doi:10.3772/j.issn.1002-0470.2018.03.002

基于社交的空间文本 skyline 查询①

陈子军②*** 郭莎莎*** 刘文远*** 刘永山***

(*燕山大学信息科学与工程学院 秦皇岛 066004)

(**河北省计算机虚拟技术与系统集成重点实验室 秦皇岛 066004)

摘 要 将社交信息应用到空间文本 skyline 查询中,提出了基于社交的空间文本 skyline(SSTS)查询。SSTS 查询中 skyline 对象的选择依赖于三个方面:与查询用户之间的距离、与查询关键字的文本相关性和在用户中的流行性。该查询引入了新型的函数计算它的社交相关性。为了提高查询者的满意度,扩展了 SSTS 查询,提出了受限的基于社交的空间文本 skyline(constrained SSTS, CSSTS)查询,同时引入了一个新颖的概念受限 skyline。针对每一种查询,应用了裁剪策略和终止条件,提高了查询速度。最后,通过实验验证了所提方法的有效性。

关键词 社交网络, skyline 查询, 文本相关性, 社交相关性, 受限 skyline

0 引 言

对于查询者而言,关键字和位置是两个主要的考虑因素。两种主要的偏好查询是 top-k 和 skyline 查询。Top-k 查询[1,2]已经被广泛研究。然而,top-k 查询中维的权重有时并不好设定。此时,skyline 查询是一个很好的选择。

Skyline 查询从给定的数据集中返回那些不被其它任何对象支配的对象,这些对象称为 skyline 对象。文献[3]提出了文本相关的空间 skyline 查询,它返回的是空间邻近、文本相关的满足条件的兴趣点。然而,随着社交网络的发展,例如 Twitter、微博等,社交网络能够帮助提高查询结果的质量。文献[4]研究了社交对推荐的影响,它的主要思想是朋友对查询用户有一定的影响。文献[5]首次提出了地理社交 skyline 查询,返回社交网络的用户集合 D 中与用户 u 紧密相关且与它的位置接近的用户。

上面的skyline查询只考虑了空间、文本和社交三个标准中的两个标准。本文将社交影响添加到空间文本 skyline 查询中,提出了基于社交的空间文本 skyline(social-based spatial-textual skyline, SSTS)查询。同时,受文献[6]的启发,提出了受限 skyline 的概念和受限的基于社交的空间文本 skyline(constrained SSTS, CSSTS)查询。

1 相关工作

Börzsönyi 等人[7]首次在关系数据库中引入了 skyline 操作。自此以后,skyline 操作在数据库领域引起了广泛的研究。Kossmann 等人[8]提出了用最近邻 NN 算法处理 skyline 查询。文献[9]提出了 Branch and Bound Skyline 算法,它是通过结点和对象到原点的最小距离来遍历 R-tree,从而得到 skyline 对象。前面提到的算法都是用于处理静态 skyline 的查找。

然而,随着推荐旅游景点、餐馆等应用的出现,动态 skyline 查询表现出了它的优势。动态 skyline

① 河北省自然科学基金(F2017203019)资助项目。

② 男,1971 年生,博士,教授;研究方向:空间数据库,时空数据库等;联系人, E-mail: zjchen@ysu.edu.cn

(收稿日期:2017-10-16)

的计算由对象派生维的属性值决定,它们依赖于查询点的输入。文献[10]引入了空间 skyline 查询的概念。空间 skyline 是由多个派生的空间属性值决定的,每一个属性值等于一个对象到一个查询点的距离。文献[3]提出了三种模型研究空间文本 skyline,并对这三种模型进行了详细的分析。文献[11]主要研究的是基于范围的空间文本 skyline 查询,通过给定一个范围,返回范围内包含的所有 skyline 对象。

随着社交网络的发展,社交在查询中被研究。文献[12]研究了基于用户签到行为的兴趣点推荐,并通过实验验证了所提出的兴趣点推荐算法比其它的推荐效果好。文献[2]考虑了朋友关系和兴趣爱好,提出了已知社交和文本的 top-k 位置查询。然而,社交在 skyline 查询中有很少的研究。文献[5]首次将社交应用到空间 skyline 查询中,提出了地理社交 skyline 查询。前面的动态 skyline 查询只是考虑空间距离、文本相关性和社交相关性三者中的两者。据本文作者所知,在 skyline 查询中很少有人将空间、文本和社交三者结合起来,并且能够可以用于兴趣点和受限兴趣点的查找。因此,本文考虑 skyline 中的空间、文本和社交属性,提出了 SSTS 和 CSSTS 查询。

2 问题描述

基于社交的空间文本 skyline(SSTS)查询 q 用三元组(u, l, T)表示,其中,$q.u$ 是查询用户,$q.l$ 是查询点的位置,$q.T$ 是关键字集合。给定数据集 D,任一兴趣点 $o \in D$ 用三元组(l, F, T)表示,其中 $o.l$ 表示兴趣点 o 的空间位置信息,通过有序数对(x_l, y_l)表示,x_l 表示纬度,y_l 表示经度;$o.F$ 表示兴趣点 o 的 Fans 集合,该集合由在 o 处签到的用户构成;$o.T$表示兴趣点 o 的文本信息,即为一组关键字的集合,$o.T=\{t_1, t_2, \cdots, t_{|o.T|}\}$。每个关键字 $t \in o.T$ 都带有一个权重 $w_t(o)$。本文采用信息检索领域的词频-逆文件频率(term frequency-inverse document frequency, TF-IDF)模型计算关键字的权重,即

$$w_t(o) = tf(t, o.T) \cdot idf(t, D) \tag{1}$$

其中,词频 $tf(t, o.T)$为 t 出现在 $o.T$ 中的频率,即为 $\frac{count(t, o.T)}{|o.T|}$;逆文档频率 $idf(t, D)$为数据集 D 的大小与包含关键字 t 的兴趣点个数之间比例的对数,一般通过 $\lg \frac{|D|}{count(t, D)}$ 计算。例如,图 1 中的数据集 $D=\{o_1, o_2, o_3, o_4, o_5\}$。以兴趣点 o_1 为例,它包含了两个关键字{cozy, expensive},通过式(1)可以计算出这两个关键字的权重:$w_{cozy}(o_1) = \frac{1}{2} \cdot \lg\left(\frac{5}{1}\right) = 0.349$;$w_{expensive}(o_1) = \frac{1}{2} \cdot \lg\left(\frac{5}{2}\right) = 0.199$。

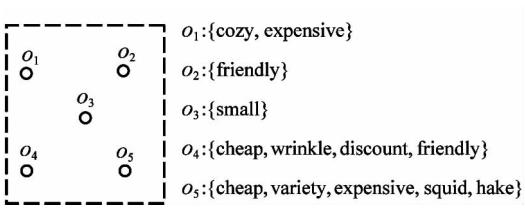

图 1 TF-IDF 模型的一个例子

根据查询的需要,首先给出查询点 q 与兴趣点 o 之间的空间文本距离和社交相关性的公式。

空间文本距离:

$$std(q, o) = \frac{d(q, o)}{Q_T(q, o)} \tag{2}$$

其中,$d(q, o)$表示兴趣点 o 到查询点 q 的欧式距离,$Q_T(q, o)$表示兴趣点 o 与查询点 q 之间的文本相关性。

$$Q_T(q, o) = (\Pi_{t_i \in q.T} w_{t_i}(o))^{\frac{1}{|q.T|}} \tag{3}$$

其中,o 的关键字集合为 $o.T=\{t_1, t_2, \cdots, t_{|o.T|}\}$,$q$ 的关键字集合 $q.T=\{\psi_1, \psi_2, \cdots, \psi_n\}$,$w_{t_i}(o)$表示 t_i 在 $o.T$ 中的权重,若 $o.T$ 中不包含 t_i,则 $w_{t_i}(o)$ 取一很小的常数0.01,$|q.T|$表示 q 的关键字个数。

社交相关性:

$$Q_S(q, o) = 1 - \frac{|o.F|}{N_U} \cdot \frac{|q.u.Friend \cap o.F|}{|q.u.Friend|} \tag{4}$$

其中,N_U 表示社交关系网中的用户总数,$|o.F|$表示兴趣点 o 的 Fans 个数,$|q.u.Friend|$表示查询用户 $q.u$ 的朋友个数,$|q.u.Friend \cap o.F|$表示查询用户 $q.u$ 的朋友在 o 处签到的朋友个数。

定义 1 基于社交的空间文本支配: 给定查询点

q,如果两个兴趣点 o_i 和 o_j 满足 $std(q, o_i) \leqslant std(q, o_j)$ 且 $Q_S(q, o_i) \leqslant Q_S(q, o_j)$,并且至少有一个满足小于条件,就称兴趣点 o_i 基于社交的空间文本支配兴趣点 o_j,简称为 o_i 支配 o_j,记为 $o_i \prec_{SST} o_j$。

定义 2 基于社交的空间文本 skyline:基于社交的空间文本 skyline 是一个包含数据集 D 中那些不能够被其它兴趣点支配的兴趣点的集合。即,$o \in$ SSTS 当且仅当 $\forall o' \in D$, $o' \not\prec_{SST} o$。

定义 3 基于社交的空间文本 skyline 查询:基于社交的空间文本 skyline 查询是指由数据集 D 中返回基于社交的空间文本 skyline。

3 索引结构

本文使用的索引结构为 SNIR-Tree[13],该索引结构包含了社交信息、文本信息和空间信息。

SSTS 查询中的 SNIR-Tree 中的每个叶子结点包含的兴趣点对象由三元组(id, $location$, F)组成,id 是数据集 D 中兴趣点对象 o 的编号,$location$ 是 o 的坐标,F 是 o 的 Fans 集合。每个叶子结点包含一个指针 InvertFile,指向该叶子结点的文本倒排表。文本倒排表中的关键字集合是该结点包含的所有兴趣点对象的关键字集合的并集,对于每一个关键字 t,存在形如(o, $w_t(o)$) 的列表,其中,o 是包含关键字 t 的兴趣点对象,$w_t(o)$ 代表 o 中关键字 t 的权重。

每个非叶子结点包含的实体由元组(cp, $rectangle$, F)组成,cp 是指向该结点的孩子结点的指针,$rectangle$ 是包含所有孩子结点的最小边界矩形(minimum bounding rectangle,MBR)框,F 是该结点所有孩子结点的 Fans 集合的并集。每个非叶子结点包含一个指针 InvertFile,InvertFile 是指向该结点的文本倒排表的指针。文本倒排表中的关键字集合是该结点包含的所有孩子关键字集合的并集。对于每一个关键字 t,列表(e, $w_t(e)$)被生成,其中,e 是一个包含 t 的孩子结点,$w_t(e) = \max_{e' \in CHILD(e)} w_t(e')$。

图 2 展示了 SSTS 查询中的 SNIR-Tree 的结构,表 1 和表 2 分别展示了非叶子结点和叶子结点的文本倒排表。

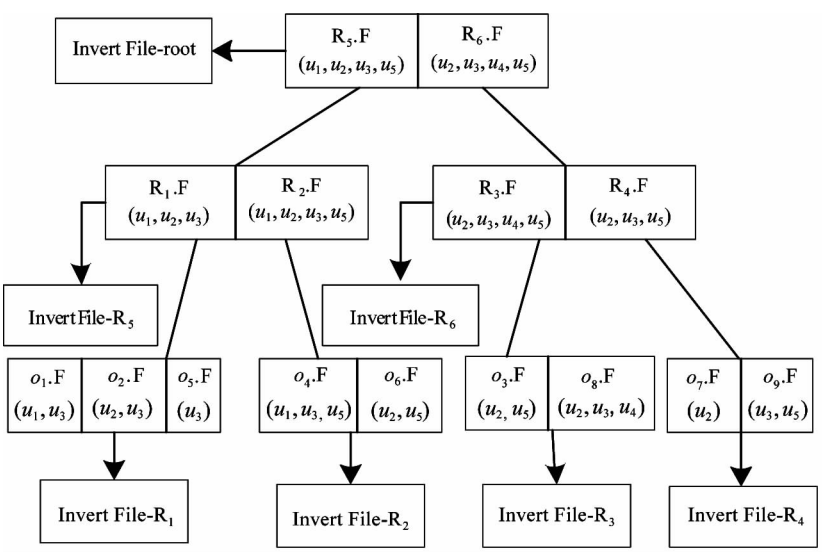

图 2 SSTS 查询中的 SNIR-Tree

表 1 非叶子结点文本倒排表

InvertFile-root	InvertFile-R_5	InvertFile-R_6
a:(R_5,7),(R_6,3)	a:(R_1,7),(R_2,4)	a:(R_3,1),(R_4,3)
b:(R_5,5),(R_6,3)	b:(R_1,5),(R_2,4)	b:(R_3,3)
c:(R_5,5),(R_6,7)	c:(R_1,5),(R_2,4)	c:(R_3,4),(R_4,7)
d:(R_5,1),(R_6,1)	d:(R_1,1)	d:(R_3,1),(R_4,1)

表 2　叶子结点文本倒排表

InvertFile-R_1	InvertFile-R_2	InvertFile-R_3	InvertFile-R_4
a:(o_1,5),(o_2,7)	a:(o_4,4)	a:(o_3,1)	a:(o_9,3)
b:(o_5,5)	b:(o_6,4)	b:(o_3,1),(o_8,3)	
c:(o_1,5),(o_5,5)	c:(o_4,4),(o_6,3)	c:(o_3,4),(o_8,3)	c:(o_7,7),(o_9,3)
d:(o_2,1)		d:(o_3,1)	d:(o_7,1)

4　查询算法

本节给出 SSTS 查询的裁剪方法、终止条件及算法处理的过程。

定义 4 *MinDist* 距离[14]:n 维欧式空间中的点 p 到同一空间内某最小边界矩形框 $R(s, t)$ 的最小距离 *MinDist*,表示为 $MinDist(p, R(s, t))$:

$$MinDist(p,R) = \sum_{i=1}^{n} | p_i - r_i |^2,$$

$$r_i = \begin{cases} s_i, & p_i < s_i \\ t_i, & p_i > t_i \\ p_i, & \text{其他} \end{cases} \tag{5}$$

为了更加容易地理解后面提出的裁剪策略,介绍一些相关的公式。

结点的文本相关性:

$$Q_T(q, e) = \left(\prod\nolimits_{t_i \in q.T} w_{t_i}(e)\right)^{\frac{1}{|q.T|}} \tag{6}$$

其中,结点 e 中的关键字集合为 $e.T = \{t_1, t_2, \cdots, t_{|e.T|}\}$,查询点 q 的关键字集合 $q.T = \{\psi_1, \psi_2, \cdots, \psi_n\}$,$w_{t_i}(e)$ 表示 t_i 在 $e.T$ 中的权重,如果 $e.T$ 中不包含 t_i,则 $w(t_i, eT)$ 为一常数0.01,$|q.T|$ 表示 q 的关键字个数。

结点的空间距离: $d(q,e) = MinDist(q,e)$ (7)

结点的空间文本距离: $std(q,e) = \dfrac{d(q,e)}{Q_T(q,e)}$ (8)

结点的社交相关性:

$$Q_S(q,e) = 1 - \frac{|e.F|}{N_U} \cdot \frac{|q.u.Friend \cap e.F|}{|q.u.Friend|} \tag{9}$$

其中,$|e.F|$ 表示结点 e 的 Fans 集合大小,即结点 e 的孩子的 Fans 集合的并集元素个数,$|q.u.Friend \cap e.F|$ 表示查询用户 u 的朋友在 e 处签到的朋友个数。

定理 1:给定一个查询点 q,对于结点 e 和它的任意孩子 e',有 $Q_T(q,e') \leqslant Q_T(q, e)$。

证明:由 SSTS 查询中的 SNIR-Tree 的性质可知,关键字 t 在 e 中的权重是 e 所有的孩子中 t 的最大权重,即 $w_t(e) = \max\limits_{e' \in CHILD(e)} w_t(e')$,因此,$w_t(e') \leqslant w_t(e)$。由于 $e.T = \bigcup\limits_{e' \in CHILD(e)} e'.T$,对于 q 的任意关键字 ψ_i,有 $w_{\psi_i}(e') \leqslant w_{\psi_i}(e)$。根据式(6)可知:$Q_T(q, e') \leqslant Q_T(q, e)$。□

定理 2:给定查询点 q,对于结点 e 和它的任意孩子 e',有 $std(q, e') \geqslant std(q, e)$。

证明:由定义 4 可以得到,$MinDist(q, e') \geqslant MinDist(q, e)$。同时,由定理 1 可得:$Q_T(q, e') \leqslant Q_T(q, e)$。因此有:$std(q,e') = \dfrac{d(q,e')}{Q_T(q,e')} = \dfrac{MinDist(q,e')}{Q_T(q,e')} \geqslant \dfrac{MinDist(q,e)}{Q_T(q,e')} \geqslant \dfrac{MinDist(q,e)}{Q_T(q,e)} = std(q,e)$ □

定理 3:给定一个查询点 q,对于结点 e 和它的任意孩子 e',有 $Q_S(q, e) \leqslant Q_S(q, e')$。

证明:由 SSTS 查询中的 SNIR-Tree 的性质可知,$e.F = \bigcup\limits_{e' \in CHILD(e)} e'.F$。故 $e'.F \subseteq e.F$,则 $|q.u.Friend \cap e'.F| \leqslant |q.u.Friend \cap e.F|$。则有:

$$\frac{|e'.F|}{N_U} \cdot \frac{|q.u.Friend \cap e'.F|}{|q.u.Friend|} \leqslant \frac{|e.F|}{N_U} \cdot \frac{|q.u.Friend \cap e.F|}{|q.u.Friend|}。$$

根据式(9)可知:$Q_S(q, e) \leqslant Q_S(q, e')$。□

4.1　裁剪方法

对于基于 SNIR-Tree 的 SSTS 的查询算法,本文

利用两个优先队列。一个队列使空间文本距离 $std(q,\cdot)$ 小的对象或结点优先出队(若两个对象或结点的 $std(q,\cdot)$ 大小相等,则 $Q_S(q,\cdot)$ 小的对象或结点优先出队),同时结点的孩子入队;另一个队列使社交相关性 $Q_S(q,\cdot)$ 小的对象或结点优先出队(若两个对象或结点的 $Q_S(q,\cdot)$ 大小相等,则 $std(q,\cdot)$ 小的对象或结点优先出队),同时结点的孩子入队。在此过程中,提出了裁剪方法。

详细的裁剪规则如下文所示,裁剪规则中的证明不包含两个对象或结点的 $std(q,\cdot)$ 与 $Q_S(q,\cdot)$ 均相等的情形,这种情形在裁剪算法中单独处理。

裁剪规则 1[15]:在按照空间文本距离 $std(q,\cdot)$ 的非递减顺序出队列的优先队列中,设 p 为出队列的对象,当前的结果集 R_1 中最后一个 skyline 对象为 R_{last}。若 $Q_S(q,R_{\text{last}})>Q_S(q,p)$,则 p 为 skyline 对象,不可以被裁剪;若 $Q_S(q,R_{\text{last}}<Q_S(q,p)$,则裁剪 p。

证明:根据优先队列的性质可知,$std(q,R_{\text{last}})\leqslant std(q,p)$。由于 R_{last} 是 R_1 中 $std(q,\cdot)$ 最大的对象,因此 R_{last} 是 R 中 $Q_S(q,\cdot)$ 最小的对象,否则,R_{last} 将被 R_1 中某个对象支配,矛盾。同理可知,若 $Q_S(q,_{\text{last}})>Q_S(q,p)$,根据定义 2 可知,$p$ 为 skyline 对象,因此,p 不可以被裁剪。若 $Q_S(q,R_{\text{last}})<Q_S(q,p)$,根据定义 1 可知,$R_{\text{last}}\prec_{SST}p$,因此,$p$ 可以被裁剪。□

裁剪规则 1 与文献[15]中的方法类似,都是利用结果集中的最后一个对象就可以判断 p 是否为 skyline 对象。

裁剪规则 2:在按照空间文本距离 $std(q,\cdot)$ 的非递减顺序出队列的优先队列中,设 p 为出队列的结点,当前的结果集 R_1 中最后一个 skyline 对象为 R_{last}。若 $Q_S(q,R_{\text{last}})<Q_S(q,p)$,则裁剪 p。

证明:由定理 2 可知,对于 p 的任意孩子 e,$std(q,e)\geqslant std(q,p)$,由优先队列的性质可知,$std(q,p)\geqslant std(q,R_{\text{last}})$,则 $std(q,e)\geqslant std(q,R_{\text{last}})$。由定理 3 可知,对于 p 的任意孩子 e,$Q_S(q,e)\geqslant Q_S(q,p)$,若 $Q_S(q,R_{\text{last}})<Q_S(q,p)$,则 $Q_S(q,R_{\text{last}})<Q_S(q,e)$。同理,可以证得 e 的孩子若为对象则会被 R_{last} 支配,依此类推,可知对于在 p 的最小边界矩形框 $p.\text{MBR}$ 中的任意对象 p',都有 $R_{\text{last}}\prec_{SST}p'$。因此,$p$ 可以被裁剪。□

裁剪规则 3[15]:在按照社交相关性 $Q_S(q,\cdot)$ 的非递减顺序出队列的优先队列中,设 p 为出队列的对象,当前的结果集 R_2 中最后一个 skyline 对象为 R_{last}。若 $std(q,R_{\text{last}})>std(q,p)$,则 p 为 skyline 对象,不可以被裁剪;若 $std(q,R_{\text{last}})<std(q,p)$,则裁剪 p。

证明:证明过程与裁剪规则 1 类似。

裁剪规则 4:在按照社交相关性 $Q_S(q,\cdot)$ 的非递减顺序出队列的优先队列中,设 p 为出队列的结点,当前的结果集 R_2 中最后一个 skyline 对象为 R_{last}。若 $std(q,R_{\text{last}})<std(q,p)$,则裁剪 p。

证明:证明过程与裁剪规则 2 类似。

4.2 算法描述

利用上面的裁剪规则,可以提高查询效率。算法 1 描述了查询处理的过程,算法 2 描述了按照 $std(q,\cdot)$ 的非递减顺序出队列计算 skyline 对象的裁剪算法 SkylinePrunes1(p),算法 3 描述了按照 $Q_S(q,\cdot)$ 的非递减顺序出队列计算 skyline 对象的裁剪算法 SkylinePrunes2(p)。

首先通过图 3 所示的例子介绍一下查询终止的过程。对象 $p_1\sim p_{11}$ 按照 $std(q,\cdot)$ 与 $Q_S(q,\cdot)$ 值的大小分布在如图 3 所示的坐标系中,为了容易描述查询处理的过程,我们使用扫描线的概念,扫描线是平行于 x 轴或 y 轴的直线,例如图 3 所示的扫描线 1 和扫描线 2。随着扫描线 1 的推进,按照 $std(q,\cdot)$ 的非递减顺序出队列的优先队列进行 skyline 对象的查找,得到当前的结果集为 R_1;随着扫描线 2 的推进,按照 $Q_S(q,\cdot)$ 的非递减顺序出队列的优先队列进行 skyline 对象的查找,得到当前的结果集为 R_2。每条扫描线在进行扫描之前,都会进行 R_1 与 R_2 中末尾元素的比较,只要不存在 $std(q,\cdot)$ 与 $Q_S(q,\cdot)$ 值均对应相等的末尾元素,扫描线就要继续推进,否则,扫描线就可以终止,称 $std(q,\cdot)$ 与 $Q_S(q,\cdot)$ 值对应相等的元素为终止对象,图 3 中扫描线 1 与扫描线 2 分别在访问到 skyline 对象 p_6 时,两条扫描线不需要再推进,p_6 即为终止对象,查询可以终止,得到结果集 $R=R_1\cup R_2=\{p_1,p_3,p_6\}\cup\{p_{11},p_8,p_6\}$

$=\{p_1, p_3, p_6, p_8, p_{11}\}$。

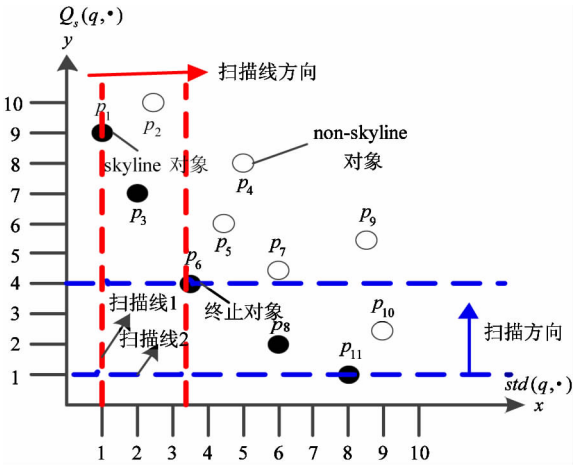

图 3　一个 SSTS 查询的例子

给定按照 $std(q,\cdot)$ 的非递减顺序出队列的优先队列 *Queue*1 和得到的最终结果集 R_1,按照 $Q_S(q,\cdot)$ 的非递减顺序出队列的优先队列 *Queue*2 和得到的最终结果集 R_2。显然有 $R_1=R_2$。

定理 4:设 R_1 和 R_2 中分别存在对象 o_1 和 o_2,R_1 中在 o_1 之前进入的对象和 o_1 构成的集合为 R_1',R_2 中在 o_2 之前进入的对象和 o_2 构成的集合为 R_2',R_1 和 R_2 中在 o_1 和 o_2 之后进入的相邻对象分别为 o_1' 和 o_2'。如果 $std(q,o_1)=std(q,o_2)$,$Q_S(q,o_1)=Q_S(q,o_2)$,则当满足下面两个条件之一时,$R_1'\cup R_2'=R_1=R_2$,即查询结果为 $R_1'\cup R_2'$。

(1) $std(q,o_1')\neq std(q,o_1)$ 或 $Q_S(q,o_1')\neq Q_S(q,o_1)$;

(2) $std(q,o_2')\neq std(q,o_1)$ 或 $Q_S(q,o_2')\neq Q_S(q,o_1)$。

证明:(1)若 $std(q,o_1')\neq std(q,o_1)$,则有 $std(q,o_1')>std(q,o_1)$,此时必有 $Q_S(q,o_1')\neq Q_S(q,o_1)$,否则有 $o_1\prec_{SST}o_1'$,与 $o_1'\in R_1$ 矛盾。若 $Q_S(q,o_1')\neq Q_S(q,o_1)$,则必有 $std(q,o_1')\neq std(q,o_1)$,否则有 $o_1\prec_{SST}o_1'$,或 $o_1'\prec_{SST}o_1$,产生矛盾。因此,得出 $std(q,o_1')\neq std(q,o_1)$ 并且 $Q_S(q,o_1')\neq Q_S(q,o_1)$,由于有 $std(q,o_1')>std(q,o_1)$,则必有 $Q_S(q,o_1')<Q_S(q,o_1)$,否则有 $o_1\prec_{SST}o_1'$,与 $o_1'\in R_1$ 矛盾。因此,必有 $o_1'\in R_2'$。对于 R_1 中在 o_1' 之后进入的任意对象 o_1'',同理可知,必有 $o_1''\in R_2'$。因此,有 $R_1'\cup R_2'=R_1=R_2$。

(2) 证明过程与(1)类似。□

定理 5:设从 *Queue*1 出队列得到的部分结果集为 R_1',从 *Queue*2 出队列得到的部分结果集为 R_2',不存在 $o_1\in R_1'$,$o_2\in R_2'$,使得 $std(q,o_1)=std(q,o_2)$,$Q_S(q,o_1)=Q_S(q,o_2)$。设最后进入 R_1' 和 R_2' 的 skyline 对象分别为 o_{1_last} 和 o_{2_last},即将进入 R_1' 和 R_2' 的 skyline 对象分别为 o_1' 和 o_2'。则下面命题成立:

(1) 若 $std(q,o_1')\neq std(q,o_{2\_last})$ 或 $Q_S(q,o_1')\neq Q_S(q,o_{2\_last})$,则不存在 $o_2\in R_2'$,使得 $std(q,o_1')=std(q,o_2)$,$Q_S(q,o_1')=Q_S(q,o_2)$;

(2) 若 $std(q,o_2')\neq std(q,o_{1\_last})$ 或 $Q_S(q,o_2')\neq Q_S(q,o_{1\_last})$,则不存在 $o_1\in R_1'$,使得 $std(q,o_2')=std(q,o_1)$,$Q_S(q,o_2')=Q_S(q,o_1)$。

证明:(1)利用反证法。若 $std(q,o_1')\neq std(q,o_{2\_last})$ 或 $Q_S(q,o_1')\neq Q_S(q,o_{2\_last})$,假设存在 $o_2\in R_2'$,使得 $std(q,o_1')=std(q,o_2)$,$Q_S(q,o_1')=Q_S(q,o_2)$。若 $std(q,o_1')\neq std(q,o_{2\_last})$,则有 $std(q,o_2)\neq std(q,o_{2\_last})$,必有 $Q_S(q,o_2)\neq Q_S(q,o_{2\_last})$,否则有 $o_2\prec_{SST}o_{2\_last}$,或 $o_{2\_last}\prec_{SST}o_2$,产生矛盾。由 $Q_S(q,o_2)\neq Q_S(q,o_{2\_last})$ 可知,$Q_S(q,o_2)<Q_S(q,o_{2\_last})$,则必有 $std(q,o_2)>std(q,o_{2\_last})$,因此,$std(q,o_1)>std(q,o_{2\_last})$,得出 $o_{2\_last}\in R_1'$,产生矛盾。

(2) 证明过程与(1)类似。□

定义 5 预备终止状态:设从 *Queue*1 出队列得到的部分结果集为 R_1',从 *Queue*2 出队列得到的部分结果集为 R_2',若存在 $o_1\in R_1'$,$o_2\in R_2'$,使得 $std(q,o_1)=std(q,o_2)$,$Q_S(q,o_1)=Q_S(q,o_2)$。则称此时算法已经进入预备终止状态。

算法进入预备终止状态后,只要对即将进入 R_1' 或 R_2' 的 skyline 对象判断是否满足定理 4 的条件,一旦满足条件则算法终止。

定理 6:设从 *Queue*1 出队列得到的部分结果集为 R_1',从 *Queue*2 出队列得到的部分结果集为 R_2',算法刚开始运行时 R_1' 和 R_2' 均为空,设最后进入 R_1' 和 R_2' 的 skyline 对象分别为 o_{1_last} 和 o_{2_last},则若满足下面条件之一,查询算法进入预备终止状态:

(1) 对每个即将进入 R_1' 的 skyline 对象 o_1',如

果 R_2'非空,且 $std(q, o_1') = std(q, o_{2\_last})$,$Q_S(q, o_1') = Q_S(q, o_{2\_last})$;则 o_1'进入 R_1'之后,查询算法进入预备终止状态,否则 o_1'进入 R_1'之后,查询算法没有进入预备终止状态。

(2) 对每个即将进入 R_2'的 skyline 对象 o_2',如果 R_1'非空,且 $std(q, o_2') = std(q, o_{1\_last})$,$Q_S(q, o_2') = Q_S(q, o_{1\_last})$;则 o_2'进入 R_2'之后,查询算法进入预备终止状态。否则 o_2'进入 R_2'之后,查询算法没有进入预备终止状态。

证明:(1) 利用归纳法。当 $R_1' = \varnothing$, $R_2' = \{o_{2\_last}\}$,若 $std(q, o_1') = std(q, o_{2\_last})$,$Q_S(q, o_1') = Q_S(q, o_{2\_last})$,则 o_1'进入 R_1'后算法进入了预备终止状态。否则,o_1'进入 R_1'后,不存在 $o_1 \in R_1'$,$o_2 \in R_2'$,使得 $std(q, o_1) = std(q, o_2)$,$Q_S(q, o_1) = Q_S(q, o_2)$,即算法没有进入预备终止状态。

利用归纳,假设算法没有进入预备终止状态,若 $std(q, o_1') = std(q, o_{2\_last})$,$Q_S(q, o_1') = Q_S(q, o_{2\_last})$,则由定义 5 可知,$o_1'$进入 R_1'后算法进入了预备终止状态;否则由定理 5 可知,o_1'进入 R_1'后算法仍然不会进入预备终止状态。

(2) 证明过程与(1)类似。□

定理 7:设当从 *Queue*1 出队列得到的部分结果集为 R_1',从 *Queue*2 出队列得到的部分结果集为 R_2'时,查询算法进入预备终止状态,存在 $o_1 \in R_1'$,$o_2 \in R_2'$,使得 $std(q, o_1) = std(q, o_2)$,$Q_S(q, o_1) = Q_S(q, o_2)$。则当满足下面两个条件之一,查询算法可终止:

(1) 在 o_1 之后进入 R_1'的对象中,找到第一个满足条件 $std(q, o_1') \neq std(q, o_1)$或 $Q_S(q, o_1') \neq Q_S(q, o_1)$的对象 o_1',在 o_1'之前的对象都进入 R_1'后,算法可以终止。

(2) 在 o_2 之后进入 R_2'的对象中,找到第一个满足条件 $std(q, o_2') \neq std(q, o_2)$或 $Q_S(q, o_2') \neq Q_S(q, o_2)$的对象 o_2',在 o_2'之前的对象都进入 R_2'后,算法可以终止。

证明:(1)设在 o_1 之后进入 R_1'的对象中,o_1'为第一个满足条件 $std(q, o_1') \neq std(q, o_1)$或 $Q_S(q, o_1') \neq Q_S(q, o_1)$的对象,则对于在 o_1 之后并且在 o_1'之前进入 R_1'的任意对象 o_1'',都有 $std(q, o_1'') = std(q, o_1)$且 $Q_S(q, o_1'') = Q_S(q, o_1)$。在 o_1'之前的对象都进入 R_1'后,由定理 4 可知,查询结果集为 $R_1' \cup R_2'$,因此,算法可以终止。

(2) 证明过程与(1)类似。□

需要说明的是,对于定理 7,若找不到满足条件的对象,当优先队列中的所有 skyline 对象都进入 R_1'或 R_2'后,算法同样可以终止。

算法 1:查询算法

输入:查询点 q,SNIR-Tree *index*;

输出:结果集 R;

```
1:  R = ∅, R1 = ∅, R2 = ∅;//R 存放最终的 skyline 对象;R1 存放按照 std(q,·)由小到大出队列得到的 skyline 对象;R2 存放按照 QS(q,·)由小到大出队列得到的 skyline 对象
2:  Queue1←NewPriorityQueue1( );//初始化优先队列 Queue1(Queue1 按照 std(q,·)非递减顺序出队列)
3:  Queue2←NewPriorityQueue2( );//初始化优先队列 Queue2(Queue2 按照 QS(q,·)非递减顺序出队列)
4:  Queue1.Enqueue(index.RootNode,0,0);
5:  Queue2.Enqueue(index.RootNode,0,0);
6:  do{
7:      while not Queue1.IsEmpty( ) do
8:          e = Queue1.Dequeue( );
9:          if e 是对象 then
10:             if R1 = = ∅ || ! SkylinePrunes1(e, R1) then //调用函数 SkylinePrunes1( )用于判断 e 是否被裁剪,若返回 false,表示 e 为 skyline
11:                 R1←e;
12:                 break;
13:         else //e 是结点
14:             if ! SkylinePrunes1(e, R1) then //调用此函数用于判断 e 是否被裁剪,若返回 false,表示 e 不可以直接被裁剪掉
15:                 for e 中的每个孩子 p do
16:                     Queue1.Enqueue(p,
                        std(q, p), QS(q, p));
17:     end while
```

18: if(R_2. size( ) >0&&R_1. LastElement( ) = = R_2. LastElement( )) then break;
19: while not *Queue*2. IsEmpty( ) do
20: e = *Queue*2. Dequeue( );
21: if e 是对象 then
22: if R_2 = = ∅ ‖ ! SkylinePrunes2(e, R_2) then //调用函数 SkylinePrunes2(e)用于判断 e 是否被裁剪,若返回 false,表示 e 为 skyline
23: R_2←e;
24: break;
25: else //e 是结点
26: if ! SkylinePrunes2(e, R_2) then //调用此函数用于判断 e 是否被裁剪,若返回 false,表示 e 不可以直接被裁剪掉
27: for e 中的每个孩子 p do
28: *Queue*2. Enqueue(p, $std(q, p)$, $Q_S(q, p)$);
29: end while
30: } while (R_1. LastElement( )! = R_2. LastElement( ))
31. 如果 *Queue*1 非空,就不断出队列,若出队列的是结点,则该结点的孩子入队列,直至找到满足定理 7 终止条件的对象 o_1',同时,在 o_1'之前的 skyline 对象都进入了 R_1。
32: R←$R_1 \cup R_2$;
33: return R;

7 ~ 17 行是关键字为 $std(q, \cdot)$的优先队列 *Queue*1 出队列,直至找到一个 skyline 对象后退出循环,9 ~ 12 行表示出队列的元素是对象时所进行的操作,10、11 行说明 e 是首次出队列的对象或者对象 e 不能被裁剪规则裁剪掉,则 e 是 skyline 对象,故加入到结果集 R_1 中。13 ~ 17 行表示 e 是结点时所进行的操作,14 行说明如果结点 e 不能被裁剪规则裁剪掉,则继续执行,15 ~ 17 行表示将 e 中的所有孩子入队列。19 ~ 29 行是关键字为 $Q_S(q, \cdot)$的优先队列 *Queue*2 出队列,直至找到一个 skyline 对象后退出循环,与 7 ~ 17 行的操作类似。18 行和 30 行都是利用定理 6 在最早的时间发现算法已经进入预备终止状态,31 行利用定理 7 使算法进入终止状态。如果 *Queue*1 为空仍然没找到满足定理 7 终止条件的对象 o_1',表明此时 *Queue*1 中的所有 skyline 对象已经都进入 R_1,查询算法可以终止。32 行得到查询结果,由定理 4 可知,当查询算法进入终止状态后,结果集 R 为 R_1 和 R_2 的并集。33 行返回最终的结果集 R。

算法 2:裁剪算法 SkylinePrunes1(p, R_1)

输入:对象或结点 p,结果集 R_1;//R_1 是按照 $std(q, \cdot)$非递减顺序出队列得到的结果集
输出:p 的状态;
说明:p 的状态为布尔值,若返回 false,如果 p 为对象,则 p 为 skyline 对象,如果 p 为结点,则 p 的孩子需要进一步判断;若返回 true,如果 p 为对象,则 p 为非 skyline 对象,如果 p 为结点,则 p 包含的对象均为非 skyline 对象。

1: if p 是对象 then
2: if R_1 = = ∅ then return false;
3: else if ($Q_S(q, R_{last}) > Q_S(q, p)$) then//裁剪规则 1
4: return false;
5: else if ($Q_S(q, R_{last}) < Q_S(q, p)$) then//裁剪规则 1
6: return true;
7: else if($std(q, p)$ = = $std(q, R_{last})$) then
8: return false; //此时 $Q_S(q, p)$ = = $Q_S(q, R_{last})$
9: else return true;
10: else//p 是结点
11: if R_1 = = ∅ then return false;
12: else if (p 符合裁剪规则 2) then
13: return true;
14: else return false;

1 ~ 9 行表示 p 是对象时,裁剪算法所进行的操作,2 行表示如果 p 为首个出队列的对象,则 p 是 skyline 对象,不能够被裁剪,3、4 行说明如果 p 满足裁剪规则 1 中的情形 $Q_S(q, R_{last}) > Q_S(q, p)$,则 p 是 skyline 对象,5、6 行说明如果 p 满足裁剪规则 1 中的情形 $Q_S(q, R_{last}) < Q_S(q, p)$,则 p 是非 skyline 对象,可以被裁剪,7、8 行说明如果 p 计算出来的值

与结果集中的元素的值相同，则 p 不可以被裁剪，它是 skyline 对象，9 行表示其他情况下，p 可以被裁剪，它是非 skyline 对象。10～16 行表示 p 是结点时裁剪算法所进行的操作，11 行表示 R_1 为空时，p 不能被裁剪，它的孩子需要入队列，12、13 行说明如果 p 满足裁剪规则 2，p 能够被裁剪，它的孩子不需要入队列，14 行表示其他情况下，p 不能被裁剪，它的孩子需要入队列。

算法 3：裁剪算法 SkylinePrunes2(p, R_2)

裁剪算法 SkylinePrunes2(p)应用的裁剪规则是裁剪规则 3 和 4，与 SkylinePrunes1(p)的思想类似。

5　CSSTS 查询算法

5.1　问题描述

用户在进行 SSTS 查询时，返回的结果对象可能比较少，使得用户的选择减少，考虑到这种情况，本节提出了一个新颖的概念“受限 skyline”。CSSTS 查询返回的是 skyline 和受限 skyline 对象。首先通过一个例子来介绍一下受限 skyline。如图 4 所示，对象 o_1～o_{24} 按照 $std(q,\cdot)$ 与 $Q_S(q,\cdot)$ 值的大小分布在图 4 的坐标系中，根据定义 3 可知，SSTS 查询的结果集 $R=\{o_1,o_3,o_4,o_9,o_{19}\}$，分别找到 R 中 skyline 对象 $std(q,\cdot)$ 与 $Q_S(q,\cdot)$ 的最大值，设为 max_$std(q,\cdot)$ 和 max_$Q_S(q,\cdot)$，根据 max_$Q_S(q,\cdot)$ 和 max_$std(q,\cdot)$ 分别作出平行于 x 轴和 y 轴的直线 1 和直线 2，直线 1、直线 2 及 R 中的 skyline 对象围成的区域称为受限区域，记为 CZ，CZ 不包含边界，因此，本文使用虚线表示，如图 4 中的 CZ 所示，在该区域内计算出来的 skyline 对象，称为受限 skyline。本图中的受限 skyline 为 $CR=\{o_5,o_8,o_{10},o_{18}\}$。

定义 6 受限 skyline：给定 SSTS 查询中的数据集 D 和结果集 R，通过 R 中的对象计算 $Q_S(q,\cdot)$ 与 $std(q,\cdot)$ 的最大值，记为 max_$Q_S(q,\cdot)$ 和 max_$std(q,\cdot)$，根据它们的值分别做平行于派生维 $std(q,\cdot)$ 和 $Q_S(q,\cdot)$ 的两条直线 l_1 和 l_2，直线 l_1、l_2 及 R 中的对象两两顺次连接围成的区域，称为受限区域 CZ，CZ 不包含边界，计算 CZ 内所有 skyline 对象的集合，称为受限 skyline(constrain-skyline，CS)。

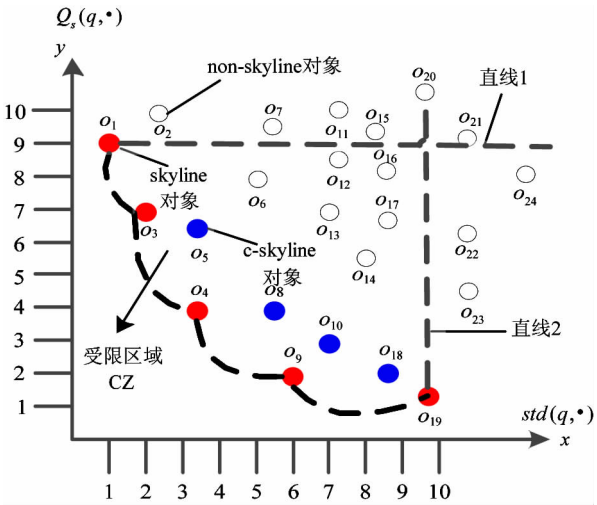

图 4　一个受限 skyline 的例子

定义 7 CSSTS 查询：给定查询点 q，数据集 D，阈值参数 k，当 SSTS 查询中的结果集个数少于 k 时，计算 CS，CSSTS 查询返回 SSTS 和 CS 的并集。

5.2　查询处理过程

首先，执行 SSTS 查询，得到结果集 R' 和优先队列的状态信息；其次，如果 R' 中的对象个数少于查询者设定的阈值参数 k，进行 CZ 的计算，得到结果集 CR；最后，CSSTS 查询返回 R' 与 CR 的并集。

详细的查询算法如算法 4 所示。

算法 4：CSSTS 查询算法

输入：查询点 q，阈值参数 k，SNIR-Tree $index$；

输出：结果集 R；

1：　$R=\varnothing$，$R'=\varnothing$，$CR=\varnothing$，$R_1=\varnothing$，$R_2=\varnothing$；//R 为最终的结果集；R' 为 SSTS 查询的结果集；CR 为 CS 对象的集合；R_1 存放按照 $std(q,\cdot)$ 由小到大出队列得到的 CS 对象；R_2 存放按照 $Q_S(q,\cdot)$ 由小到大出队列得到的 CS 对象

2：　maxSkylineSsd=0，maxSkylineSocial=0；//maxSkylineSsd 为 skyline 对象集合中最大的 $std(q,\cdot)$ 值，maxSkylineSocial 为 skyline 对象集合中最大的 $Q_S(q,\cdot)$ 值

3：　$Queue1\leftarrow$NewPriorityQueue1()；//初始化优先队 $Queue1$($Queue1$ 按照 $std(q,\cdot)$ 非递减顺序出队列)

4: *Queue*2←NewPriorityQueue2( );//初始化优先队 *Queue*2(*Queue*2 按照 $Q_S(q,\cdot)$非递减顺序出队列)

5: *Queue*3←NewPriorityQueue3( );//初始化优先队 *Queue*3(*Queue*3 按照 *std*$(q,\cdot)$非递减顺序出队列)

6: *Queue*4←NewPriorityQueue4( );//初始化优先队 *Queue*4(*Queue*4 按照 $Q_S(q,\cdot)$非递减顺序出队列)

7: *Queue*1 和 *Queue*2 分别是执行 SSTS 查询时的两个优先队列,将查询过程中这两个优先队列中被裁剪掉的对象分别添加到 *Queue*3 和 *Queue*4 中

8: R' = SSTS(q, *index*);//调用算法 1,此时的 *Queue*1, *Queue*2, *Queue*3 和 *Queue*4 已经按照 7 行的要求准备就绪

9: maxSkylineSsd = R'. max _ *std*$(q,\cdot)$;

10: maxSkylineSocial = R'. max _ $Q_S(q,\cdot)$;

11: if R'. size( ) < k then

12: *Queue*3. addAll(*Queue*1);

13: *Queue*4. addAll(*Queue*2);

14: do{

15: while not *Queue*3. IsEmpty( ) do

16: e = *Queue*3. Dequeue( );

17 if (*std*(q,e) < maxSkylineSsd&& $Q_S(q,e)$ < maxSkylineSocial) then

18: if e 是对象 then

19: if R_1 = = $\varnothing$ ‖ ! SkylinePrunes1(e, R_1) then //调用函数 SkylinePrunes1( )用于判断 e 是否被裁剪,若返回 false,表示 e 为 skyline

20: R_1←e;

21: break;

22: else //e 是结点

23: if ! SkylinePrunes1(e, R_1) then //调用此函数用于判断 e 是否被裁剪,若返回 false,表示 e 不可以直接被裁剪掉

24: for e 中的每个孩子 p do

25: *Queue*3. Enqueue(p, *std*(q,p), $Q_S(q,p)$);

26: if (*std*(q,e) ≥ maxSkylineSsd) then

27: *Queue*3. clear( );

28: break;

29: end while

30: if(R_2. size( ) > 0&&R_1. LastElement( ) = = R_2. LastElement( )) then break;

31: while not *Queue*4. IsEmpty( ) do

32: e = *Queue*4. Dequeue( );

33: if (*std*(q,e) < maxSkylineSsd&& $Q_S(q,e)$ < maxSkylineSocial) then

34: if e 是对象 then

35: if R_2 = = $\varnothing$ ‖ ! SkylinePrunes2(e, R_2) then //调用函数 SkylinePrunes2(e)用于判断 e 是否被裁剪,若返回 false,表示 e 为 skyline

36: R_2←e;

37: break;

38: else //e 是结点

39: if ! SkylinePrunes2(e, R_2) then //调用此函数用于判断 e 是否被裁剪,若返回 false,表示 e 不可以直接被裁剪掉

40: for e 中的每个孩子 p do

41: *Queue*4. Enqueue(p, *std*(q,p), $Q_S(q,p)$);

42: if (*std*(q,e) ≥ maxSkylineSocial) then

43: *Queue*4. clear( );

44: break;

45: end while

46: } while (R_1. LastElement( ) ! = R_2. LastElement( ))

47: CR←$R_1 \cup R_2$;

48: R←$R' \cup CR$;

49: return R;

1 ~ 6 行均是初始化操作。7 ~ 10 行表示执行 SSTS 查询得到的相关信息。8 行得到 SSTS 查询的结果集 R',9 行计算出 R'中最大的空间文本距离 maxSkylineSsd,10 行计算出 R'中最大的社交相关性 maxSkylineSocial。11 ~ 46 行用于计算 CS 对象。12 行将 *Queue*1 中的元素全部加入 *Queue*3,13 行将 *Queue*2 中的元素全部加入 *Queue*4,*Queue*3 和 *Queue*4 为处理 CS 对象的两个优先队列。15 ~ 29 行是按照

$std(q,\cdot)$出队列计算 CS 对象的操作,17 行表示出队列的元素 e 在 CZ 内,才能进行接下来的操作,18 ~21 行表示出队列的是对象时所进行的操作,19、20 行说明首次出队列的对象或者出队列的对象不能被裁剪掉,则它是 CS 对象,故加入结果集 R_1 中。22 ~25 行表示出队列的是结点时所进行的操作,23 行说明如果结点不能被裁剪掉,则继续处理,24、25 行表示将结点中的所有孩子入队列,进行进一步的处理。26 ~28 行说明如果出队列的元素 e 超出了 CZ,则将 *Queue*3 置空。31 ~45 行是按照 $Q_S(q,\cdot)$ 非递减顺序出队列计算 CS 对象的操作,与 15 ~29 行的操作类似,在此不再赘述。47 行将结果集 R_1 和 R_2 的元素进行合并,得到最终的 CS 对象集合 CR。49 行返回最终的结果集 R。

6　实验与结果分析

6.1　实验环境

实验环境为 AMD FX-Series FX-6330 3.62GHz 的 CPU,8GB 内存,使用 Windows 10 操作系统和 Eclipse集成开发环境。算法使用 Java 语言实现,JDK 版本为 1.8.0。

6.2　实验数据集

本实验使用数据集 Gowalla 和 Brightkite,并在这两个数据集中,为每个地点随机生成了文本信息。

6.3　实验结果分析

6.3.1　SSTS 查询

实验中为了测试裁剪的效果,在两个数据集上比较了使用裁剪策略(SSTSQ)和不使用裁剪策略(SSTSQ′)的方法。

(1) 关键字个数的影响

图 5 所示为关键字个数对运行时间和支配测试影响的对比图。查询用户朋友的个数为 20 ~ 30。由图 5(a)和图 5(b)可以看出,随着关键字个数的增加,运行时间整体呈上升趋势,这是因为,对于每一个关键字 $\psi_i \in q.T$,在倒排文件中为它创建的列表被访问。随着关键字个数的增加,在倒排文件中更多的列表被访问,因此,遍历文本倒排表的时间变长,从而计算文本相关性的时间变长,程序的运行时间变长。

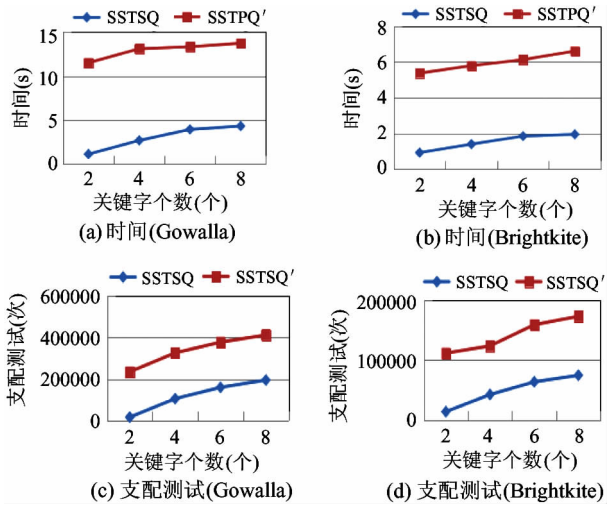

图 5　SSTS 查询中关键字个数的影响

由图 5(c)和图 5(d)可以看出,随着关键字个数的增加,支配测试整体呈上升趋势,这是因为随着关键字个数的增加,存在更多的候选 skyline 对象,需要更多的比较次数,因此,支配测试的次数增加。

(2) 查询用户的朋友个数的影响

图 6 所示为查询用户的朋友个数对运行时间和支配测试影响的对比图。关键字个数为 2。随着查询用户朋友个数的增加,运行时间整体呈上升趋势,支配测试整体也呈上升趋势。这是因为随着 u 的朋友个数的增加,可能更多的对象被 u 的朋友签到过,从而候选 skyline 对象增多,故需要比较的次数增加,运行时间变长。

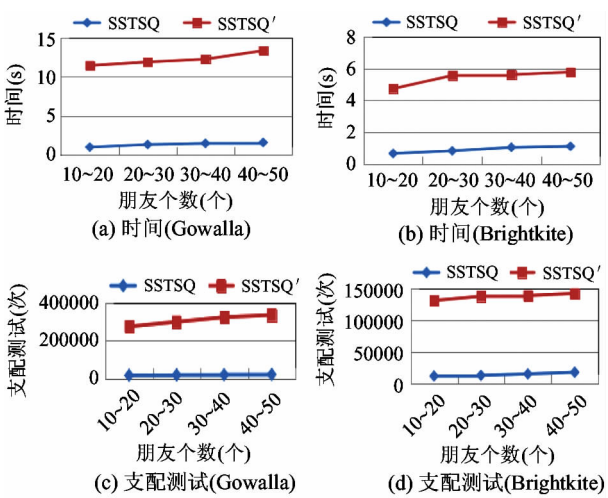

图 6　SSTS 查询中朋友个数的影响

6.3.2 CSSTS 查询

实验中为了测试 CSSTSQ 的效果,在两个数据集上比较了 CSSTSQ 和重新扫描数据集的查询 CSSTSQ′。

(1) 阈值参数 k 对时间的影响

图 7(a)和图 7(b)所示为阈值参数 k 对运行时间影响的对比图。关键字个数为 2,查询用户朋友的个数为 20 ~ 30。随着 k 的增加,运行时间整体呈上升趋势。这是因为,当 k 增加时,skyline 中对象的个数少于 k 的可能性增加,生成 CS 的可能性增加,计算 CS 需要花费一定的时间,从而程序的运行时间变长。同时,发现 CSSTSQ 的运行时间少于 CSSTSQ′的运行时间,这是因为,CSSTSQ 在计算 CS 时,保存了计算 skyline 结束后的优先队列的状态,从而可以继续扫描得到 CS。而 CSSTSQ′则是重新扫描计算 CS,明显运行时间要比 CSSTSQ 变长。

(2) 阈值参数 k 对结果集个数的影响

图 7(c)和图 7(d)所示为阈值参数 k 对结果大小影响的对比图。关键字个数为 2,查询用户朋友的个数为 20 ~ 30。随着 k 的增加,CSSTSQ 的结果集个数增加,这是因为,当 k 增加时,CS 的个数在增加。CS 的存在能够为查询者提供更多的选择机会,更大限度地满足查询者的需求。

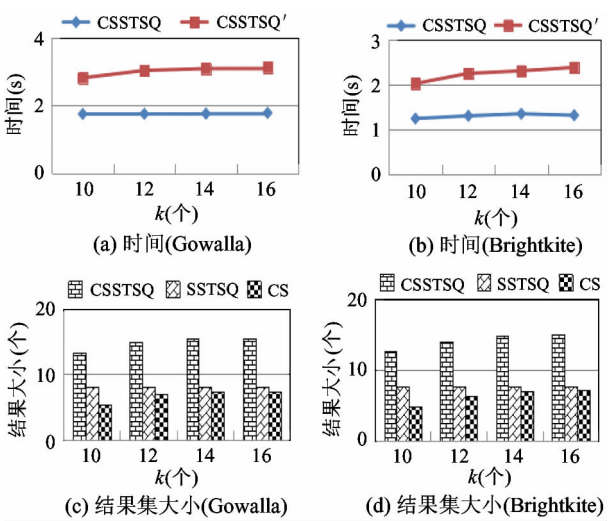

图 7 CSSTS 查询中阈值 k 的影响

7 结 论

本文将社交关系应用到空间文本 skyline 查询中,提出了基于社交的空间文本 skyline 查询,即 SSTS 查询。引入了新型的函数计算 SSTS 查询的社交相关性。为了极大地满足查询者的需求,提出了一个新颖的概念受限 skyline,进一步扩展了 SSTS 查询,提出了 CSSTS 查询。同时,在查询算法中提出了有效的裁剪策略和终止条件,提高了查询效率。实验表明,所提算法具有良好的运行效率。未来,考虑提出支持连续的基于社交的空间文本 skyline 查询的算法,同时,考虑将查询扩展到路网上。

参考文献

[1] Zhang C Y, Zhang Y, Zhang W J, et al. Inverted linear quadtree: efficient top k spatial keyword search[C]. In: Proceedings of the 29th IEEE International Conference on Data Engineering, Brisbane, Australia, 2013. 901-912

[2] 陈子军, 崔清娟, 刘文远. 已知社交和文本的 Top-k 位置查询[J]. 小型微型计算机系统, 2016, 37(10): 2199-2205

[3] Shi J M, Wu D M, Mamoulis N. Textually relevant spatial skylines[J]. *IEEE Transactions on Knowledge and Data Engineering*, 2016, 28(1): 224-237

[4] Ye M, Liu X J, Lee W C. Exploring social influence for recommendation: a generative model approach[C]. In: Proceedings of the 35th International ACM SIGIR Conference on Research and Development in Information Retrieval, Portland, USA, 2012. 671-680

[5] Emrich T, Franzke M, Mamoulis N, et al. Geo-social skyline queries[C]. In: Proceedings of the Database Systems for Advanced Applications - 19th International Conference, Bali, Indonesia, 2014. 77-91

[6] Kodama K, Iijima Y, Guo X, et al. Skyline queries based on user locations and preferences for making location-based recommendations[C]. In: Proceedings of the 2009 International Workshop on Location Based Social Networks, Seattle, USA, 2009. 9-16

[7] Börzsönyi S, Kossmann D, Stocker K. The skyline operator[C]. In: Proceedings of the 17th International Conference on Data Engineering, Heidelberg, Germany, 2001. 421-430

[8] Kossmann D, Ramsak F, Rost S. Shooting stars in the skyline : an online algorithm for skyline queries[C]. In: Proceedings of the 28th International Conference on Very

Large Data Bases, Hong Kong, China, 2012. 275-286

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

[10] Sharifzadeh M, Shahabi C. The spatial skyline queries [C]. In: Proceedings of the 32nd International Conference on Very Large Data Bases, Seoul, Korea, 2006. 751-762

[11] Li J N, Wang H Z, Li J Z, et al. Skyline for geo-textual data[J]. *GeoInformatica*, 2016, 20(3):453-469

[12] 任星怡, 宋美娜, 宋俊德. 基于用户签到行为的兴趣点推荐[J]. 计算机学报, 2017, 40(1): 28-51

[13] Wu D M, Li Y F, Choi B, et al. Social-aware top-k spatial keyword search[C]. In: Proceedings of IEEE 15th International Conference on Mobile Data Management, Brisbane, Australia, 2014. 235-244

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

[15] Regalado A, Goncalves M, Abad-Mota S. Evaluating skyline queries on spatial web objects[C]. In: Proceedings of the Database and Expert Systems Applications, Vienna, Austria, 2012. 416-423

Social-based spatial-textual skyline query

Chen Zijun[*,**], Guo Shasha[*,**], Liu Wenyuan[*,**], Liu Yongshan[*,**]

([*]School of Information Science and Engineering, Yanshan University, Qinhuangdao 066004)

([**]Key Laboratory for Computer Virtual Technology and System Integration of Hebei Province, Qinhuangdao 066004)

Abstract

Social information is applied to the spatial-textual skyline query, and a social-based spatial-textual skyline (SSTS) query is proposed. The selection of the skyline object in the SSTS query depends on three aspects: the spatial distance between the object and the query user, the textual relevance of the keywords and the popularity of users. One new function is introduced to the query to compute the social relevance. In order to improve the satisfaction of the query, the SSTS query is extended, and a constrained social-based spatial-textual skyline (CSSTS) query is proposed. In addition, a novel concept of constrained skyline is introduced. For each query, the pruning strategies and termination conditions are used to improve the query efficiency. In the end, an experiment is performed to verify the proposed method's effectiveness.

Key words: social networks, skyline query, textual relevance, social relevance, constrained skyline