文章编号:1674-2974(2020)08-0009-12 DOI:10.16339/j.cnki.hdxbzkb.2020.08.002

PSP:一种高效的偏序域上 skyline 查询处理方法

白梅,王京徽†,王习特,朱斌,李冠宇
(大连海事大学 信息科学技术学院,辽宁 大连 116000)

摘 要:为解决偏序域上的 skyline 查询问题,本文提出一种高效的偏序域上的 skyline 查询处理方法,来满足人们对查询效率日益增长的需求. 首先,为提高偏序域上 skyline 的查询效率,将倒排索引引入 skyline 查询,提出一种基于倒排的索引结构. 其次,提出基础算法 (Basic Partially-ordered Skyline Processing based on inverted index,PSP_B),PSP_B 包含两个阶段:第一阶段,能够通过映射将偏序域转化成全序域,并建立倒排索引;第二阶段,通过倒排索引提前找到扫描结束点,得到最终的 skyline 结果. 再次,在 PSP_B 的基础上,进一步提出优化算法 (Improved Partially-ordered Skyline Processing based on inverted index,PSP_I). PSP_I 通过先分组再建索引的方法能够进一步提高计算效率. 最后, 用大量的实验证明本文所提算法的正确性和高效性.

关键词:skyline 查询;倒排索引;偏序域;查询优化;算法
中图分类号:TP391 **文献标志码**:A

PSP: An Efficient Skyline Computation Method for Partially Ordered Domains

BAI Mei, WANG Jinghui†, WANG Xite, ZHU Bin, LI Guanyu
(School of Information Science and Technology, Dalian Maritime University, Dalian 116000, China)

Abstract: In order to solve the skyline query problem in partial order domain, this paper proposes an efficient query processing method for the skyline query in the partial order domain to meet people's increasing demand for query efficiency. Firstly, in order to improve the query efficiency of skyline on partially ordered domains, this paper introduces the inverted index into the skyline query and proposes an index structure based on the inverted index. Secondly, the basic algorithm PSP_B (Basic Partially-ordered Skyline Processing based on inverted index, PSP_B) is proposed. The PSP_B consists of two phases: in the first step, each partially ordered domain is converted into two fully ordered domains through mapping function, and then each fully ordered dimension is managed through the inverted index; in the second step, the scan end point is found through scanning the inverted index and the result set is ob-

* **收稿日期**:2019-06-18
基金项目:国家自然科学基金青年基金资助项目(61702072,61602076,61976032),Youth Program of National Natural Science Foundation of China (61702072,61602076,61976032); 中国博士后科学基金面上资助项目 (2017M621122,2017M611211),China Postdoctoral Science Foundation(2017M621122,2017M611211);辽宁省自然科学基金资助项目(20180540003),Natural Science Foundation of Liaoning Province (20180540003); 中央高校基本科研业务费专项资金资助项目 (3132019202),Fundamental Research Funds for the Central Universities (3132019202)
作者简介:白梅(1986—),女,辽宁本溪人,大连海事大学副教授,博士
† 通讯联系人,E-mail:jinghui940920@163.com

tained. Thirdly, based on the PSP_B, this paper further proposes an optimization algorithm PSP_I (Improved Partially-ordered Skyline Processing based on inverted index, PSP_I). The PSP_I can further improve the computational efficiency by grouping the data in advance and then indexing them. Finally, a large number of experiments prove the correctness and efficiency of the proposed algorithm.

Key words: skyline query; inverted index; partial order domain; query optimization; algorithm

近年来，随着互联网技术的发展以及信息获取设备的进步，数据库收集处理的数据量增多，进一步，数据库中存储的数据量也急剧增加. 但是，人们却很难从这些海量、庞杂的信息中挖掘出自己最想要的有价值的信息. 因此，如何快速高效地从海量数据中返回给用户最为关心的数据，成为学术界的研究热点.

skyline 查询根据用户在各个维度的偏好，利用“支配”的概念，将数值间的大小比例作为“好坏”的评价标准，直观而准确地返回给用户最为关心的结果集. 具体地，给定两个点 p_1 和 p_2，p_1 支配 p_2 指的是在所有维度上 p_1 都不比 p_2 差，并且至少在一个维度上 p_1 要严格好于 p_2. 如图 1 所示，共有 20 条图书信息记录$\{p_1,p_2,\cdots,p_{20}\}$，每一条图书信息包括了两个维度信息：图书价格和评分，评分维度利用倒数映射到[0,1]区间内，使两个维度均以小值为“优”. 例如图 1 中 p_8 在每一维都比 p_{15} 小，说明 p_8 价格和评分都比 p_{15} 要好，即 p_8 支配 p_{15}. 经过 skyline 查询后，图中最终 skyline 集合为$\{p_8,p_{12},p_{16},p_{17}\}$，其他图书记录都可以被这个集合中的点支配.

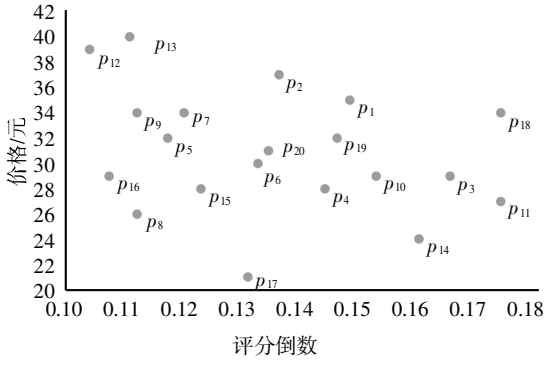


图 1 skyline 集合举例

Fig.1 The example of skyline

偏序域上的 skyline 第一次被考虑到是在 2005 年，Chan 等[1]讨论了具有偏序域属性的数据集上的 skyline 查询，他们将一个偏序域映射到两个全序域上，来适应自己提出的 skyline 算法，并针对这种映射提出了 SDC+和 BBS+算法. 但这种方法映射时成本太高，且不具有可扩展性. Sacharidis 等[2]在 2009 年提出了一种基于拓扑排序的 skyline 查询方法可以用来处理动态 skyline 查询. Liu 等[3]提出了 ZINC (Z-order Indexing with Nested Codes)算法，通过降维等方法来处理偏序维度，但当偏好关系复杂时，偏序维度降维成本将会很高. 文献 [4] 提出了一种 PSLP (Parallel Skylines with Logical Partitioning)框架，通过过滤不含 skyline 点的逻辑分区提高计算效率，但分区成本较高且过滤效率提高效果不明显.

综上所述，为了避免查询成本过高，本文将在映射的基础上引入倒排索引，提前对数据集进行过滤剪枝，通过减少对整个数据集的扫描来达到提高计算效率的目的.

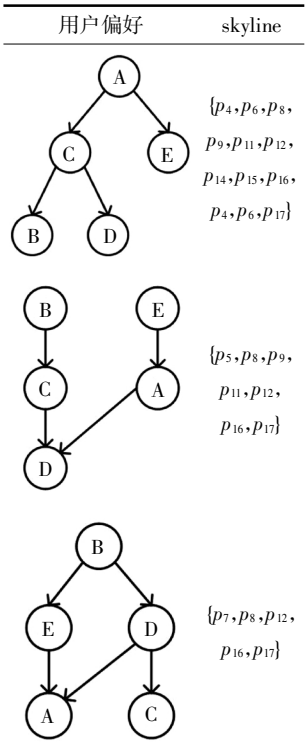
偏序域上的 skyline 查询处理，在现实生活中是一个非常有意义的议题. 例如，在为用户进行图书推荐时，用户往往希望得到评分高且价格低的图书推荐，并且不同用户对不同种类图书的偏好不同，如图 2 所示，需要将全序域与偏序域结合起来进行 skyline 查询，高效地针对不同用户进行个性化推荐. 因此，在偏序域上进行高效 skyline 查询具有极大的实际应用价值.

为了更有效地减少开销，提高轮廓查询效率，本文将倒排索引应用到查询中，首先提出了一种高效的偏序域上 skyline 查询处理方法 (Basic Partially-ordered Skyline Processing based on inverted index, PSP_B). 之后，在此算法的基础上，通过提前对数据点进行分组，提出了改进的优化算法 (Improved Partially-ordered Skyline Processing based on inverted index, PSP_I)，利用优化算法在极大程度上对冗余点

进行过滤. 实验证明,该算法能更高效地处理偏序域上的轮廓查询.

ID	全序域		
	价格/元	评分	图书种类
p_1	35	6.7	A
p_2	37	7.3	B
p_3	29	6.0	D
p_4	28	6.9	C
p_5	32	8.5	E
p_6	30	7.5	A
p_7	34	8.3	B
p_8	26	8.9	D
p_9	34	8.9	A
p_{10}	29	6.5	C
p_{11}	27	5.7	E
p_{12}	39	9.6	D
p_{13}	40	9.0	D
p_{14}	24	6.2	C
p_{15}	28	8.1	D
p_{16}	29	9.3	E
p_{17}	21	7.6	B
p_{18}	34	5.7	A
p_{19}	32	6.8	C
p_{20}	31	7.4	E

(a) 图书信息表



(b)偏序维度上用户偏好哈斯图

图 2　偏序域上的 skyline 查询举例

Fig.2　The example of skyline for partially ordered domains

总的来说,本文的主要贡献如下:

1)提出将倒排索引引入 skyline 查询领域,倒排索引将每个偏好维度上的属性按从优至劣进行排序,减少大量的冗余计算,从而提高计算效率.

2)提出了 PSP_B 算法,解决了传统算法中每次计算都对整个数据集进行扫描的问题. 算法对数据集在每个维度上建立倒排索引, 通过循环扫描策略快速找到扫描结束点来结束算法, 达到了对数据集过滤剪枝的目的,提高了计算效率.

3)在 PSP_B 算法的基础上,提出了 PSP_I 算法,在将偏序域映射到全序域之后,建立倒排索引之前,对整个数据集按文中提出的分组策略进行分组,在组内建立倒排索引. 并提出整组过滤策略, 对不含 skyline 结果点的分组进行整组过滤, 在基础算法的基础上进一步提高了剪枝效率.

4)设计了详细的性能比较实验,通过实验证明了本文提出的 PSP_B 和 PSP_I 算法可以有效地处理偏序域上的 skyline 查询问题, 并且 PSP_I 算法在查询效率上要更优于 PSP_B 算法.

1　相关工作

1.1　传统 skyline 查询

早期,人们主要关注于全序域上的 skyline 查询. Borzsonyi 等[5]第一次将 skyline 查询的概念引入数据库领域,并提出了 BNL(Block Nested Loops)和 D&C(Divide and Conquer) 算法,BNL 算法对待测元组建立临时表, 通过将每个待测元组与表内元组比较来进行轮廓查询,因此 BNL 算法性能受主内存大小的限制;D&C 算法是将数据集划分为多个分区,在每个分区内进行查询,计算出局部的 skyline 点,再将得到的结果进行合并, 然后对合并的结果再进行查询来得到最终结果. 这两种算法都会产生多次迭代,查询效率较低.

之后,Chomicki 等[6]提出了 SFS 算法,该算法是在 BNL 的基础上对数据按单调函数进行预排序,然后再进行 skyline 查询,减少了开销. Tan 等[7]提出的 Bitmap 算法先将每个元组映射成一个 m 位矢量,然后通过矢量间的计算得到最后的 skyline 集合. 之后,NN(Neural Network)算法[8]利用 R 树索引数据集,并利用 K 近邻算法来进行 skyline 查询,通过不断循环删除 skyline 点支配的区域来找到结果,因此在对高维数据进行查询时会有大量的重复查询, 导致效率低.

2005 年 Chan 等提出 BBS 算法[1],基于最近邻搜索策略,用 R-Tree 对待测数据集建立索引,将所有待测元组划在一个个最小边界矩形上. 通过对最小边界矩形左下角元组的比较来过滤冗余元组, 又进一步节省了开销.

此外, 还有许多针对特定数据环境下的 skyline 查询算法,如 P2P 网络[9]、分布式环境[10]和不确定数据环境[11]等.

1.2　偏序域上的 skyline 查询

在实际应用中, 有许多维度是无法直接通过数值间的大小比例来判断好坏的, 因此, 偏序域上的 skyline 查询在现实生活中是一个非常有意义的议题. Chan 等在 2005 年第一次讨论了具有偏序域属性的数据集上的轮廓查询[1],由于引入偏序域后,传统的轮廓查询算法将不能再有效地修剪空间,因此,他们提出将每个偏序域映射到两个全序域上来解决这一问题,并且提出了 BBS+、SDC 和SDC+三种算法. BBS+是进行投影空间转换后直接适应 BBS;SDC 和

SDC+都利用支配关系将数据进行分层,虽然减少了不必要的比较,但分层时的开销却很大.

2009 年,Sacharidis 等[2]提出了一种基于拓扑排序的 skyline 查询方法 TSS (Topologically Sorted Skylines for Partially Ordered Domains),利用拓扑排序将偏序域映射成全序域上的间隔,并且,TSS 算法可以用来处理动态 skyline 查询. 2010 年,Liu 等[3]提出了 ZINC 算法,通过降维来简化用户偏好情况,并使用嵌套码将其转化为部分阶映射到总阶数. 最后用 ZB 树来索引编码值,然而,当用户偏好复杂时,偏序降维成本将会很高.

Hsueh 等在 2017 年提出 e-DFS (extended Depth-First Search)算法[12],通过缓存之前的查询结果,将这些结果作为索引,对相似查询直接在缓存区中进行查询,该算法在相似性比较上开销大,且针对性不强, 在缓存区没有相似查询时仍要访问整个数据集.

综上所述,虽然在全序域上的 skyline 查询已经取得了丰硕的成果, 但在处理偏序域上的数据时仍缺少一种有效率的方法. 因此,本文将针对偏序域上的数据集,将偏序与全序结合起来,引入倒排索引的概念,在进行 skyline 计算之前对数据集进行过滤剪枝,将明显提高计算效率,有较高的实用价值.

2 问题描述

本文关注的是偏序域上的 skyline 查询问题,以小值为优举例, 遇到大值为优的维度需要经过预处理变成倒数后再进行计算. 为了描述方便,表 1 给出了本文的符号定义.

传统的支配关系和 skyline 的相关概念分别如定义 1 和定义 2 所示.

定义 1 (支配[5])给定 d 维数据集 P,其中两个元组 p_1 和 p_2,p_1 支配 p_2(记作 $p_1 \prec p_2$)满足下列条件:(i,j 指不同的属性维度)

1) $\forall i \in \{1,2,\cdots,d\}, p_1[i] < p_2[i]$;

2) $\exists j \in \{1,2,\cdots,d\}, p_1[j] < p_2[j]$.

定义 2 (skyline[5])给定数据集 P,其 skyline 是所有不被其他点支配的点的集合,即 $\mathrm{SKY}(P) = \{p_j \mid \nexists p_i \in P, p_i \prec p_j\}$.

偏序域由多个二元关系组成, 表明对于集合中的某些元素对,其中一个元素要优于另一个元素. 偏序一词用于表示不是每对元素都需要具有可比性. 引入偏序域后, 原有的支配定义将不再适应现有的数据集,因此给出新的支配关系.

表 1 符号定义

Tab.1 List of symbols

符号	符号含义
p_i, p_j	数据集中的数据点
SKY	轮廓点集合
P	数据集
d	数据维度
$\lvert TO\rvert$	全序维度个数
$\lvert PO\rvert$	偏序维度个数
$TO(d_1, d_2, \cdots, d_t)$	全序维度
$PO(d_1', d_2', \cdots, d_p')$	偏序维度
$\lvert D_{\text{totall}}\rvert$	映射后维度总个数
R^i	第 i 维度上的结果集
PO_i-TO_1, PO_i-TO_2	由偏序维度 PO_i 映射出的两个全序维度
p_i.count	元组 p_i 被扫描过的次数
T^i	第 i 维度上的临时表

定义 3 (偏序-支配[1])给定 d 维数据集 P,对两个元组 p_1 和 p_2,p_1 支配 p_2(记作 $p_1 \prec p_2$)满足下列条件:(其中 $i \in TO, j \in PO, m \in d$)

1) $\forall i \in TO, p_1[i]$不差于 $p_1[i]$;

2) $\forall j \in PO, p_1[j]$不差于 $p_2[j]$;

3) $\exists m \in \{TO \cup PO\}, p_1[m]$优于 $p_2[m]$.

利用图 2(a)的数据集举例说明,图 1 是对该数据集仅考虑价格和评分两个全序域, 直接在全序域上求 skyline 的结果,最终计算得到的 skyline 集合为 $\{p_8, p_{12}, p_{16}, p_{17}\}$,但在图 2(b)中,加入了偏序域上不同用户对不同种类图书的偏好, 对 3 个不同用户分别求得的 skyline 集合分别为$\{p_4, p_6, p_8, p_9, p_{11}, p_{12}, p_{14}, p_{15}, p_{16}, p_{17}\}$、$\{p_5, p_8, p_9, p_{11}, p_{12}, p_{16}, p_{17}\}$、$\{p_8, p_{12}, p_{16}, p_{17}\}$.

3 基于倒排索引的高效处理方法

本章首先介绍了文中的数据索引 (倒排索引);然后,介绍了利用倒排索引的过滤方法;最后,介绍了基于倒排索引的基础算法 PSP_B. 算法利用倒排索引对数据提前进行处理,对数据集进行过滤剪枝,使得进行 skyline 查询时不必扫描整个数据集.

3.1 倒排索引

对于具有偏序域的数据集,在建立倒排索引前,

为了方便计算,需要将偏序维度映射到全序维度.

映射规则[2]:算法采用将偏序域属性映射到两个全序域的方式来处理偏序域属性. 如图 3 所示,对于偏好哈斯图[13]进行深度优先遍历,并用间隔[po–to_1, po–to_2]进行标记,其中 po–to_1 表示节点在深度优先遍历时第一次被扫描到的时刻,po–to_2 表示节点结束扫描的时刻,即该点的子节点全部遍历结束. 偏序域上的偏好关系就可以用间隔间的覆盖来表示.

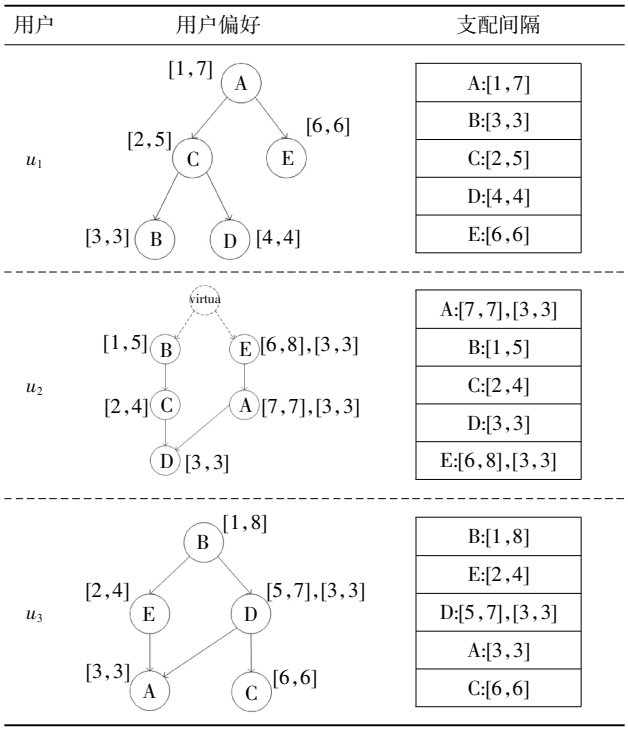


图 3　映射规则举例

Fig.3　Examples of the mapping function

如用户 u_2 所示的偏好图，在进行映射前事先为偏好图加一个虚节点,方便对偏好属性进行映射;并且对于映射出不止一个间隔的情况，按所有间隔的并集处理,如用户 u_3 所示,由于属性 D 优于属性 A,因此将属性 A 映射出的间隔并到属性D的间隔中. 假设属性 M、N 的间隔分别为 [po_1–to_1,po_1–to_2],[po_2–to_1,po_2–to_2],若属性 M 的间隔[po_1–to_1,po_1–to_2],被包含在另一个属性 N 的间隔[po_2–to_1,po_2–to_2]内,即 po_2–to_1 < po_1–to_1 并且 po_2–to_2 > po_1–to_2 那么就说明在该偏序维度上属性 M 优于属性 N. 若两个区间互不包含,则相应属性的好坏无法比较. 例如图 3 中,对于用户 u_1 来说,属性 C 映射到全序域的间隔为[2,5],属性 B 映射到全序域的间隔为[3,3],属性 C 的间隔可以覆盖属性 B 的间隔,即对于用户 u_1 来说,属性 C 要优于属性 B.

扫描结束点:为方便描述,本文引入扫描结束点的概念，我们称第一次扫描到满足结束条件 1 的元组就是扫描结束点(参考结束条件 1),扫描到扫描结束点时可以结束查询,将结果集返回给用户.

为了减少计算数据点的数量，本文采用了倒排索引结构来对数据进行管理. 倒排索引可以加快元组间支配关系的判定,快速找到扫描结束点,从而减少计算数据量. 具体地,对于所有的偏序维度按映射规则映射到全序域上,然后,针对每一维的数据,都按照从优到劣的顺序(本文例子中除 PO–TO_2 维度以大值为优外,其他维度均以小值为优)建立倒排索引如图 4 所示.

价格	评分	PO–TO_1	PO–TO_2
21:p_{17}	9.6:p_{12}	1:p_2,p_7,p_{17}	8:p_2,p_7,p_{17}
24:p_{14}	9.3:p_{16}	2:p_5,p_{11},p_{16},p_{20}	7:p_3,p_8,p_{12},p_{13},p_{15}
26:p_8	9.0:p_{13}		
27:p_{11}	8.9:p_8,p_9	3:p_1,p_3,p_6,p_8,p_9, p_{12},p_{13},p_{15},p_{18}	6:p_4,p_{10},p_{14},p_{19}
28:p_4,p_{15}	8.5:p_5		
29:p_3,p_{10},p_{16}	8.3:p_7	5:p_3,p_8,p_{12},p_{13},p_{15}	4:p_5,p_{11},p_{16},p_{20}
30:p_6	8.1:p_{15}		
31:p_{20}	7.6:p_{17}	6:p_4,p_{10},p_{14},p_{19}	3:p_1,p_3,p_6,p_8,p_9, p_{12},p_{13},p_{15},p_{18}
32:p_5,p_{19}	7.5:p_6		
34:p_7,p_9,p_{18}	7.4:p_{20}		
35:p_1	7.3:p_2		
37:p_2	6.9:p_4		
39:p_{12}	6.8:p_{19}		
40:p_{13}	6.7:p_1		
	6.5:p_{10}		
	6.2:p_{14}		
	6.0:p_3		
	5.7:p_{11},p_{18}		

图 4　倒排索引应用举例(用户 u_3)

Fig.4　The example of the inverted index application

基础循环扫描策略：由于全序域和偏序域上数据属性值的数量差距较大，导致建立的倒排表不同维度分布不同,属性值个数有差距,如图 4 所示,因此对倒排索引上每个维度 i 维护一个 times_i 变量,用来记录该维度上扫描过的点的个数，每次扫描均选择 times_i 值最小的维度,并在该维度上选择索引位置最靠前的元组进行计算,当 times_i 最小值对应多个维度时可以对这些维度按顺序依次扫描.

图 4 中,以第一次循环扫描顺序为评分、价格,PO–TO_1、PO–TO_2 为例,结束第一轮扫描后,4 个维度的 times_i 值分别为 1、1、3、3，进行第二轮扫描时,根

据基础循环扫描策略,应从 $times_i$ 值最小的维度价格和评分维度按序依次扫描,即选择价格维度,那么结束这次扫描后 4 个维度的 $times_i$ 值变为 2、1、3、3,此时应选取评分维度进行扫描. 这样一直循环扫描下去,直到算法结束(参见结束条件 1).

在 skyline 查询时,基础循环扫描策略对倒排索引进行扫描,并对扫描过的点进行扫描次数标记. 对每一个元组仅在第一次扫描到的时候进行 skyline 计算,之后扫描到时仅增加该元组的扫描次数.

3.2 冗余点过滤

在本小节中主要描述了算法过滤冗余点的过程,为描述方便首先给出结束条件 1 的内容,用来描述算法的结束条件,之后再通过引理 1 证明其正确性.

结束条件 1: 当扫描到一个元组 p_i 在所有维度上都被扫描过一次时, 若 p_i 在偏序维度 PO_m 对应多个间隔, 判断 p_i 在 PO_m 上扫描到的值是否来自同一间隔,若是,则 p_i 可以作为扫描结束点,此时可以结束算法,将结果集返回给用户;否则 p_i 不能作为扫描结束点,继续循环扫描.

引理 1[1]:若出现一个元组 p_i 至少在每一个维度上都被扫描过一次,即 p_i.count>=$|D_{total}|$,根据基础循环扫描策略, 此时在任一维度上都没有被扫描过的元组 p_j(p_j.count=0)一定不会是 skyline 点.

证明:根据引理 1,当有元组在每一个维度都扫描到一次时可以结束算法, 但由于 PO_m–TO_1、PO_m–TO_2 是由同一个偏序维度 m 映射出的且存在一个元组对应多个间隔的情况, 必须是同一个间隔扫描结束才能保证是该偏序维度扫描过一次.

证毕

图 4 中, 第一次 count 值达到 4 的元组为 p_8,此时判断,p_8 在 PO–TO_1、PO–TO_2 上扫描到的值分别为 7、3,由于 p_8 偏序域映射出的间隔为[3,3],[5,7],3 和 7 并不来自同一间隔,因此不停止算法,继续进行循环扫描. 直到扫描到元组 p_{17}.count 值为 4 时, 且在 PO–TO_1、PO–TO_2 上扫描到的值分别为 1、8, 来自同一间隔,根据引理 1 和结束条件 1,此时可以结束算法(如图 4 中阴影所示),阴影下方还没有被扫描过的点至少都可以被 p_{17} 支配, 一定不能成为 skyline 点,可以直接过滤.

为描述方便,定义 R^i 为暂时结果集,用来存放 d_i 维度上已扫描过的 skyline 点.

定理 1:对维度 D_i 建立对应的结果集R^i,那么对于扫描到在维度 D_i 上的元组 p_i,若 p_i 不被 R^i 中的元组支配,那么 p_i 一定是 skyline 点.

证明:根据支配定义,一个元组若可以支配元组 p_i,则在所有维度都不能比 p_i 差,因此,元组 p_i 当前所在维度 D_m 表现不如元组 p_i 的都不可能支配 p_i,因此仅考虑 D_m 对应的结果集 R^m 中的元组即可.

证毕

在图 4 中, 当扫描到价格维度上的第二个元组 p_{14} 时, 只需要与 R^1 中的 p_{17} 进行支配关系比较即可,如图 5 所示,不需要与 p_{12},p_7 比较,节省了比较成本.

由引理 1 和定理 1 可以过滤掉大量的冗余点,只针对最必要的元组进行 skyline 查询,极大地提高了查询效率.

R^1	R^2	R^3	R^4
21:p_{17}	9.6:p_{12}	1:p_7,p_{17}	8:p_7,p_{17}
	9.3:p_{16}	3:p_8	7:p_8
	8.3:p_7		

图 5 维度 R^i 上的结果集举例(用户 u_3)

Fig.5 The example of result set on domain R^i

算法 1:基础算法(PSP_B)

输入:D 维空间上的数据集 P,其中:

$|TO|$个全序维度 $TO(d_1,d_2\cdots)$

$|PO|$个偏序维度 $PO(d_1',d_2'\cdots)$;

输出:数据集 P 的 skyline 集合 R;

```
1. 初始化每个维度上的结果集 R^i=Φ;
2. for each d_i' ∈ PO do
3.    根据映射规则将 d_i'映射到全序域 P_i – TO_1 和 P_i – TO_2;
4. end for;
5. |D_total| = |TO|+2×|PO|; (映射后的维度总个数)
6. 对所有维度 d_i ∈ (P–TO_1∪P–TO_2∪TO)建立倒排索引;
7. while (true)
8.    根据基础循环扫描策略得到计算点 p_i,属于维度 j
9.    if(p_i.count=0)
10.         if(comparesky(p_i,R^j)=true)(见算法 2)
11.           将计算点 p_i 加入当前维度的结果集 R^j;
12.           p_i.count++;
13.         end if
14.   else
15.         p_i .count++;
16.         if(p_i.count>= |D_total|
17.           if(p_i 在 PO_m 上扫描到的值来自同一间隔)
18.              break;
19.           end if;
20.         end if;
21.   end if;
```

22. end while;
23. 对所有 R^j 取并集得到最终结果集 R;
24. return R;

算法 1 第 2~6 行根据基础循环扫描策略将偏序域映射到全序域，并建立倒排索引来维护数据. 第 8~22 行是算法主体，根据维护的 count 值来进行 skyline 计算. 其中,元组仅在第一次被扫描到的时候(即 count=0 时)进行 skyline 计算(如算法 2 所示),当 count>0 时,只记录扫描次数,不再进行 skyline 计算;当 count 值与$|D_{total}|$相等时,根据引理 1 与定理 1 判断扫描到的偏序属性是否来自同一间隔,若是,则跳出循环,执行第 23 行,返回结果集,结束算法,否则继续循环扫描,直到算法结束.

算法 2:Skyline 计算方法 comparesky(p_i,R^j)
输入:元组 p_i,p_i 所在维度对应的结果集 R^j
输出:待测 p_i 是否为最终的 skyline
1. for each $p_m \in R^j$
2.　if(p_m 偏序-支配 p_i)
3.　　return false;
4.　　end if
5. end for
6. return true;

如图 4 所示,以第一轮维度扫描顺序为价格、评分 PO-TO_1、PO-TO_2 为例,结束一轮扫描后,4 个维度的 $times_i$ 值分别为 1、1、3、3，且其中扫描过的元组{p_{17},p_{12},p_2,p_7}的 count 值为{3,1,2,2},本轮中进行了 skyline 计算的元组为{p_{17},p_{12},p_2,p_7},计入结果集的元组为{p_{17},p_{12},p_2,p_7}. 然后根据基础循环扫描策略继续进行扫描,处理价格、评分维度,扫描元组{p_{14},p_{16}},根据结束条件 1,p_{14} 与 R^1 中的 p_7、p_{17} 进行 skyline 计算,p_{16} 与相 R^2 中的 p_{12} 进行 skyline 计算. 依次循环计算,当扫描到元组 p_8 的 count 值等于 4 时(即与$|D_{total}|$相等),根据定理 1 判断扫描到 PO-TO_1、PO-TO_2 维度上的值不是来自同一个间隔,因此继续循环扫描,直到扫描到元组 p_{17} 的 count 值为 4,与$|D_{total}|$相等,此时算法结束,最终得到结果集{p_7,p_8,p_{12},p_{16},p_{17}}.

4　优化算法

在将倒排索引引入算法的基础上，本文又采用提前分组的方式对算法进行了进一步优化.

4.1　分组倒排

分组策略:给定 d 维空间上的数据集 P,根据数据集 P 的偏序维度（若有多个偏序维度则选择属性值最少的一个偏序维度)进行分组,将在该维度上拥有相同属性值的元组分到一组，例如选取图书种类作为分组维度,将具有相同种类的图书分到一组,如图 6 所示.

A	
价格	评分
30:p_6	8.9:p_9
34:p_6,p_{18}	7.5:p_6
35:p_1	6.7:p_1
	5.7:p_{18}

B	
价格	评分
21:p_{17}	8.3:p_7
34:p_7	7.6:p_{17}
37:p_2	7.3:p_2

C	
价格	评分
24:p_{14}	6.9:p_4
28:p_4	6.8:p_{19}
29:p_{10}	6.5:p_{10}
32:p_{19}	6.2:p_{14}

D	
价格	评分
26:p_8	9.6:p_{12}
28:p_{15}	9.0:p_{13}
29:p_3	8.9:p_8
39:p_{12}	8.1:p_{15}
40:p_{13}	6.0:p_3

E	
价格	评分
27:p_{11}	9.3:p_{16}
29:p_{16}	8.5:p_5
31:p_{20}	7.4:p_{20}
32:p_5	5.7:p_{11}

图 6　分组倒排举例

Fig.6　The example of grouping inversion

对数据集 P 按选定偏序维度进行分组，并将除该维度外的其他偏序维度按映射规则映射到全序维度,在组内建立倒排索引. 之后根据除分组维度外的所有维度分别建立临时表 T^i，用于存放每个分组中取值最小的一个或多个元组(本文以小值为优).

临时表的更新:进行计算时,在每个维度 i 上从对应的临时表 T^i 中取出最优值,与所在维度结果集 R^i 中的元组进行比较，若不被结果集中的点支配则加入结果集,将其从 T^i 中删除,从对应维度 i 的分组中再选择最优元组加入 T^i. 同时与基础算法一样,将记录该数据点的扫描次数值加 1. 当元组的 count 值与$|D_{total}|$-1(除分组维度外所有维度总数)相等时,根据结束条件 1,判定该组计算是否结束.

优化循环扫描策略:1)临时表选择:同基础循环扫描策略类似，对每个临时表 i 维护一个变量 $times_i'$，用来记录在临时表 T^i 中扫描过的元组的个数，每次扫描时选择 $times_i'$值最小的临时表进行扫描. 2)元组选择:每次扫描选定临时表后,选择该表内具有最优值的元组 p_i 进行计算，并根据临时表的更新策略更新临时表.

如图 6 所示,初始化时分别从每一个分组中,根据每一个维度的倒排索引取出第一个元组加入到对应维度的临时表中(图 7). 此时 T^1,T^2 的 times′均为 0,因此按顺序选取 T^1,那么第一次计算的就是 T^1 中值最优的元组 p_{17},计算完成后,此时 T^1 的 times′为 1,T^2 的 times′为 0,因此第二次对 T^2 进行扫描,选取 T^2 中值最优的元组 p_{12} 进行计算. 这样依次按优化循环扫描策略计算,直到算法结束(参见结束条件 2).

T^1(价格)				
30:p_6	21:p_{17}	24:p_{14}	26:p_8	27:p_{11}

T^2(价格)				
8.9:p_9	8.3:p_7	6.9:p_4	9.6:p_{12}	9.3:p_{16}

图 7 临时表举例

Fig.7 The example of temporary tables

4.2 整组过滤

定理 2(整组过滤方法):若扫描到的元组 p_i 在除分组维度外其他维度上均满足结束条件 1,那么可以结束元组 p_i 所在分组的计算,并且可以根据用户偏好,将在分组维度上表现比元组 p_i 差的所有分组的计算结束.

证明: 当扫描到元组 p_i 在除分组维度外其他维度上均满足结束条件 1 时,根据引理 1 此时没有扫描过的元组在除分组维度外所有维度上都不会优于 p_i,根据支配的定义,同一分组内的元组和分组维度上表现比 p_i 属性差的分组内的元组,在分组维度表现也比 p_i 差,因此不存在 skyline 元组,可以直接过滤,结束整个分组的计算.

证毕

例如在本文中,当 p_8 被扫描到两次时,通过图 2 可知,对应的分组维度图书分类上属性值为 B,对于用户 u_3 的偏好来说,根据整组过滤方法,E、D、A、C 分组都可以整组过滤,因此算法此时可以结束,没有扫描过的元组一定不会是 skyline 点.

结束条件 2:所有分组的计算都结束时算法结束.

当所有分组的计算都结束时,算法结束,此时所有维度上结果集 R^i 的并集就是最终所求的结果. 由于实际情况中偏序属性远远少于全序属性,在删除整个偏序分组时将直接过滤掉大量数据点,极大加快了计算速度. 并且在与结果集中的 skyline 点进行比较时,由于是按每个维度从最优开始的顺序,可以只与来自相同的维度的结果集中的点进行比较.

算法 3:优化算法(PSP_I)

输入:根据分组策略的到的分为 n 组后的数据集$\{P_1,P_2,\cdots,P_n\}$

输出:P 的 skyline 集合 R

```
1. 初始化每个维度上的结果集 R^i = Φ
   每个维度上的临时表 T^i;
2. while(true)
3.   根据优化循环扫描策略得到的计算点 p_i,属于维度 j
4.   if(p_i.count=0)
5.       if(comparesky(p_i,R^j)=true)
6.          将计算点 p_i 加入当前维度的结果集 R^j;
7.          p_i.count++;
8.       end if
9. end if;
10. else
11.      p_i.count++;
12.      if(p_i.count>=|D_total|-2)
13.          结束对应分组的计算;
14.          if(所有分组均结束计算)
15.             return R = {R_1∪R_2∪⋯∪R^(r-1)};
16.             end if;
17.      end if;
18. end if;
19. end while;
```

5 实验分析

本节展示了所提算法的高效性. 实验环境配置为 Inter Core i5 7300HQ 2.50 GHz CPU,8 GB 内存,1 T 硬盘,Windows 10 操作系统的 PC. 算法用 C++语言编写. 为了评估本文所提算法性能,以 ZINC 算法[3]和 PSLP 算法[4]为对比实验进行.

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集(共包含 2×10^6 条股票记录,每条股票记录包含 4 个属性:交易量、股票价格涨幅、股票类别和上市板块,其中股票类别和上市板块为偏序属性维度). 真实数据集计算结果如表 2 所示.

表 2 真实数据集实验结果

Tab.2 Result of real data set

算法	时间/s	过滤元组数量/10^6
ZINC	16.628	0.32
PSLP	11.231	0.43
PSP_B	6.243	0.83
PSP_I	4.399	1.02

通过表 2 可以看出本文提出的 PSP 算法在真实数据集上的表现,在进行 skyline 计算的速度上较之

前的算法有明显提高，并且由于高效地剪枝过滤，需要计算的数据量也明显少于 ZINC 算法和 PSLP 算法.

合成数据由文献[14]给出的数据生成器生成,包括独立分布数据集和反相关分布数据集. 默认数据集包括 5 个偏好维度,3 个全序和 2 个偏序；默认数据量规模为 10^6；默认用户偏好 DAG 的宽度为 4,深度为 8,密度为 0.6. 整体上数据服从随机分布,本章实验主要通过维度、数据集规模、哈斯图宽度和深度以及哈斯图密度的变化对实验结果的影响来验证算法的优越性,衡量实验的主要标准为 skyline 的计算时间，实验中合成数据的主要参数及其变化范围如表 3 所示.

表 3　实验参数

Tab.3　Experimental parameters

参数	默认值	变化范围
维度(全序,偏序)	(2,3)	(2,2),(2,3),(2,4),(4,2),(4,3),(4,4)
数据量规模	10^6	10^5,5×10^5,10^6,5×10^6,10^7
DAG 的宽度,深度(W,D)	(4,8)	(3,4),(4,4),(4,8),(5,8),(5,16)
DAG 的密度	0.6	0.2,0.4,0.6,0.8,1

5.1　维度变化对算法性能的影响

为了分析数据集维度对实验的影响，在固定全序维度个数的情况下模拟了偏序维度个数变化的情况,组合(t,p)表示全序维度和偏序维度个数,其中 t 表示全序维度个数,p 表示偏序维度个数. 为了更加稳定地测出算法性能，对于每种取值组合都利用数据生成器随机生成了 100 次数据集,进行 100 次实验，然后取这 100 次实验的平均值进行记录. 图 8 记录了在不同维度个数下进行 skyline 计算所消耗的时间,图 9 记录了在不同维度下计算过的元组的数量.

图 8 和图 9 表明了当偏序域维度个数增多时,4 种算法的计算时间和计算元组数量的变化. 从图中可以看出当|TO|相等,|PO|增多时 4 种算法所需时间均呈现指数上升趋势，计算过的元组数量也呈现上升趋势. 可以看出,4 种算法中 PSP_B 和 PSP_I 算法要明显优于另两种算法,这是因为当偏序域增多时,用户偏好复杂,降维成本会很高,PSP_B 算法在映射后采用倒排索引进行轮巡扫描,可以过滤掉大量冗

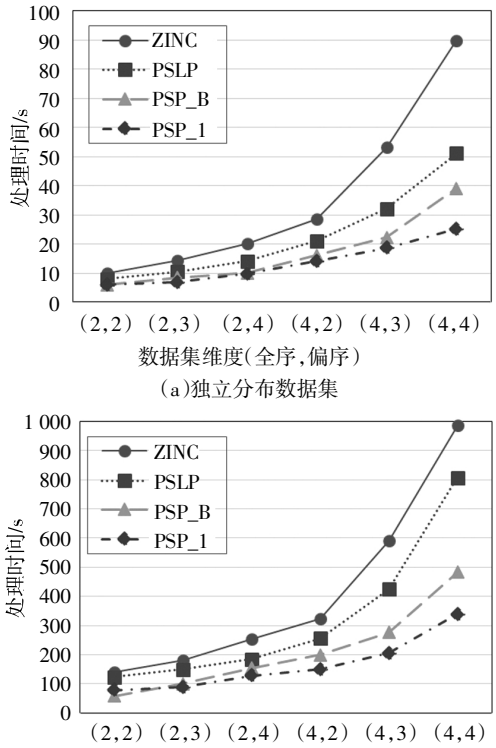


图 8　数据集维度对计算时间的影响

Fig.8　The effect of dataset dimensions on calculation time

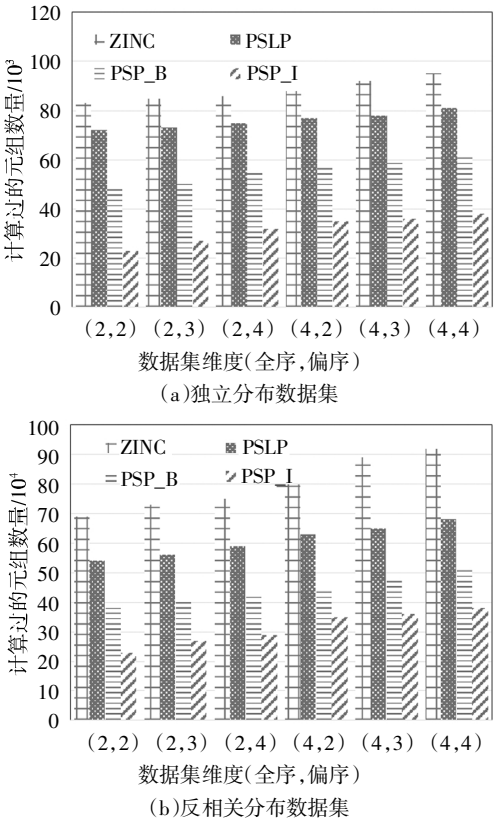


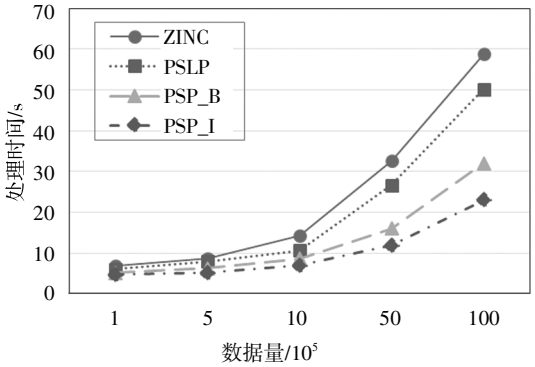
图 9　数据集维度对计算过元组数量的影响

Fig.9　Effect of dimensions on the number of calculated tuples

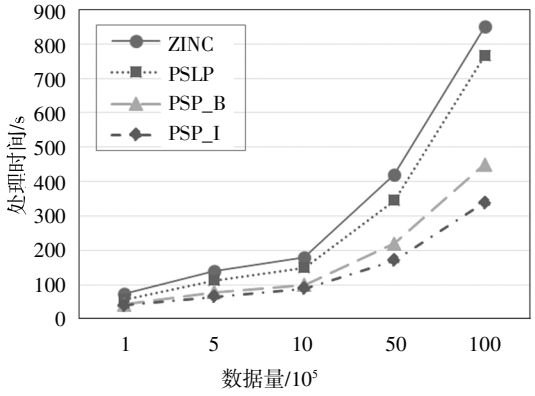
余点，节省了计算时间与计算成本,PSP_I 算法在 PSP_B 算法的基础上对数据集进行分组，通过过滤方法对整组数据进行过滤,进一步提高了过滤效率.

5.2 数据集规模对算法性能的影响

测试了数据集规模对实验的影响,在同样$|TO|=2$,$|PO|=3$ 的情况下,分别以数据量为 10^5、5×10^5、10^6、5×10^6、10^7 进行实验. 为了更加稳定地测出算法性能,采用多次实验,取平均值的方式进行实验记录. 图 10 所示为数据量不同的情况下计算 skyline 所需要的时间对比,图 11 则说明了在数据量不同的条件下查询需要计算的元组数量.



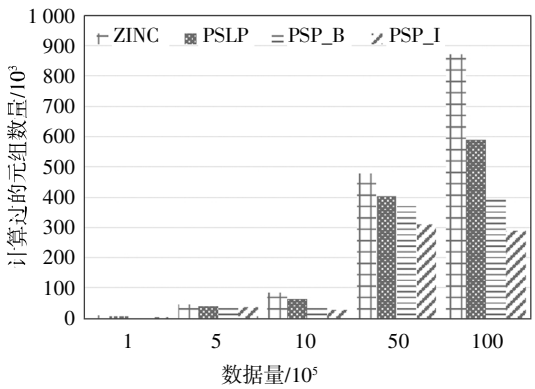
(a)独立分布数据集



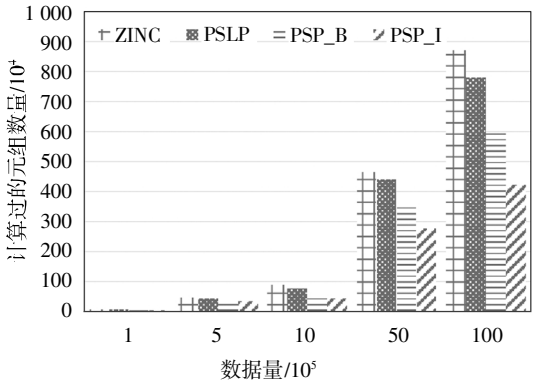
(b)反相关分布数据集

图 10 数据集规模对计算时间的影响

Fig.10 The effect of cardinality on calculation time



(a)独立分布数据集



(b)反相关分布数据集

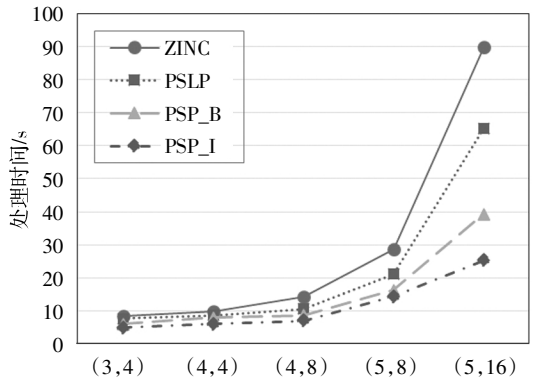
图 11 数据集规模对计算过元组数量的影响

Fig.11 The effect of cardinality on the number of calculated tuples

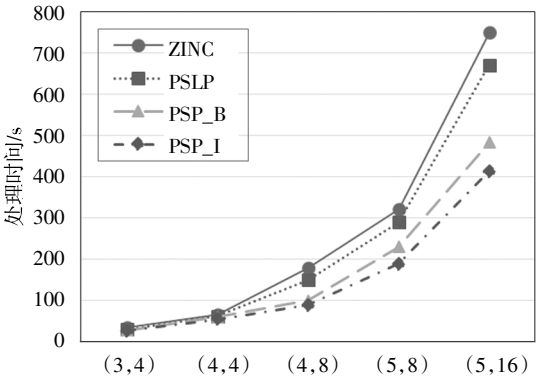
图 10 和图 11 描述了当数据集规模不同时,4 种算法进行 skyline 查询所需要的处理时间和所要计算的元组数量. 从图中可以看出，随着数据规模增大,计算所需的时间也明显增加,特别是 ZINC 算法和 PSLP 算法，在数据量增大时显出明显的劣势;在数据量增大时,PSP_B 算法通过倒排索引可以对数据集进行提前过滤，避免了大量不必要的冗余点的计算，明显加快了计算速度;PSP_I 算法在此基础上通过分组倒排可以一次性过滤掉整个分组，提高了过滤效率，进一步减少了计算时间. 实验表明 PSP_I 算法有明显的过滤剪枝效果，并且当数据量增大时对计算效率的提升效果更明显.

5.3 哈斯图宽度和深度对算法性能的影响

比较了当用户偏好哈斯图不同时 4 种算法的表现情况，实验除哈斯图形状外其他参数均采用实验默认值，并采用多次实验求平均值的方式进行实验记录. 图 12 和图 13 分别记录了在哈斯图宽度和深度不同情况下的实验结果.



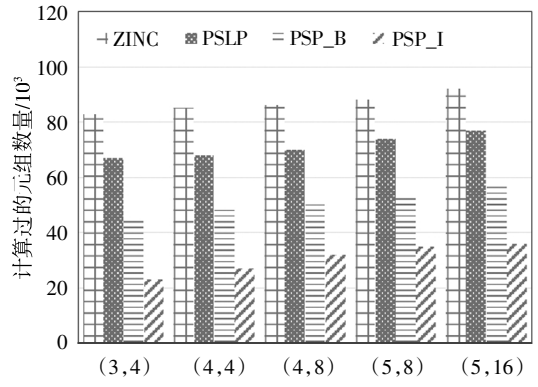
(a)独立分布数据集



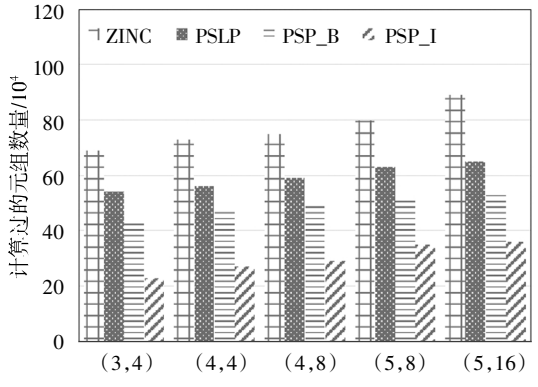
(b)反相关分布数据集

图 12　哈斯图宽度和深度对计算时间的影响

Fig.12　The effect of the depth and width of the hasse on running time



(a)独立分布数据集



(b)反相关分布数据集

图 13　哈斯图宽度和深度对计算过元组数量的影响

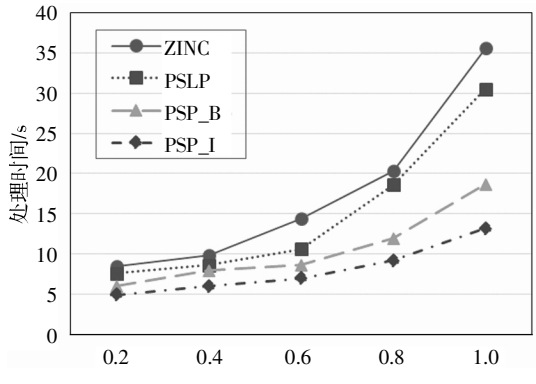
Fig.13　The effect of the depth and width of the hasse on the number of calculated tuples

从图 12 中可以看出在哈斯图宽度和深度不同的情况下,PSP_B 算法和 PSP_I 算法计算时间均少于 ZINC 和 PSLP 算法,且随着宽度和深度的增加差距逐渐明显. 图 13 则说明了 PSP_I 算法剪枝效果的优越性，该算法大量地减少了需要计算的元组数量. 通过独立分布数据集和反相关数据集的对比实验表明在不同的数据分布情况下本文提出的算法对 skyline 计算效率有明显提高,通过对数据剪枝过滤减少了计算时间.

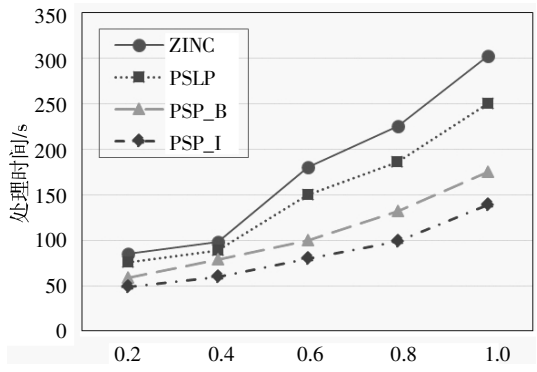
5.4　哈斯图密度对算法性能的影响

本小节的实验比较了在其他属性均取默认值的情况下，不同的哈斯图密度对实验结果的影响. 图 11 记录了哈斯图密度分别取 0.2、0.4、0.6、0.8 和 1.0 时,4 种算法的计算时间. 同之前的实验相同,采用多次实验求平均值的方法进行实验记录,结果如图 14、图 15 所示.

图 14 表明 4 种算法在哈斯图密度不同的情况下进行 skyline 计算所需要的时间. 从图中可以看出，在哈斯图密度不同的情况下本文提出的 PSP_B 算法和 PSP_I 算法所需的计算时间均少于之前的算法.图 15 表明本文提出的算法有明显的过滤效果,大量减少了元组的计算数量，证明了本文提出的算法可以有效地提高 skyline 查询的计算效率.



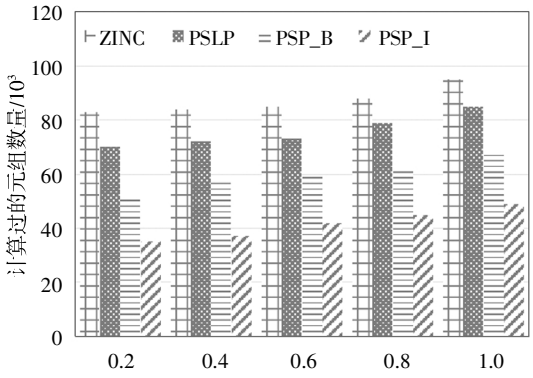
(a)独立分布数据集



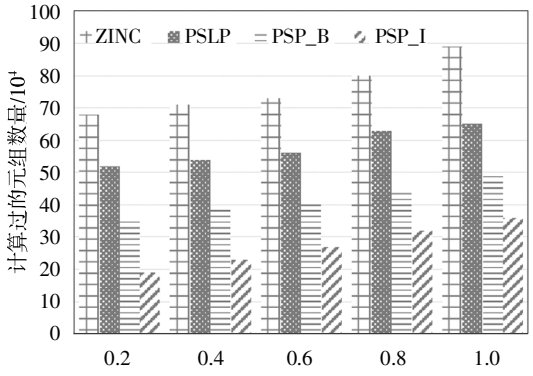
(b)反相关分布数据集

图 14　哈斯图密度对计算时间的影响

Fig.14　The effect of the density of the hasse on running time



(a)独立分布数据集



(b)反相关分布数据集

图 15 哈斯图密度对计算过元组数量的影响

Fig.15 The effect of the density of the hasse on the number of calculated tuples

6 结 论

本文对偏序域上的 skyline 查询技术进行了深入研究. 首先提出将倒排索引引入 skyline 查询领域,利用倒排索引来管理数据,在查询前进行大量的剪枝过滤, 并通过实验证明其可以有效地提高查询效率;在此基础上提出了一种优化算法,在建立倒排索引前通过分组的方式对数据进行预处理, 在进行计算时将一定不会成为结果的分组整组过滤, 从而进一步加快了过滤效率,提高计算速度. 实验结果表明, 本文提出的新算法在计算速度和过滤效果上优于之前的算法,具有合理性.

参考文献

[1] CHAN C Y,ENG P K,TAN L K,*et al*. Stratified computation of skylines with partially-ordered domains [C]//Proceedings of the ACM SIGMOD International Conference on Management of Data. Baltimore,Maryland,USA:DBLP,2005:203—214.

[2] SACHARIDIS D,PAPADOPOULOS S,PAPADIAS D. Topologically sorted skylines for partially ordered domains [C]// 2009 IEEE 25th International Conference on Data Engineering. Shanghai:IEEE,2009:1072—1083.

[3] LIU B,CHAN C Y. ZINC:efficient indexing for skyline computation [J]. Proceedings of the VLDB Endowment,2010,4(3):197—207.

[4] YIN B,GU K. Parallel skyline computation for partially ordered domains [C]// 2017 IEEE International Symposium on Parallel and Distributed Processing with Applications and 2017 IEEE International Conference on Ubiquitous Computing and Communications (ISPA/IUCC). Guangzhou:IEEE,2017:699—706.

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

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

[7] TAN K L,ENG P K,OOI B C. Efficient progressive skyline computation [C]// 27th International Conference on Very Large Data Bases. Roma:DBLP,2001:301—310.

[8] KOSSMANN D,RAMSAK F,ROST S. Shooting stars in the sky:an online algorithm for skyline queries [C]// Proceedings of 28th International Conference on Very Large Data Bases. Hong Kong:DBLP,2002:275—286.

[9] HUANG Z H,WANG Z H,GUO J K,*et al*. Efficient preprocessing of subspace skyline queries in P2P networks [J]. Journal of Software,2009,20(7):1825—1838.

[10] WU P,ZHANG C J,FENG Y,*et al*. Parallelizing skyline queries for scalable distribution [C]// International Conference on Advances in Database Technology-edbt. Heidelberg:Springer-Verlag,2006:112—130.

[11] XIN J C,BAI M,WANG G R. Efficient threshold skyline query processing in uncertain databases [C]// Seventh International Conference on Natural Computation. Shanghai:IEEE,2011:311—315.

[12] HSUEH Y L,LIN C C,CHANG C C. An efficient indexing method for skyline computations with partially ordered domains [J]. IEEE Transactions on Knowledge & Data Engineering,2017,29(5):963—976.

[13] 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[J]. 计算机学报,2016,39(10):81—104.
BAI M,XIN J C,WANG G R,*et al*. Research on dynamic skyline query processing over data streams[J]. Chinese Journal of Computers,2016,39(10):81—104. (In Chinese)

[14] PAPADIAS D,TAO Y,FU G,*et al*. An optimal and progressive algorithm for skyline queries[C]//Acm Sigmod International Conference. San Diego,California:SIGMOD,2003:467—478.