动态网络空间中的 k-NN 查询

殷晓岚[1,2]
(1.中国科学院软件研究所软件工程技术研发中心,北京 100190;2.中国科学院研究生院,北京 100049)

摘 要: 随着无线通讯应用的持续增长和定位技术的发展,如何有效率的应答大量移动对象的查询请求以及基于位置的服务(location-based services LBS)变得越来越重要,k-NN 查询是其中的重要服务功能.本文提出了一种解决动态网络中静态对象 k-NN 查询算法,该算法先将网络以目标对象为中心进行网络划分,通过定位原始对象在网络上的位置来计算位置相关查询.同时还分析了算法的复杂性,给出了实验比较.

关键词: 移动对象; 空间数据网络库; 距离索引; k-NN
中图分类号: TP311 **文献标识码:** A **文章编号:** 0372-2112 (2011) 02-0389-06

k-Nearest Neighbors Query in Dynamic Spatial Network Databases

YIN Xiao-lan[1,2]
(1. *Technology Center of Software Engineering, Institute of Software, Chinese Academy of Sciences, Beijing* 100190, *China*;
2. *Graduate School, Chinese Academy of Sciences, Beijing* 100049, *China*)

Abstract: One of the most important kinds of queries in Spatial Network Databases to support Location-Based Services is the k-Nearest Neighbors (k-NN) query. In this paper, we propose a novel approach to efficiently and accurately evaluate k-NN queries in spatial network databases using network space diagram. This approach is based on partitioning a large network to small regions, and then precomputing distances both within and across the regions. Our empirical experiments with several random data sets show that our proposed solution outperforms approaches that are based on on-line distance computation by up to one order of magnitude.

Key words: moving object; spatial network databases; distance index; k-nearest neighbors (k-NN)

1 引言

随着无线通讯的持续增长和定位技术的发展,如何有效率的应答大量移动对象的查询请求以及基于位置的服务(location-based services LBS)变得越来越重要.而 k-NN 查询是其中使用最广泛的方法之一[1].

大多数在空间数据库中的研究只考虑 Euclidean 空间[2~4],因此在进行距离相关的查询时(例如:k-NN 查询)使用 Euclidean 距离.但是在某些情况下,对象的位置和移动受到网络的约束,例如:公路、铁路、河流等等.在这些情况下,对象之间的距离使用网络距离比 Euclidean 距离更符合需求.由于对象的位置表现为受约束空间数据,需要空间网络数据库(Spatial Network Databases SNDB)的支持.每一个传统的空间数据查询方法在 SNDB 中都有相对应的方法[5].

在 Euclidean 空间进行 k-NN 查询的方法都要执行多次距离计算,而且假定这些距离计算能够快速执行.但是对于网络空间来说这个假定并不成立,所以在网络空间中直接采用 Euclidean 空间中的方法效率会很低.

最近的研究针对网络空间中的 k-NN 查询提出了一些方法.Papadias 等人[5]提出了两种方法分别叫做 Incremental Euclidean Restriction (IER)和 Incremental Network Expansion (INE),INE 方法的性能明显好于 IER.INE 方法扩展网络边的方式和 Dijkstra 方法类似.INE 方法的主要缺点在于顶点扩展方法,当一个顶点需要扩展的时候,不仅仅是在每个相邻边中检索下一个最近的对象,而是要检索每个相邻边中的所有对象.这样需要对 R 树做一次检索,而且即使边中没有目标对象这样的检索还是要进行.

Kolahdouzan 和 Shahabi[6]认为上面的方法当目标对象在网络上分布不是很密集的情况下,查询效率会很低.他们提出了一个方法 Voronoi-based approach(VN3),该方法把一个大的网络划分成几个小的 Voronoi 区域,预先计算区域内部以及跨区域之间的距离,实验分析表明 VN3 方法优于 INE 方法.但是随着 k 值的增加,计算的代价逐渐增大,这是因为有很多路线会穿越 Voronoi

收稿日期:2009-09-15;修回日期:2010-02-24
基金项目:国家自然科学基金(No.60573164);教育部留学回国人员科研启动基金(No.K6GH625029)

区域.对于稀疏的数据集,Voronoi 区域的数目比较少,但是每个区域内部结构复杂,边界顶点也比较多,这样导致计算复杂.另一方面,对于密集的数据集,Voronoi 区域的数目比较多,这样导致可选择的对象也增多,在进行判断的时候增加了复杂性.

Huang 等人[7]提出了一种 Islands 方法,该方法预先计算并存储了一定距离范围内原始对象和目标对象之间的距离.这个距离范围是 Islands 的范围.该方法结合了预先计算方法和网络扩展方法的优点.

Cho 和 Chung[8]提出了提出了一种和 Islands 方法类似的方法,与 Islands 方法不同的是该方法预先计算部分原始对象和目标对象之间的距离,这部分对象叫做 condensing 对象.另一个不同在于不是计算一定范围内的目标对象,而是一定个数的目标对象.

Almeida 和 Guting[9]提出了一种增量 k-NN 方法.该方法提出了一种索引的存贮结构来支持高效的执行 Dijkstra 方法.Gorawski 和 Gebczyk[10]提出了一种 Gorder 方法.该方法能减少在 k-NN 中数据排序、连接调度和距离计算的时间.

上述方法对网络空间中的 k-NN 查询提出了多种解决方式,但是这些方法中都假设网络是静止的,并没有对网络是动态的情况做出分析.

我们基于 Kolahdouzan 和 Shahabi[6]的算法,提出了一种在动态网络空间中的位置相关查询算法,该方法是基于空间网络划分以及部分网络空间预计算.根据空间网络划分图 NSP(network space diagram)我们能够立即得到距离原始对象最近的目标对象.我们介绍了 NSP 的一些性质,这些性质能够证明下一个距离原始对象近的目标对象是已经找到的目标对象的相邻对象.而后,我们提出了一种方法来计算原始对象与候选对象之间的网络距离,该方法用到了网络空间预计算的数据.最后我们提出了了如何处理网络空间更新算法.

2 网络空间中的划分(NSD)

网络子空间划分图是基于网络空间的子空间划分图,对象的位置受限于网络空间,对象之间的距离定义为网络距离而不是 Euclidean 距离.在网络模型化的过程将网络模型化为一个单向有权重的图. $G(N, L)$, $N=\{p_1,\cdots,p_n,p_{n+1},\cdots,p_o\}$,前 n 个顶点为目标对象,也就是子空间的中心顶点, $L=\{l_1,\cdots,l_k\}$ 为边,定义点 p 通过边到点 p_i 的距离 $d_n(p,p_i)$ 为 p 和 p_i 之间的最短网络距离.对于所有 $j\in I_n\setminus\{i\}$,我们定义:

$$Dom(p_i,p_j)=\{p\mid p\in\bigcup_{o=1}^{k} l_o, d_n(p,p_i)\leqslant d_n(p,p_j)\}$$

$$b(p_i,p_j)=\{p\mid p\in\bigcup_{o=1}^{k} l_o, d_n(p,p_i)=d_n(p,p_j)\}$$

数据集 $Dom(p_i,p_j)$定义为 p_i 对 p_j 的控制区域,在该区域中所有的顶点以及边距离 p_i 都要比距离 p_j 近.数据集 $b(p_i,p_j)$定义为 p_i,p_j 之间的边界,在上面的顶点边于 p_i,p_j 的距离一样.

$$NSP(p_i)=\bigcap_{j\in I_n\setminus\{i\}} Dom(p_i,p_j)$$

$$NSD(P)=\{NSP(p_1),\cdots,NSP(p_n)\}$$

$NSP(p_i)$定义为在网络空间中距离 p_i 比距离其他中心顶点都要更近的顶点与边的集合,与上一小节中 $SP(p_i)$的定义类似,除了子空间的边以外该定义是相互独立完全穷尽的.

图 1 是 NSD 的一个例子,图 1(a)是原始图,其中 p_1,p_2,p_3 是中心顶点, $p_4\sim p_{16}$是空间网络的顶点,图 1(b)是划分后的图.由于是单向图,边只能是全部属于一个子空间,或者是两个子空间的边界边.该例子还显示了如何连接边界点以形成划分图.

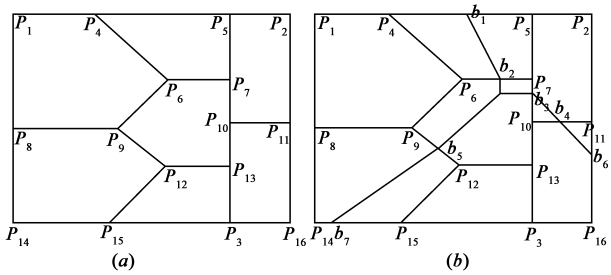

图1 网络空间NSD划分图

3 基于网络划分的距离查询算法

在本节中我们基于 Kolahdouzan 和 Shahabi[6]的算法提出了一种在动态网络空间中计算移动对象之间的距离的算法.该算法基于空间网络的划分以及子空间的距离索引.首先我们根据子空间划分的性质可以直接得到距离最近的目标对象,其次子空间的相邻空间可以得到下一个目标对象的候选对象(filter),最后根据距离索引我们可以得到原始对象与候选对象之间的距离,根据距离我们就可以得到下一个目标对象(refinement).Filter/refinement 过程交替循环进行:在每一次循环,首先要生成一组新的候选对象,然后计算原始对象到候选对象的距离,得到下一个距离最近的目标对象.这样 Filter/refinement 过程进行 k 次就可以得到距离原始对象 q 最近的 k 个目标对象.我们的算法运行有以下几个前提:

作为 filter 阶段的主要组成部分,目标对象的网络划分图需要事先计算好并存储下来,不同目标对象的网络划分图需要单独计算.

在第一次 Filter 阶段,距离 q 最近的目标对象可以通过定位包含 q 的 NSP.该过程可以使用 NSP 的索引来加速.

Refinement 阶段需要预先计算好每个 NSP 边界之

间的距离.这些预计算的距离用于计算穿过 NSP 的网络距离.

3.1 filter 阶段

我们提出的获得原始对象的候选对象的方法是基于第 3 节中提出的前两个前提.它需要提前计算好的 NSD 以及 NSP 的索引结构.我们首先介绍两条性质,这两条性质能把 filter 阶段的搜索空间限制在相邻子空间.

性质 1　对于任何在子空间 $V(p_i)$的点,它的第二近的中心顶点是 p_i 的相邻顶点.

证明:该性质可以通过反证法证明.考虑图 1(b)中 q 的最近的中心顶点是 p_4(q 在 $V(p_4)$中).现在假定 q 的第二近中心顶点是 p_3 不属于 p_4 的相邻顶点,这就要求 q 和 p_3 之间的最短路线 $L(q,p_3)$穿过至少一个 $V(p_4)$的相邻子空间($L(q,p_3)$在图 1(b)中通过点 b_1 和 b_2 穿过 $V(p_4)$).通过子空间的定义我们知道 $d_n(b_2,p_3)=d_n(b_2,p_2)$,我们可以得出 $d_n(b_1,b_2)+d_n(b_2,p_3)\geqslant d_n(b_1,p_2)$,则 $d_n(q,p_3)\geqslant d_n(q,p_2)$.这与我们的假设矛盾.

性质 2　如果 $G=\{g_1,\cdots,g_k\}\in P$ 是在 $V(g_1)$中的 q 的前 k 个最近的中心顶点,则 g_k 是$\{G\setminus g_k\}$的相邻顶点.该性质由性质 1 得出.

证明:该性质的证明于性质 1 类似.我们知道 q 到 g_k 的最短路线($L(q,g_k)$必然穿过 $V(g_k)$的一条边例如:E_k.假设 $L(q,g_k)$在点 b_k 穿过 E_k,而 E_k 是 $V(g_k)$于 $V(x)$的公共边.通过反证法来证明本性质,需要 x 不属于$\{G\setminus g_k\}$,而 $L(q,g_k)$必然要穿过 $V(x)$的至少另外一条边上的点 b_x.与性质 1 类似我们可以得到 $d_n(b_x,b_k)+d_n(b_k,g_k)\geqslant d_n(b_x,x)$,则 x 比 g_k 距离 q 要近,$x\in\{G\setminus g_k\}$,与我们的假设矛盾.

使用我们在图 2 中的 NSD,我们能很容易的描述我们的 filter 方法,使用该方法来得到候选对象集 C.对于 q 的最近的中心顶点,根据 NVP 的定义我们可以得到是 P_1,因为 $V(P_1)$包含了 q.根据性质 1 我们知道 q 的第二近的中心顶点是 P_1 的相邻顶点,$C=\{P_2,P_3,P_4,P_5,P_6\}$.在 NSD 的生成过程中,NSP 的相邻顶点已经生成了.我们将相邻信息存储在查找表中,这样查找 NSP

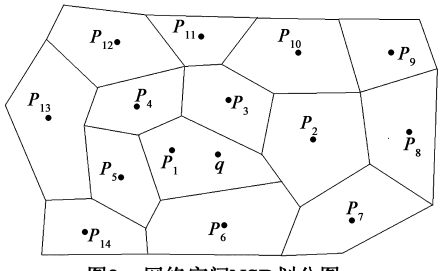

图2　网络空间NSD划分图

的相邻顶点就不需要任何空间操作.在这个阶段我们需要进行 refinement 来计算 q 到所有在 C 中的中心顶点的距离.根据距离来得到第二近的中心顶点.让我们假设第二近的中心顶点是 P_3,根据性质 2 第三近的顶点一定是$\{P_1,P_3\}$的相邻顶点,$C=\{P_2,P_4,P_5,P_6,P_{10},P_{11},P_{12}\}$.

3.2 refinement 阶段

我们在 3.1 节中讨论过,当候选顶点集 C 更新后,q 到 C 中所有顶点的距离需要计算来得到下一个最近顶点,在本节中我们讨论如何得到 q 到 C 中顶点的距离.

根据 NSD 的定义,所有能连接 NSP 中的对象和 NSP 外的对象的路线一定通过子空间的边界点,我们定义 BoP(e)为 e 的边界点.在 NSP 的生成过程中,边界点之间的的距离以及边界点于中心顶点之间的距离已经计算好并存入查询表中.如果我们的能得到外部对象到子空间边界点的距离,就能够得到对象到子空间中心顶点的距离.

在图 3 中,$N=\{p_1,\cdots,p_{14}\}$是 NSD 的中心顶点,也就是路网的目标对象,其中只有 $NSP(P_1)$的网络进行了具体描述.再该图中 $B=\{b_1,\cdots,b_{40}\}$是 NSP 的边界点.假设我们要计算 q 和 P_9 的距离,$NSP(P_9)$的边界点是 b_{34},b_{35},b_{36},则 $d_n(q,P_9)=\min(d_n(q,b_{34})+d_n(b_{34},P_9),d_n(q,b_{35})+d_n(b_{35},P_9),d_n(q,b_{36})+d_n(b_{36},P_9))$

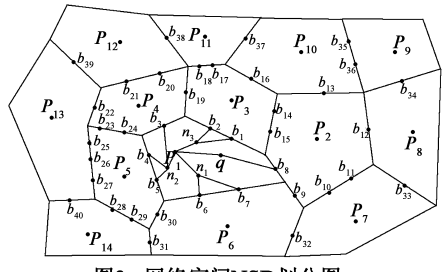

图3　网络空间NSD划分图

下面我们要计算 q 到 b_{34},b_{35},b_{36}的距离.根据上面描述,q 只能通过$\{b_1,\cdots,b_8\}$与$\{b_{34},\cdots,b_{36}\}$相连,则 $d_n(q,b_{34})=\min(d_n(q,b_1)+d_n(b_1,b_{34}),\cdots,d_n(q,b_8)+d_n(b_8,b_{34}))$

综上所述,得到 q 到 P_9 的距离或者说 q 到 C 中所有顶点的距离需要

边界点距离计算:

计算 q 到 $BoP(NSP(q))$的距离

边界点到边界点的距离计算:

计算 $BoP(NSP(q))$到候选顶点 g 的边界点之间的距离

3.2.1 边界点距离计算

该算法利用了第 3 节中的前提 3,边界点与所有内部顶点之间的距离已经预先计算.以图 3 中的 NSP 为例,$(b_1,\cdots,b_8)$相互之间的距离以及与 $P_1,(n_1,n_2,n_3)$

之间的距离已经进行了预计算.

该算法的好处是可以显著的提高性能,因为它不需要进行复杂的算法运算也不需要获取大量的位置数据用于距离计算. q 到 $BoP(NSP(q))$的距离通过检索查询表就可以获得,该算法的缺点是需要一个离线的进程来预计算以及存储网络距离.

3.2.2 边界点到边界点计算

为了计算 $BoP(NSP(q))$到了任意中心顶点的边界点的距离,我们先预计算每个 NSP 边界点之间的距离(第 3 节中的前提 3). 例如在图 4 中,我们预计算$\{b_1,\cdots,b_8\}$之间的网络距离,同时也计算$\{b_1,b_2,b_{14},\cdots,b_{19}\}$之间的网络距离. 由于每个边界点至少属于 2 个 NSP,例如 b_1 属于 $NSP(P_1)$和 $NSP(P_3)$,那么 b_1 到两个子空间边界点的距离都要计算. 由于每个 NSP 的边界点之间的距离都进行了的预计算,我们可以利用来计算任意 2 个 NSP 边界点之间的距离. 另外该方法的空间与时间复杂度都比较低,这是因为预计算只是计算边界点之间的距离,而在现实场景中边界点是计算远远少于内部顶点的,同时每个 NSP 是独立计算的.

计算 $BoP(NSP(q))$到 $BoP(NSP(g))$之间的距离的第一步是检索 q 到 g 的最短路线会通过那些 NSP,下面的性质可以通过 3.1 的性质 1、2 得出.

性质 3 在空间划分图中,如果(g_1,g_2)是两个距离 q 最近的中心顶点,则从 q 到 g_2 的最短路线只能通过$\{V(g_1),V(g_2)\}$,且穿过 $V(g_1)$和 $V(g_2)$的公共边.

证明:该性质通过反证法来证明. 如果 q 到 g_2 的最短路线通过子空间 $V(g_k)$, g_k 不属于$\{g_1,g_2\}$,那么在 $V(g_k)$中的最短路线距离 g_k 比 g_2 要近,则 q 距离 g_k 比 g_2 近,这与我们的假设矛盾. 距离来说,在图 1(b)中,假定(P_1,P_2)是 q 最近的两个中心顶点,假定 q 到 P_2 的最短路线是 $L=\langle q,b_3,b_4,P_2\rangle$,该路线通过 b_3,b_4 穿过 $NSP(P_1)$,在路段$\langle b_3,b_4\rangle$中的任意点距离 P_1 都比距离 P_2 近,这说明 P_1 是一个比 P_2 距离 q 要近的中心顶点,这与我们的假设矛盾. 这说明在 VSD 中,从 q 到 P_2 的的最短路线只能穿过 $V(P_1)$和 $V(P_2)$的公共边.

性质 4 在空间划分图中,如果$(g_1,\cdots,g_k)$是一组 q 的前 k 个最近中心顶点,则 q 的 g_k 的最短路线只能通过$\{V(g_1),\cdots,V(g_k)\}$,且一定穿过$\{V(g_1),\cdots,V(g_k)\}$的公共边.

证明:该性质是性质 3 的泛化,证明与性质 3 类似.

性质 5 在空间划分图中,如果从 q 到 g_k 的最短路线通过 $NSP(g_i)$则 g_i 比 g_k 距离 q 更近.

证明:根据性质 4 我们可以得到 q 到 g_k 最短路线只能通过$\{V(g_1),\cdots,V(g_{k-1})\}$,也就是 g_i 一定是$\{V(g_1),\cdots,V(g_{k-1})\}$中的一个,所以 g_i 比 g_k 距离 q 更近.

下面我们用图 4 中的空间划分图来说描述我们的方法如何得到 q 到 $BoP(NSP(CG))$的距离,其中 CG 是 q 的下一个最近顶点的候选顶点. 如图所示, q 的最近的中心顶点是 P_1,因为 $NSP(P_1)$包含 q. 根据性质 3, q 到第二近的中心顶点 P_i 的最短路线一定穿过 $NSP(P_1)$和 $NSP(P_i)$的公共边,为了得到 q 的第二近中心顶点,我们首先计算 q 到$(P_2,\cdots,P_6)$的临时最短路线 d_{mtn}.

$$d_{mtn}(q,P_2)=d_n(q,b_8)+d_n(b_8,P_2)$$

$$\cdots$$

$$d_{mtn}(q,P_6)=\min\{d_n(q,b_6)+d_n(b_6,P_6),\ d_n(q,b_7)+d_n(b_7,P_6)\}$$

理论上说, q 到 P_i 的临时最短距离 $d_{mtn}(q,P_i)$是 q 到 P_i 的路线中距离最短的,且该路线只能通过可能成为 q 的最近中心顶点的 NSP. 我们注意到上面所有的临时最短路线计算公式中的项,不是已经经过预计算就是能够使用我们的方法得到. d_{mtn}最下的中心顶点作为 q 的第二近的中心顶点,而断过 P_i 是 q 的下一个最近的中心顶点时 $d_{mtn}(q,P_i)=d_n(q,P_i)$. 我们假定 P_2 是距离 q 第二近的中心顶点,则第三近顶点 $P_i\in(P_3,P_4,P_5,P_6,P_7,P_8,P_{10})$, q 到 P_i 的最短路线一定通过$\{BoP(\{NSP(P_1)\cup NSP(P_2)\})\cap BoP(NSP(P_i))\}$

例如:从 q 到 P_6 的路线只能通过 b_6,b_7,b_9:

$$d_{mtn}(q,P_6)=\min\{d_n(q,b_6)+d_n(b_6,P_6),d_n(q,b_7)+d_n(b_7,P_6),d_n(q,b_8)+d_n(b_8,b_9)+d_n(b_9,P_6)\}$$

从 q 到 b_9 的实际最短路线可能通过(b_1,b_{15}),但在这个阶段,只有 P_1 和 P_2 作为 q 的最近的中心顶点,根据性质 4, $SP(q,P_6)$如果要通过 b_9 那么一定要通过 b_8. 换句话说,现阶段 q 到 b_9 的路线唯一的可能是通过 b_8. 该路线的长度用来计算 $d_{mtn}(q,P_6)$. 我们的方法将计算 q 到 $BoP(NSP(CG))$实际网络距离转换成只用计算可能最小距离. 如果我们假定 P_3 是第三近的中心顶点,则根据性质 4, q 到 b_9 的临时最短路线还需要计算(b_1,b_2)和(b_{14},b_{15}). 然后进行循环计算就可以得到 q 到所有中心顶点的距离.

3.3 索引更新

对于动态图来说,当图变化时与原始图相关的距离索引有可能会失效,我们需要对索引进行更新,当图只是局部变化时,我们并不需要更新整个索引.

当图变化时主要有两种情况,顶点变化、边变化. 顶点变化又有增加顶点和删除顶点两种情况. 边变化有增加边、删除边、边权重增加、边权重减少 4 种情况.

增加顶点、删除顶点、增加边、删除边这 4 种情况在实际交通网络中相当于道路的开通与封闭出现的频率比较慢,是以天为单位的,我们索引算法的建立时间可以满足. 边权重增加、边权重减少这 2 种情况实际交通

网络中出现的频率比较快,是以小时为单位的.

下面对这 6 种情况做具体分析:

增加边、删除边是边权重增加、边权重减少的极端情况可以归于后 2 类讨论;删除顶点相当于连接到该顶点的边全部删除,这样也可以归于边权重增加讨论;新增顶点分为 2 种情况,新增顶点在原图的边上、新增顶点不在原图的边上.对于这 2 中情况都可以分为 2 个阶段,第一阶段新增顶点,在这个阶段对于原有距离索引没有影响.第二阶段新增到该顶点的边,这个阶段又可以归于边权重减少讨论.

这样我们就将图变化的情况都归于边权重增加、边权重减少这 2 种情况.下面我们具体分析这 2 种情况:

首先是边权重增加.在建立距离索引的时候,实际上是已每个顶点为起点,建立图的生成树.当边权重增加时,增加权重的边如果不属于某点生成树的一部分,那么该点的索引不受影响.如果属于生成树的一部分,则该生成树需要从该边前的顶先重新计算.

当边权重增加减少时,无论该边是否在生成树上都需要从该边前的顶先重新计算.如果该边不在生成树上,当重新计算时如果没有影响下一点的计算则生成树不需做变化.具体算法如下:

算法 1　索引更新算法

① 判断图变化类型,权重增加转到 9,权重减少转到 2
② For i = 1 to 顶点上限
③ 检索边的前一个顶点在生成树的位置
④ 在该位置重新计算
⑤ 是否影响下一点计算,是则 7,否则 6
⑥ 该边是否在生成树上,是则 7,否则 8
⑦ 继续计算完成
⑧ Next
⑨ 结束
⑩ For i = 1 to 顶点上限
⑪ 该边是否在生成树上,是则 12,否则 13
⑫ 继续计算完成
⑬ Next
⑭ 结束

4　k-NN 查询算法

初始时间为 t_0,原始移动对象为 p_0,共有 n 个目标移动对象,分别为 $p_1, p_2, \cdots, p_n$,Δt 时间后 k-NN 算法得出在 n 个目标移动对象中距离原始移动对象 p_0 最近的 k 个对象

算法首先对存放目标对象以及原始对象到目标对象距离的队列进行初始化.然后通过索引得到距离原始对象最近的目标对象,并将该对象放入队列.接下来通过检索队列中所有对象的相邻对象,得到候选目标对象.计算原始对象与候选目标对象之间的距离得到下一个最近的目标对象.重复以上过程直到得到 k 个目标对象.

算法 2　k-NN 算法:

输入:原始移动对象 p_0、Δt
输出:k 个目标移动对象
(1) 根据原始对象所在子网得到最近的目标对象,将目标对象放入队列 q_1
(2) 根据队列中所有目标对象得到侯选对象(filter)
(3) 得到下一个目标对象 p_i,并将其放入队列 q_1.如果队列中的目标对象达到 k 个则进入 4,否则进入 2 (refinement)
(4) 队列中的目标对象为查询结果

5　算法复杂性分析

假设有网络有 m 条边,n 个顶点,则 filter 阶段 2 – 3 步需要$(\log_2 m)$次计算 .过滤算法总共需要 n $(\log_2 m)$次计算,则复杂性为 $O(n \log_2 m)$.

假设有网络有 k 个目标对象,则 1 步需要$(\log_2 k)$次计算 .则 refinement 阶段 2 – 3 步的复杂性为 $O(n)$,则精炼算法的复杂性为 $O(n)$.

k-NN 算法的复杂性为 $O(n\log_2 m)$,如果采用 Dijkstra 算法直接计算复杂性为 $O(n^3)$

6　实验分析

6.1　实验数据集

为了进行实验我们的数据为随机生成的图.生成规则如下:

(1)每个顶点最多有 5 个最少有 1 个直接连接的顶点,其中各有 1% 的顶点有 1 个或 5 个直接连接的顶点,5% 的顶点有 3 个直接连接的顶点,其余 93% 的顶点有 4 个直接连接的顶点.

(2)每条边的权重从 1 到 10,其中各有 1% 的边权重为 1 或 10,各有 5% 的边权重为 2 或 9,各有 8% 的边权重为 3 或 8,各有 15% 的边权重为 4 或 7,各有 21% 的边权重为 5 或 6.

随机生成图,3000 个顶点,图对包含不同数量的移动对象进行查询,每个图分别对包含 1000、2000、3000、5000、8000 个移动对象进行查询.

6.2　实验查询

在进行查询的时候我们对每个包含不同移动对象的图分别做 1 组查询,查询离原始对象 p_0 最近的 5 个目标移动对象,也就是 k 值分别为 5,每组查询 100 次.我们的度量标准是平均每次查询需要的时间.

6.3　实验结果

实验结果如下:

从图 4 我们得出以下几点分析:首先可以明显的看

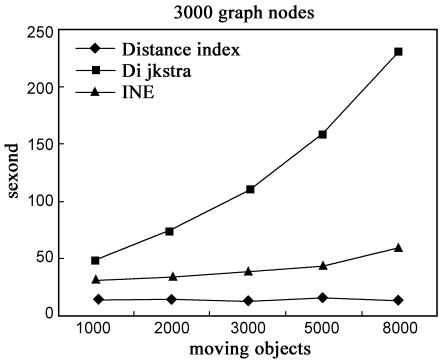

图4

出在当查询的目标对象数量变化时,采用距离索引的算法都比直接使用 Dijkstra 算法以及 INE 的效果要好.第二在图 4 中,随着包含的移动对象的增加距离索引的算法的效率保持稳定,INE 算法效率有所下降,而 Dijkstra 算法的效率在快速下降,即使包含移动对象较少时,距离索引的算法的效率也是更高.最后随着图顶点的增加,三种算法的效率都下降,但是距离索引的算法的效率比 Dijkstra 算法和 INE 算法的效率要更高,即使图顶点较少时,距离索引的算法的效率也是更高.

7 小结

在本节中我们提出了一种在动态网络空间中的位置相关查询算法,该方法是基于空间网络划分以及部分网络空间预计算.根据空间网络划分 NSP 我们能够立即得到距离原始对象最近的目标对象.我们介绍了 NSP 的一些性质,这些性质能够证明下一个距离原始对象近的目标对象是已经找到的目标对象的相邻对象.而后,我们提出了一种方法来计算原始对象与候选对象之间的网络距离,该方法用到了网络空间预计算的数据.最后我们提出了动态网络的索引更新算法.

参考文献:

[1] C S Jensen, A Friis-Christensen, T B Pedersen, D Pfoser, S Saltenis, N Tryfona. Location-based services: A database perspective[A]. ScanGIS[C]. Ås, Norway: The Department of Mapping Sciences, Agricultural University of Norway, 2001.59 – 68.

[2] G R Hjaltason, H Samet. Distance browsing in spatial databases [J]. ACM Trans Database Syst, 1999, 24(2): 265 – 318.

[3] N Roussopoulos, S Kelley, F Vincent. Nearest neighbor queries [A]. Proc of the ACM SIGMOD Intl Conf on Management of Data[C]. San Jose: ACM Press, 1995.71 – 79.

[4] R Benetis, C S Jensen, G Karciauskas, S Saltenis. Nearest and reverse nearest neighbor queries for moving objects[A]. Proceedings of the 2002 International Symposium on Database Engineering & Applications table of contents[C]. Edmonton: IEEE Computer Society, (Edmonton), 2002.44 – 53.

[5] D Papadias, J Zhang, N Mamoulis, Y Tao. Query processing in spatial network databases[A]. Proc of 29th Intl Conf on Very Large Data Bases (VLDB)[C]. Berlin: Morgan Kaufmann, 2003.802 – 813.

[6] M R Kolahdouzan C Shahabi. Voronoi-based *k* nearest neighbor search for spatial network databases[A]. Proc of 30th Intl Conf on Very Large Data Bases (VLDB)[C]. Toronto: Morgan Kaufmann, 2004.840 – 851.

[7] X Huang, C S Jensen, S Saltenis. The islands approach to nearest neighbor querying in spatial networks[A]. Proc. of the 9th Intl Symp. on Spatial and Temporal Databases (SSTD)[C]. Angra dos Reis: Springer, 2005.73 – 90.

[8] H Cho, C Chung. An efficient and scalable approach to cnn queries in a road network[A]. Proc of 31th Intl Conf on Very Large Data Bases (VLDB)[C]. Trondheim: Morgan Kaufmann, 2005.865 – 876.

[9] V T Almeida, R H Güting. Using Dijkstra's algorithm to incrementally find the *k*-nearest neighbors in spatial network databases[A]. Proceedings of the 2006 ACM symposium on Applied computing[C]. Kennesaw: Computer Security, 2006.58 – 62.

[10] M Gorawski, W Gebczyk. Realization of continuous queries with *k*-NN join processing in spatial telemetric data warehouse[A]. Proceedings of the 17th International Conference on Database and Expert Systems Applications table of contents[C]. Krakow: Springer, 2006.632 – 636.

作者简介:

殷晓岚 男,1976 年生于安徽合肥.中国科学院软件研究所博士生.研究方向为移动计算.
E-mail: xlyin@otcaix.iscas.ac.cn