

基于高维空间的在线高效子空间 Skyline 算法——CSky

周红福 官学庆 郑 凯 周傲英

(复旦大学计算机科学与工程系 上海 200433)

摘 要 Skyline 计算是要发现数据集中不被其他点支配的所有点的集合. 近来,它在实时在线服务方面的良好应用前景,使其成为数据库研究领域的一个热点. 实际应用中,用户通常期望快速、渐进地返回 Skyline 计算结果,因此文中主要讨论了高维空间子空间 Skyline 渐进查询问题. 据我们所知,现有的 Skyline 计算方法都不能直接或者通过简单修改来高效解决该种查询问题. BNL(Blocked Nested Loop)算法是一个可用来进行子空间 Skyline 计算的算法,但是,该方法低效且非渐进. 基于此,文中提出了在线高效子空间 Skyline 算法——CSky(Count the Skyline). 该算法充分利用了一个新颖数据结构——InvertS 的特征,即通过对目标数据集进行排序,存放最可能为 Skyline 点的数据于算法优先扫描的位置,这使得 CSky 算法能高效计算出任意子空间上的 Skyline;同时,CSky 每次计算子空间 Skyline 查询时,至多访问一遍数据库;再有,算法扫描一个点时,只需和当前已发现的 Skyline 点进行比较即能判断该点是否为 Skyline 点,保证了算法的渐进性. 这样,相比 BNL,CSky 大大减少了计算开销,具有其他基于索引的 Skyline 算法计算 Skyline 时的高效,且这种高效适用于所有子空间. 理论分析和实验表明,在解决高维空间子空间 Skyline 查询问题方面,CSky 性能大大优于 BNL.

关键词 轮廓;子空间;渐进算法;在线算法

中图法分类号 TP301

CSky: An Online Efficient Algorithm for Subspace Skyline Computation in High Dimensional Space

ZHOU Hong-Fu GONG Xue-Qing ZHENG Kai ZHOU Ao-Ying

(Department of Computer Science and Engineering, Fudan University, Shanghai 200433)

Abstract Skyline computation aims to find the points that are not dominated by any other point in the dataset. It has been becoming a hot topic due to its potential applications in real-time online services. Usually, such applications expect to return the first Skyline point quickly, without ransacking all the points. This paper focuses on the problem of progressive subspace Skyline queries in high dimensional space. To the best of our knowledge, the existing algorithms and their variations cannot be easily extended to support arbitrary subspace Skyline query efficiently. The BNL (Blocked Nested Loop) method can be used for subspace Skyline queries, but it is very inefficiently, and not progressive. A novel algorithm, called CSky(stands for Count the Skyline), is proposed in this paper to solve this problem. With CSky, the Skyline points can be rapidly obtained in any query subspace of a high dimensional space. It is an online algorithm based on a novel data structure called InvertS. The algorithm scans the dataset at most one pass, and the points are only compared with those detected Skyline points, thus resulting in a much smaller

收稿日期:2007-03-06;修改稿收到日期:2007-05-22. 本课题得到国家自然科学基金(60496327,60496325)资助. 周红福,男,1977年生,博士研究生,主要研究方向为数据挖掘和商务智能、传感器网络. 官学庆,男,1974年生,博士,主要研究方向为 XML 数据管理、商务智能. E-mail: gongxq@fudan.edu.cn. 郑 凯,男,1983年生,硕士研究生,主要研究方向为数据挖掘与数据流. 周傲英,男,1965年生,教授,博士生导师,主要研究领域为流数据管理、数据挖掘和商务智能、XML 数据管理以及对等计算.

computation overhead in comparison with BNL. Furthermore, CSky not only can efficiently find the Skyline like other index-based algorithms, but also do it in any subspace of the whole query space. The theoretical analysis and experimental comparison show that the CSky algorithm outperforms BNL significantly in various kinds of application cases.

Keywords Skyline; subspace; progressive algorithm; online algorithm

1 引 言

随着信息与通信技术的迅速发展,信息收集手段变得更加多样化.一方面,越来越多的数据被收集并保存在数据库中;另一方面,所收集数据的属性也愈加丰富.如何利用这些数据并从中高效获取知识成为信息化应用系统发展的目标,为此,很多研究者从不同的角度进行了各种有益的探索.表 1 是 97~98 赛季 NBA 部分球员统计数据的一个简单示例,包含有 *id*, *last name*, *minutes*, *pts*, *oreb* 等多种属性.利用这些数据,球队教练可以较全面地掌握不同球员的比赛表现.通常,这些具有丰富属性的数据对象可被看作高维空间中的向量或数据点,使用传统的数据库操作往往很难满足用户对某些数据分析的需求.例如,当选择属性 *pts*, *oreb*, *reb*, *asts*, *stl*, *blk* 作为衡量球员的主要标准时,由于球员在具体技术指标上往往各有千秋,利用传统的数据库处理技术很难评价哪些球员的综合技术统计数据较好.一种新的数据库操作——Skyline^[1],正逐渐成为当前的

研究热点之一,其目的是要发现数据集中所有不被任何其他点支配的数据点集.如该例,通过计算球员统计数据集上的 Skyline,可以在大量的球员数据中快速、有效地找出那些综合技术统计数据优异的球员.

一般,在数据集中,不被任何其他点支配的数据点被称作 SP(Skyline Point).对于两个给定的数据点 *A* 和 *B*,如果数据点 *A* 的每一维属性都不比数据点 *B* 的对应属性“差”,并且 *A* 至少有一维属性比 *B* “好”(“差”和“好”并无统一的定义,它根据用户的选择和喜好有不同的语义),则称数据点 *A* 支配数据点 *B*.例如,在表 1 所示 NBA 球员统计数据集中,球员 Jordan 在上文所列的主要技术统计属性上就支配球员 Pippen;而球员 Jordan 和 O’neal 都在某个技术统计指标上低于对方,故相互之间均不能支配对方.在这个数据集中,除 Pippen 外的每一个球员都不被任何其他球员支配,这些球员的综合技术统计数据都是相对优异的,他们构成了这个数据集合的 Skyline.

表 1 NBA 球员统计数据示例

| <i>id</i> | <i>last name</i> | <i>minutes</i> | <i>pts</i> | <i>oreb</i> | <i>reb</i> | <i>asts</i> | <i>stl</i> | <i>blk</i> |
|-----------|------------------|----------------|------------|-------------|------------|-------------|------------|------------|
| JORDAMI01 | Jordan | 3181 | 2357 | 130 | 475 | 283 | 141 | 45 |
| O’NEASH01 | O’neal | 2175 | 1699 | 208 | 681 | 142 | 39 | 144 |
| RODMADE01 | Rodman | 2856 | 375 | 421 | 1201 | 230 | 47 | 18 |
| PIPPESC01 | Pippen | 1652 | 841 | 53 | 227 | 254 | 79 | 43 |
| ROBINDA01 | Robinson | 2457 | 1574 | 239 | 775 | 199 | 64 | 192 |
| MALONKA01 | Malone | 3030 | 2190 | 189 | 834 | 316 | 96 | 70 |
| STOCKJO01 | Stockton | 1858 | 770 | 35 | 166 | 543 | 89 | 10 |

在计算数据集上的 Skyline 时,往往维数越多,不被任何其他点支配的数据点就越多.如在表 1 所示数据集中,随着统计指标项的增多,难有一个球员在所有方面优于别的球员,因此会有更多的球员不被任何其他球员支配.事实上,教练并不一定对球员的所有统计指标都感兴趣,而可能更关心在不同指标组合下球员的表现.例如,教练可能更关注进攻表现,如只对 *pts* 和 *asts* 这两项统计指标感兴趣;在这两维上,球员 Jordan, Malone 和 Stockton 构成了目标数据集的 Skyline,这有助于教练缩小考察最感兴

趣进攻球员的范围.由此可见,对于高维数据集来说,子空间上 Skyline 的计算往往更具有现实的应用价值.对于维数不高(<6)的数据空间的子空间 Skyline 计算,Papadias 等^[2]已经给出了有效的解决方案.高维数据空间的子空间 Skyline 计算问题,主要存在如下困难:

- (1) Skyline 计算本身代价较高,在最坏情况下的计算复杂度为 $O(kn^2)$. 其中, k 是数据空间的维数, n 是数据空间中数据点的个数;
- (2) 即便单个子空间 Skyline 计算所涉及的维

数不多,但由于高维空间下子空间的数目非常多(2^k).这样,高维空间下的子空间 Skyline 计算会带来的“维度灾难”问题,使得原本复杂的 Skyline 计算变得更加困难;

(3)高维数据空间和其子空间的 Skyline 之间并不是简单的包含与被包含关系;

(4)大量应用要求算法具有渐进性(progressive)和偏好性(preferable),即算法获得 SP 点的响应时间快,并且能根据用户对不同属性的喜好程度给出相应的 Skyline 结果.

针对上述困难,本文提出了一个基于 InvertS 数据结构的子空间 Skyline 计算方法——CSky.该算法利用 Skyline 定义的特点和 InvertS 结构的特性,通过计数每个数据点在结构中出现的频次,能够快速返回高维空间任意查询子空间上的 Skyline,且结果的返回是渐进的.本文的主要贡献如下:

(1)本文提出了高维空间的子空间 Skyline 查询(Subspace Skyline Query in High dimensional space,SSQH)的概念,并设计了在线算法 CSky 用于解决 SSQH 问题.对于高维数据集任意子空间上的 Skyline 计算,CSky 算法都可以通过至多一遍扫描来完成.

(2)本文引入了一个新颖的数据结构——InvertS 结构.通过对目标数据集的各维度进行排序,并将最可能为 Skyline 点的数据保存于被优先扫描的位置,使得 CSky 算法能够高效、渐进地得到高维空间子空间上的 Skyline 计算结果.

(3)设计了详细的 CSky 性能评价实验,并与 BNL^[1]算法进行了性能比较.实验结果表明,对于子空间 Skyline 查询,CSky 的性能明显优于 BNL.

本文第 2 节介绍相关的预备知识,包括子空间 Skyline 查询的定义以及 InvertS 数据结构的描述;第 3 节详细描述 CSky 算法并对其进行复杂度分析;实验及相关工作分别在第 4 节和第 5 节中介绍;第 6 节对全文进行总结,并简要介绍未来的研究方向.

2 预备知识

2.1 子空间 Skyline

子空间 Skyline 查询形式表述如下:设高维空间集 D 由属性集 $s = \{s_1, s_2, \dots, s_k\}$ 构成, D' 是由空间集 D 在任何非空属性集 $t(t \subseteq s)$ 上的投影组成,称 D 对应的 D' 上的 Skyline 查询为属性空间 s 上子空间 Skyline 查询.数据集在每一个属性集 s_i 上的投影组成的集合记作 D_i .文献[1-3]等都对 Skyline 操作

进行了相应的定义,这里采用如下定义形式:

Skyline 查询. 查询对象数据集中所有不被其他点支配的点组成的集合.

支配. 设 D_1, D_2, \dots, D_k 是 k 个全序集,记全序关系为 $<$. D 定义在笛卡儿积 $D_1 \times D_2 \times \dots \times D_k$ 上,它是有 n 个 k 维向量组成的集合.向量 $p(p_1, p_2, \dots, p_k) \in D$, p_i 表示向量 p 的第 i 维值($i = 1, 2, \dots, k$).对于 D 中任意两向量 p, q ,定义 p “支配” q ,当且仅当对所有的 i , $p_i < q_i$,且 $p \neq q$.

另外,这里定义一个新的概念 $Rank_i$:任意向量 $p \in D$ 的属性值 p_i 在集合 D_i 中的序,记作 $Rank_i$.为表述方便,以后“序”统一指由大到小的降序.

2.2 InvertS 数据结构

为便于解释 InvertS 这个数据结构,抽象 InvertS 结构如表 2 所示.首先,该结构由多个列表组成,每个列表对应数据集 D 中一维数据.

表 2 InvertS 抽象数据结构类型

| dim_1 | dim_2 | ... | dim_k |
|-------------|-------------|-----|-------------|
| B_{11} | B_{12} | ... | B_{1k} |
| B_{21} | B_{22} | ... | B_{2k} |
| ... | ... | ... | ... |
| $B_{m_1 1}$ | $B_{m_2 2}$ | ... | $B_{m_k k}$ |

其次,每一列表 dim_i 由数个桶 B_{ji} 组成,且每一列表中所含桶的数目与属性 D_i 上相异元多少相关.桶中存放指向元组的指针,这里可以理解为点的 id .另外,某一 id 放在列表的哪一个桶中由其相应属性值的 $Rank_i$ 所决定.如,点 p 在属性 s_i 上的值为 p_i ,其对应的 $Rank_i$ 不妨假设为 j ,那么 p 的 id 值就存放在桶 B_{ji} 中.若一个桶中含有多个元素,这些元素在桶中的顺序根据 id 值有序存放.

以上,介绍了 InvertS 数据结构,接下来以一个简单例子来具体说明该结构.表 3 给出了例子数据集 E ,第 1 行指的是数据集中点 id ,第 2~4 行分别指数据点在这 3 个属性 $\{s_1, s_2, s_3\}$ 上的值.这里,省略了各属性的具体单位形式以使其具有更大的代表性.表 4 给出了数据集 E 对应的 InvertS 结构.

表 3 简单例子数据

| id | s_1 | s_2 | s_3 |
|------|-------|-------|-------|
| 1 | 18 | 11 | 19 |
| 2 | 18 | 14 | 18 |
| 3 | 17 | 14 | 15 |
| 4 | 15 | 15 | 17 |
| 5 | 19 | 10 | 18 |
| 6 | 18 | 10 | 19 |
| 7 | 16 | 13 | 17 |
| 8 | 15 | 14 | 15 |
| 9 | 16 | 11 | 12 |

表 4 简单例子数据 InvertS 结构

| d_1 | d_2 | d_3 |
|---------|---------|---------|
| {5} | {4} | {1,6} |
| {1,2,6} | {2,3,8} | {2,5} |
| {3} | {7} | {3,4,7} |
| {7,9} | {1,9} | {8} |
| {4,8} | {5,6} | {9} |

首先, InvertS 结构是由三个列表组成, 原数据集 E 中每一维数据对应表 4 InvertS 结构中的一列; 其次, 每个列表由数个桶构成. 如, 列表 d_1 中包含 $\{5\}, \{1, 2, 6\}, \{3\}, \{7, 9\}, \{4, 8\}$; 同时, 每个列表不是存储该维相应的属性值, 而是存放对应的 id 值; 再次, 每个桶的放置是有顺序的, 如列表 d_1 中第一个桶是 $\{5\}$, 是因 id 是 5 的元组其在 s_1 上属性值 19 为所有元组在该属性上的最大值, 即其 $Rank_{s_1}$ 为 1. 同样的, id 为 1, 2, 6 的 $Rank_{s_1}$ 为 2, 故它们被放在第二个位置, 桶中各元素统一根据 id 值升序存放.

3 CSky 算法

本节首先详细描述了 CSky 算法, 然后给出了理论分析和证明; 算法的一些主要特性在本节最后一部分给出. 表 5 列出了算法中用到的一些概念和记号.

表 5 符号表

| 记号 | 定义 |
|-----------|--------------|
| f | 偏好函数 |
| css | 候选 Skyline 集 |
| M | 数据集的势 |
| Num | 查询空间维数 |
| $MaxFreq$ | 点频最大值 |
| SP | Skyline 点 |
| pl | invertS 结构指针 |
| $spList$ | Skyline 点列表 |

3.1 算法描述

在介绍算法之前, 先给出算法中经常用到的一个概念的定义——点频. 所谓一个点的点频是指 InvertS 结构中的点 id 在算法执行过程中被扫描到的次数. 一般, 记点 a 的点频为 $PF(a)$. 另外, 这里给出有关 CSky 算法的一些假定. (1) 算法执行前, 假定相应的 InvertS 结构已经存在; (2) 对于用户提交的查询空间, 假定预处理后, 已存储至相应的子空间结构中. 另外, 查询空间的维数应大于 1, 因为一维 Skyline 查询是平凡的.

算法 1 给出了详细描述 CSky 算法的伪代码. 首先, 算法将所有的 PF 计数器设置为零. 代码行 3~4 表明了 CSky 算法扫描 InvertS 结构中各桶的

顺序. 以表 2 抽象 InvertS 结构为例, 根据假设的用户喜好属性顺序, CSky 依次访问桶 $B_{1i_1}, B_{1i_2}, \dots, B_{1, i_{Num}-1}$, 然后从 B_{2i_1} 继续访问. 一般地, 如果算法遇到任何需访问的某列 i 的桶 B_{ji} 不存在, 那么算法结束; 已经找到所有的 SP . 现在, 考虑当访问某桶中的点 id 时, CSky 算法如何处理, 主要包括如下 3 种情形:

- (1) 如果点 p 已经被标识为 SP 点, 将其 PF 增加 1;
- (2) 如果点 p 已经被标识为被支配点, 那么什么也不用做, 处理下一个点;
- (3) 如果点 p 第一次被访问, 那么需判断其和列表 $spList$ 中点的支配关系, 算法 2 用来实现甄别点 p 和 $spList$ 中点的支配关系.

接下来, 简单讨论第三种情形. 如果 $isDominate(p, spList)$ 返回 true, 也就是说 p 被 $spList$ 中某点所支配, 那么标记 p 为被支配点; 否则, 基于偏好函数 f 值非递减序插入 css 中. 扫描完一个桶后, CSky 使用一个排序的 BNL 算法^[1] 计算 css 中的 SP 点, 并将它们合并至 $spList$ 中; 同时, 更新在 css 中新发现所有 SP 点的 PF 值并清空 css 集. 这里, 可以看出, 任一点第一次被算法扫描后, 可即时被判断是否为 SP . 最后, 算法执行过程中若存在一个 PF 计数器值等于 Num 时, 所有查询空间的 SP 被返回, 算法终止, 这个结论见下一小节定理 2 的证明. CSky 算法在判断支配关系时, 采用了一个简单的启发式规则, 表述成下面的推论(容易证明).

推论 1. 查询空间中任意两点 p, q , 若偏好函数 $f(p) \geq f(q)$, 则 q 不可能支配 p .

算法 1. CSky Algorithm ($subspace, f, pl$).
 $subspace$: The dimensional set preferred by the user
 f : the preference function
 pl : pointer to invertS structure
Output: produce the Skyline point progressively

1. Initiate PF counters and $MaxFreq$ for zero;
2. construct empty $spList$;
3. for ($i=0$; ($i < m$) && ($MaxFreq < Num$); $i++$) do
4. for ($k=0$; ($k < Num$) && ($MaxFreq < Num$); $k++$) do
5. $j=subspace[k]$;
6. for (Any Point G in B_{ij}) do
7. if p is Skyline point then
8. if ($++p'PF > MaxFreq$) then
9. $MaxFreq=p'PF$;

```
10. else
11. if ( $p$  have not been tagged as dominated point)
    then
12. if ( $isDominated(p, spList)$ ) then
13. Tag  $p$  dominated;
14. else
15. Insert  $p$  into  $css$  in non-descending order by prefer-
    ring function  $f$ ;
16.  $tempSPL \leftarrow BNL(lenofcss, css)$ ;
17. for (Any Point  $q$  in  $tempSPL$ ) do
18. Output  $q$ ;
19. if ( $++q'PF > MaxFreq$ ) then
20.  $MaxFreq = q'PF$ ;
21.  $spList \leftarrow Merge(tempSPL, spList)$ ;
22. empty  $css$  and  $tempSPL$ .
```

算法 2. $isDominated(p, list)$.

p : an point to be examined
list: an non-descending Num -dim dataset
Output: whether p is dominated by list

```
1. for (any point  $q$  in list) do
2. if  $f(q) < f(p)$  then
3. Compare( $q, p, Num$ ); //use Skyline definition
4. if (Compare( $q, p, Num$ ) is true) then
5. return(true);
6. else
7. return(false);
8. return(false).
```

为了方便理解算法,这里以一个三维查询空间的例子来说明 CSky 的算法的执行过程. 利用前面所举简单例子,原始数据和被转换后的 InvertS 结构数据分别存储在表 3 和表 4 中. 依据各维的属性值,InvertS 结构将每维分成不同的桶. 例如, d_1 在表 4 中被分成 5 个桶: $\{5\}$, $\{1, 2, 6\}$, $\{3\}$, $\{7, 9\}$ 和 $\{4, 8\}$.

现在,我们来解释 CSky 算法如何来扫描 InvertS 结构. 表 6 展示了 CSky 扫描该例 InvertS 结构的整个过程. 首先,依据用户的偏好顺序算法扫描各属性的第一个桶. 这样,算法处理桶 $\{5\}$. 这时, $spList$ 列表为空,同时 $MaxFreq$ 初值为 0. 根据前面提到的第三种情形,桶 $\{5\}$ 中所有 id 也即 5 放入 css 列表中. 然后,算法调用 $BNL(lenofcss, css)$ 来计算出 css 中的 SP 点,明显地,点 5 就是 SP 点. 因此, $tempSPL$ 和 $MaxFreq$ 分别更新为 $\{5\}$ 和 1. 同时,点 5 被标记为 SP 且作为结果输出. 合并 $tempSPL$ 和 $spList$ 后, $tempSPL$ 和 css 都被重置为空. 接下来, CSky 处理桶 $\{4\}$, 算法仍然调用 $isDominated(p, spList)$,

$spList$ 中点 5 明显不能支配点 4, 因此点 4 被放入 css 列表中. 接下来的处理方式依据上面的过程来进行, 算法处理桶 $\{2, 5\}$ 后, $MaxFreq$ 等于 3, 也就是说与整个查询空间的维数相同, 这满足算法终止条件. 因此, 所有 SP 点 $\{5, 4, 1, 2\}$ 被找出, 算法终止.

表 6 InvertS 抽象数据结构类型

| <i>bucket</i> | <i>spList</i> | <i>css</i> | <i>tempSPL</i> | <i>MaxF</i> |
|------------------|------------------|-------------|----------------|-------------|
| $B_{00}\{5\}$ | \emptyset | $\{5\}$ | $\{5\}$ | 1 |
| $B_{00}\{4\}$ | $\{5\}$ | $\{4\}$ | $\{4\}$ | 1 |
| $B_{00}\{1, 6\}$ | $\{4, 5\}$ | $\{1, 6\}$ | $\{1\}$ | 1 |
| $B_{00}\{5\}$ | $\{1, 4, 5\}$ | $\{4\}$ | $\{2\}$ | 2 |
| $B_{00}\{5\}$ | $\{1, 2, 4, 5\}$ | \emptyset | \emptyset | 2 |
| $B_{00}\{5\}$ | $\{1, 2, 4, 5\}$ | \emptyset | \emptyset | 3 |

3.2 算法分析及优化

本部分,首先我们证明了 CSky 算法是有效的, 这意味着:(1)算法执行过程中的输出一定是 Skyline 的一部分;(2)算法终止时,所有的 SP 点都已经被产生. 然后,详细分析了 CSky 算法的复杂度.

定理 1. CSky 算法扫描一个桶后,如果存在两个点 a, b ; 它们的点频分别为 $a(PF)$ 和 $b(PF)$, 且 $a(PF) > b(PF)$; 那么点 a 不可能被点 b 所支配.

采用反证法证明. 扫描一个桶后,假设存在两个点 a, b , 它们的点频分别为 $a(PF)$ 和 $b(PF)$, 其中 $a(PF) > b(PF)$, 且 b 点支配点 a . 根据假设,若 b 支配 a , 那么在所有维上, b 点的值都不比 a 点小; 因此,根据算法扫描桶的顺序,在扫描完任何一个桶后, $a(PF)$ 都不会比 $b(PF)$ 大; 这和假设 $a(PF) > b(PF)$ 矛盾.

证毕.

我们要求算法能即时判断一个新扫描点是否为 SP . 但是,在算法执行过程中,只是将新扫描点和已经发现的 Skyline 点进行比较,严格意义上说,这时新发现的 SP 还仅仅是局部 SP , 即到当前扫描桶结束时被扫描的局部数据集的 SP . 然而,结合定理 1, 就可以证明这些局部 SP 也是整个数据集的 SP , 因为仍未被算法扫描的点其点频是 0, 不可能支配这些局部 SP . 这样,证明了算法执行过程中的输出一定是 Skyline 的一部分.

定理 2. CSky 算法扫描一个桶后,如果该桶中任何一个点频等于 Num , 那么所有的 SP 点已经被输出.

采用反证法证明. 假设扫描一个桶后,若存在某点 p 的点频 $p(PF)$ 更新为 Num , 且依据算法扫描顺序,在该桶之后仍然存在另外一点 q , 满足: (1) q 是 SP ; (2) q 在算法终止之前没有被作为 SP 输出. 那么,首先,如果 q 是 SP 且在算法终止前被

扫描过;根据前一部分算法描述,那么 q 会作为 SP 点被输出. 换句话说,在算法终止前 q 没有被算法扫描过;同时我们已假设 $p(PF)$ 为 Num ,也就是说 p 点在每一维都被算法扫描到;这样,根据 CSky 算法的扫描顺序, p 在查询空间每一维上值都比 q 对应维大,即 p 支配 q ;这和 q 是 SP 的假设矛盾. 证毕.

这样,当扫描一个桶后,在该桶之后没有被扫描过的点不可能支配前面被扫描的点,这由定理 1 保证. 同时,通过前一部分算法描述,我们知道,每扫描一个桶后都会将桶中第一次被扫描到的点和已经被确定为 SP 的点进行比较,同时也会和桶内可能成为 SP 的点作比较;因此,保证了每次扫描一个桶后的输出结果一定是查询空间 Skyline 的一部分. 定理 2 保证了没有一个 SP 点没被扫描到. 综合定理 1 和 2,保证了我们输出结果就是查询空间的 Skyline.

下面对 CSky 的复杂度进行分析,假设数据集 D 的势为 N ,其属性集 s 的势为 k ,查询空间属性集的势为 $k'(k' \leq k)$. 很明显,根据算法 1,算法的复杂度和何时扫描到一个点频等于 Num 密切相关,假设此时已扫描了 N' 个点 ($N' \leq N$), N' 的具体大小和查询空间的数据分布有关,通常 $N' \ll N$. 同时,算法的复杂度还与查询空间的 Skyline 集大小有关,假设其势为 M . 根据算法 1 的 3~7 行的循环,得到 CSky 的复杂度为 $k' \cdot M \cdot N'$.

3.3 算法特点

在文献[4]中,Hellerstein 等提出了一些在线算法的准则,Kossmann 等在文献[3]中也提出了在线算法另外一些准则. 这里,本文概括在线算法为如下 5 个性质:

- ①快速性. 能够尽可能快地返回部分结果,便于用户即时选择;
- ②准确性. 返回的结果没有不满足查询条件的,换句话说,所有返回结果应该都满足查询条件;同时,所有正确结果都被返回.
- ③渐进性. 随着算法执行时间的增加,返回的结果越来越多;
- ④可控性. 用户可以对返回的结果根据自己的偏好进行一些有用的控制;
- ⑤普遍性. 算法应该是基于标准技术,不应针对每维数据的分布作假设.

CSky 完全满足上面所列的在线算法性质. 首先,利用 InvertS 结构的预排序特点,算法可以即时计算查询空间部分 SP . 这是因为,结合 CSky 算法

描述和 SP 点定义,InvertS 结构中每维开始的桶中总是存在 SP 点且被计算输出. 在我们所有实验中,输出第一个结果耗时都是毫秒数量级. 其次,在定理 1 和 2 中已经保证输出结果一定是 SP 点,而且算法是渐进地输出所有结果. 第三,随着算法的执行,返回的 SP 点逐渐增多. 至于可控性,算法本身就支持用户对维度偏好的支持. 另外,算法并没有对数据分布作任何假设,采用的是排序和两路合并等标准算法.

综上,CSky 算法满足本文所提出的在线算法的所有准则. 同时,我们发现,CSky 也支持各种 SSQH 的变种查询,如 Top- k SSQH,范围 SSQH. Top- k SSQH 就是要找到目标数据集中 Top- k SP 点;范围 SSQH 是要在用户感兴趣的区域查找出 SP 点. 限于篇幅,这里不展开描述.

4 性能验证

本节详细给出多方面的性能验证结果. 现存的算法没有专门针对 SSQH 问题的,因此这里我们通过实现文献[1]中改进的 BNL 算法,来对比 CSky 算法的性能. 以下主要从 4 个方面进行对比.(1) SSQH 的维度影响;(2) SSQH 的渐进性质;(3) 数据集大小的影响. 所有的实验处理 10 维空间中的子空间. 运行环境是处理器为 Pentium IV 2.6GHz,内存 512MB,WinXP 系统,所有算法以 C++实现.

表 7 Skyline 尺寸

| 维度 | Skyline 尺寸 | | |
|----|------------|-------|-------|
| | 相关 | 独立 | 反相关 |
| 2 | 2 | 13 | 26 |
| 3 | 1 | 11 | 426 |
| 4 | 16 | 74 | 2787 |
| 5 | 19 | 324 | 9909 |
| 6 | 36 | 1264 | 22840 |
| 7 | 49 | 4051 | 38182 |
| 8 | 107 | 7441 | 51832 |
| 9 | 156 | 14225 | 63132 |
| 10 | 390 | 22490 | 75100 |

实验中的数据集采用 Skyline 计算领域通行的人造数据集:相关的、独立的和反相关的三种数据集,产生方法来自文献[1]. 每维的值域在[1,100],数据集大小变化在[100K,500K]. 所有的实验都表明,BNL 和 CSky 时间性能和 Skyline 查询结果所得尺寸大小有关,即 Skyline 结果集越大,所耗时间越多. 而 Skyline 尺寸和数据分布、维度、数据集大小有关. 实验表明相关数据集的 Skyline 尺寸最小,而

反相关数据集最大,独立数据集在两者之间,一般情况下,维度越高,Skyline 尺寸也越大.表 7 给出了不同分布数据集在 2~10 维时的 Skyline 具体尺寸值,数据集势为 100K.

4.1 维度对 CSky 的影响

本实验考察维度变化对 CSky 算法性能的影响.采用的数据集大小是 100K.从图 1 可以看出,在不同类型数据集、各种不同维度集情况下,CSky 算

法都比 BNL 表现稳定而且耗时少.特别是在较低维度(<8)的子空间查询计算中,CSky 的 SSQH 计算 CPU 耗时比 BNL 低一个数量级.较高维度 SSQH 查询时,CSky 也只需 BNL 耗时的 1/2 弱,实际应用中,这种情形相对较少出现.这里,对每一维度集的子空间时间耗时,取相同维度数的任意 10 个子空间耗时平均(除了 10 维空间),以使两种算法 CPU 耗时的对比更具一般性,下面实验同.

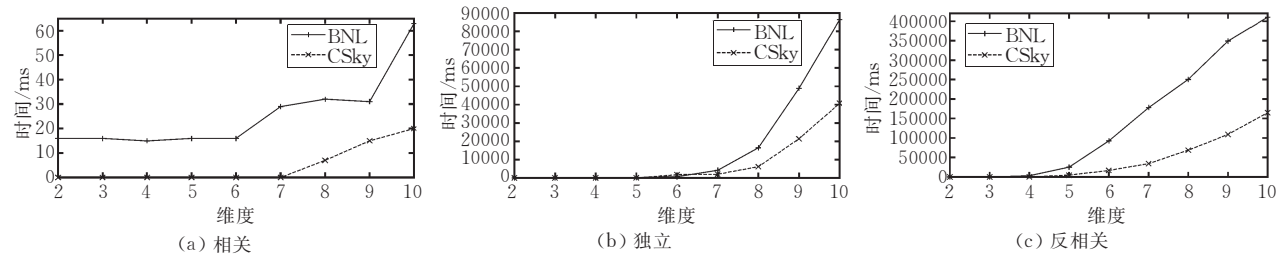


图 1 维度的影响

4.2 CSky 的渐进性质

本实验考察 CSky 算法的渐进性.由于相关数据集的 Skyline 尺寸较小,不容易反映渐进性的特点,因此,实验中选择 Skyline 尺寸较大的反相关和独立数据集.数据集大小是 100K,5 维查询子空间.图 2 横坐标是发现的 SP 点的个数,图中打点处表

示发现相应数目 SP 点所花费的时间.可以看出,CSky 几乎瞬时就能得到 Skyline 查询部分结果,随着计算的继续,渐进得到所有查询结果;而 BNL 几乎不具有渐进性.实际上,渐进性是 Skyline 计算实际应用中一个非常重要的特性.

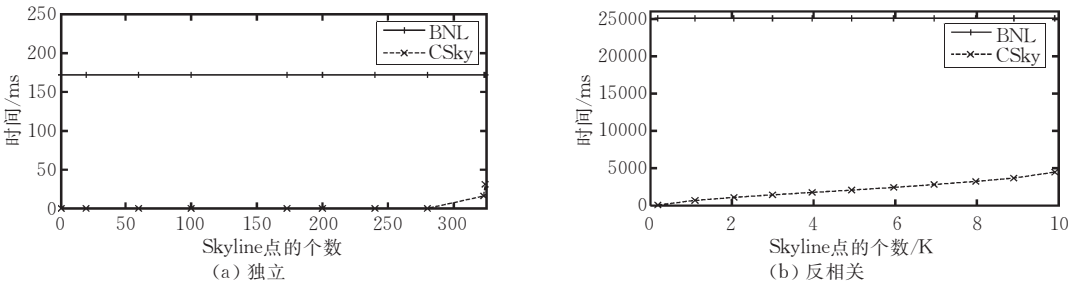


图 2 渐进特性

4.3 数据集大小的影响

本实验考察数据集大小对算法性能的影响.实验选择 5 维子空间作为查询空间,数据集选择反相关数据集,变化它的大小从 100K~500K.实验结果如图 3 所示,我们发现 BNL 和 CSky 都随着数据集的增大,CPU 耗时增加.但是,CSky 在所有的实验

中都比 BNL 用时少,同时随着数据集的增大,有着更平缓的增加,这是因为 BNL 的计算开销受数据集大小的影响更大.

5 相关工作

子空间问题常见于数据库各领域,如 OLAP 查询,聚类分析等.很多算法^[5-8]已被提出以解决这个问题.OLAP 主要在商务智能方面被广泛研究以用来解决多维查询的高效性.子空间聚类主要是通过选择相关的维,从而发现数据集子空间的聚类.一般地,我们可以将子空间聚类看作一种无监督的知识发现过程.SSQH 问题和 OLAP 查询及子空间聚类非常相似,但 SSQH 主要关注面向用户偏好的

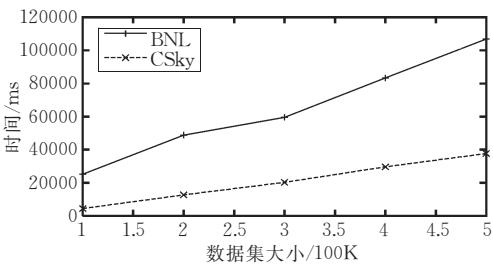


图 3 数据集大小对算法性能的影响

Skyline 计算.

Skyline 计算工作可以追溯到向量集中最大向量计算问题. 为了解决这个问题, Kung 等^[9]提出了一个复杂度为 $O(n(\log n)^{d-2})$ ($d \geq 4$) 和 $O(n \log n)$ ($d=2, 3$) 的算法. 如果数据每维独立分布, 则有更高效的线性时间复杂度的算法^[10]. 文献[1]首先在数据库领域提出了 Skyline 计算问题, 其中给出了一系列的算法, 如 Block Nested Loop (BNL) 算法, 分而治之 (Divide-and-Conquer) 算法. BNL 算法通过迭代, 将从数据集中读出的数据与内存中保存的候选 SP 比较, 从而计算出 SP 点; 分而治之的算法通过先分割数据集, 然后合并每一部分计算出的局部的 SP 点, 从而得到整个数据集的 Skyline. 自文献[1]之后, 一系列数据库领域内的 Skyline 算法被相继提出, 见文献[2-3, 11-19]. 其中, 文献[17]提出了一个 Index 算法, 主要思想是根据数据集中的各点在所有维中属性值的大小关系的特点来排序, 可以剪掉一些明显不是 SP 的点, 从而大大降低了 Skyline 计算的所费时间. 明显地, Index 算法的预排序利用了点在各维上值的特性, 因此它不能用来解决 SSQH 所要求的任意子空间 Skyline 计算问题; 分而治之的算法分割数据集的办法, 随着维度的增加其算法开销剧增使得其也不适合用来解决 SSQH 问题. BNL 可以用来计算 SSQH 查询问题, 但其本身性能限制了它的应用, 性能验证部分已经比较了它和 CSky 的性能优劣, 同时 BNL 一个更坏的性质是其不满足渐进性的特征, 只有扫描整个数据集后, BNL 才能返回 SP 点.

在文献[3]中, Kossmann 等结合了最近邻搜索的方法和 R-tree 索引数据的优点, 提出了最近邻 Skyline 计算方法, 大大改进了低维空间 Skyline 计算的性能. 文献[2]也是利用 R-tree 对数据集进行索引, 提出了 BBS 的 Skyline 计算算法, 克服了最近邻 Skyline 计算方法空间开销大以及同一点多次访问的弱点, 达到了 I/O 性能方面的最优. 这两个算法也不能用来解决 SSQH 问题, 因为它们都是利用 R-tree 来索引数据集, 我们知道, 对高维数据集利用 R-tree 索引进行 Skyline 计算, 其性能通常是不能接受的. 文献[16, 19]几乎同时提出了“Skyline Cube”概念, 强调查找所有子空间的 Skyline 的并集的操作, 充分利用了各子空间 Skyline 计算共享性质, 分别用 Bottom-Up 和 Top-down 两种策略来减少 Skyline 计算的时间复杂度. 但是, Skyline Cube 和 Skyline 本身, 两者的概念还是有较大的差别. 文

献[12]通过提出 k -dominant 概念, 解决了高维空间中的 k -dominant Skyline 查询问题, 也不宜直接用来解决高维空间的子空间查询问题. 本文针对的是 SSQH 问题, 即高维空间任意子空间的渐进高效 Skyline 计算, 提出的 CSky 算法充分利用了索引方法计算 Skyline 时的高效, 并使这种高效能被任意子空间所共享.

6 结束语

子空间 Skyline 计算是一个非常有趣且有挑战的工作. 为此, 本文提出了基于 Skyline 上下文的 SSQH 问题. 分析表明, 现有的算法不能直接应用来高效解决该问题. 为了克服这个困难, 本文提出了 CSky 算法. CSky 算法充分利用了一个数据结构 InvertS 的特性和 Skyline 定义的特点, 通过计数扫描点频, 从而能够高效、渐进地得到高维空间任意子空间的 Skyline 查询结果. 文中实验也表明了 CSky 算法能够有效地解决高维空间中子空间 Skyline 查询问题. 未来的研究工作主要从下面两个方面来考虑: 首先, 期望能够找到 Skyline 中有代表性的点, 实验中我们发现, 很多情况下 (例如独立数据集和反相关数据集), Skyline 尺寸非常大, 不便于用户直接筛选结果; 另外, 希望开发一个 Skyline 查询系统, 满足用户对任意空间的指定区域进行快速渐进 Skyline 查询.

参 考 文 献

- [1] Borzsonyi S, Kossmann D, Stocker K. The skyline operator//Proceedings of the 17th International Conference on Data Engineering. Heidelberg, Germany, 2001: 421-430
- [2] Papadias D, Tao Y, Fu G, Seeger B. An optimal progressive algorithm for skyline queries//Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data. San Diego, USA, 2003: 467-478
- [3] 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
- [4] Hellerstein J, Avnur R, Chou A, Hidber C, Olston C, Raman V, Rotha T, Haas P. Interactive data analysis: The control project. IEEE Computer, 1999, 32(8): 51-59
- [5] Agrawal R, Gehrke J, Gunopulos D, Raghavan P. Automatic subspace clustering of high dimensional data for data mining applications//Proceedings of the ACM SIGMOD International Conference on Management of Data. Seattle, Washington, USA, 1998: 95-105

[6] Han J W, Kamber M. Data Mining: Concepts and Techniques. New York: Morgan Kaufmann, 2000

[7] Parsons L, Haque E, Liu H. Subspace clustering for high dimensional data: A review. SIGKDD Explorations Newsletter, 2004, 6(1): 90-105

[8] Roussopoulos N, Kelley S, Vincent F. Nearest neighbor queries//Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data. San Jose, California, USA, 1995: 71-79

[9] Kung H T, Luccio F, Preparata F P. On finding the maxima of a set of vectors. Journal of the ACM, 1975, 22(4): 469-476

[10] Bentley J L, Clarkson K L, Levine D B. Fast linear expected-time algorithms for computing maxima and convex hulls//Proceedings of the 1st Annual ACM-SIAM Symposium on Discrete Algorithms. San Francisco, California, 1990: 179-187

[11] Chan C Y, Eng P K, Tan K L. Stratified computation of skylines with partially-ordered domains//Proceedings of the ACM SIGMOD International Conference on Management of Data. Baltimore, Maryland, USA, 2005: 203-214.

[12] Chan C Y, Jagadish H V, Tan K L, Tung K H, Zhang Z J. Finding k -dominant skylines in high dimensional space//Proceedings of the ACM SIGMOD International Conference on Management of Data. Chicago, Illinois, USA, 2006: 503-514

[13] Chomicki J, Godfrey P, Gryz J, Liang D. skyline with pre-sorting//Proceedings of the 19th International Conference on Data Engineering. Bangalore, India, 2003: 717-816

[14] Godfrey P, Shipley R, Gryz J. Maximal vector computation in large data sets//Proceedings of the 31st International Conference on Very Large Data Bases. Trondheim, Norway, 2005: 229-240

[15] Lin X M, Yuan Y D, Wang W, Lu H J. Stabbing the sky: Efficient skyline computation over sliding windows//Proceedings of the 21st International Conference on Data Engineering. Tokyo, Japan, 2005: 502-513

[16] Pei J, Jin W, Ester M, Tao Y F. Catching the best views of skyline: A semantic approach based on decisive subspaces//Proceedings of the 31st International Conference on Very Large Data Bases. Trondheim, Norway, 2005: 253-264

[17] 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

[18] Yip K Y, Cheung D W, Ng M K. On discovery of extremely low-dimensional clusters using semi-supervised projected clustering//Proceedings of the 21st International Conference on Data Engineering. Tokyo, Japan, 2005: 329-340

[19] Yuan Y D, Lin X M, Liu Q, Wang W, Yu J X, Zhang Q. Efficient computation of the skyline cube//Proceedings of the 31st International Conference on Very Large Data Bases. Trondheim, Norway, 2005: 241-252



ZHOU Hong-Fu, born in 1977, Ph.D. candidate. His research interests include data mining & business intelligence and sensor network.

GONG Xue-Qing, born in 1974, Ph.D. His research in-

terests include XML data management, business intelligence.

ZHENG Kai, born in 1983, M. S. candidate. His research interests include data mining and data stream.

ZHOU Ao-Ying, born in 1965, professor, Ph.D. supervisor. His research interests include stream data, data mining & business intelligence, XML data management, and Peer-to-Peer computing.

Background

This work is mainly supported by the National Natural Science Foundation of China under grant Nos. 60496327 and 60496325. The problem addressed in the paper is from the field of Skyline computation. Nearly, a large number of Skyline algorithms are explored in order to efficiently evaluate the Skyline in the dataset with the fixed dimensions.

The purpose of this work aims to solve the problem of Skyline computation of any subspace of high dimensional

space in order to develop a platform of multi-criteria decision making prototype system based on the Skyline view. The WebDB and P2P research group at Fudan University has produced fruitful results in the fields of data mining, Web service, business intelligence, similarity search, and security and trust management in P2P environment. This paper presents their recent research achievement in the study of the Skyline computation of high dimensional space.