软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2014,25(Suppl.(2)):157−168

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

一种支持 Top-*k* 空间关键词检索的高效压缩索引*

周 新[1,2], 张 孝[1,2], 安润功[1,2], 薛忠斌[1,2], 王 珊[1,2]

[1](数据工程与知识工程教育部重点实验室(中国人民大学),北京 100872)

[2](中国人民大学 信息学院,北京 100872)

通讯作者: 张孝, E-mail: zhangxiao@ruc.edu.cn, http://www.ruc.edu.cn

摘 要: 基于位置的服务可以指引用户找到在特定位置或区域内能够提供所需要服务的对象(比如找某个高校附近(经纬度标识)的咖啡店).向这类服务提交一个查询位置和多个关键词,该类服务返回 k 个最相关的对象,对象和查询的相关性同时考虑空间相近性和文本相似性.为了支持高效的 top-k 空间关键词查询,出现了多种混合索引,然而现有的这些索引为了提供实时响应均耗费大量存储空间.提出一种基于压缩技术的索引 CSTI,该索引显著减少了存储开销(至少减少 80%甚至到两个数据量级),同时保持高效的查询性能.大量基于真实和仿真数据集的实验结果表明,CSTI 在空间开销和响应时间上均优于已有方法.

关键词: 压缩索引;top-*k*;空间关键词检索

中文引用格式: 周新,张孝,安润功,薛忠斌,王珊.一种支持 Top-*k* 空间关键词检索的高效压缩索引.软件学报,2014,25(Suppl.(2)):157−168. http://www.jos.org.cn/1000-9825/14034.htm

英文引用格式: Zhou X, Zhang X, An RG, Xue ZB, Wang S. Efficient compressed index for top-*k* spatial keyword query. Ruan Jian Xue Bao/Journal of Software, 2014,25(Suppl.(2)):157−168 (in Chinese). http://www.jos.org.cn/1000-9825/14034.htm

Efficient Compressed Index for Top-*k* Spatial Keyword Query

ZHOU Xin[1,2], ZHANG Xiao[1,2], AN Run-Gong[1,2], XUE Zhong-Bin[1,2], WANG Shan[1,2]

[1](Key Laboratory of Data Engineering and Knowledge Engineering of the Ministry of Education (Renmin University of China), Beijing 100872, China)

[2](School of Information, Renmin University, Beijing 100872, China)

Corresponding author: ZHANG Xiao, E-mail: zhangxiao@ruc.edu.cn, http://www.ruc.edu.cn

Abstract: Location-Based services guide a user to find the object which provides services located in a particular position or region (e.g., looking for a coffee shop near a university). Given a query location and multiple keywords, location-based services return the most relevant objects ranked according to location proximity and text relevancy. Various hybrid indexes have been proposed in recent years which combine R-tree and inverted index to improve query efficiency. Unfortunately, the state-of-the-art approaches require more space in order to reduce response time. Cache mechanism is inefficient due to huge storage overhead. In this paper, a novel index based on index compressed technology (CSTI) is proposed, to answer top-k SKQ. CSTI significantly reduces storage overhead (by at least 80%), while maintaining efficient query performance. Extensive experiments based on real dataset and simulated dataset confirm CSTI is effective and efficient.

Key words: index compress; top-*k*; spatial keyword query

位置信息的重要性日益显著,据统计,搜索引擎中 20%的查询和位置相关,大量基于位置的服务方便了人们的生活.广泛使用的移动和 GPS 设备令定位和获取位置信息更容易,而社交网络的快速发展使得分享位置信息

* 基金项目: 国家自然科学基金(61070054); 国家重点基础研究发展计划(973)(2014CB340403); 国家高技术研究发展计划(863)(2012AA011001); 中央高校基本科研业务费专项资金(10XNI018)

收稿时间: 2014-05-07; 定稿时间: 2014-08-19

变得普遍.带有空间与文本信息的对象大量出现在网络中,通过位置和关键词访问对象的需求日益广泛.

为了支持高效的空间关键词检索,学者们提出多种索引方法.现有方法主要采用紧耦合空间索引和文本索引的方式来提高查询性能,在空间剪枝的同时进行文本过滤[3−8].然而目前这些方法普遍存在以下缺点:

(1) 存储开销大.无论空间优先的索引(空间索引的节点集合倒排文件),还是文本优先的索引(关键词集成空间索引 Tree),为了提高检索性能,重复存储大量信息,索引的存储空间超过原始数据的几十倍甚至上百倍.

(2) 缓存机制无效.由于紧耦合的混合索引占用大量存储空间,很难对其进行有效缓存.而缓存机制是提供实时响应的关键技术.

(3) 维护开销大.基于空间索引 R-tree 的更新代价高,而空间索引和文本索引的紧耦合增大了代价.

因此,本文提出一种新的基于压缩技术的索引,以支持 top-k 空间关键词查询.本文主要贡献如下:

(1) 新的高效压缩的索引结构.本文提出基于压缩技术的新索引结构——CSTI(compressed spatial-textual index).CSTI 采用文本优先的方式划分对象,对高频词,PCTree 索引其倒排列表,能够显著降低存储开销,提高缓存命中率,进而缩短响应时间.

(2) 新的快速 top-k 空间关键词检索算法.基于 CSTI,本文设计新的高效检索算法,快速响应 top-k 空间关键词查询.

(3) 理论和实验证明了方法的有效性和高效性.对于空间开销,本文给出了代价模型,估算 CSTI 节省的空间开销.基于真实和仿真数据集,本文做了大量实验评估.实验结果表明,CSTI 在空间开销、磁盘 I/O、响应时间上都优于目前的索引方法.空间最多减少两个数量级,最少也降低 80%.

1 问题阐述

基于位置的服务方便了我们的生活,利用百度、高德地图等容易查找感兴趣的地方,导航旅游路线.我们感兴趣的地方不仅包含空间位置信息,还包含文本描述,称其为空间文本对象.现在的手机均有定位功能,我们提交的请求不仅包含关键词信息,同时隐藏位置信息,基于位置的服务将返回和文本最相关且离查询位置最相近的对象.

定义 1(空间文本对象). 每个空间-文本对象 $o(id,loc,doc)$包含对象标识、空间属性和文本属性.$o.id$ 是对象标识;$o.loc$ 是 2 维的空间位置点,常用经纬度表示;$o.doc$ 标记对象的文本属性,由多个关键词组成.

定义 2(查询请求). 查询请求 $q(loc,doc,k)$包含 3 个属性.$q.loc$ 是用经纬度表示的 2 维空间点;$q.doc$ 是查询的关键词;$q.k$ 指定返回的结果数.

定义 3(top-k 空间关键词检索). 给定空间-文本对象集合 O,空间关键词查询 $q(loc,doc,k)$,从 O 中检索 k 个最相关的空间-文本对象.相关性同时考虑了空间相近性和文本相似性.公式(1)~公式(3)分别给出查询和对象的相关性、空间相近性和文本相似性的计算方法.

$$\tau(o,q)=\alpha\cdot\delta(o.loc,q.loc)+(1-\alpha)\cdot\theta(o.doc,q.doc) \tag{1}$$

$$\delta(o.loc,q.loc)=1-\frac{d(o.loc,q.loc)}{d_{\max}} \tag{2}$$

$$\theta(o.doc,q.doc)=\frac{\sum_{t\in q.doc}\omega_{t,o.doc}\cdot\omega_{t,q.doc}}{\sqrt{\sum_{t\in o.doc}\omega_{t,o.doc}^2\cdot\sum_{t\in q.doc}\omega_{t,q.doc}^2}} \tag{3}$$

其中,σ是调节因子,调节空间相近性和文本相关性的重要性.$d(o.loc,q.loc)$是对象 o 和查询 q 的空间距离.$d_{\max}$ 是对象间最大距离.$\omega_{t,doc}$ 是词 t 在文档 doc 中所占的权重.

例 1:图 1 给出一个 top-k 空间关键词查询的示例.空间范围内共有 8 个对象,位于 q 的旅游者寻找离自己最近且带自助的 KTV.不考虑文本相似性,对象 o_3 离旅游者最近,但 o_3 不是 KTV.虽然对象 o_6 距离旅游者稍远,然而满足关键词的查询要求,综合距离的相近性和关键词的相似性,o_6 作为 top−1 返回.

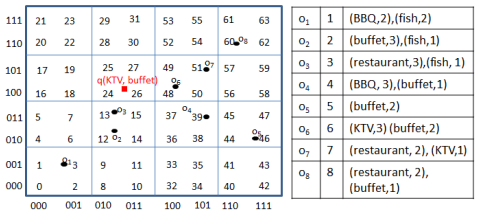

Fig.1 A sample of spatial keyword query

图 1 空间关键词查询示例

2 相关工作

IR 领域的学者们发现,仅用关键词不足以表示位置信息的邻接特性,于是探索了位置信息的表示和空间-关键词信息的检索方法[1−5].此外,为了获得空间数据库中有用的文本信息,空间数据库提供空间关键词检索,返回满足查询的空间文本对象.

数据库中的空间关键词检索主要关注传统查询的性能改善和新型查询的应用.

空间关键词检索主要有 3 种传统查询[10]:Boolean 范围查询,Boolean kNN 查询和 top-k kNN 查询(定义 3).为了提高检索性能,空间剪枝的同时进行文本过滤,组合不同空间索引(R-tree,Quad-tree,Grid,SpaceFilling Curve)和文本索引(inverted index,signature file),从而,支持不同传统查询的混合索引被提了出来.文献[10]对这些索引进行综合对比,并给出实验评估.下面详细介绍两种支持 top-k 空间关键词查询的索引.

IR-tree[7],一种空间索引优先的混合索引方式,在 R-tree 的每个节点中加入 inverted file 索引文本信息,达到空间剪枝的同时进行文本过滤.文献[8]提出一种新的混合索引——S2I.S2I 采用文本优先的方法,为具有相同关键词的对象创建倒排列表.对于非频繁词,使用文件块存储相应的倒排列表;对频繁词,S2I 构建 aR-tree[9]索引包含该频繁词的对象.相对于 IR-Tree,S2I 减少了大量的磁盘 I/O,提高了检索性能,然而随着数据量的增大,每个对象的文本信息增多,S2I 检索性能迅速下降.

除以上传统查询外,学者们(特别是国内学者)探索了新型查询,提出相应的索引架构和检索算法[11−15].

本文致力于 top-k 空间关键词检索的性能改善.目前支持 top-k 空间关键词查询的索引均是紧耦合的混合索引,为了降低响应时间而牺牲了大量空间.而本文所提出的基于压缩的空间-文本索引——CSTI 则采用另外的技术路线,即:通过压缩记录提高空间利用率降低树高、提高缓存命中率,从而改善检索性能.

3 基于压缩的空间-文本索引(CSTI)

减少索引的空间消耗不仅节省存储代价,同时缩短数据从磁盘到内存的传输时间,提高缓存命中率,改善查询性能.因此,本文提出基于压缩的空间-文本索引——CSTI.

CST 是基于倒排关键词的索引,对不同频率关键词的倒排列表区别存储,集成了 S2I 的优点.然而对于频繁词的检索,CSTI 采用了新的索引结构——前缀压缩树(PCTree).对于非频繁词的检索,CSTI 采用 Block 结构存储其倒排列表.为了支持高效的压缩,空间查找表用来维护位置信息.CSTI 主要包括 4 个组件:字典表、空间查找表、PCTree 和 Block.图 2 给出了对应于图 1 中示例的 CSTI 索引架构.

CSTI 的 Block 结构相对简单,对于倒排列表中的每个对象,记录对象标志 id 和关键词在对象中的权重(用 TF 表示),如图 2(c)所示.下面详细介绍 CSTI 的其他组件.

3.1 字典表

作为 CSTI 的重要组件,字典表存储关键词的元信息,主要包括:文档频率(df)、类型标志(type)和文件指针(pointer).文档频率表示某个关键词在所有对象中出现的次数;类型标志表明关键词对应的倒排列表采用何种方式索引;文件指针指向存储倒排列表的文件.

例 2:图 2(a)给出例 1 对应的字典表,假设文档频率大于 3 的关键词是频繁的,“buffet”采用 PCTree 索引其倒

排列表,其他关键词采用 Block 索引其倒排列表.

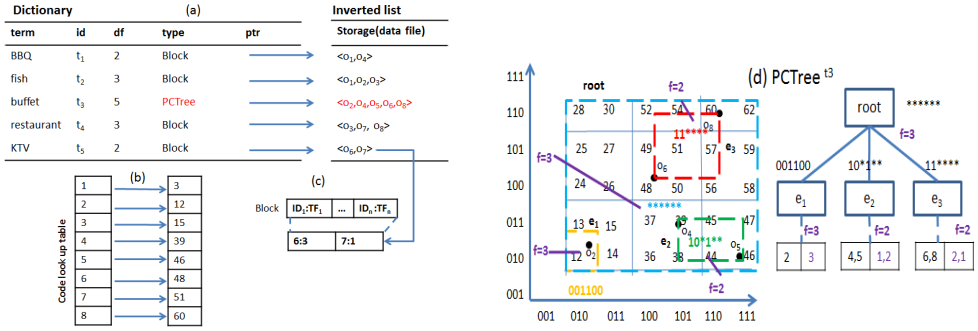

Fig.2 Structure of CSTI

图 2 CSTI 索引架构

3.2 空间查找表

为了便于压缩,CSTI 采用空间填充曲线技术将 2 维空间坐标转换为 1 维.常用的 Zorder,Hilbert 等空间填充曲线均可用来编码位置信息,CSTI 采用 Zorder 编码.

为了有效地利用压缩机制,保证空间位置相近的对象在倒排列表中位置也相近,同时减少索引的空间消耗,CSTI 按照 Zorder 编码的连续性重新为每个对象分配 id,形成空间查找表.实际存储时,对象 id 将代替 Zorder 编码表示空间特性.图 2(b)显示了例 1 对应的空间查找表.相应的 Zorder 编码和对象 id 映射关系如图 2 所示.

3.3 前缀压缩树(PCTree)

依据 Zipf 规律,少量的关键词频繁出现,大量的关键词不频繁出现,因此分开存储不同频率关键词的倒排列表具有重要意义.CSTI 分别采用 PCTree 和 Block 存储频繁词和非频繁词的倒排列表.

降低索引的存储开销,有效利用缓存机制是 CSTI 的核心工作.CSTI 的空间查找表重新分配对象 id,因此,可以高效地利用倒排索引的压缩技术.倒排索引常用的压缩算法有变长字节、Gamma coding、S9、S16、PFD 等编码方法[16].本文采用 PFD 压缩算法对频繁词的倒排列表进行压缩.

为了防止每次都要从头解压倒排列表,同时高效地进行空间剪枝,CSTI 采用前缀压缩树(prefix-based compressed tree,简称 PCTree,)合理地组织压缩数据块.

PCTree 是一棵自底向上构建的 aR-tree 变种(构建方法参照文献[17]).依据 Zorder 填充曲线的特性:(a) 如果子空间 A 包含子空间 B,那么子空间 A 的 Zorder 编码是子空间 B 的 Z-order 编码的前缀;(b) 如果空间 A 和空间 B 没有共同前缀,那么 A 和 B 一定不相交.因此,PCTree 将具有公共前缀编码的空间坐标划分到一起,形成 aR-tree 的叶节点.相比较传统的 aR-tree,该方法避免了 R-Tree 节点间的重叠,增强了空间剪枝能力.同时,CSTI 对 aR-tree 每个叶节点进行 PFD 压缩,进一步减少了空间开销.

PCTree 不仅具有 aR-tree 的优点,还增强了 aRtree 的空间剪枝能力;PCTree 压缩了叶节点,树高和节点数远少于 aR-tree,存储和检索性能均优于 aR-tree.

例 3:图 2(d)给出关键词“buffet”对应的 PCTree,左边是平面表示,右边是树形结构.对象 o_4 和 o_5 的 Zorder 编码分别为 100111 和 101110,二者具有公共前缀 10*1**.类似地,对象 o_6 和 o_8 的公共前缀为 11****,对象 o_2 自成一个叶子节点,前缀是其 Zorder 编码本身.PCTree 的节点中还存储了子树中关键词的最大权重,可以高效地计算节点和查询的相关性,空间剪枝的同时进行文本过滤.

3.4 空间开销分析

一棵存储 N 条记录的 aR-Tree,最大可能的树高如公式(4)所示,这棵 aR-tree 的节点总数可以通过公式(5)获得.PCTree 的叶节点包括元信息和压缩的数据.假设 PFD 算法的平均压缩比是 γ(大于 1),叶节点的存储代价如公式(7)所示,其中,*entrySize* 是元信息占用的空间,$(5m\times8)/\gamma$ 表示压缩后的数据量.同时,公式(8)和公式(9)分别给出

aR-Tree 和 PCTree 的空间代价模型,公式(10)计算节省的空间开销.公式(8)给出的中间节点和叶节点的总和(见公式(5))乘以每个节点的块大小即为 aR-Tree 的空间消耗;而 PCTree 的叶节点与中间节点的存储模型不同,公式(9)需要组合中间节点的空间消耗和叶节点的空间消耗.

$$h_{\max}=\lceil \log_m N\rceil-1 \tag{4}$$

$$\sum_{i=1}^{h_{\max}}\left\lceil\frac{N}{m^i}\right\rceil=\left\lceil\frac{N}{m}\right\rceil+\left\lceil\frac{N}{m^2}\right\rceil+\ldots+1 \tag{5}$$

$$m=\frac{blockSize}{entrySize}\times\beta \tag{6}$$

$$c(leaf,pctree)=entrySize+5m\times 8/\gamma \tag{7}$$

$$c(rtree)=\sum_{i=1}^{h_{\max}}\left\lceil\frac{N}{m^i}\right\rceil\times blockSize \tag{8}$$

$$c(pctree)=\sum_{i=1}^{h_{\max}-1}\left\lceil\frac{N}{5m^i}\right\rceil\times blockSize+\left\lceil\frac{N}{5m}\right\rceil\times c(leaf,pctree) \tag{9}$$

$$r_{save}=\frac{c(rtree)-c(pctree)}{c(rtree)}=\frac{4}{5}+\frac{N}{m\times\sum_{i=1}^{h_{\max}}\left\lceil\frac{N}{m^i}\right\rceil}\times\left(\frac{1}{5}-\frac{\beta}{5m}-\frac{8\beta}{\gamma\times entrySize}\right)>\frac{4}{5} \tag{10}$$

其中,$h_{\max}$ 表示最大树高;m 是 aR-tree 节点能容纳的最少 entry 数,PCTree 叶节点的容量为 5m(每个 entry 占 8B),中间节点的容量与 aR-tree 一致.blockSize 表示 aR-tree 节点的空间代价,通常设为 4 096B;β表示节点填充率,公式(6)给出 m,*blockSize*,β,*entrySize* 的关系;*entrySize* 表示 aR-tree 节点中 entry 的大小,本文为 54B.

从以上推导公式可以看出,PCTree 的空间代价至少比 aR-tree 节省 80%.同时 PCTree 叶节点减少为原来的 20%,中间节点也大量减少,导致部分 PCTree 的树高降低.同时,PCTree 避免节点间的重叠,因此,PCTree 的性能优于 aR-Tree,CSTI 能够支持快速响应 top-k 空间关键词查询.下面的第 4 节将详细阐述 top-k 空间关键词检索算法.

4 Top-k 空间关键词检索算法

本文提出的索引架构 CSTI 支持实时响应 top-k 空间关键词查询.对于单关键词检索和多关键词检索,本文给出 SSKQ 算法和 MSKQ 算法以响应查询.

4.1 单关键词检索算法

算法 SSKQ 维护一个堆 H 存放候选结果.对于查询 q,首先查找字典表,找到关键词类型,如果是非频繁词(行 3),依据文件指针找到存储位置,插入数据块中的所有对象到堆(行 4~行 7),返回前 k 个结果(行 27);如果是频繁词(行 8),遍历 PCTree 找到相应的压缩块,解压数据,插入到堆 H 中(行 9~行 21).PCTree 维护了关键词的最大权重和对象的 MBR 信息,采用 best-first 搜索策略,因此最相关的结果总是排在 H 的最前面.当找到 top-k 结果时,终止查询,返回结果(行 22~行 27).

算法 1. 单关键词检索 SSKQ(Algorithm 1. *SSKQ*(*CSTS*,q)).

Input: index CSTI,query $q(loc,t_i,k)$;

Output: top-k *objects* which have highest score.

```
1:  MaxHeap H←φ
2:  type←lookupDictionary(term)
3:  if  type is no-compressed then
4:      List l←traverseBlock(term,type)
5:      for each spatial-textual object o∈l do
6:          H.insert(o,τ(o,q))
```

```
7:          end for
8:   else
9:       H.insert(PCTree^{t_i}.root);
10:      while H.isNotEmpty() do
11:          Entry e←H.pop();
12:          if e is internal node then
13:              for each node n∈e do
14:                  H.insert(n,τ(n,q));
15:              end for
16:          else //e is a leaf-node
17:              Entry d←decompressNode(e);
18:              for each spatio-textual object o∈d do
19:                  H.insert(o,τ(o,q))
20:              end for
21:          end if
22:          if H.k is available then
23:              break;
24:          end if
25:      end while
26:  end if
27:  return top-k object in H as result
```

例 4:对应于例 1 的 top-1 查询 q(*buffet*),“buffet”检索对应的 PCTree.首先,将根节点加入大顶堆 H 中,计算其孩子和 q 的相关性,由图可知,最相关的孩子为节点 e_1,因 e_1 是叶子节点,计算叶子对象和查询的相关性,可知对象 o_2 最相关,返回结果 o_2.

4.2 多关键词检索算法

公式(1)中的相关性函数依照公式(11)分解,多关键词检索分解为多个单关键词检索,通过集成每个单关键词的部分得分(公式(12))来完成检索过程.因为对象和查询的位置信息不会发生变化,所以检索关键词 i 得出的位置信息将成为检索关键词 j 时的参照,公式(13)给出对象和查询相关性的下界.

$$\begin{aligned}\tau(o,q)&=\alpha\cdot\delta(o.loc,q.loc)+(1-\alpha)\times\theta(o.doc,q.doc)\\&=\alpha\cdot\delta(o.loc,q.loc)+(1-\alpha)\times\sum_{t\in q.doc}IDF(t)\cdot\lambda_{t,o.doc}\\&=\sum_{t\in q.doc}\left(\alpha\cdot\frac{\delta(p.loc,q.loc)}{|q.doc|}+(1-\alpha)\cdot IDF(t)\cdot\lambda_{t,o.doc}\right)\end{aligned}\tag{11}$$

$$\tau^i(o,q)=\alpha\cdot\frac{\delta(p.loc,q.loc)}{|q.doc|}+(1-\alpha)\cdot IDF(t)\cdot\lambda_{t,o.doc}\tag{12}$$

$$\tau^i_{\_}(o,q)=\alpha\cdot\frac{\delta(p.loc,q.loc)}{|q.doc|}\tag{13}$$

$$O_{\_}=\sum_{\forall i:o\in L_j}\tau^i(o,q)+\sum_{\forall j:o\notin L_j}\tau^i_{\_}(o,q)\tag{14}$$

$$O^{-}=\sum_{\forall i:o\in L_j}\tau^i(o,q)+\sum_{\forall j:o\notin L_j}\max(\mu_j,\tau(o,q))\tag{15}$$

MSKQ 算法的检索过程如下:首先为每个单关键词维护元信息(行 2~行 4).选择关键词检索源 S_i,访问其中的对象 p,计算 p 的相关性和下界 $p\_$,更新 S_i 的上界 u_i,加入 p 到候选集 C(行 6~行 11)中.更新 C 中每个对象的上

界(行 13),当一个对象的下界大于 C 中所有对象的上界时,报告该对象为结果,返回 k 个结果则终止检索过程(行 15~行 17).当结果数 m 小于 k 时,则返回 C 中有最高下界的(k–m)个对象(行 21).

多关键词检索面临的挑战是如何选择每次要集成的单关键词检索源以尽快找到 top-k 结果,提前终止检索.本文给出两种策略用于选择检索源:

(1) 轮回选择.该策略顺序选择和查询对应的关键词检索源,每轮访问检索源的顺序是一致的.该选择策略简单易行,然而它没有考虑候选结果的相关性,多次迭代才能找到最终结果.下面给出优化的选择策略.

(2) 最优选择.每次迭代,该策略均要为每个检索源预取一个对象,和查询相关性最大的对象所在的检索源将被访问.因为最相关的对象最先访问,该策略可以尽快找到总体最优的最终结果,提前终止检索过程.

例 5:对应例 1 中的查询 q(*KTV*,*buffet*),表 1 给出采用轮回选择策略的多关键词检索过程.加粗的数值展示候选对象的变化过程,通过 5 步迭代,最终 o_6 作为 top-1 结果返回.

表 1 多关键词检索的迭代过程

PCTree^{t3}	Block^{t5}	候选集 C{$p[p\_,p^{-}]$}
$\tau^{t3}(o_2,q)$=0.2676	$\tau^{t5}(o_6,q)$=0.4105	$\boldsymbol{o_2}$[0.3926,0.6781],$\boldsymbol{o_6}$[0.5355,0.6781]
$\tau^{t3}(o_5,q)$=0.2407		o_2[0.3926,0.6781],o_6[0.5355,**0.6512**],$\boldsymbol{o_5}$[0.3234,0.6512]
	$\tau^{t5}(o_7,q)$=0.2970	o_2[0.3926,**0.5646**],o_6[0.5355,0.6512], o_5[0.3234,**0.5377**],$\boldsymbol{o_7}$[0.4075,0.5377]
$\tau^{t3}(o_6,q)$=0.2242		o_2[0.3926,0.5646],o_6[**0.6346**,**0.6346**], o_5[0.3234,0.5377], o_7[0.4075,**0.5212**]

算法 2. 多关键词检索 MSKQ(Algorithm 2. *MSKQ*(*CSTI*,*q*)).

Input: index CSTI, query q(*loc*,*doc*,k);

Output: top-k *objects* which have highest score.

1: $C\leftarrow\varnothing$//*List of candidate objects*.
2: **for each** *term* $t_i\in q.doc$ **do**
3: $S_i\leftarrow source(q,t_i)$; $L_i\leftarrow\varnothing$; $u_i\leftarrow\infty$;
4: **end for**
5: **while** $\exists S_i$ such that $S_i\neq\varnothing$ **do**
6: $i\leftarrow selectSource(q.doc)$;
7: $p\leftarrow S_i.next()$; $u_i\leftarrow\tau^{i}(p,q)$; $L_i\leftarrow L_i\cup p$;
8: $p\_\leftarrow updateLowerBound(p)$;//Formula 14
9: **if** !($p\in C$) **then**
10: $C\leftarrow C\cup p$;
11: **end if**
12: **for each** $p\in C$ **do**
13: $p^{-}\leftarrow updateUpperBound(p)$;//Formula 15
14: **end for**
15: **while** $\exists p\in C$, $\forall o\in C$:$p\_\geqslant\max(o^{-})$ **do**
16: $C\leftarrow C–p$;
17: reports p, halt if k *objects* have been reported;
18: **end while**
19: **end while**
20: **if** less than k *objects* have been reported **then**
21: return the *objects* in C with *highest lower bound score* $p\_$
22: **end if**

5 实验评估

本文使用真实和仿真数据集评估CSTI索引的性能,并与目前两种经典索引IR-tree和S2I进行了对比(感谢IR-tree和S2I的作者分享源码).CSTI采用Java实现.所有的实验均在配置4GB内存、2.67GHz Intel双核处理器的PC机上完成.

5.1 数据集

本文选择1个POI(point of interest)真实数据集和仿真多个Tweet数据集进行了实验评估.POI数据集记录了欧洲多国的餐馆、景点等信息,然而POI的对象数有限,本文仿真4个不同数据量的Tweet测试集,分别是Tweet160K,Tweet320K,Tweet640K和Tweet1280K.采用随机分配对象的位置的方式模拟真实Tweet中包含的位置信息.表2给出数据集的特性.

表2 数据集特性

数据集	对象数	平均包含的关键词数	总共包含关键词数	总共包含词语数
POI160K	160 000	2.70	39 401	436 901
Tweet160K	160 000	10.04	94 735	1 685 357
Tweet320K	320 000	10.29	150 933	3 459 213
Tweet640K	640 000	10.24	249 420	6 886 321
Tweet1280K	1 280 000	10.30	414 366	13 842 791

5.2 测试结果及分析

本节对比IR-Tree,S2I和CSTI的性能,评估索引的空间开销、查询的响应时间和磁盘I/O.为了使结果可靠,每个实验都重复10次,每次响应100个查询.分别评估了结果数(k)、查询关键词数、调节因子、数据量、buffer size、不同数据集对查询性能的影响.表3给出实验用的参数设置,缺省值加粗显示.

表3 实验的参数设置

参数	值
查询结果数(k)	**10**,20,40,80,160
关键词个数	1,2,**3**,4,5
调节因子	0.1,**0.3**,0.5,0.7,0.9
数据量(Tweet)	160K,320K,**640K**,1280K
数据集	POI,**Tweet**

空间开销.表4给出基于Tweet数据集IR-Tree、S2I和CSTI的空间消耗.从表中可以看出,IR-Tree的空间开销是S2I的40倍,CSTI的200倍.相比IR-Tree,S2I的总体空间开销减少很多,但仍然是CSTI的5倍,尤其是对频繁词构建的树索引,CSTI的空间开销比S2I降低1个数量级.

表4 索引的空间开销

数据量(K)	索引	总大小(MB)	树索引大小	文件(块)索引大小	元数据大小
160	CSTI	26	5.3	14.3	6.4
	S2I	104.6	86.7	15.3	2.6
	IR-Tree	5 139.2	31	5 076	32.2
320	CSTI	53.7	18	24	11.7
	S2I	2 59.3	229	26	4.3
	IR-Tree	11 449.4	62	11 202	185.4
640	CSTI	98.2	37	39	22.2
	S2I	534.3	486	41	7.3
	IR-Tree	21 938	123	21 381	434
1 280	CSTI	173.4	70	60	73.4
	S2I	997.2	921	64	12.2
	IR-Tree	44 261	244	42 412	1 605

结果数(k)的影响.图3(a)和图3(b)分别给出索引的响应时间和磁盘I/O随结果数k值变化的情况.CSTI的响应时间和磁盘I/O都是最少的.CSTI和S2I的响应时间几乎胜过IR-Tree索引1个数量级,主要是因为这两种索引采用文本优先的方式,区别对待不同频率的关键词,从而减少了大量磁盘I/O.CSTI胜过S2I的主要原因是PCTree没有重叠,PCTree的树高和节点数较少,空间剪枝能力较强.

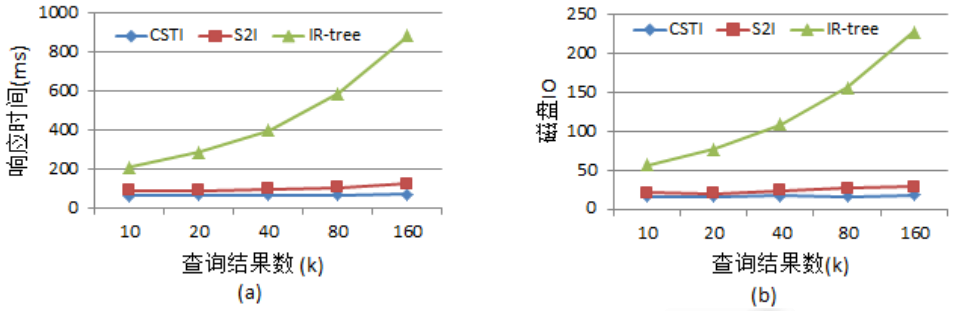

Fig.3 Respond time and I/O varying the number of results

图 3 响应时间和 I/O 随查询结果数变化情况

关键词数的影响.关键词个数对索引性能的影响由图 4 绘出.CSTI 和 S2I 的单关键词检索算法非常高效,随着查询关键词的增多,多关键词检索算法要合并多个单关键词检索的结果,因此响应时间和磁盘 I/O 均有所增多.而 IR-Tree 的性能相反,随着关键词的增多,R-Tree 节点关联的倒排索引的文本过滤能力增强,能够更快地终止检索.总的来说,CSTI 和 S2I 的性能依然胜过 IR-Tree 很多倍.而 CSTI 空间剪枝能力强,占用空间小,缓存命中率高,响应时间和磁盘 I/O 优于 S2I.

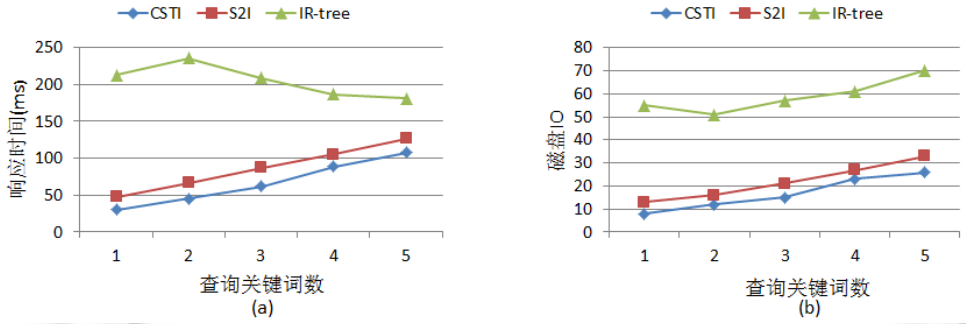

Fig.4 Respond time and I/O varying the number of query keywords

图 4 响应时间和 I/O 随查询关键词数变化情况

数据量的影响.图 5 展示出不同数据量的 Tweet 记录对索引的性能影响.随着数据量的增大,3 种索引的响应时间和磁盘 I/O 均有所增加.CSTI 和 S2I 呈线性增长,而 IR-Tree 有指数增长的趋势.对比 3 种索引的性能,CSTI 依然超过 S2I,二者都超过 IR-Tree 几乎 1 个数量级.

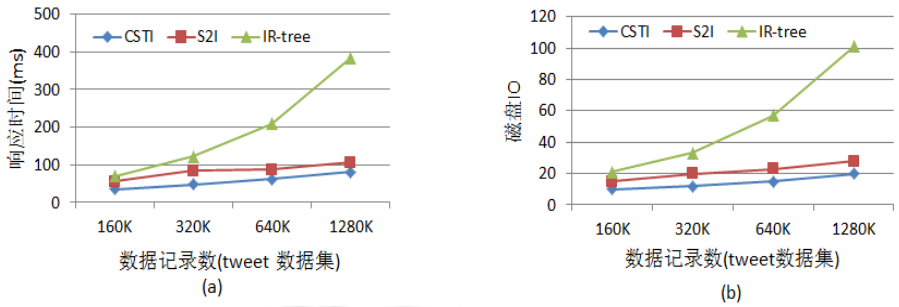

Fig.5 Respond time and I/O varying the cardinality of the Tweet dataset

图 5 响应时间和 I/O 随数据量(Tweet)变化情况

调节因子的影响.3 种索引对调节因子都很敏感,其对索引的响应时间和磁盘 I/O 的影响如图 6(a)和图 6(b)所示.CSTI 和 S2I 的性能随着空间重要性的提升而有所改善.因为增大了调节因子,CSTI 和 S2I 的空间剪枝能力

增强,提前终止了检索.IR-Tree 却有与之相反的表现,说明文本的过滤能力比空间的剪枝能力要强,返回 k 个结果的收敛速度变慢.

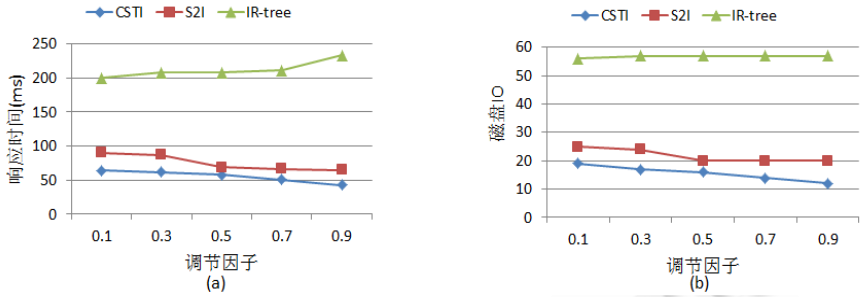

Fig.6 Respond time and I/O varying the query preference rate

图 6 响应时间和 I/O 随调节因子变化情况

Buffer size 的影响.图 7 给出 3 种索引的响应时间和磁盘 I/O 随 buffer size 的变化(从 0~64MB)情况.随着 buffer size 的增加,CSTI 和 S2I 的性能均有所改善,从变化幅度可知,CSTI 随 buffer size 的增加改善更多,这也验证了文献[10]的实验结果:随着 buffer size 的增加,空间开销越低的索引性能改善的越多.IR-Tree 尽管磁盘 I/O 有所减少(从 0~1MB 时显著减少),而总的响应时间没有缩短,主要原因是倒排文件占用空间太多,访问倒排文件消耗时间太久.

数据集的影响.图 8 给出 POI 和 Tweet 两个不同数据集对索引性能的影响.两种数据集中,CSTI 均优于 S2I 和 IR-Tree.当数据集中关键词数增多时,这种优势更加明显.

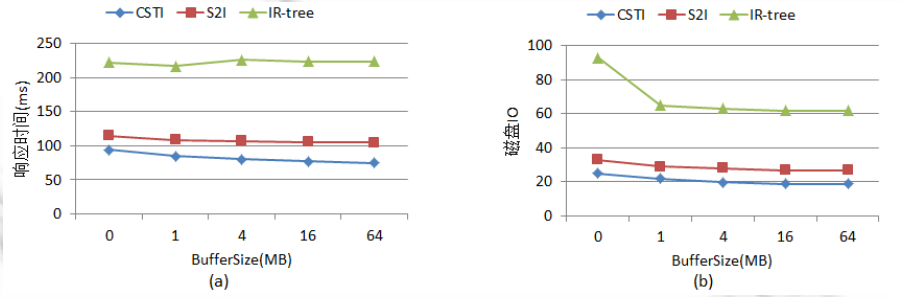

Fig.7 Respond time and I/O varying the buffer size

图 7 响应时间和 I/O 随 buffer size 变化情况

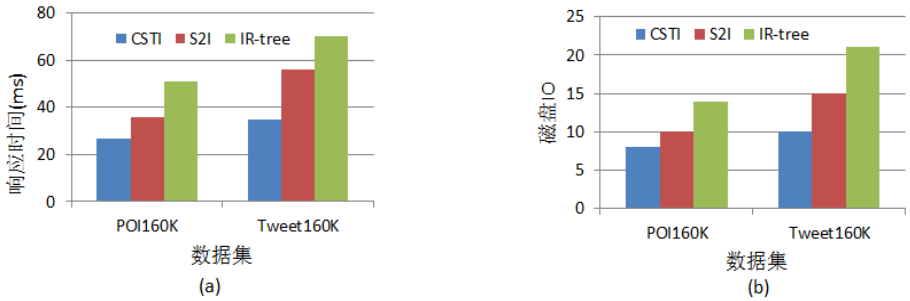

Fig.8 Respond time and I/O varying the different dataset

图 8 响应时间和 I/O 随不同数据集变化情况

6 总结与展望

本文提出一种基于压缩的索引——CSTI 和高效的检索算法以支持 top-*k* 空间关键词查询.与已有方法相比,CSTI 具有以下改进:

(1) PCTree 索引高频词的倒排列表.低频词关联对象少,直接存储节省了解压的代价,有利于快速响应查询.高频词关联对象较多,采用 PCTree 索引,增强了空间剪枝能力,极大地降低了存储开销,提高了缓存命中率,改善了查询性能.

(2) 时空开销小.CSTI 的合理设计支持实时响应查询.基于真实和仿真数据集的评估结果表明,CSTI 的空间开销节省 10 倍,时间开销也优于已有方法.

(3) 维护代价小.CSTI 中 PCTree 避免了 aR-tree 中的节点重叠,其前缀特性使其具有和 Quad-tree 相当的维护代价.空间索引采用查找表的方式将其独立出来,空间索引的更新维护代价大为降低.

尽管本文提出的方案是可行且高效的,然而空间关键词检索还有一些问题待解决.

(a) 寻找近似解.已有的方法为了找到 top-*k* 的精确解,采用 best-first 的搜索策略,而这种策略的无效扫描很多,代价昂贵,需要探索 depth-first 的搜索策略以寻找近似解,并给出与精确解的误差.

(b) 索引查询.目前的索引方法仍以索引数据为主,由于空间关键词检索和查询的关联密切,很难使得最相关的结果总排在倒排列表的最前面以提前终止检索,下一步将探索索引查询,通过探索查询日志,依据每次查询的结果重组倒排列表,提高检索性能.

References:

[1] Chen YY, Suel T, Markowetz A. Efficient query processing in geographic Web search engines. In: Chaudhuri S, Hristidis V, Polyzotis N, eds. Proc. of the ACM SIGMOD Int'l Conf. on Management of Data. Chicago: ACM, 2006. 277−288.

[2] Hariharan R, Hore B, Li C, Mehrotra S. Processing spatial-keyword queries in geographic information retrieval systems. In: Proc. of the 19th Int'l Conf. on Scientific and Statistical Database Management (SSDBM 2007). Banff: IEEE Computer Society, 2007. 16.

[3] Zhou Y, Xie X, Wang C, Gong Y, Ma WY. Hybrid index structures for location-based Web search. In: Herzog O, Schek HJ, Fuhr N, Chowdhury A, Teiken W, eds. Proc. of the 2005 ACM CIKM Int'l Conf. on Information and Knowledge Management. Bremen: ACM, 2005. 155−162.

[4] Khodaei A, Shahabi A, Li C. Hybrid indexing and seamless ranking of spatial and textual features of Web documents. In: Bringas PG, Hameurlain A, Quirchmayr G, eds. Proc. of the 21st Int'l Conf. on Database and Expert Systems Applications. Bilbao: Springer-Verlag, 2010. 450−466.

[5] Li Z, Lee KCK, Zheng B, Lee WC, Lee DL, Wang X. Ir-Tree: An efficient index for geographic document search. IEEE Trans. on Knowledge and Data Engineering, 2011,23(4):585−599.

[6] Felipe ID, Hristidis V, Rishe N. Keyword search on spatial databases. In: Alonso GA, Blakeley JLP, Chen A, eds. Proc. of the 24th Int'l Conf. on Data Engineering (ICDE 2008). Cancún: IEEE, 2008. 656−665.

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

[8] Rocha-Junior JB, Gkorgkas O, Jonassen S, Nørvåg K. Efficient processing of top-*k* spatial keyword queries. In: Pfoser D, Tao YF, Mouratidis K, Nascimento MA, Mokbel MF, Shekhar SS, Huang Y, eds. Proc. of the 12th Int'l Symp. on Advances in Spatial and Temporal Databases (SSTD 2011). Minneapolis: Springer-Verlag, 2011. 205−222.

[9] Papadias D, Kalnis P, Zhang J, Tao Y. Efficient OLAP operations in spatial data warehouses. In: Jensen CS, Schneider M, Seeger B, Tsotras VJ, eds. Proc. of the 7th Int'l Symp. on Advances in Spatial and Temporal Databases (SSTD 2001). Redondo Beach: Springer-Verlag, 2001. 443−459.

[10] Chen L, Cong G, Jensen CS, Wu D. Spatial keyword query processing: An experimental evaluation. PVLDB, 2013,6(3):217−228.

[11] Lu JH, Lu Y, Cong G. Reverse spatial and textual *k* nearest neighbor search. In: Sellis TK, Miller RJ, Kementsietsidis A, Velegrakis Y, eds. Proc. of the ACM SIGMOD Int'l Conf. on Management of Data (SIGMOD 2011). Athens: ACM, 2011. 349−360.

[12] Li GL, Feng JH, Xu J. Desks: Direction-Aware spatial keyword search. In: Kementsietsidis A, Salles MAV, eds. Proc. of the IEEE 28th Int'l Conf. on Data Engineering (ICDE 2012). Washington: IEEE Computer Society, 2012. 474−485.

[13] 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).

[14] Luo SQ, Luo YF, Zhou SG, Cong G, Guan JH: Distributed spatial keyword querying on road networks.In: Amer-Yahia S, Christophides V, Kementsietsidis A, Garofalakis MN, Idreos S, Leroy V, eds. Proc. of the 17th Int'l Conf. on Extending Database Technology (EDBT). Athens: OpenProceedings, 2014. 235−246.

[15] Zhang CY, Zhang Y, Zhang WJ, Lin XM, Cheema MA, Wang XY. Diversified spatial keyword search on road networks. In: Amer-Yahia S, Christophides V, Kementsietsidis A, Garofalakis MN, Idreos S, Leroy V, eds. Proc. of the 17th Int'l Conf. on Extending Database Technology (EDBT). Athens: OpenProceedings, 2014. 367−378.

[16] Yan H, Ding S, Suel T. Inverted index compression and query processing with optimized document ordering. In: Quemada J, León G, Maarek YS, Nejdl W, eds. Proc. of the 18th Int'l Conf. on World Wide Web (WWW 2009). Madrid: ACM, 2009. 401−410.

[17] Zhou X, Zhang X, Wang YH, Li R, Wang S. Efficient distributed multi-dimensional index for big data management. In: Wang JY, Xiong H, Ishikawa Y, Xu JL, Zhou JF, eds. Proc. of the 14th Int'l Conf. on Web-Age Information Management (WAIM 2013). Beidaihe: Springer-Verlag, 2013. 130−141.

附中文参考文献:

[13] 胡骏,范举,李国良,陈姗姗.空间数据上关键词模糊查询算法.计算机学报,2012,35(11):2237−2246.

周新(1987－),女,安徽宿州人,博士生,CCF 学生会员,主要研究领域为时空数据管理.
E-mail: zhouxin314159@163.com

张孝(1972－),男,博士,副教授,CCF 高级会员,主要研究领域为高性能数据库,非结构化数据管理,数据的智能获取与应用.
E-mail: zhangxiao@ruc.edu.cn

安润功(1992－),男,硕士生,主要研究领域为信息检索.
E-mail: anrungong@ruc.edu.cn

薛忠斌(1985－),男,博士生,主要研究领域为内存数据库,移动数据管理.
E-mail: zbxue@ruc.edu.cn

王珊(1944－),女,教授,博士生导师,CCF 会士,主要研究领域为高性能数据库,内存数据库,非结构化数据库,数据仓库与数据分析.
E-mail: swang@ruc.edu.cn