ISSN 1000-9825, CODEN RUXUEW
Journal of Software, Vol.18, No.2, February 2007, pp.268−278
DOI: 10.1360/jos180268
© 2007 by *Journal of Software*. All rights reserved.
E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

可伸缩的增量连续 *k* 近邻查询处理*

廖 巍+, 熊 伟, 王 钧, 景 宁, 钟志农

(国防科学技术大学 电子科学与工程学院,湖南 长沙 410073)

Scalable Processing of Incremental Continuous *k*-Nearest Neighbor Queries

LIAO Wei+, XIONG Wei, WANG Jun, JING Ning, ZHONG Zhi-Nong

(College of Electronic Science and Engineering, National University of Defense Technology, Changsha 410073, China)

+ Corresponding author: Phn: +86-731-4573480, Fax: +86-731-4575798, E-mail: liaowei_2000@163.com, http://ww.nudt.edu.cn

Liao W, Xiong W, Wang J, Jing N, Zhong ZN. Scalable processing of incremental continuous *k*-nearest neighbor queries. *Journal of Software*, 2007,18(2):268−278. http://www.jos.org.cn/1000-9825/18/268.htm

Abstract: To evaluate the large collection of concurrent CKNN (continuous *k*-nearest neighbor) queries continuously, a scalable processing of the incremental continuous *k*-nearest neighbor (SI-CNN) framework is proposed by introducing searching region to filter the visiting TPR-tree (time-parameterized R-tree) nodes. SI-CNN framework exploits the incremental results table to buffer the candidate objects, flushes the objects into query results in bulk, efficiently processes the large number of CKNN queries concurrently, and has a perfect scalability. An incremental SI-CNN query update algorithm is presented, which evaluates incrementally based on the former query answers and supports the insertion or deletion of both query collection and moving objects. Experimental results and analysis show that SI-CNN algorithm based on SI-CNN framework can support a large set of concurrent CKNN queries perfectly, and has a good practical application.

Key words: CKNN (continuous *k*-nearest neighbor) query; TPR-tree (time-parameterized R-tree); SI-CNN (scalable processing of incremental continuous *k*-nearest neighbor queries) framework; SI-CNN algorithm; incremental processing

摘 要: 针对基于 TPR 树(time-parameterized R-tree)索引的大量并发 CKNN(continuous *k*-nearest neighbor)查询处理,提出了一种可伸缩的增量连续 *k* 近邻查询处理(scalable processing of incremental continuous *k*-nearest neighbor queries,简称 SI-CNN)框架,通过引入搜索区域进行预裁剪以减少查询更新所需要的 TPR 树节点访问代价,并引入了增量结果表以保存候选对象,批量地更新查询结果集,具有良好的可伸缩性.基于 SI-CNN 框架提出了一种增量更新的 SI-CNN 查询处理算法,能够基于上次查询结果增量的更新查询,支持查询集合中加入或删除查询和移动对象数据集的插入、删除等动态更新操作.实验结果与分析表明,基于 SI-CNN 框架的 SI-CNN 算法可以很好地支持大量并发的 CKNN 查询处理,具有良好的实用价值.

关键词: 连续 *k* 近邻查询;TPR 树;SI-CNN 框架;SI-CNN 算法;增量处理

中图法分类号: TP311 文献标识码: A

* Supported by the National Natural Science Foundation of China under Grant No.60472031 (国家自然科学基金)
Received 2005-05-11; Accepted 2006-04-03

© 中国科学院软件研究所 http://www.jos.org.cn

连续 k 近邻(continuous k-nearest neighbor,简称 CKNN)查询[1]是时空数据库中重要的查询类型之一,用于连续查找 k 个距离查询位置最近的对象.在交通调度控制、位置服务等领域,CKNN 查询得到了广泛的关注和应用.与传统静态空间 KNN(k-nearest neighbor)查询一次计算得到的查询结果不同,CKNN 查询要求结果必须及时更新,以反映移动对象和/或查询本身的变化,因而查询处理算法更加复杂[1].

CKNN 查询所处理的对象通常为位置动态变化的移动对象数据集,并以 TPR 树(time-parameterized R-tree)[2]索引进行存储管理,而查询本身也是动态的.针对基于 TPR 树索引的大量移动对象 CNN 查询问题,Benetis 等人首先提出了 Find-NN 算法[3],利用最小距离函数裁剪准则对 TPR 树进行深度优先搜索遍历,以获得最近邻对象.Tao 等人[4]对 Find-NN 算法进行了扩展,以支持 CKNN 查询.Glenn 等人提出的 CW-KNN 算法[5]则利用搜索区域对 TPR 树索引节点进行搜索预裁剪,以提高 CKNN 查询的更新性能.Tao 等人提出的 TP-KNN 算法[6]能够有效地处理时间参数化 k 近邻查询,但对于 CKNN 查询算法则必须重复提交 TP-KNN 查询,以获得连续查询结果集,其效率非常低下.

针对实际应用中大量并发的 CKNN 查询处理问题,Mokbel 等人首先提出了 SINA 框架[7],以处理大量并发时空查询;Hu 等人提出了查询驱动更新的通用连续查询处理框架[8];Yu 等人提出了层次网格移动对象索引和查询索引,以改善 CKNN 的查询性能[9].到目前为止,Xiong 等人提出的 SEA-CNN(scalable evaluation of continuous nearest neighbor)算法[10]以及 Mortifies 等人提出的 CPM(concept ional partition model)算法[11]是支持大量并发的连续 k 近邻查询处理的具有代表性的成果.SEA-CNN 算法通过引入搜索区域和共享查询执行的思想来处理 CKNN 查询的更新操作,具有良好的可伸缩性;CPM 算法则使用概念空间划分(conceptual space partitioning)技术来提高查询更新时的空间搜索性能.上述两种算法均采用静态空间规则划分的格网(grid cell)结构来索引移动对象集,索引更新性能较差,且不支持目前通用的 TPR 树索引.

本文主要研究基于 TPR 树索引移动对象集的大量并发 CKNN 查询,提出了一种高效而实用的可伸缩的增量连续 k 近邻查询处理框架和相关算法.本文的主要贡献如下:

1) 提出了一种可伸缩的增量连续 k 近邻查询处理(scalable processing of incremental continuous k-nearest neighbor queries,简称 SI-CNN)框架,通过引入搜索区域对 TPR 树索引节点进行预裁剪以减少查询更新所需要的磁盘访问代价,并引入了基于内存的增量结果表对查询结果进行批量更新.
2) 基于 SI-CNN 框架提出了一种支持更新的 SI-CNN 查询处理算法.该算法能够基于上次查询结果进行增量更新查询,支持查询集合中加入或删除查询和移动对象数据集的插入、删除等动态更新操作.
3) 在仿真实验环境下,对本文提出的 SI-CNN 算法进行了大量的实验测试.实验结果与分析表明,基于 SI-CNN 框架的 SI-CNN 算法具有良好的查询和更新性能.

本文第 1 节介绍连续 k 近邻查询的相关基本概念.第 2 节介绍基于 TPR 树索引的 SI-CNN 查询处理框架以及 SI-CNN 查询处理算法.第 3 节给出实验结果比较与分析.最后是结论与展望.

1 连续 k 近邻查询的基本概念

本节给出了文献[4]与文献[10]中引出的相关概念.本文后续部分将直接使用这些基本定义.

定义 **1**(连续 k 近邻查询)[10]. 连续 k 近邻查询定义如下:$q=(At+B,k,T)$,其中:$At+B$ 表示查询位置 $q.Loc_t$ 的线性运动轨迹;k 表示查询 q 的最近邻对象个数;T 表示查询的时间窗口范围.

定义 **2**(查询提交时刻)[10]. 客户端向服务端提交 CKNN 查询的时刻 t_{isu}.

定义 **3**(查询更新时刻)[10]. 服务端周期性地或当缓冲区中移动对象数目超出阈值时查询进行更新的时刻 t_{upt}.

定义 **4**(查询参考时刻)[10]. CKNN 查询提交时刻 t_{isu} 或查询更新时刻 t_{upt},表示为 t_{ref}.

CKNN 查询在查询更新时刻 t_{upt} 进行更新,由于在客户端提交新的 CKNN 查询时刻 t_{isu},服务端并不立即计算当前查询的初始结果集,而是加入到查询集合中,并在查询更新时刻 t_{upt} 与其他需要更新的查询一起进行处理.因此,通常将查询参考时刻定义为查询更新时刻,即 $t_{ref}=t_{upt}$,出于简单考虑,我们令 $t_{ref}=0$.

© 中国科学院软件研究所 http://www.jos.org.cn

定义 **5**(距离函数)[4]. 在 d 维欧式空间中,给定在参考时刻 t_{ref} 以线性速度矢量运动的查询 q 与移动点对象 p,则在未来任意时刻 t,查询位置 $q.Loc_t$ 与对象 p 之间的距离定义为查询 q 与移动对象 p 之间的距离函数,

$$dist(q,p,t)=At^2+Bt+C,\text{其中 } A,B,C \text{ 均为常量.}$$

距离函数 $dist(q,p,t)$为关于时间 t 的二次凹函数,A,B,C 与在参考时刻 $t_{ref}=0$ 时查询 q 的位置和速度矢量、移动对象 p 的位置和速度矢量有关.

定义 **6**(最近邻距离)[4]. 在 d 维欧式空间中,给定 CKNN 查询 q,在任意时刻 t 查询位置 $q.Loc_t$ 到 k 最近邻对象 p 的距离定义为查询 q 的最近邻距离.

$$\min dist(t)=\{\langle dist(t),T\rangle\},\ dist(t)=At^2+Bt+C.$$

最近邻距离 $\min dist(t)$为元组$\langle dist(t),T\rangle$的集合,$dist(t)$表示在 T 时间间隔内查询 q 到 k 最近邻对象 p 之间的距离函数.最近邻距离 $\min dist(t)$给出了 CKNN 查询 q 在查询窗口范围 $q.T=[t_{ref},t_{ref}+TL]$内近邻搜索的上界,只有当更新、插入或删除的移动对象 p 在某个时间间隔$[t_s,t_e]\subseteq[t_{ref},t_{ref}+TL]$内与查询位置 q 的距离函数 $dist(q,p,t)$小于 $\min dist(t)$时,移动对象 p 才会对 CKNN 查询 q 的结果产生影响.

2 可伸缩的增量连续 *k* 近邻(SI-CNN)查询处理算法

2.1 SI-CNN查询处理思想

给定以线性速度矢量运动的 CKNN 查询 q,利用扩展 Find-NN 算法[4]对 TPR 树索引进行搜索,可以得到查询初始结果集.在查询 q 的时间窗口范围内,若存储移动对象的数据库发生变化,包括移动对象运动的更新(如速度大小和/或运动方向的改变)、移动对象的插入或删除操作等,则可能会对查询 q 的初始结果集产生影响,因此,必须对 CKNN 查询结果进行更新,以保证查询的正确性.

在理想情况下,一旦数据库发生变化,则应立即更新 CKNN 查询结果.但是,当查询的数据库移动对象规模非常大时,这种方式是不切实际且不必要的.与文献[10]中采用的方法类似,为了避免数据库频繁更新所带来的 TPR 树索引和 CKNN 查询结果的频繁更新操作,SI-CNN 查询处理将客户端提交的更新、插入和删除的移动对象缓存在缓冲区中,并按照一定的时间间隔周期性地或当缓冲区中移动对象数目超过阈值时,将缓冲区中的移动对象更新到物理磁盘(数据页面和 TPR 树索引页面)中,并更新所有的 CKNN 查询.根据实际应用,通过设置更新时间间隔和缓冲区大小,可以获得用户所需要的准确度和查询更新效率.

定义 **7**(搜索半径). 给定 CKNN 查询 q 的时间窗口 $q.T=[t_{ref},t_{ref}+TL]$及此窗口内的最近邻距离 $\min dist(t)=\{\langle dist(t),T\rangle\}$,存在一个线段方程集合 $L=\{l(t)=At+B\}$,其中,$l(t)$为其上确界,即满足$\forall t\in[t_{ref},t_{ref}+TL],l(t)\geq dist(t)$.特别地,我们定义查询 q 的搜索半径 $q.SS$ 如下:

$$q.SS=l_{\min}(t)=At+B\in L,\text{且}\forall l'(t)=A't+B'\in L,\ \int_{t_{ref}}^{t_{ref}+TL}(At+B)\mathrm{d}t<\int_{t_{ref}}^{t_{ref}+TL}(A't+B')\mathrm{d}t.$$

搜索半径 $q.SS$ 给出了 CKNN 查询 q 在查询窗口范围$[t_{ref},t_{ref}+TL]$内近邻搜索的线性上确界,若更新、插入或删除的移动对象 p 在查询窗口范围任意时刻 t 到查询位置 $q.Loc_t$ 的距离函数 $dist(q,p,t)$大于搜索半径 $q.SS$,则不会对 CKNN 查询 q 的结果产生影响.

定义 **8**(搜索区域). 在 d 维欧式空间中,给定 CKNN 查询 q,在任意时刻 $t\in[t_{ref},t_{ref}+TL]$,以查询位置 $q.Loc_t$ 为中心、搜索半径 $q.SS$ 为半径所覆盖的区域称为搜索区域 $q.SR$.

定理 **1**. 给定 CKNN 查询 q 及其搜索区域 $q.SR$,若移动对象 p 在任意时刻 $t\in[t_{ref},t_{ref}+TL]$落在搜索区域 $q.SR$ 之外,则移动对象 p 不会对查询 q 产生影响.

证明:若在任意时刻 $t\in[t_{ref},t_{ref}+TL]$,移动对象 p 落在 CKNN 查询 q 搜索区域 $q.SR$ 之外,则根据定义 6,在任意时刻 $t\in[t_{ref},t_{ref}+TL]$,移动对象 p 到查询 q 的距离函数 $dist(p,q,t)>\min dist(q,t)$,即对象 p 不可能成为查询 q 的 k 近邻对象,因此不会对查询 q 产生影响. □

定义 **9**(查询相似度,**QS**). 给定两个 CKNN 查询 $q=(At+B,k,T),q'=(A't+B',k',T')$,这两个查询的相似度 $QS(q,q')$定义为

© 中国科学院软件研究所 http://www.jos.org.cn

$$QS(q,q')=\begin{cases}\left(\dfrac{\int_{t_s}^{t_e} dist(q,q',t)\mathrm{d}t}{t_e-t_s}\right)^{-1}, & T\cap T'=[t_s,t_e]\neq\varnothing\\ 0, & \text{otherwise}\end{cases}.$$

定义 **10**(相似查询,**SQ**). 给定 CKNN 查询 $q=(At+B,k,T)$和查询集合 Q,则查询集合 Q 中与 CKNN 查询 q 相似的查询 SQ(q,Q)定义为

$$QS(q,Q)=q'=(A't+B',k',T'),$$

其中,$\forall q''=(A''t+B'',k'',T'')\in Q, QS(q,q')\geq QS(q,q'')$.

相似查询的定义给出了查询集合 Q 中与查询 q 具有最相似语义的查询.根据此定义,我们引入了 SI-CNN 动态扩充查询处理算法,其思想是:当客户端提交新的查询 q 时,服务端从查询集合 Q 中找到其相似查询 $SQ(q,Q)$,从而可以获得查询 $SQ(q,Q)$的结果集,然后将结果集中的移动对象作为候选对象进行判断后放入到查询 q 的初始结果集中,根据候选对象计算出查询 q 的最近邻距离 min$dist(t)$和搜索区域 $q.SR$,最后将查询 q 加入到查询集合 Q 中,并在查询更新时刻 t_{upt} 通过搜索 TPR 树索引得到查询 q 的初始结果.

根据定理 1 可知,对每个 CKNN 查询 q,若更新、插入或删除的移动对象在其查询窗口 $q.T=[t_{ref},t_{ref}+TL]$范围内落在搜索区域 $q.SR$ 之外,则查询 q 的结果不需要更新.由此我们引入了 SI-CNN 处理查询更新的算法,其思想如下:对于 CKNN 查询 q,当数据库发生变化时,若移动对象 p 位置落在查询 q 的搜索区域 $q.SR$ 之外,则 p 不会对查询 q 结果造成影响,忽略此对象;否则,作进一步的精化处理,判断对象 p 是否对查询 q 结果产生影响.另外,若落在查询 q 搜索区域之内的移动对象均为插入的新对象,则查询 q 结果更新不需要重新搜索 TPR 树索引,而仅仅利用查询 q 的最近邻距离 min$dist(t)$进行精化判断即可得到查询 q 更新后的结果.若有删除或更新的对象落在查询 q 的搜索区域 $q.SR$ 内,则利用最近邻距离 min$dist(t)$对这些对象进行精化判断,若对查询 q 结果产生影响,那么在更新查询 q 以后,必须重新计算搜索区域 $q.SR$.最后,必须对 TPR 树索引利用搜索区域 $q.SR$ 进行剪枝搜索,以得到正确的结果.

2.2 SI-CNN查询处理框架

传统 CKNN 查询处理机制对每个查询更新独立地执行一次 TPR 树索引扫描,对于大量需要同时更新的 CKNN 查询来说,这种重复的磁盘扫描方式效率十分低下.SI-CNN 查询处理框架采用共享查询计划(share query plan)的机制来支持大量并发的 CKNN 查询更新.在查询更新时刻 t_{upt},SI-CNN 算法首先在基于内存的查询表(圆形的查询搜索区域)和对象缓冲区(点对象)之间做连接扫描以更新查询,然后在查询表与 TPR 树索引(点对象)之间做一次连接运算,只需要一遍 TPR 树索引扫描,便可获得所有的对查询可能造成影响的候选移动对象集,将这些候选对象进行分组,并发送给不同的查询进行精化判断,得到最终的查询更新结果.

图 1 所示为 SI-CNN 系统框架,具体包括以下几个部分:

1) 查询表(query table),基于内存的序列表结构,用以存放用户提交的 CKNN 查询.其记录形式为五元组 $\langle QID,QLoc,k,T,SS\rangle$.其中,$QID$ 表示 CKNN 查询 q 的标识;$QLoc$ 表示参考时刻 t_{ref} 查询位置 $q.Loc$;k 表示 CKNN 查询最邻近对象个数;T 表示查询 q 的时间窗口范围;SS 表示查询 q 的搜索半径.在查询更新时刻 t_{upt} 必须更新查询表中的所有记录,将 $QLoc$ 更新为 t_{upt} 时的查询位置,T 起始时刻更新为 t_{upt},若此记录对应查询 q 的结果被更新,则需要重新计算搜索半径 SS.
2) 查询结果表(query results),基于磁盘的顺序表结构,存放所有 CKNN 查询的结果.其记录形式为二元组 $\langle QID,Result\rangle$.其中,$QID$ 表示 CKNN 查询 q 的标识;$Result$ 表示查询 q 的结果集$\{\langle R,T,D\rangle\}$,对于 $Result$ 的每个元组$\langle R,T,D\rangle$,R,T,D 分别表示查询 q 的 k 个最近邻对象、R 的有效时间范围、T 时间间隔内查询 q 的最近邻距离 min$dist(t)$.
3) 对象缓冲区(object buffer),基于内存的线性队列,用于缓存更新、插入或删除的移动对象.在更新时刻 t_{upt},所有对象被写入到磁盘中(更新数据页面和 TPR 树索引),并清空对象缓冲区.
4) TPR 树索引(TPR-tree),基于磁盘的移动对象索引结构,CKNN 查询通过扫描 TPR 树索引来查找最近邻

© 中国科学院软件研究所 http://www.jos.org.cn

的 k 个移动对象.

5) 增量结果表(incremental results),基于内存的线性链表结构,存放 SI-CNN 查询算法在进行查询更新时的候选移动对象(查询表分别与对象缓冲区、TPR 树索引连接操作时产生的中间结果).其元组形式为〈*QID*,*p*〉.其中,*QID* 表示记录对应的查询标识;*p* 表示移动对象(包括参考时刻位置、速度矢量等信息).增量结果表以查询标识 *QID* 进行排序,当链表中记录数目达到阈值时,将增量结果表中的记录更新到查询结果表中,同时清除缓存.

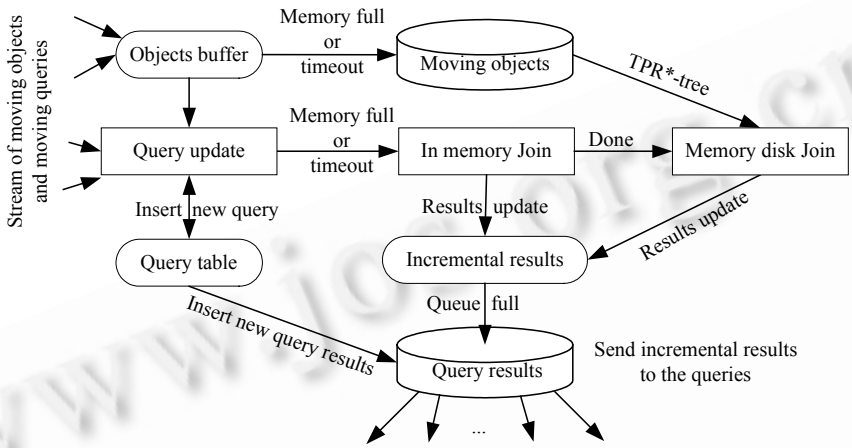


Fig.1 SI-CNN system framework

图 1 SI-CNN 系统框架

SI-CNN 框架将 CKNN 查询组织在查询表中,表中记录存放了对应查询的相关信息和搜索区域.由于每条查询记录所占用的空间非常小,所以整个查询表可以完全存放在内存中,同时便于 SI-CNN 算法在查询更新时进行查询搜索区域与移动点对象之间的连接操作和查询搜索区域的修改操作.当新的查询提交时,SI-CNN 算法生成新的查询记录并计算出其搜索区域,将此查询记录插入到查询表中.若查询表中记录的查询窗口结束时刻小于当前查询更新时刻 t_{upt},则从表中删除这一过时的查询记录.CKNN 查询结果集由于占用较大的空间而存放在基于磁盘的查询结果表中.移动对象存放在物理磁盘上,并利用 TPR 树进行索引,在每个更新时刻 t_{upt},必须对 TPR 树索引和数据页面进行更新.客户端提交的插入、删除或更新移动对象缓冲在对象缓冲区中,并在每个更新时刻 t_{upt} 与查询表进行连接判断后写入到磁盘索引及数据页面中.为了能够快速地执行 TPR 树索引与查询表之间的连接运算,且减少查询结果更新时额外的磁盘 I/O 操作,我们引入了基于内存的增量结果表来保存 SI-CNN 算法产生的中间结果,并在增量结果表记录超过阈值时,将查询结果更新批量地写入到查询结果表中.

2.3 SI-CNN查询处理算法

在更新时刻 t_{upt},SI-CNN 查询更新算法首先在对象缓冲区 O 和查询表 Q 之间做一次连接操作,计算出落在每个查询 q 搜索区域之内的移动对象,将其作为候选对象记录在增量结果表 *IR* 中;当 *IR* 满或连接操作完毕时,将 *IR* 中的更新记录批量写入到查询结果表 R 中,并标记需要进行 TPR 树索引搜索更新的查询;然后,算法在查询表 Q 和基于磁盘的 TPR 树索引之间做连接操作,将落在每个查询 q 搜索区域之内的移动对象记录在增量结果表 *IR* 中.同样地,当 *IR* 缓冲区满时,将查询更新批量写入到查询结果表 R 中.值得注意的是,算法在进行对象缓冲区 O 和查询表 Q 之间做连接操作时,若落在查询 q 搜索区域之内的对象均为插入的新对象,则显然,查询 q 结果的更新不需要搜索 TPR 树索引,只需判断这些新对象是否对查询结果有影响即可;若有更新、删除的对象落在查询 q 搜索区域内,则必须根据这些移动对象重新计算搜索区域,以保证算法在查询表 Q 和基于磁盘的 TPR 树索引之间做连接操作时剪枝搜索的正确性.

© 中国科学院软件研究所 http://www.jos.org.cn

算法 **1**. SI-CNN 查询更新算法.

输入:对象缓冲区 O,查询表 Q,查询结果序列 R,增量结果表 IR,TPR 树索引 T.

输出:更新的查询表 Q,查询结果序列 R.

步骤:

1. 在查询更新时刻 t_{upt}=0,对查询表 Q 中每个查询 q 的记录 E 依次进行下述操作:
   1.1. 若$\forall o\in O,\forall t\in q.T=[t_s,t_e]$,满足 $dist(q,o,t)>E.SS$,则更新记录 E,令 $E.QLoc=q.Loc,E.T=[t_{upt},t_e]$;
   1.2. 否则,对$\forall o\in O$,若$\exists T'\subset q.T=[t_s,t_e]$有$\forall t\in T'$,$dist(q,o,t)<E.SS$,则将元组$\langle E.QID,o\rangle$插入到增量结果表 IR 中;
   1.3. 若 IR 满或查询表 Q 访问完毕,则调用 SI-Flush 算法,将 IR 中的更新记录写入查询结果序列 R 和查询表 Q 中,并标记需要进行 TPR 树索引搜索的查询.
2. 将对象缓冲区 O 中所有对象写入磁盘数据页面,同时更新 TPR 树索引 T.
3. 对 TPR 树 T 进行深度优先搜索,从根节点开始,对每个访问节点:
   3.1. 若为非叶节点,则对节点中所有的矩形包围框 R:
      3.1.1. 若$\exists q\in Q,\exists T'\subset q.T$,使得$\forall t\in T',R_{(t)}\cap q.SR\neq\varnothing$,继续对节点进行深度优先搜索;
      3.1.2. 否则,裁剪此矩形.
   3.2. 若为叶节点,则对其中所包含的所有移动对象 p:
      3.2.1. 对$\forall q\in Q$,若$\exists T'\subset q.T=[t_s,t_e]$有$\forall t\in T'$,$dist(q,p,t)<q.SS$,则将元组$\langle q.QID,p\rangle$插入到增量结果表 IR 中;
      3.2.2. 若 IR 满或对 TPR 树 T 搜索完毕,则调用 SI-Flush 算法,将更新记录写入到查询结果序列 R,同时修改查询表 Q 中受影响查询 q 的搜索区域.
4. 得到更新的查询表 Q 和查询结果序列 R.

在 SI-CNN 查询更新算法中,步骤 1 在查询表 Q 与对象缓冲区 O 之间做基于内存的连接扫描,初步更新查询结果;步骤 3 在查询表 Q 与 TPR 树索引 T 之间做连接操作,查找落在查询表 Q 中每个查询 q 搜索区域之内的移动对象.采用深度优先策略对 TPR 树索引 T 进行搜索,若节点矩形包围框与查询 q 的搜索区域相交,则继续对此节点进行深度优先搜索.对于叶节点中的移动对象 p,查询集合 Q 中的任意查询 q,若 p 落在查询 q 搜索区域之内,则产生新的元组$\langle QID,p\rangle$插入到增量结果表 IR 中去.当增量结果表 IR 中的记录大小超过阈值或连接操作完毕时,将增量结果表 IR 中的更新记录批量写入到查询结果表中.

SI-Flush 算法用于将增量结果表 IR 中的更新记录批量写入到查询结果序列 R 中,同时修改受到影响的查询搜索区域.算法对 IR 链表进行顺序扫描,对于查询更新记录 $E=\langle QID,p\rangle$,从查询结果表 R 中得到具有标识 QID 查询 q 的结果集,若移动对象 p 对查询 q 的结果集产生影响,则将 p 加入到 q 的结果集中,同时修改查询搜索区域,直至访问完增量结果表 IR 的所有记录.在 SI-Flush 算法执行完毕之后,查询表 Q 中查询 q 的搜索区域已被修改,在此后的连接搜索时,可以更精确地对 TPR 树索引节点进行裁剪.

算法 **2**. SI-Flush 算法.

输入:查询表 Q,查询结果序列 R,增量结果表 IR.

输出:查询表 Q,查询结果序列 R.

步骤:

1. 对增量结果表 IR 中每条记录 $E=\langle QID,p\rangle$:
   1.1. 从查询结果序列 R 中查找具有 QID 标识的查询 q 的查询结果序列 R,对 R 中每个元组$\langle QID,\{\langle R,T,D\rangle\}\rangle$.
      1.1.1. 若$\exists T'\subset q.T,\forall t\in T'$满足 $dist(q,E.p,t)<D$,则将 p 加入到结果集中,同时,重新计算最近邻距离 D;
      1.1.2. 否则,忽略对象 p.
   1.2. 若查询 q 结果被修改,则重新计算搜索区域 $q.SR$,并更新查询表 Q.

© 中国科学院软件研究所 http://www.jos.org.cn

2. 若遍历完增量结果表 IR 所有记录,则清空 IR,并返回修改后的查询表 Q 和查询结果序列 R.

SI-AddQuery 算法用于处理客户端提交的新查询 q,在查询更新时刻 t_{upt} 计算初始结果集,并将查询 q 加入到查询表 Q 中.算法首先在查询表 Q 中查找 q 的相似查询 $SQ(q)$,以其结果集中的移动对象与对象缓冲区中的对象作为查询 q 的候选对象,根据候选对象计算出查询 q 的初始近似结果集,加入到查询结果序列中,并计算出查询 q 搜索区域 $q.SR$,生成新的记录$\langle q.QID,q.Loc,k,q.T,q.SS\rangle$,加入到查询表 Q 中.然后,利用 SI-CNN 查询更新算法与查询表 Q 中其他查询同时处理,以计算出准确的初始结果.若候选对象集合中对象个数 n 小于 CKNN 查询个数 k,则将查询 q 的搜索区域设为无穷大,即需要搜索整个 TPR 树,将初始结果集设为空,利用 SI-CNN 查询更新算法类似地一遍扫描求出查询 q 初始结果集,其代价是由于搜索区域过大而引起的额外的节点访问.不过,在每次 SI-Flush 算法操作之后会修改搜索区域,从而减少以后的 TPR 树索引节点搜索.

算法 **3**. SI-AddQuery 算法.

输入:查询表 Q,查询结果序列 R,查询 q.

输出:查询表 Q,查询结果序列 R.

步骤:

1. 将对象缓冲区 O 中的对象加入到查询 q 的候选对象集合中.
2. 对查询表 Q 中的所有查询 q',计算查询 q 和 q'的相似度 $QS(q,q')$,从中找到 q 的相似查询 $q'=SQ(q)$.
3. 从查询结果序列 R 中获得查询 q'的结果集,将 $q'.R$ 中的移动对象加入到查询 q 的初始近似结果候选对象.
   - 3.1. 若候选对象集合对象个数 $n\geq q.k$,则
     - 3.1.1. 从候选对象集合中计算出 q 的初始近似结果集$\{\langle R,T,D\rangle\}$,并将元组$\langle q.QID,\{\langle R,T,D\rangle\}\rangle$插入到查询结果序列 R 中;
     - 3.1.2. 根据查询 q 的初始近似结果集$\{\langle R,T,D\rangle\}$计算出查询 q 的搜索区域 $q.SR$,并将元组$\langle q.QID,q.QLoc,q.k,q.T,q.SS\rangle$插入到查询表 Q 中.
   - 3.2. 若候选对象集合对象个数 $n<q.k$,则
     - 3.2.1. 令 q 的初始近似结果集 $Results=\langle\varnothing,q.T,\infty\rangle$,将元组$\langle q.QID,Results\rangle$加入到查询结果序列 R 中;
     - 3.2.2. 生成新的元组$\langle q.QID,q.QLoc,q.k,q.T,\infty\rangle$,加入到查询表 Q 中.
4. 返回查询表 Q 与查询结果序列 R.

3 实验结果与分析

3.1 实验内容与设置

为了与文献[10,11]中的实验内容保持一致,我们使用文献[12]中基于道路网络的移动对象产生器来生成移动对象数据集和查询集合.产生器输入为德国城市 Oldenburg 的道路图,输出为在道路网络上运动的移动对象集合.数据集大小为 100k,移动对象用点坐标来表示,参考时刻 t_{ref} 每个移动对象选择一个距离其最近的道路节点为目的地进行运动.移动对象速度大小分为慢、中、快 3 种,以慢速度进行运动的移动对象在一个时间单元内所移动的距离为整个空间区域长度的 1/2500,中速度、快速度大小分别为慢速度的 5 倍和 25 倍.当移动对象到达目的地以后,会重新随机选择一个新目的地和速度大小进行运动.在以中速度运动的情况下,平均每个时间单元大约有 2k 个移动对象提交更新.在实验中,我们将时间单元大小设置为 15s.

实验数据集利用 TPR 树进行索引,TPR 树页面大小设置为 1k,中间节点扇出为 31,叶节点中可以保存大约 54 个移动点对象,索引树高度为 4 层.使用的页面缓存大小为 50k,即 50 个缓存页面,并采用最近使用(least recently used,简称 LRU)缓存替代策略.CKNN 查询产生方式与移动对象数据集产生方式相同,默认大小为 1k 个,查询时间窗口范围为$[T_{isu},T_{isu}+TL]$之间(T_{isu} 为查询提交时刻),其中,TL=50.查询近邻个数分别取 k=1,5,10,15,20.增量结果表 IR 缓冲区大小分别为 16k,24k,32k,40k,48k.可以存放 512,768,1 024,1 280,1 536 条更新记录.当查询集合中的查询由于查询时间窗口过时而被删除时,会插入新的查询.我们将查询数量 q_N 分别固定为 1k,2k,5k,7k,

© 中国科学院软件研究所 http://www.jos.org.cn

10k,平均每个时间单元会更新 20,40,100,140,200 个查询.查询结果序列页面大小为 4k,当 k=10 时,平均可以存放 7 个查询的结果集.SEA-CNN 算法及 CPM 算法中网格数目的大小参照文献[11]设置为 128×128.

实验硬件环境为 Celeron 2.4GHz 的 CPU,256MB DDR 内存和 7200 RPM 硬盘.实验参数见表 1.

Table 1 Experimental parameters

表 **1** 实验参数

Parameters	Default value	Range
Number of moving objects (k)	100	50,100,150,200,250
Number of queries (k)	1	1,2,5,7,10
Number of neighbors	10	1,5,10,15,20
Velocity of objects and queries	Middle	Slow, middle, high
Buffer size (k)	32	16,24,32,40,48

3.2 实验结果与分析

为了衡量 SI-CNN 连续 k 近邻查询算法的性能,我们比较了 SI-CNN 算法与 SEA-CNN 及 CPM 算法在处理大量并发 CKNN 查询时的节点(网格)访问代价和 CPU 计算时间代价.

如图 2(a)和图 2(b)所示为当固定 k=10 和增量结果表缓存大小 IR=32k 时,SI-CNN 算法与 SEA-CNN 及 CPM 算法在节点(网格)访问次数和 CPU 计算时间上的比较.从图中可以看出,SI-CNN 算法查询更新性能远远好于 SEA-CNN 算法,也稍好于 CPM 算法.这是由于 SEA-CNN 算法当移动对象更新后位于查询影响区域之内时,便立即对查询进行更新,极大地降低了查询更新性能;CPM 算法则优化了查询更新算法,在查询或移动对象更新时,只需访问最少的格网;而 SI-CNN 算法则利用移动对象缓冲区和增量结果表对查询更新进行批量处理,因而具有很好的 CKNN 查询更新性能.

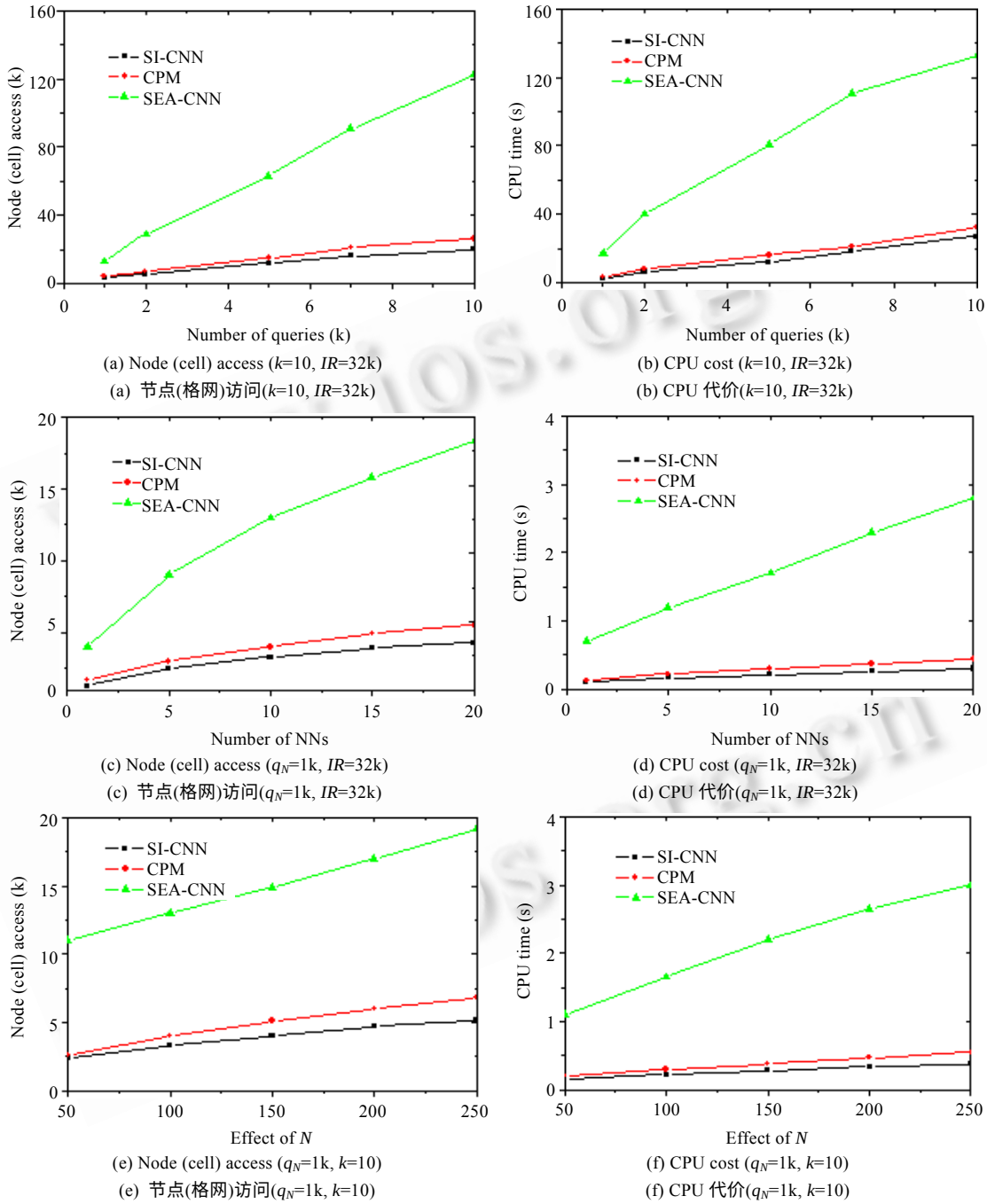
如图 2(c)和图 2(d)所示为当固定查询数目 q_N=1k 和 IR=32k 时,SI-CNN 算法与 SEA-CNN 及 CPM 算法在节点(网格)访问次数和 CPU 计算时间上的比较.可以看出:随着 k 个数的增加,CKNN 查询所需节点(网格)访问次数、CPU 计算时间基本上呈线性增长,这是因为随着 k 个数的增加,查询结果集也随之增加,查询影响区域会增大,因此查询更新需要访问更多的索引节点(网格);而 SI-CNN 算法查询结果序列所占用的磁盘页面呈线性增长,从而在将增量结果表中的更新记录批量写入查询结果序列时,需要更多的磁盘 I/O,而且查询搜索区域的计算也会由于结果集变大、变得复杂而需要更多的计算时间.

如图 2(e)和图 2(f)所示为当固定查询数目 q_N=1k 和 IR=32k 时,SI-CNN 算法与 SEA-CNN 及 CPM 算法在节点(网格)访问次数和 CPU 计算时间上的比较.从图中可以看出:随着移动数据集数目的增加,SEA-CNN 和 CPM 算法查询更新性能也会下降,但幅度并不是很大,这是由于算法使用固定网格大小,对算法性能的影响主要是由于在每个时间单元下更多的移动对象提交更新所造成的;SI-CNN 算法则是由于 TPR 树索引高度维持在 4 层,因而查询更新代价也主要是由于在每个时间单元下,更新的移动对象数目增加所造成的.但是,当移动对象数据集很大、TPR 树索引高度增加时,SI-CNN 算法节点访问代价将会显著增加.同样的问题也会出现在 SEA-CNN 和 CPM 算法中,由于每个格网中移动对象数目的增加,需要更多的磁盘页面来进行存放.值得注意的是,SEA-CNN 和 CPM 算法查询更新性能并不能完全由格网访问次数来衡量,而且还要考虑到每个格网所对应的磁盘页面个数.而在 SI-CNN 算法中,节点访问次数与磁盘页面个数是一致的.另外,由于 SEA-CNN 及 CPM 算法使用固定网格对移动对象数据集进行索引,虽然查询算法比较简单、具有良好的查询更新性能,但却导致了高昂的格网索引更新代价.相比之下,SI-CNN 算法所基于的 TPR 树在采用了批量更新算法以后,具有很好的索引更新性能,因而具有良好的实用价值.

如图 3 所示为增量结果表缓冲区大小对 SI-CNN 查询更新性能的影响.从图中可以看出:随着增量结果表 IR 缓存大小的增加,SI-CNN 查询性能会随之变好,这是由于 SI-CNN 算法性能主要受 3 部分影响:TPR 树的搜索代价,将增量结果表写入到查询结果序列时的磁盘 I/O 代价和查询搜索区域重新计算代价.增量结果表 IR 缓存越大,所需的查询结果序列磁盘访问次数会越少,CPU 计算时间也会减少.但是,由于 IR 缓存越大,查询搜索区域的更新时间间隔也越大,从而会由于搜索区域不精确而导致额外的 TPR 树节点访问,相应地需要额外的 CPU 计算

© 中国科学院软件研究所 http://www.jos.org.cn

代价.因此,随着增量结果表缓存大小的增加,SI-CNN 查询更新性能的提高并不是非常明显.在实验测试中,我们将其大小固定为 32k.



(a) Node (cell) access (k=10, IR=32k)
(a) 节点(格网)访问(k=10, IR=32k)

(b) CPU cost (k=10, IR=32k)
(b) CPU 代价(k=10, IR=32k)

(c) Node (cell) access (q_N=1k, IR=32k)
(c) 节点(格网)访问(q_N=1k, IR=32k)

(d) CPU cost (q_N=1k, IR=32k)
(d) CPU 代价(q_N=1k, IR=32k)

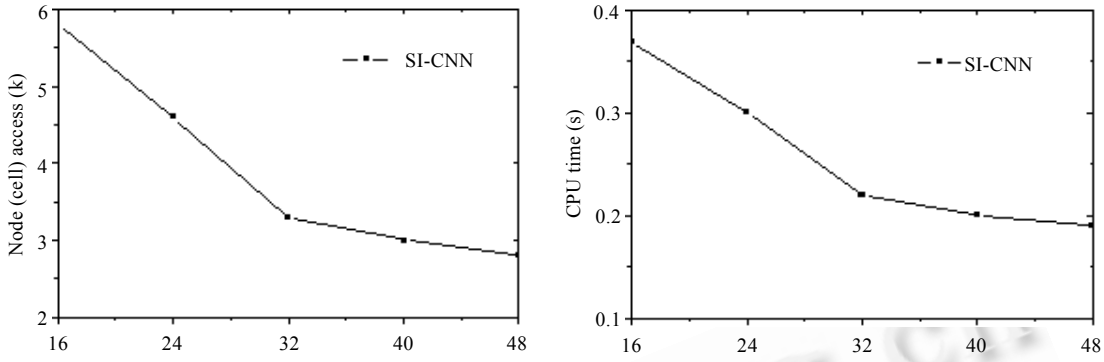
(e) Node (cell) access (q_N=1k, k=10)
(e) 节点(格网)访问(q_N=1k, k=10)

(f) CPU cost (q_N=1k, k=10)
(f) CPU 代价(q_N=1k, k=10)

Fig.2 CKNN queries update performance comparison

图 2 CKNN 查询更新性能比较

© 中国科学院软件研究所 http://www.jos.org.cn



(a) Node access (k=10, q_N=1000)
(a) 节点访问(k=10, q_N=1000)
(b) CPU cost (k=10, q_N=1000)
(b) CPU 代价(k=10, q_N=1000)

Fig.3 Impact of IR buffer size

图 3 增量结果表缓冲区大小的影响

4 结 论

本文针对基于 TPR 树索引的大量并发 CKNN 查询处理,提出了一种可伸缩的增量连续 k 近邻查询处理 SI-CNN 框架,通过引入搜索区域进行裁剪以减少查询更新所需要的磁盘访问代价,并引入了增量结果表批量的更新查询结果集.SI-CNN 框架能够高效处理大量并发的 CKNN 查询,且具有良好的可伸缩性.基于 SI-CNN 框架提出了一种支持更新的 SI-CNN 查询处理算法.SI-CNN 算法具有以下两个特征:1) 增量的(incremental),基于上次查询结果增量的更新查询;2) 可伸缩性(scalable),即支持查询集合中加入或删除查询和移动对象数据集的插入、删除等动态更新操作.实验结果与分析表明,SI-CNN 框架和 SI-CNN 算法可以很好地支持大量并发的 CKNN 查询处理,具有很好的实用价值.SI-CNN 框架不仅适用于面向移动对象的 CKNN 查询处理,而且只需简单的扩充,便可应用于基于 R 树索引的 CKNN 查询和连续窗口查询等.

References:

[1] Korn K, Muthukrishnan S, Karciauskas G, Saltenis S. Influence sets based on reverse nearest neighbor queries. In: Naughton JF, Bernstein PA, eds. Proc. of the 2000 ACM SIGMOD Int'l Conf. on Management of Data. ACM Press, 2000. 201−212.

[2] Šaltenis S, Jensen CS, Leutenegger ST, Lopez MA. Indexing the positions of continuously moving objects. In: Naughton JF, Bernstein PA, eds. Proc. of the 2000 SIGMOD Int'l Conf. on Management of Data. ACM Press, 2000. 331−342.

[3] Benetis R, Jensen CS, Karčiauskas G, Šaltenis S. Nearest neighbor and reverse nearest neighbor queries for moving objects. In: Nascimento MA, Özsu MT, Zaïane OR, eds. Proc. of the 2002 IDEAS, Int'l Symp. on Database Engineering & Applications. IEEE Computer Society, 2002. 44−53.

[4] Tao YF, Papadias D. Spatial queries in dynamic environment. ACM Trans. on Database Systems TODS, 2003,28(2):101−139.

[5] Iwerks GS, Samet H, Smith KP. Continuous k-nearest neighbor queries for continuously moving points with updates. In: Freytag JC, Lockemann PC, Abiteboul S, Carey MJ, Selinger PG, Heuer A, eds. Proc. of the 29th VLDB Int'l Conf. on Very Large Data Bases. Morgan Kaufmann Publishers, 2003. 512−523.

[6] Tao YF, Papadias D. Time-Parameterized queries in spatio-temporal databases. In: Franklin MJ, Moon B, Ailamaki A, eds. Proc. of the 2002 ACM SIGMOD Int'l Conf. on Management of Data. ACM Press, 2002. 334−345.

[7] Mokbel MF, Xiong XP, Aref WG. SINA: Scalable incremental processing of continuous queries in spatiotemporal databases. In: Weikum G, König AC, Deßloch S, eds. Proc. of the 2004 SIGMOD Int'l Conf. on Management of Data. ACM Press, 2004. 623−634.

[8] Hu HB, Xu JL, Lee DL. A generic framework for monitoring continuous spatial queries over moving objects. In: Özcan F, ed. Proc. of the 2005 SIGMOD Int'l Conf. on Management of Data. ACM Press, 2005. 479−490.

© 中国科学院软件研究所 http://www.jos.org.cn

[9] Yu XH, Pu KQ, Koudas N. Mointoring *k*-nearest neighbour queries over moving objects. In: Proc. of the 21st ICDE Int'l Conf. on Data Engineering. IEEE Computer Society, 2005. 631−642.

[10] Xiong XP, Mokbel MF, Aref WG. SEA-CNN: Scalable processing of continuous *k*-nearest neighbor queries in spatio-temporal databases. In: Proc. of the 21st ICDE Int'l Conf. on Data Engineering. IEEE Computer Society, 2005. 643−654.

[11] Mouratidis K, Hadjieleftheriou M, Papadias D. Conceptual partitioning: An efficient method for continuous nearest neighbor monitoring. In: Özcan F, ed. Proc. of the 2005 SIGMOD Int'l Conf. on Management of Data. ACM Press, 2005. 634−645.

[12] Brinkhoff T. A framework for generating network based moving objects. Geoinformatica, 2002,2(6):153−180.

廖巍(1980－),男,湖北襄樊人,博士生,主要研究领域为空间数据库,时空数据库.

熊伟(1976－),男,博士,主要研究领域为空间数据库.

王钧(1978－),男,博士生,主要研究领域为空间决策技术.

景宁(1963－),男,博士,教授,博士生导师,CCF 高级会员,主要研究领域为地理信息系统,数据库技术.

钟志农(1975－),男,博士,副教授,主要研究领域为地理信息系统.

© 中国科学院软件研究所 http://www.jos.org.cn