一种基于密度网格索引的 k-最近邻查询算法

章登义,李 想
(武汉大学计算机学院,湖北武汉 430072)

摘 要: 基于位置的服务的迅速发展对服务响应的效率提升和成本控制提出了更高的要求,本文提出了一种基于密度网格索引的 k-最近邻查询算法,该算法首先利用矩形的几何特点获取一系列候选搜索半径,随后根据移动对象的密度分布情况选择适当的候选搜索半径进行距离过滤,尽量减少不必要的内存索引单元和磁盘索引单元的访问. 实验表明,实现了本文算法的密度网格索引在 k-最近邻查询的查询效率上与 ST^2B-tree 不相上下,而查询的 I/O 代价与其他索引结构相比有明显的优势.

关键词: k-最近邻查询; 移动对象; 密度网格; 候选搜索半径
中图分类号: TP301 **文献标识码:** A **文章编号:** 0372-2112 (2017)02-0376-08
电子学报 URL: http://www.ejournal.org.cn **DOI:** 10.3969/j.issn.0372-2112.2017.02.016

A k-Nearest Neighbor Query Algorithm for Density Grid-Based Index

ZHANG Deng-yi, LI Xiang
(*School of Computer, Wuhan University, Wuhan, Hubei* 430072, *China*)

Abstract: The rapid development of location based services set higher demands on efficiency promotion and cost control of the services. In the paper, we propose a k-nearest neighbor query algorithm based on density grid index. In processing of the algorithm, a series of candidate search radii is obtained by utilizing of the geometrical features of the rectangle. Then the appropriate candidate search radii are chosen to make distance filtering according to the density distribution of the moving object, it is useful to achieve reducing the unnecessary accessing to memory index units and disk index units. Our extensive experiments show that the efficiency of the density grid index with our algorithm is about equal to ST^2B-tree on the k-nearest neighbor query, but our algorithm has obvious advantages in the cost of I/O.

Key words: k-nearest neighbor query; moving objects; density grid; candidate search radii

1 引言

随着无线传感器网络和 GPS 定位技术的迅速发展,新兴了一大批基于位置的服务,它们能够根据用户的位置信息为用户提供有针对性和个性化的信息服务,例如当顾客经过某个商圈时,其移动电话中将接收到附近商店的促销广告和打折商品信息;用户在驾驶车辆的过程中能够及时获取前往目的地道路的交通拥堵状况等. 目前,基于位置的服务所面向的用户数量在不断增加,因此如何高效存储用户的位置信息以及快速的响应用户的服务需求成为了研究的热点.

移动对象数据库(Moving Object Databases,简称 MOD)的提出为高效的管理海量移动对象位置信息建立了基础. 随着 MOD 的发展,针对多维空间中的移动对象,研究者们提出多种索引结构和 k-最近邻查询算法,但是目前采用的 k-最近邻查询算法通常没有考虑数据的分布情况,对于移动对象的密度分布情况不同的区域统一采用蛮力法进行搜索查找,导致磁盘访问开销较大.

本文针对移动对象分布不均匀的情况提出了一种基于密度网格索引的 k-最近邻查询算法,该算法利用矩形的几何特点获取一系列候选搜索半径,然后根据移动对象的密度分布情况选择适当的搜索半径访问网格单元,对访问到的网格单元内的移动对象进行距离过滤,以减少不必要的访问开销.

收稿日期:2015-09-11;修回日期:2016-01-29;责任编辑:蓝红杰
基金项目:国家自然科学基金(No.60903035,No.41001296);国家 863 高技术研究发展计划(No.2013AA12A301)

2　相关工作

2.1　空间索引

针对移动对象随时间而发生地理空间上的变化的特点,一些研究者在 R-tree[1] 的基础之上进行了改进,其中 Saltenis 等人[2] 提出的 TPR-tree 在外包矩形(Bounding Rectangles)内引入时间函数,每个对象用其位置信息、相应的时刻以及速度向量表示,使得其能够预测移动对象未来的位置. Tao 等人[3] 提出了 TPR * -tree,采用与 TPR-tree 相同的索引结构,对索引的更新算法进行改进,提升了索引的 I/O 性能. Zheng[4] 在 TPR-tree 的基础之上引入多时间版本的概念,以支持移动对象的全时态查询. 与基于 R-tree 的索引相比,基于 B + -tree 的索引结构的更新性能有显著的优势,这是由于 B + -tree 中不存在 R-tree 中节点之间空间范围重叠的情况. Jensen 等人[5] 提出了第一个基于 B + -tree 的移动对象索引结构 B^x-tree,通过使用空间填充曲线[6] 实现高维到低维的映射,该索引更新效率较高,但由于未考虑对象的速度,因此查询性能不理想. 为了提升索引的查询性能,Man 等人[7] 提出了 B^{dual}-tree,使用四维 Hilbert 空间填充曲线将移动对象的位置和速度映射为一维点,利用速度信息达到更好的查询性能. 针对数据分布的变化对索引查询性能的影响,Chen 等人[8] 提出了 ST^2B-tree,通过重建子树的方式调整空间索引的粒度来应对空间位置不断变化的移动对象的查询. 除了上述两大类索引之外,基于空间网格的索引结构近年来也成为了研究的热点. 其中 Sidlauskas 等人[9] 提出的均匀网格索引 PGrid,将空间区域划分成大小相等的网格,移动对象根据自身的坐标分属不同的单元网格,每个单元网格指向一系列存储移动对象信息的数据桶(bucket),每个数据桶(bucket)对应一个磁盘页. 由于空间对象的总数量和单元网格的容量决定了均匀网格的空间粒度,因此数据分布的情况对均匀网格的影响很大,偏斜分布的数据将导致查询时间的线性增长,同时对存储空间的利用率降低. Xiong 等人[10] 提出的 LU-Grid 采用而密度格网索引,空间区域不再划分成尺寸相同的网格,每个网格单元指向一个数据桶(bucket)作为存储单位来存放记录,数据桶的存储空间的溢出将导致网格的分裂,而网格的合并发生在两个层次上:数据桶的合并和网格的合并. Cudre-Mauroux 等人[11] 提出的存储系统中采用的四叉树结构与 LU-Grid 的索引结构类似,区别在于它将空间区域划成四个相等的子空间,任意子空间中的对象数量超过阈值则将该子空间继续划分下去直至每个子空间中的移动对象数量不超过阈值为止,若分裂自同一区域的四个子空间的对象总数量小于阈值则发生合并.

2.2　k-最近邻查询算法

基于上述的空间索引已有若干种 k-最近邻查询算法[12],其中最常见的 k-最近邻查询算法是 Best-First[3,7,13],其基本思想是将搜索路径上的节点放入一个优先队列(或者堆),队列的关键值为队列中元素到查询点的距离,距离越小优先级越高,通过节点出队的同时将该节点内的移动对象入队的方式获取查询结果点,直至出队的移动对象数量满足查询条件,这种查询算法适用于分层的空间索引[14],不需要考虑数据的分布情况. k-最近邻查询算法的过程一般分为两个阶段,在粗过滤阶段获取满足查询条件数量的候选结果点,该阶段通常采用不断扩大搜索半径的范围查询来获取对象,然后在精过滤阶段使用各种数据结构对候选结果点进行验证,例如 Hu 等[15] 提出的 k-最近邻查询方法利用 Voronoi 图[16] 划分二维空间对查询候选结果进行验证;殷晓岚[17] 对空间网络进行划分,然后通过两阶段过滤的方法得到 k-最近邻查询结果. 基于空间划分的方法还有 Mouratidis 等[18] 基于均匀网格索引提出的概念网格划分法,该方法将查询点周围的网格单元按照其与查询点的距离进行分组,按照距离由近及远的顺序访问每个分组内网格单元中的对象,直至访问到的对象数量大于等于 k 为止,然后将访问到的对象进行验证. 而 Jiang 等[19] 采用与概念网格划分法类似的方法,区别在于其将查询点周围的网格单元放入优先队列,优先队列内按照与查询点的距离大小递增排序.

3　索引结构

本文索引结构[20] 分为内存网格和磁盘网格两个部分,采用与网格文件类似的分裂和合并策略,本文的索引结构中循环的依次分裂每个维度,但是为了处理和实现的方便,我们选取中点为分裂点. 当一个磁盘网格单元分裂时,它对应的内存网格可能不受影响,合并时我们使用 kd-tree[21] 来指示磁盘网格单元及其兄弟,即分裂自同一块磁盘的两块磁盘,因此本文的索引结构中合并只会在互为兄弟的磁盘网格单元之间发生. 图 1 为本文索引结构的示意图.

如图 1 所示,假设一个磁盘网格单元最多能够存储两个对象. 图 1.1(b)中有 5 块磁盘分别是 A,B,C,D,E,九个对象(o_1 – o_9)分别存储在这五块磁盘上. 图 1.1(a)给出了对应的内存网格,由九个网格单元($cell_1$ – $cell_9$)组成,此时一个更新到达网格单元 $cell_7$,对象 o_{10} 进入了 $cell_7$,然后这个对象被写入对应的磁盘 C. 磁盘 C 此时有三个对象(o_5,o_6,o_{10}),超出了存储阈值,磁盘 C 将分裂成两块新的磁盘(F 和 G),内存网格在相同的位置分裂. 分裂操作完成后,对象 o_{11} 到达网格 $cell_9$,然后该对象被写入到磁盘 B 上. 由于有三个内存网格单

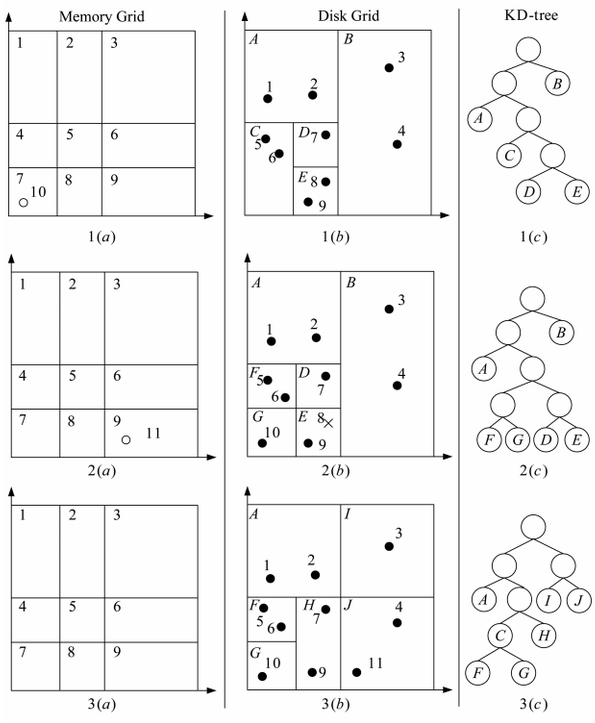


图1 索引结构示意图

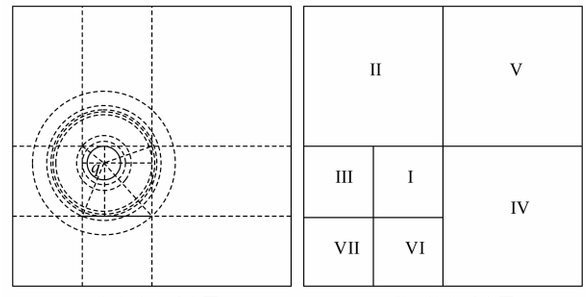
元 $cell_3$,$cell_6$ 和 $cell_9$ 对应磁盘 B,因此,磁盘 B 分裂成两块新的磁盘(I 和 J),如图可知对应的内存网格单元之前已经分裂. 图 1.2(b)中,对象 o_8 是一个过期的对象,它被清除后,磁盘 E 试图与其兄弟磁盘合并,由图 1.2(c)可知 E 的兄弟磁盘是 D,因此 D 和 E 发生磁盘合并,产生一个新的磁盘 H.

4 算法与分析

本节给出在上述密度网格索引的基础上的一种 k-最近邻查询算法的描述.

4.1 算法描述

如图 2(a)所示,本算法首先根据查询点 q 的坐标定位其所在的内存网格单元 m,假定网格单元 m 内的对象数量为 h,根据矩形的几何特点和查询点 q 的坐标,从查询点 q 向其所在的内存网格单元 m 所表示的矩形的四条边作垂线,并且向矩形的四个顶点做连线,获取到 8 个候选搜索半径,取最大的半径 $r_{\max}$,根据面积之比估计圆(q,$r_{\max}$)内的点数量,若$\frac{\pi r_{\max}^2}{s_m}h<k$,则在 m 的父节点网格上获取 8 个新的候选搜索半径,采取上述相同的操作直至满足$\frac{\pi r_{\max}^2}{s_m}h\geqslant k$. 若$\frac{\pi r_{\max}^2}{s_m}h\geqslant k$,则进入实际搜索过程,分为两个阶段,第一个阶段称为粗过滤,利用候选搜索半径与查询点 q 构成的圆的外接矩形进行范围搜索,记录搜索涉及到的磁盘页地址;第二个阶段称为精过滤,如图 2(b)所示,磁盘Ⅰ、Ⅱ、Ⅲ、Ⅳ、Ⅴ、Ⅵ、Ⅶ分别是图 2(a)中内存网格单元对应的磁盘,根据此时的搜索半径覆盖的范围,按照罗马数字的先后顺序读取磁盘内容,判断磁盘上的移动对象与查询点的欧氏距离是否小于此时的搜索半径 r.



(a) 内存搜索示意图 (b) 磁盘搜索示意图

图2 算法原理示意图

算法 1 的初始化阶段,首先维护一个临时查询结果集 temp(行 1),再根据查询点 q 的坐标获取查询点 q 所在的网格单元 m(行 2),然后根据查询点 q 和网格单元 m 获取 8 个候选搜索半径,放入集合 r(行 4). 获取到候选搜索半径之后,将 0 放入搜索半径集合 r 并按照元素的大小进行递增排序(行 5-6),然后开始遍历搜索半径集合 r 中除 0 之外的半径(行 8),然后计算当前的查询半径 r_i 与查询点 q 构成的圆的外接矩形 square(行 9),接着通过与矩形 square 范围相交的所有内存网格,获取它们对应的磁盘页,放入磁盘页集合 dlist 中(行 10). 将 dlist 中的磁盘内与查询点距离大于等于 r_{i-1} 小等于 r_i 的对象按照其与查询点的距离的大小(增序)存入临时结果 temp,且标记对象已经存入临时结果中(行 11-13),然后将临时结果并入查询结果 S 中,将 temp 清空(行 14),此时判断查询结果中对象的数量是否大于等于查询参数 k(行 15),若大于等于 k,则退出循环,查询完毕,否则继续扩大搜索半径进行查询. 最终返回 S 中的前 k 个对象最为最终的查询结果(行 19).

算法 1 KNNSearch(q,k)

输入:查询点 q,参数 k
输出:查询结果 S
开始
1. $S\leftarrow\phi$,内存网格单元 m,临时查询结果集 temp$\leftarrow\phi$
2. $m\leftarrow$FindMGCell(q) //定位查询点 q 所在的内存网格单元
3. 查询半径集合 $r\leftarrow\phi$
4. $r\leftarrow$ComputeSearchRadius(q,m) //获取最终的 8 个候选搜索半径
5. $r\cup\{0\}$
6. SelectSort(r) //根据候选半径的长度增序排序
7. 磁盘页集合 dlist$\leftarrow\phi$

```
8. for each r_i  in r  and i>0  do
9.   square←ComputerBoundSquare(q,r_i)  //计算查询半径与查询点构成的圆的外接矩形
10. dlist∪MGCellSearch(square)  //搜索当前查询半径涉及的内存网格单元对应的磁盘页
11.   for each distcell  in dlist  do
12.     temp←DiskKnnSearch (q,diskcell,r_{i-1},r_i)//放入临时结果集
13.   end for
14.   S∪temp,temp←φ
15.   if |S|≥k  then
16.       break;
17.   end if
18. end for
19. return S 中的前 k 个对象
结束
```

4.2　效率分析

假定密度网格中单元网格总数为 $l*p$,移动对象总数量为 N,一个磁盘页内的对象个数阈值为 u,在最理想的情况下执行一次 k-最近邻查询需要访问的磁盘页数量为 n.

本文的算法查询过程中消耗的时间主要分为四个部分:(1)查找查询点所在的网格单元,获取 8 个候选半径;(2)读取磁盘页;(3)判断对象与查询点的距离是否在搜索半径内;(4)将在搜索半径内的对象按照与查询点的距离的大小递增的顺序插入临时结果.

第一部分,查找查询点 q 所在的网格单元过程的最坏时间复杂度为 $O(l+p)$,计算 8 个候选半径的时间复杂度为常数,因此时间复杂度为 $O(l+p)$;第二部分,假定在使用第一个搜索半径 r_1 进行搜索时,得到的与查询相关的磁盘页集合大小为 h,如图 2(b)所示,每次扩大搜索半径将从磁盘上读取一个未被访问过的磁盘页,假定查询过程中扩展搜索半径的次数为 $i(1\leqslant i\leqslant 7)$,因此时间复杂度为 $O(h+i)$;第三部分,判断对象与查询点的距离是否在搜索半径内需进行距离计算的次数最多为 $(h+i)u$,因此时间复杂度为 $O((h+i)u)$;第四部分,假定在扩展 i 次搜索半径之后查询结束,则临时结果集内点的数量累计值的范围为 $k\sim(h+i)u$,因此将对象按照与查询点的距离的大小递增的顺序插入临时结果的时间复杂度为 $O(k)\sim O(h+i)u$. 本文算法的总时间复杂度为 $O(l+p+(h+i)(1+u)+k)\sim O(l+p+(h+i)1+2u)$(其中$(h+i)\geqslant n$).

若采用 ST^{2}B-tree 中的查询半径以固定增量扩展的方法,则消耗的时间主要分为:(1)定位查询点所在的网格单元;(2)进行扩展查询范围读取磁盘页获取候选点;(3)判断扩展的那部分范围内的点是否在当前查询范围所覆盖的正方形的内切圆内;(4)对查询的候选结果进行验证.

第一部分的时间复杂度同本文的算法相同,在最坏的情况下为 $O(l+p)$;第二部分,查询半径以固定的长度增量进行多次扩展查询,实际的时间消耗主要为磁盘页读取,假定当前查询范围的内切圆内的对象数量满足查询条件时访问磁盘页的数量为 $j(j\geq n)$,时间复杂度为 $O(j)$;第三部分,进行距离计算次数最多为 $j*u$,因此时间复杂度为 $O(ju)$;第四部分,按照与查询点距离对候选结果集进行递增排序,返回前 k 个对象为最终查询结果,由于查询结果集在总体上有序,因此采用插入排序的时间复杂度为 $O(ju)$. 因此总的时间复杂度为 $O(l+p+j(1+u)+k)\sim O(l+p+j(1+2u))$(其中$j\geq n$).

经过分析可知上述两种查询方法的时间复杂度均由主要由查询过程中访问到的磁盘页数量和移动对象与查询点的距离计算次数决定,而移动对象与查询点距离计算次数由磁盘页数量以及磁盘页内的对象数量共同决定.

证明:以图 2 为例,假设查询点 q 到第 k 个最近邻对象的距离为 d_k,若此时采用本文的方法在查询半径 r_i 为时完成查询,此时 $r_{i-1}<d_k<r_i$,则访问的磁盘页数为 i,若采用查询半径以固定增量扩展方法,其按照均匀分布估计的查询点 q 到第 k 个最近邻对象的距离 D_k,半径增量为 $\Delta r=D_k/k$,在移动对象分布稀疏时会导致 $d_k>D_k$,为满足查询条件当查询半径等于 D_k 时继续扩展,使得 $D_k+c*\Delta r\geq d_k(c>0)$,由于无法保证最后一次扩展是否会访问到不必要的磁盘页,因此该方法访问的磁盘页数量必然大于等于 i,在移动对象分布稠密时会导致 $d_k<D_k$,为满足查询条件,使得 $c*\Delta r\geq d_k(c>0)$,同理访问的磁盘页数量必然大于等于 i.

因此在移动对象分布不均匀的情况下本文算法的时间复杂度中磁盘访问数量 $h+i\leq j$,从而使得本文算法的距离计算次数$(h+i)u\leq j*u$.

5　实验结果

为了验证算法的查询性能,我们使用 C ++ 在已有框架[22]中实现了上述的密度网格索引(以下简称 DGrid)和 k-最近邻查询算法. 实验的硬件平台为:Intel® CoreTM i7-3630QM 2.4GHz CPU,16G 内存和 750GB 硬盘;软件环境为 Win7 操作系统和 QT Creator 编译系统. 实验数据为通过数据生成软件生成高斯分布数据和真实城市道路网①中的数据. 实验的比较对象为是目前比较先进的索引结构 B^x-tree,B^{dual}-tree,TPR*-tree 和 ST^{2}B-tree,其中 B^x-tree 和 ST^{2}B-tree 采用等增量扩展范围查询

① U. S. Census Bureau-TIGER/Line. http://www.census.gov/geo/www/tiger/.

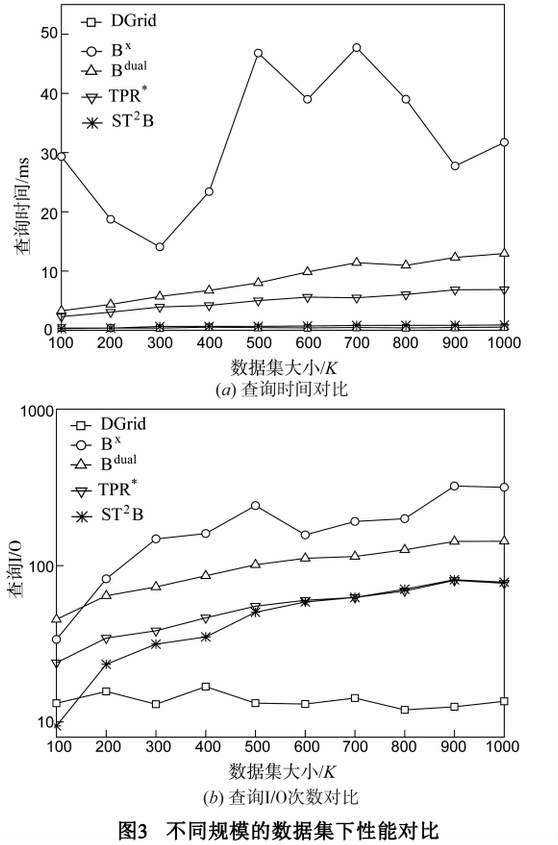
方法实现 k-最近邻查询，B^{dual}-tree 和 TPR^*-tree 采用 Best-First 算法实现 k-最近邻查询. 实验参数如表 1 所示，表中加粗的数值为默认值.

表 1 实验参数表

参数	设置
空间范围	$100000 * 100000m^2$
数据集规模	100K,200K,300K,…,**1M**
热点数量	1,**10**,100,1000,10000
查询参数 k	10,20,30,40,50,60,70,80,90,**100**
磁盘页面大小(KB)	1,2,**4**,8
缓冲区大小(页数量)	16,32,**64**,128,256
城市道路网	奥登堡,新加坡,旧金山

5.1 不同规模的数据集下的性能比较实验

图 3 分别给出了五种索引在数据集的规模从 10 万增长到 100 万的过程中执行查询所消耗的时间和 I/O 次数的变化情况. 从图 3(a)中可以看出，除 B^x-tree 以外的四种索引的查询时间略有增长，其中 DGrid 索引和 ST^2B-tree 的查询时间性能十分接近，优于其他三种参测索引. 如图 3(b)所示，DGrid 索引的查询所需的 I/O 次数变化不显著，另外四种参测索引查询所需的 I/O 次数均而呈现明显增长趋势，在数据集的规模超过 20 万之后，DGrid 索引查询消耗的 I/O 次数是五种参测索引中最少的，这是由于其他四种参测索引的查询过程中需要访问中间节点数量增加.



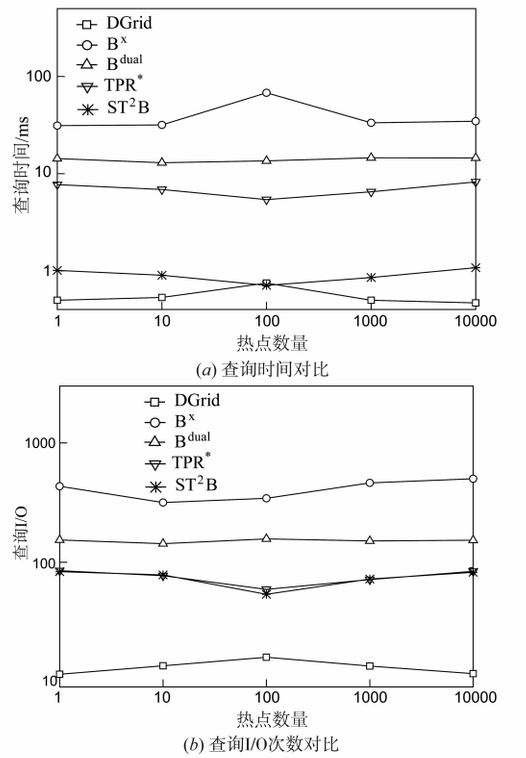
(a) 查询时间对比

(b) 查询I/O次数对比

图3 不同规模的数据集下性能对比

5.2 不同分布的数据集下的性能比较实验

图 4 分别给出了五种索引在数据集的分布情况变化过程中执行查询所消耗的时间和 I/O 的情况. 从图 4(a)可以看出当数据集的分布变得越来越不均匀时，B^x-tree 和 DGrid 索引的查询所消耗的时间有显著的增长，而 B^{dual}-tree 查询所消耗的时间几乎不受数据分布的影响，TPR^*-tree 和 ST^2B-tree 的查询时间性能有略微的提升. 当热点数量达到 100 时，数据的分布最为不均匀，此时 ST^2B-tree 的查询时间性能与 DGrid 索引相比略有优势. 如图 4(b)所示，TPR^*-tree 和 ST^2B-tree 的查询所消耗的 I/O 次数比较接近，均随着数据分布的不均匀程度的变大而减少，B^x-tree 的查询 I/O 次数随着热点数量的变化有所波动，而 B^{dual}-tree 的查询 I/O 次数变化不显著，DGrid 索引的在热点数量达到 100 时的查询 I/O 次数明显高于其他的分布情况下的查询 I/O 次数，这是由于此时的密度分布的不均匀导致一些查询需要访问更多的磁盘页.



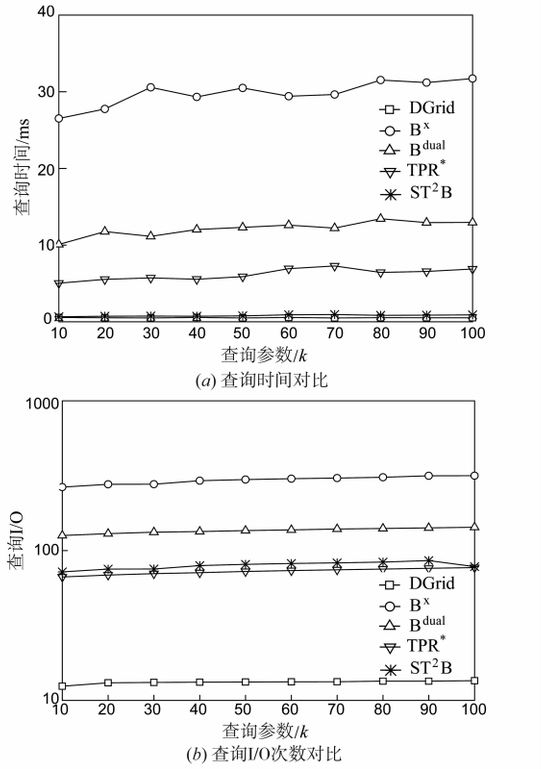
(a) 查询时间对比

(b) 查询I/O次数对比

图4 不同分布的数据集下性能对比

5.3 不同查询参数 k 下的性能比较实验

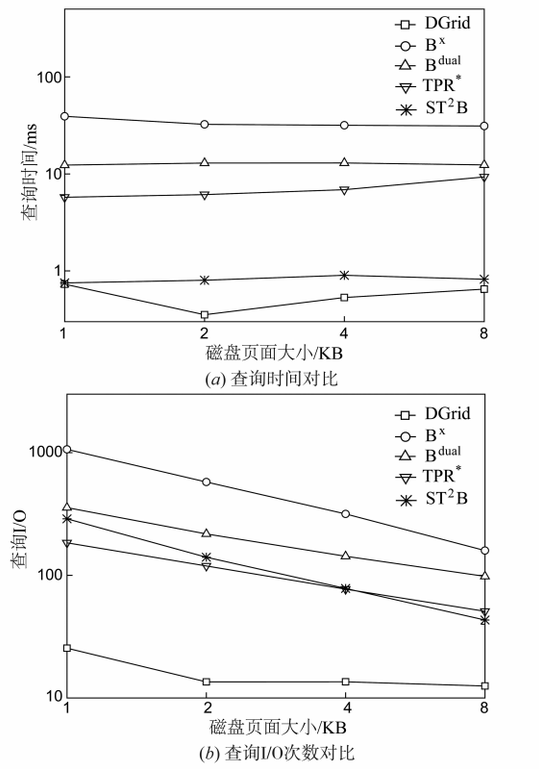
图 5 给出了查询参数 k 从 10 增大到 100 过程中五种索引的 k-最近邻查询的时间和 I/O 性能的对比情况. 如图 5(a)所示，从总体上看，参数 k 变化对五种参测索引的查询时间性能的影响较小，其中 DGrid 索引的查询



(*a*) 查询时间对比

(*b*) 查询I/O次数对比

图5　查询参数k对查询性能的影响对比实验



(*a*) 查询时间对比

(*b*) 查询I/O次数对比

图6　不同大小的磁盘页面对查询性能的影响对比实验

时间性能与 ST^{2}B-tree 十分接近. 从图 5(*b*)中可以看出,五种参测索引的查询 I/O 性能也不受查询参数 k 的影响,而 DGrid 索引的查询 I/O 性能显著优于其他四种索引.

5.4　不同大小的磁盘页面下的性能比较实验

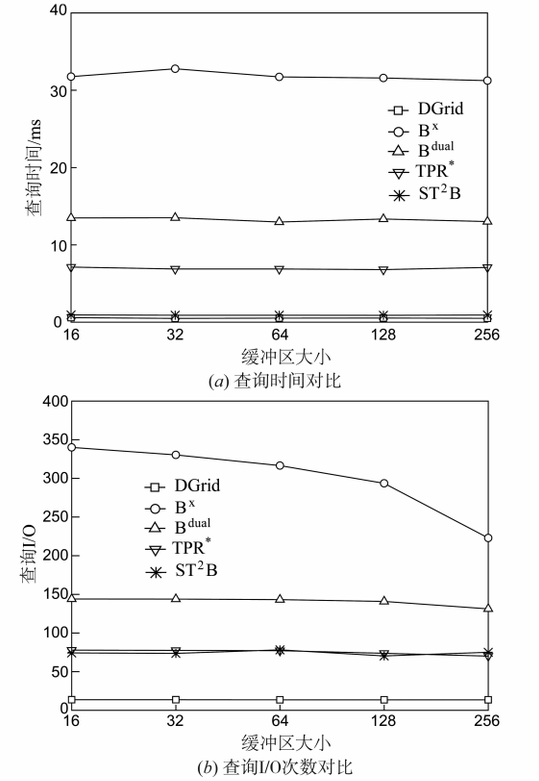
图 6 给出了磁盘页面从 1KB 增大到 8KB 的过程中五种索引的 k-最近邻查询性能对比情况. 如图 6(*a*)所示,总体而言,磁盘页面大小变化对 B^x-tree、B^{dual}-tree 和 ST^{2}B-tree 的查询时间性能影响较小,TPR*-tree 的时间性能随着磁盘页面大小的提高有所降低,而对于 DGrid 索引,当磁盘页面大小为 2KB 时,其查询时间性能达到最优. 从图 6(*b*)可以看出对于五种索引,磁盘页面大小的提高均使得其查询 I/O 次数的减少,这是由于磁盘页内的移动对象数量增长导致访问同样数量的对象所需要读取的磁盘数量减少.

5.5　不同大小的缓冲区下的性能比较实验

图 7 给出了缓冲区的页数量从 16 增大到 256 的过程中五种索引的查询性能对比情况. 如图 7(*a*)所示,总体而言,查询缓冲区的页数量变化对五种参测索引的查询时间性能影响很小. 同样,从图 7(*b*)中可以看出,除了 B^x-tree 外的其他四种索引的 k-最近邻查询 I/O 性能几乎没有受缓冲区大小变化的影响.

5.6　不同城市路网下的性能比较实验

如图 8 所示,分别给出了实验中使用的奥登堡、新加坡和旧金山三座城市的道路网. 如图 9(*a*)和(*b*),在



(*a*) 查询时间对比

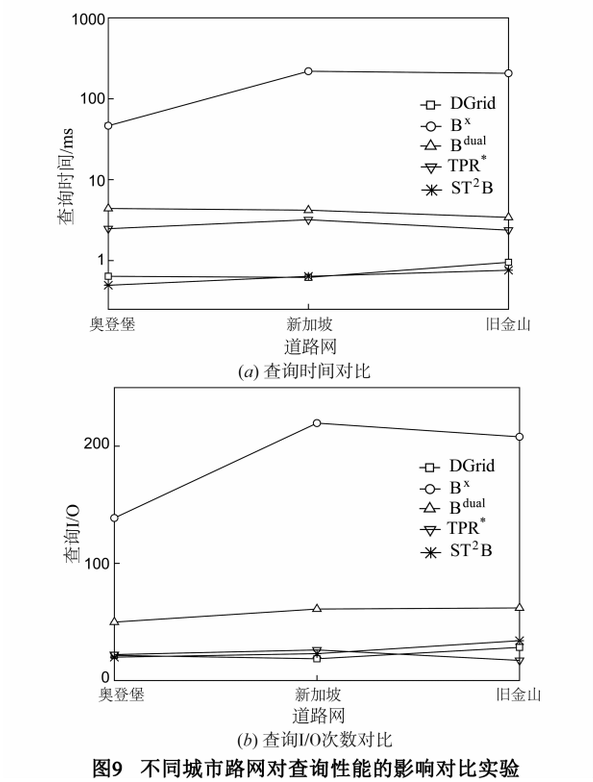
(*b*) 查询I/O次数对比

图7　缓冲区大小对查询性能的影响对比实验

奥登堡的道路网中,ST^2B-tree 的查询性能略优于 DGrid,这是由于此时移动对象的数量较少且分布均匀,同时 ST^2B-tree 的查询算法是以均匀分布为前提来计算半径扩展值. 在旧金山的道路网中,移动对象数量最多且分布变得极度不均匀,ST^2B-tree 和 DGrid 的查询算法最终得到的查询半径过大,访问多余的磁盘页,导致查询性能降低. 新加坡的道路网在移动对象数量和分布上较其他两个城市更为均衡,使得 DGrid 在五种参测索引中性能最佳.

(*a*)奥登堡 (*b*)新加坡 (*c*)旧金山

图8 城市道路网



(*a*) 查询时间对比

(*b*) 查询I/O次数对比

图9 不同城市路网对查询性能的影响对比实验

6 小结

本文提出了一种基于密度格网索引的 k-最近邻查询算法,该算法根据移动对象的分布情况和矩形的几何特性获取候选搜索半径集,在扩展查询范围的同时采用两阶段过滤的方式得到查询结果点,实验表明采用该算法的密度网格索引的查询效率与几种对比索引相比有一定的优势,且查询 I/O 性能优势较为明显.

参考文献

[1] Guttman A. R-trees: A dynamic index structure for spatial searching[A]. International Conference on Management of Data[C]. New York: ACM, 1984: 47 – 57.

[2] Ⅵaltenis S. Indexing the Positions of Continuously Moving Objects[M]. ACM SIGMOD Record. ACM, 2000. 331 – 342.

[3] Tao Y, Papadias D, Sun J. The tpr * -tree: an optimized spatio-temporal access method for predictive queries[A]. VLDB[C]. San Francisco: Morgan Kaufmann Publishers, 2003. 790 – 801.

[4] Zheng Y. A fast index method for moving objects on full temporal query[A]. Computer Research and Development (ICCRD)[C]. IEEE, 2011. 205 – 208.

[5] Jensen C S, Lin D, Ooi B C. Query and update efficient b + -tree based indexing of moving objects[A]. Thirtieth International Conference on Very Large Data Bases-volume [C]. San Francisco: Morgan Kaufmann Publishers, 2004. 768 – 779.

[6] Bader M, Bungartz H J, Mehl M. Space-Filling Curves [M]. Encyclopedia of Parallel Computing. Springer US, 2011. 1862 – 1867.

[7] Man L Y, Tao Y, Mamoulis N. The bdual-tree: indexing moving objects by space filling curves in the dual space [J]. VLDB Journal, 2008, 17(3): 379 – 400.

[8] Chen S, Ooi B C, Tan K L, et al. ST^2B-tree: a self-tunable spatio-temporal b + -tree index for moving objects[A]. International Conference on Management of Data[C]. New York: ACM, 2008. 29 – 42.

[9] Sidlauskas D, Altenis S, Jensen C S. Parallel main-memory indexing for moving-object query and update workloads [A]. Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data[C]. ACM, 2012. 37 – 48.

[10] Xiong X, Mokbel M F, Aref W G. LUGrid: update-tolerant grid-based indexing for moving objects[A]. Proceedings of the 7th International Conference on Mobile Data Management[C]. IEEE Computer Society, 2006. 13.

[11] Cudre-Mauroux P, Wu E, Madden S. TrajStore: an adaptive storage system for very large trajectory data sets[A]. IEEE International Conference on Data Engineering[C]. IEEE Computer Society, 2010. 109 – 120.

[12] Ilarri S, Mena E, Illarramendi A. Location-dependent query processing: where we are and where we are heading[J]. ACM Computing Surveys, 2010, 42(3): 1283 – 1310.

[13] Zhong R, Li G, Tan K L, et al. G-tree: an efficient index for kNN search on road networks[A]. Proceedings of the 22nd ACM international conference on Conference on in-

formation & knowledge management[C]. ACM,2013. 39 -48.
[14] You J K, Patel J M. Performance comparison of the r * -Tree and the quadtree for kNN and distance join queries [J]. Knowledge & Data Engineering IEEE Transactions on,2010,22(7):1014 -1027.
[15] Hu L, Ku W S, Bakiras S, et al. Verifying spatial queries using voronoi neighbors [A]. Proc 18th SIGSPATIAL Int'l Conf Advances in Geographic Information Systems [C]. New York:ACM,2010. 350 -359.
[16] JYHJONGFU, Lee R C T. Voronoi diagrams of moving points[J]. Lecture Notes in Computer Science,2011,08 (3):365 -379.
[17] 殷晓岚. 动态网络空间中的 k-NN 查询[J]. 电子学报,2011,39(2):389 -394.
YIN Xiao-lan. K-nearest neighbors query in dynamic spatial network databases[J]. Acta Electronica Sinica,2011, 39(2):389 -394. (in Chinese)
[18] Mouratidis K, Papadias D, Hadjieleftheriou M. Conceptual partitioning: an efficient method for continuous nearest neighbor monitoring[A]. ACM Conference on Management of Data[C]. New York:ACM,2005. 634 -645.
[19] Jiang J, Bao H, Chang E Y, Li Y. MOIST: a scalable and parallel moving object indexer with school tracking[J]. Eprint Arxiv,2012,5(12):1838 -1849.
[20] 李想,章登义,李文海. 一种采用批量操作的移动对象的密度格网索引[J]. 小型微型计算机系统,2015,36 (10):2235 -2239.
LI Xiang, ZHANG Deng-yi, LI Wen-hai. Batch operation density grid-based index for moving objects[J]. Journal of Chinese Computer Systems,2015,36(10):2235 -2239. (in Chinese)
[21] Hanan Samet. Foundations of Multidimensional and Metric Data Structures[M]. Morgan Kaufmann Publishers, University of Maryland at College Park,2006.
[22] Chen S, Jensen C S, Lin D. A benchmark for evaluating moving object indexes[A]. PVLDB[C]. San Francisco: Morgan Kaufmann Publishers,2008. 1574 -1585.

作者简介

章登义　男,1965 年生于湖北省荆州市. 武汉大学计算机学院教授,博士生导师.

李　想(通信作者)　男,1987 年生于湖北黄冈. 博士研究生,研究方向为移动对象、时空数据库、轨道数据挖掘.
E-mail:lixiang1987@ whu. edu. cn