申请上海交通大学学士学位论文

相似轨迹查询方法设计与实现

论文作者 戚　文韬

学　　号 5130309593

导　　师 朱燕民教授

专　　业 计算机科学与技术专业

答辩日期 2017 年 5 月 15 日

上海交通大学
学位论文原创性声明

本人郑重声明：所呈交的学位论文，是本人在导师的指导下，独立进行研究工作所取得的成果。除文中已经注明引用的内容外，本论文不包含任何其他个人或集体已经发表或撰写过的作品成果。对本文的研究做出重要贡献的个人和集体，均已在文中以明确方式标明。本人完全意识到本声明的法律结果由本人承担。

学位论文作者签名：_____

日　期：_____年_____月_____日

上海交通大学
学位论文版权使用授权书

本学位论文作者完全了解学校有关保留、使用学位论文的规定，同意学校保留并向国家有关部门或机构送交论文的复印件和电子版，允许论文被查阅和借阅。本人授权上海交通大学可以将本学位论文的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存和汇编本学位论文。

本学位论文属于

保　密 □，在 _____ 年解密后适用本授权书。

不保密 □。

（请在以上方框内打 ✓）

学位论文作者签名：_____　　指导教师签名：_____

日　期：_____年 _____月 _____日　　日　期：_____年 _____月 _____日

相似轨迹查询方法设计与实现

摘　要

随着 GPS 技术和移动物体追踪技术的快速发展，相似轨迹搜索和轨迹匹配在许多应用中都有着很重要意义。在本文中，我们研究并实现一种基于地理位置点集的相似轨迹搜索。在这种搜索中，问题的输入通常为一组用户定义的有序或无序的轨迹点集，问题本质在于从轨迹数据集中找到 k 条最好连接这些查询点的已有轨迹，而从一般意义上说，这 k 条轨迹也可试做相对于查询点集的相似轨迹。不同之前传统的相似轨迹查询设计，我们对相似轨迹查询的实现重点更在于用户特定的轨迹点或轨迹中在地理语义上较为重要的轨迹点。不同于以一条轨迹作为数据所产生相似轨迹结果，我们的结果能满足用户特定的查询需求。

相似轨迹获取的前提在于对相似度方法的定义。在本文的应用场景中，我们首先将将一条轨迹相似度定义为相对于查询点集的连接程度。对于用户实际运用而言，查询点集数量一般较少。本文根据这一点实现多方面的基于增长型 k 最近邻查询的相似轨迹查询方法。在空间相似度程度上，我们定义上界与下界以用于剪枝和优化查询到的相似轨迹。将 k 最近邻查询通过最好优先搜索或深度优先搜索的扩展，实现一个轨迹点在 R 树数据结构上的查询，以满足增长型 k 最近邻查询的需求，且保证了搜索的高效性和内存存储空间的保证。之后再对搜索参数进一步动态调整以加速相似轨迹搜索的过程。本文认为这种基于地理位置点的相似轨迹查询在为在路径规划、轨迹推荐、交通流量分析、拼车出行和基于地理位置的一些应用中这种基于查询点的相似轨迹查询都能够在通常意义上发挥作用。

关键词： 相似轨迹查询　k 最近邻　算法设计　优化

Design And Implementation of The Query Method of Searching Similar Trajectories

ABSTRACT

With the quick development of Global Position System technology and moving-object tracking technology, similar trajectory search and matching is becoming increasingly important in many applications. In this work, we study and implement a new type of similar trajectory searching based on geographical locations, where the input of search is a small set of user-defined location points with or without order restraint. The essence of searching problem lies in finding the k Best Connected Trajectories in the trajectory database to connect these locations points. To some extent, we consider these trajectories as the result of similar trajectory searching. Different the traditional search of similar trajectory, we focus more on the location points that users specifically define or the ones weigh more in the geographical context. This type of search can satisfy the user-defined demand more than the gerenal-purpose similar trajectory search before.

In our work, the prerequisite of searching similar trajectory is to define the similarity function. We first define the function to measure how well a trajectory connects the query location points in our application context. In practice, the number of query points tends to be relatively small. Upon this observation, we implement the search of similar trajectory based on the *Incremental k-NearestNeighbor* algorithm we proposed. The Incremental k-NearestNeighbor prunes and refines the search process by the pre-defined upper bound and lower bound of trajectory similarity. In this algorithm, we extends the k-NearestNeighbor by best-first search of depth-first search on a spatial index structure, R-tree, in order to guarantee the efficiency of searching and lower the lost of memory usage. Another contribution of our work lies in the further optimization of the search process by dynamically adjusting the searching parameters. We believe that this type of search may bring significant benefits to users in many applications, such as route planning, trajectory recommendation, traffic analysis, carpooling and location-based services in general.

KEY WORDS: Similar trajectory search,k-NearestNeighbor, Algorithm design, Optimization

目 录

插图索引

表格索引

第一章　绪论

1.1　轨迹数据处理背景阐述

计算机信息处理和存储技术的高速发展和快速普及，使得各个应用行业和科研领域的工作规模在数据上呈现爆炸式的增长。早期人们从数据规模出发用大数据（*Big Data*）对这一概念进行定义，现在大数据这一定义更多地从信息处理技术和需求方法出来，代表着我们从大数据分析与应用中所需求的新的应用和新的发展。随着计算机飞速进步的处理能力，我们能够从大数据挖掘中获取到我们所需要的应用价值。

传统企业数据、机器与传感器数据和社交数据被认为是如今大数据大致的三个类别。轨迹大数据主要属于机器与传感器数据。现代社会地理位置获取和移动计算科技进步，促使轨迹数据的大规模发展。这些轨迹数据体现了例如人类，车辆以及动物等移动物体的移动多样性。在过去十几年间，许多旨在处理、管理和挖掘轨迹数据的算法与技术许多应用中有着广泛而重要的应用价值。如今以轨迹数据挖掘为首的轨迹数据处理技术已经日趋系统且规范，从轨迹数据生成，到轨迹数据预处理，再到轨迹数据管理，最后到多样的数据挖掘任务（例如轨迹模式挖掘、轨迹异常检测、轨迹分类等等）。已有轨迹处理和轨迹挖掘的技术在相互应用中有着重要的联系与关联，轨迹数据转化成其他轨迹形式，例如图、矩阵和张量的方法也在越来越多的轨迹数据挖掘和机器学习领域有着常见的应用。

轨迹从概念上定义是一个移动物体的移动轨迹，轨迹数据可以用于许多领域的复杂分析。例如，公共交通系统可以应用过去时刻的轨迹数据分析交通流量模式并找出致使交通拥塞的原因；生物领域的动物长途迁移轨迹或是短途移动变化可以为人类提供宝贵的数据分析人类活动对生态环境的影响程度；还可以通过分析数据预测城乡车辆移动情况并及时提供符合公众出行的公共交通支持。其他应用领域也包括了路径优化设计，公共交通安全管理和基于兴趣点的用户个性化服务。

基于以上应用情景，轨迹数据挖掘在计算机科学、社会学和地理学领域都变得愈发重要。在轨技数据挖掘领域研究从深度和广度都已经取得了不错的成果，从图1–1[1] 可以看出当前轨迹数据挖掘与处理的基本研究步骤。本课题相似轨迹查询方法设计与实现主要基于其该范例中的轨迹预处理与轨迹数据索引与获取这两个领域中已存在的方法，并结合自己的理解和数据的格式实现改善和创新。

大量空间轨迹数据为我们提供了分析移动物体移动方式的可能性，这种移动方式的分析可以体现出单个轨迹所包含的某种特定移动方式或是一组轨迹所共享的相似移动方式。通常情况下相似轨迹查询是基于时空关系的查询，除此之外有些情况下一些相似轨迹查询会增加特定的查询条件，例如最快速度、偏移方向或是在规定的时间段内经过特定地理区域等等条件。在相似轨迹查询中缺少时间维度参数 (时间戳或是时间段) 是可以接受的，加入时间参数的相似轨迹查询本文将他们视为其中的一种特殊情况处理。

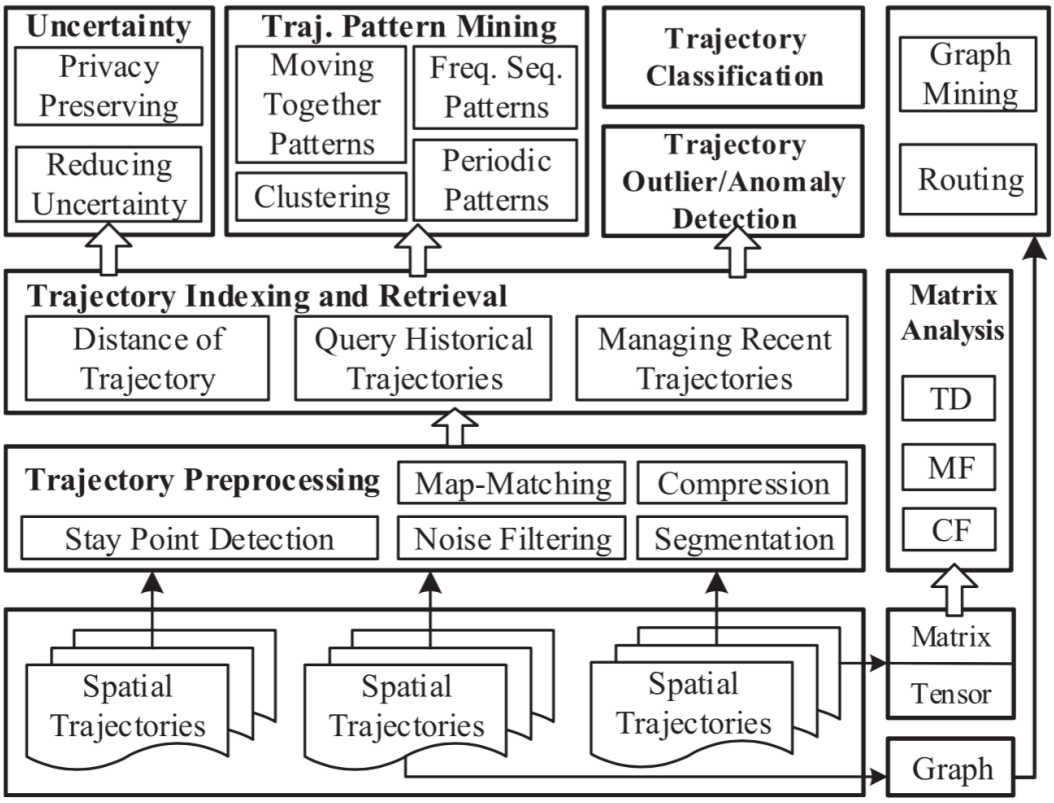

图 1–1 轨技数据挖掘范例

Fig 1–1 Paradigm of trajectory data mining

1.1.1 轨迹查询概念简介

完成在轨迹数据库中复杂的轨迹查询操作是复杂且费时的操作，因为轨迹数据库的规模一般是非常庞大的。因此，轨迹数据库的一个重要点事支持高效的轨迹索引以加速轨迹插叙过程。通常情况下，时空数据的索引技术是空间数据索引辅以时间度量参数。轨迹查询[2] 既关注经过的地理位置的拓扑位置顺序，也关注空间物体之间的距离度量，从简单的欧式距离度量到复杂的轨迹之间相似性。从大体上说，如今的轨迹查询依照时空关系分为三类:1）P-$query$，查询满足特定轨迹段或者时空关系的兴趣点或者查询针对某些兴趣点满足时空关系的轨迹；2）R-$query$，根据给定的时空区域查询轨迹或者给定轨迹查询目的区域，3）T-$query$,查询在一组轨迹数据集中查询相似轨迹或在给定的距离阈值内查询轨迹。

1.1.2 相似轨迹查询应用现状

相似轨迹查询主要是基于上述的轨迹查询方法中 P-$query$ 和 T-$query$ 展开的。

P-$query$ 主要应用在给定地理位置点后找到满足时空关系的轨迹或者轨迹段。单点轨迹查询找到针对某一给定地理位置点的最近轨迹。多点轨迹查询在给定一组地理位置点集后在轨迹数据里中

找到能在地理位置意义上连接查询点集的多条轨迹。前者用以找到某一地理位置范围内的潜在轨迹。后者在制定行进轨迹路线中有着很好的应用。$T\text{-}query$ 通常通过聚类或分类轨迹在轨迹数据库中查询轨迹。轨迹分类和聚类算法在许多应用中有着广泛的应用，例如基于移动物体特征的轨迹测或是分析路网流量结构，在轨迹集合发现共同的子轨迹以及查询与目标轨迹在欧式距离上最接近的轨迹集合。

基于 $P\text{-}query$ 的查询主要是衡量点到轨迹的中最近点的距离。目前也常通过拓展这一思路当多点的 $P\text{-}query$ 查询以评价一条轨迹连接多个查询点的好坏。在 $T\text{-}query$ 这一查询类型方面则有很多较为成熟的方法主要的不同在于他们各自的相似距离函数的定义，例如动态时间规划轨迹方法 (Dynamic Time Warping)[ref]、最长公共子序列方法 (Longest Common Subsequence)[ref]、基于编辑代价的方法 (Edit Distance With Real Penalty)[ref] 和基于序列编辑距离的方法 (Edit Distance on Real Sequences) 等等。这些方法在初期主要应用于时序相关的数据上，但是由于轨迹在某种意义上可以看成是多维度上的时序数据，上述的相似距离方法则可以应用上轨迹数据上。

1.2 相似轨迹查询问题描述

1.2.1 问题大致描述

随着轨迹数据大规模的发展与存储，在日常生活中，如何高效查询轨迹对于用户或是在工业领域都有着重要的意义。特定情况下，查询类型也会根据需求有着变化。相似轨迹查询属于轨迹查询中的一种。在这里我们队相似轨迹查询问题进行大致描述；给定一组表示一条轨迹的轨迹数据点 Q 和轨迹数据库 D，查询出在地理形状上与这些轨迹点所描述的轨迹的 k 条最相近的轨迹。

1.2.2 相似轨迹查询方法设计概述

本文研究的相似轨迹查询方法主要基于位置定的查询，即查询主要是基于一组有序或是无序的地理位置点。查询的首相要目标是从轨迹数据库中找出连接查询位置点的 k 条最佳连接轨迹 (K Best-Connected Trajectories) 使得这 k 条轨迹能够在地理位置上连接给定的未指定。不同于传统形状或其他查询标准通过给定一条轨迹的相似轨迹查询，本文的相似轨迹查询主要针对于所查找到的轨迹对于给定的一组轨迹点连接性的优劣。

如图1–2所示，通过点击地图或图像地理解码给定一组地理位置点 (图中点注释)，我们可以从数据库中获取找一条能够连接给点地理位置点原始轨迹 (图中线注释)，该实例体现出本相似轨迹查询方法在能够在包括旅游路线规划等新兴应用中更好地服务用户。与此同时，这种相似轨迹查询能还能在以上场景有所应用：旅行社或自由行游客对出行经典的路径规划；动物园能调查出动物对到某些特定地点的最短路径；交通运输部门对本地市民城乡情况的规划。

大体上，k 条最佳连接轨迹查询基于的地理位置点需要具备必要的经纬度信息 $(latitude, longitude)$。这些经纬度位置地点可以是旅游景点，不明确的沙滩或是任何一处地理坐标。用户可以通过决定轨迹连接有序或是无序以决定查询结果。例如一个简单的查询包括三个地理位置点 A,B,C

$$A_{(37.2601,122.0941)}, B_{(37.2721,122.0648)}, C_{(37.3344,122.1538)}$$

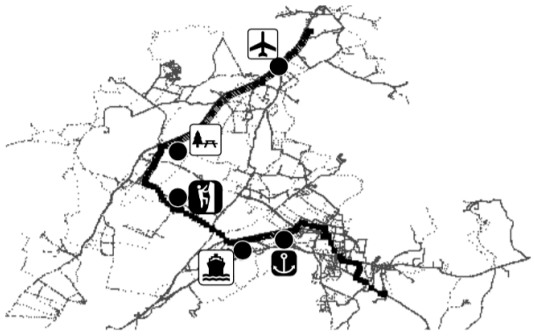

图 1–2 基于位置点的相似轨迹查询
Fig 1–2 Similar trajectory search by locations

其中 (37.2601, 122.0941) 代表 A 点的经纬度地理坐标。如果查询附带有序条件，则查询轨迹结果应保留轨迹点之间的相对顺序性，即 $A \rightarrow B \rightarrow C$。传统的相似轨迹查询方法显然无法解决查询结果与查询条件之间有序一致性。另外，本文所提出的轨迹查询方法基于任意的地理位置点使得查询模式更加多样和灵活。从本质上而言，这种这种查询方法的设计基于传统的单点查询方法以寻找针对某一位置的最为临近轨迹。

为了实现这一相似轨迹查询方法，本文首先对一组给定地理位置点中的每一个点进行基于点的查询以从数据库中找到最近的轨迹点。如果查询结果存在，通过对每一个结果的汇总所得出的轨迹从理论上而言是最接近查询点的一条轨迹且对给定的地理位置点具有良好的连接性。从这个思路出发，本文拓展 K 最近邻 (k-NearestNeighbor algorithm) 算法并提出增长性 K 最近邻 ($Incremental\ k\text{-}NN\ algorithm$) 算法。该算法增长性获取每个查询位置点的最近轨迹并不断检查以查询到的轨迹。通过自定义的轨迹相似性上界与下界，借鉴备选和筛选的算法设计思路 (candidate-refinement) 来进行优化与剪枝。利用 R 树 (R-Tree) 作为地理位置点的索引结构，算法实现过程中根据具体计算机情况和性能需求选择性使用最好优先搜索 (best-first) 方法或是深度优先搜索 (depth-first) 方法进行查询。

1.3 相似轨迹查询应用价值

相似轨迹查询系统在实际情景中具有广泛的应用价值。在路径规划应用方面上，相似轨迹查询从已记录的历史轨迹中，根据用户定义有序或无序的旅游景点顺序，给出 k 条大致符合出行路径的历史轨迹基于用户参考并选出用户自己偏好的出行路径；在轨迹推荐或拼车推荐方面上，由于限号或出行限制问题，可以通过在历史轨迹中查询相似的轨迹查询出是否有在上下班或平时出行模式较为相似的多名用户，选择在有出行限制的时段与别的用户共享出行设备；在交通分析方面，相似轨迹查询可以为公安行业根据几个具体的地理位置点锁定一辆具有车载 GPS 设备的车辆。除此之外，相似轨迹查询的具体应用还有许多，不予以赘述。

1.4 困难和挑战

相似轨迹查询方法的设计与实现存在的以下主要的困难和挑战：1) 高效准确实现相似轨迹查询。由于轨迹数据集的规模较大，通过常规查询和搜索会耗用大量时间和空间，而且在搜索过程中还要避免一些错误数据的加入。2) 基于分布式的相似轨迹查询。相似轨迹查询领域的已有工作较为丰富，但是在单机层面上的查询处理。对于轨迹大数据处理而言，将相似轨迹查询移植与分布式环境操作是必要过程。3) 实现基于相似轨迹查询功能的软件系统。轨迹数据处理的价值在于通过轨迹数据处理为每个客户提供特定服务。在相似轨迹查询中，需要提供一个用户界面为用户查询相似轨迹并将轨迹可视化的软件系统，在满足高效查询过程中需要保证良好的软件系统交互性。

1.5 论文大致结构

重写

1.6 本章小结

本章节已经初步介绍了相似轨迹查询这一概念和其相关背景，由于目前的在轨迹数据处理已经系统和规范的处理流程，轨迹数据挖掘这一领域的方法技术也已经较为成熟且丰富，通过学习传统的相似轨迹查询方法和他们各自的应用经验，本文所提出的方法在已有成果的基础上进行进一步的创新与优化，便可以使得相似轨迹查询方法与传统的相似轨迹查询有着较大的不同，且具有特定查询环境上的查询优势与性能优化。在对相似轨迹查询问题进行大致描述后，本文对解决这一问题在现实生活中有哪些具体的应用进行展开，并初步描述解决相似轨迹查询这一问题所产生的实际价值。

解决相似轨迹查询这一问题的过程中也伴随着一些具体的困难与挑战。本文通过拓展如今最为基本的 k 最近邻数据挖掘技术，以实现基础的相似轨迹查询方法为基本理论目的，移植单机运行代码至分布式环境系统为应用目标，开展毕业设计课题。

第二章　相关工作

2.1　国内外研究现状

2.1.1　相似轨迹查询方法

轨迹数据挖掘中，相似轨迹数据查询国内外有着较为丰富的研究成果。最初的相似轨迹查询通过离散傅里叶变化（Discrete Fourier Transform）将轨迹做近似匹配得到相似轨迹[3]。在此基础上，通过离散小波变换（Discrete Wavalet Transformation）在降低查询维度上的相似轨迹匹配得以实现[4]。为了能够比较不同单位长度轨迹之间相似性，Chan 和 Fu 通过动态时间规划（Dynamic Time Warping）的技术来来最小化轨迹序列之间的距离以完成相似轨迹查找[5]，虽然这一方法在处理噪音点上具有鲁棒性，但操作相对于目前算法而言过于繁琐。

较早通过欧式距离作为关键字索引轨迹的方法[6] 只能引用于在时间和空间上具有相同间隔性的轨迹数据。之后 Cai 和 Ng 提出基于切比雪夫多项式（Chebyshev polynomials）的轨迹索引和近似方法来实现相似轨迹匹配[7]。但上述方法都需要保证轨迹具有相同的长度，即相同数目的轨迹点。

Vlachos 等人用一种距离度量来转义和变换轨迹以找到相似轨迹[8]，结合动态时间规划技术，将所有轨迹变化到一个不可转变的空间再计算两个轨迹之间的相似性。Sakurai 等在动态时间规划的基础上，结合剪枝算法用下界变量来加快动态时间规划的相似轨迹查询算法[9]。Lin 和 Su 提出与时间无关、针对移动轨迹的相似搜索算法重点比较两个移动在空间上的相似性并实现查询，并在他们的实验中证明了该方法优于基于动态空间规划的相似轨迹查询。

最长公共子序列算法（Longest Common Subsequence）算法在字符串中应用也在拓展于相似轨迹查询中。最长公共子序列问题的本质描述两字符串之间的相似度，在解决问题过程中允许对两个字符串进行扩展。算法本身允许其中的某些字符不匹配，这也匹配在相似轨迹查询中允许某些点不匹配。因此最长公共子序列可高效解决噪音点问题和允许轨迹间隔不同。Vlachos 等人在最长公共子序列的基础上，提出两种针对时间变化与转移的相似度方程[10]，对于相似的子轨迹在计算中基于较大权值，并利用了高效的空间索引结构。但这一算法局需要轨迹点的采样率高度一致性。随后，一种名为 EDR（Edit Distance On Real Sequences）的方法被提出[11]。这个方法基于对距离的修改，通过三种剪枝策略以降低轨迹数据和查询数据之间的距离计算代价，在处理噪音点方面鲁棒性强于动态时间规划和最长公共子序列。但准确性上缺乏保证且对于不同采样率的轨迹处理性能较差。在此之后，ERP（Edit distance with Real Penalty)[12]、DISSIM[13] 和 MA（model-driven assignment)[14] 等等针对相似轨迹查询的算法也逐步提出且各具优势，不过他们在时间偏移、采样速率和距离阈值限制上也都有着不足。

2.1.2　相似度方程比较

图2–1对上述部分算法中涉及的相似度方程给出定义，以比较其中不同。这些方程根据自身特点与优势应用于不同的场景中[15]，包括欧氏距离方程（Euclidean Distance），动态时间规整（Dynamic Time Warping），最长公共子序列算法（Longest Common Subsequence），基于编辑代价的方法（Edit

Distance With Real Penalty）和基于序列编辑的距离方法（Edit Distance on Real Sequences）。动态时间规整（DTW）方法在比较轨迹之间相似性的过程中采用了时间偏移（time-shifting）来使得轨迹中的一些点可以尽可能多地重复出现以实现最好效果的校准。但这种方法在原有轨迹数据点出现误差（或称为噪声点）的时候会影响比较的准确性因为所有的点都需要被匹配。相比如动态时间规整方法（DTW），最长公共子序列（LCSS）选择忽略某些点以避免对他们的重排序过程，从结果上而言这种方法舍弃了偏离采样的误差点以提高了准确性，但需要人为预定距离阈值以判断什么数据属于误差点。基于编辑代价的距离方法（EDR）与 LCSS 方法类似，他们最初提出是为了解决字符串匹配问题，在轨迹数据匹配这一方面他们均采用一个阈值参数来判断两个点是否匹配，但 EDR 考虑了距离之间的衡量代价以决定是否将两个点进行匹配。在此基础上基于序列编辑的距离方法（ERP）结合 EDR 和 DTW 方法选择固定点进行距离计算。

相似度方法通常根据具体的应用进行具体的选取。但上述的相似度方法主要是基于轨迹与轨迹之前相似度的查询，在本文设计的相似轨迹查询方法上的应用度并不理想，本工作的查询条件是基于一组地理坐标点的查询，并且工作更关注与一条轨迹是否能够很好地连接上给定的一组查询点，从而提供基于轨迹点的相似轨迹结果。因此，在这样的情景下，我们需要定义一个新的相似度方程。

Definition		
DTW(R,S)	=	$\begin{cases} 0 & \text{if } m=n=0 \\ \infty & \text{if } m=0 \text{ or } n=0 \\ dist(r_1,s_1)+min\{DTW(Rest(R),Rest(S)), & \text{otherwise} \\ DTW(Rest(R),S), DTW(R,Rest(S))\} & \end{cases}$
ERP(R,S)	=	$\begin{cases} \sum_1^n dist(s_i,g),\ \sum_1^m dist(r_i,g) & \text{if } m=0,\ \text{if } n=0 \\ min\{ERP(Rest(R),Rest(S))+dist(r_1,s_1) & \\ ERP(Rest(R),S)+dist(r_1,g), & \text{otherwise} \\ ERP(R,Rest(S))+dist(s_1,g)\} & \end{cases}$
LCSS(R,S)	=	$\begin{cases} 0 & \text{if } m=0 \text{ or } n=0 \\ LCSS(Rest(R),Rest(S))+1 & \text{if } \forall d, \lvert r_{d,1}-s_{d,1}\rvert \le \epsilon \\ \max\{LCSS(Rest(R),S), LCSS(R,Rest(S))\} & \text{otherwise} \end{cases}$
EDR(R,S)	=	$\begin{cases} n,\ m & \text{if } m=0,\ \text{if } n=0 \\ \min\{EDR(Rest(R),Rest(S))+\text{subcost}, & \text{otherwise} \\ EDR(Rest(R),S)+1, EDR(R,Rest(S))+1\} & \end{cases}$

图 2–1 已有轨迹相似度函数定义[1]

Fig 2–1 Definition of existing distance functions

2.2 相关实现技术介绍

2.2.1 Flask 应用框架

Flask 框架是基于 Python 语言的一个 Web 开发微型框架。Web 框架是指用于简单实现高效编写 Web 应用的软件发开框架。在 Python 中已有的流行 Web 框架有 Django、Pyramid 等。简而言之，当用户在浏览器内输入一个待访问的网址时，会发送一个特请 HTTP 请求。与此同时，Web 框架便负

[1] $dist(ri, si)$ = L1 or L2 norm; $subcost = 0$ if $r1, t{-}s1, t$, else $subcost = 1$

责来处理这个 HTTP 请求，并分配不同的访问地址到工作人员事先已经编写好的代码，生成 HTML，最后创建附带内容的 HTTP 相应。

将 Flask 框架定义为 Web 微框架的原因是以为 Flask 本身只实现了 Web 应用中的核心内容。本质上，Flask 只依赖两个外部库：提供代码模板的 Jinja2 模板和提供 Web 服务器网关接口、路由与调用的 Werkzeurg WSGI 工作集[16]。通过第三方库来完成表单处理、用户验证、数据库操作等等任务。

2.2.2 Bootstrap

Bootstrap 是 Twitter 公司推出的一个用户 Web 前端的开源工具。作为目前最为主流的 HTML/CSS 和 JS 的开发框架，Bootstrap 通常使用于响应式分布的 Web 项目只用。作为完全开源工具，Bootstrap 中的预处理脚本可以为开发者提供可直接使用的 CSS 样式表；Bootstrap 完成一种框架应用于全平台多种设备的高度移植性；Bootstrap 提供全面的 HTML 元素、CSS 组件和 Javascript 插件，且均具有文档说明。

2.3 大规模数据集群处理

集群计算随着如今海量数据的发展在许多领域都有着广泛应用，以高效准确完成大量数据并行级的处理任务。集群计算需要提供本地化任务规划、高容错机制和数据负载平衡基本功能。除此之外，集群计算中我们也会关注某一个数据集运用在并行操作去完成指定目标任务。*MapReduce*[17] 是目前分布式处理或并行计算常用的大规模数据处理和生成的编程模型。其工作的大致思路在于用户自定义合适的 map 函数去处理初步输入的键值对数据并产生中间键值对结果，之后定义 $reduce$ 函数将中间结果以相同的关键字进行合并生成最终的结果。

对于数据密集型的应用而言，可扩展的分布式系统对于系统运行和数据处理都有着很重要的帮助。合理的分布式系统可以为系统提供在通用硬件上运行时的容错保护，并且能保证多用户请求的高度并行处理。*Hadoop* 分布式文件系统（*HDFS*）借鉴了 *Google* 文件系统（*GFS*）的大部分设计架构并实现了高度的容错保护机制并且能良好地运行于廉价的硬件设备之上。与此同时，HDFS 也保证了在应用中数据的高度吞吐速率，使得 HDFS 能高效运行具有很大数据集的任务或应用。

2.3.1 Spark 处理引擎简介

MapReduce 变成模型和 *HDFS* 可在大规模数据密集型应用良好，但对于一些需要重复使用中间数据或需要暂时保留中间数据的应用处理中，之前常用的集群计算模型 *Hadoop* 由于需要将中间数据读写与 *HDFS* 中从而产生了中间读写时间浪费，从而影响了应用性能。基于这一点，*Spark*[18] 作为在主流针对大规模数据处理的集群计算模型之一，在保证之前集群计算模型功能的同时，使用一种名为弹性分布式数据集（*Resilient Distributed Datasets*）的抽象，使得其可以将集群任务中的中间结果保存于设备的内存之中，以便之后的读写操作。因此，在大数据挖掘领域中，*Spark* 能够比 *Hadoop MapReduce* 能为高效的处理需要迭代数据的集群计算。

2.3.2 弹性分布数据集 RDD

Spark 集群计算处理引擎与之前集群处理的主要不同点即在于其引入的弹性分布式数据集（*RDD*）这一抽象概念。这一分布式内存抽象使得用户或程序员可以在容错机制的保护下在大规模集群设备中运行基于内存的数据操作。*RDD* 高效处理大数据在应用中的重用问题，作为一个并行的数据结构可以方便用户在内存中处理集群计算的中间过程数据，因地制宜分割数据集以更合理将任务分配个对应的工作节点，再结合丰富的内定操作函数以快速处理数据。而 *RDD* 提供的生成模式也能为我们设计算法提供更多思路。*RDD* 可以通过 $parallelize$ 函数将程序中已有的数据用于生成为 *RDD*，或通过对例如 *HDFS*、*Hbase* 等等的外部文件系统或外部数据源来生成 *RDD*。

2.3.3 Spark Standalone 集群模式

Spark 应用在集群模式运营中运行独立的进程组，通过驱动程序中的 *Spark* 上下文变量 *SparkContext* 来设定运行参数和初始化。运行过程主要根据 *SparkContext* 来连接如图2–2中具体不同种类的集群管理类型，并通过内定的 *Cluster Manager* 来分配应用的资源获取。初始化成功后，*Spark* 通过获取集群节点上的执行进程并准备开始执行操作和处理数据。之后，*Spark* 会将应用代码分发给各个节点并使之运行。

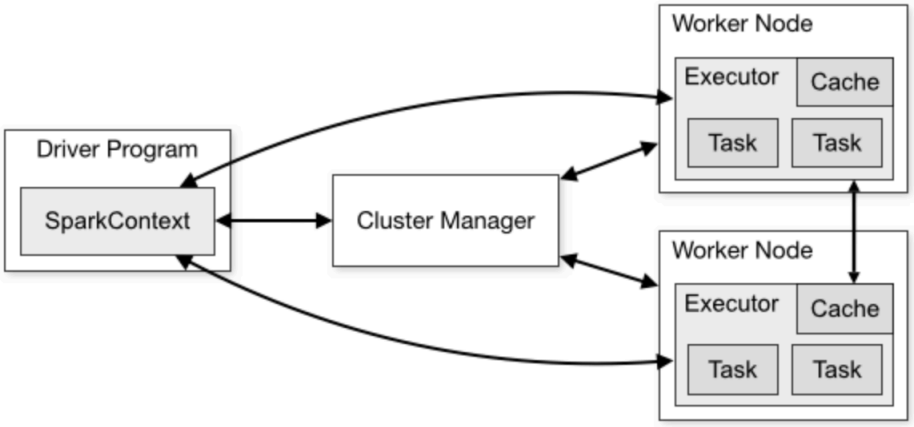

图 2–2 集群管理大致模式
Fig 2–2 Cluster Managing Mode

在相似轨迹获取这一应用中，根据我们的应用场景和硬件设置，我们采用 *Spark* 自带的 *Standalone* 集群模式，其大致设计框架如图2–3所示。

在 *Standalone* 集群模式中，我们通过一个在 *Spark* 分布式环境下的简要集群管理者来简单建立集群处理环境。在这一集群环境中，主节点（*Master Node*）为驱动程序运行的设备节点。驱动程序不仅是与用户交互信息的接口（*interface*）程序，也负责分布式运行在 *Spark* 环境中进程的运行情况。子节点们（*Slave Nodes*）为启动在工作节点中的进程提供运行环境，这些进程运行任务代码并在内存或磁盘中储存数据。在相似轨迹搜索中，我们将轨迹数据和预处理好的轨迹索引 R 树结构存储在

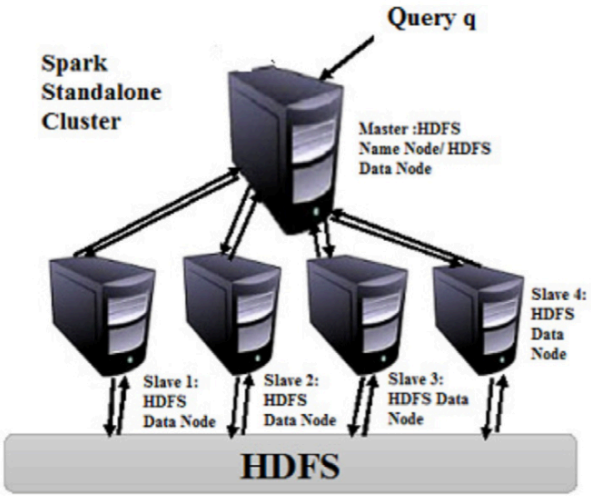

图 2–3 Spark Standalone 集群结构

Fig 2–3 Spark Standalone Architecture

HDFS 上，集群环境中的工作节点可以通过设定好的参数无需秘钥的共享 *HDFS* 上已存储好的数据，这样，我们可以将相似轨迹搜索任务进一步以分布式的方法进行处理。

2.4 轨迹数据预处理

2.4.1 WGS84 坐标系统转换至 GCJ-02 坐标系统

WGS84（World Geodetic System1984）坐标系统是 GPS 数据所基于的坐标系统，这一坐标系是通过世界卫星观测站所检测到的地理坐标。这一坐标系并不能直接应用在中国国家的地图坐标显示中，因为中国国家会测局在地理信息系统中使用的是基于 GCJ-02 的坐标系统，这一坐标系统也称为火星坐标系统。如果直接将 WGS84 坐标数据应用于使用 GCJ-02 的地图显示接口，则会造成 100 米到 700 米范围内的显示误差。同理，用户使用 GCJ-02 地图点击获得的地理位置查询点也会在相似轨迹查询中因为与 WGS84 坐标系统的偏差因素造成查询结果的不准确性。在轨迹预处理最开始先将 WGS84 坐标数据根据已有的算法[1]参考转换成 GCJ-02 系统下的坐标

2.4.2 轨迹数据简化

轨迹数据预处理中，轨迹数据的简化（或压缩）是比较重要一步。轨迹数据简化主要是指在保证轨迹的可利用性与大致准确的同时减少轨迹的点数目，以达到轨迹数据的传输、处理和存储上减少开销的目的。在本文的应用场景中，我们首先采用 *Douglas–Peucker* 算法[19] 来完成我们的轨迹简化任务。该算法的主要思路在于将通过分而治之，将曲线轨迹表示成一系列点的方法，从而减少点的数目。如今 GPS 的数据采样通常较为频繁，因此在我们对轨迹处理的范围上来所，我们可以近似地将我们所运用的数据集中的轨迹看成是一条连续的曲线，通过 *Douglas–Peucker* 算法以及我们人为设定简化阈值，我们可以高效且合理地进行轨迹简化。

[1]https://github.com/googollee/eviltransform

Douglas–Peucker 算法首先连接轨迹首尾两点 $Traj[0], Traj[Traj.length]$，遍历轨迹一遍得到离线段距离最大的点 $Traj[index]$，计算该距离并与预先设定的阈值 ϵ 比较。如果大于阈值 ϵ，则将轨迹以 $Traj[index]$ 为中点分为两端，迭代重复上述工作；如果小于阈值 ϵ，则直接将线段段作为曲线的近似以做简化。当曲线完成上述任务，依次连接处理好的子线段，完成轨迹简化任务。

2.5 轨迹数据索引与获取

空间数据结构对从一个大规模轨迹数据集中获取特定轨迹数据是十分重要的。效率问题是查询大规模数据库或数据集来获取信息的首要考虑因素。而查询效率十分依赖于合理的轨迹索引。轨迹数据根据数据特点的不同姓对索引技术也有着特殊的要求。目前主流的索引技术主要有三类：1）基于空间维度的索引，利用 R 树（R-tree）索引进行查询。通过 3DR 树（3D R-tree）或者 STR 树（STR-tree）进行带有时间维度的查询；2）利用多版本的数据结构，根据特定情况使用 MR 树（MR-tree）、HR 树（HR-tree）、MV3R 树（MV3R-tree）等等；3）将空间划分网格结构然后对应每个网格建立对应的空间索引，这类数据结构包括 MTSB 树（MTSB-tree）和 SETI。本文中我们使用 R 树这一最基本的数据结构，其满足我们对算法的实现需求。

R 树[20] 数据结构在空间数据库中应用广泛，许多轨迹索引结构大体上是基于 R 树进行拓展。R 树结构是一个平衡树结构，R 树中的每一个节点代表包含其所有子节点一个区域，这个区域通常被称为最小区域箱（Minimum Bounding Box）。节点中的每一个数据体指向对应的子节点的最小区域箱信息。R 树搜索的关键字是最小区域箱中的每一个节点。如图2–4所示的是 R 树数据结构的 2 种表现形式。在2–4(b) 中我们看到树结构而图2–4(a) 描述了数据和最小边界箱是如何分布在空间中的。

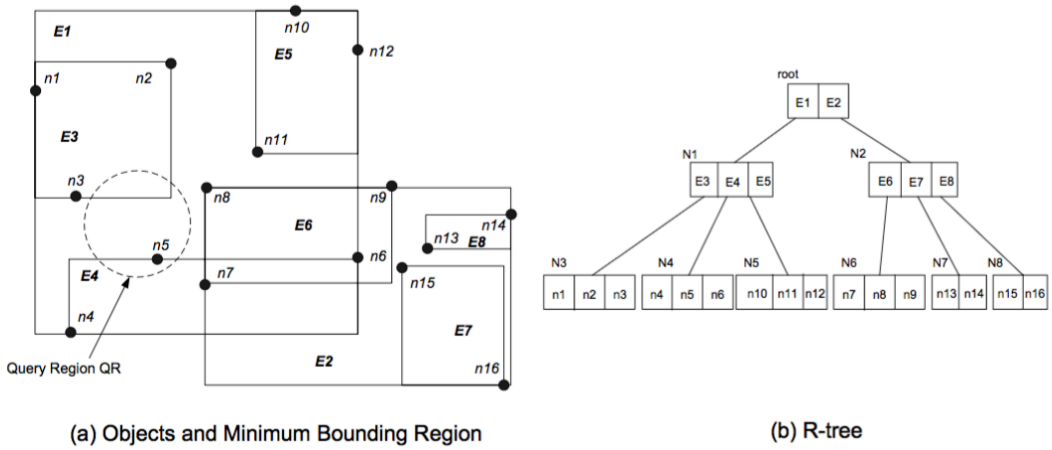

图 2–4 R 树数据结构举例

Fig 2–4 Two views of an R-tree example

在图2–4中，根节点有两个数据体 $E1$，$E2$，分别对应子节点 $N1$，$N2$。$N1$ 代表的最小边界箱包含了其子节点 $N3$、$N4$、$N5$ 以及数据体 $E1$ 所具有的最小边界箱信息。值得注意的是空间点的体

现只有在 R 树的叶节点上。R 树可以应用在范围查询和近邻查询中。本文主要使用 R 树近邻查询这一属性。给定一个查询点，R 树可以通过最优优先搜索（best-first）和深度优先搜索（depth-first）两种树遍历策略找到在数据集中最接近查询点的数据。在两种搜索策略中，查询点和每一个最小边界箱的距离被定义为变量 $mindist$。之后的搜索过程基本遵从两种搜索策略各自算法。

在 R 树中插入一个新的节点大致需要以下几个步骤。当有新的轨迹数据需要被添加到已有的 R 树种，首先为待插入的轨迹数据点找到一个合适插入的子节点中。再寻找叶结点的过程中我们会选择符合最小边界范围且对 R 树扩展度最小的一个叶结点。然后若找到 R 树叶结点数据溢出，那么我们需要对叶子结点进行分裂操作；若没有溢出，则可以将待添加的轨迹数据加入到当前已经找到的叶子节点中。最后对 R 树进行变换向上传递并对树高进行增高以完成插入操作。删除操作近似于插入过程的逆过程，在此不予以赘述。

2.6 本章小结

本章节中，本文通过图标和伪代码讨论了相似轨迹查询方法的设计与实现的基本相关工作，对本文所应用的基本定义、处理思路和数据结构有了一个初步的了解。根据本文场景定义个性化的相似度方程后，我们在查询阶段需要通过多点输入的条件下进行归集查询。由于输入数据的轨迹点数目相对较少，我们可以借助已有的数据结构进行空间距离上搜索。利用 R 树和基于 k 最邻近的进行方法拓展是本文实现算法的基本思想，以快速的搜索并获取数据。根据上述本文工作相关工作描述，我们可以得出实现本文算法的先决条件目前都是基于只有已有的成熟工作。在下一章节中，本文将开始对相似轨迹查询方法进行理论讨论。

第三章　相似轨迹查询方法实现

3.1　相似轨迹查询问题描述

相似轨迹查询传统意义上是根据一条已有历史轨迹，在轨迹数据库中查询出与这一条轨迹在地理位置上形状相似的一条或多条轨迹。在本文中，我们将输入轨迹进一步简化为一组轨迹查询点集 Q。Q 在地理位置上保留轨迹原有的形状，通过对点集 Q 进行 *k 最佳连接*查询，从轨迹数据库 D 中得出 k 条最相似的轨迹集合 T' 作为输出。由于系统设计与用户的交互，性能指标方面查询结果应满足高效、实时且具有良好的准确性。

3.2　k 最佳连接

一个轨迹数据库中存储了大量的原始车载 GPS 轨迹或是已经预处理过的车载 GPS 轨迹。这里的轨迹由一系列的地理位置点组成 $\{p_1, p_2, p_3, \cdots, p_m\}$，其中 p_i $1 \leq i \leq m$ 代表一个由经度和维度构成的地理位置点而 m 代表轨迹中点的数目。本文所定义的 k 最佳连接查询（k Best-Connected Rrajectory Query）的输入由一组查询点 Q 组成。q_j $1 \leq j \leq n$ 和 p_i 定义相同，其中 n 是查询点的数目。这里用户可以选择选择是否在查询中指定轨迹连接依照查询点的先后顺序，即是否选择查询有序性。若选择查询有序性，则查询点集 Q 为认为是从 q_1 到 q_m 有序点集。

$$Q = \{q_1, q_2, q_3, \cdots, q_n\}$$

在搜索最好连接轨迹这一上下文中，相似度方程的定义需要和传统方法有所不同，在这里我们将相似度方程定义为一条轨迹连接查询点的好坏程度。因此，本文首要考虑一条轨迹到每一个查询点的距离，我们简要定义距离一个查询点 q_i 到一条轨迹 $R = \{p_1, p_2, p_3, \cdots, p_m\}$ 的距离为 D_q, 即

$$D_q(q_i, R) = \min_{p_j \in R} \{D_e(q_i, p_j)\} \tag{3–1}$$

式3–1中，$D_e(q_i, p_j)$ 是指查询点 q_i 和轨迹点 p_j 之间的欧氏距离，因此通常意义上相似度距离 D_q 代表从查询点 q_i 到轨迹上任一一点距离的最短距离。当我们找到轨迹上一点 p_j 是离查询点 q_i 的最短距离点时，我们将 $< q_i, p_j >$ 作为最短匹配点对。在无序查询点击中，我们定义查询点集 Q 和轨迹 R 之间的相似度为 $Sim(Q, R)$。

定义 3.2.1. 轨迹 $R = p_1, p_2, \cdots, p_n$ 而查询点为 q，$< q, p_i >$ 表示一组匹配点对。对于 $\forall p_i \neq p_j$，$d_e(p_i, q) \leq d_e(p_j, q)$，那么 $< p_i, q >$ 是轨迹 R 到查询点 q 的最短匹配点对。

$$Sim(Q, R) = \sum_{i=1}^{n} e^{-D_q(q_i, R)} \tag{3–2}$$

式3–2将每个查询点对 $Sim(Q, R)$ 的贡献值通过自然对数去反得以体现，即根据自然函数的单调性，查询点离轨迹越近，则 $-D_q(q_i, R)$ 越大，以自然对数为底取幂的值也越大，最后使得 $Sim(Q, R)$

的值也越大。从用户人为角度和地理语义角度上看，一条轨迹与所有的查询点被定义为相似当且仅当这条轨迹和所有的查询点都十分接近。

图3–1通过距离说明查询点和轨迹之间的匹配关系。如图3–1(a) 所示，查询点 q_1、q_2 和 q_3 分别与轨迹 R 上的轨迹点 p_6、p_4 和 p_7 最近匹配，根据式3–2可以得出，$Sim(Q,R) = e^{-D_q(q_1,p_6)} + e^{-D_q(q_2,p_4)} + e^{-D_q(q_3,p_7)} = e^{-1.5} + e^{-0.1} + e^{-0.1}$。

另一方面，选择查询点和轨迹之前进行有序查询时，顺序性是需要在查询过程中予以考虑。对于查询点 q_i 而言，最近匹配点或许并不在是距离上最近的轨迹点 p_j。因此，相似度方程在此步骤中应该适当调整。我们再借用图3–1予以说明。假设以下用户场景：用户希望查询出一条以 $q_1 \rightarrow q_2 \rightarrow q_3$ 为顺序的相似轨迹，显然图3–1(a) 中的顺序并不再符合用户需求。实际的有序查询结果顺序如图3–1(b) 是 $p_3 \rightarrow q_4 \rightarrow q_7$。在考虑有序性的相似查询规程中，我们的目标是在保持查询有序性的同时追求每一对匹配点对相似度的最大贡献值，即从图3–1(b) 可以看出 $< q_1, p_3 >, < q_2, p_4 >, < q_3, p_7 >$ 这样的三对匹配点是所有有序匹配对中使得相似度最大的情况。

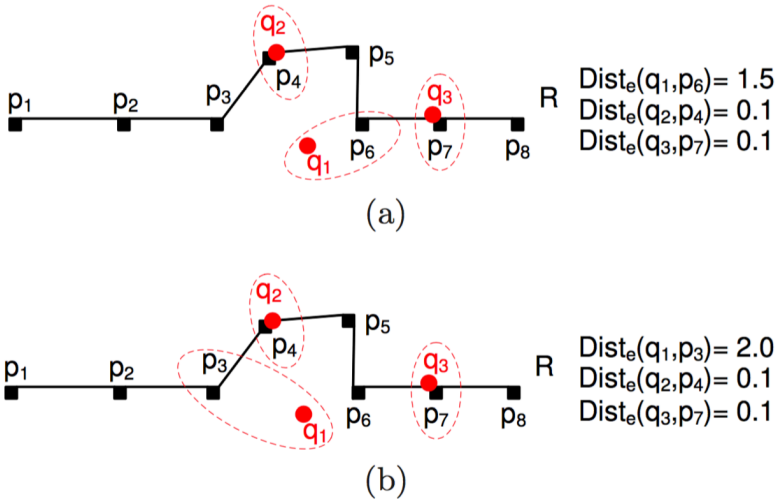

图 3–1 查询点与轨迹点之间的匹配

Fig 3–1 Match between query points and trajectory point

有序查询的相似性计算和无序查询也有区别。给定一组有序的查询点 $Q_o = \{q_{o1}, q_{o2}, q_{o3}, \cdots, q_{on}\}$ 和一条已有轨迹 R，我们通过递归思想为有序查询重新定义相似度方程为 $Sim_o(Q,R)$，式3–3。其中 $Head(x)$ 函数代表 x 中的第一个点，例如 $Head(Q)$ 是查询点 q_1；同时 $Rest(x)$ 表示 x 去掉 x 第一个点之后剩余的部分，例如 $Rest(Q)$ 代表 $\{q_2, q_3, \cdots, q_n\}$。在式3–3中，通过递归的想法，本文将对 $Sim_o(Q,R)$ 的求最大值问题分为对两个子问题的求解，即分别计算 $Sim_o(Rest(Q),R)$ 和 $Sim_o(Q,Rest(R))$ 的最大值问题。当 $Head(Q)$ 和 $Head(R)$ 的两个轨迹点匹配的时候，我们可以将 $e^{-D_e(Head(Q),Head(R))}$ 提前计算并加入当后面计算的 $Sim_o(Rest(Q),R)$ 之中。在这种情况下，$Head(R)$ 需要为下一轮的比较计算继续保留，因为对于 $Rest(Q)$ 中的查询点来说，$Head(R)$ 依旧有可能成为最佳匹配点。而当当 $Head(Q)$ 和 $Head(R)$ 不匹配的时候时我们则跳过 $Head(R)$ 计算 $Sim_o(Q,Rest(R))$。这种求解思路类似于动态规划的思路，这也为我们再后面优化过程中通过动态

规划的来解决这一问题提供了参考。式3–3结合了动态时间规整（DTW）利用重复点和最长公共子序列（LCSS）省略不匹配点的优点来计算相似度方程。

$$Sim_o(Q,R) = max \begin{cases} e^{-D_e(Head(Q),Head(R))} + Sim_o(Rest(Q),R) \\ Sim_o(Q,Rest(R)) \end{cases} \tag{3–3}$$

根据相似度方程，本文可以对 k 最佳连接查询有以下定义 3.1.2.：

定义 3.2.2. 给定一组轨迹集合 $T = R_1, R_2, R_3 \cdots, R_n$、一组查询点 $Q = q_1, q_2, q_3, \cdots, q_n$ 和对应的相似度方程 Sim，k 最佳连接查询则可以从轨迹集合 T 中找到 k 条轨迹 T'，满足式3–4。其中 Sim 根据用户定义选择是否考虑有序性。：

$$Sim(Q,R_i)_{R_i \in T'} \geq Sim(Q,R_j)_{R_j \in T-T'} \tag{3–4}$$

3.3 相似轨迹查询处理过程

3.3.1 算法变量符号定义及解释

表3–1提供了本文本章节之后所涉及的基本符号及其注释。

符号标记	符号注释	符号标记	符号注释
N	一条轨迹的轨迹点总数目	m	一组查询点总数目
$D_e(q_i,p_j)$	点 q_i 和点 p_j 之间的欧氏距离	C	轨迹备选集
$D_e(q_i,p_j)$	点 q_i 和点 p_j 之间的欧氏距离	ϵ	搜索范围阈值
$D_q(q_i,R)$	点 q_i 和轨迹 R 之间的最短距离	ρ	轨迹点密度
r	λ-NeareatNeighbor 搜索半径	ξ	查询点 q_i 对相似度上界贡献
μ, υ	优化搜索权值	UB_{ns}	未在备选集中轨迹的相似度上界
LB	备选集中轨迹的相似度下界		

表 3–1 本文符号列表及其对应注释
Table 3–1 A list of notations and explanations

3.3.2 问题概述

轨迹数据为一组有序点集，轨迹 R 可以被表示为 $R = p_1, p_2, \cdots, p_n$，其中 p_i 是轨迹 R 的在时间顺序上的第 i 个轨迹点。对于本文应用而言，查询点集 Q 被定义为一组点集 $Q = q_1, q_2, \cdots, q_m$，且根据具体情况定义是否有序。在根据上文设计的相似性距离和 k 最佳连接定义后，我们将我们相似轨迹查询任务等价转化为 k 最佳连接查询，并产生下面的定义：

定义 3.3.1. 给定已有的轨迹数据集 D，和一条待查询轨迹 R_q。我们通过轨迹简化算法[19, 21] 将待查询轨迹 R_q 转换成一组查询点集 Q。通过 k 最佳连接查询方法，从轨迹数据集 D 中获取 k 条轨迹，集合为 $D' = R_1, R_2, \cdots, R_k$，满足式3–5。其中 Sim 根据用户定义选择是否考虑有序性。：

$$Sim(Q,R_i)_{R_i \in D'} \geq Sim(Q,R_j)_{R_j \in D-D'} \tag{3–5}$$

我们称 D' 轨迹数据集合为对于轨迹 R_q 的 k 条最相似轨迹查询结果。

3.3.3 相似轨迹查询算法实现

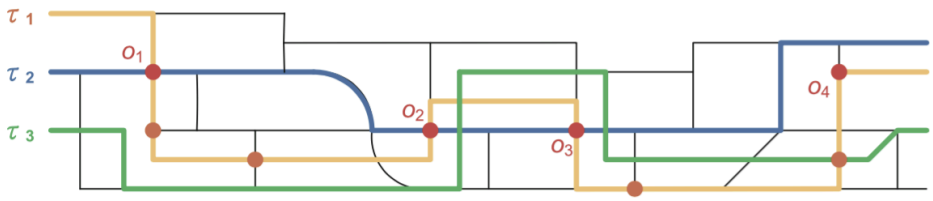

图 3–2 基于位置点的相似轨迹查询
Fig 3–2 Search similar trajectory by location points

如图3–2所示，本文所实现的相似轨迹查询算法主要为针对地理位置点集的轨迹搜索算法[22]。将轨迹数据按特定需求分类索引在 R 树数据结构上后，该相似轨迹查询算法在轨迹点 k 最近邻查询算法[23] 基础上，实现了增长型 k 最近邻查询算法[24]，结合备选和筛选（candidate-refinement）的剪枝思路[25]，通过轨迹相似度上界与下界对搜索空间进行优化处理，以满足空间搜索算法的高效性和准确性。

在这里，轨迹相似度上界与下界所满足的搜索剪枝关系用定理3.1描述

定理 3.1 (相似度上下界)**.** *假设对于相似轨迹插叙的 k 最佳连接算法没有有序性限制，我们可以在对查询点集进行一轮 k 最近邻查询 (k=λ) 之后的轨迹备选集 C 中，选取一个包含 k 条轨迹的一个轨迹子集 C'。当 $\min_{R_x \in C'} LB(R_x) \geq UB_{us}$ 这一条件满足时，我们可以从轨迹备选集 C 中获得 k 条最佳连接轨迹，即 k 条与查询点集最相似的轨迹。*

证明. 首先对于轨迹子集 C' 中的某一条轨迹 R_a（$R_a \in C'$）而言，轨迹 R_a 满足 $Sim(Q, R_a) \geq LB(R_a)$。与此同事，对于轨迹备选集 C 之外的轨迹 R_b（$R_b \notin C$），轨迹 R_b 满足 $UB_{ns} \geq Sim(Q, R_b)$。当上述定理成立时，即 $\min_{R_a \in C'} LB(R_a) \geq UB_{ns}$，我们可以推断出 $\forall R_a \forall R_b \big(R_a \in C' \wedge R_b \notin C\big)$，$Sim(Q, R_a) \geq Sim(Q, R_b)$ 成立。这也证明了对于查询点集 Q 得到的 k 最佳连接的结果轨迹在这个时候应该全部在轨迹备选集 C 中。 □

算法大致流程如图3–3。通过函数主体首先定义初始化几个中间变量。*while* 循环实现增长型 k 最邻近的每一轮查询。查询中，对查询点集 Q 中的每一个查询点进行 k 最近邻方法查询，对查询结果中的每一个轨迹点所在的轨迹都加入轨迹备选集 C。判断轨迹备选集 C 的基数大小是否满足条件。如果满足条件，计算此时归集备选集中所有轨迹的相似度下界大小并用一个数组 $LB[]$ 进行保存，同时计算未在轨迹备选集中的轨迹的相似度上界大小。运用堆排序或优先队列的思想将数组 $LB[]$ 中选取 k 个最大相似度下界及相对应的轨迹。如果3.1满足，即选出来 k 条轨迹中的最小轨迹相似度下界大于等于未在轨迹备选集中的轨迹相似度上界，则说明我们需要查询的 k 条最相似轨迹已经存在于轨迹备选集 C 中，并且其他未扫描到的轨迹可以忽略不予以计算与检查。

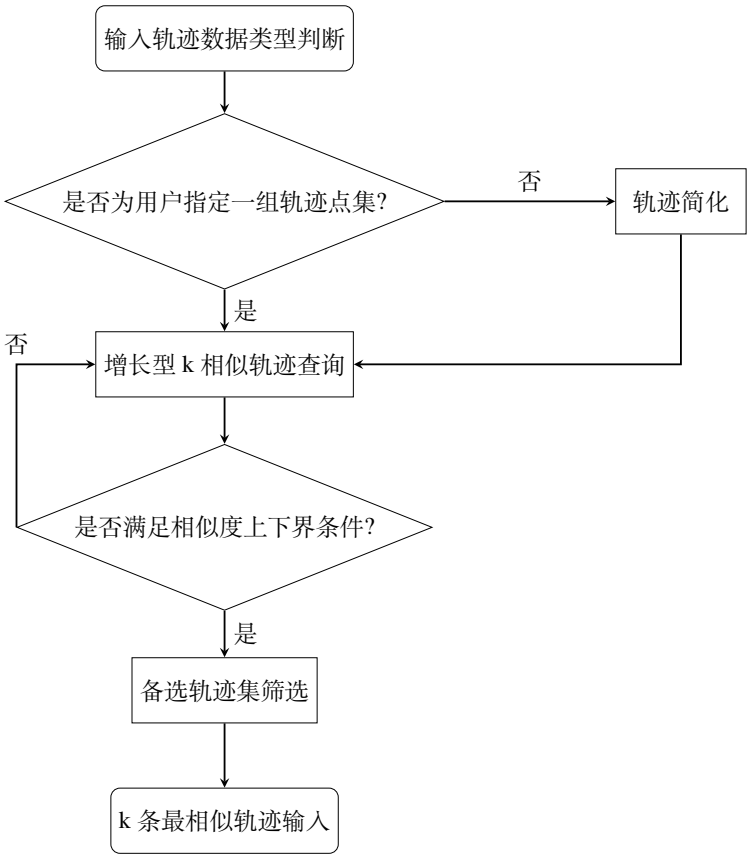

图 3–3 相似轨迹查询算法流程图
Fig 3–3 Searching algorithm flow chart

3.4 算法优化

3.4.1 λ 动态增长优化

在增长型 k 最近邻查询算法中，对于每一次的 k 最近邻查询 λ-NN(q_i) 而言，搜索范围 λ 都是动态增加 $\Delta\lambda$，即每一轮循环中，对于查询点集中的每一个查询点 q_i，搜索范围在数目上是相等的。但值得提出的时，在针对地理位置点进行相似轨迹查询这一上下文中，查询点对于结果的重要性并不是完全一致的。主观而言，有些位置点相对于其他位置点来说具有更重要或更优先的查询级别；从算法角度讨论，每个查询点 q 的结果 λ-NN(q) 对于构建轨迹备选集 C、决定轨迹相似度上下界均有着不同的影响程度。举例来说，对于两个查询点 q_i 和 q_j，在 λ 相同的情况下，如果 $D_e(q_i, p_i^\lambda) > D_e(q_j, p_j^\lambda)$，则对于查询点 q_i 所查找的范围更大，即 $e^{-D_e(q_i,p_i^\lambda)} < e^{D_e(q_j,p_j^\lambda)}$。根据，$UB_{ns} = \sum_{i=1}^{m} e^{-D_e(q_i,p_i^\lambda)}$，我们可以根据结果推出在降低未在备选集中的轨迹相似度上界的过程中，查询点 q_i 比查询点 q_j 效果更好，更有帮助。在定理 3.1.2. 中，未在备选集中的轨迹相似度上界越低，则定理条件越容易满足，即增长型 k 最近邻查询算法可以更快得出结果。我们需要分析 λ 对每个查询点搜索的影响来决定如何动态增加搜索范围。首先我们定义每个查询点 q_i 对于 UB_{ns} 的影响为 $\xi(q_i)$

$$\xi(q_i) = e^{-D_e(q_i, p_i^{\lambda})}$$

显然，当 $\xi(q_i)$ 的值越小时，则相对应的 UB_{ns} 也将越小。接着我们定义 ρ 为某一范围内轨迹点的密度值，定义 $r = D_e(q_i, p_i^{\lambda})$ 为对查询点 q_i 进行 k 最近邻查询时的搜索半径。在 k 最近邻查询这一范围内，我们可以粗略计算出轨迹点的密度值 ρ 等于

$$\rho = \frac{\lambda}{\pi r^2}$$

根据轨迹点密度和搜索半径的关系，我们重写 $\xi(q_i)$ 为

$$\xi(q_i) = e^{-D_e(q_i, p_i^{\lambda})} = e^{-r} = e^{-\sqrt{\frac{\lambda}{\pi\rho}}}$$

在这一步，我们的首要目标是明确 $\xi(q_i)$ 影响因子的下降速度与 λ 之间的关系，根据 λ 的变化所造成的影响赋予查询点 q_1 到 q_m 不同的 $\Delta\lambda$ 变化值，即对于不同的查询点，除了初始第一轮查询之外，之后 $(\lambda + \Delta\lambda)$ 的值都是各自生成的。本文将 $\xi(q_i)$ 为关于 λ 的微分值 $\frac{d\xi}{d\lambda}$ 的绝对值定义为下降速率 $Decay(q_i)$

$$\frac{d\xi}{d\lambda} = \frac{d}{d\lambda} e^{-\sqrt{\frac{\lambda}{\pi\rho}}} = -\frac{1}{2}(\pi\rho\lambda)^{-\frac{1}{2}} * e^{-\sqrt{\frac{\lambda}{\pi\rho}}} \tag{3–6}$$

根据式3–6，我们可以用 λ 和搜索半径 r 来计算轨迹点密度 ρ，因此可以改写下降速率为

$$Decay(q_i) = |\frac{d\xi}{d\lambda}| = \frac{r}{2\lambda} e^{-r} \tag{3–7}$$

根据式3–7，我们可以得知，对于一个固定的 λ 值来说，下降速率 $Decay(q_i)$ 会随着搜索半径 r 的不断增长，先初步上升（$r \in (0, 1]$）后逐渐下降 $r \in (1, \infty)$）。我们可以得知在对于查询结果较为稀疏的查询点（即搜索半径 r 较大）在一开始赋予较大的查询权重值。但随着搜过过程的进行，当搜索半径 r 不断增长达到某一个值得时候，一些相对密集的查询点结果会使得其对应的下降速率变大。这一结论使得我们在搜索和查询过程中重点关注查询点结果较为密集的查询点，这样也能使得我们能更快更有效地在每一轮查询之后降低未在轨迹备选集中轨迹的相似度上界值 UB_{ns}。但随之产生的问题在于，当搜索半径 r 和 λ 都足够大的时候，我们下降 UB_{ns} 会因为 $\frac{d\xi}{d\lambda}$ 趋近于 0 而变得不再有效。

满足定理3.1需要上下界两个变量对条件的同时满足。因此，在关注未在轨迹备选集中轨迹的相似度上界值 UB_{ns} 对增长型 k 最近邻查询的影响时，我们可以在加速增长型 k 最近邻查询算法的时候考虑相似度下界这一因素。当备选集中轨迹的相似度下界 LB 增长越快的时候，定理3.1也就越容易成立。提高相似度下界 LB 所要面对的问题在于，一条轨迹的相似度下界有可能是源于多个查询点所产生查询结果，并且想要预测在搜索过程中什么时候 λ-NN(q_i) 的结果中的某一点和轨迹上的某一点恰好是同一个点也是不太容易的。换言之，我们问题主要在于定量描述每一个查询点对于相似度下界增长的影响。借此，我们基于每一轮重新查找到的新轨迹数目来定义一个启发式搜索的取回速率 $Ratio(q_i)$

$$Ratio(q_i) = \frac{Number(q_i)}{\Delta\lambda} \tag{3–8}$$

式3–8中 $\Delta\lambda$ 为当前循环轮次 λ 的值与上一轮循环中 λ' 值的差值（$\lambda > \lambda'$），而 $Number(q_i)$ 表示在当前循环轮次搜索中获取的轨迹数目多少。基本思想在于，轨迹备选集 C 的基数值范会随着搜索过程中新轨迹数目的增长而增长。在这样的归集备选集 C 中，轨迹相似度下界会曾铮的更快，再根据定3.1，我们也更有可能找到目标寻求的 k 条最相似轨迹。

结合考虑上文所提及的下降速率 $Decay(q_i)$ 和取回速率 $Ratio(q_i)$，我们可以对每一个查询点指定对应的 λ 查询增长值 $\Delta\lambda(q_i)$

$$\Delta\lambda(q_i) = \gamma\big(\alpha\frac{Decay(q_i)}{\sum_{i=1}^{m} Decay(q_i)} + \beta\frac{Ratio(q_i)}{\sum_{i=1}^{m} Ratio(q_i)}\big) \tag{3–9}$$

式3–9中，α 和 β 是本文定义的权值，γ 定义为 $\gamma = mk2^r$ 其中 r 为算法增长型 k 最近邻查询的当前循环轮次数。这样，我们摈弃原先对每一个查询点都增长相同的 λ 值这一处理思路，选择通过式3–9的方法应用于每一个查询点上以对每个查询的进行不同的 λ 增量处理。这样的预先处理会在挖掘出相对重要的轨迹查询点上花费一定时间，但也加速了整个增长型 k 最近邻查询算法的搜索过程。这样的预处理时间由于优化整个算法过程，因此是可接受的。注意到我们在每一轮 λ 增量的总值是

$$\sum_{i=1}^{m}\Delta\lambda(q_i) = \gamma\big(\alpha\frac{\sum_{i=1}^{m} Decay(q_i)}{\sum_{i=1}^{m} Decay(q_i)} + \beta\frac{\sum_{i=1}^{m} Ratio(q_i)}{\sum_{i=1}^{m} Ratio(q_i)}\big) = \gamma(\alpha+\beta)$$

为了保证在每一轮增长型 k 最近邻查询过程中获取的结果轨迹点数据恒定，我们将 $\alpha+\beta$ 设定为 1，其中可以设定 $\alpha = \beta = 0.5$，这样每一轮我们获取的点的数据为 γ

3.4.2 动态规划计算有序轨迹相似度

在前文中我们提及查询的有序性和用户指定有关。在进行有序查询的过程中，之前的算法是基于递归进行实现的：通过去不断匹配轨迹和查询点来进行子递归，从而计算出轨迹和有序查询点集之间的相似度大小。但基于递归相似度计算会占用大量的时间。因此在本上，我们通过动态规划的思路来计算某一条轨迹 R 和查询点集 Q 的相似度，借此来优化算法在有序查询中的处理性能。

假设 $M[i][j]$ 是我们需要解决查询问题的子问题的有序相似度，即 $Sim_{order}(\{q_1, q_2, q_3, \cdots, q_i\}, \{p_1, p_2, p_3, \cdots, p_j\})$。对于动态规划思路而言，当我们获取到 $M[i-1][j]$ 和 $M[i][j-1]$ 的值时，我们可以通过比较 $e^{-D_e(Head(Q),Head(R))} + M[i-1][j]$ 和 $M[i][j-1]$ 的值来决定 $M[i][j]$ 的最大值。如果值 $e^{-D_e(Head(Q),Head(R))} + M[i-1][j]$ 较大，我们可以得出目前的一对匹配点对为 $< p_i, p_j >$，并令 $M[i][j] = e^{-D_e(Head(Q),Head(R))} + M[i-1][j]$，反之，我们略过对 p_j 的目前和之后匹配，并令 $M[i][j] = M[i][j-1]$。这一动态规划的思路自底向上的解决了 $M[i][j]$ 的求值问题，其中 m 为查询点集的基数大小而 n 为轨迹点数目。在算法最后通过范围二维数组中的值来表示查询点集 Q 和轨迹 R 之前有序相似度。算法的复杂性为 $O(mn)$，在具体应用中由于 m 的值相对于 n 来说普遍较小，所以我们可以将算法复杂性近似看成是线性的。

3.5 分布式相似轨迹查询算法实现

在之前的工作描述中，我们对于相似轨迹查询的实现总会提及查询点集的数目相对较少这一前提。对于单机处理而言，如果将一整条轨迹的轨迹数据点作为输入或者查询点集过多，由于增长型 k

最近邻查询会对每一个查询点都进行 λ-NN 的搜索处理，因此整个相似轨迹的查询过程会显得相对缓慢。但有些时候，将一条轨迹作为相似轨迹查询的输入的确更简单且更人性化[26]。在硬件设置对算法性能的约束下，借助基于地理位置点的轨迹简化方法，我们可以对前一章所涉及的相似轨迹查询方法通过加入 *Spark* 分布式集群处理的手段，做到以一条轨迹（或数量更多的查询点）为输入的相似轨迹查询操作。具体实现思路在于，通过基于地理位置点的轨迹简化方法将一条轨迹简化为一组数量相对于单机查询要多的查询点集；然后将查询点通过分布式集群操作分配给各个子节点来进行相对于各集群节点的相似轨迹查询操作，各个子节点通过访问 HDFS 获取轨迹数据和轨迹 R 树索引；最后将结果以轨迹为关键字，以相似度为权值进行求和，选择相似度和最高的 k 条轨迹作为查询结果。

3.5.1 Spark 分布式相似轨迹查询

Spark 集群环境使得我们可以将代码分发给各个工作节点使他们处理对数据集相同的操作，这为分布式搜索相似轨迹提供了实现的基础。单机实现相似轨迹搜索为保证运行性能，给定的输入集查询点需要保证数量在某一程度上相对较小。但对于分布式处理而言，这一约束可以通过集群集计算处理予以取消。给定一条原始轨迹 $Traj$，我们可以先通过基于地理位置点的轨迹简化在保留轨迹形状和轨迹中的重要位置点的同时，一定程度上减少轨迹数目。事实上，如果集群设备性能较好，我们可以略去集群计算相似轨迹前对轨迹简化这一步骤。由于本文实验设备限制，通过在集群处理前的轨迹简化能在保证结果正确性的过程中，稳定处理性能。因此，我们将轨迹简化作为集群计算前的预处理过程。

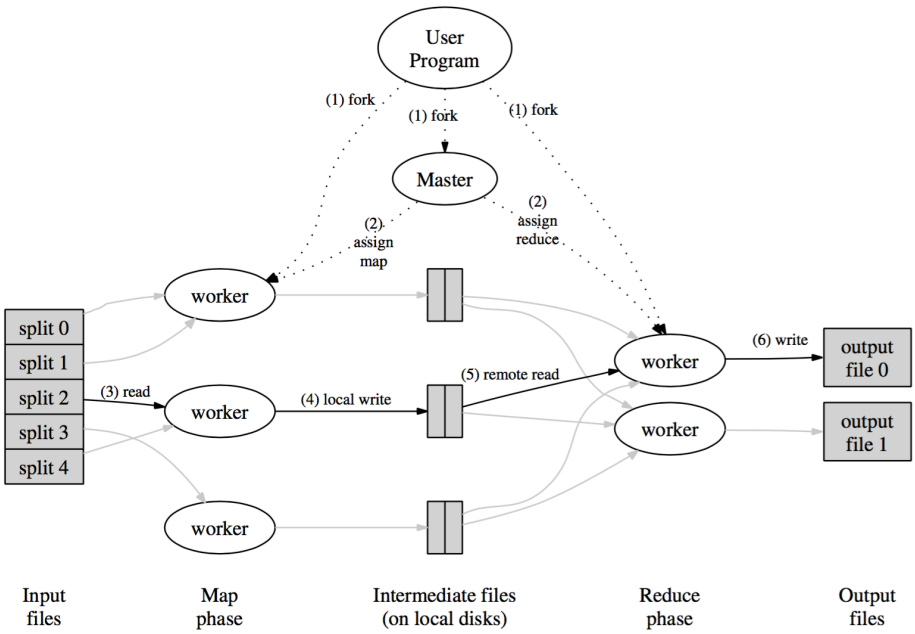

图 3–4 Spark 分布式相似轨迹查询

Fig 3–4 Spark distributed search similar trajectories

图3–4为 *MapReduce* 的大致处理框架，我们根据这一框架设计出分布式查询相似轨迹的大致算法。在算法中，假设我们已经完成对原始轨迹的简化得到已处理好的简化轨迹 $Traj'$[21]。首先我们通过 *Spark* 内的 *partition* 函数将输入数据分割，并分发给每一个工作节点（*Worker Node*）。对于每一个工作节点而言，他们都有了输入数据的一部分轨迹点。对于每一个工作节点而言，他们都可以将自己所拥有的部分轨迹点作为相似轨迹查询的查询点集，由于工作节点通过设置可以直接访问 *HDFS* 上已存储的轨迹数据，我们通过主节点（*Master Node*）将增长型 k 最近邻查询代码分发给每一个工作节点并予以上运行。每个工作节点将针对各自输入查询点集所得出的 k 条最相似轨迹及其对应的相似度大小作为输出。通过对轨迹作为关键字合并中间查询结果，对相似度求和按相似度大小降序排列轨迹，再从中选取 k 条轨迹作为最终的结果。

实现细节如算法3–1所示。其中在运行过程中初始化 *Spark* 运行所需的上下文变量并设置主节点信息。在 *map* 阶段对查询点集进行查询搜索，在 *reduce* 阶段进行关键字合并。

算法 3–1 分布式相似轨迹查询算法

输入: 相似轨迹查询数目 k, 一条原始轨迹数据 $Traj$

输出: k 条最相似轨迹 $k\text{-}Trajs$

```
Traj' ← TS(Traj);        //简化轨迹
Initialise Spark Context sc and set Master information;
RDD ← sc.parallelize(Traj', partitionNumber);
k-Trajs ← RDD.map(iknn)        //工作节点分布式进行查询处理
        .flatMap(lambda x:x)        //查询结果平铺成为一维列表或数组
        .reduceByKey(lambda x,y:x+y)        //以轨迹为关键字做相似度求和
        .sortBy(lambda x:x[1], descending)        //降序排列
        .collect();
return k-Trajs;
```

3.5.2 多请求分布式相似轨迹查询

从工业应用角度，多用户同时进行相似轨迹处理时，可以针对对多请求的分布式相似轨迹查询处理。将请求数据集根据 *Spark* 环境默认的分割方法或自定义的分割方法，将多请求分割成多个子集，然后分配个集群中的子节点，让他们各自处理处理一部分请求。在实际生活中，作为一个交付使用的应用，在某个时段可能有多个用户请求发送给服务器端，在配置有集群环境的情况下，可以实现对请求的分布式处理，具体思路类似于上文分布式相似轨迹查询，不予以赘述。

3.6 本章小结

本章节在相关工作的基础上，定义了一些轨迹相关的符号变量，并基于这些符号变量和已有的相关工作，从问题的本质出发，设计了基于 k 最近邻查询的增长型 k 最近邻算法。通过将 k 最佳连接查询结果等价视作为 k 最相似轨迹查询结果，实现相似轨迹查询。在实现过程中，通过对搜索范围的剪枝优化、对搜索参数的动态设定和对查询有序性无序性的特殊处理，结合备选和筛选的处理

思路，将相似轨迹搜索过程的处理性能实现在用户可接受范围内。

另一方面，随着如今 GPS 技术的发展和车载数据的激增，轨迹数据挖掘的数据规模通常较大，因此在处理轨迹数据过程中，单机本地操作或许无法满足轨迹挖掘的特定需求。因此，我们可以通过分布式集群处理的方法来完成大数据轨迹数据挖掘。分布式如今发展较为成熟，为我们提供了许多适应于不同应用的大数据分布式处理框架。根据 *MapReduce* 的编程模型，我们自定义对输入、中间过程和输出的处理方法，借助分布式框架 *Spark* 提供的接口方法能够较好的完成本文的相似轨迹查询在分布式上的实现。

第四章　相似轨迹查询系统设计

4.1　本章序言

随着相似轨迹查询这以在科研领域和工业应用中的不断发展，开发相似轨迹查询系统为用户在搜索相似轨迹中提供更方便更可视化的结果。为了明确关于相似轨迹查询系统的设计需求、应用领域与系统功能，在本章节中，本文首先对相似轨迹查询系统的设计与实现作细节说明和具体描述。我们对本章节的目的以及之后说明中会涉及到的名词概念进行解释，以便于之后章节的拓展。该系统是主要以 Python 进行后端数据处理，系统运行于以 Python 为基础的 Flask Web 内部依赖 Werkzeurg 的工作集服务器。用户只需要通过进行简单操作便可以实现个性化的相似轨迹查询操作，并了解本系统软件的基本工作原理。

4.1.1　系统应用范围

相似轨迹查询系统是一个基于 *GPS* 轨迹数据的服务器端应用系统。根据轨迹用户提供的输入轨迹或一组基于 GPS 数据的轨迹点集，系统帮助用户在地理语义上查询出与输入最相似的 k 条结果轨迹。系统可以免费从 Github 上下载并运行在可以运行 Python 程序的平台上。

本系统在应用中，需要连接互联网以调用基于 Javascript 的地图服务接口，实现地图显示和轨迹显示等等基础功能。所有系统所需要的轨迹数据以文件系统存储于服务器端。在系统运行过程中，用户可以通过浏览器登录页面进行系统访问，结合用户自身查询需求系统得出不同的查询结果。本系统也支持展示出所有地图上具体位置信息和查询具体地理位置的功能。

4.1.2　系统名词定义

表4–1所示。

符号标记	符号注释
用户	使用相似轨迹查询系统进行轨迹查询人员
管理员	相似轨迹查询系统管理员，负责运行权限和控制系统
轨迹数据	移动物体上空间上按时间顺序的移动数据记录
GPS	全球定位系统
DESC	描述
DEP	依赖

表 4–1 相似轨迹查询系统涉及名词定义

Table 4–1 A list of definitions and explanations in the system of searching similar trajectories

4.2 大体描述

本文设计的相似轨迹查询系统运行运行于基于 Python 语言的 Flask Web 框架，运行过程中需要连接互联网以使用地图接口，系统内部数据为上海私家车数据。系统针对的用户为一般用户和登录用户。一般用户可以目的轨迹点集进行相似轨迹查询，实现路径规划等具体应用，用户在运用过程中需要在地图上点击位置以实现轨迹点击输入；登录用户在一般用户具有功能基础上，用户的历史轨迹在轨迹数据库中有存储，可以实现以一条轨迹为输入的相似轨迹查询，实现拼车或轨迹推荐等具体应用。本系统目前已有依赖于 Flask 框架和地图接口提供。

4.2.1 系统设计框架

相似轨迹查询系统主要由前后端两部分组成。后端部分主要以轨迹预处理、相似轨迹查询和查询请求分析处理为主；前端以部分地图轨迹可视化、查询结果输出、地理位置搜索等部分组成。

针对相似轨迹查询系统的后端处理流程思路主要为，在后台运行服务器代码，接收来自浏览器前端的请求。将请求分析与提前预先设定的资源定位符函数结合，对于数据类型选择是否进行轨迹数据预处理。针对同一的输入轨迹数据点集，进行相似轨迹数据查询功能服务。然后将结果作为请求返回返回给用户。实现一次相似轨迹查询请求过程。与此同时，系统前端主要以实现数据可视化为主，以一个窗口形式将地图数据显示。对于用户感兴趣的地理位置坐标点和历史轨迹数据，可以通过点击地图或历史轨迹数据项目将数据在地图窗口上得以可视化展示。

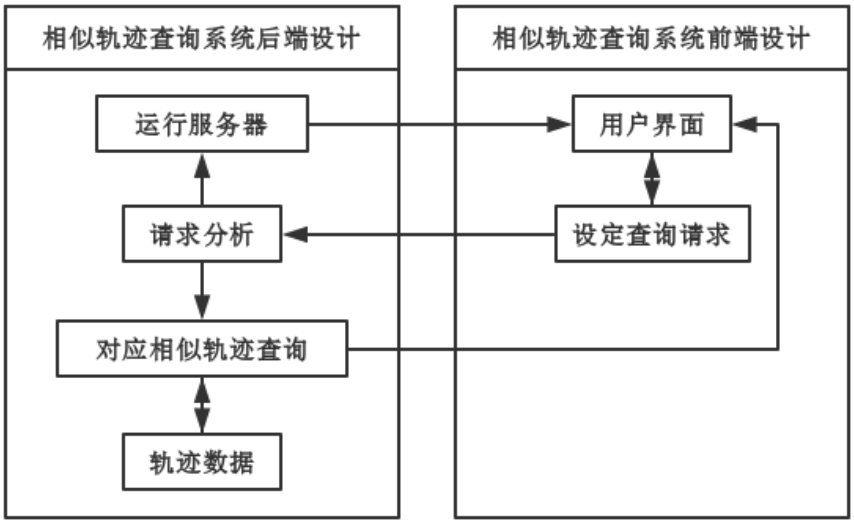

图 4–1 相似轨迹查询系统设计框架

Fig 4–1 System design architecture

4.2.2 系统功能描述

在相似轨迹查询系统中，一般用户和登录用户都可以使用相似轨迹查询功能。查询结果与用户定义的输入轨迹数据类型（为轨迹点集或一条历史轨迹）、查询的日期、相似轨迹数目阈值和查询是否有序性有关。

相似轨迹查询的主要是通过在地图上以不同颜色和不同粗细的折线端进行表示。在查询结果中，根据与查询输入相似度的相关性，越相似的查询结果轨迹将以越显著的参数表示出来，用户可以一目了然在地图上得出最符合查询的结果的一条或多条轨迹。

4.2.3 系统限制描述

4.2.4 用户类与特点

4.3 系统具体需求分析

4.3.1 系统需求说明目的

在完成对相关工作的研究和系统市场的前景分析后，本文提出相似轨迹查询系统设计需求说明。相似轨迹查询查询系统设计需求说明的目的在于对于本系统进行具体描述，明确索要开发的系统应该具备的功能与界面，为系统分析及移植开发提供清晰的基础需求描述，并以此为基础进一步满足后续设计与开发。本文所设计开发的相似轨迹查询系统目标是为用户提供高效、准确的查询服务平台。系统针对目前轨迹信息管理的实际情景，较为全面地满足用户的查询需求，初步实现系统所制定的设计初衷。需求说明从字面上作为软件系统发展的指导和完备部分，解释说明了系统中的限制条件、应用交互接口以及具体功能。

4.3.2 用户界面需求

4.3.3 用户功能需求

第五章　结论

5.1　设计思路总结

本文在相似轨迹查询设计和实现之前，先查阅了相关资料。了解并学习相关与轨迹数据挖掘的基础知识后，开始初步设计出与传统相似轨迹查询不同的相似轨迹处理算法。在问题本质上，我们研究并实现一种通过一组有序或无序的轨迹点从轨迹数据库中查询 k 条最佳连接这些轨迹点的轨迹，从地理语义上而言，我们可以将这 k 条最佳连接轨迹等价看我们目标查找的 k 条最相似轨迹。这类相似轨迹查询不同于之前已有的工作，它能在轨迹推荐、交通路况分析和生物站点分配等等中有着广泛的应用空间。在实际情景和本文上下文中，我们都限定这一组查询点集的数目相对较少，这一限定条件使得我们可以利用一些空间索引和查询方法来实现我们的搜索过程。轨迹简化这一预处理技术在本文实现过程中起着很重要作用，我们可以即利用轨迹简化技术来减少我们对设备存储空间的需求，也可以在查询相似轨迹的过程中将一条轨迹做为输入，后者便于用户用轨迹输入取代繁琐的查询点集。

本文设计出增长型 k 最近邻相似轨迹查询方法。在定义适合的轨迹相似度方程之后，主体算法基于 k 最近邻分类查询算法，结合备选和筛选的处理思路来进行查询。在 k 最近邻查询中，使用便于空间搜索的 R 树数据结构来完成备选过程处理；定义轨迹相似度的上界与下界，在循环过程中以上下界为剪枝条件来不断筛选出符合条件的 k 条相似轨迹。在优化过程中，根据地理位置点语义上重要性的不同，因地制宜动态设定查询范围，以更快速地完成查询过程。在初步了解分布式集群处理理念后，本文将相似轨迹查询应用与分布式处理框架相结合，可以将输入范围扩大至整条轨迹的同时以保证搜索数据，借助 *MapReduce* 的编程处理框架和已有的分布式存储技术，本文也保证了更准确的查询结果。

我们通过现实生活中采集的 GPS 轨迹数据作为数据集，对本文设计出的相似轨迹查询方法做算法评价。**(等写完实验再来写)**

5.2　未来工作展望

轨迹数据挖掘的系统框架在绪论章节有大致的讨论，在本文所设计的相似轨迹查询算法设计中，仅仅通过轨迹简化预处理和轨迹查询完成了初步任务。在实际应用中，对轨迹数据点的修正、降噪预处理、选择更合理的轨迹索引结构以及在相似轨迹查询中辅以轨迹模式挖掘与轨迹分类都能够在一定程度上对轨迹准确性或是算法性能有提高。

其次对于分布式框架而言，本文所设计的分布式操作仅借助简单的操作，以完成初步的查询任务。在时间允许的情况下可以从 *Spark* 提供的丰富的弹性分布式数据集操作接口去更合理的安排数据处理流程和方式。相对于本文通过单一设备搭建本地集群环境而言，将工作环境移植备性能更高的集群环境并且按照节点数目和数据大小设定自定义集群处理环境也能够大幅度提高。

参考文献

[1] ZHENG Y. Trajectory data mining: an overview[J]. ACM Transactions on Intelligent Systems and Technology (TIST), 2015, 6(3): 29.

[2] ZHENG Y, ZHOU X. Computing with spatial trajectories[M]. [S.l.]: Springer Science & Business Media, 2011.

[3] AGRAWAL R, FALOUTSOS C, SWAMI A. Efficient similarity search in sequence databases[J]. Foundations of data organization and algorithms, 1993: 69–84.

[4] CHAN K.-P, FU A W.-C. Efficient time series matching by wavelets[C]// Data Engineering, 1999. Proceedings., 15th International Conference on. IEEE. [S.l.]: [s.n.], 1999: 126–133.

[5] BERNAD D. Finding patterns in time series: a dynamic programming approach[J]. Advances in knowledge discovery and data mining, 1996.

[6] YANAGISAWA Y, AKAHANI J.-I, SATOH T. Shape-based similarity query for trajectory of mobile objects[C]// International Conference on Mobile Data Management. Springer. [S.l.]: [s.n.], 2003: 63–77.

[7] CAI Y, NG R. Indexing spatio-temporal trajectories with Chebyshev polynomials[C]// Proceedings of the 2004 ACM SIGMOD international conference on Management of data. ACM. [S.l.]: [s.n.], 2004: 599–610.

[8] VLACHOS M, GUNOPULOS D, DAS G. Rotation invariant distance measures for trajectories[C]// Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining. ACM. [S.l.]: [s.n.], 2004: 707–712.

[9] SAKURAI Y, YOSHIKAWA M, FALOUTSOS C. FTW: fast similarity search under the time warping distance[C]// Proceedings of the twenty-fourth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems. ACM. [S.l.]: [s.n.], 2005: 326–337.

[10] VLACHOS M, KOLLIOS G, GUNOPULOS D. Discovering similar multidimensional trajectories[C]// Data Engineering, 2002. Proceedings. 18th International Conference on. IEEE. [S.l.]: [s.n.], 2002: 673–684.

[11] CHEN L, ÖZSU M T, ORIA V. Robust and fast similarity search for moving object trajectories[C]// Proceedings of the 2005 ACM SIGMOD international conference on Management of data. ACM. [S.l.]: [s.n.], 2005: 491–502.

[12] CHEN L, NG R. On the marriage of lp-norms and edit distance[C]// Proceedings of the Thirtieth international conference on Very large data bases-Volume 30. VLDB Endowment. [S.l.]: [s.n.], 2004: 792–803.

[13] FRENTZOS E, GRATSIAS K, THEODORIDIS Y. Index-based most similar trajectory search[C]// Data Engineering, 2007. ICDE 2007. IEEE 23rd International Conference on. IEEE. [S.l.]: [s.n.], 2007: 816–825.

[14] SANKARARAMAN S, AGARWAL P K, MØLHAVE T, et al. Model-driven matching and segmentation of trajectories[C]// Proceedings of the 21st ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems. ACM. [S.l.]: [s.n.], 2013: 234–243.

[15] MORSE M D, PATEL J M. An efficient and accurate method for evaluating time series similarity[C]// Proceedings of the 2007 ACM SIGMOD international conference on Management of data. ACM. [S.l.]: [s.n.], 2007: 569–580.

[16] Jinja2 and WSGI. Website. `http://werkzeug.pocoo.org/docs/0.12/` `http://jinja.pocoo.org/docs/2.9/`.

[17] DEAN J, GHEMAWAT S. MapReduce: simplified data processing on large clusters[J]. Communications of the ACM, 2008, 51(1): 107–113.

[18] ZAHARIA M, CHOWDHURY M, FRANKLIN M J, et al. Spark: Cluster Computing with Working Sets.[J]. HotCloud, 2010, 10(10-10): 95.

[19] VISVALINGAM M, WHYATT J D. The Douglas-Peucker Algorithm for Line Simplification: Re-evaluation through Visualization[C]// Computer Graphics Forum. Vol. 9. 3. Wiley Online Library. [S.l.]: [s.n.], 1990: 213–225.

[20] SELLIS T, ROUSSOPOULOS N, FALOUTSOS C. The R+-Tree: A Dynamic Index for Multi-Dimensional Objects.[R]. 1987.

[21] CHEN Y, JIANG K, ZHENG Y, et al. Trajectory simplification method for location-based social networking services[C]// Proceedings of the 2009 International Workshop on Location Based Social Networks. ACM. [S.l.]: [s.n.], 2009: 33–40.

[22] QI S, BOUROS P, SACHARIDIS D, et al. Efficient point-based trajectory search[C]// International Symposium on Spatial and Temporal Databases. Springer. [S.l.]: [s.n.], 2015: 179–196.

[23] ROUSSOPOULOS N, KELLEY S, VINCENT F. Nearest neighbor queries[C]// ACM sigmod record. Vol. 24. 2. ACM. [S.l.]: [s.n.], 1995: 71–79.

[24] CHEN Z, SHEN H T, ZHOU X, et al. Searching trajectories by locations: an efficiency study[C]// Proceedings of the 2010 ACM SIGMOD International Conference on Management of data. ACM. [S.l.]: [s.n.], 2010: 255–266.

[25] TANG L.-A, ZHENG Y, XIE X, et al. Retrieving k-nearest neighboring trajectories by a set of point locations[C]// International Symposium on Spatial and Temporal Databases. Springer. [S.l.]: [s.n.], 2011: 223–241.

[26] SHANG S, DING R, ZHENG K, et al. Personalized trajectory matching in spatial networks[J]. The VLDB Journal, 2014, 23(3): 449–468.

致　谢

本科四年级下学期几个月的时间内，我从毕业设计的论题提交到现在最终完成这一份毕业论文，从一开始对轨迹数据挖掘领域毫无涉及不知道自己该从什么角度下手，到现在能够大概了解轨迹数据挖掘的系统框架并实践自己在相似轨迹数据查询这一领域的初步应用，努力所带来的收获远超过这一篇毕业论文所提及的内容。在此期间，十分感觉在我奋斗过程中不断鼓励我指导我的导师，为我提供各种帮助和建议的同学。

感谢我毕业设计的导师朱燕民老师，在我完成自己毕业设计开始会为我提供入手的方向和，不厌其烦地在我陷入不知所措的时候提供最及时的指导。朱燕民老师每周对毕业设计进度的关心让我在完成毕业设计的过程中脚踏实地，一步一步完成自己的既定目标，并在每周的汇报过程中基于我当前阶段最中肯的建议，让我在这一过程中时时刻刻都充满着前进的动力。

感谢徐亚楠学长、张博文学长和黄凯欣学长在我实现相似轨迹查询和学习分布式计算过程中给予的帮助和指导，让我能够从正确的方向了解并学习相关领域的知识。感谢朱灏和吴思禹两位同学在实现毕业设计中提供的帮助与建议。

最后感谢上海交通大学对我本科四年的培养，让我从四年的本科教育学习中逐渐入门计算机这一广大的知识领域，明晰了自己未来发展的方向，让我在有限的时间里为未来的求学生涯与工作领域打下了扎实的知识基础。

DESIGN AND IMPLEMENTATION OF THE QUERY METHOD OF SEARCHING SIMILAR TRAJECTORIES

Trajectories are the traveling history of moving objects such as a person, a vehicle or an animal. The ever-evolving development in location acquisition and the real-time GPS technology have generated massive spatial trajectory data and can be used for complex analysis across diverse study fields. In this context, a lot of techniques involving mining trajectory data have been fostering a extensive range of applications. The utility of trajectory data depends largely on the efficient and effective trajectory query and search processing in trajectory databases. In fact, trajectory search is aimed to evaluate the spatiotemporal relationships among spatial data objects. Among them, similar trajectory search has long been an attractive and hot topic which helps in various applications in spatial trajectory databases.

In essence, We design a new type of searching similar trajectory, differing from the conventional ones that finds similar trajectories to a specified one, in which context the query input is only a small number of location points with or without the restraint of order. The target of our searching is substantially to find the k best connected trajectory to the query points, and we consider these result trajectory similar to the query points in the shape geographically. We lay the emphasis more on how well a trajectory connects these query points for the reason that user may focus more on some specific locations when looking for similar trajectory. The basic idea of designing the algorithm is candidate and refinement, which gathers all the potentially similar trajectory as a candidate trajectory set and then preserves k the most similar trajectory as the result to be expected. his novel search type is helpful in many trajectory-related applications such as route planning and trajectory recommendation.

In this paper, we study and fulfil this type of search by the help of k best connected trajectory query. In order to search for the similar trajectory from a large-scale trajectory databases, we are supposed to first define the appropriate similarity function to score how well a trajectory connects the query points. The Eulerian distance function serves well in time of measuring the contribution of each matched points to the similarity. Using the exponential function assists us assign a larger contribute to a closer matched pair of points while lower the dedication of a unmatched or dissimilar ones.

Next, to answer the k best connected query for search similar trajectories. We should first make the full use of the practical observation that the number of query locations tends to be relatively small in general, because users' focuses are impossibly shared to each one of the trajectory points. In this way, we decide to search the k closest trajectories for each query points one by one with the help of existing spatial trajectory indexing and retrieval techniques, and then merge the intermediate query results of each points to generate the final k most similar trajectories. The cost of this method is theoretically considered as the cost of search over the trajectory indexing structure multiplying the number of query points. The latter one can be relatively

small upon the above observation. So the cost greatly depends on the indexing structure we adopt. Here, referring some related word, we utilise the commonly used R-tree index while searching the matched points from similar trajectories. After we index the points of all the trajectories from the database by a single R-tree, we can acquire the closest trajectory with regard to a specific query points. This is because if we get the nearest points to a query point, the trajectory that contains this nearest point must be the potential target trajectory to the query points.

Indexing the points in a single R-tree helps us efficiently find the closest trajectory points with regard to a query location. The searching method we adopt is k-NearestNeighbor algorithm. However, this method only cannot handle the searching task because of the incompleteness of single search and process. Therefore, we extend the k-NearestNeighbor to the one with incremental search range, named incremental k-NearestNeighbor, which search the R-tree index continually until the expected similar trajectory are found. The crucial problem in this method is how to prune the unqualified candidate trajectory, refine the possible set of temporary found trajectory and speed up the process of incremental k-NearestNeighbor. In order to handle this key point, we follow to define the lower bound of similarity of the potential trajectories and the upper bound of similarity of unqualified trajectories. By comparing these two similarity, we are able to decide whether the k expected similar trajectories have been contained in the candidate during the searching process, which obviously could tell us when and where to terminate the costly search procedure.

By using lower bound and upper bound, we have a chance to establish a reasonable pruning strategy for the similar trajectory searching to avoid searching the whole trajectory databases. At this time, one critical question is what search range we should select in the process of searching to promise that the expected k most similar trajectory are found in the candidate set. The dilemma of how to set the search range is that, on one hand, if we set a large value to the search range, it could probably acquire the complete candidate trajectory set at the cost of a huge search space for each query points; on the other hand, the small search range may give rise to the incompleteness of the candidate set that not all the potential trajectories are included in the set and then results in the false dismissal. Given the two bounds mentioned above, we determine to dynamically adjust the search range for the query points respectively, rather than choose the fixed increment for every search loop. With this thought, we initially choose a positive search range for all the query points. However, for each point, if the generated candidate trajectory set cannot satisfy the theorem that determines whether the process is supposed to be stopped, we increase the search region given the search point's weight among the set of query points respectively. Furthermore, taking the decay rate and retrieval ratio as supplemental factors of the query point, we can formulate the proper way to regulate the search range for each point.

The input of search similar trajectory so far is a set of query points. Considering the users' demand, a single trajectory as the input tends to provide a better interactive experience between users and applications. Thus, we combines the current design with trajectory simplification and distributed computing technologies to fulfil the searching of similar trajectory given a specific trajectory in the real world. The input format of a trajectory does not change that the essence of query is the set of query points. But we should choose some points among trajectory points to weigh enough to represent the whole trajectory first. The trajectory simplification based on the locations can handle this concern smoothly by segmenting a trajectory, computing

the weight of each point and then choose the number of points important enough to represent a complete trajectory. The simplified result could be relatively large compared to the incremental k-NearestNeighbor algorithm. Now, this paper adopt the distributed computing architecture in the design and implementation. We partition the simplified result to each worker node in the cluster environment and extend the MapReduce programming method. Each worker node can access the trajectory data by Hadoop Distributed File System and then do the search job individually in the map stage as if working in a single machine. The individual results are seen as intermediate and then we merge these results in the reduce stage to select the k most similar trajectory as the final result. The trajectory simplification technique and distributed computing help us extend our method in searching the similar trajectory not only by a set of query points, but also a specific trajectory. Besides, the distributed computing idea can contribute to the multiple requests from many users at the same time, as it can partition the request set and send each of them to one of the worker node.