软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2020,31(10):3197−3215 [doi: 10.13328/j.cnki.jos.005810]

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

支持 OR 语义的高效受限 Top-*k* 空间关键字查询技术*

潘 晓[1], 于启迪[1], 马 昂[1], 孙亚欣[1], 吴 雷[1,2], 郭景峰[2]

[1](石家庄铁道大学 经济管理学院,河北 石家庄 050043)

[2](燕山大学 信息科学与工程学院,河北 秦皇岛 066004)

通讯作者: 潘晓, E-mail: smallpx@stdu.edu.cn

摘 要: 近些年,随着定位系统和移动设备的普及,空间文本对象的数量日益庞大,基于位置的地理信息服务在人们的生活中发挥着越来越重要的作用.对于空间关键字查询搜索的研究亦如火如荼.然而,现有许多研究工作只适用于 AND 语义,支持 OR 语义的搜索研究相对较少.当用户放松对关键字匹配的要求时,支持 OR 语义的搜索技术显得尤为重要.针对这一问题,在聚集线性四分树的基础上,利用线性四分树上物理存储的 Morton 码与逻辑空间位置的对应性,提出了基于虚拟网格的 VGrid 算法.该算法可同时支持 OR 语义和 AND 语义.最后,通过在真实数据集上进行大量实验,验证了所提算法的有效性和高效性.

关键词: 倒排线性四分树;OR 语义;空间文本对象;空间关键字查询;移动计算

中图法分类号: TP311

中文引用格式: 潘晓,于启迪,马昂,孙亚欣,吴雷,郭景峰.支持 OR 语义的高效受限 Top-*k* 空间关键字查询技术.软件学报, 2020,31(10):3197−3215. http://www.jos.org.cn/1000-9825/5810.htm

英文引用格式: Pan X, Yu QD, Ma A, Sun YX, Wu L, Guo JF. Efficient algorithm of top-*k* spatial keyword search with OR semantics. Ruan Jian Xue Bao/Journal of Software, 2020,31(10):3197−3215 (in Chinese). http://www.jos.org.cn/1000-9825/5810.htm

Efficient Algorithm of Top-*k* Spatial Keyword Search with OR Semantics

PAN Xiao[1], YU Qi-Di[1], MA Ang[1], SUN Ya-Xin[1], WU Lei[1,2], GUO Jing-Feng[2]

[1](School of Economics and Management, Shijiazhuang Tiedao University, Shijiazhuang 050043, China)

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

Abstract: In recent years, with the popularization of positioning system and mobile devices, the numbers of spatial-textual objects increase extraordinarily. Location-based services using geographical information play a critical role in daily lives. Spatial keyword search has attracted more attention from academia and industry. However, many of the existing techniques can only be appliable on AND semantics. There is relatively less research supporting OR semantics. When the users do not require the exact keyword matching, the search technology that supports OR semantic is particularly important. To solve this problem, this study proposes a virtual grid-based query algorithm VGrid. VGrid is an aggregate linear quadtree (AIL) based algorithm, utilizing the easy transformation between the Morton codes and the spatial locations in the space. The algorithm can support both OR and AND semantics. Finally, a series of experiments is conducted on a real dataset, and the effectiveness and efficiency of the proposed algorithm are verified.

* 基金项目: 国家自然科学基金(61472340, 61303017); 河北省自然科学基金(F2018210109); 河北省教育厅重点项目(ZD2018040); 引进留学人员资助项目(C201822); 河北省基础研究团队项目(2019JT70803); 石家庄铁道大学第 4 届优秀青年科学基金(Z661250444); 国家级大学生创新创业训练计划(201710107006, 201710107007)

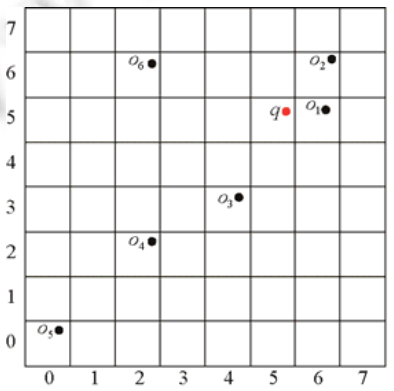
Foundation item: National Natural Science Foundation of China (61472340, 61303017); Natural Science Foundation of Hebei Province of China (F2018210109); Department of Education Key Project of Hebei Province (ZD2018040); Foundation of Introducing Overseas Student (C201822); Basic Research Team Project of Hebei Province (2019JT70803); The 4th Outstanding Youth Foundation of Shijiazhuang Tiedao University (Z661250444); College Innovative Training Program Foundation of China (201710107006, 201710107007)

收稿时间: 2018-03-21; 修改时间: 2018-07-19, 2018-09-27; 采用时间: 2019-01-08

Key words: inverted linear quad-tree; OR semantics; geo-textual object; spatial keywords query; mobile computing

随着智能手机和移动终端的普及,越来越多的应用中出现了地理位置与文本信息交融的现象[1,2].一方面,越来越多的场所,例如商店、饭店、游乐场等,都附加了与其地理位置相关的文本描述信息;另一方面,文本信息也通过地名、街道地址等特征与地理信息相关联.研究表明,大约有五分之一的互联网搜索与地理位置相关,包括地名、邮政编码等[3].在同时含有空间信息和文本信息的对象上进行空间文本查询(简称为空间关键字查询)是当前研究的热点问题之一[4],即给定查询点位置和文本信息,从大量空间文本对象中找到符合查询要求的对象.

Top-k 空间关键字搜索是空间关键字查询领域中非常重要的操作之一,其根据给定的评分函数在数据集中返回得分最高的 k 个对象[5−13].现有大部分空间关键字查询工作最先考虑的是满足用户所有文本要求和位置邻近性要求的对象,可称其为基于 AND 语义的查询.然而,基于 AND 语义的查询结果需完全匹配查询关键字,有时会使用户错失一些较好的选择.以图 1 为例,空间中的每个对象(即黑色点 o_1 至 o_6)均具有地理位置坐标和相应的关键字集合,其中每一个关键字对应一个权值,表示该关键字的重要程度(如用户评分).例如,对象 o_2 是一个电影院,对象 o_4 是一个综合性商场.假设用户初到该城市,在查询点 q(红色点)的位置查找一家电影院,并想喝杯咖啡.基于 AND 语义的 Top-2 查询将返回对象集合{o_4,o_5},因为仅此两个对象同时包含{cinema,coffee}两个关键字.观察图 1 中的对象,很明显,对象 o_2 和 o_1 均部分满足用户要求,且对象 o_2 和 o_1 在对应查询关键字上的权重均大于对象 o_4 和对象 o_5 在相应查询关键字上的权重(设权重越高越好),此外,q 到对象集合{o_1,o_2}的距离较到对象集合{o_4,o_5}更近.换句话说,如果用户更看重品质,基于 AND 语义查询返回的结果将错过更符合用户品质要求的对象(即对象 o_2 和对象 o_1),并且,该对象距离查询用户位置并不远.本文关注的基于 OR 语义的受限 Top-k 空间关键字查询可解决此类问题.



(a)

对象	位置	关键字及权重集合
o_1	(6.5, 5.8)	**coffee** (0.176)
o_2	(6.5, 6.5)	**cinema** (0.301)
o_3	(4.5, 3.5)	swim (0.151), **coffee** (0.088)
o_4	(2.6, 2.6)	**coffee** (0.044), **cinema** (0.075), library (0.119), swim (0.075)
o_5	(0.5, 0.5)	**coffee** (0.059), **cinema** (0.100), library (0.159)
o_6	(2.5, 6.5)	swim (0.301)

(b)

Fig.1 A simple example

图 1 查询示例

研究者针对 Top-k 空间关键字搜索问题提出了许多索引技术和查询算法[14].其中最普遍使用的是 IR-tree 索引.在该索引中,根据所有对象的地理位置建立一棵 R 树,每个节点关联一个倒排文件.当数据量较小时,IR-tree 处理效率较高,而当数据量增多时,其查询和更新效率就会下降;此外,由于在空间中只建立了一棵树,树的维护时间较长[15,16].为了解决上述不足,文献[13]提出了一种倒排线性四分树索引结构 IL-Quadtree.IL- Quadtree 为每个关键字都建立了一棵线性四分树,快速返回符合查询关键字要求的 k 个对象.当用户进行查询时,只访问查询关键字所对应的线性四分树,直接忽略掉那些不包含目标关键字的对象.然而,文献[16]仅满足 AND 语义要求.本文从查询 OR 语义出发,综合考虑用户的地理位置和查询关键字与空间文本对象匹配的情况,返回在用户空间范围要求内的空间文本相似性最高的前 k 个对象,即支持 OR 语义的受限 Top-k 空间关键字查询.

本文采用倒排聚集线性四分树 AIL 索引所有的空间文本对象.即在倒排文件中存储所有关键字,倒排文件

中的每个关键字指向包含该关键字的所有对象组成的聚集线性四分树.在 AIL 的基础上,本文提出了两种查询处理算法 VQuad 和 VGrid.受文献[16]的启发,VQuad 算法将整个空间区域看成一个虚拟四分树,基于此,虚拟四分树采用最小最佳优先原则返回同时支持 OR 语义与空间距离约束的结果.VQuad 算法是一个自顶向下的空间搜索算法.然而,线性四分树在物理上存储的并不是空间层次型结构,造成了在非空间层次结构上构建层次结构信息时,VQuad 重复遍历线性四分树.我们发现,线性四分树中的空间编码相当于将整个区域划分为虚拟网格,线性四分树中的对象位置与网格单元中的码值具有一一对应的关系.利用该对应关系,在网格中可以通过 $O(1)$ 的时间复杂度访问任意单元的邻近单元,避免了线性四分树的重复遍历.基于上述想法,本文提出了一种基于虚拟网格的高效查询算法 VGrid.

综上所述,本文贡献总结如下.

- 在倒排聚集线性四分树的基础上,提出了一种支持 OR 语义的基于虚拟四分树的 Top-k 空间关键字搜索算法 VQuad.
- 从线性四分树中空间位置的编码特点出发,提出了一种基于虚拟网格遍历的高效 OR 语义查询的 Top-k 空间关键字搜索算法 VGrid.
- 在真实数据集上进行了大量的实验,验证了所提方法在支持 OR 语义和 AND 语义上的有效性和高效性.

本文第 1 节回顾 Top-k 空间关键字查询领域的相关工作.第 2 节给出问题的形式化描述以及相关定义.第 3 节阐述聚集倒排线性四分树索引结构 AIL,并详细介绍在该树上的相关操作.第 4 节提出基于 OR 语义的受限 Top-k 空间关键字查询处理算法 VQuad 和 VGrid.第 5 节阐述在真实数据上进行的一系列实验,并对实验结果进行分析.最后,第 6 节对全文进行总结.

1 相关工作

空间关键字查询的文本约束可分为 4 种[11]:完全匹配、部分匹配、模糊匹配和布尔约束.文献[8,16–20]属于完全匹配约束(即 AND 语义约束),查询完全满足用户关键字要求的对象.适用于用户对文本相似性要求较高的情况.文献[7,12,15,21–24]属于部分匹配约束(即 OR 语义约束),查询满足用户部分关键字要求的对象.相较于完全匹配约束,当查询用户对文本相似性要求的严格程度不那么高时,可以采用部分匹配的方式.模糊匹配[25–29]要求对象关键字与查询关键字进行字符串相似度匹配.布尔约束[30–33]是用一个布尔表达式指定必须包含或可包含的关键词以及不包含的关键词.

就查询结果的排序方式而言,排序公式一般为空间距离和文本相关性的某种组合.例如,文献[30,34]仅依据查询对象与被查询对象的空间距离作为排序依据;文献[35]以空间和文本相关性的比值作为排序依据;文献[16,36–39]采用的是空间和文本相关性的线性组合;文献[16,35–39]使用户对于空间距离与文本相关性的重要程度有一定的选择权,但不能完全约束空间距离,导致查询返回的结果有距离用户过远的可能.本文采取空间距离与文本相关性的线性组合排序,并且增加了对空间距离的约束,充分考虑了用户对空间距离与文本重要程度的实际需求.

现有的在空间文本对象上建立的索引结构可分为空间优先和文本优先两类.空间优先的索引是先根据空间划分建立索引,然后在每个节点中存放有关的关键字信息.空间优先的索引中使用最多的是 IR-tree[19,22]和 IR^2-tree[18]等.IR-tree 为 R-tree 中的每个节点关联了一个倒排文件.IR^2-tree 为 R-tree 中的每个节点关联了签名文件,签名文件概括了一个节点所包含的关键字信息.当数据量较小时,IR-tree 和 IR^2-tree 的处理较为高效,当数据量增大时,这类索引的效率将明显下降,并且剪枝困难,影响到算法效率[15].另一类是文本优先的索引,这类索引是在现有文本索引(如倒排文件)上进行的改进,将每一个关键字映射到一个含有空间信息的数据结构上,如 I^3[15]、S2I[13]、IL-Quadtree[16]等.I^3 索引使用四分树(quadtree)将位置空间进行划分,提出关键字单元作为基本存储单元,根据不同关键字在单元格中出现的次数,分为频繁和非频繁的关键字.对于非频繁的关键字,直接用一个页面存储相关对象,通过查找表直接映射到数据文件中的物理位置;对于频繁的关键字,为频繁的关键字设计头文件指向关键字单元格所在磁盘页,其中,头文件是一个类似于四分树的结构,每个节点存储关于频繁关键字的概要信息.IL-Quadtree 针对每个关键字建立一棵线性四分树.通过检测不同树的相同节点,检验该区域是否包

含所有的关键字,以实现剪枝.

本文在研究内容上,就文本约束而言,属于部分匹配约束;就排序方式而言,采用空间与文本相似性的线性组合;就索引结构而言,采用文本优先的聚集倒排线性四分树.与现有工作的区别主要体现在以下 3 点:第一,本文将整个区域划分为虚拟网格,网格中每一个单元均具有唯一的码值标识,非空网格单元以聚集线性四分树的形式存储.第二,利用单元码唯一并与单元坐标可相互转换的性质,邻近单元可通过 $O(1)$时间复杂度的方法计算来获得,极大地加快了查询速度.第三,空间和文本相关性的线性组合同时考虑了空间距离与文本相关性,并且在此基础上增加了对空间距离的约束,通过对空间距离的约束有效地缩小了可查询的空间范围.

2 问题的形式化定义

空间中任意对象均包含地理位置和文本关键字集合,记为 $o=(loc,T)$,其中,空间位置 $o.loc=(x,y)$,文本关键字集合 $o.T=\{t_1,t_2,...,t_n\}$,t_i 是文本关键字.任意两个空间文本对象的相似性包括空间相似性和文本相似性两个方面.

定义 1(空间相似性). 任意两个空间文本对象在空间中的相似程度用空间相似性表示.查询对象 q 与被查询对象 o 的空间相似性定义为

$$f_s(q,o)=\frac{\delta(q.loc,o.loc)}{\delta_{\max}} \tag{1}$$

其中,$\delta_{\max}$ 表示空间中任意两点的最远距离,$f_s(q,o)$可采用任意两点间的距离公式,本文采用的是欧几里德距离.两个对象空间距离越近,$f_s(q,o)$的值越小,空间相似性越大.

定义 2(文本相似性). 空间文本对象 o 中的每个关键字都被赋予一个权重,代表该关键字在对象 o 中的重要程度.对于任意的关键字 t,在对象 o 中的重要程度记为 $w_{t,o}$.$w_{t,o}=tf_{t,o}\times idf_t$,其中,$tf_{t,o}$ 是词频,idf_t 表示逆向文件频率.两个对象 q 和 o 的文本相似性为

$$f_t(q,o)=1-\frac{\sum_{t\in q.T}w_{t,o}}{\max P} \tag{2}$$

$\sum_{t\in q.T}w_{t,o}$ 是所有查询关键字在 o 中的权重加和.max$P$ 是每个关键字在所有对象中的最大权重加和,用以归一化计算.$f_t(q,o)$的值越小,文本相似性越大.定义 2 中的文本相似性定义支持查询关键字的 OR 语义.具体地,即使查询 $q.T$ 中的部分关键字不被包含在对象 o 中(即 $w_{t,o}=0$),其文本相似性也不会为 0.

定义 3(空间文本相似性). 结合定义 1 和定义 2,任意两个空间文本对象的相似性为

$$f(q,o)=\alpha f_s(q,o)+(1-\alpha)f_t(q,o) \tag{3}$$

其中,$\alpha(\alpha\in[0,1])$是一个可调节参数,用以调节空间距离与文本相似性之间的重要程度.$f(q,o)$的值越小,q 与 o 的空间文本相似性就越大.

本文问题描述如下:设空间文本对象集合 $O=\{o_1,o_2,\ldots,o_n\}$,用户查询 $q=(loc,T,d,k)$.loc 是用户查询所在位置,T 是由一系列查询关键字组成的集合,d 是用户可以接受的最远距离,k 即最近邻对象个数,其中,d 的优先级比 k 高.受限 Top-k 空间关键字查询即从 O 中查找在距离 d 范围内空间文本相似性最高的 k 个被查询对象.其形式化表示见定义 4.

定义 4(受限 Top-k 空间关键字查询). 设查询对象 q 和被查询集合 O,一个对象 $o\in O$ 是查询 q 的受限 Top-k 关键字查询结果当且仅当对象 o 满足

$$|o'|o'\in O\ \&\ f(q,o')\leqslant f(q,o)|\leqslant k \text{ 并且 } f_s(q,o)<d \tag{4}$$

3 倒排聚集线性四分树 AIL

本文采用倒排聚集线性四分树索引所有的空间文本对象.倒排聚集线性四分树是倒排表与聚集线性四分树的结合.倒排表中的每一个关键字指向一棵聚集线性四分树.图 2 是基于图 1 中的空间文本对象建立的倒排聚集线性四分树示例.

一个关键字对应一棵聚集线性四分树.该树由包含该关键字的所有空间文本对象组成,那些不包含该关键

字的对象不会在该聚集线性四分树中被索引.“聚集”是指在树中每一个节点中都存储一个聚集值,即该关键字在此节点下的最大权重.如图 4 所示,“coffee”对应的线性四分树的根节点中的 0.176 表示树下所有的对象{o_1,o_3,o_4,o_5}中“coffee”的权重均不大于 0.176.线性四分树源于四分树,与四分树不同的是,线性四分树以 B+–树的形式存储空间位置的编码值,不是直接存储空间位置(编码方法将在本节后面加以阐述).此外,线性四分树中不存储不包含任何对象的空间码值.若采用四进制 Morton 码对图 1 所示的空间进行编码,其结果可如图 3 所示.包含“coffee”关键字的对象有{o_1,o_3,o_4,o_5},将这些对象所在位置对应的 Morton 码用 B+–树索引起来即形成“coffee”关键字对应的聚集线性四分树,如图 4 所示.类似地,“cinema”对应的聚集线性四分树如图 5 所示.图 4 和图 5 中的两棵聚集线性四分树是图 2 中“coffee”和“cinema”的关键字对应的聚集线性四分树的详细版.

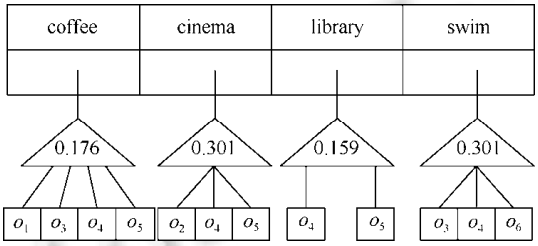


Fig.2 Aggregate inverted linear quadtree

图 2 聚集倒排线性四分树

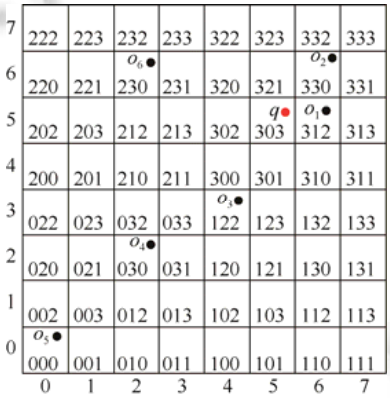


Fig.3 Encoded space

图 3 空间编码

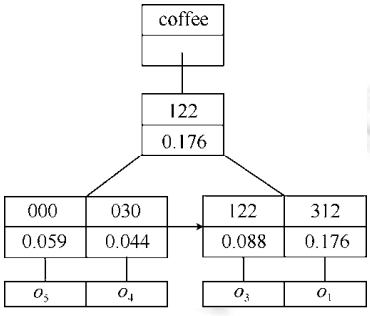


Fig.4 Aggregate linear quadtree w.r.t. coffee

图 4 “coffee”对应的聚集线性四分树

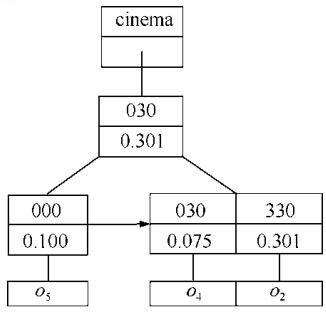
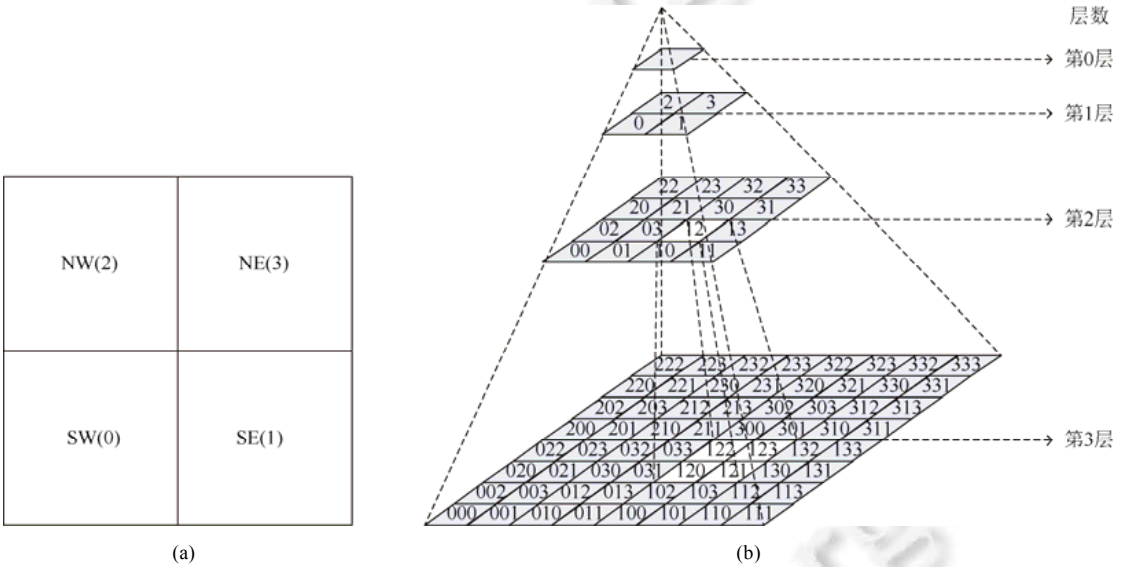


Fig.5 Aggregate linear quadtree w.r.t. cinema

图 5 “cinema”对应的聚集线性四分树

空间位置可遵循一定的规则进行编码,常用的有四进制或十进制 Morton 码,本文采用四进制 Morton 码.文献[40]对空间位置编码方法进行了详细阐述,这里仅作简要说明.编码过程需要借助四分树,设四分树最大层次

为 *r*.自顶向下地对空间位置进行四等分直至层数 *r*.在任意层次 *h*(*h*≤*r*)下,将 SW、SE、NW、NE 这 4 个方向的等分区域分别标记为 0、1、2、3,如图 6(a)所示.在空间上建立四分树(如图 6(b)所示).四分树上一个 *h* 层的空间位置可用一个 *h* 位的四进制串表示,称为位置 Morton 码.如在图 6(b)所示的第 3 层,任意一个空间位置都是一个 3 位的四进制串.四进制串中从左向右的任意一位表示的是该位置在相应层次的方向.具体地,第 3 层的 Morton 码左侧第 1 位数字表示该节点在四分树深度为 1 时区域位于的方向.如图 6(b)所示的中间 4 个区域的编码为 120、121、122、123,其中左侧第 1 位的“1”表示这 4 个区域均处于四分树第 1 层的 SE 方向.左侧第 2 位数字表示深度为 2 时该节点所属区域的方向.继续上面的例子,左侧第 2 位的“2”表示这 4 个区域均处于四分树第 2 层划分的 NW 方向.第 3 位的 0、1、2、3 表示第 3 层上 4 个方向的划分.在线性四分树中索引的均是底层(最深层)的 Morton 码.因此,采用 Morton 码对图 1 所示的空间进行编码,图 3 和图 6(b)所示的最底层叶子节点编码是一致的.



(a) (b)

Fig.6 Encoded Morton

图 6 Morton 编码

线性四分树是将上述非空叶子节点的四进制 Morton 码值以数字的形式索引起来.综上所述,聚集线性四分树 AIL 的创建算法如算法 1 所示.

算法 1. 构建倒排聚集线性四分树 AIL.

Input:空间文本对象集合 *O*,关键字域值 *D*;

Output:倒排聚集线性四分树 AIL.

1. 为所有的查询关键字创建倒排表 *AIL*;
2. **for** 每一个对象 *o*∈O
3. *cm*=计算对象 *o* 的 Morton 码;
4. **for** 关键字 *w*∈*o.T* //对于对象 o 中的每一个关键字
5. **if** *cm* 不在关键字 *w* 对应的 B+–树中
6. 将 *cm* 插入到 *w* 对应的 B+–树;
7. **else**
8. 在 *cm* 对应的叶子节点中添加对象 *o*;
9. 为 *AIL* 中所有的节点添加聚集值;

4 基于 OR 语义的受限 Top-*k* 空间关键字查询算法

本文的研究目标是基于*AIL*,从带有空间位置和文本信息的对象集合*O*中,寻找在受限范围内与查询*q*空间文本相似性最高的 *k* 个对象.*AIL* 融合了空间与文本信息,其中的线性四分树实质存储的是空间位置编码,倒排文件中存储了文本信息.所以,基于 *AIL* 进行 Top-*k* 空间关键字查询有两种思路:(1) 将线性四分树转化为空间上的四分树,改进已有经典算法,这是设计 VQuad 算法的初衷(详见第 4.2 节);(2) 从线性四分树的空间编码入手,直接在编码后的空间上进行查询,这是设计 VGrid 算法的动机(详见第 4.3 节).在介绍具体算法之前,我们先来说明在两种算法中都将用到的定义和定理.

4.1 相关定义

在后续算法中需要计算查询点到一个覆盖多个对象的矩形的空间文本相似性.因此,下面首先定义扩展的空间文本对象,然后说明如何利用定义 3 计算查询点到扩展空间文本对象的空间文本相似性.

定义 5(扩展的空间文本对象). 扩展的空间文本对象 *R* 包含地理位置和文本关键字集合两个部分,其形式化的表示为 $R=(loc,T)$. 该对象的空间位置 *loc* 用一个矩形表示,该矩形可覆盖 *R* 下的每一个空间文本对象.*T* 为 *R* 下的所有覆盖对象的关键字集合的并集,其中,针对每一个属于 *T* 的关键字由两个元素组成(t,w),*t* 是关键字本身,*w* 是该关键字在 *R* 中的最大权重.

以图 7 为例说明定义 5.图 7 显示了一个扩展的空间文本对象 *R*,其中,*R.loc* 覆盖对象 o_3 和 o_4.*R.T*={coffee(0.088),cinema(0.075),library(0.119),swim(0.151)}.

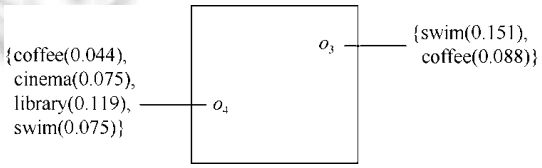


Fig.7 Extended spatio-textual object *R*

图 7 扩展的空间文本对象 *R*

在扩展的空间文本对象上,依然可以利用公式(3)计算查询 *q* 与扩展空间文本对象 *R* 的空间文本相似性.其中,空间相似性采用查询 *q* 到矩形 *R.loc* 的最小空间距离,文本相似性采用公式(2)即可.

下面的定理 1 证明了查询 *q* 与扩展的空间文本对象 *R* 的空间文本相似性是查询 *q* 到 *R* 中覆盖的任意对象的空间文本相似性的上限.

定理 1. 对于 *R* 下覆盖的任意对象 o,$f(q,R)\leqslant f(q,o)$.

证明:从空间和文本两个角度证明.

从空间的角度,易知对于 *R* 中包含的任意对象 *o*,对象 *o* 到查询点 *q* 的空间距离不小于 *R* 到 *q* 的空间距离,即 $\delta(q.loc,R.loc)\leqslant\delta(q.loc,o.loc)$. 因此,由公式(1)可知 $f_s(q,R)\leqslant f_s(q,o)$.

从文本的角度,易知对于 *R* 中的任意对象 o,$o.t\in R.T$,对象 *o* 对应于查询 *q* 的文本权重不大于 *R* 对应于查询 *q.T* 的文本权重,即 $\sum_{t\in q.T}w_{t,o}\leqslant\sum_{t\in q.T}R.T.W_{t,o}$. 因此,由公式(2)可知 $f_t(q,R)\leqslant f_t(q,o)$.

综合上述空间和文本两个角度,由公式(3)可得 $f(q,R)\leqslant f(q,o)$.由于 $f(q,o)$的值越小越好,因此查询 *q* 与 *R* 的文本相似性是查询 *q* 到 *R* 中覆盖的任意空间文本对象的空间文本相似性的上限. □

4.2 VQuad算法

VQuad 算法的基本思想是采用最小最佳优先原则,利用 *AIL* 中存储的空间位置重建虚拟四分树[16],在虚拟四分树上寻找满足 OR 语义的空间受限的 *k* 最近邻查询结果.文献[16]在虚拟四分树上进行 Top-*k* 关键字查询处理,但其仅实现了 AND 语义.VQuad 算法改进了文献[16]以支持 OR 语义的空间受限查询.下面首先简要介绍线性四分树与虚拟四分树的转换过程.接下来,在虚拟四分树的基础上详细介绍 VQuad 查询处理方法.

4.2.1 虚拟四分树

建立虚拟四分树的目的是将查询中不同关键字对应的聚集线性四分树整合起来,通过计算虚拟四分树的节点与查询之间的空间文本相似性,将不满足查询要求的节点较早地剪枝掉.虚拟四分树之所以称为“虚拟”在于其物理上并不真实存在.由于四分树中每个层次在线性四分树的区域编码已知,所以根据树的层次以及区域编码能够建立一棵虚拟四分树.图 8 是与图 6(b)一样的虚拟四分树.唯一不同的是,这里将每个层次的编码扩展为 r(=3)位.空缺位用 X 字符补位.虚拟四分树的根节点即整个区域编码为 XXX(其中,X 可取{0,1,2,3}中的任意值).第 1 层的 4 个区域编码分别为 0XX、1XX、2XX、3XX(仅左侧第 1 位有意义).虚拟四分树的第 2 层节点区域编码范围是 00X~33X(仅左侧前 2 位有意义),虚拟四分树的第 3 层节点区域编码范围是 000~333.

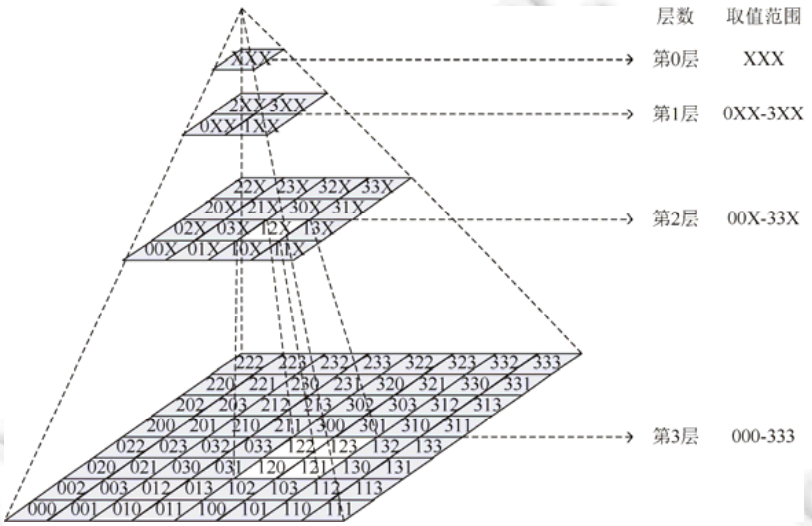


Fig.8 Virtual quadtree

图 8 虚拟四分树

在查询过程中需要计算虚拟四分树上的一个区域与查询之间的空间文本相似性.然而,虚拟四分树仅是逻辑上的概念,在物理上实际存储的是以 B+–树组织起来的码值.因此,在计算空间文本相似性时,在空间方面,用区域中的最大 Morton 码值与最小 Morton 码值的横纵坐标所围成的区域限定区域范围,表示为(最小 Morton 码值,最大 Morton 码值).通过区域码值与空间横纵坐标的转换即可确定该区域的空间位置及空间相似性.在文本方面,将区域中所有单元的关键字权重最大值加和作为该区域对应查询的文本权重.由此,可利用公式(3)计算区域与查询间的空间文本相似性.例如,图 8 中第 2 层编码为 12X 的区域,其在 B+–树上实际对应的编码为{120,121,122,123}.该区域对应查询要求的文本权重,即 4 个单元内对应查询关键字权重最大值的加和.

4.2.2 VQuad 算法流程

VQuad 算法逻辑上将整个空间用四分树进行划分,利用最小最佳优先原则选择符合查询要求的 Top-k 个空间文本对象.算法 2 展示了 VQuad 算法的具体过程.在算法 2 中,用栈 *nbs* 存放从虚拟四分树中取得的节点.栈中元素以节点到查询 q 之间的空间文本相似性的值升序排序.首先,找到查询 q 包含的所有关键字对应的聚集线性四分树 *btset*(第 2 行),将虚拟四分树的根节点入栈 *nbs* 中(第 3 行).当栈 *nbs* 不为空时,从 *nbs* 中取栈顶 e(第 5 行).如果 e 是空间文本对象,则将该对象存入结果集 R 中(第 8 行~第 9 行).如果 e 是节点,判断 e 是叶子节点或非叶子节点,如果 e 是非叶子节点,则将节点 e 所在区域进行逻辑上四等分,计算 e 的 4 个孩子节点与查询 q 之间的空间距离,判断该空间距离是否在用户允许范围 d 内.如果在空间限制范围内,则运用公式(3),计算 e 的孩子节点与查询 q 之间的空间文本相似性,并将该孩子节点码值及其空间文本相似性 $f(q,e')$入栈 *nbs*(第 10 行~第 16 行).如果 e 为叶子节点,则将 e 在不同的线性四分树中包含的所有对象取出,判断取出对象空间上是否在用户允许范围 d 内.如果是,则计算对象与查询 q 之间的空间文本相似性并入栈 *nbs* 中(第 18 行~第 23 行).最后,当结果

集 R 中的对象个数达到 k 时,算法终止.

需要说明的是,VQuad 算法不仅可以支持 OR 语义,也可以支持 AND 语义.即只需在取出线性四分树上某一层次的节点时,判断其是否在所有的聚集线性四分树中存在.如果在所有查询要求的关键字对应的线性四分树中均存在,则执行算法的相关计算,即修改算法 2 中的第 13 行和第 20 行.

4.2.3 运行示例

我们用图 1 来说明算法 VQuad 的运行过程.设 *AIL* 的层数 r=3,查询点 q={(5.8,5.8),{coffee,cinema},3,1},其中,d=3,k=1.

首先,虚拟四分树根节点的区域码值为(000,333).通过公式(3)计算查询点 q 与虚拟四分树根节点的空间文本相似性 $f(q,code)$=0.344.将{(000,333),0.344}入栈 *nbs* 中.由于 *nbs* 不为空,栈顶出栈 e=(000,333).计算 e 的 4 个孩子节点的码值,即{(000,033),(100,133),(200,233),(300,333)}.判断 4 个孩子均在查询点 q 的 d 范围内.计算 4 个孩子节点与查询 q 的空间文本相似性,见表 1 中第 3 列.将节点码值及其与查询对应的空间文本相似性值按升序存入栈 *nbs* 中(见表 1 中第 4 列).取栈顶 e=(300,333),因为单元(300,333)是四分树的中间节点,计算 e=(300,333)的 4 个孩子节点的码值,即{(300,303),(310,313),(320,323),(330,333)}.计算查询点 q 到 d 范围内的孩子节点的空间文本相似性,见表 1.将节点码值及其与查询对应的空间文本相似性值按升序存入栈 *nbs* 中(见表 1 中第 2 行).

重复上述操作,直至表 1 第 4 行,取当前栈顶 e=(330,330).因为单元(330,330)是叶子节点.从与查询关键字对应的 B+–树中取出 330 单元以下的对象,即{o_2}.因为 o_2 与查询点 q 的距离小于 d,则计算查询点 q 到对象 o_2 的空间文本相似性.将对象及其与查询 q 的空间文本相似性的值入栈 *nbs* 中(见表 1 中第 4 行).第 5 行中,取栈顶 e=o_2,因为 o_2 是空间文本对象,将 o_2 存入结果集.此时 k=1,且栈 *nbs* 中栈顶元素的空间文本相似性上限小于当前查询结果中最差的空间文本相似性,由定理 1 可知,程序停止.输出结果集 R={(o_2,0.510)}.

Table 1 A running example of VQuad

表 1 VQuad 算法运行实例

序号	结果集 R	被访问的单元	被访问单元划分后的 4 个孩子节点及与查询 q 对应的空间文本相似性	栈 *nbs* 状态
1	{}	{(000,333),0.344}	{(000,033),0.671},{(100,133),0.698},{(200,233),0.764},{(300,333),0.343}	{(300,333),0.343},{(000,033),0.671},{(100,133),0.698},{(200,233),0.764}
2	{}	{(300,333),0.343}	{(300,303),0.700},{(310,313),0.576},{(320,323),0.707},{(330,333),0.485}	{(330,333),0.485},{(310,313),0.576},{(000,033),0.671},{(100,133),0.698},{(300,303),0.700},{(320,323),0.707},{(200,233),0.764}
3	{}	{(330,333),0.485}	{(330,330),0.485},{(331,331),0.743},{(332,332),0.743},{(333,333),0.760}	{(330,330),0.485},{(310,313),0.576},{(000,033),0.671},{(100,133),0.698},{(300,303),0.700},{(320,323),0.707},{(331,331),0.743},{(332,332),0.743},{(333,333),0.760},{(200,233),0.764}
4	{}	{(330,330),0.485}	{o_2,0.510}	{o_2,0.510},{(310,313),0.576},{(000,033),0.671},{(100,133),0.698},{(300,303),0.700},{(320,323),0.707},{(331,331),0.743},{(332,332),0.743},{(333,333),0.760},{(200,233),0.764}
5	{o_2,0.510}	{o_2,0.510}	{}	{(310,313),0.576},{(000,033),0.671},{(100,133),0.698},{(300,303),0.700},{(320,323),0.707},{(331,331),0.743},{(332,332),0.743},{(333,333),0.760},{(200,233),0.764}

算法 2. 基于虚拟四分树的 OR 语义查询排序算法 VQUAD.

Input:*ALT*:聚集倒排线性四分树,k:要求返回对象的个数,d:空间距离约束,q:查询对象;

Output:R:前 k 个 f 值最小的对象集合.

1. R=∅;*nbs*=∅;
2. *btset*=*AIL* 中所有与 $q.T$ 相对应的聚集线性四分树;
3. 将虚拟四分树的根节点入栈 *nbs*;

4. **while** $nbs\neq\varnothing$ **do**
5. $e\leftarrow$从栈 nbs 中出栈顶;
6. **if** $|R|=k$ **then**
7. **break;**
8. **else if** e 是一个对象 **then**
9. $R=R\cup e$;
10. **else if** e 是一个非叶子节点 **then**
11. **for** e 每一个子节点 e' do
12. **if** $\delta(q.loc,e'.loc)\leqslant d$ **then**
13. **for** *btset* 中的每一棵聚集线性四分树
14. 计算 e'中关键字对应的权重加和;
15. 计算 $f(q,e')$;
16. 将子节点 e'和 $f(q,e')$入栈 *nbs*;
17. **else**
18. **for** 叶子节点 e 中的每一个对象 o
19. **if** $\delta(q.loc,o.loc)\leqslant d$ **then**
20. **for** *btset* 中的每一棵聚集线性四分树
21. 计算 o 中关键字对应的权重加和;
22. 计算 $f(o,q)$;
23. 将对象 o 和 $f(q,o)$入栈 *nbs*;
24. **return** R;

4.3 VGird查询算法

因为 VQuad 算法在计算四分树某一层节点与查询点的空间文本相似性时,需要不断重复地访问线性四分树,进行 Morton 码与空间位置的转换,从而影响了查询效率.实际上,无需将线性四分树构建成虚拟四分树,可直接从线性四分树上的 Morton 码出发,通过二进制的位运算获得单元的邻近区域和区域中的空间文本对象.基于这样的动机,本文提出了基于虚拟网格的 VGrid 查询算法.VGrid 将整个空间看成一个虚拟网格,网格中每一个单元均有唯一的 Morton 码.利用单元的 Morton 码与单元位置坐标可相互转换的性质,邻近单元可通过公式(5)在 $O(1)$时间复杂度下计算获得,提高了查询效率.

(1) VGrid 算法流程

算法的基本思想是以查询 q 所在位置为中心,从中心单元开始,循环寻找中心单元的邻近 8 个单元(如图 9 中的 n_0~n_7 所示)中包含的对象,计算该对象与查询 q 的空间文本相似性,不断更新查询结果集直至获得满足空间限制的 Top-k 结果.为了防止空间单元的重复访问,采用 *visit* 布尔集合来标识单元是否已被访问.

具体地,首先找到查询点 q 所在的单元,确定相应的四进制 Morton 码,记为 *code*(第 2 行).找到查询 q 包含的所有关键字对应的聚集线性四分树 *btset*(第 3 行).根据定义 3 计算查询 q 到单元 *code* 的空间文本相似性,以(*code*,$f(q$,*code*))的形式存入栈 *nbs*(第 4 行).*nbs* 中的元素以空间文本相似性升序排序.取 *nbs* 栈顶 *nbs_t*.若栈顶对应的码值 *nbs_t.code* 存在于线性四分树集合 *btset* 中的任意一棵树中,则从具有 *nbs_t.code* 的线性四分树中取出 *nbs_t.code* 单元下的对象组成 *Oset*(第 7 行~第 8 行).计算 *Oset* 中的对象与查询 q 的空间文本相似性,将满足空间距离约束 d 的对象放入不断更新的候选结果集 *RSet* 中,以查询 q 到该对象的空间文本相似性为排序关键字(第 9 行~第 11 行).当 *Rset* 中的第 k 个对象的空间文本相似性值大于栈顶元素的空间文本相似性值时,说明空间中存在比候选集中更符合用户要求的查询结果.此时,利用公式(5)寻找栈顶单元的邻近 8 个单元,将 8 个单元中满足空间距离约束 d 并且未被访问的单元的 *code* 值及其与查询的空间文本相似性存入 *nbs*,并在 *visit* 中将 *code* 单元标识为已访问(第 12 行~第 17 行).重复第 6 行～第 19 行,直至候选结果集$|Rset|=k$,且 *Rset* 中对象的第 k 个

对象的空间文本相似性值优于 *nbs* 中栈顶元素的空间文本相似性值.

算法 3. 基于虚拟网格的 OR 语义查询排序算法 VGrid.

Input:*ALT*:聚集倒排线性四分树,*q*:查询对象,*d*:空间距离约束,*k*:要求返回的对象个数;

output:*RSet*:前 *k* 个 *f* 值最小的对象集合.

1. *RSet*:=∅;*nbs*:=∅;
2. *code*=*q* 的位置点所对应的单元;
3. *btset*=所有与 *q.T* 相对应的 *AIL*;
4. *nbs*←{(*code*,*f*(*q*,*code*)};
5. **while** (|*Rset*|<*k*||*Rset.f*≥*nbs.top.f*)
6. *nbs_t*=从栈 *nbs* 中取栈顶;
7. **if** (*nbs_t.code* 存在于 *btset* 中任意一棵树中)
8. *Oset*=分别取出在 *btset* 中所有 *code* 值等于 *nbs_t.code* 的单元对应的对象;
9. for (*Oset* 中的每一个对象 *o*)
10. if (*δ*(*q.loc*.*o.loc*)≤*d*)
11. 将 *f*(*q*,*o*)更新入 *RSet*;
12. **if** (|*Rset*|<*k*||*Rset.f*≥*nbs_t.f*)
13. *code*=调用公式(4)计算 *nbs_t* 的邻近 8 个单元;
14. **if** (*code* 没有被标记为 *visit* & *δ*(*q.loc*,*e.loc*)≤*d*)
15. 计算 *f*(*q*,*code*);
16. *code* 被标记为 *visit*;
17. 将 *code* 入栈 *nbs*;
18. **else**
19. **break**;
20. **return** *RSet*;

这里需要说明两点:(1) 为保证算法的正确性,在算法 3 的第 15 行中计算虚拟网格中的一个单元到查询 *q* 的空间文本相似性时,单元格关键字权重采用的是该关键字在空间中的全局最大值;(2) VGrid 算法可同时支持 OR 语义和 AND 语义.在完成 AND 语义查询时仅需将算法 3 中的第 7 行修改为被查询单元或对象是否存在于查询关键字对应的所有线性四分树即可.

(2) 运行实例

继续以图 1 和查询点 *q*={(5.8,5.8),{coffee,cinema},3,1}的例子说明算法 3(VGrid)的运行过程.首先找到查询点 *q* 所在的单元,确定相应的 *code* 值为 303.在 *visit* 中将 303 标记为已访问.通过公式(3)计算查询点 *q* 到 303 单元的空间文本相似性 *f*(*q*,303)=0.700.将(303,0.700)入栈 *nbs* 中.设关键字 coffee、cinema 分别对应的线性四分树为 bt_1 和 bt_2(即图 3 和图 4).栈顶元素 303 出栈.虽然单元 303 到查询点 *q* 的距离小于 *d*,但 303 没有在 bt_1 和 bt_2 中,说明 303 中没有对应查询文本的对象.利用公式(5)计算 303 的相邻 8 个单元,即{300,301,310,312,330,321,320,302}.计算查询 *q* 到每一个单元的空间文本相似性,见表 2 的第 1 行.将单元 Morton 码值及其与查询对应的空间文本相似性值按升序存入栈 *nbs* 中,并在 *visit* 中将这些单元标记为已访问.

从栈 *nbs* 中取栈顶 330.因为 330 到查询点 *q* 的距离小于 *d*,取出树 bt_1 和 bt_2 中 330 单元对应的对象,去重后得 *h*={o_2}.计算 o_2 与查询 *q* 的空间文本相似性 *f*=0.510,并存入结果集 *R*={(o_2,0.510)}.此时,结果集 *R* 中对象 o_2 的空间文本相似性大于栈顶元素与查询 *q* 的空间相似性(即 0.510>0.485).根据公式(4)计算栈顶 330 相邻 8 个单元,即{303,312,313,331,333,332,323,321}.其中,{303,312,321}均已被访问,因此将剩余单元到查询点 *q* 的距离小于 *d* 的单元,即{313,331,333,332,323}入栈,见表 2 第 2 行.将{313,331,333,332,323}标记为已访问.由于此时结果集 *R* 中 o_2 与查询 *q* 空间文本相似性小于栈顶元素对应的空间文本相似性(即 0.510<0.576).所以,程序终止.输出

的结果集 R={(o_2,0.510)}.

Table 2 Running instance of VGRID

表 2 VGRID 算法运行实例

结果集 R	被访问的单元	邻近 8 个单元及其与查询对应的空间文本相似性	栈 *nbs* 状态
{}	303(0.700)	300(0.740),301(0.728),310(0.729),312(0.576), 330(0.485),321(0.707),320(0.729),302(0.728)	330(0.485),312(0.576),321(0.707),301(0.728), 302(0.728),310(0.729),320(0.729),300(0.740)
(o_2,0.510)	330(0.485)	303(0.700),312(0.579),313(0.742),331(0.743), 333(0.760),332(0.743),323(0.742),321(0.707)	312(0.576),321(0.707),301(0.728),302(0.728), 310(0.729),320(0.729),300(0.740),313(0.742), 323(0.742),331(0.743),332(0.743),333(0.760)

(3) 邻近 8 个单元的计算方法

Morton 码[23]是空间网格划分后每一个单元(cell)的唯一标识,与单元的空间坐标可进行相互转换.利用这个性质,通过二进制位运算很容易获得某单元周围邻近 8 个单元的码值乃至位置坐标.下面首先说明 Morton 码与空间坐标的相互转换方法,接下来说明如何通过二进制的位运算在 $O(1)$时间内获得邻近单元的码值.

已知某单元的十进制坐标为(x,y),其 Morton 码的具体计算方法如下.先将单元位置坐标(x,y)的值转化为二进制形式,令 $x=x_{r-1}...x_1x_0$,$y=y_{r-1}...y_1y_0$,其中,$x_i,y_i\in\{0,1\}$,r 为线性四分树划分的深度.该单元对应的二进制编码为 $n=y_{r-1}x_{r-1}...y_1x_1y_0x_0$.例如,图 3 中单元 303 的坐标为(5,5),将两个坐标转化为二进制,分别是 x=110,y=110,则该单元对应的编码为 n=110011,转化为四进制后即 Morton 码为 303.

在算法 3 的第 13 行需要查找中心单元邻近的 8 个单元,如图 9 所示.其中,任意单元的 8 个邻近单元的计算方法如公式(5)[36].设中心单元的 Morton 码值为 *code*,则有:

$$m_q = n_q \oplus_q \Delta n_i = (((n_q \mid t_y) + (\Delta n_i \wedge t_x)) \wedge t_x) \mid (((n_q \mid t_x) + (\Delta n_i \wedge t_y)) \wedge t_y) \tag{5}$$

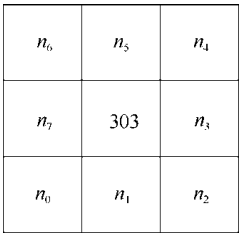


Fig.9 The eight neighbor cells for the cell 303

图 9 303 单元的 8 个邻近单元

在公式(5)中,m_q 是邻近单元 Morton 码的二进制表示.n_q 是中心单元 *code* 的二进制表示.t_x 和 t_y 是两个二进制常数:t_x=01...0101 表示“01”重复 r 次,t_y=10...1010 表示“10”重复 r 次.Δn_i 是基本方向增量之一,即计算中心单元任意方向的单元码值时坐标的变化量,8 个方位的基本增量即 Δn_0=(−1,−1),Δn_1=(0,−1),Δn_2=(1,−1),Δn_3=(1,0),Δn_4=(1,1),Δn_5=(0,1),Δn_6=(−1,1),Δn_7=(−1,0).采用上文单元码值的计算方法,将 Δn_i 由坐标值转换为 Morton 码(见表 3 第 2 列).公式(5)采用的是位运算:“+”表示相加;“|”表示 OR;“^”表示 AND.设线性四分树划分深度 r=3,计算中心单元 303(转换成二进制为 110011)的邻近 8 个单元码值的过程见表 3.

Table 3 Increment of direction

表 3 方向增量

n_i	Δn_i	公式(4)带入后的 m_q	Morton 码值
n_0	Δn_0=111111	(((110011|101010)+(111111∧010101))∧010101)|(((110011|010101)+(111111∧101010))∧101010)=110000	300
n_1	Δn_1=101010	(((110011|101010)+(101010∧010101))∧010101)|(((110011|010101)+(101010∧101010))∧101010)=110001	301
n_2	Δn_2=101011	(((110011|101010)+(101011∧010101))∧010101)|(((110011|010101)+(101011∧101010))∧101010)=110100	310
n_3	Δn_3=000001	(((110011|101010)+(000001∧010101))∧010101)|(((110011|010101)+(000001∧101010))∧101010)=110110	312
n_4	Δn_4=000011	(((110011|101010)+(000011∧010101))∧010101)|(((110011|010101)+(000011∧101010))∧101010)=111100	330
n_5	Δn_5=000010	(((110011|101010)+(000010∧010101))∧010101)|(((110011|010101)+(000010∧101010))∧101010)=111001	321
n_6	Δn_6=010111	(((110011|101010)+(010111∧010101))∧010101)|(((110011|010101)+(010111∧101010))∧101010)=111000	320
n_7	Δn_7=010101	(((110011|101010)+(010101∧010101))∧010101)|(((110011|010101)+(010101∧101010))∧101010)=110010	302

5 实 验

5.1 实验设置

实验采用 Foursquare 上真实的签到数据集[41,42],包括纽约(NYC)和洛杉矶(LA)两个城市用户的签到数据.

图 10 和图 11 展示了将两个数据集导入 QGIS 后,签到兴趣点 POI(point of interests)的分布情况.表 4 列出了数据集的统计信息,包括 POI 数量、键字数量以及每个 POI 上的平均关键字数.实验中的所有算法用 Java 实现.运行于 Intel(R) i5 2.30GHzCPU 处理器、4GB 内存的 Windows 8 计算机上.实验中,所有的 POI 组成被查询点集合.查询点集合从 POI 集合中随机抽取 10 000 个.随机抽取的点位置即查询点位置信息.查询点的文本关键字按照不同等级的词频随机分配.文本关键字的等级是根据关键字词频的上四分位、中位数划分的 3 个等级(即高频、中频和低频词).实验默认设置见表 5.

文献[16]是第 1 篇在线性四分树上进行 Top-k 关键字查询的代表性工作,但其仅支持 AND 语义.本文提出的 VQuad 算法是在文献[16]中提出的基于虚拟四分树算法基础上的改进,所以在支持 OR 语义方面,实验仅对比了 VQuad 算法和 VGrid 算法在两个不同数据集上平均查询时间的变化情况(见第 5.2 节).为了验证两种算法在 AND 语义方面的有效性,第 5.3 节对两种算法支持 AND 语义的实验结果进行了对比分析.实验变动参数有关键字个数(l)、返回结果数(k)、数据集大小等.由于文献[16]已验证了倒排线性四分树与经典索引结构的对比情况,这里没有再对索引结构大小等进行赘述.

Fig.10 LA check-in datasets

图 10 LA 签到数据集

Fig.11 NYC check-in datasets

图 11 NYC 签到数据集

Table 4 Statistics for the dataset

表 4 数据集的统计信息

Property	LA	NYC
POI 数量	215 614	206 416
关键字数量	175 704	240 781
每个对象的平均关键字数量	1.34	1.32

Table 5 Default setting

表 5 实验默认设置

实验设置的参数	字母表示	默认值
每个查询点的文本关键字数	l	3
查询返回的结果数	k	10
空间和文本比重参数	α	0.3
线性四分树划分深度	R	3
空间距离约束	d	10%×空间大小

5.2 支持OR语义的算法性能对比

图 12 对比展示了 VQuad 和 VGird 算法在两个不同真实数据集上的查询平均处理时间.从图 12 观察到两种算法在 LA 和 NYC 数据集上的性能表现类似.从图 10 和图 11 可以看出,两个签到数据集 POI 分布类似且 POI 个数接近.无论是在 LA 的数据集上还是在 NYC 的数据集上,VGird 的平均处理时间均优于 VQuad.其原因在于,VQuad 算法需要多次访问查询关键字对应的线性四分树.具体地,VQuad 查询过程中采用的虚拟四分树是一个逻辑概念上的结构,物理上并不存在.因此,每次访问到某一个层次的四分树节点时,需要进行物理上线性四分树存储的 Morton 码到虚拟四分树的转换(见第 4.2.1 节).同时,VQuad 算法在计算非叶子节点与查询点的空间文本相似性时,需要该节点下覆盖对象的查询关键字的最大权重.然而,线性四分树上存储的是虚拟四分树上叶子

节点中所有对象在对应关键字的最大权重.因此,虚拟四分树的非叶子节点的关键字最大权重需要从该节点下的所有叶子节点获得,导致线性四分树的重复访问,影响了查询效率.相比之下,在空间方面,VGird 算法中通过二进制的位运算(见公式(5))直接获得附近单元位置,利用 *visit* 布尔数组防止了单元的重复访问;在文本关键字方面,VGird 直接运用的是聚集线性四分树中存储的每个关键字的最大权重.因此,查询效率得到了明显提高.

图 13 对比展示了查询平均处理时间在不同查询关键字个数 l 下的变化情况.这里用 LVQuad 和 LVGrid 分别表示在 LA 数据集运行 VQuad 算法和 VGrid 算法.NVQuad 和 NVGrid 分别表示在 NYC 数据集上运行 VQuad 算法和 VGrid 算法.查询关键字数量从 1 增加到 5.随着查询关键字个数的增加,VQuad 和 VGrid 均需要访问更多的关键字对应的线性四分树,因此查询平均处理时间会随着关键字个数的增加而增加.图 13 印证了我们的想法,两种算法的查询平均处理时间均随着查询关键字个数的增加亦在增长.然而,两种算法的增加幅度是逐渐降低的.这是因为在 OR 语义环境下,更多的查询关键字意味着用户对查询需求的放松.线性四分树中会有更多的候选结果供选择.在 k 确定的前提下,一定程度地舒缓了查询时间的增长幅度.通过对比图 13 中的 4 条曲线可以发现,无论是在 LA 数据集还是在 NYC 数据集下,VGrid 算法的查询效率均比 VQuad 算法要好,由图 13 可以看出,VGrid 比 VQuad 平均快 2.5 倍.

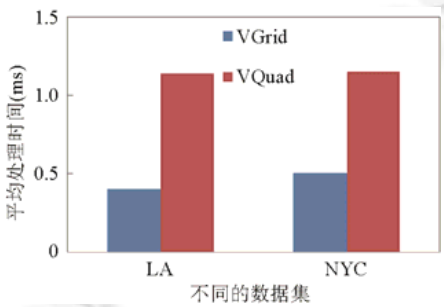


Fig.12 Performance on different datasets

图 12 不同数据集上的查询平均处理时间

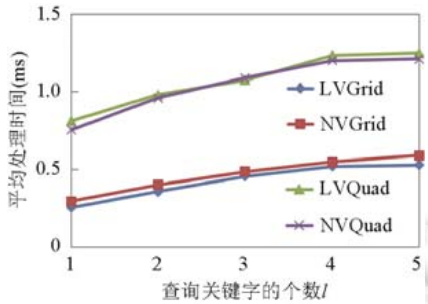


Fig.13 Performance on different number of query keywords

图 13 不同查询关键字数量上的查询平均处理时间

图 14 对比展示了查询平均处理时间在不同查询返回结果个数 k 下的变化情况.观察图 14 发现,两种算法的平均处理时间均随着查询返回结果的个数 k 的增加而增加.显而易见,由于用户提高了查询要求,两种算法均需要更多的时间在空间中寻找满足查询要求的结果.同时,从图 14 还可以发现,随着 k 的增大,在相同的数据集上,VQuad 算法的增长幅度比 VGrid 平均要大 0.05ms.这是因为,随着查询返回结果个数的增加,VQuad 算法访问了虚拟四分树上更多的非叶子节点,增加了对关键字权重的计算次数,而 VGrid 算法可直接在线性四分树上获取关键字权重,相比之下提高了查询的效率.最终,从图 14 可以看出,VGrid 算法在两个数据集的查询平均处理时间相差不大(约差 0.07ms).

我们从原始数据集中随机抽取 50 000、100 000、150 000、200 000 个 POI 对象组成了不同的被查询对象集合.从图 10 和图 11 可以看出,两个数据集中的 POI 不是均匀分布的.所以,不同大小的数据集不是在整个空间上均匀增长,而是随着 POI 分布的密集程度而增长.图 15 对比显示了查询平均处理时间在不同 POI 数据集大小下的变化情况.可以看出,整体上来看,两种算法的性能均随着数据集数量的增加而下降.平均而言,随着 POI 数量的增加,在四分树上一个节点或网格单元下覆盖的 POI 数量会增多.这就造成了从一个四分树节点或网格单元下需要抽取更多的 POI 对象进行检验.但无论在哪个数据集上,VGrid 的性能都是优于 VQuad 的.我们发现,两种算法在数据从 150 000 增长到 200 000 时,NYC 数据集上的增长幅度高于 LA 数据集.通过分析图 10 和图 11 后发现,在相同的 POI 集合大小下,NYC 数据集比 LA 的数据集分布得更为分散.整体上来看,两种算法在处理时间上是高效的,VQuad 在 1.1ms 以下,VGrid 在 0.43ms 以下.

图 16 对比展示了查询平均处理时间在不同α值下的变化.α是一个可调节参数,用来调节空间相似性与文本相似性的重要程度.α越大表示空间相似性的重要程度越大,反之,文本相似性的重要程度越大.由图 16 可以看

出,当α从 0.1 变到 0.9 时,两种算法在 LA 数据集和 NYC 数据集上的平均查询处理时间均保持平稳,两种算法的性能始终保持大约 0.5ms 的差距.

由于 VGrid 中递归计算邻近 8 个单元,为了验证算法在空间搜索方面的效率,图 17 对比展示了 VGrid 算法在两个数据集上的空间搜索占比的变化情况.空间搜索占比即查找到用户要求的 k 个最近邻结果,算法遍历过的空间与整个空间面积比值.k 从 10 增加到 50.此时,d 被设置为无穷大.由图 17 可以看出,查询搜索空间占比会随着查询返回结果个数的增加而增加.这是比较自然的现象,因为满足要求的结果分布在更远的单元.有趣的是,相同 k 值设置下 NVGrid 要找到查询结果,比 LYGrid 搜索的空间更广.这从另一方面验证了在 NYC 上的查询平均处理时间比在 LA 上要更长.即使 k 增长到 50,搜索空间的占比也低于 4.5%.

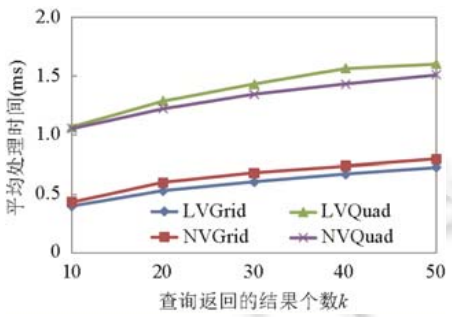


Fig.14 Performance on different k

图 14 不同查询结果数量上的查询平均处理时间

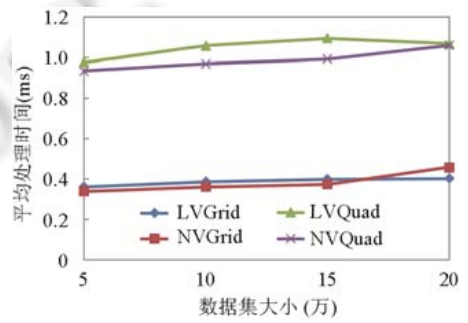


Fig.15 Performanceon different datasize

图 15 不同大小的数据集上的查询平均处理时间

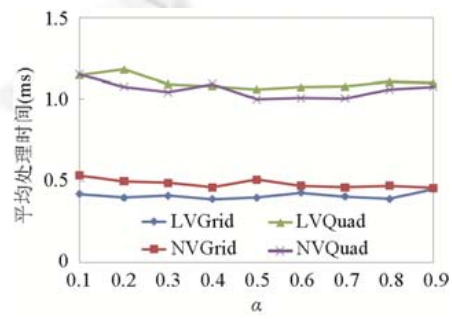


Fig.16 Performance on different α

图 16 不同参数值α上的查询平均处理时间

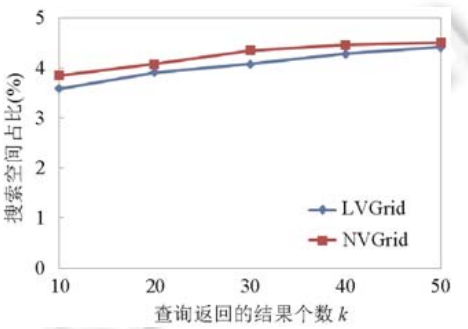


Fig.17 Percentage on different k

图 17 不同查询结果数量上的搜索空间占比

5.3 支持AND语义的算法性能对比

本文提出的 VGrid 算法和 VQuad 算法可同时支持 OR 语义和 AND 语义.为了全面验证算法的性能,本节对比展示了两种算法在支持 AND 语义方面的性能.

图 18 对比展示了 VQuad 和 VGrid 算法在支持 AND 语义时在两个真实数据集上的平均查询处理时间.与支持 OR 语义的情况相比,其相同之处是,在两个数据集上,算法 VGrid 的平均处理时间依然均优于 VQuad;不同之处在于,从整体上来讲,VQuad 在 OR 语义上运行的时间比 AND 语义长 0.2ms,而 VGrid 在 AND 和 OR 语义上的运行时间很近,平均都是 0.49ms.这是因为,VQuad 算法的处理单位是虚拟四分树上的节点,而 VGrid 的处理单位是网格单元.基于虚拟四分树的 VQuad 本质上是一种自顶向下的算法,在支持 AND 语义时,节点剪枝效果更明显,而 VGird 本质上是基于中心单元的遍历,所以 AND 语义与 OR 语义差别不大.

图 19 对比展示了查询平均处理时间在查询关键字数量从 1 增加到 5 的变化情况.随着查询关键字个数的增加,VQuad 和 VGrid 的查询平均处理时间均有所增加,其原因与支持 OR 语义的原因相同,这里不再赘述.在 l=1 时,AND 语义与 OR 语义无差异.当 l 继续增加时,4 条线均在 l 增加到 3 时发生了陡变.当 l 增加到 4、5 时,增长

速率减缓.图 20 对比展示了 AND 语义下两种算法在不同返回结果个数 k 下的变化情况.从图 20 不难发现,两种算法的平均处理时间大体上均随着查询返回结果的个数 k 的增加而增加.从两组数据来看,VGrid 较 VQuad 性能更稳定.平均来讲,在不同数据集上,VQuad 的平均查询处理时间为 1.25ms,VGrid 的平均查询处理时间是 0.69ms.图 21 和图 22 对比展示了 POI 数据集大小、参数α值变化时算法的性能.两者的变化趋势与图 15、图 16 类似,这里也不再赘述.

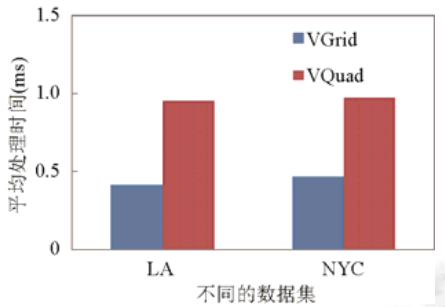


Fig.18 Performance on different datasets w.r.t AND constraints

图 18 不同数据集上支持 AND 语义的查询效率

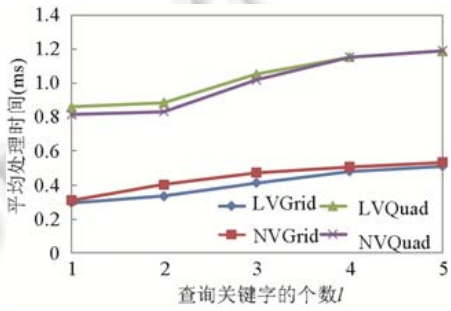


Fig.19 Performance on different number of query keywords w.r.t AND constraints

图 19 不同查询关键字数量上支持 AND 语义的查询效率

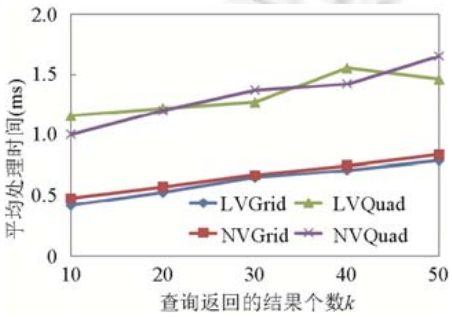


Fig.20 Performance on different number results w.r.t AND constraints

图 20 不同查询结果数量上支持 AND 语义的查询效率

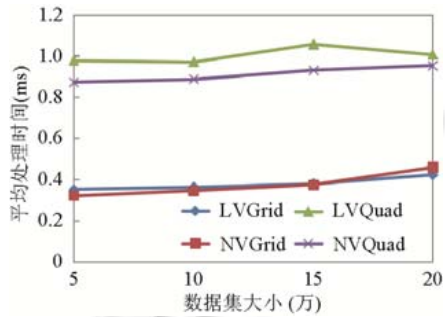


Fig.21 Performanceon different size of query datasets w.r.t AND constraints

图 21 不同大小的数据集上支持 AND 语义的查询查询效率

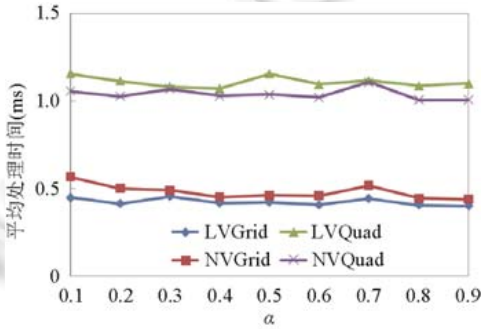


Fig.22 Performance on different α

图 22 不同参数值α上支持 AND 语义的查询效率

6 总 结

基于位置的地理信息服务在人们的生活中发挥着越来越重要的作用,针对空间文本对象查询的研究成为工业界和学术界关注的研究热点问题之一.为了给用户提供更多高品质的选择结果,本文针对基于聚集倒排线性四分树的高效 OR 语义的受限 Top-*k* 空间关键字查询的问题进行了研究.综合考虑空间距离、空间文本相似程度的需求,基于聚集倒排线性四分树,分别提出基于虚拟四分树的 VQuad 和基于虚拟网格的 VGrid 算法.两种算法均可同时支持 AND 语义和 OR 语义.通过一系列的实验发现,由于 VGrid 直接利用了线性四分树上空间编码的特点,在所有的实验设置中其性能均优于 VQuad 且算法性能更稳定.未来考虑将此技术思想应用到在道路网络上的查询研究中.

References:

[1] Zhu H, Yang X, Wang B, Lee WC. Range-based obstructed nearest neighbor queries. In: Proc. of the ACM SIGMOD. 2016. 2053−2068.

[2] Li Y, Huang Z, Zhu R, Li G, Shu L, Tian S, Ma M. Parameterized spatio-textual publish/subscribe in rode sensor networks. IEEE Access, 2017,5(99):22940−22952.

[3] Sanderson M, Kohler J. Analyzing geographic queries. In: Proc. of the Int'l ACM SIGIR Conf. 2004.

[4] Chan KH, Long C, Wong CW. On generalizing collective spatial keyword queries. IEEE Trans. on Knowledge and Data Engineering, 2018,30(9):1712−1726.

[5] Han XX, Yang DH, Li JZ. TKEP: An efficient top-*K* query processing algorithm on massive data. Chinese Journal of Computers, 2010,33(8):1405−1417 (in Chinese with English abstract).

[6] Zhu R, Wang B, Yang X, Zheng B, Wang G. SAP: Improving continuous top-*k* queries over streaming data. IEEE Trans. on Knowledge and Data Engineering, 2017,29(6):1310−1328.

[7] Hu HQ, Li GL, Bao ZF, Feng JH, Wu YW, Gong ZG, Xu YQ. Top-*k* spatio-textual similarity join. IEEE Trans. on Knowledge and Data Engineering, 2016,28(2):551−565.

[8] Wang Y, Jian X, Yang ZH, Li J. Query optimal *k*-plex based community in graphs. Data Science and Engineering, 2017,2(4): 274−274.

[9] Yang J, Zhang Y, *et al*. Categorical top-*k* spatial influence query. World Wide Web, 2017,20:175−203.

[10] Cong G, Jensen CS, Wu D. Efficient retrieval of the top-*k* most relevant spatial Web objects. Proc. of the VLDB Endowment, 2009, 2(1):337−348.

[11] Chan KH, Li C. Hybrid indexing and seamless ranking of spatial and textual features of Web documents. In: Proc. of the SSTD. 2017. 357−375.

[12] Li ZS, Lee KC, Zheng BH, Lee WC, Lee D, Wang XF. IR-tree: An efficient index for geographic document search. IEEE Trans. on Knowledge and Data Engineering, 2011,23(4):585−599.

[13] Rochajunior JB, Gkorgkas O, Jonassen S, Nørvåg K. Efficient processing of top-*k* spatial keyword queries. In: Proc. of the SSTD. 2011. 205−222.

[14] Liu XP, Wan CX, Liu DX, Liao GQ. Survey on spatial keyword search. Ruan Jian Xue Bao/Journal of Software, 2016,27(2): 329−347 (in Chinese with English abstract). http://www.org.jos.cn/1000-9825/4934.htm [doi: 10.13328/j.cnki.jos.004934]

[15] Zhang D, Tan KL, Tung AKH. Scalable top-*k* spatial keyword search. In: Proc. of the EDBT. 2013. 359−370.

[16] Zhang CY, Zhang Y, Zhang WJ, Lin XM. Inverted linear quadtree: Efficient top *k* spatial keyword search. In Proc. of the ICDE. 2013. 901−912.

[17] Wu DM, Yiu ML, Cong G, Jensen CS. Joint top-*k* spatial keyword query processing. IEEE Trans. on Knowledge and Data Engineering, 2012,24(10):1889−1903.

[18] Felipe ID, Hristidis V, Rishe N. Keyword search on spatial databases. In: Proc. of the ICDE. 2008. 656−665.

[19] Chen L, Xu JL, Lin X, Jensen CS, Hu HB. Answering why-not spatial keyword top-*k* queries via keyword adaption. In: Proc. of the ICDE. 2016. 697−708.

[20] Chan KH, Long C, Wong CW. Inherent-cost aware collective spatial keyword queries. In: Proc. of the SSTD. 2017. 357−375.

[21] Zhang DX, Chan CY, Tan KL. Processing spatial keyword query as a top-*k* aggregation query. In: Proc. of the SIGIR. 2014. 355–364.

[22] Wu DM, Cong G, Jensen CS. A framework for efficient spatial Web object retrieval. VLDB Journal, 2012,21(6):797–822.

[23] Chen LS, Cong G, Cao X, Tan KL. Temporal spatial-keyword top-*k* publish/subscribe. In: Proc. of the ICDE. 2015. 255–266.

[24] Liu J, Ke D, Sun H, Ge Y, Zhou X. Clue-based spatio-textual query. Proc. of the VLDB Endowment, 2017,10(5):529–540.

[25] Wang J, Gao H, Li J, Yang D. An index supporting spatial approximate keyword search on disks. Journal of Computer Research and Development, 2012,49(10):2142–2152 (in Chinese with English abstract).

[26] Hu J, Fan J, Li GL, Chen SS. Top-*k* fuzzy spatial keyword search. Chinese Journal of Computers, 2012,35(11):2237–2246 (in Chinese with English abstract).

[27] Yao B, Li FF, Hadjieleftheriou M, Hou K. Approximate string search in spatial databases. In: Proc. of the ICDE. 2010. 545–556.

[28] Lin X, Xu JL, Hu HB. Reverse keyword search for spatio-textual top-*k* queries in location-based services. In: Proc. of the ICDE. 2017. 375–386.

[29] Zheng K, Zhou XF, Fung PC, Xie KX. Spatial query processing for fuzzy objects. VLDB Journal, 2012,21(5):729–751.

[30] Cary A, Wolfson O, Rishe N. Efficient and scalable method for processing top-*k* spatial Boolean queries. In: Proc. of the SSDBM. 2010. 87–95.

[31] Gao YJ, Qin X, Zheng BH, Chen G. Efficient reverse top-*k* Boolean spatial keyword queries on road networks. IEEE Trans. on Knowledge and Data Engineering, 2015,27(5):1205–1218.

[32] Chen G, Zhao JW, Gao YJ, Chen L, Chen R. Time-aware Boolean spatial keyword queries. IEEE Trans. on Knowledge and Data Engineering, 2017,29(11):2601–2614.

[33] Zhao PP, Fang HL, Sheng VS, Li ZX, Xu JJ, Wu J, Cui ZM. Monochromatic and bichromatic ranked reverse Boolean spatial keyword nearest neighbors search. World Wide Web, 2017,20(1):39–59.

[34] Li GL, Feng JH, Xu J. DESKS: Direction-aware spatial keyword search. In: Proc. of the ICDE. 2012. 474–485.

[35] Wu DM, Yiu ML, Jensen CS, Cong G. Efficient continuously moving top-*k* spatial keyword query processing. In: Proc. of the ICDE. 2011. 541–552.

[36] Huang WH, Li GL, Tan KL, Feng JH. Efficient safe-region construction for moving top-*k* spatial keyword queries. In: Proc. of the CIKM. 2012. 932–941.

[37] Shi JM, Wu DM, Mamoulis N. Textually relevant spatial skylines. IEEE Trans. on Knowledge and Data Engineering, 2016,28(1): 224–237.

[38] Choudhury FM, Culpepper JS, Sellis T, Cao X. Maximizing bichromatic reverse spatial and textual *k* nearest neighbor queries. Proc. of the VLDB Endowment, 2016,9(6):456–467.

[39] Xie X, Lin X, Xu J, Jensen CS. Reverse keyword-based location search. In: Proc. of the ICDE. 2017. 375–386.

[40] Aizawa K, Motomura K, Kimura S, Kadowaki R, Jia F. Constant time neighbor finding in quadtrees: An experimental result. In: Proc. of the Int'l Symp. on Communications, Control and Signal Processing. 2008. 505–510.

[41] Bao J, Zheng Y, Mokbel MF. Location-based and preference-aware recommendation using sparse geo-social networking data. In: Proc. of the SIGSPATIAL. 2012. 199–208.

[42] Wei LY, Zheng Y, Peng WC. Constructing popular routes from uncertain trajectories. In: Proc. of the KDD. 2012. 195–203.

附中文参考文献:

[5] 韩希先,杨东华,李建中.TKEP:海量数据上一种有效的 Top-*K* 查询处理算法.计算机学报,2010,33(8):1405–1417.

[14] 刘喜平,万常选,刘德喜,廖国琼.空间关键字搜索研究综述.软件学报,2016,27(2):329–347. http://www.org.jos.cn/1000-9825/4934.htm [doi: 10.13328/j.cnki.jos.004934]

[26] 胡骏,范举,李国良.空间数据上 Top-*k* 关键词模糊查询算法.计算机学报,2012,35(11):2237–2246.

潘晓(1981－),女,博士,副教授,CCF 专业会员,主要研究领域为数据管理,数据挖掘,移动计算,隐私保护.

于启迪(1996－),女,学士,主要研究领域为数据管理,查询,移动计算.

马昂(1992－),女,硕士生,主要研究领域为数据管理,数据挖掘,移动计算,隐私保护.

孙亚欣(1994－),女,学士,主要研究领域为数据管理,查询,移动计算.

吴雷(1980－),男,博士生,讲师,CCF 专业会员,主要研究领域为数据管理,数据挖掘,移动计算,隐私保护.

郭景峰(1962－),男,博士,教授,博士生导师,CCF 高级会员,主要研究领域为数据库理论及应用,数据挖掘,社交网络,图像处理.