# 一种基于改进网格多维 TTI 索引的动态 Top-$k$ 查询算法

邓丹苹[1)] 秦小麟[1),2)] 李博涵[1),2),3)] 郑 伟[1)] 刘 亮[1),2)] 李 雪[1),4)]

[1)](南京航空航天大学计算机科学与技术学院 南京 210016)

[2)](江苏省软件新技术与产业化协同创新中心 南京 210016)

[3)](江苏省易图地理信息科技股份有限公司 江苏 扬州 225000)

[4)](昆士兰大学信息技术与电子工程学院 布里斯班 澳大利亚 4067)

**摘 要** Top-$k$ 查询是目前海量数据在动态环境中高效处理的重要方法之一.在许多实际应用中,满足用户偏好的 top-$k$ 查询一般由两个部分组成:选择条件和排序函数.用户可自行设置排序函数,也可选择对不同数据子集进行查询.在传统数据库领域中已经对 top-$k$ 算法进行了深入的研究,但是现有的方法不适用于大量目标对象的属性值发生动态变化的情况.在查询过程中由于目标对象的属性值发生改变可能导致查询结果的改变,从而对算法性能有更高的要求.围绕动态 top-$k$ 计算问题,在网格索引的基础上提出了 TTI 索引,通过 TTI 索引中的概要信息高效计算网格 $k$ 支配能力并划分影响区和自由区.根据划分的区域裁剪数据集并降低数据动态变化时需重新计算发生的概率.实验中采用多种数据集进行测试,分别与 top-$k$、RankCube 和 CIA 算法进行了比较.实验结果验证了算法的有效性,实验数据表明在静态情况下,该文算法的查询效率可比传统 top-$k$ 算法最多快至 8 倍,动态情况下可比传统 top-$k$ 算法最多快 10 倍.

**关键词** Top-$k$ 查询;网格索引;分区;概要;动态

**中图法分类号** TP311 **DOI 号** 10.11897/SP.J.1016.2019.01827

# A Dynamic Top-$k$ Query Based on the Improved Grid Multi-Dimensional Index TTI

DENG Dan-Ping[1)] QIN Xiao-Lin[1),2)] LI Bo-Han[1),2),3)] ZHENG Wei[1)] LIU Liang[1),2)] LI Xue[1),4)]

[1)](*College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing* 210016)

[2)](*Collaborative Innovation Center for Novel Software Technology and Industrialization of Jiangsu Province, Nanjing* 210016)

[3)](*Jiangsu Easymap Geographic Information Technology Corp Ltd., Yangzhou, Jiangsu* 225000)

[4)](*School of Information Technology and Electrical Engineering, The University of Queensland, Brisbane, Australia* 4067)

**Abstract** Efficient processing of Top-$k$ queries is a crucial requirement in many dynamic environments that involve massive amounts of data. Observed in many real scenario, a Top-$k$ query often consists of two components to reflect a user's preference: selection condition and ranking function. Users can set their own sort function, simultaneously can choose to query different interesting subsets of the data. For traditional database, scholars have studied in the Top-$k$ algorithm. However,

收稿日期:2017-09-27;在线出版日期:2018-05-25.本课题得到国家自然科学基金(61728204,61672284,61373015,61300052,41301407)、中国民用航空局安全能力建设基金(AS-SA2015/21)、南京航空航天大学科研基地创新基金(NJ20160028)、青年科技创新基金(NT2018028)、智能电网保护和运行控制国家重点实验室基金(国防预研领域)、江苏高校优势学科建设工程资助项目(No. BK20130819)、澳大利亚研究理事会发现项目(ARC DP140100104)资助.**邓丹苹**,硕士,主要研究方向为时空数据库、情境感知.E-mail:369351945@qq.com.**秦小麟**,博士,教授,博士生导师,中国计算机学会(CCF)高级会员,主要研究领域为空间与时空数据库、分布式数据管理与安全.**李博涵**(通信作者),博士,副教授,中国计算机学会(CCF)教师会员,主要研究方向为空间与时空数据库、地理信息系统等.E-mail:bhli@nuaa.edu.cn.**郑 伟**,硕士,主要研究方向为数据库管理、推荐系统.**刘 亮**,博士,讲师,主要研究方向为数据安全管理、云数据管理和无线传感器网络.**李 雪**,博士,教授,主要研究领域为社交媒体舆论分析大数据分析、序列知识发现、分布式高速时变数据流挖掘.

previous works do not adapt to the situation when object attribute values is dynamic or the data amount is great. To address the problem of dynamic Top-$k$ query, we propose a Trunk Tree Index (TTI) structure based on grid index. To remedy the lack ability of dealing these data, we describe a novel grid index method and give the GID-Top-$k$ algorithm. The TTI index is determined according to the dominating relation between grids, and the dominating relation is further divided into father and son relation, brother relation and so on. TTI index can acquire the position of the grid subscript when there exists insert, delete and update. Furthermore, the process of pruning in Grid Index based Dynamic Top-$k$ can effectively computing the $k$ dominated capability through the summary of the grid in the Trunk tree Index, and minimizing the number of $k$ dominated capability to be counted by looking for determinant units. Divide the influence area and free area according to the division of the region. Finally, pruning the data set, and if the data changes in the free area, there is no need to recalculate Top-$k$ result. It can reduce the probability of recalculation when data changes frequently. The building blocks and calculation modules of the multi-dimensional TTI index are given, the grid dominance relationships, the relationship between brothers and sons are clarified from the two-dimensional to the multi-dimensional case with TTI. The pruning rules and calculation of free area and influence are related with the operation of searching grid, which avoid the stochastic search in grid using linked list. The cost of building and maintaining of the index is also analyzed with query efficiency. The experiment chooses different datasets to test, which compared with the baseline algorithm Top-$k$, RankCube and CIA algorithm separately. Because grid index is different from general index, the n dimensions on query space is divided into such as large area of the grid, under the condition of the same configuration parameters in other experiments, the distribution of data in each grid density on the algorithm performance is going to have an impaction of the presented algorithms. Experimental results verify the effectiveness of the proposed algorithm. Experimental results verify the effectiveness of the algorithm, the experimental data show that in the static case, the query efficiency of the algorithm in this paper up to eight times faster than the traditional Top-$k$ algorithm, dynamic cases up to 10 times faster than the existing comparison objects. We will develop the improved Top-$k$ queries to be an approach for big data analysis, and multi-criteria retrieval applications.

**Keywords** Top-$k$ Query; grid index; partition; summary; dynamic

## 1 引 言

Top-$k$ 查询通过用户的偏好返回 $k$ 个最佳纪录,用户偏好由选择条件和排序函数组成. 例如 top-$k$ 查询的一个应用场景如下,在餐馆查询数据库系统当中,每个餐厅记录的属性有口味、环境、服务、人均消费水平、距离等等,用户可以通过在线查询来挑选合适的餐厅. 每个用户的需求是不同的,也就是每个用户用来筛选 $k$ 个结果的偏好函数不同,比如 $f=0.1\times$环境$+0.6\times$口味$+0.3\times$距离,用户可以根据自己的喜好找到最满足要求的 $k$ 个结果.

在上例中,偏好函数的类型是线性的,实际应用中还存在二次、指数、对数正态等形式[1],不同的用户不仅对偏好函数中的参数有着不同规定,关注的属性维度也不一样. 大众点评"餐饮风向标"显示,截至 2012 年 10 月 31 日,大众点评收录的上海现存的餐厅数量总计达 60 113 家,5 年间扩容幅度超过 3 倍. 而北京现存的餐厅数量总计 55 288 家,是 10 年前的 2.5 倍[2],2014 年达到 91 445 家,季度增长率达到 3.3%,该数据目前还在迅速增长. 在类似大众点评、美团等线上查询网站中,随着新的用户评价加入,每个餐厅的各项属性值都会发生变化. 在以上应用场景中都存在一个亟待解决的问题:即如何在动态变化的大量数据集中进行有效的 top-$k$ 查询. 传统的 top-$k$ 算法通过对所有记录进行遍历和全排序从而返回得分最高的 $k$ 个,这种情况下无论 $k$ 取多少都不会影响算法的运行时间. 如果在数据集合较大时,

用户只要求选出 top10 的餐厅，全遍历及全排序需要花费很大的代价并且没有必要[3]. 排序列表类型的方法的使用前提是数据集可放入内存，并且一旦数据发生动态变化，需要重新建立列表，并不适应大量数据动态变化时的情况[4]. 例如，层次模型采用了数据预处理的方法，提高了在线查找的速度，但在动态变化的情况下，维护层次模型的代价较大[3]. 再如，视图方法在数据动态变化时需要重新计算其对应的实例视图，同样不适用于动态变化的数据集，并且该类算法效率对权重系数的变化较敏感[5]. 此外，基于概要的方法在离线时把数据区域分成若干个等大网格，并根据参数 $k$ 的大小以网格为最小单位划分影响区和自由区，当增删的记录位于影响区时可能会引起 top-$k$ 集合的变化，而自由区中的记录变化从理论上保证对 top-$k$ 集合无影响[6]. 该方法可满足不同用户的偏好，提高了在线查询速度，可较好适应数据集的动态变化. 但是传统的基于概要的方法并未对网格建立高效索引，当网格数量增加时，查找定位网格速度较慢，并且划分区域时不能利用网格在索引中的位置提高计算效率，继而在数据动态变化导致频繁更新区域时花费代价较大.

针对以上问题，本文提出 GID-Top-$k$(Grid Index based Dynamic Top-$k$ Computation Algorithm)算法，在已有算法的基础上，为网格建立 TTI(Trunk Tree Index)树形索引. TTI 索引在数据进行插入，删除等操作时可依据网格下标对其进行定位，提高了建立和维护索引结构的效率. TTI 索引可根据网格在索引中的位置和网格的概要信息进行裁剪，通过判断网格是否具有“$k$ 支配能力”以及确定索引中的“剪枝单元”来划分自由区和影响区，提高了剪枝效率. 其中索引加载模块负责将初始数据插入 TTI 索引. 算法的计算模块负责将网格分区并计算初始 top-$k$ 结果，并当数据变化发生在影响区时，负责重新划分区域及计算.

本文的主要贡献归纳为如下几点：

(1) 在数据划分成网格的思想上提出了 TTI 树形索引，并利用网格下标提出了快速建立 TTI 树形索引的算法.

(2) 在 TTI 属性索引上提出了快速划分影响区及自由区的算法，利用已遍历网格的位置及概要信息用于裁剪未遍历的网格，提高了划分区域的效率.

(3) 利用真实数据和人工数据进行全面测试，检验了所提出的 GID-Top-$k$ 算法的有效性.

本文第 2 节介绍相关工作及现有研究中存在的不足；第 3 节为研究的问题过程中所需的基本知识；第 4 节介绍 GID-Top-$k$ 算法在二维空间中的应用；第 5 节将 GID-Top-$k$ 算法扩展到多维空间，并分析其代价；第 6 节进行实验验证，并与已有算法在不同条件下进行对比和分析；第 7 节进行总结与展望.

## 2　相关工作

排序列表类型的代表性方法有 TA[4]，BPA[7]，NRA[8]，TKEP[9]，TMS[10]. TA[4] 算法首先对每个维度上的所有记录进行排序并生成多张排序列表. 利用排序列表计算出当前阈值，并用当前阈值找出所有大于阈值的记录. 算法执行过程中，顺序扫描每个排序好的列表，直至遇到记录指示器，随机访问其他列表并计算偏好函数值. 已扫描的记录通过排序获得 top-$k$ 查询结果. 该方法缺点是返回结果数量不等于 $k$，返回结果数量取决于阈值，阈值太大导致返回结果过少，阈值太小则导致返回结果过多. BPA[7] 利用最佳位置策略可使算法收敛的更快，并且降低执行代价，一定程度上提高了效率. NRA[8] 算法处理 top-$k$ 查询时只支持顺序读，执行过程分为增长阶段和收缩阶段，增长阶段不断累积候选元组直至得到阈值，收缩阶段通过剪切元组得到 top-$k$ 结果. TKEP[9] 在 NRA 算法的基础上加入了早剪切操作，可减少在海量数据集上 NRA 算法增长阶段需要维护的元组数量. TMS 提出了基于排序列表的多维选择问题，利用层次化结构的选择属性网格对原数据表执行水平划分，并根据其属性值进行降序排列，并提出了裁剪操作，来删除不满足条件的记录，提高了算法性能.

层次方法有 Onion[3]，DG[11]，AppRI[12]. 层次方法利用了得分最高的记录一定可在数据集的凸包中被找到的性质(对任何线性偏好函数适用)，该方法用外层包围内层的几何形式计算并储存凸包. 当用户发起线性 top-$k$ 查询时，从最外层的凸包向内层进行，任意线性偏好函数都可以在前 $k$ 层中找到查询结果. 该方法的缺点是在数据集动态变化时，需要花费很大的代价来维护层次模型.

视图方法主要有 Prefer[5]，LPTA[13]. 视图方法选择任意偏好函数预先计算一组排序的列表，并且对每个偏好函数，计算出对应的实例视图. 对于任意给定的偏好函数 $f$，在预先存储好的偏好函数中找到与 $f$ 最相似的一个，并找到它对应的实例视图，通过遍历该视图中的记录子集，可高效返回 $k$ 个结

果. Prefer 算法可以应用在非线性偏好函数中,前提是需要维护不同类型偏好函数对应的视图集. 这类算法在用户偏好函数与预处理函数越相似时处理的速度越快. 这类方法的缺点是对权重系数变化十分敏感,并且同样不适用于动态变化的数据集.

基于概要的方法一般使用网格对数据集进行划分,将每个网格顺序存储于列表当中并记录网格中数据点. 在查询时通过网格的概要信息计算其中数据点的近似函数分值,以裁剪不属于结果集合的数据点,在满足条件的网格单元中,通过进一步访问数据点计算偏好函数值并排序,最后得到 $k$ 个查询结果. 文献[14]证明了在内存中 Q-index 网格索引比 R 树更加高效. RankCube 算法[9]在历史数据集中采用了概要方法进行多维选择查询,该方法构建网格较快但计算过程较粗糙,适用于需要快速建立索引的数据. TMA 和 SMA 算法[6]首次提出了自由区和影响区的概念,并利用队列和网格中记录偏好函数的最高值设置阈值计算 top-$k$ 结果. 该算法的缺点是需要得到网格中偏好函数值最高的记录才能确定是否裁剪该网格,索引结构的维护代价较大. GDG 算法[15]通过求出所有网格的支配网格集和逆支配网格集对网格进行裁剪. 文献[16]提出了一种不规则网格索引,可以适应数据分布不均匀的情况,但是网格尺寸的动态变化也会增加维护代价. CIA 算法[17]在建立网格的基础上通过模糊的方法来提高查找效率,缺点是在 $k$ 值较小时返回结果数量与 $k$ 值相差很多,返回结果个数不精确.

文献[18]基于 BMW 算法基础上提出了采用两层索引的 BMW-$t$ 和 BMW-CS 算法,BMW-CS 算法速度较快但是可能会丢失之前查询的 top-$k$ 结果集. 文献[19]在 BMW-CS 算法的基础上做进一步改进提出 BMW-CSP 算法,在保证结果集不丢失的前提了提高了效率,但是需要消耗更多内存资源. 文献[20]提出了支持室内障碍空间的 top-$k$ 算法,利用 R 树和支配关系的思想裁剪集合减小计算代价,并且在目标对象发生变化时,利用动态变化集合与查询结果集合的关系判断不需要重新计算的 top-$k$ 查询,提高了算法效率. 反向 top-$k$ 查询和 top-$k$ 查询有着密切的关系. RTA 算法[21]是一种有效解决反向 top-$k$ 查询算法,但是 RTA 算法需要访问所有的用户并根据每个用户的权值单独执行一次 top-$k$ 操作. BBR 算法[22]不需要访问每个用户,减少了遍历代价,但是在维度增加时,算法性能有所下降. 文献[23]在反向 top-$k$ 查询的基础上提出了反向空间偏好 top-$k$ 查询,该查询的查询结果与反向 top-$k$ 查询结果相同. 不同之处在于对象的属性不是直接给出,而是由对象与周边设施的空间位置关系计算得到. 实际上,关于 top-$k$ 查询的研究已经从传统关系型数据库转换到分布式平台,支持代价较大的高维向量并行连接操作[24-25]. 而对于 top-$k$ 查询的一些变体,如支持动态属性的向量的 top-$k$ 支配查询将在基于距离的度量空间中基础上衍生出增强版 top-$k$ 查询①. 目前这些静态和动态的 top-$k$ 查询在大数据分析中都有着频繁的应用.

从以上分析可以看出,无论是静态或动态环境,采用索引或非索引的数据,它们对 top-$k$ 查询的裁剪效率都会受到大量目标对象的属性值变化频率的影响,在动态情况更甚. 而现有的基于网格索引的研究方法中,虽然都利用了网格概要信息进行裁剪,但是存储网格一般使用的链表形式,在访问网格时使用随机查找的方式. 在建立网格时,需要获得网格中的最优记录,最劣记录和记录数量等概要信息,或者需要计算网格所有支配节点以及逆支配节点,以确定该网格属于自由区还是影响区,在查询结构的建立以及维护方面都需花费较大的代价. 本文使用 TTI 索引存储网格,可根据网格下标快速访问网格,并利用网格的位置就和相关网格的记录数量进行裁剪,减少了网格中概要信息的数量,提高了建立及维护算法的效率.

## 3 问题描述及定义

### 3.1 基于概要的一般 top-$k$ 查询

大多数基于概要的 top-$k$ 查询借助索引解决 top-$k$ 问题. 索引把数据空间划分成等大的网格,并通过网格的概要信息确定网格之间的关系. 利用网格间的关系实现裁剪操作,得到 $k$ 个结果. 进而,把整个数据空间划分成自由区和影响区,当变化的数据发生在自由区时不影响结果集.

例如,用户关注的餐厅属性分别是价格和口味. 价格和口味是相对静态的属性,假设其在查询过程中不变,首先根据价格和口味两个维度上的属性值将餐厅记录划分入网格. 然后,以网格为最小单位将餐厅记录集合划分成自由区和影响区,并将自由区中的网格记录裁剪,在影响区中计算 $k$ 个结果,返回

① http://pdfs.semanticscholar.org/ce66/1789ac6793e884e162f8476753236e27616c.pdf

结果，概要查询结束. 当某些用户关注的属性数量较多，这时二维概要查询已经不能满足要求，用户不仅关心餐厅的价格和口味属性，还关心距离和排队时间属性. 然而，距离是和用户自身的位置有关的，会随着用户的移动发生变化，排队时间也会随着排队人数的增减而动态改变，从而导致距离和排队时间两个属性在用户发起查询的时刻和用户得到查询结果时刻的改变，这种改变可能会影响到返回的结果集. 概要查询可通过判断影响区中是否存在属性值发生变化的记录确定结果集合是否需要重计算. 可节约计算资源，实现算法的动态性调整. 在给出问题的形式化定义之前，表 1 给出本文符号的注释表.

**表 1　注释表**

| 符号 | 名称 |
|---|---|
| $p$ | 目标对象 |
| $p.id$ | 目标对象标识符 |
| $p.x_i$ | 属性分值 |
| $\varphi_i$ | 权重系数 |
| $Score$ | 得分函数 |
| $dominate$ | 支配关系 |
| $n$ | 属性维度 |
| $\delta$ | 网格单位 |
| $Cell$ | 网格单元集合 |
| $C_{i,j}$ | 网格单元 |
| $q$ | 查询请求 |
| $RN(C_{i,j})$ | $C_{i,j}$ 中记录数量 |
| $Area$ | 区域标识符 |
| $Plink$ | 记录头结点 |
| $maxlstep$ | 最大向左步数 |
| $maxrstep$ | 最大向右步数 |
| $calculate\_kRN$ | $k$ 支配能力函数 |

在概要 top-$k$ 查询中，首先将用户 $u$ 所要查找的记录集合放入数据空间中，记录在 $n$ 维空间中的位置由它的 $n$ 个属性值决定. 在二维空间中，得到数据集在两个维度上的最大值 $max_{x1}$ 和 $max_{x2}$，对数据集中任意记录 $p$，设其在两个维度上的原始属性值为 $p.o_1, p.o_2$. 将记录 $p$ 的属性值范围变换到 0～1 的值之间，令 $p.x_1=p.o_1/max_{x_1}$，$p.x_2=p.o_2/max_{x_2}$. 可以得到其变换后的属性值 $p.x_1, p.x_2$. 记录 $p$ 表示为〈$p.id, p.x_1, p.x_2$〉，其中，$p.id$ 表示记录 $p$ 的唯一标识符，$p.x_1, p.x_2$ 是记录 $p$ 在属性 $x_1$ 和 $x_2$ 上的值，$p.x_1, p.x_2 \in [0,1]$. 记录 $p$ 的表示类似文献[26-29]中处理多维数据流的方法. 然后，使用网格对这些有效数据进行划分：在 $1\times1$ 的二维空间中，网格在每个属性维度上的长度为 $\delta$. 所以，位于二维空间的 $i$ 行 $j$ 列的网格 $C_{i,j}$ 包含所有属性 $x_1$ 属于 $[i\times\delta, (i+1)\times\delta]$ 并且属性 $x_2$ 属于 $[j\times\delta, (j+1)\times\delta]$ 的记录. 相反的，已知记录 $p$，它的属性值为 $(p.x_1, p.x_2)$ 它所在的网格编号 $C_{i,j}$ 可以通过 $i=\lfloor p.x_1/\delta \rfloor$，$j=\lfloor p.x_2/\delta \rfloor$ 计算得到. 所有二维空间上的网格组成网格单元集 $Cell$. 定义 1 和 2 给出概要 top-$k$ 查询中数据记录间关系的定义. 定义 3 网格间关系的定义，不失一般性，选择二维空间数据（即记录拥有两个属性，其取值范围在 0 到 1 之间）为例说明.

**定义 1**（单调偏好函数[6]）. 单调递增偏好函数 $f$，在任意维度 $x_i$ 上，记录 $p_1, p_2$ 满足 $p_1.x_i \geq p_2.x_i$，并且存在维度 $x_j$，使得 $p_1.x_j > p_2.x_j$. 则 $score(p_1)=f(p_1.x_1, p_1.x_2, \cdots, p_1.x_n) > f(p_2.x_1, p_2.x_2, \cdots, p_2.x_n)=score(p_2)$. 对应的，易知单调递减偏好函数满足 $score(p_1)=f(p_1.x_1, p_1.x_2, \cdots, p_1.x_n) < f(p_2.x_1, p_2.x_2, \cdots, p_2.x_n)=score(p_2)$.

**定义 2**（支配关系[30]）. 假设两个记录 $p_1, p_2$ 的 $n$ 个属性分别是 $(p_1.x_1, p_1.x_2, \cdots, p_1.x_n)$，$(p_2.x_1, p_2.x_2, \cdots, p_2.x_n)$，若 $\forall i, p_1.x_i \geq p_2.x_i$ 成立，且 $\exists j$，使得 $p_1.x_j > p_2.x_j$ 则称 $p_1$ 支配 $p_2$. 记做 $dominate(p_1, p_2)$. 显然，支配关系具有传递性，若 $dominate(p_1, p_2)$ 并且 $dominate(p_2, p_3)$，则 $dominate(p_1, p_3)$.

**定义 3**（网格间的支配关系）. 在二维空间上，设 $C_{a1,a2}, C_{b1,b2}$ 属于网格集合 $Cell$，对 $\forall p_a \in C_{a1,a2}$，$\forall p_b \in C_{b1,b2}$，满足 $p_a$ 支配 $p_b$，则，$C_{a1,a2}$ 支配 $C_{b1,b2}$. 记做 $dominate(C_{a1,a2}, C_{b1,b2})$. 显然，$C_{a1,a2}$ 支配 $C_{b1,b2}$ 等价于 $a1>b1 \cap a2>b2$. 并且，网格间的支配关系同样具有传递性.

在二维空间中，用户发起查询 $q$，需要查找偏好函数值排名第 $k$ 的记录，设该记录为 $p_k$. 偏好函数为 $score(x_1, x_2)=\psi_1\times x_1+\psi_2\times x_2$（$\psi_i$ 为 $x_i$ 属性维度的权重系数）. 如图 1 所示，直线 $score(p_k.x_1, p_k.x_2)=\psi_1\times p_k.x_1+\psi_2\times p_k.x_2$ 将数据空间分成两个部分：影响区表示为 $\{(x_1, x_2) \mid \psi_1\times p_k.x_1+\psi_2\times p_k.x_2 \leq \psi_1\times x_1+\psi_2\times x_2, x_1\leq 1 \cap x_2 \leq 1\}$；自由区表示为 $\{(x_1, x_2) \mid \psi_1\times p_k.x_1+\psi_2\times p_k.x_2 \geq \psi_1\times x_1+\psi_2\times x_2, 0\leq x_1 \cap 0 \leq x_2\}$. 在数据集更新时，若有数据被移入或

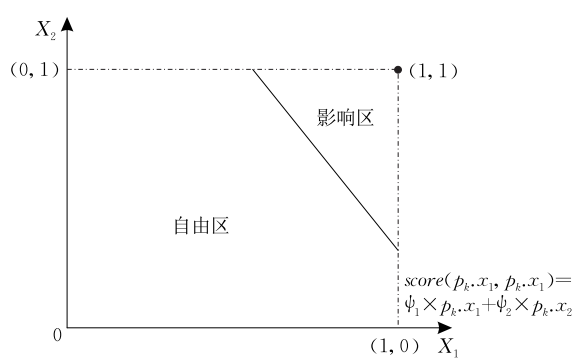



图 1　自由区和影响区示意图

移出影响区,则会对查询 $q$ 的结果集合产生影响,这时需要重新计算 $k$ 个结果并重新划分影响区和自由区.如果自由区的数据发生删除或添加等改动,不会影响查询 $q$ 的结果集.文献[20]中的 SPD-Top-$k$ 算法采用类似的方法对动态变化的数据与结果集进行比较,之后再二次分类,可以提高算法效率,避免在每一次数据变化时都重新进行 top-$k$ 排序.

这种概要方法的优点是对自由区和影响区精确划分,影响区内只有 $k$ 个数据,可以保证影响区面积最小化,这样改动数据留在影响区的概率也就随之减小.但是,这种方法需要对数据集进行全排序并计算第 $k$ 个最优数据,并且一旦改动数据落在影响区,则需再次对数据进行全排序,初始查询代价和维护代价都很大.使用网格索引划分数据空间,虽然弱化了影响区的精度,使影响区内的数据大于 $k$ 个,但是网格索引利用了网格间的支配关系可以快速确定影响区,提高维护效率.对数据空间进行网格划分后如图 2 所示.

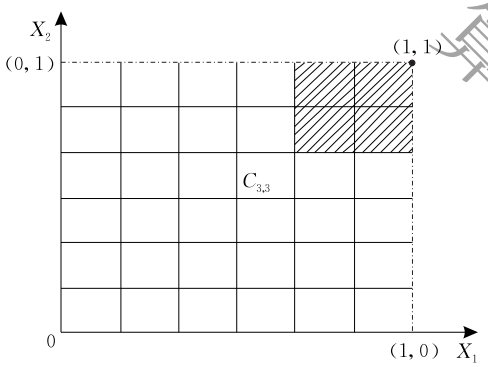



图 2 网格单元关系

任何单调偏好函数 $f$,得分最高的点都落在靠近点(1,1)角落的数据空间.在 Skyline 查询中,数据点(1,1)支配数据集中所有其他点.根据定义 2,所有数据点都被其右上方的数据点所支配.根据定义 1,2,若 $dominate(p_1,p_2)$,则 $score(p_1)>score(p_2)$.性质可推广到网格,若 $dominate(C_{a1,a2},C_{b1,b2})$,则对 $\forall p_a\in C_{a1,a2},\forall p_b\in C_{b1,b2}$,满足 $score(p_a)>score(p_b)$.对任意网格 $C_{i,j}$,若所有支配 $C_{i,j}$ 的网格中记录数量之和大于等于 $k$,则 $C_{i,j}$ 中的点不可能出现在 top-$k$ 的结果集当中,可将 $C_{i,j}$ 归入自由区.例如,图 2 中 $C_{3,3}$,阴影区域为支配其的网格 $C_{4,4},C_{5,5},C_{4,5},C_{5,4}$,若 $RN(C_{4,4})+RN(C_{5,5})+RN(C_{4,5})+RN(C_{5,4})\geq k$,则 $C_{3,3}$ 不包含 top-$k$ 结果,将 $C_{3,3}$ 划分为自由区,又由于支配关系具有传递性,所有被 $C_{3,3}$ 支配的网格也划分为自由区.

利用上述方法,只需遍历数据空间中网格直到所有网格都被划分到自由区和影响区为止.需要注意的是这样操作所需遍历的网格数较多,并且由于传统算法将网格存储在链表当中,定位某个网格时需要遍历所有链表上的网格节点,计算和查找所花费代价较大,在记录频繁更新时,维护效率很低.基于此,本文提出的 TTI 树形索引在划分网格的方法上进一步组织数据,可减小定位网格和计算网格所花费的时间和需要遍历的网格数量,以提高查询和维护的效率.

### 3.2 TTI 索引结构

以二维空间为例介绍 TTI 属性索引,在第 5 节中进一步介绍多维空间中的 TTI 索引.基于网格分区的 TTI 索引,整体结构如图 3 所示.

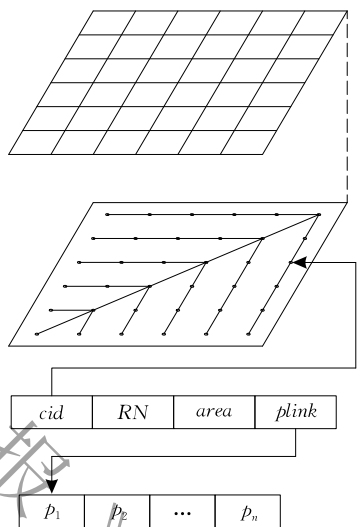



图 3 TTI 索引结构示意图

在 TTI 索引可维护网格间的支配及层次关系,每个节点表示一个网格,网格在索引中的位置由其下标 $i,j$ 决定,根节点为 $O$,节点中包括网格的唯一标识 $cid$,网格中记录数量 $RN(cid)$,$area$ 为区域标识,有两种情况:网格属于自由区 $area$ 的值为 0,属于影响区 $area$ 的值为 1.指向网格内数据的指针 $plink$,网格内的数据用链表方式存储.数据点 $p$ 包括其唯一标识 $pid$,其属性值 $p.x_1,p.x_2$.

由于 TTI 索引是根据网格间支配关系确定的,需要对支配关系进一步划分.两个网格 $C_{a1,a2},C_{b1,b2}$,若存在 $dominate(C_{a1,a2},C_{b1,b2})$,则 $C_{b1,b2}$ 为 $C_{a1,a2}$ 的子孙.网格中除了父子关系以外,还存在兄弟关系,若 $a1=b1\cap a2>b2$ 或者 $a1>b1\cap a2=b2$,则称 $C_{a1,a2}$ 为 $C_{b1,b2}$ 的胞兄,$C_{b1,b2}$ 为 $C_{a1,a2}$ 的胞弟,在图 4 中,$C_{3,3}$ 的胞弟有 $C_{3,2},C_{3,1},C_{3,0},C_{2,3},C_{1,3},C_{0,3}$;$C_{3,3}$

的胞兄有 $C_{3,4}$，$C_{3,5}$，$C_{4,3}$，$C_{5,3}$.

**定理 1.**　若 $C_{a1,a2}$ 为 $C_{b1,b2}$ 的胞兄，兄弟网格 $C_{b1,b2}$ 和 $C_{a1,a2}$ 相互不支配，但对任意网格 $C$，若存在 $dominate(C_{b1,b2},C_{i,j})$ 则一定存在 $dominate(C_{a1,a2},C_{i,j})$.

证明.　因为 $dominate(C_{b1,b2},C_{i,j})$，则 $b1>i$，$b2>j$，又因为 $a1=b1\cap a2>b2$ 或者 $a1>b1\cap a2=b2$，所以 $a1>i\cap a2>j$. 所以存在 $dominate(C_{a1,a2},C_{i,j})$.　　证毕.

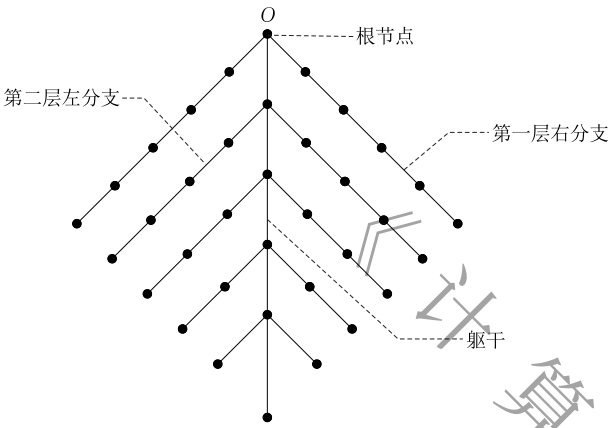



图 4　二维 TTI 树形结构示意图

如图 4 所示，TTI 索引由根节点为 $O$，躯干，左分支，右分支组成. 其中只有躯干上的节点有左中右孩子，称作躯干节点. 其它节点均只有一个孩子，称为左分支节点和右分支节点. 每个节点表示一个网格. 躯干节点与其连接的左右分支组成一层次，从根节点开始从高层次向低层次递减.

**定义 4**(TTI 索引).　若存在躯干节点为网格 $C_{i,j}$，则其左孩子为网格 $C_{i-1,j}$，其右孩子为 $C_{i,j-1}$，中孩子为 $C_{i-1,j-1}$. 若存在左分支节点 $C_{i,j}$，则其孩子为 $C_{i-1,j}$. 若存在右分支节点 $C_{i,j}$，则其孩子为 $C_{i,j-1}$.

从以上定义可以发现，每个躯干节点的中孩子为其子孙，左右孩子为其胞弟. 其中左分支节点的孩子为其左胞弟，右分支节点的孩子为其右胞弟. TTI 索引的左右分支由兄弟关系组成，躯干由父子关系组成. 事先规定两个具有兄弟关系并且位于同一层次同侧分支的网格之间的表示方式，EB 表示胞兄，YB 表示胞弟. 比如，$C_{3,3}$ 和 $C_{3,0}$ 之间的距离为 3，则 $C_{3,0}$ 为 $C_{3,3}$ 距离为 3 的右胞弟，记作 $C_{3,0}=YB(C_{3,3},3,right)$，$C_{3,3}$ 为 $C_{3,0}$ 距离为 3 的胞兄，记作 $C_{3,3}=EB(C_{3,0},3)$. 除躯干节点外，其他节点的兄弟没有左右之分. 层次与层次之间靠躯干节点的父子关系连接. 比如 $C_{3,3}=S(C_{5,5},2)$，表示 $C_{3,3}$ 为 $C_{5,5}$ 距离为 2 的子辈；$C_{5,5}=F(C_{3,3},2)$，表示 $C_{5,5}$ 为 $C_{3,3}$ 距离为 2 的父辈. 如图 4 所示，根节点在顶层的 6 层 TTI 结构.

利用 TTI 可实现网格的快速建立和访问，从根节点开始，遍历其左右分支，从顶层向下，并在访问过程中快速判定网格划分影响区和自由区，从而实现快速返回 top-$k$ 结果.

## 4　基于网格分区的二维 GID-Top-$k$ 算法

### 4.1　索引加载模块

建立索引模块首先需要将数据空间进行网格划分，网格在每个属性维度上的长度为 $\delta$，根据网格下标建立 TTI 树形索引，并初始化 $RN(cid)=0$，$plink\rightarrow$ null，$area=0$，记录根节点的下标 $O.i$，$O.j$，给出一记录 $p$，在 $1\times1$ 的二维空间中，它的属性值为 $(p.x_1,p.x_2)$ 它所在的网格编号 $C_{i,j}$ 可以通过 $p.i=[p.x_1/\delta]$，$p.j=[p.x_2/\delta]$ 计算得到.

算法思路：通过改变 $p.i$ 和 $p.j$ 的值，使 $p.i$ 和 $p.j$ 的值向 $O.i$，$O.j$ 靠近，当 $p.i=O.i\cap p.j=O.j$ 时，说明找到 $p$ 所在网格. 从根节点开始向下遍历，每向下一层 $p.i$ 和 $p.j$ 加 1，当 $p.i=O.i$ 或者 $p.j=O.j$ 时，说明已经找到 $p$ 所在的层(第 3～7 行)，此时考虑 3 种情况：

(1) $p.i=O.i\cap p.j=O.j$，$p$ 为当前层次的躯干节点，初始化网格(第 15 行)，算法结束；

(2) $p.i=O.i\cap p.j\neq O.j$，说明 $p$ 位于当前层次的右分支，$O.j-p.j$ 的值为指针向右走的步数(第 8～10 行)；

(3) $p.i\neq O.i\cap p.j=O.j$，说明 $p$ 位于当前层次的左分支，$O.i-p.i$ 的值为指针向左走的步数(第 11～14 行).

遍历结束，将 $p$ 插入对应的网格中，并将对应网格中的 $RN(C_{i,j})$ 值加 1，将 $area$ 的值初始化为 0(第 15～16 行). 对数据集 $P$ 中的数据进行一一插入，可得到每个网格中 $RN(cid)$ 的值和 $plink$ 指向的数据. 在 TTI 索引中，插入数据点时不需要遍历所有网格. 以下是将一个数据插入网格的算法. 循环该算法直到网格数据集 $Cell$ 为空.

**算法 1.**　初始化网格 $C$ 中的 $RN(C)$，$plink$ 的值，$area$ 值

输入：数据点 $p$，根节点 $O$，网格集合 $Cell$

输出：数据点 $p$ 所在的网格 $C$ 以及网格 $C$ 中的 $RN(C)$，$plink$

```
1.  q→O
    /* 初始化查找指针,指向根节点 o */
2.  p.i=[p.x1/δ], p.j=[p.x2/δ]
    /* 通过 p 的属性值判断 p 所属网格 */
3.  while (p.i≠O.i && p.j≠O.j)
4.   q=q→midchild, p.i++, p.j++
     /* 用指针 q 找到 p 所在层次的躯干节点 */
5.   if (p.i≠O.i && p.j≠O.j)
      /* 如果没有找到 p 所在层次的躯干节点 */
6.    continue
7.   end if
8.   while (p.j≠O.j)
      /* 如果网格在右分支,q 向右孩子移动 */
9.    q=q→rightchild, p.j++
10.  end while
11.  while (p.i≠O.i)
      /* 如果网格在左分支,q 向左孩子移动 */
12.   q=q→leftchild, p.i++
13.  end while
14. end while
15. RN(q→cell)++, p→cell.plink, area=0
16. return q→cell
17. end
```

### 4.2 计算模块及更新维护

初始计算模块需先将每个网格划入自由区和影响区,这样 top-$k$ 查询可将自由区中的记录裁剪,并可根据记录发生动态变化的区域判断是否重新进行查询.最简单的划分区域方法就是遍历索引中每个网格,判断所有支配其的网格中记录数量之和是否大于等于 $k$,若大于等于 $k$,则当前网格中的记录不可能存在 top-$k$ 结果集中,可划入自由区,$area$ 值不变,默认为 0;若小于 $k$,则需进一步判断网格中记录是否属于 top-$k$ 结果集,划入影响区,$area$ 值更改为 1.但是该方法需要计算索引中所有网格,效率较低.在划分区域的过程中通过计算网格的"$k$ 支配能力"和每层索引的"剪枝单元"来裁剪一些不可能属于影响区的网格,减少需要遍历的网格数量,提高算法效率.

**定义 5**($k$ 支配能力). 存在网格 $C$,若 $C$ 和 $C$ 胞兄节点上的网格和所有支配 $C$ 的网格中记录数量之和大于等于 $k$,则称网格 $C$ 是具有 $k$ 支配能力的网格.

**定理 2.** 在 TTI 索引中,存在网格 $C$,$C$ 是具有 $k$ 支配能力的网格,则所有被 $C$ 所支配的网格中不存在 top-$k$ 的结果.

证明. 设 $C$ 是具有 $k$ 支配能力的网格,任意一个元素 $C_{i,j}$,满足 domain($C$,$C_{i,j}$),根据定理 1,$C$ 胞兄节点上的网格同样支配 $C_{i,j}$,又根据支配关系的传递性,支配 $C$ 的网格,同样支配 $C_{i,j}$.$C$ 和 $C$ 胞兄节点上的网格和所有支配 $C$ 的网格中记录数量之和大于 $k$,所以 $C_{i,j}$ 中不可能存在 top-$k$ 结果.

证毕.

根据定义 5 和定理 2,利用 *calculate_kRN* 函数计算网格 $C$ 和 $C$ 胞兄节点上的网格和所有支配 $C$ 的网格中记录数量之和,可判断被 $C$ 支配的网格属于自由区还是影响区,例如,当指针指向 $C_{5,4}$ 节点时,计算其胞兄为 $C_{5,5}$,$C_{5,6}$,$C_{6,4}$,支配其的网格为 $C_{6,5}$,$C_{6,6}$.若满足 $RN(C_{5,4})+RN(C_{5,5})+RN(C_{5,6})+RN(C_{6,4})+RN(C_{6,5})+RN(C_{6,6})\geq k$,则 $C_{5,4}$ 支配的网格可划入自由区.在 TTI 索引中,还可以利用网格在索引的位置做裁剪,使得在划分区域的同时对尽量少的网格调用 *calculate_kRN* 函数.

**定理 3.** 在 TTI 索引中,若胞兄节点是具有 $k$ 支配能力的网格,则胞弟节点一定也是具有 $k$ 支配能力的网格.

证明. 设 $C_{a1,a2}$ 为 $C_{b1,b2}$ 的胞兄,则 $a1=b1\cap a2>b2$ 或者 $a1>b1\cap a2=b2$,设网格集合 $U_{a1}$ 和 $U_{a2}$ 为支配 $C_{a1,a2}$ 的网格集合和 $C_{a1,a2}$ 的胞兄网格集合,网格集合 $U_{b1}$ 和 $U_{b2}$ 为支配 $C_{b1,b2}$ 的网格集合和 $C_{b1,b2}$ 的胞兄网格集合,由 $C_{a1,a2}$ 具有 $k$ 支配能力得到 $RN(U_{a1})+RN(U_{a2})+RN(C_{a1,a2})\geq k$.设 $C_{c1,c2}$ 为 $U_{a2}$ 中的任意一个网格,则 $(a1=c1\cap a2<c2)\cup(c1>a1\cap a2=c2)$,四种情况排列组合得到 $C_{c1,c2}$ 为 $C_{b1,b2}$ 的胞兄或者 $C_{c1,c2}$ 支配 $C_{b1,b2}$.根据定理 1,集合 $U_{a1}$ 的任意网格 $C_{d1,d2}$,一定有 $dominate(C_{d1,d2},C_{b1,b2})$,所以 $RN(U_{a1})+RN(U_{a2})\geq RN(U_{b1})+RN(U_{b2})$,最后得出 $C_{b1,b2}$ 也具有 $k$ 支配能力.

**定义 6**(剪枝单元). 剪枝单元为距离躯干节点最近并且具有 $k$ 支配能力的网格,二维 TTI 索引每层有两个左右剪枝单元分别位于左右分支.

**定理 4.** 在二维 TTI 索引中,假设当前层次的左右剪枝单元与躯干节点的距离为 $lefttab$,$righttab$,则下一层次左右剪枝单元与躯干节点的距离小于等于 $lefttab-1$,$righttab-1$.

证明. 设躯干节点中的网格 $C_{a,a}$,它的儿子节点 $C_{a-1,a-1}=S(C_{a,a},1)$,是右剪枝单元,$YB(C_{a,a},righttab,right)=C_{a,a-righttab}$,设下一层次与躯干节点的距离为 $righttab-1$ 的节点为 $YB(C_{a-1,a-1},righttab-1,right)=C_{a-1,a-righttab}$,又因为 $C_{a-1,a-righttab}$ 为 $C_{a,a-righttab}$ 的胞弟,$C_{a,a-righttab}$ 是具有 $k$ 支配能力

的网格，所以 $C_{a-1,a-righttab}$ 也是具有 $k$ 支配能力的网格. 所以，下层次右剪枝单元与躯干节点的距离一定小于等于 $righttab-1$. 左剪枝单元同理.　　证毕.

根据定理 3、4，只需找出当前层次中的剪枝单元，就可以最大限度的对低层次不属于影响区的节点进行裁剪. 这里需要注意，剪枝单元不可以用于裁剪本层次的节点，只可用于裁剪低层次的节点.

算法思路：根据定理 1，设置左右标记对网格节点进行裁剪，初始化左右分支的标记值 $lefttab$，$righttab$ 为无穷大(第 1 行)，算法通过不断更新缩小左右标记值以减少需要遍历的节点. 从顶层的根节点开始(第 2 行)，指针往左遍历与躯干节点距离小于 $lefttab$ 的网格节点直到找到剪枝单元(第 4～10 行)，用中间变量 $k\_tab$ 记下当前分支剪枝单元与本层躯干节点的距离，若没找到，$k\_tab$ 的值为无穷大(第 5 行)，用上层次的 $lefttab$ 来确定当前分支需要遍历的最大步数，遍历当前分枝结束后，判断是否用变量 $k\_tab$ 更新 $lefttab$(若 $k\_tab>lefttab$，说明没找到剪枝单元，不更新 $lefttab$)(第 8 行). 结束一侧分支后重置 $k\_tab$ 值为无穷大(第 11 行，19 行). 同理遍历右分支(12～18 行). 当前层次的 $lefttab$ 和 $righttab$ 可用于裁剪下一层次的网格节点，每向下一层次，更新标记值，将 $lefttab$ 和 $righttab$ 减 1(第 19～20 行). 当 $lefttab$ 和 $righttab$ 都小于等于 0 时，本层中只剩下一个躯干节点需要遍历，直到 $lefttab$ 和 $righttab$ 都为负数，算法结束.

**算法 2.**　网格分区算法.

输入：网格集合 $Cell$，每个网格中的 $RN(cid)$，根节点 $O$

输出：网格集合 $Cell$，每个网格中 $area$ 值

子函数说明：$calculate\_kRN$ 函数判断输入网格 $C$ 是否具有 $k$ 支配能力

```
1.  q→O, righttab=+∞, lefttab=+∞, k_tab=+∞
    /* 初始化变量指针 q, righttab, lefttab, R 用来判
       断是否更新标记值并暂时存储更新后标记值 */
2.  while (righttab≥0 ‖ lefttab≥0)
    /* 顶层开始向下遍历 */
3.   i=0, j=0, ql=q, qr=q
     /* 初始化左右指针指向当前层次的躯干节点，
        初始化左右分支计数变量 i, j */
4.    while (ql≠null && i≤lefttab)
      /* 每层中先遍历左分支
5.     if k_tab>i && calculate_kRN(ql)>k then
         k_tab=i
       /* 用 k_tab 判断是否找到函数值大于 k 的节
          点并记下其与躯干节点的距离 */
6.     end if
7.     ql.area=1, ql=ql→leftchild, i++
8.     if lefttab>k_tab then lefttab=k_tab
       /* 更新标记值 */
9.     end if
10.  end while
11.  k_tab=+∞
12.  while (qr≠null && j≤righttab)
      /* 同理遍历右分支 */
13.   if k_tab>j && calculate_kRN (qr)>k then
        k_tab=j
14.   end if
15.   qr.area=1, qr=qr→rightchild, j++
16.   if righttab>k_tab then righttab=k_tab
17.   end if
18.  end while
19.  k_tab=+∞
20. q=q→midchild, lefttab--, righttab--
      /* 指向下一层 */
21. end while
22. end
```

若要计算网格 $C$ 的 $k$ 支配能力，则需找到 $C$ 胞兄节点上的网格和所有支配 $C$ 的网格，利用 TTI 索引层次之间的关系，在每个层次中查找 $C$ 胞兄节点上的网格和所有支配 $C$ 的网格，从高层开始查找范围递减. $calculate\_kRN$ 函数的具体实现如算法 3 所示.

算法思路：通过当前节点与根节点位置关系，计算在遍历过程中指针从每层躯干节点出发需要向左和向右走的步数，记为 $maxlstep$，$maxrstep$，顶层的根节点向左需要走过的步数为 $maxlstep=O.i-C.i$，向右需要走过的步数为 $maxrstep=O.j-C.j$(第 1 行)，先遍历左分支(4～6 行)，再遍历右分支(7～9 行)，在这一过程中遍历的节点数为 $maxlstep+1$，$maxrstep+1$(左右分支均包括躯干节点)，每遍历一个节点，获取节点上网格的 $RN$ 值，累加入 $result$ 变量(第 5 行)，在这一过程中，躯干节点的 $RN$ 值被累加了两次，在计算当前层次结束时减去一次当前层次躯干节点上网格 $RN$ 值(第 10 行). 每向下一层，将步数 $maxlstep$ 和 $maxrstep$ 减 1(第 10 行)，直到 $maxlstep$ 或者 $maxrstep$ 的值小于 0，算法结束. 例如，在图 4 中判断网格 $C_{3,4}$ 是否具有 $k$ 支配能力，根节点为 $C_{5,5}$，$maxlstep=5-3=2$，$maxrstep=5-4=1$，在顶层累加左分支中走两步得到 $RN(C_{5,5})$，$RN(C_{4,5})$，$RN(C_{3,5})$ 值，右分支中走一步 $RN(C_{5,5})$，

$RN(C_{5,4})$值，减去一次 $RN(C_{5,5})$；$maxlstep$ 减一后为的值 1，$maxrstep$ 减一后为的值 0，指针向下一层，累加左分支中 $RN(C_{4,4})$，$RN(C_{3,4})$值，累加右分支 $RN(C_{4,4})$，减去一次 $RN(C_{4,4})$；$maxlstep$ 减一后为的值 0，$maxrstep$ 减一后为的值－1. 至此算法结束，返回累加值.

**算法 3.** 计算当前网格是否具有 $k$ 支配能力.

输入：网格集合 $Cell$，要判定的网格 $C_{i,j}$，每个网格的 $RN$ 值，根节点 $O$

输出：$C_{i,j}$ 和 $C_{i,j}$ 的胞兄和所有支配 $C_{i,j}$ 的网格中记录数量之和

```
1.  q→O,O.i−C.i=maxlstep,O.j−C.j=maxrstep,
    result=0
    /* 初始化变量，计算 maxlstep,maxrstep */
2.  while (maxlstep≥0 && maxrstep≥0)
3.    i=0, j=0, ql=q, qr=q
      /* 从顶层开始 */
4.    while (i≤maxlstep)
      /* 先计算左分支 */
5.    result=RN(ql)+result, ql=q→leftchild,i++
6.    end while
7.    while (j≤maxrstep)
      /* 再计算右分支/ */
8.    result=RN(qr)+result, qr=q→rightchild,j++
9.    end while
10.   result=result−RN(q), q=q→midchild,
      maxlstep−−,maxrstep−−
      /* 减去一次躯干节点上的 RN 值，指针指向下
         一层更新 maxlstep 和 maxrstep 的值 */
11.   end while
12. return result
13. end
```

通过计算模块，TTI 索引中初始自由区和影响区已确定，可在影响区中通过快速排序算法选出 $k$ 个最佳记录，得到初始 top-$k$ 结果集.

由于记录的属性值会在在查询的过程中发生变化，从而导致系统计算的 $k$ 个结果集合和实际结果集合产生差异. GID-Top-$k$ 算法支持查询过程中记录属性值的动态变化，并可根据动态变化发生的区域快速返回新的 top-$k$ 结果集. 首先，数据的变化会引起 TTI 索引结构的改变，例如存在一发生变化的记录 $p$，初始属性值为 $p.x_1$，$p.x_2$，变化后的属性值为 $p.x_1'$，$p.x_2'$. 根据属性值 $p.x_1$，$p.x_2$ 可定位记录 $p$ 当前所在的网格，删除该记录，并更改网格的概要信息，将记录数量减一，并记下网格的区域标识符. 然后根据属性值 $p.x_1'$，$p.x_2'$ 定位记录 $p$ 更新后所属的网格，插入该记录，并更改网格的概要信息，将记录数量加一，并记下网格的区域标识符. 若更新前及更新后的网格区域标识符均表示自由区，由于 top-$k$ 结果集合只可能存在于影响区当中，所以，记录 $p$ 的变化不影响 top-$k$ 查询的结果集. 在实际查询过程中，发生变化的可能不止一个数据，而是一个数据集，称其为动态变化集合. 对于动态变化集合，需要对其中的记录逐一进行判定，直到动态变化发生在影响区，则可确定需要重新计算 top-$k$ 结果，此时不再读取区域标识符，只需进行网格概要信息的改变和数据的删除及插入完成索引结构的更新即可，待更新完成后，需要重新划分自由区及影响区. 若在索引结构的更新过程中不涉及影响区网格的变化，则不需要进行结果集合的重计算.

## 5 多维属性处理及代价分析

### 5.1 多维 TTI 索引的加载模块

TTI 索引向多维度进行扩展，多维空间中的网格支配关系，兄弟父子关系，和 $k$ 支配能力的定义类似于二维空间，限于篇幅不再赘述，仅对处理方法和计算代价进行补充说明.

与在二维空间中一样，$n$ 维 TTI 索引，仅有唯一主躯干和主根节点 $O$，主躯干上的网格节点集合为 $\{C_{1,1,\ldots,1},C_{2,2,\ldots,2},\cdots,C_{r,r,\ldots,r}\}$，同样采用分层策略，不同层次的网格节点互不相交，每层存在 $n$ 个主分支，每个主分支相当于一个 $n-1$ 维空间上 TTI 索引的躯干，躯干上每个节点都是其分支上低维 TTI 索引的根节点，例如，$n-1$ 维 TTI 索引上的躯干节点集合为 $n-2$ 维 TTI 索引上根节点. 一个 $n$ 维 $r$ 层的 TTI 索引(不妨设最高层主分支上的节点数也为 $r$)，存在 $n\times r$ 个主分支，$n\times r$ 个 $n-1$ 维 TTI 索引，但注意这些 $n-1$ 维 TTI 索引的层次数并不相同，而是随着 $n$ 维 TTI 索引层数的降低而递减. 例如，在第 $w$ 层中($w$ 的值范围为 $1\sim r$)，$n$ 个 $n-1$ 维 TTI 索引的根节点均为 $C_{w,w,\ldots,w}$. 在第 $w$ 层的第 $m$ 个主分支($m$ 的值范围为 $1\sim n$)，其躯干节点集合为 $\{C_{w,w,\ldots,w},C_{w+1,w+1,\ldots,w,\ldots,w+1},\cdots,C_{r-w+1,r-w+1,\ldots,w,r-w+1}\}$，第 $m$ 维度的值始终为 $w$. 进一步用三维空间上 TTI 索引为例说明，在如图 5 所示的三维空间中，$C_{0,0,0}$ 为主根节点，黑色封闭区域内分别表示第 0 层次上的三个二维空间，黑色网格节点表示主躯干，第一层第一个二维空间躯干，第一层第二个二维空间躯干，第一层第三个二维

空间躯干. $C_{0,0,0}$ 也为第一层三个二维 TTI 索引的根节点. 图 5 所示为该三维 TTI 索引的其中一层的展开图.

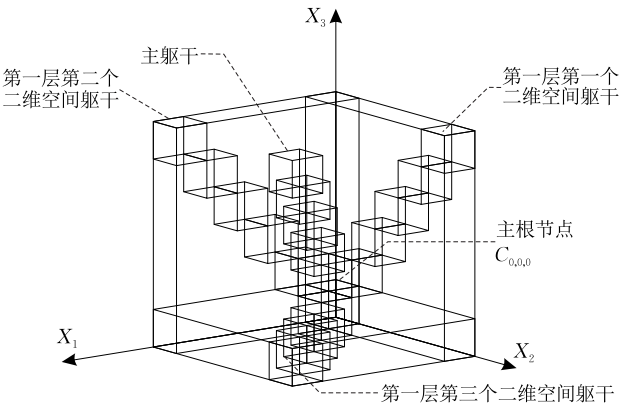



图 5　三维空间部分 TTI 索引立体图

与二维空间一样，多维 TTI 索引的建立过程就是查找对应的网格，将目标对象集合中记录一一插入. 查找原则是：根据网格下标与根节点下标的差来确定遍历方向与步数，并且只遍历躯干节点直至找到最后一个分支. 将记录 $p$ 的下标表示为 $p.x_1, p.x_2, \cdots, p.x_n$，通过 $p.i=[p.x_i/d]$ 得到（$i$ 值得范围为 $1\sim n$），$p$ 所在网格的下标为 $p.1, p.2, \cdots, p.n$，根节点 $O$ 的下标为 $O.1, O.2, \cdots, O.n$. 查找的步骤如下：

(1) 判断是否存在 $i$，使得 $p.i=O.i$，若不存在，转向步骤(2)，若存在，转向步骤(3).

(2) 则将 $p$ 的每个下标加一，指针向下走一步，方向为 $n$ 维索引的主躯干，转向步骤(1).

(3) 先不移动指针，但是可以判断一定可以通过本层次第 $i$ 个 $n-1$ 维空间找到记录 $p$ 所在的网格，此时将 $p$ 所在网格和根节点 $O$ 进行降维，删除第 $i$ 个维度和 $p.i, O.i$ 的值，并将查找范围缩小到本层第 $i$ 个 $n-1$ 维空间，将 $n-1$ 赋值给 $n$，若 $n$ 的值为 1，到第(4)步骤，否则到步骤(1).

(4) $n=1$ 表示已经找到网格所在的一维分支，将指针往分支方向向下遍历直到找到网格，算法结束.

以三维空间为例说明查找过程. 三维记录 $p$ 所在网格下标为 $p.1=4, p.2=3, p.3=1$，根节点 $O$ 的下标 $O.1=5, O.2=5, O.3=5$，首先判断是否存在 $i$（$i$ 的值为 $1\sim3$），使得 $p.i=O.i$，判断结果是不存在，所以将 $p.1, p.2, p.3$ 加 1，得到 $p.1=5, p.2=4, p.3=2$，并将指针遍历至节点 $C_{4,4,4}$. 再次判断是否存在 $i$（$i$ 的值为 $1\sim3$），使得 $p.i=O.i$，存在 $i=1$ 满足条件，可确定要定位的的节点位于本层第 $i$ 个二维空间中，图 6 为该层 TTI 索引的展开图，将查找范围缩小到图 6 中虚线包围区域，并将维度 $i$ 上 $p.1$ 和 $O.1$ 删除，将维度 $n$ 降为 2，只考虑根节点 $O$ 和 $p$ 所在节点第 2 和 3 维度上的值，判断是否存在 $i$（$i$ 的值为 $2\sim3$），使得 $p.i=O.i$，判断结果是不存在，将 $p.2, p.3$ 加 1，指针向灰色区域的躯干方向移动，得到 $p.2=5, p.3=3$，存在 $i=2$ 满足条件，将维度 $i$ 上 $p.2$ 和 $O.2$ 删除，将维度 $n$ 降为 1，表示已经找到网格所在的一维分支，将指针往分支方向向下遍历两步直到找到网格，算法结束. 实线区域内为本层次中查找网格走过的路径.

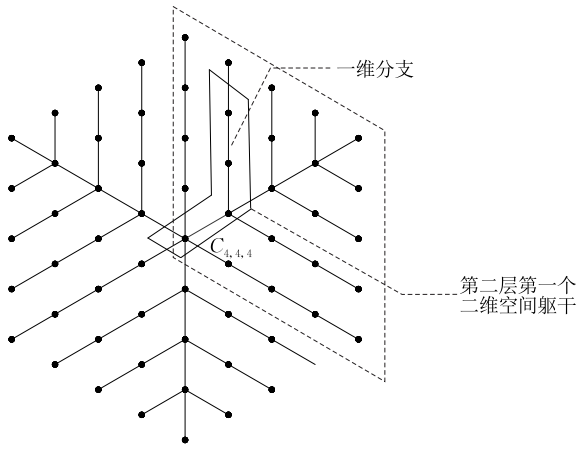



图 6　三维 TTI 索引单层展开图

### 5.2　多维 TTI 索引的计算模块

**定义 7**（剪枝单元）.　$n$ 维 TTI 索引每层有 $n$ 个剪枝单元分别位于 $n$ 个 $n-1$ 维空间中，为距离主躯干节点最近并且具有 $k$ 支配能力的网格.

在 $n$ 维 TTI 索引中并没有选择遍历低维空间上所有节点找出剪枝单元，而是仅遍历一条路径确定一个剪枝单元，该条路径就是 $n-1$ 维空间上的躯干节点集合. 原因是，首先遍历所有节点查找的剪枝单元虽然具有更强大的剪枝能力，但是计算代价很大；其次，在 $n-1$ 维 TTI 索引中的躯干节点集合，是具有父子关系的，但是这个父子关系在 $n$ 维空间中，是不成立的，因为在第 $n$ 维空间中，它们位于一个层次，代表在第 $n$ 个维度上，网格属性是同级的，它们在 $n$ 维空间中属于胞兄胞弟的关系. 不过，一旦确定了 $w$ 层第 $m$ 个 $n-1$ 维 TTI 的躯干节点具有 $k$ 支配能力，则 $w-1$ 层第 $m$ 个 $n-1$ 维 TTI 对应位置的躯干节点及其所带的所有分支，也就是该节点上所有低维空间，可以被直接剪切，使其剪枝能力最大化，分别位于 $w$ 及 $w-1$ 层，保证了它们在第 $n$ 个维度上的父子关系，也就保证了它们在 $n$ 维空间上的父子关系.

与在二维 TTI 索引中一样，可利用层次与层次

间的关系，确定当前层次所有具有 $k$ 支配能力的网格与本层躯干的最大距离.在查找剪枝单元时，可裁剪不可能成为剪枝单元的网格，减少实际需要计算的网格数量，提高算法效率.裁剪策略为：在第 $w$ 层次的第 $m$ 个 $n-1$ 维 TTI 索引中，遍历躯干节点所在的路径直至找到第一个具有 $k$ 支配能力的网格节点，确定其为剪枝单元，并记下其与 $n-1$ 维 TTI 索引根节点的距离 $cuttab\_m$（相当于二维空间中的 $righttab$ 和 $lefttab$）.则在 $w-1$ 层次对应的第 $m$ 个 $n-1$ 维 TTI 索引中，躯干上具有 $k$ 支配能力的剪枝节点与根节点的距离一定小于等于 $cuttab\_m-1$，并且距离子跟节点大于 $cuttab\_m-1$ 的子躯干节点及其分支，可以被剪切.

**定理 5.** 在 $n$ 维 TTI 索引中，假设在当前 $w$ 层第 $m$ 个 $n-1$ 维 TTI 索引中，与根节点距离为 $cuttab\_m$ 的躯干节点为剪枝单元，则 $w-1$ 层次对应的第 $m$ 个 $n-1$ 维 TTI 索引中，剪枝单元与根节点的距离小于等于 $cuttab\_m-1$.

证明. 设 $w$ 层第 $m$ 个 $n-1$ 维 TTI 索引根节点上的网格为 $C_{w,w,\ldots,w}$，与根节点距离为 $cuttab\_m$ 的躯干节点上的网格为 $C_{w+cuttab\_m,w+cuttab\_m,\ldots,w,\ldots,w+cuttab\_m}$（第 $m$ 个维度值始终为 $w$）.$w-1$ 层次对应的第 $m$ 个 $n-1$ 维 TTI 索引根节点上的网格为 $C_{w-1,w-1,\ldots,w-1}$，与根节点距离为 $cuttab\_m-1$ 的躯干节点上的网格为 $C_{w+cuttab\_m,w+cuttab\_m,\ldots,w-1,\ldots,w+cuttab\_m}$（第 $m$ 个维度值始终为 $w-1$），由于 $C_{w+cuttab\_m,w+cuttab\_m,\ldots,w-1,\ldots,w+cuttab\_m}$ 为 $C_{w+cuttab\_m,w+cuttab\_m,\ldots,w,\ldots,w+cuttab\_m}$ 的胞弟，$C_{w+cuttab\_m,w+cuttab\_m,\ldots,w,\ldots,w+cuttab\_m}$ 是具有 $k$ 支配能力的网格，所以 $C_{w+cuttab\_m,w+cuttab\_m,\ldots,w-1,\ldots,w+cuttab\_m}$ 也是具有 $k$ 支配能力的网格.所以，$w-1$ 层次对应的第 $m$ 个 $n-1$ 维 TTI 索引中，剪枝单元与根节点的距离小于等于 $cuttab\_m-1$.

从根节点出发，从高层次向低层次遍历，在每个层次中，遍历 $n$ 个 $n-1$ 维 TTI 索引的躯干节点，遍历的最大步数由集合 $Cut\{cuttab\_1,\cdots,cuttab\_n\}$ 决定（$cuttab\_m$ 决定第 $m$ 个 $n-1$ 维 TTI 索引），设在当前 $n-1$ 维 TTI 索引的躯干走过的步数为 $presenttab$，$k\_tab$ 用于标记找到剪枝节点走过的步数，和用于更新 $Cut$ 集合的值，指针从根节点开始，初始化 $m=1$，$presenttab=0$，$Cut$ 集合中每个元素和 $k\_tab$ 为无穷大.当集合 $Cut$ 中存在小于 0 的元素时，算法结束，具体裁剪步骤如下：

(1) 指针向当前层次第 $m$ 个 $n-1$ 维 TTI 索引的躯干向下遍历一步，将 $presenttab$ 值加一，若指针为空或者 $presenttab>cuttab\_m$，转向步骤(3)；否则，计算当前节点 $k$ 支配能力.若当前节点具有 $k$ 支配能力，转向步骤(2)，若不具有，转向步骤(1)；

(2) 若 $k\_tab$ 大于 $presenttab$，将 $presenttab$ 赋值给 $k\_tab$，否则转向步骤(1)；

(3) 将 $k\_tab$ 赋值给 $cuttab\_m$，$m$ 的值加一，若 $m$ 小于等于 $n$，重置 $presenttab=0$，$k\_tab$ 为无穷大，返回步骤(1)，若 $m$ 大于 $n$，说明本层次遍历完毕，转向步骤(4)；

(4) 指针向主躯干节点往下走一步，将 $Cut$ 集合中每个元素的值减 1，判断 $Cut$ 集合中是否存在值小于 0 的元素，若不存在，重置 $m=1$，$presenttab=0$，$k\_tab$ 为无穷大；若存在，算法结束.

在计算网格 $C$ 的 $k$ 支配能力时，可以通过分别一一定位 $C$ 的胞兄和支配 $C$ 的节点计算 $RN$ 之和来解决.

### 5.3 多维 TTI 索引的代价分析

**定理 6.** 假设空间维度为 $n$ 维，每个维度上有 $r$ 个取值的网格空间中，TTI 索引最多只需要遍历 $r$ 个网格和进行 $n$ 次自加 1 运算，就可查找空间中任意一个网格.

证明. 假设从 $C_{0,\ldots,0}$ 出发，需要查找的网格为 $C_{p1,p2,\ldots,pn}$，其中 $p_1,p_2,\cdots,p_n$ 均小于等于 $r$，不妨假设 $p_1\leq p_2\leq\cdots\leq p_n$.接下来证明根据 TTI 索引的遍历规则，找到 $C_{p1,p2,\ldots,pn}$ 所花费的步数一定小于 $r$.根据遍历规则，第一次先走 $p_1$ 步，到达网格 $C_{p1,p1,\ldots,p1}$，忽略 $p_1$ 所在的维度，继续查找；第二次（$p_2-p_1$）步，到达 $C_{p1,p2,\ldots,p2}$，忽略 $p_2$ 所在的维度，继续查找；第 $n$ 次（$p_n-p_{n-1}$）步，到达 $C_{p1,p2,\ldots,pn}$，自加次数为 $n$，将 $n$ 次所走的步数相加：$p_1+(p_2-p_1)+\cdots+(p_n-p_{n-1})=p_n$.总步数为 $p_n$，由于 $p_n\leq r$，最后得出遍历所需总步数小于等于 $r$.

基于网格的 top-$k$ 查询算法，它们的裁剪规则和计算自由区及影响区规则都是基于多次查找网格的操作，而传统算法通常将网格用链表形式存储，或未说明网格的具体存储模式，查找方式为随机查找.例如传统的网格算法，在一个 $n$ 维，每个维度上有个 $r$ 个取值的网格空间中，空间上一共有 $r^n$ 个网格，查询并未在网格上建立索引，随机查找一次平均需要遍历的网格数为 $r^n/2$，查找一个网格的时间复杂度为 $O(r^n)$，建立一个网格索引需要对每个目标对象都进行一次查找网格算法，如果目标对象集合较大，传统算法建立和维护网格索引花费代价也会很大.TTI 索引的时间复杂度为 $O(r)$，很大程度上增加了

定位网格的速度，适合目标对象属性发生动态变化时索引结构需要频繁调整的情况. 虽然初始建立索引仍旧需要花费一定的代价，不过由于索引的建立只依赖于数据的数量以及维度，不依赖用户给出的权重系数及 $k$ 值. 所以多维索引可在线下建立，不花费用户的查询时间，不影响返回结果的效率.

在计算 $k$ 支配能力时，对于网格 $C_{p1,p2,\cdots,pn}$，$C_{p1,p2,\cdots,pn}$ 和其胞兄和支配其的网格表达式满足，在第 $i$ 个维度上的值小于等于 $p_i$，所以 $C_{p1,p2,\cdots,pn}$ 和其胞兄和支配其的网格数量总数为 $p_1\times p_2\times\cdots\times p_n$. 在传统算法中，查找一个节点平均需要遍历 $r^n/2$ 个节点，则查找 $p_1\times p_2\times\cdots\times p_n$ 个节点需要遍历 $p_1\times p_2\times\cdots\times p_n\times r^n/2$ 个网格. 而 TTI 索引只需要遍历 $p_1\times p_2\times\cdots\times p_n\times r$ 个网格以及进行 $p_1\times p_2\times\cdots\times p_n\times n$ 次加一运算. 虽然划分的网格数目可以人为减少，但是这样会降低算法裁剪能力，不利于算法总体性能，所以在实际操作中，网格数量需要维持在一定的量. 定位网格的速度优势不仅仅体现在建立网格上，计算网格的 $k$ 支配能力同样需要大量定位网格的操作，而计算 $k$ 支配能力的快慢和需要计算 $k$ 支配能力的网格数量决定了划分自由区及影响区效率.

## 6　实验与结果

### 6.1　实验设置

采用 C 语言在 Windows 10 环境下对 GID-Top-$k$ 算法性能进行测试. 实验 PC 机的配置为：Intel(R) Core i5，8 GB 内存，2 TB 硬盘，64 位操作系统. 实验数据为综合商场真实的商品信息数据，其中商品大类 1403 种，商品种类总数是 66 231 个. 其中测试数据来源于数据堂，数据集为电子商务数据中的电商评论数据. 本文根据提出的 GID-Top-$k$ 算法，与 RankCube[9] 算法和 Top-$k$ 算法和 CIA 算法[13] 进行了对比. 实验在不同条件下测试了 GID-Top-$k$ 算法的性能，并且为了避免实验结果的随机性，查询时间选取 10 次测试结果的平均值. 其中为了验证 GID-Top-$k$ 算法中动态调整的有效性，实验过程中人工对餐厅的部分属性进行随机更新.

### 6.2　参数 $k$ 大小对查询性能的影响

如图 7 所示，为参数 $k$ 大小对 4 种算法在 50 000 个数据对象的 2 维真实数据集上运行时的影响. 从图中实验结果可以看出，RankCube 算法和 GID-Top-$k$ 算法的查询时间都逐渐增大，但是，top-$k$ 算法和 CIA 算法对参数 $k$ 变化不敏感，这是由于 top-$k$ 算法需要对数据进行全排序，所以查询时间不受参数 $k$ 的影响，而 CIA 算法在 $k$ 值较小时，实际返回结果数量与 $k$ 值相差较大（实际返回结果数均大于 500），所以算法基本不受 $k$ 值影响. GID-Top-$k$ 算法的查询时间增加的比较缓慢，这是因为 GID-Top-$k$ 算法中分配影响区域的面积增加较缓慢，GID-Top-$k$ 算法逐步检索网格，$k$ 值增大会导致数据访问量的逐步增大，说明在 GID-Top-$k$ 算法中，$k$ 值对网格节点具有裁剪作用，$k$ 值的增大导致可裁剪的网格数减少，故查询时间增加. 由于 CIA 算法并不是精确的 top-$k$ 查询，GID-Top-$k$ 算法在 $k$ 值变化时与之比较并没有效率上的优势. 当 $k=250$ 时，GID-Top-$k$ 算法查询速度是 RankCube 算法的 1.7 倍，当 $k=50$ 时，GID-Top-$k$ 算法查询速度是 top-$k$ 算法的 7.7 倍.

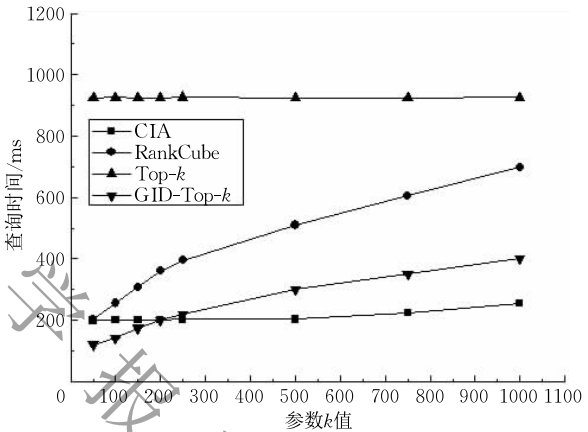



图 7　参数 $k$ 大小对查询性能的影响

### 6.3　静态场景下目标对象集规模对查询性能的影响

图 8 所示为 4 种算法在静态 2 维人工合成数据对象数量变化时的性能. 实验结果显示，GID-Top-$k$ 算法和 CIA 算法的查询时间增加较为平缓，查询效率总体表现差不多. 在数据集合较小时，GID-Top-$k$ 算法所耗费的时间大于 top-$k$ 算法，这是因为 GID-Top-$k$ 算法需要对目标对象建立网格索引，划分影响区和自由区，动态变化判断等步骤，需消耗额外时间. 但是，随着数据集的增大，GID-Top-$k$ 算法在效率上的优越性逐渐体现，而 top-$k$ 算法由于需要对数据集进行全排序，在数据量较大时，查询性能大幅下降. 同样，RankCube 算法性能也降低了，这是因为在大量数据中找到满足条件的元组需要做更多的随机访问. GID-Top-$k$ 算法对网格建立了索引，所以对数据集的尺寸较不敏感，可在大数据量情况下性能表现良好.

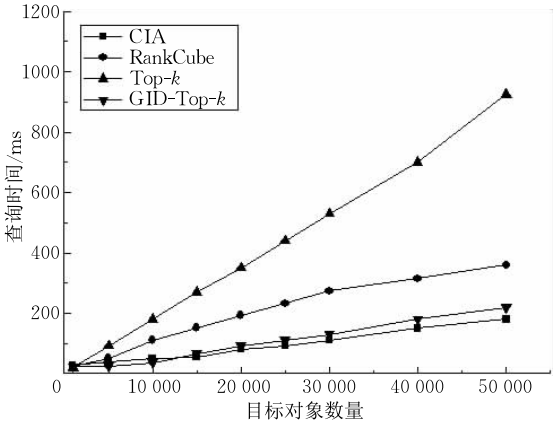



图 8 静态场景下目标对象集规模对查询性能的影响

### 6.4 动态场景下目标对象集规模对查询性能的影响

图 9 所示实验为在图 8 的基础上，人工对数据集中 5%的目标对象的属性值进行随机修改得到的实验结果. Top-$k$ 和 CIA 算法没有考虑数据属性动态变化的情况，所以只要有数据发生变动，为保证查询结果的准确性，需频繁的重新运行算法，降低了效率. 而提出的 GID-Top-$k$ 算法只有在改动发生在影响区内才需进行计算，并且当数据变化时，可以快速重定位网格，维护索引结构代价较小，明显提高了算法效率，使之更适应数据动态变化时的情况. 实验结果表明查询间增幅较为平缓. 在目标对象集合较大时，RankCube 算法在静态中受限的性能，在动态情况下并没有明显的提升，这是因为数据的变化需要不断的维护索引结构以保证结果的准确性.

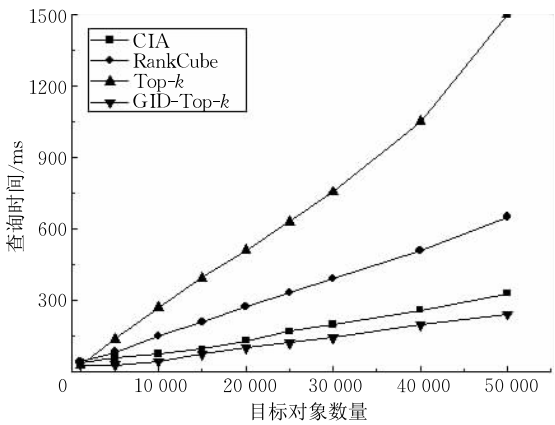



图 9 动态场景下目标对象集规模对查询性能的影响

### 6.5 目标对象的属性个数对查询性能的影响

对于同一个目标对象，不同用户关注的属性维度是不一样的，由于 CIA 算法不支持属性个数的扩展，图 10 所示为其他三种算法在计算 50000 个真实数据集时，数据属性维度的大小对算法性能的影响. 因为本文并未涉及高维子空间选择算法的研究，考虑到高维全空间查询在实际应用中较少出现，就其花费代价来说性价比较低，所以本文在实验中只验证了最多 6 个维度. 从图中可以看出 top-$k$ 算法随着数据对象的属性个数增加，查询时间缓慢增长，这是由于算法需要多消耗部分时间在累加计算上，在对每个数据对象调用聚合函数时代价增大. GID-Top-$k$ 算法对于属性维度的增加体现在 TTI 索引中每层分支数量的增加，查询过程不变，需要多遍历的网格数量只是有增加，所以对整体算法的性能并没有太大影响. RankCube 算法的查询时间随着目标属性维度增加略有下降再升高，这是因为 RankCube 算法是以 4 个维度为基础建立的，当维度小于 4 时，需要将网格向低维度投影，因此需要访问更多网格，算法效率降低. 总体来说，GID-Top-$k$ 算法和 top-$k$ 算法对目标对象属性个数不敏感.

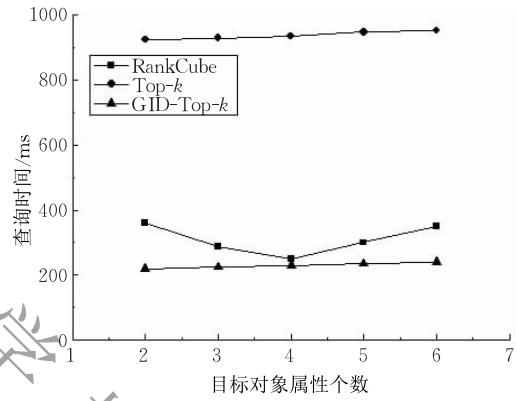
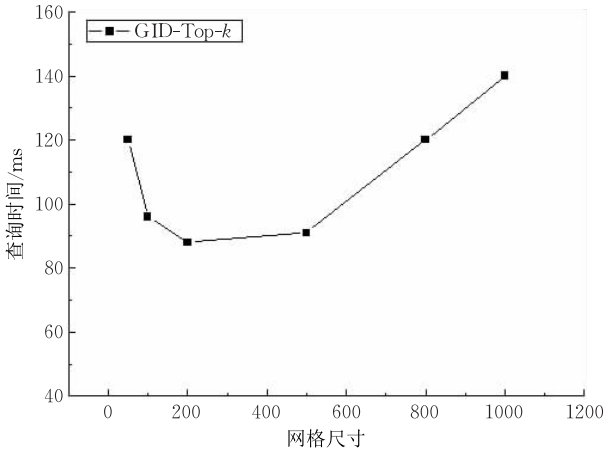



图 10 目标对象的属性个数对查询性能的影响

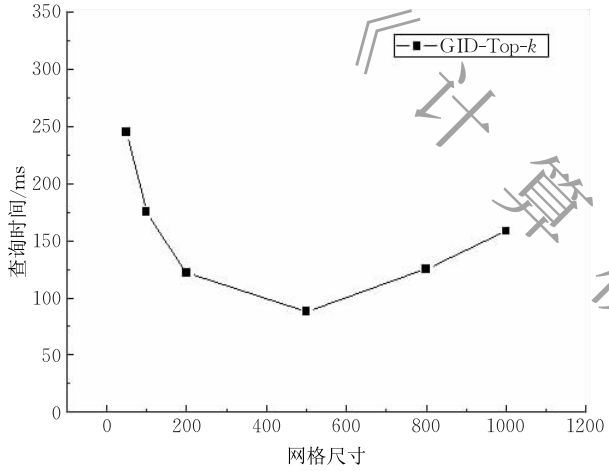
### 6.6 网格尺寸对查询性能的影响

图 11 所示测试网格尺寸大小对 GID-Top-$k$ 算法的影响，实验采用两组数据规模分别为 20000 和 50000 的二维人工数据集合进行测试，称为 A 和 B 组. 通过改变网格边长来改变每个网格内的数据量以测试算法性能. 当网格尺寸从小变大，两组算法性能均先升高后降低，这是因为网格尺寸较小时，TTI 索引中的点数较多，划分区域时需要遍历较多节点，并且每个网格内的记录数量较少，无法体现出概要方法的优势之处，当网格尺寸较大时，虽然索引的结构比较简单，可是划分自由区和影响区时较为粗糙，会导致影响区面积较大，数据变动时重新计算的可能性较大，并且每个网格中存储的数据链表较长，不利于整体算法性能. A 组实验网格尺寸为 100 时算法性能最佳，B 组实验网格尺寸为 500 时算法性能最佳. 说明随着数据量的增大，算法性能最佳时刻的网格尺寸也增大. 当 A 组网格尺寸值在 100～500 间时，B 组网格尺寸值在 200～800 间，查询时间对尺寸变

化不敏感，这是因为此时访问外层网格索引的效率和内层网格数据的效率之和正好达到平衡状态.




(a) 数据规模为20 000网格尺寸对GID-Top-$k$查询性能的影响

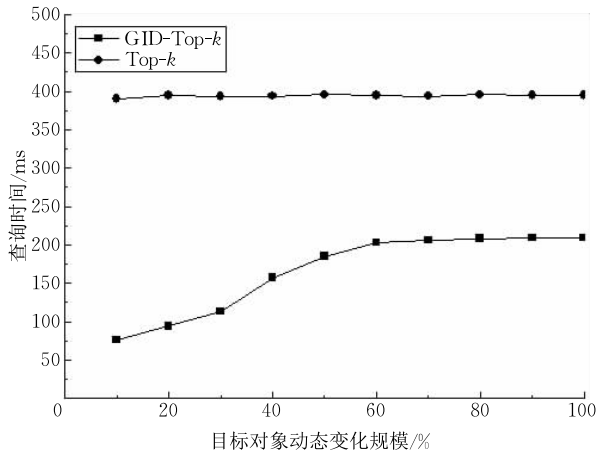



(b) 数据规模为50 000网格尺寸对GID-Top-$k$查询性能的影响

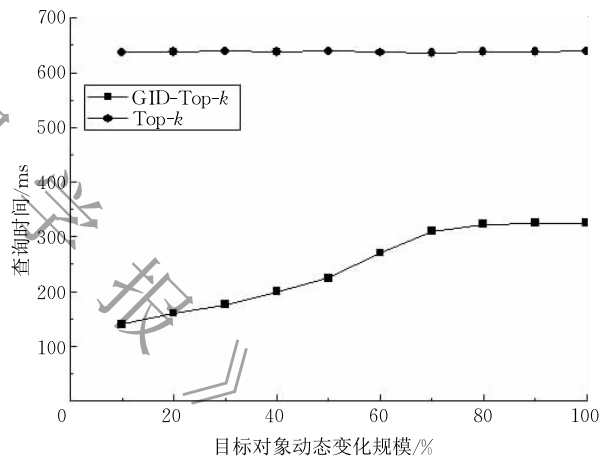
图 11　网格尺寸对 GID-Top-$k$ 查询性能的影响

### 6.7　目标对象动态变化集合规模对查询性能的影响

GID-Top-$k$ 算法需要判断发生动态变化的数据点是否属于影响区来决定是否重新运行计算模块. 所以动态变化集合的规模的大小会直接影响进行重新计算的概率，进而影响算法查询效率. 实验测试了目标对象数量分别为 10 000 和 20 000 时，不同比例动态变化的二维人工合成数据集下传统 top-$k$ 算法，GID-Top-$k$ 算法的运行时间，如图 12(a)所示为目标对象数量为 10 000 时查询时间变化情况，随着目标对象动态变化集合比例的增加，传统 top-$k$ 算法的查询性能变化不大，这是因为在传统 top-$k$ 中一旦有数据发生改变就需重新进行全排序运算，重新计算的概率与动态变化的规模无关. GID-Top-$k$ 算法随着目标对象动态变化集合比例的增加，查询时间也随之增加，实验还发现当 GID-Top-$k$ 算法动态变化比例增加到 30%时，算法的查询时间增速出现明显提升，在动态变化比例增加到 50%时趋于稳定，这是因为此时需要进行重新计算的概率趋于稳定. 图 12(b)为目标对象为 20 000 时的情况，由于网格尺寸的原因，使得数据集对变化比率较不敏感，但是算法的初始代价也较高，总体来说，数据集的大小不会改变算法性能的整体走势.




(a) 数据规模为10 000动态变化规模对查询性能的影响




(b) 数据规模为20 000动态变化规模对查询性能的影响

图 12　动态变化规模对查询性能的影响

### 6.8　目标对象分布密度对查询性能的影响

由于网格索引不同于一般的索引，需要将查询空间在 $n$ 个维度上划分成等大的网格区域，所以，在其他实验配置参数相同的情况下，数据在每个网格上分布的密度对算法性能会产生一定影响. 为了便于改变数据分布的密度，本节实验采用人工数据集对算法进行测试，实验设置了 3 组目标对象数量均为 10000 的二维数据集，均对它们建立 10 层的 TTI 索引. 第一组有 50%的目标对象均位于第一层次，第二组仅有 5%的目标对象位于第一层次，第三组目标对象在数据空间均匀分布，实验测试在 $k$ 值增大的情况下，三组实验数据的性能变化情况，因为若

算法受 $k$ 值影响程度和查询时间可以看出算法的裁剪性能,受 $k$ 值影响越大,查询时间越短,算法性能越好(可参考 top-$k$ 算法).图 13 所示实验结果表明,GID-Top-$k$ 算法的效率对目标对象在数据空间的分布较为敏感.第一个数据集中,GID-Top-$k$ 算法无法裁剪至少 50%的目标对象,而不能裁剪的对象最后全都需要进行排序以选出 $k$ 个结果,从实验结果可看出,GID-Top-$k$ 算法基本不受 $k$ 值影响,说明 GID-Top-$k$ 算法的裁剪性能已经大大降低.第二组实验数据设置很适合 GID-Top-$k$ 算法,性能得到明显提升,但是随着 $k$ 值增大,查询时间的增幅变大,说明算法性能的提升会受到 $k$ 值的限制.第三组实验数据中,GID-Top-$k$ 算法适中,增幅也较平稳,说明 GID-Top-$k$ 算法在该数据集中裁剪方法是有效的.

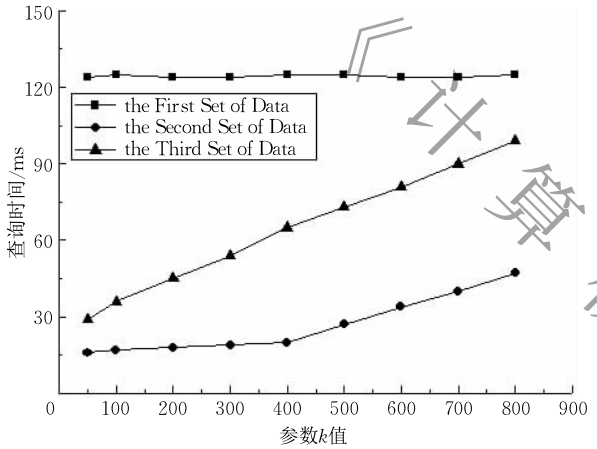



图 13 参数 $k$ 在三种数据分布数据集下对查询性能的影响

## 7 结束语

本文在目标对象属性值动态变化的情况下基于网格索引提出了一种 GID-Top-$k$ 查询算法.GID-Top-$k$ 在 top-$k$ 查询基于概要的方法上提出了 TTI 树形索引,在用 TTI 树形索引实现快速访问的基础上结合了概要方法中的快速预计算和剪枝过程.与传统基于概要的 top-$k$ 算法相比,GID-Top-$k$ 算法为所有网格建立索引,并利用网格在索引中的位置和网格的概要信息做裁剪,大大减少需要访问的网格数量并提高了算法效率.通过实验对 GID-Top-$k$ 算法和已有 top-$k$ 算法在多种情况下进行分析和对比.实验结果显示 GID-Top-$k$ 算法在数据集合较大时更能体现其性能的优越性.并且对目标对象数据维度大小及目标对象属性动态变化等因素不敏感.本文工作还能从以下两个方面深入:(1)将 TTI 索引应用到数据流 top-$k$ 查询中,考虑在线划分影响区和自由区;(2)在基于网格动态 top-$k$ 查询中加入多维选择功能.

## 参 考 文 献

[1] Xin D, Han J, Cheng H, et al. Answering Top-$k$ queries with multi-dimensional selections: The ranking cube approach //Proceedings of the 32nd International Conference on Very Large Data Base (VLDB). Trondheim, Norway, 2006: 463-474

[2] Han Jing, Zhou Shu-Fang, Luo Wen. Public Comment Catering Weathervane Series: China Food & Beverage Market Data Report. Beijing: Chinese Business Publisher, 2013 (in Chinese)
(韩璟,周淑芳,骆雯. 大众点评餐饮风向标系列:中国餐饮市场数据报告. 北京:中国商业出版社, 2013)

[3] Chang Y C, Bergman L, Castelli V, et al. The Onion technique: Indexing for linear optimization queries. ACM SIGMOD Record, 2000, 29(2): 391-402

[4] Fagin R, Lotem A, Naor M. Optimal aggregation algorithms for middleware//Proceedings of the 20th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems. Santa Barbara, USA, 2001: 102-113

[5] Hristidis V, Koudas N, Papakonstantinou Y. PREFER: A system for the efficient execution of multi-parametric ranked queries. ACM SIGMOD Record, 2001, 30(2): 259-270

[6] Mouratidis K, Bakiras S, Papadias D. Continuous monitoring of Top-$k$ queries over sliding windows//Proceedings of the ACM SIGMOD International Conference on Management of Data. Chicago, USA, 2006: 635-646

[7] Akbarinia R, Pacitti E, Valduriez P. Best position algorithms for Top-$k$ queries//Proceedings of the International Conference on Very Large Data Bases (VLDB). Vienna, Austria, 2007: 495-506

[8] Mamoulis Nikos, Yiu Man Lung, Cheng Kit Hung, et al. Efficient Top-$k$ aggregation of ranked input. ACM Transactions on Database Systems, 2007, 32(3): 19

[9] Han Xi-Xian, Yang Dong-Hua, Li Jian-Zhong, et al. TKEP: An efficient Top-$k$ query processing algorithm on massive data. Chinese Journal of Computers, 2010, 33(8): 1405-1417 (in Chinese)
(韩希先,杨东华,李建中等. 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)

[10] Han Xi-Xian, Liu Xian-Min, Li Jian-Zhong, et al. TMS: A novel algorithm for Top-$k$ queries with multi-dimensional selections on massive data. Journal of Computer Research and Development, 2017, 54(3): 570-585 (in Chinese)
(韩希先,刘显敏,李建中等. TMS:一种新的海量数据多维选择 Top-$k$ 查询算法. 计算机研究与发展, 2017, 54(3): 570-585)

[11] Zou L, Chen L. Pareto-based dominant graph: An efficient indexing structure to answer Top-$k$ queries. IEEE Transactions on Knowledge & Data Engineering (TKDE), 2011, 23(5): 727-741

[12] Xin D, Chen C, Han J. Towards robust indexing for ranked queries//Proceedings of the International Conference on the 32nd Very Large Data Bases (VLDB). Seoul, Korea, 2006: 235-246

[13] Das G, Gunopulos D, Koudas N, et al. Answering Top-$k$ queries using views//Proceedings of the International Conference on Very Large Data Bases (VLDB). Seoul, Korea, 2006: 451-462

[14] Kalashnikov D V, Prabhakar S, Hambrusch S E, et al. Efficient evaluation of continuous range queries on moving objects//Proceedings of the International Conference on Database and Expert Systems Applications. Springer-Verlag, Aix-en-Provence, France, 2002: 731-740

[15] Gan Liang, Jin Xin, Jia Yan, et al. Gridded dominant graph: A more efficient index structure for Top-$k$ queries based on reverse dominant point set. Journal of Computer Research and Development, 2010, 47(10): 1771-1784(in Chinese)
(甘亮，金鑫，贾焰等. GDG:一种基于逆支配点集的 Top-$k$ 高效查询索引方法. 计算机研究与发展，2010，47(10)：1771-1784)

[16] Kontaki M, Papadopoulos A N, Manolopoulos Y. Continuous Top-$k$ dominating queries in subspace//Proceedings of the Panhellenic Conference on Informatics. Samos Island, Greece, 2008: 31-35

[17] Chen D, Sun G Z, Gong N Z. Efficient approximate Top-$k$ query algorithm using cube index//Proceedings of the Asia-Pacific Web Conference on Web Technologies and Applications. Beijing, China, 2011: 155-167

[18] Rossi C, Moura E S D, Carvalho A L, et al. Fast document-at-a-time query processing using two-tier indexes//Proceedings of the International ACM SIGIR Conference on Research and Development in Information Retrieval. Dublin, Ireland, 2013: 183-192

[19] Daoud C M, Edleno S D M, Carvalho A, et al. Fast Top-$k$ preserving query processing using two-tier indexes. Information Processing & Management, 2016, 52(5): 855-872

[20] Li Bo-Han, Zhang Chao, Li Dong-Jing, et al. A DSP-Topk query optimization algorithm supporting indoor obstacle space. Journal of Computer Research and Development, 2017, 54(3): 557-569(in Chinese)
(李博涵，张潮，李东静等. 支持室内障碍空间的 DSP-Topk 查询优化算法研究. 计算机研究与发展，2017，54(3)：557-569)

[21] Vlachou A, Doulkeridis C, Kotidis Y, et al. Monochromatic and bichromatic reverse Top-$k$ queries. IEEE Transactions on Knowledge & Data Engineering, 2011, 23(8): 1215-1229

[22] Vlachou A, Doulkeridis C, Nørvåg K, et al. Branch-and-bound algorithm for reverse Top-$k$ queries. IEEE Transactions on Knowledge & Data Engineering, 2013, 23(8): 481-492

[23] Li Miao, Gu Yu, Chen Mo, et al. Efficient processing method for reverse Top-$k$ spatial preference queries. Journal of Software, 2017, 28(2): 310-325(in Chinese)
(李淼，谷峪，陈默等. 一种针对反向空间偏好 Top-$k$ 查询的高效处理方法. 软件学报，2017，28(2)：310-325)

[24] Ma You-Zhong, Ci Xiang, Meng Xiao-Feng. Parallel Top-$k$ join on massive high-dimensional vectors. Chinese Journal of Computers, 2015, 38(1): 86-98(in Chinese)
(马友忠，慈祥，孟小峰. 海量高维向量的并行 Top-$k$ 连接查询. 计算机学报，2015，38(1)：86-98)

[25] Ilyas I F, Beskales G, Soliman M A. A survey of Top-$k$ query processing techniques in relational database systems. ACM Computing Surveys, 2008, 40(4): 1-58

[26] Koudas N, Ooi B C, Tan K L, et al. Approximate NN queries on streams with guaranteed error/performance bounds//Proceedings of the 30th International Conference on Very Large Data Bases (VLDB). Toronto, Canada, 2004: 804-815

[27] Mouratidis K, Papadias D, Hadjieleftheriou M. Conceptual partitioning: An efficient method for continuous nearest neighbor monitoring//Proceedings of the ACM SIGMOD International Conference on Management of Data. Baltimore, USA, 2005: 634-645

[28] Xiong Xiao-Peng, Mokbel M F, Aref W G. SEA-CNN: Scalable processing of continuous $k$-nearest neighbor queries in spatio-temporal databases//Proceedings of the International Conference on Data Engineering (ICDE). Tokyo, Japan, 2005: 643-654

[29] Raptopoulou K, Papadopoulos A N, Manolopoulos Y. Fast nearest-neighbor query processing in moving-object databases. GeoInformatica, 2003, 7(2): 113-137

[30] Borzsony S, Kossmann D, Stocker K. The skyline operator//Proceedings of the International Conference on Data Engineering (ICDE). Heidelberg, Germany, 2001: 421-430

**DENG Dan-Ping**, M. S. Her research interests include spatial and spatio-temporal databases and context aware.

**QIN Xiao-Lin**, Ph. D., professor, Ph. D. supervisor. His research interests include spatial and spatio-temporal databases, data management and security in distributed environment.

**LI Bo-Han**, Ph. D., associate professor. His research interests include spatial and spatio-temporal databases, geographic information system.

**ZHENG Wei**, M. S. His research interests include data management and recommendation system.

**LIU Liang**, Ph. D., lecturer. His research interests include data security, cloud data management and wireless sensor networks.

**LI Xue**, Ph. D., professor. His research interests are opinion analysis on social media, big data analytic, knowledge discovery from sequences, mining distributed high-speed time-variant data streams.

**Background**

This paper focuses on the research for query processing issues on massive amount of data based on a novel index TTI. Top-$k$ is an crucial operation in multiple fields since it returns $k$ most important objects among potential huge answer space according to a given ranking. However, for massive data, random access is prohibitively expensive. Previous works do not adapt to the situation when object attribute values is dynamic or the data amount is great. Trunk Tree Index (TTI) structure is based on grid index. It can remedy the lack ability of dealing these data before. A novel grid index method called GID-Top-$k$ algorithm can decrease the random access by the technique of advanced grid. TTI index can acquire the position of the grid subscript when insert, delete and update happen. In addition, the process of pruning in Grid Index based Dynamic Top-$k$ can effectively compute the determinant unit through the summary of the grid in the Trunk tree Index, and divide the Influence area and Free area. Finally, pruning the data set and reducing the probability of recalculation when data changes frequently. Now we devote to developing the improved Top-$k$ queries to apply in big data analysis, multi-criteria retrieval and so on.

This work is supported in part by the National Natural Science Foundation of China (Nos. 61728204, 61672284, 61373015, 61300052, and 41301407), the Natural Science Foundation of Jiangsu Province (No. BK20130819), the Youth Science and Technology Innovation Funding (No. NT2018028), the CSC (No. 201406835051), the Innovation Funding (No. NJ20160028), the State Key Laboratory of Smart Grid Protection and Control, Civil Aviation Administration of China (No. AS-SA2015/21), the Australian Research Council Discovery Project (No. ARC DP140100104).