

上海交通大学

SHANGHAI JIAO TONG UNIVERSITY

学士学位论文

BACHELOR'S THESIS



论文题目 相似轨迹查询方法设计与实现

学生姓名 戚文韬
学生学号 5130309593
指导教师 朱燕民教授
专业 计算机科学与技术专业
学院(系) 计算机科学与工程系

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 最佳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系统查询到的相似轨迹。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、拼车出行等等基于地理位置的应用中，这种基于查询点的相似轨迹查询都能发挥作用。再根据相似轨迹查询算法设计出符合实际用户需求的相似轨迹查询系统，从需求分析入手，进行系统功能设计并进行系统测试。

根据系统测试，基于 k 最佳连接查询算法设计的系统能够在 0.2 秒至 1 秒内给出相似度准确性在 97.2% 以上的无序相似轨迹查询结果和 72.5% 以上有序相似轨迹查询结果。

关键词： 相似轨迹查询 k 最近邻 算法实现 优化 系统设计

Design and Implementation of An Efficient Similar Trajectory Search Algorithm

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 from 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 general-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 extend 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 cost 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. Based on the proposed algorithm related to searching similar trajectories, we develop a system for searching similar trajectories which meets the practical requirements among the trajectories users. Starting from the system requirement analysis, we design the module part of the whole system respectively and make a test for this system.

The test of this system based on the k best connected trajectories tells that the system can give the result within 0.2 to 1 seconds with the accuracy of 97.2% for searching without order restraint and the accuracy of 72.5% with order restraint.

KEY WORDS: Similar trajectory search, k-NearestNeighbor, Algorithm implementation, Optimization, System Design

目 录

插图索引	vi
表格索引	vii
算法索引	viii
第一章 绪论	1
1.1 大数据发展概述	1
1.2 轨迹数据处理现有工作的概述及评价	1
1.2.1 轨迹查询简介	2
1.2.2 相似轨迹查询应用现状	3
1.2.3 相似轨迹查询技术现状概述	3
1.3 相似轨迹查询问题描述	3
1.3.1 问题大致描述	3
1.3.2 相似轨迹查询方法设计概述	3
1.4 相似轨迹查询应用价值	5
1.5 困难和挑战	5
1.6 论文结构	5
1.7 本章小结	5
第二章 相关工作	6
2.1 国内外研究现状	6
2.1.1 相似轨迹查询方法	6
2.1.2 相似度方程比较	7
2.2 相关实现技术介绍	8
2.2.1 Flask 应用框架	8
2.2.2 Bootstrap	9
2.2.3 大规模数据集群处理	9
2.2.4 轨迹数据预处理	11
2.2.5 轨迹数据索引与获取	12
2.3 本章小结	13
第三章 相似轨迹查询方法实现	14
3.1 相似轨迹查询问题描述	14
3.2 k 最佳连接定义	14

3.3	相似轨迹查询处理过程	17
3.3.1	算法变量符号定义及解释	17
3.3.2	相似轨迹查询问题概述	17
3.3.3	相似轨迹查询算法实现	18
3.4	分布式相似轨迹查询算法实现	18
3.4.1	Spark 分布式相似轨迹查询	19
3.4.2	多请求分布式相似轨迹查询	20
3.5	本章小结	21
第四章 相似轨迹查询系统需求分析		22
4.1	本章序言	22
4.2	需求分析概述	22
4.2.1	系统需求背景	22
4.2.2	系统需求说明目的	22
4.2.3	系统应用定位分析	22
4.2.4	系统应用范围	23
4.2.5	系统可行性分析	23
4.3	功能需求	24
4.3.1	用户功能需求	24
4.3.2	用户界面需求	24
4.4	性能需求	25
4.4.1	系统查询准确率	25
4.4.2	系统查询时间特性	25
4.4.3	系统适应性	26
4.5	本章小结	26
第五章 相似轨迹查询系统设计与实现		27
5.1	本章序言	27
5.1.1	系统设计编写背景	27
5.1.2	系统设计编写原则	27
5.1.3	系统名词定义	27
5.1.4	用户类与特点	27
5.1.5	系统运行环境	28
5.2	系统设计	28
5.2.1	系统设计分析	28
5.2.2	系统功能描述	29
5.2.3	系统设计框架	29
5.2.4	系统处理流程	30
5.3	接口设计	30

5.3.1 外部接口	30
5.3.2 内部接口	31
5.3.3 用户接口	31
5.4 系统总体结构	32
5.4.1 系统组成模块	32
5.4.2 系统功能模块详细设计	32
5.5 系统用户界面设计	33
5.6 本章小结	33
第六章 系统测试与评价	35
6.1 引言	35
6.1.1 编写目的	35
6.1.2 针对用户	35
6.1.3 缺陷定义	35
6.1.4 测试环境数据	36
6.2 系统测试概要	37
6.3 系统测试描述	37
6.3.1 功能测试描述	37
6.3.2 性能测试描述	37
6.4 系统测评结果	37
6.5 系统测试评价	39
6.6 本章小结	40
第七章 结论	41
7.1 相似轨迹查询系统设计总结	41
7.2 未来工作展望	41
参考文献	42
致 谢	44

插图索引

1-1 轨迹数据挖掘范例	2
1-2 基于位置点的相似轨迹查询	4
2-1 已有轨迹相似度函数定义	8
2-2 集群管理大致模式	10
2-3 Spark Standalone 集群结构	11
2-4 R 树数据结构举例	12
3-1 查询点与轨迹点之间的匹配	15
3-2 基于位置点的相似轨迹查询	18
3-3 相似轨迹查询算法流程图	19
3-4 Spark 分布式相似轨迹查询流程	20
4-1 相似轨迹查询系统可行性	23
4-2 相似轨迹查询系统用户功能需求	24
4-3 相似轨迹查询用户界面需求	25
4-4 系统查询结果准确性	25
5-1 相似轨迹查询系统设计框架	29
5-2 相似轨迹查询系统处理流程图	30
5-3 相似轨迹查询系统数据层次	31
5-4 相似轨迹查询系统模块组成	32
5-5 用户功能窗口	34
5-6 用户地图界面设计	34
6-1 上海市私家车轨迹数据样例	36
6-2 相似轨迹查询数目 k 和查询时间的关系	38
6-3 相似轨迹查询点数目和查询时间的关系	38
6-4 相似轨迹查询数目 k 和查询准确度的关系	39
6-5 相似轨迹查询点数目和查询准确度的关系	39

表格索引

3-1 公式符号列表及其对应注释	17
5-1 相似轨迹查询系统涉及名词定义	28
5-2 相似轨迹查询系统运行环境	28
6-1 系统测试选取轨迹数据格式	36
6-2 分布式测试与单机测试比较	40

算法索引

3-1 有序相似度算法 dp_Similarity(Q,R)	16
3-2 分布式相似轨迹查询算法	21

第一章 绪论

1.1 大数据发展概述

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。

大数据的定义以类型多、容量大、存取快、价值高为主的数据集合。现在世界上运用大数据推动经济发展、提升政府服务和提高用户体验正在成为一种主流的趋势，发达国家和发展中国家都在制定着与本国发展相符的大数据战略性文件以推动大数据在各个领域的发展和应用。目前，我国互联网和移动互联网的用户数目庞大，拥有丰富且系统的饿数据资源储备与广大应用市场优势。坚持大数据的驱动发展和部署范围，并拓展深化大数据的实际应用，已经成为稳增长、调结构、促改革、惠民生和推动政府治理能力现代化的内在需要和必然选择。

在大众创业和万众穿行的创新驱动新格局，充分利用已有的大数据红利并促进激发大众创业的活力是目前利用大数据发展的总体目标。政府在这一方面也鼓励从业与各个行业的人员利用已有开放共享的大数据资源，进行资源整合与分类，提升大数据的治理能力。在这一基础上推动相关产业一起创新发展，利用类型不同、数量众多的大数据资源来培育出新兴的产业领域，助力我国在目前阶段的经济转型。与此同时，大数据安全发展也需要得以保障，在开发大数据应用的时候提高管理水平和保证良性发展是我们再开发利用过程中所注意的方面。

1.2 轨迹数据处理现有工作的概述及评价

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，公共交通安全管理以及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。在轨迹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。

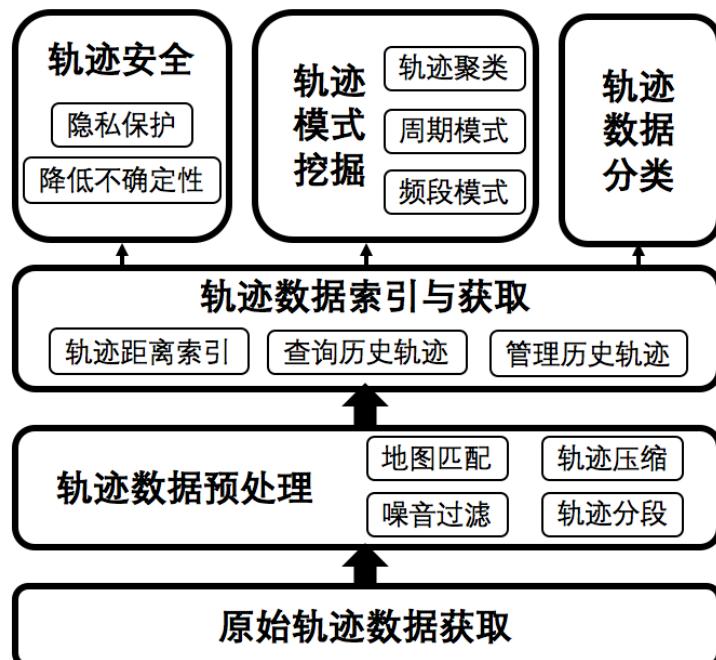


图 1-1 轨迹数据挖掘范例
Fig 1-1 Paradigm of trajectory data mining

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。

1.2.1 轨迹查询简介

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是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。

1.2.2 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-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-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-query* 查询以评价一条轨迹连接多个查询点的好坏。在 *T-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)、最长公共子序列方法 (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。

1.2.3 相似轨迹查询技术现状概述

相似轨迹查询技术目前主要通过轨迹分类、轨迹聚类或计算轨迹距离度量这三种大概方法来实现。在这三类方法的实现中，对于相似轨迹之前匹配的模型建立和算法改进和对长度较长数据的分段算法是问题解决中的主要关注点。

1.3 相似轨迹查询问题描述

1.3.1 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。

1.3.2 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^[3] 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。

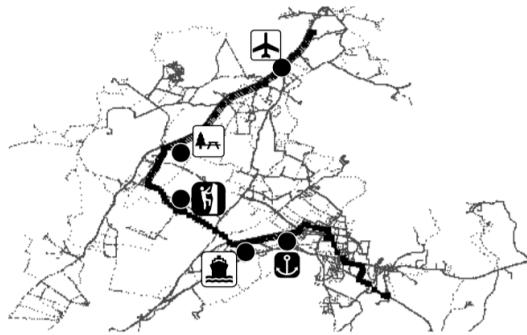


图 1-2 基于位置点的相似轨迹查询^[3]

Fig 1-2 Similar trajectory search by locations

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

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

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-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。

1.4 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。

1.5 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。

1.6 论文结构

毕业设计论文主要结构为：本章绪论介绍轨迹数据处理和相似轨迹查询的相关背景，以及在该背景下对相似轨迹查询这一问题的概要描述，在该问题下所面临处理困难和挑战；第二章以相关工作为主，介绍本文相似轨迹处理查询方法设计与实现所设计到的相关理论算法基础和有关设计技术支持；第三章从实现算法入手，介绍具体的实现细节和分布式实现方法；本文第四章和第五章分别介绍开发相似轨迹查询系统的需求分析和设计概要，前者作为设计指导，明确本文应该从什么方向针对性地去进行系统设计；第六章作为系统测试和评价章节；最后对本文所涉及的工作进行总结和未来展望。

1.7 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技术方法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。最初相似轨迹查询通过首先对一段序列数据进行离散傅里叶变化（Discrete Fourier Transform），根据帕塞瓦尔定理在 $O(n \log n)$ 实现时间复杂度内将轨迹做近似匹配得到相似轨迹^[4]。在此基础上，通过离散小波变换（Discrete Wavelet Transformation）在降低查询维度上的相似轨迹匹配得以实现^[5]。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）的技术来最小化轨迹序列之间的距离以完成相似轨迹查找^[6]，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^[7] 只能应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^[8]。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定义和变换轨迹以找到相似轨迹^[9]，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^[10]。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。

一种名为 Fréchet^[11] 距离的相似度方程在相似轨迹查询中有着比较普遍的应用。假设一个人和一只狗用一根绳子连接，从起始点各自沿着一条路径行走到终点，并允许他们以不同的速度前进但不允许往回走。直观理解而言，Fréchet 距离就是狗绳距离：人和狗各自在走完各自路径所需要的最短的狗绳长度。两个各有 m 和 n 个离散点组成的轨迹之前的 Fréchet 距离可以在 $O(mn \log(m + n))$ 时间复杂度内计算，对于离散型的轨迹点，^[11] 称之为离散 Fréchet 距离，可以通过动态规划算法在 $O(mn)$ 时间复杂度内计算。之后 Agarwal 等人^[12] 也提出并验证了离散 Fréchet 距离在次二次时间复杂度内的计算方法。

但该算法对于两条交错轨迹的相似度计算应用上存在一些需要提高的地方。对于给定的一对轨迹 T_1 和 T_2 ，可能会有大量的潜在轨迹点对能够产生理论上合理的 Fréchet 距离。因此通过最小化 Fréchet 距离来表示轨迹之间的相似度并不一定能够满足之后的需求。为了解决这个问题，研究人员通过用将 Fréchet 距离平均化而不是最小化来表示轨迹之间的相似度。这个思路的具体实现基于目前比较流行的动态时间规整（Dynamic Time Warping）技术。动态时间规整技术最初是提出并应用是在对声音信号的匹配上^[13]，在相似轨迹查询这一领域，动态时间规整技术可以较有效地查询采样率不同的相似轨迹。因为这一技术通过对两条轨迹之前所有点都进行一一匹配，结果准确性会因为一些噪音点或者轨迹点偏差而受到影响。由于测量造成的误差在目前轨迹处理技术中还缺乏准确有效地预处理技术。针对这种潜在的轨迹数据误差，最长公共子序列算法^[14] 等算法得以提出以在相似轨迹

查询中对误差数据具有较好的鲁棒性。

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^[14]，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^[15] (Edit Distance On Real Sequences) 的方法被提出。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。类似该工作，Sayan Ranu 等人提出了 EDwP^[16] (Edit Distance with Projections) 的相似轨迹查询方法。其主要思路在于对轨迹之间相互投影，使得经过投影部分的轨迹序列在地理形状上和时序上保持一致性或相似性。在投影过程中对已投影的轨迹用自定义的一种名为 TrajTree 的所以结构进行存储，其中越靠近根节点的轨迹数据表示具有越多一致或相似部分的轨迹。作者也通过实验验证该算法对噪音点处理上的强鲁棒性和 TrajTree 数据结构在进行 k 最近邻查询时的高效性，不过该方法过程对轨迹相似度的查询中对时间间隔较大的轨迹数据查询性能相对较差。

在此之后，针对时序相似度获取的 ERP (Edit distance with Real Penalty)^[17] 算法被提出。这一算法被认为是 L1 范式和距离修改方程的较好结合算法：L1 范式应用在相似度距离方程中而用距离修改方程解决时间偏移问题。由于 ERP 算法是基于距离修改方程，该算法可以在 k 最近邻查询剪枝过程中处理三角不等式带来的相似度问题。而在轨迹存储上，ERP 算法选择了 B^+ 树来存储轨迹数据以在该算法中减少存储需求和节约可能的 I/O 开销。这种结合这可以看成是 GEMINI (GEnome MINing) 对时序数据索引的拓展工作。

DISSIM^[18] 和 MA (model-driven assignment)^[19] 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。

另一方面，在生物计算方面的序列校准领域的研究技术，例如找到 DNA 或蛋白质的序列相似度，也通过在某些情况下使用于相似轨迹的查询领域。书^[20]中提及的通过建立模型来分析生物序列的方法，被拓展为对两条轨迹序列的校准。这一方法在处理相似轨迹时会有较好的应用性能。但是对于采样率不同的轨迹，基于方法^[20]的相似轨迹处理会使得轨迹间原本相似的部分由于算法建模中一一对应原则导致被视作不相似的。

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^[21]，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 (或称为噪音点) 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	
$DTW(R, S)$	$= \begin{cases} 0 & \text{if } m = n = 0 \\ \infty & \text{if } m = 0 \text{ or } n = 0 \\ dist(r_1, s_1) + min\{DTW(Rest(R), Rest(S)), \\ DTW(Rest(R), S), DTW(R, Rest(S))\} & \text{otherwise} \end{cases}$
$ERP(R, S)$	$= \begin{cases} \sum_1^n dist(s_i, g), \sum_1^m dist(r_i, g) & \text{if } m = 0, \text{ if } n = 0 \\ min\{ERP(Rest(R), Rest(S)) + dist(r_1, s_1) \\ ERP(Rest(R), S) + dist(r_1, g), \\ ERP(R, Rest(S)) + dist(s_1, g)\} & \text{otherwise} \end{cases}$
$LCSS(R, S)$	$= \begin{cases} 0 & \text{if } m = 0 \text{ or } n = 0 \\ LCSS(Rest(R), Rest(S)) + 1 & \text{if } \forall d, r_{d,1} - s_{d,1} \leq \epsilon \\ max\{LCSS(Rest(R), S), LCSS(R, Rest(S))\} & \text{otherwise} \end{cases}$
$EDR(R, S)$	$= \begin{cases} n, m & \text{if } m = 0, \text{ if } n = 0 \\ min\{EDR(Rest(R), Rest(S)) + subcost, \\ EDR(Rest(R), S) + 1, EDR(R, Rest(S)) + 1\} & \text{otherwise} \end{cases}$

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 应用的软件开发框架。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

¹ $dist(r_i, s_i) = L1 \text{ or } L2 \text{ norm}$; $subcost = 0$ if $r_1, t-s_1, t$, else $subcost = 1$

用的 Werkzeug WSGI 工作集^[22]。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。

2.2.3 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*^[23] 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。

2.2.3.1 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*^[24] 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。

2.2.3.2 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*。

2.2.3.3 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^[25] 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。

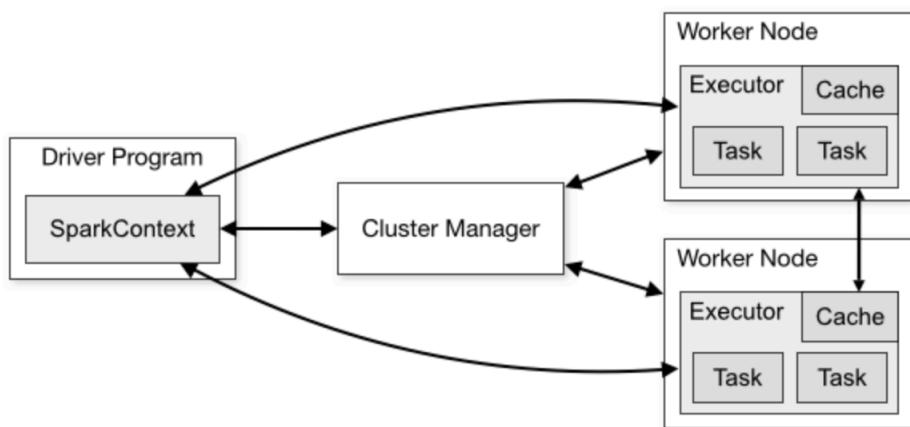


图 2–2 集群管理大致模式^[25]

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^[26] 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 *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。

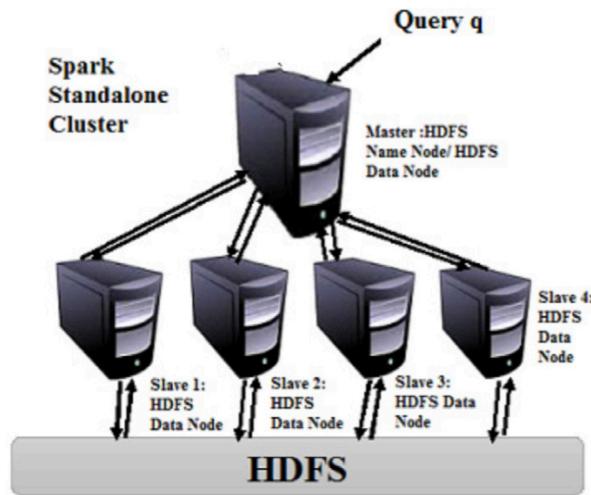


图 2-3 Spark Standalone 集群结构^[26]

Fig 2-3 Spark Standalone Architecture

2.2.4 轨迹数据预处理

2.2.4.1 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，因为中国国家测绘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

2.2.4.2 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^[27] 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。

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

¹<https://github.com/googollee/eviltransform>

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。

2.2.5 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^[28, 29] 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^[28] 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。

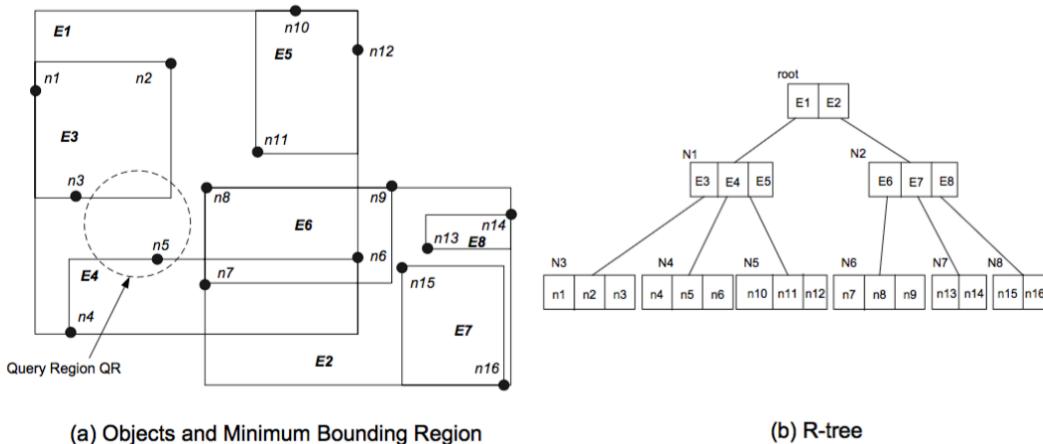


图 2-4 R 树数据结构举例^[28]
Fig 2-4 Two views of an R-tree example

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。

2.3 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 最佳连接定义

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 $\{p_1, p_2, p_3, \dots, p_m\}$ ，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。

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

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 $R = \{p_1, p_2, p_3, \dots, p_m\}$ 的距离为 D_q ，即

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

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 $\langle q_i, p_j \rangle$ 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 $R = p_1, p_2, \dots, p_n$ 而查询点为 q ， $\langle q, p_i \rangle$ 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 $\langle p_i, q \rangle$ 是轨迹 R 到查询点 q 的最短匹配点对。

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

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^[3]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 $\langle q_1, p_3 \rangle, \langle q_2, p_4 \rangle, \langle q_3, p_7 \rangle$ 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。

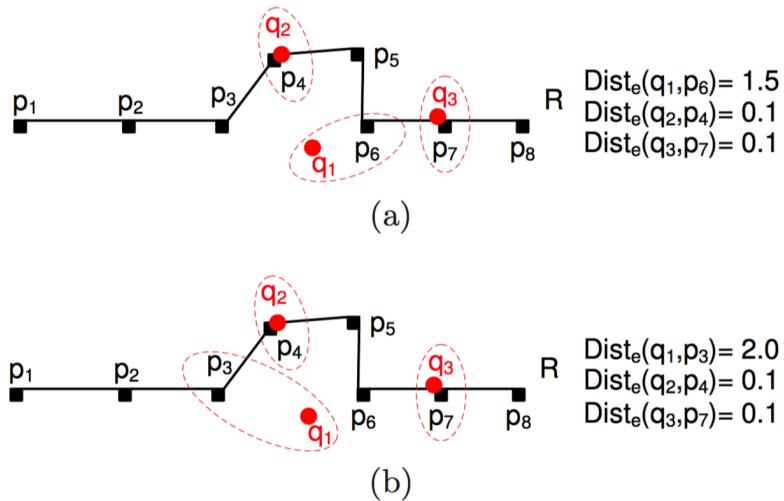


图 3-1 查询点与轨迹点之间的匹配^[3]
Fig 3-1 Match between query points and trajectory point

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 $Q_o = \{q_{o1}, q_{o2}, q_{o3}, \dots, q_{on}\}$ 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 $\{q_2, q_3, \dots, q_n\}$ 。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 $Head(Q)$ 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。

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

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 $T = R_1, R_2, R_3 \dots, R_n$ 、一组查询点 $Q = q_1, q_2, q_3, \dots, q_n$ 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。:

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

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。

算法 3-1 有序相似度算法 dp_Similarity(Q,R)

输入：查询点集 $Q = q_1, q_2, q_3, \dots, q_m$, 轨迹 $R = p_1, p_2, p_3, \dots, p_n$

输出：查询点集 Q 和轨迹 R 之间的有序相似度 $Sim_{order}(Q, R)$

```

1: Initialise 2-dimensional array  $M[m + 1][n + 1]$ ;
2:  $M[i][0] \leftarrow 0$  for  $1 \leq i \leq m$ ;
3:  $M[0][j] \leftarrow 0$  for  $1 \leq j \leq m$ ;
4: for  $1 \leq i \leq m$  do
5:   for  $1 \leq j \leq n$  do
6:     if  $e^{-D_e(Head(Q), Head(R))} + M[i - 1][j] > M[i][j - 1]$  then // $q_i$  和  $p_j$  匹配
7:        $M[i][j] \leftarrow e^{-D_e(Head(Q), Head(R))} + M[i - 1][j]$ ; //重复  $p_j$ 
8:     else
9:        $M[i][j] \leftarrow M[i][j - 1]$ ; //略过  $p_j$ 
10:    end if
11:   end for
12:   return  $M[m][n]$ ;
13: end for

```

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 $Sim_{order}(\{q_1, q_2, q_3, \dots, q_i\}, \{p_1, p_2, p_3, \dots, p_j\})$ 。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 $\langle p_i, p_j \rangle$ ，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

$M[i][j] = M[i][j - 1]$ 。由于历史轨迹点的数目各不相同，对于重复使用的点，我们需要将相似度计算结果正则化以符合实际。

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。

3.3 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 -NearestNeighbor 搜索半径	ξ	查询点 q_i 对相似度上界贡献
μ, v	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 相似轨迹查询问题概述

轨迹数据为一组有序点集，轨迹 R 可以被表示为 $R = p_1, p_2, \dots, p_n$ ，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 $Q = q_1, q_2, \dots, q_m$ ，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^[27, 30] 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 $D' = R_1, R_2, \dots, R_k$ ，满足式3-5。其中 Sim 根据用户定义选择是否考虑有序性。：

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

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。

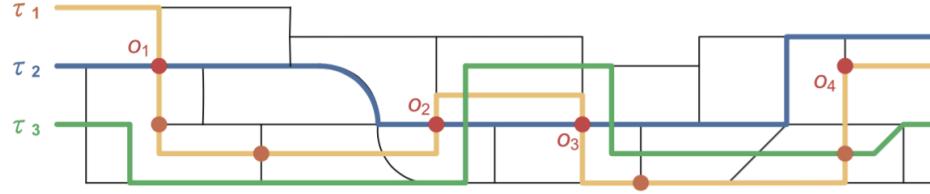


图 3-2 基于位置点的相似轨迹查询^[31]
Fig 3-2 Search similar trajectory by location points

3.3.3 相似轨迹查询算法实现

如图3-2^[31]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^[32]。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^[33]基础上，实现了增长型 k 最近邻查询算法^[3]，结合备选和筛选（candidate-refinement）的剪枝思路^[34]，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 ($k=\lambda$) 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 $\min_{R_x \in C'} LB(R_x) \geq UB_{ns}$ 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 $\forall R_a \forall R_b (R_a \in C' \wedge R_b \notin C), Sim(Q, R_a) \geq Sim(Q, R_b)$ 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??^[3] 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。

3.4 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

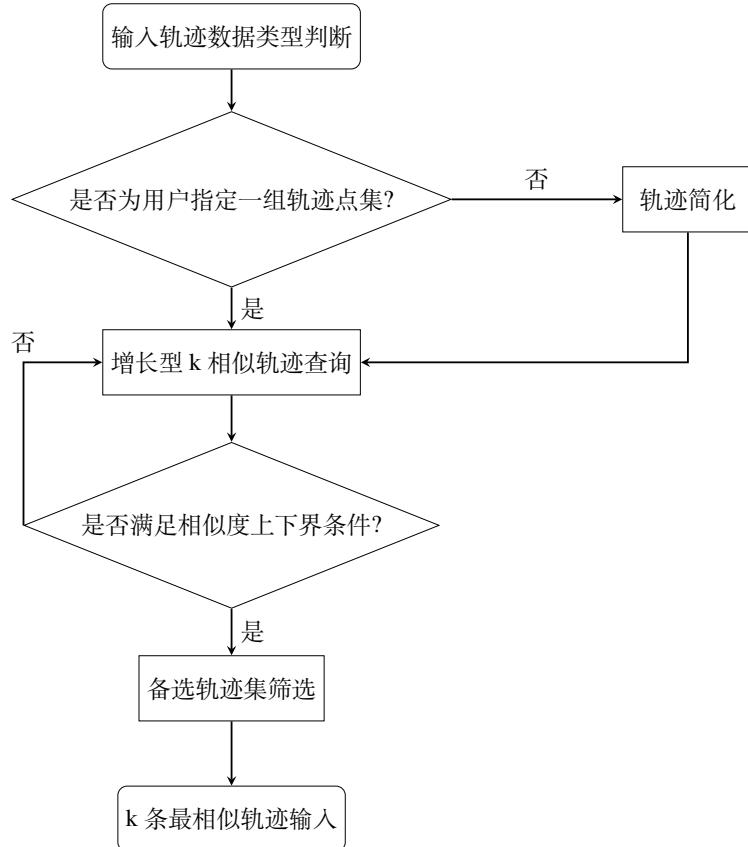


图 3-3 相似轨迹查询算法流程图

Fig 3-3 Searching algorithm flow chart

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^[31]。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。

3.4.1 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。

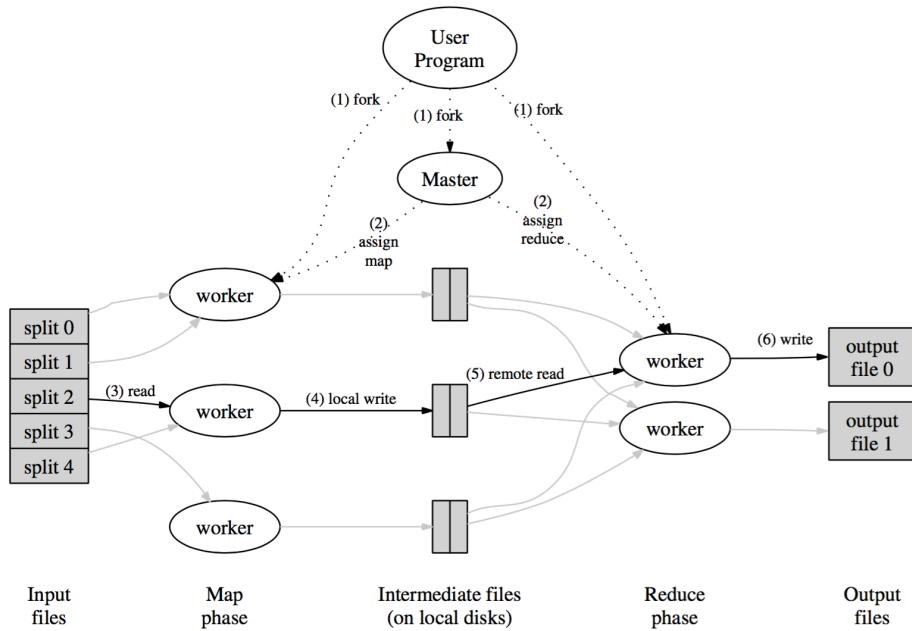


图 3-4 Spark 分布式相似轨迹查询流程^[23]

Fig 3-4 The flow of Spark distributed search similar trajectories

图3-4^[23]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'*^[30]。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3-2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。

3.4.2 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

算法 3-2 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 $k\text{-}Trajs$

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

集, 然后分配个集群中的子节点, 让他们各自处理一部分请求。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。

3.5 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 相似轨迹查询系统需求分析

4.1 本章序言

本文中，相似轨迹查询系统需求分析需要明确：在该查询系统中，针对不同的用户类型提供的轨迹查询点集或者一条历史轨迹输入，在满足系统交互原则的基础上，准确高效通过可视化的方式输出系统所处理出的相似轨迹结果。在开展相似轨迹查询系统设计之前，我们首先对要设计的系统所涉及的实际需求进行描述。在定义好该系统所需要解决的应用问题和需要满足的用户需求之后，在因地制宜设计系统框架。

4.2 需求分析概述

目前相似轨迹查询这一领域的研究成果在学术方面有着丰富和较为系统的成果，不过目前这些学术成果移植到系统应用中的样例较为稀缺。在国内开发的地图应用或者提供以地图搜索为主体的公司在相似轨迹查询这一领域所投入的精力并不多，导致基于相似轨迹查询的系统软件发展投入在实际应用中的比例相对较少。

4.2.1 系统需求背景

随着 GPS 技术不断发展和私家车数量的激增，生活中每天都会产生海量数据规模的轨迹数据。用户想要根据特定需求查询轨迹（例如本文所要解决的查询相似轨迹），通过传统的软件系统难以高效查询出所期望的结果。我们通过基于目前主流的算法实现，设计出快速、准确的相似轨迹查询系统，借助网页地图接口将查询结果清晰了然展示在用户界面上。系统在实现相似轨迹查询的基础上，可以为用户提供轨迹路径规划和轨迹推荐等基于相似轨迹查询的应用服务。

4.2.2 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。

4.2.3 系统应用定位分析

相似轨迹查询系统软件设计的初始主要目标为外出旅游对景点进行路径规划、外出前查询起始地点路径和进行日常拼车需求的轨迹数据使用用户。由于社会人员拥有私有交通工具的比例上升以

及配置 GPS 轨迹设备的流行性，越来越多的用户需要借助可视化的轨迹数据管理方法来查询和管理自己的轨迹数据，其中相似轨迹查询便是其中需求之一。相似轨迹查询问题可以在上述的三种初始应用中，借助系统内部的地图可视化接口，满足用户需求。

4.2.4 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。

4.2.5 系统可行性分析

本文通过搭建 Web 框架实现相似轨迹查询系统，从如图4-1几个方面阐述搭建相似轨迹查询系统在用户人群中使用的可行性：

- 网络应用基础设施完善：现在在中国，互联网的发展使得人们可以通过 PC 终端和移动端访问各类网站。网络应用在企业公司和家庭用户中都有着广泛的普及，这为用户方便使用相似轨迹查询系统提供可能性。
- 地图数据处理数据优秀：在国内外诸如谷歌地图、百度地图和高德地图等等公司提供的基于各种开发平台的地图应用程序接口使得系统开发人员能够跨平台的开发出各种有关地图数据的应用。且各种接口功能丰富，性能优秀，可视化程度良好。
- 轨迹数据规模庞大：轨迹数据生成和处理技术的成熟发展使得系统所存储的轨迹数据规模在一定的用户使用规模上是具备大数据规模的，使得系统通过相似轨迹处理得到结果的准确性和实用性进一步得到保证。
- 网络应用技术普及：网络技术、Web 技术、各种网络数据库技术的发展为本文开发基于 Web 框架的系统软件提供可能性，这些主流技术的系统发展是本文开发与设计系统的主要支持，为本文系统的实现提供保障。

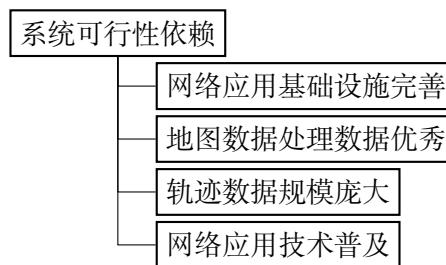


图 4-1 相似轨迹查询系统可行性

Fig 4-1 Feasibility of searching similar trajectories system

4.3 功能需求

4.3.1 用户功能需求

1. 一般查询用户:
 - 基于轨迹点集的相似轨迹查询功能: 一般查询用户在地图界面上点击地理位置或根据查询输入窗口输入查询地点, 同时可以增加或删除查询点。查询轨迹点集会以小图标的形式显示上地图上。结合查询日期条件, 搜索出相似轨迹。
2. 登录用户:
 - 登录功能: 登录用户 ID 和密码登录到用户界面, 显示用户的历史轨迹。
 - 显示历史轨迹数据功能: 登录用户可以选择一条历史轨迹数据, 展示与地图上
 - 基于历史轨迹的相似轨迹查询功能: 在选择一条历史轨迹时候, 登录用户在设定好查询时间参数后可以进行相似轨迹查询功能。
 - 基于轨迹点集的相似轨迹查询功能: 一般查询用户在地图界面上点击地理位置或根据查询输入窗口输入查询地点, 同时可以增加或删除查询点。查询轨迹点集会以小图标的形式显示上地图上。结合查询日期条件, 搜索出相似轨迹。

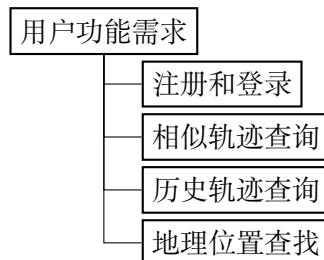


图 4-2 相似轨迹查询系统用户功能需求
Fig 4-2 User function requirement in the system

4.3.2 用户界面需求

- 用户登录界面: 实现用户账户与密码登录。
- 功能选择界面: 进入相似轨迹查询系统后选择用户所需要的功能。
- 地图显示界面: 在一般用户和登录用户模式中都可以显示城市地图信息, 通过高德地图^[35] 接口实现附属界面功能。
- 地理位置点输入界面: 在输入界面中输入地址位置名称, 显示对应的地理经纬度坐标并在地图上显示具体位置。
- 查询条件设定界面: 相似轨迹查询窗口, 结合用户自定义条件作为相似轨迹查询的附属输入参数。
- 历史轨迹显示界面: 显示登录用户的历史轨迹数据条目。

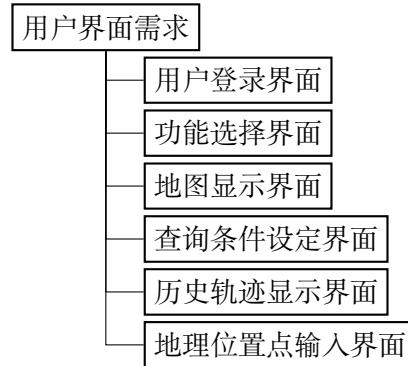


图 4-3 相似轨迹查询用户界面需求
Fig 4-3 User interface requirement in the system

4.4 性能需求

4.4.1 系统查询准确率

相似轨迹查询应保证查询结果与输入轨迹数据相似度较高。在地理语义上，相似轨迹查询结果应该尽可能地靠近查询输入点集或是输入轨迹。以图4-4^[26]为例，其中 T_1 、 T_2 和 T_3 是历史轨迹，轨迹查询点集为圆圈表示，从图中我们可以看出 T_1 相对于 T_2 和 T_3 ，是更相似于输入的历史轨迹，因此在系统查询中，若以上述查询点集作为输入，首先应输出 T_1 以保证查询准备率。

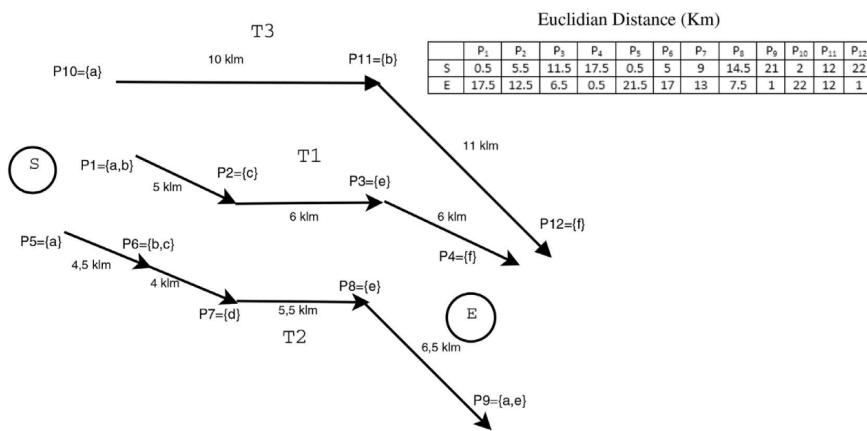


图 4-4 系统查询结果准确性^[26]
Fig 4-4 Accuracy of searching

4.4.2 系统查询时间特性

在相似轨迹查询系统中，单机查询点集操作普遍在一秒内完成；若输入点过多或者以一条轨迹作为查询输入，所需的查询时间会到两秒。在交互型系统中，这样这时间操作在可接受范围之内。

4.4.3 系统适应性

相似轨迹查询系统可以根据用户特定的查询需求满足有序或无序、有关轨迹日期、查询数目限制等类型。在满足运行条件的情况下可正常运行于各种场景。

4.5 本章小结

系统软件需求分析是在对系统如何开发的概要描述。本章节主要针对相似轨迹查询系统在目前研究环境中，要解决的主要问题进行相似的需求分析，明确系统的问题要求，了解输入与输出，并从用户的角度描述了系统应该具备哪些实际的功能，达到什么的性能标准。与此同时，本章节内容也分析了相似轨迹查询系统在功能性与非功能性中的大致组成框架，为从事相关领域的软件工程师确定用户的基本使用需求。在明确这些需求之后，本文才开始对相似轨迹查询系统进行针对性的设计与构思，为下一章节相似轨迹查询系统的设计与实现提供基础。

第五章 相似轨迹查询系统设计与实现

5.1 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 Werkzeug 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。

5.1.1 系统设计编写背景

待开发的系统为相似轨迹查询系统。本章节设计概要可作为系统设计人员、系统测试人员、系统使用用户的参考资料与文档。设计实现服务对象为需要进行相似轨迹查询的互联网用户。

5.1.2 系统设计编写原则

- 准确性：本章节所描述的针对系统的需求，设计系统的功能与行为与目标软件系统产品实现期望相符。
- 完整性：包括全部有实际应用意义的系统需求，在功能、性能和设计方面均满足所需的需求。依照 < 系统软件设计规范说明书 >^[36] 的要求，将描述系统需求所需的流程图和名词定义给出完整描述。
- 易读性：本章节内容通过简要文字，辅以插图和图标，让读者对象能够清楚了解系统设计中的需求内容。
- 可验证性：对于本章节中所编写的系统需求分析与设计内容，读者可通过实际系统进行功能、性能和设计方面上的检查以校对是否与编写内容一致。

5.1.3 系统名词定义

表5-1所示。

5.1.4 用户类与特点

相似轨迹查询系统的使用者主要分为轨迹查询用户和管理用户。

对于轨迹查询用户而言，我们通过是否具有在系统中存储历史轨迹数据分为登录用户和一般用户。一般用户是系统的流动用户，在系统轨迹数据库中不具备历史轨迹。一般用户只能通过向系统输入一组轨迹查询点集以完成相似轨迹查询功能。登录用户可以根据账户信息和密码登录相似轨迹查询系统，使用已有的历史轨迹数据或是自定义的一组轨迹点集实现相似轨迹查询功能。

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

表 5-1 相似轨迹查询系统涉及名词定义

Table 5-1 A list of definitions and explanations in the system of searching similar trajectories

相似轨迹查询系统中的管理员主要负责登录用户的权限管理和对历史数据的修改处理操作。管理员对于相似轨迹查询系统中违反查询和使用规范的用户可以修改其登录权限以阻止其进一步违规操作。管理员还可以对历史轨迹进行增加、修改和删除操作，以保证系统数据库的实时性和准确性。

5.1.5 系统运行环境

表5-2对本系统运行环境进行了描述。

序号	环境需求	环境参数
01	系统操作系统	Mac OS 10.11.6
02	分布式操作系统	Ubuntu14.04.1 amd64
03	单机 PC 设备参数	处理器 2.6 GHz Intel Core i5 且内存大于等于 4G
04	分布式 PC 设备参数	处理器 2.6 GHz Intel Core i5 且内存大于等于 1G
05	服务器平台	Flask0.12/Python2.7.6

表 5-2 相似轨迹查询系统运行环境

Table 5-2 A list of environment parameters in the system of searching similar trajectories

5.2 系统设计

本文设计的相似轨迹查询系统运行于基于 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。

5.2.1 系统设计分析

将相似轨迹查询系统以 Web 框架搭建，其最大的好处是给予用户在查询上最大程度的便利。这种便捷主要体现在用户只要连接上互联网，就可以直接通过浏览器访问系统，在线进行相似轨迹的查询。这种网络操作不需要用户在下载系统到终端或者移动端便可以进行实时的相似轨迹查询功能。

所以在设计相似轨迹查询这一系统时，主要功能实现均为用户查询而服务，即最重要的一点是完成用户功能部分。

5.2.2 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。

在完成登录功能或跳选到直接查询后，用户可以在系统界面中完成查询任务。相似轨迹查询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。登录用户可以在地图上点击历史轨迹数据显示系统数据库中存储的某一条与自己相关的历史轨迹。

5.2.3 系统设计框架

如图5-1所示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。

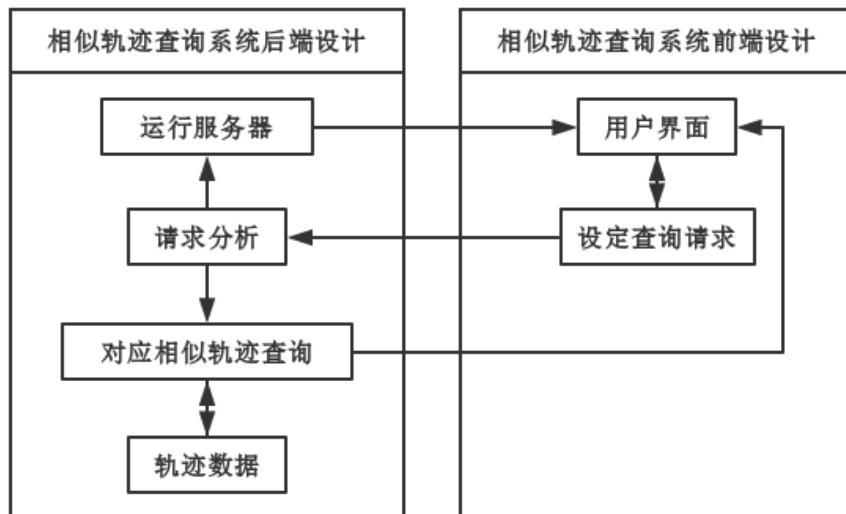


图 5-1 相似轨迹查询系统设计框架

Fig 5-1 System design architecture

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。

5.2.4 系统处理流程

图5-2展示了相似轨迹查询系统的主要流程。进入相似界面之后，用户可以直接使用系统（即不需要登录）使用系统进行相似轨迹查询，在获取满意的查询结果之后退出系统。同时，如果需要运用登录功能，系统会在登录界面中提醒用户是选择轨迹历史用户登录还是管理员模式登录。对于轨迹历史用户，可以实现相对应的轨迹查询类型并和一般用户一样在得到满意的查询结果之后退出系统。对于管理员而言，他们主要负责对用户权限和轨迹数据处理这一部分工作，在完成管理员工作之后他们也可以按流程退出系统。

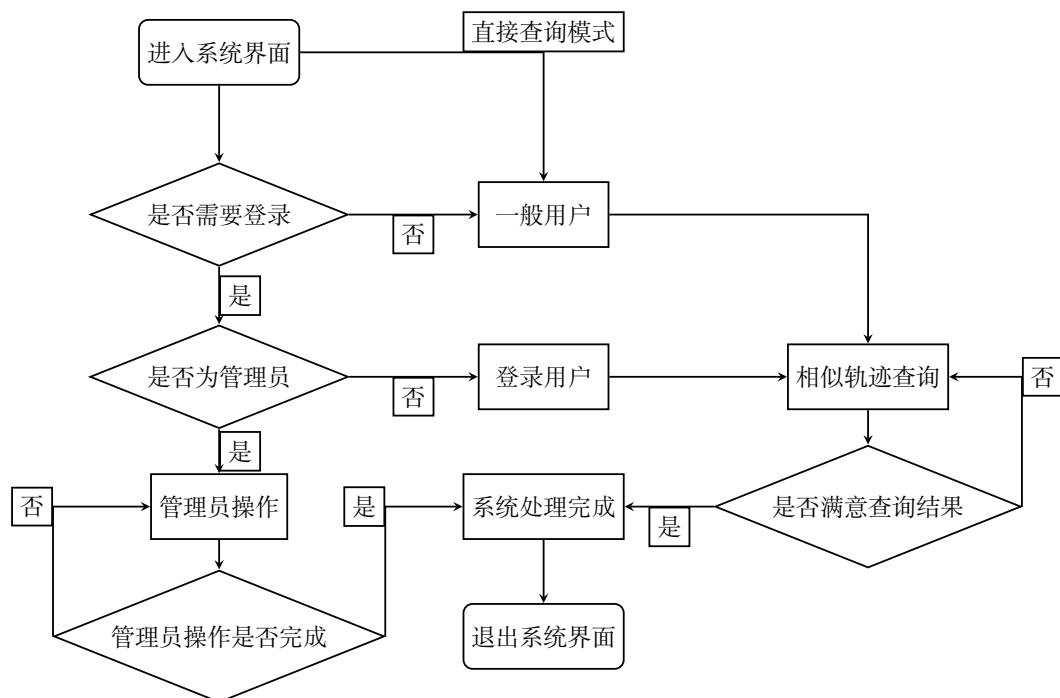


图 5-2 相似轨迹查询系统处理流程图
Fig 5-2 A flow chart of searching similar trajectory system

5.3 接口设计

5.3.1 外部接口

由于相似轨迹查询系统没有直接的硬件系统设计，该系统中没有直接的外部硬件接口。存储于系统中的 GPS 轨迹数据由系统内部文件系统或数据库来管理，其中的服务器或数据的连接操作由设

备内部的操作系统负责。运行环境中对系统所需要运行的硬件设备和操作系统环境在表5-2予以表述。

5.3.2 内部接口

相似轨迹查询系统内部数据传递主要由三层结构实现

- 系统表达层
- 系统逻辑层
- 系统数据层

系统表达层使用 flask 提供的 Jinja2^[22] 模板实现系统的可视化界面和网页交互界面，采用基于 Bootstrap 的设计框架进行直接表达；逻辑层采用 Python 中的修饰符和 flask 中的将请求与数据处理结合；数据层通过直接对分布式文件系统读写或 sqlite3 数据库连接进行数据处理。

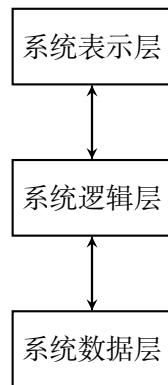


图 5-3 相似轨迹查询系统数据层次
Fig 5-3 Data passing level in the system

为了满足上述三层结构的正常实现，我们设计的内部接口主要为以下几种。表达层与逻辑层本文实现系统采用值传递的接口方式。表达层获取逻辑层传递的值进行界面刷新处理，同时逻辑层接收表达层的请求值以进行后续处理。逻辑层与数据层之间使用文件系统或数据库已有的命令操作，逻辑层借用底层提供的接口实现对数据的处理操作，数据层对逻辑层提供数据查询支持。

5.3.3 用户接口

用户必须具有支持地图加载功能浏览器，本文测试系统所使用的浏览器为 *Safari9.2.1*。相似轨迹查询系统主要通过用户 Web 页面界面进行实时的信息交互与查询处理，达到信息传递和查询结果可视化的目的。该系统主要提供以下几种用户接口：

- 用户登录与注销接口
- 功能选择接口
- 地理位置点查询接口
- 相似轨迹查询接口
- 历史轨迹展示接口

5.4 系统总体结构

根据系统处理流程：

- 系统初始时，进入系统主界面。可以完成一般操作或选择登录实现历史轨迹操作。
- 如果用户登录成功，则会显示用户信息和对应的历史轨迹以完成用户需求工作。
- 当用户完成查询工作，可以选择注销退出用户界面或直接选择关闭浏览器以退出系统。

我们大致按照下述章节来对系统进行总体结构设计。

5.4.1 系统组成模块

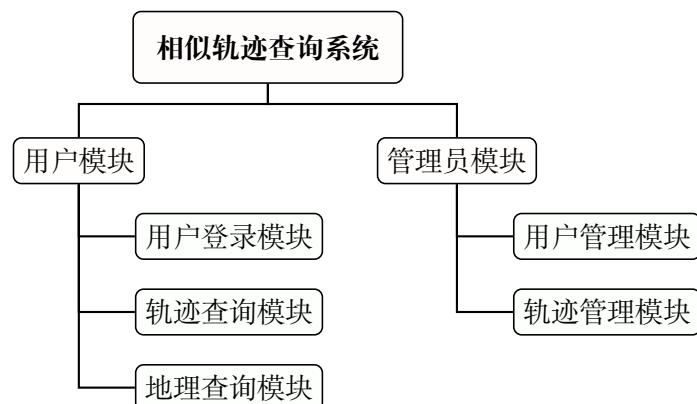


图 5-4 相似轨迹查询系统模块组成

Fig 5-4 Modules of Searching similar trajectories system

如图5-4所示，本文相似轨迹查询模块主要由以下两个部分组成：

1. 用户模块

- 用户登录模块：完成用户登录功能，用户通过在轨迹数据中的 id 号码和密码进行密钥登录。
- 轨迹查询模块：完成用户相似轨迹查询功能，可以将相似轨迹查询结果展示在地图上。
- 地理查询模块：完成用户对地理位置点的查询，用户在不知道地理点具体位置时候可以通过这一查询功能完成对地理位置在地图上的定位。

2. 管理员模块

- 用户管理模块：完成管理员对登录用户的权限管理和信息管理。
- 轨迹管理模块：完成管理员对系统内部轨迹数据的增删改除功能。

5.4.2 系统功能模块详细设计

1. 用户登录模块

- 模块编号：M1
- 模块名称：用户登录模块
- 模块输入：用户名信息与对应密码
- 模块处理：决定用户信息与对应密码是否匹配

- 模块输出：若匹配则跳转到功能页面，否则重新输入

2. 轨迹查询模块

- 模块编号：M2
- 模块名称：轨迹查询模块
- 模块输入：查询轨迹点集或历史轨迹
- 模块处理：根据输入进行相似轨迹查询
- 模块输出：在地图上输出 k 条最相似轨迹查询结果

3. 地理查询模块

- 模块编号：M3
- 模块名称：地理查询模块
- 模块输入：查询地理点名称信息
- 模块处理：调用地图查询接口寻找对应定理信息的坐标
- 模块输出：在地图上显示具体位置并在界面中显示地理点坐标

4. 用户管理模块

- 模块编号：M4
- 模块名称：用户管理模块
- 模块输入：用户信息和对应操作
- 模块处理：管理员对相应用户进行权限限制操作
- 模块输出：被限制用户暂时无法使用系统

5. 轨迹管理模块

- 模块编号：M5
- 模块名称：轨迹管理模块
- 模块输入：轨迹信息
- 模块处理：管理员根据输入的轨迹信息对轨迹数据库进行更新操作
- 模块输出：更新后的系统轨迹数据库

5.5 系统用户界面设计

图5–5和图5–6分别为用户使用功能的窗口和用户使用系统功能的主要界面设计实现截图。

5.6 本章小结

本章基于系统需求分析，将上一章节中对系统的需求描述转换成具体接口设计结构与对应模块设计结构：将一个复杂的相似轨迹查询系统以小模块子系统的结构进行分解，并建立模块之间的数据传递和功能调用关系，确定对应接口对应的数据传递层。在这里，对相似轨迹查询系统进行设计阐述的主要目的是将需求逻辑转化为功能模块实现，以便开发人员在开发这一系统时能清晰明了明确自己负责的模块开发部分，也便于用于理解系统的运行逻辑情况。



图 5-5 用户功能窗口

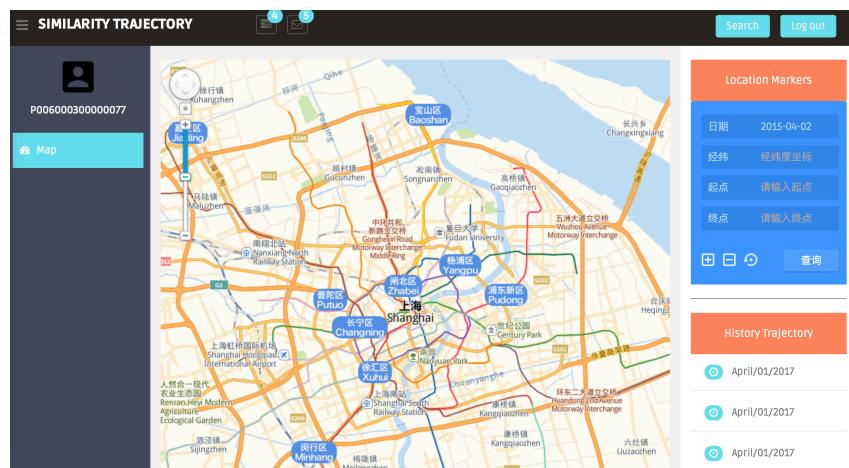


图 5-6 用户地图界面设计
Fig 5-6 User interface design

第六章 系统测试与评价

在系统软件开发中，系统测试与评价是本文关注的重点：系统的开发是否具备完整的基础功能、系统查询过程中能稳定在什么层次的性能水平，以及系统是否有足够的缺陷应对措施都是我们所需要关注的问题。

本章节主要给出系统测试文档。依据前面章节所提出的系统需求分析和系统设计实现文档，对现阶段开发的系统进行功能与性能在综合测试与评价。在测试过程中我们是主要依赖于测试原理和测试手段：测试原理主要根据系统测试理论指导，测试手段是运行系统进行实际应用查询以获取测试数据。另一方面，我们在测试过程中尽可能地模拟实际应用过程，以从用户的角度去测试系统是否具备实用性，因此系统运行环境中我们使用的数据集以实际轨迹数据为主。

6.1 引言

6.1.1 编写目的

编写本章节测试与评价文档主要有以下几个目的：

- 验证系统内部算法实现的正确性与可行性。
- 对系统产品的质量进行评价。通过对系统测试结果的数据分析以评估所设计实现的系统是否符合预期使用价值。
- 分析测试过程中使用的数据和信息，为之后的产品拓展提供参考资料。
- 分析系统产品目前阶段所存在的性能不足和使用缺陷，为下一阶段工作的提高做准备。

6.1.2 针对用户

- 相似轨迹查询系统开发人员
- 相似轨迹查询系统管理人员
- 相似轨迹查询系统使用用户

6.1.3 缺陷定义

- 软件系统未实现需求分析和系统设计中的功能
- 软件系统未满足用户或客户的期望与需求
- 软件系统未满足在需求分析中预计的功能准确性与完备性。
- 软件测试人员难以理解系统设计而无法测试
- 用户个体对系统使用过程后认为系统体验不佳、系统查询效果不良。

6.1.4 测试环境数据

6.1.4.1 测试数据集

上海私家车 OBD 轨迹数据集^[37]。其中有三种数据格式的数据。在系统测试中，我们选择数据定义为 GPS_routine_3 的第三种数据类型私家车轨迹数据，如表6-1所示：

字段名称	含义	类型	备注
id	记录号		自增 id
obd_id	obd 设备号	String	
经纬度	地理坐标		
obd 端时间		时间	yyyy-mm-dd 时分秒
车辆状态	速度、发动机转速		

表 6-1 系统测试选取轨迹数据格式

Table 6-1 A list of data fields of private trajectory data

本文选择其中时间范围为 2015 年 4 月份的轨迹数据。原始轨迹数据的一条数据包含表6-1中的所有的字段值，其中数据项目以时间顺序依次存储。系统处理过程中，假设在第三种数据结构下，如果一个 obd 用户的两个坐标点时间间隔超过半个小时，则两个坐标点属于不同的两天轨迹数据。基于这一假设，系统在预处理阶段将每个用户的轨迹点划分到对应的轨迹数据中，生成用户的轨迹历史数据。



图 6-1 上海市私家车轨迹数据样例^[37]

Fig 6-1 An example of Shanghai OBD private trajectories data

6.1.4.2 测试设备环境

1. 单机测试环境

- Mac OS X EI Caption 操作系统
- 2.6 GHz Intel Core i5 处理器

- 8GB 1600 MHz DDR3 内存
- 2. 分布式集群处理环境
 - Ubuntu14.04 amd64 操作系统
 - 2.6 GHz Intel Core i5 处理器
 - 1GB DDR3 内存

6.2 系统测试概要

本章节大体从系统的功能和性能两个方面入手对本文所设计并实现的相似轨迹查询系统进行测试。在功能测试方面，测试主要的目的在于考察系统实现是否完成需求分析与系统设计中的预期功能，保证用户在使用系统过程中能够正常使用任一个系统的子功能从而保证用户使用系统的流畅性与完整性；在性能测试方面，测试主要的目的在于分析系统实现是否能够高效并准确地为用户提供查询结果或是其他交互方面为用户提供良好的使用体验，在网页系统中，良好的交互性与结果的准确性是一个系统在用户群中不断使用的根本保证。

6.3 系统测试描述

6.3.1 功能测试描述

本文所设计的功能测试中的各类操作主要针对组成相似轨迹查询系统的单一功能模块，每个操作步骤对应的就是一个功能点，即功能模块。由于本文所设计的相似轨迹查询系统由用户模块和管理员模块两个大部分组成，因此我们需要再分别对两个主要模块内部的功能模块进行系统使用测试。

6.3.2 性能测试描述

在单机测试环境中，本文所设计的性能测试主要是为了获取系统在各种应用或者不同输入参数的情况下对查询效率和查询准确的反应情况。由于该系统的主要针对群体是用户对象，因此性能是本次测试中的重点考虑因素，一个良好的系统应该能够给予用户较好的交互体验。由于系统设计与实现并不一定合理，本文尽可能的考虑系统在查询过程中的上限性能，找出在现阶段中系统可能仍然存在的不足或亟待改善的部分，为今后系统的拓展和升级提供参考思路。

对于分布式性能的测试，由于系统设备限制问题，本文在本次测试中仅给出在分布式环境在系统完成任务时所需要的查询时间。我们在测试中证明对于分布式，在对于目前查询性能影响较大的时间因子在于读取 HDFS 文件系统中。也将相似轨迹查询在分布式集群处理环境下的性能提高目标放在今后系统能够运行在更优秀的集群环境中。

6.4 系统测评结果

对于单机测试本文设定一下测试条件：

- 测试变量：相似轨迹查询数目和查询位置点数目。
- 测试指标：相似轨迹查询时间和查询准确度。

- 测试类型：相似轨迹无序查询和有序查询。

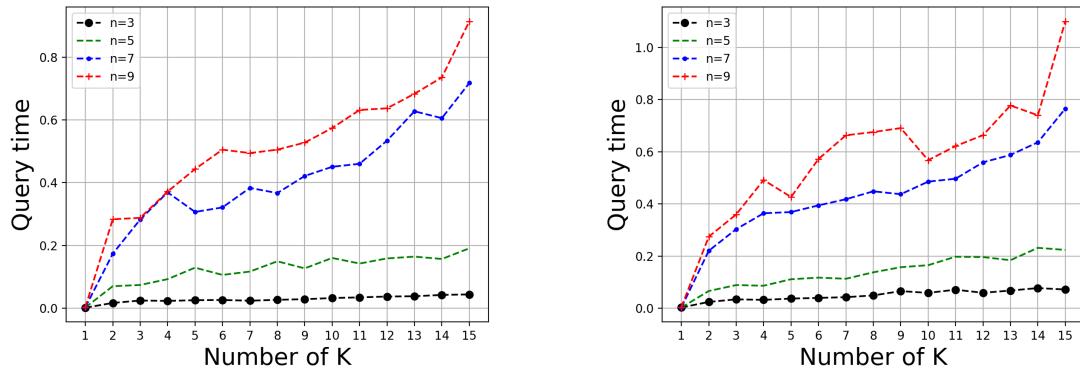


图 6-2 相似轨迹查询数目 k 和查询时间的关系 (左图无序查询, 右图有序查询)

Fig 6-2 Query time for the number of k trajectories

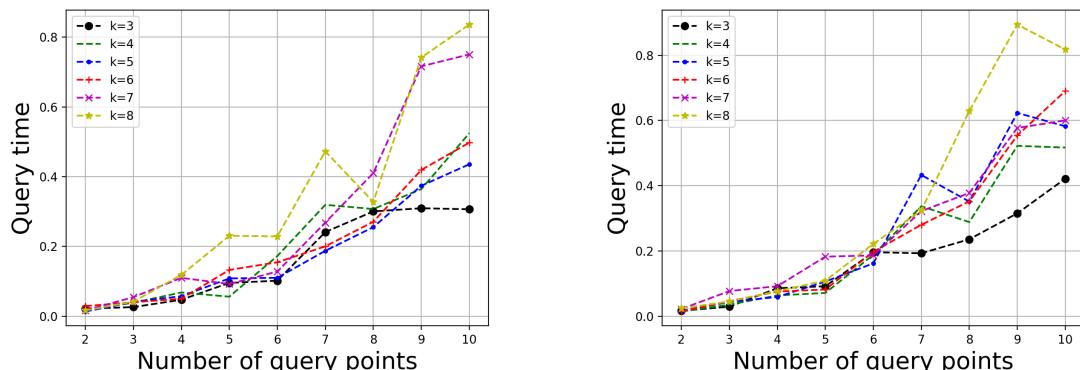


图 6-3 相似轨迹查询点数目和查询时间的关系 (左图无序查询, 右图有序查询)

Fig 6-3 Query time for the number of query points

图6-2和图6-3展示了测试系统变量与查询时间之间的关系。由图中我们可以看出随着查询变量（查询轨迹数目或查询点数目）的上升，查询时间在一定程度上与查询变量成正相关的关系。总体而言，查询变量越大，那系统查询出正确的结果所需要的时间也就越多。在图中对于一条时间曲线会出现下降的情况，本文分析主要是由于轨迹数据集范围问题，即对于本次测试数据，对于某组测试数据能找到更多的相似轨迹，因此系统结束通过剪枝能够更快地退出查询循环，导致查询时间较少，曲线下降。

图6-4和图6-4展示了测试系统变量与查询准确度之间的关系。首先本文对准确度的计算度量为以计算出的轨迹相似度除以一个理想化的相似度（即完全匹配轨迹的相似度）。在测试相似轨迹查询数目对准确率的影响中，由图6-4，两者主要为负相关关系。对于无序查询而言，轨迹查询数目对查询准确率影响并不太大；对于有序查询而言，根据测试结果应在一定范围内尽量选择查询大于三条以上轨迹查询数目以保证准确率。在测试查询点数目对准确率的影响中，由图6-5，两者关系要具

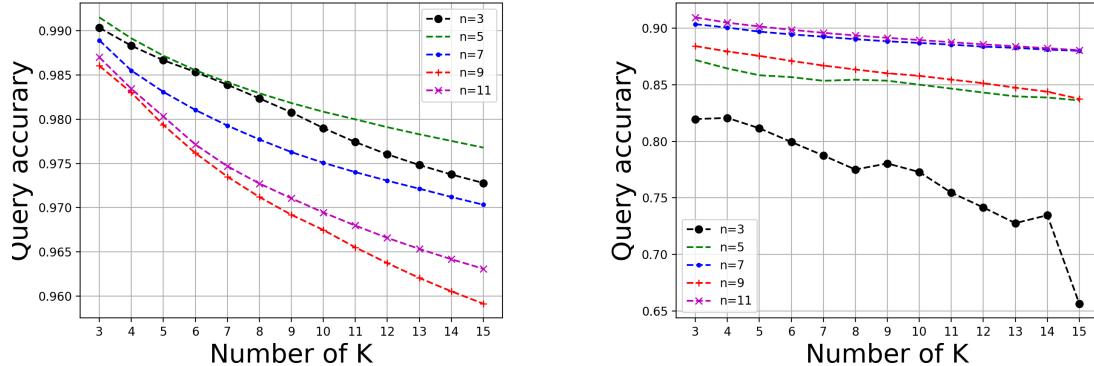


图 6-4 相似轨迹查询数目 k 和查询准确度的关系 (左图无序查询, 有图有序查询)

Fig 6-4 Query accuracy for the number of k trajectories

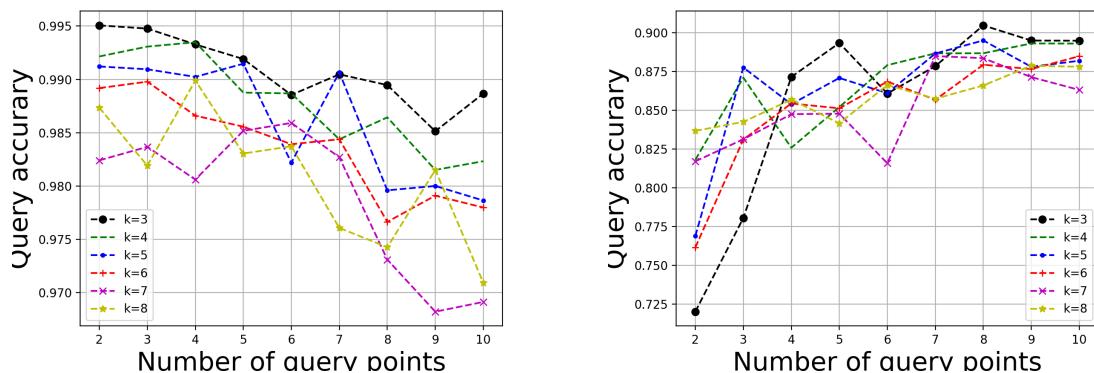


图 6-5 相似轨迹查询点数目和查询准确度的关系 (左图无序查询, 有图有序查询)

Fig 6-5 Query accuracy for the number of query points

体分析。对于无序查询而言，查询点数目对查询准确率影响依旧并不太大，且有随着查询点数目增多准确率降低的趋势；但对于有序查询而言，我们可以发现随着查询点数目的增加准确率反而增加。因此，用户可以先决定无序和有序的查询类型再选择合理查询点的数目以查询到较为准确的结果。

对于分布式集群运行相似轨迹查询系统，本文在分布式测试环境中对读取计算和单机测试环境中读取计算时间。根据表6-2可以看到，在分布式环境和单机环境中，内存直接计算的时间对比几乎相近，可以认为两者在直接计算中处理耗时一致。但在读有关计算中（分布式环境主要由集群节点访问 HDFS 读取轨迹数据），分布式耗时较单机环境相比十分严重，本文根据查询资料认为这一问题主要由测试设备参数和网络连接共同限制，导致分布式运行速率达不到理想值。

6.5 系统测试评价

根据上述的测试结果，本文从测试结果所对应的功能和性能两个方面对相似轨迹查询系统做出客观评价。

序号	分布式环境		单机环境	
	读有关计算	内存直接计算	读有关计算	内存直接计算
1	5.017481	0.000755	0.39443	0.000274
2	2.020407	0.001006	0.52876	0.00182
3	4.515193	0.000701	0.063135	0.000179
4	5.647829	0.000522	0.135711	0.000175
5	32.505241	0.001405	1.92676	0.000467

表 6-2 分布式测试与单机测试比较 (数据单位: 秒 (sec))

Table 6-2 Comparison between cluster machines and single machines result

- 从系统功能方面，相似轨迹查询系统完成了以用户为主体对象的用户模块和一部分管理模块功能，初步完成了一个面向用户群里的相似轨迹查询，在必要功能组成上具备完整性。不过功能组成相对简单，在之后基于现阶段的系统设计与实现任务中，应该更多地结合实际应用以丰富完整系统的功能组成。
- 从系统性能方面，从测试结果提供的查询时间折线图可以观察得出结论：系统较好地实现了内部的算法部分，单机环境查询结果具有高效性和准确性，能够在大部分实际应用中给予用户良好的查询体验。
- 测试过程中也发现了一些不足之处：功能组成相对简单，在之后基于现阶段的系统设计与实现任务中，应该更多地结合实际应用以丰富完整系统的功能组成；性能测试阶段也反应出在测试过程中存在一些误差点，这些误差产生的主要原因主要是由于数据集轨迹较大导致其中一些噪音点没有预先除去，在之后的测试中应当予以注意。

6.6 本章小结

本章节主要对本文根据相似轨迹查询算法所设计并实现的相似轨迹查询系统进行测试和评价。在测试过程中，参考了目前对软件系统测试的基本方法和理论指导，从功能和性能两个主要方面入手对系统展开测试并得出基本结果。基于这一结果，本文再对系统本文进行评价操作，以符合实际情况。

第七章 结论

7.1 相似轨迹系统设计总结

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作我们目标查找的 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，从实际应用需求的角度设计出符合用户需求和数据特点的相似轨迹查询系统，依照对系统本身的需求对系统功能、界面和性能模块做针对性的设计和规划，并对实现好的相似轨迹查询系统做性能评价。

7.2 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[J]. 2011.
- [3] CHEN Z, SHEN H T, ZHOU X, et al. Searching trajectories by locations: an efficiency study[J]. 2010: 255–266.
- [4] AGRAWAL R, FALOUTSOS C, SWAMI A. Efficient similarity search in sequence databases[J]. Foundations of data organization and algorithms, 1993: 69–84.
- [5] CHAN K.-P, FU A W.-C. Efficient time series matching by wavelets[J]. 1999: 126–133.
- [6] BERNAD D. Finding patterns in time series: a dynamic programming approach[J]. Advances in knowledge discovery and data mining, 1996.
- [7] YANAGISAWA Y, AKAHANI J.-I, SATOH T. Shape-based similarity query for trajectory of mobile objects[J]. 2003: 63–77.
- [8] CAI Y, NG R. Indexing spatio-temporal trajectories with Chebyshev polynomials[J]. 2004: 599–610.
- [9] VLACHOS M, GUNOPULOS D, DAS G. Rotation invariant distance measures for trajectories[J]. 2004: 707–712.
- [10] SAKURAI Y, YOSHIKAWA M, FALOUTSOS C. FTW: fast similarity search under the time warping distance[J]. 2005: 326–337.
- [11] ALT H, GODAU M. Computing the Fréchet distance between two polygonal curves[J]. International Journal of Computational Geometry & Applications, 1995, 5(01n02): 75–91.
- [12] AGARWAL P K, AVRAHAM R B, KAPLAN H, et al. Computing the discrete Fréchet distance in subquadratic time[J]. SIAM Journal on Computing, 2014, 43(2): 429–449.
- [13] RABINER L R, JUANG B.-H. Fundamentals of speech recognition[J]. 1993.
- [14] VLACHOS M, KOLLIOS G, GUNOPULOS D. Discovering similar multidimensional trajectories[J]. 2002: 673–684.
- [15] CHEN L, ÖZSU M T, ORIA V. Robust and fast similarity search for moving object trajectories[J]. 2005: 491–502.
- [16] RANU S, DEEPAK P, TELANG A D, et al. Indexing and matching trajectories under inconsistent sampling rates[J]. 2015: 999–1010.
- [17] CHEN L, NG R. On the marriage of l_p -norms and edit distance[J]. 2004: 792–803.
- [18] FRENTZOS E, GRATSIAS K, THEODORIDIS Y. Index-based most similar trajectory search[J]. 2007: 816–825.

- [19] SANKARARAMAN S, AGARWAL P K, MØLHAVE T, et al. Model-driven matching and segmentation of trajectories[J]. 2013: 234–243.
- [20] DURBIN R, EDDY S R, KROGH A, et al. Biological sequence analysis: probabilistic models of proteins and nucleic acids[J]. 1998.
- [21] MORSE M D, PATEL J M. An efficient and accurate method for evaluating time series similarity[J]. 2007: 569–580.
- [22] Jinja2 and WSGI. Website. <http://werkzeug.pocoo.org/docs/0.12/> <http://jinja.pocoo.org/docs/2.9/>.
- [23] DEAN J, GHEMWAT S. MapReduce: simplified data processing on large clusters[J]. Communications of the ACM, 2008, 51(1): 107–113.
- [24] ZAHARIA M, CHOWDHURY M, FRANKLIN M J, et al. Spark: Cluster Computing with Working Sets.[J]. HotCloud, 2010, 10(10-10): 95.
- [25] Cluster Mode Overview. <http://spark.apache.org/docs/latest/cluster-overview.html#cluster-manager-types>.
- [26] HONGZHI W, BELHASSENA A. Parallel Trajectory Search Based on Distributed Index[J]. Information Sciences, 2017.
- [27] VISVALINGAM M, WHYATT J D. The Douglas-Peucker Algorithm for Line Simplification: Re-evaluation through Visualization[J]. 1990, 9(3): 213–225.
- [28] SELLIS T, ROUSSOPOULOS N, FALOUTSOS C. The R+-Tree: A Dynamic Index for Multi-Dimensional Objects.[R]. 1987.
- [29] Rtree: Spatial indexing for Python. <http://toblerity.org/rtree/>.
- [30] CHEN Y, JIANG K, ZHENG Y, et al. Trajectory simplification method for location-based social networking services[J]. 2009: 33–40.
- [31] SHANG S, DING R, ZHENG K, et al. Personalized trajectory matching in spatial networks[J]. The VLDB Journal, 2014, 23(3): 449–468.
- [32] QI S, BOUROS P, SACHARIDIS D, et al. Efficient point-based trajectory search[J]. 2015: 179–196.
- [33] ROUSSOPOULOS N, KELLEY S, VINCENT F. Nearest neighbor queries[J]. 1995, 24(2): 71–79.
- [34] TANG L.-A, ZHENG Y, XIE X, et al. Retrieving k-nearest neighboring trajectories by a set of point locations[J]. 2011: 223–241.
- [35] AutoNavi Map Javascript Application Program Interface. <http://lbs.amap.com/api/javascript-api/summary/>.
- [36] TUW R P, FRAUNHOFER I. Software requirement specification[J]. 2002.
- [37] Networking Research Lab at Department of Computer Science, Engineering at Shanghai Jiao Tong University. <http://202.120.37.78:8080/nrl>.

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 AN EFFICIENT SIMILAR TRAJECTORY SEARCH ALGORITHM

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.