

一种道路网络中移动对象的K近邻多查询处理算法

郝兴 王凌 孟小峰

(中国人民大学信息学院, 北京 100872)
{haoxing, jingyiwang, xfmeng}@ruc.edu.cn

摘要 在实际应用中, 服务器时常会收到许多查询请求, 因此如何高效地进行多查询处理, 并且获得良好的吞吐量具有重要的意义。本文研究了道路网络中连续的 K 近邻多查询处理技术。在已知查询点位置和运动速度的情况下, 将道路网络上的查询点进行聚类, 提出了基于聚类的 K 近邻多查询处理算法, 使同一聚类中的查询共享执行, 从而大大提高多查询处理的效率。算法还进一步利用扩展树存储查询结果, 提高连续的 K 近邻多查询处理性能。理论分析和实验结果表明本文提出的算法是可行的, 高效的。

关键词 K 近邻查询; 多查询处理; 道路网络; 聚类

中图法分类号 TP392

Multiple KNN Queries Processing for Moving Objects in Road Networks

Xing Hao, Ling Wang, and Xiaofeng Meng
(School of Information, Renmin University of China, Beijing 100872)

Abstract In the real world applications, servers often receive a lot of query requests, so how to process these queries efficiently and achieve better I/O efficiency is a challenging goal. This paper studies the multiple KNN queries processing for moving objects in road networks. Assume that the location and velocity of the query points in the road network are given, we cluster the query points and propose a Clu-MQN (Clustering based Multiple Queries in road Network) algorithm, which processes the queries in the same cluster synchronously, so the efficiency of query processing is greatly improved. Also the expansion tree is used to store the query results in order to improve the efficiency of processing multiple continuous k nearest neighbor queries. Theoretical and experimental analysis proved the feasibility and efficiency of the algorithm.

Keywords k nearest neighbor query; multi queries processing; road network; cluster

1. 引言

在空间数据库研究中, K 近邻 (KNN) 查询问题受到广泛关注, 这类查询被频繁地使用于地理信息系统 (GIS) 等应用中。KNN 查询可以定义为: 给定一个空间对象集合, 一个查询点, 找到距离查询点最近的 K 个对象。例如, 由装载着 GPS 设备的汽车向服务器发出一个 KNN 查询请求, 要求找到距离该车最近的 5 个饭店。根据查询点及对象所处环境不同, KNN 查询可以分为两类。一是基于自由空间的 KNN

查询[1,2]。二是基于空间网络 (如道路网络) 的 KNN 查询[3,4]。本文研究的就是基于道路网络的 KNN 查询处理。在自由空间中, 对象的运动是不受约束的, 而且对象间距离是欧几距离, 而在道路网络中, 对象之间的距离是基于最短路径计算的网络距离, 因此自由空间上的 KNN 算法难以运用到空间网络上的 KNN 查询中, 移动对象的运动受到道路网络限制, 查询处理需要沿着网络进行遍历。

收稿日期: 2007-06-20

基金项目: 本文得到国家自然科学基金项目(课题号: 60573091), 北京市自然科学基金 (课题号: 4073035), 教育部新世纪优秀人才支持计划的资助。

现有基于空间网络的 KNN 查询研究主要集中在如何高效地响应单个查询。而在实际应用中,中央服务器时常会收到许多查询请求。如何高效的处理多个查询(即多查询处理),以获得良好的系统吞吐量更有实际意义。在空间数据库中,已有的多查询处理研究大多假设对象在欧几空间中运动[6]。文献[5]提出了空间网络中的 KNN 多查询问题,它们通过在主存中缓存先前的 KNN 查询结果来减少多查询处理的磁盘 I/O 代价。但是,这种方法处理的是基于道路网络的静态的多个 KNN 查询,即假设查询点是静态对象点。而本文关注的问题是如何高效处理基于道路网络的连续的多个 KNN 查询,即假设查询点是连续运动的移动对象。

道路网络的单个 KNN 查询处理方法可以分为两类:在线计算方法[3],即增量的扫描网络直到 K 个近邻找到为止;预计算方法[4],即基于预计算形成的数据结构进行 KNN 的查询处理。第一类方法捕捉网络的动态特性,例如有用对象的出现与消失,并且使用基于网络扩展的搜索方法。由于使用在线计算的数据结构,这类方法可以捕捉网络连通性并且易于更新。然而,与在线计算方法相比较,第二种方法的查询性能更好,但是难于处理道路网络和对象点的频繁更新。本文提出的方法把上述两类方法相结合。首先将道路网络上的查询点进行聚类以共享预计算的查询结果,并且利用扩展树增量地连续返回 KNN 查询结果。

本文主要研究道路网络中多 KNN 查询处理,提出一种基于聚类的高效连续查询处理算法。主要贡献总结如下:

1) 在已知查询点位置和运动速度的情况下,对道路网络上的查询点进行聚类,实现同一聚类的查询共享执行。从而减少了重复的查询操作,可大大提高多查询处理的效率。

2) 利用扩展树来处理查询并存储和维护查询结果。首先,根据查询点聚类和初始查询结果建立扩展树,并保存从查询点到最远查询结果的距离的路网信息。然后,随着查询点和对象点的位置更新,增量的更新扩展树使其包含所有可能的查询结果,从而实现连续的多 KNN 查询处理。

3) 在模拟实验环境下,对提出的多查询处理算法进行了大量实验测试。实验结果表明所提出的算法具有可行性和高效性。

本文第 2 节介绍相关工作;第 3 节讨论 KNN 多查询处理算法,包括聚类预处理过程,扩展树建立和扩展树的动态维护过程;第 4 节给出实验结果与分

析;最后是结论与展望。

2. 相关工作

随着 kNN 查询处理技术的发展和用户查询请求的增多,研究者开始更多关注多 KNN 查询处理,考虑如何降低多查询代价,提高总的查询效率。

文献[5]中主要介绍了三种多查询处理的方法:基于共享的 S_kNN;基于聚类的 C_kNN;以及两者的结合 SC_kNN。S_kNN 方法是将查询点的结果缓存,当出现新的查询点时,可以利用这些缓存的结果。在算法中,设置一个 D 值,当一个查询点 q 扩展距离为 D 并且没有发现缓存的查询点,则将 q 的查询结果缓存。C_kNN 方法是对查询点聚类。在一个聚类区域内,所有邻接点都在该区域内的结点称为内部结点,存在邻接点不在该区域内的结点称作边界结点。在该算法中,缓存边界结点的 kNN,这样,所有聚类内部的查询点在扩展过程中都不必向聚类区域外扩展。该算法在查询点比较密集的区域,减少了扩展的次数,提高了查询效率。然而在查询点稀疏区域,仍然需要逐一进行扩展。SC_kNN 方法是 S_kNN 和 C_kNN 方法的结合,既缓存密集区域边界点的 kNN 结果,也按 D 距离存储查询点的结果。结合了两种算法的优点,但同时也增大了存储的数据量。

SEA-CNN (Shared Execution Algorithm) 算法[6]中定义了两个区域: Answer region 是以查询点为中心,包含所有查询结果的最小圆; Searching region 为包含所有可能的查询结果的最小圆。算法根据查询点的 Answer region 以及数据点和查询点的变化,计算出查询点的 Searching region。并在此区域中查找查询结果。该算法提供了欧几空间上的多查询处理。

3. 查询处理

这一部分将介绍如何将查询点聚类,并利用聚类结果,高效的查找结果。

首先,将查询点按照位置和运动的相似性聚类。将聚类结果看作一个整体进行处理。

其次,利用扩展树[7]存储查询结果及路径信息,即保存从查询点到 dmax (dmax 为查询点到最远查询结果的距离)处的所有路网信息。扩展树的意义在于,只有影响到扩展树的数据点会影响到查询结果。

最后,当查询点的运动及数据点的更新影响到扩展树,即改变查询结果时,需要更新扩展树和查询结果。即扩展树的维护。

3.1 预处理——聚类的形成

本节首先引入文献[8]中的网络定义,然后通过提出一个新的聚类块的概念来定义道路网络中查询点的聚类。

定义 1: 一个网络可以表示为一个无向带权图 $G=(V, E, W)$, 其中 V 是顶点(结点)的集合, E 是边的集合, W 为正的实数集合($W: E \rightarrow IR+$), 表示边所对应的权值。网络中的一个对象位于网络中的边 e ($e \in E$)上, 对象在网络中的位置可以表示为一个三元组 (n_i, n_j, pos) (取代欧几空间坐标表示), 其中 n_i 和 n_j 是对象所在网络边的两个结点, $pos \in [0, W(e)]$ 是对象离结点 n_i 的距离。

根据道路网络的聚类特点, 我们通过提出一个新的聚类块的概念来定义道路网络中的查询点对象聚类问题。

定义 2: 聚类块(Cluster Block, 简称CB). 一个CB表示为 $(O, n_a, n_b, head, tail, ObjNum)$ 其中 O 为对象的集合 $O = \{o_1, o_2, \dots, o_i, \dots, o_n\}$ ($o_i = (oid_i, n_a, n_b, pos_i, speed_i, next_node)$), 不失一般性, 假设 $pos_i < pos_{i+1}$ ($1 \leq i \leq n-1$), 满足 $|pos_{i+1} - pos_i| \leq \epsilon$ ($1 \leq i \leq n-1$). $head, tail$ 分别为聚类块的起点位置和终点位置, 通过相对于边起点的相对距离来表示, $ObjNum$ 为CB中对象的个数。既然所有的对象都在相同的边 (n_a, n_b) 上, 聚类块的位置由 $(head, tail)$ 来决定, 其长度为 $|tail-head|$ 。

聚类块与数据点之间的网络距离定义为聚类的中心位置 $((tail+head)/2)$ 到数据点的网络距离。

聚类处理过程实际上是多查询处理的一个预处理过程, 为了减少查询点维护的代价, 引入了聚类块的定义。一个聚类块是一组在当前和将来一段时间都彼此靠近的移动查询点的集合。我们限制一个聚类块中查询点以相同的方式运动在相同的路段上。为了使得一个聚类块中的查询点足够的密集, 聚类块中每对相邻查询点之间的网络距离不超过一个系统阈值 ϵ 。我们假设移动查询点分段线性运动, 在每个路段上运动速度稳定, 并且下一个运动的路段已知。聚类过程将道路网络上的查询点分组为聚类块的集合。具体过程为扫描道路网络上的所有路段, 若路段上存在查询点, 将查询点分组成相应的 CB, 保证任意 CB 内的查询点集合满足任意两个相邻查询点之间的距离不大于 ϵ 。这个阶段可以过滤掉那些没有查询点的路段, 形成一个初始的分组结果, 减少不必要的冗余的网络搜索过程。

通过将查询点聚类成 CB 的形式, 可以将同一个 CB 中的多个查询点看作一个查询点, 共享执行查询, 减少查询处理次数, 提高查询处理的效率。

3.2 查询初始化处理

在算法的初始阶段, 利用 Dijkstra 算法, 找到每个聚类的初始扩展树和查询结果。即, 从聚类块开始, 扩展路网直到找到 k 个查询结果。图 3-1 为查询初始

化的伪代码。

查询初始化算法:

```

1 For (每一个聚类 CB)
2   计算所有查询点中查询结果最大值 kmax
3   初始化小顶堆 H 和 PE, 设置 dis_kmax=∞, disd=disc=0, posc 为 CB 中心位置
4   设 n1, n2 为 posc 所在的边, n1 为初始结点, 将 n1 放入 H, 标志值设为 0, 将 n2 的标志值设为 1
5   While (H 中的 n 点距 posc 的距离 < dis_kmax)
6     取出 n, 将 n 的邻接结点 nadj 放入 H, 如果 nadj 没有标记过将 nadj 的标志值设为和 n 相同
7     For (每条 nnadj)
8       For (检查 nnadj 上所有的数据点 p)
9         If (dis(p, posc) < dis_kmax)
10          将 p 放入结果集即 CB 中所有查询点的结果集并记录 dis(q, posc), 如果 p 为结果集中第 kmax 个数据点, dis_kmax = dis(q, posc)
11          If (mnadj 中有一个结点的标志值为 1)
12            p 的标志值为 1,
13          else p 的标志值为 0
14          将 mnadj 放入 PE
15        End for
16      End for
17    End while
18    根据 dis_kmax 计算 PE 中边的边界, 构造扩展树
19    找到标记为 0 的查询结果中距离 posc 最远的数据点 p, disd = dis(p, posc), 如果没有标记为 0 的查询结果, 则找到最近的查询结果 p, disd = dis(p, posc)
20    扩展所有边界结点标志值存在 1 的边, 找到第一个不在结果集中的数据点 p', disc = dis(p', posc), 调整扩展树
21  End for

```

图 3-1 查询初始化的伪代码

对每一个CB进行相同的处理。首先, 初始化小顶堆H和PE, H存放要扩展的边的初始结点, PE存放所有可能的扩展树上的边。得到CB所在的边 n_1n_2 , 将终结点 n_2 标记为 1, 表示CB向这个方向运动。然后循环的找到下一步将要扩展到的边, 将其上的数据点进行检查并放入结果集, 每一个数据点有一个标志, 表示随着CB的运动, 数据点与CB的距离会增加还是减少, 如果标志为 1, 表示距离会缩小。当下一个要扩展的边的起始结点到CB的距离大于 dis_kmax 时, 算法结束。最后根据 dis_kmax 构造扩展树。找到标记为 0 的查询结果中距离CB最远的数据点 p , 这个数据点可能成为第一个离开CB查询结果的点。而算法 20 行的 p' 表示这个数据点可能成为第一个代替 p 成为查询结果的数据点。

如图 3-2, CB 为聚类的中心位置, 它从 n_1 向 n_2 运动, $p_1 \sim p_8$ 为数据点, $kmax=4$ 。如图 3-2 中黑色标记为 CB 的扩展树。将此时的 kNN 结果 $\{p_1, p_2, p_3, p_4\}$ 记录到结果集中, $tstart$ 记录为 0。

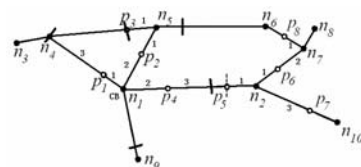


图 3-2 查询初始化

标记为 0 的查询结果包括 p_1, p_2, p_3 , 其中距离

CB最远的是 p_3 ，到CB的距离为4，所以 $disd=4$ 。边界结点标志值存在1的边只有 n_1n_2 ，扩展 n_1n_2 ，找到第一个数据点为 p_5 ，距CB的距离为5， $disc=5$ ，调整扩展树为图3-2中虚线标记部分。

3.3 增量式连续查询处理

3.3.1 基于查询点位置更新的查询处理

由于查询点的运动，有的点会离开扩展树，而有些点会进入扩展树，为了找到这个临界的位置，定义如下距离。

定义 dis 为：

$$dis = (disc - disd) / 2$$

当CB运动超过 dis 距离时，需要更新它的扩展树，此时由于CB的运动使原来的查询结果不再有效。

在图3-2的例子中。 $dis = (5 - 4) / 2 = 0.5$ ，当CB运行到距 n_1 距离为0.5的位置时， p_5 成为它的新的查询结果，而 p_3 将不再是它的查询结果。因为此时，CB距 p_5 和 p_3 的距离为 $disd + dis$ 和 $disc - dis$ 均为 $(disc + disd)$ ，即4.5，如图3-3。而CB会距 p_5 越来越远相反距 p_3 越来越远。此时需要把 p_5 加入查询结果集，而从查询结果集中删除 p_3 。并且记录下此时的时间 $tchange$ 。作为上一个查询结果的有效时间结束点和下一个查询结果的有效时间起始点。

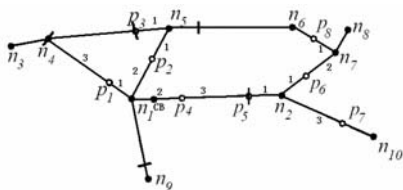


图 3-3 基于查询点位置更新的查询处理

3.3.2 基于数据点位置更新的查询处理

在任意时刻，如果有任意数据点从扩展树外进入到扩展树内部，或有结点从扩展树内部离开，都要更新扩展树。

在图3-2的例子中，如果在 $[0, tchange]$ 时间段，数据点位置更新为图3-4中的情况。 p_5 向 n_1 方向移动了距离1， p_3 向 n_4 方向移动了距离1。此时，要把扩展树扩张到 p_3 的新的位置，如图3-4中标记的位置，这时的扩展树包含了所有的可能的查询结果。再修剪该扩展树，直到找到正确的查询结果。

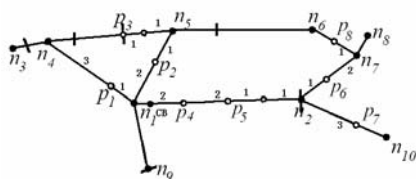


图 3-4 基于数据点位置更新的查询处理

更新扩展树算法伪代码如图3-5。

更新扩展树算法

```

1 obj 为更新的数据点
2 if (obj 进入扩展树)
3     newdis=dis kmax, 将 obj 标记后放入结果集
4 if (obj 离开扩展树)
5     newdis=dis (obj, posc)
6 if (obj=nil)
7     disc=dis (候选点 q', posc)
8 newdis=max(newdis, disc)
9 for (每个扩展树中的边并且不是整条边在扩展树中)
10    扩展该边到距 posc 为 newdis 的距离, 并把扩展中遇到的边做标记后放入 PE, 遇到的新数据点标记后放入结果集
11 检查结果集中的数据点, 找到前 kmax 个点, 并记录 dis kmax
12 根据 dis kmax 计算 PE 中边的边界, 构造扩展树
13 找到标记为 0 的查询结果中距离 posc 最远的 p, disd=dis(p, posc), 如果没有标记为 0 的查询结果, 则找到最近的查询结果 p, disd=-dis(p, posc)
14 扩展所有边界结点标志值存在 1 的边, 找到第一个不在结果集中的数据点 p', disc=dis(p', posc), 调整扩展树
    
```

图 3-5 更新扩展树的伪代码

如果 $disc$ 大于 $2 * (dis(n_1, n_2) - posc) + disd$ ，则定为 $disc = 2 * (dis(n_1, n_2) - posc) + disd$ ，因为此时，CB需向这个方向运行超过 $dis = (2 * (dis(n_1, n_2) - posc) + disd - disd) / 2 = dis(n_1, n_2) - posc$ 的距离，才会改变查询结果，而此时CB已走到该条边的结束点。

使用扩展树可以处理连续的KNN查询处理，使查询结果能够适应数据更新的变化。

4 实验结果与分析

这部分我们比较了基于聚类的Clu-MQN算法与不使用聚类的查询算法（简称Conv）的效率。分析了数据点的个数，查询点的个数，查询结果的个数，数据点的更新次数，以及查询点的聚类比率对查询处理效率的影响。

我们使用brinkhoff[9]基于网络的移动对象产生器产生实验数据。图4-1为实验中使用的参数及其取值。

Parameter	Default	Range
10^{-3} *Number of objects	10	1,5,10,15,20
Number of queries	500	100,300,500,700,1000
Number of NNs	20	5,10,20,30,50
Objects agility	10%	0,5,10,20,50(%)
Cluster of queries	80%	90,80,70,60,50(%)

图 4-1 实验参数及取值

图4-2显示了当数据点增多时，对算法效率的影响。随着数据点的增多，两种算法的效率都有所提高，这是因为，数据点变得密集，使扩展树减小，扩展树的维护代价也减小了。但是随着数据点的增多数据点更新频繁，调用扩展树更新算法次数增加，所以算法效率的提高不再明显。同时，当数据点增多，Clu-MQN算法比Conv的优势减小，Clu-MQN的优势在于扩展树少，不必维护所有查询点的扩展树，只要维护CB

的扩展树,因此当扩展树的维护代价减小, Clu-MQN 的优势不再明显,相反,由于数据点的增多使得构造初始存储结构的复杂性加大, Clu-MQN 算法反而比 Conv 算法效率低。

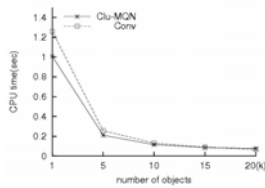


图 4-2 数据点个数对查询性能的影响

图 4-3(a)显示了查询点个数对查询性能的影响。随着查询点的增多,算法效率降低。当查询点比较少时, Conv 比 Clu-MQN 效率高,因为此时只有少部分的查询点,聚类不太明显,且 Clu-MQN 在初始阶段要建立聚类存储结构,使得算法复杂度比 Conv 高。但当查询点增多,更多的查询点聚类,扩展树的构造和维护代价都比 Conv 降低。4-3(b)显示了 k 的个数对查询性能的影响。当 k 增大,表示查询结果的扩展树增大,因此扩展树的建立以及维护代价都将增大,此时, Clu-MQN 的优势就显现出来。因此它的效率会比 Conv 越来越好。

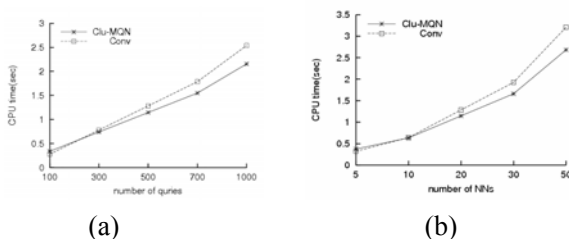


图 4-3 查询点的个数和 K 的个数对查询性能的影响

图 4-4(a)显示了数据点更新率对算法性能的影响。首先,数据点更新次数增多,扩展树的维护工作相应增加,因此两种算法的效率都有所降低,且 Conv 的降低幅度会比 Clu-MQN 增大。4-4(b)显示了查询点聚类率(聚类数与查询点数比率)对算法性能的影响。查询点的聚类率的减少即查询点相似度的提高,会大大提高 Clu-MQN 的处理效率,而对 Conv 没有任何影响。

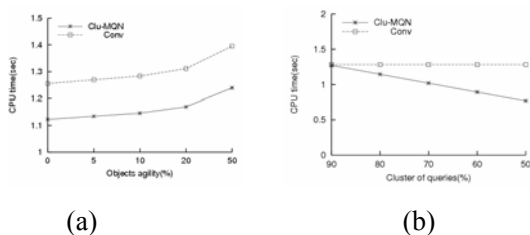


图 4-4 数据点更新率与查询点聚类率对查询性能的影响

5. 结论和未来工作

本文研究了道路网络中连续多 KNN 查询处理技

术。在已知查询点位置和运动速度情况下,对道路网络上的查询点进行聚类,实现同一聚类的查询共享执行,从而大大提高多查询处理的效率。利用扩展树进行查询处理和查询结果的存储,并且不断维护扩展树,从而实现连续的 KNN 多查询处理。最后,基于模拟环境进行实验,实验结果表明所提出算法具有可行性和高效性。未来我们将在道路网络上其他查询类型的多查询处理方面进行研究,并进一步考虑查询结果的效率和精度的评估。

参考文献

1. Jun Zhang, Manli Zhu, Dimitris Papadias, Yufei Tao, Dik Lun Lee. Location-based Spatial Queries. In: Proc of the SIGMOD 2003. San Diego, USA: ACM Press, 2003. 443-454
2. Yufei Tao, Dimitris Papadias. Time-parameterized queries in spatio-temporal databases. In: Proc of the SIGMOD 2002. Madison, USA: ACM Press, 2002. 334-345
3. Victor Teixeira de Almeida, Ralf Hartmut Güting. Using Dijkstra's algorithm to incrementally find the k-Nearest Neighbors in spatial network databases. In: Proc of the SAC 2006. Dijon, France ACM Press 2006. 58-62
4. Mohammad R. Kolahdouzan, Cyrus Shahabi, Voronoi-Based K Nearest Neighbor Search for Spatial Network Databases. In Proc of the VLDB 2004. Toronto, Canada. ACM Press 2004, 840-851
5. Xuegang Huang, Christian S. Jensen, Simonas Saltenis, Multiple k Nearest Neighbor Query Processing in Spatial Network Databases. In Proc of the ADBIS 2006. Thessaloniki, Hellas. 266-281.
6. Xiaopeng Xiong, Mohamed F. Mokbel, Walid G. Aref. SEA-CNN: Scalable Processing of Continuous K-Nearest Neighbor Queries in Spatio-temporal Databases. In Proc of the ICDE 2005. Tokyo, Japan. 643-654.
7. Kyriakos Mouratidis, Man Lung Yiu, Dimitris Papadias, Nikos Mamoulis. Continuous Nearest Neighbor Monitoring in Road Networks. In Proc of the VLDB 2006. Seoul, Korea. 43-54.
8. Man Lung Yiu, Nikos Mamoulis. Clustering Objects on a Spatial Network. In Proc of the SIGMOD 2004. Paris, France. 443-454
9. Thomas Brinkhoff: A Framework for Generating Network-Based Moving Objects. *GeoInformatica* 2002 6(2): 153-180

郝兴, 女, 1985 年生, 主要研究方向为移动数据管理。

王凌, 女, 1975 年生, 主要研究方向为移动数据管理。

孟小锋, 男, 1964 年生, 教授, 博士生导师, 主要研究方向为 Web 数据管理、XML 数据库、移动数据管理。

郝兴

E-mail: haoxing@ruc.edu.cn

中国人民大学品园一楼 425 室 100872

手机: 13522668662

座机: 010-62514994