ISSN 1000-9825, CODEN RUXUEW
Journal of Software, Vol.19, No.6, June 2008, pp.1386−1400
DOI: 10.3724/SP.J.1001.2008.01386
© 2008 by *Journal of Software*. All rights reserved.
E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

Skyline 查询处理*

魏小娟[1], 杨 婧[1], 李翠平[1,2+], 陈 红[1,2]

[1](中国人民大学 信息学院,北京 100872)
[2](数据工程与知识工程国家教育部重点实验室,北京 100872)

Skyline Query Processing

WEI Xiao-Juan[1], YANG Jing[1], LI Cui-Ping[1,2+], CHEN Hong[1,2]

[1](School of Information, Renmin University of China, Beijing 100872, China)
[2](Key Laboratory of Data Engineering and Knowledge Engineering of the Ministry of Education, Beijing 100872, China)
+ Corresponding author: E-mail: cuiping_li@263.net

Wei XJ, Yang J, Li CP, Chen H. Skyline query processing. *Journal of Software*, 2008,19(6):1386−1400. http://www.jos.org.cn/1000-9825/19/1386.htm

Abstract: This paper gives a survey on current Skyline queries techniques. It first introduces the background in which Skyline queries appear. Then it presents in-memory algorithms in Skyline query problem. Facing to the situation of large data sets, it further presents the techniques about Skyline query processing by two cases, with or without indices respectively. Evaluations of Skyline query methods are discussed after that. This paper also introduces the novel query model-SKYCUBE which is applied to process multi-Skyline queries in various subspaces and related research based on it. Additionally, it introduces the efficient algorithm to solve Skyline queries in various applicant environment and the extension of the Skyline query processing. Finally, this paper proposes several directions for further research on the topic of Skyline query processing.
Key words: Skyline query; Skyline point; dominance; multi-objective optimization; SKYCUBE

摘 要: 对目前的Skyline查询方法进行分类和综述.首先介绍Skyline查询处理问题产生的背景,然后介绍Skyline查询处理的内存算法,并从带索引和不带索引两个方面对现有的外存Skyline查询处理方法进行分类介绍,在每组算法后,都对该组算法进行了性能评价,然后介绍不同子空间上的多Skyline查询处理模型——SKYCUBE的概念和相关研究.另外,还介绍了不同应用环境下解决Skyline查询处理的策略以及Skyline查询处理问题的扩展,最后归结出Skyline查询处理后续研究的几个方向.
关键词: Skyline查询;SP;控制关系;多目标优化;SKYCUBE
中图法分类号: TP311 **文献标识码**: A

Skyline 查询[1]也称为 Pareto(帕类托,在不损害他方利益的条件下,自身已达最优),在多目标的选取决策中

* Supported by the National Natural Science Foundation of China under Grant Nos.60673138, 60603046 (国家自然科学基金); the Program for New Century Excellent Talents in University of China (新世纪优秀人才支持计划); the Program for Excellent Talents in Beijing of China under Grant No.35607025, (北京市优秀人才培养资助项目)
Received 2007-07-17; Accepted 2007-09-04

© 中国科学院软件研究所 http://www.jos.org.cn

非常有用.它和 Convex Hulls、Top-K 查询、Nearest Neighbor 查询等都有关系.具体地讲,Skyline 查询处理是指从给定的一个 D-维空间对象集合 S 中选择一个子集,该子集中的任意一个点都不能被 S 中的任意一个其他点所控制.所谓控制关系是指给定一个 D-维空间中的多个对象(集合 S),如果对象 p 至少在某一维上优于另一个对象 q,而在其他维上都不比对象 q 差(p 优于或等于 q),则说 p 能够控制 q.

Skyline 查询是一个典型的多目标优化的问题[2].对它的研究最早可以追溯到 1975 年.Kung 在文献[3]中针对二维或三维数据提出了一种查询复杂度为 $O(n\log_2 n)$的算法,针对大于三维的数据,提出了查询复杂度为 $O(n(\log_2 n)^{d-2})$的算法.后来,Bentley 在假定各维数据分布独立的情况下,提出了人们期望的线性查询算法[4].所有这些研究都假定数据集比较小,可以放入内存,所提出的算法也都是内存算法.

数据库领域的研究人员对 Skyline 查询的研究始于 2001 年,最早由 Borzsonyi 等人提出[1].主要关注在数据量很大、无法放入内存的情况下,如何处理 Skyline 查询.最近几年,对 Skyline 查询的研究大体上可以分为 4 类:

1) 单 Skyline 查询处理算法.该类算法假定所有的 Skyline 对象都处在某一个特定的 D-维空间中,返回的结果集合只有 1 个.根据查询过程中是否借助索引,单 Skyline 查询处理算法又分为两类:不带索引算法和带索引算法.前者假定没有任何索引存在,通过扫描整个数据集(至少 1 次)来返回 Skyline 查询的结果;后者通过引入适当的索引结构,如 R-树,来提高查询处理的效率.

2) 多 Skyline 查询处理算法.针对现实生活中不同的用户可能有不同的兴趣和偏好,需要在不同的子空间中处理 Skyline 查询的需求,数据仓库和 OLAP 领域的研究者对在不同子空间上进行 Skyline 查询的研究产生了浓厚的兴趣,提出了 SKYCUBE[5]的概念.SKYCUBE 借用传统的 Data Cube 的多维层次结构,提出了有效的同时计算多个 Skyline 查询的思想.该类算法主要包含针对 SKYCUBE 的计算、维护和压缩等.

3) 不同应用环境下的 Skyline 查询处理.主要包括 Web 信息系统中的 Skyline 查询处理、P2P 网络环境下的 Skyline 查询处理、数据流环境下的 Skyline 查询处理、移动的公路网络环境下的 Skyline 查询处理等.

4) Skyline 查询处理问题的扩展.例如,文献[6]中首次扩展了空间数据库中不同数据点之间的控制关系的概念,将其用于经济学框架下的商业分析,提出了控制关系分析的概念;文献[7]针对高维空间下出现在 Skyline 查询结果中的点非常多,从而导致该结果在很多时候对用户失去意义的问题,提出了 k-Dominant 的概念,等等.

可以看出,Skyline 的查询处理问题已经引起了国内外研究者的高度重视,近几年,在 SIGMOD,VLDB,KDD,PODS,ICDE,ICDM 等相关的高水平国际会议上发表了许多高质量的论文,展示出大量的研究成果.在 TKDE,TODS 等期刊中也发表了大量成果.然而目前,国内和国际上还没有将 Skyline 查询处理的发展情况、核心技术和研究成果进行整体上的介绍.鉴于 Skyline 查询处理在多规则决策应用方面的重要价值和在实时在线服务方面的良好应用前景,为了捕捉 Skyline 查询处理的发展动态,对 Skyline 查询处理研究有一个总体上的把握,促进国内迅速跟上国际研究的步伐,综述这方面的工作十分有意义.

本文在分析国内外相关研究工作的基础上对 Skyline 查询处理技术进行了综述.本文第 1 节介绍 Skyline 查询处理问题产生的背景、动机和应用场景,并对问题的定义进行具体描述.第 2 节详述单 Skyline 的查询处理算法,包括内存处理算法和外存处理算法、带索引和不带索引的处理算法等.第 3 节具体介绍多 Skyline 的查询处理算法,包括 SKYCUBE 的计算、维护和压缩等.第 4 节介绍不同应用环境下 Skyline 的查询处理算法.第 5 节介绍 Skyline 查询处理问题的扩展.最后总结全文,并展望未来的研究工作.

1 Skyline 查询处理问题

1.1 问题描述

Skyline 查询问题也称为 Pareto 最优或极大向量问题.它是指从给定的一个 N-维空间的对象集合 S 中选择一个子集,该子集中的点都不能被 S 中的任意一个其他点所控制,满足这个条件的点称为 SP(skyline point).这里的控制关系是指给定一个 N-维空间中的多个对象(对象集 S),若存在这样两个对象 $P=(p_1,p_2,\ldots,p_N)$和 $Q=(q_1,q_2,\ldots,q_N)$,对象 P 在所有维上的属性值都不比对象 Q 差,且至少在一维上的属性值优于对象 Q,则称 P 控制 Q.

先来看一个经典的例子——饭店的选择入住问题[1].假设某个旅客到 Nassau 旅游,他/她想找一家价格便宜

© 中国科学院软件研究所 http://www.jos.org.cn

且距海滨近的饭店入住.图 1 中的每个小圆点都代表一家饭店,*X* 轴表示饭店的价格,*Y* 轴表示饭店到海滨的距离.可以看出,该旅客只需要考虑位于折线上的那些点(饭店)就可以了.不在折线上的点(饭店)根本用不着考虑,因为总是可以在折线上找到一点(饭店),或者价格上更便宜,或者距海滨更近.这时,我们说,折线上的点控制了所有其他不在折线上的点,而折线上的点之间不存在控制关系.所有位于折线上的点构成 Skyline 查询的结果集.

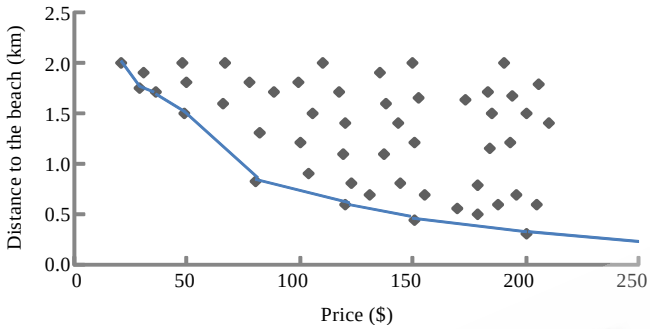

Fig.1 Skyline of hotels in Nassau (Bahamas)

图 1 有关 Nassau 饭店的 Skyline 查询问题

Borzsonyi 等人[1]提出用如下的一个 SQL 查询来表达一个 Skyline 查询:

*Select**, *From Hotels*, *Skyline of Price* min, *Distance* min

其中,min 表示价格和距离两个属性值都取 min 值的操作.这里的 min 也可以换成其他条件,如 max,joins, group-by 等等.

1.2 Skyline查询和Top-*K*查询

Skyline 查询与 Top-*K* 查询不同,但它们之间又具有一定的相似性.实际上,它们都用于解决传统的多目标优化问题,只是采用的手段不同.解决传统的多目标优化问题通常有 3 种不同的方法[8].

1) 转化成单目标优化的问题(Top-*K* 查询).主要思想是通过一个单调的加权函数将对象集合 *S* 中每个对象的多个属性进行聚集,得到一个单一值,通常被称为 Score 值,然后将所有的对象按照其 Score 值进行排序,再选出前 *K* 个最大或最小的对象,即为所要的查询结果.

2) Pareto 方法(Skyline 查询).不是将问题转化为单目标优化的问题,而是直接采用多目标优化算法解决原始的多目标问题.多目标优化算法最终返回的结果集是一系列平行的、互不受控制的解(位于 Skyline 上的多个 SP).这里的“控制”是指一个数据点在每一维上都不比另一点“差”,而且必须至少在一维上比另一点要“好”.其中,“差”和“好”并无统一的定义,根据用户的查询和选择条件的语义而定.

3) 词典序方法.即为不同的属性安排不同的优先级,然后按照优先级的高低来选择各个属性进行优化.按照优先级从高到低的顺序依次比较对象相应属性值的大小,若其属性值相同,则继续比较各个对象在下一个优先级上的属性值大小,直到比较出明确的优劣为止.

2 单 Skyline 查询处理算法

2.1 内存处理算法

文献[3,4]针对数据量小、可以放入内存时的求最大向量集问题的 Skyline 查询提出了解决算法.根据向量间的控制关系的概念[4]:若 *P* 的每个分量都大于等于 *Q* 对应的分量,则称向量 *P* 控制向量 *Q*.这样,若一个向量不被组中的任何其他向量所控制,这个向量就是这个组中的最大向量.针对最大向量问题,Kung 在文献[3]中针对 2 维或三维数据提出了查询复杂度为 $O(n\log_2 n)$ 的算法,针对大于三维的数据,提出了一种查询复杂度为 $O(n(\log_2 n)^{d-2})$ 的算法.后来,Bentley 在假定数据在各维上的属性值独立且无重复的情况下,提出了人们期望的线性时间的查询算法[4].

© 中国科学院软件研究所 http://www.jos.org.cn

2.2 外存处理算法

这些算法关注的都是在数据量大、无法放入内存的情况下对 Skyline 查询问题的处理.从带索引和不带索引两个方面对现有的单 Skyline 查询处理算法进行分类.

2.2.1 不带索引的算法:BNL/SFS/D&C/Bitmap/LESS

BNL(block-nested-loops)算法[1]:该算法是对待测元组建立临时表 T,T 由一个个的临时表 T_i 组成,要读取的第 1 组输入放在临时表 T_0 中.然后在主存中维持一个窗口队列,以收集相互间不存在控制关系的 Skyline 元组,窗口队列初始化为空.算法开始时,第 1 个读取的元组自然地被放进窗口队列中.然后,每当从当前临时表队列 T_i 中读入一个元组 p 时,就用该元组和窗口队列中已有的所有元组依次进行比较,可能出现以下的 3 种情况:

1) 窗口中存在元组控制元组 p:p 被删除,以后的迭代中也不再考虑 p.

2) 窗口中存在元组被元组 p 控制:从窗口队列中删除被 p 控制的元组,以后的迭代中也不再考虑这些元组,p 插入窗口队列中.

3) 元组 p 和窗口中所有元组都不存在控制关系:若窗口中有足够空间,则将 p 插入窗口队列中;若空间不够,则将 p 写入下一个临时队列 T_{i+1} 中.

每当算法扫描到 T_i 队列的末尾时,就有一些 Skyline 元组被确定了.若 T_{i+1} 队列为空,则算法终止,此时,所有窗口队列中的元组都是 SP;否则,在第 1 个写入 T_{i+1} 队列的元组产生前就已写入窗口队列中的元组是 SP,其他窗口队列中的元组再进行下一轮的处理,重复这个比较的过程,新的当前处理队列是 T_{i+1} 队列.每次迭代结束,都输出窗口队列中已和临时表中的所有元组比较过的元组.这些元组既不受控于其他元组,也不会控制还在考虑中的任何元组.其他的元组若在下一次迭代过程中没有被删除,则被输出.易知,一个元组越早被插入到窗口队列中,在下一次迭代中就越可能早地被输出.为了跟踪一个窗口队列中可能被输出的元组,算法为每个窗口队列中的元组和写入临时文件中的元组都设置了一个时间戳.这个时间戳实际上就是一个记录元组顺序的计数器,因此,若从临时文件中读入的元组的时间戳是 t,则可输出窗口队列中所有时间戳小于 t 的元组.算法利用时间戳保证了不会出现两个元组重复比较的现象和算法能够有效地及时终止的条件.

该算法的改进思想是将窗口队列变成一个能自动组织队列,替换窗口中的元组,替换策略是保证窗口队列中的元组尽可能控制更多的元组.

SFS(sort-filter-skyline)算法[9]:该算法是在 BNL 算法的基础上对数据集进行了预排序.BNL 算法的 I/O 开销依赖于迭代的次数和每次迭代要处理的数据集的大小.因为内存中窗口大小有限,不一定能装下所有的 SP,所以对要处理的元组先排序,过滤掉一些受控元组,因此,便有了 SFS 算法.SFS 算法的思想是,对临时表中的元组按照 SP 的选取规则先做一个拓扑排序,然后用 BNL 算法处理这些元组.故一旦有一个临时表 T_i 中的元组被加入窗口中,它就一定是 SP 了.因为 T_i 有序,所以它后面的元组不可能控制它,这样就不需要做替换的检查,减少了开销.

D&C(divide-and-conquer)算法[1]:该算法将数据集划分为几部分,使得每部分都可以放入内存,然后使用内存算法分别计算每一部分的 SP,最后通过合并每部分的 SP 去除其中受控的数据点来得到最终的 SP 集.该算法的具体流程是:

1) 计算输入的点在某维 d_p 上的中值(这里的中值可以指精确中值,也可以指模糊中值)m_p,并按这个中值把输入的点集划分为两部分 P_1,P_2.其中,P_1 集合包含所有在 d_p 上的属性值优于 m_p 的元组,P_2 集合包含其他剩余的元组.

2) 分别计算 P_1 的 SP 集合 S_1 和 P_2 中的 SP 集合 S_2.计算时,P_1 和 P_2 不断地迭代划分,直到每个部分只包含一个或没有元组为止.

3) 通过合并全部的集合 S_1 和 S_2,得到最终的 SP 集合.具体地说,就是删去 S_2 中被 S_1 中的元组控制的元组.

该算法最具挑战性的一步是第 3 步.如图 2 所示,该步的思想是用另一个维度 $d_g(d_g\neq d_p)$上的中值进一步划分 S_1 和 S_2 集合.结果我们得到了 4 个部分:$S_{1,1}$,$S_{1,2}$,$S_{2,1}$,$S_{2,2}$.$S_{1,i}$ 在 d_p 上优于 $S_{2,i}$,$S_{i,1}$ 在 d_g 上优于 $S_{i,2}(i=1,2)$.现在,我们只需合并 $S_{1,1}$ 和 $S_{2,1}$,$S_{1,1}$ 和 $S_{2,2}$,$S_{1,2}$ 和 $S_{2,2}$.这一步的先进性在于我们无须合并 $S_{1,2}$ 和 $S_{2,1}$,因为这两个集合中的元组是一定不会存在控制关系的.递归地调用归并函数来实现对 $S_{1,1}$ 和 $S_{2,1}$ 以及其他每两部分之间的合并,也

© 中国科学院软件研究所 http://www.jos.org.cn

就是说,$S_{1,1}$ 和 $S_{2,1}$ 又被划分了.归并函数的迭代终止条件是所有的维都被考虑过了,或者有一个划分为空或只包含 1 个元组.

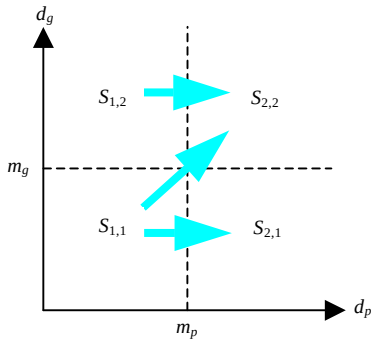

Fig.2 Basic merge

图 2 基本合并

算法的改进:

(1) 划分的界限不一定必须是中值;

(2) 将 2-路划分改进为 m-路划分;

(3) 第 3 步的最初划分中的归并函数可以用一个 Bushy-路归并.

Bitmap 算法[10]:该算法是采用位图结构来判断一个点是否是 SP,算法满足渐进性,无须遍历整个数据集,找到一个 SP 便可以及时返回,因为按位操作计算速度很快,故可以很快地判断出一个点是否是 SP.

下面是对这一算法的描述及一些定义的说明:

(1) 待测数据集为 D,包含的都是 d 维空间的点;

(2) 在第 i 维上有 k_i 个不同的属性值($1\le i\le d$);

(3) p_{ij} 代表第 i 维上的第 j 个不同的属性值,不失一般性,不妨设 $p_{i1}>p_{i2}>\ldots>p_{ik_i}$,所有维上属性值的范围都映射到[0…1];

(4) 本算法考虑取大操作为优(MAX).

算法基于的规则:

一个点 $x=(x_1,x_2,\ldots,x_d)$用一个 m 位的向量表示,其中,x_i 用 k_i 位表示,则 m 为所有 k_i 的总和,即为点 x 在所有维上不同属性值的个数之和,每位依次为 $p_{i1},p_{i2},\ldots$,如果 x_i 是 i 维上的第 p_{iq} 个不同属性值,则 x_i 的 k_i 个位上的值分别为:第 1 位到第 $q-1$ 位设置为 0,第 q 位到第 k_i 位设置为 1.这样就把每个数据点转换为了相应的点阵结构.

设 BS_{ij} 代表位片在第 i 维上的第 j 个不同的属性值.根据以下规则来判断一个点 $x=(x_1,x_2,\ldots,x_d)$是否是 SP:

让 $A=BS_{1q_1}\,\&\,BS_{2q_2}\,\&\ldots\&\,BS_{dq_d}$,则当且仅当第 n 个点在所有维上的属性值都优于(或等于)其他点 x 在相应维上的属性值时,A 的第 n 位置为 1;

让 $B=BS_{1q_1-1}\,\&\,BS_{2q_2-1}\,\&\ldots\&\,BS_{dq_d-1}$,则当且仅当第 n 个点在某些维上的属性值优于其他点 x 在相应维上的属性值时,B 的第 n 位置为 1;

让 $C=A\&B$,则易得出:当且仅当第 n 个点在每一维上的属性值都优于(或等于)其他点 x 在相应维上的属性值,且在某些维上的属性值优于其他点 x 在相应维上的属性值时(满足 SP 的定义),C 的第 n 位置为 1.

这样,无须遍历整个数据集,算法就能判断出一个点是否是 SP,从而提前返回查询结果.

LESS(linear elimination sort for skyline)算法[11]:该算法结合了 SFS,BNL 和 FLET 算法的某些方面,是 SFS 算法的改进算法.它与 SFS 算法最大的两个不同之处在于:

1) 在第 0 步时,LESS 算法加入了一个消除过滤窗口(简记为 EF 窗口)来对待测数据集先做一个外部排序,进而提前删除了一些不可能是 SP 的待测数据;

© 中国科学院软件研究所 http://www.jos.org.cn

2) LESS 算法结合了 1)中的外部排序和 SFS 算法中第 1 步的过滤 SP.

值得一提的是,在 LESS 算法中,若 EF 窗口有剩余空间,就将输入记录的副本放进去;否则,若 EF 窗口已满并且有比输入记录熵值还小的记录,就把这个记录写入窗口,替换原窗口中的记录.除此之外,保持 EF 窗口不变.这样做,可以保证更有效地严格筛选.

对于不带索引的多种算法的性能评价小结:

BNL 算法具有广泛的适用性,适用于任何维的数据,不满足渐进性,会出现误诊现象(把非 SP 在中间计算中误诊为 SP 的现象).SFS 算法需要预处理过程(排序),可有效剪枝,加速计算,输出无阻塞,流水线式输出;D&C 算法适用于数据集小的情况,数据集大时,划分过程 I/O 开销大,不满足渐进性,会出现误诊现象;Bitmap 算法,需要预处理过程(创建 bitmap),满足渐进性,响应时间短,不适于动态数据库和不同的查询条件;LESS 算法,平均时间复杂度为 $O(kn)$,最好的时间复杂度为 $O(kn)$,最坏的时间复杂度为 $O(kn^2)$.

2.2.2 带索引的算法:Using B-trees/Using an R-tree/Index/NN/BBS/BBS$^+$/SDC/SDC$^+$

Using B-trees 算法[1]:该算法的基本思想是,对于二维空间上的 Skyline 查询问题,用 B-tree 对所有元组建立一个有序的索引,然后扫描整个索引,得到按序排列的元组.这样,仅仅需要比较一个元组和它的前驱元组中最近的是 SP 的前驱元组,即可判断出该元组是否是 SP,从而先过滤出一些 SP.基于这些元组在各个维上的索引排序,同时按序扫描两个维上的元组而不用扫描全部索引,就可以得到第 1 个 SP.然后作出以下判断:

1) 按索引序在这两个维上的属性值都在已找出的 SP 之后的元组一定受控于这个 SP,因此一定不是 SP;

2) 其他的元组可能是 SP,也可能不是 SP.

不过,该算法适用范围很有限,仅在数据集小、第 1 个匹配的 SP 很快能找到的情况下效率才会高.若在一个执行了 join 或 group-by 操作的结果集上进行 Skyline 查询,该算法就不适用了.

Using an R-tree 算法[1]:该算法的基本思想是,对于任意空间上的 Skyline 查询问题,用 R-tree 对所有元组建立一个有序的索引,然后深度优先遍历 R-tree,每当遇到一个新的元组时就进行剪枝.可以采用最近邻居搜索的策略来分支定界提高算法效率.文献[1]还提出了在将来的工作中,可以探究不同的启发式规则来对 R-tree 先进行选枝,再做进一步的深度优先遍历.

该算法仅适用于 R-tree 记录了 Skyline 查询语句中的所有维度上的属性信息的情况.当然,R-tree 还可包含其他属性信息.像 Using B-trees 算法一样,仅当数据集小时,该算法具有优越性,一旦存在谓词判定,或针对高维空间下的 Skyline 查询问题,该算法性能则会变得很差.若在一个执行了 join 或 group-by 操作的结果集上进行 Skyline 查询,则该算法也不再适用.

Index 算法(a B$^+$-tree-based algorithm)[10]:该索引方案采取的是将高维数据转换为一维数据处理的方式,然后对转换后的数据建立一个 B$^+$树索引.设待处理集为$(x_1,x_2,\ldots,x_d)$,$x_{\max}$ 代表数据点在 d-维空间中的最大属性值,这个属性值所在的维度记为 $d_{\max}$,相应地,转换后的数据 $y=x_{\max}+d_{\max}$.

该算法的基本思想是,将各个维上的属性值按降序排列,基于各个维上的排序,同时按序扫描各维上的元组很容易得到第 1 个匹配的 SP.第 1 个 SP 一定在某个维度上拥有最大的属性值,所以,在找第 1 个 SP 时,只要找那些在某维上的属性值为最大的点即可.而在各维上都排在已找出的 SP 之后的点,一定受控于那个 SP,因此可以直接删除这样的点,节省 I/O 开销.

NN(nearest neighbor)算法[12]:该算法的基本思想是利用搜索到的最近邻居来递归地划分数据空间.先对待测数据集建立一个 R-tree 索引,再找到距离当前枢轴点 P 最近的邻居节点 I,然后根据点 I 进行空间划分,设现在进行的是 d-维空间上的查询,点 I 的坐标是 $(n_1,n_2,\ldots,n_d)$,则空间被划分为 $[0,n_1],[0,\infty],[0,\infty],\ldots$ 和 $[0,\infty],[0,n_2],\ldots,[0,\infty]$和…和$[0,\infty],[0,\infty],\ldots,[0,n_d]$,在找到一个 SP 后,这些划分空间放入到一个 To-do list 中,当 To-do list 非空时,就用 NN 算法从 To-do list 中拿出一个划分空间继续迭代划分.直到一个划分空间为空时,它就不再被划分,每个新找到的 SP 都会递归地调用 d 次 NN 算法.这里存在一个问题,当搜索空间维数大于二维时,会出现重复划分某些空间的问题.文献[12]提出了几种消除重复划分的方法:Laisser-faire,Propagate,Merge,Fine-grained partitioning.这里就不再一一介绍.

© 中国科学院软件研究所 http://www.jos.org.cn

BBS(branch-and-bound skyline)算法[13]:该算法也是基于最近邻居搜索策略.算法用 R-Tree 对待搜索数据集建立索引,每个上层的节点 e_i 对应下层的一个最小边界矩形(MBR),叶子节点则对应一个数据点,规定每个点的最小距离(mindist)等于它在各维上属性值的坐标之和,一个 MBR 的 mindist 等于这个矩形的左下角的点在各维上属性值的坐标之和.算法从 R-Tree 的根节点入手,将其中所有的数据点根据它们的 mindist 序插入到一个堆中.然后从有着最小的 mindist 的点入手,对它进行扩展.即从堆中拿出这个点,把它的孩子节点插入到堆中(堆中仍然按 mindist 有序),以后仍然是选出堆中拥有最小的 mindist 的节点进行扩展.迭代就这样进行下去.当堆中被拿出的节点是叶子节点时,它就是找到的 SP.在扩展节点时,要注意堆中相邻的两个节点是最近邻居关系,但即使是这样,前一个节点的 mindist 比后一个节点的 mindist 要小,也不能说明后一个节点被前一个节点控制而对后一个节点剪枝不再扩展,因为若后一个节点是一个非叶子节点,它的孩子节点中就有可能含有 SP,只有被已找出的 SP 控制的孩子节点才能被剪枝不再继续扩展.算法终止的条件是堆为空.

BBS$^+$算法[14]:该算法是对 BBS 算法上的直接改进算法,改进算法与原算法的差别在于:1) BBS$^+$算法中的 R-Tree 索引是建立在部分有序的属性值的转换值域*之上的索引,因此,BBS 算法中控制关系的比较在 BBS$^+$算法中变成了 m-控制关系的比较;2) 因为 BBS$^+$算法计算的中间查询结果集可能会出现误诊现象,所以,BBS$^+$算法中的 *UpdateSkylines*()函数需要检测和去除误诊结果(即要比较每一个新找到的 SP 和已找出的中间 SP 集).

SDC(stratification by dominance classification)算法[14]:文献[14]针对属性域部分有序的 Skyline 查询问题提出了一个基于控制关系分类的分层算法.该算法也是采用基于属性值的转换值域上的一个分层策略.为避免对不必要的控制关系的检查,通过探索域变换的特点,该算法把运行的中间结果集分成了两层:一层一定是 Skyline 查询结果的点集;另一层是可能被误诊为 Skyline 查询结果的点集.具体做法是,对每一个由控制变换 f_i 得到的部分有序的属性 A_i,由它的部分有序 DAG $G_i=(D_i,E_i)$得到一棵生成树 ST_i,这样,每一个 D_i 中的点都有它的入度值和出度值.对于每个 D_i 中的点 v,若它在 G_i 中的直接入径也在 ST_i 中,则称它是完全被覆盖的;否则,称 v 部分被覆盖.类似地,每个 D_i 中的点 v,若它在 G_i 中的直接出径也在 ST_i 中,则称它是完全覆盖的;否则,称 v 为部分覆盖.在算法中找到的新的 SP,若满足这里的完全覆盖关系,则属于第 1 层结果集,直接可以作为最终 SP 结果返回;若满足部分覆盖关系,则属于第 2 层结果集,还需要进一步比较以去除可能存在的误诊 SP.然后,算法根据分类的控制关系,通过 3 个模块——① 缩小控制比较,② 优化控制比较,③ 有效计算,把计算的 Skyline 中间结果集分为 4 个子集,避免不必要的中间结果的比较和检查,以得到最终的 SP 集.

SDC$^+$算法[14]:这种算法比 SDC 算法表现出的先进性在于:在 SDC 算法中,Skyline 的中间结果集被分为两层——一层是完全覆盖关系的中间 SP 集,另一层是部分覆盖关系的中间 SP 集,从而保证了前一个集合中出现的 Skyline 中间查询结果一定是最终的 SP,而后一个则是可能被误诊的 SP.这样虽然避免了 BBS$^+$算法中对每一个新找到的 SP 都需和已找到的 SP 集进行比较的开销,但是若找到的 SP 是第 2 层的中间 SP,则仍需大量的比较和检测.而实验证明,前一层的中间结果点集只占很少的一部分,因此,SDC$^+$算法提出的改进方案是将数据划分为更多层的策略.这种算法中,数据被划分为了两个以上的层,第 i 层的点都一定不能控制第 i–1 层的点.这样,找到第 i–1 层中的 SP 集后,无须检查第 i 层中的点的情况,这些查询结果就可以被返回了.算法的先进性表现在保证了只需层内比较,层间不会出现 SP 的误诊情况.

对于待索引的多种算法的性能评价小结:

Using B-trees 算法适用于二维空间,但不具有普遍的适用性,需要一个预处理过程(为查询建立索引);Using an R-tree 算法适用于高于二维的空间,但同样也不具有普遍的适用性,需要预处理过程(为查询建立索引);Index 算法需预处理过程(建索引),响应时间快,满足渐进性,但不适于不同查询条件和不同维的子空间查询;NN 算法

---

* 用一个域转换函数来完成属性值转换工作:将部分有序的属性进行转换.为了避免维数诅咒现象(即 Skyline 查询结果的数量随着维数的增加呈指数增长),文献并没有采用常用的从部分域到全域的变化方法,即将部分有序的属性值转换为布尔值来表示其之间的控制关系,而是构造一个同构变换 f:对域 D_i 上每个部分有序的属性值 A_i,构造一个一对一的域变化 f_i,它把每个 $v\in D_i$ 变换成一个间隔 $f_i(v)\in N\times N$,其中,N 代表自然数.这个域变化函数 f_i 是这样定义的:对于任意两个不同的 $v,v'\in D_i$,若 $f_i(v)$包含 $f_i(v')$,则 v 控制 v'.

© 中国科学院软件研究所 http://www.jos.org.cn

需预处理过程(建索引),适合不同的查询条件和动态数据库,满足渐进性,但算法性能低,对于 3 维以上的数据,算法的空间开销大;BBS 算法仅适用于属性域全有序的查询处理,I/O 性能高,适合动态数据库,不同的查询条件和不同维的子空间查询,满足渐进性;BBS+算法适用于属性域部分有序的查询处理,是 BBS 的改进算法,但会出现误诊现象,不满足渐进性;SDC/SDC+算法适用于属性域部分有序的查询处理,响应时间短,渐进性好,SDC^+是 SDC 的改进算法.

3 多 Skyline 查询处理算法

第 2 节介绍的算法都假定所有的 Skyline 对象都处在某一个特定的 d-维空间中,Skyline 查询所涉及的属性集组合不发生变化,即所有用户所关心的度量指标相同.但现实生活中,不同的用户可能有不同的兴趣和偏好,需要在不同的子空间中处理 Skyline 查询.比如,某房地产公司有多处房产等待出售,每一处房产都有价格(price)、与市中心的距离(dist)、房龄、拥有的房间个数等属性.有的顾客可能对前面两个属性比较感兴趣,只在 price 和 dist 这两个属性上作 Skyline 查询,而有的顾客可能对价格和房龄比较感兴趣,希望在这两个维上作 Skyline 查询.这样,对一个 d-维的数据集来说,共有 2^d 种不同类型的 Skyline 查询,它们的查询结果是不一样的.最近,数据仓库和 OLAP 领域的研究者对在不同子空间上进行 Skyline 查询的研究产生了浓厚的兴趣.Yuan 等人[5]提出了 SKYCUBE 的概念,利用 Data Cube 的多维层次结构,提出了有效的、能够同时计算多个 Skyline 查询的算法.为了减少查询的响应时间,文献提出了在这些相关的 Skyline 查询间采用一种信息共享的策略,基本的思想是预先计算出一个给定数据集的所有可能的 Skyline 查询.尽管 SKYCUBE 在概念上沿袭了传统的 Data Cube 的概念,但是,计算 SKYCUBE 比计算 Data Cube 更具有挑战性,因为:(1) SKYCUBE 的子空间不能完全由父空间推算出来,例如,存在这样的情况:在父空间中不是 SP 的点却有可能是子空间的 SP;(2) 计算 Skyline 比计算 Data Cube 时常用到的划分或排序算法的代价要大.

3.1 SKYCUBE

3.1.1 SKYCUBE 的结构

所谓 SKYCUBE,就是一个给定数据集的所有非空的子空间的 Skyline 查询结果的全体.其中的每个子空间的 SP 集合被称为一个 Cuboid.

3.1.2 SKYCUBE 的计算:BUS,TDS

为了减少查询的响应时间,文献[5]提出了利用 SKYCUBE 的层次结构,在相关的 Skyline 查询间采用信息共享的策略,并且基于这种策略,提出了同时计算多个相关的 Skyline 查询的算法——Bottom-Up 算法和 Top-Down 算法,弥补了现存算法中各个查询间信息独立、只能独立地处理单个 Skyline 查询问题的不足.下面来介绍这两种算法的基本思想.

BUS 计算(bottom-up skycube computation)算法[5]:算法的基本思想是按自底向上的顺序逐层计算出 SKYCUBE 中的每个 Cuboid.对每个 Cuboid 的计算,采用的算法是类似于 SFS 的一个嵌套循环算法.鉴于 naïve 算法的效率低就在于:(1) 独立地计算每个立方体;(2) 计算整个 SKYCUBE 需要 2^d-1 次的排序.为了解决这两个问题,算法采用的应对策略分别是共享结果和共享排序.其中,共享结果基于的理论是在数据各维上的属性值都不相同的条件下,每个子 Cuboid 中的 SP 一定是它的父 Cuboid 的 SP,因此,算法用所有子 Cuboid 的并集作为计算父 Cuboid 的初始集合,这个求并运算,用位操作在线性时间内就可以完成;而共享排序的策略是指在计算一个 Cuboid 时,总是让数据在这个 Cuboid 中所有维的属性值域最大的那个维上保持输入有序,由嵌套循环算法的本质,这样做并不会影响计算结果的正确性,但却可以将计算 SKYCUBE 需要排序的次数从 2^d-1 减少到 d,而且有效地防止了误诊现象的出现.另外,还可以在排序前通过一个过滤的过程来进一步减少比较点间控制关系的开销.基于这两种共享策略,BUS 算法按照自底向上的顺序一层一层地计算 SKYCUBE 中的每个 Cuboid.在计算每个 Cuboid 时,算法按序检查每个数据点和已找到的 SP 的关系,若这个数据点已经是其子 Cuboid 的一个 SP,则将它直接插入父 Cuboid 中;否则通过调用一个检查函数来比较它与现有 SP 的控制关系,判断它是不是一个新的 SP.最后要注意的是,在数据各维上的属性值可能出现相同值的情况下,算法采用的方法是将这些有相同属

© 中国科学院软件研究所 http://www.jos.org.cn

性值的数据点放入到一个缓冲区中,先计算出它们的 SP 作为候选集,再拿这些候选集和已找到的 SP 作比较计算.

TDS 计算(top-down skycube computation)算法[5]:算法运用的共享策略是共享划分、合并以及共享父结果.算法首先找到能够覆盖所有 Cuboid 的最少路径的集合,这里的路径指的是 Cuboid 中不同维度的组合,然后用 SDC 算法来计算每条路径上的 Cuboid.为了利用共享父亲策略的优势,算法总是从未计算的路径中选择组合维数最多的路径进行计算.但实际上,由于父 Cuboid 和子 Cuboid 并不是包含关系,所以,采用共享父结果策略可能造成一些子 Cuboid 集中 SP 的丢失,而这些丢失的数据点一定是与父 Cuboid 中的 SP 在某些维上有相同的属性值.因此,为了找出这些丢失的数据点,对于父 Cuboid 中的每个 SP,我们都收集和它在这个父 Cuboid 的维度组合上有着相同属性值的数据点,为了加速这一收集过程,对整个数据集在每维上的属性值都进行了一个预排序.

3.1.3 特定子空间上的 Skyline 查询:SUBSKY 算法

考虑到对于一个给定的多维数据点的集合,并不是每个用户都需要所有子空间上的 Skyline 查询结果,实际上,用户往往只是根据自己的偏好对一个特定的子空间上的查询感兴趣,而现存的算法大都需要至少扫描整个数据集一遍,必然会带来在其他子空间上计算的额外开销.针对这一问题,Pei 等人在文献[15]中提出了一种针对特定子空间下的 Skyline 查询的算法——SUBSKY 算法.算法采用维度压缩的方案,将多维的数据集合转换为一维的值,然后用一棵 B-Tree 对转换后的数据建立索引,通过搜索 B-Tree 来进行子空间的 Skyline 的查询计算.但是该算法的不足之处在于,它对维度敏感且要全部的结果计算完毕方才产生第 1 个输出.由于算法只用到了关系技术,所以它可以适用于任何商业数据库系统.

SUBSKY 算法[15]的基本思想是:假定一个 d-维空间,这个空间上的每个维度的值域都为[0,1].定义空间的最大角对应的点为锚点,它的坐标在各维上都是 1,每个数据点 p 的坐标值可以转换为一个一维的值 $f(p)$,记 $f(p)=\max(1-p[i])$(其中,$p[i]$为点 p 在第 i 维上的属性值).这样,对于一个数据点 p,它的 $f(p)$若小于子空间上的每一个 SP 的 $1-p_{sky}[i]$,即 $f(p)<\mathrm{Min}(1-p_{sky}[i])$,则它一定不是子空间中的 SP.由这一思想,就得出了一种计算子空间的 SP 的快速算法——SUBSKY 算法.算法首先按照数据集中数据点的 $f(p)$值降序排序,然后同时维护几个队列:

1) 目前找到的 SP 集合 S_{sky};

2) U 的值,定义 U 为最大的 $\mathrm{Min}(1-p_{sky}[i])$,其中,$p_{sky}\in S_{sky}$.当 U 大于下一个点 p 的 $f(p)$值时,算法终止.

改进思想:为了让算法具有更普遍的适用性,可以为每一簇点集合分别选择一个锚点:首先对点分簇,即将点投射到一条斜率为–1 的直线上,投影在一起的点为一簇,然后定义分簇的点的锚点为这簇点右上角那个点的坐标值.这样,前面基本思想中用到的锚点坐标(1,1,…,1)也相应地变为该簇点的锚点的坐标.其他比较和剪枝过程都依照上面的基本思想进行.

3.2 压缩的SKYCUBE

SKYCUBE 中存放了预计算出来的所有非空子空间的 Skyline 查询结果,在一定程度上提高了 Skyline 的查询效率.但当数据发生变化时,需要对这些查询结果进行维护,维护代价十分昂贵.而且,SKYCUBE 中还存在着大量冗余数据,如果能将这些冗余数据去掉,那么既可以减少存储空间,也可以减少维护的代价.为此,文献[16]对 SKYCUBE 进行了压缩,提出了一个新的存储模式 CSC(compressed skycube),同时提出了对 CSC 的计算、查询和维护算法.

另外,Pei 等人在文献[17,18]中结合 SP 的语义提出的子空间上的 Skyline 分析,引入了 Skyline 组和决定子空间的概念,并且设计了另一种能够捕捉子空间语义的多 Skyline 计算方法:利用 Skyline 组和它们的决定子空间为多维子空间的 Skyline 查询提供信息,在文献[19]中提出了另一种基于语义压缩的 SKYCUBE,文献[19]还提出了计算这种压缩的 SKYCUBE 的算法——Stellar 算法.

4 不同应用环境下的 Skyline 查询处理算法

Skyline 查询返回一组有意义的对象,这些对象在各维上都不被其他对象所控制,从而支持用户在复杂的情况下进行决策,这使得它在许多领域都有着广泛的应用,如多标准决策支持系统以及用户偏好查询等.为了更好

© 中国科学院软件研究所 http://www.jos.org.cn

地适应在不同环境下的应用,最近两年,对 Skyline 问题的研究逐渐地趋向于在具体应用环境下进行.下面分别介绍在几个不同的应用环境下 Skyline 问题的查询处理策略.这些应用环境包括 Web 信息系统、分布式 P2P 网络、数据流和公路网络.

4.1 Web信息系统

由于 Web 资源分布独立,之前介绍的所有集中式的 Skyline 查询处理算法不能用于处理 Web 信息系统中的 Skyline 查询问题.为了处理分布式环境下的 Skyline 查询,文献[20]首次提出了一种有效的基于 Web 上的分布式 Skyline 算法(a Web based distributed Skyline algorithm).该算法的整个流程可分为 3 个阶段:

第 1 阶段:按每个记录在各维上的属性值降序分别建立有序队列 $S_i(1\le i\le n)$,然后依次从上述的 n 个有序队列 $S_1,\ldots,S_n$ 中检测待测对象 O_{new},建立一个合并的中间队列 P.若 O_{new} 不在 P 中,则将它加入 P,并在 P 中记下它在相应的 S_i 上的属性值;若 P 中已有该对象,就将这个对象新检测到的属性值添加在它的记录中相应的 S_i 下,直到有一个对象 O_{known} 在所有维上的属性值都已知为止.

第 2 阶段:对于 S_i 中 O_{known} 之后的对象,若是与它在该维上属性值相同,将这些对象记录和其属性值也添入队列 P 中;若是比它在该维上的属性值要小,则这部分的检测终止.对于队列 P,按照各对象记录在各维 S_i 上属性值的已知与否建相应 n 个队列 K_i,每个 K_i 队列仍按属性值降序排列.这是一个按序存取(sorted access)的过程.

第 3 阶段:检测每个 K_i 队列,若该队列中存在拥有多于一个已知属性值的对象,则比较它与该 K_i 队列的其他对象,在 P 中去除那些受控的对象;对于 K_i 队列中只知道一个属性值的对象,随机访问它在其他维度上的属性值,得出它和该 K_i 队列中其他对象的控制关系,在 K_i 和 P 中去除受控的对象.这是一个随机存取(random access)的过程.最后,返回结果集 P.

文献[20]还利用两条启发式规则提出了改进算法,改进算法及时地终止了每阶段的检测工作,从而提高了效率.并且,为了避免出现维度诅咒的问题,文献[20]提出了一种新型的抽象模式,以便尽快得到大致的 SP 集合,然后再细化改良.

4.2 P2P网络

P2P 网络中的数据分布存储在不同的节点上,每个节点既是客户机,又是服务器,没有一个统一的中央控制节点.由于网络规模巨大,每个节点只有有限的资源,故为 P2P 系统中的 Skyline 查询建立和维护一个专门的索引是没有价值和不现实的,故现存的索引算法并不适合于一个高度动态的 P2P 系统.为此,文献[21]研究了 P2P 网络环境下的 Skyline 查询处理问题,提出了 P2P 网络上的 Skyline 算法(a P2P based Skyline algorithm).算法利用语义网的本质特征,在不影响查询结果精确性的前提下,有效地限制了执行查询包括的网络范围.该算法可分为 4 个阶段:

第 1 阶段,定位源簇:根据 SSW(semantic small world)的簇命名规则可以很容易地确定最优值所在的簇,并把最优值所在簇定义为起始簇(C_0),其簇 ID 为 0.C_0 中节点不会被其他簇中的节点控制.因此,在计算出 C_0 中的 SP 之后,可直接将结果由起始节点返回给用户.然后,我们记最接近最优值的那个 SP 值为界限值 v_{bound},用于以后步骤中的剪枝.

第 2 阶段,Inter-Cluster 剪枝:利用一个节点的联系表来实现选择 Skyline 查询的顺序和对搜索空间的剪枝.根据一个给定簇的语义范围,选定代表该簇的对象,它的属性值在所有维上均取最优值.如果这个最优对象被 v_{bound} 控制,就称该簇被 v_{bound} 控制.从 C_0 中的任意一个节点开始,对于该节点的相邻簇,如果这些簇不被 v_{bound} 所控制,那么向这些相邻簇附送 Skyline 查询,并标记它们以避免重复访问.定义那些从其他簇接收到 Skyline 查询的节点为所在簇的代表节点 P_{rep},P_{rep} 从它的联系表中选择要附送 Skyline 查询的簇.当节点的所有相邻簇中找不到没有被访问且不被 v_{bound} 所控制的簇时,这个过程终止.

第 3 阶段,Intra-Cluster 剪枝:对于上面步骤所得到的簇,我们继续使用 v_{bound} 对其中的节点进行筛选.根据节点的语义信息,采用计算簇的语义范围的方式来计算节点的语义范围.根据这些信息范围,如果一个节点被 v_{bound} 所控制,那么这些节点就被剪枝掉.簇的代表节点 P_{rep} 用来存储 Intra-Cluster 路径以及接受其他节点附送来

© 中国科学院软件研究所 http://www.jos.org.cn

的 Skyline 查询.

第 4 阶段,计算 SP:对于前面步骤最后得到的节点,从其本地数据中得到 SP,并返回给所在簇的 P_{rep}.由 P_{rep} 使用一个内存 Skyline 算法来合并它们,合并后的簇 SP 将被传送给 p_0,p_0 对不断接收到的簇 SP 用一个内存 Skyline 算法进行合并,同时,p_0 记录了传输路径.一旦所有传输路径都被剪枝了,p_0 就停止等待接收 SP 并将结果返回给用户.

另外,对于更广泛的 P2P 网络,尽管节点可以分簇,但是并不使用语义信息来划分和管理簇,因此,上面提到的算法不能适用于所有 P2P 网络.文献[21]采用了一个强制推动计算进行的手段,提出了一个近似的 P2P 网络 Skyline 查询算法——Single-Path Approximate 算法.在有超级节点的 P2P 网络中,高网速和高性能的计算机被自动设置为超级节点.超级节点作为其他用户的索引服务器,每个超级节点连接一些数量有限的节点.文献[22]为有超级节点的 P2P 网络中的子空间的 Skyline 查询提出了一种基于阈值的算法——SKYPEER.为了有效地处理子空间上的 Skyline 查询,文献[22]还通过定义扩展的 Skyline 集(ext-skyline)来扩展控制关系的概念.在扩展的定义条件下,一个点仅当它在各维上的属性值都比另一个点在相应维上的属性值小时,它才被另外的那个点所控制.SKYPEER 算法推动了节点间的 Skyline 查询的传递,极大地减少了需要转移的数据量;另外,通过阈值控制计算范围,减少了通信代价和执行时间.

4.3 数据流

流数据具有数据量大且速度快、无限连续到达、随时间具有不可预测性和多变性等特点,从而给在数据流环境下进行 Skyline 查询处理带来了严峻的挑战.它要求 Skyline 查询处理具有实时性和强的管理性,便于快速捕获流数据的特征.数据流应用一般运行在存储容量有限(相对于数据流的大小)的环境下,因此,为了减少 Skyline 查询结果集,改善其可管理性,文献[23]考虑对数据流中最近的 N 个元素计算其 Skyline 结果集——n-of-N Skyline 查询,即在一个数据流的最近 N 个元素中,查询任意最近 n 个元素($n\le N$)的 SP 集合,以支持快速的数据流上的 Skyline 在线查询.文献[23]处理的流数据是一种基于滑动窗口模式,存在插入、删除和新到来数据等情况的待测数据集.针对这种数据流上的 Skyline 查询处理,提出了 Stabbing The Sky 算法.该算法主要考虑数据插入和删除时如何对全空间上的 Skyline 进行维护.算法的缺陷在于,没有考虑如何计算子空间上的 Skyline,而且算法的维护代价较高.另外,文献[23]还对这种算法作了拓展,给出了计算(n_1,n_2)-of-N Skyline 查询问题的算法.(n_1,n_2)-of-N Skyline 查询问题,即在一个数据流的最近 N 个元素中,查询任意最近 n_1 个元素到最近 n_2 个元素($n_1\le n_2\le N$)中的 SP.

4.4 公路网络

前面所述的有关 Skyline 查询处理的研究主要是针对一个查询参考点的单源 Skyline 查询问题,包括针对一个固定参考点的绝对 Skyline 查询问题和针对一个用户指定参考点的相对 Skyline 查询问题.文献[24]首次针对公路网中多源的 Skyline 查询处理问题提出了解决方法.这是一个公路网中两点的网络距离需要联机计算的相对 Skyline 查询问题.

不失一般性,文献[24]提出的算法中仍然采用取小操作为优.Skyline 查询可以是绝对的,也可以是相对的.其中,绝对的 Skyline 查询是基于数据集中数据点静态属性值上的最小值计算;而相对的 Skyline 查询是指给定数据集中数据点和一个用户给定的查询点间的差异最小,也称为动态 Skyline 查询.在公路网中,基于不同的应用环境,两点之间的距离可以用 3 种方式表示:欧氏空间距离、网络空间距离以及表面物理距离.文献[24]提出了 3 种采用不同策略来减少计算网络距离开销的多源公路网 Skyline 查询处理算法:CE(the collaborative expansion)算法,EDC(the euclidean distance constraint)算法,LBC(the lower bound constraint)算法.

5 Skyline 查询处理问题的扩展

5.1 高维空间下的Skyline查询处理问题

Skyline 查询在高维环境下面临一个巨大挑战,那就是维数诅咒现象,即 Skyline 查询结果的数量随着维数

© 中国科学院软件研究所 http://www.jos.org.cn

的增加呈指数增长.当维数很大时,得到的 Skyline 查询结果集巨大,用户还需要在这巨大的结果集中做选择,这样的查询结果就不再有意义了.

为了消除高维空间下产生的维数诅咒现象,文献[25]提出了一个新的度量尺度——Skyline 频率,即 SP 在不同子空间的 Skyline 查询中作为查询结果返回的次数,以此来决定不同 SP 的价值.给定一个 n-维空间的对象集合.一个对象的 Skyline 频率由 2^n-1 个不同的 Skyline 查询决定,每个查询是对一个可能的非空属性子集上的查询.易知,一个数据点的 Skyline 越高频率越有价值,因为它在越少的维度组合上受控.这样,问题转化成了 Top-k 的 Skyline 频率查询问题,并且,文献[25]为其提供了一种近似算法,有效地避免了以往的查询算法在不同维度的空间上伸缩性差和需要在指数个非空子空间中都进行 Skyline 查询的问题.下面来介绍这种计算 Top-k 的 Skyline 频率的近似算法.

Top-k Frequent Skyline Computation[25]:算法基于的思想是,一个点的每个控制子空间被这个点的最大控制子空间集所覆盖.如果一个子空间中存在其他点能控制点 p,就称这个子空间为点 p 的控制子空间.记点 p 的控制子空间的数目为其控制频率 $d(p)$,则其 Skyline 频率为 $2^n-d(p)-1$.于是,Top-k 的 Skyline 频率计算问题就转化为查询 Bottom-k 的控制频率的点的问题.

首先,初始化最大的可能控制频率值μ为 $2^{|S|}-1$,Top-k 频率的 SP 集合 R 为空集,然后对于每个数据集 D 中的点 p,调用函数 *ComputeMaxSubspaceSet*()计算点 p 的所有最大控制子空间的集合 M,该函数采用的计算方式是比较点 p 和其他点的控制关系的方法,然后调用函数 *CountDominatingSubspaces*()计算点 p 的控制频率 $d(p)$,若 $|R|<k$ 或者 $d(p)<\mu$(当$|R|=k$ 时去除 R 中有着最高控制频率的点),则将点 p 插入到 R 集合中,更新μ为 R 中的最高控制频率,循环结束.最后,返回 R 集合.

算法核心的两个步骤:

第 1 步:调用函数 *ComputeMaxSubspaceSet*()来计算每个数据点的最大控制子空间集.记 $DS(q,p)$为点 q 控制点 p 的子空间集,用一个子空间对(U,V)表示,其中,在 U 中的每维上,q 的属性值都小于 p 的属性值;在 V 中的每维上,q 的属性值都等于 p 的属性值.这样,算法计算每个新来的数据点的最大控制子空间集 M,M 由一系列子空间对的形式表示:$M=\{(U_1,V_1),(U_2,V_2),\ldots,(U_n,V_n)\}$,其中,$(U_i,V_i)$对应于每个数据集中的其他点 q_i 的 $DS(q_i,p)$.算法采用比较点间不同维度上的属性值的方式来计算 $DS(q,p)$,用位图来记录表示控制子空间.值得一提的是,这一步算法中采用的优化策略是只精确计算可能具有 top-k 的 Skyline 频率的 SP 的最大控制子空间集以减少精确计算,提高算法效率.

第 2 步:调用函数 *CountDominatingSubspaces*()来计算每个点的控制频率.一个点的最大控制子空间集 M 所覆盖的子空间数即为一个点的控制频率,算法提出基于精确和模糊计数的两种方法来统计 M 覆盖的控制子空间数.

另外,为了实现在高维空间上找到更重要和更有意义的点,文献[7]提出了一个新的概念,k-dominant(k-控制).所谓点 p k-控制点 q 是指存在 k 维,在这个 k 维的子空间上点 p 等于或优于点 q(其中,$k\le d$,全空间为 d 维),并且至少在这 k 维中的一维上点 p 是优于点 q 的,因此,k-控制的 SP 集合就是指不被任何其他点 k 控制的点集合.随之产生的问题是,k-控制查询得到的 SP 之间是没有传递性的,所以,现存的 Skyline 算法不适用于 k-控制 Skyline 查询.文献[7]针对这个问题提出了 3 种解决算法.

One-Scan 算法[7]:首先,将数据集 D 中数据点按照其各维上的属性值之和非递增排序,并初始化 k-控制 SP 集合 R 为空集,非 k-控制但满足完全控制的 SP 集合 T 为空集.对于每个 D 中的点 p,首先将它和 T 中的点进行比较:若 T 中存在点 p'被点 p 控制,就从 T 中删除 p';否则,若点 p'控制点 p,或者 $p'=p$,就不考虑 p 了.但是,若点 p 是一个不重复的 SP,就拿 p 继续和 R 中其他的点比较以检查它是不是 k-控制的 SP.对于 R 中任意的点 p',若点 p k-控制点 p',则把点 p'从 R 移到 T 中,因为这样的点不会是 k-控制的 SP.最后比较点 p 和 R 中的点,若 p 是 k-控制的,就将点 p 插入到 R 中;否则,点 p 就是唯一的 SP,因此把它插入 T 中.当所有 D 中的点都被检查后,R 就变成了 D 中所有 k-控制 SP 的集合.

Two-Scan 算法[7]:在此算法中需要维持完全控制的 SP 集合来计算 k-控制 SP.而完全控制的 SP 集合往往比

© 中国科学院软件研究所 http://www.jos.org.cn

k-控制 SP 集合要大得多,所以,维持 *T* 集合需要很大的空间消耗和计算代价.Two-Scan 算法采用扫描 *D* 集合两遍的策略来避免这个代价的发生.在第 2 遍扫描中,判断 *R* 中的一个点 p'是否是 *k*-控制 SP,只需比较 p'和 *D*-*R* 集合中比它早到的点就可以了,对于那些 *D* 中出现的比点 p'晚到的点,在第 1 次扫描中它们已经与 p'比较过了.

Sorted Retrieval 算法[7]:与前两种算法顺序扫描数据集 *D* 来计算 SP 的方法不同,该算法引进了一个启发式函数 *FindNextDimensions*()来选择需要处理的下一维 S_i,并对其进行检测处理.

5.2 Skyline查询和控制关系分析的结合

在上面的研究工作的基础上,文献[6]首次从微观经济学的角度扩展了空间数据库中不同数据点之间的控制关系的概念,将其用于经济学框架下的商业分析,提出了控制关系分析(dominant relationship analysis,简称 DRA)的概念.控制关系分析在实际中,尤其是市场分析中具有非常重要的作用,比如顾客在购买商品之前,可以通过控制关系的分析来比较不同厂家的产品,从而选出最优的商品或最优的候选商品集.厂家可以通过控制关系分析,推断自己的产品在市场中的定位,从而调整营销策略等等.该文将多维数据组织成一个传统的数据方体 DADA(data cube for dominant relationship analysis),并用所能控制的点数作为度量,借用传统的数据方体预计算思想,将数据分析所要用到的信息预先存储下来.该模型不仅可以处理传统的 Skyline 查询,而且还可以处理许多其他查询,如线性优化查询(LOQ)、子空间查询(SAQ)、比较控制查询(CDQ)等.

5.3 Skyline查询和Top-*k*查询的结合

文献[26]研究的问题是,如何从点集中选出 *k* 个 SP,使得至少被这 *k* 个点中的一个点所控制的点的数量最大,这些点被认为是 SP 中最具代表性的 SP.针对这一问题,在二维空间中基于现有的算法,文献[26]给出了一种动态算法;而在等于或大于三维的空间中,这个问题被证明是一个 NP 难问题,解决这个问题所需的多项式时间近似系数是 1–1/e,该文采用 FM 概率计算方法建立了一种高效率、可伸缩、基于索引的随机算法,加快了计算速度.

6 总结和展望

信息与通信技术的迅速发展使得信息收集手段变得更加丰富.通常,这些被收集的数据对象可以看成是高维空间的向量或者点,而每一维可视作数据的一个属性.随着信息爆炸时代的到来,越来越多的数据存放在数据库中,使得利用这些信息并从中高效获取知识成为一种必然需要.为此,不同领域的学者从各自的方向和角度进行了有益的探索.Skyline 查询处理作为一种新的数据库操作符(skyline operator),受到了越来越多的重视,被广泛应用于各种决策支持系统中.

目前,绝大多数的研究工作围绕 4 个方面展开:单 Skyline 查询处理方法、多 Skyline 查询处理方法、不同应用环境下的 Skyline 查询处理和针对 Skyline 查询处理问题的扩展.其中,关于单 Skyline 查询处理方法,呈现出了大量的研究成果.从内存到外存,从无索引到有索引.多 Skyline 查询处理方法主要包含 SKYCUBE 方法和 CSC 方法.后者是在前者基础上的一个压缩结构.不同应用环境下的 Skyline 查询处理介绍了在 Web 信息系统、P2P 系统、数据流环境和公路网络环境下的 Skyline 查询处理问题.针对 Skyline 查询处理问题的扩展,主要介绍了在高维空间下存在高维诅咒现象的情况下如何对 Skyline 查询进行处理.另外,还介绍了 Skyline 查询和控制关系分析、Top-*K* 查询相结合的问题.

通过对这些已有研究的分析和总结,我们归结出 Skyline 查询处理问题的后续研究的几个方向:

(1) 扩展 Skyline 查询处理的应用环境,如传感器网络环境下的 Skyline 查询处理问题有待进一步研究; 无线传感器网络由基站和大量的无线传感器节点构成,其上的一种典型查询是多属性查询,而 Skyline 正是用于解决多属性决策问题的有效方法.但是,由于无线传感器网络中的节点拥有的能量有限,为 Skyline 算法的应用带来了极大的困难.因此,研究传感器网络中的 Skyline 计算问题是一个非常有趣而又具有挑战性的课题.

(2) 扩大 Skyline 查询处理的结合范围,如可以将 Skyline 查询和时态查询、范围查询、*k*NN 查询等相结合;如何将在某个时间窗口上进行的时态聚集查询和 Skyline 查询相结合,尤其是如何将时态聚集计算和 Skyline 计算同时进行是一个非常具有挑战性的问题;其次,范围查询也可以与 Skyline 查询相结合,范围查询的问题也

© 中国科学院软件研究所 http://www.jos.org.cn

可以转换成空间搜索的问题.另外,控制关系分析(dominant relationship)目前也已成为数据库领域研究人员关注的一个热点问题.考虑到市场中存在竞争对手且资源受限的情况,可以考虑将 Skyline 查询与基于 *k*-最近邻居(*k*-nearest neighbor,简称 *k*NN)的控制关系相结合.

(3) 近似 Skyline 查询处理,为了提高查询处理的速度,可以将控制关系的定义在某些方面进行弱化,从而在损失少量精度的情况下更快地得到查询结果.

References:

[1] Borzsonyi S, Kossmann D, Stocker K. The skyline operator. In: Proc. of the 17th Int'l Conf. on Data Engineering. Heidelberg, IEEE Computer Society Press, 2001. 421–430. http://www.dbis.ethz.ch/research/publications/38.pdf

[2] Balke WT, Güntzer U. Multi-Objective query processing for database systems. In: Proc. of the Internet Conf. on Very Large Data Bases (VLDB 2004). 2004. 936– 947. http://www.vldb.org/conf/2004/RS24P1.PDF

[3] Kung HT, Luccio F, Preparata FP. On finding the maxima of a set of vectors. Journal of the ACM, 1975,22(4):469–476. http://portal.acm.org/citation.cfm?id=321910&dl=ACM&coll=portal

[4] Bentley JL, Kung HT, Schkolnick M, Thompson CD. On the average number of maxima in a set of vectors and applications. Journal of the ACM, 1978,25(4):536–543. http://portal.acm.org/citation.cfm?id=322092.322095

[5] Yuan Y, Lin X, Liu Q, Wang W, Yu JX, Zhang Q. Efficient computation of the skyline cube. In: Proc. of the 31st Int'l Conf. on Very Large Databases. ACM, 2005. 241–252. http://www.vldb2005.org/program/paper/tue/p241-yuan.pdf

[6] Li CP, Ooi BC, Tung AKH, Wang S. DADA: A data cube for dominant relationship analysis. In: Proc. of the ACM SIGMOD Int'l Conf. on Management of Data (SIGMOD 2006). 2006. http://www.comp.nus.edu.sg/~ooibc/dadacube.pdf

[7] Chan CY, Jagadish HV, Tan KL, Tung AKH, Zhang ZJ. Finding *k*-dominant skylines in high dimensional space. In: Proc. of the ACM SIGMOD Int'l Conf. on Management of Data (SIGMOD 2006). Chicago, 2006. 503–514. http://www.eecs.umich.edu/db/files/k_dominant.pdf

[8] Freitas AA. A critical review of multi-objective optimization in data mining. A Position Paper: ACM SIGKDD Explorations, 2004, 6(2):77–86. http://portal.acm.org/citation.cfm?id=1046456.1046467&coll=GUIDE&dl=GUIDE

[9] Chomicki J, Godfrey P, Gryz J, Liang D. Skyline with presorting. In: Proc. of the IEEE Int'l Conf. on Data Engineering. Los Alamitos: IEEE Computer Society Press, 2003. http://www.cs.sfu.ca/CC/843/jpei/Skyline/Chomicki-Presorting-ICDE03.pdf

[10] Tan KL, Eng PK, Ooi BC. Efficient progressive skyline computation. In: Proc. of the 27th Int'l Conf. on Very Large Data Bases. 2001. 301–310. http://www.vldb.org/conf/2001/P301.pdf

[11] Godfrey P, Shipley R, Gryz J. Maximal vector computation in large data sets. In: Proc. of the 31st Int'l Conf. on Very large Data Bases. 2005. 229–240. http://www.vldb2005.org/program/paper/tue/p229-godfrey.pdf

[12] Kossmann D, Ramsak F, Rost S. Shooting stars in the sky: An online algorithm for skyline queries. In: Proc. of the Int'l Conf. on Very Large Data Bases. 2002. http://www.dbis.ethz.ch/research/publications/43.pdf

[13] Papadias D, Tao Y, Fu G, Seeger B. Progressive skyline computation in database systems. ACM Trans. on Database Systems, 2005, 30(1):41–82. http://delab.csd.auth.gr/courses/c_mmdb/skyline.pdf

[14] Chan CY, Eng PK, Tan KL. Stratified computation of skyline queries with partially-ordered domains. In: Proc. of the ACM SIGMOD Int'l Conf. 2005. 203–214. http://www.comp.nus.edu.sg/~chancy/sigmod05-skyline.pdf

[15] Tao Y, Xiao X, Pei J. SUBSKY: Efficient computation of skylines in subspaces. In: Proc. of the 22nd Int'l Conf. on Data Engineering (ICDE 2006). 2006. http://www.cse.cuhk.edu.hk/~taoyf/paper/icde06.pdf

[16] Xia T, Zhang DH. Refreshing the sky: The compressed skycube with efficient support for frequent updates. In: Proc. of the ACM SIGMOD Int'l Conf. on Management of Data (SIGMOD 2006). 2006. 491–502. http://portal.acm.org/citation.cfm?id=1142473.1142529

[17] Pei J, Jin W, Ester M, Tao YF. Catching the best views of skyline: A semantic approach based on decisive subspaces. In: Proc. of the 2005 Int'l Conf. on Very Large Data Bases (VLDB 2005). 2005. http://www.vldb2005.org/program/paper/tue/p253-pei.pdf

© 中国科学院软件研究所 http://www.jos.org.cn

[18] Pei J, Yuan Y, Lin X, Jin W, Ester M, Liu Q, Wang W, Tao Y, Yu JX, Zhang Q. Towards multidimensional subspace skyline analysis. ACM Trans. on Database Systems (TODS), 2006,31(4):1335−1381. http://www.cs.sfu.ca/~jpei/publications/skycube-jn-final.pdf

[19] Pei J, Fu AWC, Lin X, Wang H. Computing compressed skyline cubes efficiently. In: Proc. of the 23rd IEEE Int'l Conf. on Data Engineering (ICDE 2007). 2007. http://www.cs.sfu.ca/~jpei/publications/mdskyline-icde.pdf

[20] Balke WT, Guntzer U, Zheng JX. Efficient distributed skylining for Web information systems. In: Proc. of the 9th Int'l Conf. Extending Database Technology (EDBT 2004). 2004. 256−273. http://www.l3s.de/~balke/paper/edbt04.pdf

[21] Li HJ, Tan QZ, Lee WC. Efficient progressive processing of skyline queries in peer-to-peer systems. In: Proc. of the 1st Int'l Conf. on Scalable Information Systems (INFOSCALE 2006). 2006. http://www.cse.psu.edu/~wlee/Publications/wlee%20Infoscale06b.pdf

[22] Vlachou A, Doulkeridis C, Kotidis Y, Vazirgiannis M. SKYPEER: Efficient subspace skyline computation over distributed data. In: Proc. of the 23rd IEEE Int'l Conf. on Data Engineering (ICDE). 2007. http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=4221690

[23] Lin X, Yuan Y, Wang W, Lu H. Stabbing the sky: Efficient skyline computation over sliding windows. In: Proc. of the 21st IEEE Int'l Conf. Data Engineering (ICDE 2005). 2005. 502−513. http://portal.acm.org/citation.cfm?id=1053724.1054088

[24] Deng K, Zhou XF, Shen HT. Multi-Source skyline query processing in road networks. In: Proc. of the 23rd IEEE Int'l Conf. on Data Engineering (ICDE 2007). 2007. 796−805. http://www.itee.uq.edu.au/~zxf/_papers/MUSO.pdf

[25] Chan CY, Jagadish HV, Tan KL, Tung AKH, Zhang ZJ. On high dimensional skylines. In: Proc. of the 10th Int'l Conf. On Extending Database Technology (EDBT 2006). 2006. 478−495. http://www.comp.nus.edu.sg/~atung/renmin/edbt06.pdf

[26] Lin XM, Yuan YD, Zhang Q, Zhang Y. Selecting stars: The *k* most representative skyline operator. In: Proc. of the 23rd Int'l Conf. on Data Engineering (ICDE 2007). 2007. http://www.cse.unsw.edu.au/~lxue/publication/icde07b.pdf

魏小娟(1985－),女,新疆乌鲁木齐人,硕士生,主要研究领域为数据挖掘技术,Skyline 计算.

李翠平(1971－),女,副教授,CCF 会员,主要研究领域为数据仓库,数据挖掘,数据流.

杨婧(1983－),女,博士生,主要研究领域为数据挖掘技术,Skyline 计算.

陈红(1965－),女,教授,博士生导师,CCF 高级会员,主要研究领域为数据挖掘,流数据管理,传感器,数据库.

© 中国科学院软件研究所 http://www.jos.org.cn