空间数据上 Top-k 关键词模糊查询算法

胡 骏 范 举 李国良 陈姗姗
(清华大学计算机科学与技术系数据库研究组 北京 100084)

摘 要 基于位置的服务(LBS)变得日益普及,越来越多的研究开始关注如何对空间中的兴趣点(POI)做有效的检索.现有的方法提出了空间数据上的关键词检索,研究如何根据查询的位置和关键词找到相关的 POI 点.然而,现有方法主要对查询关键词进行精确匹配,不能支持模糊查询:当查询关键词与底层数据存在微小差异的时候,LBS 系统不能返回相关的结果.为了满足移动用户的模糊查询需求,文中对空间数据上的 Top-k 关键词模糊查询问题进行研究:给定一组 POI 点,检索与查询关键词近似匹配且空间上距离相近的 Top-k 个结果.为了提供高效的模糊查询,文中首先定义了一种新型的相关性函数,综合考虑了文本相似性和空间距离,进而提出了一种有效的索引结构 RegionTrie,并基于 RegionTrie 设计了高效的 Top-k 算法.真实数据集上的实验结果表明,文中提出的 Top-k 算法十分高效,性能远好于对比方法.

关键词 基于位置的服务;空间数据上的关键词检索;字符串近似匹配
中图法分类号 TP311 **DOI 号**: 10.3724/SP.J.1016.2012.02237

Top-k Fuzzy Spatial Keyword Search

HU Jun FAN Ju LI Guo-Liang CHEN Shan-Shan
(*Database Research Group, Department of Computer Science and Technology, Tsinghua University, Beijing* 100084)

Abstract Location-Based Services (LBS) have become more and more popular recently. Existing LBS systems employ a spatial keyword search method to provide services, which finds the relevant POIs by considering textual relevance and spatial distance when given a set of points-of-interest (POIs). Existing methods only allow exact matches for query keywords and fail to support fuzzy search. To provide error-tolerance search experiences, we study the top-k fuzzy spatial keyword search problem in this paper: Given a set of POIs and a query with location and keywords, we find the relevant POIs having similar keywords with the query. It calls for efficient algorithms to provide real-time search for mobile users. To address this challenge, we introduce a novel function to quantify the relevance between POIs and the query, by considering the similarity between keywords and spatial distance. Then, we devise an effective index structure, called RegionTrie to organize the POIs and develop efficient search algorithm based on the RegionTrie. We conducted experiments on real datasets, and the experimental results show that our algorithms achieve high performance.

Keywords location-based services; spatial keyword search; string similarity search

1 引 言

近年来,由于移动设备(如智能手机)上全球定位系统 GPS 的普及,基于位置的服务(LBS)得到了学术界和工业界的广泛关注.很多基于位置的服务,如 AT&T 公司的位置信息服务[①]等,都得到了普及和应用,带给了用户位置相关的检索体验.

收稿日期:2012-06-05;最终修改稿收到日期:2012-09-04.**胡 骏**,男,1990 年生,硕士研究生,主要研究方向为数据库. E-mail: hjguyue@gmail.com. **范 举**,男,1984 年生,博士,主要研究方向为非结构化和结构化数据查询、万维网数据管理、空间和文本上的查询和推荐. **李国良**,男,1980 年生,博士,讲师,主要研究方向为数据清洗与融合、文本空间数据处理、众包数据管理. **陈姗姗**,女,1989 年生,主要研究方向为数据库.

① http://www.wireless.att.com/lbs/

现有的 LBS 系统采用关键词检索的方式帮助用户从空间数据库中找到位置相关的结果[1-11]. 具体来说,假设空间数据库中有一组兴趣点(Points Of Interest,POI 点),其中每个 POI 点都包含位置信息和一定的文本信息. 给定用户的位置和一组查询关键词,LBS 系统返回从空间和文本上都与查询相关的 POI 点. 举例来说,假设用户希望找到附近的星巴克咖啡店,她可以输入关键词"starbucks",系统会根据 POI 点与查询位置的距离以及是否包含关键词"starbucks"返回相关的 POI 点作为结果.

现有方法需要 POI 点精确地匹配到查询关键词,不支持关键词的模糊匹配. 然而,由于移动设备自身的特点,如屏幕小、键盘输入不方便等,用户很可能在输入关键词的时候出现错误. 例如,用户可能将关键词"starbucks" 误输为"sterbuck",从而无法得到想要的结果. 针对这一问题,Yao 等人[6]提出了空间数据上的字符串近似匹配方法,该方法使用编辑距离来度量查询与 POI 点中关键词的相似性,并设定一个阈值:编辑距离小于等于该阈值时,则认为相应的关键词彼此近似. 例如,给定阈值 2,关键词"sterbuck"与"starbucks"彼此近似,因为它们的编辑距离为 2,小于等于阈值. 因此,当查询关键词为"sterbuck"时,也可以模糊匹配到包含"starbucks"的 POI 点,从而实现容错的目的. 该方法存在以下两点局限性:首先,对于不同的查询关键词,很难设定一个统一的阈值:较长的查询可能需要比较大的阈值;相反,较短的查询需要的阈值也会小一些. 其次,Yao 等人仅提供了一种近似的解法,即结果可能存在着一定的误差,可能会引入错误以及遗漏结果.

本文提出一种空间数据上 Top-k 关键词模糊查询方法,帮助用户在输入存在错误的情况下,找到最相关的 Top-k 个 POI 点. 提出的方法同样使用编辑距离来度量关键词之间的相似性. 然而,与现有方法相比,该方法不需要设定阈值,而是将编辑距离融入到结果的排序函数中, 因此比现有的方法更加灵活. 另一方面,本文提出的方法能够根据排序函数找到精确的 Top-k 结果,避免了近似算法可能引入的错误.

提供空间数据上的 Top-k 关键词模糊查询存在以下几点挑战:首先,需要设计支持容错的相关性函数,从而判断哪些 POI 点与查询更加相关;其次,需要设计高效的 Top-k 算法,为移动用户返回实时的查询结果. 针对这些挑战,本文首先提出了一种新型的相关性函数,综合考虑编辑距离、关键词权重和空间距离. 给定相关性函数,设计了一种有效的结构 RegionTrie 对空间数据库中的 POI 点进行索引,该索引根据文本的前缀和空间分布将数据中的 POI 点组织成一个层次的结构. 基于 RegionTrie,我们设计了高效的 Top-k 算法,该算法可以通过增量的方式对编辑距离进行计算,并有效地估计查询与 POI 点的空间距离. 此外,本文还设计了有效的剪枝策略,从而进一步提高算法的效率.

综上所述,本文的贡献在于:

(1) 首次提出了空间数据上的 Top-k 关键词模糊查询问题,从而支持查询的容错.

(2) 提出了有效的索引结构 RegionTrie,并基于该结构设计了高效的 Top-k 算法.

(3) 在真实数据集上进行了实验,实验结果表明,提出的方法具有很高的检索效率.

本文第 2 节对相关的工作进行综述;第 3 节给出问题的形式化描述;第 4 节提出 RegionTrie 的索引结构,并给出基于该索引的 Top-k 算法;第 5 节进一步对 Top-k 算法进行优化,提出增量计算的方案;第 6 节报告实验的结果;最后,第 7 节对全文进行总结.

2 相关工作

本节对两方面的相关工作:空间数据上的关键词检索和字符串近似查询进行综述.

空间数据上的关键词检索得到了非常广泛的研究[1-11]. 根据查询模型的不同可以将现有研究分为:(1) k 近邻查询,返回与查询文本相关且距离最近的 k 个 POI 结果. 这方面现有的工作多将文本索引融合到空间的 R 树索引中[3,5];(2) 区域查询,返回与查询文本上相关且空间上在一定区域内的 POI 结果[1-2]. 为了支持容错,Yao 等人[6]提出了空间数据上的字符串近似匹配方法,通过设定编辑距离的最大阈值找到与查询关键词不一定完全一致的 POI 点,并基于 R 树给出了一种近似的检索算法.

与现有的研究相比,本文主要研究 Top-k 关键词模糊查询问题,将编辑距离融合到排序的相关性函数中,因而无需给定编辑距离的阈值,提供了更加灵活的查询容错方式;同时,研究精确的 Top-k 算法,避免了近似算法带来的错误.

字符串近似查询也得到了广泛的研究[12-17],主要考虑给定一组字符串,如何从中找到与查询字符串最相似的字符串. 很多工作都使用编辑距离来衡

量字符串的相似性，计算相似性大于一定阈值的字符串. Xiao 等人[11]讨论了一种 Top-k 方法，即找到与查询最相似的 k 个字符串，其基本的思路是不断地变化编辑距离的阈值，从而找到前 k 个结果.

本文讨论的问题与字符串近似查询也有着显著的不同. 除了文本相似性，本文研究的问题需要进一步考虑空间距离，因为其更为复杂.

3　问题的形式化描述

数据模型：空间数据库中的一组 POI 对象 $O=\{o_1,o_2,\cdots,o_{|O|}\}$. 其中任一对象 $o\in O$ 包含空间位置信息 l 和文本信息 T，表示为 $o=(l,T)$. 具体来说，空间信息 $o.l$ 为二维空间内的地理位置点 (x_l,y_l)；本文信息 $o.T=\{t_1,t_2,\cdots,t_{|T|}\}$ 为一组关键词的集合. 每个关键词 $t\in o.T$ 都带有一个权重 $w(t)$，用来表示关键词 t 对于 o 的重要程度. 本文采用信息检索领域经典的 TF-IDF 模型计算关键词的权重，即

$$w(t)=\mathrm{tf}(t,T)\cdot\mathrm{idf}(t,O) \tag{1}$$

其中，词频 $\mathrm{tf}(t,T)$ 为 t 出现在 $o.T$ 中的频率 $\frac{\mathrm{count}(t,T)}{|T|}$；逆文档频率 $\mathrm{idf}(t,O)$ 为包含关键词 t 的对象个数的倒数，可以通过 $\log\frac{|O|}{\mathrm{count}(t,O)+1}$ 进行计算. 图 1 给出了 7 个对象 $o_1\sim o_7$ 的示例，以对象 o_1 为例，它包含了空间上的一个位置点和两个关键词 t_1 和 t_2. 通过式(1)，可以计算出这两个关键词的权重.

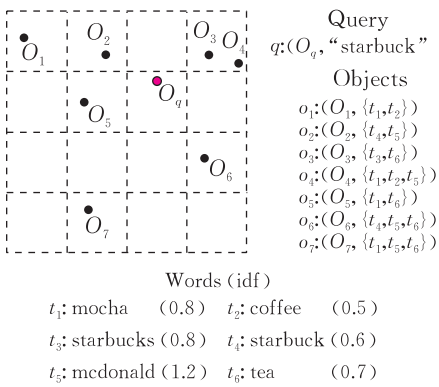

图 1　空间数据上 Top-k 模糊关键词查询示例

查询模型：空间相关的关键词查询 q，同样包含空间位置 $q.l$ 和文本信息 $q.T$. 为了对问题进行简化，本文主要讨论单关键词的情况，即查询 q 仅包含一个关键词 $q.t$，并把多关键词查询问题作为下一步的研究工作. 查询的结果是集合 O 中在空间和文本上都与查询 q 最相关的前 k 个对象(简称为 Top-k 结果). 下面具体定义文本和空间相关性.

对于文本，本文允许查询关键词 $q.t$ 与对象中的关键词 $o.t$ 不完全一致，存在微小的差异. 具体来说，要考虑以下两方面的因素：

(1) 查询关键词 $q.t$ 与对象中任意关键词 $o.t\in o.T$ 之间的编辑距离 $ed(q.t,o.t)$. 编辑距离定义为将一个关键词变成另一个关键词所需要最少的编辑操作(即查询、删除和修改)个数. 例如，关键词“starbucks”和“sterbuck”的编辑距离为 2，因为可以通过一个替换操作(将 a 替换为 e)和一个删除操作(删掉最后一个 s)将前一个关键词变成后一个. 编辑距离越小，$q.t$ 与 $o.t$ 越相似，查询与对象 o 在文本上也就越相关.

(2) 关键词 $o.t$ 对于对象 o 的权重 $w(t)$. 权重越大，则关键词 $o.t$ 对于对象 o 就越重要，查询与对象 o 在文本上也就越相关.

下面给出文本相关性的形式化定义. 考虑查询关键词 $q.t$ 和对象 o 的文本信息 $o.T$，文本相关性表示为分值 $S_{\mathrm{T}}(q,o)$，并通过下式进行计算：

$$S_{\mathrm{T}}(q,o)=\frac{w(t^*)/w_{\max}}{[1+ed(q,t^*)]^m} \tag{2}$$

其中，关键词 t^* 为 $o.T$ 中与查询 $q.t$ 编辑距离最小的关键词，即 $t^*=\arg\min_{t\in o.T}\{ed(q.t,o.t)\}$；$w_{\max}$ 为全局最大的关键词权重，其目的是将权重规范到区间 $[0,1]$；参数 m 用来调节权重和编辑距离的重要程度，实验中取 $m=2$.

另一方面，本文采用欧几里德距离来衡量查询与对象的空间相关性：距离越小，则认为对象与查询就越相关. 可以将空间相关性表示为分值 $S_{\mathrm{L}}(q,o)$，并通过下式进行计算：

$$S_{\mathrm{L}}(q,o)=1-\frac{\mathrm{dis}(q,o)}{\mathrm{dis}_{\max}} \tag{3}$$

其中 $\mathrm{dis}(q,o)$ 为空间上点 $q.l$ 与 $o.l$ 之间的欧几里德距离，$\mathrm{dis}_{\max}$ 为全局可能出现的最大距离，其目的是将距离规范到区间 $[0,1]$.

将文本相关性和空间相关性进行线性组合，可以得到查询 q 与对象 o 之间的综合相关性：

$$S(q,o)=\alpha S_{\mathrm{T}}(q,o)+(1-\alpha)S_{\mathrm{L}}(q,o) \tag{4}$$

其中参数 α 用来调节文本与空间的相对重要程度. 基于以上的模型，可以对问题进行形式化定义：

问题定义：考虑一组对象 $O=\{o_1,o_2,\cdots,o_{|O|}\}$. 给定查询 $q=(l,t)$，根据相关性函数 $S(q,o)$ 对所有的对象进行打分，并计算出分值最大的 Top-k 个对象.

下面通过一个实例对问题做进一步的说明. 图 1

给出了空间上的 7 个对象 $o_1 \sim o_7$ 和查询 q，考虑 $k=3$的时候，Top-k 的结果应该为$\{o_3, o_2, o_6\}$. 从文本上看，对象 o_3 中包含关键词“starbuck”，与查询关键词的编辑距离为 0；对象 o_2 和 o_6 中包含关键词“starbucks”，与查询关键词的编辑距离为 1；从空间上看，o_3 和 o_2 跟查询 q 的距离最小，o_6 的位置则远一些. 综合文本和空间，可以看出：对象 o_3 的分值最大，o_2 和 o_6 的分值次之. 它们的分值比其它对象大，为 Top-3 结果.

4 基于索引 RegionTrie 的算法

本节给出一种基于索引 RegionTrie 的 Top-k 算法. 第 4.1 节对索引结构 RegionTrie 进行定义；第 4.2 节给出 RegionTrie 的构建方法；第 4.3 节介绍如何使用 RegionTrie 计算与查询最相关的 Top-k 结果.

4.1 RegionTrie 结构

本文提出一种新的索引结构——RegionTrie. 该索引结构基于传统的前缀树（即 Trie）索引. 在前缀树中，每个结点存储一个字母；任意一条从根结点到叶子结点的路径表示一个完整的单词；根结点到非叶子结点表示一个前缀. 通过前缀树，可以便利地找到具有某一前缀的所有完整关键词. 与前缀树相比，RegionTrie 进一步扩充了空间信息. 具体来说，给定一个前缀，包含该前缀的所有对象具有一定的空间分布. 例如在图 1 的实例中，包含前缀“m”的对象为 o_1 和 o_5，它们分布在空间的左上方. 为了记录这部分空间信息，RegionTrie 为每个前缀都扩充一个或多个区域，用来表示包含该前缀且分布在区域内的对象.

下面给出 RegionTrie 的形式化描述. RegionTrie 为一个层次结构，其中每个结点 n 都对应前缀$n.p$和一个空间区域$n.R$. 特别地，父亲结点 n 和任一孩子结点 n_c之间具有以下的性质：

(1) 前缀包含性：即父亲结点对应的前缀$n.p$是孩子结点对应前缀$n_c.p$的前缀.

(2) 区域包含性：即父亲结点对应的区域$n.R$在空间上包含孩子结点对应的区域$n_c.R$.

不难看出，RegionTrie 实际上是通过不同的前缀和区域将 O 中的对象进行了层次化的组织. 不同于传统的 Trie 树，RegionTrie 中同一个前缀可能对应着多个结点，因为它们在空间上对应了不同的区域.

下面给出 RegionTrie 的一个实例（如图 2 所示）. 考虑左图中的 3 个对象，分别包含关键词“abc”、“abd”和“ac”. 右图中的结点‘a’对应着前缀‘a’和左图中全区域. 右图中的孩子结点（‘b’,0）对应着前缀“ab”和左图中的小区域 0（第 4.2 节讨论如何获得子结点所在的区域）. 可以看出：全区域包含区域 0；且“a”是“ab”的前缀. 同理，孩子结点（‘b’,3）和（‘c’,1）与父结点也具有前缀和区域的包含性.

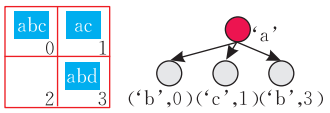

图 2 RegionTrie 的一个实例

4.2 RegionTrie 的构建

本文采用自顶向下的算法建立 RegionTrie. 对于对象集合 O，算法一方面通过前缀包含性，另一方面通过空间划分将对象组织在层次化的 RegionTrie 上. 特别地，本文中采用和索引结构 QuadTree 类似的策略对空间进行划分. 图 3 和图 4 为示例图.

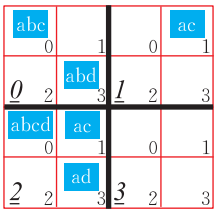

图 3 建立 RegionTrie 时的空间划分示例

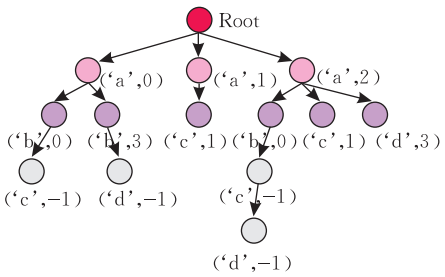

图 4 根据图 3 建立的 RegionTrie 示例

具体的算法如下：从根结点开始，将空间 4 等分，对于每一小块空间，建立传统的 Trie 树的第 1 层，4 小块空间共对应 4 个子 Trie 树的第 1 层，每个结点都要带上相应的区域编号（如图 3 所示）. 接着继续 4 等分 4 个小空间得到更小的空间，并分别在自己的空间内建立传统 Trie 树的下一层，并标上区域编号. a 按此不断继续，先划分空间，再建立子 Trie 树，以此类推. 当空间划分足够精细，即划分达到一定深度的时候即可停止对空间划分，并在以划分出的最小空间上直接建立剩余部分的 Trie 树.

在示例的 RegionTrie 建立过程中，假如划分两

层之后满足空间的精细程度，则设定划分限制为两层，图 4 中可见第 3 层之后就把区域标号均标上了‘−1’.

4.3　基于 RegionTrie 的 Top-k 算法

首先在对象集合 O 上建立一棵 RegionTrie 树，在每个叶子结点 $leaf=(p,R)$ 放一个倒排列表，记录 O 中哪些对象既包含关键词 p 又被区域 R 所包含. 形式化地，倒排列表中的每一项代表空间数据库 O 中的一个对象 o，它满足 $p\in o.T$ 且 $o.l\in R$. 根据 RegionTrie 建立的规则，倒排列表中记录的位置一定在该结点对应的区域之内. 倒排列表中每一项的顺序按照对应词在文本中的权重来排序，由从大到小的顺序来排.

为了得到 Top-k 结果，本节采用递增编辑距离的方式：优先考虑编辑距离最小的对象，再考虑空间相关性，同时进行文本和空间的大量剪枝.

算法描述：从编辑距离等于 0 开始检索，首先得到编辑距离为 0 的词，也就是找到所有编辑距离为 0 的叶子结点，然后遍历倒排列表，对于每一项，根据位置信息和该词的权重计算出综合分数值，放到一个优先队列(最小堆)中，这个最小堆保持 k 的大小，堆内的大小按照每个点的综合分数值来衡量. 当加入一个新的点之后使得堆的大小等于 $(k+1)$ 时，删掉堆顶的点，从而保持前 k 个最大的综合分值 $S(q,o)$. 计算完编辑距离为 0 的结点之后，计算编辑距离为 1 的结点，再计算编辑距离为 2 的点，如此不断地增长. 当满足如下两个条件之一时便可以停止下来：

(1) 确保已经得到了 Top-k 结果. 前面提到把当前的结果集存放在一个最小堆中，这个最小堆的顶点就是代表了当前的第 k 个大的综合分数值，把它作为一个界. 如果能确保剩余所有点按照最好情况计算，它们的分数值都已经比这个界小，则停止计算.

(2) 编辑距离太大的情况，定义上限，在计算到编辑距离等于上限时，就不再增长编辑距离了.

给出算法的伪代码如下：

算法 1.　基于 RegionTrie 的 Top-k 算法.

```
search(q, k)
Input: 查询 q, 数值 k
Output: 存放 Top-k 结果的优先队列 topKQueue
1. initialSearch();
2. STOP=false;
3. while(!STOP){
4.     nodes=getNodes(editDistance);
5.     topKQueue=processNodes(nodes, topKQueue);
6.     editDistance++;
7.     STOP=updateState();
   }
Function  processNodes (nodes, topKQueue)
Input: 一组结点 nodes, 优先队列 topKQueue
Output: 存放 Top-k 结果的优先队列 topKQueue
1. for(Node node:nodes){
2.     for(Entry entry:node.invertedList){
3.         score=calculate(entry);
4.         topKQueue.add(score,entry);
5.         if(topKQueue.size()>K)
6.             topKQueue.peek();
7.     }
8. }
9. return topKQueue.
Function  updateState ( )
Output: STOP: true or false (should be stopped or not)
1. if(editDistance>=limit)
2.     return true;
3. bestScore=estimate();
4. if (bestScore<bound)
5.     return true.
```

上面代码中 search 函数为整个核心框架，其中 getNodes 函数为给定编辑距离下，求出词库中所有与查询词编辑距离为给定值的词所在的结点. processNodes 函数为处理这些结点的函数，主要是完成计算结点上的倒排列表的工作. updateState 函数更新全局状态，判断是否停止. 关于 getNodes 函数下面会详细说明.

回到前面的例子，我们在计算过程始终维护着一个大小为 K 的优先队列，这个优先队列是一个最小堆，把堆顶的分数作为界. RegionTrie 的特性可以快速地把例子中离 query 很远的 o_1 和 o_7 先排除了.

5　基于 RegionTrie 的增量计算方法

5.1　增量式地计算编辑距离

仔细分析上面的算法过程，可以发现每次更新编辑距离时，都会从根结点开始重新计算. 事实上，给定查询词 $q.t$，如果已经计算出某个词，如“abcde”和 $q.t$ 的编辑距离后，可以增量地估计“abcdef”和 $q.t$ 的编辑距离，从而减少不必要的计算. 举例来说，当计算完编辑距离为 0 之后开始计算编辑距离为 1 的情形，需重新计算，没有利用编辑距离为 0 的中间结果.

下面讨论如何增量地计算编辑距离. 计算编辑距离的经典动态规划方法可以形象地用填写表格的方式来表示. 例如，当我们来计算“mocha”和“monica”之间的编辑距离时，如下图 5(a) 所示.

		m	o	c	h	a
	0	1	2	3	4	5
m	1	0	1	2	3	4
o	2	1	0	1	2	3
n	3	2	1	1	2	3
i	4	3	2	2	2	3
c	5	4	3	2	3	3
a	6	5	4	3	3	3

(a)

		m	o	c	h	a
	0	1	2	3		
m	1	0	1	2	3	
o	2	1	0	1	2	3
n	3	2	1	1	2	3
i		3	2	2	2	3
c			3	2	3	3
a				3	3	3

(b)

图 5 填表格计算编辑距离

图 5(a)的图即为传统算法,图 5(b)是对传统算法的改进,只计算必要的格子. 基于上述观察,图 6 给出了具体的计算方法. 对于图中的表格,最上一行表示查询词 q,最左列表示某个词 w,表格灰色处格子填写的是编辑距离,现在要计算 q 和 w 之间的编辑距离. 观察灰色表格中第$[i,j]$位置的格子,里面已经填写了编辑距离 e. 此时,对于格子$[(i+1),j]$和$[i,(j+1)]$,它们的编辑距离均为$(e+1)$;对于格子$[(i+1),(j+1)]$,取决于 q 的$(j+1)$和 w 的$(i+1)$的字符是否相同,相同的话就是 e,不相同的话就是$(e+1)$.

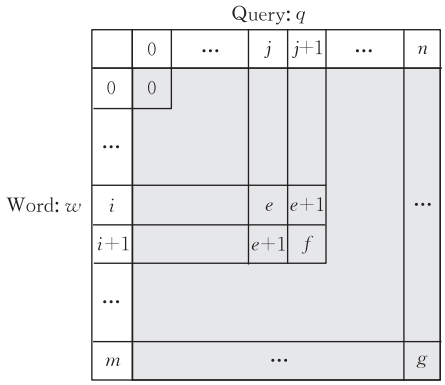

图 6 编辑距离的增量计算方法

假设表格已经填完了编辑距离小于等于 e 的格子,现在要填满编辑距离为$(e+1)$的格子. 可以这样完成,对于每一个编辑距离为 e 的格子.

步骤 1. 纵向横向直接加 1.

对于已经填了 e 的格子,它的右方和下方的格子如果还没有填写,直接加 1.

步骤 2. 右下方的格子延伸判断.

对于已经填了 e 的格子,它右下方的格子有可能是 e,也有可能是$(e+1)$,前面已经说明. 如果是 e 的话我们称它满足“匹配”条件,在这个格子中填上 e 之后,还需要继续判断它的右下方,因为它右下方的格子有可能继续满足“匹配”条件,需要不断地往右下方寻找,直到填满所有右下方为 e 的格子;如果是$(e+1)$的话,也就是直接不满足“匹配”条件,此时在这个格子中填上$(e+1)$就可以停止了.

编辑距离的增量计算方法:根据编辑距离的大小,按照从小到大的顺序,依次在表格中标记数字,每个数字都代表着字符串 w 的一个子串和查询 q 的一个子串之间的编辑距离. 首先,标记完所有编辑距离为 0 的格子,再标记编辑距离为 1 的格子,之后再标记编辑距离为 2 的格子,以此类推. 直到整个表格最右下角的格子被标记了数字就停止,因为此时已经得到了最终要求的编辑距离,最右下角的数字就代表了 q 和 w 之间的编辑距离.

5.2 基于 RegionTrie 的增量计算思想

前面的“填表格算法”可以得到的只是一个词和查询词之间的编辑距离表格,但是需要求的是对象集合 O 中的所有词和查询词之间的编辑距离,一种比较直接的方法是遍历所有的词,并计算编辑距离. 显然,这种方法的效率较低,下面讨论如何结合 RegionTrie 结构和上述增量计算的思想进行求解.

首先定义 RegionTrie 树的结点 n 和查询词 q 的编辑距离. 考虑从根结点到结点 n 对应的字符串 s,本文将结点 n 与查询词 q 之间的编辑距离定义为 s 与 q 之间的编辑距离. 如果把查询词 q 的前 j 个字符组成的子串定义为 $q[1,j]$,则结点 n 和 $q[1,j]$的距离表示结点 n 对应的字符串 s 和子串 $q[1,j]$之间的编辑距离.

而这个编辑距离可以对应到表格上的一点,如图 7 所示.

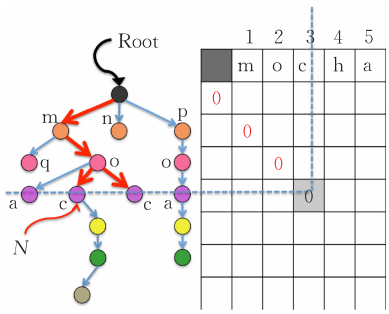

图 7 RegionTrie 计算编辑距离示意图 1

上面图 7 中的结点 N 和 $q[1,3]$(即“moc”)之间的距离可以在表格中的深色格子中表示出来,值为结点 N 对应的字符串“moc”和 $q[1,3]$之间的编辑距离 0. 同样,图 8 中结点 M 和 $q[1,3]$(即“moc”)之间的距离为 M 对应的字符串“poa”和“moc”之间的编辑距离 2,如图 8 表格中深色格子所示.

我们把结点 N 和 $q[1,3]$表示为一个组合$(N,3)$,它的值定义为前面描述的距离 0;把结点 M 和

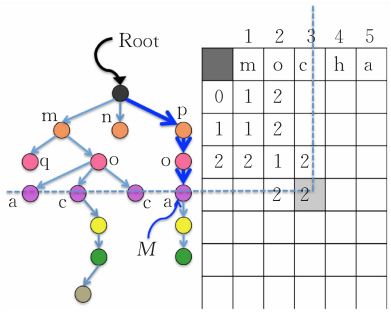

图 8　RegionTrie 计算编辑距离示意图 2

$q[1,3]$表示为组合$(M,3)$，它的值定义为前面描述的距离 2. 接下来要给出算法递增式地找到所有编辑距离为 e 的组合. 把所有编辑距离为 e 的组合的集合记为 $E[e]$，假设现在已经得到了 $E[0],E[1],\cdots,E[e-1]$，需要找到 $E[e]$的所有元素.

对于 $E[e-1]$中的任意一个元素(N,j)，N 是 RegionTrie 树上的结点，进行下面两种类型的扩展.

(1) 普通扩展

对 N 的任意一个子结点 Nc，如果 Nc 结点存储的字符和查询词 q 的第$(j+1)$位不同，则可以知道从根结点到 Nc 的字符串与查询词的子串 $q[1,j+1]$之间的编辑距离为 e，于是可以把$(Nc,j+1)$加入到 $E[e]$中. 另外，不难知道(Nc,j)的值和$(N,j+1)$的值也是 e，所以把(Nc,j)和$(N,j+1)$加入 $E[e]$之中. 我们把这个过程叫做普通扩展.

(2) 延伸扩展

在普通扩展之后，如果把$(Nc,j+1)$加入到 $E[e]$中，则应该继续对结点 Nc 进行分析. 因为，当 Nc 还有子结点的字符和查询词 q 的$(j+2)$位相同时，还需要继续往下扩展…，直到所有子结点的字符和查询词的下一个字符都无法吻合时，才可以停止扩展. 我们把这个过程叫做延伸扩展.

5.3　基于 RegionTrie 的增量算法

初始化：将(Root,0)加入 $E[0]$集合中，其中 Root 为 RegionTrie 树的根结点. 此时，$E[0]$还没有填满，需要对(Root,0)进行延伸扩展，才能得到完整的 $E[0]$.

接下来，假设已经得到了完整的 $E[0],E[1],\cdots,E[e-1]$，通过扩展 $E[e-1]$来得到 $E[e]$. 具体过程是，对于每一个 $E[e-1]$的元素，先进行普通扩展，再进行延伸扩展，最后得到 $E[e]$. 注意每个组合(N,j)最多只出现一次，也就是只属于一个 $E[e]$，以最先出现的为准，之后扩展过程再次出现就直接跳过.

可以根据上述思路替换 RegionTrie 树的 Top-k 算法：每次更新编辑距离为 e 时，就利用 RegionTrie 树上的“编辑距离递增”算法，从 $E[e-1]$得到$E[e]$. 而 $E[e]$中的所有叶子结点对应的字符串即为词库中与查询词 q 编辑距离为 e 的词.

```
Function　getNodes(editDistance)
Input：编辑距离的值 editDistance
Output：RegionTrie 中的一组结点 nodes
1. extend the E[editDistance−1] to get E[editDistance];
2. for(Entry entry:E[editDistance]){
3.      node=entry.node; j=entry.position;
4.      if(node is a leaf node && j==query.length)
5.          nodes.add(node);
6. }
```

getNodes 函数通过扩展全局的 $E[e-1]$得到 $E[e]$，之后判断 $E[e]$中哪些元素既是叶子节点又到了查询词 q 的最右端，如满足则加入要处理的节点集合中.

5.4　剪枝策略

RegionTrie 树上的“编辑距离递增”算法利用了编辑距离小于等于$(e-1)$的中间信息，但随着查询字符串长度的增长，RegionTrie 就往下走得越深.

长度剪枝：针对编辑距离的最大限制，可以在长度上进行过滤. 例如，查询字符串长度为 n，如果允许的编辑距离限制为 L，那么符合条件的词一定满足长度在$(n-L)$到$(n+L)$之间. 所以，让每个结点记录子结点中(如果自身是叶子结点还要包括自身)的词的长度集合，当这个长度集合中没有一个长度在$(n-L)$到$(n+L)$之间时，可以不再考虑这个点和它的子结点.

位置剪枝：位置剪枝[18]是长度剪枝的进一步优化，它基于之前提到的编辑距离增量计算方法.

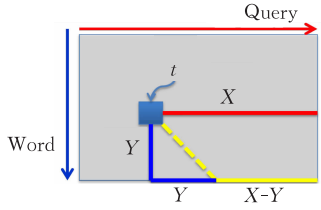

图 9　编辑距离的位置剪枝示意

设 t 为当前格子的编辑距离，$X-Y$ 是接下去至少要增长的编辑距离数，$limit_{editDistance}$ 为编辑距离最大限制. 剪枝条件为 $t+(X-Y)>limit_{editDistance}$. 当满足上面的条件时，便可以剪枝，从而大大减少扩展元素的个数.

6 实 验

6.1 实验设置

数据集:通过 twitter 的 API 抓取移动用户发的 tweet,每条 tweet 的信息包含了文字内容和发 tweet 时的地点,可以看作一个 POI 点.实验中抓取了一百万条 tweet,对每一条的文本进行了分词并滤掉停用词(包括标点和一些其它符号),统计得到数据集中共包含不重复的词 289 633 个.

查询集:分为短查询集和长查询集,每个集合包含 100 条查询.其中位置信息随机抽取,文本信息按照查询词的长度分为两种类型:平均长度在 8 以内的为短查询集,大于 8 的为长查询集.查询词在文档频率较高的词中随机抽取,这样就可以避免一些特别生僻的词(只出现过一次,查询 Top-k 在文本和空间上都没有近似结果).

参数设置:文本相关性式(2)中,参数 m 设置为 2;RegionTrie 树建立时,空间划分最大深度取 4;在综合相关性式(4)中,参数 α 先给定 0.5 进行对比,第 2 步测试分析 α 从 0.1~0.9 变化的情况.

实验环境:处理器为 2.3 GHz Intel Core i5;内存为 4 GB;操作系统为 Linux.

6.2 对比方法

为了将本文提出的索引结构 RegionTrie 树以及基于 RegionTrie 树的增量式算法进行对比,实验考虑以下对比方法.

对比方法采用传统的前缀树(Trie)来作为文本的索引,并在每个叶子结点上维护倒排列表,存储包含相应关键词的 POI 点.和 RegionTrie 树不同的是,Trie 树上的结点不带有位置信息.为了提高基于 Trie 方法的效率,也在每个结点上存储了它以及子结点中的最大文本权重值.

对比方法的框架和本文讨论的基于 RegionTrie 的 Top-k 算法基本一致,核心的框架还是 search 函数.其中 search 函数中调用的 getNodes 函数有所不同,原来返回的是 RegionTrie 树上的结点,而现在返回的是 Trie 树上的结点.同时 getNode 函数中选取结点的策略也不同,这里选取结点的策略参考文献[19],该策略可以返回指定编辑距离下 Trie 树上的结点.

6.3 实验结果

索引大小. 结点数和整个树形索引结构(包括倒排列表)的存储空间比较,见表 1 和表 2.

表 1 结点数 (单位:K)

Trie	RegionTrie(空间划分高度限制)						
/	0	1	2	3	4	5	6
36	64	68	76	84	96	108	117

表 2 树形结构存储空间 (单位:MB)

Trie	RegionTrie(空间划分高度限制)						
/	0	1	2	3	4	5	6
871	871	982	1171	1361	1648	1946	2183

效率. 测试分析效率的算法有

基于 Trie 树的对比方法.

基于 RegionTrie 树的增量算法.

考虑剪枝策略的增量算法.

(1)空间和文本权重相同.对于综合相关性式(4)的参数 α 取值 0.5,测试结果如图 10、图 11 所示.

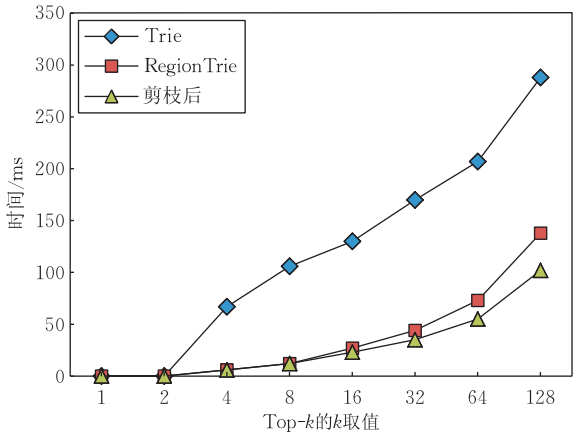

图 10 短查询集结果

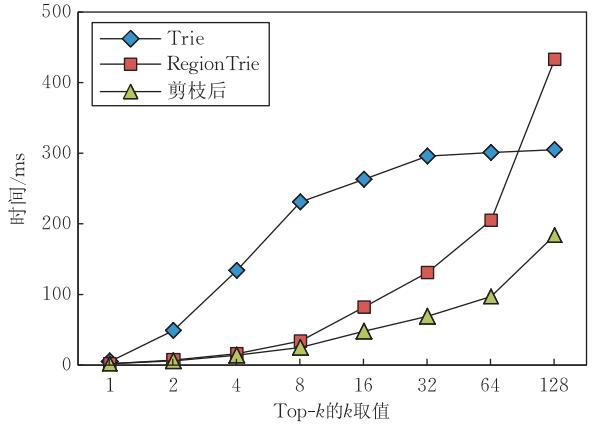

图 11 长查询集结果

从上面的结果来看,不论是短查询集还是长查询集,效率上都有很大的提高.例如,当 $k=32$ 时,对于查询集,考虑剪枝的增量算法相比对比方法 Trie 效率提高了 5 倍.对于长查询集,提高了 4 倍.

对于短查询集,基于 RegionTrie 树上增量算法

已经有不错的效果，因为 RegionTrie 结构同时考虑空间和文本，能进行快速地滤掉. 对于长查询集，虽然 RegionTrie 结构能快速过滤，但是树的搜索深度很深，中间过程计算量变大，需要进行剪枝优化，之后可以达到更好效果，实验结果也验证了这样的分析.

(2) 空间和文本权重不同. 变化参数 α，比较两种查询集下对比算法 Trie 和考虑剪枝的优化算法的性能，结果如图 12 所示.

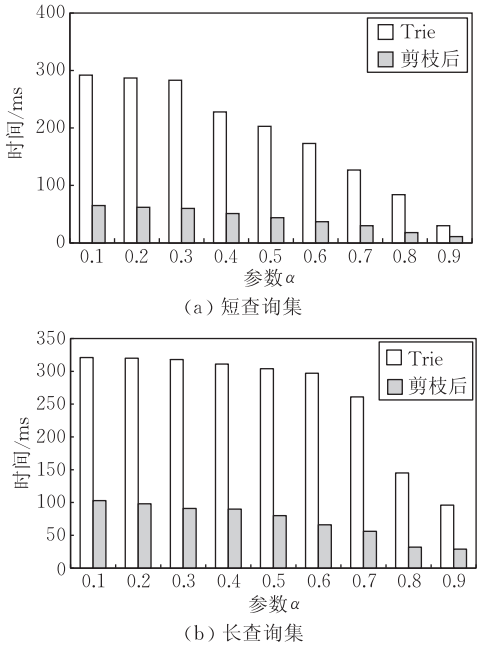

图 12　变化参数 α(横轴参数)之后的效率对比

可以从图 12 中看到，变化参数 α，使得空间和文本的权重发生变化，但无论偏重空间还是偏重文本，效率都有很大的提高，至少保持在 3 倍以上.

有效性是指能否满足用户的查询需求，当用户选择了参数 α 之后，我们提供的结果能否匹配用户对于空间和文本上的需求. 为了验证本文提出的方法的有效性，我们在 30 名学生中做了如下实验，让每个学生分别在参数 α 取不同大小的值时进行查询，并给出 0～10 分的满意度评价，最终以平均分来衡量有效性并得到如图 13 的结果.

可以从图 13 中看到，当参数 α 取边界值 0 和 1，只考虑空间或只考虑文本，此时满意度达到峰值，因为结果准确用户就一定满意；当参数 α 取中间值 0.5 时，用户均衡考虑空间和文本，满意度也相对较高；当参数 α 取值 0.25 和 0.75 时，用户略偏重空间或略偏重文本，满意度也大于 8，但略低于参数 α 取其它值的情况.

(1) 有效性分析

我们定义中给出的综合相关性分数是基于参数

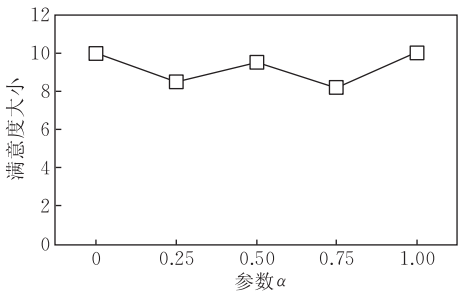

图 13　变化参数 α 时得到满意度评分

α 的线性函数，变化趋势为线性. 用户可以根据自己的偏好方便选择参数 α 的取值，如偏好文本的重要性时在 0.5～1 之间选择 α，偏好空间上的重要性时就在 0～0.5 之间选择 α 从而保证搜索的有效性.

(2) 高效性分析

我们的方法取得高效率的原因在于：(i) 提出的 RegionTrie 结构同时考虑了文本的相似性和空间距离，可以实现有效的剪枝；(ii) 基于 RegionTrie 的增量算法和剪枝策略能够有效地计算 Top-k 结果.

7　结　论

本文研究了空间数据上 Top-k 关键词模糊查询问题. 首先定义了一种新型的相关性函数，融合了编辑距离、关键词权重、空间距离等 3 方面因素；其次提出了一种新型的索引结构 RegionTrie，同时根据文本和空间特性对 POI 点进行有效的组织. 基于该索引结构，设计了高效的 Top-k 算法，并给出了有效的编辑距离增量计算方法和剪枝策略. 实验结果表明，提出的方法较之对比方法在性能上有着显著的提高.

参　考　文　献

[1] Zhou Y, Xie X, Wang C, Gong Y, Ma W-Y. Hybrid index structures for location-based web search//Proceedings of the CIKM. Bremen, Germany, 2005: 155-162

[2] Chen Y-Y, Suel T, Markowetz A. Efficient query processing in geographic web search engines//Proceedings of the SIGMOD. Chicago, IL, 2006: 277-288

[3] Felipe I D, Hristidis V, Rishe N. Keyword search on spatial databases//Proceedings of the ICDE. Cancun, Mexico, 2008: 656-665

[4] Zhang D, Chee Y M, Mondal A, Tung A K H, Kitsuregawa M. Keyword search in spatial databases: Towards searching by document//Proceedings of the ICDE. Shanghai, China, 2009: 688-699

[5] Cong G, Jensen C S, Wu D. Efficient retrieval of the top-k

most relevant spatial Web objects. Proceedings of the VLDB Endowment, 2009: 2(1): 337-348

[6] Yao B, Li F, Hadjieleftheriou M, Hou K. Approximate string search in spatial databases//Proceedings of the ICDE. Long Beach, California, USA, 2010: 545-556

[7] Cao X, Cong G, Jensen C S. Retrieving top-*k* prestige-based relevant spatial Web objects. Proceedings of the VLDB Endowment, 2010, 3(1): 373-384

[8] Wu D, Yiu M L, Jensen C S, Cong G. Efficient continuously moving top-*k* spatial keyword query processing//Proceedings of the ICDE. Hannover, Germany, 2011: 541-552

[9] Cao X, Cong G, Jensen C S, Ooi B C. Collective spatial keyword querying//Proceedings of the SIGMOD Conference. Athens, Greece, 2011: 373-384

[10] Roy S B, Chakrabarti K. Location-aware type ahead search on spatial databases: Semantics and efficiency//Proceedings of the SIGMOD Conference. Athens, Greece, 2011: 361-372

[11] Li G, Xu J, Feng J. Desks: Direction-aware spatial keyword search//Proceedings of the ICDE. Washington, DC, USA, 2012: 474-485

[12] Arasu A, Chaudhuri S, Ganjam K, Kaushik R. Incorporating string transformations in record matching//Proceedings of the SIGMOD Conference. Vancouver, BC, Canada, 2008: 1231-1233

[13] Chakrabarti K, Chaudhuri S, Ganti V, Xin D. An efficient filter for approximate membership checking//Proceedings of the SIGMOD Conference. Vancouver, BC, Canada, 2008: 805-818

[14] Chaudhuri S, Ganjam K, Ganti V, Motwani R. Robust and efficient fuzzy match for online data cleaning//Proceedings of the SIGMOD Conference. San Diego, California, USA, 2003: 313-324

[15] Li C, Lu J, Lu Y. Efficient merging and filtering algorithms for approximate string searches//Proceedings of the ICDE. Cancun, Mexico, 2008: 257-266

[16] Gravano L, Ipeirotis P G, Jagadish H V, Koudas N, Muthukrishnan S, Srivastava D. Approximate string joins in a database (almost) for free//Proceedings of the VLDB. Roma, Italy, 2001: 491-500

[17] Xiao C, Wang W, Lin X, Shang H. Top-*k* set similarity joins//Proceedings of the ICDE. Shanghai, China, 2009: 916-927

[18] Li G, Deng D, Wang J, Feng J. PASS-JOIN: A partition-based method for similarity joins. Proceedings of the VLDB Endowment, 2011, 5(3): 25-264

[19] Ji S, Li G, Li C, Feng J. Efficient interactive fuzzy keyword search//Proceedings of the WWW. Madrid, Spain, 2009: 371-380

HU Jun, born in 1990, M. S. candidate. His research interest is in database.

FAN Ju, born in 1984, Ph. D.. His research interests include structural and non-structural data query, Web data management, space and text query and recommendation.

LI Guo-Liang, born in 1980, Ph. D., lecturer. His research interests include data cleaning and integration, text space data processing, crowdsourcing data management.

CHEN Shan-Shan, born in 1988. Her research interest is in database.

Background

Due to the well-known Global Positioning System utilized in mobile phones, Location-Based Services have become very popular recently; more and more research starts focusing on how to effectively search spatial POIs.

Existing LBS systems employ a spatial keyword search method to provide services, which finds the relevant POIs by considering textual relevance and spatial distance when given a set of points-of-interest (POIs). Specifically, each POI in the spatial database contains spatial information and certain text information; when given users' location information and certain keywords, LBS systems will return spatially and textually relevant POIs.

However, because of mobile systems' features, such as small screens and the inconvenience of keyboard input, users are more likely to make mistakes when inputting keywords; for example, users may mistakenly input "sterbuck" for "starbucks". In this way, users may get unexpected results. Existing methods only allow exact matches for query keywords and fail to support fuzzy search, which means that when queries slightly differs from existed data, LBS systems will fail to return relevant results.

To address this challenge, we focus on fuzzy search and introduce a novel function to quantify the relevance between POIs and the query, by considering the similarity between keywords and spatial distance. Then, we devise an effective index structure, called RegionTrie to organize the POIs and develop efficient search algorithm based on the RegionTrie. We conducted experiments on real datasets, and the experimental results show that our algorithms achieve high performance.