# 路网中高吞吐量移动对象实时查询算法

薛忠斌[1,2,3] 白利光[1] 何 宁[1] 周 烜[2,3] 周 歆[2,3] 王 珊[2,3]
(神华国华(北京)电力研究院有限公司 北京 100069)[1]
(教育部数据工程与知识工程重点实验室(中国人民大学) 北京 100872)[2]
(中国人民大学信息学院 北京 100872)[3]

**摘 要** 随着无线通信技术、空间定位技术和移动计算技术的快速发展,基于位置的查询成为数据库领域的一个重要研究问题。研究了路网中移动对象的KNN查询,一系列的算法被提出用于解决移动对象的KNN查询问题。然而,这些算法关注于查询的快速响应问题或者专注于解决移动对象的快速更新问题。随着移动对象数量的不断增加,当查询和更新大量涌入时,吞吐量成为一个更重要的问题。针对移动对象的更新数据流和查询数据流,提出了一种基于内存的高吞吐量移动对象KNN查询算法——DSRNKNN算法,用于处理路网中移动对象的KNN查询。DSRNKNN算法采用了基于快照的模式。在每个快照中,DSRNKNN算法通过重新构建索引的方式避免了复杂的索引维护操作,充分发挥了硬件的性能;通过每次执行一组查询的方式,充分利用查询内和查询间的并行,增加了数据的局部性,提高了算法的效率。在基于实际路网生成的数据集上对算法进行了测试,实验验证了DSRNKNN算法具有很好的性能表现。

**关键词** 时空数据库,移动对象,KNN查询,主存
**中图法分类号** TP301 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2017.03.004

## Throughput Oriented Real-time Query Processing Algorithm for Moving Objects in Road Network

XUE Zhong-bin[1,2,3] BAI Li-guang[1] HE Ning[1] ZHOU Xuan[2,3] ZHOU Xin[2,3] WANG Shan[2,3]
(Guohua (Beijing) Electric Power Research Institute Co. Ltd, Beijing 100069, China)[1]
(MOE Key Laboratory of Data Engineering and Knowledge Engineering (Renmin University of China), Beijing 100872, China)[2]
(School of Information, Renmin University of China, Beijing 100872, China)[3]

**Abstract** With the rapid development of wireless communication technology, space positioning technology and mobile computing technology, location based queries have become an important research issue in the area of database. In this paper, we studied the problem of moving object KNN query in road network. A series of algorithms have been proposed to process KNN queries of moving objects. However, these algorithms are either designed for fast response time or high update performance. With the increasing of moving objects, when both queries and updates arrive at a very high rate, the throughput becomes more important. For the query stream and object update stream, we proposed a high throughput main memory algorithm——dual stream road network KNN (DSRNKNN) algorithm, which is used for moving object KNN query in road network. DSRNKNN uses a snapshot approach. In each snapshot, DSRNKNN builds a new index structure based on the update of the moving objects, which avoids complex index maintenance operations and gives full play to the performance of the hardware. DSRNKNN executes a batch queries at each run, making full use of inner-and inter-parallel, which increases the data locality and improves the efficiency of the algorithm. We conducted a comprehensive performance study of the proposed techniques by using the real network generated data. The results show that DSRNKNN is highly efficient.

**Keywords** Spatial temporal database, Moving object, KNN query, Main memory

到稿日期:2015-11-27 返修日期:2016-04-29

**薛忠斌**(1985—),男,博士后,主要研究领域为内存数据库、移动对象数据库;**白利光**(1959—),男,主要研究领域为发电企业经营管理;**何 宁**(1979—),男,高级工程师,主要研究领域为企业大数据处理、发电企业信息化建设;**周 烜**(1979—),男,博士,副教授,主要研究领域为数据库;**周 歆**(1991—),男,硕士生,主要研究领域为高性能数据库;**王 珊**(1944—),女,硕士,教授,主要研究领域为高性能数据库新技术、内存数据库新技术、Video数据库技术、数据仓库与大数据管理与分析技术等。

## 1　应用背景

近年来随着无线通信技术、移动定位技术和互联网技术的快速发展，大量的移动设备如智能手机、平板电脑和各种车载设备等逐渐融入人们的日常生活。移动设备的广泛普及和无线通信技术的快速发展，促使移动对象的查询处理成为研究的热点。

本文充分运用大内存和多核等新硬件特性，解决了当路网中的查询数据流和对象更新数据流大量涌入时，查询响应的实时性和系统吞吐量最大化问题。随着硬件技术的不断发展，大内存和多核处理器逐渐成为市场的主流。内存容积的不断增大，允许工作负载全部放入内存，从而提高了查询的响应时间。处理器核数的增多，允许多个查询并发执行，从而提高了系统的吞吐量。

本文主要有以下贡献：

(1)本文针对路网中移动对象的位置更新数据流和用户查询数据流大量涌入的场景，提出了一种高吞吐量的移动对象实时 KNN 查询算法——DSRNKNN(Dual Stream Road Network KNN)。该算法主要应对查询动态变化的场景，但也能很好地应对查询固定的场景。

(2)该算法充分利用大内存和多核的特性，通过采用快照内重新构建索引的方式，避免了对索引的复杂更新维护操作，充分发挥了硬件的性能。

(3)算法采用多个查询同时执行的方式，充分利用数据的局部性，提高系统的吞吐量。在算法的执行过程中，通过设计合理的内存存取模式和数据布局方式，充分利用查询内和查询间的并行，提高了数据的空间局部性和缓存命中率，从而实现了系统的高吞吐量。

(4)在基于实际路网生成的数据集中，通过与已有算法进行实验对比，验证了算法的有效性。

## 2　相关工作

### 2.1　路网中移动对象的 KNN 查询算法

随着应用需求的不断发展，路网中移动对象的 KNN 查询问题逐渐成为研究的热点。

*UNICONS* 算法[1] 主要解决了路网中移动对象的持续 KNN 查询问题。IMA 算法[2] 采用快照的方式，在每个快照中处理多用户并发连续 KNN 查询。文献[3]中提出了一种新的时空查询框架——ROAD。文献[4]在 ROAD 框架的基础上，提出了一种 SQUARE 算法。在 SQUARE 算法中，采用 20-80 原则，对查询比较频繁的区域的查询结果进行存储，基于存储的查询结果，快速响应对该区域的查询，提高查询的效率。文献[5]讨论了在大数据的情况下，用户需要排队以等待查询响应，用户的排队等待时间严重影响了服务质量。同时，在查询过程中，查询具有同质性，即在同一个时刻，用户的查询相似，一个查询的结果可以被多个查询共用，从而减少查询的响应时间，提高系统的吞吐量。

### 2.2　多核查询算法

随着计算机硬件技术的不断发展，学术界对基于大内存和多核等新硬件的算法也展开了相关研究。

文献[6-7]分析了在不同的处理器架构下，内存中 hash join 算法的性能表现。文献[8]分析了内存中多核并行 join 算法(sort merge 算法和 radix join 算法)在 SIMD 技术和 NUMA 架构中的性能表现。文献[9]通过充分利用大内存和多核的特性，处理了欧式空间下的移动对象范围查询和 KNN 查询问题。文献[11]针对当前流行的自适应索引算法，讨论并验证了多核自适应索引算法的性能表现。

综上所述，针对路网中移动对象的 KNN 查询，产生了丰富的研究成果。但已有的成果不能很好地应对高并发的查询数据流和更新数据流，因此需要重新考虑位置服务的特点，面向特定的位置服务，充分利用当前大内存和多核的硬件特点，设计出合理的算法，提高位置服务的质量。

## 3　预备知识

DSRNKNN 算法关注于处理路网中移动对象的 KNN 查询。路网定义如下。

**定义 1**　路网用一个无相加权图 $G=(V,E)$ 来表示，其中 $V$ 为节点的集合，代表路网中的连接处，$E$ 为无相边的集合，代表路网中两个节点之间的线段，同时 $E\subseteq V\times V$。

**定义 2**　对于任意路网 $G(E,V)$，每条边表示为 $e(v_1,v_2)$，其中 $v_1,v_2$ 为两个联通的节点，$v_1$ 为起始节点，$v_2$ 为最终节点，每条边的权重为非负值。

DSRNKNN 算法用一个服务器管理移动对象，对移动对象进行周期性采样，移动对象会向服务器发送一个更新信息，包含 id 和坐标。移动对象的位置为一个时间函数。文献[2]也采用了该方法，它对移动对象的位置提供了一个很好的近似。对于到来的查询，也在同一个服务器中进行处理。用户在 $T_1$ 时刻提交的查询基于对 $T_0$ 时刻形成的位置快照进行计算，其中 $T_0\leqslant T_1$，$T_1-T_0<\Delta T$，$\Delta T$ 为一个固定的时间间隔，查询结果在 $T_0+\Delta T$ 时间段内有效。

**公理 1**　对于在路径 $\overrightarrow{n_i,n_j}$ 中的 KNN 查询 $q$，其查询结果 $Rq\subseteq(O(n_i,n_j)\cup R_{n_i}\cup R_{n_j})$，其中 $R_q$ 为查询 $q$ 的 KNN 结果，$O(n_i,n_j)$ 为路径 $\overrightarrow{n_i,n_j}$ 中的移动对象，$R_{n_i}$ 为节点 $n_i$ 的 KNN 结果，$R_{n_j}$ 为节点 $n_j$ 的 KNN 结果。

证明：公理 1 能够自证明，在此省略。

## 4　DSRNKNN 算法

在 DSRKNN 算法中，主要针对路网中高并发的查询数据流和移动对象数据流，充分运用大内存和多核处理器等新硬件的特性，处理移动对象的更新和 KNN 查询。

### 4.1　索引移动对象和查询

在算法中，采用快照的方式对移动对象进行周期性采样。在每个快照中，所有移动对象位置信息存储在对象缓存器 OB 中。算法充分运用计算机多核的特性，采用多线程并行处理的方式，对移动对象进行聚集索引。首先，把对象缓存器 OB 中的数据按照使用的线程数进行划分，每个线程处理一块移动对象数据。对于每个线程中的移动对象，先计算出移动对象与它所在路径中两个端点的距离，以确定移动对象所在的聚集节点。当对所有的移动对象计算完成，并得到每个移动对象的聚集节点信息后，分配一块存储空间 OT 用于存储聚

集的移动对象。然后采用多线程处理的方式，以 p. cluster 为哈希值，通过 radix hash 方式[6]把 OB 中所有的移动对象聚集到 OT 表中。

对于查询，采用类似的处理方式，把位于 QB 中的所有查询按照其所在的边进行聚集，通过 radix hash 函数，最终存储到 QT 表中。

### 4.2 处理查询

通过 4.1 节的操作，把移动对象和查询按照所在的节点和所在的边进行聚集索引，分别存储在 OT 和 QT 表中。充分利用多核的特性，通过批处理的方式，一次处理多个查询。算法采用队列存储查询所涉及的边。在算法的执行过程中，每个线程通过 round-robin 的方式处理一条边上的查询，各个线程间相互独立，从而避免了线程间的竞争，提高了算法的效率。

**算法 1** Search_Node 算法

```
Input:n,d(q,n),q. kNN_dist
Output:R
1. if n. kNN is computed
2.   for each object oi in n. kNN
3.     if d(q,n)+d(n,oi) <=q. kNN_dist then
4.       R=R ∪{oi,d(q,n)+d(n,oi)}
5.     end if
6.   end for
7. else
8.     for all the vertices ni that n adjacent
9.       push ni and d(n,ni) into queue Q
10. end for
11. while Q is not empty
12. pop Q with node ni and distance d(n,ni)
13. if d(ni,n) < n. kNN_dist then
14.     if ni. kNN is computed then
15. for each object oi in ni. kNN
16. if d(ni,oi)+d(ni,n) < q. kNN_dist then
17. n. kNN=n. kNN ∪{oi,d(ni,oi)+d(ni,n)}
18.           end if
19.       end for
20. else
21.   search all the objects oi in ni
22. if d(ni,oi)+d(ni,n) < n. kNN_dist
23. n. kNN=n. kNN ∪{oi,d(ni,oi)+d(ni,n)}
24. for all the vertices nj that ni adjacent
25.   if d(ni,n)+ d(ni,nj) < n. kNN_dist then
26. Push nj and d(n,ni)+d(ni,nj) into queue Q
27.     end if
28. end for
29. push n and d(q,n) into PQ
30.     end while
31. end if
32. return R
```

算法 1 给出了在节点中处理查询的算法。当查询执行的过程中节点 $n$ 的 KNN 结果存在时，直接在节点 $n$ 的 KNN 结果集中进行查询，得到查询结果，不需要继续探索，如算法中第 1－6 行所示。否则，就要对节点 $n$ 的所有邻接点进行搜索，构建节点 $n$ 的 KNN 查询结果。在处理过程中，首先把节点 $n$ 的所有邻近点放入队列 Q 中，如算法中第 8－10 行所示。对于队列 Q 中的节点 $n_i$，当两个节点间的距离小于 $n$ 中 KNN 查询最远节点的距离时，需要从 $n_i$ 节点中查找节点 $n$ 的 KNN 结果。当节点 $n_i$ 中的 KNN 查询结果已经得到时，直接在 KNN 结果中进行查找，如代码中第 14－19 行。若节点 $n_i$ 中无 KNN 查询结果时，在以 $n_i$ 为中心聚集的所有对象中进行查找，同时把节点 $n_i$ 的邻接点放入队列 Q 中，如代码中的第 21－30 行。当得到节点 $n$ 的 KNN 查询结果后，把节点 $n$ 放入队列 PQ 进行处理。

图 1 给出了 DSRNKNN 算法的示例。在两个线程执行时，首先线程 1 处理路径$\overrightarrow{n_1,n_2}$中的查询 $q_1$ 和 $q_2$，线程 2 处理路径$\overrightarrow{n_3,n_4}$上的查询 $q_3$。在执行的过程中，对于线程 1 需要构建节点 $n_1$ 和 $n_2$ 的 KNN 结果集，线程 2 需要构建节点 $n_3$ 和 $n_4$ 上的 KNN 结果集。当查询继续执行时，线程 1 处理$\overrightarrow{n_3,n_5}$路径上的查询 $q_5$，线程 2 处理$\overrightarrow{n_5,n_6}$路径上的查询 $q_6$。对于线程 1，因为节点 $n_3$ 的 KNN 结果集已经得到，索引只需要查询节点 $n_5$ 的结果集。对于线程 2 需要构建节点 $n_5$ 和 $n_6$ 的 KNN 结果集，节点 $n_5$ 在两个查询中共用，所以只需要一个线程查询。在查询的节点 $n_5$ 的 KNN 结果集时，当搜索到节点 $n_3$ 后就不需要再对节点 $n_2$ 和 $n_4$ 进行搜索。因为 $n_3$ 的 KNN 结果集已经得到，所以减少了搜索的范围。通过图 1 的例子可以看到，在查询的执行过程中只需要搜索很少的节点就能完成 KNN 的查询，同时在多线程执行过程中，很多节点的 KNN 结果集已经得到，不需要重复计算，提高了算法的查询效率。

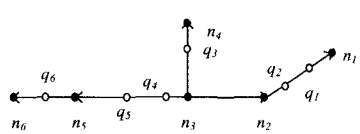


图 1 DSRNKNN 查询示例

## 5 实验结果与分析

### 5.1 实验设置

本节通过与 INE 算法进行实验对比，验证了 DSRNKNN 算法的有效性。在实验中，与文献[10-11]相同，使用了两个实际的路网数据集：CA 和 SF。这两个数据集被广泛地用于评价路网中查询的性能表现[4,12]，其中 CA 为 California 的实际路网，包含 21048 个节点和 21693 条路径，SF 为 San Francisco 的实际路网，包含 174956 个节点和 223001 条边。用开源的移动对象路径生成器 MOTO[13]生成数据。MOTO 是在数据生成器 Brinkhoff[14]的基础上形成的。MOTO 采用了一种基于路网的对象布局方式，所有的移动对象都随机分布在一个给定的路网中，设定移动对象的最大车速 $S_{max}=60\text{m/s}=216\text{km/h}$。数据生成器还按照现实中城市人口的分布进行了修改，以确保更新最频繁的区域，即查询最多的区域。

所有的算法用 C/C++实现，用 g++在最高的优化等级下进行编译。实验运行在 32 核（2 Intel E5-2670@2.6 GHz，物理核数为 16，采用超线程技术）的计算机上，使用了 SUSEOS 11 (64-bit)系统，有 256G RAM，片上的内存被所有的线程共享。

### 5.2 算法性能表现

在 DSRNKNN 算法中对于到来的查询数据流和移动对象更新数据流采用了快照的处理方式。在每个快照中，对于移动对象的更新采用重新构建索引的方式，移动对象聚集在距离最近的节点上。通过这种方式避免了传统算法中复杂的索引维护操作，充分发挥了硬件的特性。

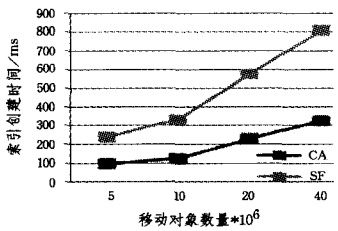


图 2　不同数量的移动对象的索引创建时间

首先测试了对于不同数量的移动对象创建索引时所用的时间，通过图 2 可以看到，随着移动对象数量的增加，索引创建的时间也随之增加；同时，随着移动对象数量的增加，SF 中索引的创建时间远远大于 CA 中所用的时间。因为在 SF 路网中，节点的数量远远大于 CA 中的数量，索引创建的时间主要与路网中节点的数量相关。

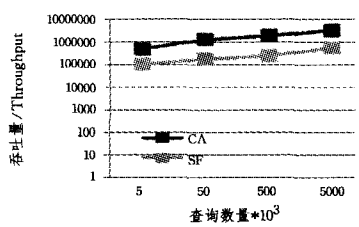


图 3　不同查询时系统的吞吐量

在 DSRNKNN 算法中，对查询按照所在的路径进行聚集，把同一条路径上的所有查询存储在查询表的相邻位置。在执行的过程中，对于查询采用了批处理的方式，通过一次处理多个查询，消除了查询的排队等待时间，提高了系统的吞吐量。图 3 测试了同时执行不同数量的查询时，系统吞吐量的变化，横坐标为查询的数量(5000～5000000)，纵坐标为系统的吞吐量(在此采用了对数的方式)。从图 3 可以看出，随着查询数量的增加，系统的吞吐量也随之增加，并且呈线性增长。

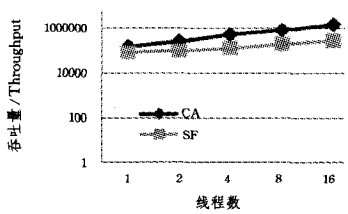


图 4　不同线程下系统的吞吐量

在 DSRNKNN 算法中，充分利用硬件多核的特性，采用多线程并行处理的方式。在执行过程中，每个线程处理路网中的一条路径上的查询。通过使用一个队列，采用 round-robin 的方式消除了线程间的竞争。图 4 显示了针对 CA 和 SF 数据集，在不同线程下系统吞吐量的变化，横坐标为采用的线程数，纵坐标为系统的吞吐量(在此采用了对数的方式)。从图 4 中可以看到系统的吞吐量随线程数的增加呈线性增长。

在 DSRNKNN 算法中，对于移动对象的 KNN 查询基于公理 1 进行计算，即路径上的 KNN 查询结果包含在路径两个端点的 KNN 结果集中。因此在执行过程中，首先计算出路径两个端点的结果集，然后再计算查询的 KNN 结果。当路径中包含多个查询时，只需要计算两个端点的 KNN 结果，即可得到路径上所有查询的结果，通过这种方式减少了查询的搜索范围。同时，对于路径上的节点，当在执行过程中搜索到某个节点已经有 KNN 结果集时，就不需要通过这个节点继续扩展，从而进一步减少了查询的搜索范围。图 5 对比了不同 $K$ 值时，系统吞吐量的变化。随着 $K$ 值的增加，路径中节点 KNN 结果的搜索范围也需要进一步增加，需要进一步扩大搜索范围。从图 5 中可以看到，随着 $K$ 值的不断增加，系统的吞吐量逐渐下降。

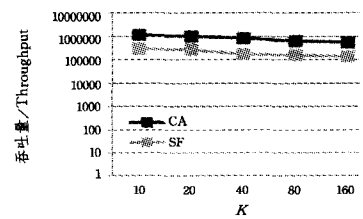


图 5　不同 $K$ 值时系统的吞吐量

### 5.3 对比实验

本节主要对比了 INE 算法和 DSRKNN 算法在不同查询数量和 $K$ 值下，算法的性能表现。

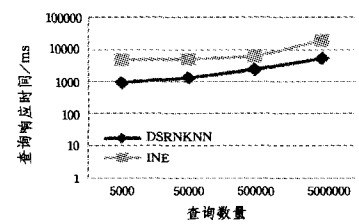


图 6　INE 算法与 DSRNKNN 算法的对比

图 6 显示了在不同查询数据量下，DSRNKNN 算法和 INE 算法的性能表现。从图 6 中可以看到，随着查询数量的增多，INE 算法和 DSRNKNN 算法的响应时间都增加，同时 DSRNKNN 算法的查询性能优于 INE 算法。

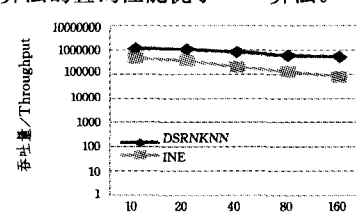


图 7　INE 算法与 DSRNKNN 算法在不同 $K$ 值下系统的吞吐量

图 7 显示了在不同的 $K$ 值下，INE 算法和 DERNKNN 算法的性能表现。INE 算法在执行过程中采用了路网扩展的方式，随着查询 $K$ 值的增加，它需要搜索遍历的路网范围增加。同时，算法中不同的查询之间相互独立，查询得到的结果不能重用，从而增加了算法的执行时间，降低了算法的吞吐量。DSRNKNN 算法采用对移动对象和查询构建索引的方式，使查询过程中很多结果得到重用，从而提高了算法的吞吐量。

**结束语**　本文主要解决了路网环境下当移动对象的更新

(下转第 41 页)

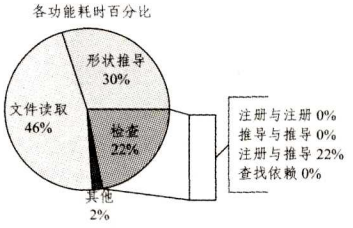


图 8　离线分析中各个分项的时间开销(Chombo 的 relax())

实验表明文件信息的读取开销巨大，为了优化这部分开销，将来需把离线分析改为在线分析。实验中访存区域的逻辑形状推导的开销也较大，这主要是因为轨迹信息的处理量较大。工具的分析检查则耗时较少，这得益于数据压缩的表示。

由图 8 可知，在所有的检查工作中，注册与推导的全检查是最耗时的部分，而其他由用户给出的特定 task 号的抽样性检查的耗时则几乎可以忽略。

**结束语**　本文介绍的 AceMeshCheck 能帮助用户定位 AceMesh 程序中并行相关的错误，并提供有用的反馈信息，从而降低 AceMesh 应用的并行化难度。相比 StarSscheck[3]，本工作的特色有：能检查一些全局性的错误；能解决网格应用中多维逻辑区域的形状识别问题。

未来会增加 AceMesh 编译器对 AceMeshCheck 的支持，自动把 AceMesh 制导转化为 AceMeshCheck 的宏功能接口。此外，也会进一步降低 AceMeshCheck 的时间开销，使其性能接近 Memcheck。

## 参 考 文 献

[1] ETSION Y, CABARCAS F, RICO A, et al. Task Superscalar: An Out-of-Order Task Pipeline[C]//Proceedings of the Annual International Symposium on Microarchitecture (MICRO-43). 2010: 89-100.

[2] GAUTIER T, LIMA J V F, MAILLARD N, et al. XKaapi. A Runtime System for Data-Flow Task Programming on Heterogeneous Architectures[C]//27th IEEE International Parallel & Distributed Processing Symposium (IPDPS). 2013: 1299-1308

[3] CARPENTER P M, REMIREZ A, AYGUADE E. Starsscheck: A Tool to Find Errors in Task-Based Parallel Programs[M]// Euro-Par 2010-Parallel Processing. Springer Berlin Heidelberg, 2010: 2-13.

(上接第 19 页)

数据流和查询数据流大量涌入时的高吞吐量的移动对象 KNN 查询问题，提出了一种新的基于内存的移动对象 KNN 查询算法——DSRNKNN。算法采用了快照的处理方式，在每个快照内，基于移动对象的更新，重新构建索引，避免了复杂的索引维护操作，充分发挥了硬件的特性。对于查询，采用了批处理的方式。基于查询所在的边进行聚集，每次处理一条边上的查询。在执行过程中，首先计算边上两个端点的 KNN 结果，基于端点的 KNN 结果，能够快速地得到该边上所有查询的结果。同时，节点可能被多条边共有，只需要对节点计算一次，便可以为多边使用，减少了搜索范围，提高了查询效率。通过对 California 和 San Francisco 的实际路网的数据集进行实验，验证了算法的有效性。

## 参 考 文 献

[1] CHO H J, CHUNG C W. An efficient and scalable approach to CNN queries in a road network[C]//Proceedings of the 31st International Conference on Very Large Data Bases. VLDB Endowment, 2005: 865-876.

[2] MOURATIDIS K, YIU M L, PAPADIAS D, et al. Continuous nearest neighbor monitoring in road networks[C]//Proceedings of the 32nd International Conference on Very large Data Bases. VLDB Endowment, 2006: 43-54.

[3] LEE K C K, LEE W C, ZHENG B, et al. ROAD: a new spatial object search framework for road networks[J]. IEEE Transactions on Knowledge and Data Engineering, 2012, 24(3): 547-560.

[4] CHEN Y J, CHUANG K T, CHEN M S. Coupling or decoupling for KNN search on road networks?: a hybrid framework on user query patterns[C]//Proceedings of the 20th ACM International Conference on Information and Knowledge Management. ACM, 2011: 795-804.

[5] CHEN Y J, CHUANG K T, CHEN M S. Spatial-temporal query homogeneity for KNN object search on road networks[C]// Proceedings of the 22nd ACM International Conference on Conference on Information & Knowledge Management. ACM, 2013: 1019-1028.

[6] BALKESEN C, TEUBNER J, ALONSO G, et al. Main-Memory Hash Joins on Modern Processor Architectures [J]. IEEE Transactions on Knowledge & Data Engineering, 2015, 27(7): 1754-1766.

[7] TEUBNER J, ALONSO G, BALKESEN C, et al. Main-memory hash joins on multi-core CPUs: Tuning to the underlying hardware[C]// 2013 IEEE 29th International Conference on Data Engineering (ICDE). IEEE, 2013: 362-373.

[8] KIM C, KALDEWEY T, LEE V W, et al. Sort vs. Hash revisited: fast join implementation on modern multi-core CPUs[J]. Proceedings of the Vldb Endowment, 2009, 2(2): 1378-1389.

[9] SIDLAUSKAS D, SALTENIS S, JENSEN C S. Processing of extreme moving-object update and query workloads in main memory[J]. The Vldb Journal, 2014, 23(5): 817-841.

[10] LI F, CHENG D, HADJIELEFTHERIOU M, et al. On trip planning queries in spatial databases[M]// Advances in Spatial and Temporal Databases. Springer Berlin Heidelberg, 2005: 273-290.

[11] LI F. Real Datasets for Spatial Databases: Road Networks and Points of Interest[OL]. http://www.cs.fsu.edu/ lifeifei/SpatialDataset.htm. 2004

[12] LEE K C K, LEE W C, ZHENG B. Fast object search on road networks[C]//Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology. ACM, 2009: 1018-1029.

[13] DITTRICH J, BLUNSCHI L, SALLES M A V. Indexing moving objects using short-lived throwaway indexes[M]// Advances in Spatial and Temporal Databases. Springer Berlin Heidelberg, 2009: 189-207.

[14] BRINKHOFF T. A framework for generating network-based moving objects[J]. GeoInformatica, 2002, 6(2): 153-180