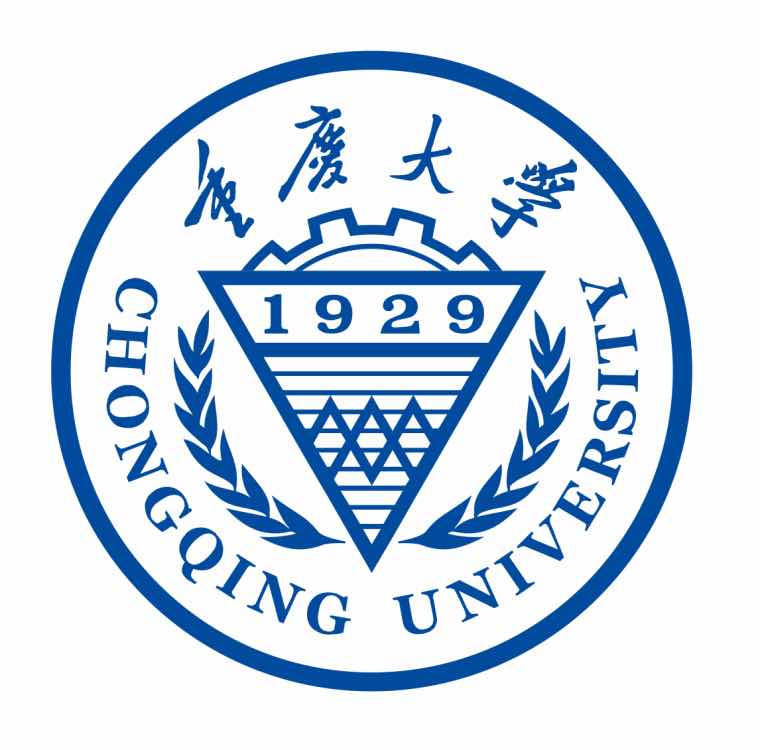
重庆大学本科学生毕业设计（论文）

# 基于启发算法的多旅行商问题优化算法的研究与实现



学 生：程小桂

学 号：20154330

指导教师：吴全旺

专 业：计算机科学与技术

重庆大学计算机学院

2019年6月

**Graduation Design(Thesis) of Chongqing University**

**MTSP based on heuristic algorithm study and implementation**

# IMG_1332

# **Undergraduate: Cheng Xiaogui**

# **Supervisor: Wu Quanwang**

# **Major: Computer Science and Technology**

### College of Computer Science

### Chongqing University

### June 2019

摘 要

本文设计了一种基于限制路径城市节点数和交叉避免的蚁群算法，解决了最小化最长路径的多旅行商问题。

本文首先探究了普通蚁群算法的设计过程，然后根据最小化最长路径的优化目标对蚁群算法的设计过程进行特殊设计，并且使用交叉避免算法对生成的路径进行优化。之后我们根据基准数据随机生成了20个测试问题，把设计的蚁群算法在这20个测试问题上运行，进行参数调优。最后我们将优化参数后的蚁群算法在这20个问题上重新运行，并记录优化目标结果和运行时间，将这些结果和Lingo 17精确算法求解在这20个问题上运行的结果进行对比，判断算法的优劣。

最后的测试结果显示我们的算法可以取得非常高的准确度并且运行时间非常小，说明我们的算法可以有效地求解最小化最长路径的多旅行商问题。

关键词：蚁群算法，有限路径城市节点，交叉避免，精确算法, MinMax

**ABSTRACT**

This paper discusses a new ant colony algorithm based on limited number of route cities and cross avoidance to handle MinMax case of MTSP.

At the very beginning, common procedure of design ant colony algorithm is discussed, then in order to handle specific features of our objective function, some strategies is imposed to the algorithm. What’s more, we go on optimizing the solutions by cross avoidance algorithm. In order to test the performance of the algorithm, 20 problems are generated by benchmark data eil51. Parameter tuning is firstly done on these problems. Then algorithm runs on those problems, using parameters from parameter tuning, and the objective result and time is registered. At last, these results are compared with those generated by using Lingo 17 exact algorithms.

From the comparison, a thrilling result is presented, which ensures the accuracy and efficiency of our algorithm.

**Key words: ACO**, **limited route cities**, **cross avoidance, exact algorithm, MinMax**

目录

摘要,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,, ,,,,,,,,,,,,,,Ⅰ

ABSTRACT,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,Ⅱ

[1 绪论](#_Toc16857_WPSOffice_Level1) [1](#_Toc16857_WPSOffice_Level1)

[1.1 引言](#_Toc1542_WPSOffice_Level2) [1](#_Toc1542_WPSOffice_Level2)

[1.2 课题研究意义](#_Toc1774_WPSOffice_Level2) [1](#_Toc1774_WPSOffice_Level2)

[1.3 国内外研究现状](#_Toc26770_WPSOffice_Level2) [3](#_Toc26770_WPSOffice_Level2)

[1.4 本章小结](#_Toc7040_WPSOffice_Level2) [6](#_Toc7040_WPSOffice_Level2)

[2 重点研究内容及技术实现途径](#_Toc1542_WPSOffice_Level1) [8](#_Toc1542_WPSOffice_Level1)

[2.1 重点研究内容](#_Toc11351_WPSOffice_Level2) [8](#_Toc11351_WPSOffice_Level2)

[2.2 技术实现途径](#_Toc10593_WPSOffice_Level2) [8](#_Toc10593_WPSOffice_Level2)

[2.2.1 开发环境配置](#_Toc10593_WPSOffice_Level3) [8](#_Toc10593_WPSOffice_Level3)

[2.2.2 蚁群算法研究](#_Toc24133_WPSOffice_Level3) [8](#_Toc24133_WPSOffice_Level3)

[2.2.3 多旅行商问题解决方案](#_Toc19413_WPSOffice_Level3) [12](#_Toc19413_WPSOffice_Level3)

[3 问题阐述](#_Toc1774_WPSOffice_Level1) [14](#_Toc1774_WPSOffice_Level1)

[3.1 问题定义和相关符号](#_Toc24133_WPSOffice_Level2) [14](#_Toc24133_WPSOffice_Level2)

[3.2 优化模型公式](#_Toc19327_WPSOffice_Level2) [14](#_Toc19327_WPSOffice_Level2)

[4. 算法描述](#_Toc26770_WPSOffice_Level1) [16](#_Toc26770_WPSOffice_Level1)

[4.1 基于有限城市访问数量的蚁群算法](#_Toc28752_WPSOffice_Level2) [16](#_Toc28752_WPSOffice_Level2)

[4.1.1 转移概率](#_Toc27914_WPSOffice_Level3) [16](#_Toc27914_WPSOffice_Level3)

[4.1.2 信息素更新](#_Toc30987_WPSOffice_Level3) [17](#_Toc30987_WPSOffice_Level3)

[4.1.3 算法流程](#_Toc9566_WPSOffice_Level3) [17](#_Toc9566_WPSOffice_Level3)

[4.2 使用交叉避免优化蚁群算法](#_Toc19413_WPSOffice_Level2) [17](#_Toc19413_WPSOffice_Level2)

[4.2.1 交叉检测](#_Toc4310_WPSOffice_Level3) [18](#_Toc4310_WPSOffice_Level3)

[4.2.2 交叉消除](#_Toc15504_WPSOffice_Level3) [19](#_Toc15504_WPSOffice_Level3)

[4.2.3 交叉避免算法描述](#_Toc29891_WPSOffice_Level3) [19](#_Toc29891_WPSOffice_Level3)

[5. 实验结果](#_Toc7040_WPSOffice_Level1) [23](#_Toc7040_WPSOffice_Level1)

[5.1 产生问题](#_Toc27914_WPSOffice_Level2) [23](#_Toc27914_WPSOffice_Level2)

[5.2 Lingo 17](#_Toc30987_WPSOffice_Level2) [23](#_Toc30987_WPSOffice_Level2)

[5.3 参数调优](#_Toc9566_WPSOffice_Level2) [23](#_Toc9566_WPSOffice_Level2)

[5.3.1 旅行商访问城市数量的上限和下限](#_Toc20740_WPSOffice_Level3) [24](#_Toc20740_WPSOffice_Level3)

[5.3.2 蚂蚁的数量](#_Toc16725_WPSOffice_Level3) [25](#_Toc16725_WPSOffice_Level3)

[5.3.3 初始信息素浓度](#_Toc26534_WPSOffice_Level3) [27](#_Toc26534_WPSOffice_Level3)

[5.3.4 挥发率](#_Toc3145_WPSOffice_Level3) [28](#_Toc3145_WPSOffice_Level3)

[5.3.5 停止条件](#_Toc9547_WPSOffice_Level3) [30](#_Toc9547_WPSOffice_Level3)

[5.4 和Lingo 17运行结果比较](#_Toc4564_WPSOffice_Level2) [33](#_Toc4564_WPSOffice_Level2)

[5.5 结果讨论](#_Toc4310_WPSOffice_Level2) [34](#_Toc4310_WPSOffice_Level2)

[6. 结论](#_Toc11351_WPSOffice_Level1) [35](#_Toc11351_WPSOffice_Level1)

[致谢](#_Toc10593_WPSOffice_Level1) [36](#_Toc10593_WPSOffice_Level1)

[参考文献](#_Toc24133_WPSOffice_Level1) [37](#_Toc24133_WPSOffice_Level1)

1 绪论

1.1 引言

旅行商问题是一个经典的NP难问题，它的基本基本定义是：给定一组城市和一个旅行商，让找出一种方案满足旅行商从一个城市出发访问完所有城市并且每个城市有且仅被访问一次的要求下，使得该路径的长度最小。多旅行商问题作为旅行商问题的一种扩展，它扩充了旅行商的数量，让多个旅行商从一个节点出发走出多条路径使得每个城市节点有且仅被访问一次。根据目标函数的不同，多旅行商问题的主要类别是最小化所有路径和以及最小化最长路径长两种。本文讨论的问题就是基于最小化最长路径目标的多旅行商问题。

蚁群算法最先被 Colorni, Dorgio 和Maniezzo 提出。作为一种元启发式算法，蚁群算法可以非常高效地解决旅行商问题。本文也将实现一种限制路径长的蚁群算法来解决多旅行商问题。

蚁群算法主要受到自然界蚂蚁的行为的启发而设计的。在自然界中，蚂蚁在行径过程中会在行进的路径上留下信息素，由此之后的蚂蚁根据之前的蚂蚁留下的高信息素浓度的路径行进。这样，越来越多的蚂蚁选择最短的路径，从而造成整个蚁群选择找到一条到达食物的最短路径。

蚁群算法虽然在解决旅行商问题方面非常高效，但是不同的问题需要设计不同的方案，因为蚁群算法有许多参数需要针对特定的问题进行调节，而且不同的问题的限制条件需要有针对性的设计算法。

1.2 课题研究意义

和旅行商问题比起来，多旅行商问题是它的扩展并且拥有更广泛的适用性，因为多旅行商问题可以处理多于一个旅行商的问题[1]。多旅行商问题的主要应用包括以下问题并由表1所示：

1）印刷调度：Gorenstein[2]提出了有多种版面的期刊的打印的调度问题。打印机有五对滚筒，纸张两面同时被打印，版面有4页、6页、8页三种类型，调度问题是哪种版面在哪个滚筒上印刷和印刷时间多长。在旅行商问题看来，盘面转换需要的时间就是旅行商问题中路径上的权重。同样的情景是产品调度问题，对于预先印制的报纸插入广告，广告商要对不同的地区投放不同的广告，这就产生了调度问题[3]。这个问题中不同的地区代表旅行商问题的城市，生产线代表了旅行商

2）人员调度：Svestka和Huckfeldt[4]提出了存款收集的路径规划问题。分行接收的人们的存款需要收集到总行，那么收集存款的职员的行进路线的规划就构成了优化目标。同样，维护工人查看设备的路径问题，邮车经过不同的邮箱的路径问题都是这一类问题

3）校车路径规划：Angel[5]研究了校车路径规划问题，相较于一般多旅行商问题添加了一些限制条件。该调度的优化目标是找到一种装载策略以使路径数量和路径总长最小，并且保证不超载和对路径访问的时间限制。

4）访问调度：Gilbert和Hofstra[6]提出了多旅行商问题在旅游业中旅行社工作人员和旅游景点工作人员会面的应用。在这个应用中，旅行社工作人员作为问题中的旅行商而旅行景区则代表城市。

5）任务计划：任务计划一般适用于自动化机器人的应用场景，比如建造、军事侦察、仓库自动化、邮局自动化和行星探索。任务计划一般情况是计划制订者有N个机器人和M个目的地，有一个让机器人返回的基地，目标就是让这些机器人找出最好的路径使得这些机器人在规定的时间内访问完所有的目标。Brummit和Stentz[7]研究了多旅行商问题在任务计划中的应用并探究了非结构化环境中的应用。在协同式机器人领域，Yu等提出了将自主机器人的规划转化成一种多旅行商问题的变体。同样Ryan等研究了自主无人飞行器的路径规划问题，并将它们转化成有时间窗口的多旅行商问题。

6）热轧调度：在钢铁冶金领域，热轧调度是为了在满足轧制规范条件，用户订单紧急程度等约束下，生成多个轧制序列分配各不同的轧机，使得整个轧制过程的花费最小化。在这个过程中，每个轧机代表一个旅行商，每个订单代表一个城市，城市间的距离是两个不同订单之间在轧机上转换所需要的花费。由此，热轧调度就转换成了非堆成的多旅行商问题。

7）全球导航定位系统检测网络的设计：全球导航定位系统是由许多卫星组成的空间系统，它在灾难预警和管理、环境和农业监测、地图导航等应用中起到关键的作用。全球导航定位系统检测网络的主要用处是为了定位地球上的一些不知道的点的地理位置。这些不知道的点都安装了卫星接收装置，并且有一组相应的观察进程来定位它们的位置。当有多个卫星接收器或者多个工作时间的时候，如何给不同的卫星接收器调度观察进程的顺序就是一个多旅行商为问题。

表1.1 多旅行商的应用

|  |  |  |
| --- | --- | --- |
| 序号 | 应用场景 | 应用实例 |
| 1 | 印刷调度 | 印刷调度[2]  预制广告调度[3] |
| 2 | 人员调度 | 存款收集[4] |
| 工人维护设备规划 |
| 邮车发送邮件路径规划 |
| 3 | 校车路径规划 | 校车路径规划[5] |
| 4 | 访问调度 | 旅游业洽谈[6] |
| 5 | 任务计划 | 机器人任务计划[7] |
| 协同机器人规划 |
| 自主无人机路径规划 |
| 6 | 热轧调度 | 钢铁订单调度 |
| 7 | 全球导航系统监测网络 | 监测进程的调度 |
| 合计 | 7 | 12 |

1.3 国内外研究现状

多旅行商问题的研究主要有三种方法，分别是精确计算法、启发式算法或元启发式算法、以及将多旅行商问题转化成单旅行商问题再用单旅行商问题求解[8]。精确计算法因为复杂度非常高，非常不适合大规模数据集；启发式算法主要有Lin-kernigham算法、自组织神经网络、遗传算法和蚁群算法。三种方法的发展如下并如表2所示：

1）精确计算法

Laporte和Nobert[8]最早提出了不将多旅行商问题转化为单旅行商问题的直接算法和反转算法。启发他们做出这样的算法的原因是将多旅行商问题转化成但旅行商问题会引入许多局部最优解，从而给求解问题带来更大的困难。而他们提出的直接算法和反转算法主要通过取消部分约束条件来完成。对于直接算法来说，它取消了算法中对于子回环消除约束，在算法产生一个整数解之后，检测该整数解是否存在子回环，如果存在就使用约束条件对这个解消除子回环。反转算法和直接算法的不同之处在于它在产生整数解之前就执行了子回环检测和消除的约束。实验结果表明，反转算法的性能比直接算法的性能更好，它在Cyber173的计算机上求解出有100多个节点的问题。该算法的一个有趣的性质是，它的执行时间随着旅行商的数量的增加变小，而将多旅行商问题转化成单旅行商问题的转化算法的执行时间随着旅行商数量的增加而边长。

Ali和Kennington[9]提出一种分支限界法，它对度数约束进行拉格朗日松弛，并且使用子梯度算法求解拉格朗日二重问题。最后该算法求解了一个规模达到100的非对称多旅行商问题和一个问题规模达到59的对称多旅行商问题。

Gavish和 Srikanth [10]最早使用分支限界法来解决大规模的对称多旅行商问题。他们给分支界定法引入了一个下界，这个下界通过解决取消了度数约束的拉格朗日问题获得。这个拉格朗日问题使用有度数限制的横跨所有节点的最小生成树进行求解。拉格朗日乘子通过子梯度优化过程进行更新，并且快速敏感性分析技术也被用来减少问题的规模。作者也进行了额外的计算分析并且解决了达到500个节点的非欧几里德问题，旅行商的数目取值从2到10以2递增；解决了节点数目达到100，旅行商数目达到10个的欧几里德问题。试验结果表明随着问题规模的增大，拉格朗日松弛算法得到的差异在变小，而当规模大于400的时候，差异几乎为0. Gavish他们同时将多旅行商问题转化为单旅行商问题，但是当旅行商人数大于4时转化所得到的多旅行商问题远比同等规模的初始问题要难解决得多。

Gromicho等[11]提出了另外一种解决多旅行商问题的精确算法。这个算法解决了固定旅行商数量的非对称多旅行商问题。该算法基于取消了子回环消除约束的QA约束条件。因为QA问题可以在多项式时间里面得到解决。通过r叉树或反r叉树加强对带有QA约束条件分支限界法的下界动态调控，实验表明，使用这种方法对于使用QA松弛无法产生好的边界的情况中对于提升下边界的影响非常巨大。同时，这种分支界定法跟只有QA松弛条件的分支界定法比起来更加优秀，从节点数量来看，少了百分之10或者10倍。特别是对于对称的多旅行商问题，算法产生了很大的提升。利用搭载了25MHz的80386的CPU的IBM PS/70的电脑，使用这个算法解决了120个城市节点的问题并且旅行商的人数从2到12，每次增加一个。

2）启发式或者元启发式算法

Russell提出了最早的一种解决多旅行商问题的启发式算法。尽管这个算法基于在扩展图上将多旅行商转换为单旅行商问题，这个算法依旧是Lin和Kernighan开发的针对单旅行商问题的一种启发式算法的扩展。Potvin等也起出了一种基于交换过程的启发式算法。

Fogel提出了一种使用进化程序的并行方法解决多旅行商问题。这个方法考虑了两个旅行商并且优化目标是最小化这两个旅行商路径的差值。规模为25和50个城市的两个问题用来测试这个算法并且获得了近乎最优解的结果。

人工神经网络是解决组合优化的有效方法，当前也有部分技术成功应用到单旅行商问题，但是应用到多旅行商问题的研究还比较少，多数是基于对单旅行商的自组织网络进行扩展。其中，Wacholder等[12]扩展了Hopfield和Tank的人工神经网络模型用来求解多旅行商问题，但是模型十分复杂也很难找到可行解。Hsu等[13]提出一种基于自组织特征图的神经网络方法，并且将多旅行商问题转化成m个并行的单旅行商问题。

最早使用遗传算法解决多旅行商问题的是Zhang[14],最近的研究是Tang[15]所研究的使用遗传算法解决热轧问题，该方法主要将问题抽象成多旅行商问题，将它转化成单旅行商问题并且应用改进的遗传算法计算一个解。在国内比较成功的用例还包括哈尔滨工程大学于2002年使用遗传算法求解多旅行商模型并解决了多个机器人的路径规划问题，使得4个机器人在32个矿井的作业更加顺畅。近年来使用遗传算法求解多旅行商的研究更加深入，但大都集中在染色体的编码的研究上，当前已有的方案有单染色体、双染色体、多染色体、两部分染色体和与具体问题无关的编码方式。实验表明不管何种编码方式都会产生单旅行商退化问题，而且因为算法计算效率严重依赖于编码方案的选择。

最近，Song使用扩展的模拟退火算法解决了每个旅行商花费固定的旅行商 问题[15]，这个方法在400个城市，3个旅行商的条件下进行测试，在IBM PC-586条件下获得51分钟的成绩。

利用蚁群算法解决多旅行商问题的研究成果均来自于我国。最早是由东北大学信息科学与工程学院于2006年提出采用ACO解决多旅行商问题，前提条件是每个旅行商所经过的城市数量被限定，通过增加虚拟城市将多旅行商问题转化为单旅行商问题，然后利用改进的ACO算法进行求解，实验证明此方法是有效的。虽然不能求出所有问题的最优解，但是能够计算得到比较好的优化解，尤其是可以求出较大规模问题的优化解，但是该方法与其它方法相比需要更长的求解时间[16]。

3）将多旅行商问题转换成单旅行商问题

将多旅行商问题转化为单旅行商问题的基本策略最早是由Gorenstein提出[17]，其主要思想是对于一个有m个旅行商的多旅行商问题，增加m-1个虚拟城市，用来对不同的旅行商访问的城市进行间隔，并且将这些虚拟城市之间的直达距离设为无穷大，以阻止旅行商访问的城市序列中出现不合理的排列。Svestka等[18] 提出了另一种转化方法，将最初的距离矩阵增加m-1行及m-1列，每一个新行和新列都是原来矩阵的第一行和第一列的复制。这种算法最初用来解决分配问题， 城市规模可以达到60个。Bellmore等[19]采用将一个有n个城市、m个旅行商的非对称多旅行商问题转化为标准的有n+m-1个城市的标准非对称单旅行商问题。Hong等[20]也采用了类似的方法，通过把一个有n个城市、m个旅行商的对称多旅行商问题转化为n+m+4个城市的标准对称单旅行商问题。Rao[21]扩展了Bellmore和Hong的方法，将一个非对称旅行商的问题转化为n+m-1个城市的标准对称旅行商问题。

除了从一个中心城市出发的多旅行商问题之外，还有从若干不同中心城市出发的多旅行商问题，对于这类问题也有一些研究者采用将其转化为单旅行商问题的方法。Laporte等通过将此类问题转化为分配问题，在此基础上使用分支限界方法求解问题规模为80、旅行商人数为3的多旅行商问题。但是这是一个不完整的转化，因为转化后的问题有一个非分配约束。对于这种情况，Yang通过复制中心城市并引入虚拟中心城市，将一个多中心城市的多旅行商问题转化为对称单旅行商问题。为此，Kara等为多旅行商问题提出一个多项式线性规划模型，并且说明了解决转化后的问题远比解决原始问题要困难得多。

虽然将多旅行商问题转化为单旅行商问题来解决是一种简单可行的方法，但是随着问题规模的扩大，转化之后的单旅行商问题将严重退化，使得问题的求 解难度更大。

表1.2 多旅行商的解决方法

|  |  |  |
| --- | --- | --- |
| 序号 | 应用场景 | 应用实例 |
| **1** | 精确计算法 | 整数线性规划  割平面法  分支界定法  拉格朗日松弛+分支界定法 |
| **2** | 启发式算法 | 简单启发式算法  进化算法  模拟退火  禁忌搜索  遗传算法  神经网络 |
| **3** | 转换方法 | 非对称多旅行商问题转换成单旅行商问题  对称多旅行商问题转化成单旅行商问题  多起点多旅行商问题转换成单旅行商问题 |

1.4 本章小结

作为对于单旅行商问题的一种扩展，多旅行商问题比单旅行商问题更具有实用性，在现实生活中拥有着广泛的应用，比如印刷调度、人员调度、校车路径规划、访问调度、任务计划、热轧调度、全球导航系统监测网络等问题。多旅行商问题主要有三种求解方案，分别是精确算法、启发式算法和将多旅行商问题转化成单旅行商问题。作为一个NP完全问题，多旅行商问题无法使用精确算法进行有效地求解，这也是当前更多应用启发式算法求解多旅行商问题的根本原因所在。

在本文中，我们计划使用蚁群算法来求解多旅行商问题。蚁群算法在求解旅行商问题的实验结果中被证明是非常有效的启发式算法，我们设计的蚁群算法引入了对每个旅行商访问路径的城市节点数目的上限和下限的限制条件，并且设计了交叉检验和交叉避免算法来进行结果的优化

2 重点研究内容及技术实现途径

2.1 重点研究内容

本课题主要研究内容是基于改进蚁群算法的多旅行商问题优化算法的研究与实现，解决最小化最长路径的多旅行商问题。主要研究内容包括：

1）阅读蚁群算法等元启发算法相关的论文和书籍，系统地了解相关知识和原理，了解蚁群算法的特点并实现蚁群算法。

2）阅读并掌握旅行商问题的基本特征和常用求解算法，讨论多旅行商问题的特征和解决方法，考虑蚁群算法的解决方法。

3）编程实现求解旅行商问题的蚁群算法，并与现有的其他常用求解算法进行对比，设计对比标准，由此方便讨论算法的性能。

4）通过查阅相关文献，研究如何针对多旅行商问题的特性，对蚁群算法进行改进，从而实现更佳的优化性能。

5）学习算法分析与设计、数据结构、Java程序开发、面向对象程序设计、软件工程等方面的课程，编程实现求解多旅行商问题的改进蚁群算法，分析和验证所设计的改进算法的性能优劣。

重点和难点在于掌握蚁群算法的基本原理和旅行商问题的优化算法。完成基于改进蚁群算法的多旅行商问题优化算法是本课题的重点，难点是基于多旅行商问题的特性，通过改进蚁群算法获取更佳的优化性能。

2.2 技术实现途径

2.2.1 开发环境配置

开发环境选择的操作系统选择的是Linux，选择的编译器是Pycharm，开发语言选择Python，使用numpy科学计算包以及matplotlib图形可视化工具进行算法的开发和可视化工作。

2.2.2 蚁群算法研究[22]

1）蚁群算法原理

如图1所示，蚂蚁从蚁穴E点可以经过两条路径EDFBA或者EDCBA到达食物点A。开始的时候，蚂蚁在E点可以往左边或者往右边走，概率上来说各占一半。我们假设蚂蚁的行进速度一样，对于行进到D点的蚂蚁，它们有两个选择，然而对于选择经过F的路径的蚂蚁，他们到达食物点A的时间明显长于经过C点的蚂蚁，由此路径路径DCB上积累信息素的速度会比路径DFB快，从而导致其他蚂蚁在选择路径的时候会更倾向与选择信息素浓度更高的路径DCB，由此形成了最短路径EDCBA。

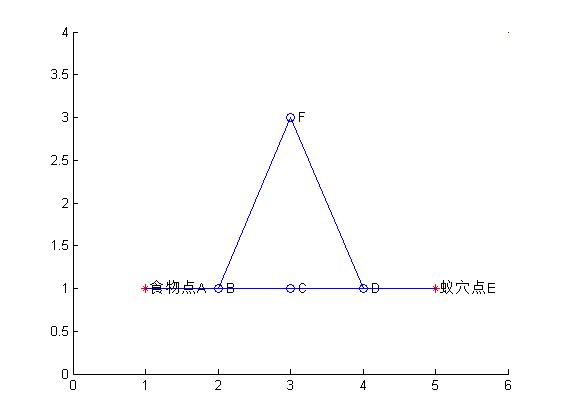


图2.1 蚂蚁寻路图示

2）蚁群算法解决单旅行商问题问题算法

根据蚂蚁寻路问题的启示，我们产生了下面表3.1的蚁群算法的计算过程来解决TSP问题

表2.1 蚁群算法基本步骤

|  |  |
| --- | --- |
| 步骤 | 具体描述 |
| 1 | 对相关参数进行初始化，包括蚁群规模、信息素因子、启发函数因子、信息素挥发因子、信息素常数、最大迭代次数等，以及将数据读入程序，并对数据进行基本的处理，比如将城市的坐标位置转化为城市间的矩阵 |
| 2 | 将蚂蚁放于不同的出发点，计算其下一个访问城市，直至所有蚂蚁访问完所有城市 |
| 3 | 计算各个蚂蚁经过的路径长度L，记录当前迭代次数中的最优解，同时对各个城市连接路径上的信息素浓度进行更新 |
| 4 | 判断是否达到最大的迭代次数，否则返回步骤2 |
| 5 | 输出程序结果并根据需要输出程序寻优过程的相关指标 |

该算法的基本流程图如图2所示

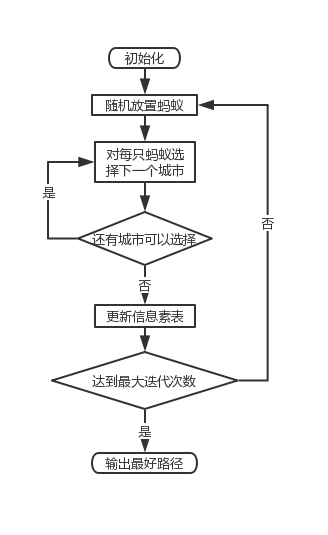


图2.2 蚁群算法基本流程图

1. 蚁群算法优化方案-交叉避免

在产生解的过程中，可能会出现如图3的路径交叉的情况。研究蚁群算法的特征我们可以发现如果可以优化部分路径，那么全局路径的长度也可以被优化。由此如果我们可以优化交叉部分的路径，我们就可以优化整个路经长。从图3四边形性质我们可以看到，对角线之和总是大于对边之和，因为对于对角线和对边总有如下不等式成立

 (3.1)

由此可以利用四边形对角线和大于对边和的性质对路径进行优化，把对边 舍去并添加两条对边使得构成回路，进而让将路径长减少。

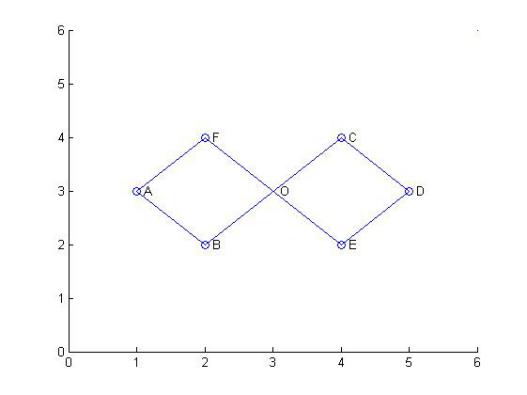


图2.3 路径交叉示例

具体的算法步骤如下表4所示

表2.2 交叉避免算法步骤

|  |  |
| --- | --- |
| 步骤 | 具体描述 |
| **1** | 初始化方程系数，对于给定的两条线段，计算它们的直线的方程的系数 |
| **2** | 判断两条线段是否平行或重叠，如果平行或者重叠就退出 |
| **3** | 计算两个线段对应的直线的交点 |
| **4** | 判断交点是否在两条线段上，如果在就进行路径调整，否则退出 |

算法流程图如图2.4所示。

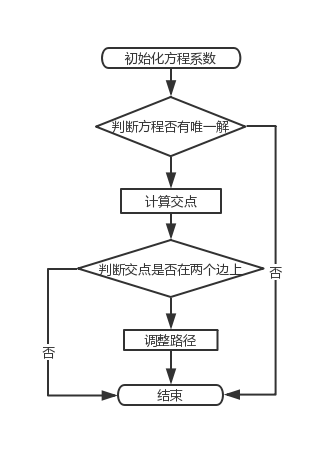


图2.4 交叉避免算法流程

2.2.3 多旅行商问题解决方案[22]

1）多旅行商问题类型[23]

一般情况下，多旅行商问题分为四种类型：

第一种：m个旅行商从同一个城市出发访问其中一定数量的城市，即只有一个中心城市，使得每个城市必须被某一个旅行商访问而且只能访问一次，最后回到出发城市。

第二种：m个旅行商从m个不同城市出发访问其中一定数量的城市，使得每个城市必须被某一个旅行商访问而且只能访问一次，最后回到各自的出发城市。

第三种：m个旅行商从同一个城市出发访问其中一定数量的城市，使得每个城市须被某一个旅行商访问而且只能访问一次，最后到达m个不同的城市。

第四种：m个旅行商从m个不同城市出发访问其中一定数量的城市，使得每个城市必须被某一个旅行商访问而且只能访问一次，最后到达同一个城市。常见的多旅行商问题为前两种，例如快递公司送货的问题就是第一种，全国连锁店的送货问题就是第二种。

按照多旅行商问题的一般性定义，第一、二种多旅行商问题就在这个定义范围之内，第三、四种加上约束条件“各个旅行商的终点不为出发城市”，为一般定义的多旅行商问题的变种。如果在一般的多旅行商问题基础上，再加上一些约束则演变出变种的多旅行商问题。如果两个城市之间的代价在两个方向上是一样的，称为对称多旅行商问题，反之称为非对称多旅行商问题。

2）蚁群算法解决多旅行商问题问题[25]

改进的蚁群算法解决多旅行商问题问题主要在于选择下一个城市的时候蚂蚁的可选城市列表的区别，具体包括下表5两个方面：

表2.3 起点城市可选条件

|  |  |
| --- | --- |
| 步骤 | 具体描述 |
| 蚂蚁可以回到起点城市条件 | 当前蚂蚁已经访问了L个城市  剩余蚂蚁不足以平均获得K个访问城市 |
| 起点城市必须在禁忌表条件 | 蚂蚁没有访问到K个城市  蚂蚁是当代最后一个并且存在未访问城市  剩余蚂蚁平均获得超过L个可访问城市 |

3）蚁群算法的参数设置问题[25]

蚁群算法拥有大量的参数，而不同的参数设置对实验结果的影响是很巨大的，为了获得良好的参数，需要使用实验的方法获得对参数的设置的指导。通过查阅资料，根据文献[26]的实验结果，有如下表6的设置参数的方法：

表2.4 蚁群算法参数设置

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 参数符号 | 参数说明 | 参考数值 |
| **1** | α | 信息素因子 | [1,4] |
| **2** | β | 启发函数因子 | [3,4.5] |
| **3** | ρ | 信息素挥发因子 | [0.2,0.5] |
| **4** | Q | 信息素常熟 | [10,1000] |
| **5** | Iter\_max | 最大迭代次数 | [100,500] |

3 问题阐述

3.1 问题定义和相关符号

对于一个关于一组城市的完全图和一个花费矩阵，其中V代表这一组城市节点，A代表任意两个城市节点的边，代表第i个和第j个城市之间的距离。花费矩阵矩阵可以是对称的也可以是非对称的。 城市节点被划分成出发节点V和客户节点，因此。最初的时候，出发城市1有m个旅行商。

对于优化目标为最小化最长路径的多旅行商问题，就是就是要给这m个旅行商的每个人找到一条路径，从起始城市出发，最后回到出发节点。每个城市节点有且仅被一个旅行商访问一次，并且优化所有路径中最长的那一条路径，让它最小化。

3.2 优化模型公式

Soheil Ghafurian 和 Nikbakhsh Javadian给出了多出发节点多旅行商问题MMTSP的优化模型公式，解决了固定终点的多出发点多旅行商问题。我们借鉴这个优化模型并根据我们问题的目标进行修改。

首先，我们定义了一个二进制变量，它说明第k个旅行商所走过的路径。



对于每个旅行商来说,  代表从起点城市出发到第i个城市走过的城市的数量，它包括了第i个城市。 在这个问题中，我们引入 和  作为每个旅行商访问城市数量的限制。其中，  是每个旅行商最少访问城市的数量，  是每个旅行商最多访问城市的数量。因此，对于每个客户节点即非起点城市 , 有不等式  成立。并且对于每个旅行商访问的最优一个客户节点 , 它的  满足下面的不等式 。

对于本文所讨论的最小化最长路径的多旅行商问题，他的优化模型如下：



其中， , ,并且  是旅行商的数目。

在这个公式中，第一个约束条件限制了有且仅有m个旅行商从出发城市起程，约束条件2限制了所有的旅行商必须回到出发城市节点1， 第三个约束条件保证所有的客户节点城市都被访问了有且仅有一次，第四个约束条件保证访问一个客户城市节点的旅行商必须从那个客户节点城市离开。第五和第六个约束条件分别约束了每个旅行商访问城市节点数目的上限和下限。因为如果一个旅行商只访问了一个客户城市节点也满足第五和第六个约束条件，但是却打破了对于访问城市节点数目下限的要求，所以第七个约束条件把访问城市节点数目为一个的情况排除了。第八个约束条件是防止产生客户节点子回环的约束条件。客户节点子回环指的是在客户节点中产生的环路，因为这个环路没有起点城市作为起点和终点，所以不满足我们对于旅行商问题的定义，因此是无效的解。如果没有子回环的约束条件，前面的约束条件不足以产生没有子回环的解。

1. 算法描述

4.1 基于有限城市访问数量的蚁群算法

最初，起点城市有antnum个人工蚂蚁。对于每个人工蚂蚁来说，它从起点城市触发，不断地从可以访问的城市集合里面根据一定的概率选择一个来访问，直到最后回到起点城市，完成了对一个子路径的构造。之后它还从起点城市出发按照上面的方法构造一个子路径，直到最后这个蚂蚁构造了m个子路径。这样就构造出了一个解。

4.1.1 转移概率

我们根据M. Dorgio, V. Maniezzo提出的关于计算转移概率的方法，但是因为我们的问题的特殊性，我们需要做一些改进。我们主要对保存当前无法被访问的节点的禁忌表  的构造方法添加了一些策略。对于当前处在第i个城市节点的人工蚂蚁，它下面访问第j个城市节点的概率可以用下面的公式计算。



其中  代表当前在i城市的蚂蚁能够访问访问的城市集合。  和  控制信息素浓度和距离倒数的相对重要程度。

对于选择下一个城市的算法，需要考虑下面的策略才能构造有效的解：

1. 人工蚂蚁必须返回起点城市并且结束当前路径的情况：
   1. 人工蚂蚁已经访问了H个城市，此时如果不回到起点城市，那么就会突破对于路径的访问城市数量的上限
   2. 剩余可以被访问的城市的数量不足以平均分配给剩下的路径，使得剩下的路径的城市数量大于L，此时如果不回到起点城市，那么就无法满足所用的路径的城市数量的下限
2. 起点城市1不能可访问城市列表的情况：
   1. 人工蚂蚁当前访问的城市的数量小于L，此时如果起点城市出现在可访问城市列表中就可能回到起点，从而造成路径的城市节点数目小于L的情况
   2. 这个蚂蚁当前在构造最后一个路径，此时如果蚂蚁回到起点城市，就会打破访问所有城市节点的限制条件
   3. 当前剩下的城市节点数目平均到每条剩下的路径得到的数目超过了H，此时如果蚂蚁回到起点城市就无法路径中城市节点数目小于上限。

4.1.2 信息素更新

算法的每一轮结束以后，需要更新信息素矩阵。我们让  代表第t轮中从第i个节点到第j个节点的边上的信息素浓度。那么该信息素浓度矩阵的更新的公式是：



其中  代表地种群中第k个蚂蚁在第t到第t+1轮中间在城市i到城市j的路径上留下的信息素浓度。  的计算公式是：



其中 是第k个蚂蚁产生的路径在目标函数下得到的数值。也就是说，这个数值是第k个蚂蚁产生的路径的最长的子路径的长度。

4.1.3 算法流程

算法会被重复执行，最多 循环。在每一轮计算里面，有  个蚂蚁按照上面所说的方法构造 个路径。在每轮计算结束以后，信息素浓度矩阵使用这些路径来更新新的信息素浓度矩阵。

算法将会在执行了 轮或者当前蚁群中的最优解没有对于当前全局最优解有提升的时候结束。

• 初始化 

• Do:

* 1. 用上面讨论的方法构造  个解
  2. 每轮的最后用上面使用的算法更新信息素浓度矩阵 

• While( 当前蚁群中的最优解提升当前全局最优解并且算法执行轮数小于)

• 返回最优解

4.2 使用交叉避免优化蚁群算法

对于旅行商问题来说，下面两个性质保证了通过交叉避免可以优化结果

性质1） 优化局部路径可以引起优化整个路径

从图1中我们可以发现，计算整条路径的长度的计算公式可以是：



从中我们可以看到，如果我们优化了部分路径 , 我们就可以优化整条路径的长度 

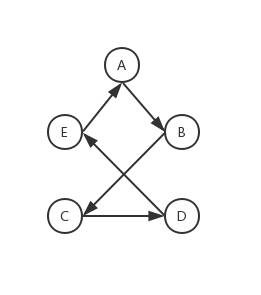


图4.1 某条可能的路径

性质2） 在四边形中，对边的和小于对角边的和

从图2我们可以看到，下面的不等式一定成立。这个结论启发我们如果我们讲有交叉的路径把它的交叉解开，让路径从走对角边变成走对边，我们就可以减少整条路径的长度。所以我们的工作就是调整这个交叉。



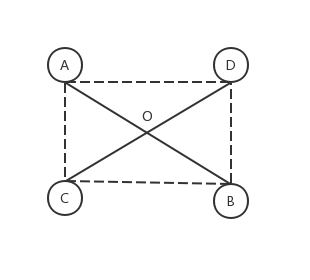


图4.2 一个四边形

4.2.1 交叉检测

如图2所示，假设路径中交叉的的四个点分别是 ,并且他们的交点是 , . 因此有下面的线性方程组：



其中,



如果这个方程组有解, 那么. 如果方程满足这个条件，那么我们就可以就算这个方程组的解。计算方程组的接的公式是：



如果这个路径在这四个点中间有交叉，那么就要满足下面三个条件中的一个



或者,



或者,



4.2.2 交叉消除

从图3中我们可以看到，最开始的时候，路径是 . 然而，经过消除交叉之后，新的路径是 . 从这两个路径的对比中，我们可以看到消除交叉的过程实际上可以简化成将有交叉的那个部分的路径的起点和终点中间的所有点进行颠倒。也就是说，对于上面的路径的消除交叉的操作是将有交叉的部分路径的起点A和终点E之间的所有的点倒转。由此我们对于交叉避免的算法就是将发生交叉的部分路径的起点城市和终点城市之间走过的城市的顺序进行颠倒。

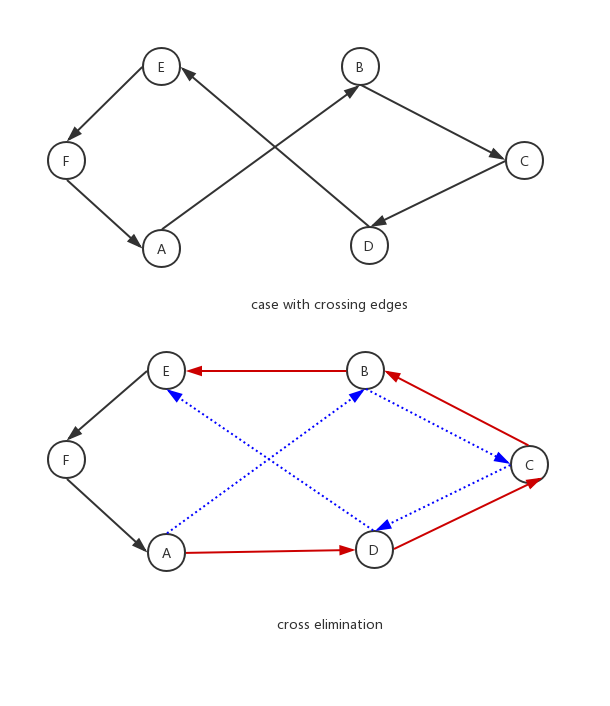


图4.3 交叉消除

4.2.3 交叉避免算法描述

• while :

• 应用交叉检测算法检测是否出现交叉

• 如果存在交叉，对于出现交叉的部分路径使用交叉消除算法来消除交叉

• 如果没有交叉，退出循环

1. 实验结果

5.1 产生问题

对于单出发点的多旅行商问题，有一些基准的数据，这些数据是 eil51, berlin52, eil76, and rat99。但是当前的算法还没有对这些数据进行精确的求解，也没有对于这些数据的最优解的计算，因为这些数据的近似解。但是近似解对于我们测试算法的计算方法不合适。因为我们使用的测试算法的计算方法是把最优解和蚁群算法产生的解的比作为评价算法的标准。从这个标准来说，它的数值在0到1之间并且数值越大说明算法的效果越好，算法产生的解越逼近于最优解。

我们用eil51的基准数据来产生一些规模比较小的，可以在有限时间里面求解的问题。更细致的来说，我们从eil51的基准数据中随机取出n个城市。其中，对于每个n，我们生成了4个问题。所以我们一共产生了20个问题。

5.2 Lingo 17

作为一款知名的解决线性规划，非线性规划和整数规划的优化软件，Lingo 17 使用精确算法可以很强大地计算出优化模型的最优解。同样，Lingo 17 也具有对于精确算法的普遍的缺陷，那就是对于NP完全问题， Lingo 17使用的精确算法不能在有效时间内求解出这种问题在大规模条件下的问题的最优解。

在我们产生问题的过程中，我们发现Lingo 17很有效的求解出有很多城市的多旅行商问题。比如我们产生了城市数量为20的一个问题，使用三个旅行商，结果我们花费了几天的时间才找到了最优解。所以我们选择城市的是让城市的数量小于20。

在整个实验章节，我们使用Lingo 17来求解问题的最优解。

5.3 参数调优

使用蚁群算法求解优化问题的一个非常困难的地方就在于参数的调整。这也就是说，对于参数的一点小小的改变就会对整个求解最优解的效果产生巨大的差异，从而影响算法的性能。一般说来，参数调优的算法整体上采用的实验方法，我们也会利用实验方法来调节参数。

从《MATLAB在数学建模中的应用》给出的对于参数的建议，我们将α和β分别设置成1和5. 表1是对n=10的四个问题在不同k的数值的使用Lingo 17 进行精确求解的时间花费的结果。从中我们可以看到总体上来说，精确算法在相同问题上用不同的k值进行求解的时间花费大体上随着k的增加而变大。由此，为了在有效时间内求解问题，我们将k值设置成2.

表5.1 用不同k值解决n=10的四个问题花费的CPU时间（单位 秒）

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| k | 问题一 | 问题二 | 问题三 | 问题四 | 平均时间 |
| 1 | 0.14 | 0.66 | 0.67 | 0.38 | 0.46 |
| 2 | 1.23 | 10.98 | 15.94 | 7.39 | 8.88 |
| 3 | 9.83 | 98.98 | 140.4 | 114 | 90.8 |
| 4 | 67.91 | 112.28 | 82.98 | 129.84 | 98.25 |

5.3.1 旅行商访问城市数量的上限和下限

对于最小化最长路径的多旅行商问题，他的主要应用就是平衡负载，让所有旅行商经过的路径长度相差不大。因此我们假设旅行商访问城市的数量也相差不大，从而让每个旅行商访问的城市的数量在一个范围里面，也就是设置一个上下限。为了找到最合适的范围和上下界，我们使用不同的上下界解决产生的问题。算法在每个问题上运行10次并且保存最好的5次的产生的解和运行时间。算法的其他参数分别进行如下的设置。算法的结果见表2，表3，表4，表5.

表5.2 n=12的问题在不同的数量间隔下解的平均比值

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 数量间隔 | 问题一 | 问题二 | 问题三 | 问题四 | 平均比值 |
| (5, 6) | 0.98 | 1.00 | 0.99 | 0.96 | 0.98 |
| (4, 7) | 0.98 | 1.00 | 1.00 | 0.96 | 0.98 |
| (3, 8) | 0.97 | 1.00 | 1.00 | 0.94 | 0.98 |

表5.3 n=14的问题在不同的数量间隔下解的平均比值

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 数量间隔 | 问题一 | 问题二 | 问题三 | 问题四 | 平均比值 |
| (6, 7) | 1.00 | 0.97 | 1.00 | 1.00 | 0.99 |
| (5, 8) | 0.98 | 0.97 | 1.00 | 1.00 | 0.99 |
| (4, 9) | 0.98 | 0.94 | 1.00 | 0.98 | 0.97 |

表5.4 n=16的问题在不同的数量间隔下解的平均比值

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 数量间隔 | 问题一 | 问题二 | 问题三 | 问题四 | 平均比值 |
| (7, 8) | 0.96 | 0.99 | 0.97 | 0.94 | 0.96 |
| (6, 9) | 1.00 | 0.98 | 0.98 | 0.96 | 0.98 |
| (5, 10) | 0.98 | 0.97 | 0.98 | 0.98 | 0.97 |

表5.5 n=18的问题在不同的数量间隔下解的平均比值

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 数量间隔 | 问题一 | 问题二 | 问题三 | 问题四 | 平均比值 |
| (8, 9) | 0.93 | 1.00 | 0.98 | 0.93 | 0.96 |
| (7, 10) | 0.94 | 0.98 | 0.97 | 0.92 | 0.95 |
| (6, 11) | 0.95 | 0.98 | 0.97 | 0.92 | 0.95 |

从上面的结果来看，我们可以看到对于每个n值，对应的间隔的设置可以从表6看到

表5.6 不同n值对应的优化的数量间隔

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| n | 12 | 14 | 16 | 18 |
| 间隔 | (4,7) | (6,7) | (6,9) | (8,9) |

5.3.2 蚂蚁的数量

在蚁群算法中，蚂蚁的数量影响算法的精确度和算法的收敛的时间。一般说来蚂蚁数量越大，算法越容易收敛到局部最优解，算法的收敛速度越快。而蚂蚁数量越小，寻找全局最优解的能力越强，相对的，算法的收敛速度越慢。因此如何平和解的精度和算法的运行时间是需要考虑的问题。为了找到合适的蚂蚁数量，我们对每个问题应用蚁群算法求解10次并且将最好的5次记录下来。算法对每个问题的数量间隔按照上面讨论的结果设置，对于其他参数，分别进行如下的设置。实验结果中表7展示了对于不同蚂蚁数量不同n的所有问题的最优解对蚁群算法产生的解的比值的平均值，表8展示了蚁群算法运行的平均时间，图1展示了平均比值和时间的变化趋势。

表5.7 不同antnum对不同n的问题的最优解和蚁群算法求出的最好解的比值平均值

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| antnum | N=12 | N=14 | N=16 | N=18 | 平均比值 |
| 1 | 0.92 | 0.94 | 0.92 | 0.88 | 0.91 |
| 2 | 0.94 | 0.97 | 0.93 | 0.91 | 0.94 |
| 3 | 0.96 | 0.98 | 0.94 | 0.94 | 0.95 |
| 4 | 0.97 | 0.99 | 0.94 | 0.93 | 0.96 |
| 5 | 0.97 | 0.99 | 0.95 | 0.94 | 0.96 |
| 6 | 0.97 | 0.98 | 0.96 | 0.95 | 0.97 |
| 7 | 0.97 | 0.99 | 0.97 | 0.96 | 0.97 |
| 8 | 0.96 | 0.99 | 0.97 | 0.95 | 0.97 |
| 9 | 0.97 | 0.99 | 0.95 | 0.95 | 0.97 |
| 10 | 0.96 | 0.99 | 0.97 | 0.94 | 0.96 |
| 11 | 0.98 | 0.99 | 0.97 | 0.96 | 0.97 |
| 12 | 0.98 | 0.99 | 0.96 | 0.96 | 0.97 |
| 13 | 0.98 | 0.99 | 0.97 | 0.95 | 0.97 |
| 14 | 0.97 | 0.99 | 0.98 | 0.96 | 0.97 |
| 15 | 0.98 | 0.99 | 0.98 | 0.96 | 0.97 |
| 16 | 0.98 | 0.99 | 0.97 | 0.96 | 0.97 |
| 17 | 0.98 | 0.99 | 0.98 | 0.96 | 0.98 |
| 18 | 0.97 | 0.99 | 0.98 | 0.95 | 0.97 |
| 19 | 0.98 | 0.99 | 0.97 | 0.96 | 0.97 |
| 20 | 0.98 | 0.99 | 0.98 | 0.96 | 0.98 |
| 21 | 0.97 | 0.99 | 0.98 | 0.96 | 0.97 |
| 22 | 0.98 | 0.99 | 0.98 | 0.97 | 0.98 |
| 23 | 0.98 | 0.99 | 0.97 | 0.95 | 0.97 |
| 24 | 0.98 | 0.99 | 0.99 | 0.97 | 0.98 |
| 25 | 0.98 | 0.99 | 0.99 | 0.95 | 0.98 |

表5.8 不同antnum对不同n的问题的平均时间（单位 秒）

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| antnum | N=12 | N=14 | N=16 | N=18 | 平均时间 |
| 1 | 0.06 | 0.06 | 0.07 | 0.08 | 0.07 |
| 2 | 0.09 | 0.11 | 0.12 | 0.14 | 0.12 |
| 3 | 0.13 | 0.16 | 0.18 | 0.21 | 0.17 |
| 4 | 0.17 | 0.21 | 0.24 | 0.27 | 0.22 |
| 5 | 0.21 | 0.26 | 0.29 | 0.34 | 0.27 |
| 6 | 0.25 | 0.32 | 0.35 | 0.40 | 0.33 |
| 7 | 0.30 | 0.39 | 0.40 | 0.47 | 0.39 |
| 8 | 0.46 | 0.42 | 0.46 | 0.53 | 0.47 |
| 9 | 0.52 | 0.46 | 0.52 | 0.60 | 0.52 |
| 10 | 0.50 | 0.53 | 0.57 | 0.69 | 0.57 |
| 11 | 0.50 | 0.57 | 0.63 | 0.75 | 0.61 |
| 12 | 0.62 | 0.61 | 0.68 | 0.84 | 0.69 |
| 13 | 0.57 | 0.62 | 0.74 | 0.92 | 0.71 |
| 14 | 0.63 | 0.66 | 0.79 | 0.95 | 0.76 |
| 15 | 0.71 | 0.71 | 0.85 | 0.99 | 0.81 |
| 16 | 0.75 | 0.76 | 0.91 | 1.05 | 0.87 |
| 17 | 0.75 | 0.84 | 0.96 | 1.15 | 0.93 |
| 18 | 0.73 | 0.90 | 1.04 | 1.48 | 1.03 |
| 19 | 0.76 | 0.92 | 1.14 | 1.59 | 1.10 |
| 20 | 0.80 | 1.01 | 1.20 | 1.62 | 1.16 |
| 21 | 0.84 | 1.16 | 1.24 | 1.72 | 1.24 |
| 22 | 0.87 | 1.12 | 1.29 | 1.78 | 1.27 |
| 23 | 0.92 | 1.11 | 1.38 | 1.86 | 1.32 |
| 24 | 1.12 | 1.13 | 1.43 | 2.06 | 1.43 |
| 25 | 1.09 | 1.18 | 1.50 | 2.23 | 1.50 |

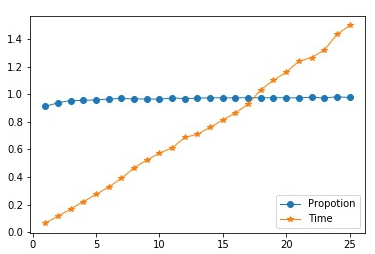


图5.1 不同antnum的平均比值和时间的变化趋势

从图1我们可以看到，算法性能的主要提升在蚂蚁数量小于10之前。然而蚂蚁数量为10所得到的对所有问题的平均运行时间几乎是蚂蚁数量为7的两倍，并且蚂蚁数量为7和蚂蚁数量为10的两种情况在算法性能上差异不大，因此我们忽略蚂蚁数量为7之后的性能提升，把蚂蚁数量antnum设置成7。

5.3.3 初始信息素浓度

一般说来，初始信息素浓度不会影响算法的性能。为了确认我们的假设，我们让蚁群算法在所有的问题上运行了10次，并且记录了其中最好的5次结果，包括每个解得到的最长回路的长度和运行时间。除了上面讨论的参数，我们把其他参数分别进行如下设置。实验结果如表9，表10和图2所示。

表5.9 不同初始trialmatrix对不同n的最优解和蚁群算法的最好解的平均比值

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| trailmatrix | N=12 | N=14 | N=16 | 平均比值 |
| 1 | 0.97 | 0.98 | 0.96 | 0.97 |
| 10 | 1.00 | 0.97 | 0.95 | 0.97 |
| 100 | 0.99 | 1.00 | 0.97 | 0.96 |
| 1000 | 0.92 | 1.00 | 0.95 | 0.97 |

表5.10 不同初始trialmatrix对不同n的平均时间（单位 秒）

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| trailmatrix | N=12 | N=14 | N=16 | N=18 | 平均比值 |
| 1 | 0.407 | 0.436 | 0.523 | 0.612 | 0.494 |
| 10 | 0.321 | 0.428 | 0.488 | 0.630 | 0.466 |
| 100 | 0.324 | 0.405 | 0.480 | 0.601 | 0.452 |
| 1000 | 0.357 | 0.418 | 0.516 | 0.599 | 0.472 |

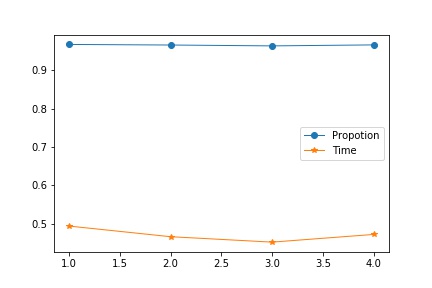


图5.2 不同初始信息素浓度的平均比值和时间变化趋势

从图2中我们看到，不同的出事信息素浓度不会影响算法的精度和效率，这也证明了我们之前的结论。所以我们将初始信息素浓度设置成1。

5.3.4 挥发率

从经验看来，挥发率越小，那么信息素浓度积累越快，算法收敛越快，算法越容易陷入局部最优解；挥发率越大，信息素浓度积累越慢，算法越容易找到全局最优解，但是相对来说，算法收敛速度变慢，算法运行时间变长。因此，如何平衡算法精度和算法运行时间，找到合适的挥发率，我们设置了下面的实验。我们让挥发率设置成从0.1到0.9，每次间隔0.1. 我们让蚁群算法在所有问题上对不同的挥发率上运行10次并且记录最好的5次结果的目标数值和运行时间。算法参数根据上面讨论的结果进行设置，对于没有讨论的参数，我们设置如下。算法的结果可以从表11，表12和图3中看到。

表5.11 不同对不同n的最优解和蚁群算法的最好解的平均比值

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | N=12 | N=14 | N=16 | N=18 | 平均比值 |
| 0.1 | 0.976 | 0.988 | 0.965 | 0.955 | 0.971 |
| 0.2 | 0.977 | 0.986 | 0.972 | 0.970 | 0.976 |
| 0.3 | 0.976 | 0.988 | 0.975 | 0.958 | 0.974 |
| 0.4 | 0.973 | 0.987 | 0.961 | 0.955 | 0.969 |
| 0.5 | 0.976 | 0.988 | 0.961 | 0.960 | 0.971 |
| 0.6 | 0.975 | 0.986 | 0.971 | 0.941 | 0.968 |
| 0.7 | 0.972 | 0.989 | 0.965 | 0.951 | 0.969 |
| 0.8 | 0.965 | 0.986 | 0.963 | 0.953 | 0.967 |
| 0.9 | 0.965 | 0.981 | 0.965 | 0.952 | 0.966 |

表5.12 不同对不同n的平均CPU时间（单位 秒）

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | N=12 | N=14 | N=16 | N=18 | average |
| 0.1 | 0.330 | 0.420 | 0.486 | 0.570 | 0.452 |
| 0.2 | 0.306 | 0.420 | 0.490 | 0.572 | 0.447 |
| 0.3 | 0.318 | 0.416 | 0.506 | 0.612 | 0.463 |
| 0.4 | 0.332 | 0.407 | 0.500 | 0.597 | 0.459 |
| 0.5 | 0.362 | 0.435 | 0.490 | 0.583 | 0.467 |
| 0.6 | 0.366 | 0.443 | 0.497 | 0.669 | 0.493 |
| 0.7 | 0.363 | 0.445 | 0.534 | 0.678 | 0.505 |
| 0.8 | 0.368 | 0.416 | 0.543 | 0.662 | 0.497 |
| 0.9 | 0.346 | 0.438 | 0.506 | 0.613 | 0.476 |

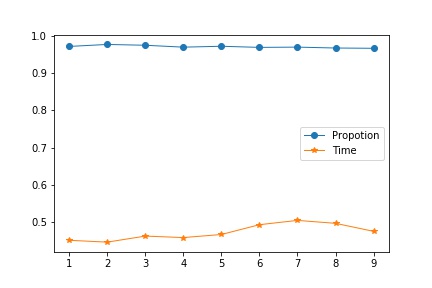


图5.3 不同挥发率的平均比值和CPU时间的变化趋势

从图3我们可以看到，随着挥发率的增加，算法耗费的时间变多。综合运行时间和算法精度的条件下，我们可以看到把挥发率设置成0.2是很合适的。一方面他的精度损失比其他的结果更小，同事运行时间也很小。

5.3.5 停止条件

蚁群算法会在运行一定的轮数之后收敛，也就是算法对于目标没有任何提升，而之后的轮数就没有意义，反而会让算法的运行时间变长，因此找到合适的最大运行轮数有一定的意义。为了找到合适的最大轮数，我们所有问题用不同的运行最大轮数运行了10次并且记录了目标结果最好的5次的目标结果和运行时间。算法的参数根据上面的讨论进行设置。算法的运行结果如表5.13，表5.14和图5.4所示。

表5.13 不同运行轮数对不同n的问题的最优解和蚁群算法最好解的平均比值

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| cyclenum | n12 | n14 | n16 | n18 | average |
| 1 | 0.931 | 0.935 | 0.891 | 0.884 | 0.910 |
| 2 | 0.950 | 0.976 | 0.931 | 0.888 | 0.936 |
| 3 | 0.938 | 0.983 | 0.946 | 0.897 | 0.941 |
| 4 | 0.962 | 0.982 | 0.924 | 0.927 | 0.949 |
| 5 | 0.955 | 0.986 | 0.948 | 0.916 | 0.951 |
| 6 | 0.965 | 0.989 | 0.947 | 0.922 | 0.956 |
| 7 | 0.962 | 0.986 | 0.946 | 0.912 | 0.952 |
| 8 | 0.967 | 0.986 | 0.950 | 0.928 | 0.958 |
| 9 | 0.967 | 0.986 | 0.951 | 0.948 | 0.963 |
| 10 | 0.968 | 0.987 | 0.943 | 0.939 | 0.959 |
| 11 | 0.968 | 0.987 | 0.952 | 0.935 | 0.960 |
| 12 | 0.971 | 0.986 | 0.951 | 0.931 | 0.960 |
| 13 | 0.968 | 0.987 | 0.955 | 0.947 | 0.964 |
| 14 | 0.971 | 0.987 | 0.946 | 0.933 | 0.959 |
| 15 | 0.967 | 0.986 | 0.956 | 0.960 | 0.967 |
| 16 | 0.975 | 0.987 | 0.955 | 0.957 | 0.968 |
| 17 | 0.981 | 0.988 | 0.961 | 0.945 | 0.969 |
| 18 | 0.970 | 0.989 | 0.960 | 0.956 | 0.968 |
| 19 | 0.985 | 0.988 | 0.967 | 0.957 | 0.974 |
| 20 | 0.975 | 0.987 | 0.960 | 0.951 | 0.968 |
| 21 | 0.975 | 0.988 | 0.965 | 0.944 | 0.968 |
| 22 | 0.973 | 0.990 | 0.967 | 0.963 | 0.973 |
| 23 | 0.981 | 0.988 | 0.966 | 0.959 | 0.973 |
| 24 | 0.982 | 0.986 | 0.967 | 0.966 | 0.975 |
| 25 | 0.984 | 0.990 | 0.964 | 0.962 | 0.975 |

表5.14 不同运行轮数对不同n的问题的平均CPU时间（单位 秒）

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| cyclenum | N=12 | N=14 | N=16 | N=18 | average |
| 1 | 0.023 | 0.031 | 0.036 | 0.043 | 0.033 |
| 2 | 0.035 | 0.052 | 0.064 | 0.067 | 0.054 |
| 3 | 0.049 | 0.069 | 0.084 | 0.096 | 0.075 |
| 4 | 0.063 | 0.095 | 0.117 | 0.124 | 0.099 |
| 5 | 0.077 | 0.118 | 0.131 | 0.153 | 0.120 |
| 6 | 0.099 | 0.133 | 0.168 | 0.182 | 0.145 |
| 7 | 0.120 | 0.151 | 0.211 | 0.197 | 0.170 |
| 8 | 0.137 | 0.171 | 0.250 | 0.220 | 0.194 |
| 9 | 0.156 | 0.196 | 0.231 | 0.248 | 0.208 |
| 10 | 0.166 | 0.219 | 0.246 | 0.289 | 0.230 |

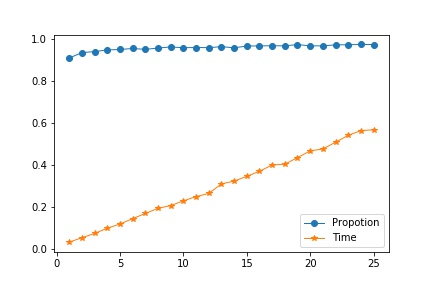


图5.4 不同运行轮数的平均比值和平均时间

从途中我们可以看到主要的优化结果的提升发生在运行轮数小于18之前。而之后的轮数不仅没有提升算法的结果，反而花费更多的时间，因此我们把运行轮数设置成18.

5.4 和Lingo 17运行结果比较

为了测试蚁群算法的精度和效率，我们把蚁群算法的运行结果和Lingo 17的运行结果进行比较。所有问题都用蚁群算法计算了10次并且最好的结果被保存下来，而且所有问题也被Lingo 17使用精确算法求解了相关的优化模型计算了一次。蚁群算法的参数的设置根据上面的讨论进行设置。算法的结果可以从表15看到。同时，图5.5给出了n=18的四个问题的对应的结果，我们可以看到比较好的效果。

蚁群算法使用Python在Pycharm上进行实现，运行的平台是AMD A8-7200P Radeon, 8核，2.4GHz。而Lingo 17解决优化模型的计算平台是阿里云的云计算平台，服务器的配置是Intel(R) Xeon(R) Platinum 8163处理器，2.5GHz。

表5.15 蚁群算法结果和Lingo优化模型结果比较

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 问题种类 | 问题顺序 | 蚁群算法最好目标结果 | Lingo 最优目标结果 | 最优解结果和最好结果的比值 | 蚁群算法运行时间 | Lingo运行时间 |
| N=12 | 1 | 100.303 | 100.300 | 1.000 | 0.308 | 4.170 |
| 2 | 140.413 | 140.410 | 1.000 | 0.261 | 119.830 |
| 3 | 142.795 | 142.790 | 1.000 | 0.300 | 4.520 |
| 4 | 122.464 | 121.320 | 0.991 | 0.316 | 2.010 |
| N=14 | 1 | 119.314 | 116.440 | 0.976 | 0.378 | 18.520 |
| 2 | 150.538 | 145.700 | 0.968 | 0.349 | 99.350 |
| 3 | 146.108 | 146.100 | 1.000 | 0.387 | 7.590 |
| 4 | 133.785 | 133.780 | 1.000 | 0.373 | 142.000 |
| N=16 | 1 | 128.979 | 119.310 | 0.925 | 0.503 | 12.080 |
| 2 | 131.526 | 131.520 | 1.000 | 0.501 | 45.360 |
| 3 | 147.257 | 144.010 | 0.978 | 0.489 | 623.500 |
| 4 | 173.070 | 172.020 | 0.994 | 0.443 | 629.810 |
| N=18 | 1 | 130.582 | 128.330 | 0.983 | 0.514 | 130.640 |
| 2 | 139.497 | 139.490 | 1.000 | 0.507 | 157.890 |
| 3 | 145.565 | 145.170 | 0.997 | 0.564 | 273.160 |
| 4 | 165.479 | 163.900 | 0.990 | 0.607 | 1828.010 |

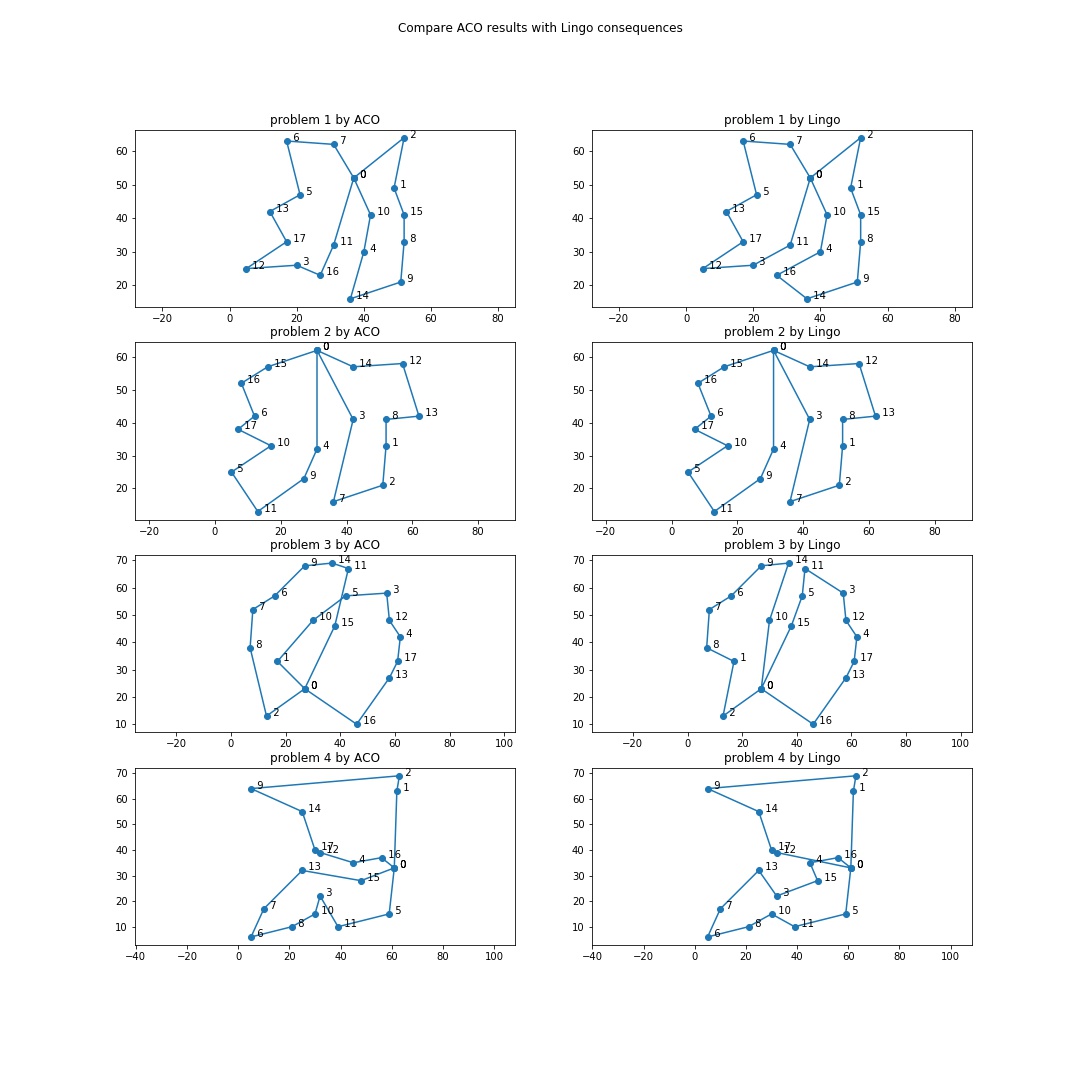


图5.5 ACO和Lingo结果对比

5.5 结果讨论

从表15看到，和Lingo优化模型的效果比起来，本文所实现的蚁群算法的精度和效率令人兴奋。大多数情况下，和最优解果比起来，蚁群算法提供的解的精度丢失不大，而且蚁群算法所需要的运行时间也非常小。对于n=18的一组问题，平均的精度的损失大约在百分之0.8，而运行时间上蚁群算法的平均时间是0.55秒，Lingo运行时间是597.42秒，蚁群算法大概节省了百分之99.9的运行时间。这个结果显示出我们算法的精度的优良和运行效率的良好。

1. 结论

在本文中，我们首先讨论了课题研究背景和意义，指出多旅行商问题在现实生活中的广泛的应用。其次，我们讨论了解决多旅行商问题的现有的方案，我们可以看到对于大规模的多旅行问题而言无法使用精确算法求解多旅行商问题的优化问题，只能能使用启发式算法来求解多旅行商问题。我们指出，蚁群算法是解决旅行商问题的优秀的算法并且将设计一种蚁群算法来解决多旅行商问题。

之后我们探讨了课题研究的重点和蚁群算法的一般流程。

最后，基于对算法特征的理解，我们设计了一种基于有限访问城市数量和交叉避免的蚁群算法，用来解决目标函数为最小化最长路径的多旅行商问题。首先我们讨论了该蚁群算法的设计和交叉避免的算法流程，其次我们用eil51这个基准数据产生了20不同规模的测试问题，此外我们进行了参数调优。最后，我们用调节的参数解决所有问题并且把得到的结果和Lingo产生的最优解进行比较。从比较的结果看到，我们的算法的精度和效率都十分令人满意。

致谢

大学四年就要过去了，我要感谢在这四年热心帮助过我的老师和同学。大学四年里，承蒙曾令秋老师、刘铎老师、陈贤彰老师，吴全旺安咯是的照顾，让我在学习生活中收获到丰富的知识和技能，也让我见识到不同的工作态度和风采，开拓了我的视野和胸襟。

同时也感谢陪我一起参加比赛的同学，他们的每个人都有着让人着迷的闪光点，他们一丝不苟、胸怀荣誉、不懈奋斗的精神让人十分着迷和钦佩。

这个大学里面取得的成绩和收获都离不开老师和同学们的帮助，这里真心感谢他们的陪伴和支持。同时这也是人生十分美好的回忆和依恋。

参 考 文 献

1. Tolga Bektas.The multiple traveling salesman problem: an overview of formulations and solution procedures[J].The International Journal of Management Science,2006,34(1):209–219
2. Gorenstein S.Printing press scheduling for multi-edition periodicals[J].Management Science,1970,16(6):B373–83.
3. Carter AE, Ragsdale CT.Scheduling pre-printed newspaper advertising inserts using genetic algorithms[J}.Omega,2002,30:415–21.
4. Svestka JA, Huckfeldt VE.Computational experience with an m-salesman traveling salesman algorithm[J].Management Science,1973,19(7):790–9.
5. Angel RD, Caudle WL, Noonan R, Whinston A.Computer assisted school bus scheduling[J].Management Science,1972,18:279–88.
6. Gilbert KC, Hofstra RB.A new multiperiod multiple traveling salesman problem with heuristic and application to a scheduling problem[J].Decision Sciences,1992,23:250–9.
7. Brummit B, Stentz A. Dynamic mission planning for multiple mobile robots[J].Proceedings of the IEEE international conference on robotics and automation,1996.3:2396 - 2401
8. LaporteG, NobertY.A cutting planes algorithm for them-salesmen problem[J].Journal of the Operational Research Society,1980,31:1017-1023.
9. AliAI,KenningtonJL.The asymmetric m-traveling salesmen problem:aduality based branch-and-bound algorithm[J].Discrete Applied Mathematics,1986,13(2-3):259-276.
10. GavishB,SrikanthK.An optimal solution method for large-scale multipletraveling salesman problems[J].Operations Research,1986,34(5):698-717.
11. Gromicho J, Paixao J, Branco I.Exact solution of multiple traveling salesman problems[M]//Mustafa Akgül, et al., editors, Combinatorial optimization. NATO ASI Series, 1992, 82:291-292.
12. WacholderE, HanJ, MannRC. A neural network algorithm for the multiple traveling salesmen problem[J]. Biology Cybernetics, 1989, 61(1):11-19.
13. HsuC, TsaiM, ChenW. Astudy of feature-mapped approach to the multiple travelling salesmen problem [C]//IEEE International Symposiumon Circuitsand Systems. Singapore:IEEEPress, 1991, 3:1589-92.
14. ZhangT, Gruver WA, Smith MH. Team scheduling by genetic search[C]//Proceedings of the 2nd international conference on intelligent processing and manufacturing of materials. Honolulu, HI, USA:IEEEPress, 1999, 2:839-844.
15. Tang L, Liu J, Rong A, Yang Z. A multiple traveling salesman problem model for hot rolling scheduling in Shang ai Bao shan Iron & Steel Complex[J]. European Journal of Operational Research, 2000, 124(2):267-282.
16. ]Pan J J, Wang D W. An Ant Colony Optimization Algorithm for Multiple Travelling Salesman Problem[C]// Proceedings of the First International Conferenceon Innovative Computing, Informationand Control. Beijing:IEEEPress, 2006, 1:210-213.
17. Gorenstein S. Printing press scheduling for multi-edition periodicals[J]. Management Science, 1970, 16(6):373-83.
18. Svestka J A, Huckfeldt V E. Computational experience with an m-salesman traveling salesman algorithm[J]. Management Science, 1973, 19(7):790-799.
19. Bellmore M, Hong S. Transformation of multi salesmen problem to the standard traveling salesman problem[J]. Journal of the ACM(JACM), 1974, 21(3):500-504.
20. Hong S, Padberg M W. A note on the symmetric multiple traveling salesman problem with fixed charges[J]. Operations Research, 1977, 25(5):871-874.
21. Rao M R. A note on the multiple traveling salesman problem[J]. OperationsResearch, 1980, 28(3):628-632.
22. Marco Dorigo, Luca Maria Gambardella.Ant Colony System: A Cooperative Learning Approach to the Traveling Salesman Problem[J].IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION,1997,1(1):53-66
23. Soheil Ghafurian∗, Nikbakhsh Javadian. An ant colony algorithm for solving fixed destination multi-depot multiple
24. traveling salesmen problems[J]. Applied Soft Computing, 2011, 11:1256–1262
25. Yu Qingsheng, Lin Dongmei, Wang Dong. An Overview of Multiple Traveling Salesman Problem[J]. China Academic Journal Electronic Publishing House, 2012, 31(2):166-168