OIQ-tree：一种支持大规模空间文本数据流上连续 k 近邻查询的索引

杨 茸 牛保宁

（太原理工大学信息与计算机学院 山西 晋中 030600）

摘 要 空间文本数据流上连续 k 近邻查询（Continuous k-nearest neighbor Queries over Spatial-Textual data streams，CkQST）能在空间文本对象组成的数据流上检索并实时更新 k 个包含指定关键字的空间邻近对象，是空间文本数据流上连续查询（Continuous Queries over Spatial-Textual data streams，CQST）的一种，以预订（subscribe）的方式广泛应用于广告定位、微博分析、地图导航等领域. 求解 CkQST 采用 CQST 的求解框架——构建空间文本混合索引组织查询，利用索引的空间过滤和文本过滤能力，为不断到来的对象匹配查询. 该框架的求解效率取决于索引的过滤能力，提高索引过滤能力的主要途径是将查询的空间搜索范围映射到索引结构的最小区域，减少需要验证的查询数量. 这一途径适用于查询空间搜索范围很少变化的情况. 对于 CkQST，覆盖 k 个最邻近对象的空间范围随着符合文本匹配条件的对象的数量的变化而变化，与之对应的索引项需要同步更新，代价高. 针对这一问题，本文选择能够高效支持空间范围变化的 Quad-tree 和关键字查找的倒排索引，构成空间文本混合索引，组织 CkQST. 在空间过滤方面，提出内存代价模型 VUMBCM（Verification and Update of Memory-Based Cost Model），通过平衡索引更新代价和验证代价，优化查询空间搜索范围到 Quad-tree 节点的映射. 在文本过滤方面，采用基于块的有序倒排索引，组织 Quad-tree 节点内的查询，以快速定位需要验证的查询，避免对倒排列表中大量不可能匹配查询的访问；批量处理包含共同文本项的对象，提高文本验证时的对象吞吐量. 由此构建的混合索引，称为 OIQ-tree. 实验表明，OIQ-tree 中的代价模型及基于块的有序倒排索引能够支持 CkQST 的高效求解. 与目前先进的索引技术相比，当查询规模达到 2000 万时，因数据流中对象的变化导致的索引平均更新时间降低了 46%，数据流中对象的平均处理时间降低了 22%.

关键词 空间文本查询；数据流；空间文本索引；k 近邻；连续查询

中图法分类号 TP391 **DOI 号** 10.11897/SP.J.1016.2021.01732

OIQ-tree: Support Continuous k Nearest Neighbor Queries over Large-Scale Spatial-Textual Data Streams

YANG Rong NIU Bao-Ning

(*College of Information and Computer, Taiyuan University of Technology, Jinzhong, Shanxi* 030600)

Abstract The continuous k nearest neighbor queries over spatial textual data streams (CkQST for short) retrieve and continuously monitor at most k nearest neighbor objects to the user specified location containing all the user specified keywords over the data streams composed of spatial textual objects, which is a type of continuous queries over spatial textual data streams and has been widely used in a wide variety of location based applications, such as location aware targeting of advertisements, analysis of microblogs and mobile navigation services etc. by way of subscriptions. Evaluating CkQST utilizes the solution framework of evaluating the generic continuous queries over spatial textual data streams, i. e. selecting a spatial index and a textual index to form a hybrid

收稿日期：2019-12-27；在线发布日期：2021-01-16. 本课题得到国家自然科学基金（61572345）资助.**杨 茸**，博士研究生，中国计算机学会（CCF）会员，主要研究方向为空间数据管理与分析. E-mail：13546728560@126.com. **牛保宁**（通信作者），博士，教授，中国计算机学会（CCF）会员，主要研究领域为数据库系统的性能管理、音频检索、大数据管理及分析. E-mail：niubaoning@tyut.edu.cn.

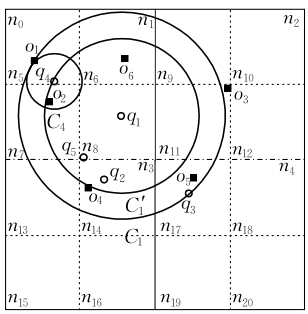
spatial textual index to organize the queries, and matching the incoming objects continuously generated utilizing the spatial and textual filtering capabilities of the index. In this framework, the evaluation efficiency depends on the filtering ability of the index, and the major approach to improving the filtering ability of the index is to map the spatial search range of the queries to the smallest area of the index structure to reduce the number of queries being verified by the objects over data streams, which is suitable for the situations where the search range of the queries rarely changes. For CkQST, the spatial range covering k nearest neighbor qualified objects frequently changes with the number of the objects containing all the query keywords, and accordingly the index should be updated synchronously, which requires very expensive cost. To solve the problem, this paper selects the Quad-tree integrated with an inverted index to construct a hybrid spatial textual index to organize CkQST, where the Quad-tree can efficiently support the frequent change of the spatial range of CkQST, and the inverted index can efficiently support the keyword query. With respect to the spatial filtering, a memory based cost model VUMBCM (Verification and Update of Memory Based Cost Model) is proposed to optimize the mapping the search range of CkQST to the Quad-tree nodes by trading off the verification cost and update cost of index; With respect to the textual filtering, a block based ordered inverted index is proposed to organize CkQST at the Quad-tree nodes, which can quickly locate the promising queries and avoid verifying a large number of unpromising queries in the posting lists. Additionally, the ordered inverted index allows multiple objects over data streams containing common texts to be processed in a batch, which can improve the throughput performance during textual verification. The above hybrid index integrated the Quad-tree with the block based ordered inverted index and the cost model is called OIQ-tree. The extensive experiments on real world and synthetic datasets demonstrate that the proposed index OIQ-tree can efficiently evaluate CkQST. Compared with the state of the art techniques, when the number of the subscribed queries reaches 20 million, the average index updating time caused by the incoming objects over the data streams decreases by 46%, and the average incoming objects processing time decreases by 22%.

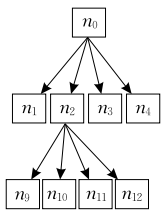
Keywords　spatial-textual query; data streams; spatial-textual index; k nearest neighbor; continuous query

1　引　言

空间文本数据流上连续 k 近邻查询(Continuous k-nearest neighbor Queries over Spatial-Textual data streams,CkQST)是在空间文本对象数据流上检索并实时更新 k 个包含指定关键字(文本)的空间邻近对象,广泛应用于各类基于位置的服务中.

图 1 描述了一个贯穿全文的 CkQST 实例. t_0 时刻,系统中有 5 个用户预订的 CkQST $\{q_1,\cdots,q_5\}$ 和 5 个已经到来的空间文本对象 $\{o_1,\cdots,o_5\}$,查询和对象的经纬度位置分别用小空心圆圈和小实心方框标出,包含的文本集合及到期时刻在图的下方列出. 每个查询需要检索2个包含指定文本的最邻近对象.





查询	文本集	到期时刻	对象	文本集	到期时刻
q_1	$\{w_1,w_2,w_3\}$	t_5	o_1	$\{w_1,w_2,w_3,w_7\}$	t_7
q_2	$\{w_1,w_2,w_5\}$	t_6	o_2	$\{w_1,w_2,w_4,w_5\}$	t_8
q_3	$\{w_1,w_2,w_7\}$	t_5	o_3	$\{w_1,w_2,w_4,w_7\}$	t_5
q_4	$\{w_1,w_2\}$	t_4	o_4	$\{w_1,w_2,w_3,w_5,w_6\}$	t_6
q_5	$\{w_1,w_2,w_5\}$	t_4	o_5	$\{w_1,w_2,w_6,w_7\}$	t_6

图 1　CkQST 实例

空间区域用三层 Quad-tree[1] 组织，并对节点依次编号，根节点为 n_0. 以 q_1 的计算为例，当前时刻返回结果集$\{o_4,o_1\}$. 以 q_1 为中心、覆盖$\{o_4,o_1\}$的最小圆形区域 C_1 被称为 q_1 的空间搜索范围. t_1 时刻，新对象 o_6 到达，经纬度位置在图 1 中标出，包含文本集$\{w_1,w_2,w_3\}$. 它包含 q_1 及 q_4 全部文本，落入 q_1 的空间搜索范围 C_1 内，但是没有落入 q_4 的空间搜索范围 C_4 内，q_1 的结果集更新为$\{o_6,o_4\}$，空间搜索范围更新为 C_1'，q_4 的结果集及空间搜索范围不变. t_2 时刻，o_6 到期，C_1' 内满足文本匹配条件的对象数量不足 2 个，重新为 q_1 计算结果，新的结果集更新为$\{o_4,o_1\}$，空间搜索范围更新为 C_1.

挑战. 作为空间文本数据流上连续查询(Continuous Queries over Spatial-Textual data streams，CQST)的一种，求解 CkQST 采用 CQST 的求解框架，即根据查询特性，选取恰当的空间索引及文本索引，构建空间文本混合索引组织查询；利用索引的空间和文本过滤能力，采用合适的过滤策略，快速为数据流中的对象过滤大量不可能匹配的查询，提高二者的匹配效率[1-13]. 利用已有索引技术组织 CkQST 时，主要有以下两个挑战.

(1) 在空间过滤方面，求解 CkQST 需要判断对象落入了哪些查询的空间搜索范围内. 空间索引 R-tree、Quad-tree 等通常被用来组织 CQST[1-13]，查询的空间搜索范围可以对应多个节点集合，不同的集合对应的空间索引的过滤能力及更新代价也不同. 通常空间索引拥有强的过滤能力与低的更新代价是矛盾的. 对于 CkQST，随着符合文本匹配条件的对象的到来及到期，其空间搜索范围随之改变. 如何映射 CkQST 的空间搜索范围，使对应的索引既有强的过滤能力、又有低的更新代价是求解 CkQST 首先要解决的问题. 图 1 中，t_0 时刻 q_1 的空间搜索范围 C_1 可映射为下列任意一个覆盖 C_1 的非重叠节点集：$\{n_0\}$、$\{n_1,n_2,n_3,n_4\}$、$\{n_1,n_2,n_3,n_{17}\}$、$\{n_5,n_6,n_7,n_8,n_2,n_3,n_4\}$、$\{n_1,n_9,n_{11},n_{13},n_{14},n_{17}\}$. 若映射为$\{n_0\}$，仅需更新$\{n_0\}$中相应的倒排列表，但是落入$\{n_0\}$的对象均需要与 q_1 比对验证，即索引更新代价小，过滤能力弱. 若映射为$\{n_1,n_9,n_{11},n_{13},n_{14},n_{17}\}$，需要更新 6 个节点中相应的倒排列表，但是这 6 个节点覆盖的区域小于$\{n_0\}$覆盖的区域，可以过滤更多的对象，索引过滤能力强，但是更新代价高. 若对象服从均匀分布，后者的验证代价比前者减少 7/16，但是后者的索引更新代价至少是前者的 6 倍. 因此，为 CkQST 的空间搜索范围与空间节点集合找到一个最优关联是一个具有挑战性的问题. 随着数据流规模的增加，这一问题更加突出.

提高 CQST 空间过滤能力主要有两种途径：① 将查询的空间搜索范围映射到索引结构的最小区域，以减少需要验证的查询数量[1-6]. 这一途径适用于查询空间搜索范围很少变化的情况. 查询空间搜索范围的频繁更新会导致高昂的索引重建代价；② 将查询的空间搜索范围映射到哈希桶或网格[14-16]可以避免索引的重建，但是失去了查询及对象的空间分布信息，在验证对象与查询是否匹配时，需要逐一验证桶内所有查询. 对于 CkQST，采用第一种途径会导致高昂的索引更新代价，采用第二种途径会降低索引的过滤能力，增加验证代价. 两者都不能高效地支持 CkQST 的求解.

(2) 在文本过滤方面，求解 CkQST 需要验证对象包含了哪些查询的全部关键字. 倒排索引通常被用来组织 CQST[1-2,7-10]，包含共同文本的多个查询被插入该文本对应的列表. 若对象包含该文本，则验证列表中查询的其它文本是否与对象的文本匹配. 影响倒排索引性能的主要因素是被频繁使用的文本对应的倒排列表会很长，造成验证(遍历)该列表内查询的时间长. 目前解决该问题主要有三种途径：① 比较查询包含文本对应的列表，将查询插入到最短的列表，即低频文本对应的列表，以调整各列表中的查询数量[1-2]. 由于查询数量庞大，包含低频文本的查询依然很多，列表可能依然很长；② 增加文本划分深度，如有序关键字字典树[2-4]. 由于索引计算复杂度高、更新代价大，不适用于节点内查询频繁更新的场景；③ 倒排列表中的查询按相似度排序[8]，但是 CkQST 中没有对应的相似度概念. 以上三种途径都不能高效支持 CkQST 的文本过滤. 此外，当包含相同文本的对象频繁到来时，该文本对应的倒排列表在短时间内被频繁访问，形成性能瓶颈.

根据 CkQST 查询特性，本文采用 Quad-tree 作为空间索引，在每个 Quad-tree 节点，集成一个有序倒排索引，以适应查询空间搜索范围的动态变化，称其为 OIQ-tree. 为解决上述挑战，本文提出一个内存代价模型，综合考虑索引的过滤能力及更新代价，将 CkQST 空间搜索范围映射到一组最佳空间节点集. 在空间节点内，采用基于块的有序倒排索引，以快速定位需要验证的查询，避免对倒排列表中大量不可能匹配查询的访问；支持对象的批量验证，提高对象吞吐量. 本文的创新点如下：

(1) 提出内存代价模型 VUMBCM(Verification

and Update of Memory-Based Cost Model). VUMBCM 综合考虑索引中查询的验证代价及更新代价，自适应地把 CkQST 的空间搜索范围映射到具有不同粒度的 Quad-tree 节点集合，解决索引频繁更新导致的效率问题.

(2) 为了提高对象与查询的验证效率，在 Quad-tree 节点采用基于块的有序倒排索引组织查询的文本，以减少需要验证的查询数量；采用对象批量处理技术，实现多个包含共同文本的对象共享对倒排列表的访问，减少对倒排列表的访问次数.

(3) 真实数据集上的实验表明，OIQ-tree 及其适配的策略具有高效性及可扩展性，能够很好地支持大规模数据流上的 CkQST 求解.

本文第 2 节介绍求解 CkQST 的相关工作；第 3 节形式化 CkQST；第 4 节介绍 OIQ-tree 的关键技术及相应的对象处理方法；第 5 节介绍本文的实验，并对实验结果进行分析；第 6 节对本文工作进行总结.

2　相关工作

求解 CkQST，一方面需要选择恰当的空间文本混合索引组织 CkQST，另一方面需要运用恰当的过滤策略，尽可能地提高索引的过滤能力，降低对象与查询的验证代价及索引更新代价. 本节依次讨论现有组织 CQST 的索引、支持查询空间搜索范围到空间节点的映射策略及支持对象批量处理的文本索引策略.

组织 CQST 的索引. 根据对象与 CQST 空间搜索范围的匹配模式，可将组织 CQST 的索引分为两类：(1) 支持在指定空间搜索范围内匹配对象的索引[1-6]，用于连续布尔范围查询. IQ-tree[1] 由 Quad-tree 和 Ranked key 倒排列表构建，把查询的空间搜索范围映射到 Quad-tree 的多个节点，批量更新，以减少查询插入或删除导致的索引更新代价. FAST[2] 由空间金字塔及自适应关键字索引构建，根据查询包含关键字的频率决定将查询插入 Ranked key 倒排列表还是有序关键字字典树. Ap-tree[3] 及 Ap-tree$^+$[4] 由网格和有序关键字字典树构建，把一组查询划分为空间节点或文本节点，以适应查询的空间及文本分布. 混合索引[5] 由网格和 OpIndex[17] 构建，利用网格组织查询范围以适应查询移动. R^t-tree[6] 由 R-tree 和倒排索引构建，在 R-tree 各层节点记录其后代节点内查询的一个或多个关键字，以增加节点的文本过滤能力；(2) 支持在整个空间区域内匹配对象的索引[7-11]，用于连续近似 Top-k 查询. 混合索引[7] 由 R-tree 和倒排索引构建，在 R-tree 节点中集成的倒排列表带有空间邻近度下界属性. 混合索引[8] 由 Quad-tree 和倒排索引构建，在每个节点的倒排列表中，查询根据偏好参数分组，且组内按相似度升序排列. 混合索引[9] 及 IQ*-tree[10] 由 Quad-tree 和倒排索引构建，在每个节点，按照查询与节点的空间邻近度将查询划分到多个桶，在桶中建立基于块的倒排索引，每个块维护一个最低文本相似度. RI-tree[11] 由 R-tree 和动态区间树构建. 为了进一步加快数据流中对象处理，Wang 等人[8]、Chen 等人[12] 和 Mahmood 等人[13] 研究了分布式服务器集群上 CQST 查询负载分配优化算法. 此外，支持空间搜索范围可变的索引[14-16]，用于连续空间 k 近邻查询. 这类索引将查询的空间搜索范围映射到哈希桶或网格，不考虑文本因素.

上述索引利用空间索引(Quad-tree[1,8-10]、金字塔[2]、网格[3-5] 或 R-tree[6-7,11] 等)组织查询范围，在空间节点内利用文本索引(Ranked key 倒排列表[1-2]、有序关键字字典树[2-4]、倒排索引[7-10] 或区间树[11] 等)组织查询关键字，并引入过滤策略，以尽快处理到来的对象[1-13]. 对于空间索引，Quad-tree、网格及金字塔是空间驱动型索引，节点的边界不依赖于数据的空间分布，迭代地将空间区域划分为多个均匀、不重叠的区域，数据更新不会引起索引结构的调整；R-tree 是数据驱动型索引，叶节点按照数据的空间范围或位置构建，非叶节点由叶节点迭代构建，叶节点的更新会引起整个 R-tree 结构的更新. 对于文本索引，倒排索引及 Ranked key 倒排列表中的数据是无序的，当对列表中的数据进行验证时，需要逐一验证；有序关键字字典树中的数据是有序的，只需要验证包含文本项的分支，与倒排索引相比，索引构建复杂，更新代价大.

CkQST 检索包含全部指定关键字的对象，需要为每个 CkQST 维护一个合适的空间搜索范围，以保证既可找到 k 个满足条件的对象，又不需要验证过多的对象. 这样，CkQST 的空间搜索范围随着满足文本约束的对象的到来及到期而动态变化. 因此，支持 CkQST 的混合索引既要有高效的过滤能力，又要有较低的索引更新代价.

支持查询空间搜索范围到空间节点的映射策略. 根据上述讨论，求解 CkQST 是在动态变化的空间搜索范围内验证到来的对象是否满足文本约束，因

此,如何组织 CkQST 动态更新的空间搜索范围是关键.

在空间索引选择上,多粒度空间索引 R-tree 和 Quad-tree 支持空间范围重组. R-tree 叶节点更新会导致从叶节点到根节点的更新,Quad-tree 则不会,因此,本文利用 Quad-tree 组织 CkQST 的空间搜索范围.

在查询空间搜索范围到空间节点集合映射方面,IQ-tree[1]通过一个代价模型将倒排列表中的查询的空间搜索范围映射到磁盘的一组 Quad-tree 节点集,以减少查询插入或删除导致的索引更新 I/O 代价,索引更新频率低. FAST[2]采用空间金字塔组织查询的空间搜索范围,从根节点开始,根据包含的文本为查询寻找插入节点,将包含低频文本的查询插入到较低层的少量区域较大的节点中,将只包含高频文本的查询插入到较高层的多个区域较小的节点中.该索引顺序插入新提交的查询不考虑查询空间范围的整体分布. Ap-tree[3]及 Ap-tree+[4]根据查询的空间分布将一组查询划分到 f 个空间网格. 三类索引在组织连续布尔范围查询时都表现了良好的性能.但是其映射是一次性的,不考虑查询空间搜索范围的动态更新.如果对其组织的查询空间搜索范围频繁更新,会导致索引结构不断重建.本文利用 Quad-tree 节点多粒度、同一粒度节点非重叠的特性,根据提出的代价模型 VUMBCM,为 CkQST 的空间搜索范围自适应地选取最优的 OIQ-tree 节点,在索引过滤能力和更新代价之间取得平衡.

支持对象批量处理的文本索引策略. 对到来对象分批处理是降低查询验证代价的有效途径. 自适应关键字索引[2]按照预定顺序将查询插入 Ranked key 倒排列表或者有序关键字字典树. 节点内的查询的文本分布发生变化会导致查询在 Ranked key 倒排列表和有序关键字字典树间反复调整,因此不适用于节点内查询频繁更新的场景. Ap-tree[3]及 Ap-tree+[4]采用有序关键字字典树组织查询,将查询划分到 f 个有序文本区间,多个对象遍历同一文本区间时,可批量处理对象.但是由于其迭代地为节点内查询寻找最优划分,所需要的时间代价为文本数量的平方.此外,节点内查询关键字分布发生变化时,需要重新划分文本区间,不适用于节点内查询频繁更新的场景. Wang 等人[8]在空间节点内的倒排列表中,将查询按相似度升序排列,可批量处理对象,但是不适用于精确的 k 近邻查询.

此外, Wu[18]、Huang[19]、Guo[20]、Zheng[21]、Salgado[22]、Oh[23]等人研究空间文本移动连续查询,使查询在移动过程中也能得到满足约束条件的结果,如车主在汽车行驶过程中查找心仪的加油站.这类连续查询专注于为移动的查询或对象建立安全区域.

本文对空间节点内倒排列表中的 CkQST 按包含的文本排序,配合分块策略,解决包含相同文本的对象的批量处理问题.

3 问题定义

本节形式化 CkQST.

定义 1. 空间文本对象. 数据流上一个空间文本对象表示为 $o=(loc,\psi,t_e)$,其中 $o.loc$ 表示对象的地理位置,用经纬度表示,$o.\psi$ 表示对象的关键字,是一组属于词汇集 $\mathcal{V}$ 的文本集合,$o.t_e$ 表示对象的到期时刻.数据流上空间文本对象集记为 $\mathcal{O}$.

定义 2. 空间文本数据流上连续 k 近邻查询. 一个 CkQST 表示为 $q=(loc,\psi,k,t_e)$,其中 $q.loc$,$q.\psi$ 及 $q.t_e$ 遵循与空间文本对象类似的意义,分别表示查询 q 的地理位置,关键字及到期时刻,$q.k$ 表示查询检索对象的数量. 在 $q.t_e$ 时刻前,持续为 q 计算 k 个包含指定关键字的对象集,记为 $q(\mathcal{O})=\{o\in\mathcal{O}\mid o.\psi\supseteq q.\psi\wedge dist(o,q)\leq q.kdist\}$,这里 $dist(o,q)$ 表示 o 与 q 的欧氏距离,$q.kdist$ 表示当前时刻 q 与第 k 个最邻近结果的距离,被称为搜索半径. 为查询 q 实时维护一个圆心为 $q.loc$,半径为 $q.kdist$ 的圆形空间搜索范围,记为 $q.SR_k$.

定义 2 中的“连续查询”是相对于一次查询(Ad-hoc query)而言的. 一次查询只计算提交时刻的结果. 连续查询是用户预订(subscribe)的、在一段时期内有效的查询,在有效期内,随着数据流中对象的到来,系统需要及时更新查询结果.

在应用程序中,空间文本对象是商家发布的促销广告或者网站上发布的最新突发新闻,而 CkQST 是用户提交的搜索请求. 在下文中,如果没有歧义,“对象”指空间文本对象,“查询”指 CkQST,“k”指 $q.k$,“数据流”指空间文本数据流. 为了算法处理方便,将词汇集 $\mathcal{V}$ 中的文本按字母表顺序从小到大依次映射为 0 到 $|\mathcal{V}|-1$ 之间的整数,并用 w 表示. 查询和对象包含的文本集有序,并使用以下符号:$q.\psi[i]$ 表示 $q.\psi$ 中第 i 个关键字,$q.\psi[i:j]=\bigcup_{i\leq l\leq j} q.\psi[l]$ 表示 $q.\psi$ 中从第 i 个到 j 个关键字间的所有关键字构

成的集合，$q.\psi[:i]=\bigcup_{1\le l\le i} q.\psi[l]$ 表示 $q.\psi$ 中从第 1 个关键字到第 i 个关键字间的所有关键字构成的集合，$q.\psi[i:]=\bigcup_{i\le l\le |q.\psi|} q.\psi[l]$ 表示 $q.\psi$ 中自第 i 个关键字之后的所有关键字构成的集合. 对象 o 遵循类似的符号.

问题陈述. 给定海量的 CkQST 集 $\mathcal{Q}$ 及不断更新的空间文本数据流 $\mathcal{O}$，为每个 CkQST 在空间文本数据流上实时检索最多 k 个、最邻近的、且包含其全部关键字的对象.

根据前面章节的讨论，本文 CkQST 由 OIQ-tree 组织，以快速处理数据流中到来的对象. 对象由 IL-Quadtree[24] 组织，以快速查找 CkQST 的 k 个最邻近的结果. 表 1 列出了本文中常用的符号.

表 1　符号

符号	描述
$q(q.loc,q.\psi,q.k,q.t_e)$	CkQST(描述 q 的地理位置、关键字、检索对象的数量及到期时刻)
$o(o.loc,o.\psi,o.t_e)$	数据流中的对象(描述 o 的地理位置、关键字及到期时刻)
$q(\mathcal{O})(q.kdist,q.SR_k)$	q 的结果集(搜索半径及空间搜索范围)
$\mathcal{Q},\mathcal{O}$	CkQST 集及数据流中的对象集
$q.\psi[:i],q.\psi[i:j],q.\psi[i:]$	q 包含的关键字子集
$\|q.\psi\|,\|o.\psi\|$	q 及 o 包含的关键字数量
N,n	OIQ-tree 的节点
$N.R,Area(N)$	节点 N 的空间区域及区域的面积
$PL_{w_{i1}w_{i2}}$	关键字 w_{i1},w_{i2} 对应的倒排列表
$\|\mathcal{B}\|$	倒排列表包含的块数量
b,b_r	倒排列表中的块
$b_r.\psi$	b_r 块中查询的第三个关键字集
$b_r.\min w,b_r.\max w$	块中查询第三个关键字的最小、最大值
$\|b\|$	块 b 中的查询数量
$\theta_{\Delta w}$	块关键字区间划分长度
$\mathcal{V}(\|\mathcal{V}\|)$	词汇集合(词汇集中的文本数量)
$C_V(q,N)$	q 关联节点 N，q 的验证代价
$p_o(N)$	对象落入节点 N 的概率
$p_V^w(w_j)$	N 中前三个关键字为 w_{i1},w_{i2},w_j 的查询被验证的概率
$E_V(q\|N)$	q 插入 N 中，查询的验证代价期望值
$\Delta C_V,\Delta C_U$	查询映射到当前节点与映射到其子节点的验证代价及更新代价差值
$N.height,\theta_h$	节点 N 所在的树高及 OIQ-tree 的高度

4　OIQ-tree

本节讨论 OIQ-tree——一个高效组织 CkQST 的索引. 4.1 节介绍 OIQ-tree 索引；4.2 节提出有序倒排索引；4.3 节介绍查询空间搜索范围映射到 OIQ-tree 节点的内存代价模型 VUMBCM；对象处理算法在 4.4 节给出；4.5 节给出索引构建及维护算法.

4.1　OIQ-tree 索引

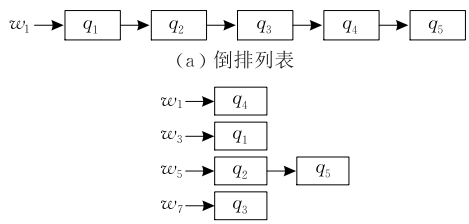
Quad-tree 节点具有多粒度、同一粒度节点非重叠的特点，OIQ-tree 利用 Quad-tree 组织查询的空间搜索范围，以适应该范围的动态变化；在 Quad-tree 的节点中集成有序倒排索引，组织查询的关键字，从而构成空间文本混合索引.

查询空间搜索范围的组织. OIQ-tree 利用 Quad-tree 组织查询的空间搜索范围，需要将该范围映射到 Quad-tree 的一组节点，即将查询插入到这组节点的倒排索引中，这种映射被称为关联.

定义 3.　关联. 给定任意查询 q 及一组节点集 $NS=\{n_i,\cdots,n_j\}$，NS 中所有节点的空间区域的并集覆盖查询空间搜索范围，即 $\bigcup_{n_l\in NS} n_l.R\supseteq q.SR_k$，节点集内任意两个节点 n_{i1},n_{i2} 空间区域不重叠，即 $n_{i1}.R\cap n_{i2}.R=\varnothing$，且与查询空间搜索范围有交集，即 $q.SR_k\cap n_{i1}.R\neq\varnothing$. 若将 q 插入到节点 $n_i,\cdots,n_j$ 的倒排索引中，则称 q 关联节点集 NS. $n_i.R$ 表示节点 n_i 的空间区域.

图 1 中，t_0 时刻 q_1 的空间搜索范围为 C_1，节点集 $NS=\{n_1,n_9,n_{11},n_{13},n_{14},n_{17}\}$ 满足定义 3 中的条件，即 NS 中所有节点的空间区域的并集覆盖 C_1，任意两个节点的空间区域均不重叠，任意节点空间区域均与 C_1 有交集，故 q_1 可关联节点集 NS. 通过第 1 节关于求解 CkQST 的挑战的讨论可知，查询与 Quad-tree 节点的关联是一个具有挑战性的问题.

查询关键字的组织. 在空间节点中，OIQ-tree 利用基于块的有序倒排索引组织查询. 在倒排索引中，关键字对应的倒排列表中的数据项通常是无序的. 为了便于验证到来的对象是否包含查询的全部关键字，查询只插入到其中 1 个关键字对应的倒排列表中. 图 2 是对图 1 中的 5 个查询建立的倒排列表，图 2(a)是将查询插入到第 1 个关键字对应的倒排列表，图 2(b)是将查询插入到低频关键字对应的倒排列表，即 Ranked key 倒排列表. 当对象到来时，需要为其验证相应倒排列表中所有的查询，其中大部分查询由于对象不包含它的其余关键字而与对象不匹配.



(a) 倒排列表

(b) Ranked key倒排列表

图 2　倒排列表示例

OIQ-tree 使用有序倒排列表，避免对列表中所有查询的验证. 有序倒排列表中的查询按照包含的关键字升序排列，在验证该列表时，一旦对象的关键字集小于当前查询关键字集，就可以终止验证. 图 1 中的 5 个查询插入到第 1 个关键字对应的倒排列表中，则有序倒排列表如图 3 所示. 新对象 o_6 到来时，验证图中列表，o_6 的关键字集小于 q_2 的关键字集，故当验证到 q_2 时，终止该列表的验证.

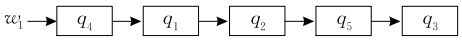


图 3 单关键字构建有序倒排列表示例

4.2 有序倒排索引

验证对象与查询是否匹配，对象与查询关键字的比对是验证的主要代价，如何利用有序倒排索引减少这种比对次数是提高评估 CkQST 性能的关键. 本文从构建有序倒排索引的关键字数量和在倒排列表中快速定位需要验证的查询两个方面，讨论提升对象与查询验证效率的途径.

4.2.1 思路

构建有序倒排索引的关键字数量不同，索引的验证代价及存储代价也不同. 若构建有序倒排索引采用查询的一个关键字，倒排列表数量及其占用存储空间少，但是倒排列表长，验证代价大. 若构建有序倒排索引采用查询的全部关键字，对象与需要验证的倒排列表中的查询都满足文本匹配条件，不需要文本比对，但是倒排列表数量及占用存储空间急剧增长. 若词汇集包含 100 万个文本项，查询包含的关键字数量不超过 5 个，倒排列表数量为 $10^{6\times5}$！因此，需要权衡验证代价与存储代价. 4.2.3 节给出构建有序倒排索引的关键字数量对在索引上进行查找、插入、删除操作的性能的影响，5.2 节通过实验验证这一影响. 在此基础上本文选定采用两个关键字构建有序倒排索引.

根据上述讨论可知，倒排列表中只有少部分查询与对象匹配，快速定位列表中需要验证的查询是另一个提高对象与查询验证效率的途径. 在有序倒排列表中，查找、插入、删除查询代价与列表长度相同. 但是若将该列表分为多个块，则对列表的操作转变为对其中某个块的操作. 为了尽快定位到需要验证的查询，本文将有序倒表列表中的查询按块组织，实现验证只发生在块内，从而提高对象与查询的验证效率.

按照上面讨论的思路，下面给出分块有序倒排索引的形式化定义，讨论构建有序倒排索引的关键字数量与索引的查找、插入、删除操作的性能及存储代价的关系，从而证明分块有序倒排索引可以支持 CkQST 关键字的高效验证.

4.2.2 分块有序倒排索引

本节给出采用两个关键字构建的有序倒排列表的形式化定义，即利用查询的前两个关键字构建有序倒排列表，列表中所有查询按照包含的关键字升序排列.

定义 4. 有序倒排列表/有序倒排索引. 给定节点 n 中一组待插入查询 $q_1,q_2,\cdots,q_i$，这些查询的前两个关键字均为 w_{i1},w_{i2}，对这组查询按照包含的关键字从小到大排序，并依次插入倒排列表中，称该倒排列表为有序倒排列表，记为 $PL_{w_{i1}w_{i2}}$. 特别地，若查询只包含一个关键字，则其插入的倒排列表记为 $PL_{w_{i1}w_{i1}}$. 利用有序倒排列表组织节点中的查询，得到节点下的有序倒排索引. 在下文中，如果没有歧义，“列表”指有序倒排列表.

为了快速定位列表中需要验证的查询和加快列表的构建，把列表分块. 按照查询包含的第三个关键字，定位查询所在的块.

定义 5. 块. 给定节点 n 中有序倒排列表 $PL_{w_{i1}w_{i2}}$ 及待插入查询 q，$q.\psi[1:2]=\{w_{i1},w_{i2}\}$. 若将 q 插入 $PL_{w_{i1}w_{i2}}$ 中的第 $r(r\geq0)$ 个块，记为 b_r，仅当 $q.\psi[3]\in((r-1)\cdot\theta_{\Delta w},r\cdot\theta_{\Delta w}]$（即 $r=\lceil q.\psi[3]/\theta_{\Delta w}\rceil$）. 其中，在 b_0 块中插入只包含 $\{w_{i1},w_{i2}\}$ 的查询，称 $\theta_{\Delta w}$ 为块关键字区间划分长度. 此外，记块 b_r 中第一个查询的第三个关键字为 $b_r.\min w$（该值为块中查询的第三个关键字的最小值，即 $b_r.\min w=\min\limits_{q_i\in b_r}q_i.\psi[3]$）. 类似地，记块 b_r 中最后一个查询的第三个关键字为 $b_r.\max w$（该值为块中查询的第三个关键字的最大值，即 $b_r.\max w=\max\limits_{q_i\in b_r}q_i.\psi[3]$）. 定义 $b_r.\psi$ 为关键字集 $b_r.\psi=\{w|w\in[b_r.\min w,b_r.\max w]\}$.

例 1. 令 OIQ-tree 树高为 3，块关键字区间划分长度 $\theta_{\Delta w}=4$. 图 1 中 $q_1,\cdots,q_5$ 构成的 OIQ-tree 如图 4 所示. q_1,q_2,q_3 关联根节点 $\{n_0\}$，q_4 关联节点

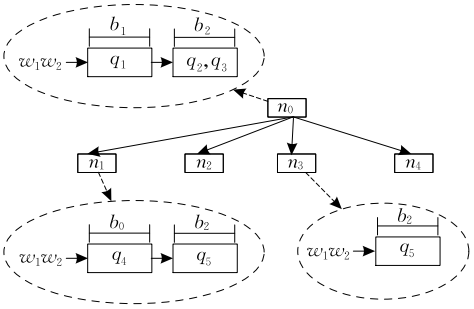


图 4 OIQ-tree 示例

$\{n_1\}$，q_5 关联节点 $\{n_1, n_3\}$. 在节点 n_0 中，倒排列表 $PL_{w_1w_2}$ 包含两个块 b_1 及 b_2，b_1 中的查询满足 $1=\lceil q.\psi[3]/4\rceil$，且 $b_1.\min w=b_1.\max w=w_3$，且 $b_1.\psi=\{w_3\}$. 类似地，$b_2.\min w=w_5$，$b_2.\max w=w_7$，$b_2.\psi=\{w_5, w_6, w_7\}$.

在有序倒排列表中，查询按照关键字分块排列. 在验证对象与查询是否匹配时，可根据倒排列表的块关键字区间划分长度，找到需要验证的块(见定理 1). 在验证列表中的块时，若对象包含的关键字集小于块中某查询的关键字集，则可终止对该列表的验证；若对象包含的关键字集大于块中所有查询的关键字集，则可终止对该块的验证(定理 2).

定理 1.　给定节点 n 及落入该节点的对象 o，对 o 中任意两个关键字 w_{i1}，w_{i2} ($w_{i1}<w_{i2}$)，如果在 n 中存在倒排列表 $PL_{w_{i1}w_{i2}}$，且在该列表中存在块 b_r 中的查询 q，o 是 q 的查询结果，则必然 $\exists j$，$o.\psi[j]>w_{i2}$，$r=\lceil o.\psi[j]/\theta_{\Delta w}\rceil$.

证明.　若在 $o.\psi\backslash\{w\in o.\psi\mid w\le w_{i2}\}$ 中不存在文本满足 $r=\lceil o.\psi[j]/\theta_{\Delta w}\rceil$，即不存在文本满足 $(r-1)\cdot\theta_{\Delta w}<o.\psi[j]\le r\cdot\theta_{\Delta w}$. 但是根据定义 5，对 $\forall q_i\in b_r$，$(r-1)\cdot\theta_{\Delta w}<q_i.\psi[3]\le r\cdot\theta_{\Delta w}$，故 o 至少不包含 b_r 中所有查询的第三个关键字，o 不是 b_r 中查询的结果. 与假设矛盾，定理成立.　　证毕.

定理 2.　给定节点 n 及落入该节点的对象 o，对 o 中任意两个关键字 w_{i1}，w_{i2} ($w_{i1}<w_{i2}$)，如果在 n 中存在倒排列表 $PL_{w_{i1}w_{i2}}$，则验证 o 与该列表中任意块 b_r 中的查询是否匹配，以下结论成立.

(1) 如果 b_r 中查询的第三个关键字的最小值大于 o 中所有关键字，即 $b_r.\min w>\max\{w\in o.\psi\}$，则 o 不可能是从 b_r 开始的块中任意查询的结果，终止对 $PL_{w_{i1}w_{i2}}$ 的验证. 特别地，若 $\exists q_i\in b_r$，$q_i.\psi[3]>\max\{w\in o.\psi\}$，则终止对 $PL_{w_{i1}w_{i2}}$ 的验证.

(2) 如果 b_r 中查询的第三个关键字的最大值小于 o 中剩余待验证的最小的关键字，即 $b_r.\max w<\min\{w\in o.\psi\mid w>w_{i2}\}$，则 o 不可能是 b_r 块中任意查询的结果，终止 o 对 b_r 的验证.

证明.　(1) 如果 $b_r.\min w>\max\{w\in o.\psi\}$，则有 $b_r.\min w=\min\limits_{q_i\in b_r}q_i.\psi[3]>\max\{w\in o.\psi\}$，即对 $\forall q_i\in b_r$，o 不包含 q_i 的第三个关键字，故 o 不是 b_r 块中查询的结果，同理对 b_r 块之后的任意块 b_{r+j} 都有 $b_{r+j}.\min w>b_r.\min w>\max\{w\in o.\psi\}$，故提前终止对 $PL_{w_{i1}w_{i2}}$ 的验证. 类似地，对 $\forall q_i\in b_r$，满足 $q_{i+j}.\psi[3]>q_i.\psi[3]>\max\{w\in o.\psi\}$，$o$ 不可能是 q_i 后任意查询的结果，终止对 $PL_{w_{i1}w_{i2}}$ 的验证.

(2) 如果 $b_r.\max w<\min\{w\in o.\psi\mid w>w_{i2}\}$，即 $b_r.\max w=\max\limits_{q_i\in b_r}q_i.\psi[3]<\min\{w\in o.\psi\mid w>w_{i2}\}$，$o$ 不可能是 b_r 块中查询的结果.　　证毕.

4.2.3　构建有序倒排索引的关键字数量的选取

本节分析当构建有序倒排索引的关键字数量不同时，索引存储代价及查找、插入、删除代价的变化.

存储代价. 利用 Hash 结构实现有序倒排索引，且 Hash 函数值唯一，则索引需要占用一块连续的、很大的内存空间. 当索引中倒排列表数量很多时，这是比较困难的. 同其它工作[1-13]，本文采用 VS 中的 Map 结构实现有序倒排索引. 构建有序倒排索引的关键字数量越多，倒排列表的数量越多，占用的存储空间越大. 5.2 节中实验表明，当查询数量为 500 万，索引采用两个关键字构建比采用单个关键字构建多占用 1.5 GB 内存.

查找、插入、删除代价. 在有序倒排列表中进行查找、插入、删除操作，性能在最坏的情况下相同，都需要遍历倒排列表中的一个或多个块. 以对象查找匹配的查询为例，分析倒排列表采用不同数量的关键字构建时的性能优劣. 在空间节点中处理对象分为三步：找到需要验证的倒排列表集；在相应倒排列表中找到需要验证的块；在所有块中找到需要验证的查询进行验证.

定理 3.　给定节点 n 及落入该节点的对象 o，若采用 m ($m\ge1$) 个关键字构建有序倒排索引，则节点内最多有 $|\mathcal{V}|^m$ 个有序倒排列表，这里 $|\mathcal{V}|$ 表示词汇集 $\mathcal{V}$ 中的文本数量. 若用 $|Q|$ 表示验证的倒排列表中 b_0 块中的查询数量，则 o 在 n 中查找匹配查询的代价(简称查找代价)有以下结论.

(1) 若对象包含一个关键字，查找代价为 $O(\log|\mathcal{V}|^m+|Q|)$.

证明.　假定 $o.\psi=\{w_{i1}\}$，对象访问由 w_{i1} 确定的倒排列表，查找代价为 $O(\log|\mathcal{V}|^m+|Q|)$. 这里 Q 表示只包含关键字 w_{i1} 的查询集合.　　证毕.

(2) 若对象包含两个关键字，查找代价为 $O(3\log|\mathcal{V}|^m+|Q|)$. 特别地，当采用单个关键字构建有序倒排索引时，查找代价为 $O(2\log|\mathcal{V}|+\log|\mathcal{B}|+|b|+|Q|)$.

证明.　假定 $o.\psi=\{w_{i1}, w_{i2}\}$，对象访问由 w_{i1}，w_{i2} 确定的倒排列表，可分为三种情形，由 w_{i1} 确定的倒排列表，由 w_{i2} 确定的倒排列表及由二者确定的倒排列表，故查找代价为 $O(3\log|\mathcal{V}|^m+|Q|)$. 这

里 Q 表示最多只包含文本 w_{i1} 和 w_{i2} 的查询集合. 若采用单个关键字构建有序倒排索引,只会访问由 w_{i1} 和 w_{i2} 分别确定的倒排列表,且在 w_{i1} 确定的倒排列表中访问两个块,一个块内查询只包含 w_{i1},另一个块内查询除包含 w_{i1} 外,可能包含 w_{i2},故查找代价为 $O(2\log|\mathcal{V}|+\log|\mathcal{B}|+|b|+|Q|)$. 这里 Q 表示只包含文本 w_{i1} 或 w_{i2} 的查询集合. 证毕.

(3) 若对象包含的关键字数量大于等于三,查找代价为 $O\left(|o.\psi|^m\cdot\left(\log|\mathcal{V}|^m+|o.\psi|\cdot\left(\log|\mathcal{B}|+\frac{|b|}{(|\mathcal{B}|\cdot|b.\psi|)^{m-1}}\right)\right)+|Q|\right)$.

证明. 若对象包含的关键字数量大于等于三,且对象的每个关键字都映射在倒排列表不同的块中,访问的倒排列表可分为三类:第一类由少于 $m(m>1)$ 个关键字确定的倒排列表,查找代价为 $\sum_{i=1}^{m-1}\binom{i}{|o.\psi|}\cdot(\log|\mathcal{V}|^m+|Q|)$,这里 Q 表示最多只包含 o 的 $m-1$ 个关键字的查询集合;第二类由 m 个关键字确定的倒排列表,且包含对象最后一个关键字 $o.\psi[|o.\psi|]$,查找代价为 $\binom{m-1}{|o.\psi|-1}\cdot(\log|\mathcal{V}|^m+|Q|)$,这里 Q 表示只包含 o 的 m 个关键字,且其中一个是 $o.\psi[|o.\psi|]$ 的查询集合;第三类由 m 个关键字确定,且不包含对象的最后一个关键字,则这样的倒排列表共有 $\binom{m}{|o.\psi|-1}$ 个,每个倒排列表的查找代价为 $O\left(\log|\mathcal{V}|^m+|o.\psi|\cdot\left(\log|\mathcal{B}|+\frac{|b|}{|\mathcal{B}|^{m-1}\cdot|b.\psi|^{m-1}}\right)\right)$,这里 Q 表示只包含 o 的 m 个关键字,且不包含 $o.\psi[|o.\psi|]$ 的查询集合. 故所需要的查找代价为 $O\left(|o.\psi|^m\cdot\left(\log|\mathcal{V}|^m+|o.\psi|\cdot\left(\log|\mathcal{B}|+\frac{|b|}{(|\mathcal{B}|\cdot|b.\psi|)^{m-1}}\right)\right)+|Q|\right)$. 这里 Q 表示最多只包含 o 的 m 个关键字的查询集合. 证毕.

由定理 3 可知,若采用一个关键字构建有序倒排索引,对象可快速定位需要验证的倒排列表,但是倒排列表的块中包含的查询数量较多,需要花费较长的时间验证块中的查询;而采用多关键字构建有序倒排索引,情形正好相反. 5.2 节中给出了构建有序倒排索引的关键字数量对索引及算法性能的影响.

4.3 内存代价模型 VUMBCM

通过第 1 节关于求解 CkQST 的挑战的讨论可知,若将 CkQST 的空间搜索范围映射到区域较大的节点,索引更新代价小,但是查询的验证代价较大;反之,若映射到多个区域较小的节点,查询的验证代价变小,但是索引更新代价较大. 如图 1 中,t_0 时刻 q_1 的空间搜索范围 C_1 映射节点集 $\{n_0\}$ 及 $\{n_1,n_9,n_{11},n_{13},n_{14},n_{17}\}$ 时,查询的验证代价及索引的更新代价区别很大. 因此,需要一个代价模型平衡二者,为查询的空间搜索范围寻找最优关联节点集. 为此,本节提出内存代价模型 VUMBCM. 给定任意查询的空间搜索范围,该模型从 OIQ-tree 根节点开始,依次计算查询关联当前节点及其子节点集的验证代价及索引更新代价,选择具有最小验证代价及索引更新代价和的节点集作为查询的关联节点集. 其中,验证代价由验证的对象数量与单次查询的验证代价期望值乘积得到,更新代价由查询插入倒排列表相应块中的更新代价期望值得到. 下面是相关形式化定义和证明.

定义 6. 最小外界节点. 给定任意查询 q 及当前时刻的空间搜索范围 $q.SR_k$,若存在节点 N,满足 $q.SR_k\subseteq N.R$,且对 N 的所有子节点 n_i 都有 $q.SR_k\not\subset n_i.R$,则称 N 为查询 q 的最小外界节点. 其中 $N.R$ 及 $n_i.R$ 表示节点 N 及 n_i 的空间区域. 在图 1 中,q_1 的最小外界节点为根节点 n_0,即只有 n_0 能覆盖 q_1 的空间搜索范围,而其任意子节点都不能.

(1) 查询验证代价

给定查询 q 及最小外界节点 N,若 q 关联 N,则固定时间间隔内,查询 q 的验证代价记为 $C_V(q,N)$,用式(1)估计. 不失一般性,假定 q 及到来对象包含的关键字数量均大于 2,且 $q.\psi[1:3]=\{w_{i1},w_{i2},w_{i3}\}$,验证对象与查询是否匹配平均代价为单位时间.

$$C_V(q,N)=Num_o\times p_o(N)\times p_V^q(q|N)\times E_V(q|N) \tag{1}$$

其中 Num_o 表示固定时间间隔内数据流上到来对象的数量. $p_o(N)$ 为对象落入节点 N 的概率. 若数据流中对象的地理位置服从均匀分布,则有 $p_o(N)=Area(N)/Area(root)$,其中 $Area(N)$ 和 $Area(root)$ 分别表示节点 N 的空间区域面积和整个空间区域的面积. 若对象地理位置不服从均匀分布,在每个节点中维护一个值,记录该节点中落入对象的数量,记为 $Num_o^N(N)$. $p_V^q(q|N)$ 表示 q 插入节点 N 中被验证的概率. 假定 q 插入 N 的 b_r 块中,则验证对象与查询是否匹配,对象需要包含 q 的前两个关键字及 $b_r.\psi$ 中自 q 开始的查询的第三个关键字,用式(2)来估计.

$$p_V^q(q|N)=p_V^B(b_r)\times\frac{b_r.\max w-w_{i3}+1}{|b_r.\psi|} \quad (2)$$

这里 $p_V^B(b_r)$ 表示块 b_r 被验证的概率. 若倒排列表为空，则直接用节点内包含 $\{w_{i1},w_{i2},w_{i3}\}$ 的对象数量占比估计；若倒排列表 $PL_{w_{i1}w_{i2}}$ 非空，则 $p_V^B(b_r)$ 用 $\max\limits_{w_j\in b_r.\psi} p_V^w(w_j)+\frac{1}{|b_r.\psi|-1}\left(\sum\limits_{w_j\in b_r.\psi}p_V^w(w_j)-\max\limits_{w_j\in b_r.\psi}p_V^w(w_j)\right)$ 估计，$p_V^w(w_j)$ 表示落入节点 N 中的对象包含 $\{w_{i1},w_{i2},w_j\}$ 的概率，即前三个关键字为 $\{w_{i1},w_{i2},w_j\}$ 的查询被验证的概率. $|b_r.\psi|$ 表示 $b_r.\psi$ 中文本项的数量. 在式(1)中，$E_V(q|N)$ 表示 q 插入节点 N 中的验证代价期望值，用 b_r 块中所有查询的验证代价期望值估计，见式(3).

$$E_V(q|N)=\sum_{w_j\in b_r.\psi}p_V^w(w_j)\times(Num_q)_{\le w_j} \quad (3)$$

这里 $(Num_q)_{\le w_j}$ 表示从块中第一个查询开始到前三个关键字为 $\{w_{i1},w_{i2},w_j\}$ 的查询为止的查询数量.

类似地，若查询 q 关联节点集 NS，则验证代价记为 $\sum\limits_{n_i\in NS}C_V(q,n_i)$，用式(4)估计.

$$\sum_{n_i\in NS}C_V(q,n_i)=Num_o\times\sum_{n_i\in NS}p_o(n_i)\times p_V^q(q|n_i)\times E_V(q|n_i) \quad (4)$$

对查询 q，从最小外界节点开始，迭代寻找最优关联节点集. 在每次迭代中，选择查询是关联于当前节点还是关联于其子节点. 二者的验证代价差值记为 ΔC_V，通过式(5)估计.

$$\Delta C_V=\sum_{n_j\in INS\cup n}C_V(q,n_j)-\sum_{n_j\in INS\cup n.child}C_V(q,n_j) \quad (5)$$

其中 INS 是迭代过程中得到的中间结果. 初始时 $INS=\varnothing$，n 是最小外界节点，$n.child$ 是 n 的子节点，且其与查询的空间搜索范围交集不为空. 如果 $\Delta C_V\le 0$，终止迭代.

(2) 索引更新代价

在 OIQ-tree 中插入或删除一个查询时，需要更新查询关联的所有节点，产生索引更新代价. 值得注意的是，若需要删除某查询，在节点中标记该查询，只有其再次被访问时才删除，故本文忽略查询的删除代价. 在 OIQ-tree 节点中插入一个查询，首先需要找到查询待插入的块，再在块中找到待插入的位置进行插入. 若查询 q 关联最小外界节点 N，则查询只需要插入到节点 N 的列表 $PL_{q.\psi[1]q.\psi[2]}$ 中，更新代价记为 $C_U(q,N)$，用式(6)来估计.

$$C_U(q,N)=\log|\mathcal{B}|+\frac{1}{|b_r|+1}\sum_{i=1}^{|b_r|+1}i=\log|\mathcal{B}|+\frac{|b_r|}{2}+1 \quad (6)$$

这里 $|\mathcal{B}|$ 表示查询所属倒排列表的总块数，b_r 为 q 待插入块，$|b_r|$ 表示块 b_r 中的查询数量. 假定 q 插入块中每个位置的概率相同，均为 $\frac{1}{|b_r|+1}$，则用 $\frac{1}{|b_r|+1}\sum\limits_{i=1}^{|b_r|+1}i$ 来估计 q 插入块中的代价.

如果查询 q 关联节点集 NS，则索引更新代价用式(7)来估计.

$$\sum_{n_i\in NS}C_U(q,n_i)=\sum_{n_i\in NS}\left(\log|\mathcal{B}_i|+\frac{|b_{r_i}|}{2}+1\right) \quad (7)$$

查询关联当前节点与关联其子节点的索引更新代价差值记为 ΔC_U，用式(8)来估计.

$$\Delta C_U=\sum_{n_j\in INS\cup n.child}C_U(q,n_j)-\sum_{n_j\in INS\cup n}C_U(q,n_j) \quad (8)$$

为了平衡查询的验证代价及索引的更新代价，引入一个规一化参数 $\theta_U(0<\theta_U\le 1)$，表示更新操作与验证操作的权重比，即查询一经插入，至少会被验证 $1/\theta_U$ 次. 如果 $\Delta C_V\ge\theta_U\cdot\Delta C_U$，将查询关联节点 n 的一组子节点集，否则，关联节点 n.

给定查询 q 及其空间搜索范围 $q.SR_k$，函数 $ChooseNodeSet(N,q)$ 从 q 的最小外界节点开始，在其后代节点上迭代计算 ΔC_V 及 ΔC_U，为 q 选择一组最优的关联节点集. 函数首先判断 q 的最小外界节点 N 的高度是否达到阈值 θ_h，若是，则关联该节点(1 行). 若否，在其后代节点中寻找最优关联节点(2～13 行). 定义变量 NS 为最终关联节点集，初始化为 $\varnothing$，队列 $NodeQueue$ 存放与 $q.SR_k$ 相交的非空节点，并按照与查询的距离按从远到近排序，初始化为最小外界节点 N(2 行). 若 $NodeQueue$ 非空，第一个节点(记为 n)出队，判断 n 的高度是否达到阈值 θ_h，若是，则直接关联该节点(4～6 行)；若否，得到与 $q.SR_k$ 相交的子节点集 nsc(7～10 行). 计算查询关联当前节点 n 与关联其子节点集的验证代价及更新代价的差值即 ΔC_V 及 ΔC_U(11 行). 如果 $\Delta C_V\ge\theta_U\cdot\Delta C_U$，更新代价较小，查询关联子节点集，将子节点集加入节点队列中，否则，关联该节点(12～13 行).

计算复杂度. 函数 $ChooseNodeSet(N,q)$ 的主要计算代价是在 q 的最小外界节点的后代节点中计算 C_V 及 C_U. 为 OIQ-tree 节点中的列表及块维护相关参数，计算 C_V 及 C_U 需要 $O(1)$. 因此函数的计算复杂度为 $O((4^{\theta_h}-1)/3)$. 其中 θ_h 为 OIQ-tree 的高度，即在最坏的情况下需要访问所有节点.

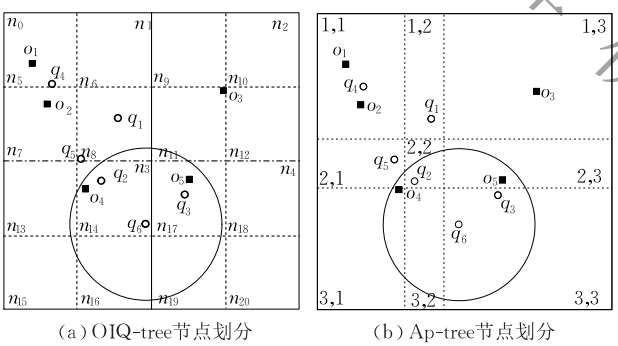
函数1. 选择关联节点集函数*ChooseNodeSet*(N,q).

输入:q,查询;N,最小外界结点

输出:覆盖 $q.SR_k$ 的最优非重叠的结点集

1. IF $N.height \geq \theta_h$ THEN RETURN N;
2. $NS \leftarrow \varnothing$; $NodeQueue \leftarrow N$;
3. WHILE $NodeQueue$!=NULL
4. $n \leftarrow NodeQueue[1]$;
5. IF $n.height \geq \theta_h$ THEN
6. $NS \leftarrow n$; CONTINUE;
7. $nsc \leftarrow \varnothing$;
8. FOR $i=0;3$
9. IF $n.child[i].R \cap q.SR_k \neq \varnothing$ THEN
10. $nsc \leftarrow n.child[i]$;
11. 计算 ΔC_V,ΔC_U;
12. IF $\Delta C_V \geq \theta_U \cdot \Delta C_U$ THEN $NodeQueue \leftarrow nsc$;
13. ELSE $NS \leftarrow n$;
14. RETURN NS;

例 2. 在图 1 的示例中,假定 t_8 时刻,新查询 q_6 预订,地理位置在图 5 中标出,包含的关键字为 $\{w_1, w_2, w_6\}$. 首先确定其空间搜索范围(图中的圆形区域),利用代价模型为其寻找最优关联节点集.



(a) OIQ-tree节点划分 (b) Ap-tree节点划分

图 5 q_6 空间搜索范围的映射

q_6 的最小外界节点为根节点即 n_0. 取 $\theta_U=0.5$. 初始时,令 $NS=INS=nsc=\varnothing$,NS,INS 及 nsc 如上文所述. 若将 q_6 关联根节点 n_0,则其应插入 $PL_{w_1w_2}$ 的 b_2 中(见图 4). 此时落入 n_0 中的对象共有 5 个,故 $Num_o^N(n_0)=5$;在 b_2 中,$b_2.\psi=\{w_5, w_6, w_7\}$,落入 n_0 中对象包含 $\{w_1, w_2, w_5\}$ 的概率为 $p_V^w(w_5)=2/5$,类似地有 $p_V^w(w_6)=2/5$,$\max\limits_{w_j \in b_r.\psi} p_V^w(w_j)=p_V^w(w_7)=3/5$,故 $p_V^B(b_2)=1$,故 $p_V^q(q_6 \mid n_0)=p_V^B(b_2)\times\frac{b_2.\max w-w_6+1}{|b_2.\psi|}=\frac{2}{3}$. 在 b_2 块中,从第一个查询开始到前三个关键字为 $\{w_1, w_2, w_5\}$ 的查询数量为 $(Num_q)_{\leq w_5}=1$,类似地有 $(Num_q)_{\leq w_6}=2$,以及 $(Num_q)_{\leq w_7}=3$,故 q_6 的验证代价期望值为 $E_V(q_6 \mid n_0)=\sum\limits_{w_j \in b_2.\psi} p_V^w(w_j)\times(Num_q)_{\leq w_j}=3$;$C_V(q_6, n_0)=Num_o\times p_o(n_0)\times p_V^q(q_6 \mid n_0)\times E_V(q_6 \mid n_0)=10$,$C_U(q_6, n_0)=\log|\mathcal{B}|+\frac{|b_2|}{2}+1=\log 2+2=3$.

若将 q_6 关联 n_0 的子节点集,则 $nsc=\{n_1, n_2, n_3, n_4\}$,$q_6$ 应插入每个节点的 $PL_{w_1w_2}$ 的 b_2 中. 对节点 $n_i \in nsc$,若 n_i 内查询待插入的倒排列表或块为空,则有 $p_V^q(q_6 \mid n_i)=E_V(q_6 \mid n_i)=p_V^w(w_6)$. 故对 n_2 及 n_4,有 $C_V(q_6, n_2)=0$,$C_V(q_6, n_4)=1$,$C_U(q_6, n_2)=1$,$C_U(q_6, n_4)=1$. 在 n_1 中,落入 n_1 的对象数量为 $Num_o^N(n_1)=2$,b_2 块被验证的概率为 $p_V^B(b_2)=1/2$,查询被验证的概率为 $p_V^q(q_6 \mid n_1)=1/4$,q_6 的验证代价期望值为 $E_V(q_6 \mid n_1)=1/2$,故 $C_V(q_6, n_1)=1/4$,$C_U(q_6, n_1)=3/2$(注意,更新时 b_0 单独更新,故非 b_0 块数为 1). 在 n_3 中,落入 n_3 的对象数量为 $Num_o^N(n_3)=1$,b_2 块被验证的概率为 $p_V^B(b_2)=1$,查询被验证的概率为 $p_V^q(q_6 \mid n_3)=1/2$,q_6 的验证代价期望值为 $E_V(q_6 \mid n_3)=3$,故 $C_V(q_6, n_3)=3/2$,$C_U(q_6, n_3)=3/2$. 故 $\sum\limits_{n_i \in nsc} C_V(q_6, n_i)=11/4$,$\sum\limits_{n_i \in nsc} C_U(q, n_i)=5$,则有 $\Delta C_V=\sum\limits_{n_i \in INS \cup n_0} C_V(q_6, n_i)-\sum\limits_{n_i \in INS \cup nsc} C_V(q_6, n_i)=7.25$,$\Delta C_U=\sum\limits_{n_i \in INS \cup nsc} C_U(q_6, n_i)-\sum\limits_{n_i \in INS \cup n_0} C_U(q_6, n_i)=2$,$\Delta C_V \geq \theta_U \cdot \Delta C_U$,将 $\{n_1, n_2, n_3, n_4\}$ 加入到节点队列中,继续划分.

优先划分离查询地理位置较远的节点,故优先划分 n_2,此时 $INS=\{n_1, n_3, n_4\}$. n_2 中只有 n_{11} 与 q_6 查询空间搜索范围相交,故 $nsc=\{n_{11}\}$,而 $C_V(q_6, n_{11})=C_V(q_6, n_2)$,且 $C_U(q_6, n_{11})=C_U(q_6, n_2)$,故 $\Delta C_V=\Delta C_U=0$,n_{11} 替换 n_2,且 n_{11} 节点高度达到最大,不再划分,$NS=\{n_{11}\}$;继续划分 n_1,此时 $INS=\{n_{11}, n_3, n_4\}$. n_1 中只有 n_8 与 q_6 查询空间搜索范围相交,故 $nsc=\{n_8\}$,n_8 中没有对象,故 $C_V(q_6, n_8)=0$,$C_U(q_6, n_8)=1$,$\Delta C_V=0.25$,$\Delta C_U \leq 0$,故 n_8 替换 n_1,且 n_8 高度达到最大,$NS=\{n_{11}, n_8\}$;继续划分 n_4,此时 $INS=\{n_8, n_{11}, n_3\}$. n_4 中 n_{17} 和 n_{19} 与 q_6 搜索范围相交,故 $nsc=\{n_{17}, n_{19}\}$,$C_V(q_6, n_{17})=C_V(q_6, n_4)$,$C_V(q_6, n_{19})=0$,故 $\Delta C_V=0$,$C_U(q_6, n_{17})=C_U(q_6, n_{19})=1$,$\Delta C_U=1$,故取 n_4,$NS=\{n_{11}, n_8, n_4\}$;继续划分 n_3,与 n_4 类似,故最终 $NS=\{n_{11}, n_8, n_4, n_3\}$. 按照 IQ-tree[1] 的节点映射策略,利用查询的验证次数来替换原代价模型中的磁盘 I/O 数,则 q_6 关联根节点,虽然更新代价最小,但是随着数据流中对象不断到来,其验证代价在不断增大;按照 Ap-tree[3] 及 Ap-tree$^+$[4] 中的节点映射策略,q_6 映射为网格

$\{c_{2,1},c_{2,2},c_{2,3},c_{3,1},c_{3,2},c_{3,3}\}$，如图 5(b)所示. 若 q_6 查询空间搜索范围变大或变小，需要调整索引结构，以使查询验证代价最小.

4.4　对象处理算法

在短时间间隔内，对访问同一倒排列表的多个对象，按照其访问的块及块中的关键字进行分组，采用组匹配技术，批量处理多个对象. 定义数据结构 $\langle bid,\langle w,oidset\rangle\rangle$，一个数据项表示访问倒排列表某块中某关键字的对象集. 这里 bid 表示倒排列表中的块号，w 表示对象包含的关键字，对任意 bid，有多个 $\langle w,oidset\rangle$ 项，其中 $\lceil w/\theta_{\Delta w}\rceil=bid$，$\langle w,oidset\rangle$ 表示包含 w 的对象集为 $oidset$. 为了下文描述方便，用 $wset_{bid}$ 表示由 bid 确定的关键字集，用 $oidset_{bid,w}$ 表示由 bid 及 w 确定的对象集.

算法 1 描述了访问倒排列表 $PL_{w_{i1}w_{i2}}$ 的一组用 $\langle bid,\langle w,oidset\rangle\rangle$ 表示的对象集处理算法，该算法遵循过滤-验证策略，利用 4.2 节中的定理 1、定理 2 提前终止一个块或者一个倒排列表的验证. 首先，若 b_{bid} 为 b_0，则验证对象集 $oidset$ 内对象是否落入 b_0 中查询的空间搜索范围内，若是，则对象是查询的结果，将查询对象对添加到 QOS 中，返回 QOS，算法执行结束(1～6 行). 若 b_{bid} 不为 b_0，令由块 b_{bid} 确定的关键字集为 $wset_{bid}$，则对 $wset_{bid}$ 中的关键字 w_j，按照 4.2 节定理 2，若 $b_{bid}.\min w>w_j$，判断下一个关键字(8 行)；若 $b_{bid}.\max w<w_j$，该块终止验证(9 行). 否则需要依次验证块中每个查询 q，若 $q.\psi[3]>w_j$，判断下一个关键字(11 行)；若 $q.\psi[|q.\psi|]<w_j$，对象不可能是 q 的结果；否则判断对象集中的对象是否包含了查询的全部关键字，且落入查询的空间搜索范围内，若都满足，则将查询对象对添加到 QOS 中(13～15 行). 此外，对所有返回的查询对象对 $\langle q,o\rangle$，更新 q 的搜索半径；添加新项 $\langle dist(q,o),o\rangle$ 到 q 的结果列表，并删除结果列表中距离最远的项；检查缩小后的 q 的空间搜索范围是否与其关联节点都相交，在不相交的节点中标记删除 q；在相交的节点及其子节点内寻找 q 的最优关联节点.

计算复杂度. 算法 1 描述了在节点的有序倒排列表内处理一组对象的过程. 最坏情况下，对象只能单独处理. 4.2 节中描述了单个对象在一个节点内查找匹配查询集的时间复杂度为 $O(|o.\psi|^2\cdot(\log|\mathcal{W}|^2+|o.\psi|\cdot(\log|\mathcal{B}|_{\max}+|b|_{\max})))$，其中 $|\mathcal{V}|$，$|o.\psi|$ 如上所述，$|\mathcal{B}|_{\max}$ 是倒排列表中包含的最多块数，$|\mathcal{B}|_{\max}\le\lceil|\mathcal{V}|/\theta_{\Delta w}\rceil$，$|b|_{\max}$ 为块中包含的最多查询数. 单个对象在所有覆盖该对象的非空节点集中过滤-验证查询的时间复杂度是 $O(\theta_h\cdot|o.\psi|^2\cdot(\log|\mathcal{W}|^2+|o.\psi|\cdot(\log|\mathcal{B}|_{\max}+|b|_{\max})))$，其中 θ_h 为 OIQ-tree 的高度，即在最坏的情况下需要访问从根节点到叶节点的所有覆盖对象的节点数.

算法 1.　对象处理算法.

输入：$PL_{w_{i1}w_{i2}}$，需要访问的倒排列表；$\langle bid,\langle w,oidset\rangle\rangle$，按块及关键字分组的对象集

输出：QOS，查询对象对 $\langle q,o\rangle$，其中 o 是 q 的 k 近邻结果

```
1.  IF bid==0&&b0!=NULL THEN
2.    FOR ∀o∈oidset
3.      FOR ∀q∈b0
4.        IF dist(o,q)≤q.kdist THEN
5.          QOS←⟨q,o⟩;
6.    RETURN QOS;
7.  FOR wj∈wset_bid
8.    IF b_bid.minw>wj THEN CONTINUE;
9.    IF b_bid.maxw<wj THEN BREAK;
10.   FOR ∀q∈b_bid
11.     IF q.ψ[3]>wj THEN BREAK;
12.     IF q.ψ[|q.ψ|]<wj THEN CONTINUE;
13.     FOR ∀o∈oidset_{bid,wj}
14.       IF dist(o,q)≤q.kdist &&o.ψ⊇q.ψ THEN
15.         QOS←⟨q,o⟩;
16. RETURN QOS;
```

4.5　索引构建及维护

当新查询提交时，首先在对象索引 IL-Quadtree[24] 中为其寻找 k 近邻结果，并确定空间搜索范围，最后利用算法 2 将其插入 OIQ-tree 中.

算法首先查找 q 的最小外界节点(1 行)，并利用函数 $ChooseNodeSet()$ 计算一组最优关联节点集 NS(2 行). 对于 NS 中的每个节点 n，将 q 插入相应的倒排列表中(3～23 行). 如果 q 只包含一个关键字，则直接插入倒排列表 $PL_{q.\psi[1]q.\psi[1]}$ 的 b_0 块中(4～8 行). 如果 q 包含多于一个关键字，则首先检查当前节点倒排列表 $PL_{q.\psi[1]q.\psi[2]}$ 是否存在，若不存在，构建倒排列表(9～10 行). 如果 q 包含两个关键字，则将其直接插入倒排列表 $PL_{q.\psi[1]q.\psi[2]}$ 的 b_0 块中(11～15 行). 否则，需要依据 $q.\psi[3]$ 在 $PL_{q.\psi[1]q.\psi[2]}$ 中找到映射块 b_r(16 行). 如果该块不存在，则构建，并将 q 插入块 b_r 中(17～19 行). 否则，遍历块 b_r 中每个查询 q_i，找到第一个满足 $q_i.\psi[3:]\ge q.\psi[3:]$ 的查询 q_i，并将 q 插入到 q_i 之前(20～23 行).

算法 2.　查询插入算法.

输入：q，查询；$root$，OIQ-tree 根结点

输出：将 q 插入索引中

```
1.  N←为 q 寻找最小外界节点;
```

```
2.  NS←ChooseNodeSet(N,q);
3.  FOR n∈NS
4.    IF |q.ψ|==1 THEN
5.      IF ¬∃PL_{q.ψ[1]q.ψ[1]} THEN
6.        构建新块b0及PL_{q.ψ[1]q.ψ[1]},并将b0插入列表;
7.      将q插入PL_{q.ψ[1]q.ψ[1]}中b0块的尾部;
8.      CONTINUE;
9.    IF ¬∃PL_{q.ψ[1]q.ψ[2]} THEN
10.     构建新列表PL_{q.ψ[1]q.ψ[2]};
11.   IF |q.ψ|==2 THEN
12.     IF ¬∃b0 THEN
13.       构建新块b0,并将b0插入PL_{q.ψ[1]q.ψ[2]};
14.     将q插入PL_{q.ψ[1]q.ψ[2]}中b0块的尾部;
15.     CONTINUE;
16.   r=⌈q.ψ[3]/θ_Δw⌉;
17.   IF ¬∃br THEN
18.     构建新块br,将q插入br,br插入PL_{q.ψ[1]q.ψ[2]};
19.     CONTINUE;
20.   FOR ∀qi∈br
21.     IF qi.ψ[3:]≥q.ψ[3:] THEN
22.       将q插入到qi之前;
23.       BREAK;
```

计算复杂度. 算法 2 的计算代价主要包含三部分:寻找 q 的最小外界节点,在 OIQ-tree 中寻找最优关联节点集,并将查询插入到所有关联节点的倒排列表中. 第一部分的计算代价为 $O(\theta_h)$,第二部分的计算代价为 $O((4^{\theta_h}-1)/3)$,第三部分主要依赖于有序倒排索引的过滤性能. 在最坏的情况下,将查询插入相应的倒排列表的计算复杂度为 $O(|NS|\times(\log|\mathcal{B}|_{\max}+|b|_{\max}))$,其中 $|NS|$ 是关联节点的数量,$|b|_{\max}$,θ_h 和 $|\mathcal{B}|_{\max}$ 如上所述.

5 实验验证与结果分析

本节评估本文索引及算法的有效性和可扩展性. 5.1 节给出本文的实验环境,包括硬件平台、数据集、比较算法及评价指标;5.2 节测试四个参数(构建有序倒排索引的关键字数量(记为 m)、块关键字区间划分长度 $\theta_{\Delta w}$、查询的更新与验证操作比 θ_U 及查询检索的对象数量 k)对索引及算法性能的影响;5.3 节测试不同数据集、数据集大小、查询包含的关键字数量对索引及算法性能的影响.

5.1 实验环境

硬件平台:实验用 VS2013 实现,并运行在英特尔 i7-8700K 3.7 GHz 和 32 GB 内存的 Windows 10 机器上. 与经典的空间文本数据流系统一样[2-13],本文使用内存计算模型,将索引加载到内存中以支持实时响应.

数据集. 本文采用三类数据集进行实验评估. TWEETS 是从 Twitter 网站上收集的 2012 年 4 月至 10 月有地理标记的推特[9],是本节默认的实验数据集. GN 是从 US Board 网站上得到的美国地名,每条记录包含地理标记及较短的文本描述①. GOWALLA 是从 Gowalla 网站上收集的有地理标记的签到数据②,为每条记录随机分配少于 50 个文本项,这些文本项来自 20 Newsgroups③. 清洗所有源数据集,删除非英文字符,统一英文大小写,得到测试数据集. 三类数据集统计信息如表 2 所示.

表 2 数据集统计信息

数据集	数据集大小/万	相应文本集大小/万	每条记录包含的平均文本数
TWEETS	2000.00	179.88	8.73
GN	228.62	20.24	3.48
GOWALLA	64.43	6.12	26.00

对象流. 令数据集对应的空间区域两点间最大距离为 d. 对数据集中的每条记录,生成一个对象. 取其所有文本作为对象的关键字,将经纬度向水平方向及垂直方向移动 $0.01\%d$ 到 $1\%d$ 作为对象的地理位置,并随机使对象进入系统或从系统删除模拟真实环境对象的到来及到期.

查询集. 对数据集中的每条记录,生成一个查询. 随机取 j 个文本作为查询的关键字,其中 $1\leq j\leq 5$,取经纬度作为查询的地理位置,指定的整数值作为查询检索的对象数量 k,并随机使查询进入系统或从系统删除模拟真实环境查询的预订及删除.

比较算法. 本文查询索引及对象处理算法与 IQ-tree[1]、FAST[2] 及 Ap-tree[3] 比较. 它们都需要提前生成查询的空间搜索范围,此项工作在每次测试前进行. IQ-tree 被加载到内存,故替换原代价模型中的磁盘 I/O 数为查询的验证次数. Ap-tree 空间划分扇区大小为 200,查询划分阈值为 40,KL-散度阈值为 0.001. FAST 频繁关键字阈值为 10. 本文除了测试 OIQ-tree 索引性能外,还测试构建有序倒排索引关键字的数量对索引及算法性能的影响.

评价指标. (1)索引构建时间,指数据集中所有查询得到空间搜索范围后构建索引的时间;(2)对象平均处理时间,指对象到来时,在查询索引中查找受影响查询并修改查询结果的时间;(3)索引平均更新时间_O,指新对象到来或已有对象到期导致的

① http://geonames.usgs.gov/
② https://snap.stanford.edu/data/loc-gowalla.html
③ http://people.csail.mit.edu/jrennie/20Newsgroups

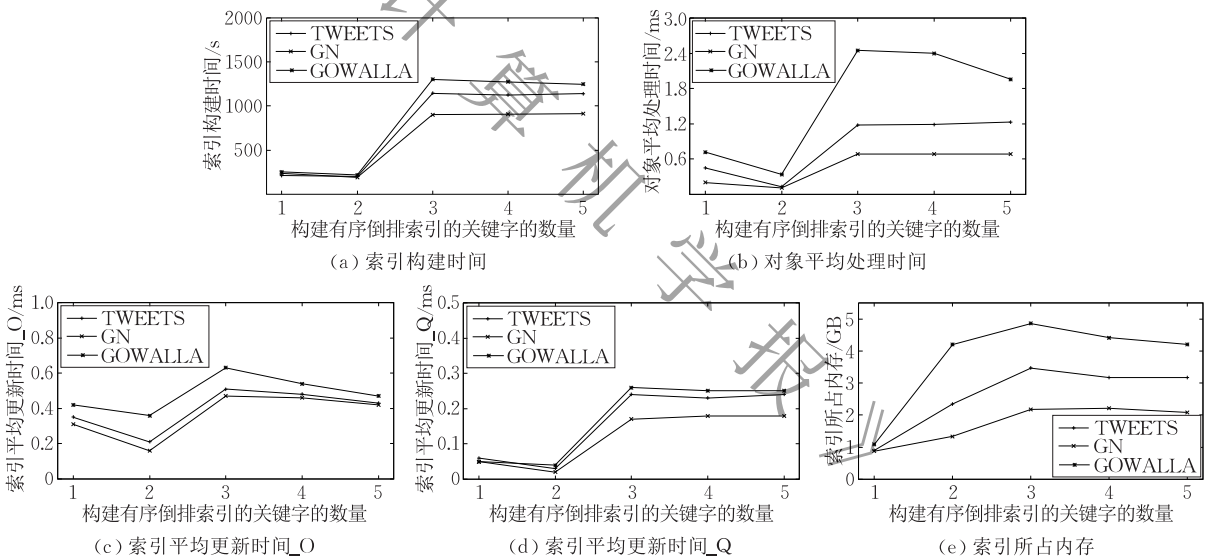
索引平均更新时间，即受影响的查询得到新的空间搜索范围后，索引的更新时间；(4) 索引平均更新时间_Q，新提交查询导致的索引更新时间，即查询得到空间搜索范围后插入索引的时间；(5) 索引大小，查询索引占用的内存.

默认参数设置. 默认情况下，查询的关键字数量为 1～5 的随机数，查询检索的对象数量为 20，块关键字区间划分长度 $\theta_{\Delta w}$ 为 1000、查询的更新与验证操作比 θ_U 为 0.001. 默认测试集：在每类数据集上生成三组测试集，每组包含 200 万对象及 200 万查询，评价指标取三组测试结果的平均值.

5.2　参数测试

在三类数据集中随机选择 500 万对象及 500 万查询构建对象索引及查询索引. 本节测试以下参数：构建有序倒排索引的关键字数量(记为 m)、块关键字区间划分长度 $\theta_{\Delta w}$、查询的更新与验证操作比 θ_U 及查询检索的对象数量 k.

构建有序倒排索引的关键字数量 m. 在三类数据集上测试 m 对索引及算法性能的影响. 图 6 描述了当 m 由 1 增加到 5 时，评价指标的变化. 三类数据集的评价指标变化趋势相似，且当 $m=2$ 时，都表现了较优的性能，因此，在下文测试中，取 m 为 2. 在 GOWALLA 数据集中，查询及对象包含的关键字较多，因此五个评价指标的值都远大于其它两类数据集. 由 4.2 节定理 3 可知，若 m 取较小值，节点内包含较少的倒排列表，但是需要花费较长的时间验证列表内的查询；相反，若 m 取较大值，则需要花费较长的时间查找需要验证的倒排列表. 这一点在图 6 中也得到了验证. 此外，m 越大，倒排列表数量越多，占用的内存越大. 值得注意的是，当 $m>3$ 时，在单个倒排列表内查询的验证代价及更新代价均变小，内存代价模型将查询映射到少量区域较大的低层节点，即查询插入的空间节点变少，故评价指标均开始减小.



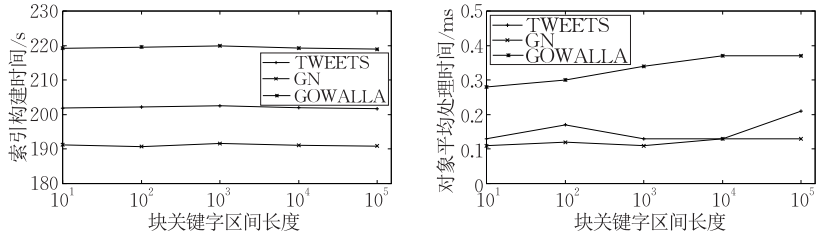
(a) 索引构建时间　(b) 对象平均处理时间　(c) 索引平均更新时间_O　(d) 索引平均更新时间_Q　(e) 索引所占内存

图 6　构建有序倒排索引关键字的数量 m 不同时评价指标的变化

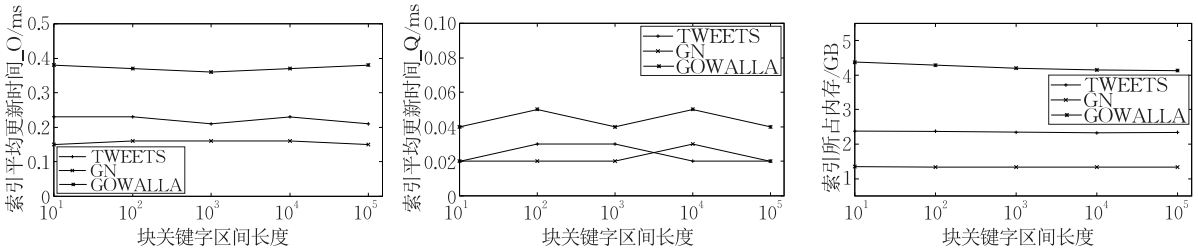
块关键字区间划分长度 $\theta_{\Delta w}$. 在三类数据集上测试 $\theta_{\Delta w}$ 对索引及算法性能的影响. 当 $\theta_{\Delta w}$ 从 10 增加到 100000 时，评价指标变化如图 7 所示. 随着 $\theta_{\Delta w}$ 的增加，评价指标均发生了轻微的变化. 这是因为若 $\theta_{\Delta w}$ 取较小值，倒排列表中包含较多的块，每个块中包含较少的查询，算法花费较长的时间定位查询所在的块；反之，若 $\theta_{\Delta w}$ 取较大值，算法花费较长的时间定位查询在块中的位置. 此外，$\theta_{\Delta w}$ 变大，列表中包含的块变少，因此索引占用内存减小.

更新与验证操作的权重比 θ_U. 在三类数据集上测试 θ_U 对索引及算法性能的影响. 图 8 描述了当 θ_U 从 0.0001 增加到 0.1 时，评价指标的变化. θ_U 取 0.0001 时，查询寻找关联节点，验证代价起主要作用，故查询关联高层的多个区域较小的节点，索引构建时间及索引平均更新时间_Q 较长，对象平均处理时间及索引平均更新时间_O 较短，索引占用内存较大；而当 θ_U 变大，索引更新操作的比重变大，因此查询被映射到低层的少量区域较大的节点，故索引构建时间及索引平均更新时间_Q 缩短，而对象平均处理时间及索引平均更新时间_O 变长，索引占用内存变小.

查询检索的对象数量 k. 在三类数据集上测试 k 对索引及算法性能的影响. 图 9 描述了当 k 从 10 增加到 50 时，评价指标的变化. 当 k 较小时，查询检索

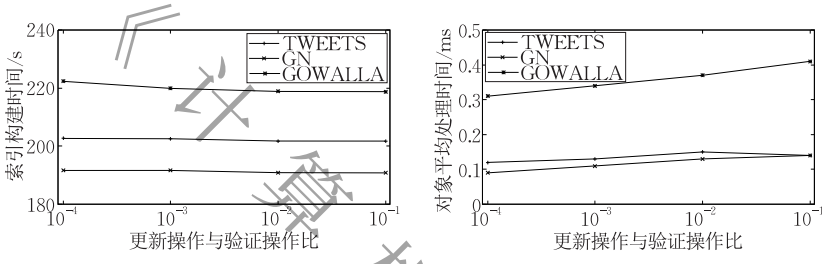


(a) 索引构建时间 (b) 对象平均处理时间

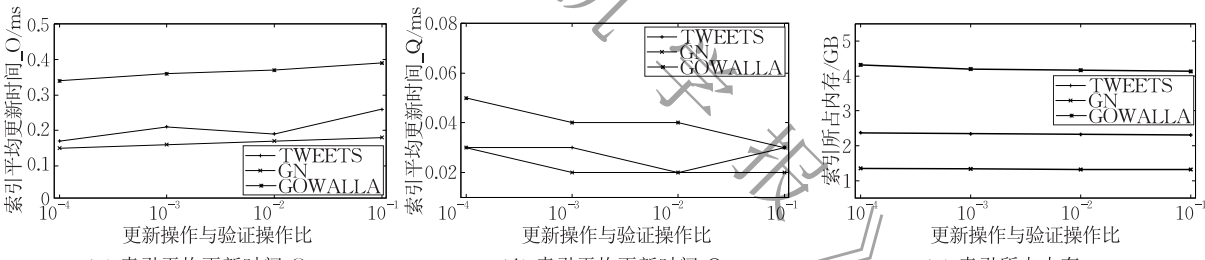


(c) 索引平均更新时间_O (d) 索引平均更新时间_Q (e) 索引所占内存

图 7 块关键字区间划分长度 $\theta_{\Delta w}$ 不同时指标的变化

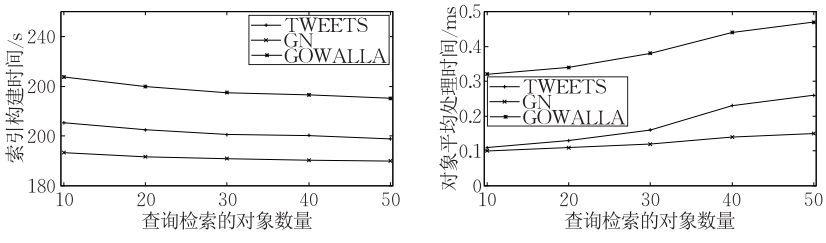


(a) 索引构建时间 (b) 对象平均处理时间

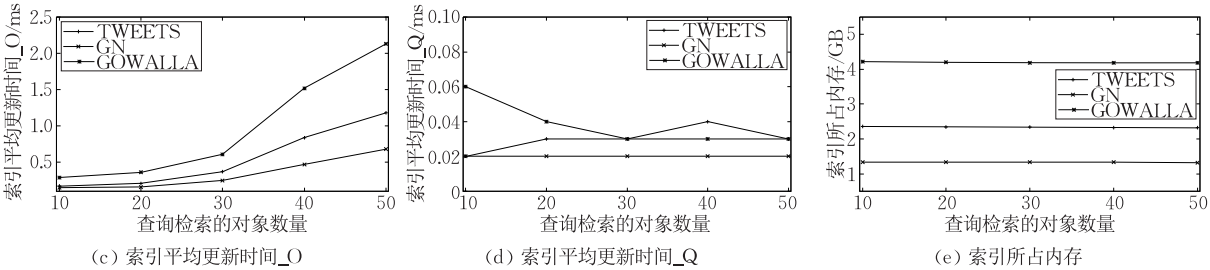


(c) 索引平均更新时间_O (d) 索引平均更新时间_Q (e) 索引所占内存

图 8 查询更新与验证操作的权重比 θ_U 不同时评价指标的变化



(a) 索引构建时间 (b) 对象平均处理时间



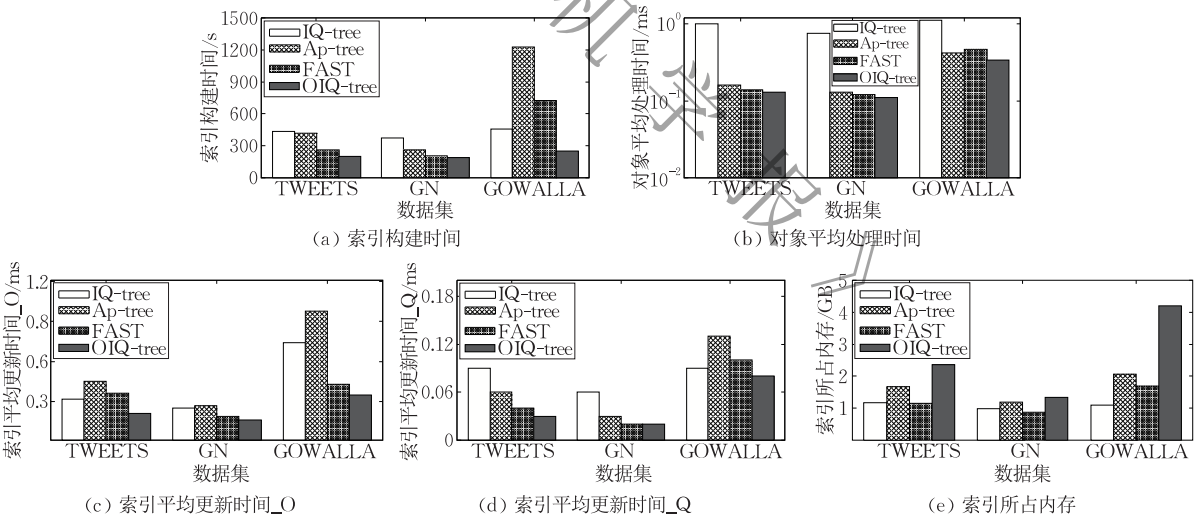
(c) 索引平均更新时间_O (d) 索引平均更新时间_Q (e) 索引所占内存

图 9 查询检索对象数量不同时评价指标的变化

的对象数量较少，查询的空间搜索范围较小，查询关联高层的多个区域较小的节点，故索引构建时间及索引平均更新时间_Q 较长，占用内存较大，而对象的平均处理时间及索引平均更新时间_O 较短；反之，当 k 变大时，查询检索的对象数量增多，查询的空间搜索范围变大，查询关联低层的少量区域较大的节点，故索引构建时间及索引平均更新时间_Q 缩短，索引占用内存减小，对象的平均处理时间及索引平均更新时间_O 增加.

5.3　性能评估

查询集. 在三类数据集上随机选择 500 万对象及 500 万查询构建对象索引及查询索引，评估 OIQ-tree 索引策略的高效性. 图 10 描述了在三类数据集上 IQ-tree、Ap-tree、FAST 及 OIQ-tree 的索引性能. OIQ-tree 的索引构建时间、对象平均处理时间、索引平均更新时间_O 及索引平均更新时间_Q 均是最短的；FAST 的索引性能与本文 OIQ-tree 最接近. IQ-tree 侧重考虑查询的空间分布，其文本采用 Ranked key 倒排列表，过滤能力较弱；Ap-tree 综合考虑查询的空间及文本分布，但是索引构建及更新较复杂；FAST 综合利用 Ranked key 倒排列表及有序关键字字典树优势，集成空间金字塔，但是其不考虑查询整体分布，依据查询预订顺序更新索引，先提交的查询更倾向于插入到低层节点，索引空间过滤能力较弱. OIQ-tree 中内存代价模型综合考虑查询的空间及文本分布，平衡索引的验证代价与更新代价，使查询均匀地插入索引空间节点，因此在节点内，需要为对象验证的查询数量适中；OIQ-tree 中有序倒排索引采用两个关键字构建，其过滤性能接近于有序关键字字典树，更新性能接近于 Ranked key 倒排列表，故 OIQ-tree 的性能较优. 但是 OIQ-tree 占用内存最大，这是由有序倒排索引的结构决定的. 在三类数据集中，GN 数据集五个评价指标值都是最小的，GOWALLA 数据集五个评价指标值都是最大的. 这是因为 GN 数据集中记录包含的文本数量较少，而 GOWALLA 数据集中记录包含的文本数量较多. 在 GOWALLA 数据集上，Ap-tree 的对象平均处理时间较 FAST 优，这是因为当数据集中大量查询只包含高频关键字时，与 FAST 相比，Ap-tree 综合考虑查询的空间与文本分布，尽可能地区分查询，即索引过滤能力更强，故对象平均处理时间较短.

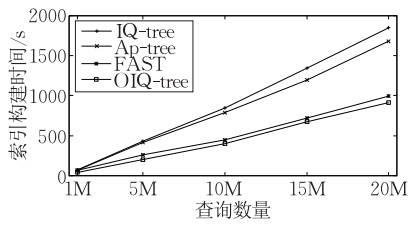


(a) 索引构建时间　(b) 对象平均处理时间　(c) 索引平均更新时间_O　(d) 索引平均更新时间_Q　(e) 索引所占内存

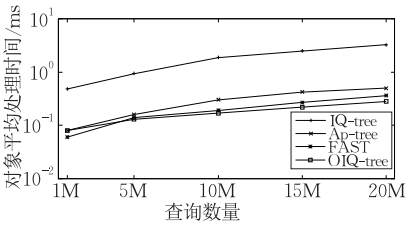
图 10　不同数据集索引性能

查询集大小. 在 TWEETS 数据集上测试索引策略的可扩展性，构建对象索引及查询索引的数据集大小从 100 万增加到 2000 万，图 11 显示了数据集变大时 IQ-tree、Ap-tree、FAST 及 OIQ-tree 评价指标的变化. 当查询数量从 100 万增加到 2000 万时，五个评价指标对应的值均增加. 数据集越大，索引节点中插入的查询越多，索引构建时间、对象平均处理时间、索引平均更新时间_O 及索引平均更新时间_Q 越长. 对象平均处理时间 OIQ-tree 最短，IQ-tree 最长. 当查询数量增加到 2000 万时，OIQ-tree 处理对象平均用时 0.28 ms，比 Ap-tree 快了 44%，比 FAST 快了 22%，说明本文索引策略及算法具有很好的可扩展性. 索引平均更新时间_O OIQ-tree 最短，Ap-tree 最长. 这是因为 OIQ-tree 中查询的关联节点集适应数据流中对象的更新；而 Ap-tree 中更新的查询增多时，需要重建部分节点，故索引更新时间_O 最长；但是随着查询数量增加，Ap-tree 按关键字划分的节点数量增加，故索引更新时间_O 增加

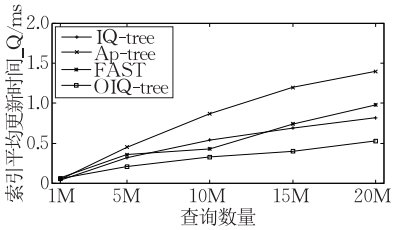
变缓;与 Ap-tree 相比,FAST 只更新节点内部分查询,因此索引更新时间_O 较短. IQ-tree 节点内插入查询不需要排序,故当查询数量不断增加时,索引更新时间_Q 没有明显的增加.



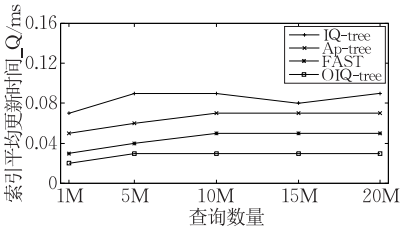
(a) 索引构建时间



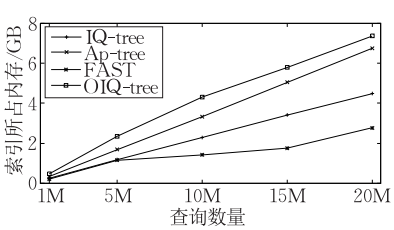
(b) 对象平均处理时间



(c) 索引平均更新时间_Q



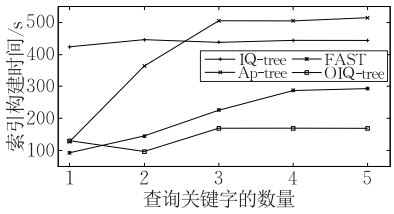
(d) 索引平均更新时间_Q



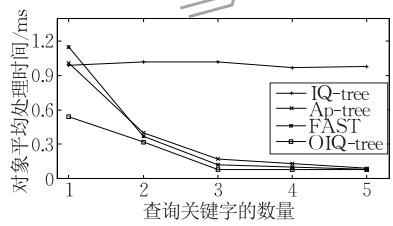
(e) 索引所占内存

图 11 TWEETS 数据集大小不同时评价指标的变化

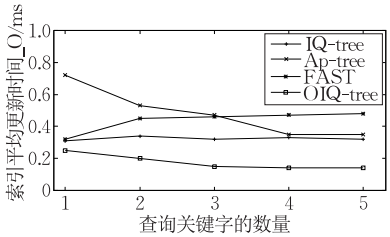
查询关键字数量 $|q.\psi|$. 在 TWEETS 数据集上测试索引策略的可扩展性,当 $|q.\psi|$ 从 1 增加到 5 时,IQ-tree、Ap-tree、FAST 及 OIQ-tree 的评价指标变化如图 12 所示. IQ-tree 侧重考虑查询的空间分布,将查询插入到其最低频的关键字对应的列表中,评价指标随 $|q.\psi|$ 的变化不敏感. 而 Ap-tree、FAST 及 OIQ-tree 均考虑查询的文本分布,因此评价指标会随着 $|q.\psi|$ 的变化而变化. Ap-tree 依据查询中包含的关键字计算索引的划分方式. $|q.\psi|$ 越大,Ap-tree 中文本节点越多,树的高度越高,各节点内包含的查询越少,因此索引构建时间及索引平均更新时间_Q 变长,占用内存增大,但是对象平均处理时间及索引平均更新时间_O 变短. 在 FAST 中,$|q.\psi|$ 越大,更多查询关键字成为高频关键字,查询更倾向于关联索引较高层节点,因此对象平均处理时间减少,但是索引平均更新时间_Q、索引构建时间及索引平均更新时间_O 增加,占用内存增大. 在 OIQ-tree 中,$|q.\psi|$ 越大,构建有序倒排索引时,查询排序需要比较的关键字越多,索引构建时间及索引平均更新时间_Q 变长;但是有序倒排索引过滤能力增强,故对象平均处理时间及索引平均更新时间_O 变短.



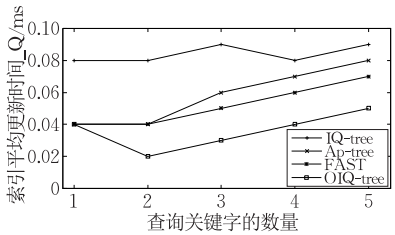
(a) 索引构建时间



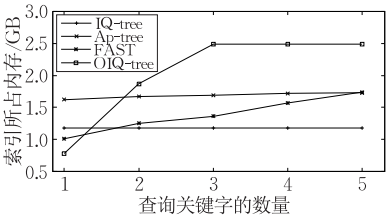
(b) 对象平均处理时间



(c) 索引平均更新时间_O



(d) 索引平均更新时间_Q



(e) 索引所占内存

图 12 查询关键字数量不同时评价指标的变化

6　结束语

在大规模空间文本数据流查询服务中，本文提出 CkQST 解决方案. 利用 OIQ-tree 组织 CkQST，通过内存代价模型 VUMBCM，将 CkQST 映射到最优关联空间节点集，以尽可能适应数据流中对象不断到来导致的查询空间搜索范围动态更新；通过基于块的有序倒排索引，快速定位需要验证的查询，避免对倒排列表中大量不可能匹配查询的访问，并实现对象的批量处理. 实验结果表明，本文索引策略及对象批处理算法具有高效性及很好的可扩展性. 在未来的工作中，将重点考虑查询的重新评估问题.

参　考　文　献

[1] Chen L S, Cong G, Cao X. An efficient query indexing mechanism for filtering geo-textual data//Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data. New York, USA, 2013: 749-760

[2] Mahmood A R, Aly A M, Aref W G. FAST: Frequency-aware indexing for spatio-textual data streams//Proceedings of the 2018 IEEE 34th International Conference on Data Engineering. Paris, France, 2018: 305-316

[3] Wang X, Zhang Y, Zhang W J, et al. AP-tree: Efficiently support location-aware publish/subscribe. The VLDB Journal, 2015, 24(6): 823-848

[4] Deng Z, Wang M, Wang L Z, et al. An efficient indexing approach for continuous spatial approximate keyword queries over geo-textual streaming data. International Journal of Geo-Information, 2019, 8(2): 57-76

[5] Guo L, Zhang D X, Li G L, et al. Location-aware pub/sub system: When continuous moving queries meet dynamic event streams//Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data. Melbourne, Australia, 2015: 843-857

[6] Li G L, Wang Y, Wang T, Feng J H. Location-aware publish/subscribe//Proceedings of the 2013 ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. Chicago, USA, 2013: 802-810

[7] Hu H Q, Liu Y Q, Li G L, et al. A location-aware publish/subscribe framework for parameterized spatio-textual subscriptions//Proceedings of the 2015 IEEE 31st International Conference on Data Engineering. Seoul, South Korea, 2015: 711-722

[8] Wang X, Zhang W J, Zhang Y, et al. Top-k spatial-keyword publish/subscribe over sliding window. The VLDB Journal, 2017, 26(3): 301-326

[9] Chen L S, Cong G, Cao X, Tan K L. Temporal spatial-keyword top-k publish/subscribe//Proceedings of the 2015 IEEE 31st International Conference on Data Engineering. Seoul, South Korea, 2015: 255-266

[10] Chen L S, Shang S. Approximate spatio-temporal top-k publish/subscribe. World Wide Web, 2019, 22(5): 2153-2175

[11] Hulawale S, Burghate B. A novel RI-tree based top-k subscription matching for location-aware publish/subscribe. International Journal of Advance Research and Innovative Ideas in Education, 2016, 2(3): 3928-3935

[12] Chen Z D, Cong G, Zhang Z J, et al. Distributed publish/subscribe query processing on the spatio-textual data stream//Proceedings of the 2017 IEEE 33rd International Conference on Data Engineering. San Diego, USA, 2017: 1095-1106

[13] Mahmood A R, Daghistani A, Aly A M, et al. Adaptive processing of spatial-keyword data over a distributed streaming cluster//Proceedings of the 2018 ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems. Seattle, USA, 2018: 219-228

[14] BÖhm C, Ooi B C, Plant C, Yan Y. Efficiently processing continuous k-NN queries on data streams//Proceedings of the 2007 IEEE 23rd International Conference on Data Engineering. Istanbul, Turkey, 2007: 156-165

[15] Xiong X P, Mokbel M F, Aref W G. SEA-CNN: Scalable processing of continuous k-NN Queries in spatio-temporal databases//Proceedings of the 2005 IEEE 21st International Conference on Data Engineering. Tokyo, Japan, 2005: 643-654

[16] Yu X H, Pu K Q, Koudas N. Monitoring k-nearest neighbor queries over moving objects//Proceedings of the 2005 IEEE 21st International Conference on Data Engineering. Tokyo, Japan, 2005: 631-642

[17] Zhang D X, Chan C Y, Tan K L. An efficient publish/subscribe index for e-commerce databases//Proceedings of the 2014 VLDB Endowment 40th International Conference on Very Large Data Bases. Hangzhou, China, 2014: 613-624

[18] Wu D M, Yiu M L, Jensen C S, Cong G. Efficient continuously moving top-k spatial keyword query processing//Proceedings of the 2011 IEEE 27th International Conference on Data Engineering. Hannover, Germany, 2011: 541-552

[19] Huang W H, Li G L, Tan K L, Feng J H. Efficient safe-region construction for moving top-K spatial keyword queries//Proceedings of the 2012 ACM International Conference on Information & Knowledge Management. Maui, USA, 2012: 932-941

[20] Guo L, Shao J, Aung H H, Tan K L. Efficient continuous top-k spatial keyword queries on road networks. GeoInformatica, 2015, 19(1): 29-60

[21] Zheng B L, Zheng K, Xiao X H, et al. Keyword-aware continuous kNN query on road networks//Proceedings of the 2016 IEEE 32nd International Conference on Data Engineering.

Helsinki, Finland, 2016: 871-882

[22] Salgado C, Cheema M A, Ali M E. Continuous monitoring of range spatial keyword query over moving objects. World Wide Web, 2018, 21(3): 687-712

[23] Oh S J, Jung H R, Kim U M. An efficient processing of range spatial keyword queries over moving objects//Proceedings of the 2018 IEEE 32nd International Conference on Information Networking. Chiang, Mai, 2018: 525-530

[24] Zhang C Y, Zhang Y, Zhang W J, Lin X M. Inverted linear Quadtree: Efficient top k spatial keyword search. IEEE Transactions on Knowledge & Data Engineering, 2016, 28(7): 1706-1721

YANG Rong, Ph. D. candidate. Her research interests include spatial data management and analysis.

NIU Bao-Ning, Ph. D., professor. His research interests include performance management of DBMSs, audio retrieval, and big data management and analysis.

Background

The continuous k nearest neighbor query over Spatial-Textual data streams (CkQST) is the core operation in numerous location-based publish/subscribe systems and has been widely used in a variety of location-based applications. The existing works aim to mainly investigate the continuous Boolean range query and top-k query. The common approach for evaluating generic continuous queries over spatial-textual data streams consists of selecting an appropriate spatial index and a textual index to form a hybrid spatial-textual index, and exploiting it with appropriate spatial and/or textual filtering strategies to process the incoming objects according to the features of queries. Compared with the continuous Boolean range query and top-k query, the search range of CkQST covering k nearest neighbor objects changes frequently with the arrival and expiration of qualified objects, which requires the index to have both strong filtering ability and low update cost, which is challenging because the existed spatial-textual indexes don't update efficiently as the dataset frequently updated. This work extends Quad-tree with an inverted index, and exploits it with some techniques to efficiently evaluate CkQST. Compared with the state of the art techniques, when the number of the subscribed queries reaches 20 million, the average index updating time caused by the incoming objects over the data streams decreases by 46%, and the average incoming objects processing time decreases by 22%.

This work is supported by the National Natural Science Foundation of China (No. 61572345). The project aims to study the query interactions in predicting query response time, i. e. when multiple queries concurrent run, we predict and evaluate the query response time. In this project, we predict and evaluate the response time of queries across multiple domains, such as audio retrieval, image retrieval and spatial-textual data retrieval. Predicting and Evaluating query response time is a fundamental issue for many tasks related to managing database systems. We have proposed some basic methods to predict and evaluate the response time of queries in database, and have done a lot of work in the fields of audio retrieval, image retrieval and spatial-textual data retrieval, and published many high quality papers in the important journals at home and abroad.