2014 级毕业设计 (论文)

基于优化算法的航班选择系统的设计与开发

年　　级: 2014 级

学　　号: 2014271055

姓　　名: 李恩宾

专　　业: 电子信息科学与技术

指导教师: 李昊璇

2018 年 4 月

基于优化算法的航班选择系统设计与开发

学生姓名: 李恩宾 指导老师: 李昊璇

内容摘要 本文致力于使用随机优化技术来解决航班选择问题。曾为团体安排过旅游计划的人都知道，计划的制定要求有许多不同的输入，比如：每个人的航班时间表是什么，要租用多少辆汽车，哪个机场是最通畅的等等。许多的输出结果也必须考虑，比如：总的成本、候机时间、起飞时间。因为我们无法将这些输入用一个简单的公式映射到输出，所以要找到最优解，就必须借助于优化算法。

智能优化算法是可以较好地解决受多种变量的影响，存在许多可能解的问题的算法。对于结果因不同变量的组合而产生很大变化的问题，智能优化算法也可以很好解决。优化算法通常是通过尝试不同的解并给这些解打分以确定其质量的方式来找到一个问题的最优解。本文主要使用了两种不同优化算法：退火算法与遗传算法。通过对比不同算法的结果，选出最优算法方案。

关键词：遗传算法 退火算法 航班选择

Design and Development of Flight Selection System Based on Optimization Algorithm

Abstract This article focuses on the use of stochastic optimization techniques to solve the flight selection problem. Anyone who has organized travel plans for the group knows that there are many different inputs for the development of the plan, such as: what is the schedule of flight for each person, how many cars to hire, which airport is the most smooth, and so on. Many output results must also be considered, such as: total cost, waiting time, and departure time. Because we cannot map these inputs to the output with a simple formula, to find the optimal solution, we must resort to optimization algorithms.

The stochastic optimization technique is a good solution to the impact of multiple variables, there are many possible solutions to the problem, and the results of the combination of these variables have a great change. Optimization algorithms usually find the optimal solution to a problem by trying different solutions and scoring these solutions to determine their quality. This paper mainly uses two different optimization algorithms: annealing algorithm and genetic algorithm. By comparing the results of different algorithms, the optimal algorithm scheme is selected.

Key Words: Genetic Algorithm Annealing Algorithm Flight Selection

目录

1 绪论

1.1 研究背景

随着时代的发展，航空运输逐渐在生产生活中占据重要地位，跨国客运与货运更是离不开航空运输的便利。但是航班数量的不断增加，使得航空管理系统已难以承受，机场容量将成为未来航空运输业发展的瓶颈。

根据民航局发布的报告 2007-2015 年末，我国民用航空航线条数从 1,506 条增至 3,326 条，民用航空通航机场数从 148 个增至 210 个，运输飞机在册架数从 1,134 架增至 2,650 架，运输总周转量从 365.30 亿吨公里增长到 851.65 亿吨公里[1]。但仅 2017 年 5 月民航局运输司、民航局消费者事务中心和中国航空运输协会共受理消费者投诉 2044 件[2]。航班延误已经成为航空运输中的重要问题。

图 1 目前大多数网站的购票方案

目前的各类旅游网站与购票网站只提供单一的购买选择而不能很好的为消费者提供最优的航班选择。为了综合航班的各项数据，给出最优的建议本文将使用智能优化算法解决航班选择问题。

1.2 优化算法的发展

每个人在生产生活中都会遇到各式各样的问题，而如何较好的找到解决方案，更有效率地解决问题就需要合适的优化算法来实现。交通出行的成本升高，人们就需要寻找到一种最优化出行方案的算法来降低出行成本。这就像每个人和每个企业都会考虑的在一定成本上如何让利润最大化，或者如何尽可能的缩减成本而提高利润一样。优化算法是这样一类数学方法的总称，它是研究在一定的约束条件下如何根据某些因素[3]，或成本函数来达到使某个活某一些指标最优的方法。

随着现代生活科技的提高，遇到的各种问题也越来越需要优化算法的帮助，这体现了优化算法在当今时代的重要性，一般来说优化算法分为启发式与非启发式。非启发式算法常用数学方法得到最优解，主要有牛顿迭代法、拉格朗日乘子法、梯度下降法、共轭梯度法等。上述方法主要用来处理数学问题中的最优化，但实际中的问题通常很难用单一数学公式描述因此启发式优化算法的应用也非常广泛，解决了很多困难问题。常用的启发式优化算法有蚁群算法、遗传算法、退火算法、贪婪算法等。

1.3 论文的结构和安排

为了解决 1.1 中的航班选择问题，本文尝试利用优化算法求解旅客出行时应该选择哪个航班可以尽可能的优化出行路线的问题。优化算法的优劣决定旅客的等待时间与旅行时的花费，合适的优化算法能够最优化出行路线，最小化出行成本[4]。

在第 2 章中我们介绍了航班选择问题的由来与具体问题的转化，并且确定了比较不同方案之间优劣的成本函数，这对评价优化算法的效果是重要的。在第 4 章中我们具体使用了四种不同方式解决航班选择问题，其中包括随机搜索、爬山算法、模拟退火算法和遗传算法。最后在第 5 章中我们将对比讨论各个算法的优劣，并找到最优算法。

2 相关工作的说明

2.1 Python 语言介绍

本文之后实现算法将使用 Python 语言实现，Python 语言是由 Guido van Rossum 发明的一种动态程序语言[5]。一般 Python 程序的语法与结构如图 2所示，从中可以看出 Python 具有良好的易读性和简洁的语法。

```
175     # Build the initial population
176     pop = []
177     for i in range(popsize):
178         vec = [random.randint(domain[i][0], domain[i][1])
179                for i in range(len(domain))]
180         pop.append(vec)
181
182     # How many winners from each generation?
183     topelite = int(elite * popsize)
```

图 2 Python 程序

与常见的动态类型语言一样，Python 具有动态类型与垃圾回收机制，这两者保证了程序编写的方便性，使得人们能够更加专注与算法本身的实现而不被其他细节末枝干扰。并且 Python 具有庞大的标准库提供各式各样的封装函数，保证了程序的简洁性。相比于 C++、Java 或 C# 之类的语言，Python 语言编写的程序可读性高，代码简洁。对于之后要实现的智能算法来说，Python 语言可以带来快速开发、测试和修改等良好的特性。

2.2 团体旅游问题简介

为了能够具体说明航班选择问题的细节，本文所用示例是为来自不同地区又去往同一地点旅游的人安排购票方案，既选择不同的航班。在本例中有关乘客和他们来自何地的具体细节如表 1所示，而他们的共同目的地则是 LGA (位于纽约的机场)。从表格中来看，一共 6 人的团体旅游项目往返机票需要 12 张，我们需要做的是如何安排好这十二张机票的购买方案。并且由于是一次团体旅游活动，所以所

表 1 需要安排的成员信息

姓名	Seymour	Franny	Zooey	Walt	Buudy	Les
来自	BOS	DAL	CAK	MIA	ORD	OMA

有需要安排的成员都需要在同一天到达又要在同一天离开，并且去机场时搭乘同样的交通工具。通过上面的了解我们知道，为了使旅行过程最优化我们需要尽可能的为每一位成员安排恰当的航班以达到出行的最优化配置，这就是航班选择的具体内容。

2.3 数据的获取与处理

我们从相关的航班网站中下载对应的航班数据并保存在文本文件中，之后算法会调用其中的数据来确定机票的各项信息。其中的数据包含以逗号分隔的不同航班的出发地点、目的地、起飞时间、到达时间、机票费用的信息。其中一项如下所示：

$$LGA, OMA, 6:20, 12:22, 942$$

它表达的含义是本次航班从 LGA 飞往 OMA 并且在早上 $06:20$ 时起飞，并在下午 $12:22$ 到达，这张机票售价 942\$, 在航班网站的订票界面显示如图 3。

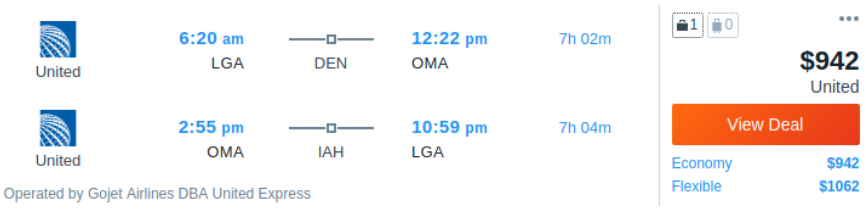

图 3 网站显示数据

为了能在 Python 程序中使用这些数据，我们将把这些数据加载为一个字典，字典是 Python 中的一种可变容器的模型，可存储任意类型对象。字典中包括键值与数据的对应。在此字典中以航班的起止点为键，详细信息为值。具体结构将如下所示：字典将在程序中被调用，查询时只需要知道键值就可以

$$\begin{aligned} Dic = \quad & [\\ & \ldots \\ & (LGA, OMA) = (6:20, 12:22, 942), \\ & \ldots \\ &] \end{aligned}$$

得到对应的航班信息，如 $Dic[(LGA, OMA)]$ 将会给出 $(6:20, 12:22, 942)$，这在算法实现中是非常方便的。

2.4 问题的程序实现

为了简化航班选择在程序中的描述，我们把每个人所在地点一天中飞往纽约的航班按照时间顺序排列——0 代表了一天中的第一次航班，1 是第二次，以此类推。所以所有人所乘坐航班的选择可以描述为这样一个数组 $[3, 4, 1, 1, 4, 6]$。该数组的排列与表 1相同，第一项的 3 表示 Seymour 将搭乘 BOS 的第三次航班飞往纽约。再加上返回时的航班选择最终结果将会是下面这样一个数组：

$$S = [2, 5, 2, 4, 6, 5, 1, 3, 3, 2, 1, 3]$$

上述数组所对应航班列表如表 2所示。

如果我们要对出行进行优化配置，那么最简单的优化方案既是让所有航班机票的价格最低即可，但是从表 2中可以看出，即使在同一天返回，时间的差别也是巨大的。即使忽略价格问题，由于团体旅行是成员一起往返于机场就会导致如果有成员的机票时间太早，而有些成员的时间又太晚造成时间上的浪费。为了解决这个问题，找到各个成本之间的平衡点，我们需要一种方法来为各个安排方案进行评估。

表 2 具体安排的情况

成员	出发地	出发时间	价格	返回时间	价格
Seymour	BOS	9:45-11:50	$172	13:39-15:30	$ 74
Franny	DAL	13:54-18:02	$294	9:49-13:51	$229
Zooey	CAK	9:15-12:14	$247	12:01-13:41	$267
Walt	MIA	12:05-15:30	$330	15:23-18:49	$150
Buddy	ORD	15:58-18:40	$173	14:19-17:09	$190
Les	OMA	13:37-15:08	$250	8:04-10:59	$136

2.5 成本函数的确定

成本函数是优化算法的核心[6]，在航班选择问题中成本函数的目标就是衡量一个选择方案的好坏程度。本文所讨论的优化算法的目的就是寻找到一组解使得成本函数的返回值最低。

在复杂问题中确定成本函数，根据多个不同变量确认选择方案的优劣程度是比较困难的。因此在航班选择问题中，我们主要讨论如下几个影响因素：

价格因素 机票所价格占据出行成本的大部分，因此在成本函数中占据较大比重。

旅行时间 每个人乘坐飞机所消耗的时间。

等待时间 如果有成员的机票较早，那么其他成员将花费的等待成本。

出发成本 如果出发时间较早，也会增加成员的出发成本，因为这需要要求成员尽早起床。

车辆租用 如果集体租用同一辆汽车，那么我们需要在同一天早于租用时间返还车辆，否则将会多付一天租用费。

确定上述影响因素后，我们需要将所有因素结合为一个成本函数并且确定每个因素的重要性。根据不同因素的影响效果，成本函数将返回所有影响因素组合在一起的一个值。在航班选择问题中我们将在飞机上的时间成本设定为每分钟等价为一美元的成本，在机场等待的时间成本为每分钟等价 0.5 美元，这样我们就把时间成本与价格成本结合在了一起。

结合机票的价格成本，飞行的时间成本，成员在机场等待的成本还有汽车租用的成本，最终我们可以得到成本函数的具体算法过程如下算法 1所示。

算法 1 成本函数

输入: 航班选择序列 S
输出: 序列 S 的出行成本 C

```
function EvaluateCost(S)
    C ← CostOfTicket(S)          ▷ 机票价格成本
    C ← C + CostOfTravel(S)      ▷ 飞行时间成本
    C ← C + CostOfWait(S)        ▷ 等待时间成本
    C ← C + CostOfEarly(S)       ▷ 出发时间成本
    C ← C + CostOfRent(S)        ▷ 汽车租用成本
    return C
end function
```

其中在算法 1中的函数 CostOfWait(S) 中，我们需要针对特殊情况做具体的处理。既我们允许任何需要等待两个小时或以上的人可以独自离开，具体算法实现如算法 2所示。

成本函数确定后，我们需要找到方法使得成本函数的输出最小[7]，既使得航班选择的最终成本最低。在航班选择问题中，出发与返回共有 12 个航班，每个航班又大约有 10 种可能。因此尝试每种可能并找出最优解是不太现实的，我们需要具体的优化算法来确定哪种方案才能使得成本函数取得最小值。

算法 2 等待时间成本

输入: 航班选择序列 S
输出: 序列 S 的等待时间成本 C

```
function CostOfWait(S)
    S' ← SortByLeftTime(S[5 : 11])          ▷ 将返回时的序列按照时间排序
    for i = 5 → 11 do
        C ← C + 0.5 × i × CostOfTime(S'[i])          ▷ 时间成本
        if (S'[i + 1].LeftTime − S'[i].LeftTime) > 120 then          ▷ 时间超过 2 小时
            i ← i + 1          ▷ 跳过下一个成员
        end if
    end for
    return C
end function
```

之后本文将会使用 Python 语言建立不同的优化算法，并且针对每种算法做具体的测试与对比，并最终选择效果最好的方案。

3 可行性分析

3.1 技术可行性

Python 语言致力于让程序变得简洁明了，让开发变得方便快捷[8]，完全可以胜任航班选择优化算法的实现。从图 3中网站所给的航班数据中可以看出机票是成对出现的，也就是说每两个地点间都有往返机票，保证了不会出现无解的情况使算法出现问题。

3.2 操作可行性

算法程序由 Python 开发，而 Python 程序语言是一个跨平台语言。只要安装了对应的运行环境无论运行 Windows、linux 还是 MacOS 的电脑都可以无障碍使用，如图 4就是在 linux 系统上运行 Python 环境。

```
> ~/ python
Python 3.6.5 (default, Apr 14 2018, 13:17:30)
[GCC 7.3.1 20180406] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

图 4 Python 环境

3.3 经济可行性

开发算法所用到的程序语言开发工具完全免费，并可以在官网下载。机票订购网站的数据也可以自由访问，而无需付费，完全具有经济可行性。

4 问题求解

4.1 本章结构

为了选出最优算法，在本章中将会实现四种不同优化算法，通过运行不同优化算法处理已有的航班数据得到结果。然后成本函数对比各个算法所得出的结果的优劣，选出效果最好的优化算法，其具体流程如图 5所示。

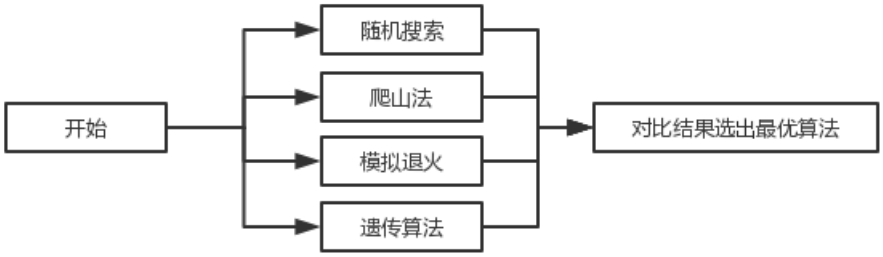

图 5 本章主要流程

4.2 随机搜索

随机搜索的本质是随机猜测可能的安排方案并且通过与之前方案的对比，通过成本函数找到可行的解[9]。

加入随机搜索算法是为了与其后要讨论的其他优化算法做对比，以衡量算法的优劣，并介绍一般优化算法的调用过程与基本结构。随机搜索算法的结构如算法 3所示。

算法 3 随机搜索

输入: 可选的航班号序列 D
输出: 航班序列 S

```
function RandomSearch(D)
    Best ← 100000                                  ▷ 成本初始化为一个大数
    for i = 0 → 1000 do                            ▷ 随机搜索 1000 次
        S′ ← [Random(0, D[i]) for i in Len(D)]     ▷ 随机生成一个航班序列
        if EvaluateCost(S) < Best then             ▷ 如果比上一最好的成本更低
            Best ← EvaluateCost(S)                 ▷ 设置当前为最优
            S ← S′                                 ▷ 更新最优航班序列
        end if
    end for
    return S
end function
```

作为一个对比基准，随机搜索的结果如表 3所示，其中的序列既为 2.4 中的航班序列。

表 3 随机搜索算法得出的结果

成员	出发地	出发时间	价格	返回时间	价格
Seymour	BOS	18:34-19:36	$136	8:23-10:28	$149
Franny	DAL	7:53-11:37	$433	19:57-23:15	$512
Zooey	CAK	20:30-23:11	$114	18:17-21:04	$259
Walt	MIA	18:23-21:35	$134	12:37-15:05	$170
Buddy	ORD	12:44-14:17	$134	14:19-17:09	$190
Les	OMA	13:37-15:08	$250	16:35-18:56	$144
序列		[8, 1, 9, 8, 4, 5, 7, 1, 8, 6, 1, 6]			
成本		6234			

4.3 爬山算法寻找最优解

从 4.2可以看出随机搜索的效率是十分地下的。在随机搜索中我们只能盲目的寻找下一个可能的选择方案而没有利用我们已经找到的较好的解。在航班选择问题中一个较好的选择方案的解很可能与其他较好方案的解接近。为了能够利用好已被发现的较好的选择方案而发现更优的方案，我们使用一个能搜索临近更优解的算法。

4.3.1 算法的描述

爬山算法从一个已有的选择方案开始，在这个选择方案临近的解集中寻找更好的解决方案。这类似于爬山从山坡上向下走，如图 6所示。在图中当前解处算法将根据当前解左右的成本取值找到一个更

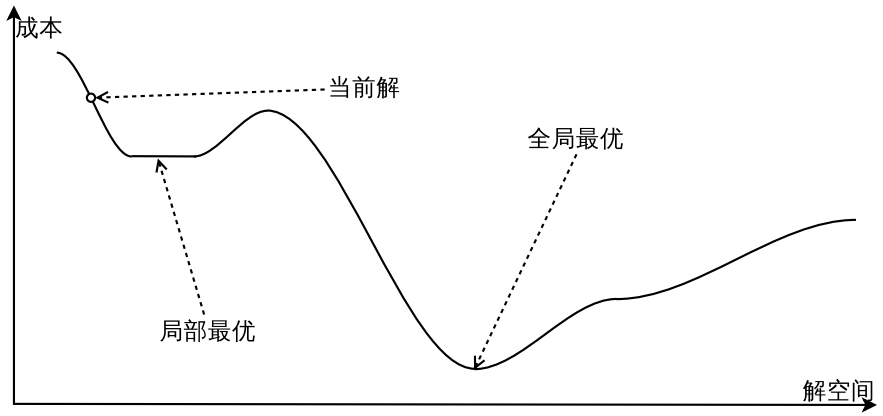

图 6 爬山法

优解，也就是当前解的右边。最终根据算法当前解会顺着斜坡一直走下去，直到没有更优解[10]。

在航班选择问题中使用爬山算法解决问题的关键是，先随机生成一个航班选择方案，然后再寻找与之相临的方案。在相邻的方案中寻找更优方案。

在本例中我们将与当前方案相比可以让某人乘坐的航班稍早或稍晚的方案作为相邻解。我们对每个搜索到的方案进行成本计算，并且安排在相邻解集中的最优解作为下一次的初始解，具体算法如算法 4所示。

最终爬山法所获得的结果如表 4所示，可以看出使用了更加可行的算法后，爬山法与随机搜索相比有了很大进步。

表 4 爬山算法得出的结果

成员	出发地	出发时间	价格	返回时间	价格
Seymour	BOS	18:34-19:36	$136	10:33-12:03	$ 74
Franny	DAL	10:30-14:57	$290	18:44-22:42	$351
Zooey	CAK	18:35-20:28	$204	10:32-13:16	$139
Walt	MIA	11:28-14:40	$248	18:07-21:30	$355
Buddy	ORD	18:48-21:45	$246	14:19-17:09	$190
Les	OMA	13:37-15:08	$250	11:07-13:24	$171
序列	[8, 3, 8, 3, 8, 5, 3, 4, 8, 3, 4, 6]				
成本	4079				

算法 4 爬山算法

输入: 可选的航班号序列 D
输出: 航班序列 S

```
function HillClimbSearch(D)
    S ← [Random(0, D[i]) for i in Len(D)]                ▷ 随机生成一个航班序列
    while True do
        Neighbors ← [ ]                                  ▷ 相邻解的列表
        for i = 0 → Len(D) do                            ▷ 生成临近解集
            if S[i] > D[i][0] then                       ▷ 如果某人航班号大于起始航班号
                S[i] ← S[i] − 1                          ▷ 选择到更小的航班号
                Neighbors.Append(S)                      ▷ 加入到相邻解集中
            else                                         ▷ 是起始航班号
                S[i] ← S[i] + 1                          ▷ 选择到更大的航班号
                Neighbors.Append(S)                      ▷ 加入到相邻解集中
            end if
        end for
        Best ← EvaluateCost(S)                           ▷ 保存上次最优解
        for S′ in Neighbors do
            if EvaluateCost(S′) < EvaluateCost(S) then   ▷ S' 为更优解
                S ← S′                                   ▷ 把更优解赋值给 S
            end if
        end for
        if EvaluateCost(S) = Best then                   ▷ 如果没有更好的解
            Break                                        ▷ 跳出循环
        end if
    end while
    return S
end function
```

4.3.2 局部最优解

爬山算法的效果一般比随机搜索的效果要好，但爬山算法有个重要缺陷[11]。从图 7可以看出，如果爬山算法如果找不到比当前最优解更好的解，它就会陷入局部的最优解而无法跳出。

一般来说对于在 2.5中建立的适应度函数，我们将其记为 $f(s)$，其中 s 代表了不同的方案，S 则是所有方案的集合。那么对于 $f(s)$ 的全局最优解与局部最优解为

$$f(s_0) = min\{f(s)\} \qquad s_0, s \in S$$
$$f(d_0) = min\{f(d)\} \qquad d_0, d \in D \quad D \subset S$$

在图 7中如果爬山算法的初始解生成在虚线左边，如 A 点和 B 点，那么根据算法中的永远只找更好的解的规则，对于 A 点, 解会逐渐向右前进。对于 B 点则会向左前进，最终在爬山算法的运行逻辑下最终都会陷入 C 点而不能到达全局最优。

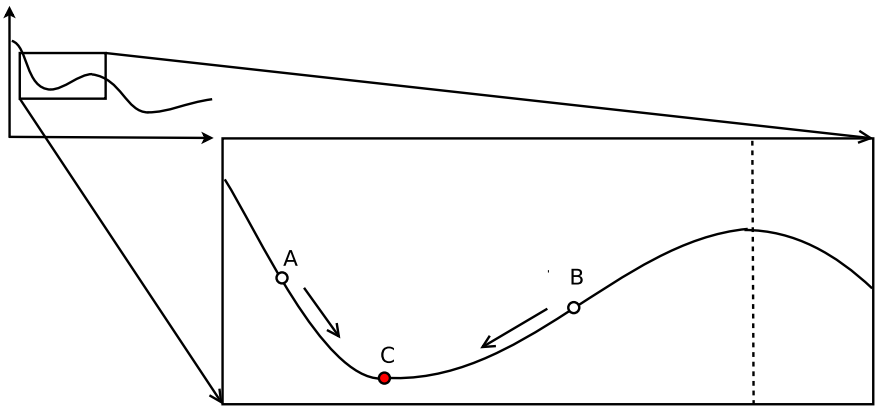

图 7 局部最优解

而局部的最优解不一定是全局的最优解，为了解决爬山算法容易陷入局部最优的问题。我们需要更加智能，能够跳出局部最优解找到全局最优解的优化算法。

4.4 模拟退火算法

模拟退火算法是用来给定近似全局最优解的启发式算法之一，它可以解决上述局部最优解的问题。这个名称的灵感来自于冶金学上的退火，一种涉及材料加热和控制冷却以增加晶体尺寸并减少其缺陷的技术。

4.4.1 模型的建立

模拟退火同样从一个随机解出发，在每次迭代过程中，算法会随机选择解中的某个元素进行改变。如果得到的结果更好，即成本函数更低，这个最优解将会成为下一次迭代的开始[12]。与上述爬山法不同的是，即使某个解可能看起来更差，模拟退火算法仍有可能将它当作下一次迭代的开始。这是因为算法引入了温度参数 T 来控制接受较差解的程度。

算法的迭代过程如图 8所示，随着温度 T 的降低，算法也越来越接近全局最优解。在模拟退火算法最开始的阶段算法会更容易接受那些比较差的解，随着温度 T 的降低算法将会变得越来越不能接受更差的解，直到算法的最后阶段，模拟退火将要完成时，算法只会接受更优的解[13]。在算法运行过程中，更高成本的解能被接受的程度与温度的关系由公式 1所示[14]，其中 h 代表了更高成本解的成本，l 代表了原本解的成本。

$$p = e^{-\frac{h-l}{T}} \tag{1}$$

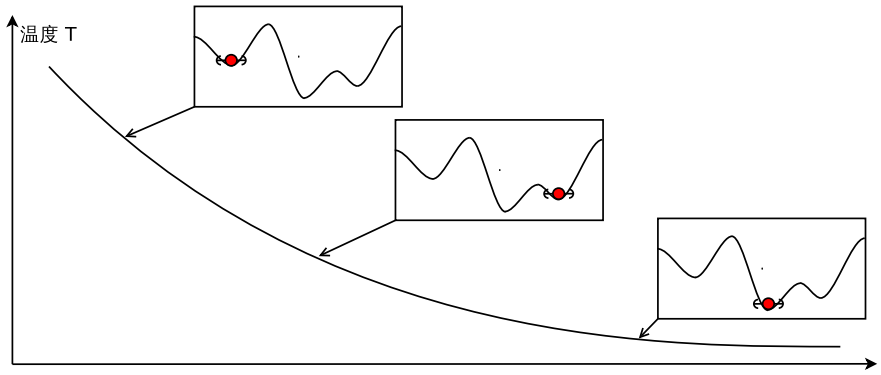

图 8 模拟退火算法

从公式 1可以看出，在最开始时模拟退火算法的温度非常高，因此 p 的值将会非常接近 1。即代表此时系统几乎完全接受成本最高的解，随着温度 T 的下降，p 的值将会越来越接近 0。综上所述，具体程序如算法 5所示。

算法 5 模拟退火算法

输入: 可选的航班号序列 D，初始温度 T，降温比率 C
输出: 航班序列 S

```
function AnnealingSearch(D, T = 10000, C = 0.95)
    S ← [Random(0, D[i]) for i in Len(D)]          ▷ 随机生成一个航班序列
    while T > 0.1 do                                ▷ 在温度降低之前一直循环
        l ← Len(S)                                  ▷ 获得序列的长度
        i ← Random(0, l)                            ▷ 随机选择一项进行改变
        s ← Random(−1, 1)                           ▷ 随机选择改变方向
        S′ ← S                                      ▷ 生成下一解
        S′[i] ← S′[i] + s                           ▷ 应用更改，随机变化
        ea ← EvaluateCost(S)                        ▷ 先前解的成本
        eb ← EvaluateCost(S′)                       ▷ 下一解的成本
        if ea < eb or Random(0, 1) < e^(−(eb−ea)/T) then   ▷ 模拟退火概率
            S ← S′                                  ▷ 赋值给下一解
        end if
        T ← T × C                                   ▷ 降温
    end while
    return S
end function
```

最终通过模拟退火算法得到的结果如表 5所示，可以看到模拟退火的结果相比随机搜索与爬山算法都要优秀。同时，对比三个算法的时间复杂度可以发现，退火算法的执行时间明显减少。

表 5 模拟退火算法得出的结果

成员	出发地	出发时间	价格	返回时间	价格
Seymour	BOS	17:11-18:30	$108	8:23-10:28	$149
Franny	DAL	7:53-11:37	$433	6:09- 9:49	$414
Zooey	CAK	6:08- 8:06	$224	13:37-15:33	$142
Walt	MIA	14:01-17:24	$338	15:23-18:49	$150
Buddy	ORD	15:58-18:40	$173	10:33-13:11	$132
Les	OMA	11:08-13:07	$175	15:07-17:21	$129
序列	[7, 1, 0, 5, 6, 3, 6, 1, 6, 1, 6, 1]				
成本	3679				

4.5 遗传算法

遗传算法是模仿生物进化过程的一种算法，它以自然选择和遗传学中的复制、变异和交叉等自然规律为理论依据。利用遗传算法求解航班选择问题，首先需要算法随机生成初始种群，即一组初始的可行解，再重复对种群进行选择、交叉、变异等遗传操作直到种群成熟[15]。然后根据成本函数对整个种群的适应度即成本进行排序，成本越低，适应度越高。

4.5.1 变异与交叉算子

为了能更好的在航班选择问题中使用遗传算法，我们将个体的适应度，也就是成本加入个体的基因中，构成的种群个体结构如图 9所示。

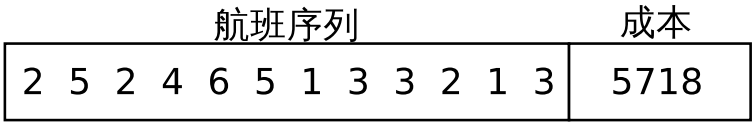

图 9 种群中的个体

种群中被选出的一组新的最优解是由之前种群按照适应度排序后稍做修改得来的[16]，其中突变算子是指让种群中某个个体的某些基因突变的算子，其在种群迭代中其过程如图 10所示，变异不一定是好的，通常情况下变异会使基因的适应度减小，即成本增大，但变异给算法提供了一种搜索全局最优解的可能[17]。

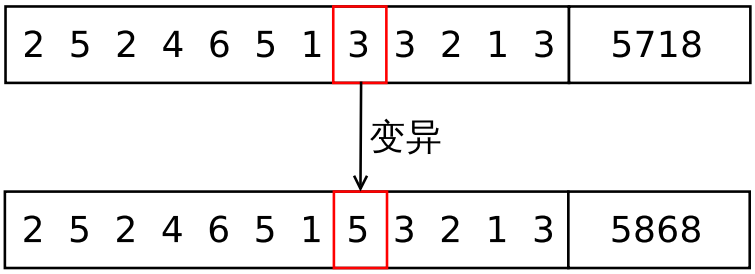

图 10 突变算子

同样的对于算法中的交叉算子[18]，它是使有效基因遗传下去的主要算子，在交叉过程中算法随机选取一个交叉点并在交叉点随机选取一个不定长度的序列。于此同时在另一个个体中也选取同样长度的一条序列，双方个体交换这个序列即为交叉过程，如图 11所示。

上述两种算子是遗传算法中常用的用来寻找全局最优解，或者跳出局部最优解的搜索方案[19]。在航班选择问题中使用变异和交叉算子将会使算法能够更好地搜索到最终的优化方案。从图 11来看当一个适应度较低的个体与适应度较高的个体交叉时，适应度较高个体的优秀基因成功降低了原个体的成本，既提高了个体的适应度。

4.5.2 算法的执行过程

正如 4.5.1中描述的，遗传算法通过变异与交叉算法从已有的种群中构造一个新的种群，之后又以新的种群开始构造下一个种群。经过足够长的时间的迭代，遗传算法所得到的解将会越来越优越，具体实现如算法 6所示。

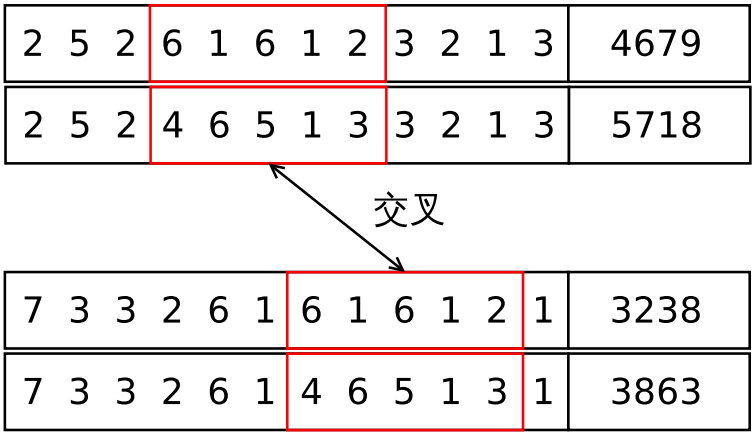

图 11 交叉算子

算法 6 遗传算法

输入: 可选的航班号序列 D，种群数量 N，变异比率 M，交叉比率 E
输出: 航班序列 S

```
function GeneticSearch(D, N = 500, M = 0.2, E = 0.2)
    Pop ← [ ]
    for i in Range(N) do                                  ▷ 生成初始种群
        S ← [Random(0, D[i]) for i in Len(D)]             ▷ 随机生成一个航班序列
        Pop.Append([S, EvaluateCost(S)])                  ▷ 加入种群
    end for
    for i in Range(100) do                                ▷ 循环 100 代
        Elite = Sort(Pop, Pop[][−1])[0 : 0.2 × N]         ▷ 选出适应度最好的 20%
        while Len(Pop) < N do                             ▷ 当种群未满时
            if Random(0, 1) < M then
                Pop.Append(Muate(Elite))                  ▷ 添加变异个体
            end if
            if Random(0, 1) < E then
                Pop.Append(Cross(Elite))                  ▷ 添加交叉个体
            end if
        end while
    end for
    S ← Elite[0][0 : −1]                                  ▷ 返回种群中最好的个体
    return S
end function
```

从表 6来看，最终得到的结果相比于上面提到过的所有算法都要好，对比模拟退火算法的结果可以看出即使两种算法都能跳出局部最优解，但遗传算法的全局最优解搜索能力要好于模拟退火算法。

表 6 遗传算法得出的结果

成员	出发地	出发时间	价格	返回时间	价格
Seymour	BOS	12:34-15:02	$109	8:23-10:28	$149
Franny	DAL	7:53-11:37	$433	10:51-14:16	$256
Zooey	CAK	10:53-13:36	$189	9:58-12:56	$249
Walt	MIA	9:15-12:29	$225	12:37-15:05	$170
Buddy	ORD	12:44-14:17	$134	7:50-10:08	$164
Les	OMA	7:39-10:24	$219	11:07-13:24	$171
序列		[4, 1, 3, 2, 4, 1, 3, 1, 4, 1, 4, 1]			
成本		2404			

5 总结与展望

5.1 结果分析

根据我们在第 4 章中算法的使用与求解，可以看出随机搜索作为对比的基准首先介绍，但随机搜索效果太差，基本上很难给出令人满意的结果; 作为对随机搜索的优化爬山法提供了一个更合理的思路[20]，即根据当前解来找到下一个解，只接受更优解，但爬山法的方式决定了它的固有缺陷，即难以跳出局部最优解。

而为了解决爬山法容易陷入局部最优解而无法跳出的问题我们引入了模拟退火算法，它由物理模型引入在算法的初期可以有效的跳出局部最优解，找到全局最优解; 作为模拟退火的对比，遗传算法通过模拟生物群落进化的方式交换优秀基因，具有更高的全局搜索性，能够更好的找到选择方案。

表 7 四种算法对比结果

算法	方案成本
随机搜索	6234
爬山法	4079
模拟退火算法	3679
遗传算法	2404

综上所述，对比四种优化算法实验中得到结果总结如表 7所示，从中可以看出遗传算法所得到的方案成本最低，即遗传算法最适合于航班选择问题。

5.2 解决问题意义

在生活节奏不断加快的现代社会，无论任何问题如何快速而又正确地找到解决方案是本文所要讨论的。航班选择是其中具有代表性的一个问题，通过文中优化算法的使用，我们最终优化了出行成本，相比于随机作出的决定优化算法所给出的方案成本降低了 159%，这大大降低了出行所需的成本节约了时间与金钱，提高了效率。

5.3 进一步工作与展望

为了能让更多人了解优化算法的作用，降低人们的生活成本，我们接下来的工作可以将优化算法与航空公司的订单网站结合起来，为出行的人们提供在线的实时的服务。当人们在网站上预订机票时，后台系统将会自动给出优化结果，给出建议。实现网络上的订单优化，这将节约大量社会成本。

6 参考文献

[1] 肖芸. 航空业发展对中国旅游业的影响研究 [J]. 经济研究导刊, 2012(20): 43–45.

[2] 黄小荣. 航班收益分析与最佳航班安排 [J]. 中国民航大学学报, 2001, 19(6): 19–22.

[3] 戴书文. 组合优化中启发式算法的研究分析 [J]. 淮南职业技术学院学报, 2005, 5(1): 72–74.

[4] SEGARAN T. Programming Collective Intelligence: Building Smart Web 2.0 Applications[M]. 2007.

[5] ROSSUM G V, DRAKE F L. Python 3 Reference Manual[J]. Department of Computer Science [CS], 1995, 111(254): 1–52.

[6] 王凌. 智能优化算法及其应用 [M]. [S.l.]: 施普林格出版社, 2001.

[7] 陈冬芳, 薛继伟, 张漫. 全局最优化算法及其应用 [J]. 东北石油大学学报, 2005, 29(1): 89–93.

[8] OLIPHANT T E. Python for Scientific Computing[J]. Computing in Science & Engineering, 2007, 9(3): 10–20.

[9] 贺红, 马绍汉. 随机算法的一般性原理 [J]. 计算机科学, 2002, 29(1): 90–92.

[10] HUHN P. Interior Point Methods for Linear Optimization[J]. Mathematical Methods of Operations Research, 2007, 65(1): 193–194.

[11] XIAO W, DUNFORD W G. A modified adaptive hill climbing MPPT method for photovoltaic power systems[C] // Power Electronics Specialists Conference, 2004. Pesc 04. 2004 IEEE. 2004: 1957–1963 Vol.3.

[12] 王雪梅, 王义和. 模拟退火算法与遗传算法的结合 [J]. 计算机学报, 1997(4): 381–384.

[13] 高尚. 模拟退火算法中的退火策略研究 [J]. 航空计算技术, 2002, 32(4): 20–22.

[14] KIRKPATRICK S, VECCHI M P. Optimization by simulated annealing[M]. [S.l.]: Morgan Kaufmann Publishers Inc., 1987: 339–348.

[15] 赵静, 但琦, OTHERS. 数学建模与数学实验 [M]. [S.l.]: 北京: 高等教育出版社, 2008.

[16] 张铃, 张拔. 遗传算法机理的研究 [J]. 软件学报, 2000, 11(7): 000945–952.

[17] GOLDBERG D E. Genetic Algorithm in Search Optimization and Machine Learning[J]. Addison Wesley, 1989, xiii(7): 2104–2116.

[18] 胡中功, 李静. 群智能算法的研究进展 [J]. 自动化技术与应用, 2008, 27(2): 13–15.

[19] 苗夺谦, 胡桂荣. 知识约简的一种启发式算法 [J]. 计算机研究与发展, 1999, 36(6): 681–684.

[20] THOMAS, H.CORMEN, CHARLES, et al. 算法导论 (原书第 3 版)[J]. 计算机教育, 2013(12): 51–51.

7 致谢

在这次毕业设计中，我学习和了解到了很多知识，首先我要感谢我的指导老师和同学们的无私帮助。其次我要感谢各位科学工作者的优秀论文，本文所用算法均由 Python 实现，所以我还要感谢 Python 语言的发明人 Guido van Rossum 先生让算法的实现变得简单易懂。最后本文在 linux 上由 LaTeX 编译而成所以我还要感谢 linux 的发明者 Linus Benedict Torvalds 先生和 LaTeX 的发明人 Leslie Lamport 先生，感谢他们的无私奉献。