# 旅游路线规划问题

## 摘 要

随着国民经济的快速发展，旅游活动成为现代生活的重要组成部分。本题要求为自驾游爱好者制定全国 201 个 5A 景点旅游计划。本文采用分层优化方法， 建立两层带两个时间窗的旅行商问题模型、修正的带时间窗多车辆路径问题模型， 采用改进的遗传算法并借助 MATLAB 编程求解，在第一问中得到以出游年数最 短为目标的自驾出游方案。第二问中，在对两层模型目标和约束修正的基础上， 为旅游者提供以体验最佳、费用最优的多交通方式出行的旅游规划。第三问中， 本文将模型进行推广，为北京的自驾游游客制定了游览全国 5A 景点的十年旅游 计划，并为旅游者和相关旅游部门提供可借鉴性建议。第四问中，基于第二问所 建立的模型，增加游览时间比（游览时间占总出游时间的比例）最大的目标，将 4A 景区纳入出游考虑范围，制定了更为合理的十年旅游计划。 对于问题1，本文采用分层优化的思想，下层为各省会城市辐射区内景点的 路径规划，以出行时间最短为目标，在景点开放时间、车行区间等限制因素下， 建立带两个时间窗的旅行商问题模型，采用双种群遗传算法并借助MATLAB 编 程软件进行求解，得到每个省会城市辐射区内景点的最短时间游览路线。在下层 优化的基础上，上层模型以旅游年数最少为目标，将201 个景点出游路径规划简 化为31 个省市游览路线优化，对带时间窗的多车辆路径问题（MVRPTW）模型 进行修正，使同一城市游览时间超过 15 天限制时拆分为两次游览。对于问题 2，本文首先比较“自驾”、“飞机+租车”和“高铁+租车”三种交通方 式的费用-距离曲线，得到高铁优先考虑、一天车程范围内采用自驾方式出游，两天以上车程选用飞机出行的结论。在此基础上，对第一问两层优化模型进行修正，以总游览年数最少、自驾出行次数最少和费用最优为一、二级目标，增加票价、车油费过路费、住宿费、租车费等费用约束。对于问题 3，本文将前两问中的模型进行推广，将出发地由西安改为北京， 运用问题2 中的模型和算法进行求解，得到最优的十年旅游计划，并给出每次出 行方式、出行行程、天数和游览景区信息，有效验证了模型的可行性和适用性。 最后根据不同优化目标的最优旅游计划为旅游者和旅游部门分别提出三条和四 条建议。对于问题 4，基于问题 2 中模型，在游览 5A 级景区的基础上，将 4A 级景 区纳入考虑范围，并引入游览时间比（游览时间/总出行时间）为目标函数，对问 题2 中得到的出行规划进行优化，有效利用了“间隙”时间，得到出行体验更佳、 更加合理的出行规划，并给出十年出行规划。

**关键词**：旅行商问题(TSP) 多车辆路径问题(MVRP) 遗传算法(GA) 旅游计划

目 录

**[一、问题重述](#_Toc28288_WPSOffice_Level1)** **[1](#_Toc28288_WPSOffice_Level1)**

[需要考虑如下问题：](#_Toc7971_WPSOffice_Level2) [1](#_Toc7971_WPSOffice_Level2)

**[二、问题分析](#_Toc7971_WPSOffice_Level1)** **[2](#_Toc7971_WPSOffice_Level1)**

[2.1 问题一的分析](#_Toc14340_WPSOffice_Level2) [2](#_Toc14340_WPSOffice_Level2)

[2.2 问题二的分析](#_Toc28240_WPSOffice_Level2) [2](#_Toc28240_WPSOffice_Level2)

[2.3 问题三的分析](#_Toc21584_WPSOffice_Level2) [2](#_Toc21584_WPSOffice_Level2)

[2.4 问题四的分析](#_Toc9368_WPSOffice_Level2) [2](#_Toc9368_WPSOffice_Level2)

**[三、模型假设](#_Toc14340_WPSOffice_Level1)** **[3](#_Toc14340_WPSOffice_Level1)**

**[三、算法介绍](#_Toc28240_WPSOffice_Level1)** **[4](#_Toc28240_WPSOffice_Level1)**

**[四、模型的建立与求解](#_Toc21584_WPSOffice_Level1)** **[5](#_Toc21584_WPSOffice_Level1)**

[4.1 问题一的求解](#_Toc5653_WPSOffice_Level2) [5](#_Toc5653_WPSOffice_Level2)

[4.2问题二的求解](#_Toc25931_WPSOffice_Level2) [10](#_Toc25931_WPSOffice_Level2)

[4.3 问题三的求解](#_Toc23603_WPSOffice_Level2) [12](#_Toc23603_WPSOffice_Level2)

[4.4问题四的求解：](#_Toc6726_WPSOffice_Level2) [13](#_Toc6726_WPSOffice_Level2)

**[参考文献](#_Toc9368_WPSOffice_Level1)** **[14](#_Toc9368_WPSOffice_Level1)**

**[附录](#_Toc5653_WPSOffice_Level1)** **[15](#_Toc5653_WPSOffice_Level1)**

## 

## 一、问题重述

一位自驾游爱好者拟制定旅游计划遍历全国 201 个5A 级景区。该旅游爱好

者自身的限制条件：

（1）每年外出旅游时间不超过 30 天；

（2）每年外出旅游的次数不超过 4 次；

（3）每次旅游的时间不超过 15 天；

（4）根据个人偏好，每个 5A 景区有最少游览时间；

时间窗限制条件为：

（5）行车时间限定于每天 7:00 至19:00 之间，每天开车时间不超过 8 小时；

（6）景区开放时间统一为 8:00 至18:00。

在每天的行程安排上，有以下规则：

（7）若安排全天游览，开车时间控制在 3 小时内；

（8）若安排半天景点游览，开车时间控制在 5 小时内；

（9）在每个省会城市至少停留 24 小时感受风土人情（不安排景区浏览）；

对于道路行车，速度限制如下：

（10）高速公路上的行车平均速度为 90 公里/小时；

（11）普通公路上的行车平均速度为 40 公里/小时。

#### 需要考虑如下问题：

（1）行车线路采用高速优先原则，即先通过高速公路到达与景区邻近的城市，再自驾到景区，以常住地在西安市为例，规划设计游遍201 个5A 级景区旅游线路，确定花费年数及每一次旅游的具体行程。

（2）出行方式综合考虑全程自驾、先乘坐高铁或飞机到达省会城市后再租车自驾到景区等出行方式，租车费用 300 元/天，油费和高速过路费另计，租车和还车需在同一城市。住宿费简化为省会城市和旅游景区 200 元/人•天，地级市150 元/人•天，县城 100 元/人•天。高速公路的油耗加过路费平均为 1.00 元/公里，普通公路上油耗平均为 0.60 元/公里。假设每一个景区最长逗留时间不超过最少时间的2 倍，选择高铁出行要求当天乘坐高铁的时间不超过 6 个小时，且至多安排半天的游览。该旅游爱好者一家 3 人同行，要求设计一个十年游遍所有 201 个5A 景区的费用最优、旅游体验最好的旅游线路，并给出每一次旅游的具体线路。

（3）能否在第二问所建立的模型基础上加以推广，可以为全国的自驾游爱好者规划设计类似的旅游线路，进而给出常住地在北京市的自驾游爱好者的十年旅游计划；根据上述三问的结果给旅游爱好者和旅游有关部门提出建议。

（4）自2007 年 3 月 7 日至2015 年7 月13 日，全国旅游景区质量等级评定委员会分29 批共批准了 201 家景区为国家5A 级旅游景区。结合国家4A 级景区名单，请更为合理地规划该旅游爱好者的十年旅游计划。

## 二、问题分析

本题要求根据不同的条件约束不同的目标规划旅游路线，具体问题分析如 下：

#### 2.1 问题一的分析

本问中要求制定以西安为常驻地的最短年数 201 个5A 景点出游计划，并给 出每次具体行程。本文采用分层优化思想，下层以出行时间最短为目标，建立带 两个时间窗的旅行商问题模型，求每个省会城市辐射区内景点的最短时间游览路 线。上层模型以旅游年数最少为目标，将 201 个景点出游路径规划简化为 31 个 省市游览路线优化，对带时间窗的多车辆路径问题模型进行修正，采用遗传算法 并借助MATLAB 编程求解，得到 201 个 5A 景区出游总年数最少的出游计划。

#### 2.2 问题二的分析

比较“自驾”、“飞机+租车”和“高铁+租车”三种交通方式的费用-距离曲线，求 得出行方式选择与距离关系。在此基础上，对第一问两层优化模型进行目标、约 束的修正，以总游览年数最少、自驾出行次数最少和费用最优为一、二级目标， 增加票价、车油费过路费、住宿费、租车费等费用约束，以求最优的十年出游计 划。

#### 2.3 问题三的分析

本题将前两问中的模型进行推广，将出发地由西安改为北京，运用问题 2 中 的模型和算法进行求解，得到最优的十年旅游计划，以此验证模型的可行性和适 用性，最后根据不同优化目标的最优旅游计划为旅游者和旅游部门提出可行性建 议。

#### 2.4 问题四的分析

本题基于问题 2 中模型，在游览5A 级景区的基础上，将 4A 级景区纳入考 虑范围，并引入游览时间比（游览时间/总出行时间）为目标函数，对问题 2 中 得到的出行规划进行优化，有效利用了“间隙”时间，从而得到出行体验更佳、 更加合理的十年出行规划。

## 三、模型假设

1. 假设采用高速优先的策略，即先通过高速公路到达与景区邻近的城市，再自驾到景区。

2. 假设在两个景点邻近城市之间的道路均为高速公路，在高速公路上的行车平均速度为 90 公里/小时；景点邻近城市到邻近景点的道路为普通公路，驾驶距离与时间参照附件 1。

3. 据题意可假设“一天”等于“8 个小时”。

4. 假设某省会城市辐射区景点是指与该省行政区域范围内城市邻近的所有景点（包括省内、省外景点）。

5. 某一省市内景点邻近城市的选取根据附件 1 中景点与各城市距离时间数据确定。

6. 游客若需要租车游览，均在省会城市租还车。

7. 由于同一省内各城市间距离较近，且需要在省会城市租还车，因此为节省费用，省内出行均采用自驾或租车方式。

8. 假设“体验最佳”是指游览 201 个景点总年数最少，并尽可能减少自驾出游次数，即对于远距离的景点城市尽量选用飞机、高铁交通方式。

9. 假设“体验最佳”为第一级目标，“费用最优”为第二级目标，两者权重值 P1：P2=3:2.

10. 由于门票、饮食等费用，不影响出游路线规划，为简化计算，假设出游费用不包括门票、饮食等费用。

11. 若游客当天 19：00 结束驾驶时，位于两个城市的驾车途中，则假设当晚的住宿费用与地级市的住宿费用相同，为 150 元/天人。

## 三、算法介绍

遗传算法（Genetic Algorithm）是模拟[达尔文](https://baike.baidu.com/item/%E8%BE%BE%E5%B0%94%E6%96%87/23890" \t "https://baike.baidu.com/item/_blank)[生物进化](https://baike.baidu.com/item/%E7%94%9F%E7%89%A9%E8%BF%9B%E5%8C%96/3311428" \t "https://baike.baidu.com/item/_blank)论的自然选择和[遗传学](https://baike.baidu.com/item/%E9%81%97%E4%BC%A0%E5%AD%A6/233918" \t "https://baike.baidu.com/item/_blank)机理的生物进化过程的计算[模型](https://baike.baidu.com/item/%E6%A8%A1%E5%9E%8B/1741186" \t "https://baike.baidu.com/item/_blank)，是一种通过模拟自然进化过程搜索[最优解](https://baike.baidu.com/item/%E6%9C%80%E4%BC%98%E8%A7%A3/5208902" \t "https://baike.baidu.com/item/_blank)的方法。遗传算法是从代表问题可能潜在的解集的一个[种群](https://baike.baidu.com/item/%E7%A7%8D%E7%BE%A4/798044" \t "https://baike.baidu.com/item/_blank)（population）开始的，而一个种群则由经过[基因](https://baike.baidu.com/item/%E5%9F%BA%E5%9B%A0/227875" \t "https://baike.baidu.com/item/_blank)（gene）编码的一定数目的个体(individual)组成。每个个体实际上是[染色体](https://baike.baidu.com/item/%E6%9F%93%E8%89%B2%E4%BD%93/195881" \t "https://baike.baidu.com/item/_blank)(chromosome)带有特征的实体。染色体作为[遗传物质](https://baike.baidu.com/item/%E9%81%97%E4%BC%A0%E7%89%A9%E8%B4%A8/4196140" \t "https://baike.baidu.com/item/_blank)的主要载体，即多个基因的[集合](https://baike.baidu.com/item/%E9%9B%86%E5%90%88/2908117" \t "https://baike.baidu.com/item/_blank)，其内部表现（即[基因型](https://baike.baidu.com/item/%E5%9F%BA%E5%9B%A0%E5%9E%8B/2574789" \t "https://baike.baidu.com/item/_blank)）是某种基因组合，它决定了个体的形状的外部表现，如黑头发的特征是由染色体中控制这一特征的某种基因组合决定的。因此，在一开始需要实现从[表现型](https://baike.baidu.com/item/%E8%A1%A8%E7%8E%B0%E5%9E%8B/5106257" \t "https://baike.baidu.com/item/_blank)到基因型的[映射](https://baike.baidu.com/item/%E6%98%A0%E5%B0%84/410062" \t "https://baike.baidu.com/item/_blank)即[编码](https://baike.baidu.com/item/%E7%BC%96%E7%A0%81/80092" \t "https://baike.baidu.com/item/_blank)工作。由于仿照基因编码的工作很复杂，我们往往进行简化，如[二进制](https://baike.baidu.com/item/%E4%BA%8C%E8%BF%9B%E5%88%B6" \t "https://baike.baidu.com/item/_blank)编码，初代种群产生之后，按照适者生存和优胜劣汰的原理，逐代（generation）演化产生出越来越好的近似解，在每一代，根据问题域中个体的[适应度](https://baike.baidu.com/item/%E9%80%82%E5%BA%94%E5%BA%A6/1052009" \t "https://baike.baidu.com/item/_blank)（fitness）大小选择（selection）个体，并借助于自然遗传学的遗传[算子](https://baike.baidu.com/item/%E7%AE%97%E5%AD%90/970194" \t "https://baike.baidu.com/item/_blank)（genetic operators）进行组合交叉（crossover）和变异（mutation），产生出代表新的解集的种群。这个过程将导致种群像自然进化一样的后生代种群比前代更加适应于环境，末代种群中的最优个体经过[解码](https://baike.baidu.com/item/%E8%A7%A3%E7%A0%81/10944752" \t "https://baike.baidu.com/item/_blank)（decoding），可以作为问题近似最优解。

遗传算法也是计算机科学人工智能领域中用于解决最优化的一种搜索启发式算法，是进化算法的一种。这种启发式通常用来生成有用的解决方案来优化和搜索问题。进化算法最初是借鉴了进化生物学中的一些现象而发展起来的，这些现象包括遗传、突变、自然选择以及杂交等。遗传算法在适应度函数选择不当的情况下有可能收敛于局部最优 [1]  ，而不能达到全局最优。

遗传算法的[基本运算](https://baike.baidu.com/item/%E5%9F%BA%E6%9C%AC%E8%BF%90%E7%AE%97" \t "https://baike.baidu.com/item/_blank)过程如下：

a)初始化：设置进化代数计数器t=0，设置最大进化代数T，随机生成M个个体作为初始群体P(0)。

b)个体评价：计算群体P(t)中各个个体的[适应度](https://baike.baidu.com/item/%E9%80%82%E5%BA%94%E5%BA%A6" \t "https://baike.baidu.com/item/_blank)。

c)[选择运算](https://baike.baidu.com/item/%E9%80%89%E6%8B%A9%E8%BF%90%E7%AE%97" \t "https://baike.baidu.com/item/_blank):将选择算子作用于群体。选择的目的是把优化的个体直接遗传到下一代或通过配对交叉产生新的个体再遗传到下一代。选择操作是建立在群体中个体的[适应度](https://baike.baidu.com/item/%E9%80%82%E5%BA%94%E5%BA%A6" \t "https://baike.baidu.com/item/_blank)评估基础上的。

d)交叉运算：将交叉算子作用于群体。遗传算法中起核心作用的就是交叉算子。

e)[变异运算](https://baike.baidu.com/item/%E5%8F%98%E5%BC%82%E8%BF%90%E7%AE%97" \t "https://baike.baidu.com/item/_blank)：将变异算子作用于群体。即是对群体中的个体串的某些[基因座](https://baike.baidu.com/item/%E5%9F%BA%E5%9B%A0%E5%BA%A7" \t "https://baike.baidu.com/item/_blank)上的基因值作变动。

群体P(t)经过选择、交叉、[变异运算](https://baike.baidu.com/item/%E5%8F%98%E5%BC%82%E8%BF%90%E7%AE%97" \t "https://baike.baidu.com/item/_blank)之后得到下一代群体P(t+1)。

f)终止条件判断:若t=T,则以进化过程中所得到的具有最大[适应度](https://baike.baidu.com/item/%E9%80%82%E5%BA%94%E5%BA%A6" \t "https://baike.baidu.com/item/_blank)个体作为[最优解](https://baike.baidu.com/item/%E6%9C%80%E4%BC%98%E8%A7%A3" \t "https://baike.baidu.com/item/_blank)输出，终止计算。

## 四、模型的建立与求解

### 4.1 问题一的求解

据题意可知，要求线路设计采用高速优先策略，先通过高速公路到达与景区邻近的城市，再自驾到景区，因此，与景区邻近的城市可作为路线规划中的必经节点。其二，要求旅游爱好者在每个省会城市除去游览景区外，至少停留 24 个小时感受风土人情，因此，每个省会城市也是路线选择的必经节点。在必须到达省会城市、景点邻近城市的要求基础上，本题采用分层优化思想，以简化求解。在下层路径优化中，以必经节点省会城市为起点，对省会城市辐射区内景点邻近城市进行路线规划，由此可以得到 31 个省会城市辐射区内景点最优旅游路线方案。在此基础上，每个省会城市辐射区内景点游览总时间等价于该省会城市所需观光时间，上层优化中原有的 201 个 5A 景区路线规划由此简化为 31 个省会城市路线优化。通过两层优化模型的求解，从而可以制定游览年数最少的 201 个5A 景区出游计划。

#### 4.1.1下层路径优化：

**模型建立：**

本层的带两个时间窗的旅行商问题模型以履行总时间最少为目标，以行车时

间、景点开放时间等限制因素为约束，如下所示：

**目标函数**：



**模型求解：**

本模型的输入数据为各省市内任意景点邻近城市之间的距离、邻近城市至景点的距离、每个景点的游览时间等。其中，景点邻近城市的选取根据题目所给景点至各城市的距离时间数据确定，则 31 个省市邻近城市选取及景点邻近城市之间的OD 矩阵如附录 1 所示。本文通过双种群遗传算法[3][4]，对问题进行求解，具体步骤如下：

1. 编码

我们采用简单易用的整数编码方法。对于有 n 个访问点的 TSP 问题，将景点所城市从 1 到 n 进行编号，每个从 1 到 n 的全排列都可以作为该问题的一个解，即染色体个体。例如有 5 个景点城市的 TSP 问题，{3，5，4，2，1}就是一个合法的染色体，表示车辆对 5 个省会城市的访问路线是 3-5-4-2-1。

1. 产生初始群体

随机产生初始群体

1. 求取适应度函数

利用遗传算法寻优，就是要找到适应度最大的染色体，而软时间窗约束的TSP 的目标函数是计算复杂度会较高。由于适应度函数一般要求非负，因此将目标函数转化为适应度函数，这里我们通过变换来达到目的。此处为染色体适应度，为一常数，为初始种群最佳染色体的旅行成本，为染色体所对应的旅行成本。

#### **遗传算子的设计**

（1） 交叉算子采用正交交叉方式。记当前的第 K 代种群为先对种群进行两两配对，按照预先产生的正交阵对种群进行交叉，得到种群Bk。

（2） 对交叉后得到的种群Bk中的每个体Xi以预先给定的变异概率进行变异。

（3） 对变异后的种群中最差的 S 个体进行更新操作，并得的种群为Dk。

#### **5. 种群交叉**

将两个种群中最优解取出，再在每个种群中随机选取 num 个染色体，将这num+1 个染色体互换，进入对方种群。从而形成最优的遗传信息。

#### **6. 终止条件**

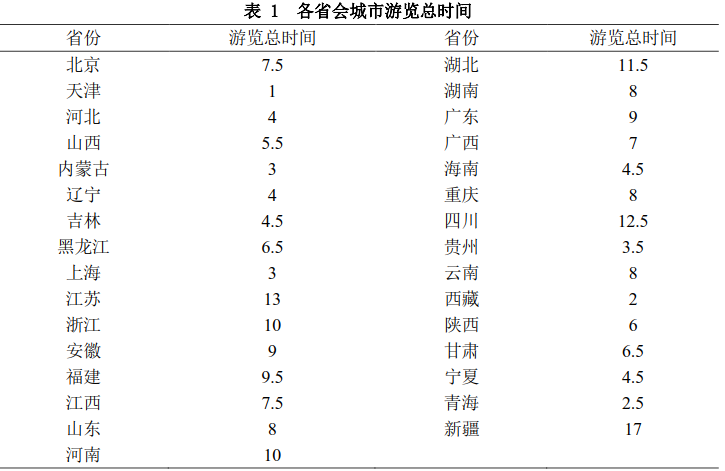
由于遗传算法具有较大的随机性，这里使用几条基于启发式规则的终止条件：

（1） 若算法迭代到 100 代，则终止算法。

（2） 若某代群体中染色体的平均适应度与当代最佳染色体适应度的比值大于0.9，则终止算法。

1. 若最佳染色体连续保持 10 代，则终止算法。

通过 MATLAB 程序的的算法实现，得到每个省会城市辐射区内景点游览的最佳游览路线及最短时间（已添加 1 天省会城市逗留时间），如表1 所示。



#### 4.1.2上层路径优化 ：

通过下层优化求解，201 个景点最短时间的驾车游览规划可以简化为 31 个

省市的驾车游览路径优化。由题意可知，要求旅游爱好者每年外出旅游时间不超

过30 天，每年外出次数不超过 4 次，每次旅游时间不超过 15 天，求所有省会城市最短游览时间的出游方案。本层优化模型与下层的区别之处在于：其一，由于是多年的多次出游，本层不再是求通过所有节点（省会城市）的一条回路，而是满足出游时间限制的多个回路。其二，到达省会城市的时间不再要求，而在下层中到达景点时间有开业时间段限制。其三，由于某一个省会城市的出游时间就可能超过出游时间 15 天的限制，因此，存在两次到达同一省会城市的情形，而下层旅行商问题中每一个节点的到达和离开次数均是一次。 考虑如上因素，可将该层路径优化问题等价为带时间窗的多车辆路径（MVRPTW）问题。其中，出游次数等价于车辆数量，每次出游的时间限制等价于车辆载重量约束，每个省会城市的游览时间等价于每个节点的货物需求量。因此，可以运用多车辆路径问题的求解思路构建游客最短时间游览规划模型。

**模型的建立**:

**目标函数：**



**模型的求解:**

在车辆调度问题的研究中，常常将复杂的问题转化为相对简单的问题，然后进行求解，从而间接得到复杂问题的解。本题需要游客从西安多次出行遍历所有31个省会城市，可视为多个游客同时从西安出发，花费最短的时间遍历所有 31个省会城市，因而可以转化为单车场多车辆调度规划问题。但是本题的单车场多车辆调度规划问题与传统的单车场多车辆调度规划问题有所不同，单个配送点的配送量可能大于配送车辆的载重。针对上述两点，我们做出如下改进：针对需求超过配送车辆载重的需求点，将该需求点拆分成多个可由一辆车辆完成分配的需求点。在此基础上，本文对传统的多车辆路径问题的遗传算法[8]进行改进，以求解本题模型。

**求解模型的基本遗传算法:**

遗传算法是一种模仿生物进化过程的全局随机搜索方法。遗传算法的基本思想为：从优化问题的一个种群（一组可行解）开始，按照适者生存和优胜劣汰的原理，逐代演化产生出越来越好的一个种群（一组可行解）。

基本遗传算法的执行过程如下：

第一步：初始化群体大小 N，交叉概率 Pe，变异概率 Pm 等参数，随机生成处事种群：X(0)；

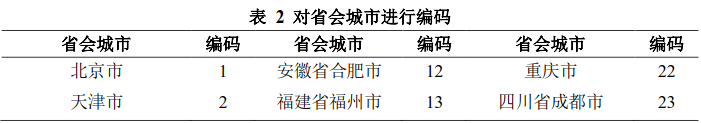
第二步：计算种群中个体的适应度；

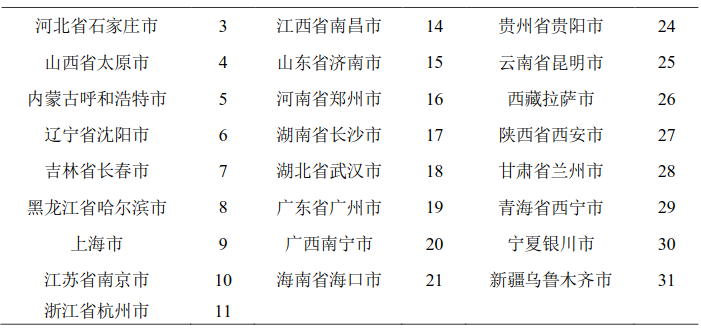
第三步：按照遗传策略，对第 t 代种群X（t）进行选择操作、交叉操作和变

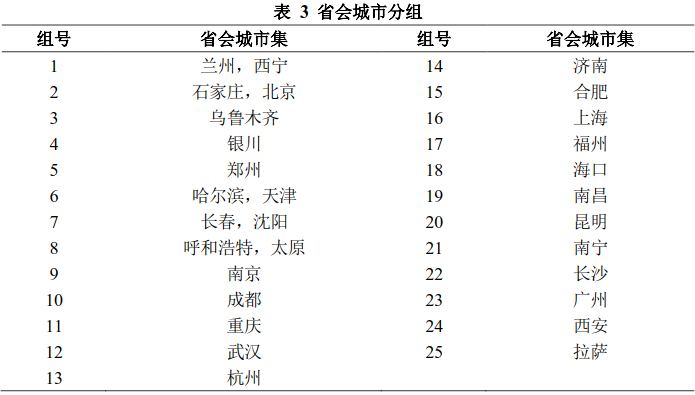
异操作，形成下一代的种群 X(t+1)；

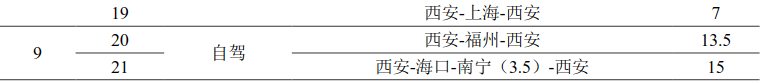
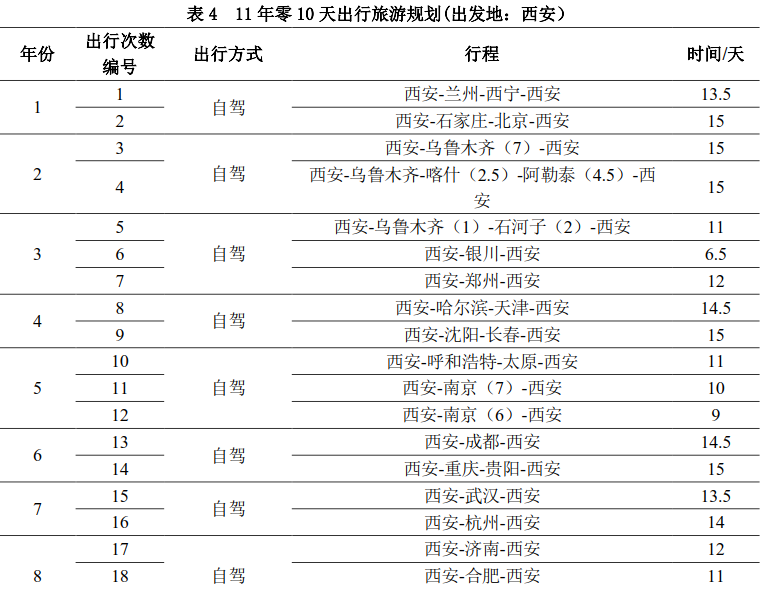
第四步：判断算法是否满足停止准则，如果不满足，则返回到第二步；如果满足，则输出种群中的最大适应度值的个体作为最优解 X\*，终止计算。基本遗传算法使用三种遗传算子，即选择运算使用比例选择算子、交叉算子使用单点交叉算子、变异运算使用基本位变异算子或均匀变异算子。

编码方式：传统的二进制编码及其基于二进制编码的基本遗传操作并不适用于求解 VRP 问题，因此我们采用基于整数符号形式的编码方式，即个体染色体编码串中的基因值是一个无数值含义，只代表被车辆访问客户编号的符号集，如{1，2，3，4，…}等。







通过算法设计和借助 PYTHON程序（主要程序详见附录 2），得到 31 个省会城市的出游年数最少的旅游方案，即全国 201 个 5A 景区的出游方案，出游总时间为 11 年零 10 天，如表 4 和图4 所示。

### 4.2问题二的求解

由题意知，第二问中要求制定费用最优、体验最佳的 201 个 5A 景点旅游计划。游客在每一次出行时有三种交通方式选择：自驾、机票+省内租车、高铁+省内租车。飞机、高铁的选择在节省西安至省会城市路途时间和成本的同时，也增加了每一省市内租车的额外成本。因此，只有当飞机、高铁的节约成本大于额外增加的租车费用时，才满足费用最优要求。

#### 4.2.1 交通方式选择

就西安至各省市的往返程而言，由下图可知，高铁成本远低于自驾、航空两

种交通方式，航空、自驾费用-距离曲线的交点 A 介于（720km，1440km）之间，

即一天至两天的行程距离之间。当距离在 A 之前时，自驾游比飞机出行费用更

优，加上后期城市至景区的租车费用，飞机更不具备优势；当省际城市距离超过

A 时，航空是费用最少的出行方式。因此，从西安至各省会城市的出游，如果有

高铁班次，优先选择高铁交通方式，若没有，若行车时间在 1 天之内，选择自驾方式；当行车时间在 1-2 天之间，进一步比较整个出行旅程中飞机节省成本大与后期租车成本大小；当行车时间大于 2 天时，优先选择飞机交通方式。

#### 4.2.2路径优化：

本题假设“体验最佳”包含两个含义：其一，游览 201 个景点总年数最小，其二，尽可能减少自驾出游次数，即对于远距离的景点城市尽量选用飞机、高铁交通方式。本题同样采用分层优化方法，目标函数为总游览时间最短、自驾次数最少、总游览费用最优。

**模型的建立：**

下层优化是对某一省会城市辐射区内景点的旅游规划，由于同一省内各城市间距离较近，为节省费用，省内出行均采用自驾或租车方式。游客若需要租车游览，均在省会城市租还车。并且是否租车、自驾次数由上层交通方式选择决定，因此租车费用、票价因素和自驾次数最少目标均将在上层考虑。本层的优化目标为总游览年数最短、总费用最优，优化的费用仅考虑油费、过路费、住宿费。

目标函数：



模型的求解：

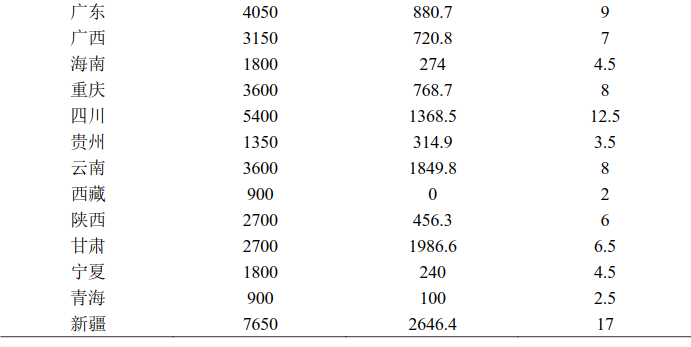
模型求解算法沿用问题一的求解算法----双种群遗传算法。由于问题二考虑了

出行费用、住宿费用、租车费用等，目标函数是要求总游览时间最短、自驾次数

最少、总游览费用最优，因而目标函数发生了变化。所以解决本模型的遗传算法

的适应度函数，应当相应做适当调整。





### 4.3 问题三的求解

问题三是问题二的推广。对于全国的自驾游爱好者来说，问题二所建模型可以基于其出发地计算出 10 年旅游规划路线。基于问题二所建立的模型，在问题三中只需要修改出发地点和模型输入变量即可得到自北京出发的 10 年旅游规划。

5.3.1 建议的提出

通过对比时间最短的自驾出游方案和体验最佳、费用最优的组合方式出游方案的结果，本文对旅游者和相关旅游部门提出部分可借鉴性的建议，以期望推动中国旅游业稳步发展。

5.3.2.1 对旅游者的建议

对于旅游者而言，有选择出游交通方式的自主权，可供选择的方式有自驾游、“飞机+租车”、“高铁/铁路+租车”等。随着全国交通网络完善，以及游客对个性化出游的追求，自驾游已经越来越普遍。但自驾游在满足自由旅游的同时，其高强度的驾车时长要求会大大降低旅游体验效果。旅游者应当根据旅游景点的分布不同，适当选择出游交通方式。

（1） 中短途自驾，提升旅游体验

完全自驾出游只建议中短途距离的游览，当城市之间的车程超过两天时，长距离的驾车不仅不能达到休闲放松的目的，也存在较大的安全隐患。因此，为使旅游体验最大化，可在中短途（两天车程之内）的旅程中选择自驾出游方式。

（2） 组合出游，高铁优先

建议高铁+租车、飞机+租车的组合出游方式，既可以满足自驾出游的个性化、自由的要求，也能够避免长距离驾车路途的劳顿。由自驾、高铁和飞机的费用-距离曲线可以看出，同等距离条件下，高铁的费用远低于其他两种方式。与自驾相比，高铁不仅速度占优势，也可减少驾车路途劳累，因此当出行的省市与出发地有高铁班次时，高铁应作为首要选择。

（3）目标明确，合理规划

确定每次出游的目的，合理规划线路，选择组合游览景点。由第一二问可知，以时间最短为目标的最佳出游路径与体验最佳、费用最优的出游方案完全不同。旅游者应当根据自身经济、时间等条件，制定合理的旅游路径，使旅游效用最大化。

### 4.4问题四的求解：

同问题二

## 参考文献

1. A. Beham. Parallel tabu search and the multiobjective vehicle routing problem with time windows. In 21th International Parallel and Distributed Processing Symposium, pages 1-8. IEEE Computer Society, 2007.
2. 李敏, 吴浪, 张开碧. 求解旅行商问题的几种算法的比较研究[J]. 重庆邮电大学学报(自然科学版), 2008, 20(5): 624-626.
3. Deb, K., Pratap, A., Agarwal, S., & Meyarivan, T. A. M. T. (2002). A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE transactions on evolutionary computation, 6(2), 182-197.
4. E. Alba and B. Dorronsoro. Computing nine new best-so-far solutions for capacitated VRP with a cellular genetic algorithm. Inform. Process. Lett.,98(6):225-230,2006.

## 附录

源程序：

TSP.py

|  |
| --- |
| # -\*- coding: utf-8 -\*-  """TSP.py  TSP问题  """  import sys  import random  import math  import time  import Tkinter  import threading  from GA import GA  class MyTSP(object):  """TSP"""  def \_\_init\_\_(self, root, width=800, height=600, n=32):  self.root = root  self.width = width  self.height = height  self.n = n  self.canvas = Tkinter.Canvas(  root,  width=self.width,  height=self.height,  bg="#ffffff",  xscrollincrement=1,  yscrollincrement=1  )  self.canvas.pack(expand=Tkinter.YES, fill=Tkinter.BOTH)  self.title("TSP")  self.\_\_r = 5  self.\_\_t = None  self.\_\_lock = threading.RLock()  self.\_\_running = False  self.nodes = []  self.nodes2 = []  self.ga = None  self.order = []  self.\_\_bind\_events()  self.new()  def \_\_bind\_events(self):  self.root.bind("q", self.quite)  self.root.bind("n", self.new)  self.root.bind("e", self.evolve)  self.root.bind("s", self.stop)  def title(self, s):  self.root.title(s)  def new(self, evt=None):  self.\_\_lock.acquire()  self.\_\_running = False  self.\_\_lock.release()  self.clear()  self.nodes = [] # 节点坐标  self.nodes2 = [] # 节点图片对象  for i in range(self.n):  x = random.random() \* (self.width - 60) + 30  y = random.random() \* (self.height - 60) + 30  self.nodes.append((x, y))  node = self.canvas.create\_oval(  x - self.\_\_r,  y - self.\_\_r, x + self.\_\_r, y + self.\_\_r,  fill="#ff0000",  outline="#000000",  tags="node",  )  self.nodes2.append(node)  self.ga = GA(  life\_count=50,  mutation\_rate=0.05,  judge=self.judge(),  mk\_life=self.mk\_life(),  x\_func=self.x\_func(),  m\_func=self.m\_func(),  save=self.save()  )  self.order = range(self.n)  self.line(self.order)  def distance(self, order):  """得到当前顺序下连线总长度"""  distance = 0  for i in range(-1, self.n - 1):  i1, i2 = order[i], order[i + 1]  p1, p2 = self.nodes[i1], self.nodes[i2]  distance += math.sqrt((p1[0] - p2[0]) \*\* 2 + (p1[1] - p2[1]) \*\* 2)  return distance  def mk\_life(self):  def f():  lst = range(self.n)  random.shuffle(lst)  return lst  return f  def judge(self):  """评估函数"""  return lambda lf, av=100: 1.0 / self.distance(lf.gene)  def x\_func(self):  """交叉函数"""  def f(lf1, lf2):  p1 = random.randint(0, self.n - 1)  p2 = random.randint(self.n - 1, self.n)  g1 = lf2.gene[p1:p2] + lf1.gene  # g2 = lf1.gene[p1:p2] + lf2.gene  g11 = []  for i in g1:  if i not in g11:  g11.append(i)  return g11  return f  def m\_func(self):  """变异函数"""  def f(gene):  p1 = random.randint(0, self.n - 2)  p2 = random.randint(self.n - 2, self.n - 1)  gene[p1], gene[p2] = gene[p2], gene[p1]  return gene  return f  def save(self):  def f(lf, gen):  pass  return f  def evolve(self, evt=None):  self.\_\_lock.acquire()  self.\_\_running = True  self.\_\_lock.release()  while self.\_\_running:  self.ga.next()  self.line(self.ga.best.gene)  self.title("TSP - gen: %d" % self.ga.generation)  self.canvas.update()  self.\_\_t = None  def line(self, order):  """将节点按 order 顺序连线"""  self.canvas.delete("line")  def line2(i1, i2):  p1, p2 = self.nodes[i1], self.nodes[i2]  self.canvas.create\_line(p1, p2, fill="#000000", tags="line")  return i2  reduce(line2, order, order[-1])  def clear(self):  for item in self.canvas.find\_all():  self.canvas.delete(item)  def quite(self, evt):  self.\_\_lock.acquire()  self.\_\_running = False  self.\_\_lock.release()  sys.exit()  def stop(self, evt):  self.\_\_lock.acquire()  self.\_\_running = False  self.\_\_lock.release()  def mainloop(self):  self.root.mainloop()  if \_\_name\_\_ == "\_\_main\_\_":  MyTSP(Tkinter.Tk()).mainloop() |

GA.py

|  |
| --- |
| # -\*- coding: utf-8 -\*-  """GA.py  遗传算法类  """  import random  from Life import Life  class GA(object):  def \_\_init\_\_(self, x\_rate=0.7, mutation\_rate=0.005, life\_count=50, gene\_length=100, judge=lambda lf, av: 1,  save=lambda: 1, mk\_life=lambda: None, x\_func=None, m\_func=None):  self.x\_rate = x\_rate  self.mutation\_rate = mutation\_rate  self.mutation\_count = 0  self.generation = 0  self.lives = []  self.bounds = 0.0 # 得分总数  self.best = None  self.life\_count = life\_count  self.gene\_length = gene\_length  self.\_\_judge = judge  self.save = save  self.mk\_life = mk\_life # 默认的产生生命的函数  self.x\_func = (x\_func, self.\_\_x\_func)[x\_func == None] # 自定义交叉函数  self.m\_func = (m\_func, self.\_\_m\_func)[m\_func == None] # 自定义变异函数  for i in range(life\_count):  self.lives.append(Life(self, self.mk\_life()))  def \_\_x\_func(self, p1, p2):  # 默认交叉函数  r = random.randint(0, self.gene\_length)  gene = p1.gene[0:r] + p2.gene[r:]  return gene  def \_\_m\_func(self, gene):  # 默认突变函数  r = random.randint(0, self.gene\_length - 1)  gene = gene[:r] + ("0", "1")[gene[r:r] == "1"] + gene[r + 1:]  return gene  def \_\_bear(self, p1, p2):  # 根据父母 p1, p2 生成一个后代  r = random.random()  if r < self.x\_rate:  # 交叉  gene = self.x\_func(p1, p2)  else:  gene = p1.gene  r = random.random()  if r < self.mutation\_rate:  # 突变  gene = self.m\_func(gene)  self.mutation\_count += 1  return Life(self, gene)  def \_\_get\_one(self):  # 根据得分情况，随机取得一个个体，机率正比于个体的score属性  r = random.uniform(0, self.bounds)  for lf in self.lives:  r -= lf.score;  if r <= 0:  return lf  def \_\_new\_child(self):  # 产生新的后代  return self.\_\_bear(self.\_\_get\_one(), self.\_\_get\_one())  def judge(self, f=lambda lf, av: 1):  # 根据传入的方法 f ，计算每个个体的得分  last\_avg = self.bounds / float(self.life\_count)  self.bounds = 0.0  self.best = Life(self)  self.best.set\_score(-1.0)  for lf in self.lives:  lf.score = f(lf, last\_avg)  if lf.score > self.best.score:  self.best = lf  self.bounds += lf.score  def next(self, n=1):  # 演化至下n代  while n > 0:  # self.\_\_getBounds()  self.judge(self.\_\_judge)  new\_lives = [Life(self, self.best.gene)]  # self.bestHistory.append(self.best)  while len(new\_lives) < self.life\_count:  new\_lives.append(self.\_\_new\_child())  self.lives = new\_lives  self.generation += 1  # print("gen: %d, mutation: %d, best: %f" % (self.generation, self.mutationCount, self.best.score))  self.save(self.best, self.generation)  n -= 1 |

Life.py

|  |
| --- |
| # -\*- coding: utf-8 -\*-  """Life.py  生命类  """  import random  class Life(object):  def \_\_init\_\_(self, env, gene=None):  self.env = env  self.score = 0  if gene is None:  self.\_\_rnd\_gene()  elif isinstance(gene, list):  self.gene = []  for k in gene:  self.gene.append(k)  else:  self.gene = gene  def \_\_rnd\_gene(self):  self.gene = ""  for i in range(self.env.gene\_length):  self.gene += str(random.randint(0, 1))  def set\_score(self, v):  self.score = v  def add\_score(self, v):  self.score += v |

结果图：

