现，利用式(6)来求解一个元组的增量支配体积，会产生大量的重复运算和不必要的计算. 因此，我们提出改进的贪心算法策略，并给出相关的优化策略.

4.2.2　优化贪心算法的描述 OGA

针对基础贪心算法，我们从两个方面给出了优化策略. 首先改进了式(6)，可以避免贪心算法在计算过程中造成的重复运算；其次，提出了过滤策略，避免了一些元组参与计算，提高了计算效率.

优化策略 1. 式(6)的改进.

以图 9 为例，当结果集为 $\{s_1\}$ 时，用式(6)求解 $IncreSize(s_i,\{s_1\})$ 时，需要计算中间值 $IntSize(\{s_i,s_1\})$. 而当结果集为 $\{s_1,s_3\}$ 时，求解 $IncreSize(s_i,\{s_1,s_3\})$，需要计算中间值 $IntSize(\{s_i,s_1\})$，$IntSize(\{s_i,s_3\})$ 以及 $IntSize(\{s_i,s_1,s_3\})$. 通过比较，可以发现有重复项 $IntSize(\{s_i,s_1\})$. 随着结果集中的元组原来越多，每次加入新元组时都需要重复计算之前计算过的所有中间值. 因此，本文对式(6)提出了增量式的改进，可以利用 $IncreSize(s_i,\{s_1\})$ 的结果求解 $IncreSize(s_i,\{s_1,s_3\})$，避免计算重复的中间值.

假定集合 N^+ 比集合 N 多了一个元组 n，那么可以得到式(7)：

$$i\text{-}Sets(N^+)=i\text{-}Sets(N)\cup\{n\triangleright(i-1)\text{-}sets(N)\} \quad (7)$$

其中，i-$Sets(N)$ 表示在集合 N 中任取 i 个元组构成的所有组合的集合，$n\triangleright(i-1)$-$set(N)\}$ 表示在 $(i-1)$-$Sets(N)$ 中所包含的所有组合中都添加元组 n. 例如，给定 $i=3$，$N=\{s_1,s_2,s_3,s_4\}$，$N^+=\{s_1,s_2,s_3,s_4,n\}$，那么可以知道 3-$Sets(N)=\{\{s_1,s_2,s_3\},\{s_1,s_2,s_4\},\{s_1,s_3,s_4\},\{s_2,s_3,s_4\}\}$ 以及 2-$Sets(N)=\{\{s_1,s_2\},\{s_1,s_3\},\{s_1,s_4\},\{s_2,s_3\},\{s_2,s_4\},\{s_3,s_4\}\}$. 而 3-$Sets(N^+)$ 可以由 2 部分组成：(1)包含了 3-$Sets(N)$ 中所有的元素；(2)对 2-$Sets(N)$ 中的每个元素中添加一个元组 n. 所以，3-$Sets(N^+)=\{\{s_1,s_2,s_3\},\{s_1,s_2,s_4\},\{s_1,s_3,s_4\},\{s_2,s_3,s_4\},\{s_1,s_2,n\},\{s_1,s_3,n\},\{s_1,s_4,n\},\{s_2,s_3,n\},\{s_2,s_4,n\},\{s_3,s_4,n\}\}$.

根据式(7)，可以得到如式(8)：

$$\sum_{\forall S_j\in i\text{-}Set(N^+)} IntSize(\{s_m\}\cup S_j)=\sum_{\forall S_j\in i\text{-}Set(N)} IntSize(\{s_m\}\cup S_j)+\sum_{\forall S_j\in (i-1)\text{-}Set(N)} IntSize(\{s_m,n\}\cup S_j) \quad (8)$$

根据式(6)和(8)，可以用 $IncreSize(s_i,N)$ 的结果增量计算 $IncreSize(s_i,N^+)$ 的值，避免了大量的重复计算，具体如式(9)所示.

$$\begin{aligned}IncreSize(s_i,N^+)=&\ IncreSize(s_i,N)-\Big[IntSize(\{s_i,n\})+\\&(-1)^1\sum_{\forall S_j\in 1\text{-}Set(N)} IntSize(\{s_i,n\}\cup S_j)+\cdots+\\&(-1)^{|N|-1}\sum_{\forall S_j\in(|N|-1)\text{-}Set(N)} IntSize(\{s_i,n\}\cup S_j)+\\&(-1)^{|N|}IntSize(\{s_i,n\}\cup N)\Big]\end{aligned} \quad (9)$$

利用式(9)求解 $IncreSize(s_i,N^+)$，需要的时间复杂度为 $O(2^{|N|})$，而利用式(6)求解 $IncreSize(s_i,N^+)$，需要的时间复杂度为 $O(2^{|N|+1})$，所以利用式(9)求解增量支配面积可以减少一半的计算量.

下面，对式(9)进行扩展，如果集合 N^* 比集合 N 多出多个元组，即 $N\subseteq N^*$，且 $N^*-N=\{s_1',s_2',\cdots,s_n'\}$，$\forall s_i\notin N^*$，已知 $IncreSize(s_i,N)$，可以用式(10)求解 $IncreSize(s_i,N^*)$.

$$IncreSize(s_i,N^*)=$$
$$IncreSize(s_i,N)-\Big[\sum_{\forall s_j\in N^*-N}IntSize(\{s_i,s_j\})+$$
$$(-1)^1\sum_{\forall S_j\in 2\text{-}Set(N^*)-2\text{-}Set(N)}IntSize(\{s_i\}\cup S_j)+\cdots+$$
$$(-1)^{|N|-1}\sum_{\forall s_j\in N^*-N}IntSize(\{s_i,s_j\}\cup N)+\cdots+$$
$$(-1)^{|N^*|-|N|-1}IntSize(N^*)\Big] \tag{10}$$

对比式(6)和(10),当已知 $IncreSize(s_i,N)$的值,用式(10)计算 $IncreSize(s_i,N^*)$,可以少算$O(2^{|N|})$个值.

如图9所示,已知 $IncreSize(s_4,\{s_1,s_3\})=0.049$. 利用式(9),可以求得 $IncreSize(s_4,\{s_1,s_2,s_3\})=IncreSize(s_4,\{s_1,s_3\})-[IntSize(\{s_4,s_2\})-IntSize(\{s_4,s_2,s_1\})-IntSize(\{s_4,s_2,s_3\})+IntSize(\{s_4,s_1,s_2,s_3\})]=0.049-(0.032-0.022-0.008+0.008)=0.039$.

优化策略2. 减少冗余元组的增量计算.

观察图9中的例子,可以发现如下引理.

引理3. 对于任意元组 s_i,如果集合 $S_1\subseteq S_2$,那么 $IncreSize(s_i,S_1)\geq IncreSize(s_i,S_2)$.

证明. 根据定义5,可以很容易得到该结论.

以图9为例,所有的元组都按照其支配体积进行由小到大排序后,当结果集为$\{s_1\}$时,可以计算出 $IncreSize(s_3,\{s_1\})=0.078$,而排在 s_5 之后的元组的支配体积都小于等于0.072,由此可见,贪心算法在选择第二个结果元组时,一定不会选择排在 s_5 之后的元组,因此不需要计算排在 s_5 之后的元组的精确增量体积. 根据此观察,可以得到如下定理.

定理3. 用 G_i 表示贪心策略执行 i 次后,共含有 i 个结果元组. 如果 $\forall s_m,s_n\notin G_i$, $\exists j<i$,并且 $IncreSize(s_m,G_i)\geq IncreSize(s_n,G_j)$,那么 s_n 一定不会是贪心策略的第 $i+1$ 个结果元组.

证明. 根据增量支配体积的定义可知,$\forall s_n\notin G_i$ 且 $j<i$,那么 $IncreSize(s_n,G_j)\geq IncreSize(s_n,G_i)$,根据假设已知 $IncreSize(s_m,G_i)\geq IncreSize(s_n,G_j)$,那么 $IncreSize(s_m,G_j)\geq IncreSize(s_n,G_j)$,所以 s_n 一定不会成为贪心策略的第 $i+1$ 个结果元组.

为了使用定理3快速确定贪心策略中的结果元组,我们希望尽快地得到那些更大的增量支配体积作为过滤值. 根据简单的观测,可以发现,通常支配体积大的元组,针对任意贪心结果集合 G_i 的增量支配体积也容易更大,因此我们把所有元组按照其支配体积进行由大到小的排序,并形成集合$\{s_1,s_2,\cdots,s_n\}$. 在贪心算法的每次迭代过程中,我们依次计算元组相对于当前贪心结果集 G_j 的增量支配体积,并把得到的最大增量支配体积作为 G_j 的过滤值,记作 t_j.

为了达到快速过滤的效果,对于还没有计算真实增量支配体积的元组 s_i,可以快速计算出 s_i 的一个上界值. s_i 的上界值表示的是该元组针对当前贪心结果集 G_j 的增量支配体积的上界,记作 $upbound(s_i)$. 通过比较当前贪心结果集的过滤值和每个元组的上界值,可以快速完成元组的过滤.

上界值的选取:为了达到快速过滤的效果,对于元组 s_i 的上界值采用如下过程进行计算:初始化时,s_i 的初始上界值是 s_i 的支配体积;当 $upbound(s_i)$ 大于当前过滤值时,s_i 的上界值更新为 $DomSize(s_i)-\max\{\forall s_j\in G_j, IntSize(\{s_i,s_j\})\}$,记作 s_i 针对 G_j 的更新上界值;如果该上界值依然大于过滤值,则计算 s_i 对当前贪心结果集 G_j 的增量支配体积,并把该支配体积作为 s_i 的最新上界值.

结合优化策略1和优化策略2,我们可以给出优化贪心算法的描述:(1)初始时,计算每个元组的支配体积,并把该体积作为每个元组的初始上界值,而后选择支配体积最大的元组加入到贪心结果集合中;(2)在贪心算法的第 $j+1$ 次迭代时,设贪心集合 G_j 的初始过滤值 $t_j=0$. 按照元组的支配体积大小依次处理元组 s_i:如果 s_i 的上界值 $upbound(s_i)$ 大于 t_j,快速求得 s_i 的更新上界值为 $DomSize(s_i)-\max\{\forall s_j\in G_j, IntSize(\{s_i,s_j\})\}$. 如果 s_i 的更新上界值依然大于 t_j,利用式(9)或(10)求得 s_i 针对 G_j 的增量支配体积 $IncreSize(s_i,G_j)$,并把 s_i 的上界值 $upbound(s_i)$ 更新为 $IncreSize(s_i,G_j)$. 同时如果满足 $IncreSize(s_i,G_j)$ 大于 t_j,把 t_j 更新为 $IncreSize(s_i,G_j)$;(3)重复过程(2),直到贪心结果集合中有 k 个元组,算法结束. 优化贪心算法的过程如算法4所示.

算法4. 优化贪心算法的描述过程.

输入:多维 skyline 集合 SKY,参数 k

输出:近似的 k-MCS 结果集 G

1. 初始计算每个元组 s_i 的支配体积 $DomSize(s_i)$;
2. 设每个元组 s_i 的初始上界值 $upbound(s_i)=DomSize(s_i)$;
3. 把所有元组按照支配体积由小到大排序;
4. 选择支配体积最大的元组加入到结果集 G 中;
5. WHILE(集合 G 的大小<=k)
6. 初始化当前 G 的过滤值 $t=0$;
7. 初始当前支配体积最大元组 max_t 为空;
8. FOR(依次处理 SKY-G 中的每个元组 s_i)

```
9.    IF (upbound(s_i)>t 且 upbound(s_i)==
        DomSize(s_i))
10.     upbound(s_i)=DomSize(s_i)-max{∀s_j∈G,
        IntSize({s_i,s_j})} //计算更新上界值
11.   ENDIF
12.   IF(upbound(s_i)>t)
13.     用式(9)或(10)计算 IncreSize(s_i,G);
14.     upbound(s_i)=IncreSize(s_i,G);
15.   ENDIF
16.   IF(IncreSize(s_i,G)>t)
17.     max_t=s_i;
18.     t=IncreSize(s_i,G);
19.   ENDIF
20. ENDFOR
21. 把 max_t 元组加入到 G 中;
22. ENDWHILE
```

如算法 4 中的第 10 行所示，当元组的初始上界值无法满足过滤要求时，为了避免冗余计算，我们首先快速计算出元组的更新上界值用来过滤. 通常情况下利用更新上界值，就可以过滤掉大量的非结果元组. 因此，我们不直接计算该元组的增量支配体积.

为了避免重复计算，在优化贪心算法执行过程中，我们需要保留如下值：每个元组的支配体积以及一个相交支配体积表. 相交支配体积表用来保存任意两两元组间的相交支配体积. 该表格不仅有助于更新上界值的计算，其保存的结果在计算增量支配体积时，也会被频繁用到.

如图 10 所示，是轮廓集合 $\{s_1, s_2, \cdots, s_7\}$ 的相交支配体积表，保留了任意两个元组之间的相交支配体积，如 $IntSize(\{s_1, s_3\})=0.05$. 利用相交支配体积，可以快速确定任意元组的更新上界值. 如果当前贪心集合 $G=\{s_1, s_3\}$，对于元组 s_4，可以确定它的更新上界值为 $Domsize(s_4)-\max\{0.055, 0.028\}=0.057$.

	s_1	s_2	s_3	s_4	s_5	s_6
s_1(0.5,0.5,0.45)	–	–	–	–	–	–
s_2(0.8,0.2,0.2)	0.055	–	–	–	–	–
s_3(0.2,0.2,0.8)	0.05	0.032	–	–	–	–
s_4(0.3,0.8,0.2)	0.055	0.032	0.028	–	–	–
s_5(0.7,0.2,0.7)	0.045	0.048	0.048	0.018	–	–
s_6(0.4,0.7,0.6)	0.06	0.024	0.036	0.048	0.027	–
s_7(0.8,0.9,0.1)	0.011	0.016	0.004	0.016	0.005	0.008

图 10　相交支配体积表

如图 11 所示，是优化贪心算法的执行过程. 首先，选择计算每个元组的支配体积，并作为它们的初始上界值. 按元组的支配体积进行由大到小排序后，选择 s_1 加入到贪心集合 G 中；在第 2 次迭代时，首先计算 $IncreSize(s_2, G_1)=0.073$，并得到 $t_1=0.073$. 而后计算 $IncreSize(s_3, G_1)=0.078$，并更新 $t_1=0.078$. 由于 $Domsize(s_4)=0.112>t_1$，需要计算 s_4 的更新上界值为 0.057，并过滤掉 s_4. 对于后续的所有元组，初始上界值就小于 0.078，可以直接过滤，直接把 s_3 加入到结果集中 $G_2=\{s_1, s_3\}$；在第 3 次迭代时，首先计算 $IncreSize(s_2, G_2)=0.061$，由于 0.061 大于后续所有元组的更新上界值，可以直接确定把 s_2 加入到结果集中 $G_3=\{s_1, s_3, s_2\}$；第 4 次迭代时，首先计算 $IncreSize(s_4, G_3)=0.061$，后续的所有元组的上界值都小于该过滤值，因此把 s_4 加入到贪心结果集中. 4 次迭代后贪心结果集中共有 4 个结果元组，$G_4=\{s_1, s_3, s_2, s_4\}$.

	$DomSize(s_i)$	$upbound(s_i)$	$IncreSize(s_i,\{s_1\})$	$upbound(s_i)$	$IncreSize(s_i,\{s_1,s_3\})$	$upbound(s_i)$	$IncreSize(s_i,\{s_1,s_2,s_3\})$
s_1(0.5,0.5,0.45)	**0.1375**	–	–	–	–	–	–
s_2(0.8,0.2,0.2)	0.128	0.073	0.073	–	**0.061**	–	–
s_3(0.2,0.2,0.8)	0.128	0.078	**0.078**	–	–	–	–
s_4(0.3,0.8,0.2)	0.112	0.057	0.057	0.057	–	–	**0.039**
s_5(0.7,0.2,0.7)	0.072	0.072	–	0.024	–	0.024	–
s_6(0.4,0.7,0.6)	0.072	0.072	–	0.036	–	0.036	–
s_7(0.8,0.9,0.1)	0.018	0.018	–	0.007	–	0.007	–

图 11　优化贪心算法执行过程($k=4$)

利用贪心策略计算 k-MCS 结果时，可以发现，随着加入到贪心结果集中的元组不断变多，计算每个元组的增量支配体积的计算量是呈指数级增加的，且后加入的元组的增量支配体积也越来越小，为此，为了避免后续庞大的计算量，我们提出了提前终止的贪心优化算法.

4.2.3　ε-截断优化贪心算法的描述 ε-OGA

ε-截断优化贪心算法可以提前终止贪心算法的计算. 具体地，当贪心算法迭代到 m 次时，贪心结果集中只有 m 个结果元组. 如果 $m<k$，结果集中的元组数目不够，仍需要选出新的结果元组. 根据当前上界值，可以快速确定未加入元组的最大增量支配体积. 如果最大增量支配体积只占当前结果集支配体积的很小一部分，可以称后续所有元组对增量支配体积的贡献很小，可以提前终止贪心计算. 对于后续的元组不需要知道它们精确的增量支配体积，只需

要进行一个上界估计，并把需要的元组加入到的结果集中.

具体地，设定一个阈值 ε，在进行第 m 次贪心迭代时，得到增量支配体积最大的元组为 s_m，把 s_m 加入结果集得到 G_m. 对于所有未加入的元组，它们的上界值一定小于 s_m 的增量支配体积. 当它们中最大的 $k-m$ 个上界值之和小于 $\varepsilon \times DomSize(G_m)$，贪心算法提前终止. 同时更新所有元组的增量支配体积上界，并选择其中 $k-m$ 个上界值最大的元组加入到结果集中.

根据上述过程，已知第 m 次贪心迭代后的结果集 G_m 的支配体积为 $DomSize(G_m)$ 时，根据引理 3，贪心算法 OGA 得到的结果集合 G_k 的支配体积 $DomSize(G_k) \leq DomSize(G_m)+\varepsilon \times DomSize(G_m)$. 而算法 ε-OGA 得到的结果集合 G'_k 的支配体积 $DomSize(G'_k) > DomSize(G_m)$，因此可以得到 $\frac{DomSize(G'_k)}{DomSize(G_k)} > \frac{1}{1+\varepsilon}$. 因此，对比算法 OGA，$\varepsilon$-OGA 算法仅牺牲了 $\varepsilon/(1+\varepsilon)$ 的精度，就可以获得大量的速度提升.

增量支配体积的更新策略：对于任意没有加入结果集合的其它数据元组，按照其支配体积进行排序，并保留之前计算出的增量支配体积上界. 而后，对于任意元组 s_i，计算 s_i 和当前结果集中每个元组的相交支配体积，并选取其中相交支配体积最大的 d（维度值）个元组组成集合 $d\text{-}IntSet(s_i)$，求 s_i 对集合 $d\text{-}IntSet(s_i)$ 的增量支配体积，如果该体积小于 s_i 的原上界，则把 s_i 的上界值更新为该增量支配体积. 而后，选择上界值最大的元组加入到贪心结果集中. 按照此方法进行迭代，直到结果集中的元组个数为 k 个为止. ε-截断优化贪心算法描述如算法 5 所示.

算法 5. ε-截断优化贪心算法的描述过程.

输入：多维 skyline 集合 SKY，参数 k，截断值 ε

输出：近似的 k-MCS 结果集 G

1. 初始设置贪心结果集 $G=\varnothing$；
2. WHILE(集合 G 的大小<=k)
3. 按算法 4 中的方法把支配体积最大的元组加入到结果集 G 中；
4. sum := 不在 G 中的最大的 $k-m$ 个上界值之和；
5. IF(sum<G 的支配体积)
6. BREAK；
7. ENDIF
8. ENDWHILE
9. WHILE(集合 G 的大小<=k)
10. FOR(依次处理 SKY-G 中的每个元组 s_i)
11. 在 G 中选出跟 s_i 相交支配体积最大的 d 个元组组成集合 $d\text{-}IntSet(s_i)$；
12. 计算 s_i 对 $d\text{-}IntSet(s_i)$ 的增量支配体积 $newup$；
13. IF($newup$<$upbound(s_i)$)
14. $upbound(s_i)=newup$；
15. ENDIF
16. ENDFOR
17. 选择上界值最大的元组加入到 G 中；
18. ENDWHILE

算法 5 的 11 行中，之所以选择跟 s_i 相交支配体积最大的 d 个元组，是因为在 d 维空间中，选择与 s_i 重叠部分最大的 d 个元组，能够在保证计算效率的同时，尽可能准确地估计 s_i 的上界值. 值得注意的是，当维度 $d>10$ 时，只选择相交体积最大的 10 个元组. 原因是超过 10 个元组后，计算每个元组的增量支配体积将耗费大量的时间.

如图 11 和图 12 所示，给出了 ε-截断优化贪心算法的举例. 给定 $\varepsilon=0.4$，$k=5$ 时，如图 11 所示，按照算法 4 进行了 3 次迭代后，贪心结果集 $G=\{s_1, s_3, s_2\}$，此时 G 的支配体积为 0.1357+0.078+0.061=0.2747. 剩余上界值最大的 2 个元组为 $\{s_4, s_6\}$，它们的上界值之和为 $0.057+0.036=0.093<\varepsilon \times 0.2747$，因此可以提前结束算法 4，采用算法 5 的 9～18 行的截断方法来选取剩余的 2 个元组. 如图 12 所示，当前 $G=\{s_1, s_3, s_2\}$，查阅图 10，对于任意剩余元组 s_i，在 G 中与 s_i 相交支配体积最大的 3 个元组都是 s_1, s_2, s_3，因此，计算 s_i 相对于 $\{s_1, s_3, s_2\}$ 的增量支配体积作为每个元组最新上界值，并选取上界值最大的元组 s_4 加入到结果集合中，$G=\{s_1, s_3, s_2, s_4\}$. 之后再对剩余的元组从 G 中选取相交支配体积最大的 3 个元组，其中，s_5 的 3 个最大相交元组为 $\{s_1, s_2, s_3\}$，s_6 的 3 个最大相交元组为 $\{s_1, s_3, s_4\}$，s_7 的 3 个最大相交元组为 $\{s_1, s_2, s_4\}$，而后计算相应的增量支配体积作为第 5 次迭代的上界. 而后选择上界最大的元组 s_5 加入到结果集中. 最终的贪心结果集为 $G=\{s_1, s_3, s_2, s_4, s_5\}$.

	$G=\{s_1,s_2,s_3\}$		$G=\{s_1,s_2,s_3,s_4\}$	
	Max3 Tuples	*upbound*(s_i)	*Max3 Tuples*	*upbound*(s_i)
s_1(0.5, 0.5, 0.45)	–	–	–	–
s_2(0.8, 0.2, 0.2)	–	–	–	–
s_3(0.2, 0.2, 0.8)	–	–	–	–
s_4(0.3, 0.8, 0.2)	s_1, s_2, s_3	0.039	–	–
s_5(0.7, 0.2, 0.7)	s_1, s_2, s_3	0.003	s_1, s_2, s_3	0.003
s_6(0.4, 0.7, 0.6)	s_1, s_2, s_3	0.006	s_1, s_2, s_4	0.002
s_7(0.8, 0.9, 0.1)	s_1, s_2, s_3	0.002	s_1, s_2, s_4	0.002

图 12 ε-截断优化贪心算法执行过程($\varepsilon=0.4$，$k=5$)

根据算法 3 到算法 5 的描述可以发现，基础贪心算法 3 的时间复杂度为$\sum_{i=0}^{k}(M-i)\times 2^i=O((M-k+1)\times 2^{k+1}-M)$，其中 M 是整体 skyline 元组的数目. 算法4 中优化贪心算法的时间复杂度为$O((M-k+1)\times 2^k-M)$，优化策略 1 可以减少一半的计算代价. 优化策略 2 可以过滤掉一些冗余元组的计算，但是不会改变算法的时间复杂度. 同时，优化贪心算法需要保存一个相交支配表，并对每个 skyline 元组存一个 *bound* 值一个增量值，所以优化贪心算法的空间复杂度是 $O(M^2)$.

在算法 5 中，如果在第 $L(L<k)$ 次达到截断条件，那么算法5的时间复杂度为$O((M-L)\times 2^{L-2}+\left(M-\frac{k+L}{2}\right)\times 2^{d+1}\times(k-L))$. 由此分析可知，算法 5 的时间复杂度最低，处理最快. 算法 5 对于每个元组，需要保留一个 *bound* 值，以及 d 个与该元组覆盖体积最大的元组. 同时，算法 5 也需要保留一个相交支配表，因此算法 5 的空间复杂度也是 $O(M^2)$. 值得注意的是，如果 ε 值设的过小，算法提前终止时，L 过大，影响算法效率，为此，我们通过选择合适的 ε 值，使得算法终止时，L 值不超过 10.

5　实验分析

5.1　实验设置

在本节中，使用 Visual C++语言实现了算法 OPA、OGA 和 ε-OGA. 实验环境为 Intel i7-4790 CPU@3.6GHz；8GB 内存；1TB 硬盘和 Windows 7 操作系统.

在本节中，将本文算法 OPA、OGA 和 ε-OGA 和文献[3]中的 PBA、GA、ε-GA 算法，文献[31]中的 RT 算法进行了比较. 其中，OPA 是本文的二维 k-MCS 处理算法；PBA 是文献[3]的二维 k-MCS 处理算法；OGA、ε-OGA 是本文的多维 k-MCS 处理算法；GA、ε-GA 是文献[3]中的多维 k-MCS 处理算法，RT 是文献[31]中的贪心优化算法(按文献[31]进行了参数设置 $l=8$，error$=0.1$). 本文分别用真实数据和合成数据验证算法性能. 真实数据采用的是股票数据和森林防火监控数据. 股票数据(来自网站 https://uqer.io/，共包含十二个月内的共 798532 条股票记录，每条股票记录包含 4 个属性：日成交量、股票涨幅率、今日最低价和流通市值). 按照日成交量、股票涨幅率越大越好、最低价和流通市值越小越好的原则来选取 skyline 元组. 对于二维算法，我们选取了流通市值和最低价两个维度来进行实验，共包含 29 个 skyline 元组. 而对于多维算法，选用了股票记录的所有 4 个维度，共包含 453 个 skyline 元组. 森林防火监控数据共包含 30949 个元组，4 个维度，分别是温度、湿度、光照、气压，按照维度、光照、气压越小越好，湿度越大越好的原则得到了 109 个 skyline 元组.

如表 2 中所示，在处理二维数据时，本文提出的算法 OPA 和算法 PBA[3] 都是准确算法，因此，它们计算的支配面积相等. 但是 OPA 的计算效率要比 PBA 算法高. 是因为已知 l-最大组合来求解$(l+1)$-最大组合时，PBA 和 OPA 分别采用 l-最大区间[3] 和 l-阶前缀支配表来求$(l+1)$-最大组合. 尽管 l-最大区间中需要填写的值较少，但是每个值的填写需要消耗 $2l$ 次乘法以及 $4l$ 次加减法. 而 l-阶前缀支配表中填写的值较多，但是每个值的填写仅需要 1 次加法. 主要原因是 l-最大区间不能复用之前$(l-1)$-最大区间中计算过的值，而 l-阶前缀支配表可以复用$(l-1)$-阶前缀支配表中的已知值. 因此，OPA 的算法效率要好于 PBA.

表 2　真实数据集结果

	算法		时间/ms	支配体积
	二维算法			
股票数据集	OPA	$k=10$	11.7843	0.9479
		$k=15$	13.6122	0.9481
	PBA	$k=10$	23.6234	0.9479
		$k=15$	25.3123	0.9481
	多维算法			
股票数据集	OGA	$k=10$	2898.94	0.5705
		$k=15$	259091	0.5821
	ε-OGA ($\varepsilon=0.01$)	$k=10$	1410.37	0.5692
		$k=15$	7050.23	0.5809
	GA	$k=10$	24678.91	0.5705
		$k=15$	1271545.12	0.5821
	ε-GA ($\varepsilon=0.01$)	$k=10$	1201.32	0.5689
		$k=15$	4836.18	0.5801
	RT ($l=8,e=0.1$)	$k=10$	4019.4	0.5201
		$k=15$	107749	0.5212
森林防火监控数据	OGA	$k=10$	1931.63	0.8719
		$k=15$	76896.6	0.8720
	ε-OGA ($\varepsilon=0.01$)	$k=10$	150.49	0.8719
		$k=15$	204.22	0.87195
	GA	$k=10$	4140.63	0.8719
		$k=15$	171601	0.8720
	ε-GA ($\varepsilon=0.01$)	$k=10$	6.02	0.8717
		$k=15$	45.667	0.8718
	RT ($l=8,e=0.1$)	$k=10$	1175	0.8717
		$k=15$	55276	0.8718

在处理多维数据时，本文提出的算法和文献[3,31]中的算法都是近似算法，采用贪心原则来求解 k-MCS 问题. 其中 OGA 算法和 GA 算法所求的结果集相同，得到的支配体积也相同，但是 OGA 的运行效率要远远优于 GA 的运行效率. 这是因为 OGA 与 GA 相比，利用 4.22 节中优化策略 1，运用已知的中间结果，可以节省 50%的计算量来得到增量支配体积. 利用优化策略 2，避免了大量不必要元组的增量体积计算. 因此 OGA 要远好于 GA. 而 ε-OGA 和 ε-GA 都是对贪心算法的提前终止，照比 ε-GA 算法，算法 ε-OGA 在计算效率上比 ε-GA 略差，但是在计算准确性上有所提升. 尤其是随着 k 值的增加，ε-GA 的精确度下降比较严重，因为其采用了一种简单的误差截断方法，即达到误差条件后，后续元组不再进行计算，直接加入结果集. 而本文的算法 ε-OGA 在满足截断条件后，后续元组依然会进行快速过滤，从而在保证效率的基础上，提供了更高的精确度. RT 算法采用的加速贪心策略，通过一些方案，对原始数据集进行一定舍弃后，得到 P、Q 候选集合，如果 P 集合中元组数目超过 k 个，则在 P 集合中采用贪心策略求出最终结果. 如果 P 集合元组数目少于 k 个，则把 Q 中元组对当前 P 增量支配体积最大的元组加入 P 中. 这就导致随着 k 值的增加，RT 算法的时间消耗与 OGA 和 GA 算法一样，也呈指数级增加，效率比 OGA 和 GA 好，但是精度不如 OGA 和 GA. 数据维度不超过 7 时，skyline 元组间的相交支配体积较大，本文截断算法 ε-OGA 和 ε-GA 中的参数 ε 可以很快发挥作用，因此，算法 ε-OGA 和 ε-GA 在结果集不到 8，就可以提前结束运算，保证 ε 精度的前提下，有极好的时间效果. 而 RT 算法需要在缩减的数据集中采用贪心策略进行计算，并不能提前结束算法，所以在维度不高的数据集中，当 k 值较大时，RT 算法的效率远不如 ε-OGA 和 ε-GA，同时精度也不如 ε-OGA 和 ε-GA.

合成数据是由 skyline 查询标准测试数据生成器[1]生成的，包括独立、相关和反相关数据集. 由于相关数据集中的 skyline 元组数目过少，skyline 元组数目甚至不超过 k 值，无法有效验证本文算法性能. 因此，本文只采用独立和反相关数据集来测试算法性能.

独立数据[1]：数据是随机分布的.

反相关数据[1]：当所有的数据都标准化到[0,1]范围后，经过点〈0,0,…,0〉，〈1,1,…,1〉可以确定一条直线 line，垂直于 line 的平面有无数多个，将最中间的平面记为 plant. 首先，以 plant 为均值，一个非常小的值作为标准差，采用正态分布概率选取这些平面；接着，在选定的平面内，以均匀分布来生成数据点. 反相关分布中，元组在某一维上表现的“好”，在其它几维或 1 维上的表现就会“较差”. 因此，反相关数据集中的 skyline 元组数目最多. 生活中的反相关分布是经常出现的. 例如文献[3]中所列举的酒店各维属性分布就满足反相关分布.

合成数据的主要参数变化范围及默认值如表 3 所示. 因数据成反相关分布时，skyline 结果较多，skyline 数据元组的支配能力较弱，故 ε 的默认值设得较大.

表 3 实验参数

参数	默认值	变化范围
数据集大小	10^6	10^5, 5×10^5, **10^6**, 5×10^6, 10^7
数据维度	4	2, 3, **4**, 5, 6
k 值	15	10, 12, **15**, 17, 20
ε	0.01(随机)	**0.01**, 0.05, 0.1, 0.5
	0.1(反相关)	0.01, 0.05, **0.1**, 0.5

为了获得稳定的实验结果，在真实数据集中，我们重复了 10 次实验，求得了运行时间的平均值. 而在合成数据集中，我们每次生成了 5 组不同的数据集，记录的是 5 组数据集结果的平均值. 由于算法 GA 和 OGA 在某些数据集中运行的时间过长，因此，对于所有的反相关数据集，$k>15$ 时，和维度>4 的数据集，我们只运行了一次 GA 和 OGA 算法，其他算法都是 5 组数据集的结果平均值.

5.2 二维算法的性能测试

本节测试了 2 维准确算法的性能，对比了本文算法 OPA 和文献[3]中的算法 PBA. 由于在独立分布下，二维数据集下 skyline 元组数目不够多，所以，本小节仅比较了反相关数据分布下算法的性能影响，主要记录了算法的处理时间以及结果集的支配面积大小与 skyline 全集支配面积的对比(图中标记为 fullsky).

5.2.1 数据集大小对算法性能的影响

本小节主要比较了数据集大小对算法性能的影响，如图 13 和图 14 所示. 由图 13 可以看出，随着数据量的增加，skyline 元组的数目相应增加，会导致算法的运行时间呈线性增加. 由图 14 可以看出，随着数据量增加，结果集的支配面积也相应增加，这是由于随着数据量的增加，skyline 元组数目将变多，使得 skyline 元组的支配面积也能相应增加. 图中 fullsky 表示的是整体 skyline 集合的支配面积.

由图 13 和图 14 可以发现，本文提出算法 OPA 比算法 PBA 有更好的运行效率，约提高 20%左右. 同时，OPA 和 PBA 都是准确算法，拥有相同的支配面积. 图 13 中括号里面的数据是对应数据集的全部 skyline 元组数目，后续的图 17 和图 19 中横坐标括号里面的数据都是对应数据集中的完整 skyline 集合的元组数目.

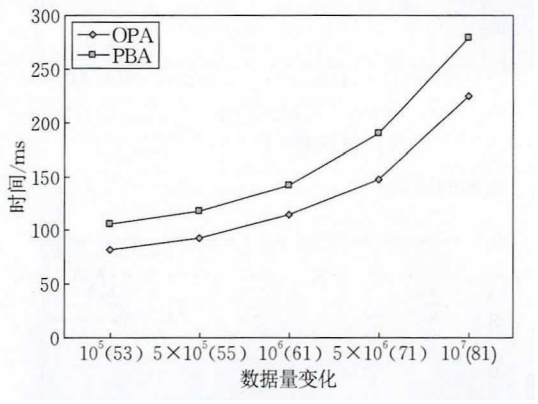


图 13　数据量变化对算法运行时间的影响

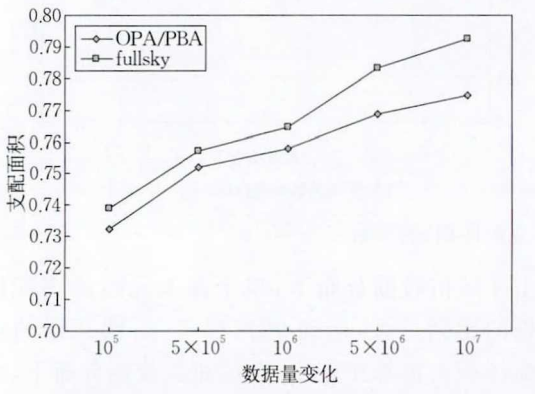


图 14　数据量变化对支配面积(支配体积)的影响

5.2.2　k 值变化对算法性能的影响

本节主要比较了 k 值变化对算法性能的影响，如图 15 和图 16 所示. 由图 15 可以看出，随着 k 值的增加，算法的查询时间将线性增加. 本文算法 OPA 的性能比 PBA 更好，提高约 20%. 由图 16 可以看出，

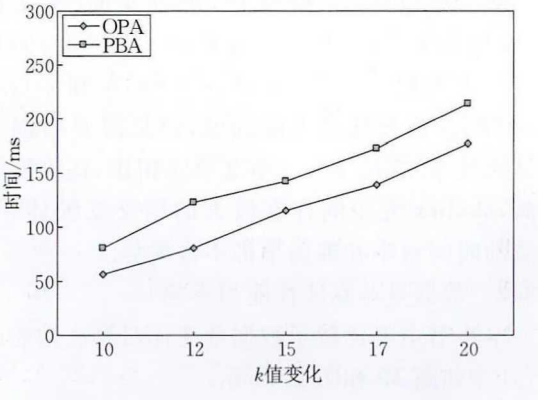


图 15　k 值变化对算法运行时间的影响

随着 k 值增加，选出的结果集的支配面积将变大，慢慢贴近整体 skyline 的支配面积. 综上，本文算法 OPA 照比 PBA 具有更高的效率和相同的准确率.

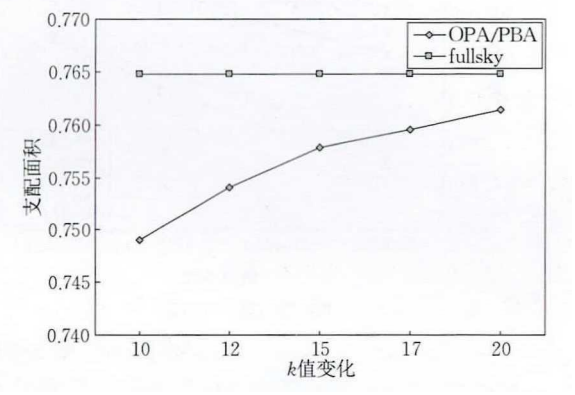


图 16　k 值变化对支配面积(支配体积)的影响

5.3　多维算法的性能测试

本节测试了多维近似算法的性能，对比了本文算法 OGA，ε-OGA 与文献[3]中的 GA、ε-GA 算法，以及文献[31]中的算法 RT. 主要记录了算法的响应时间，以及结果的支配体积的对比. 由于图例中无法引入符号 ε，因此，后续图中图例为 e-OGA 代表算法 ε-OGA，e-GA 代表算法 ε-GA.

5.3.1　维度变化对算法性能的影响

本节主要比较了维度变化对算法性能的影响，具体如图 17 和图 18 所示.

如图 17 所示，随着维度的增加，数据集的 skyline 结果数目是呈指数级增加的，因此，导致算法的运行时间也是呈指数级增加的. 对比图 17(a)和(b)可以发现，反相关数据集中的算法运行时间要远超同参数下的独立数据集中的算法运行时间. 这是由于在同参数下，反相关数据集中的 skyline 结果数目要远多于独立数据集中的 skyline 数目. 对比算法 ε-OGA 和 ε-GA，可以发现 ε-OGA 的运行效率略差于 ε-GA，在随机数据集中，随着维度的增加，时间相差比例由 45%降低到 30%左右，差值由 0.02s 上升到 10s. 在反相关数据集中，ε-OGA 的运行效率比 ε-GA 差，比例由 40%降低到 20%左右，差值由 1s 上升到 200s 左右. 这是由于 ε-OGA 与 ε-GA 相差的时间取决于终止算法时结果集的大小，结束越晚，相差的比例越少，结束越早，相差的比例越多，但是相差时间很短. 随着维度增加，ε-OGA 需要额外计算的时间增加，但是因为算法总体运行时间的增加，额外计算占据的比例减少. 对比 OGA 和 GA 算法，OGA 算法的运行效率要远比 GA 算法好. 在随机数据集中，随着维度增加，时间相差比例由 70%下降

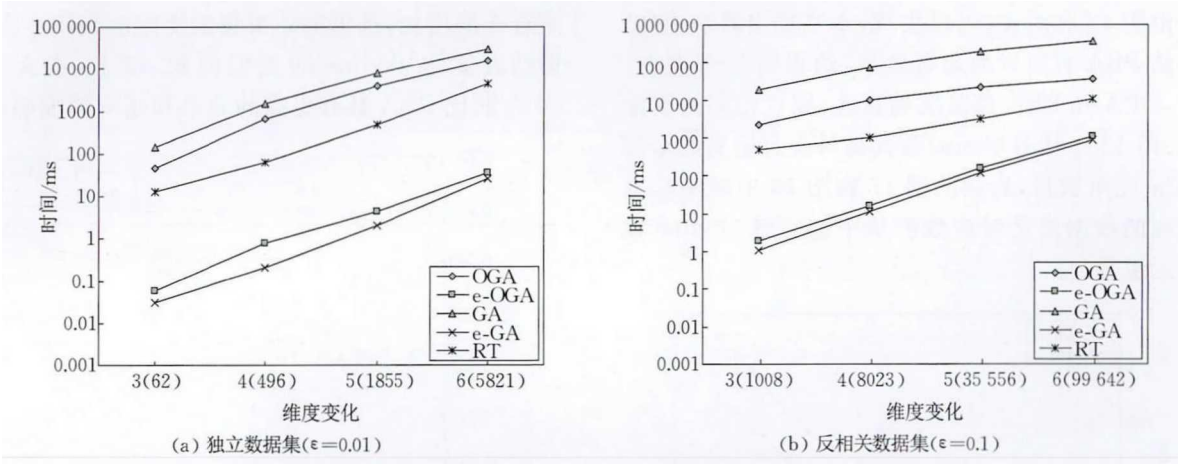


图 17 维度变化对算法运行时间的影响

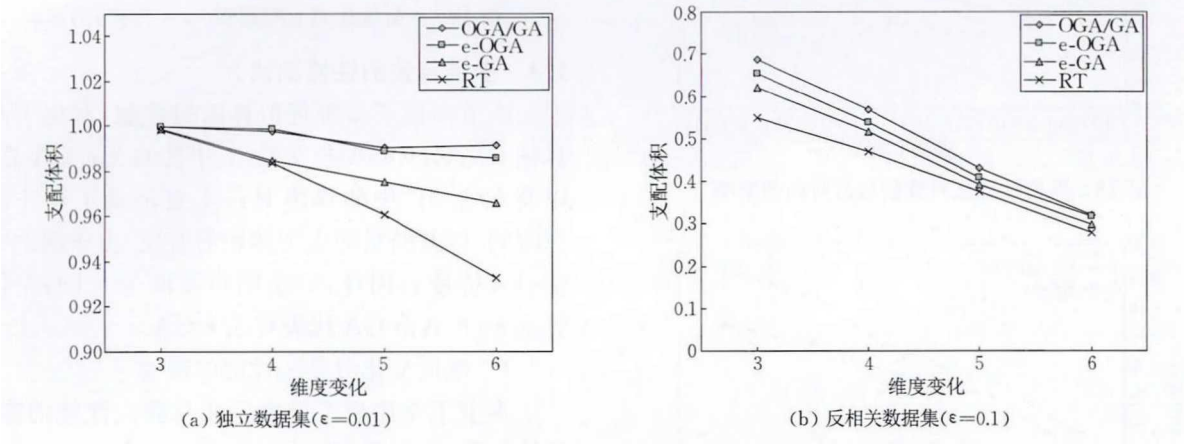


图 18 维度变化对支配面积(支配体积)的影响

到 50%,相差时间由 100s 上升到 15000s. 在反相关分布中,相差比例由 96%下降到 89%,相差时间由 1600s 上升到 330000s. 这是由于随着维度增加,优化策略 2 能过滤掉的元组比例减少,而反相关分布中,优化策略 2 能过滤掉的元组比例较高,所以时间效率提升更加明显. 算法 RT 的运行时间比 OGA 和 GA 要好,但是却比 ε-OGA 和 ε-GA 慢很多,这是由于 ε 值的引入,导致算法 ε-OGA 和 ε-GA 可以在结果集等于 10 时提前终止,而算法 RT 尽管候选集数目有所减少,却需要计算到 $k=15$. 而根据时间复杂度分析可知,影响贪心算法效率的最主要因素是 k 值的大小,因此,RT 的运行时间比 ε-OGA 和 ε-GA 慢很多,是 ε-OGA 和 ε-GA 算法运行时间的几百倍.

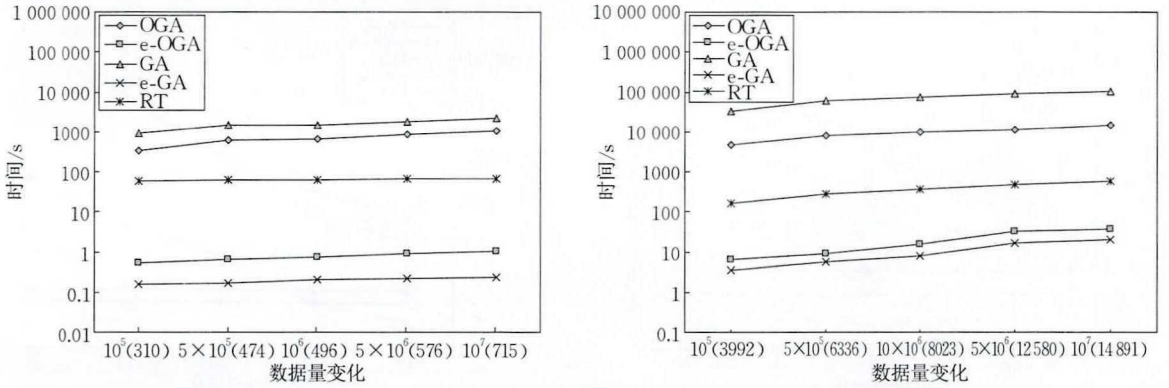
如图 18 所示,随着维度的增加,算法的支配体积是不断减少的,这是由于 skyline 点的性质导致的. OGA 和 GA 算法具有相同的支配体积,即它们具有相同的准确度. 而相同数据集下,算法 ε-OGA 的支配体积要比 ε-GA 高,在随机数据集下,准确度提高在 1%左右,随着维度增加,提升比例上升,这是由于随机数据分布下,单个最大元组的支配体积占据的比例过大,达到 90%以上,后续元组的增量支配体积占据都比较小. 在反相关数据分布下,精度提升在 10%左右,并随着维度增加,提升比例增加,但是提升的数值下降. 这是由于反相关分布下,随着维度的增加,结果集能覆盖的体积越来越小. 因此,算法 ε-OGA 比 ε-GA 具有更高的准确度. RT 算法因为随机舍弃了一些元组,导致算法的结果集的支配体积最小.

综合比较图 17 和图 18,可以发现,本文算法 OGA 与 GA 算法相比,具有相同的准确度,但是运行效率大大提高. 而本文算法 ε-OGA 和 ε-GA 相比,ε-OGA 具有更高的准确度,但是需要略损失一些算法效率. 算法 RT 与本文算法相比,在维度不是太高,skyline 元组间存在较大的相交支配体积时,算法的时间效率和准确率都不占优势.

5.3.2 数据量对算法性能的影响

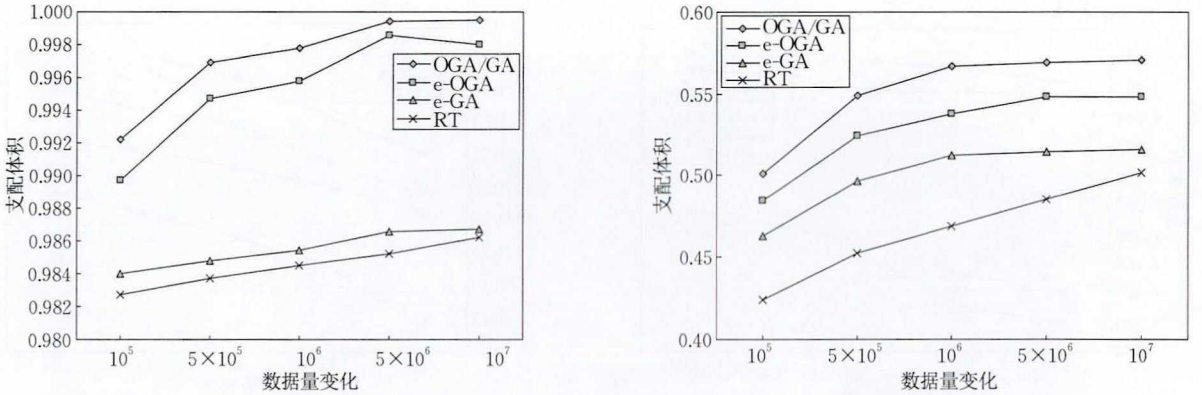
本小节主要比较了数据量变化对算法性能的影响,具体如图 19 和图 20 所示.

如图 19 所示,随着数据量的增加,算法的运行



(a) 独立数据集(ε=0.01)　　(b) 反相关数据集(ε=0.1)

图 19　数据量变化对算法运行时间的影响



(a) 独立数据集(ε=0.01)　　(b) 反相关数据集(ε=0.1)

图 20　数据量变化对支配体积的影响

时间线性增加，这是由于 skyline 结果数目随着数据量增加呈线性增加. 同样，反相关数据集中的算法运行时间远超过独立数据集中的算法运行时间. 对比算法 ε-OGA 和 ε-GA，可以发现 ε-OGA 的运行效率略差于 ε-GA. 在独立数据集中，ε-OGA 与 ε-GA 算法的差值比例约在 75%左右，相差值不超过 0.8 s. 反相关数据集中，差值比例约在 50%，相差值不超过 15 s，相差比例不会随着数据量增加而改变. 对比 OGA 和 GA 算法，OGA 算法的运行效率要远比 GA 算法好. 独立分布下，差值比例在 60%左右，相差值最大达到 1200 s，反相关分布下，相差比例在 86%左右，相差值最大达到 90 000 s. 相差比例不会随着维度发生明显变化. 算法 RT 的运行时间比 OGA 和 GA 算法要快，比 ε-OGA 和 ε-GA 慢很多.

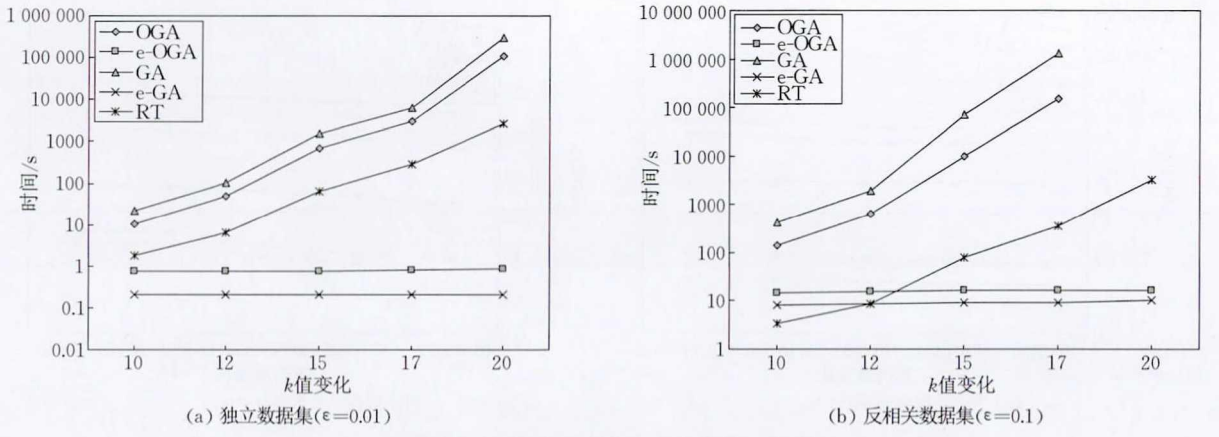
如图 20 所示，随着数据量的增加，算法的支配体积整体呈上升趋势. OGA 和 GA 算法具有相同的支配体积，即它们具有相同的准确度. 而相同数据集下，算法 ε-OGA 的支配体积要比 ε-GA 高，因此，算法 ε-OGA 比 ε-GA 具有更高的准确度. 独立分布下，准确度提升在 1%左右. 反相关分布下准确率提升在 5%左右，随数据量增加变化不大. 算法 RT 的精度最差，因为随机删除了一些元组. 在反相关数据集中，算法 RT 的精度表现要更差一些，因为反相关数据集中，算法 RT 舍弃的元组比例要更多.

综合比较图 19 和图 20，可以发现，本文算法 OGA 与 GA 算法相比，具有相同的准确度，但是运行效率大大提高. 而本文算法 ε-OGA 和 ε-GA 相比，ε-OGA 具有更高的准确度，但是需要略损失一些算法效率. ε-OGA 与 RT 算法相比，无论是准确度还是算法效率，都有更好的表现.

5.3.3　k 值变化对算法性能的影响

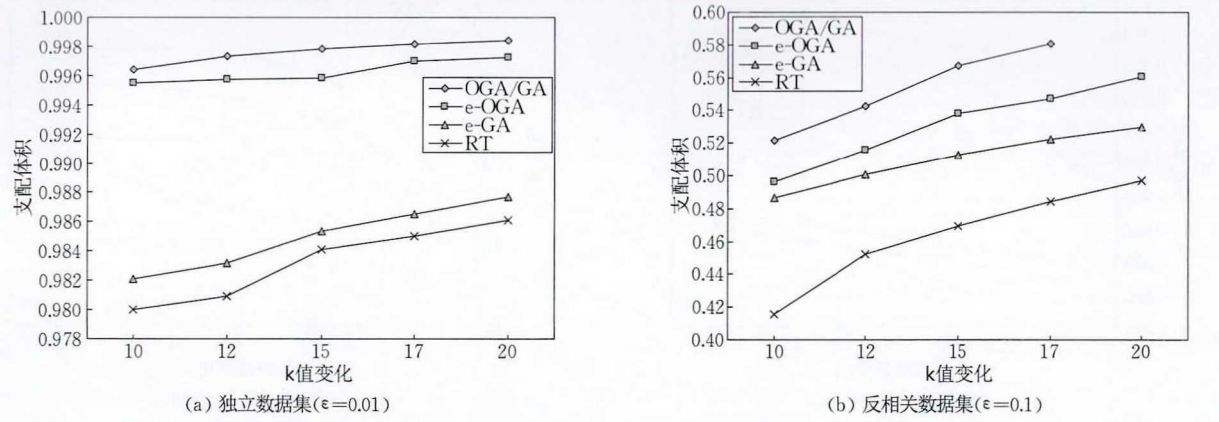
本小节主要比较了 k 值变化对算法性能的影响，具体如图 21 和图 22 所示.

如图 21 所示，随着 k 值的增加，算法 OGA、GA 与 RT 的运行时间呈指数增加，当 $k=20$ 时，反相关数据分布下，算法 OGA 与 GA 的运行时间过长，没有记录结果. 随着 k 值的增加，算法 ε-OGA 和 ε-GA 的运行时间变化不大. 独立分布下相差值约为0.05 s，反相关分布下相差值约为 6 s，基本不会随着 k 的变化而变化. 因为 ε-OGA 需要的额外计算与 k 值关系



(a) 独立数据集(ε=0.01) (b) 反相关数据集(ε=0.1)

图 21 k 值变化对算法运行时间的影响



(a) 独立数据集(ε=0.01) (b) 反相关数据集(ε=0.1)

图 22 k 值变化对算法运行时间的影响

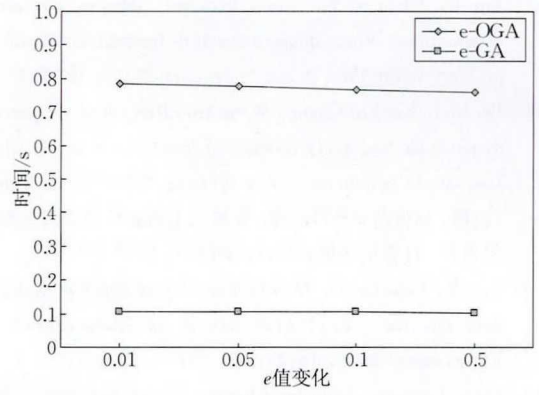
不大,受 d 的影响较大. 因此,算法 ε-OGA 和 ε-GA 对 k 值变化不敏感. 总体上,反相关数据集中的算法运行时间远超过独立数据集中的算法运行时间. 对比算法 ε-OGA 和 ε-GA,可以发现 ε-OGA 的运行效率略差于 ε-GA,但差距很小. 对比 OGA 和 GA 算法,OGA 算法的运行效率要远比 GA 算法好. 随着 k 的增加,独立数据集中相差比例由 50% 增加至 70%,最大差值 200 000 s. 反相关数据集中相差比例由 70% 上升到 90%,最大相差值达到 1 200 000 s,这是由于随着 k 的增加,优化策略 2 能过滤掉的元组数目相应增加,需要的计算量大大减少. 在相关数据集上,RT 算法的运行时间比 ε-OGA 和 ε-GA 慢很多,这是由于相关数据集中,ε-OGA 和 ε-GA 大概在结果集大小为 8 时,提前结束算法. 但是在反相关数据集,$k=10$ 时,RT 算法比 ε-OGA 和 ε-GA 快,这是由于在反相关数据中,ε-OGA 和 ε-GA 大概在结果集大小为 10 时,结束算法,而 RT 又随机删除了一些元组,因此结束计算较快.

如图 22 所示,随着 k 值的增加,算法的支配体积整体呈上升趋势. OGA 和 GA 算法具有相同的支配体积,即它们具有相同的准确度. 而相同数据集下,算法 ε-OGA 的支配体积要比 ε-GA 高,随机分布下,随着 k 值的增加,精度的提升越来越低,这是因为随机分布环境下,后加入元组的增量体积照比整体支配体积的影响过小. 从图 22(b)可以看出,随着 k 值的增加,ε-OGA 和 ε-GA 差距将变大,由 0.01 增加到了 0.04,这是由于 ε-GA 算法在 k 达到一定值后,不再计算,只是将结果简单加入. 而反相关数据集下,整体支配体积不是太大,后加入的元组对整体支配体积还有一定的影响. *RT* 算法的精确度最低,因为删除了一些元组.

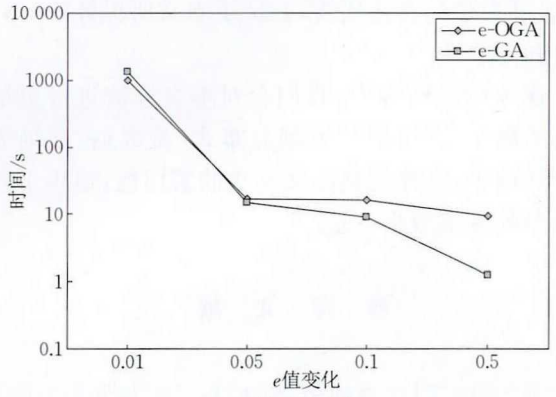
综合比较图 21 和图 22,可以发现,本文算法 OGA 与 GA 算法相比,具有相同的准确度,但是运行效率大大提高. 而本文算法 ε-OGA 和 ε-GA 相比,ε-OGA 具有更高的准确度,但是需要略损失一些算法效率.

5.3.4 ε 值变化对算法性能的影响

由于 ε 值只存在算法 ε-OGA 和 ε-GA 中,因此,本小节只比较了 ε 值变化对算法 ε-OGA 和 ε-GA 性能的影响,具体如图 23 和图 24 所示.

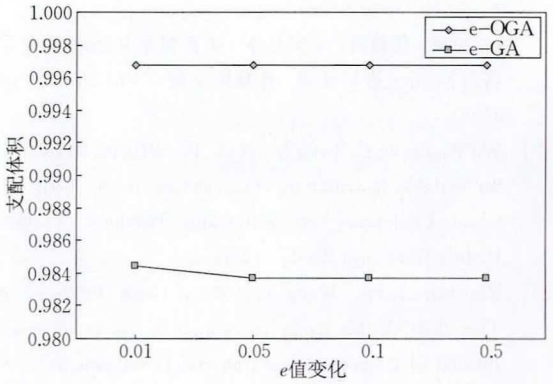


(a) 独立数据集

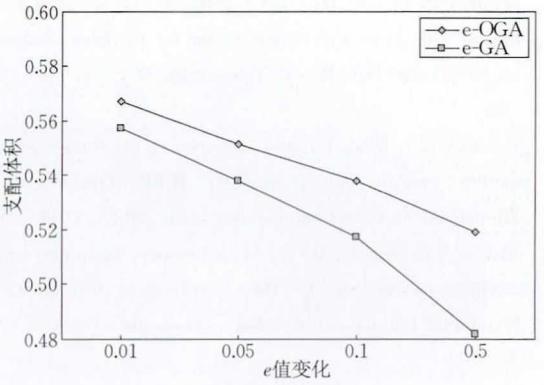


(b) 反相关数据集

图 23　ε 值变化对算法运行时间的影响



(a) 独立数据集



(b) 反相关数据集

图 24　ε 值变化对算法运行时间的影响

如图 23(a)和图 24(a)所示，独立数据集中，ε 值变化对算法的性能基本没有影响，因为独立分布中，贪心算法初始加入的 skyline 元组支配体积增加较快，后加入的元组支配体积贡献过少，很快能够满足算法 ε-OGA 和 ε-GA 的结束条件. 因此 ε 从 0.01 到 0.5 变化时，对算法运行效率和支配体积基本没有影响.

如图 23(b)和图 24(b)所示，反相关分布中，ε 值变化对算法性能影响较大，原因是反相关分布中，后加入的 skyline 元组也可能对整体支配体积提供较大的贡献，随着 ε 值减少，算法运行时间增加，整体上 ε-OGA 运行增加时间较 ε-GA 更慢一些，在 ε= 0.01 时，算法 ε-GA 的运行时间反超 ε-OGA. 随着 ε 的增加，算法的准确度都呈下降趋势.

综合比较图 23 和图 24，可以发现，在独立分布下，本文算法 ε-OGA 和 ε-GA 相比，ε-OGA 具有更高的准确度，但是需要略损失一些算法效率，处理时间都在 1 s 内. 而在反相关分布下，随着 ε 值的减少，算法 ε-OGA 在运行时间和支配体积上都比 ε-GA 表现更好.

通过上述大量的实验，验证了本文所提算法 OPA、OGA 和 ε-OGA 的有效性以及高效性. OPA 与现有算法 PBA 相比，拥有更好的效率和相同的准确率. OGA 与 GA 相比，拥有更好的效率和相同的准确率. ε-OGA 和 ε-GA 相比，拥有更好的准确率，只牺牲了很少的运行时间. ε-OGA 与 RT 算法相比，拥有更好的准确度和更好的时间效率.

6　总　结

本文对基于最大覆盖的代表轮廓 k-MCS 问题进行了深入的研究. 首先，针对 2 维数据集，提出基于前缀支配表的前缀优化算法 OPA，对比以往算法，该算法利用优化的支配面积计算公式和前缀支配表，可以通过少量的加减法运算，完成 2 维上的 k-MCS 求解. 同时利用 2 个优化策略，可以减少非必要的支配面积求解. 其次，针对多维数据集，提出了优化贪心算法 OGA，对比以往的基础贪心算法，OGA 利用改进的集合支配体积和元组增量支配体积的上下界估计策略，可以减少 50%以上的计算量. 在 OGA 的基础上，提出了提前截断算法 ε-OGA，在牺牲少量精度的前提下，大大提高运算效率. 最

后，通过大量的对比实验，验证了本文所提算法的有效性和高效性.

在今后的研究中，我们会对本文算法进行更加深入的研究，并将其扩展到分布式、流数据、高维等复杂环境中，从而提高本文算法的实用性，适用于各种复杂的现实数据环境.

参 考 文 献

[1] Borzsonyi S, Kossmann D, Stocker K. The skyline operator//Proceedings of the International Conference on Data Engineering. Washington, USA, 2001: 421-430

[2] Godfrey P, Shipley R, Gryz J. Maximal vector computation in large data sets//Proceedings of the International Conference on Very Large Data Bases. Trondheim, Norway, 2005: 229-240

[3] Bai M, Xin J, Wang G, et al. Discovering the k representative skyline over a sliding window. IEEE Transactions on Knowledge & Data Engineering, 2016, 28(8): 2041-2056

[4] Malene S-S, Sean C, Ira A. Maximum coverage representative skyline//Proceedings of the International Conference on Extending Database Technology. Bordeaux, France, 2016: 702-703

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

[6] Tao Y, Ding L, Lin X, et al. Distance-based representative skyline//Proceedings of the International Conference on Data Engineering. Shanghai, China, 2009: 892-903

[7] Bentley J-L, Kung H-T, Schkolnick M, et al. On the average number of maxima in a set of vectors and applications. Journal of the ACM, 1978, 25(4): 536-543

[8] Kalyvas C, Tzouramanis T. A survey of skyline query processing. 2017. https://xueshu.baidu.com/usercenter/paper/show?paperid=93cb7ef734ff8213eb9f5c28b0e9bf7a&site=xueshu_se

[9] Chomicki J, Godfrey P, Gryz J, et al. Skyline with presorting //Proceedings of the 19th International Conference on Data Engineering. Bangalore, India, 2003: 717-719

[10] Tan K-L, Eng P-K, Ooi B-C. Efficient progressive skyline computation//Proceedings of the 27th International Conference on Very Large Data Bases. Roma, Italy, 2001: 301-310

[11] Kossmann D, Ramsak F, Rost S. Shooting stars in the sky: An online algorithm for skyline queries//Proceedings of the 28th International Conference on Very Large Data Bases. Hong Kong, China, 2002: 275-286

[12] Papadias D, Tao Y, Fu G, et al. An optimal and progressive algorithm for skyline queries//Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data. San Diego, USA, 2003: 467-478

[13] Liu B, Chan C-Y. Zinc: Efficient indexing for skyline computation//Proceedings of the 36th International Conference on Very Large Data Bases. Singapore, 2010: 197-207

[14] Bai Mei, Xin Jun-Chang, Wang Guo-Ren, et al. Research on dynamic skyline query processing over data streams. Chinese Journal of Computers, 2016, 39(10): 2007-2030(in Chinese)
(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)

[15] Tao Y, Papadias D. Maintaining sliding window skylines on data streams. IEEE Transactions on Knowledge & Data Engineering, 2006, 18(3): 377-391

[16] Ding Lin-Lin, Xin Jun-Chang, Wang Guo-Ren, et al. Efficient Skyline query processing of massive data based on Map-Reduce. Chinese Journal of Computers, 2011, 34(10): 1785-1796(in Chinese)
(丁琳琳，信俊昌，王国仁等. 基于 Map-Reduce 的海量数据高效 Skyline 查询处理. 计算机学报，2011，34(10)：1785-1796)

[17] Wu P, Zhang C, Feng Y, et al. Parallelizing skyline queries for scalable distribution//Proceedings of the 10th International Conference on Extending Database Technology. Berlin, Germany, 2006: 112-130

[18] Xin Jun-Chang, Wang Guo-Ren, Gong Pi-Zhen, et al. Threshold skyline query processing in uncertain databases. Journal of Computer Research and Development, 2009, 46(z2): 126-132(in Chinese)
(信俊昌，王国仁，公丕臻等. 不确定数据库中的阈值轮廓查询处理. 计算机研究与发展，2009，46(z2)：126-132)

[19] Chan C-Y, Jagadish H, Tan K-L, et al. Finding k-dominant skylines in high dimensional space//Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data. New York, USA, 2006: 503-514

[20] Xia T, Zhang D, Tao Y. On skylining with flexible dominance relation//Proceedings of the International Conference on Data Engineering. Cancún, Mexico, 2008: 1397-1399

[21] Xin Jun-Chang, Bai Mei, Dong Han, et al. An efficient processing algorithm for ρ-dominate skyline query. Chinese Journal of Computers, 2011, 34(10): 1876-1884(in Chinese)
(信俊昌，白梅，东韩等. 一种 ρ-支配轮廓查询的高效处理算法. 计算机学报，2011，34(10)：1876-1884)

[22] Zhang Z, Lu H, Ooi B-C, et al. Understanding the meaning of a shifted sky: A general framework on extending skyline query. VLDB Journal, 2010, 19(2): 181-201

[23] Papadias D, Tao Y, Fu G, et al. Progressive skyline computation in database systems. ACM Transactions on Database Systems, 2005, 30(1): 41-82

[24] Kontaki M, Papadopoulos A-N, Manolopoulos Y. Continuous top-k dominating queries. IEEE Transactions on Knowledge & Data Engineering, 2012, 24(5): 840-853

[25] Nanongkai D, Sarma A-D, Lall A, et al. Regret-minimizing representative databases. Proceedings of the VLDB Endowment, 2010, 3(1): 1114-1124

[26] Chester S, Thomo A, Venkatesh S, et al. Computing k-regret minimizing sets. Proceedings of the VLDB Endowment, 2014, 7(5): 389-400

[27] Assent I. Taking the big picture: Representative skylines based on significance and diversity. VLDB Journal, 2014, 23(5): 795-815

[28] Sarma A-D, Lall A, Nanongkai D, et al. Representative skylines using threshold-based preference distributions// Proceedings of the International Conference on Extending Database Technology. Hannover, Germany, 2011: 387-398

[29] Evangelos D, Bernhard S. Efficient computation of reverse skyline queries//Proceedings of the 33rd International Conference on Very Large Data Bases. Vienna, Austria, 2007: 291-302

[30] Huang X, Zheng J. Deletion-robust k-coverage queries// Proceedings of the Database Systems for Advanced Applications. Chiang Mai, Thailand, 2019: 215-219

[31] Mirzasoleiman B, Badanidiyuru A, Karbasi A, et al. Lazier than lazy greedy//Proceedings of the Association for the Advancement of Artificial Intelligence. Austin, USA, 2015: 1812-1818

[32] Minoux M. Accelerated greedy algorithms for maximizing submodular set functions//Proceedings of the Optimization Techniques. Berlin, Germany, 1978: 234-243

BAI Mei, Ph.D., associate professor. Her research interests include data management, cloud computing, and query processing and optimization.

WANG Xi-Te, Ph. D., associate professor. His research interests include big-data management and parallel data processing.

LI Guan-Yu, Ph. D., professor. His research interests are intelligent information processing, semantic Internet of Things.

NING Bo, Ph. D., associate professor. His current research interests include data management and privacy preserving.

ZHOU Xin, Ph. D., lecturer. Her research interests include data management and machine learning.

Background

As an important operator for multi-criteria decision making, Skyline queries can find the data that users are really interested in from a large amount of data. The skyline is a good recommendation set for users if the size of skyline is small. However, with the increase of dimensionality and distribution of the dataset, the number of skyline tuples may be too large. It will be impractical for users to choose suitable skyline points after browsing all skyline points. Hence, the full skyline set becomes meaningless. To tackle this problem, the concept of k representative skyline has been proposed. Considering the representativeness and stability of representative skylines, we choose to study the concept of k-maximum coverage skyline (k-MCS for short) problem in this paper, which has high representativeness and good stability.

Compared with the previous k-MCS algorithms, the proposed algorithms in this paper have better efficiency. Firstly, we propose a prefix-based optimization algorithm (OPA for short) for the k-MCS problem in 2-dimensional datasets. Using the prefix-dominance-table, OPA can obtain the k-MCS result with a small number of addition and subtraction operations. Secondly, considering the k-MCS problem in d-dimensional ($d \geq 3$) datasets is NP-hard problem, two optimaization greegy algorithms OGA and ε-OGA are proposed. Compared with basic greedy algorithm, OGA reduces more than 50% calculations. By introducing the parameter ε, ε-OGA can greatly speed up the calculation effiency by sacrificing a small amount of precision. Finally, the effectiveness and efficiency of the proposed algorithms OPA, OGA and ε-OGA are verified by a large number of experiments.

Compared with previous algorithms, the algorithms in this paper have better computational efficiency by using the optimization formulas to compute the dominance size of a data set. Meanwhile, some unnecessary calculations can be reduced by using the corresponding filtering strategies. Hence, the proposed algorithms in this paper can solve the k-MCS problem more efficiently.

This research is partially supported by the National Natural Science Foundation of China under Grant Nos. 61702072, 61602076, 61976032, the China Post doctoral Science Foundation under Grant Nos. 2017M621122, 2017M611211, the Natural Science Foundation of Liaoning Province under Grant No. 20180540003, and the Fundamental Research Funds for the Central University under Grant No. 3132019202.