文章编号:1001-2486(2007)01-0081-05

基于扩展时空距离度量的连续 k 近邻查询方法*

廖 巍,吴秋云,陈宏盛,景 宁,钟志农
(国防科技大学 电子科学与工程学院,湖南 长沙 410073)

摘 要:针对基于 TPR 树索引的连续 k 近邻查询,引入了一种新的时空距离度量最小最大距离函数作为 TPR 树索引搜索时节点剪枝上界。提出了一种采用最优优先策略的基于扩展时空距离度量的连续 k 近邻查询 STM-CNN 算法,利用最小距离函数进行 TPR 树索引节点搜索时访问排序,并使用最小最大距离函数对 TPR 树索引进行剪枝界定。

关键词:连续 k 近邻查询;TPR 树;最小最大距离函数;STM-CNN 算法
中图分类号:TP392 **文献标识码**:A

Continuous k-nearest Neighbor Queries Based on Extended Spatio-temporal Distance Metrics

LIAO Wei, WU Qiu-yun, CHEN Hong-sheng, JING Ning, ZHONG Zhi-nong
(College of Electronic Science and Engineering, National Univ. of Defense Technology, Changsha 410073, China)

Abstract: In order to process CKNN queries on the basis of TPR-tree efficiently, a new spatio-temporal distance metrics minmaxdist(t) was presented as a pruning upper bound. Also a CKNN queries algorithm STM-CNN which can search in best-first manner was developed by means of mindist(t) and minmaxdist(t) metrics, in which STM-CNN algorithm visits TPR-tree nodes according to mindist(t) order, and pruning the nodes with minmaxdist(t).

Key words: CKNN queries; TPR-tree; minmaxdist(t); STM-CNN algorithm

连续 k 近邻(continuous k-nearest neighbor, CKNN)查询是时空数据库中重要的查询类型,用于连续查找 k 个距离查询点最近的对象。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。与传统的 KNN 静态空间查询一次计算便可得到结果不同,在时空数据库中由于查询对象和/或查询本身的动态性,CKNN 查询返回结果通常带有时态语义,因而查询处理算法更加复杂。

CKNN 查询所处理的对象通常为时空数据集,其中包含大量移动对象和静态对象,分别以 R 树和 TPR 树索引进行存储管理,查询本身也可能是静态或动态的。传统 KNN 静态空间查询通常采用最小距离(mindist)和最小最大距离(minmaxdist)两种空间距离度量[1]来对 R 树进行剪枝界定,并使用深度优先(depth-fist)、最优优先(best-first)等搜索算法[2]来查找 k 个距离查询点最近的对象。但是这种静态空间距离度量对于 CKNN 查询,尤其是在处理基于 TPR 树(TPR-tree)[3]索引的移动对象动态查询时并不适合于搜索时节点的剪枝界定。

为了解决基于 TPR 树索引的大量移动对象 CKNN 动态查询问题,研究者提出了基于最小距离函数度量[6-7]、影响时间度量[4]以及搜索区域剪枝[5]等方法来加速 CKNN 查询的搜索。Benetis 等人提出的 Find-NN 算法[6]利用最小距离函数 mindist(t)作为连续近邻查询的裁剪准则来对 TPR 树进行剪枝搜索,其限制是处理连续单近邻查询且只能使用并非最优的深度优先遍历算法。Tao 等人在文献[7]中对 Find-NN 算法进行扩展以支持连续 k 近邻查询,但仍采用深度优先算法对 TPR 树索引进行搜索遍历。

* 收稿日期:2006-05-16
基金项目:国家自然科学基金资助项目(60472031)
作者简介:廖巍(1980—),男,博士生。

1 基于 TPR 树时空距离模型

传统的 KNN 静态空间查询采用最小距离与最小最大距离来对 R 树索引进行剪枝搜索。在时空动态环境下,对上述两种空间距离度量进行扩展可用于 TPR 树索引的剪枝搜索。在 d 维欧氏空间中,给定参考时刻 t_{ref}、查询点 q 及其运动矢量 $\boldsymbol{V}$,包含移动对象的矩形包围框 MBR 及速度包围框 VBR。最小距离函数(mindist(t))表示在未来时刻 t 查询点 q 距离矩形包围框 MBR 中移动对象的最小距离。Find-NN 算法利用最小距离函数对 TPR 树进行深度优先遍历搜索。

为了支持最优优先等搜索策略,我们引入了新的时空距离度量最小最大距离函数(minmaxdist(t)),其含义表示在 t 时刻保证 MBR 中存在至少一个移动对象能够落在查询点 q 距离范围内的最小距离。最小距离函数和最小最大距离函数分别给出了在任意时刻 t 查询点 q 到包含移动对象的 MBR 距离的下界和上界。利用这两个时空距离度量可以对基于 TPR 树索引的连续 k 近邻查询时进行节点访问排序和剪枝界定。

定义 1 给定 d 维欧氏空间 $E_{(d)}$ 中运动点 p 在参考时刻 t_{ref} 时坐标 $P=<p_1,p_2,\cdots,p_d>$,及速度矢量的上界 $V^-=<v_1^-,v_2^-,\cdots,v_d^->$,$V^+=<v_1^+,v_2^+,\cdots,v_d^+>$,$<v_i^-,v_i^+>$ 表示在参考时刻 t_{ref} 第 i 维运动速度下界和上界。则在任意时刻 t,点 p 所能够到达的范围定义为点 p 的邻域矩形 Range(p,t)。

$$\text{Range}(p,t)=<P+V^-\times t,P+V^+\times t>$$

最小最大距离函数(minmaxdist(t))为 d 维欧氏空间中任意时刻 t 查询点 q 到矩形 R 中任意对象或子节点 MBR 的最小距离。最小最大距离函数定义了连续近邻搜索时节点裁剪准则,在连续查询时间窗口范围内任意时刻 t,若节点到查询点 q 的最小距离函数始终大于最小最大距离函数,则对其进行剪枝界定。考虑到 TPR 树节点的 MBR 并非最小的,且 MBR 中移动对象的运动无法获得,对静态空间中的最小最大距离加入时态语义进行扩展所得到的最小最大距离函数在实际中很难进行具体的定义和计算。为简单起见,我们将最小最大距离函数定义如下。

定义 2 d 维欧氏空间 $E_{(d)}$ 中,假定 p 为参考时刻 t_{ref} 矩形框 R 顶点中距离查询点 q 为 minmaxdist(q, R)的顶点,则查询点 $q_{(t)}$ 在任意时刻 t 距离同维矩形 $R_{(t)}$ 的最小最大距离函数 minmaxdist(q,R,t)定义为 q 到顶点 p 邻域矩形 Range(p,t)的最大距离。

$$\text{minmaxdist}(q,R,t)=\max\Big(\sum_{1\leqslant k\leqslant d}|q_k^t-p_k^t|^2,\forall P_{(t)}=<p_1^t,p_2^t,\cdots,p_d^t>\in\text{Range}(p,t)\Big)$$

定理 1 给定任意时刻 t 查询点 q 及包含移动对象集合 $O_{(t)}=\{\boldsymbol{o}_i^t,1\leqslant i\leqslant m\}$ 的矩形 R,则下述命题为真:

$$\exists\, o_{(t)}\in O_{(t)},\|(q_{(t)},o_{(t)})\|\leqslant\text{minmaxdist}(q,R,t)$$

证明 由定义 2 可知,在参考时刻 t_{ref} 至少存在一个对象落在 $\|q,p\|=\text{minmaxdist}(q,R)$ 范围内,令此对象为 o,在任意时刻 t 其邻域矩形为 Range(o, t),由于 o 的运动速度落在矩形 R 的速度包围框之内,因此在任意时刻 t,查询点 q 到 Range(o,t)的最大距离始终小于点 q 到 Range(q,t)的最大距离,而 o 包含在其邻域矩形 Range(o,t)中,则有 $\|(q_{(t)},o_{(t)})\|\leqslant\text{minmaxdist}(q,R,t)$,得证。

2 连续近邻查询算法 STM-CNN

利用上述给定的时空距离度量最小距离函数 mindist(t)和最小最大距离函数 minmaxdist(t),参照静态空间中近邻查询的搜索和剪枝算法,下面给出采用最优优先搜索策略的连续 k 近邻查询 STM-CNN 算法。STM-CNN 算法假定用户提交具有时间结束条件的查询,而对于其他结束条件的连续近邻查询则不支持(如查询‘在得到预定次数结果变化后终止查询’)。为简单起见,我们令连续近邻查询时间窗口范围为 $[t_{isu},\ t_{isu}+TL]$,其中 t_{isu} 表示查询的提交时刻。

定义 3 给定 CKNN 查询及时间窗口 $[t_{isu},\ t_{isu}+TL]$,设查询返回结果集大小为 n,则查询时间窗口被 $n+1$ 个时间点分割为 n 个时间间隔,在每个时间间隔内查询结果相同。我们称这些时间点为分

割点(split),特别地,时间点 t_{isu} 和 $t_{isu}+TL$ 称为边界分割点。

连续 k 近邻查询结果是形式为$<R,T>$元组的集合,其中 R 表示满足当前瞬时查询条件的结果集,T 则表示 R 的有效时间间隔。为了能够支持基于扩展时空距离度量的连续 k 近邻查询算法,将返回结果形式扩展为$<R,T,D>$,其中 R,T 含义不变,仍然表示查询结果、有效时间间隔,而 D 则表示在 T 时间间隔内查询结果 R 中对象(矩形包围框或移动对象)到查询点 q 之间的距离函数的集合。对于单近邻 STM-CNN 查询而言,若 R 为矩形包围框,则 D 表示 R 在 T 时间间隔内距离查询点 q 的最小最大距离函数 $\mathrm{minmaxdist}(q,R,t)$;若 R 为移动对象 p,则 D 表示在 T 时间间隔内 p 到查询点 q 的距离函数 $\mathrm{dist}(q,p,t)$。

具体区别在于 k 近邻 STM-CNN 算法结果集元组$<R,T,D>$中 D 是 k 个二次时间函数 $\mathrm{dist}(q,t,j)$的集合,其中 $1\leqslant j\leqslant k$。在进行近邻查询搜索时,只有当节点满足 $\forall t\in[t_{isu},t_{isu}+TL],\forall j=1,2,\cdots,k,\mathrm{mindist}(q,R,t)>\mathrm{dist}(q,t,j)$时,才会对此节点进行裁剪;否则,对于结果集中任意元组$<R,T,D>$,若存在某个时间间隔$[t_s,t_e]\subseteq T,\forall t\in[t_s,t_e],\exists j(1\leqslant j\leqslant k)$使得 $\mathrm{minmaxdist}(q,R',t)<\mathrm{dist}(q,t,j)$,则将时刻 t_s、t_e 加入到分割点序列中去,同时删除此元组,并增加三个新元组$<R_1,T_1,D_1>$,$<R_2,T_2,D_2>$,$<R_3,T_3,D_3>$,其中 $R_1=R_3=R$、$R_2=R'\cup(R-R_j)$、$D_2=\mathrm{minmaxdist}(q,R',t)\cup(D-\mathrm{dist}(q,t,j))$、$D_1=D_3=D$。$T_2=[t_s,t_e]$,$T_1\cap T_2\cap T_3=\Phi$,$T_1\cup T_2\cup T_3=T$,其含义是在时间间隔 T_1、T_2、T_3 内距离查询点 q 最近的矩形框分别为 R_1、R_2、R_3,相应的距离函数分别为 D_1、D_2、D_3。另外,由于新增的元组同样可能存在某个时间间隔$[t_s,t_e]\subseteq T,\forall t\in[t_s,t_e],\exists j(1\leqslant j\leqslant k)$使得 $\mathrm{minmaxdist}(q,R',t)<\mathrm{dist}(q,t,j)$,因此算法必须对新增元组进行上述判断,同时修改精确距离度量以减少搜索时不必要的节点访问。

算法 1　k 近邻 STM-CNN 算法。

输入:TPR 树索引、查询点 q、查询时间窗口$[t_{isu},t_{isu}+TL]$;

输出:近邻对象集合 NN、分割点序列 $\mathrm{split}(q)$。

步骤:

(1) 初始化近邻对象集合 $NN=\Phi$、分割点序列 $\mathrm{split}(q)=\Phi$,距离函数 $\mathrm{dist}(t)=\infty$,优先队列 $H=\Phi$。

(2) 获得 TPR 树索引根节点,计算根节点中每个矩形包围框 R 在 t_{isu} 时刻的最小最大距离 $\mathrm{minmaxdist}(q,R,t_{isu})$,从中找到 k 个最小 $\mathrm{minmaxdist}(q,R,t_{isu})$的矩形包围框 R_j,令 $\mathrm{dist}(t,j)=\mathrm{minmaxdist}(q,R_j,t)$,$\mathrm{split}(q)=\{t_{isu},t_{isu}+TL\}$。对所有的节点矩形包围框 R。

若 $\forall t\in[t_{isu},t_{isu}+TL],\forall j=1,2,\cdots,k(\mathrm{mindist}(q,R,t)>\mathrm{dist}(q,t,j))$,裁剪此节点;否则,令 $M(R,q)=\int_{t_{isu}}^{t_{isu}+TL}\mathrm{mindist}(q,R,t)\mathrm{d}t$,将元组 $<M(R,q),R>$ 插入到优先队列 H 中去。

(3) 若 H 不为空,则重复以下操作。

1) 从优先队列 H 中取出具有最小 key 值的元组 E。

2) 若 E 是非叶节点,则对节点 E 中的所有矩形包围框 R。

① 若 $\forall t\in[t_{isu},t_{isu}+TL],\forall j=1,2,\cdots,k(\mathrm{mindist}(q,R,t)>\mathrm{dist}(q,t,j))$,则裁剪此矩形;

② 对于任意 $i(0<i<n)$,其中 n 为分割点集合 $\mathrm{split}(q)$的大小,若有 $\forall t\in[t^-,t^+](t_i\leqslant t^-\leqslant t^+\leqslant t_{i+1})$,$\exists j=1,2,\cdots,k$,使得 $\mathrm{minmaxdist}(q,R,t)<\mathrm{dist}(q,t,j)$,令 $\mathrm{dist}(q,t,j)\leftarrow\mathrm{minmaxdist}(q,R,t)(t\in[t^-,t^+])$,同时修改分割点集合,令 $\mathrm{split}(q,t)\leftarrow\mathrm{split}(q,t)\cup\{t^-,t^+\}$;

③ 令 $M(R,q)=\int_{t_{isu}}^{t_{isu}+TL}\mathrm{mindist}(q,R,t)\mathrm{d}t$,将元组 $<M(R,q),R>$ 插入到优先队列 H 中去。

3) 若 E 是叶节点,则对节点 E 中的所有对象 p,且 $p\neq q$:

① 若有 $\forall t\in[t_{isu},t_{isu}+TL],\forall j=1,2,\cdots,k(\mathrm{dist}(q,p,t)>\mathrm{dist}(q,t,j))$,则跳过 p;

② 对于任意 $i(0<i<n)$,其中 n 为分割点集合 $\mathrm{split}(q)$的大小,若有 $\forall t\in[t^-,t^+](t_i\leqslant t^-\leqslant t^+\leqslant t_{i+1})$,$\exists j=1,2,\cdots,k$,使得 $\mathrm{minmaxdist}(q,p,t)<\mathrm{dist}(q,t,j)$,令 $\mathrm{dist}(q,t,j)\leftarrow\mathrm{minmaxdist}(q,p,$

t)($t\in[t^-,t^+]$),同时修改分割点集合令 split(q,t)←split(q,t)∪ $\{t^-,t^+\}$。

4) 从优先队列 H 中删除元组 E。

(4) 输出查询结果集 NN,分割点集合 split(q)。

3 实验结果与分析

3.1 实验内容与设置

实验数据采用随机生成,数据集大小为 100KB,移动对象用点坐标来表示,均匀分布在10 000×10 000空间域内。移动对象运动用如下方式模拟,从 LA[Tiger](包含 128K 个二维矩形框)实际数据集中抽取 5000 个 MBR,其中心作为移动对象运动目的地,参考时刻 t_{ref}每个移动对象随机选择一个目的地运动,速度大小在[10,30]之间均匀分布,移动对象到达目的地后则会重新随机选择一个新目的地和速度大小进行运动。数据集中移动对象不会消失或加入新的对象,只存在速度更新操作。令 $o.Dist$ 表示移动对象起始地与目的地之间的距离,$o.vel$ 表示此对象的运动速度大小,则此对象在 $o.Dist/o.vel$ 时间间隔后发出更新请求。数据集的 $o.Dist$ 和 $o.vel$ 平均值大小分别为 3000 和 20。

我们使用 TPR 树对实验数据集进行索引,TPR 树页面大小设置为 1K,中间节点扇出大约为 31,叶节点中可以保存 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,并使用最近使用(LRU)缓存替代策略。查询性能用处理连续 k 近邻查询所需要的平均节点访问次数(NA)和 CPU 计算时间来衡量。连续 k 近邻查询产生方式如下:(1)查询点 q 位置在10 000×10 000空间域内随机分布;(2)查询点 q 速度大小在[5,20]均匀分布,并沿着随机选择的目的地方向运动;(3)查询时刻窗口范围为[T_{isu}, $T_{isu}+TL$](T_{isu}为查询提交时刻),其中 $TL=10,20,30,40,50$。在每个时间单元用户提交 10 个连续 k 近邻查询。

实验硬件环境为 Celeron 2.4GHz 的 CPU,256MB DDR 内存和 5400 RPM 硬盘。

3.2 实验结果与分析

为了衡量 STM-CNN 算法的搜索代价,将近邻查询对象个数固定为 $k=1$ 和 $k=5$,并将查询时间窗口范围由 10 个时间单元逐渐增加到 50 个单元,以比较算法在不同查询条件下的节点访问代价和 CPU 计算时间。

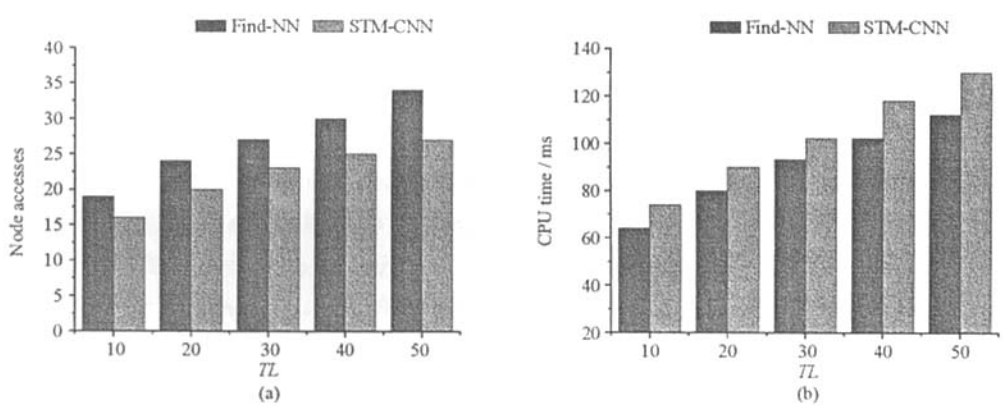

图 1 单近邻连续查询算法性能比较

Fig. 1 Single CKNN queries algorithms performance comparison

图 1 和图 2 分别比较了 STM-CNN 算法与 Find-NN 算法在 $k=1$ 和 $k=5$ 时查询所需的节点访问代价和 CPU 计算时间代价。从图中可以看出,随着查询窗口 TL 的增加,STM-CNN 算法与 Find-NN 算法的节点访问次数和 CPU 计算时间都呈线性增长的趋势。这是由于查询时间窗口 TL 越大,其返回结果集中元组数目会越多,因此在近邻搜索时需要访问更多的 TPR 树索引节点,相应的 CPU 计算代价也会由于结果集变大而需要较多的时间。由图 1(a)和图 2(a)可看出,在同等条件下,STM-CNN 算法比 Find-NN 算法需要更少的节点访问次数,尤其是在近邻对象个数 k 和查询窗口 TL 比较大时,STM-CNN 算法比 Find-NN 算法有更好的查询性能,其代价是较高的 CPU 计算时间,但是与磁盘访问的代价

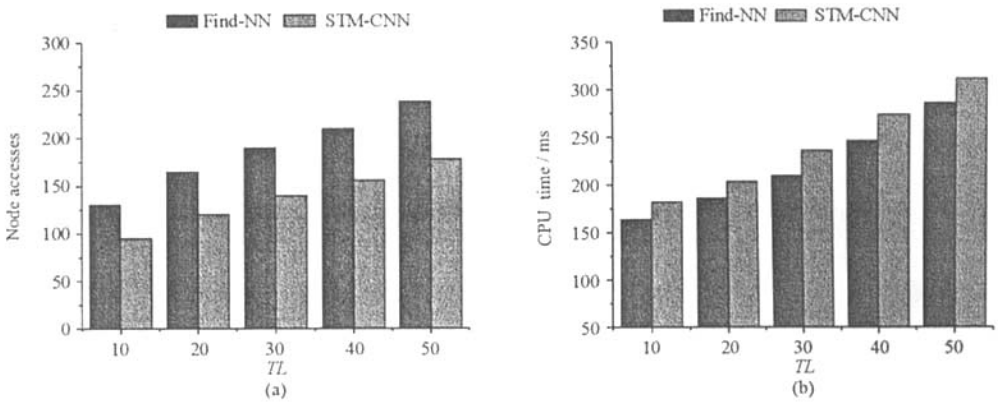

图 2　k 近邻连续查询算法性能比较($k=5$)

Fig. 2　CKNN queries algorithms performance comparison ($k=5$)

相比,可以忽略不计。

实验结果表明,采用最优优先访问策略的 STM-CNN **算法性能明显优于基于深度优先策略的** Find-NN **算法。由于** STM-CNN **算法使用全局优先队列来对节点访问顺序进行排列,在每次节点搜索时从中选取最好的节点进行访问,并同时对搜索裁剪距离度量进行修改,缩小搜索区域以减少不必要的节点访问。而** Find-NN **算法在搜索时只能对节点访问顺序进行局部最优选择,且只有在访问叶节点时才会对搜索裁剪距离度量进行修改,因此不可避免地导致额外的节点访问。**

4　结 论

针对基于 TPR **树的连续 k 近邻查询,提出了一种新的时空距离度量最小最大距离函数作为搜索裁剪时的上界,并给出了一种采用最优优先策略的连续 k 近邻查询** STM-CNN **算法。与基于最小距离函数的** Find-NN **算法等现有连续 k 近邻查询算法相比,**STM-CNN **算法具有更好的查询磁盘搜索性能,其代价是稍高的** CPU **计算时间。**

参 考 文 献:

[1] Roussopoulos N, Kelley S, Vincent F. Nearest Neighbor Queries[C]//Proc. of the 1995 ACM SIGMOD Intl. Conf. on Management of Data, 1995.

[2] Hjaltason G R, Samet H. Distance Browsing in Spatial Databases[C]//ACM Transactions on Database Systems TODS, 1999.

[3] Saltenis S, Jensen C S, et al. Indexing the Positions of Continuously Moving Objects[C]//Proc. of the 2000 SIGMOD Intl. Conf. on Management of Data, 2000.

[4] Tao Y P, Papadias D. Time-parameterized Queries in Spatio-temporal Databases[C]//Proc. of the 2002 ACM SIGMOD Intl. Conf. on Management of Data, 2002.

[5] Xiong X P, Mokbel M F, Aref W G. SEA-CNN: Scalable Processing of Continuous k-nearest Neighbor Queries in Spatio-temporal Databases[C]//Proc. of the 21st ICDE Intl. Conf. on Data Engineering, 2005.

[6] Benetis R, Jensen C S, Karciauskas G, et al. Nearest Neighbor and Reverse Nearest Neighbor Queries for Moving Objects[C]//Proc. of IDEAS, 2002.

[7] Tao Y F, Papadias D. Spatial Queries in Dynamic Enviroment[J]. ACM Transactions on Database Systems TODS, 2003.