密级：_____

浙江大学

硕 士 学 位 论 文

论文题目 基于旅游文记的旅游景点推荐及行程路线规划系统

作者姓名 胡乔楠

指导教师 陈刚 教授

学科(专业) 计算机应用技术

所在学院 计算机科学与技术学院

提交日期 2015-05

A Dissertation Submitted to Zhejiang University for the Degree of Master of Engineering

TITLE: travelogue based tourist attractions recommending trip and route planning system

Author: Hu Qiaonan

Supervisor: Chen Gang

Subject: Computer Application Technology

College: Computer Science and Technology

Submitted Date: 2015-05

摘要

旅游业的火热以及各种旅游网站的兴起，使得旅游网站上的旅游文记迅速增长。本文设计了一个基于旅游文记的旅游景点推荐以及行程路线规划系统，不仅挖掘出大量旅游文记当中的旅游景点，还结合这些景点给游客提供旅游方面的服务。目前旅游网站上，对旅游文记的挖掘应用并不透彻,所提供的旅游服务也有欠缺,旅游景点和旅游路线旅游地图的结合并不密切。本文所设计的系统迎合了游客的需求，解决了旅游文记挖掘的应用问题。

在本文所设计的系统当中，设计并实现了名为 MutipleGuide 的旅游路线规划算法。该算法考虑了旅游景点满意度，旅游景点时间，以及旅游景点费用这三个因素，另外用户也可以选择不同的起点和终点进行路线规划。当有多个结果时，使用天际线算法进行最大域筛选。本算法更加人性化，得到的结果也更加多样化。该算法在运行时间和运行占用内存方面也十分高效，本文做了大量的实验进行对比，验证了该算法的高效性。

本文所设计的系统当中,还实现了其他三个功能,分别是热门旅游景点推荐，关联旅游景点推荐和旅游路线规划。本文详细讲解了本系统的整个设计框架，并展示了本系统的使用过程。从系统的使用来看，本系统可以很好的解决游客所遇到的景点行程以及路线的问题，证明了本系统的可用性和实用性。

关键词： 旅游文记，路线规划，景点推荐，本文挖掘

Abstract

With the development of tourism and the rise of various tourism website, the travelogue on the tourist sites grows rapidly. In this paper, we design a travelogue based tourist attractions recommending, travel trip and route planning system. This system not only digs out tourist attractions from a large number of travelogue, but also provide travel service to tourist with these attractions. On tourism website, the application of travelogue mining is not thorough, the travel services is not enough, and the combination of tourist attractions and tourist routes with tourist map is not close. The system in this paper meets the needs of tourists, and solve the application problems of travelogue mining.

The system designed in this paper, using a kind of travel route planning algorithm named MutipleGuide. This algorithm considers three factors of a route, the satisfaction, time and cost of tourist attractions. In addition, user can choose different start point and end point to do route planning. When multiple results run out by this algorithm, it can use skyline algorithm to get maximum field screening. This algorithm is more humane, and the result is more diversified. This algorithm is very efficient in running time and memory. We has done a lot of comparative experiments to verify its high efficiency.

In the system designed by this paper, it has other three functions, which are popular attractions recommending, associated attractions recommending and travel route planning. This paper not only explains in detail the design of the whole framework of this system, but also show the whole use of it. From the use of the system, it can well solve the problems of attractions recommending, travel trip and route planning which is usually encountered by tourists, and prove the availability and practicability of the system.

Keywords： travelogue, trip planning, attractions recommending, text mining

目录

图目录

表目录

第1章 绪论

1.1 课题背景和研究意义

目前旅游业蓬勃发展，在经济增长和拉动就业方面的作用日益受到国家的重视。计算机数据挖掘技术的发展，使得其能解决旅游业在发展过程中的很多问题，给大家带来更多便利。随着生活水平的提高，越来越多的人们走到世界各地去旅游。由此催生了一大批旅游相关的网站，国外的网站有 Spaces，Travelpod，IgoUgo,Travel-Blog 等，国内的有携程网，去哪儿等。在这些旅游网站上，人们不仅可以查看别人的旅游文记，预览对应风景的图片。自己也可以发布旅游文记，上传照片，供后面的人进行参考查阅。现在旅游网站上的旅游相关的数据数以亿计，根据专业旅游网站 travelpod 统计，仅在一周内人们就在其网站上分享了约八万条旅游经验。

如此大量的数据引发了新的问题，游客在查阅旅游网站时不得不花大量的时间去阅读旅游文记，从而筛选出满意的旅游景点或者旅游路线。传统的搜索引擎能够在一定程度上帮助游客进行信息筛选，但是搜索引擎提供的信息一般比较零散，信息之间没有很强的关联性。另外，对于行程安排，最优路线选择等问题，又是旅游业和计算机结合所形成的新问题。以一位游客第一次来到夏威夷游玩为例，一般游客会遇到如下四个问题：

1：来到夏威夷，游客肯定很想知道这个地方最热门的景点有哪些。另外，夏威夷是一个洲，游客还想了解，到夏威夷洲下面每一个郡当中的热门景点。如考爱郡（Kauai County）的热门景点是哪些。

2：游客在了解景点的过程中，发现某个景点十分吸引他（她），但这个景点并不是热门景点，了解的人不像热门景点那么多。于是他（她）很想问有没有和这个景点相关联的景点，从中找出他（她）喜欢的。

3：当游客选择了一些景点如夏威夷火山公园，冒纳基火山，普纳撸海滩等景点之后，游客便希望在几天之内游览完这些景点，如何安排景点才能够节省时间而不用走很多弯路。这个就是旅游行程规划的问题。

4：当某一天，游客来到夏威夷大岛，岛上有很多景点，但是游客时间只有四个小时，并且游客的有一定费用预算。如何在这四个小时内，并且在一定费用的前提下，选择一条最优的路线，让游客获得最大的满意度。

从目前旅游网站的应用来看，大多只进行了景点推荐，并且仅仅是热门景点推荐，也就是游客遇到的问题 1。而相关联的景点推荐涉及很少，行程规划也没有特定的算法来实现，都是让游客自己根据前人的经验去进行选择和安排。关于路线规划算法，由于这个问题本身是 NP 难问题，并且涉及到时间、费用和满意度等因素，尽管有一些研究，本文未发现使用这个算法来帮助游客的网站。

旅游业的一个特点就是地理位置相关性，就是每一个景点和路线都可以在地图上准确找出来。而目前很多旅游网站和地图结合的并不紧密。让游客没法提前或者在旅游完之后了解自己游玩的景点在地图上的分布情况，无法给游客直观的体验。表 1.1 就显示了国内外热门旅游网站，在旅游文记和地图的结合情况。只有 travelPod 在用户游玩之后可结合自己的旅游文记在地图上当中标明旅游的地点和路线，这也是事后标明。在现有技术背景下，完全可以对现有的旅游文记进行挖掘。让游客在游玩之前，就能够在地图上得到旅游景点和路线的直观印象。

表 1.1 国内外旅游网站旅游文记可视化情况

网站名称	乐途网	同程网	携程网	travelPod	travelBlog
关于地点的地图	无	无	有	有	无
整个线路地图	无	无	无	有	无

为了解决上述问题，并给游客带来便利，首先必须对旅游文记进行挖掘，提取文记当中的旅游景点并找出旅游景点之间的关联关系。这需要使用文本挖掘技术，并且还需要有详细的地理名词作为帮助，准确无误提取出旅游景点的地理和行政领域等信息。关于旅游路线规划，需要选择合适的算法来获得节省时间的行程安排。另外一个难点就是路线规划问题，在景点满意度，景点费用，旅游时间的限制下，如何从这么多景点当中选择出不仅是合适并且最优的一系列景点，组成一条路线，供游客参考。由于路线规划问题是 NP 难问题，如何才能够根据旅游路线规划的特点，在理想的时间内计算并得到结果，即便在景点较多时，算法仍然能有效计算，这又是另外一个挑战。在应用方面，需要把景点，行程以及路线，和地图结合起来。在选择景点的同时，就能在地图上看到其位置。在规划行程和路线的同时，也能够在地图上看到行程的分布和路线的走向。

经过这些年的技术发展，有些问题已经得到解决。已经有研究能够很好解决旅游当中的部分问题，并在商业当中应用。当本人仔细研究了这些研究成果之后，就发现其中仍然有很多不足之处。如真实旅游数据的匮乏导致景点的推荐不够真实有效，文本挖掘不够充分导致景点之间的关联性不强，很多研究只有在单一方面或景点或路线等上进行研究，没有融合各种功能于一体的系统。更重要的是，路线规划由于其问题本身的难度和复杂性，一直存在进一步进行研究的价值。

1.2 本文工作与贡献

传统的旅游景点推荐，一般是根据景点的热门程度或者推荐算法进行个性化推荐，而考虑层级热门景点推荐的研究很少。在旅游路线规划方面，关于两个不同点之间，考虑时间和费用两个因素的算法仍然是个空缺。而旅游景点的满意度的评判没有固定标准，因此本文以热门程度加上纠正因子来作为满意度分数，并重点考虑路线规划算法的设计与实现。本人所设计的系统弥补了层级热门旅游景点推荐和多因素旅游路线规划研究的不足，同时该系统融合了从景点到路线一体的功能，迎合游客的需要。

为了得到真实的数据，本文从外文旅游网站上爬取了八万篇旅游文记作为数据基础，并使用文本挖掘和地理名词提取技术，获取了这八万篇旅游文记当中有关的地理名词和景点名词。为了解决地理和景点名词提取的误差，本文还加入了自然语言的一些规则来提高准确率和召回率。经过筛选，从八万篇旅游文记当中获得了一百万个地理名词，作为景点推荐的基本数据。为了能够方便把新的旅游文记及其地理名词加入到后台数据库当中，本文搭建了一个系统来进行后台数据处理。有关后台数据库设计会在 5.2 节当中详细介绍。基于获得的景点名词，本文使用了地理本体树来进行不同政治层级的旅游景点推荐，使用关联规则算法来进行关联景点推荐。使用最小生成树聚类算法来进行旅游行程规划，并提出一种新的旅游路线规划算法。本文利用 J2EE 技术，设计一个 B/S 模式的系统。这个系统融合了上述算法，提供了有关旅游的多层次服务。本文的工作和贡献主要有如下四点：

（1） 设计并实现了一个基于旅游文记旅游景点及行程路线规划系统。该系统和谷歌地图实时交互，让用户直观看到景点和行程在地图上的分布。

（2） 设计了一种起点到终点的旅游路线规划算法， 这种路线规划算法考虑了时间，费用，满意度三个因素。

（3） 本文做了大量的横向和纵向对比实验，在景点个数，时间，以及费用三个因素变化较大时，仍然验证了本文提出算法在时间和内存方面的高效性。

（4） 根据旅游文记的景点，实现了地理本体树，关联规则挖掘算法，最小生成树聚类算法和 MutipleGuide 算法来实现景点推荐，行程路线规划功能，并用实验证实了算法和系统的有效性。

1.3 本文组织

本文总共分为六章，每一章内容如下：

第一章介绍本文工作的背景和课题意义，介绍了本文工作的必要性和贡献。

第二章介绍相关工作和技术，主要是一些已有的技术和研究，作为整个系统

又是必须用到的，并且是必不可少的一部分功能。

第三章介绍结合旅游文记和地理名词，如何实现旅游景点推荐和旅游行程规划，展示了实验结果，验证了所用算法的有效性。

第四章重点讲解本文提出的旅游路线规划算法 MutipleGuide 算法，从这个算法的问题定义，到实现过程，以及最后用了大量的实验从各个方面来验证本算法的高效性。

第五章展示了本文所设计的旅游景点推荐和旅游行程路线规划系统，讲解了本文所设计系统的功能以及系统的使用情况。

第六章总结了本文的工作，并对未来研究进行展望。

1.4 本章小结

本章总结了本文的课题背景和研究意义，本阐明了本文的工作与贡献，给出了本文的组织结构。

第2章 相关工作和技术研究

2.1 旅游文记挖掘和旅游景点推荐

这一节介绍旅游文记挖掘和旅游景点推荐方面的研究

2.1.1 旅游文记挖掘相关研究

在[1]当中，主要抓住与旅游有关的多媒体信息，比如照片，视频，还有照片当中包含的地理位置，如 GPS 坐标，坐标的序列。把这些图片，地理坐标，时间上的因素和景点的序列等，综合在一起进行数据挖掘，得到大众的集体旅游智慧。通过这些编成一个个旅游景点。尹华罡研究海量时空数据的内容[2]，从海量图片数据入手，提出了一种快速挖掘用户的线路的算法，并用搜索引擎对线路建立索引，挖掘出用户的信息，从而给用户推荐个性化的路线。照片上通常会生成照片拍摄时的准确位置（GPS）和时间，用数据挖掘来增强路线从而提高用户体验。最后构成一个基于用户历史信息的个性化路线推荐算法，设计并实现了一种交互式的规划系统 Photo2Trip。关于数字图片打标签，有专门的文章[3]讲解如何做到。这是一个端到端的系统，很多图片通过 GIS 拍摄，并通过这个系统打标签。在海量图片的推荐方面，[4]主要使用 Flickr,ImageNet/WordNet,ConceptNet 三个内容来给图片生成推荐算法。把图片和知识库链接起来，形成一个知识网络，效果符合人的主观理解。

佘新伟[5]主要是为游客提供旅游推荐服务，比如准确的旅游信息，旅游行程规划。该论文使用了 J2EE 当中的一些主流技术，比如 Structs, Sprint,Hibernate。该论文使 1200 条模拟的游客旅游日志作为数据源。Yin H[6]等人描述一个交互式的旅行规划系统，名叫“Photo2Trip”。这个系统能够通过旅行地点（北京，巴黎，纽约），旅行时间（夏天，冬天，月份）以及旅行的类型喜好（喜欢风光景点，历史景点），来给用户推荐多个目的地之间的旅游路线，以及推荐旅游景点内部的线路。用户还可以通过添加或者减少景点来得到个性化的路线。

Xin Lu[7]等人还写了一篇短文，同样描述类似的工作，这篇文章偏重描述系统。除此之外，[8]中把旅游文记可视化，可视化的方法是在提取旅游文记当中的地名词之后，再给地名词添加对应的图片。这样，一篇旅游文记一般都会包含若干个地理名词，那么与这些地理名词有关的风景图片，就会组成一个可视化的“旅游文记”。作者的主要贡献是能够很好的度量地理名词和图片之间的相似性。在[9]当中，使用了协同过滤算法，特点是能够挖掘每个文章的置信度。

2.1.2 景点推荐相关研究

给用户推荐旅游景点，通常有两种类型，热门旅游景点推荐和个性化旅游景点推荐。热门的景点，通常是某一个景点在某一个区域或者时间段十分有名，属于这个某个地区代表性的景点。比如，去北京就要去万里长城和天安门广场，去上海就要去看东方明珠，到纽约就要去看自由女神像。当游客有了热门景点之后，就可以在一个陌生的城市安排自己的将要玩的旅游景点列表。对于第二种类型，个性化旅游景点推荐，这个需要考虑用户个性化的因素。既可以通过用户个人的喜好，也可以通过用户过去的历史景点信息来进行推荐。

研究旅游景点推荐的文章已经有很多，[10]这篇文章总结了 09 年之前有关旅游推荐系统的一些系统和技术，包括手持设备和台式设备系统。从中可以看出，已经有大量的文章使用协同过滤技术在手持设备和台式设备上进行旅游景点推荐。[11]利用贝叶斯网络和层次分析法来给用户提供在一个陌生旅游地的个性化推荐服务，该推荐主要考虑被推荐用户以及其他用户的旅游行为，属于基于用户的旅游行为推荐技术。该文根据旅游网站上有关旅游景点的信息，构建旅游本体树来存储一个旅游景点的地点，时间，费用等属性，然后构建贝叶斯网络来评估用户对这些属性的偏好，从而达到旅游景点推荐的目的。[12]根据用户所产生的 GPS 轨迹来进行热门旅游景点以及个性化旅游景点推荐。[13]根据伪用户对旅游及观点的投票，设计了一个增强的协同过滤系统，能够在移动环境下给用户提供个性化的 PIO(point of interest)推荐。该系统以推荐饭馆为例，结果表明该系统能够满足移动环境下的可扩展性，低延时性以及便利性三个要

求。[13]Geowhiz[14]CityVoyager,[15][16][17] 使用用户在现实生活当中所留下来的历史位置的记录来推荐这些文章，其主要给用户推荐一些有特征的地理相关事务，如饭馆或者士多店。[14]主要给用户推荐商店，其利用用户历史地理位置信息来评估用户的个性化喜好。[16]从用户的地理位置历史记录以及地理空间的层次化属性当中挖掘出人所在地理空间上的相似度。接下来[17]在其以用户为中心的 CF 模型中使用了这个相似度，并进行个性化的朋友和地点推荐。[18]根据 GPS 和用户的评论使用协同过滤算法来进行一个活动到地点的推荐。比如，用户给出一项行为如购物，这个系统会推荐最好的 k 个可以购物的地方。相反，如果用户给出地点北京奥林匹克公园，那么这个系统会给出这个区域当中最活跃的 k 个地点。

本文从旅游文记当中挖掘出旅游景点有关的信息，给予区域性的热门旅游景点推荐。利用数据挖掘技术，给予用户相关联的热门旅游景点推荐。并结合用户和旅游文记的关系，利用协同过滤技术，给予用户个性化的旅游景点推荐。

2.2 关联规则挖掘和聚类算法

2.2.1 关联规则算法介绍

关联规则分析是用来在大型数据库当中发现变量之间有用联系的方法。它使用不同的度量规则来发现数据库当中变量之间的强联系。关联规则一般用最小支持度和最置信度来进行筛选，满足一定阈值的事物会被筛选出来。进行关联规则分析的首要步骤是进行频繁项挖掘。经典的频繁项挖掘算法有 Apriori 算法[19]。是用来挖掘数据库当中的频繁项集合的算法，该算法自底向上进行挖掘，首先挖掘出包含项数少的频繁项集合，然后逐步扩充，直到所有项数的频繁项都被挖掘出来。为了提高 Apriori 算法的效率，有五种变形，分别是基于散列技术，基于事务压缩技术，基于划分技术，基于抽样方法和动态项集技术,这些都是类 Apriori 算法。为了解决 Apriori 产生候选集合的不足，一些不同于基于 Apriori 方法的算法提出，挖掘频繁项集的模式增长方法 FP-growth 算法[20]该算法第一次扫描数据库后获得项数频度关联，在进行第二次数据库扫

描时，就把数据库当中项的频集压缩成一颗频繁模式树，再利用这棵树来进行频繁模式挖掘。 FP-growth 算法显著地降低了搜索开销，其效率大约比 Apriori 快一个数量级。非类 Apriori 算法有 H-mine[21]和 Opportune Project[22]，该算法利用一种超链接数据结构来来进行频繁项挖掘，其在内存和速度上相比 Apriori 有明显优势。Opportune Project 算法。上述算法在时间效率和空间效率比 Apriori 算法提高了很多。当最小支持度阈值比较低时，会产生大量的频繁项集合，使用上述算法会产生频繁项挖掘风暴。为了解决这个问题，[23]提出了 A-Close 的频繁闭集挖掘算法。Jian Pei,Jiawei Han, 和 Runying Mao[24]提出了 CLOSET 算法这个算法特点在于使用了 FP-tree 的数据结构，前缀路径压缩技术和基于划分的投射机制。[25]综合了前任所提出的有效策略和新开发的如混合投影和项跳过技术。本文将使用这种综合的方法对频繁项进行低阈值的挖掘。

表 2.1 事务 id 和其对应的项目

事务 ID	项目	事务 ID	项目
T1	I3，I4	T6	I4，I5
T2	I1，I4	T7	I1，I2，I3
T3	I2，I3，I5	T8	I2，I5
T4	I1，I4，I6	T9	I3，I4
T5	I2，I5，I6	T10	I4，I5

2.2.2 关联规则挖掘基本概念

（1）项集(ItemSet)，$\tau = \{I_1, I_2, I_3\}$是项的集合，每个事务 T 都是一个非空项集，并且$T \in \tau$。每一个事务都有一个标示符，称为 TID。例如表 2.1 当中$\tau = \{I_1, I_2, I_3, I_4, I_5, I_6\}$,T 总共有 10 个，分别标记为 T1-T10。在本文当中，τ代表一个地理名词集合，Ti 代表一篇文档，而 Ii 代表一个地理名词。

（2）关联规则（Association Rule），有事务 A 和 B，关联规则是$A \to B$的蕴含式，并且$A \neq \emptyset$，$B \neq \emptyset$ 。例如设事务 A={I3}，B={I4}，$A \to B$是一个关联规则。

（3）支持度和置信度（Support and confidence）。有关联规则$A \to B$在事物集 D 中成立，并且有支持度 sup，置信度 conf。若 sup 指相对支持度，那么 sup 就是事物 D 中包含 A 并 B 的百分比。若 sup 指绝对支持度，sup 就是事务 D 中 A 并 B 的频度。置信度就是 D 当中包含 A 的事务当中同时也包含 B 的事务的百分比。例如表 2.1 当中，设事务 A={I3}，B={I4}，$A \to B$的绝对 sup 就是 A 和 B 同时出现次数为 2，相对 sup 就是出现的次数除以总的总的事务数，相对 sup($A \to B$)=2/10=20%。$A \to B$的置信度就是 A 出现的次数 4，这 4 次当中，只有 2 次出现了 B，那么 conf($A \to B$)=2/4=50%。

（4）频繁项集（frequent itemset)。如果项集 A 的最小支持度满足预定的最小支持度阈值(min_sup)，那么 A 是频繁项集。频繁 k 项集通常记为 Lk。例如表 2.1 当中，设最小支持度阈值为 2，设项 A={I3，I4})，B=({I5,I6}),$sup(A) = 2 \geq 2$，$sup(B) = 1 < 2$。那么项 A 就是频繁项，项 B 不是频繁项

（5）闭频繁项集(closed frequent itemset)。项集 A 在数据集 D 中是闭的（closed），如果不存在真超集 B 使得 B 与 A 在 D 中具有相同的支持度计数。如果 A 既是闭的，又是频繁项集，那么 A 就是闭频繁项集。例如在表 2.1 中，设最小支持度阈值为 2，有项 A={I3，I4}。项 A 就是闭频繁项集。因为 $sup(A) = 2 \geq 2$，并且不存在 A 的真超集 B，使得 B 和 A 具有相同的支持度计数。

设 C 是数据库 D 中满足最小支持度阈值 min_sup 的闭频繁项集的集合，那么 C 和他的计数信息可以用来导出频繁项集的完整集合，也就是说，闭频繁项集包含了频繁项的完整信息。举一个例子，假如事务数据库当中有两个事务{{I1，I2，…,I50}，{I1,I2,…,I100}}，设最小支持度是 1。那么这个数据库当中的频繁闭项集的集合就是{{I1，I2，…,I50}:2,{I1,I2，…，I100}：1}，那么从

这个集合当中可以推导出 sup({I1，I45})=2，sup({I28,I88})=1 等等，可以看出所有的频繁项都可以由这个集合导出。

先验性质（Apriori property）：频繁项集的所有非空子集也一定是频繁的。根据频繁项集的定义，如果项集 I 不是频繁项集，也就是 s(I)< min_sup。如果把任意一个项集加入到 I 当中，使 I’=I+A，那么 I’不可能比 I 获得更大的支持度，那么$s(I) < \min_sup$。

反单调性(antimonotone)：如果一个集合不是频繁项，那么它的所有超集也都是不频繁的。

2.3 旅游行程规划和 skyline 算法

2.3.1 旅游行程规划算法

比如游客选择了杭州的十个景点，分别是断桥，雷峰塔，虎跑，动物园，灵隐寺，九溪，城隍阁，西溪湿地，钱塘江大桥，宋城，游客想要在三天内游玩这些景点，下面就是如何安排每天行程的问题，让每一天的景点都比较合理。或者游客每天不想走太远，只想在景点之间相距只有 8KM 的范围内活动，如何确定玩几天，每天玩哪些景点才能够比较好的满足游客的要求。本文应用 MST 算法能够根据用户的需求很好的解决这一问题。

抽象描述，行程规划问题就是解决某一些点的聚类问题。传统的解决聚类问题的算法有 K-means[26]和 EM[27]等，这类算法的优点是结构简单，但是在处理复杂簇问题时，容易陷入局部解。而谱图聚类能够很好的解决复杂簇的问题。使用谱图聚类解决问题时，首先需要对数据进行如下抽象。把一个个数据描述成一张图当中的一个个点，数据和数据之间的关系抽象成图中点和点之间的边。然后根据某个优化目标，使用谱图聚类算法把这张图切割成若干个互相不连通的子图，每一个子图叫做一个簇。现在谱图聚类算法已经有很多，如离线谱图聚类[28,29]方法和增量谱图聚类[30,31]方法。本文使用最小生成树聚类算法[32,33]，能够根据聚类个数要求，进行聚类的划分，同样可以根据点与点之间边的距离，进行聚类的划分。

2.3.2 最小生成树聚类算法

很多场景下都会使用最小生成树(MST)聚类算法，MST 聚类算法有其自身的优越性，处理数据不收形状和维度的影响。MST 聚类算法首先需要计算出最小生成树。对于一个带权边的无向图，如何选择一棵生成树，使得这棵树所有边的权值和最小，这课树就是最小生成树。最小生成树的算法有普利姆和克鲁斯卡尔，这两个算法的复杂度分别是$O(v^2)$和$O(elog(e))$。本文采用克鲁斯卡尔算法。最小生成树算法已经很成熟，就不介绍，描述最小生成树聚类的算法的步骤。下面先描述按照天数来聚类的算法，设旅程的天数为 n。

(1)获取行程景点的数据，构造全连接图。

(2)利用克鲁斯卡尔算法计算出最小生成树

(3)把最小生成树剩余的边按照其权值从大到小排序，然后由大到小切断 n-1 条边。

(4)返回 n 个子图。

若按照景点之间距离的长短来进行行程规划，算法的前两步和按照天数的聚类算法都是相同的，只有最后两步不同。最后两步算法如下，设最远距离为 m：

(3)把最小生成树剩余的边按照其程度从大到小排序，切断两点之间距离大于 m 边，行程 k 个子图。

(4) 返回 k 个子图。

举一个例子，如图 2.1 是一个包含 7 个景点的旅游地图，每个景点编号分别是 A，B，C，D，E，F，G，点之间的连线代表两个景点之间有通路，边的权值代表两个景点之间的距离。

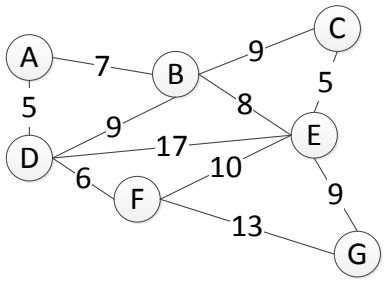

图 2.1 一张包含 7 个景点的旅游地图

下面一位游客想在三天内游玩这七个景点，如何安排。另外一位游客希望旅行的时候，在景点之间走的路程不超过 6KM。算法首先根据这张图生成一棵最小生成树，如图 2.2 当中的 MSTTree。接下来，根据第一位游客的需求，把边按照由大到小的顺序切断，首先切断景点 E 和景点 G 之间的边，行程两个子图，接下来切断 B 和 E 之间的边，行程三个景点。那么这三天的行程安排如下，第一天游玩 G 这个景点，第二天游玩 C 和 E 景点，第三天游玩 A,B,D,F 景点。从结果上来看，能够把距离远的景点和相聚较近的景点分开，结果令人满意。根据另外一位游客的需求，则把最小生成树当中边的值超过 6 的全部切断，结果如图 2.2 当中的第三幅图。具体过程和之前类似，也达到了理想效果。

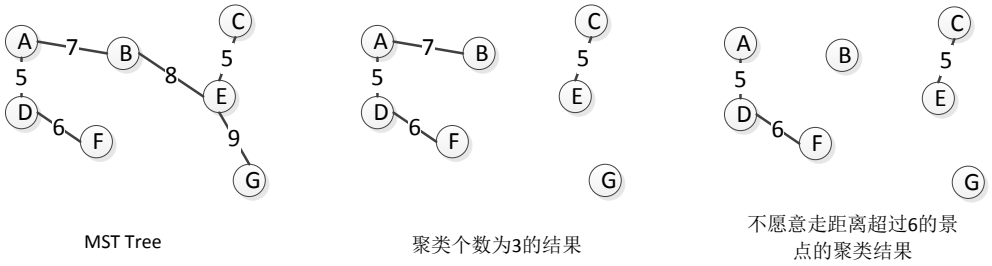

图 2.2 MSTTree 以及两种策略得到的行程规划结果

Skyline 查询问题，是在多变量限制条件下进行最优值求解的算法[34]。假设 V 是 k 维空间的一个向量集合，这个 k 维空间是$U_1 * U_2 * \ldots * U_k$。如何在向量集合 V 当中寻找一组最大的集合解 r，使得在 r 当中，任何一个向量不被另外一个向量所控制。在集合 V 当中而不再集合 r 当中的向量，都被集合 r 当中的向量所控制。一个 k 维空间当中，一个向量v_1控制另外一个向量v_2的含义是，v_1在任何一维 i 上都不差于v_2上对应维度的值，而v_1在至少一个维度 j 上要优于v_2对应的维度。早在 1975 年，Kung 就对这个问题进行了研究[35]。当集合中的向量维度在 2 维或者 3 维时，skyline 算法的复杂度是$O(n log_2^n)$。当向量的维度大于三维时，复杂度上限为$O(n log_2^{n^{d-2}})$。

2.4 J2EE 技术和谷歌地图 API

J2EE（Java 2 Platform Enterprise Edition）技术是一套不同于传统技术体系的

技术架构，这套技术是为了快速便捷的搭建企业级解决方案。这套技术架构总共包含 13 种技术规范，如 JDBC(Java Database Connectivity)，JNDI(Java Name and Directory Interface)，EJB(Enterprise JavaBean)等。本文主要使用了其中的 JDBC，JSP（Java Servlet Page）,Java Servlet,EJB 这四种技术。JDBC 提供了 Java 访问数据库的统一方式，屏蔽了不同数据库访问之间的差异。JSP 在 HTML 代码当中嵌入了 Java 代码，客户端访问并提交了 JSP 页面之后，后台服务器把这个 JSP 请求转换成 Java 代码，处理完之后再返回。Java Servlet 是 Java 服务器端处理请求的类，它同样响应请求并返回结果，但是 Java Servlet 并不能作为页面显示。EJB 包括会话（session） bean，实体(entity) bean，和消息驱动（message-driven） bean，主要负责消息的传递，数据的存储等。

谷歌地图提供一组开放的 api 接口，通过在 HTML 代码当中嵌入谷歌地图的 JavaScript 代码，就能够获得谷歌地图提供的服务。谷歌地图提供的功能包括地图显示，地图的放大缩小，地点标记，显示线条和形状等。获得谷歌地图的使用许可需要在网上申请 Google 地图 API key。

2.5 本章小结

本章介绍了和本文研究有关的研究和技术，包括其他研究者使用关于旅游文记的一些工作，旅游景点推荐已有的一些研究，关联规则挖掘的研究以及基本概念，旅游行程规划的概念以及可使用的算法，旅游路线规划的研究情况，Skyline 算法的概念，J2EE 当中的一些关键技术，谷歌地图 API 的使用。

第3章 旅游景点推荐以及行程规划

3.1 基于地理本体树的热门旅游景点推荐

基于地理本体树的热门旅游景点推荐就是解决如下问题，把旅游景点按照一定的地理规则进行聚类，并对每个聚类当中的每个旅游景点按照其热门程度排序，把每个聚类当中最热门的旅游景点推荐给游客。本文从大量的旅游文记当中挖掘出旅游景点相关的地理名词，并通过地理名词辞典为地理名词精确匹配详细的地理空间属性。本文利用空间属性可以很好地构建地理本体树来帮助本文对地理相关数据进行分析。

3.1.1 地理本体树结构

地理本体树是由四元组构成的模型，包括地理概念，地理公理，地理关系和地理实例[36]。其中以地理概念和地理关系为主轴衍生出其他两个元素。本文当中地理概念就是一个个旅游景点，地理实例就是一个个地理名词，地理关系就是景点和景点之间的地理位置划分，这个地理位置最小到行政区域。本文从旅游文记当中挖掘出地理名词，并根据其空间关系和行政区域划分，构建了地理本体树。从如图 3.1 是一棵地理本体树。这棵树的根节点是 United State，第二层是洲级别的节点，图当中显示了 Hawaii 和 Florida 洲。Hawaii 洲下面总共有五个县，本文当中显示了 Hawaii 洲下面的三个县，并进一步显示了县当中的几个更小一级的 POI，这些地方很多都是旅游景点。根据这课地理本体树，游客可以方便的选择某一个小镇上的热门旅游景点，同样，也可以得到某一个郡或者某一个洲下面的热门旅游景点。

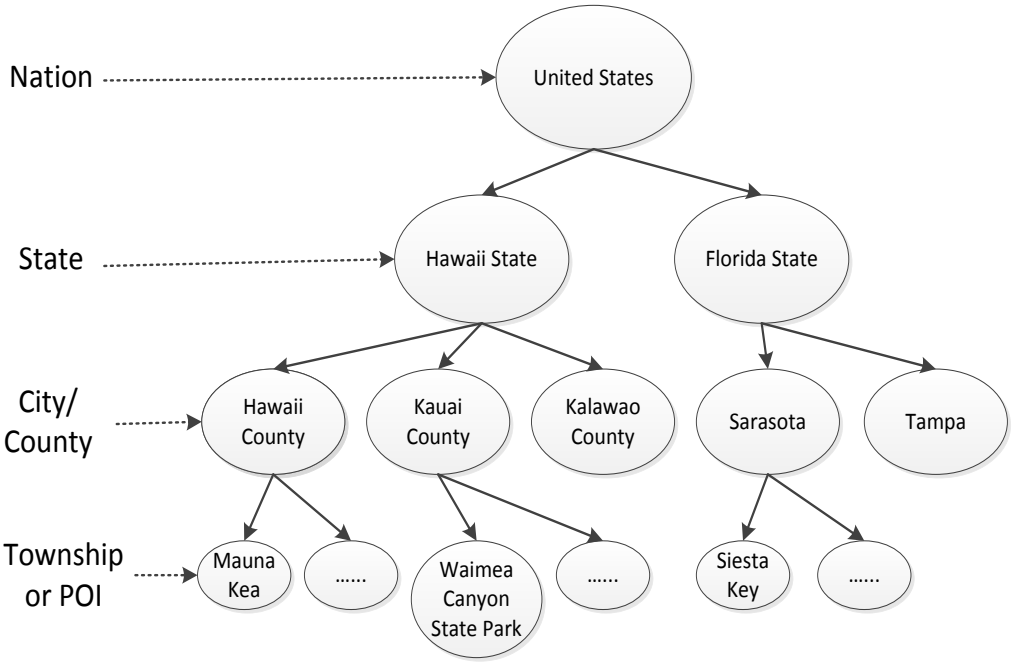

图 3.1 根据行政级别划分的地理本体树

3.1.2 基于地理本体树的推荐算法和数据结构

为了能够得到如图 3.1 的地理本体树，首先需要获得每一个景点的从国家一直到这个景点的全部名称，这个可以通过地理名词提取技术获取。然后就是根据全部地理名词的全部名称为本体树从根到叶子节点的路径来建树的过程。整个建树流程如图 3.2 所示。其中比较重要的数据结构就是每个本体树的节点，这个节点不仅需要存储当前节点的简单地理名词，当一个地理名词同时是一个地理实体的时候，还需要存储当前节点的全线地理名词。另外，这个节点需要记录当前地理名词出现的次数，也就是一个景点的热度。最后这个节点还要有指向下一个地理本体树节点的链接。在得到地理本体树之后，遍历地理本体树不同的层可以得到不同领域的热点地理名词。采用层次遍历法遍历最后一层可以得到每个郡的旅游景点，接着按照景点地理名词出现的次数排序，取前 k 个，可以得到某一个郡的 topk 热门景点推荐。同理，遍历倒数第二层，可以得到每个洲的热门 topk 热门景点。景点推荐的效果见 3.4 节。

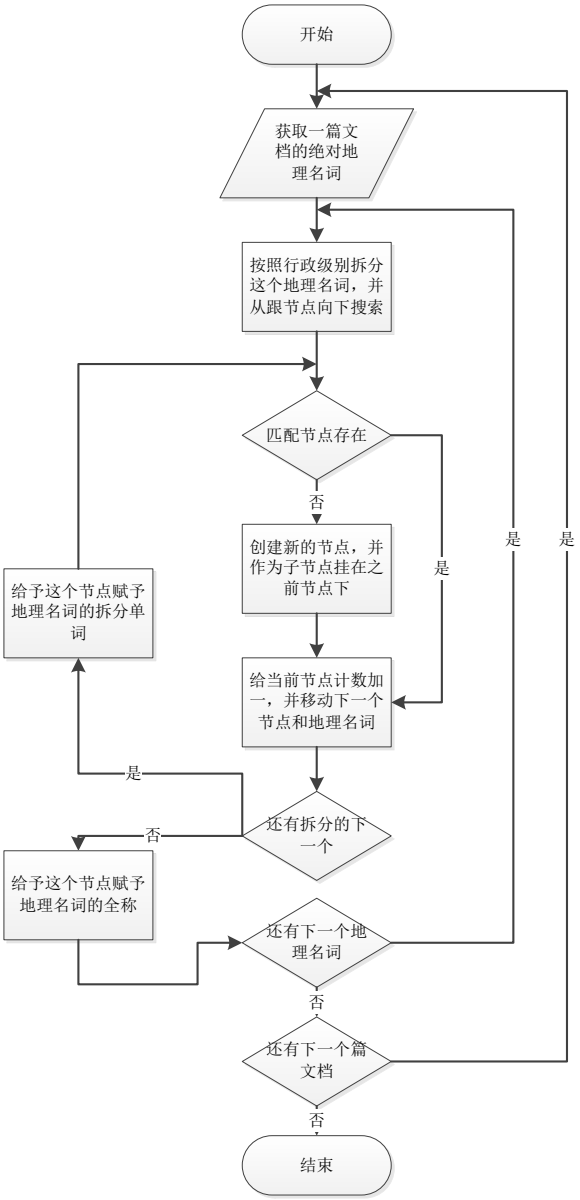

图 3.2 根据文档获取地理本体树的算法流程图

3.2 基于关联规则的热门旅游景点推荐

基于关联规则的热门旅游景点推荐算法，旨在解决如下问题，当游客获得了某些旅游景点并安排到自己的旅游行程单之后，游客希望系统能够根据自己选择的景点，自动的给他推荐另外一些相关联的旅游地点或者景点，从而扩展其

旅游景点的选择范围。本文利用关联规则挖掘算法，从大量旅游文记当中挖掘出一系列相关联的旅游景点集合推荐给用户。

3.2.1 FP-tree 构建和挖掘过程

频繁模式增长（Frequent-Pattern Growth）:将代表频繁项集的数据库压缩到一棵频繁模式树（Frequent-Pattern Tree），该树扔保留项集的关联信息。然后，把这种压缩后的数据库划分成一组条件数据库，每个数据库关联一个频繁项，并分别挖掘每个条件数据库。例如图 3.3 就是根据表 2.1 在设置最小支持度为 2 时，得到的一棵频繁模式(FP-tree)树,建树过程后面再详细讲解。

条件频繁模式树(conditioned Frequent-Pattern Tree)。通过 FP 树，根据表头挖掘出每一项的条件频繁模式树，也叫做条件数据库，最后根据这个条件 FP 树得到频繁模式。条件数据库的挖掘过程后面再详细讲解。

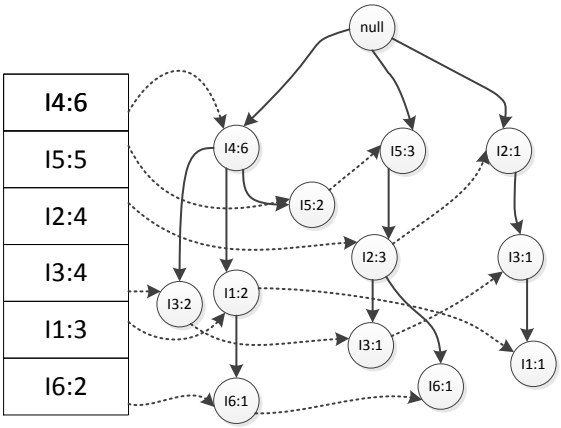

图 3.3 FP-tree

例如表 2.1 当中有如下 10 个事务，构建频繁模式树的过程如下：

（1） 第一遍扫描数据库，得到每个一项的支持度计数，并且按照他们的支持度计数递减排列。例如，根据表 2.1 的数据第一步得到如下排列的一项频繁项集合,如表 3.1。

表 3.1 一项频繁项

I1:3	I2:4	I3:4	I4:6	I5:5	I6:2

按照支持度递减排序后，就是如图 3.3 左边的表头当中每一项的排列。

（2）删除按照支持度计数递减的一项集合当中，计数小于最小支持度的项，只剩下一项频繁项。同样如图 3.3，由于最小支持度是 2，所有的一项都是频繁的，所以没有删除某一项。

（3）第二次扫描数据库，每个事务中的项都按 L 中的次序处理，并对每个新开头事务创建一个分支。对于已有的前缀路径，在每个频繁模式树的节点中给计数加一。

例如：第一项 T1={I3，I4}，按照次数大小的排序，就是{I4，I3},从根节点开始向下查找节点，I4 并不存在，那么新建节点 I4，节点的初始计数为 1。然后顺着 I4 向下找 I3，I3 不存在，于是创建新节点，节点计数同样为 1，并从 I4 指向 I3。第二项，T2={I1，I4}，按照次数大小的排序就是{I4,I1}，首先从根节点向下找，I4 存在，那么给 I4 的计数加一，并开始查找 I1。I1 不存在，那么新建节点 I1，I1 计数为 1。从 I4 指向新建节点 I1，第二项添加完毕。表 2.1 当中剩余的 8 项也是同样的添加过程。

（4）建立表头项来进行检索。表头项的大小为事务中一项频繁项的个数。对于每个新增加的节点都从表头项连接点的尾部添加一个链接指向这个节点。例如图 3.3 当中，当添加第一个事务 T1={I3，I4}时，I4 是新建的，那么表头节点当中的 I4 这一项添加一个连接指向 FP-tree 当中新建的节点 I4，I3 也是新建的，同样从表头节点当中的 I3 这一项添加一个连接指向新建的 I3。当第二次在 FP-tree 当中新建节点 I3 时，则从表头当中链表的尾部，也就是上一个被指向的 I3 节点，指向这个新建的 I3。图 3.3 中表头节点的箭头都用虚线表示。

从频繁模式树的构建算法可以看出，整个算法只需要两次扫描数据库。第一次扫描数据库获得一项频繁项，第二次扫描数据库，是为了根据排序的一项频繁项和数据库当中的事务来构建频繁模式树。之后进行频繁项挖掘的过程中，只需要扫描频繁模式树，相对于扫描整个数据库，极大的节省了算法的复杂度。

条件数据库的挖掘过程。

前缀路径：某一个节点的前缀路径，表示从 FP-tree 的根节点开始，一直到该节点的一条唯一路径（这条路径不包括根节点和当前节点）。

条件模式基：条件模式基是 FP-tree 某一个项出现的所有节点的前缀路径和，这个路径当中每一个节点的计数都等于当前节点的计数。

（1）从头表的尾部开始扫描表头项

（2）对于表头项所在链表当中的每一项，挖掘出其中的前缀路径，取出链表当中每一个节点的前缀路径，并根据链表上节点的计数设置前缀路径计数，构成条件模式基。

（3）对于每一项的条件模式基，构造这一项的条件模式树，这棵条件模式树包含了关于这一项的所有频繁项的信息。

表 3.3 FP-tree 挖掘过程

项	条件模式基	条件模式树
I6	{I4，I1：1}，{I5，I2:1}	null
I1	{I4:2},{I2，I3:1}	{I4:2}
I3	{I4:2}，{I5，I2:1},{I2:1}	{I4:2},{I2:2}
I2	{I5:3}	{I5:3}
I5	{I4:2}	{I4:2}

表 3.3 就解释了以表 2.1 和图 3.3 所示的数据挖掘频繁项的过程，解释如下，首先从头表尾部 I6 开始扫描，I6 的连接当中有两个节点，这两个节点的前缀路径分别是{I4，I1：1}，{I5，I2:1}，这两项构成条件模式基，根据这两个条件模式基，无法在条件模式树的根节点下添加新的分支，因为最小支持度是 2，因此，挖掘 I6 的条件模式树无法得到新的频繁项。接着扫描头表上的 I1，I1 的链表上有两个节点，其前缀路径分别是{I4:2}和{I2，I3:1}，满足最小支持度的只有{I4:2}，因此 I1 的条件模式树是{I4:2}，根据这棵条件模式树得到的频繁项{I4，I1:2}。接着扫描头表上的 I3，I3 的链表上有三个节点，其前缀路径分别是{I4:2}，{I5，I2:1}和{I2:1},由三条前缀路径导出的条件模式树有

两个分支，分别是{I4:2}和{I2:2}，根据这棵条件模式树得到的频繁项{I4，I3:2}和{I2，I3：2}。接着扫描头表上的 I2，I2 的链表上有两个节点，其前缀路径分别是{I5:3}和根节点 null，满足最小支持度的只有{I5:3}，因此 I2 的条件模式树是{I5:3}，根据这棵条件模式树得到的频繁项{I5，I2:3}。接着扫描头表上的 I5，I5 的链表上有两个节点，前缀路径分别是{I4:2}和根节点 null，满足最小支持度的只有{I4:2}，因此 I1 的条件模式树是{I4:2}，根据这棵条件模式树得到的频繁项{I4，I5:2}。I4 是表头的最后一项，总是靠根节点最近，不需要再挖掘。每一项的条件模式树如图 3.4。

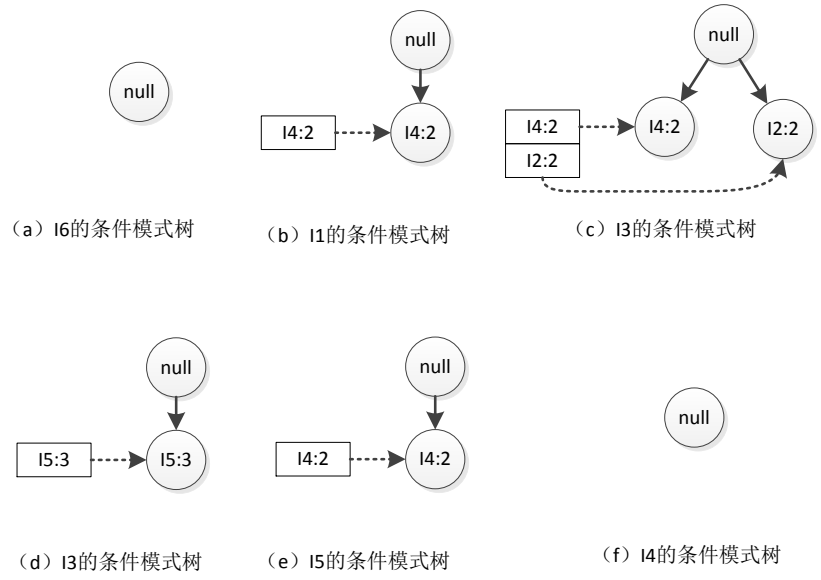

图 3.4 每一项的条件模式树

3.2.2 Closet+算法特点以及实现

Closet+算法是基于 FP-tree，采用深度优先搜索，水平数据存储格式，混合投影方法并加入了项合并子集剪枝，项跳过等优化策略的频繁闭项集挖掘算法。

在挖掘频繁项的过程当中可以采用深度优先搜索和广度优先搜索两种，广度优先搜索就是先搜索完大小为一的频繁项集合，然后再搜索大小为二的频繁项集合，依次得出最后的频繁项集合。之前在挖掘 FP-tree 时采用的是深度优先搜索，先搜索完和某一个项相关的所有频繁项，然后再搜索和另外一项相关的频繁项。

深度优先搜索适合事务当中项较多的数据库，并且在 FP-tree 上使用深度优先搜索具有更高的效率。Closet+算法采用深度优先搜索策略，结合条件模式树的特点，能够更加高效的挖掘包含较多项事务的数据库。

产生条件数据库的过程中，有两种投影方法，一种是自底向上的投影，另外一种是自顶向下进行投影。之前描述的都是自底向上的挖掘过程，而 CLOSET+采用混合投影的方法，即在数据稠密的数据集当中采用自底向上的投影方法，在数据稀疏的数据集当中采用自顶向下的投影方法。本文只采用自底向上的投影方法即可解决问题。

数据分垂直存储格式和水平存储格式。水平存储格式如表 2.1 当中描述，以文档的 id 为索引来检索一项事物当中的每一项,形式化表示就是{TID:itemset}。而数据同样可以用项-TID 集合的格式来存储，形式化表示就是{item：TIDset}，其中 item 就是每一个项的 id，TIDset 就是包含这个项的事务的集合,这种数据格式就是垂直存储格式。如表 3.2 就是把表 2.1 当中的数据转化成垂直格式。本文当中就是以地理名词为索引，指向包含这个地理名词的文档 ID 的集合。当数据量很大时，垂直数据格式需要为每一项维护的事务 ID 集合就非常大，并且需要产生的中间项事务 ID 集合长度也非常大，因此需要耗费很多内存空间。CLOSET+算法采用水平存储格式，在进行频繁模式树挖掘时就能节省大量的内存空间，因为不同的项 ID 可以共享前缀路径。

表 3.2 数据的垂直存储格式

项目 ID	事务	项目 ID	事务
I1	T2，T4，T7	I4	T1，T2，T4，T6，T9，T10
I2	T3，T5，T7，T8	I5	T3，T5，T6，T8，T10
I3	T1，T3，T7，T9	I6	T4，T5

项合并：如果包含频繁项集 A 的每个事务都包含项集 B，但不包含 B 的任何真超集，则 A 并 B 形成一个闭频繁项集，并且不必再搜索包含 A 但不包含 B 的任何项集。

例如，数据库当中有两个事务{{I1，I2，…,I50}，{I2,…,I100}，最小支持度是 2。当挖掘 I50 时，其前缀路径为 {I1,I2,…,I49:2},{I2,I3,…,I49:1}，可以发现，I50 的所有事务都包含 {I2,I3,…,I49}，但不包含{I2,I3,…,I49:1}的任何真超集。那么 { I2,I3,…,I49,I50：2}可以形成一个闭频繁项集，并且不必在搜索包含 I50 但不包含{I2,I3,…,I49}的集合。

子项集剪枝：如果频繁项集 A 是一个已经发现的闭频繁项集 B 的真子集，并且 sup(A)=sup(B)，则 A 和 A 在集合枚举树中的所有后代都不可能是闭频繁项集，因此可以剪枝。例如，数据库当中有两个事务{{I1，I2，…,I50}，{I2,…,I100}}，最小支持度是 2。在挖掘 I50 时，根据项合并优化，可以得到{I2,I3,…,I49,I50：2}是一个闭频繁项集。当挖掘 I49 时，发现 I49 是闭频繁项集{I2,I3,…,I49,I50}的真子集，那么就可以对 I49 进行剪枝。同理，可以对 I48，I47…I2 进行剪枝。这样，挖掘这个数据库的闭频繁项只需要挖掘 I50 之后，就停止了。

项跳过：在深度优先挖掘闭项集时，每一层都有一个与头表和投影数据库相关联的前缀项集 A。如果一个局部频繁项 p 在不同层的多个头表中都具有相同的支持度，则可以将 p 从较高层头表中剪裁掉。例如，考虑上面的例子，数据库当中有两个事务{{I1，I2，…,I50}，{I2,…,I100}},最小支持度是 2。在计算 I50 的投影数据库时，发现 I49 在 I50 的条件数据库当中和 I49 在全局数据库当中有相同的支持度，那么就可以跳过 I49 这一项。同理可以跳过 I48，I47，…，I2，只需要对 I50 进行条件 FP-Tree 之后，既可以停止了。

3.2.3 基于关联规则的旅游景点推荐算法

在上面几小节当中，详细介绍了 Closet+算法的原理和实现过程，这一节主要讲解如何利用 Closet+算法来实现旅游景点推荐，终点介绍整个旅游景点推荐算法相关的数据结构。

关联规则的挖掘分为两步，第一步是挖掘出频繁项，第二部是根据频繁项来导出不同项之间的关联关系。之前描述的有关 Closet+算法的过程和优化策略仅仅实现了第一步，就是挖掘出频繁项集合，而在本文当中就是利用 Closet+算法所生成的频繁项集合来实现旅游景点推荐。

为游客进行关联规则旅游景点推荐的过程分为三步，首先，就是数据准备。Closet+算法首先运行挖掘出当前旅游文集集合当中的闭频繁项集合，并存储到数据库当中。第二步，游客会选择或者输入几个旅游景点。第三步根据游客输入的景点名称，从数据库的频繁项集合当中选择包含这些景点的集合，并把其他的景点推荐给用户。

下面列出整个过程的一些重要数据结构

FPTreenode 类:存储 FP-tree 节点信息，其中包括项的 id(itemId)，节点的父亲节点链接(parent)，节点的兄弟链接(sibling)，节点的孩子信息(children)，该节点的计数次数(occurence)。

docWordTable：全局的事务数据表格，以文档的 ID 为键，文档当中单词的 id 集合为值的一个索引表格。

frequentOneSet:存储频繁一项集合的列表，也就是之前描述的头表，其中的元素是 FPTreenode，按照每个节点的 occurrence 由大到小排序。

frequentClosedSet:存储频繁闭项集合，有一个集合来存储 item 的 id，并统计其支持度。

frequentClosedItems：存储所有的频繁闭项集合，集合当中是一个频繁闭项集合。下面介绍基于关联规则的旅游景点推荐所使用的一些方法如表 3.3-3.6。

表 3.3 获取头表的方法

函数名	getFrequentOneSet
功能	获取 FP-Tree 当中作为索引的头表
参数	阈值 thr，整个文档单词表格 docWordTable
返回值	按照从支持度从大到小排序的一项频繁项列表

表 3.4 创建频繁模式树的方法

函数名	createFPTree
功能	建立频繁模式树
参数	整个文档单词表格 docWordTable 头表项 oneFrequentSet
返回值	频繁模式树的根节点

表 3.5 挖掘频繁闭项集合的方法

函数名	getFrequentClosedSet
功能	使用 Closet+挖掘频繁闭项集合
参数	阈值 thr，整个文档单词表格 docWordTable， 头表项 onFrequentSet
返回值	所有的频繁闭项集合列表

表 3.6 推荐旅游景点的方法

函数名	getAssociationList
功能	根据旅游景点来获取相关旅游景点
参数	旅游景点，频繁闭项集合
返回值	相关的旅游景点列表

3.3 旅游行程规划

3.3.1 最小生成树聚类算法的数据结构和功能函数

为了方便表达点的添加和边的添加，本文在实现当中把点和边各当成一个类来实现。重要的数据结构如下:

MapPoint，有属性 id 表示这个景点的唯一编号，latitude 这个景点的经度,longitude 这个景点的纬度,name 这个景点的名字，occurrence 这个景点出现的次数。

Edge,有两个 MapPoint 分别是起点和终点，w 这条边的权重。由于该图表达的是无向图，于是起点和终点没有区别。

Graph,包含一系列 MapPoint 集和边集 Edge 的一张图。

GraphSet，存储一系列连通分图。

下面介绍 MST 聚类算法的一些方法如表 3.7-3.9

表 3.7 最小生成树方法

函数名	KruskalMST
功能	用 Kruskal 算法获得最小生成树
参数	原始的图 Graph
返回值	一棵最小生成树

表 3.8 按照天数来聚类的方法

函数名	Cluster_LimitCount
功能	对最小生成树实现聚类
参数	最小生成树 Graph，天数 n
返回值	个数为 n 的连通子图

表 3.9 按照边的长度来聚类的方法

函数名	Cluster_LimitWeight
功能	对最小生成树实现聚类
参数	最小生成树 Graph，边的大长度
返回值	个数为 k 的连通子图

3.4 景点推荐和行程规划算法测试

前文当中描述了基于地理本体树的热门旅游推荐，基于关联规则挖掘算法的景点推荐和行程规划算法。在这一节当中，针对这三个算法给出相应的实验。本文从 travelBlog 和 travelPod 上爬取了八万多篇旅游文记，并提取了每一篇文档的地理名词。由于数据量比较大，本节以夏威夷洲的两千多篇文记为实验样本进行实验。本文的地理名词当中有经纬度信息，从而可以计算出两个景点之间的距离。由于本文是在地理名词的基础上进行景点的推荐，在地理名词当中，难免有些表达的不是景点或者表达为另外的方式，这个在景点名词提取算法改善后本算法的效果会同样改善。本文工作的实验环境为 MAC OS X 系统，处理器为 Intel Core i5 四核处理器，CPU 主频为 2.6GHZ，内存为 8GB。

3.4.1 基于地理本体树的热门旅游景点推荐

基于热门旅游景点推荐，首先提取每篇文章的地理名词，然后把这些地理名词按照地理本地树进行聚类，最后在每一个类别当中按照地理名词出现的次数由大到小排序。本节利用 3.1 节当中的地理本体树散发，首先展示 Hawaii 洲的五个郡当中排在前列的地理名称如表 3.10，然后经过筛选找出属于典型代表景点的地理名词，并在表格当中标出热度。

除了得到每个郡下面的热门景点外，还可以根据郡来统计每个郡的总体热门程度和平均热门程度。总体热门程度表示这个郡总体获得的频度，平均热门程度

代表这个地区的总体频度除以景点的个数。表 3.11 当中以 Hawaii 的五个郡为例，列举他们的总体热度和平均热度。

表 3.10 Hawaii 五个洲当中每个郡的热门景点

序号	景点名称	纬度	经度	词频
1	Hawaii，Hawaii County，Big Island	19.54943	-155.56228	506
2	Hawaii，Hawaii County，Hilo	19.72972	-155.09	276
3	Hawaii，Hawaii County，Kona	19.63820	-155.97860	184
4	Hawaii，Hawaii County，volcanoes national park	19.72972	-155.09	92
5	Hawaii，Hawaii County，Mauna Loa	19.47861	-155.60833	60
6	Hawaii，Hawaii County，Kailua Kona	19.64536	-155.99367	46
7	Hawaii，Hawaii County，Captain Cook	19.49694	-155.92166	46
8	Hawaii，Hawaii County，Waipio Valley	20.1225	-155.59361	25
1	Hawaii，Kauai County, Kauai	22.0110799	-159.70543	260
2	Hawaii，Kauai County, Kilauea	22.18916	-159.41333	65
3	Hawaii，Kauai County, Waimea Canyon State Park	21.99291	-159.66907	64
4	Hawaii，Kauai County, Waimea Bay	21.95156	-159.66761	41
5	Hawaii，Kauai County, Kalalau Trail	22.20027	-159.62027	36
6	Hawaii，Kauai County, Spouting Horn Park	21.88777	-159.49638	18
7	Hawaii，Kauai County, Hanalei bay	22.20889	-159.50709	15
1	Hawaii，Kalawao County, Catholic Church	21.18055	-156.95166	4
2	Hawaii，Kalawao County, Protestant Church, 11, 44	21.18111	-156.95194	1
3	Hawaii，Kalawao County, National Historic Landmark	21.19222	-156.98583	1
1	Hawaii, Honolulu County, Honolulu	21.30694	-157.85833	738
2	Hawaii，Honolulu County, Oahu	21.43333	-157.96666	551
3	Hawaii，Honolulu County, Waikiki beach	21.27861	-157.82833	263
4	Hawaii，Honolulu County, Pearl Harbor	21.35269	-157.96962	231
5	Hawaii，Honolulu County, Diamond head	21.26130	-157.80474	193
6	Hawaii，Honolulu County, Hanauma Bay	21.26873	-157.69311	130
7	Hawaii, Honolulu County, Polynesian	21.33638	-157.87388	111
8	Hawaii, Honolulu County, North Shore	21.58444	-158.10722	104

续表 3.10

序号	景点名称	纬度	经度	词频
9	Hawaii, Honolulu County, USS Arizona	21.36476	-157.94998	76
10	Hawaii, Honolulu County, Sunset Beach	21.667269	-158.04886	74
11	Hawaii,Honolulu County, Kailua	21.39666	-157.75430	71
1	Hawaii, Maui County, Maui	20.818987	-156.44260	744
2	Hawaii, Maui County, Lahaina	20.87833	-156.6825	181
3	Hawaii, Maui County, Mauna Kea	20.92277	-156.31083	141
4	Hawaii, Maui County, Haleakala	20.701406	156.17331	135
5	Hawaii, Maui County, Molokini	20.63174	-156.494064	69
6	Hawaii, Maui County, Kihei	20.78361	-156.46611	48
7	Hawaii, Maui County, Sandy beach	21.11480	-156.73788	27
8	Hawaii, Maui County, Samoa	20.71951	-155.98759	24
9	Hawaii, Maui County, Green Sand Beach,	20.91205	-156.69252	21

表 3.11 夏威夷洲下五个郡总体景点热门程度

	Hawaii County	Kauai County	Kalawao County	Honolulu County	Maui County
景点个数	700	274	6	1129	440
总体热度	3492	1459	14	6530	2648
平均热度	4.98	5.32	2.33	5.78	6.01

从总体上来看，Hawaii 的五个郡的热度都近似，除了 Kalawao 郡之外，其他的景点都有 5 左右的平均热度。五个郡的总体热度和他们的行政面积成正比，从表格当中的数据来看，选择 Honolulu 郡的游玩很多景点，并且每个景点的热度也排在前列。

3.4.2 基于关联规则的旅游景点推荐

基于关联规则的旅游景点推荐，就是利用 3.2 节当中所讲诉的频繁闭集挖掘算法，首先统计旅游文记当中每个景点出现的次数，然后根据阈值去除非频繁的景点，接下来使用 FP-Tree 数据结构来挖掘频繁闭项集合。本文在挖掘过程

当中，设置阈值为 3。根据 3.2 节的算法，首先计算出每一个单个景点的次数也就是支持度，然后从中去除不频繁的景点。表 3.12 中当中以夏威夷洲的旅游文记为数据，显示了部分达到支持度的景点。

表 3.12 达到阈值的一些景点名称

编号	名称	频度	编号	名称	频度
1	Hawaii	860	11	Lahaina	86
2	Honolulu	429	12	Hanauma Bay	82
3	Maui	330	13	Mauna Kea	72
4	Oahu	316	14	Haleakala	68
5	Waikiki beach	186	15	Haleakala	65
6	the North Shore	168	16	Volcanoes National Park	65
7	Diamond head	137	17	Sunset Beach	55
8	Kauai	135	18	USS Arizona Memorial	52
9	Pearl Harbor	126	19	Kailua Beach Park	45
10	Kona	114	20	Waimea	48

表 3.13 中显示了最终进行频繁闭项集合的挖掘结果，根据其中的一个景点，可以推荐另外的景点给用户。总共有一万多个频繁闭项集合，展示其中的 10 项频繁闭项集合。

从结果上来看，其中的 Japanese 是一个很不合理的景点，本人经过阅读旅游文记以及查看地图之后，发现，Japanese 是和 Pearl Harbor 有关系的一个旅游景点。由于 Pearl Harbor 曾经被日本人偷袭，因此在 Pearl Harbor 会有和日本有关的场所，用来纪念这次战争。Japanese 是代表 Japanese Cultural Center of Hawai。

比如用户选中了 Polynesian 这个景点之后，系统就会推荐 Diamond head，Pearl Harbor,Waikiki beach 等景点给用户。

表 3.13 频繁闭项集合

编号	频繁闭项集合	频度
1	Polynesian,Japanese,Hawaii,Diamond head,Honolulu	3
2	USS Missouri,Japanese,Pearl Harbor,Diamond Head crater,USS Arizon,USS Bowfin,Diamond head	3
3	USS Missouri,Pearl Harbor,Pacific Aviation Museum,Diamond Head crater,Oahu,Honolulu	3
4	Polynesian,Japanese,Pearl Harbor,Waikiki beach,USS Arizona Memorial,Oahu,Honolulu	3
5	Japanese,USS Missouri,USS Arizona Memorial,Arizona,Hawaii,Honolulu,USS Bowfin	3
6	Japanese,USS Missouri,Pearl Harbor,Diamond Head crater,USS Arizona,Honolulu,USS Bowfin,Diamond head	3
7	Japanese,Pearl Harbor,USS Arizona Memorial, Hawaii,Oahu,Australia,Honolulu	3
8	Japanese,USS Missouri,Pearl Harbor,USS Arizona Memorial,Arizona,USS Bowfin,Honolulu,Japan	3
9	Japanese,USS Missouri,Pearl Harbor,Diamond Head crater,USS Arizona， USS Bowfin,Honolulu,Diamond head	3
10	National Park,Kona,Pahoa,Waimea,Hilo,Mauna Kea,Farmers Market,Hawaii,Maui	3

3.4.3 旅游行程规划算法

在 3.3 节当中提出了 MST 聚类算法来解决行程规划的问题。在本节当中将使用真实数据来验证本算法的效果。本实验假设用户选取的景点是表 3.10 当中 Honolulu County 当中编号为 2-11 的十个景点，然后用户要求在 3 天之内游玩这十个景点，那么行程规划的结果如下：

第一天：Pearl Harbor，USS Arizona，Oahu，Polynesian，Waikiki beach，Diamond head，Hanauma Bay

第二天：North Shore，Sunset Beach

第三天：Kailua

另外，如果用户以距离为限制进行行程规划，行程规划的结果如下，路程限制是 11000m：

第一天：Pearl Harbor，USS Arizona，Oahu，Polynesian，Waikiki beach，Diamond head

第二天：North Shore ，Sunset Beach

第三天：Hanauma Bay

第四天：Kailua

这十个景点的经纬度和频度在表 3.10 当中都有显示，就不继续在表格当中列出。根据经纬度可以得到这十个景点之间相互距离，如表 3.14 当中所示。下面验证本算法的有效性，根据这十个景点行程的图生成一棵最小生成树如图 3.5，最后再给出在这课最小生成树上按照不同规则进行聚类的结果如图 3.6。表 3. 14 当中，给出了这十个景点之间的距离关系。

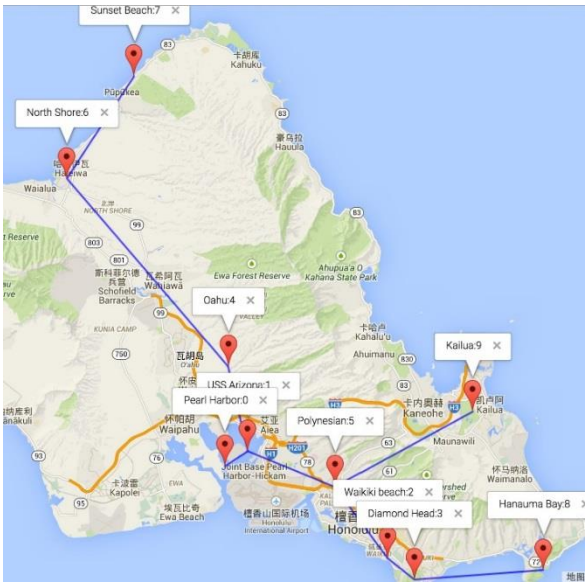

图 3.5 十个景点的最小生成树

表 3.14 景点与景点之间的距离关系

名称	名称	距离（km）	名称	名称	距离（km）
Oahu	Waikiki beach	22	Pearl Harbor	Kailua	22
Oahu	Pearl Harbor	8	Diamond head	Hanauma Bay	11
Oahu	Diamond head	25	Diamond head	Polynesian	11
Oahu	Hanauma Bay	33	Diamond head	North Shore	47
Oahu	Polynesian	14	Diamond head	USS Arizona	18
Oahu	North Shore	22	Diamond head	Sunset Beach	51
Oahu	USS Arizona	7	Diamond head	Kailua	15
Oahu	Sunset Beach	27	Hanauma Bay	Polynesian	20
Oahu	Kailua	22	Hanauma Bay	North Shore	55
Waikiki beach	Pearl Harbor	16	Hanauma Bay	USS Arizona	28
Waikiki beach	Diamond head	3	Hanauma Bay	Sunset Beach	57
Waikiki beach	Hanauma Bay	14	Hanauma Bay	Kailua	15
Waikiki beach	Polynesian	7	Polynesian	North Shore	36
Waikiki beach	North Shore	44	Polynesian	USS Arizona	8
Waikiki beach	USS Arizona	15	Polynesian	Sunset Beach	41
Waikiki beach	Sunset Beach	48	Polynesian	Kailua	14
Waikiki beach	Kailua	15	North Shore	USS Arizona	29
Pearl Harbor	Diamond head	19	North Shore	Sunset Beach	11
Pearl Harbor	Hanauma Bay	30	North Shore	Kailua	42
Pearl Harbor	Polynesian	10	USS Arizona	Sunset Beach	35
Pearl Harbor	North Shore	29	USS Arizona	Kailua	20
Pearl Harbor	USS Arizona	2	Sunset Beach	Kailua	42
Pearl Harbor	Sunset Beach	35			

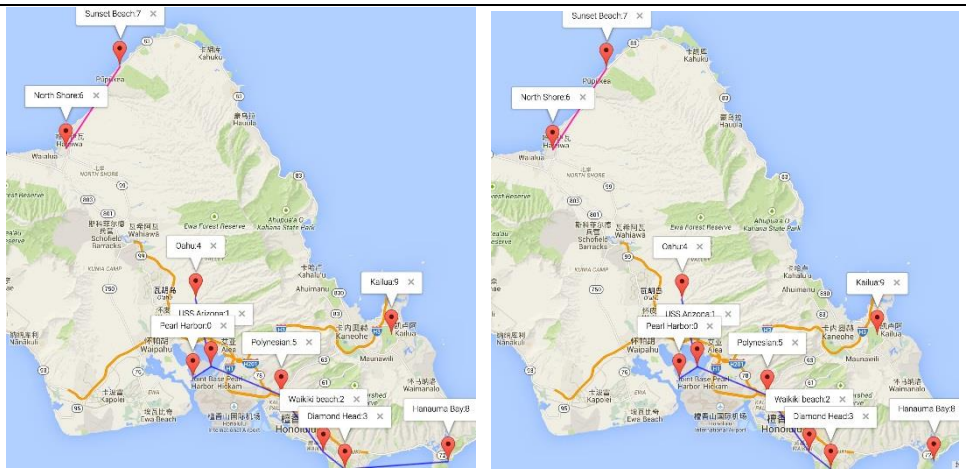

图 3.6 显示了根据 3 天（左）和以阈值为 11KM（右）得到的聚类结果

3.5 本章小结

本章主要讲述景点推荐和行程规划这两个功能。在 3.1 节当中介绍了热门旅游景点推荐算法，基于地理本体树的热门旅游景点推荐，并在 3.1.1 和 3.1.2 小节当中介绍了地理本体树的结构和算法总体流程。在 3.2 节当讲解了关联规则算法的实现过程，并详细介绍了一种频繁闭项集合算法 Closet+算法，在，3.2.1 节介绍了使用频繁增长模式树的数据结构以及挖掘方法，并在 3.2.2 节当中介绍了 Closet+算法所特有的频繁闭项集合挖掘策略，最后在 3.2.3 节当中给出算法的函数组成和结构。3.3 节介绍行程规划算法，在 3.3.1 节当中给出算法的函数组成和结构。在 3.4 节当中，针对之前的三种功能，给出了详细的实验数据和结构，证明本节当中的三个旅游景点相关算法都是有效的。

第4章 旅游路线规划算法

4.1 综述

一位游客来到一个陌生的城市旅游，一般会遇到这样的问题：“早上九点从宾馆出发，如何安排旅游路线，才能够游玩令自己最满意的景点，并且自己在旅游上的花费也在预算之内，最后在下午五点前坐上到达另一个城市的飞机”。一个热门旅游城市会有很多个旅游景点，一位游客可能并不需要全部游玩一遍，通常也来不及全部都游玩。接下来的问题就是“选择哪些景点，并以什么样的顺序来游玩这些景点，才能够使用户获得的满意度最高”，这属于一种最优路线求解问题。哪些景点会令游客满意，这里主要考察两个因素，一个是景点的热度，第二个是用户本身对于景点的偏好。如何来计算景点的热度，已经有很多研究工作，[7][11][13][37-39]，是根据 GPS 定位系统留下来的用户历史路线轨迹，并通过一定的规则来计算景点的热度，最后得到一个一定范围内的数值来代表景点的受欢迎程度。数值越大代表景点越受欢迎。如果单使用景点热度来进行景点选择和路线规划，根据推荐系统当中的理论，最终得到的结果会是少量热门的景点。所以实际当中需要加入用户的偏好，使得推荐的景点更加多样化。本文没有根据用户轨迹来获得景点的热度，而是通过景点名词在旅游文记当中出现的次数也称为景点词频来反映景点的热度。从理论上来看，旅游文记表达的内容更加丰富，使用景点词频来表达热度切实可行。而用户对于景点的偏好，之前的研究[8]利用地理话题模型进行了主题分析，并且能够根据旅游特征来推荐旅游景点。根据这个模型，可以得到用户对于各种景点特色的喜好，并预测其对其他景点的喜好程度，同样可以用一个数值来表达用户对于景点的偏好。对于每一个用户 U，其对于每一个景点的满意度计算如公式(4.1)：

$$US_i = (AS + \alpha SS_{\mathrm{i}})W_i \qquad \text{公式(4.1)}$$

其中 AS 是每个旅游景点的热门程度，对于一定规模的旅游文记，这个数值是固定的。SS_{i}是用户个性化喜好，α作为系数使得SS_{i}的分数和 AS 达到一个数量级，W_i是归一化参数。获得用户对于景点的满意度计算方式后，接下来就是考虑如何获得景点之间的路程时间和单个景点的游玩时间和费用，旅游的时间可以根据线路之间的长短来估计，而景点的游玩时间，可以根据旅游经理的经验，以及旅游景点区域大小来估计。景点的费用可以在景区网站上获得，同样可以咨询旅游经理。费用方面，本算法以一个个体为单位考虑，团体票费用暂时不考虑。

当景点满意度，景点之间的路程时间，景点的游玩时间和费用都获得之后，我们可以获得一张景点旅游图，如图 4.1. 图中有四个景点，a1 到 a4，每个景点都有一个三元组所表示的属性，分别是每个景点的满意度，停留时间和费用消耗。对于每两个景点之间的连线上，根据公式 4.1 计算出所需要的旅行时间。另外两个点 au 代表用户所在的地点，at 代表用户在游完所有的景点之后，所要达到的地点。也就是所谓的起点和终点，起点和终点的满意度，停留时间和费用消耗默认为 0。

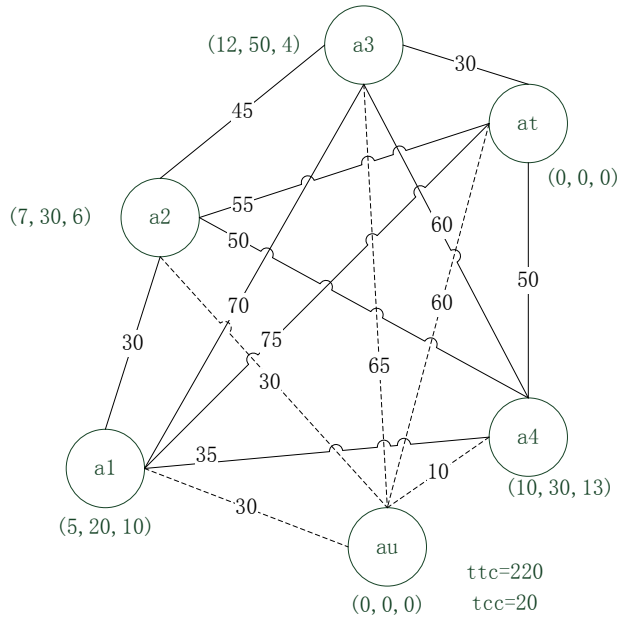

图 4.1 一个旅游地图的例子

接下来就需要考虑如何来根据用户所给的限制条件来选择景点和游玩顺序。影响旅游路线的因素有游玩时间，路程长短，旅游费用等因素。前人所提出的旅行规划算法，大多数只考虑景点的热门程度[40-43]，少数研究[44-45]以游玩时间为限制来进行旅游路线规划，目前所知同时考虑了时间，费用和满意度的路线推荐算法只有[46]。这篇文章需要用户指定旅游关键字，并根据关键字找到一条最优路线。和旅游路线规划比起来，用户所给的关键字给限定了某些特殊的地点。本文除了考虑用户满意度之外，同时还考虑游玩时间和旅游费用这两个重要因素。因为，可能最好玩的景点，用户根本来不及去游玩。另外，最好玩的旅游路线可能费用昂贵。在时间和费用两个因素的影响下求最优解，这是一个天际线(skyline)查询问题。当所有景区都免费，或者景区费用未知时，就退化成单值优化问题。旅游路线规划问题，就是游客指定起点（宾馆）和终点（飞机场），然后系统返回最优路线，已经有很多研究[7,44,46]给出了算法。这些算法中有精确的最优解算法，也有近似解算法。相同算法框架和数据集下下，近似解算法要比精确解算法至少快一个数量级。

4.2 问题描述

参照图 4.1 当中，本文所描述的最优路径问题是产生一系列景点序列，这些序列以起点开头，以终点结尾，这些序列的时间并和费用都必须满足用户的旅行时间和费用预算。而这条路径需要使用户获得的满意度最大。本文所研究的最优旅行路线问题和旅行商问题[47]不同，旅行商问题是在给定的 n 个点当中，找到一条最短的路径经过这 n 个点。旅行商问题的结果需要显示这 n 个点，但是最优路线查找问题是为了找 n 个点子集的一个排列。这属于 NP 完全问题，当景点的数量增加时，最优路线查找的计算时间会直线上升。表 4.1 当中显示按照暴力法求解最优路线所需要检查的路线列表。关于表格当中四列数字的含义在下一节当中有详细解释。

表 4.1 暴力法搜索旅游路线的过程

路线	时间	满意度	费用
$\langle a_1 \rangle$	125	5	10
$\langle a_2 \rangle$	115	7	6
$\langle a_3 \rangle$	145	12	4
$\langle a_4 \rangle$	90	10	13
$\langle a_1, a_2 \rangle$	165	12	16
$\langle a_1, a_3 \rangle$	165	12	16
$\langle a_1, a_4 \rangle$	165	12	16
$\langle a_2, a_1 \rangle$	185	12	16
…	…	…	…
$\langle a_1, a_2, a_3 \rangle$	235	24	20
$\langle a_1, a_3, a_2 \rangle$	300	24	20
$\langle a_2, a_1, a_3 \rangle$	260	24	20
$\langle a_2, a_3, a_1 \rangle$	320	24	20
$\langle a_3, a_1, a_2 \rangle$	320	24	20
$\langle a_3, a_2, a_1 \rangle$	315	24	20
$\langle a_1, a_2, a_4 \rangle$	240	22	29
…	…	…	…
$\langle a_1, a_2, a_3, a_4 \rangle$	345	34	33
…	…	…	…

在研究过前任的算法之后，本人就发现这些研究当中通常有如下假设，就是任意两点之间都是连通的，并且任意两点之间是线段连接。这样的假设方便计算，同样也方便展示和用户理解。本文在旅游文记当中提取的地理名词包含有经纬度信息，根据两个景点的经纬度坐标，就可以得到两个景点之间的直线距离。但是，本文在根据两点之间的距离计算路程时间时，遇到如下问题。地理位置上两点之间的直接距离并不等于实际环境当中人们从一个点走到另一个点

的距离。当实际环境当中，两点之间没有障碍的情况下，比如在草原，荒漠等地带，两点之间的直线距离等于人们所走的路径距离。在城市，山川等地方，两点之间的路径需要沿着固定的曲线，或者根本没有固定的道路。本人根据地图上两点之间的直线距离和实际距离的对比，发现，两点之间的实际距离就算在城市这样有规定道路的地方，其实际距离也只是略微大于直线距离。另外，两点之间的实际距离和直线距离还有如下关系，随着两点之间的直线距离增加，其实际距离和直线距离会越来越接近。本文考虑自驾游的旅行方式，根据汽车行驶速度限制，制定了公式(4.2)来得到路程时间，x 是路程长度，单位为米（m），y 为旅行时间，单位为分钟(minute)。

$$y = \begin{cases} 5 & (0 \le x \le 1000) \\ 0.0022x + 3.78 & (1000 \le x \le 10000) \\ 0.0007x + 8 & (10000 \le x \le 60000) \\ \frac{x}{1000} & (x \ge 60000) \end{cases} \qquad \text{公式（4.2）}$$

本文采取精确最优解算法，并且本文的研究也基于两点间直接可达的假设，本文深入研究了 TripMine[12]算法之后,提出 MultipleGuide 算法。TripMine 仅仅考虑起点和终点重合的情况来进行路线挖掘，并且只考虑了景点的时间因素.本算法终点可以设置在任意位置，并且有两个因素制约路线的选择，由于最优解不仅仅受一个因素影响，得到的初步结果还需要天际线查询算法进行进一步筛选。TripMine 算法提出了三个优化策略，本文在仔细研究了 TripMine 算法的整个过程之后，在其优化的基础上，又加入了两个优化策略。实验结果表明，本文提出的算法在时间和内存方面都优于 TripMine。由于本文增加了费用因素，本文在使用 TripMine 算法时也要额外加入了费用限制。本文所处理的问题更加复杂，得到的结果也更加多样化。

4.3 旅游路线规划算法

这一章将讲述本文所提出的旅游路线规划算法，表 4.2 当中总结了本章当中一些符号的含义。

表 4.2 关键符号的含义

符号	含义
a;A	a 代表一个景点；A 代表一个景点集合
s;S	s 代表一个旅游线段；S 代表一个旅游线段集合
DOS(a)	代表一个旅游景点所获得的满意度
ST(a)	代表一个景点的停留时间
AC(a)	一个旅游景点所需要的花费
TT(s)	表示走完一条旅游线段所需要的时间
M	代表一张包含旅游景点和旅游线段的旅游地图，M＝（A，S）
tr	代表一条包含若干个旅游景点的旅游路线，景点和景点之间是有序的
TS(tr)	代表一条旅游路线所获得的满意度
TC(tr)	代表一条旅游路线所需要的花费
TRT(tr)	代表一条旅游路线所需要的时间
a_u	用户的起点位置
a_t	用户的终点位置

4.3.1 路线规划算法基本概念

定义 1. 景点集合(Attraction Set)。公式表示为$A = \{a_1, a_2, \dots a_i, \dots, a_{|k|}\}$。对于每一个$a_i \in A$,景点$a_i$都有三个属性，分别是满意度（Degree Of Satisfaction）DOS(a_i),停留时间（Stay Time）ST（a_i）和景点费用(Attraction Cost)AC(a_i)，旅游景点满意度就是之前所描述用户对一个景点的总体满意程度。停留时间代表一个旅游景点的平均游玩时间，景点费用代表一个景点的总体游玩费用。例如图 4.1 当中，以a_1为例，$DOS(a_1) = 5$，$ST(a_1) = 20$，$AC(a_1) = 10$

定义 2.旅游线段集合(Segment Set)。公式表示为 S=$\{s_1, s_2, \dots s_i, \dots, s_{|k|}\}$。对于每一个旅游线段$s_i \in S$，$s_i = (a_x, a_y)$,其中$a_x, a_y$是景点集合 A 当中的景点，并且这两个景点不能相等。景点之间的线段长度含义是前文所描述的两点之间的直线距离，为了方便计算，直接用前文所描述的路程时间长短来描述线段的长短。因此对于每一个线段$s_i \in S$都有一个属性旅行时间（Travel Time）TT(a_x, a_y)。例如图 4.1 当中有 15 条旅游线路，其中包括$s_1 = (a_1, a_2)$，$s_2 = (a_2, a_3)$，其中TT(s_1) = TT$(a_1, a_2) = 30$，TT(s_2) = TT$(a_2, a_3) = 45$。

定义 3.旅游地图(Travel Map)。公式表示为M = （A，S）。图 4.1 就代表一个旅行图，这个旅行图当中有四个旅游景点，其中a_u是起点，a_t是终点。起点和终点也属于旅游景点，他们的特殊性在于不贡献满意度和停留时间。在这个旅行地图 M 当中，有 6 个旅游景点，15 条旅游线段。那其中的a_1和a_2来举例子，DOS(a_1)=5，ST（a_1）=20，AC(a_i)=10，DOS(a_2)=7，ST（a_2）=30，AC(a_2)=5。从图 4.1 中可以看出a_1和a_2之间的旅行时间是 30，因此TT$(a_1, a_2) = 30$。

定义 4.旅游路线（Trip Route）。公式表示为tr = $\langle a_1, a_2, \dots, a_2, \dots, a_j \rangle$，其中$a_i \in A$并且$a_i \neq a_u$，$a_i \neq a_t$，因为路线的起点和终点已经固定了，不需要在旅行路线当中再次显示。j 代表路径的长度。tr 当中景点和景点之间是有序的，比如图 4.1 当中，一个简短的路径$\langle a_1 \rangle$代表从a_u出发，经过a_1，然后再从a_1到达终点a_t。

定义 5.旅游路线满意度（Trip Route Satisfaction）。旅游满意度是相对于一条旅游路线而言的，针对一条旅游路线tr = $\langle a_1, a_2, \dots, a_i, \dots, a_j \rangle$，旅游满意度TS(tr)的定义如公式（4.3）。在表 4.3 当中列出了图 4.1 当中所有有效的旅游路线，如表 4.3 当中，TS($\langle a_1, a_2 \rangle$) = 7 + 5 = 12。也就是游客选择旅游路线$\langle a_1, a_2 \rangle$获得的满意度是 12。

$$\text{TS(tr)} = \sum_{i=1}^{j} DOS(a_i) \qquad \text{公式（4.3）}$$

定义 6.旅游花费(Trip Cost)。旅游花费同样是一条路线的属性，针对一条旅游路线tr = $\langle a_1, a_2, \dots, a_i, \dots, a_j \rangle$，旅游花费 TC(tr)的定义如公式（4.4）。例如表 4.3

当中$\mathrm{TC}(\langle a_1, a_2\rangle) = 10 + 5 = 15$。在本文中，旅游花费的单位是美元。若游客选择这条路线，需要花费 15 美元的旅游费用。

$$\mathrm{TC}(\mathrm{tr}) = \sum_{i=1}^{j} AC(a_i) \qquad \text{公式（4.4）}$$

定义 7.旅游路线时间(Trip Route Time)。对于一段旅游路线$\mathrm{tr} = \langle a_1, a_2, \dots, a_2, \dots, a_j\rangle$,TRT(tr)的表达式如公式（4.5）。例如$\mathrm{TRT}(a_u, \langle a_1, a_2\rangle, a_t) = \mathrm{TT}(a_u, a_1) + \mathrm{TT}(a_2, a_u) + \mathrm{TT}(a_u, a_1) + \mathrm{TT}(a_1, a_2) + \mathrm{ST}(a_1) + \mathrm{ST}(a_2) = 30 + 55 + 30 + 30 + 20 = 165$也就是说游客选择路线$\langle a_1, a_2\rangle$需要花 165 分钟完成，表 4.3 当中也列出了一些其他旅游路线的时间。

$$\mathrm{TRT}(a_u, \mathrm{tr}, a_t) = \mathrm{TT}(a_u, a_1) + \mathrm{TT}(a_j, a_t) + \sum_{i=1}^{j} TT(a_i, a_{i+1}) + \sum_{i=1}^{j} ST(a_i) \qquad \text{公式(4.5)}$$

定义 8.旅游时间限制(Trip Time Constraint)。也就是游客完成一趟旅行所可以支配的最长时间，这个时间是游客给出的，如图 4.1 所示，旅游地图的时间限制 ttc=220 分钟。

表 4.3 所有有效的旅游路线

旅游路线	时间	满意度	花费	旅游路线	时间	满意度	花费
$\langle a_1\rangle$	125	5	10	$\langle a_1, a_3\rangle$	200	17	14
$\langle a_2\rangle$	115	7	6	$\langle a_2, a_3\rangle$	185	19	10
$\langle a_3\rangle$	145	12	4	$\langle a_2, a_4\rangle$	190	17	19
$\langle a_4\rangle$	90	10	13	$\langle a_4, a_2\rangle$	175	17	19
$\langle a_1, a_2\rangle$	165	12	16	$\langle a_4, a_3\rangle$	180	22	17
$\langle a_2, a_1\rangle$	185	12	16				

定义 9.旅游费用限制(Trip Cost Constraint)。ttc 表示希望在一趟旅行当中所花费货币的最大值，这个值同样是游客给出的，如图 4.1 所示，旅游地图的费用限制 tcc=20 美元。

定义 10.有效的旅游路线(Valid Trip Route)。如图 4.1 设a_u为起点a_t为终点，ttc 为时间限制，tcc 为费用限制。一个有效的$\mathrm{tr} = \langle a_1, a_2, \ldots, a_2, \ldots, a_j \rangle$必须得满足$\mathrm{TRT}(a_u, \mathrm{tr}, a_t) \leq \mathrm{ttc}$并且$\mathrm{TC}(\mathrm{tr}) \leq \mathrm{tcc}$。也就是这条旅游路线的时间开销必须小于或者等于时间限制，费用开销必须小于或者等于费用限制，否则这条旅游路线是无效的。参照表 4.3，旅游路线$\langle a_1, a_2 \rangle$是有效的，$\mathrm{TRT}(a_u, \langle a_1, a_2 \rangle, a_t)$的时间为 165 满足时间限制，并且费用$\mathrm{TC}(\langle a_1, a_2 \rangle)$为 20 满足费用限制。但是线路$\langle a_1, a_4 \rangle$就是无效的旅游路线，尽管其时间$\mathrm{TRT}(a_u, \langle a_1, a_4 \rangle, a_t)$为 165 满足时间限制，因为的费用$\mathrm{TC}(\langle a_1, a_4 \rangle)$是 23 超过了费用限制。表 4.3 当中列出了所有有效的旅游路线。

定义 11.候选景点集合（Candidate Attraction Set）.设景点集合$\mathrm{A} = \{a_1, a_2, \ldots, a_i, \ldots, a_k\}$,取 A 中的 k 景点可以组成某一条旅游路线，那么 A 就是 k 阶候选集合。例如候选集合$\{a_1, a_2\}$是一个二阶候选景点集合。如表 4.1，列举了一阶到三阶的所有候选景点集合。

定义 12.有效的旅游景点集合（Valid Attraction Set）.设$\mathrm{A} = \{a_1, a_2, \ldots, a_i, \ldots, a_k\}$,如果取集合 A 中的每一个景点，能够组成一条有效的旅游路线，那么这个旅游景点集合就是有效的。相反，如果这个景点集合当中景点组成的所有路线都是无效的，那么这就是一个无效的旅游景点集合。举一个例子，如图 4.1 当中，景点集合$\{a_3, a_4\}$是一个有效的景点集合，这个集合可以生成两个旅游路线$tr_1 = \langle a_3, a_4 \rangle$，$tr_2 = \langle a_4, a_3 \rangle$。尽管$\mathrm{TRT}(a_u, tr_1, a_t) = 255$是无效的，但是$\mathrm{TRT}(a_u, tr_2, a_t) = 180$是有效的，因此景点集合$\{a_3, a_4\}$是有效的。

定义 13.极大旅游路线(Maximal Trip Route)。设a_u为起点a_t为终点，ttc 为时间限制，tcc 为费用限制。极大旅游路线就在有效的旅游路线，找出旅游路线的旅游满意度最高的旅游路线。可能有多条极大旅游路线，这些极大旅游路线获得的满意度都是最大的。如图 4.1 当中，旅游路线$\langle a_1, a_2 \rangle$，$\langle a_1, a_3 \rangle$，$\langle a_3, a_4 \rangle$都是有效的，$\langle a_4, a_3 \rangle$的旅游满意度最高，所有$\langle a_4, a_3 \rangle$是极大旅游路线。

定义 14.最优旅游路线(Optimal Trip Route)。设a_u为起点a_t为终点，ttc 为时间限制，tcc 为费用限制。最优旅游路线就是在极大旅游路线当中，找到在时间和费用上最优的旅游路线。举一个例子，目前有四条极大旅游路线tr_1, tr_2, tr_3, tr_4，这

四条旅游路线的旅游路线时间，旅游满意度和旅游花费分别为tr_1：$<200,22,10>, tr_2:<210,22,8>, tr_3:<210,22,12>, tr_4:<200,22,11>$。其中，只有$tr_1$和$tr_2$是最优旅游路线，另外两个都不是。因为$tr_1$在费用上优于$tr_4$，$tr_2$在费用上优于$tr_3$。尽管$tr_1$在时间上优于$tr_2$，但是$tr_2$在费用上优于$tr_1$，他们相互都不能够控制对方，并且他们也无法被其他极大路线所控制，因此tr_1和tr_2是最优旅游路线。在图 4.1 当中，极大旅游路线只有$\langle a_3, a_4\rangle$，因此$\langle a_3, a_4\rangle$也是最优旅游路线。

4.3.2 基本性质和定理

性质 1 旅游满意度常数性质。设景点集合 A=$\{a_1, a_2, \dots, a_i, \dots, a_k\}$，那么由集合 A 当中所有景点组成的所有路线，其满意度都是不变的。设有$tr_k = \langle a_{1k}, a_{2k}, \dots, a_{ik}, \dots, a_{jk}\rangle, tr_l = \langle a_{1l}, a_{2l}, \dots, a_{il}, \dots, a_{jl}\rangle$，其中对于所有$a_k \in tr_k$，$a_k \in A$，对于所有$a_l \in tr_l$，$a_l \in A$。$tr_k$和$tr_l$取到集合 A 当中所有景点，那么$tr_k$和$tr_l$长度相等，并且其所有的景点都相同。那么$\mathrm{TS}(tr_k) = \mathrm{TS}(tr_l)$。由于$\mathrm{TS}(tr_k) = \sum_{i=1}^{k} DOS(a_i)$，$\mathrm{TS}(tr_l) = \sum_{i=1}^{k} DOS(a_i)$，因此$\mathrm{TS}(tr_k) = \mathrm{TS}(tr_l)$。

性质 2 旅游费用常数性质。设景点集合 A=$\{a_1, a_2, \dots, a_i, \dots, a_k\}$，那么由集合 A 当中所有景点组成的路线，其费用都是不变的。证明过程和性质 1 类似，就不再赘述。

定理 1 有三个不同的旅游线段分别是$s_1 = (a_x, a_y), s_2 = (a_y, a_z), s_3 = (a_x, a_z)$，这三个旅游线路的时间分别为$\mathrm{TT}(s_1)$，$\mathrm{TT}(s_2)$，$\mathrm{TT}(s_3)$，那么任意两个线段的时间加起来，都不会小于第三条线路的时间。这相当于三角行两边之和大于第三边性质，就不深入证明。

根据定理 1，可以得出定理 2，一条旅游路线的子路线的时间，必然不会超过这条路线。设 $\mathrm{tr} = \langle a_i, a_k, \dots, a_l, \dots, a_j\rangle$是一条旅游路线，其子路线为 $\mathrm{tr'tr} = \langle a_{ik}, a_{kk}, \dots, a_{lk}, \dots, a_{jjk}\rangle$，用户的起点和终点分别是$a_u$和$a_t$，那么$\mathrm{TRT}(a_u, \mathrm{tr}, a_t) \geq \mathrm{TRT}(a_u, \mathrm{tr'}, a_t)$。

证明：首先证明父路线比子路线多一个景点的情况。假设 tr 比 tr'多一个景点 a，那么景点 a 有三种情况出现在 tr 当中，分别出现在 tr 的开头，结尾和中部。下面可以分成三种情况证明。(1)a 是 tr 的第一个景点。其第二个景点为a_i，那么 $\mathrm{TRT}(a_u, \mathrm{tr}, a_t) - \mathrm{TRT}(a_u, \mathrm{tr}', a_t) = \mathrm{TT}(a_u, \mathrm{a}) + \mathrm{TT}(\mathrm{a}, a_i) - \mathrm{TT}(a_u, a_i)$，根据性质 1，可以得出$\mathrm{TRT}(a_u, \mathrm{tr}, a_t) \geq \mathrm{TRT}(a_u, \mathrm{tr}', a_t)$。(2)a 是 tr 的最后一个景点，其倒数第二个景点为a_j。那么 $\mathrm{TRT}(a_u, \mathrm{tr}, a_t) - \mathrm{TRT}(a_u, \mathrm{tr}', a_t) = \mathrm{TT}(a_j, \mathrm{a}) + \mathrm{TT}(\mathrm{a}, a_t) - \mathrm{TT}(a_j, a_t)$，根据性质 1，得出$\mathrm{TRT}(a_u, \mathrm{tr}, a_t) \geq \mathrm{TRT}(a_u, \mathrm{tr}', a_t)$。(3)a 是 tr 中间的一个点，假设 a 左边和右边的点分别为a_i和a_j，那么 $\mathrm{TRT}(a_u, \mathrm{tr}, a_t) - \mathrm{TRT}(a_u, \mathrm{tr}', a_t) = \mathrm{TT}(a_i, \mathrm{a}) + \mathrm{TT}(\mathrm{a}, a_j) - \mathrm{TT}(a_i, a_j)$。根据性质 1，得出 $\mathrm{TRT}(a_u, \mathrm{tr}, a_t) \geq \mathrm{TRT}(a_u, \mathrm{tr}', a_t)$。下面扩展到一般的情况，当 tr 比 tr'多 n 个景点时，假设 n 个景点的集合为A_n，我们首先从集合A_n中拿出一个景点a_{n1}加入到 tr'组成tr'_1，刚才已经证明$\mathrm{TRT}(a_u, tr'_1, a_t) \geq \mathrm{TRT}(a_u, \mathrm{tr}', a_t)$，接下来再从$A_n$当中拿出一个景点$a_{n2}$加入到$tr'_1$组成$tr'_2$，同样有$\mathrm{TRT}(a_u, tr'_2, a_t) \geq \mathrm{TRT}(a_u, tr'_1, a_t)$，以此类推，当拿出第 n 个点之后，$tr'_n = \mathrm{tr}$，最后证明 $\mathrm{TRT}(a_u, \mathrm{tr}, a_t) \geq \mathrm{TRT}(a_u, \mathrm{tr}', a_t)$。

由定理 2 得出推论 1，无效旅游路线的父路线是无效的。

证明，设有旅游路线$\mathrm{tr} = \langle a_i, a_k, \dots, a_l, \dots, a_j \rangle$，用户的起点和终点分别是$a_u$和$a_t$，旅游时间限制是 ttc。如果 tr 是一条无效的旅游路线，也就是$\mathrm{TRT}(a_u, \mathrm{tr}, a_t) > ttc$。那么其父路线$\mathrm{tr}' = \langle a_{ik}, a_{kk}, \dots, a_{lk}, \dots, a_{jk} \rangle$是一条无效的旅游路线。有定理 2 得出$\mathrm{TRT}(a_u, \mathrm{tr}', a_t) \geq \mathrm{TRT}(a_u, \mathrm{tr}, a_t)$，所以有$\mathrm{TRT}(a_u, \mathrm{tr}', a_t) > \mathrm{ttc}$。

4.4 算法框架以及优化策略

4.4.1 算法框架

（1）候选景点集合产生过程

设有 k 阶候选集合 $CA_1 = \{a_{1m}, a_{2m}, \dots, a_{\mathrm{im}}, \dots, a_{km}\}$和$CA_2 = \{a_{1\mathrm{n}}, a_{2n}, \dots, a_{\mathrm{in}}, \dots, a_{kn}\}$,那么这两个 k 阶候选集合可以产生 k+1 阶候选集合如果

$a_{im} = a_{in}(1 \le i \le k-1)$并且$a_{km} \ne a_{kn}$，这个 k+1 阶候选集合$CA_3 = \{a_{1m}, a_{2m}, \dots, a_{\mathrm{im}}, \dots, a_{km}, a_{kn}\}$。

（2）旅游路线生成过程

候选集合从第一阶开始产生，并且在产生完所有第 k 阶候选集合之后，产生第 k+1 阶候选集合。对于每一个 k 阶候选集合，首先检查集合费用，根据性质 1 和性质 2，就可以判断这个 k 阶候选集合有没有可能是有效的集合。如果这个集合当中所有景点的费用满足旅游费用限制，那么就进一步检查这个景点所生成的旅游线路是否符合旅游时间限制。否则，这个集合就可以被判定就是一个无效的旅游景点集合。

生成其排列并检查是否含有有效的旅游路线。这个需要用到全排列生成算法，复杂度是 O（k!）。全排列生成算法已经很成熟，就不赘述。如果能够找到一条有效的旅游路线，就赋予这个旅游景点集合相应的旅游路线时间，这个旅游路线时间并不是这个旅游景点集合最短时间的旅游路线集合。这个景点集合最短的旅游路线可能有很多，本文并不需要把一个景点的最优旅游路线都显示出来。由于本文最终需要产生极大旅游路线并最终产生最优旅游路线，极大旅游路线是由旅游路线满意度决定的，所以并不需要立刻就把一个旅游集合的最短时间的旅游路线计算出来。如果一个候选集合由全排列生成的所有旅游路线都是无效的，那么这个候选集合就是无效的。以图 4.1 为例，图 4.2 中显示如何检查判断一个候选旅游景点集合是无效的。

根据推论 1 可以得出，推论 2 一个候选集合的父集合也是无效集合。设有候选集合$\mathrm{CA} = \{a_1, a_2, \dots, a_i, \dots, a_k\}$，其父集合$\mathrm{CA'} = \{a_1, a_2, \dots, a_i, \dots, a_n\}$，并且候选结合 CA 是无效的，也就是对于候选集合 CA 所生成的每一条旅游路线 tr，都有$\mathrm{TC(tr)}$大于旅游费用限制或者$\mathrm{TRT(tr)}$大于旅游时间限制。对于$\mathrm{CA'}$生成的每一条旅游路线 tr’，其子候选集合 CA 都能生成一条旅游路线tr_i与之对应，使得 tr’是其父旅游路线。因此$\min(\mathrm{TRT(tr')}) \ge \min(\mathrm{TRT}(tr_i))$，而$\min(\mathrm{TRT}(tr_i))$高于旅游时间限制，因此$\min(\mathrm{TRT(tr')})$也高于旅游时间限制。关于旅游费用限制的证明很明显，就不详细描述了。

因此，在发现了第 k 阶的无效旅游景点集合之后，这些无效的旅游景点集合就不能够成为产生下一阶旅游景点集合的候选集合。这里用到了剪枝的思想，从而可以降低整个算法的复杂度。以图 4.1 为例，图 4.2 中显示了其候选集合产生过程。其中，一阶候选集合都是有效的旅游景点集合。在所有二阶景点集合当中，候选集合$\{a_1, a_4\}$是无效的。在第 3 阶候选集合当中，所有的旅游景点集合都是无效的。第 k 阶旅游景点候选集合无法产生下一阶旅游景点集合时，旅游路线生成过程停止。

（3）极大旅游路线生成。设置一个全局极大旅游景点集。每当生成一个有效的旅游景点集合之后，就拿这个旅游景点集合与当前的极大旅游集合的旅游路线满意度对比，如果当前极大旅游景点集合为空，那么当前的有效集合就是极大旅游候选集合。如果其满意度和当前极大景点集合的一样大，那么就把这个有效的旅游景点集合也加入到极大旅游景点集当中，如果其满意度大于当前极大景点集的满意度，那么就清空当前的极大旅游景点集，并加入当前旅游景点集合。以图 4.2 为例，当一阶候选集合产生之后，极大旅游景点集当中只有一个旅游景点集合$\{a_3\}$。当第二阶候选集合产生之后，候选集合$\{a_3, a_4\}$产生极大旅游景点集当中的唯一的旅游景点集合。当第三阶旅游景点产生之后，极大旅游景点集不变。

（4）最优旅游路线生成。旅游路线生成过程停止之后，得到一个极大旅游景点集，这个集合当中每一个旅游景点集合所能产生的满意度都是最大的。针对每一个旅游景点集合，生成时间最短的一条旅游路线，一条旅游景点集合可能产生多条时间最短的旅游路线，由于不同的旅游顺序对于旅游时间限制和旅游费用限制并无影响，所以只取其中的一条作为代表，这个过程需要检查这个旅游景点集合的所有排列的旅游路线。在所有代表最短旅游时间的旅游路线当中，根据天际线查询来求得最优旅游路线集合。同样以图 4.2 为例，3 阶候选旅游景点集合生成之后，极大旅游景点集合有$\{a_3, a_4\}$，接下来求$\{a_3, a_4\}$的最短时间旅游路线，求得的结果是$\langle a_4, a_3\rangle$。由于极大旅游景点集只有一个景点，不需要使用天际线查询来找最优解，最优旅游路线就是 tr=$\langle a_4, a_3\rangle$。

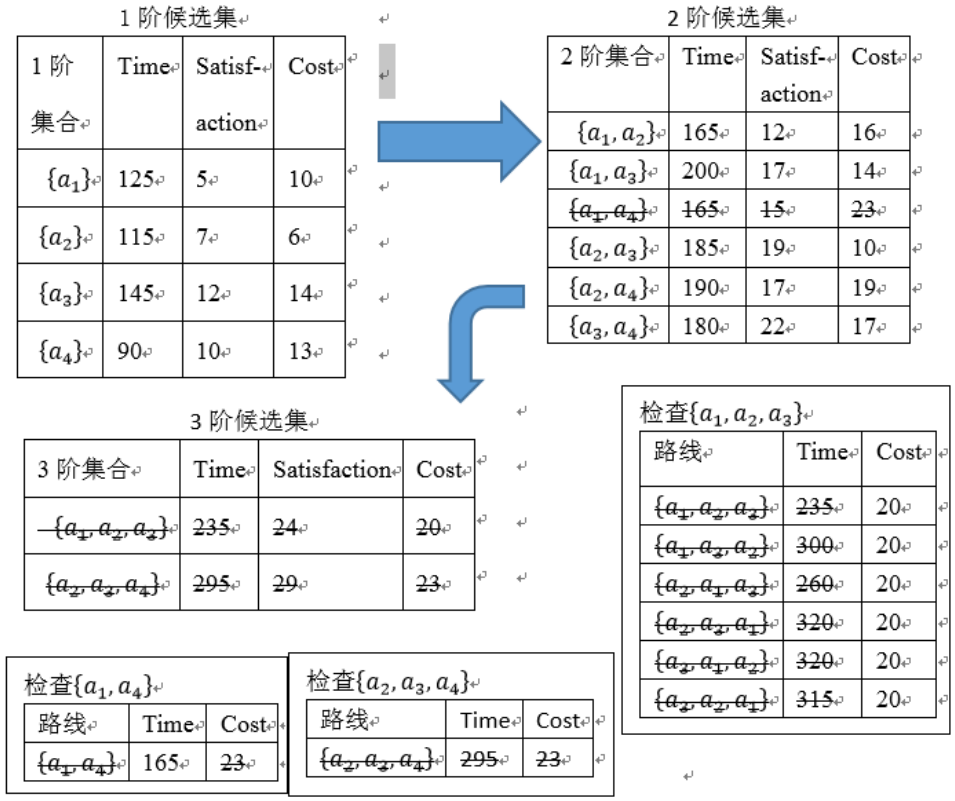

图 4.2 候选集合产生和检查无效候选集合的过程

4.4.2 优化策略

1. 排列检查优化

设有候选集合$\mathrm{CA}=\{a_1,a_2,\ldots,a_i,\ldots,a_k\}$，检查 CA 的全排列生成的旅游路线时，并不总是一定需要 O（k!）的复杂度，就算是在 CA 的所有排列的旅游路线时间都超过用户所给旅游时间限制情况下，可以使用剪枝的方法，减少计算时间。排列优化剪枝策略如下，在生成 CA 的全排列的过程中，就开始计算某一个排列生成旅游路线所需要的时间，当发现时间已经超过旅游时间限制后，就不需要继续生成这个排列。继而生成下一个排列。以图 4.2 中为例，当检查候选集合$\{a_1,a_2,a_3\}$时，在生成排列$\{a_2,a_3,a_1\}$时，当排列已经有$\{a_2,a_3\}$，并进入 1 的时候，当前旅游路线所消耗的时间已经达到 225，不需要继续加入a_1。从而可以提高返回检查其他的排列。

2. 排序优化

把一阶候选集合按照旅游路线费用长短从大到小排列，如果旅游费用一样，那么就按照旅游路线时间从大到小排列。从候选集合产生过程来看，每一阶候选集产生，排在前面的景点集合能够和后面更多的景点集合生成新的景点集合。那么，如果一阶候选旅游集合经过排序之后，那么生成的二阶候选集合当中，排在第一位的候选集合其费用和时间都会相对比较大，那么其将会长生更少的三阶旅游候选集合。在某些情况下，排序优化可以显著减少候选集合生成。当起点和终点重合，并且不考虑费用因素的情况下，这个优化效果更佳明显。图 4.3 是经过排序后的候选集产生和检查过程，由于本文例子特殊，所以根据本例子从排序优化中看不到计算时间的减少。

3.下界检查优化

设有 k 阶候选集合$\mathrm{CA} = \{a_1, a_2, \dots, a_i, \dots, a_k\}$，这个集合生成的旅游路线排列的最短时间下界由三部分组成（1）集合当中 k 个景点的停留时间只和$\sum_{i=1}^{k} a_i$。（2）k 个景点中，和 k 个景点相连当中的最短的 k-1 条边，需要从C_k^2条路线当中去寻找。（3）起点到这 k 个景点的最短距离和终点到这 k 个景点的最短距离之和。证明如下，由于需要经过这 k 个景点，那么最短旅游路线的时间必然包含这 k 个景点的停留时间。对于第二点，旅游路线需要经过 k 个点，那么每个点所延伸出来的最小线肯定是这条路的最小值。由于从起点出发，并回到终点，那么这条路线无法以小于和起点终点连接的最短路线经过。证明完毕。

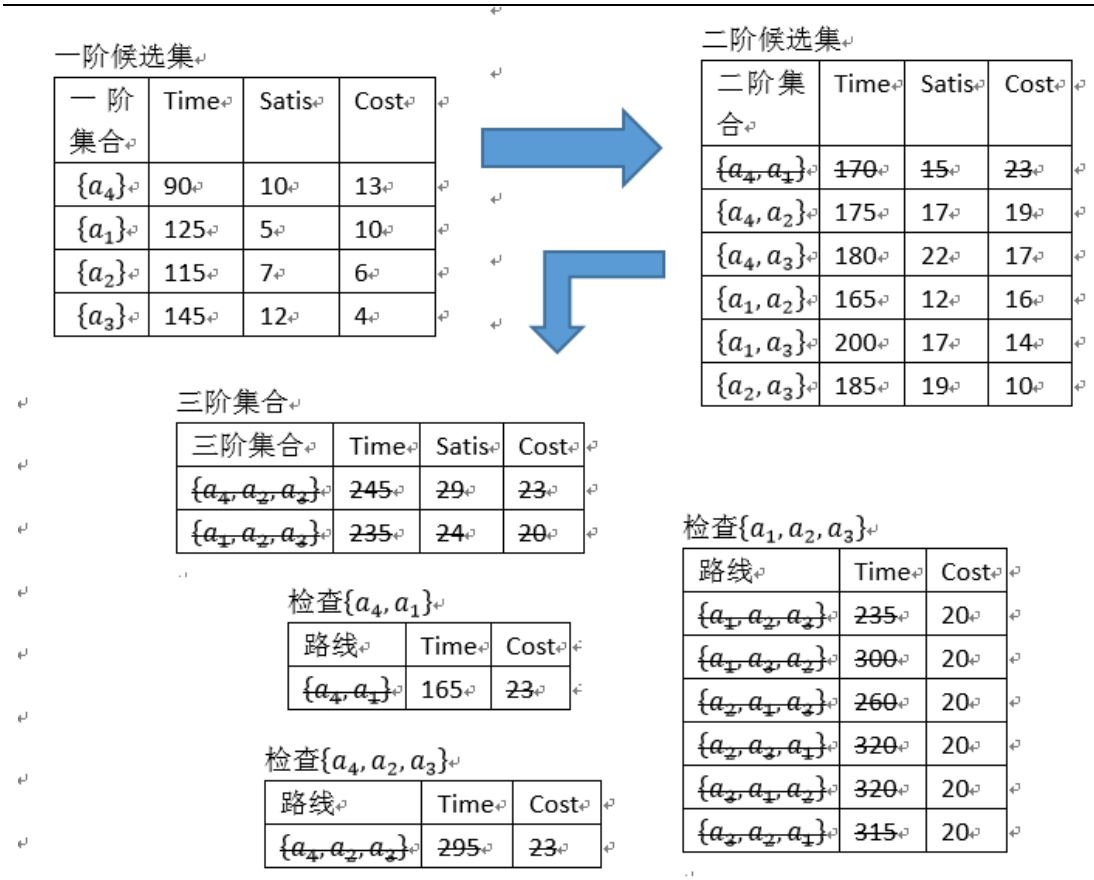

图 4.3 经过排序优化后的候选集合产生过程和候选集检查过程

因此，在减产一个候选集合是否是有效之前，先判断这个候选集合的最短时间下界是否大于旅游路线时间限制。如果大于就可以直接判定该候选集合无效。下界检查的复杂度为$O(k^2)$，极大节省了本来需要 0(k!)的计算时间。以图 4.2 当中的候选集合$\{a_1,a_2,a_3\}$为例进行最短下界时间检查。$\{a_1,a_2,a_3\}$当中，三个集合的停留时间为 100 分钟，和a_1相连的最短旅游线段$s=(a_1,a_2)$，长度是 30。和a_2相连的最短边（除了之前连接过的景点）$s=(a_2,a_3)$，长度为 45。和起点相连的获得最短时间的景点是a_1长度为 30。和终点相连获得最短时间的景点是a_3长度为 30。那么候选集合$\{a_1,a_2,a_3\}$的最短下界时间为 235，大于旅游时间限制，因为就不需要再进行全排列路线的时间检查。

4. 无效子集优化

由推论 2 得出，无效子集合的父集合也是无效的集合。尽管这条规则用来作

为剪枝策略减少候选集合，这条规则还可以用来判断候选集合是否是无效的。用一个无效集把所有无效候选集合都存储起来设为US（Invalid Set），当生成新的集合后，检查该候选集合是否有存在子集属于无效集，如果有，那么这个集合也无效的。无效子集检查算法的复杂度为$O(2^n)$，同样小于排列检查复杂度O(k!)。

5. 无效候选集优化

当逐个检查某一阶候选集当中的景点集合时，若剩余的候选集的满意度都小于当前极大解时，余下的候选集合不可能产生极大旅游路线，但是余下的候选集合可能仍然是有效的，不能立即删除。但是对于满意度也小于当前极大满意度，并且没有其他集合能够与之产生新的集合时，这个集合也就不需要再检查其是否有效，可以立刻舍去。如图 4.3，当检查完候选集合$\{a_4, a_3\}$之后，$\{a_4, a_3\}$变成极大候选集合，最大满意度是 22。剩余的三个候选集合$\{a_1, a_2\}$，$\{a_1, a_3\}$和$\{a_2, a_3\}$的满意度都比 22 要小，但是$\{a_1, a_2\}$和$\{a_1, a_3\}$有可能产生新的极大候选集合，所以不能够舍弃。而由于没有候选集合可以和$\{a_2, a_3\}$产生下一阶候选集，所以不需要检查$\{a_2, a_3\}$的时间来判断其是否有效，就可以直接舍弃这个候选集合。

4.4.3 MutipleGuide 算法

这一节将给出 MutipleGuide 算法的伪代码以及核心优化策略和核心构造过程的伪代码，算法以及优化策略的意义见上一节。

（1） MutipleGuide 算法整体框架

算法 4.1 当中，首先初始化 mts,invalidSet,其中 mts 代表极大旅游路线集合，invalidSet 代表无效的景点候选集合（行 1-2）。接着初始化 cas，也就是初始化每一个一阶候选集合（行 3-6）。接着就是排序优化，用来减少后面集合的生成（行 7）。接着进入循环，根据当前阶候选集合，产生下一阶候选集合。接下来检查当中的每一个集合是否有效，为了提高算法的效率，在使用当前阶集合时，首先检查集合当中的花费是否满足时间限制，复杂度为 O（n），可以缩短检查时间。然后再进行下界检查优化，无效子集优化，经过这两个检查都判定为有效的情况下，再进行基本的全排列检查（行 10-13）判断是否是有效候选集合。这三个检查，只要有一个判定为无效，那么这个候选机会就会被删除，并加入到无效集合当中（行

26-27）。

算法 4.1 给出了 MutipleGuide 算法的伪代码

Algorithm 4.1 MutipleGuide(ttc,tcc,M)

Input：游客的旅游时间限制(ttc)，游客的费用限制(tcc)，旅游地图(M)，包括景点集合(A)以及线段集合(S),景点集合当中包括每个景点的停留时间(ST(a))和满意度(DOS(a))和费用(AC(a))。

Output:最优旅游路线集合(Optimal Travel Route)

```
1. Let mts be empty
2. Let invalidSet be empty
3. for every a belong to A and a!=a_u and a!= a_t
4. add a to c.attraction
5. add c to cas
6. end for
7. sortAttractions(cas);
8. while(cas!=null)
9.       for every c belong to cas
10.         if checkCost(c,ttc)
11.             if checkTimeLowBound(cM,ttc))
12.                 if subSetChecking(c,invalidSet,M);
13.                     if  checkValid(c,visit,pm,index,M,ttc,time)
14.                         if(mts==null)
15.                             add c to mts;
16.                         else if TS(c)>mts[0].satisfaction
17.                             mts.clear();
18.                             add c to mts;
19.                         else
20.                             add c to mts;
21.                         end if
22.                         checkUnusedCAS(cas,mts);
23.                        continue;
24.                     end if
25.         end if
26.         add c to invalidSet
27.         cas1.remove(c)
28.       end for
29.      constructCAS(cas,M);
30.    end while
31. otr = skylineQuery(mts)
32. return otr
```

若当前集合是有效候选集合，就尝试把当前集合加入到极大旅游路线集合当中。是否加入的主要依据是当前候选集合的满意度和极大集合当中满意度的对比，若当前的大于极大集合满意度，就用当前候选集合替代极大集合。若等

于，就加入到极大候选集合（行 14-21）。接下来，需要使用无效候选集合优化来去除无效的候选集合（行 22）。当所有集合的有效性都检查完之后，就开始构造下一阶候选集合（行 31），最后当下一阶没有候选集和，跳出循环（行 9）。使用天际线查询来得到 mts 的最优解（行 31）。由于排序优化是一个按照候选集合的费用和旅游时间的排序过程，比较简单，就不进一步写伪代码了。下面给出其他函数过程的伪代码。

（2） 下界检查优化算法

算法 4.2 给出了下界检查优化的伪代码

Algorithm 4.2 checkTimeLowBound(c,M,ttc)

Input:候选集合 c，旅游替代图 M，旅游时间限制 ttc

Output: false if the attraction set is invalid,otherwise true

```
1. Let checkedSet cs be empty
2. Let lowerBound
   lb=min(au,c.attractions)+min(at,c.attractions);
3. for every a in c.attractions
4.     lb += a.stayTime;
5.     add a to checkedSet;
6.     let minTime = MAXIMAL;
7.     for every b in c.attractions and b is not in checkedSet
8.         if TT(a,b)<minTime
9.             minTime = TT(a,b);
10.            link = b;
11.    end for
12.    if checkedSet.size!=c.size
13.        lb += minTime;
14. end for
15. if(lb>ttc)
16.    return false;
17. else
18.    return true;
```

首先将 cs，lb 进行初始化，cs 将存储已经检查过的景点集合。最小下界值 lb 设置为起点到旅游集合景点当中的最小距离与终点到旅游集合的最小距离的和（行 1-2），接下来就逐个检查旅游集合当中的每个景点到其他景点的最短距离，每经过一个景点，累加这个景点的停留时间，并把这个景点加入到 checkedSet 当中(行 3-5)。然后在旅游集合景点当中找到还未检查过的景点，逐一对比检查，筛选出该景点和另外景点的最小距离，并加入到 lb 当中（行 7-14）。当所有景点加入 checkedSet 之后，循环结束，lb 就是该景点集合的最小

下界。如果 lb 超过 ttc，那么这个景点集合就 被判断为无效。如果 lb 没有超过 ttc，那么这个景点将进入下一个检查（行 15-18）。

（3） 无效子集优化算法

算法 4.3 给出了无效子集优化的伪代码,其中 invalidSet 就是把所有检查无效的子集加入到这个集合中，便于以后检查无效集合。

Algorithm 4.3 subSetChecking(c,invalidSet,M)

Input：候选集合 c，无效集合 invalidSet，旅游地图 M

Output:false if the subset of c equals to any set in invalidSet otherwise true.

```
1.    for each subSet ss of c
2.       if(invalidSet.contains(ss))
3.          return false;
4.       end if
5.    end for
6.    return true;
```

逐个检查候选集合 c 的每个子集(行 1),子集生成算法很成熟,就不详细写了。如果发现 invalidSet 包含其中的某一个子集，那么这个候选集合 c 肯定是无效的（行 2-4）。当检查完 c 的所有子集之后仍未找到一个被 invalidSet 包含的集合，那么返回 true，在算法 4.1 当中，将对候选集合 c 做常规时间检查。

（4） 排列检查优化

这是一个使用剪枝优化的递归方式求解全排列的算法，若当前已经生成排列的长度和景点集合的大小相等，则计算这个排列的总共使用时间（行 1-2），一个排列就相当于一个路线。若满足则返回真，否则返回假（行 3-6）。刚才的代码是一个排列生成之后的递归出口，下面进入递归生成代码。visit 存储当前已经访问过的景点，循环遍历 c 当中的每一个景点，若没被访问过，则考虑把当前景点加入当前排列的下一位（行 9-10）。先计算出加入当前景点所需要的时间和，若和小于 ttc，则，继续寻找排列的下一个旅游景点，也就是递归调用 checkValid（行 11-15）。接下来的代码就是处理递归调动返回的结果，以及递归调用完成后需要清除的信息（行 16-21）。

算法 4.4 经过优化的排列检查伪代码，

Algorithm 4.4 checkValid(c,visit,pm,index,M,ttc,time)

Input: 候选集合 c，辅助集合 visit，生成的排列 pm，当前景点的索引,旅游地图 M，旅游时间限制 ttc，当前已经累计的旅游时间 time

Output: true if c is a valid set,otherwise false

```
1. if pm.size==c.size
2.     time += TT(M.getAttraction(index),at);
3.     if time<=ttc
4.         return true;
5.     else
6.         return false;
7.     end if
8. end if
9. for every attraction a in c
10.     if a is not in visit
11.         add a to visit
12.         time2 = time +
    a.StayTime+TT(M.getAttraction(index),a);
13.         if time2<=ttc
14.             pm.add(a);
15.             result =
    checkPerutationOPT(c,visit,pm,a.getIndex,ttc,time)
16.             if result is true
17.                 return true;
18.             pm.remove(a);
19.         end if
20.         visit.remove(a);
21.     end if
22. end for
23. return false;
```

（5）无效集合检查优化

首先把这一阶所有的候选集和都复制到副本 copyCas 当中（行 1），copyCas 是存储还未找到配对的候选集合，初始值所有集合都还未进行配对。接着判断剩余还未被检查的候选集合当中，是否有集合的旅游满意度大于极大旅游满意度，如果没有，那么就进行下一阶候选集和模拟生成（行 2-6）。如果没有，就可以直接返回（行 7，23）。下一阶候选集合生成过程如下，把候选集合当中的景点两两配对(行 8-9)。如果这两个集合当中有一个还未成功配对，那就开始检查这两个集合能够成功配对（行 10）。两个集合若只有一个景点不同，其他景点都相同，那么这两个集合就可以成功配对，并从副本 copyCas 当中去

除这两个集合（行 11-18）。模拟配对完成后，检查是否还有未成功配对的候选集和，把他们从 cas 当中删除（行 21-22）。

算法 4.5 如何进行无效集合优化检查

Algorithm 4.5 checkUnusedCAS(cas,mts)

Input: 当前所有的候选集合(cas),当前极大旅游集 mts

Output: 经过删减的 cas

```
1.    Let copyCas as cas;
2.    for every c in cas and c is unChecked
3.        if c.RouteSatisfaction >= mts.getRouteSatisfaction
4.            exist = true;
5.            break;
6.    end for
7.    if exist
8.        for every c1 in cas
9.            for every c2 in cas and c2!=c1
10.               if copyCas.contains c1 or copyCas.contains
   c2
11.                   Let canJoin=true;
12.                   for i be 0 to c1.size()-2
13.                       if c1.getAttraction(i) !=
   c2.getAttraction(i)
14.                           canJoin = false;
15.                           break;
16.                   end for
17.                   if canJoin
18.                       remove c1,c2 from copyCas
19.           end for
20.       end for
21.       if copyCas.size >0
22.           remove a in copyCas from cas
23.   return cas;
```

4.5 旅游路线规划算法实验

本节主要通过实验展示 BF(Brute Force),TripMine,MutipleGuide 三种算法在运行时间和内存占用两个方面的对比。Brute Force 是使用纯粹的子集生成加全排列检查策略，复杂度为$O(\sum_{i=1}^{n}(c_n^i \times n!))$，效率非常低，根本没法做实验。本文采取经过优化的 BF 算法来进行实验，其在运行时间方面要优于原始的 BF 算法。表 4.4 显示了本文三种算法所使用的优化策略和求解过程，对应优化策略的

解释参照 4.3 节。本文工作的实验环境为 MAC OS X 系统，处理器为 Intel Core i5 四核处理器，CPU 主频为 2.6GHZ，内存为 8GB。

表 4.4 三种算法主要策略对比

算法	优化策略
BF	下界检查优化，排列检查优化。求出所有子集，并逐一检查每个子集的排列。
TripMine	排序优化，下界检查优化，无效候选集优化。从一阶候选集一直生成到 k 阶候选集，并逐一检查每一个候选集合的全排列。
MutipleGuide	排序优化，下界检查优化，无效候选集优化，排列检查优化，无效子集优化。从一阶候选集逐步生成下一阶候选集合，并在中途利用无效子集来减小候选集合，利用排列优化来降低排列检查复杂度。

4.5.1 实验数据和实验参数

本文从真实的旅游文记当中提取出地理名词，在纠正了错误语义的地理名词之后，重新获得地理名词的一系列地理特征，其中的经纬度就是实验当中需要用到的数据。本文的使用的旅游景点全都是在美国的夏威夷洲，在每次实验当中，都选择两个相对比较远的地点作为旅游路线的起点和终点。表 4.5 列出了文本的实验参数。在所有实验当中，前三个参数都是变化的，并且本文每次固定两个参数，变化第三个参数来查看算法的运行结果。可变化参数当中，标记有下划线并表示为黑体的数值，代表针对其它参数做实验时，这个参数的默认值。后四个参数所有实验都是共同享有的，与每一个实验无关。景点的评分和费用使用模拟器生成相应的数值，景点的评分根据景点出现的次数以及用户的偏好归一化生成。

表 4.5 实验参数

参数	取值范围
景点个数	10,20,**30**,40,50,60,70,80
费用范围(dollar)	0,20,40,60,**80**,100,120,140,160,180,200
时间范围(minute)	30,60,90,120,150,180，**210**,240,270,300,330,360,390,420,450,480
地图大小(KM^2)	60
景点停留时间(minute)	30-100
景点评分	1-10
景点费用(dollar)	0-50

4.5.2 实验展示

本文针对算法运行的时间和内存，分别在三种场景下对这三种算法进行了六组对比实验：

(1)景点数目不同费用大小和时间相同的内存占用和运行时间比较

(2)费用大小不同景点个数和时间相同的内存占用和运行时间比较

(3)时间不同费用大小和景点数目相同的内存占用和运行时间比较

在以上三种场景下，本文将统计三种算法的运行时间和内存占用情况，从而证明本文提出的算法不仅优于原始的 BF 算法，并且比经过优化的 TripeMine 算法更加正确和高效。 由于每次实验所得出的运行时间和内存占有稍有不同，为了消除随机性误差，本文的大多数实验数据都是经过了 10 次运行得到平均值取整的结果。对于每一组实验，本文首先描述实验结果，然后给出实验统计表格，并展示曲线图。

(1)景点数目不同费用大小和时间相同的运行时间对比实验

第一组实验比较特殊，本文分成了五个算法来对比，这五个算法分别是 MG（MutipleGuide），TMSS（Trip Mine with Satisfaction Optimal）,TMPE(Trip

Mine with Permutation Optimal),TM(Trip Mine),BF(Brute Force)。其中 TMSS 对应本文提出的无效候选集合优化，TMPE 对应本文提出的排列检查优化。后面算法的英文简写和刚才描述的一致。本实验的时间显示为 210 分钟，费用限制为 80 美元。实验结果如图 4.4 和表 4.6。BF 算法由于运行时间随着景点个数增长太高，无法在图中显示，当景点个数达到 40 个时，BF 算法的运行时间就达到了 176050274ms，也就是两天多的时间。从图 4.4 可以看出，前四种算法都优于 BF 算法。而前四种算法在景点数目达到 60 个之前，运行时间都很接近。景点数目超过 60 个之后，本文提出的三种算法的运行时间主键少于 TM 算法。并且加入了两种优化策略的 MG 算法运行时间是最短的，比 TM 算法少了十分之一左右。通过这个实验表明，本文提出的算法在旅行时间和旅游费用相同的情况下，不仅本文提出的优化策略能够降低原本算法的运行时间，本文提出的 MG 算法也优于 TM 算法。接下来的五个实验，不在统计 TMSS，TMPE 的结果。

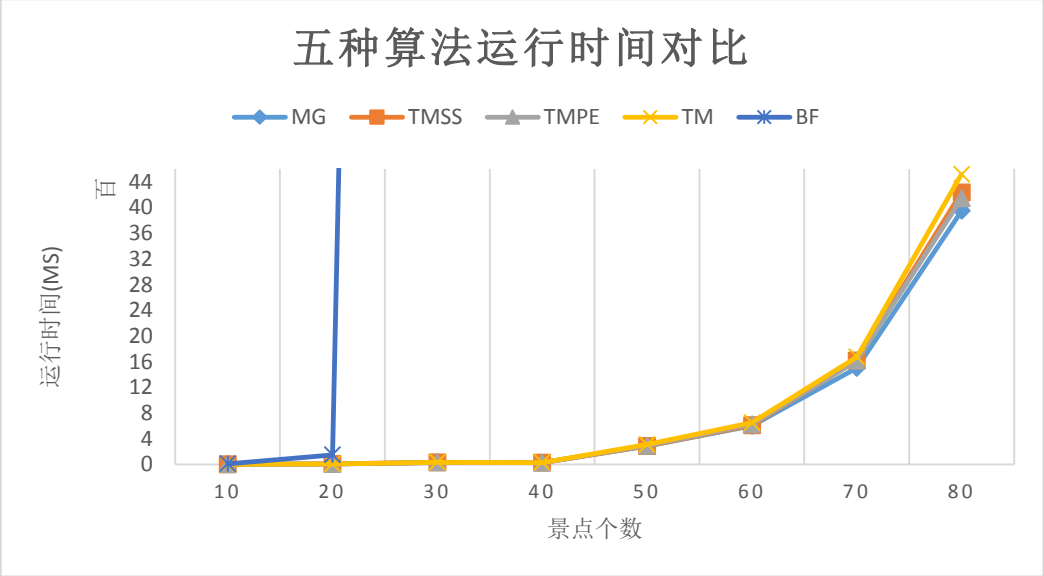

图 4.4 五中算法在景点个数不同情况下运行时间

(2) 费用大小不同景点个数和时间相同运行时间对比实验

这次试验 MG 和 TM 算法都是在景点为 50 个时进行实验，实验结果如表 4.6 和图 4.5。由于 MG 和 TM 算法在景点为 30 个时，时间区别较小，不方便展示。

表 4.6 费用不同三种算法的运行时间

费用限制/dollar	0	20	40	60	80	100
MG	1	8	47	170	258	284
TM	1	5	48	175	278	307
BF（att30）	93649	94540	90401	100806	106949	104815
费用限制/dollar	120	140	160	180	200	
MG	285	288	291	281	283	
TM	303	305	310	308	317	
BF（att30）	112796	111243	112612	107876	103013	

另外，由于 BF 在景点为 50 个时，需要数个月甚至数年才能运行完，所以 BF 算法的景点个数是 30 个，尽管景点个数少，其运行的时间由于大大高于另外两种算法，无法在折现图当中很好显示，因此给出了这次实验的统计数据，如表 4.6，时间单位为 ms。从表格当中可以看出，BF 算法的时间稳定在 112796ms，远远大于 MG 和 TM 算法的运行时间。根据图中的数据，当费用在 60 美元之前，两种算法的运行时间比较接近，当费用继续增加时时，MG 算法的时间逐渐小于 TM 算法。最后 MG 算法的时间平均比 TM 的少十分之一。实验结果表明，本文提出的 MG 算法在费用变大的情况下，运行时间少于 TM 算法。

(3)时间不同费用大小和景点数目相同的运行时间对比实验

实验结果如图 4.6，图中显示，BF 算法的运行时间随着旅游时间的改变变化不大，稳定在一百万毫秒左右。和前面两次实验结果类似，MG 算法和 TM 算法在前期运行时间较短时，区别不大。随着游玩时间的增加，当游玩时间超过 330 小时之后，MG 算法的时间逐渐少于 TM 算法，并且稳定少于五分之一左右。实验表明，当时间增加时，MG 算法表现出更加高效的性能。

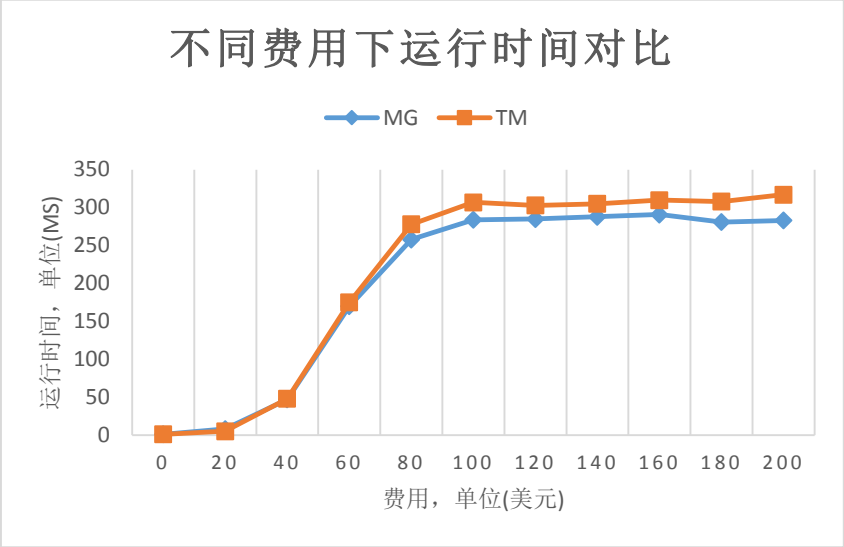

图 4.5 两种算法在费用不同情况下的运行时间

(4) 景点数目不同费用大小和时间相同内存占用对比实验

本实验的费用限制为 80 美元，时间限制为 210 分钟。实验结果如图 4.7，从图中可以看出，BF 算法的内存占从景点为 20 开始，就高于其他两个算法，并一直高于其他两个算法。当景点个数为 20 到 30 个是，MG 算法的内存约为 TM 算法内存的一半，随着景点个数继续增加，MG 所占内存始终少于 TM 算法，大于少占用十分之一。

(5) 费用大小不同景点个数和时间相同内存对比对比实验

本实验的景点个数为 30 个，旅游时间限制为 210 分钟。实验结果如图 4.8，从图中可以看出，BF 算法的内存占用不收费用增长的影响。而随着费仲的增加，MG 算法和 TM 算法的内存占用都出现了起伏。费用小于 80 时，MG 算法内存占用小于 TM 算法，当费用高于 80 之后，两种算法的内存占用起伏不定。这是由于在候选集合产生的过程当中，加入了费用检查优化，当费用不同时，不同的集合会被筛选掉，从而可以产生的新一阶的候选集合也不同。尽管内存上下波折，两种算法的内存占用仍然远远小于 BF 算法。从这个实验当中得出，MG 算法的内存占用在费用小于 80 时，优于 TM 算法，费用大于 80 时，内存占用变化无规律。

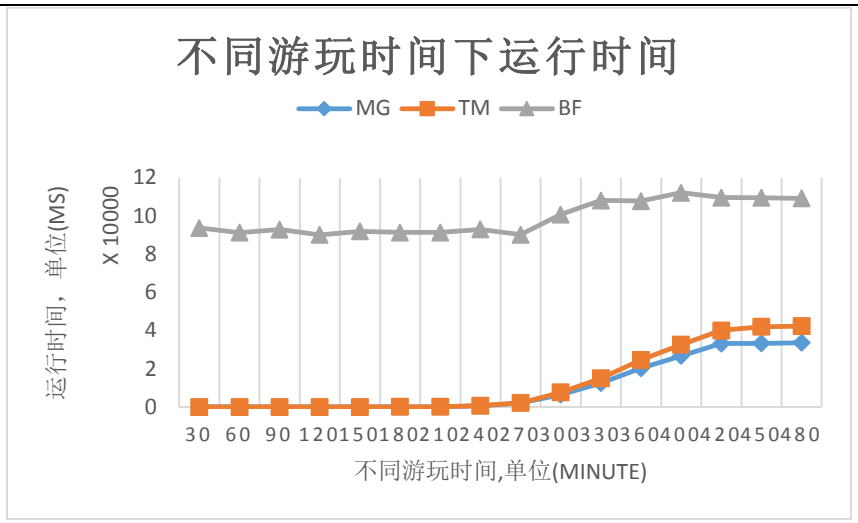

图 4.6 三种算法在时间不同情况下的运行时间

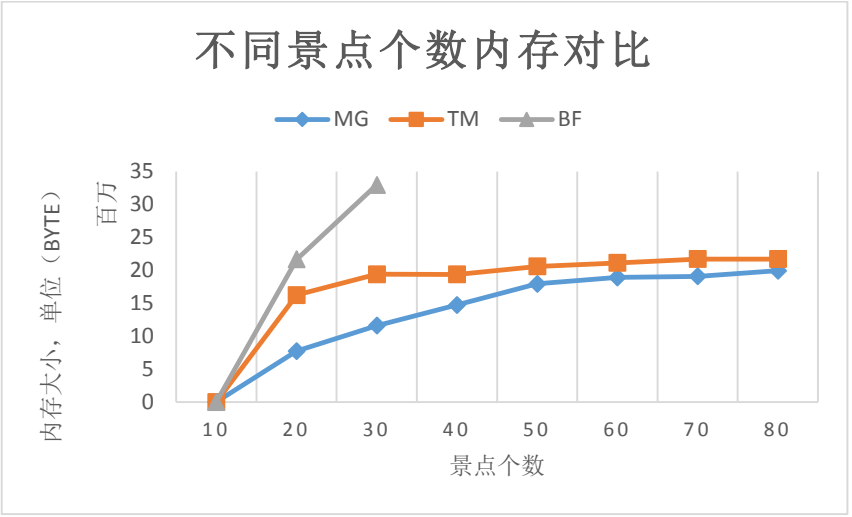

图 4.7 三种算法在景点个数不同情况下的内存占用对比

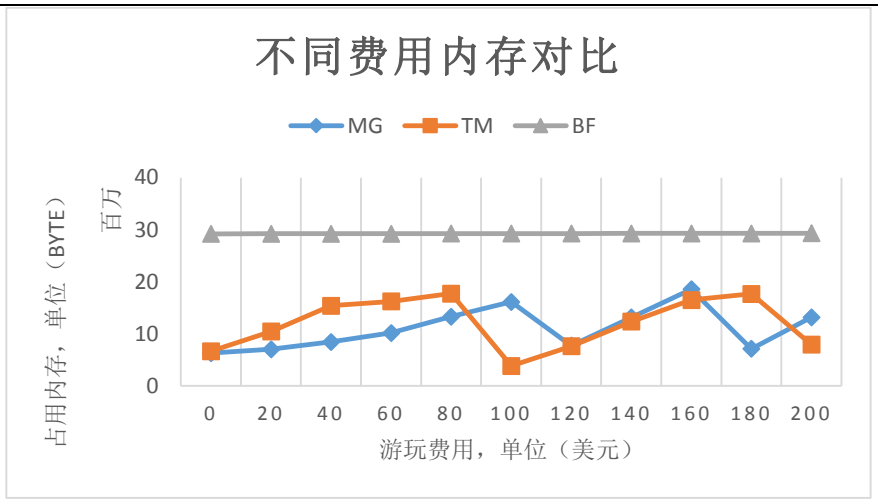

图 4.8 三种算法在旅游费用不同情况下的内存占用对比

(6)时间不同费用大小和景点数目相同内存对比实验

本实验的景点个数为 30，旅游费用显示为 80 美元。实验结果如图 4.9，从图中可以看出，BF 算法和上一个实验规律一样，随着旅游费用的增加，稳定在 45M 左右，并且远远高于其他两个算法。当旅游时间小于 150 时，TM 和 MG 内存占用相近，当旅游时间达到 180 分钟时，TM 算法有一个快速增加斜坡，最后占用内存和 BF 算法相同。而 MG 算法的内存占用增长坡度在 360 分钟时来临，并且幅度也小于 TM 算法。从图中可以得出，当旅游时间增加时，MG 算法的内存占用明显优于 TM 算法。

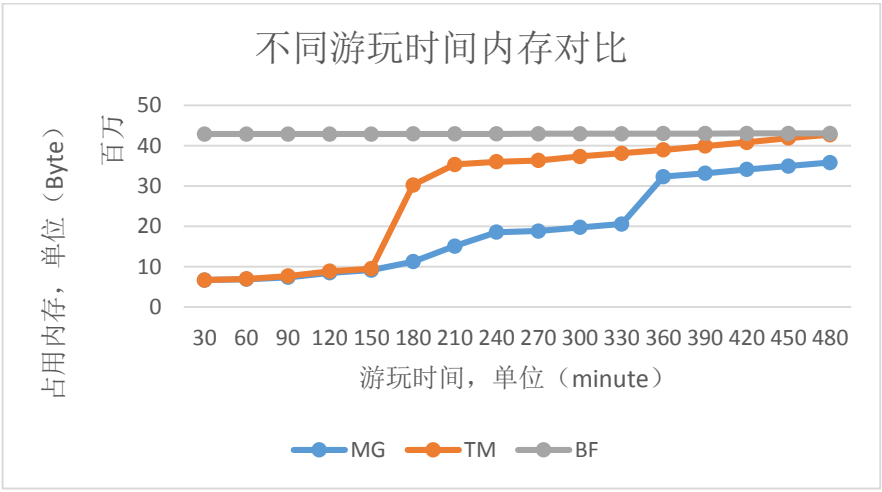

图 4.9 三种算法在旅游时间不同情况下的内存占用对比

综合以上六组实验，可以得出以下结论，自身性能方面 MutipleGuide 算法在景点个数小于 40 个时，运行时间都小于 1 秒钟，运行时间高效。景点个数小于 40 个时，运行内存也在 20M 以下，占用内存也不大。另外，当旅游时间增加到 480 分钟，旅欧费用增加到 200 没眼，Mutipeuide 算法都能够高效运行。横向对比方便，在六组实验当中，在运行时间和占用内存上都极大优于 BF 算法。和 TM 算法对比上，在景点个数小时，性能基本一致。当景点个数增多，旅游时间增加或者旅游费用增加时，MutipleGuide 算法在运行时间和占用内存上都优于 TM 算法。

4.6 本章小结

这一章主要着重讲解了旅游路线规划算法。在 4.1 和 4.2 节当中讲解了旅游路线规划算法的应用场景，和问题的模型和目标。在 4.3 节当中讲解了本算法的必要定义和作为基础的性质定理。在 4.4 节当中描述了算法使用的优化策略并提出 MutipleGuide 算法。最后在 4.5 节当中，就 MutipleGuide 算法和其他两种算法在时间和内存使用方面进行了实验对比，表明本算法的高效性。

第5章 系统设计与展示

基于前面章节所描述的算法，本文实现了一个在线旅游景点以及行程路线规划系统。

5.1 系统功能和框架

本系统基于 J2EE 技术，实现了前文所描述的算法，并且把算法封装成功能模块融入到系统当中。这一节介绍整个系统的功能和架构。

5.1.1 功能设计

根据前文所描述的算法，本系统有四个大的功能，分别是热门旅游景点推荐，关联景点推荐，旅游行程规划，旅游路线规划。每一个功能都配以景点选择与存储和地图展示功能，增加用户体验。本文采用 google 地图 API 来实现地图展示。图 5.1 描述是系统功能组成。系统中四大功能所采用的算法在前面章节当中已经详细讲解，本章着重系统的使用和展示。

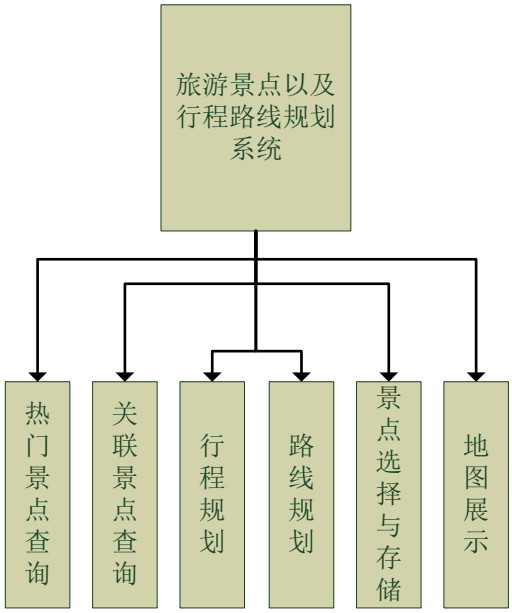

图 5.1 系统功能图

5.1.2 系统架构

本在线旅游系统总体采用 J2EE 技术，使用传统的 MVC（Model View Control）架构实现，是业务，数据和界面三层分离，架构图如图 5.2 所示。在客户层，也就是 MVC 的 View 层，以 HTML 和 JSP 网页形式进行客户端内容展示，使用了 JAVASCRIPT 技术支持与后台进行异步交互，使用 AJAX 技术进行局部刷新。在应用层，也就是 MVC 的 Control 层，使用 Java Servlet 来相应客户端的请求，调用相关的 java 类来完成业务逻辑，并以 json 对象形式返回数据。在数据层，也就是 MVC 的 Model 层，使用 Hibernate 技术封装数据访问逻辑，负责完成业务逻辑的数据读写任务。最后数据存储在 Mysql 当中，Hibernate 通过 JDBC 来与数据库进行交互。

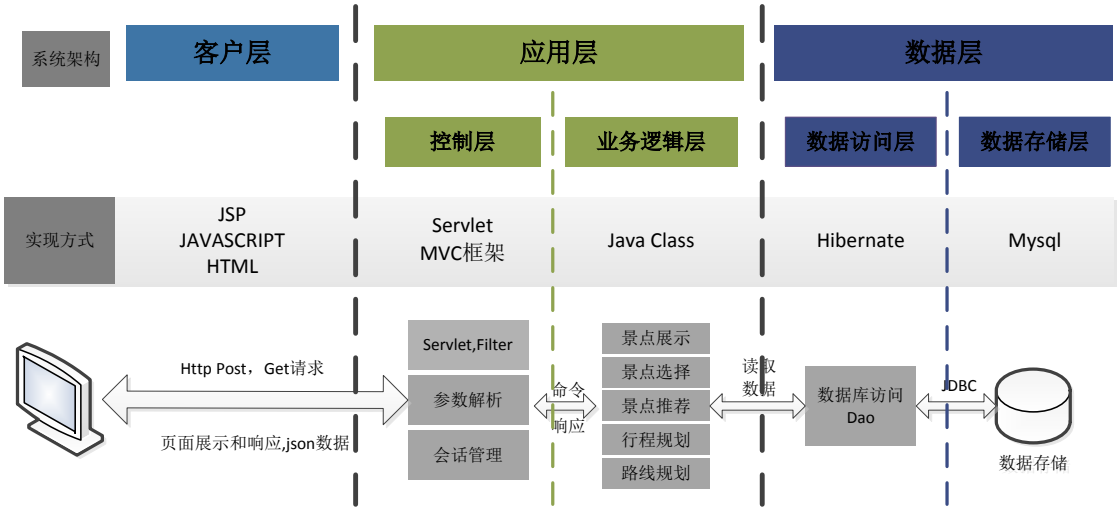

图 5.2 系统架构图

5.2 数据库设计

本文工作所设计的数据库主要包含两部分，一部分是为处理旅游文记以及地理名词相关任务所设计的数据库，这部分数据库包含旅游文记，详细的地理名词信息，地理名词提取过程需要的中间缓存信息等。这部分属于数据预处理信息，在本系统当中并不直接用到。第二部分就是本在线旅游系统所直接用到的信息，

其信息是通过第一部分所设计的数据库数据通过本章所写的算法产生，提供给在线旅游系统专用，包括景点信息和景点关联规则。这样在进行景点查询时，就直接到数据库当中获取数据，省掉了多余的算法计算时间。

下面将展示本文所设计的主要数据库表格，属于数据预处理的有旅游文记（表 5.1）和详细地理名词（表 5.2），属于在线旅游系统的有旅游景点（表 5.3）和旅游景点关联规则（表 5.4）。

1. 旅游文记（travelogue）信息

如表 5.1 所示，doc_id 是其主属性，其中还 country，geo_info 代表其中地理位置相关信息，表明和这篇文章相关的地理位置，author 就是用户。Author 是协同过滤的必要数据。text 就是文记的字符串信息，word_count 是旅游文记当中的单词统计信息。

所有旅游文记相关的信息都存储在 travelogue 表当中，本文提取了美国所有洲的旅游文记信息，在本系统当中，只使用了夏威夷洲的数据来进行展示。

表 5.1 travelogue 的主要属性

属性名	属性类型
doc_id	int(18)
country	varchar(50)
geo_info	varchar(180)
title	varchar(500)
source	varchar(20)
author	varchar(200)
text	longtext
word_count	int(18)

2. 地理名词（my_location_sense）详细信息

最终包含了地理名词详细信息的是 my_location_sense，它主要负责详细解释一个地名词，其表的数据项如表 4.3。经纬度是这个地点的地理位置，senseFeature 是代表这个地方的主要地形，而 senseFeatureDetails 代表这个地方的人口，海拔，行政级别等。对于 locationSense，不一定能够都找到其解释。对于 LocationSenseId, 0 表示查询有结果，但是没有全匹配的。 -1 表示没有查询到任何结果；-3 表示在同一篇文档中和之前的重复，记录在 duplicatelocationSense 中;-4 表示这篇文档没有 name，也就是无法有 locationSense。

表 5.2 my_location_sense 的数据项和数据类型

属性名	属性类型
locationSenseName	varchar(100)
latitude	double
longitude	double
locationSenseTime	int(18)
locationSensePriorProbability	double
senseFeature	varchar(100)
senseFeatureDetails	varchar(100)
doc_id	int(18)
word_id	int(18)
originalName	varchar(50)
adoptedName	varchar(50)

3. 旅游景点（WebGeoname）信息

这张表格存储的是已经计算好的热门旅游景点的信息，专门提供给在线旅游系统进行数据访问，取了夏威夷洲的数据进行计算。WebGeonameId 作为旅游景点的主属性，WebGeonameContent 是旅游景点的全线名称，Latitude 和 Longitude 就是经纬度，Occurrence 是其在文章中出现的次数，也代表了景点的热度。

4. 旅游景点关联规则（WebGeonameContent）信息

这张表格同样是专门给在线旅游系统使用，是通过关联规则算法计算得到的结果。

WebRuleAttId 是其主属性，WebRuleAttRules 是相关联的一组景点，存储是这组景点的 Id,以逗号隔开。景点 Id 取自 WebGeoname 当中的主属性 WebGeonameId。Occurrence 是这组景点出新过的次数，也就是关联规则算法当中的支持度。ruleSize 是这个规则有多少个景点，方便统计。文章中描述的另外两个算法，由于只能够根据用户所选择的景点实时生成，所以没法提前计算好并存入数据库当中。用户在使用时，是经过行程规划算法和路线规划算法计算得到的结果。

表 5.3 WebGeoname 的数据项和数据类型

属性名	属性类型
WebGeonameId	int(18)
WebGeonameContent	varchar(100)
Latitude	double
Longitude	double
Occurence	int(18)

表 5.4 WebGeonameContent 的数据项和数据类型

属性名	属性类型
WebRuleAttId	int(18)
WebRuleAttRules	varchar(100)
Occurence	int(18)
ruleSize	int(18)

5.3 系统功能展示

这一节将展示系统当中的每一个功能，包含四个主要功能。四个功能的系统界面详细，包含三个主要模块，分别是地图显示，景点信息选择和展示，已选择景点操作。如图 5.3 是热门旅游景点展示的系统界面，其他三个功能界面和图 5.3 所展示的类似。橙色框内是所有内容展示的区域，正上方黑色加粗大字表示所处

于的功能，左下角蓝色框内会显示旅游景点的相关信息，右下角绿色框内显示景点在地图上的信息。

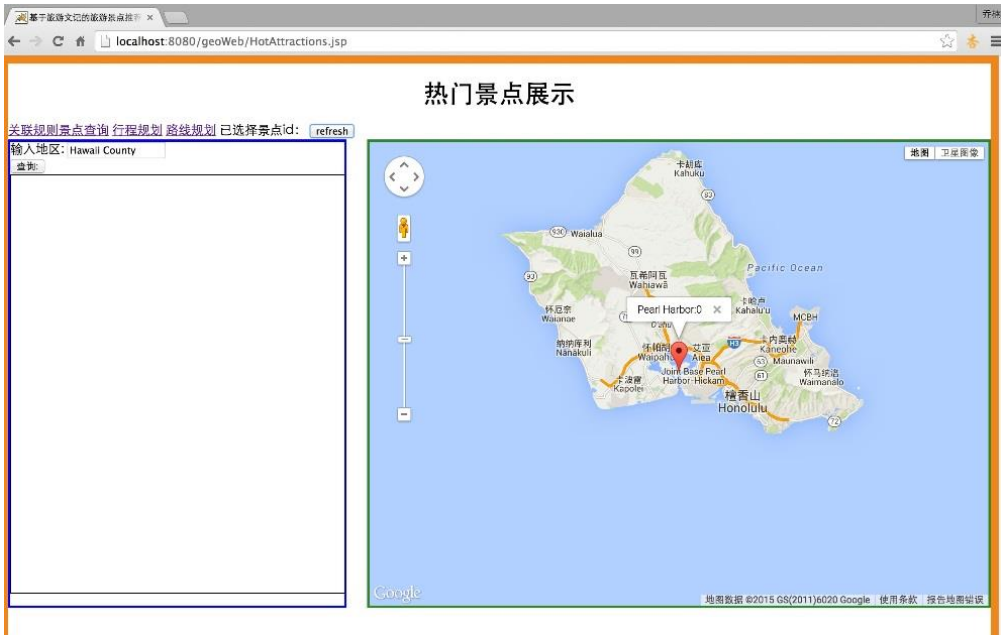

图 5.3 系统界面展示

5.3.1 旅游景点查询

旅游景点推荐分为热门旅游查询和关联旅游景点查询。热门旅游景点，就是查询某一个区域内的热门旅游景点。用户输入某一个区域，比如 Hawaii County，就可以得到这个区域内的最热门旅游景点。并在地图上显示相应的位置。关联规则景点查询，用户输入一个景点名词，就可以得到和这个景点有关联关系的景点，同样在地图上显示其位置。

1. 热门旅游景点查询

根据行政区域划分，可以查询郡，洲和国家以内的热门旅游景点。本节以 Hawaii County 为例，查询 Hawaii 郡以内的热门旅游景点，查询结果如图 5.4 所示。使用 Ajax 技术，可以使蓝色和绿色框当中的信息会被局部刷新

蓝色框内会出现一个表格，表格当中显示最热门的 20 个景点，表格当中每一列的内容如表格第一行所指示，在这就不详细讲解。表格的最后一列有一个 select

按钮，可以供用户来选择景点。被选择的景点会存放在 jsp 的 session 当中，并在“已选择的景点 id”之后，显示用户已经选择的景点。从图 5.4 来看，用户还未选择任何一个景点。右下角绿色框内显示最热门的 10 个旅游景点的位置，并用红色气球进行标记，气球上会显示这个景点的 Id，景点的单名（不包含其比起大的行政区的名字），以及这个景点的热度排名。

图 5.4 热门旅游景点查询结果

2. 景点选择，删除和刷新

景点选择，删除和刷新功能在后面三个功能当中也会用到，在这里先讲解，后面用到功能相同。现在点击“add”按钮，选择三个景点，在“已选择的景点 id”之后会出现所选择景点的 id，但是绿色框内地图的景点标记仍然没有变化。如图 5.5 所示。

另外，用户想删除某个景点，只需要点击景点 id 上的“delete”删除按钮，用户就可以删除所选择的景点。对于已选择的景点，若想看看这些景点的分布位置，那就点击“refresh”按钮，就可以看到所选择景点在地图上的分布，如图 5.6 所示。

3. 关联规则景点查询

图 5.4 当中有另外三个功能页面的链接，点击关联规则景点查询，即跳到干练规则景点查询页面。关联规则景点查询界面和热门景点查询结构一样，需要输入

的数据不同。输入想要查看的景点之后，便可得到关联规则景点查询的结果，同样在蓝色框当中以表格形式显示。

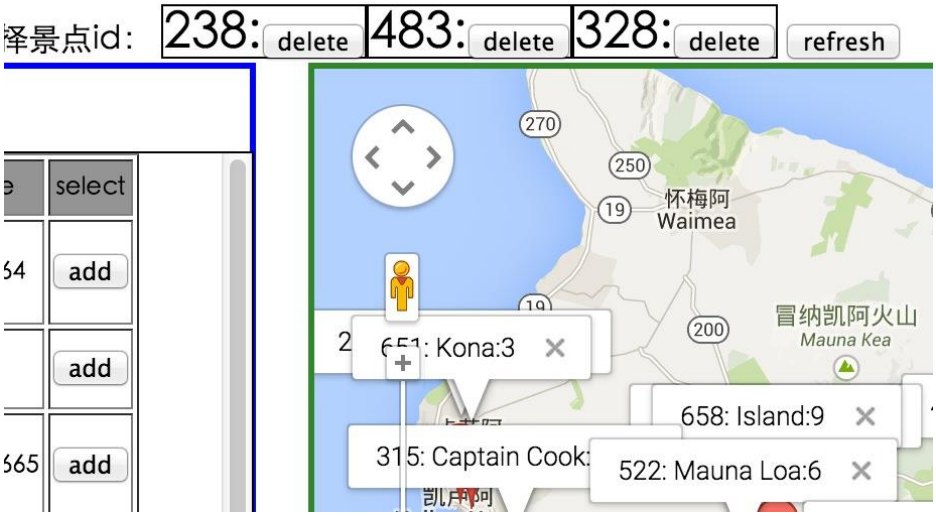

图 5.5 景点添加删除功能

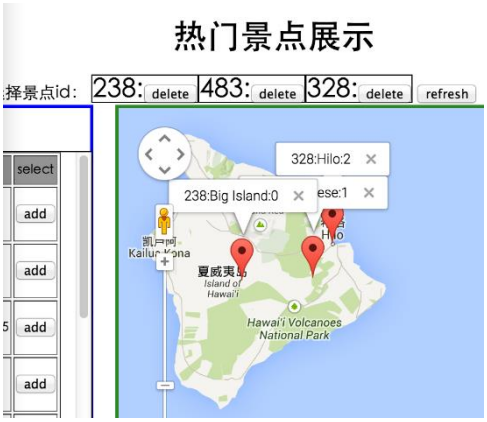

图 5.6 刷新已选择景点结果

图 5.7 是关联规则景点的查询结果，用户输入“Hilo”点击查询，会得到两个表格。第一张表格显示包含“Hilo”的关联规则，第二张表格显示关联规则当中的景点详细信息。在表格展示的同时，右边的地图也会显示第二张表格当中景点的信息。

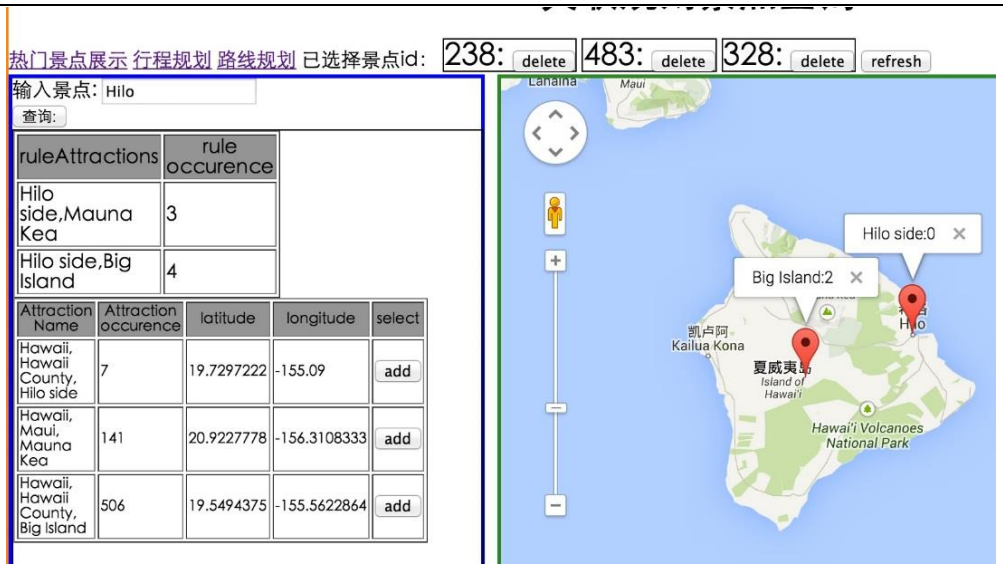

图 5.7 关联规则旅游景点查询结果

5.3.2 旅游行程规划

点击行程规划，就可以到行程规划页面，用户输入需要安排的天数之后，就可以得到行程规划的结果。如图 5.8 所示，用户已经选择了 10 个景点，并安排三天之内游玩这三个景点，那么在蓝色框当中会得到三天行程的列表，在右边绿色框当中会得到三天行程每一天的分布。其中安排在一天当中的行程用红色线连接在一起。

图 5.8 行程规划结果

若对于景点的选择不满意，用户可以灵活删除某个景点，再得到新的行程规划列表和景点地图分布，在这就不进行图片展示。

5.3.3 旅游路线规划

点击路线规划，来到路线规划界面，和前面所讲界面类似。路线规划就是在选择起始点的情况下，在时间和费用的限制下，在一系列景点当中选出一条满意度最高的路线。由于限制因素不止一个，所以根据天际线算法筛选出最优的一系列解。因此，旅游路线规划需要用户输入所允许的最大时间，最能承受的最大费用，以及起点和终点的位置。算法当中并不会考虑起点和终点所消耗的时间和费用。

图 5.9 景点的起始点选择

点击“选择起始点按钮”，会显示景点的详细列表，在列表的最后一列，可以让用户选择景点的起始点，用户输入时间和费用限制后，就可以开始进行路线规划，如图 5.9 所示。点击 “路线规划”按钮就能够得到路线规划的结果，如图 5.10 所示。蓝色框展示每条路线的详细信息，地图上红色线显示一条从起点到终点的最优路线。

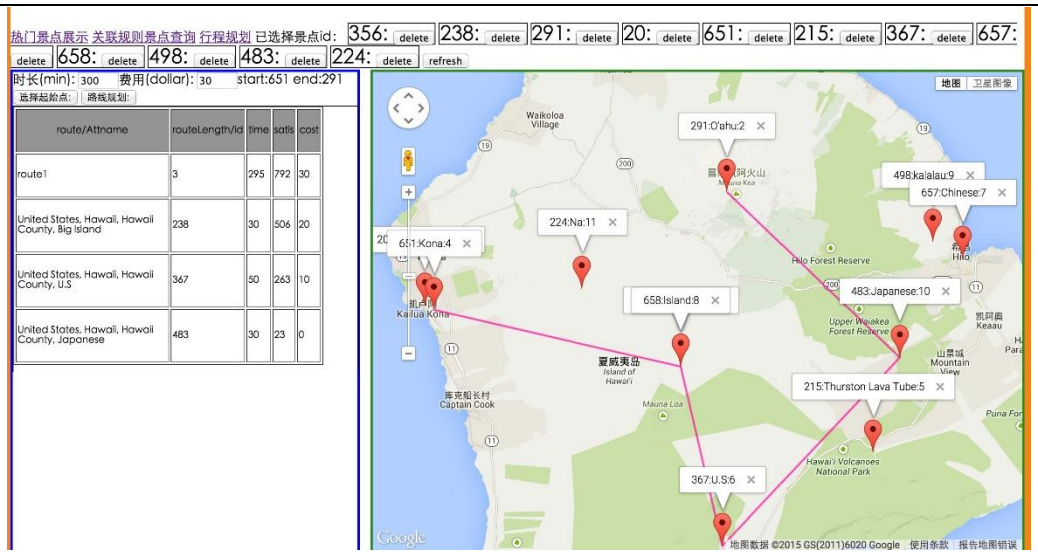

图 5.10 旅游路线规划结果

5.4 本章小结

本章描述在线旅游景点推荐以及行程规划系统的设计与展示，主要用来验证前面所描述算法在实际使用当中的效果。在 5.1 节当中给出了在线旅游系统的功能和整体架构，在 5.2 节当中给出了支持本文整个系统所进行的数据库设计。5.3 节当中，从一个用户的角度，详细描述了本系统每个功能的使用过程，并配以系统展示图片展现了系统所给出的结果。整个系统表现出良好的有效性和可用性。

第6章 总结与展望

6.1 本文工作与贡献

在旅游行业持续稳定增长的大背景下，本人为了解决游客在旅游过程当中所面临的一些问题，设计了一个系统来解决这些问题。面对旅游文记大量增长的情况，本系统有专门的文本挖掘模块，可以提取出旅游文记当中的地理名词从而获取其中的旅游景点。利用所获得的旅游景点的地理信息以及旅游景点和旅游文记的关系，实现了旅游景点推荐，实现了旅游行程规划和旅游路线规划。针对旅游行业当中的一个难题旅游路线规划问题，本人在深入研究了相关的文献之后，根据旅游的实际情况，提出了 MutipleGuide 算法，该算法考虑费用和时间两个因素求解最优的旅游路线，并且算法非常高效。

总结起来，本文的主要贡献有如下四点:

(1) 设计并实现了一个交互式的旅游景点推荐及行程路线规划系统。

这个系统以大量的旅游文记数据来源，实现了从旅游文记到旅游景点推荐及行程路线规划的整个流程。首先需要对旅游文记进行文本挖掘，提取其中的地理名词和旅游景点，其次，要把旅游景点的详细信息保存起来，用来进行热门旅游景点推荐。另外，要把旅游景点之间的关系保存起来，用来做关联性旅游景点推荐。最后要使用合适的算法达到路线规划和行程规划的目的，在使用当中系统和地图实时结合起来，通过系统展示证明了系统的有效性和易用性。

(2) 设计并实现了旅游路线规划算法 MutipleGuide 算法

旅游路线规划算法以满意度为优化目标，找出一条在时间，费用等因素限制下，让用户最满意的路径。该算法是一类 NP 难问题，因此解决的思路很多。另外，在满意度值的选择方面，也有很多因素考虑。MutipleGuide 算法可以同时考虑时间和费用的因素，并且起点和终点可以任意设置。让游客在选择路线时，可

以更加迎合自身的需求。

(3) 通过实验验证了 MutipleGuide 算法的高效性

本文做了大量的对比试验，首先让 MutipleGuide 算法与未优化的算法进行纵向对比，然后再让 MutipleGuide 算法与 BrouteForce 和 TripeMine 算法进行横向对比，不仅对比运行时间，还对比内存的占用情况。实验结果表明，MutipleGuide 算法不仅可以在景点个数，旅游时间，旅游费用等数量较大时获得理想的运行效果，并且在运行时间和内存占用上仍然展现其高效性。

(4) 应用合适的算法来实现旅游景点推荐和路线规划

针对旅游景点推荐的算法很多，本人结合地理本体树理论实现了层级化的热门景点推荐，结合关联规则算法实现了关联规则景点推荐，结合最小生成树聚类算法实现了旅游行程规划。以本文当中的数据进行实验，表明了本人所使用算法的有效性。在与系统结合时，亦展现了本算法的可用性。

6.2 未来研究展望

尽管本文所实现的系统解决了旅游当中的一些问题，未来仍然可以从以下几个方面进一步探究：

(1) 考虑从更多方面进行旅游景点推荐，比如借助旅游文记作者的个人信息和旅游习惯进行个性化旅游景点推荐。加入读者的评价来判断所推荐景点的好坏，选择最优的景点推荐算法。

(2) 在从旅游文记到旅游景点和旅游路线方面的流程，可以改为数据实时更新系统。现在使用离线更新系统，更新过程需要重新跑算法。改为实时更新可以节省时间。

(3) 在考虑用户对旅游景点的满意度方面，可以考虑跟多的因素，例如游客的个性化喜好，游客对景点的初步打分等。

(4) 找出更多的优化策略，进一步提升旅游路线规划算法的运行速度

参考文献

[1] Majid A. 基于地理标签的社会媒体数据挖掘的智能旅游推荐研究[D]. 浙江大学, 2012

[2] 尹华罡. 基于海量时空数据的路线挖掘与检索[D]. 中国科学技术大学, 2012

[3] Toyama K, Logan R, Roseway A. Geographic location tags on digital images[C]//Proceedings of the eleventh ACM international conference on Multimedia. ACM, 2003: 156-166.

[4] Xie L, He X. Picture tags and world knowledge: learning tag relations from visual semantic sources[C]//Proceedings of the 21st ACM international conference on Multimedia. ACM, 2013: 967-976.

[5] 余新伟. 在线旅游行程规划系统关键技术研究与实现[D]. 西安电子科技大学. 2013

[6] Yin H, Lu X, Wang C, et al. Photo2trip: an interactive trip planning system based on geo-tagged photos[C]//Proceedings of the international conference on Multimedia. ACM, 2010: 1579-1582.

[7] Lu X, Wang C, Yang J M, et al. Photo2trip: generating travel routes from geo-tagged photos for trip planning[C]//Proceedings of the international conference on Multimedia. ACM, 2010: 143-152.

[8] Hao Q, Cai R, Wang C, et al. Equip tourists with knowledge mined from travelogues[C]//Proceedings of the 19th international conference on World wide web. ACM, 2010: 401-410.

[9] Zhang S, Zhang R, Liu X, et al. A Personalized Trust-Aware Model for Travelogue Discovering[C]//Web Intelligence and Intelligent Agent Technology (WI-IAT), 2012 IEEE/WIC/ACM International Conferences on. IEEE, 2012, 3: 112-116.

[10] Kabassi K. Personalizing recommendations for tourists[J]. Telematics and Informatics, 2010, 27(1): 51-66.

[11] Huang Y, Bian L. A Bayesian network and analytic hierarchy process based personalized recommendations for tourist attractions over the Internet[J]. Expert Systems with Applications, 2009, 36(1): 933-943.

[12] Zheng Y, Xie X. Learning travel recommendations from user-generated GPS traces[J]. ACM Transactions on Intelligent Systems and Technology (TIST), 2011, 2(1): 2.

[13] Horozov T, Narasimhan N, Vasudevan V. Using location for personalized POI recommendations in mobile environments[C]//Applications and the Internet, 2006. SAINT 2006. International Symposium on. IEEE, 2006: 6 pp.-129.

[14] Takeuchi Y, Sugimoto M. CityVoyager: an outdoor recommendation system based on user location history[M]//Ubiquitous intelligence and computing. Springer Berlin Heidelberg, 2006: 625-636.

[15] Zheng Y, Chen Y, Xie X, et al. GeoLife2. 0: a location-based social networking service[C]//Mobile Data Management: Systems, Services and Middleware, 2009. MDM'09. Tenth International Conference on. IEEE, 2009: 357-358.

[16] Li Q, Zheng Y, Xie X, et al. Mining user similarity based on location history[C]//Proceedings of the 16th ACM SIGSPATIAL international conference on Advances in geographic information systems. ACM, 2008: 34.

[17] Zheng Y, Chen Y, Li Q, et al. Understanding transportation modes based on GPS data for web applications[J]. ACM Transactions on the Web (TWEB), 2010, 4(1): 1.

[18] Zheng Y, Zhang L, Ma Z, et al. Recommending friends and locations based on individual location history[J]. ACM Transactions on the Web (TWEB), 2011, 5(1): 5.

[19] Agraval R, Srikant R. ‘Fast Algorithms for Mining Association Rules in Large Data Bases[C]//20th Inter-national Conference on Very Large Databases, Santiagom. 1994.

[20] Han J, Pei J, Yin Y. Mining frequent patterns without candidate generation[C]//ACM SIGMOD Record. ACM, 2000, 29(2): 1-12.

[21] Pei J, Han J, Lu H, et al. H-mine: Hyper-structure mining of frequent patterns in large databases[C]//Data Mining, 2001. ICDM 2001, Proceedings IEEE International Conference on. IEEE, 2001: 441-448.

[22] Liu J, Pan Y, Wang K, et al. Mining frequent item sets by opportunistic projection[C]//Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining. ACM, 2002: 229-238.

[23] Pasquier N, Bastide Y, Taouil R, et al. Discovering frequent closed itemsets for association rules[M]//Database Theory—ICDT'99. Springer Berlin Heidelberg, 1999: 398-416.

[24] Pei J, Han J, Mao R. CLOSET: An Efficient Algorithm for Mining Frequent Closed Itemsets[C]//ACM SIGMOD workshop on research issues in data mining and knowledge discovery. 2000, 4(2): 21-30.

[25] Wang J, Han J, Pei J. Closet+: Searching for the best strategies for mining frequent closed itemsets[C]//Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining. ACM, 2003: 236-245.

[26] Lloyd S. Least squares quantization in PCM[J]. Information Theory, IEEE Transactions on, 1982, 28(2): 129-137.

[27] Dempster A P, Laird N M, Rubin D B. Maximum likelihood from incomplete data via the EM algorithm[J]. Journal of the royal statistical society. Series B (methodological), 1977: 1-38.

[28] Ng A Y, Jordan M I, Weiss Y. On spectral clustering: Analysis and an algorithm[J]. Advances in neural information processing systems, 2002, 2: 849-856.

[29] Shi J, Malik J. Normalized cuts and image segmentation[J]. Pattern Analysis and Machine Intelligence, IEEE Transactions on, 2000, 22(8): 888-905.

[30] Ning H, Xu W, Chi Y, et al. Incremental Spectral Clustering With Application to Monitoring of Evolving Blog Communities[C]//SDM. 2007: 261-272.

[31] Alzate C, Suykens J A K. Multiway spectral clustering with out-of-sample extensions through weighted kernel PCA[J]. Pattern Analysis and Machine Intelligence, IEEE Transactions on, 2010, 32(2): 335-347.

[32] Jain A K, Murty M N, Flynn P J. Data clustering: a review[J]. ACM computing surveys (CSUR), 1999, 31(3): 264-323.

[33] Zahn C T. Graph-theoretical methods for detecting and describing gestalt clusters[J]. Computers, IEEE Transactions on, 1971, 100(1): 68-86.

[34] [Balke W T, Güntzer U. Multi-objective query processing for database systems[C]//Proceedings of the Thirtieth international conference on Very large data bases-Volume 30. VLDB Endowment, 2004: 936-947.

[35] Kung H T, Luccio F, Preparata F P. On finding the maxima of a set of vectors[J]. Journal of the ACM (JACM), 1975, 22(4): 469-476.

[36] 王敬贵,杜云艳,苏奋振,周成虎. 基于地理本体的空间数据集成方法及其实现[J]. 地理研究,2009,03:696-704

[37] Liu X, Yang X. A generalization based approach for anonymizing weighted social network graphs[M]//Web-Age Information Management. Springer Berlin Heidelberg, 2011: 118-130.

[38] Hao Q, Cai R, Wang X J, et al. Generating location overviews with images and tags by mining user-generated travelogues[C]//Proceedings of the 17th ACM international conference on Multimedia. ACM, 2009: 801-804.

[39] Zheng Y, Zhang L, Xie X, et al. Mining interesting locations and travel sequences from GPS trajectories[C]//Proceedings of the 18th international conference on World wide web. ACM, 2009: 791-800.

[40] Kisilevich S, Keim D, Rokach L. A novel approach to mining travel sequences using collections of geotagged photos[M]. Springer Berlin Heidelberg, 2010.

[41] [41] Lee C S, Chang Y C, Wang M H. Ontological recommendation multi-agent for Tainan City travel[J]. Expert Systems with Applications, 2009, 36(3): 6740-6753.

[42] Soo V W, Liang S H. Recommending a trip plan by negotiation with a software travel agent[M]//Cooperative Information Agents V. Springer Berlin Heidelberg, 2001: 32-37.

[43] Yu C C, Chang H P. Personalized location-based recommendation services for tour planning in mobile tourism applications[M]. Springer Berlin Heidelberg, 2009.

[44] Lu E H C, Lin C Y, Tseng V S. Trip-mine: An efficient trip planning approach with

travel time constraints[C]//Mobile Data Management (MDM), 2011 12th IEEE International Conference on. IEEE, 2011, 1: 152-161.

[45] 鲍金玲, 杨晓春, 王斌, 等. 一种支持约束关系的高效的行程规划算法[J]. 小型微型计算机系统, 2013, 34(12): 2702-2707.

[46] Cao X, Chen L, Cong G, et al. Keyword-aware optimal route search[J]. Proceedings of the VLDB Endowment, 2012, 5(11): 1136-1147.

[47] Applegate D L, Bixby R E, Chvatal V, et al. The traveling salesman problem: a computational study[M]. Princeton University Press, 2011.

致谢

逝者如斯夫，不舍昼夜，转眼间两年半的研究生已经接近尾声，我的毕业论文也圆满完成。回首研究生这几年，我感觉自己在研究方面学习到了很多，同样在生活上也领悟了很多。

首先，感谢我的硕士研究生导师陈刚教授。在研究生学习生活中，陈老师给予了我莫大的帮助和鼓励。从最开始为研究方向而迷茫，到最后在逐步完成实验和论文的过程中，陈刚老师都给予了莫大的鼓励和帮助。研究生学习的三年时间中，我从陈老师身上不仅仅学到了许多数据挖掘推荐系统领域的基础知识，而且陈刚老师总是对于数据挖掘领域当前最新研究热点及研究方向有准确的把握，使得我们能够在科研工作中快速的取得进展。另外，我还要感谢陈珂老师，在我做研究遇到困难时，陈珂老师总能够给与我恰当的指导。我还要感谢寿黎旦老师，寿黎旦老师对数据库领域的研究特别熟悉，寿老师经常教导做学术的心态和方法。还有胡天磊老师，胡天磊老师经常在生活上给与我帮助，并讨论计算机的一些新技术。最后我还要感谢伍赛老师，他交给我的论文写作技巧让我受益匪浅。

接下来，我要感谢我的父母。无论遇到什么样的困难，父母总是尽量支持我。在生活上父母也给无微不至的照顾我。感谢你们 24 年来的养育之恩，感谢你们作为我人生第一老师的所有付出。

再次，感谢我的所有同学以及实验室的兄弟姐妹，感谢刘博文、姚雨程、祁雅萍、罗妙辉、林秋霞、王俊俏、李梦雯、王振杰、何平、邝昌浪、周宇、赵王军、吴逸、叶茂伟、李环、唐钦正是有了你们的陪伴才使得枯燥的校园生活中平添了几分乐趣。

最后，感谢在百忙之中抽出时间阅读并给予指正的所有论文评阅老师。

署名　胡乔楠 2015 年 5 月 4 日星期一