软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2015,26(9):2297−2310 [doi: 10.13328/j.cnki.jos.004704]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

排序的相互 *k*-Skyband 查询算法*

蒋 涛[1], 张 彬[1], 余法红[1], 柳 晴[2], 周傲英[3]

[1](嘉兴学院 数理与信息工程学院,浙江 嘉兴 314001)

[2](浙江大学 计算机科学与技术学院,浙江 杭州 310027)

[3](华东师范大学 软件学院,上海 200062)

通讯作者: 张彬, E-mail: jxbinzhang@gmail.com

摘 要: 不同于传统的 *k*-Skyband 查询方法,提出一种相互 *k*-Skyband 查询(M*k*SB),它从对称角度执行 Skyline 查询,找出所有既在 *q* 的动态 *k*-Skyband(D*k*SB)中又在 *q* 的反向 *k*-Skyband(R*k*SB)中的数据对象.进一步地,为了更好地支持用户决策和数据分析,排序操作被引入到M*k*SB算法中.因为M*k*SB需要执行*q*的D*k*SB和反向R*k*SB,故它需要遍历索引多次,从而导致了大量冗余的I/O开销.利用信息重用技术和若干有效的修剪方法,M*k*SB将多次的索引搜索合并成单次,极大地降低了I/O访问次数.同时,证明了基于窗口查询的M*k*SB(WM*k*SB)算法具有最低的 I/O 代价.在真实与合成数据集上的实验结果表明,所提出的算法是有效的且明显胜过基于 BBS 的算法,尤其 WM*k*SB 算法具有极少的 I/O 开销,通常能够减少 95%以上的冗余 I/O.

关键词: 算法;排序;*k*-Skyband;相互 *k*-skyband;空间数据库

中图法分类号: TP311 **文献标识码**: A

中文引用格式: 蒋涛,张彬,余法红,柳晴,周傲英.排序的相互 *k*-Skyband 查询算法.软件学报,2015,26(9):2297−2310. http://www.jos.org.cn/1000-9825/4704.htm

英文引用格式: Jiang T, Zhang B, Yu FH, Liu Q, Zhou AY. Randed processing for mutual *k*-Skyband query. Ruan Jian Xue Bao/Journal of Software, 2015,26(9):2297−2310 (in Chinese). http://www.jos.org.cn/1000-9825/4704.htm

Randed Processing for Mutual *k*-Skyband Query

JIANG Tao[1], ZHANG Bin[1], YU Fa-Hong[1], LIU Qing[2], ZHOU Ao-Ying[3]

[1](College of Mathematics Physics and Information Engineering, Jiaxing University, Jiaxing 314001, China)

[2](College of Computer Science and Technology, Zhejiang University, Hangzhou 310027, China)

[3](Software Engineering Institute, East China Normal University, Shanghai 200062, China)

Abstract: This paper proposes a novel Skyline query: mutual *k*-Skyband (M*k*SB) query. Unlike the traditional *k*-skyband query methods, M*k*SB executes the Skyline query from a symmetric perspective, and retrieves all the objects which are among both the dynamic *k*-Skyband (D*k*SB) of a specified query object *q* and the reverse *k*-Skyband (R*k*SB) of *q*. Furthermore, the ranking operation is introduced into M*k*SB due to its importance in data analysis and decision support. Since M*k*SB needs to perform D*k*SB and R*k*SB of *q*, it traverses the index multiple times, incurring much redundant I/O overhead. The proposed algorithms reduce multiple traversals to a single one, using the information reuse technology and several effective pruning heuristics that significantly cut down I/O accesses. Meanwhile, it is proved that M*k*SB based on window query (WM*k*SB) has the lowest I/O cost. Extensive experiments are conducted on both real and synthetic datasets, and the experimental results show that the proposed algorithms are efficient and outperform their competitors, i.e. the

* 基金项目: 浙江省自然科学基金(LY14F020038); 国家自然科学基金(61379033, 61003049); 嘉兴学院南湖学院科研重点资助项目

收稿时间: 2014-03-19; 定稿时间: 2014-07-31; jos 在线出版时间: 2015-02-02

CNKI 网络优先出版: 2015-02-02 15:28, http://www.cnki.net/kcms/detail/11.2560.TP.20150202.1528.006.html

basic algorithm based on BBS (branch and bound Skyline). Especially, WM*k*SB has the least I/O cost and often reduces more than 95% redundant I/O accesses.

Key words: algorithm; ranking; *k*-Skyband; mutual *k*-Skyband; spatial database

Skyline 计算在许多领域具有潜在的应用价值,例如:多标准决策(multicriteria decision making)[1]、空间位置导航服务(例如:在线地图服务)[2]、商业数据分析(例如:超市位置选取、产品用户偏好分析)[3,4]、环境监测(例如:火灾温度、瓦斯浓度)[5]等领域.

基本的 Skyline 计算方法可以分为两类:Skyline 查询算法[2,3]和反向 Skyline 查询(RSQ)算法[4−6].Papadias 等人[1]首次提出基于分支界限的 Skyline(branch and bound Skyline,简称 BBS)算法,其主要思想依据维护的最小堆来访问索引节点,从而保证每次处理的节点(entry)与坐标原点具有最小距离,进而使得距离原点最近的数据对象为 Skyline 对象,然后继续这个搜索过程,将渐进地(progressive)获得其他的 skyline 对象,并使得整个算法具有最优的 I/O 效率[1].进一步地,他们提出了扩展 Skyline 边界线为厚度为 k 的 Skyline 边界带,此即 *k*-Skyband.当指定查询对象 q 时,基本的 *k*-Skyband 就成为了动态 *k*-Skyband(D*k*SB),记为 D*k*SB(q).显然,D*k*SB 本质上是一种放松 Skyline 控制关系的计算,它使得不被任意其他对象的控制关系变成了最多可以被 k 个其他对象控制(k 也表示边界线的厚度).Liu 等人[7]提出反向的 *k*-Skyband(R*k*SB)查询算法,主要思想是修改 BBS 算法使得适应 *k*-Skyband 查询以及预计算方法和一些启发式的修剪策略.记查询对象 q 的 R*k*SB 为 R*k*SB(q)).进一步地,Miao 等人[8]解决了不完整数据集上的 *k*-Skyband 查询.Feng 等人[9]则利用多核体系计算机并行计算 *k*-Skyband 查询.

然而,这些 *k*-Skyband 算法仅从查询对象 q 的正向角度考虑 *k*-Skyband 的查询需求,即要求查询的结果(例如:对象 p)在 q 的动态 *k*-Skyband 中(即 $p\in$D*k*SB(q)),或者 q 的反向角度执行 *k*-Skyband 查询,即要求 q 处在某对象 p 的动态 *k*-Skyband 中($q\in$D*k*SB(p)).因而,这些算法不能直接用来处理体现相互关系的应用.例如:个人匹配服务.事实上,考虑数据之间的相互关系是非常重要的.例如:文献[10]中提出的相互 skyline 即考虑了这种需求,因为返回的查询结果 p 既在查询对象 q 的 skyline 中,也在 q 的反向 skyline 当中.为此,本文提出了体现相互关系的 *k*-Skyband 查询类型,即相互 *k*-Skyband(M*k*SB)查询.类似地,M*k*SB 查询中返回的结果既在 q 的动态 *k*-Skyband 中,也在 q 的反向 *k*-Skyband 中.

M*k*SB 查询在许多需要体现相互关系的应用场景中具有非常重要的意义.例如:考虑一个交友网站中的基于个性特征匹配的服务,且假定每个用户的爱好(喜欢音乐、体育、游戏等)表示用户的个性,每个用户某个爱好的喜欢程度表示为 0~1 之间的一个实数值(注意,不同用户的爱好程度可以相同).对于给定的查询用户 q(例如:Bob),希望根据他的爱好找出的好朋友与他的志趣都比较接近,这样 Bob 可以发出一个 *k*-Skyband 来寻找这样的朋友.假定,返回的查询结果中有位称作 Alice 的女性朋友,Bob 非常喜欢,且希望与她约会.那么,Alice 是否愿意呢?如果 Alice 也发出一个 *k*-Skyband 查询,发现 Bob 并不在她的查询结果当中(这表明 Bob 相对她获取的查询结果中的其他人来说,Bob 距离 Alice 的个性特性较远).Alice 很可能会拒绝 Bob 的约会请求,因为她认为 Bob 并不符合她的个性特性.反之,如果 Bob 在 Alice 的查询结果当中,那么 Alice 则有可能接受 Bob 的约会申请.这是因为 M*k*SB 查询保证了 Bob 和 Alice 在个性特性上是相互匹配的.

显然,相互 *k*-Skyband 查询可以通过分别执行一次查询对象 q 的正向 *k*-Skyband 和反向 *k*-Skyband,然后计算 D*k*SB(q)和 R*k*SB(q)的交集来实现.然而,这种简单的实现方法效率很低,存在大量冗余的 I/O,不适合现实的应用场景.例如:具有较少能量和计算能力的传感器网络应用中[11].另一方面,直接的 M*k*SB 查询返回的结果缺乏排序操作,使得用户不易选择.为此,本文进一步引入排序操作.例如:选取一个单调函数并取排序函数值最小的前 m 个数据对象作为结果,从而提出了排序的相互 *k*-Skyband 查询处理算法.本文利用了信息重用技术思想,将 M*k*SB 查询中的多次 *k*-Skyband 查询对应的 R-teee 搜索合并成单次搜索,并提出了一些高效修剪方法,很好地解决了 M*k*SB 查询效率问题.

本文的主要贡献如下:

(1) 提出了一种新颖的 *k*-Skyband 查询:相互的 *k*-Skyband(M*k*SB),它满足了一些应用需求.

(2) 提出了 3 种 M*k*SB 查询算法:基本的 M*k*SB(Basic M*k*SB,简称 BM*k*SB)、动态的 M*k*SB(M*k*SB based on

Dynamic *k*-Skyband,简称 DM*k*SB),以及基于窗口查询的 M*k*SB(M*k*SB based on Window query,简称 WM*k*SB).他们的主要思想包括:信息重用技术、提前终止方法以及一些高效修剪策略.

(3) 设计了详尽的性能评价实验,实验结果表明,WM*k*SB 算法通常能减少 95%以上的冗余 I/O 成本,且具有最优的 I/O 效率.

本文第 1 节是问题的定义和描述.第 2 节介绍 M*k*SB 查询工作原理.第 3 节给出 DM*k*SB 和 WM*k*SB 算法.第 4 节通过实验对 3 种算法的性能进行比较验证.最后,第 5 节总结全文.

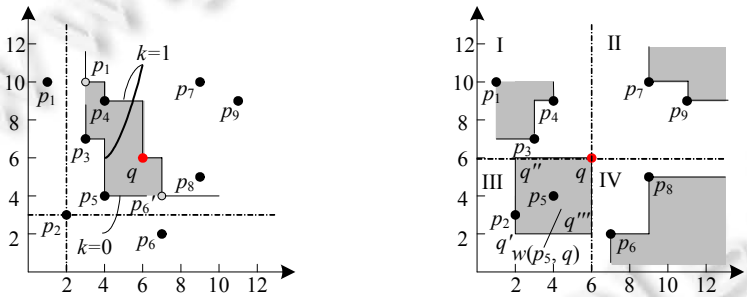
1 问题定义与描述

1.1 问题定义

相互 *k*-Skyband 查询是一种对称的 Skyline 查询,体现了查询对象 q 和数据对象 p 之间的相互关系.其查询结果中的对象 p 同时出现在 q 的 D*k*SB 和 R*k*SB 中.

定义 1(动态 *k*-Skyband(D*k*SB)). 给定查询对象 q,参数 k 和数据集 P,对象 q 的 D*k*SB 查询(记为 D*k*SB(q))返回一个最大子集 $R\subseteq P$,使得 R 中的任意对象 $r\in R$,最多被 k 个其他对象控制.

动态 *k*-Skyband 查询实质是将原数据集 P 相对于参考对象 q 映射形成新的数据集 P',然后在 P'上执行 *k*-Skyband 运算.其中,k 表示边界线(即 skyline)的厚度.当 k=0 时,*k*-Skyband 即是原始的 Skyline 计算.例如:图 1(a)中 p_2 的动态 0-Skyband 和 1-Skyband 查询结果分别为{p_3,p_5}和{p_3,p_5,p_1,p_4,q,p_6}.注意,D0SB(q)⊂D1SB(q).



(a) p_2 的动态 1-skyband (b) q 的反向 1-skyband

Fig.1 Illustration of 1-Skyband query

图 1 1-Skyband 查询示例

定义 2(反向 *k*-Skyband(R*k*SB)). 给定查询对象 q 参数 k 和数据集 P,q 的 R*k*SB 查询(记为 R*k*SB(q))返回一个最大子集 $R\subseteq P$,使得 R 中的任意对象 $r\in R$ 的动态 *k*-Skyband(D*k*SB(r))包含 q.

为了说明 R*k*SB 查询的执行过程,定义 3 给出了全局 *k*-Skyband 查询的概念.

定义 3(全局 *k*-Skyband(G*k*SB)). 给定查询对象 q,参数 k 和 d 维的数据集 P,对象 q 的 G*k*SB 查询(记为 G*k*SB(q))返回 q 的各个坐标分区中的 *k*-Skyband 查询结果的并集.例如:图 1(b)中 q 的坐标分区 I,II,III 和 IV 的 1-Skyband 分别为{p_3,p_4},{p_7,p_9},{p_5,p_2}和{p_6,p_8},G*k*SB(q)={p_3,p_4,p_7,p_9,p_5,p_2,p_6,p_8},且 G0SB(q)⊂G1SB(q).

事实上,根据定义 2 直接求 q 的 R*k*SB 具有较高的成本.通常是首先执行 q 的 G*k*SB 查询获得一个候选集 S_c,然后为 S_c 中的每个对象 $o\in S_c$ 执行 o 和 q 形成的窗口(记为 $w(o,q)$)查询[4].如果窗口中包含不多于 k 个其他数据对象(o 和 q 除外),则 o 为 R*k*SB(q)的一个查询结果;否则,o 不是 R*k*SB(q)的查询结果.例如:图 1(b)中灰白色区域为对象 p_5 的 1-Skyband 的窗口 $w(p_5,q)$.由 q 出发联向 p_5 并延长获得 q 相对于 p_5 的对称点(记为 q'),然后分别作经过 q 和 q'的水平线和垂直线,获得两个交点 q''和 q''',连接 q''至 q,再至 q''',最后至 q'组成的矩形区域,即为 $w(p_5,q)$,它仅包含对象 p_2,故 $p_5\in$R1SB(q).而 $w(p_1,q)$则包含了 p_3 和 p_4,故 $p_1\notin$R1SB(q).注意:上述定义 1~定义 3 存在关系 D*k*SB(q)⊆G*k*SB(q)且 R*k*SB(q)⊆G*k*SB(q).

根据查询对象 q 的 DkSB 和 RkSB 定义,易获得 q 的相互 k-Skyband(MkSB),其定义如下.

定义 4(相互 k-Skyband(MkSB)). 给定查询对象 q,参数 k 和数据集 P,对象 q 的 MkSB 查询(记为 MkSB(q))检索出所有既在 DkSB(q)中又在 RkSB(q)中的数据对象.它可以表示为 MkSB(q)={$p\in P|p\in$DkSB(q)$\wedge p\in$RkSB(q)}或 MkSB(q)={$p\in P|p\in$DkSB(q)$\wedge q\in$RkSB(p)}.

相互 k-Skyband 查询返回的结果可能太多,故一般用户只需要排序后的前 m 个结果,此即下面的定义 5.

定义 5(排序的相互 k-Skyband(SMkSB)). 给定查询对象 q,单调上升排序函数 F,多维数据集 P,参数 k 和 m,SMkSB 查询找出 m($m\leqslant$|MkSB(q)|)个 MkSB 对象,使得它们组成的集合 $R\subseteq P$ 有最低的分数 $score(R)=\sum_{p\in R}F(p)$,其中,$F(p)$是对象 p 在函数 F 中的函数值.这里分数越低,结果越好.

1.2 基本处理算法

基本的 MkSB 算法 BMkSB 遵循分支界限[1]搜索框架,先获得 DkSB 查询结果,再应用参数 m 约束条件,然后获取最终结果.它首先以 q 为参考对象调用文献[1]中的动态 k-Skyband[1]搜索算法获得 q 的动态 k-Skyband 查询结果集合 S_c,然后以 $o\in S_c$ 为参考对象每次调用动态 k-Skyband 算法(实际上,它是反向的动态 k-Skyband 查询)获得 o 的动态 k-Skyband 查询结果集合 S_{rc},并判断是否 q 被 S_{rc} 中最多 k 个对象控制.如果是,则说明对象 o 是 q 的相互 k-Skyband 对象.然后,通过维护一个最多包含 m 个查询结果的队列 S_r 来实现 MkSB 的排序.当 S_c 中的所有对象都判断结束后,队列 S_r 中的所有数据对象即是查询对象 q 的排序值最小的 m 个相互 k-Skyband 对象.

假设数据集 P 上的 R-tree 索引有 log|T_P|层,最小堆尺寸为 H_1,动态 k-Skyband 的数目为|R_d|,则 BMkSB 算法具有下面的定理.

定理 1. 给定排序参数 m,BMkSB 算法的时间和空间复杂度分别为 $O((1+\min(m,|R_d|))\times\log|T_P|)$和 $O(2H_1)$.

证明:BMkSB 执行动态 k-Skyband 查询需要 $O(\log|T_P|)$次 I/O 访问,使用一个尺寸为 H_1 的最小堆.另外,候选集中的每个对象 p 在执行反向动态 k-Skyband 同样需要 $O(\log|T_P|)$次 I/O 访问,尺寸为 H_1 的最小堆,这个查询过程最少需执行 m 次,最多|R_d|次.因此,其总时间复杂度为 $O((1+\min(m,|R_d|))\times\log|T_P|)$,总空间复杂度为 $O(2H_1)$. □

注意,当需要排序的对象数目 $m>|R_d|$时,BMkSB 执行 RkSB 查询的实际次数为|R_d|;否则,$m\leqslant|R_d|$时,BMkSB 执行 RkSB 查询的实际次数为 m,故其时间复杂度函数中应使用最小值 min 函数,而不是最大值 max 函数.

2 相互 k-Skyband 查询原理

显然,BMkSB 算法需两个步骤才能完成,即动态 k-Skyband 和反向 k-Skyband,从而需多次访问 R-tree 树,存在大量冗余的 I/O 成本.下面将讨论改进它的基本原理.

2.1 信息重用技术

BMkSB 算法通过执行一次反向的 k-Skyband,来判断是否 $q\in$DkSB(p),从而判断候选对象 p 是否属于 MkSB(q)(其中,p 来自第 1 阶段获得的候选集 S_c).显然,此过程需要一次完整的索引访问过程.事实上,如果能够将该索引(例如:R-tree)访问过程中的节点(中间索引节点或叶子的数据对象节点)信息(即最小边界矩阵 MBR 信息)保存下来,并能在后面的反向搜索过程中利用,那么 MkSB 查询将仅需一次完整的 R-tree 遍历.

考虑图 2 中的动态 1-Skyband(D1SB)查询过程.假设单调上升排序函数 F(=min{$L_1(e,q)$})为节点 e 与 q 的 L_1(坐标之差绝对值)距离最小值,即 q 与 e 的 MBR 的四条边的最小距离值.首先节点 N_1(即 e_1)和节点 N_2(即 e_2)被插入辅助堆 H 中.然后,访问节点 e_1,最小堆 H={e_2,e_3,e_4}.接着,由于 $F(e_2)$=min{$L_1(e_2,q)$}=1.5 小于 $F(e_3)$=2,访问节点 N_2(即 e_2),最小堆 H={e_3,e_5,e_6,e_4}.紧接着 N_3(即 e_3)被访问,数据节点 a,b,c 被插入 H 中.此时,数据节点 c 位于堆 H={c,e_5,e_6,e_4,b,a}的顶部且不被任何对象控制,因而它成了动态 1-Skyband 结果集 R_d 的第 1 个元素.此时,将节点 c 相对 q 进行映射(图 2 中二维空间中为左右、对角线和上下方向),得到 c',c'',c'''点(灰色圆点).进一步得到 I,II,III,IV 这 4 个阴影区域,即节点 c 相对于 q 的动态控制区域 DDR_q(dynamic dominating region).由于节点 N_4(即 e_4),N_5(即 e_5),N_6(即 e_6)和点 a 完全包含在 $DDR(c)$中,N_4,N_5,N_6 和 a 被 R_d={c}控制.接着扩展 e_5,数据点 f 控制了 a 和 e_4,故 a 和 e_4 被修剪.然后扩展节点 e_6,数据点 i 和 j 被 R_d={c,f}控制,故 i 和 j 被修剪.最终,得到 D1SB(q)的查

询结果为 R_d={c,f,b,g}.

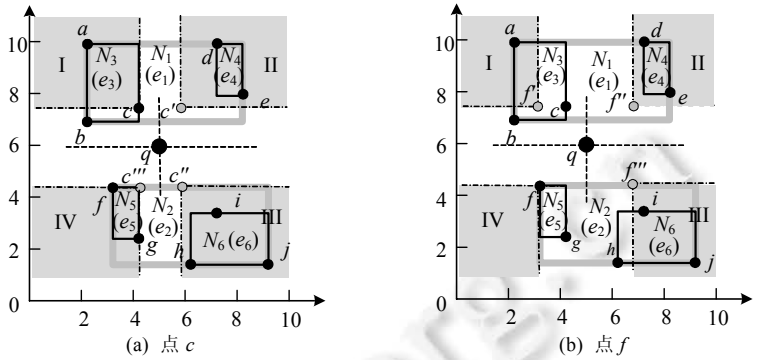


Fig.2 Dynamic dominating region of dynamic 1-Skyband query

图 2 动态 1-Skyband 的动态控制区域

上述动态 1-Skyband 搜索过程中辅助堆 H 内容和结果集 R_d 的变化情况见表 1.其中,堆的内容为(节点,节点与 q 之间的最小距离)对.带删除线的节点或数据点表示被 R_d 中至少两个数据点控制,从而可以修剪掉.

Table 1 The auxiliary heap content during dynamic 1-Skyband

表 1 动态 1-Skyband 搜索过程辅助堆的内容

Action	Heap content(H)	R_d
Access root	(e_1,1)(e_2,1.5)	Null
Expand e_1	(e_2,1.5)(e_3,2)(e_4,4)	Null
Expand e_2	(e_3,2)(e_5,2.5)(e_6,3.5)(e_4,4)	Null
Expand e_3	(c,2.5)(e_5,2.5)(e_6,3.5)(e_4,4)(b,4)(a,7)	{**c**}
Expand e_5	(f,3.5)(e_6,3.5)~~(e_4,4)~~(b,4)(g,4.5) ~~(a,7)~~	{c,**f**}
Expand e_6	(b,4)~~(i,4.5)~~(g,4.5)(h,5.5)~~(j,8.5)~~	{c,f,**b**}
Access g	(g,4.5)~~(h,5.5)~~	{c,f,b,**g**}

上述动态 1-Skyband 搜索过程中重用信息堆 H_r 内容和结果集 R_d 的变化情况见表 2.

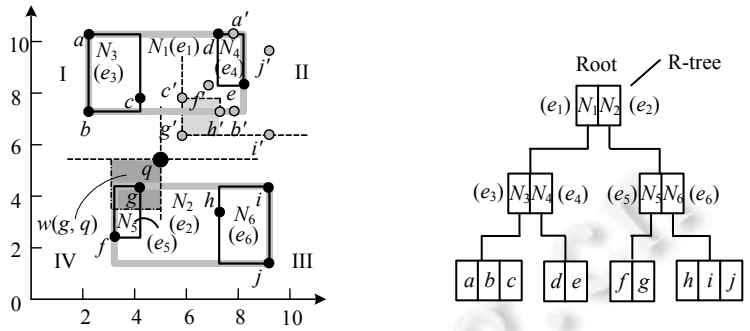
Table 2 The reuse heap content during dynamic 1-Skyband

表 2 动态 1-Skyband 搜索过程重用堆的内容

Action	Heap content(H_r)	R_d
Access *root*	(e_1,1)(e_2,1.5)	Null
Expand e_1	(e_2,1.5)(e_3,2)(e_4,4)	Null
Expand e_2	(e_3,2)(e_5,2.5)(e_6,3.5)(e_4,4)	Null
Expand e_3	(c,2.5)(e_5,2.5)(e_6,3.5)(e_4,4)(b,4)(a,7)	{**c**}
Expand e_5	(c,2.5)(f,3.5)(e_6,3.5)(e_4,4)(b,4)(g,4.5)(a,7)	{c,**f**}
Expand e_6	(c,2.5)(f,3.5)(e_6,3.5)(e_4,4)(b,4)(g,4.5)(i,4.5)(h,5.5)(a,7)(j,8.5)	{c,f}
Access b and g	(c,2.5)(f,3.5)(e_4,4)(b,4)(g,4.5)(i,4.5)(h,5.5)(a,7)(j,8.5)	{c,f,**b**,**g**}

上面的分析过程表明重用堆 H_r 的信息随着动态 1-Skyband 的执行,访问过程中的节点信息存储在 H_r 中,如果某节点仍需扩展,则该节点从 H_r 中被删除而其孩子节点被插入 H_r,直到动态 1-Skyband 执行完成.例如: 访问节点 N_3 前 H_r={e_3,e_5,e_6,e_4},访问节点 N_3 后,其孩子节点{a,b,c}被插入 H_r 中,而节点 N_3={e_3}被删除,此时 H_r={c,e_5,e_6,e_4,b,a}.

尽管文献[10]也使用了重用方法,但它们仅保存第 1 个动态 1-Skyband 结果之前访问的节点信息.例如:图 3 中获得 q 的 D1SB 的第 1 个结果 g 后 H_r={e_5,e_1,e_6}.然后,求得 D1SB(q)={g,c,h,i}(图 3(a)中 II 区中虚线相连的点,其中灰色圆点为其他分区在 II 分区中的映射点)的过程中堆 H_r 的内容不发生变化.然而本文的重用堆 H_r 随着整个相互 1-Skyband 执行完成而发生变化.例如:在求得 c∈D1SB(q)时,H_r={e_5,e_3,e_6,e_4}.这样,执行对象 c 的窗口查询时不会产生新的 I/O.然而,文献[10]的信息重用方法却需 1 次 I/O 访问(因为 e_3∉H_r).



(a) q 的 G1SB (b) R-tree 结构

Fig.3 Illustration the use of H_r

图 3 重用堆 H_r 使用实例

2.2 合并多次索引搜索

如果 MkSB 算法执行动态 k-Skyband 时将访问信息保存在重用堆 H_r 中.那么,它再执行多次反向动态 k-Skyband 就可以求得排序后的前 m 个相互 k-Skyband 的查询结果.

另一方面,相互 k-Skyband 可以在动态 k-Skyband 的候选结果集 S_c 的基础上,结合重用堆 H_r 的内容,通过执行窗口查询[4]来求得最终的结果,即下面的定理 2.

定理 2. 给定查询对象 q 和多维数据集 P,候选结果集 S_c=DkSB(q),如果窗口查询 $w(p\in S_c,q)$包含超过 k 个其他对象(p 和 q 除外),则 p 不是相互 k-Skyband 对象.

证明:假设任意对象 $p\in$MkSB(q),则有 $p\in$DkSB(q),即 p 是 q 的动态 k-Skyband.根据集合 DkSB(q)和 GkSB(q)的关系有:DkSB(q)⊆GkSB(q),又 MkSB(q)⊆DkSB(q)⊆GkSB(q),故有 $p\in$GkSB(q).根据已知条件知,GkSB 对象 p 的窗口 $w(p,q)$包括的其他对象个数超过 k 个,因而 q 相对于 p 被至少 k+1 个对象控制.换句话说,q 不在 p 的 DkSB 查询结果当中.根据 RkSB 的定义知,$p\notin$RkSB(q).既然 $p\in$DkSB(q)又 $p\notin$RkSB(q),从而 $p\notin$MkSB(q). □

定理 2 具有重要意义,它保证了在执行 DkSB 查询的基础上,通过窗口查询可以获得 MkSB 查询结果,而不必执行返回结果很多的 GkSB 查询.例如:图 3(a)中执行 $g\in$D1SB(q)的窗口查询 w(g,q)就可以判断 $g\in$M1SB(q),即 g 是相互 1-Skyband 的结果.

既然,H_r 可以在 DkSB 中使用,也可以在窗口查询中使用,可以证明下面的定理 3 是正确的.

定理 3. 给定查询对象 q 和多维数据集 P,如果使用前述重用堆 H_r 来保存 q 的 DkSB 执行过程中的节点信息,无论是通过多次窗口查询或 RkSB 查询来完成 MkSB 查询,MkSB 算法不会产生漏报或多报.

证明:显然,H_r 完整地保存了数据集 P 中的所有数据对象不同层次的索引信息.例如:图 3(a)中数据对象 a,b,c 为第 0 层的叶子节点,而数据对象 d,e 为第 1 层的中间索引节点 e_4.这样,通过 H_r 一定可以访问到所有节点的 MBR 信息.定理 2 已经证明了通过窗口查询的方法可以获得 MkSB,故而不会产生多报或漏报.另一方面,对于候选集 S_c 的某个对象 $p\in S_c$,以 p 为参考对象并使用 H_r 也可以求得 p 的动态 k-Skyband 结果集合 S_{rc}.根据定义 2 如果 $q\in S_{rc}$,那么对象 p 就是一个 MkSB 对象.因此,这种方法也不会产生多报或漏报. □

定理 3 也是非常重要的,它保证了信息重用方法的正确性.这样,就可以合并 MkSB 的 R-tree 遍历成单次 R-tree 遍历,从而极大地降低 I/O 访问次数.

2.3 讨 论

本节讨论重用堆 H_r 的空间复杂度.显然,前述方法的有效性依赖于 SMkSB 在第 1 阶段执行 DkSB 过程中 H_r 的空间复杂度.下面从理论上进行分析论证.

假设数据集 P 包含 E 个数据对象,索引中根节点包括α个节点,平均每个节点包括μ个孩子节点,索引包含 L 层,每次 I/O 操作页面可以容纳β个节点,则根节点需$\lceil \alpha/\lambda \rceil$次 I/O 访问,其他节点平均需$\lceil \mu/\beta \rceil$次 I/O 访问.易求出

首个动态 k-Skyband 点需要的最小 I/O 次数为

$$N(1)=\lceil \alpha/\beta \rceil+(L-1)\times\lceil \mu/\beta \rceil \tag{1}$$

重用堆 H_r 的尺寸$|H_r|$至少如下面的公式(2):

$$|H_r|=\alpha+(L-1)\times\mu \tag{2}$$

同时索引层次满足下面的公式(3):

$$E=\alpha+\alpha.\mu+\alpha.\mu^2+\ldots+\alpha.\mu^{L-1} \tag{3}$$

联立等式(1)~等式(3)得到下面的公式(4)和公式(5):

$$|H_r|=\alpha+(\log_{\mu}^{E\cdot(\mu-1)/\alpha+1}-1)\times\mu \tag{4}$$

$$N(1)=\lceil \alpha/\beta \rceil+(\log_{\mu}^{E\cdot(\mu-1)/\alpha+1}-1)\times\lceil \mu/\beta \rceil \tag{5}$$

公式(4)和公式(5)分别反映了文献[10]重用堆的大小和 I/O 次数,它们的空间复杂度是比较低的.如果所有 DkSB 对象在一个节点,则本文重用堆 H_r 的最大尺寸与文献[10]的相同.否则,此时重用堆在后续的 DkSB 对象求出时可能会继续扩展,以至 H_r 继续增大,直到 DkSB 执行完.访问的 I/O 与此类似,每当节点扩展时,将存在 I/O 操作,但同时由于 H_r 变大,大部分的访问信息可以从 H_r 获得,从而以后需要的 I/O 操作将逐渐减少(虽然,总的 I/O 次数仍可能增加).

基于表 2 的观察可以发现:在求得 DkSB 时,H_r 中包含了很多数据对象.在这种情况下,一种方法是限定 H_r 中的第 1 层节点不能扩展,从而缩减 H_r 的尺寸.

3 排序的相互 k-Skyband 查询处理

排序的相互 k-Skyband 采用边搜索边修剪的方式实现,运用约束条件尽早终止计算,充分重用已有信息,最大限度减少 I/O 访问和 CPU 时间.

3.1 DMkSB算法

DMkSB 算法的基本思路与 BMkSB 算法类似.主要不同之处在于:(1) DMkSB 执行动态 k-Skyband 查询时会将运行过程中的访问信息保存在重用堆 H_r 中;(2) DMkSB 在判断当前对象 $p\in$DkSB(q)是否是相互 k-Skyband 而执行反向动态 k-Skyband 时使用 H_r,极大地减少了 I/O 次数;(3) DMkSB 边搜索边修剪可以在其第 5 行的 DkSB(q)和第 14 行的 DkSB(p)分别提前终止计算,而不像 BMkSB 完整地执行完它的 DkSB(q)和反向 DkSB(p).前者可提前终止是因为当前节点 e 的低边界排序函数值 $LB_F(e)=\min\{F(x)|\forall x\in e\}$比查询结果中的第 m 个最小的排序函数值大,DkSB(q)不必要全部执行完;后者可提前终止是因为如果$\exists(k+1)$个对象 $p'\in D$ 相对于 p 控制 q,DkSB(p)不必要执行完.DMkSB 执行 DkSB(q)提前终止计算的推理体现在下面的定理 4.

定理 4. 给定排序函数 $F(.)$和排序参数 m,对象 $o_1,o_2,\ldots,o_m$ 是目前获得的 m 个 MkSB 结果对象,节点 e 的最小排序值 $LB_F(e)$为 $\min\{F(x)|\forall x\in e\}$,$m\_score$ 表示 m 个结果对象中的最大排序函数值 $\max\{F(o_i)|o_i,\forall i\in[1,m]\}$.如果满足条件 $LB\_F(e)\geqslant m\_score$,则可以安全地修剪掉节点 e.

证明:既然节点 e 的任意数据对象的最小排序值 $LB_F(e)$都比 m_score 大,那么一定存在至少 m 个结果对象 $o_1,o_2,\ldots,o_m$ 的分数值比 e 中的任意对象的分数值小.因此,节点 e 可以安全地修剪掉. □

DMkSB 的详细算法描述见算法 1.

算法 1. DMkSB(T,k,F,m,q,S_r).

输入: /*T: index; k and m: skyband and ranked parameter; $F(.)$: preference function; q: query object*/;

输出: S_r: result set.

1. $S_r=\varnothing,S_c=\varnothing,H=\varnothing,H_r=\varnothing,m\_score=+\infty$;
2. Insert *root* into H and H_r with (e,*mindist*(e,q));
3. **WHILE** H is not empty **DO**
4. Pop the entry (e,*mindist*(e,q)) of H;

5. **IF** $LB\_F(e)\geqslant m\_score$ **THEN BREAK**;
6. **IF** e is not contained in (k+1) $DDR_q(o\in S_c)$ **THEN**
7. **IF** e is an intermediate entry **THEN**
8. **FOR** each child entry $e_i\in e$ **DO**
9. **IF** $e_i\notin k$+1 $DDR_q(o\in S_c)$ **THEN**
10. Insert (e_i,*minidist*(e_i,q)) into H;
11. Insert all $e_i\in e$ into H_r, Delete e from H_r;
12. **ELSE** //e is a data object
13. $S_c=S_c\cup\{e\}$; //e is a DkSB object w.r.t. q
14. *rflag*=RDkSB(T,k,e,q,H_r);
15. **IF** *rflag*==TRUE **THEN**
16. **IF** $|S_r|<m$ **THEN**
17. $S_r=S_r\cup e\}$; //e is a MkSB object w.r.t. q
18. Update *m_score* using largest score in S_r;
19. **ELSE** //$|S_r|\geqslant m$
20. let o' be the object in S_r with m-th smallest score;
21. **IF** $F(e)<m\_score$ **THEN**
22. $S_r=S_r\cup\{e\}$;
23. Remove o' from S_r;
24. Update *m_score* using new m-th smallest score;
25. **RETURN** S_r;

DMkSB 执行动态 k-Skyband 的方法是:判断节点 e 是否完全包含在求得的 DkSB 候选集合 S_c 中的 k+1 个数据对象 $o\in S_c$ 的动态控制区域 $DDR_q(o)$中(算法第 6 行和第 9 行).显然,如果节点 $e\subseteq DDR_q(o)$,则 e 被 o 动态控制.第 11 行更新重用堆 H_r,即从中删除当前节点 e,并将 e 的所有孩子节点 $e_i\in e$ 插入 H_r 中.为了提高 H_r 更新效率,可以采用二分搜索方法.第 13 行添加对象 e 到 DkSB 候选集合 S_c 中,第 14 行以 e 为参考对象,使用重用堆 H_r,调用反向 DkSB 查询算法 RDkSB(行 14),判断是否存在对象 p'相对当前对象 $e\in$DkSB(q)控制 q.如果存在,则该算法返回 FALSE,并保存到变量 *rflag* 中;否则 *rflag* 为真(行 15),然后算法在第 16 行~第 24 行实现对获得的最多 m 个查询结果的排序和动态更新.其中,RDkSB 算法伪代码描述如算法 2 所示.

算法 2. RDkSB(T,k,p,q,H_r).

输入: /*T: index; k: skyband parameter; p: candidate object$\in$DkSB(q); q: query object; H_r: reuse heap*/;

输出: *rflag*: an Boolean value whether q is dominated k+1 object w.r.t current object $p\in$DkSB(q).

1. $S_c=\varnothing$,$H=\varnothing$; //S_c—result set of RkSB(q)
2. **WHILE** H_r is not empty **DO**
3. Pop the entry (e,*mindist*(e,q)) of H_r;
4. Push the entry (e,*mistdist*(e,p)) into H;

5~13. Lines 3~4, 6~10, and 12~13 of DMkSB but q is replaced by p in lines 4, 6, 9 and 10

14. **IF** $\exists$(k+1) points in e dominates q w.r.t. p **THEN**
15. **BREAK**, **RETURN** FALSE;
16. **RETURN** TRUE;

RDkSB 算法的最好优先(best first)搜索框架与算法 DMkSB 类似.算法的 2~4 行将重用堆中的信息弹出,并根据 DkSB(q)中的当前候选对象 p 重新计算节点 e 的最小距离 *mistdist*(e,p),然后插入搜索辅助堆 H 中.注意,由于是以 p 为参考对象进行动态 DkSB 查询,因而第 5 行~第 13 行使用 p 代替 q 计算节点 e 的最小距离 *mindist*(e,

p)和动态控制区域 $DDR_p(o\in S_c)$.第 14 行判断当前节点 e 是否存在 k+1 个对象相对于 p 控制 q.如果是,根据 k-Skyband 定义,则当前候选对象 p 一定不是 MkSB 对象,从而可以终止计算(行 15),并返回 FALSE;否则,计算出所有的 RkSB 对象,并返回 TRUE(行 16),因为 $q\in$DkSB(p).

从 DMkSB 算法的描述和分析中,容易得到下面的 I/O 性能定理 5.

定理 5. 假定 q 的 DkSB 查询包含$|R_d|$个数据对象,排序函数参数为 m,则 BMkSB 算法从磁盘装入访问 H_r 中实体 1+min(m,$|R_d|$)次,而 DMkSB 算法仅仅装入 1 次.

3.2 WMkSB算法

WMkSB 的实现框架与 DMkSB 完全一致.不同之处在于:它通过调用窗口查询,以当前 DkSB(q)的候选对象 p 与 q 的窗口 $w(p,q)$是否包含 k+1 其他数据对象 $o\in w(p,q)$来判断 p 是否是 MkSB 对象(定理 2),并利用重用堆 H_r 信息和实现提前终止计算.其实质是利用窗口查询来代替反向的 DkSB 计算(例如:DkS$B(p)$,其中,$p\in$DkSB(q)).仅替换 DMkSB 算法第 14 行中的 RDkSB 函数为 k-Skyband 窗口查询函数 SBWinQ,便可以得到算法 3.需要说明的是,尽管 DMkSB 和 WMkSB 算法中代码区别较少,但是他们的实现方法的原理完全不同,WMkSB 的实现原理体现在定理 2.

算法 3. WMkSB(T,k,F,m,q,S_r).

输入: /*T: index; k and m: skyband and ranked parameter; F(.): preference function; q: query object*/;

输出: S_r: result set.

1~25. Lines 1~25 of DMkSB but the function RDkSB of line 14 is replaced by the function of window query SBWinQ

SBWinQ 利用重用信息时,可使用 RDkSB 算法中行第 2 行~第 4 行的代码.而它判断窗口 $w(p,q)$是否包含 k+1 其他对象(例如:图 3 中 g 点的窗口 $w(g,q)$不包含其他对象,故是一个 MkSB 对象),则可以使用类似 BBS 算法的 R-tree 索引遍历过程,而需要将控制关系的判断语句更改为节点的 MBR 是否与 $w(p,q)$相交或包含 $w(p,q)$这种位置关系的判断语句即可,并使用布尔查询方法:即遇到 k+1 个其他数据对象 $p'(p'\neq p\wedge p'\neq q)\in w(p,q)$时,立刻终止计算,并返回 FALSE.从 WM$k$SB 算法的描述分析中,可以得到类似于定理 5 的关于 WMkSB 算法的 I/O 性能定理.

4 实验评估

实验采用真实数据集 NBA[12]、Hou[12](抽取前三维组成实际数据集),它们分别包含了 5 维 17K、6 维 127K 个数据对象以及服从不同分布特性的合成数据集 *Correlated*(*Corr*)和 *Independent*(*Indep*),数据空间为[1,10000].查询对象 q 从数据集中随机选取 50 个,实验结果取 50 次运行的平均值.除了排序参数 m(排序的查询结果数目)实验外,所有实验中排序参数 m 设置为 16,排序函数 F 为 $\min\{L_1(x,q)|\forall x\in e\}$.实验通过两个重要的性能指标 I/O 访问次数和 CPU 运行时间(不包括 I/O 时间)来评估 BMkSB,DMkSB 和 WMkSB 的性能(由于目前未出现同类算法,故本文算法未与其他算法比较).

其他参数设置如下.R-tree 页面尺寸和 cache 块大小为 4 096 字节.实验在双核 3.2GHz、500G 硬盘的 PC 机上运行,编程语言为 C++.注意,实验绘图中的折线表示 I/O 访问次数,对应左边的对数纵坐标,柱状图表示 CPU 运行时间,对应右边的纵坐标(其中,数据维度变化实验中使用对数坐标).绘图中的 RHS 表示重用堆包含索引节点的数目.需要说明的是,类似于文献[6]和文献[12]中的 R-tree 实现代码,本文的 R-tree 代码也是基于磁盘实现(即访问 1 次索引节点将产生 1 次 I/O 操作).

4.1 真实数据集上的算法性能

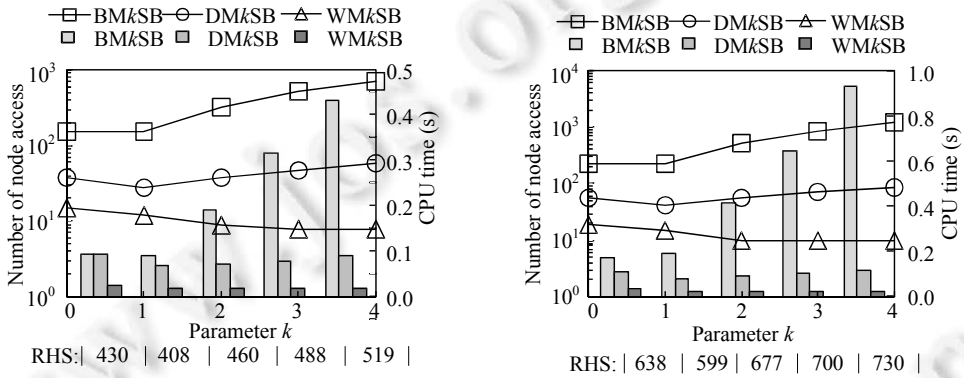
(1) 参数 k 的影响

第 1 组实验比较算法在真实数据集上随参数 k 大小(skyband 厚度参数)的性能(cache 块数目为 0).图 4 的实验结果反映:随着 k 的增大,BMkSB 和 DMkSB 算法 I/O 节点访问次数和 CPU 运行时间(单位:s)都逐渐增大;然而 WMkSB 的 I/O 成本和 CPU 时间逐渐减少且有最小的成本.实际上,当 k 的值由 0 分别增加到 1,2,3 和 4 时,3

种算法在 NBA 和 Hou 数据集上的相互 *k*-Skyband 的数目分别为 13,13,27,36,46,以及 15,15,31,42,53.这也在一定程度上解释了图 4 中 I/O 成本和 CPU 时间的增长趋势.3 种算法的 I/O 次数分别处在不同的数量级上,其中 DM*k*SB 的 I/O 次数仅与排序参数 *m* 处于一个数量级.这是因为 M*k*SB 结果对象在 *k* 较小(例如:*k*=1)时,一般处于索引的同一节点上;同时重用堆技术的使用减少了大量的冗余 I/O 访问.例如:*k*=4 时,WM*k*SB 减少了 BM*k*SB 算法 99.14%以上的 I/O.

(2) 数据集尺寸(*s*)的影响

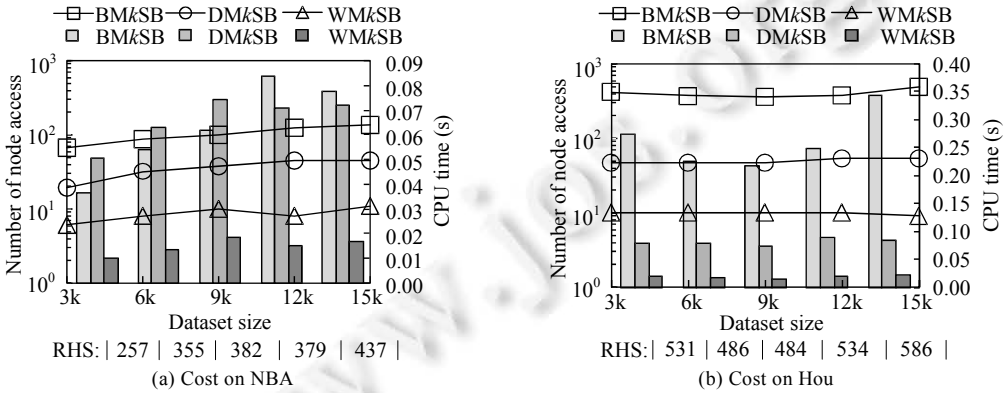
第 2 组实验比较算法在真实数据集上随着数据集尺寸 *s* 变化的性能.参数 *k* 固定为 2,cache 块数目为 0.图 5 的实验结果反映:增大数据集尺寸,3 种算法的 I/O 成本和 CPU 运行时间沿着增长趋势扩大.但有趣的现象是 3 种算法的局部性能可能下降.这是因为,I/O 访问不仅与数据尺寸相关,也与数据分布和 *q* 的位置有关.由于使用了重用技术,故 DM*k*SB 的 I/O 成本稍差于 WM*k*SB 却比 BM*k*SB 好得多.



(a) Cost on NBA (b) Cost on Hou

Fig.4 Performance on real datasets vs. *k* (cache blocks=0)

图 4 算法在真实数据集上随 *k* 变化的性能(cache 块数目=0)



(a) Cost on NBA (b) Cost on Hou

Fig.5 Performance on real datasets vs. dataset size (*k*=2, *b*=0)

图 5 算法在真实数据集上随数据集尺寸变化的性能(*k*=2, *b*=0)

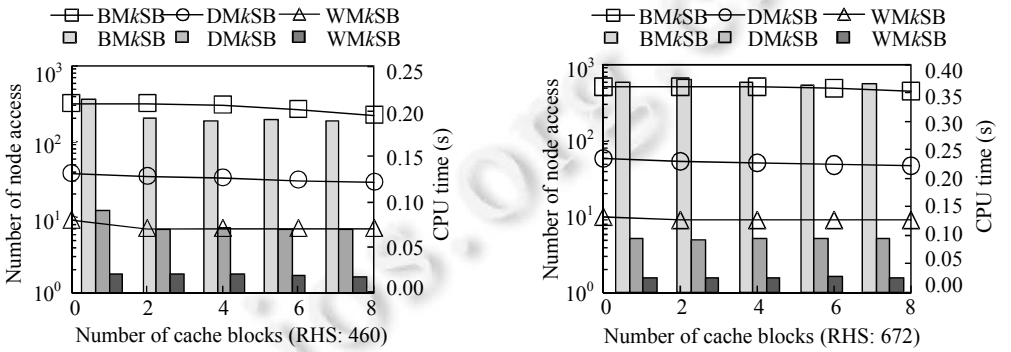
(3) Cache 块数目(*b*)的影响

第 3 组实验比较算法在真实数据集上随 cache 块数目 *b* 变化的性能(*k*=2).图 6 的实验结果体现了重用方法远比增加 cache 大小的方法更能减少 I/O 次数.例如:在 Hou 上,16K 的 cache 仅能减少 1.56%的 I/O 访问,而 DM*k*SB 和 WM*k*SB 的重用技术却分别减少 88.67%和 96.68%的 I/O 访问.因为 cache 基于局部性原理来保存节点信息, 常常下次访问的节点并不存在 cache 中.另一方面, 重用技术的运用减少了许多不必要的计算,使得

WM*k*SB 比 DM*k*SB 快 3~5 倍,比 BM*k*SB 快至少 10 倍以上.

(4) 排序参数(*m*)的影响

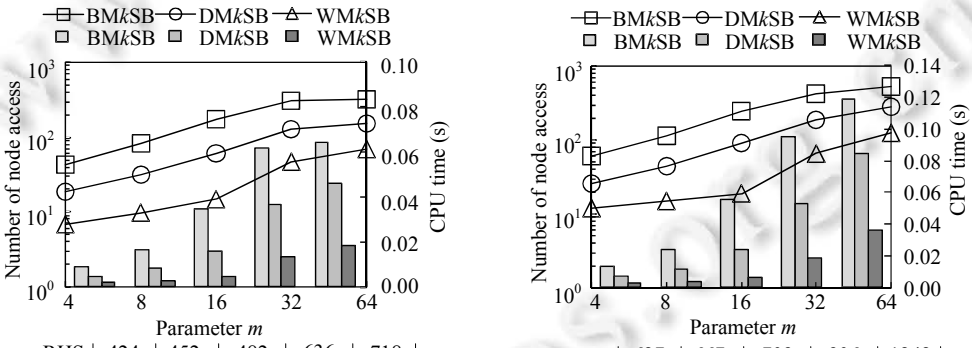
第 4 组实验比较排序参数 *m*(要求返回的查询结果数目)在真实数据集上对各种算法的影响.图 7 中的实验结果表明,3 种算法都将随着 *m* 的增加,I/O 次数和 CPU 运行时间都将增大,但是算法 WM*k*SB 具有最好的性能.BM*k*SB 和 DM*k*SB 算法在参数 *m* 增大到 32 之后,I/O 次数将缓慢增长;而 WM*k*SB 算法在 *m* 增大到 16 后,在 16~32 的范围内,I/O 次数较快速地增长.这是因为 *m* 较小时,WM*k*SB 返回的结果可以在一个节点上找到.



(a) Cost on NBA (b) Cost on Hou

Fig.6 Performance on real datasets vs. cache blocks (*k*=2)

图 6 算法在真实数据集上随 cache 块数目变化的性能(*k*=2)



(a) Cost on NBA (b) Cost on Hou

Fig.7 Performance on real datasets vs. *m* (*k*=2, *b*=0)

图 7 算法在真实数据集上随 *m* 变化的性能(*k*=2, *b*=0)

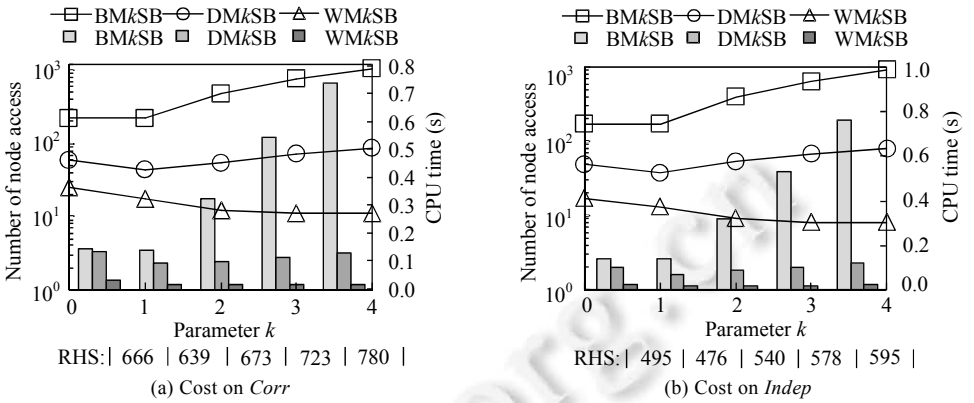
4.2 合成数据集上的算法性能

(1) 参数 *k* 的影响

第 1 组实验评估算法在 512K 尺寸 3 维的合成数据集上随着 *k* 变化的性能.图 8 的实验结果表明:增大 *k*,实验结果和现象及原因与图 4 类似.WM*k*SB 仍具有与参数 *m* 一个数量级的 I/O 访问次数.

(2) 数据集大小 *s*(size)的影响

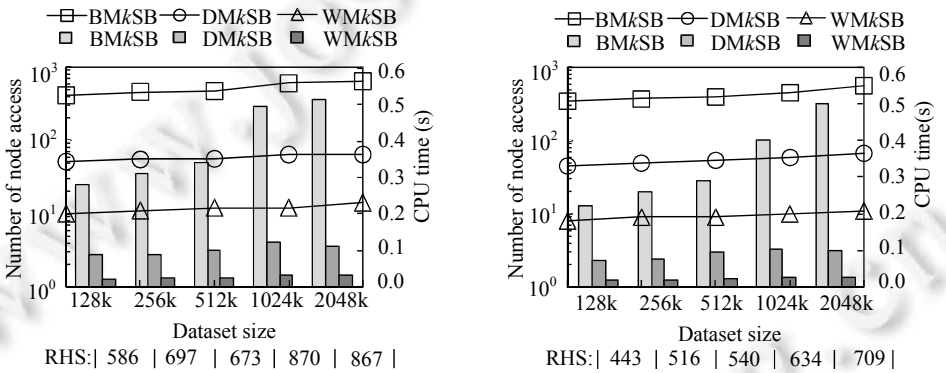
第 2 组实验比较算法在 3 维的合成数据集上随着数据集尺寸 *s* 变化时的性能.数据集尺寸分别为 128K,256K,512K,1024K 和 2048K,参数 *k*=2,*b*=0.图 9 的实验结果和相应的现象及原因类似于图 5.



(a) Cost on *Corr* (b) Cost on *Indep*

Fig.8 Performance on synthetic datasets vs. k (b=0)

图 8 算法在合成数据集上随 k 变化的性能(b=0)



(a) Cost on *Corr* (b) Cost on *Indep*

Fig.9 Performance on synthetic datasets vs. s (k=2, b=0)

图 9 算法在合成数据集上随 s 变化的性能(k=2, b=0)

(3) Cache 块数目 b(cache blocks)的影响

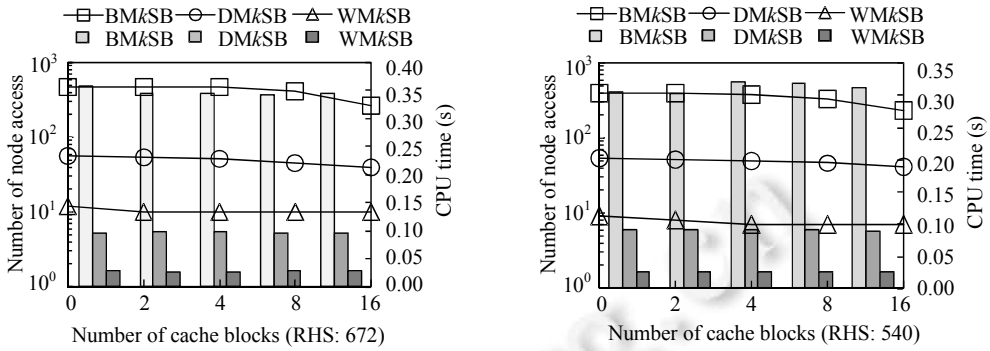
第 3 组实验报告 512K 大小 3 维的合成数据集随着 b 变化的性能(k=2).图 10 中的实验结果及相应现象及原因类似于图 6:与使用 H_r 相比,增大 b 仅能很少程度上减少 I/O 访问次数,仅当 cache 较大的时才有较为明显的效果(例如:b=32);但对 CPU 运行时间几乎没有影响.WMkSB 最优 I/O 性能充分表明了重用信息策略的优势.

(4) 数据维度 d 的影响

第 4 组实验比较算法在 512K 的合成数据集上随着维度 d 变化的性能(k=2,b=0).从图 11 的实验结果中可以看出,随着 d 的增大,BMkSB 算法的 I/O 次数明显地增大,DMkSB 缓慢增大,而 WMkSB 算法仅仅轻微增大.在 5 维情形的 *Indep* 数据集上,DMkSB 减少了 BMkSB 算法 98.6%的 I/O 次数,而 WMkSB 又减少了 DMkSB 算法 96.1%的 I/O 次数,总体上 WMkSB 仅有 BMkSB 算法 0.055%的 I/O 访问.另一方面,WMkSB 算法在 *Indep* 上,其 CPU 运行速度比 BMkSB 快 1780 倍,比 DMkSB 也快了 13 倍.这说明维度 d 对算法性能有较大影响.由于 *Corr* 对角线分布、*Indep* 均匀分布,*Corr* 比 *Indep* 具有更多 I/O 访问.

(5) 排序参数(m)的影响

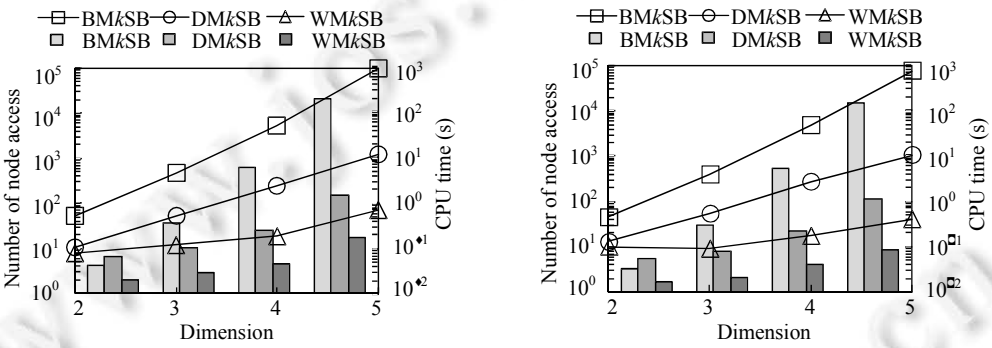
第 5 组实验评估合成数据集上排序参数 m 对各种算法的影响(数据集尺寸为 512K,维度为 3).图 12 中的实验结果及相应现象及原因类似于图 7:WMkSB 在 I/O 成本和 CPU 运行时间都优于其他两种算法.



(a) Cost on *Corr* (b) Cost on *Indep*

Fig.10 Performance on synthetic datasets vs. b (k=2)

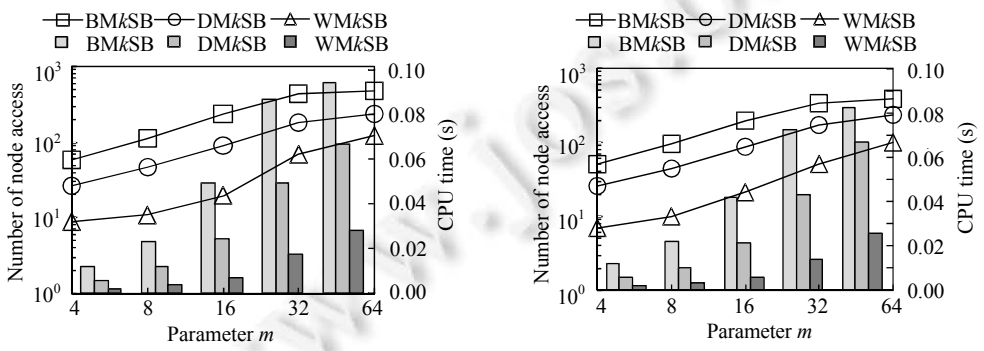
图 10 算法在合成数据集上随 b 变化的性能(k=2)



(a) Cost on *Corr* (b) Cost on *Indep*

Fig.11 Performance on synthetic datasets vs. d (k=2, b=0)

图 11 算法在合成数据集上随着维度 d 变化的性能(k=2, b=0)



(a) Cost on *Corr* (b) Cost on *Indep*

Fig.12 Performance on synthetic datasets vs. m (k=2, b=0)

图 12 算法在合成数据集上随着 m 变化的性能(k=2, b=0)

5 结 论

相互 k-Skyband 查询从对称角度设计应用,在商业数据分析、个性化匹配等方面具有重要的应用价值.本文

提出了 3 种高效的排序相互 k-Skyband 查询算法:BMkSB,DMkSB 和 WMkSB.算法采用 BF 搜索策略,利用重用堆思想和尽早终止计算等方法,极大地减少了算法的冗余 I/O 访问,加速了 CPU 的运行效率.最后,实验和理论分析都证明了基于窗口查询和信息重用的 WMkSB 算法在 skyband 厚度、数据集大小和维度以及排序参数 m 上,都具有最优的性能.WMkSB 是目前为止最优的相互 k-Skyband 查询算法:在 I/O 效率上,它几乎保持了 $O(m)$的线性 I/O 访问次数.

References:

[1] Papadias D, Tao Y, Fu G, Seeger B. Progressive skyline computation in database systems. ACM Trans. on Database Systems, 2005, 30(1):41−82. [doi: 101145/1061218.1061320]

[2] Sharifzadeh M, Shahabi C, Kazemi L. Processing spatial skyline queries in both vector spaces and spatial network databases. ACM Trans. on Database Systems, 2009,34(3):Article 14. [doi: 10.1145/1567274.1567276]

[3] Chen L, Lian X. Efficient processing of metric skyline queries. IEEE Trans. on Knowledge and Data Engineering, 2009,21(3): 351−365. [doi: 10.1109/TKDE.2008.146]

[4] Dellis E, Seeger B. Efficient computation of reverse skyline queries. In: Proc. of the VLDB Conf. ACM, 2007. 291−302.

[5] Lian X, Chen L. Monochromatic and bichromatic reverse skyline search over uncertain databases. In: Proc. of the ACM SIGMOD Conf. Vancouver: ACM, 2008. 213−226. [doi: 10.1145/1376616.1376641]

[6] Gao Y, Liu Q, Zheng B, Chen G. On efficient reverse skyline query processing. Expert Systems with Applications, 2014,41(7): 3237−3249. [doi: 10.1016/j.eswa.2013.11.012]

[7] Liu Q, Gao Y, Chen G, Li Q, Jiang T. On efficient reverse k-skyband query processing. In: Proc. of the DASFAA Conf. Busan: Springer-Verlag, 2012. 544−559. [doi: 10.1007/978-3-642-29038-1_39]

[8] Miao X, Gao Y, Chen L, Chen G, Li Q, Jiang T. On efficient k-skyband query processing over incomplete data. In: Proc. of the DASFAA Conf. Wuhan: Springer-Verlag, 2013. 424−439. [doi: 10.1007/978-3-642-37487-6_32]

[9] Feng X, Gao Y, Jiang T, Chen L, Miao X, Liu Q. Parallel k-skyband computation on multicore architecture. In: Proc. of the APWeb. LNCS 7808, Sydney: Springer-Verlag, 2013. 827−837. [doi: 10.1007/978-3-642-37401-2_79]

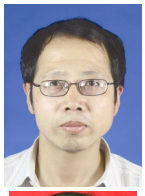
[10] Zhang B, Jiang T, Yue G, Li G. Mutual skyline queries based on reusing technology. Journal of Huazhong University of Sicence and Technology (Nature Edition), 2010,38(7):111−115 (in Chinese with English abstract). [doi: 10.13245/j.hust.2010.07.026]

[11] Wang G, Xin J, Chen L, Liu Y. Energy-Efficient reverse skyline query processing over wireless sensor networks. IEEE Trans. on Knowledge and Data Engineering, 2012,24(7):1259−1275. [doi: 10.1109/TKDE.2011.64]

[12] Tao Y, Xiao X, Pei J. Efficient skyline and top-k retrieval in subspaces. IEEE Trans. on Knowledge and Data Engineering, 2007,19(8):1072−1088. [doi: 10.1109/TKDE.2007.1051]

附中文参考文献:

[10] 张彬,蒋涛,乐光学,李国徽.基于重用技术的相互 Skyline 查询算法.华中科技大学学报(自然科学版),2010,38(7):111−115.



蒋涛(1973－),男,湖南衡阳人,博士,副教授,主要研究领域为时空数据库技术,Skyline 计算.

张彬(1978－),女,副教授,主要研究领域为数据库查询技术.

余法红(1977－),男,博士,讲师,主要研究领域为人工智能,复杂网络.

柳晴(1988－),男,博士生,主要研究领域为 Skyline 查询.

周傲英(1965－),男,博士,教授,博士生导师,CCF 高级会员,主要研究领域为空间数据库,数据挖掘,P2P 计算和系统,Web 搜索.