DOI: 10.3785/j.issn.1008-973X.2018.01.019

基于多线程的不确定移动对象连续 k 近邻查询

齐建鹏[1],于彦伟[1], 王创存[1], 曹磊[2], 宋鹏[1]
(1. 烟台大学 计算机与控制工程学院,山东 烟台 264005;
2. 麻省理工学院 计算机科学与人工智能实验室,马萨诸塞州 剑桥 02139)

摘 要: 针对不确定数据下的大规模连续 k 近邻查询请求,基于不确定移动对象连续 k 近邻查询的 Rate 方法,提出高效的基于多核多线程的并行查询处理框架. 根据查询对象的运动速度与相对位置确定查询请求间是否采用查询复用,确定查询复用时的距离边界. 提出密度网格扩展的多线程数据分发方法,解决了负载均衡问题,将空间位置相邻的查询请求划分到同一线程,提高查询复用率. 通过多线程间的内存共享机制,对计算过的移动对象的预测区域实现计算复用. 在大规模交通数据集上验证了所提算法的有效性与查询性能,相比传统的 Rate 方法,所提并行算法的加速比可达 37.

关键词: k 近邻查询;不确定移动对象;查询预测;查询复用;多线程
中图分类号: TP 391 **文献标志码**: A **文章编号**: 1008-973X(2018)01-0142-09

Multi-threading based continuous k-nearest neighbor queries for uncertain moving objects

QI Jian-peng[1], YU Yan-wei[1], WANG Chuang-cun[1], CAO Lei[2], SONG Peng[1]
(1. *School of Computer and Control Engineering, Yantai University, Yantai* 264005, *China*;
2. *CSAIL, Massachusetts Institute of Technology, Cambridge, Massachusetts* 02139, *USA*)

Abstract: An efficient multi-core and multi-threading based framework was proposed for searching k-NNs of large-scale queries with uncertain locations based on the continuous k-NN query method called Rate for uncertain moving objects. The velocities and locations of different query objects were used to judge whether employing query reuse and give the bound of reuse distance. The density grid based multi-threaded data partition method was proposed to resolve the problem of load balance, and neighboring queries were grouped into the same thread to improve the reusability. The obtained predicted areas of moving objects can be reused by building shared memory over multi-core and multi-threading. The experiments conducted on large scale datasets demonstrated the effectiveness and efficiency of the proposed methods, and the proposed optimized parallel method reached about 37 speed-up compared with Rate.

Key words: k-nearest neighbor query; uncertain moving object; query prediction; query reuse; multi-threading

收稿日期: 2017-05-12. **网址**: www.zjujournals.com/eng/fileup/HTML/201801019.htm
基金项目: 国家自然科学基金资助项目(61403328, 61572419, 61773331, 61703360);山东省重点研发计划资助项目(2015GSF115009);山东省自然科学基金资助项目(ZR2014FQ016);山东省高等学校科技计划项目(J17KA091);烟台大学研究生科技创新基金资助项目(YDZD1712).
作者简介: 齐建鹏(1992—),男,硕士生,从事数据挖掘并行计算研究. orcid.org/0000-0001-6150-9773. E-mail:jianpengqi@126.com
通信联系人: 于彦伟,男,博士,讲师. orcid.org/0000-0001-6941-2132. E-mail:yuyanwei@ytu.edu.cn

海量轨迹数据包含了丰富的时空特征信息，记录了人、物或事件的移动轨迹信息，体现了各类移动对象的位置变化，正在深刻地影响着人们的日常生活，在城市规划、智能交通等领域发挥了极大的作用[1-2]. 时空查询技术作为位置信息服务的重要技术支撑之一，已经在实际应用中发挥了重要作用，典型的 k 近邻查询[3]是时空数据领域广泛应用的查询类型之一，一直是学术界与工业界共同关注的研究热点[4].

面向移动对象的 k 近邻查询算法旨在查找距离给定位置或对象最近的 k 个移动对象. 连续 k 近邻查询是将 k 近邻算法应用到动态实时环境下，对移动对象进行连续查询以实现实时动态更新. 笔者等[5]面向不确定移动对象，提出针对不确定移动对象的连续 k 近邻查询处理方法，利用最近一段时间的位置数据预测移动对象在未来 I 时间区间内的可能区域，采用距离区间表示查询对象与可能区域的距离，利用基于模糊可能度判定的排序方法查找 k 近邻. 在面向海量用户查询的实际应用系统中，需要应对高并发量的查询请求需求，串行算法所带来的延迟往往是不能忍受的. 另外，针对当前常用计算平台的性能配置，传统单线程查询方案难以充分利用多核处理器的计算优势. 基于多核多线程的不确定移动对象连续 k 近邻查询处理是当前解决这一问题的主要方法.

在实际环境下常常存在一部分邻近的用户同时查询附近的移动对象，这使得查询对象在地理位置上具有较大的空间相关性. 将相距较近的查询对象分到相同组别，组内其他用户的查询请求结果往往与邻近用户的结果一致. 利用空间相关性对邻近查询对象的结果进行有效复用，可以提升大规模并发查询请求时的处理效率.

在确定数据下的 k 近邻查询方面，已有较多的研究工作[6-15]. 目前，针对确定数据的并行 k 近邻查询算法已经有所发展[13-15]. 赵亮等[14-15]结合多核多线程技术，提出在道路网下的移动对象连续 k 近邻查询算法，使用任务并行和数据并行的方法优化连续查询过程，该算法实现了 1.51～1.7 的加速比. 上述算法都是针对确定的移动对象数据，查询的是当前时刻下的 k 近邻，针对未来一小段时间内的预测问题，没有相应的解决方案.

在不确定数据下的 k 近邻查询方面，Huang 等[16]考虑不确定速度的移动对象连续 k 近邻查询问题，提出有效代价的 P^2KNN（probability-based possible-KNN）算法，在给定的查询时间区间内找出每个时刻上的 k 近邻对象. Fan 等[21]提出基于道路网的模糊连续 k 近邻查询算法，处理速度不确定的移动对象查询问题；该方法考虑了移动对象的运动状态，利用基于路网的距离区间模型计算移动对象与查询对象间的最大和最小距离，采用 vague 模糊集计算 k 近邻. 在传统的单线程环境下，对于大量位置邻近的查询请求都进行 k 近邻的计算，会造成重复查询，导致响应延迟问题.

本文针对不确定移动对象，提出高效的基于多核多线程的连续 k 近邻并行查询处理方法. 本文的主要贡献如下：1）提出面向不确定移动对象的 k 近邻查询复用方法，根据临近查询对象之间的运动速度与相对位置确定是否采用查询复用，并确定两查询对象采用复用时的距离边界. 2）提出基于密度网格扩展的多线程负载均衡数据分发方法，利用网格密度对查询请求进行划分，解决多线程并行条件下的负载均衡问题；将空间位置相邻的查询请求尽量划分到同一线程，提高查询复用率. 3）实现了多线程间的内存共享机制，对计算过移动对象的预测区域通过建立内存共享区来避免重复计算，以实现计算复用. 4）采用 Oldenburg 城市上模拟的大规模交通数据集，设计综合实验评估，验证了所提算法的有效性与查询性能.

1　问题定义

给出一些重要的定义和表示，对不确定数据的连续 k 近邻并行查询问题进行描述.

定义 1（移动对象）　一个移动对象由唯一的标识符表示，记为 o_i. o_i 的移动轨迹数据是一个多维时空位置点序列，每个位置点为五元组 (t, x, y, v_x, v_y) 表示，其中 t 为采样时间，x、y 分别为 o_i 在 t 时刻的位置坐标，v_x、v_y 分别为 o_i 在该位置时 x 和 y 方向上的速度分量.

对于采样频率不固定的移动对象 o_i，轨迹数据可以表示为轨迹序列 $s_i = \{(t_1, x_1, y_1, v_{x1}, v_{y1}), (t_j, x_j, y_j, v_{xj}, v_{yj}), \cdots (t_k, x_k, y_k, v_{xk}, v_{yk})\}$. 其中，$s_{i,j}$ 为移动目标 o_i 在时刻 t_j 时的位置，相应地，用 $s_{i,(j,k)}$ 表示 o_i 从时刻 t_j 到时刻 t_k 区间上所采集的轨迹序列. 包含 n 个移动对象的集合表示为 $DB_o = \{o_1, o_2, \cdots, o_n\}$，不确定轨迹数据集表示为 $S = \{s_1, s_2, \cdots, s_n\}$.

采用基于时间的滑动窗口 W 来处理移动对象实时的位置数据，窗口长度标记为 w，将最小的时间刻度单位定义为一个时间点，则 W 包含的时间点数 $|W| = w$.

针对基于时间的滑动窗口 W 的不确定 k 近邻查询 Query(q,I),于彦伟等[5]提出基于速度变化率的 k 近邻预测查询方法 Rate,给定一个查询点 q,通过预测周围移动对象在 I 时间区间内可能到达的区域,然后结合 vague 模糊集对 q 在下一时刻的 k 近邻进行排序. 如图 1 所示,s_i 为移动对象 o_i 的轨迹数据,包含 4 个位置点. 当 q 查询预测 $t=18$(当前时间 $t_c=17$,$I=1$)时的 k 近邻时,首先根据下式预测 o_i 在 $t=18$ 时可能到达的区域 $s_{i,18}$.

$$\left.\begin{aligned} s_{i,t_c+I} &= s_{i,t_j}+v_{i,j}(I+t_c-t_j)+\frac{1}{2}a\,(I+t_c-t_j)^2,\\ a_{i,j} &= \frac{\Delta v}{\Delta t}=\frac{v_{i,j+1}-v_{i,j}}{t_{j+1}-t_j}. \end{aligned}\right\} \tag{1}$$

当采用与当前时间临近的多个以往时间点上的速率变化时,可得一块可能到达的区域,如图 1 的阴影部分所示. Rate 方法既考虑了最近一个时刻的位置与速度的权重,又考虑了最近一段时间速度与采样时刻的变化情况. 采用多速率变化的 Rate 方法获取的可能到达区域有效刻画了不确定移动对象在 t_c+I 区间内可能存在的位置.

使用最近距离与最远距离的区间$[d,\ D]$为查询对象到移动对象间的距离. 对于查询对象 q 到两个移动对象 o_1、o_2 的距离区间,分别设为$[d_1,\ D_1]$和$[d_2,\ D_2]$. 若要获得 q 的 k 近邻,则需要比较 q 到不同移动对象的距离区间,Rate 采用 vague 模糊集方法,利用模糊可能度判定的方法排序 k 近邻. 如性质 1 所述,若 q 到 o_1 距离相比 q 到 o_2 的距离更近的概率 $p(o_1<o_2)$ 大于等于 0.5,则满足 $D_2-d_1\geqslant D_1-d_2$. 通过验证 $D_2-d_1\geqslant D_1-d_2$ 是否成立,可以判断查询对象 q 到 o_1 是否更近于 o_2. 若 o_1 与除 o_1 之外的移动对象 o_i 都计算一遍 $p(o_1<o_i)$,则可以确定 o_1 是查询对象 q 的第几近邻. 对所有移动对象检查一遍,可以获取到查询对象 q 的 k 近邻. 与 Huang 等[16]提出的 P^{2}KNN 方法相比,Rate 方法的查询正确率平均提高 18%,效率平均提升 66%.

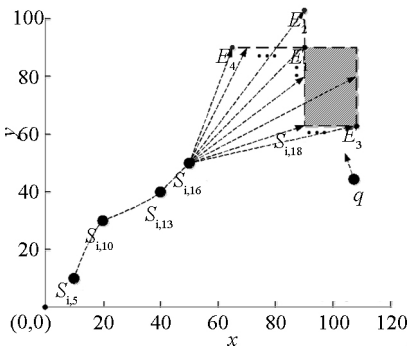


图 1 基于速度变化率的预测

ig. 1 Position predicting based onrate

性质 1 $p(o_1<o_2)\geqslant 0.5$ 当且仅当 $D_2-d_1\geqslant D_1-d_2$.

问题定义(不确定移动对象 k 近邻多线程查询 Multi-Query(Q,I)). 给定移动对象集合 DB_o、不确定轨迹数据序列 S、时间窗口 W、当前查询时刻 t_c、查询请求集合 Q、近邻数量 k 和线程数 M,Multi-Query(Q,I) 查询从 M 个线程并行返回 Q 中所有查询对象 q 在接下来时间 I 内最可能的 k 个最近邻移动对象.

涉及到的符号及相应含义如表 1 所示.

表 1 涉及的重要符号名称及解释

Tab. 1 Symbolsand descriptors

名称	解释
DB_o	移动对象集合 $DB_o=\{o_1,\ o_2,\cdots,\ o_n\}$
L_{grid}	网格密度列表,$L_{grid}=\{\rho_{j,k}\mid j,k=1,2,3,\cdots,m\}$
M	线程数量 M
o_i	移动对象 o_i
o_k^i	查询点 q_i 的第 k 近邻
O_k^i	查询点 q_i 的 k 近邻查询结果集合
Q_i	第 i 组查询对象的请求集合
Q	查询对象集合,$Q=\{Q_i\mid i=1,2,\cdots,M\}$
s_i	移动对象 o_i 的位置轨迹序列 s_i
$S_{\langle O\rangle,t}$	移动对象集合 O 中的所有移动对象在 t 时刻的位置
S	不确定轨迹数据集 $S=\{s_1,\ s_2,\cdots,s_n\}$
$\rho_{j,k}$	网格$<j,\ k>$的密度,即网格第 j 行第 k 列包含的查询点数

2 基于多线程的 k 近邻并行查询算法

2.1 多线程 k 近邻并行查询框架

针对不确定移动对象,给出基于多核多线程的连续 k 近邻并行查询 Multi-Query(Q,I)算法的框架,如算法 1 所示.

算法 1 Multi-Query(Q,I)

Inputs: DB_o, S, W, Q, k, M;

Outputs: k-NNs for each query object in Q;

01: $Q_{i=1,\ldots,M}$ gridPartition(Q,M);

02: FOR $i\leftarrow$ 1 to M PAR-DO

03: QueryThread(Q_i, I).

采用基于密度网格扩展的多线程数据分发方法，将查询请求根据地理空间划分成数量大致相等的 M 份；M 个线程并行执行使用数据复用优化的 k 近邻查询算法来获取查询对象的前 k 近邻. 2.2 与 2.3 节将分别介绍查询复用优化和基于密度网格扩展的数据分发方法.

2.2　邻近查询对象间的查询复用

考虑到用户在进行 k 近邻查询时当前位置是已知的并且查询的是在未来很小一段时间 I 内的大致位置，使用匀速运动方式对查询对象在 I 时间区间的位置进行预测.

对于同一时刻发起请求的多个查询对象，若相距较近，则考虑是否可以复用已查询到的结果来减少冗余的 k 近邻查询过程. 如图 2 所示，设定已对 q_i 查询完毕，并返回 q_i 的 k 近邻集合 O_k^i. 当 q_j 进行查询时，为了能够复用 q_i 的查询结果，需要考虑 q_j 与 q_i 之间的距离 $d(q_i, q_j)$ 可以直接复用 q_i 的查询结果时的条件. 若使得 q_j 的 k 近邻刚好为 q_i 的 k 近邻，如图 2 的虚线圆圈，即 q_j 的第 k 近邻刚好覆盖到 o_k^i，不超出 o_{k+1}^i，则需要满足 $2R_{qj}+d(q_i,o_{k+1}^i)-d(q_i,o_k^i)=2d(q_i,o_{k+1}^i)$. $R_{qj}=d(q_i,q_j)+d(q_i,o_k^i)$，可以得出 $d(q_i,q_j)=[d(q_i,o_{k+1}^i)-d(q_i,o_k^i)]/2$. 为了使得 q_j 的 k 近邻刚好包含 o_k^i，而不包括 o_{k+1}^i，所以 $d(q_i, q_j)$ 应该满足 $[0,(d(q_i,o_{k+1}^i)-d(q_i,o_k^i))/2)$，才能直接复用 q_i 的计算结果.

针对移动对象的 k 近邻查询预测问题，在复用查询结果时需要考虑查询对象之间的速度方向关系. 比如两个查询对象相背而行，则在未来 I 时刻内对查询结果进行复用，将会导致查询准确率降低. 利用两查询对象的余弦相似度来判定是否采用查询复用. 如图 3 所示，设定当前 q_i 已经查询完毕，继续对 q_j 进行查询，根据式(2)的余弦相似度，利用两查询对象的速度向量 v_i 与 v_j 计算相似度 $\mathrm{sim}(q_i,q_j)$.

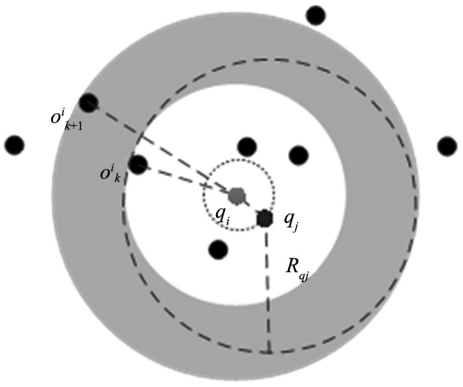


图 2　查询点间查询复用示例

Fig. 2　Query reuse between two query points

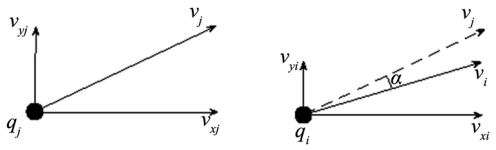


图 3　查询点之间的速度关系

Fig. 3　Velocity relations between query points

在实际情况下，同向运动的查询对象之间的距离较近较小，相似度越高，因此考虑速度方向夹角 $\alpha\in[0,\pi/6]$ 时对查询点进行查询复用，即当余弦相似度区间满足 $[\sqrt{3}/2,1]$ 时考虑查询复用.

$$\mathrm{sim}(q_i,q_j)=\cos\alpha=\frac{\boldsymbol{v}_i\cdot\boldsymbol{v}_j}{|\boldsymbol{v}_i||\boldsymbol{v}_j|}. \tag{2}$$

算法 2　QueryThread(Q, I)

Inputs: DB_o, S, W, Q, k;

Outputs: k-NNs for each query object in Q;

01: FOR $q_i\in Q$ DO

02: IF QRs=ϕ THEN

03: $\mathrm{Can}_i\leftarrow$PR-Tree(S, MAX_HITS, q_i);

04: ELSE

05: index ← reuseQuery(q_i, QRs);

06: IF index ≥ 0 THEN

07: $\mathrm{QR}_i\leftarrow$ QRs.get(index);

08: QR_i.isReuse ← TRUE; Goto line 12;

09: ELSE

10: $\mathrm{Can}_i\leftarrow$PR-Tree(S, MAX_HITS, q_i);

11: $\mathrm{QR}_i\leftarrow$Query(q_i, Can_i, I);

12: QRs←QRs ∪ QR_i;

13: RETURN QRs;

reuseQuery(q_i, QRs)

14: FOR QR_j in QRs DO

15: IF ！QR_j. isReuse THEN

16: IF $\mathrm{sim}(q_i,q_j)\in[\sqrt{3}/2,1]$ AND $d(q_i,q_j)<(d(q_j,o_{k+1})-d(q_j,o_k))/2$

17: THEN RETURN j;

18: RETURN -1;

Query(q_i, Can_i, I)

19: mutex ← 1;

20: FOR $o_j\in\mathrm{Can}_i$ DO

21: $s_j\leftarrow$get(S, o_j);

22: IF $s_{j,t_c+I}\notin\mathrm{MS}_{t_c+I}$ THEN

23: P(mutex);

24: compute s_{j,t_c+I};

25: $\mathrm{MS}_{t_c+I}\leftarrow\mathrm{MS}_{t_c+I}\cup\{s_{j,t_c+I}\}$;

26: V(mutex);

27: $QR_i \leftarrow$queryKNN(q_i, $S_{\{Can_i\},t_c+I}$);

28: RETURN QR_i;

算法2描述多线程下考虑查询复用时的k近邻查询算法. 若当前查询为第一个查询对象q_i($i=1$),则采用PRTree索引获取在当前时刻t_c下q_i周围MAX_HITS个移动对象集合Can_i作为q_i的k近邻的候选项集合. 04~09行描述的是对查询点q_i检查是否可以采用查询复用的过程. 使用reuseQuery()函数从已查询过的查询点中检查是否满足复用条件,即速度相似度与距离是否在复用范围以内;如果满足条件,则返回复用结果,否则,回退到最原始的计算方法,即对查询点的临近对象使用PR-Tree进行过滤(第10行). 为了避免查询复用的漂移,每次查询复用只针对真实计算过的k近邻查询,不会对复用结果再次复用,因此使用QR.isReuse进行区分,如第15行所示. 第11行表示采用不确定k近邻查询方法Rate返回q_i的前k近邻. 为了共享已经计算的移动对象区域预测结果,建立内存共享区MS_{t_c+I}用于暂存已经计算过的s_{j,t_c+I}的信息,如第20~27行所示. 对于q_i的每个候选邻近对象o_j,首先在共享内存区MS_{t_c+I}中查找在t_c+I时刻的预测区域s_{j,t_c+I},若不存在,则计算出o_j在t_c+I时刻的预测区域s_{j,t_c+I}. 第27行描述的是针对q_i对已经计算出的候选区域集合$S_{\{Can_i\},t_c+I}$查找q_i的前k近邻. 有关MS_{t_c+I}共享内存区的计算复用详细实现方法见2.3节的计算复用. 考虑到多线程操作,因此采用加锁操作,避免重复计算(第23、26行).

2.3 负载均衡的多线程k近邻查询

针对多线程并行k近邻查询,对移动对象计算过的预测区域进行共享是需要考虑的关键问题. 为了使得更多相邻查询对象复用查询的计算结果,需要对查询对象集合Q在地理空间上进行划分. 引入计算复用与密度网格扩展划分的概念来解决以上两个关键问题.

计算复用. 在同一线程内,只有当两查询对象满足复用距离时才能直接复用查询结果. 针对不确定移动对象的k近邻查询,对查询对象周围的移动对象可能到达区域的计算占据较大的时间开销. 对于互相之间距离较近的查询对象,查询范围通常存在较大的重叠区域,因此,在满足复用条件的前提下,可以直接共享已完成的移动对象可能到达区域的计算结果. 在多线程并行计算时,不同线程上的查询对象的查询所涉及到的相关移动对象集合往往存在重叠区域. 为了使得不同线程上查询对象能够共享已完成的移动对象可能到达区域的计算结果,实现多线程内存共享将有效加大查询过程中的计算复用. 为了避免重复计算,通过建立计算结果内存共享区MS_{future}来暂存已计算的移动对象集合在未来时间future $= t_c+I$时刻可能到达的区域,以实现计算复用.

密度网格扩展划分. 在实际应用中,大部分查询对象的分布是不均匀,例如在城市中,人口密集区域、商场商业区以及交通站点等位置查询请求往往过于密集,住宅区、公园、工厂等区域查询比较稀少. 在多线程数据分发时,采用随机方式虽然能够保证负载均衡,但会导致查询对象间的数据复用率降低. 网格划分是目前为止公认较高效的空间划分方法. 为了提升算法整体效率,即尽量增加查询复用率,又保证负载均衡,本文结合查询对象在位置上的相关性,提出基于密度网格扩展的数据分发策略.

为了方便描述,定义查询对象集合$Q=\{q_i \mid i=1,2,\cdots,n\}$,划分集合为$Q_M=\{Q_i \mid i=1,2,\cdots,M\}$,$|\bar{Q}|=|Q|/M$,$\rho_{j,k}$表示网格中对应单元格$<j,k>$(第$j$行第$k$列)的密度,即所包含查询对象的数量. 首先,对$Q$集合所在区域进行网格划分,将查询对象在空间位置上映射到相应网格内,根据每个网格包含的查询对象数量计算网格密度$\rho_{j,k}$并按照降序排序得到网格密度列表L_{grid}. 然后,从L_{grid}的第一个网格开始,执行扩展操作,直至得到M个划分. 若当前网格密度$\rho_{j,k}<|\bar{Q}|$,则使网格$<j,k>$向周围未划分的网格进行扩展,扩展方式按以下原则进行. 1) 若当前网格周围的临近网格未被其他网格扩展,则当前网格向周围网格扩展,直到网格$<j,k>$及扩展网格所包含的查询对象数量刚好不小于$1.2\bar{Q}$结束(根据经验可知,将$0.8\bar{Q}$~$1.2\bar{Q}$视为负载平衡状态). 2) 若当前网格周围的临近网格已被扩展或临近网格不包含任何查询对象,则不再对当前网格进行扩展. 在扩展操作完毕后,将会得到M个划分,接下来继续执行平衡操作. 若存在$|Q_i|<0.8|\bar{Q}|$,首先将未划分的网格归入到Q_i中,使得满足$0.8|\bar{Q}|\leqslant|Q_i|\leqslant1.2|\bar{Q}|$. 若已不存在未划分的网格,则循环选择$Q_j=\max\{|Q_j| \mid j=1,2,\cdots,M\}$,并将$Q_j$所包含的查询对象按以下平衡原则分配到$Q_i$. 1) 若$Q_j$是由高密度网格扩展获得,则选择某扩展网格的查询对象分配到Q_i. 2) 若Q_j仅包含一个高密度网格,则随机选取Q_j中的$(|Q_j|-|Q_i|)/2$个查询对象到Q_i. 该密度网格的划分方法优先将密集区域及相邻网格划分在同一线程,提高了数据复用率;考虑了每个划分包含数量相近的查

询对象，进而实现了多线程间的负载均衡.

算法 3. gridPartition(Q, M)

Inputs：Q, M;

Outputs：$Q_M=\{Q_i \mid i=1,2,\cdots,M\}$;

01：G←mapping(Q);/* Mapping Q into Grid* /

02：L_{grid}←density(G);

03：WHILE $|L_{\text{grid}}|\neq 0$ DO

04：g←L_{grid}.getfirst();

05：Q_i←expendCell(g); i++;

06：IF $i=M$ THEN break;

07：Q←fetchQ_i(Q, L_{grid});

08：Q←balanceMParts(Q);

09：RETURN Q;

balanceMParts(Q)

10：FOR $i\leftarrow M$ to 1 DO

11：IF $|Q_i|<0.8\bar{Q}$ THEN

12：$Q_j=\max\{|Q_j| \mid j=1,2,\cdots,M\}$;

13：move q from Q_j to Q_i;

算法 3 描述了基于密度网扩展的数据分发策略. 首先将所有查询对象映射到网格，得到按照网格密度降序排列的列表L_{grid}，如 01、02 行所示. 然后循环从L_{grid}取出第一个网格，对该网格进行扩展，直到获取 M 个划分(如 03～06 行所示). 扩展过程按照所述的 2 点扩展原则进行扩展，每次扩展后网格从L_{grid}中删除. 之后，将剩余L_{grid}分配到较少的划分中(第 07 行). 最后，根据 M 个划分的查询数量，使用平衡原则对 Q 的 M 个划分进行平衡操作，如 balanceMParts(Q)函数.

如图 4 所示，假设当前线程数 $M=4$，$|Q|=40$，此时每个线程应当大致分得$\bar{Q}=40/4=10$ 个查询点. 首先将查询点所覆盖的区域划分成 5×5 网格，

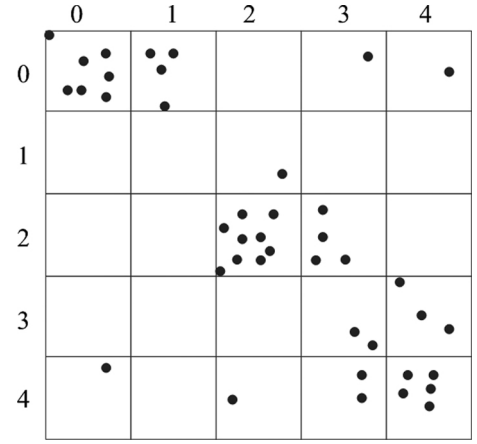


图 4　基于密度网格的划分示例

Fig. 4　Grid-based partition

并将查询点映射到相应网格，按照网格密度降序排列得到列表$L_{\text{grid}}=\{\rho_{2,2}=9,\ \rho_{0,0}=7,\ \rho_{4,4}=5,\ \rho_{0,1}=4,\ \rho_{2,3}=4,\ \rho_{3,4}=3,\ \rho_{3,3}=2,\ \rho_{4,3}=2,\ \rho_{0,3}=1,\ \rho_{0,4}=1,\ \rho_{1,2}=1,\ \rho_{4,0}=1,\ \rho_{4,2}=1\}$. 第一步首先得到$\rho_{2,2}=9<1.2|\bar{Q}|$，因此对网格<2，2>进行扩展，当扩展到{<2,2>,<2,3>,<1,2>}后得到划分$|Q_1|=14$，以满足$|Q_1|>1.2|\bar{Q}|$. 此时，$L_{\text{grid}}=\{\rho_{0,0},\ \rho_{4,4},\ \rho_{0,1},\ \rho_{3,4},\ \rho_{3,3},\ \rho_{4,3},\ \rho_{0,3},\ \rho_{0,4},\ \rho_{4,0},\ \rho_{4,2}\}$. 继续计算 Q_2，由于$\rho_{0,0}<|\bar{Q}|$，从单元格<0，0>向周围扩展；当扩展至{<0,0>,<0,1>}后，周围不再有包含查询点的网格，停止扩展. 此时，Q_2所包含的查询对象数量为$|Q_2|=11$. $L_{\text{grid}}=\{\rho_{4,4},\ \rho_{3,4},\ \rho_{3,3},\ \rho_{4,3},\ \rho_{0,3},\ \rho_{0,4},\ \rho_{4,0},\ \rho_{4,2}\}$. 继续计算 Q_3，对$\rho_{4,4}$扩展得到单元格{<4,4>, <3,3>, <3,4>, <4,3>}，对应划分$|Q_3|=12$. 此时，$L_{\text{grid}}=\{\rho_{0,3},\ \rho_{0,4},\ \rho_{4,0},\ \rho_{4,2}\}$，对应最后一个划分$|Q_4|=4$. 划分阶段结束，由于存在$|Q_4|<0.8|\bar{Q}|$，进入平衡阶段，通过选择 $Q_1=\max\{|Q_j| \mid j=1,2,3,4\}$对 Q_1 进行再分配. 由于 Q_1 是由高密度网格{<2,2>, <2,3>,<1,2>}得到，将单元格<2,3>对应的查询点分配给 Q_4，刚好得到$|Q_4|=8$. 此时，所有划分都满足平衡状态，最终基于密度网格扩张的划分结果为$|Q_1|=10$，$|Q_2|=11$，$|Q_3|=12$，$|Q_4|=8$.

3　实验与结果

3.1　实验数据与测试方法

实验平台采用 Intel Xeon E5-2660 处理器，8 核，48 GB 内存，Windows Server 2012 操作系统，所有算法全部由 Java 实现.

实验数据. 数据集 Oldenburg 由 Thomas Brinkhoff 实现的基于网络的移动对象生成器，该生成器可以生成德国奥尔登堡城市的交通数据[23]. 使用的数据包含 5 000 个移动对象、905 000 个位置采样点；根据生成器的参数设置，生成的移动对象的速度类型包含 6 类(类似于行人、自行车、机动车等不同的速度). 为了模拟不确定数据，将该数据集分为 3 个版本，分别为全部数据 Full、随机删除 10% 采样点的数据集 Full-10% 以及随机删除 20% 采样点的数据 Full-20%.

评估方法. 采用准确率评估方法，Precision=$(R\cap D)/R$，其中 D 为查询时刻从完整数据中查询的真实 k 近邻结果，R 为所提算法在删除数据之后

获得的 k 近邻预测结果. 对于效率评估,通过变化各重要参数,测量单次查询所执行的 CPU 时间来评估算法的性能.

对比方法. 对提出优化方法的组合策略及 Rate 方法进行综合对比分析. 组合方案如下:不考虑数据复用、随机分发查询请求、无共享内存的多线程查询方法(basic random no-shared memory, BRN);不考虑数据复用、随机分发查询、共享内存的多线程查询方法(basic random shared memory, BRS);考虑数据复用、随机分发查询、共享内存的多线程查询方法(optimized random shared memory, ORS);考虑数据复用、密度网格分发查询、共享内存的多线程查询处理方法(optimized grid shared memory, OGS). 所有方法的实现都采用 PR-Tree 索引[24].

3.2 实验和结果

正确性评估. 首先,在 Oldenburg 三组数据集上验证了所提方法的正确率,并与串行算法 Rate 进行对比分析. 参数设置如下:$W=10$,查询对象数量 $Q=1\ 500$,$I=1$,查询时间间隔 $\Delta t=2$,$k=40$. 如图 5所示为 Rate 与 OGS 方法分别在 2、4、6、8 个线程上并行查询的正确率 P. 图中,Full 为所有样本,Full-10%表示在原有数据的基础上删除了 10%,Full-20%表示在原有数据的基础上删除了 20%. 可以看出,在 3 个数据集上,OGS 方法在多个线程上的并发查询结果几乎都与 Rate 方法的查询结果完全一致,验证了 OGS 方法的有效性. 随着删除数据的增多,不确定移动对象的数量增加,导致所有算法的正确率都下降,但在 Full-20%数据集上仍能达到近 80%的正确率.

性能评估. 下面在 Full-20%数据集上评估重要参数对所提算法查询效率的影响. 在没有明确指定参数变化下的默认参数设置如下:$W=10$,查询请求数量 $Q=500$,$I=1$,查询时间间隔 $\Delta t=2$,$k=40$,线程数 $M=8$.

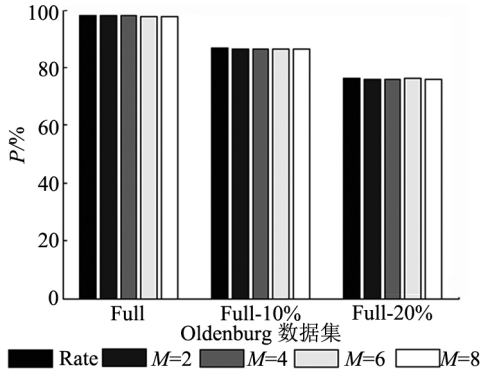
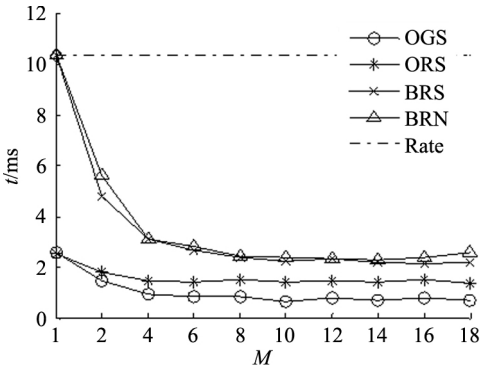


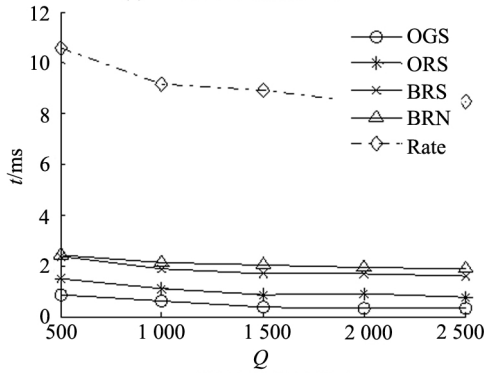
图 5 不同数据集上的正确率对比

Fig. 5 Precision comparison on different datasets

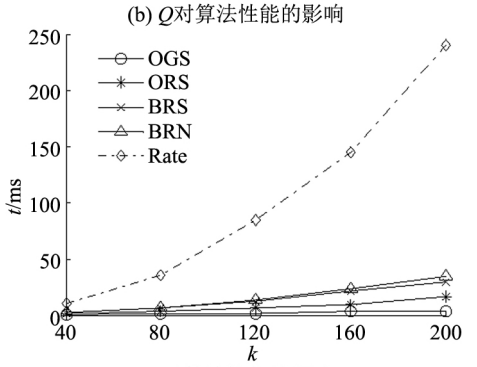
首先,通过变化 M 评估并行线程数量对 4 种组合算法性能的影响. 如图 6(a)所示,采用数据复用与密度网格分发的 OGS 方法所消耗的时间最少,采用数据复用与随机分发的 ORS 次之,BRN 使用最多的 CPU 时间. 随着 M 的增加,单次查询的平均时间不断降低,但是当 $M\geqslant 10$ 以后,所有算法趋于平稳. 这是由于该实验平台 CPU 为八核,当线程数超过该数量后,不能进一步提升并行度;当线程过多时,单核上线程之间的切换将带来一定的开销. 当 $M=10$ 时,OGS 方法相对于串行 Rate 的加速比已达 16.59. 通过对比 OGS、ORS 与 BRS、BRN 方法可以发现,采用查询复用方法对算法效率有很大的提升. 通过对比 OGS 与 ORS 方法可以发现,利用



(a) M对算法性能的影响



(b) Q对算法性能的影响



(c) k对算法性能的影响

图 6 OGS、ORS、BRS、BRN、Rate 算法的效率对比

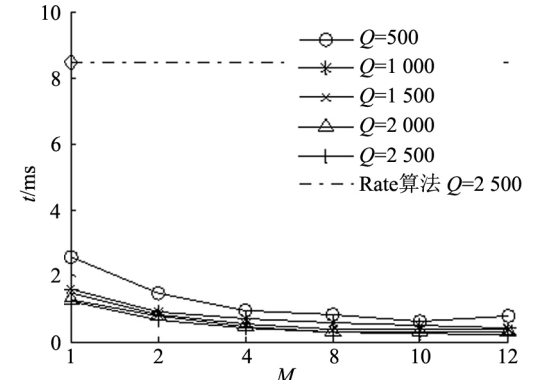
ig. 6 Efficiency comparison on OGS, ORS, BRS, BRN, Rate

密度网格分发策略进一步提升了数据复用率.

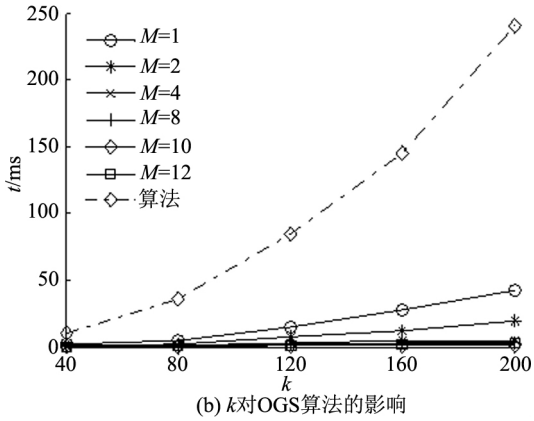
图 6(b)描述了 Q 对算法性能的影响. 随着 Q 的不断增加，虽然算法消耗的总时间是上升的，但是单次查询的平均时间呈下降趋势. 这是因为 Q 越多，相近的查询请求越多，数据复用率越高. 同样，OGS 方法使用了最少的 CPU 时间，当 $Q=2\ 500$ 时，OGS 方法的效率相比于 BRN 提高了 5.27 倍. 这证明了所提的 OGS 方法相对于 Q 具有较高的可扩展性.

图 6(c)描述了参数 k 对算法效率的影响，当 k 增大时，单次查询的平均消耗时间都上升，这是因为当增大 k 时，PRTree 索引需要搜索的临近移动对象数量相应增加，需要计算的可能到达的区域与排序邻居数量增加，导致平均消耗时间增加. 从总体上来看，OGS 消耗最少的时间，并且增长缓慢. 与 BRN、BRS 和 ORS 相比，OGS 分别平均提升了 536%、481%和 200%的效率. 这验证了 OGS 相对参数 k 具有较好的可扩展性.

为了进一步考查 OGS 算法的扩展性能，分别通过变化 M、k、Q 对算法进行评估. 首先，评估了 M 对不同查询请求数量下 OGS 算法性能的影响. 如图 7(a)所示，当增大 M 时，OGS 算法处理每次查询请求所消耗的时间不断降低，当线程数达到 10 之后



(a) M对OGS算法的影响



(b) k对OGS算法的影响

图 7　线程数量 M 与参数 k 对 OGS 算法的影响

Fig. 7　Efficiency of OGS w. r. t M and k

算法性能趋于平稳，这是由硬件平台的限制导致的. 当 Q 增多时，单次查询请求的平均消耗时间降低. 与 Rate 算法相比，当 $M=8$，$Q=2\ 500$ 时，并行后的加速比最高可达 28.27.

图 7(b) 给出参数 k 对各线程数下 OGS 算法效率的影响. 可以发现：相对于参数 k，OGS 算法在多线程环境下展现了更好的扩展性. 随着 k 的增大，单次查询的平均响应时间都上升，但线程数越多，上升越缓慢. 这说明所提的复用策略在多线程下有效提升了算法效率. 当 $M=8$ 时，与 Rate 算法相比，效率平均提升了 37 倍.

4　结　语

本文提出多核多线程环境下面向不确定移动对象的连续 k 近邻并行查询算法. 针对不确定移动对象已计算的可能到达区域，设置内存共享区用于不同线程之间的计算复用. 针对空间位置邻近的查询请求，设计查询复用的方法，确定了复用条件. 利用密度网格扩展的方法，实现了负载均衡的多线程数据分发. 实验结果表明，提出的多线程方法与串行算法相比，加速比达到 16.5 以上. 下一步将考虑引入路网和城市兴趣点，进一步研究有效的不确定移动对象连续 k 近邻并行查询策略.

参考文献(References)：

[1] ZHENG Y, CAPRA L, WOLFSON O, et al. Urban computing: concepts, methodologies, and applications [J]. **ACM Transactions on Intelligent Systems and Technology**, 2014, 5(3): 38.

[2] 周傲英，杨彬，金澈清，等. 基于位置的服务：架构与进展[J]. 计算机学报，2011，34(7)：1155-1171.
ZHOU Ao-ying, YANG Bin, JIN Che-qing, et al. Location-based services: architecture and progress [J]. **Chinese Journal of Computers**, 2011, 34(7): 1155-1171.

[3] TAO Y, PAPADIAS D, SHEN Q. Continuous nearest neighbor search [C]// **International Conference on Very Large Data Bases**. Hong Kong: VLDB Endowment, 2002: 287-298.

[4] ZHENG Yu. Trajectorydata mining: an overview [J]. **ACM Transactions on Intelligent Systems and Technology**, 2015, 6(3): 1-41.

[5] 于彦伟，齐建鹏，宋鹏，等. 面向不确定移动对象的连续 K 近邻查询算法[J]. 模式识别与人工智能，2016，29(11)：1048-1056.
YU Yan-wei, QI Jian-peng, SONG Peng, et al. Continuous K-nearest neighbor queries for uncertain moving

objects [J]. **Pattern Recognition and Artificial Intelligence**, 2016, 29(11): 1048-1056.

[6] KOLAHDOUZAN M R, SHAHABI C. Alternativesolutions for continuous K nearest neighbor queries in spatial network databases [J]. **Geoinformatica**, 2005, 9(4): 321-341.

[7] YU X, PU K Q, KOUDAS N. Monitoring k-nearest neighbor queries over moving objects [C]// **IEEE International Conference on Data Engineering (ICDE)**. Tokyo, Japan: IEEE, 2005: 631-642.

[8] XIONG X, MOKBEL M F, AREF W G. SEA-CNN: scalable processing of continuous k-nearest neighbor queries in spatio-temporal databases [C]// **IEEE International Conference on Data Engineering (ICDE)**. Tokyo: IEEE, 2005: 643-654.

[9] **郝兴, 王凌, 孟小峰. 一种道路网络中移动对象的k近邻多查询处理算法**[C]// **中国数据库学术会议. 海口: 中国计算机学会**, 2007: 113-118.
HAO Xing, WANG Ling, MENG Xiao-feng. Multiple KNN queries processing for moving objects in road networks [C]//**National Database Conference**. Haikou: CCF, 2007: 113-118.

[10] **孙圣力, 林硕. 一个高效的连续k近邻查询改进算法** [J]. **计算机研究与发展**, 2013(**增**1): 80-89.
SUN Sheng-li, LIN Shuo. A improved algorithm for efficient continuous KNN queries [J]// **Journal of Computer Research and Development**, 2013(supple.1): 80-89.

[11] MOURATIDIS K, YIU M L, PAPADIAS D, et al. Continuousnearest neighbor monitoring in road networks [C]// **Proceedings of the 32nd Very Large Data Bases Conference (VLDB)**. Seoul: VLDB, 2006: 43-54.

[12] HUANG Y K, CHEN Z W, LEE C. Continuous K-nearest neighbor query over moving objects in road networks [C]// **Advances in Data and Web Management, Joint International Conferences**. Suzhou: [s.n.], 2009: 27-38.

[13] **廖巍, 吴晓平, 严承华, 等. 多用户连续k近邻查询多线程处理技术研究**[J]. **计算机应用**, 2009, 29(7): 1861-1864.
LIAO Wei, WU Xiao-ping, YAN Cheng-hua, et al. Research on multi-threading processing of concurrent multiple continuous k-nearest neighbor queries [J]. **Journal of Computer Applications**, 2009, 29(7): 1861-1864.

[14] **赵亮, 陈荦, 景宁, 等. 道路网中的移动对象连续K近邻查询**[J]. **计算机学报**, 2010, 33(8): 1396-1404.
ZHAO Liang, CHEN Luo, JING Ning, et al. Continuous K nearest neighbor queries of moving objects in road networks [J]. **Chinese Journal of Computers**, 2010, 33(8): 1396-1404.

[15] **赵亮, 景宁, 陈荦, 等. 面向多核多线程的移动对象连续K近邻查询**[J]. **软件学报**, 2011, 22(8): 1805-1815.
ZHAO Liang, JING Ning, CHEN Luo, et al. Continuous K nearest neighbor queries over moving objects based on multi-core and multi-threading [J]. **Journal of Software**, 2011, 22(8): 1805-1815.

[16] HUANG Y K, CHEN C C, LEE C. Continuous K-nearest neighbor query for moving objects with uncertain velocity [J]. **Geoinformatica**, 2009, 13(1): 1-25.

[17] **王艳秋, 徐传飞, 于戈, 等. 一种面向不确定对象的可见k近邻查询算法**[J]. **计算机学报**, 2010, 33(10): 1943-1952.
WANG Yan-qiu, XU Chuan-fei, YU Ge, et al. Visible k nearest neighbor queries over uncertain data [J]. **Chinese Journal of Computers**, 2010, 33(10): 1943-1952.

[18] **陈子军, 任彩平, 刘文远. 路网中查询点速度不确定的连续k近邻查询方法**[J]. **小型微型计算机系统**, 2011, 32(3): 430-434.
CHEN Zi-jun, REN Cai-ping, LIU Wen-yuan. Continuous K-nearest neighbor query for query points with uncertain velocity [J]. **Journal of Chinese Computer Systems**, 2011, 32(3): 430-434.

[19] **王宝文, 胡云, 陈子军, 等. 路网中速度不确定移动对象的k近邻查询**[J]. **小型微型计算机系统**, 2012, 33(8): 1756-1760.
WANG Bao-wen, HU Yun, CHEN Zi-jun, et al. k-nearest neighbor query for moving objects with uncertain velocity in road network [J]. **Journal of Chinese Computer Systems**, 2012, 33(8): 1756-1760.

[20] LI G, LI Y, SHU L C, et al. CkNNquery processing over moving objects with uncertain speeds in road networks [C]// **Proceedings of the 13th Asia-Pacific Web Conference on Web Technologies and Applications**. Beijing, China: Springer, 2011: 65-76.

[21] FAN P, LI G, YUAN L, et al. Vague continuous K-nearest neighbor queries over moving objects with uncertain velocity in road networks [J]. **Information Systems**, 2012, 37(1): 13-32.

[22] SISTLA A P, WOLFSON O, XU B. Continuous nearest-neighbor queries with location uncertainty [J]. **VLDB Journal**, 2015, 24(1): 25-50.

[23] BRINKHOFF T. Generatingnetwork-based moving objects [C]// **International Conference on Scientific and Statistical Database Management**. Washington, DC: IEEE, 2000: 253-255.

[24] ARGE L, BERG M D, HAVERKORT H, et al. The priority R-tree: a practically efficient and worst-case optimal R-tree [J]. **ACM Transactions on Algorithms**, 2008, 4(1): 9.