中国研究生创新实践系列大赛
“华为杯”第十六届中国研究生数学建模竞赛

学　　校	杭州电子科技大学
参赛队号	**19103360018**
队员姓名	**1.周家洛**
	2.高丹蓓
	3.施江玮

中国研究生创新实践系列大赛
“华为杯”第十六届中国研究生数学建模竞赛

题　目　　智能飞行器航迹规划模型

摘　　要：

复杂环境下航迹快速规划是智能飞行器控制的一个重要课题。本文分析智能飞行器系统自身结构局限性以及飞行环境导致飞行任务失败等多种原因，建立了多目标规划模型，利用禁忌搜索算法和遗传算法求解出最优的智能飞行器航行轨迹方案。

针对问题一，在确保智能飞行器能够成功到达目的地的条件下，本文考虑了多种约束条件，建立了多约束条件下航迹长度最小化和误差校正点数最小化的**双目标优化模型**。模型求解时，首先**引入规范函数进行无量纲化处理**并通过**线性加权法**将**该模型转化为单目标优化模型**，以降低模型复杂度；然后通过约束条件将符合条件的误差校正点根据航迹长度最短原则，使用**贪心算法**初始化解，再使用**禁忌搜索算法**，优化初始解，**在 120 次迭代后得到较优解，运行时间为 70.64s，改善情况为 6.55%**。对数据集 1 求解出的最佳航行轨迹方案为：A→503→69→237→115→338→457→555→436→B，经过误差校正点数 **8** 个，航迹长度为 **104898m**，比 AB 直线距离增加了 **4.41%**；对数据集 2 求解出的最佳航行轨迹方案为：A→163→114→8→309→305→123→45→160→92→93 →61→292→B，经过误差校正点数 **12** 个，航迹长度为 **109342m**，比 AB 直线距离增加了 **6.11%**。该算法**时间复杂度**为 $O(n^2)$。本文通过深度优先搜索策略进行遍历，**验证**两组数据最优解的校正点数为 8 和 12，**该算法针对两组数据最优解的点数均为最优**；同时，两组数据航迹长度比仅比 AB 直线距离增加 **4.41%**和 **6.11%**，**验证了算法的有效性**。

针对问题二，在确保智能飞行器能够成功到达目的地的条件下，需要考虑飞行器转向轨迹。为此本文参考 **Dubins** 曲线，分析了飞行器在三维空间中从 A 点到 B 点的转向过程，设计了飞行器转向约束，根据优先级尽可能短的航迹长度>尽可能少的误差校正点，在此基础上改进问题一的双目标优化模型，建立了带多约束条件的航迹长度最小化和误差校正点数最小化的**双目标优化模型**。模型求解时，先引入**规范函数进行无量纲化处理**并通过**线性加权法**将该模型**转化为单目标优化模型**，再使用**贪心算法**求解初始化解，然后引入了**集中和分散机制**，提出并使用**改进禁忌搜索算法**优化初始解，一定程度上解决了禁忌搜索算法在求解优化问题时存在着的停滞现象，进一步提高了禁忌搜索算法的搜索质量和收敛速度，**在 120 次迭代后得到较优解，运行时间为 74.41s，改善情况为 6.79%**。对数据集 1 求解出的最佳航行轨迹方案为：A→503→69→237→233→598→561→448→485→B，经过误

差校正点数 **8** 个，航迹长度为 **104960m**，比 AB 直线距离增加了 **4.47%**，比问题一最优方案距离增加了 **0.02%**；对数据集 2 求解出的最佳航行轨迹方案为：A→163→114→8→309→305→123→45→160→92→93→61→292→B，经过误差校正点数 **12** 个，航迹长度为 **109411m**，比 AB 直线距离增加了 **6.18%**，比问题一最优方案距离增加了 **0.06%**。该算法时间复杂度低于 $O(n^2)$。本文通过深度优先搜索策略进行遍历，**验证**两组数据最优解的校正点数为 8 和 12，**该算法针对两组数据最优解的点数均为最优**；同时，两组数据航迹长度比仅比 AB 直线距离增加 **4.47%**和 **6.18%**，仅比第一问最优方案增加了 **0.02%**和 **0.06%**，**验证了算法的有效性**。

针对问题三，由于无法确保智能飞行器能够成功抵达目的地，因此需尽量降低失败概率，同时使航迹长度尽可能小，经过的校正点个数尽可能少。为此本文考虑了飞行器成功到达目的地的多种约束条件，建立了带多约束条件的失败概率最小化、航迹长度最小化和误差校正点数最小化的**多目标优化模型**。通过模型求解时，先引入**规范函数进行无量纲化处理**并通过**线性加权法将该模型转化为单目标模型**，再使用**贪心算法**求解初始化解，然后将禁忌搜索算法与遗传算法结合，提出并使用**遗传禁忌混合搜索算法**优化初始解，**在 120 次迭代后得到较优解，运行时间为 87.86s，改善情况为 6.13%**。对数据集 1 求解出的最佳航行轨迹方案为：A→578→417→80→237→607→33→194→450→448→485→302→612→B，经过误差校正点数 **12** 个，航迹长度为 **108439m**，比 AB 直线距离增加了 **7.93%**，成功抵达终点概率 **100%**；对数据集 2 求解出的最佳航行轨迹方案为：A→169→322→100→137→194→190→296→250→243→73→82→44→211→321→279→301→38→287→99→326→B，经过误差校正点数 **19** 个，航迹长度为 **147271m**，比 AB 直线距离增加了 42.91%，成功抵达终点概率 **65.01%**。该算法时间复杂度 $T(n)=O(n^4/\lambda)$（λ 为种群数量）。本文通过深度优先搜索策略进行遍历，**验证**两组数据最优解的校正点数为 12 和 19，**该算法针对两组数据最优解的点数均为最优**；同时，两组数据航迹长度比比 AB 直线距离增加 7.93%和 42.91%，**验证了算法的有效性**。

关键词：多目标规划；贪心算法；禁忌搜索算法；时间复杂度

目录

1. 问题重述

1.1 问题背景

随着科学技术特别是航空科学技术的飞速发展，智能飞行器逐渐进入了人们的生活当中，在军用和民用领域的使用上都得到了显著增加[1]。智能飞行器的飞行航线快速规划在航行任务策划中起着核心作用，其目的是确定由出发地到目的地的最优航线。由于智能飞行器系统本身的结构局限性，其自身无法利用定位系统位置进行校正，由此会产生一定的定位误差进而影响任务进程，甚至造成任务无法完成。为了使智能飞行器得到最优的利用，本文在尽量减少航行路线距离，降低能源消耗，并且考虑系统定位误差限制等诸多约束条件的前提下，例如图 1 所示的智能飞行器航行规划区域，研究了如何自动快速计算得到智能飞行器的最优航线问题。

根据上述问题，以 A 点作为出发地，B 点作为目的地，其航线规划有下列限制条件：

(1) 在飞行期间，对智能飞行器进行动态的定位。在飞行器航行单位距离（1m）的情况下，其垂直定位误差和水平定位误差分别增加一个单位δ。只有在保证飞行器到达目的地时，即时的垂直定位误差和水平定位误差都小于θ个单位的条件下，进行航线规划。

(2) 为满足限制条件 1，需对飞行器的垂直定位误差和水平定位误差进行垂直和水平的校正。在所给航行规划区域内，存在部分垂直校验点和水平校验点，可分别用作垂直定位误差和水平定位误差的清零（即校正），从而在实时校正的同时进行航线规划。

(3) 在 A 点处，初始垂直定位误差为 0，初始水平定位误差为 0。

(4) 垂直校正点只能校正垂直定位误差（垂直定位误差清零），不可校正水平定位误差（水平定位误差不变）。

(5) 水平校正点只能校正水平定位误差（水平定位误差清零），不可校正垂直定位误差（垂直定位误差不变）。

(6) 垂直校正条件：垂直定位误差小于α_1，水平定位误差小于α_2。

(7) 水平校正条件：垂直定位误差小于β_1，水平定位误差小于β_2。

(8) 飞行器转弯时所能达到的最小半径为 200m。

1.2 需要解决的问题

在本文中，我们需要根据附件 1 和附件 2 所给数据集结合上述限制条件建立一个从出发点到达目的地的飞行器航线规划模型，并按要求完成如下问题：

1．在限制条件（1）~（7）的约束下，考虑两个优化目标，建立数学模型，并根据此模型结合附件 1 和附件 2 所给的两个数据集，设计算法分别规划出各自条件下的飞行器航行路线，并对该算法进行有效性和复杂度的分析。

2．在前一问的基础上，同时考虑限制条件（8），同样考虑前一问的两个优化目标，对前一问的模型进行改进，并根据改进后的模型结合所给的两个数据集，改进算法分别规划出此问题下的飞行器航行路线，并对该改进的算法进行有效性和复杂度的分析。

3． 在问题一模型的基础上，进一步考虑环境因素对误差校正的影响，对该模型的目标函数增加概率因子，并赋予权重。根据改进后的模型结合所给的两个数据集，进一步设计算法分别规划出此问题下的飞行器航行路线，并对该算法进行有效性和复杂度的分析。

2. 问题分析

飞行器航线规划问题在很早就被提出并研究，该航线规划相关的文献也较多。大部分都是在缩短总航线的基础上，考虑了飞行质量和飞行安全等一些限制条件；在航线规划的基础上，进一步将飞行器的路线进行平滑和优化。为解决该问题，一般用到的算法有 A* 算法、Dijkstra 算法、粒子群算法、遗传算法等[2~5]。这些算法各有优劣，有些传统算法得到的结果精确，但随着模型复杂度和问题规模的增加，算法计算所需时间也更大；一些智能算法计算速度较快，适应性高，但容易陷入局部最优解。

针对上述背景下的多约束条件航线规划问题，本文从目标函数的角度入手，分析了各种情况下定位误差实时变化状态并建立了带约束的多目标优化模型；同时，根据问题的限制条件具体分析，考虑了多种约束条件，采用了禁忌搜索算法和遗传算法，针对实际应用对算法进行改进。

2.1 问题一分析

前提：根据题目所给参数有以下两种场景：针对附件 1，满足垂直校正条件下的最大垂直定位误差 $\alpha_1=25$，最大水平定位误差 $\alpha_2=25$；满足水平校正条件下的最大垂直定位误差 $\beta_1=20$，最大水平定位误差 $\beta_2=25$；到达目的地 B 点时，垂直定位误差和水平定位误差都小于 $\theta=30$；单位飞行距离下，垂直定位误差和水平定位误差所增加的单位定位误差 $\delta=0.001$。 针对附件 2，满足垂直校正条件下的最大垂直定位误差 $\alpha_1=20$，最大水平定位误差 $\alpha_2=10$；满足水平校正条件下的最大垂直定位误差 $\beta_1=15$，最大水平定位误差 $\beta_2=20$；到达目的地 B 点时，垂直定位误差和水平定位误差都小于 $\theta=20$；单位飞行距离下，垂直定位误差和水平定位误差所增加的单位定位误差 $\delta=0.001$。

条件：根据上述问题背景，按照（1）~（7）限制条件的要求对附件 1 和附件 2 所提供的数据分别进行航线规划。

目标：基于上述前提和条件，在保证最短航线距离的基础上，减少定位误差校正次数，进而建立多目标航迹规划函数。并在该模型下，按照所采用的算法，完成其有效性和复杂度的讨论。

2.2 问题二分析

前提：与问题一采用同样的参数设置，即同样也考虑以下两种场景：针对附件 1，满足垂直校正条件下的最大垂直定位误差 $\alpha_1=25$，最大水平定位误差 $\alpha_2=25$；满足水平校正条件下的最大垂直定位误差 $\beta_1=20$，最大水平定位误差 $\beta_2=25$；到达目的地 B 点时，垂直定位误差和水平定位误差都小于 $\theta=30$；单位飞行距离下，垂直定位误差和水平定位误差所增加的单位定位误差 $\delta=0.001$。 针对附件 2，满足垂直校正条件下的最大垂直定位误差 $\alpha_1=20$，最大水平定位误差 $\alpha_2=10$；满足水平校正条件下的最大垂直定位误差

$\beta_1=15$，最大水平定位误差$\beta_2=20$；到达目的地 B 点时，垂直定位误差和水平定位误差都小于$\theta=20$；单位飞行距离下，垂直定位误差和水平定位误差所增加的单位定位误差$\delta=0.001$。

条件：根据问题背景，按照（1）~（8）限制条件的要求对附件 1 和附件 2 所提供的数据分别进行航线规划。与问题一不同的是，问题二是在额外考虑限制条件（8）的前提下进行模型的建立。限制条件（8）涉及了三维飞行空间中的转弯问题，由于飞行器转弯的最小半径的限制，所建模型需要在讨论航线方位偏转角的基础上，考虑飞行航线的规划，并包括各种不同情况的转弯情形。与此同时还要满足（1）~（7）限制条件下实时定位误差校正约束。

目标：基于上述前提和条件，在保证最短航线距离的基础上，减少定位误差校正次数，进而建立多目标航迹规划函数。并在该模型下，按照所采用的算法，完成其有效性和复杂度的讨论。

2.3 问题三分析

前提：与问题一采用同样的参数设置，即同样也考虑以下两种场景：针对附件 1，满足垂直校正条件下的最大垂直定位误差$\alpha_1=25$，最大水平定位误差$\alpha_2=25$；满足水平校正条件下的最大垂直定位误差$\beta_1=20$，最大水平定位误差$\beta_2=25$；到达目的地 B 点时，垂直定位误差和水平定位误差都小于$\theta=30$；单位飞行距离下，垂直定位误差和水平定位误差所增加的单位定位误差$\delta=0.001$。 针对附件 2，满足垂直校正条件下的最大垂直定位误差$\alpha_1=20$，最大水平定位误差$\alpha_2=10$；满足水平校正条件下的最大垂直定位误差$\beta_1=15$，最大水平定位误差$\beta_2=20$；到达目的地 B 点时，垂直定位误差和水平定位误差都小于$\theta=20$；单位飞行距离下，垂直定位误差和水平定位误差所增加的单位定位误差$\delta=0.001$。

与问题一不同的是，航迹规划需要考虑飞行器飞行期间环境因素的变化对定位误差校正点校正概率的影响，在模型的建立中，额外增加校正概率的权重。其中失败校正的概率为 20%，若校正失败，则经过该校正点的剩余定位误差为$\min(error,5)$，也就是说经过该校正点后，定位误差取原误差（校正前）和 5 的较小值。

条件：根据问题背景，与问题一想相同，按照（1）~（7）限制条件的要求对附件 1 和附件 2 所提供的数据分别进行航线规划。

目标：基于上述前提和条件，在保证最短航线距离的基础上，减少定位误差校正次数，并且使得所规划的航迹到达目的地的校正成功概率最大化，由此建立多目标航迹规划函数。

3. 模型假设

1. 假设飞行器均按规划好的航线自主飞行，无须人工控制；
2. 假设飞行器不会出现故障，迫降和损坏等问题；
3. 假设忽略飞行器起飞和降落过程对飞行路程的影响；
4. 假设飞行器为质点，忽略飞行器大小。

4. 符号说明

符号	符号说明
$Distance$	航迹长度变量
$Times$	经过校正区域进行校正的次数
x_{ij}	路径决策变量
d_{ij}	i 点与 j 点的欧氏距离
$\vec{v}$	飞行器当前飞行运动矢量
$P(x,y,z)$	飞行器所在位置坐标
$Ph(x,y,z)$	距飞行器当前位置最近的水平校正点坐标
$Pv(x,y,z)$	距飞行器当前位置最近的垂直校正点坐标
N	误差校正点的数量
s_i	第 i 段飞行器正常飞行（无需校正）的路程
p	飞行器无法成功到达目的地的概率
η_i	飞行器在第 i 个校正点能否到达下一个校正点的判决变量
p_s	误差校正点校正成功的概率

5. 问题一：模型建立与求解

5.1 模型建立

5.1.1 短航迹-少校正次数双目标规划模型

优化目标：（1）飞行器从出发地 A 航行至目的地 B 的航线距离尽可能短；（2）经过校正区域进行校正的次数尽可能少。因此，定义航迹长度变量 *Distance*，即飞行器从出发地 A 航行至终点 B 所经过的总路程；定义经过校正区域进行校正的次数 *Times*。

目标函数：采用短航迹长度-少校正次数双目标优化模型，即同时优化航迹长度和校正次数双变量。目标函数可定义为：

$$\min z_1 = Distance \tag{5-1}$$

$$\min z_2 = Times \tag{5-2}$$

决策变量：（1）定义决策变量 x_{ij}，当前校正点（或起点）i 的下一个最优校正点（或终点）j 是否连通，用以确定最优路径。

$$x_{ij} = \begin{cases} 0 & i点和j点没有连接 \\ 1 & i点和j点存在连接 \end{cases} \tag{5-3}$$

同时，使用向量 $\mathbf{x} = [x_{ij}^{(1)}, x_{ij}^{(2)}, ... x_{ij}^{(k)} ...]$ 记录每一次的决策，其中 $x_{ij}^{(k)}$ 表示第 k 次选择连接的 i，j 两点，用向量 **x** 来记录最佳路径。

目标函数中的 *Distance* 和 *Times* 可以通过决策变量表示为：

$$\begin{aligned} Distance &= \sum_{i=0}^{N}\sum_{j=0}^{N} x_{ij} \times d_{ij} \\ Times &= \sum_{i=0}^{N}\sum_{j=0}^{N} x_{ij} \end{aligned} \tag{5-4}$$

其中 d_{ij} 表示 i 点与 j 点的欧式距离。

约束条件：（1）飞行器垂直校正条件：垂直定位误差 ε_v 小于 α_1，水平定位误差 ε_h 小于 α_2；（2）飞行器水平校正条件：垂直定位误差 ε_v 小于 β_1，水平定位误差 ε_h 小于 β_2；（3）飞行器到达目的地前，垂直误差和水平误差都应小于 θ。

在探讨约束条件前，定义校正器类型变量 T_i：

$$T_i = \begin{cases} 0 & 第i个校正点是水平校正点 \\ 1 & 第i个校正点是垂直校正点 \end{cases} \tag{5-5}$$

约束条件分析如下：

1）当飞行器从起点出发时，即 k=1 时，会选择一个合适的水平校正点或垂直校正点。两种情况的约束条件汇总如下表 5-1：

表 5-1 飞行器从起点选择校正点约束条件

情况	到达校正点时的误差	约束条件
选择水平校正点	$\varepsilon_h = \delta \times d_{0j} \times x_{0j}$	$\varepsilon_h \le \beta_2$
	$\varepsilon_v = \delta \times d_{0j} \times x_{0j}$	$\varepsilon_v \le \beta_1$
选择垂直校正点	$\varepsilon_h = \delta \times d_{0j} \times x_{0j}$	$\varepsilon_h \le \alpha_2$
	$\varepsilon_v = \delta \times d_{0j} \times x_{0j}$	$\varepsilon_v \le \alpha_1$

因此，选择合适的校正器类型变量 T_i，当 k=1 时的约束条件可以表述为：

$$\begin{cases} k = 1 \\ \delta \times d_{0j} \times x_{0j} \times (1 - T_i) \le \beta_1 \\ \delta \times d_{0j} \times x_{0j} \times (1 - T_i) \le \beta_2 \\ \delta \times d_{0j} \times x_{0j} \times T_i \le \alpha_1 \\ \delta \times d_{0j} \times x_{0j} \times T_i \le \alpha_2 \end{cases} \tag{5-6}$$

2）当飞行器从第 i 个校正点出发时，即 k>1 时，会选择一个合适的水平校正点或垂直校正点。两种情况的约束条件汇总如下表 5-2：

表 5-2 飞行器从校正点选择下一校正点约束条件

情况		到达校正点时的误差	约束条件
选择水平校正点	上一次经过水平校正点	$\varepsilon_h = \delta \times \left(d_{ij}^{(k)} \times x_{ij}^{(k)} \right)$	$\varepsilon_h \le \beta_2, \theta$
		$\varepsilon_v = \delta \times \left(d_{ij}^{(k)} \times x_{ij}^{(k)} + d_{ij}^{(k-1)} \times x_{ij}^{(k-1)} \right)$	$\varepsilon_v \le \beta_1, \theta$
	上一次经过垂直校正点	$\varepsilon_h = \delta \times \left(d_{ij}^{(k)} \times x_{ij}^{(k)} + d_{ij}^{(k-1)} \times x_{ij}^{(k-1)} \right)$	$\varepsilon_h \le \beta_2, \theta$
		$\varepsilon_v = \delta \times \left(d_{ij}^{(k)} \times x_{ij}^{(k)} \right)$	$\varepsilon_v \le \beta_1, \theta$
选择垂直校正点	上一次经过水平校正点	$\varepsilon_h = \delta \times \left(d_{ij}^{(k)} \times x_{ij}^{(k)} + d_{ij}^{(k-1)} \times x_{ij}^{(k-1)} \right)$	$\varepsilon_h \le \alpha_2, \theta$
		$\varepsilon_v = \delta \times \left(d_{ij}^{(k)} \times x_{ij}^{(k)} \right)$	$\varepsilon_v \le \alpha_1, \theta$
	上一次经过垂直校正点	$\varepsilon_h = \delta \times \left(d_{ij}^{(k)} \times x_{ij}^{(k)} \right)$	$\varepsilon_h \le \alpha_2, \theta$
		$\varepsilon_v = \delta \times \left(d_{ij}^{(k)} \times x_{ij}^{(k)} + d_{ij}^{(k-1)} \times x_{ij}^{(k-1)} \right)$	$\varepsilon_v \le \alpha_1, \theta$

因此，选择合适的校正器类型变量 T_i，当 k>1 时的约束条件可以表述为：

$$\begin{cases} k>1 \\ \delta\times\left(d_{ij}^{(k)}\times x_{ij}^{(k)}\times(1-T_i)+d_{ij}^{(k-1)}\times x_{ij}^{(k-1)}\times T_i\right)\le\beta_1,\theta \\ \delta\times\left(d_{ij}^{(k)}\times x_{ij}^{(k)}\times(1-T_i)+d_{ij}^{(k-1)}\times x_{ij}^{(k-1)}\times(1-T_i)\right)\le\beta_2,\theta \\ \delta\times\left(d_{ij}^{(k)}\times x_{ij}^{(k)}\times T_i+d_{ij}^{(k-1)}\times x_{ij}^{(k-1)}\times T_i\right)\le\alpha_1,\theta \\ \delta\times\left(d_{ij}^{(k)}\times x_{ij}^{(k)}\times T_i+d_{ij}^{(k-1)}\times x_{ij}^{(k-1)}\times(1-T_i)\right)\le\alpha_2,\theta \end{cases} \tag{5-7}$$

3)决策变量的约束条件。由于定义的决策变量 x_{ij} 的目的是寻找一条最优路径，即 x_{ij} 构成的邻接矩阵是一个有向无环图，且不是一种树结构。这意味着每个校正点最多只能经过一次，且必须保证从起点到终点是一条通路。

首先应当约束邻接矩阵每一列的和小于等于 1，这样可以确保每个校正点最多经过一次；

其次需要约束起点到终点是一条通路。本文使用向量 $\mathbf{x}=[x_{ij}^{(1)},x_{ij}^{(2)},...x_{ij}^{(k)}...]$ 记录每一次的决策，其中 $x_{ij}^{(k)}$ 表示第 k 次选择连接的 i，j 两点，展开为 $x_{ij}^{(k)}(i_k,j_k)$。应当使第 k 次选择的点 $x_{ij}^{(k)}(i_k,j_k)$ 纵坐标与第 k+1 次选择的点 $x_{ij}^{(k)}(i_k,j_k)$ 横坐标相等，即 $j_k=i_{k+1}$。

因此，决策变量的约束条件可以表述为：

$$\begin{cases} \sum_{j=0}^{N}x_{ij}\le 1 \\ j_k=i_{k+1} \\ i,j\in(0,1,2,...N) \end{cases} \tag{5-8}$$

综上所述，所建立的双目标优化模型可以表述为：

$$
\begin{aligned}
&min\, z_1 = Distance = \sum_{i=0}^{N}\sum_{j=0}^{N} x_{ij} \times d_{ij} \\
&\min\, z_2 = Times = \sum_{i=0}^{N}\sum_{j=0}^{N} x_{ij} \\
&s.t.\begin{cases}
\begin{cases}
k = 1 \\
\delta \times d_{0j} \times x_{0j} \times (1 - T_i) \le \beta_1 \\
\delta \times d_{0j} \times x_{0j} \times (1 - T_i) \le \beta_2 \\
\delta \times d_{0j} \times x_{0j} \times T_i \le \alpha_1 \\
\delta \times d_{0j} \times x_{0j} \times T_i \le \alpha_2
\end{cases} \\
\begin{cases}
k > 1 \\
\delta \times \left(d_{ij}^{(k)} \times x_{ij}^{(k)} \times (1 - T_i) + d_{ij}^{(k-1)} \times x_{ij}^{(k-1)} \times T_i \right) \le \beta_1, \theta \\
\delta \times \left(d_{ij}^{(k)} \times x_{ij}^{(k)} \times (1 - T_i) + d_{ij}^{(k-1)} \times x_{ij}^{(k-1)} \times (1 - T_i) \right) \le \beta_2, \theta \\
\delta \times \left(d_{ij}^{(k)} \times x_{ij}^{(k)} \times T_i + d_{ij}^{(k-1)} \times x_{ij}^{(k-1)} \times T_i \right) \le \alpha_1, \theta \\
\delta \times \left(d_{ij}^{(k)} \times x_{ij}^{(k)} \times T_i + d_{ij}^{(k-1)} \times x_{ij}^{(k-1)} \times (1 - T_i) \right) \le \alpha_2, \theta
\end{cases} \\
\begin{cases}
\sum_{j=0}^{N} x_{ij} \le 1 \\
j_k = i_{k+1} \\
i, j \in (0,1,2,\ldots N)
\end{cases} \\
\alpha_1, \alpha_2, \beta_1, \beta_2, \delta, \theta > 0 \\
d_{ij}^{(k)} = \left\| \overrightarrow{i^{(k)} j^{(k)}} \right\|
\end{cases}
\end{aligned}
\tag{5-9}
$$

5.2 模型求解

5.2.1 双目标转化为单目标

在本问题中，由最小化航迹变量 *Distance* 和最小化经过矫正区域进行校正的次数 *Times* 建立了两个目标函数，选取两个值 $0 < \lambda_i < 1 (i = 1,2)$，作为这两个目标的线性加权系数，并且 $\sum_{i=1}^{2} \lambda_i = 1$。由于本问题中两个目标函数的量纲不同，因此需要归一化目标函数，对其进行无量纲化处理。此时目标函数分别通过函数 $\alpha_{z_1} = \dfrac{z_1 - min\, z_1}{max\, z_1 - min\, z_1}$，$\alpha_{z_2} = \dfrac{z_2 - min\, z_2}{max\, z_2 - min\, z_2}$ 规范。此时，由于目标函数归一化导致整个目标函数缩小至变化细微，因此需要乘以 100，这样迭代时目标函数区分明显。该问题就转化为了单目标数学规划问题，转化后的单目标函数可以写为：

$$\min z_{z_1} = (\lambda_1 \alpha_{z_z} + \lambda_2 \alpha_{z_2}) \times 100 \tag{5-10}$$

因此，模型可以表述为：

$$\min z_{z_1} = (\lambda_1 \alpha_{z_z} + \lambda_2 \alpha_{z_2}) \times 100$$

$$s.t. \begin{cases} \begin{cases} k = 1 \\ \delta \times d_{0j} \times x_{0j} \times (1 - T_i) \le \beta_1 \\ \delta \times d_{0j} \times x_{0j} \times (1 - T_i) \le \beta_2 \\ \delta \times d_{0j} \times x_{0j} \times T_i \le \alpha_1 \\ \delta \times d_{0j} \times x_{0j} \times T_i \le \alpha_2 \end{cases} \\ \begin{cases} k > 1 \\ \delta \times \left(d_{ij}^{(k)} \times x_{ij}^{(k)} \times (1 - T_i) + d_{ij}^{(k-1)} \times x_{ij}^{(k-1)} \times T_i \right) \le \beta_1, \theta \\ \delta \times \left(d_{ij}^{(k)} \times x_{ij}^{(k)} \times (1 - T_i) + d_{ij}^{(k-1)} \times x_{ij}^{(k-1)} \times (1 - T_i) \right) \le \beta_2, \theta \\ \delta \times \left(d_{ij}^{(k)} \times x_{ij}^{(k)} \times T_i + d_{ij}^{(k-1)} \times x_{ij}^{(k-1)} \times T_i \right) \le \alpha_1, \theta \\ \delta \times \left(d_{ij}^{(k)} \times x_{ij}^{(k)} \times T_i + d_{ij}^{(k-1)} \times x_{ij}^{(k-1)} \times (1 - T_i) \right) \le \alpha_2, \theta \end{cases} \\ \begin{cases} \sum_{j=0}^{N} x_{ij} \le 1 \\ j_k = i_{k+1} \\ i, j \in (0,1,2,...N) \end{cases} \\ \alpha_1, \alpha_2, \beta_1, \beta_2, \delta, \theta > 0 \\ d_{ij}^{(k)} = \left\| \overrightarrow{i^{(k)} j^{(k)}} \right\| \end{cases} \tag{5-11}$$

接下来便转化为单目标的禁忌搜索算法求解目标函数。

5.2.2 基于单目标规划的禁忌搜索算法

禁忌搜索算法是一种亚启发式的随机搜索算法，它从一个初始可行解出发，选择一系列的特定搜索方向作为试探，选择实现让特定的目标函数值变化最多的移动[6]。为了避免陷入局部最优，禁忌搜索算法采用了一种灵活的记忆技术，其基本思想是在搜索过程中将近期的历史搜索存放在禁忌表中，防止算法重复搜索，即之前搜索过的区域不会在迂回搜索，这样便大大提高了搜索的效率，减少了程序的时间复杂度。

随着迭代的不断进行，禁忌表将不断进行更新，当迭代经过一定的次数之后，原先进入禁忌表的移动便会从禁忌表中移除。

（1）解的编码与更新

编码方式采用二进制编码，编码的每一位就是解的相应维的取值，为了能清楚说明解是如何编码与更新的，这里设 $N=6$，即总误差校正点数量为 6，如图 5-1 所示。

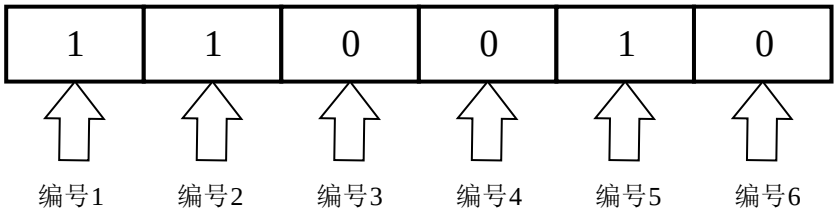

图 5-1 误差校正点编码分配问题

该解表示，飞行器需经过编号为 1，2，5 的误差校正点，而无需经过编号为 3，4，6 的误差校正点。

为了优化算法，我们需要解的移动。在这里我们考虑使用简单移动，即新解与原始解有且只有一个分量不同。为了算法的可行性，两个解多个分量的不同使用多次简单移动组合来解决。

在移动之前，首先需要考虑约束条件。我们设计了判别函数，只有满足约束条件的误差校正点才能移动。若在约束条件下（既满足误差校正点类型又满足到达该校正点时可以进行误差校正），那么简单移动$\langle i,j\rangle \to \langle i,k\rangle (j \neq k)$表示无人机第 i 次校正从误差校正点 j 转移到误差校正点 k。

（2）候选集的构造

候选集的选择一般由邻域中的邻居组成，这里我们选取前 10 个距离目标误差校正点最近的同类型误差校正点。

首先移动解的单个分量，在寻找可移动分量的过程中，两个分量的类型应该是一致的，在相互移动的过程中，若移动该点时满足到达该校正点时可以进行误差校正，那么将其加入候选集。下面为产生候选集的算法，该流程图如图 5-2 所示：

候选集产生算法步骤

Step1：选定类型匹配的误差校正点；
Step2：遍历选定的误差校正点；
Step3：选取前 10 个距目标误差校正点最近的误差校正点；
Step4：若在相互移动的过程中，移动该点时满足到达该校正点时可以进行误差校正，则标记为可插入分量；
Step5：将可插入分量集合记为候选集，选择时按轮盘选择法进行选择。

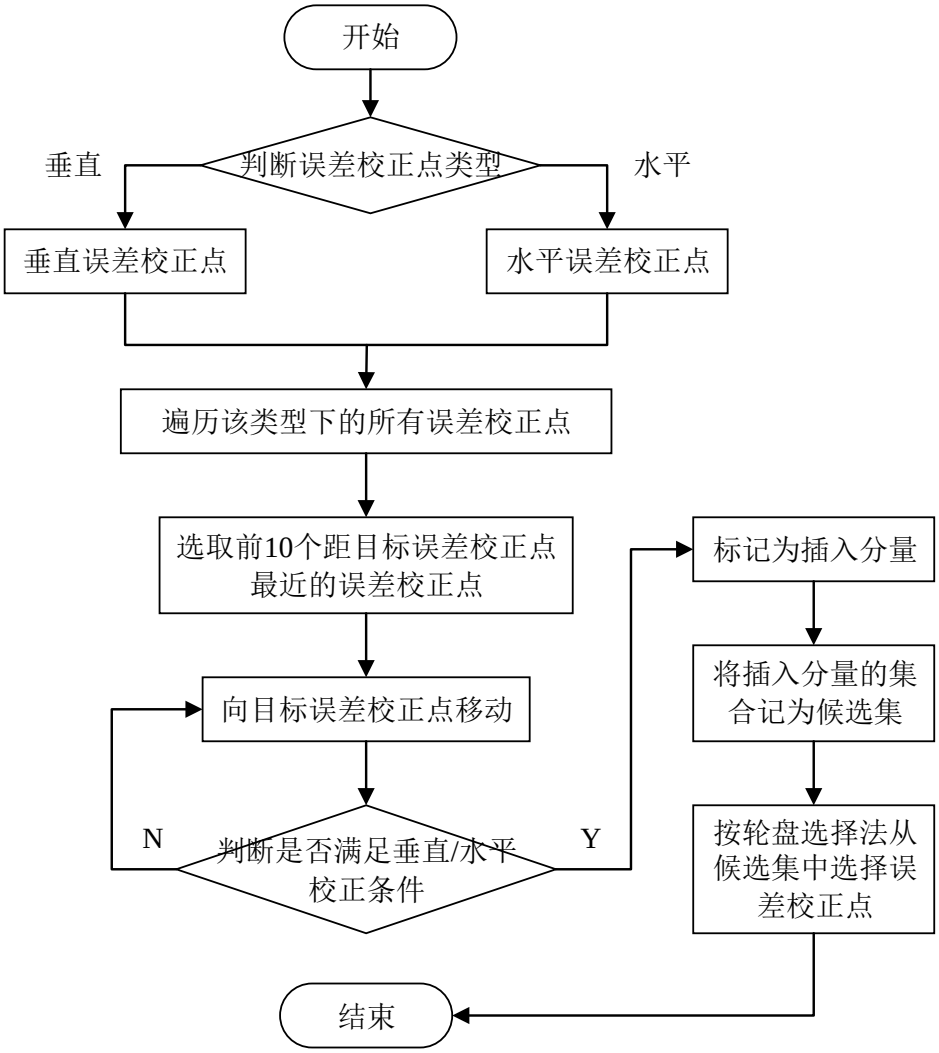

图 5-2 候选集产生算法

（3）禁忌搜索

其基本思想是在搜索过程中将近期的历史搜索存放在禁忌表中，防止算法重复搜索，即之前搜索过的区域不会在迂回搜索，这样便大大提高了搜索的效率，有效避免算法陷入循环和局部最优解。其核心时禁忌表的管理，禁忌表分为禁忌对象和禁忌长度。

其中，禁忌对象是导致解变化的重要因素。在本问题中，我们将简单移动和它的逆向移动作为禁忌对象。在禁忌期内 $\langle i,j\rangle \to \langle i,k\rangle (j \neq k)$ 和 $\langle i,k\rangle \to \langle i,j\rangle (j \neq k)$ 都将被禁止，从而避免迂回搜索。

禁忌长度即禁忌表大小，当一个对象进入禁忌表，只有在经过禁忌表长度后才能移出。如果禁忌表长度很短，此时该算法局部搜索能力强，解禁范围大，程序占用的内存和时间小，但是很容易陷入局部最优。如果禁忌表长度很长，此时对象在禁忌表内的时间很长，可以避免陷入局部最优，保证了解的广域性，但程序占用的内存较大，运行时间较长。因此，权衡两种思路，我们使用 $\sqrt{N}$ （N 为误差校正点的数量）表示禁忌表长度。该算法流程图如图 5-3 所示：

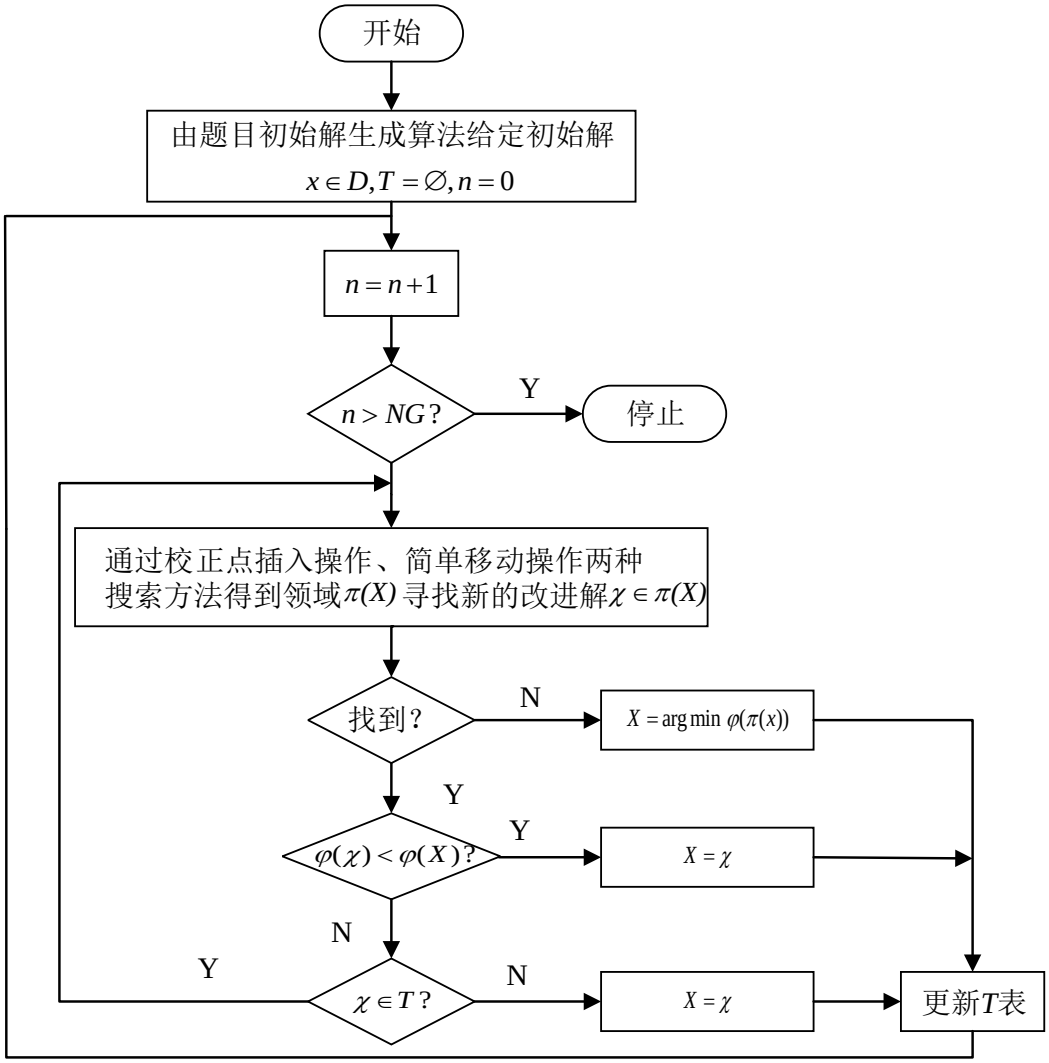

图 5-3 禁忌搜索算法流程图

（4）初始化解

这里我们使用贪心算法生成初始解。首先产生长度为 N（N 为误差校正点的数量）的全 0 列表。根据航迹长度最短原则，选出所有满足约束条件的点，组成一个集合，然后从该集合中选出离 B 点最近的点作为目的地，并将对应的编号标记为 1 同时将该点作为出发点。反复进行上述操作直到无人机可以直接飞到 B 点。具体流程如下，该流程图如图 5-4 所示：

贪心算法步骤

Step1：将 A 点作为出发点；
Step2：遍历误差校正点，选出所有满足约束条件的下一点，组成一个集合；
Step3：从该集合中选出离 B 点最近的点，作为目的地；
Step4：若选中的该点可以直接飞到 B 点，则算法结束，保存路线，否则将该点作为出发点返回 Step2。

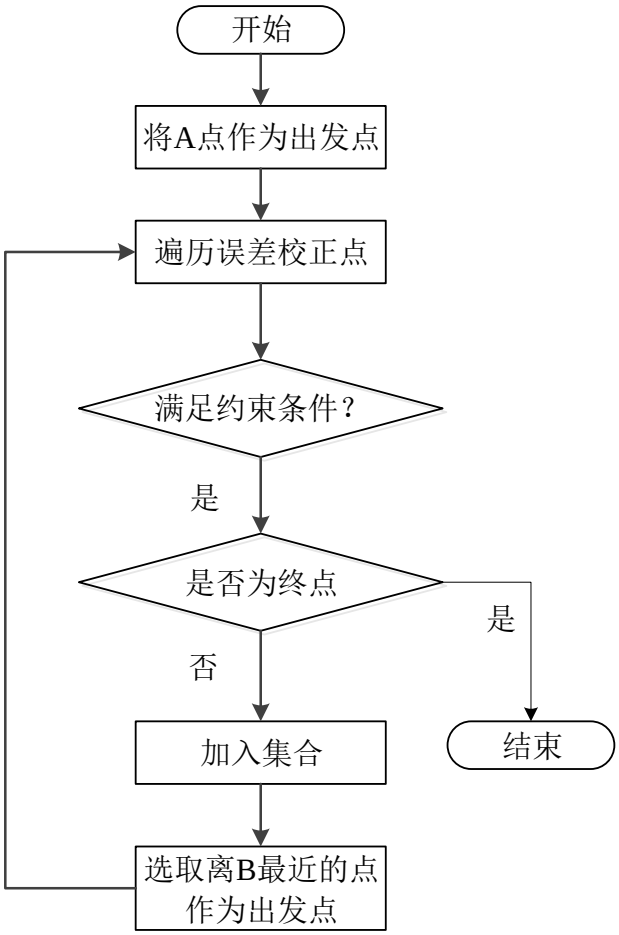

图 5-4 初始解生成流程图

（5） 目标解的迭代

通过分析表中数据，可以看到迭代次数在 120 次以上时，解的改善情况变化不大，可以认为在 120 次以后能得到较优解，而此后迭代次数上升会导致运行时间增加，因此具体迭代次数可以综合运行时间和改善情况，各迭代次数解情况如下表 5-3 所示。

表 5-3 各迭代次数解的情况

迭代次数	运行时间（s）	目标函数值	航迹距离（m）	校正次数	总体改善情况
30	13.95	293	112329	7	~
60	32.44	280	107253	8	4.64%
90	51.69	276	105865	8	1.45%
120	70.64	275	104898	8	0.363%

5.2.3 求解结果及分析

（1）航迹规划和误差校正点数

本问题采用禁忌搜索算法迭代得到较优解，根据解的编码方式进行解码，得到满足条件的航迹规划，最终得到的航迹规划路线如下。

数据集 1 航迹规划路线如下图 5-5 所示，具体数据如下：

$A \rightarrow 503 \rightarrow 69 \rightarrow 237 \rightarrow 155 \rightarrow 338 \rightarrow 457 \rightarrow 555 \rightarrow 436 \rightarrow B$，航迹长度为 104898m，误差校正点 8 点。

表 5-4 数据集 1：飞行器经过的误差校正点编号与校正前后误差

校正点编号	校正前垂直误差	校正前水平误差	校正点类型	校正后垂直误差	校正后水平误差
0	0	0	出发点 A	~	~
503	13.38791985	13.38791985	1	0	13.38791985
69	8.807342267	22.19526212	0	8.807342267	0
237	21.30682538	12.49948312	1	0	12.49948312
155	11.20081737	23.70030049	0	11.20081737	0
338	23.38661745	12.18580008	1	0	12.18580008
457	12.81307763	24.99887772	0	12.81307763	0
555	24.50332039	11.69024275	1	0	11.69024275
436	7.355848347	19.0460911	0	7.355848347	0
612	22.31369179	14.95784344	B	22.31369179	14.95784344

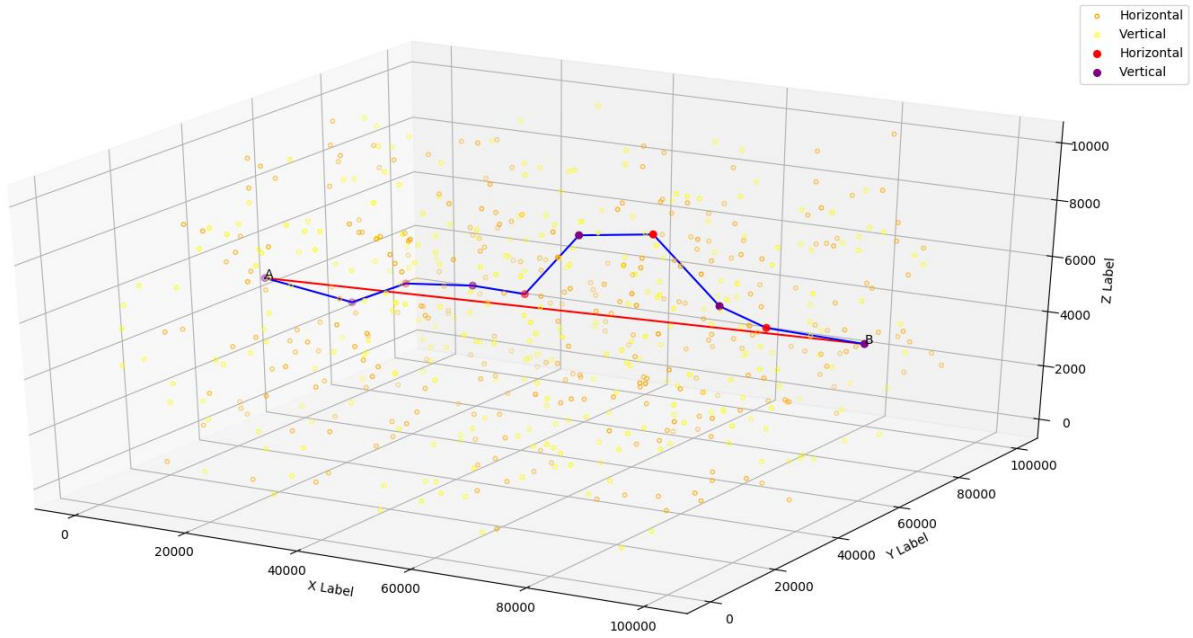

图 5-5 数据集 1 航迹 3D 图

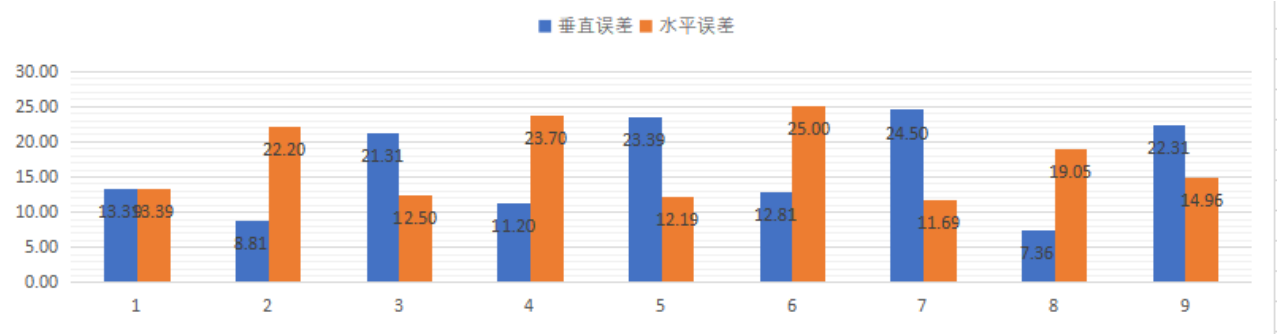

图 5-6 数据集 1：各个校正点误差分布图

结合数据表 5-4，由上柱状图 5-6 可以直观地看出，飞行器到达垂直校正点时，垂直误差较大；飞行器到达水平校正点时，水平误差较大。误差都在约束条件之内尽可能大，且安排的校正点交替出现，校正点位置安排合理，能够在误差即将超出限制条件时及时校正。航行轨迹沿着 AB 方向上下穿梭，安排较为合理，航行轨迹长度为算法最优解，航迹长度 104898m，仅使用 8 个误差校正点，AB 直线距离为 100465m，航迹长度仅比 AB 直线距离增加了 4.41%。

为了进一步说明本文针对问题一所设计的算法求解出的结果是最优解，我们选取了部分数据集 1 其他解果，参考下表 5-5：

表 5-5 数据集 1 中不同校正点个数下的结果

校正点个数	航迹路径	航迹长度
7	无	无
8	A→503→294→91→282→33→315→403→594→501→B	105081m
	A→303→199→15→148→278→369→214→397→B	105865m
9	A→285→303→366→607→170→540→250→340→277→B	111462m
	A→578→64→80→148→155→278→369→214→397→B	108316m
10	A→285→303→562→351→418→278→369→214→555→18→B	112722m
	A→521→303→64→607→170→278→375→457→340→425→B	112721m

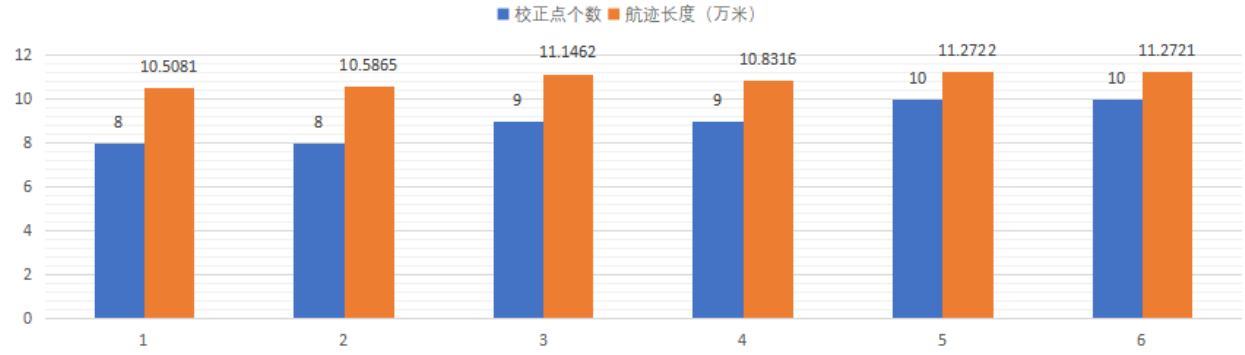

图 5-7 校正点个数及航迹长度对比柱状图

由表 5-5 和图 5-7 可以观察到，在我们选取的部分结果中，校正点个数为 7 个时，无法找到任意一条可行路径；且随着校正点个数增加，航迹长度与校正点个数呈正相关。可以从这个角度说明本文求解出的 8 个校正点，航迹长度 104898m 为算法的最优解。

数据集 2 航迹规划路线如下图 5-8 所示，具体数据如下：

$A \rightarrow 163 \rightarrow 114 \rightarrow 8 \rightarrow 309 \rightarrow 305 \rightarrow 123 \rightarrow 45 \rightarrow 160 \rightarrow 92 \rightarrow 93 \rightarrow 61 \rightarrow 292 \rightarrow B$，航迹长度为 109342m，误差校正点 12 点。

表 5-6 数据集 2：飞行器经过的误差校正点编号与校正前后误差

校正点编号	校正前垂直误差	校正前水平误差	校正点类型	校正后垂直误差	校正后水平误差
0	0	0	出发点 A	~	~
163	13.28789761	13.28789761	0	13.28789761	0
114	18.62205093	5.334153324	1	0	5.334153324
8	13.92198578	19.2561391	0	13.92198578	0
309	19.44631118	5.524325401	1	0	5.524325401
305	5.968714547	11.49303995	0	5.968714547	0
123	15.17310764	9.204393096	1	0	9.204393096
45	10.00616142	19.21055451	0	10.00616142	0
160	17.49129596	7.485134547	1	0	7.485134547
92	5.776163625	13.26129817	0	5.776163625	0
93	15.26088202	9.484718396	1	0	9.484718396
61	9.834209702	19.3189281	0	9.834209702	0
292	16.38812359	6.553913884	1	0	6.553913884
326	6.960509275	13.51442316	B	6.960509275	13.51442316

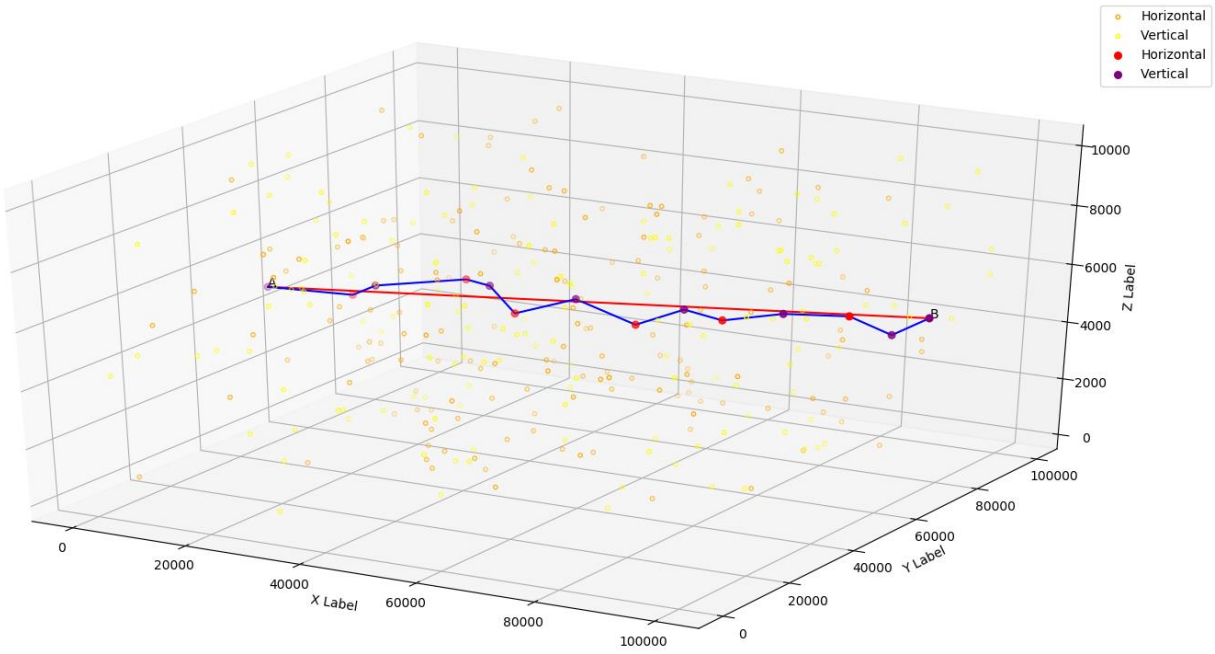

图 5-8 数据集 2 航迹 3D 图

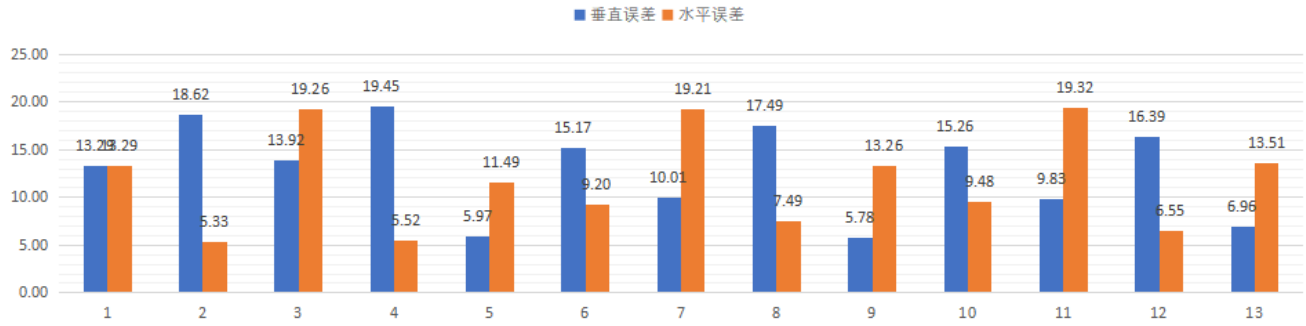

图 5-9 数据集 2：各垂直/水平误差校正点误差分布图

结合数据表 5-6，由上柱状图表 5-9 可以直观地看出，飞行器到达垂直校正点时，垂直误差较大；飞行器到达水平校正点时，水平误差较大。误差都在约束条件之内尽可能大，且安排的校正点交替出现，校正点位置安排合理，能够在误差即将超出限制条件时及时校正。航行轨迹沿着 AB 方向上下穿梭，安排较为合理，航行轨迹长度为算法最优解，航迹长度 109342m，仅使用 12 个误差校正点，AB 直线距离为 103045m，航迹长度仅比 AB 直线距离增加了 6.11%。

为了进一步说明本文针对问题一所设计的算法求解出的结果是最优解，我们选取了部分数据集 2 其他解果，参考下表 5-7：

表 5-7 数据集 1 中不同校正点个数下的结果

校正点个数	航迹路径	航迹长度
11	无	无
12	A→163→114→8→309→305→123→45→160→92→93→61→292→B	109342m
	A→163→114→8→309→121→123→45→160→92→93→61→292→B	110772m

13	A→184→163→114→8→309→54→123→115→160→92→93→61→292→B	112315m
	A→184→163→114→8→309→121→123→45→160→92→93→61→166→B	113883m
14	A→184→163→114→8→309→54→123→115→160→92→93→61→292→135→B	115793.m
	A→184→163→114→8→309→121→123→45→160→92→93→61→292→135→3B	115933m

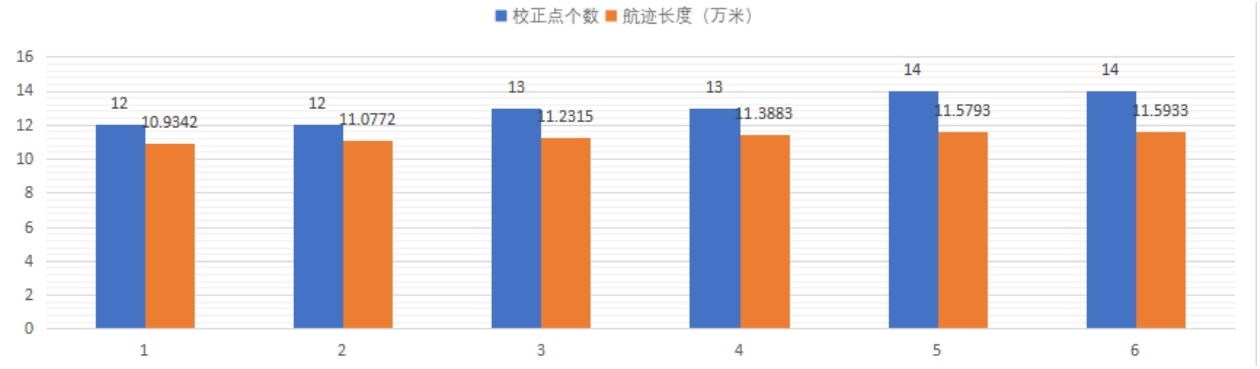

图 5-10 校正点个数及航迹长度对比柱状图

由表 5-7 和图 5-10 可以观察到，在我们选取的部分结果中，校正点个数为 11 个时，无法找出任意一条可行的路径；且随着校正点个数增加，航迹长度与校正点个数呈正相关。可以从这个角度说明本文求解出的 12 个校正点，航迹长度 109342m 为算法的最优解。

5.3 模型评估

5.3.1 算法的有效性和复杂度

双目标规划问题的最主要特点是各个目标间的矛盾性和不可度量性。目标间的矛盾性是指如果采用某一个方案试图去优化一个目标的值，则可能会导致另一个目标的值变差。目标间的不可度量性是指各目标间一般并没有一致的量纲导致没有统一的度量标准，因此不能直接进行比较。基于上述性质，不能将双目标规划问题直接归并为单目标规划问题来解决。

在本模型中，我们以航迹路线最小化、误差校正点总数最小化的优先级顺序作为主线，将双目标规划模型通过无量纲化处理和参考优先级顺序，引入线性加权系数，使复杂的多目标规划模型求解转换为单目标规划模型，极大降低了问题的求解难度。本问题首先通过贪心算法来获取较好的初始解，这大大减少了算法后续寻找最优解的时间。同时禁忌搜索算法使用禁忌表来避免重复搜索，这大大提高了搜索的效率，有效避免算法陷入循环和局部最优解，从而实现全局优化。结果表明禁忌搜索算法在飞行器航迹规划这一 NP-hard 的求解过程中，展现了较好的寻优能力，具有良好的有效性，比该算法路径短的，即便存在，校正点数也会比我们的多；比这个校正点数少的，距离肯定大于该算法求出的路径。禁忌搜索算法各操作时间复杂度见下表 5-4：

表 5-4 禁忌搜索算法各操作时间复杂度

操作名称	时间复杂度
产生初始解	n^2

生成邻域	$C_n^k(k-opt)$
判断禁忌表	$n\times l$（n 为问题规模，l 为禁忌表长度）
解除禁忌表	$n\times l$

禁忌搜索算法时间复杂度为$O(\text{Max_GEN}\times(n^2+n+\text{l}))$，在程序运行过程中，找到最优解的时间数据集 1 为 66.78s，数据集 2 为 54.23s。

5.3.2 灵敏度分析

本文在设计算法时，引入了一个全局搜索常量 k，用于在当前时间搜索距离飞行器当前位置最近的 k 个可行校正点。如何选取合适的 k 值是一个值得思考的问题，如果 k 的值选取较小，则只能在少量的备选点内寻找合适的校正点，最终结果可能与全局最优解相差较大；k 值选取较大，增加了程序的时间开销，空间开销，且较多的备选点可能出现混淆，甚至出现“绕圈”的情况。因此，本文设计了对算法中 k 值的灵敏度分析，尽可能选取一个合适的 k 值，使结果趋向最优解。

首先分析数据集 1：如下图 5-9 和下图 5-10 所示，可以观察到，当 k 取 5 时，航迹长度较大；当 k 取 10 时，航迹长度在整体上最小；当 k 大于 10 时，航迹长度开始增大。

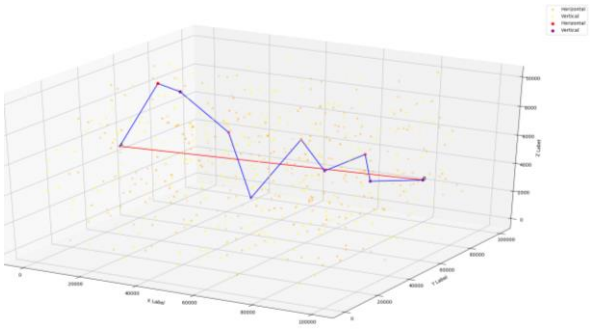

图 5-9(a)：k=5；轨迹长度：112329m

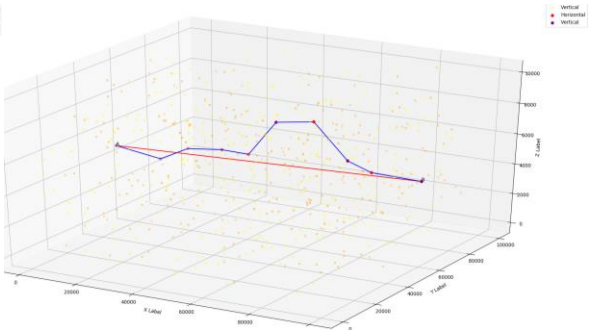

图 5-9(b)：k=10；轨迹长度：104898m

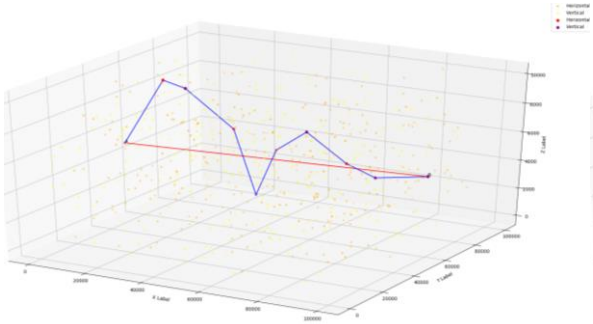

图 5-9(c)：k=15；轨迹长度：105865m

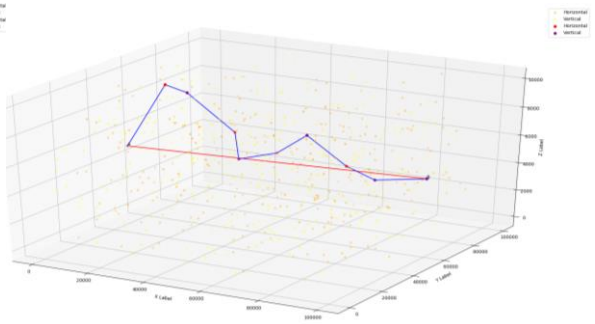

图 5-9(d)：k=20；轨迹长度：107253m

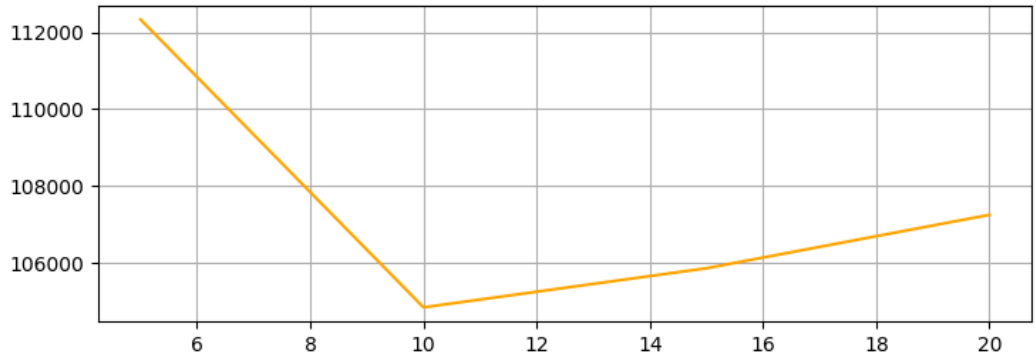

图 5-10 数据集 1 航行轨迹长度随 k 值变化曲线

再分析数据集 2：如下图 5-11 和下图 5-12 所示，可以观察到，当 k 取 5 时，航迹长度较大；当 k 取 10 时，航迹长度在整体上最小；当 k 大于 10 时，航迹长度开始增大。

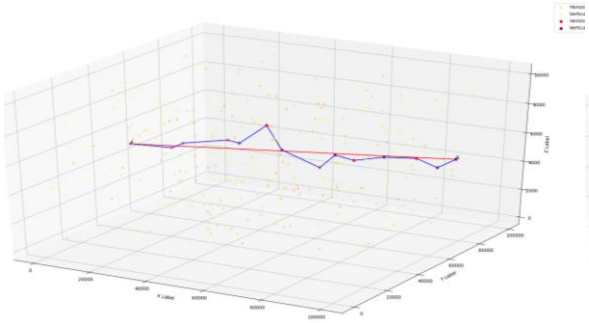

图 5-11(a)：k=5；航机长度：111585m

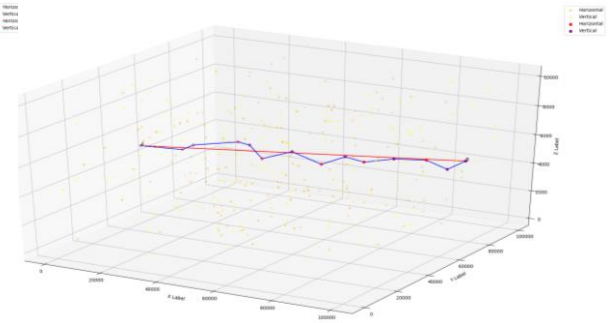

图 5-11(b)：k=10；航机长度：109342m

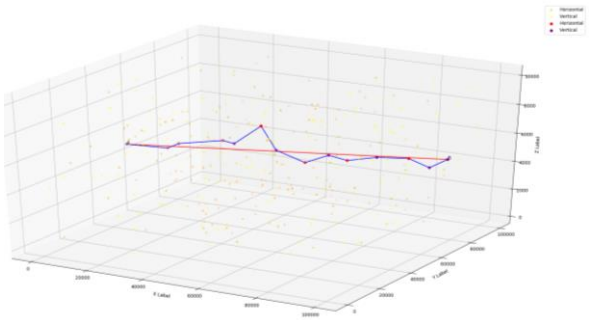

图 5-11(c)：k=15；航机长度：109786m

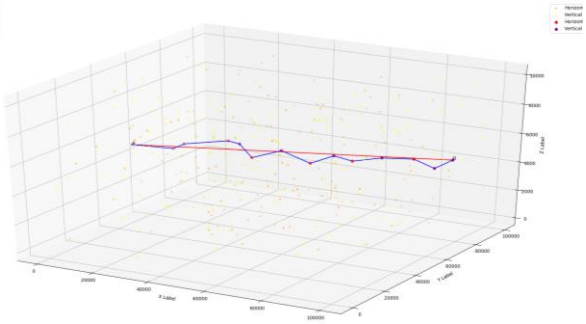

图 5-11(d)：k=20；航机长度：110772m

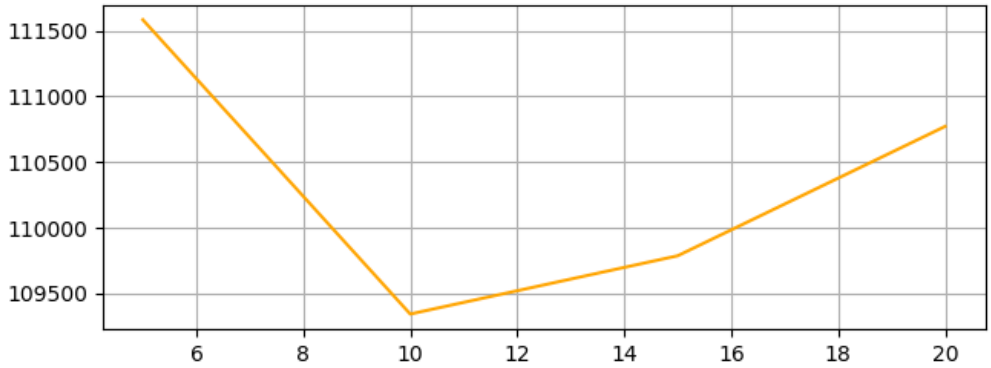

图 5-12 数据集 2 航行轨迹长度随 k 值变化曲线

可以得出结论，当 k 取值偏小时，航迹路径规划不合理，会漏掉较多优良的校正点，航迹长度显著偏大；当 k 取值在 10 附近时，则不会遗漏附近优良的校正点，航迹路径规划合理，航机长度趋于最优值。

6. 问题二：模型建立与求解

6.1 模型建立

6.1.1 短航迹-少校正次数-圆弧式转向双目标规划模型

优化目标：（1）飞行器从出发地 A 航行至目的地 B 的航线距离尽可能短；（2）经过校正区域进行校正的次数尽可能少。因此，定义航迹长度变量 *Distance*，即飞行器从出发地 A 航行至终点 B 所经过的总路程；定义经过校正区域进行校正的次数 *Times*。

目标函数：采用短航迹长度-少校正次数双目标优化模型，即同时优化航迹长度和校正次数双变量。目标函数可定义为：

$$\min z_3 = Distance \tag{6-1}$$

$$\min z_4 = Times \tag{6-2}$$

决策变量：（1）定义决策变量 x_{ij}，当前校正点（或起点）i 的下一个最优校正点（或终点）j 是否连通，用以确定最优路径。

$$x_{ij} = \begin{cases} 0 & i点和j点没有连接 \\ 1 & i点和j点存在连接 \end{cases} \tag{6-3}$$

同时，使用向量 $\mathbf{x} = [x_{ij}^{(1)}, x_{ij}^{(2)}, ... x_{ij}^{(k)} ...]$ 记录每一次的决策，其中 $x_{ij}^{(k)}$ 表示第 k 次选择连接的 i，j 两点，用向量 **x** 来记录最佳路径。

目标函数中的 *Distance* 和 *Times* 可以通过决策变量表示为：

$$\begin{aligned} Distance &= \sum_{i=0}^{N}\sum_{j=0}^{N} x_{ij} \times d_{ij} \\ Times &= \sum_{i=0}^{N}\sum_{j=0}^{N} x_{ij} \end{aligned} \tag{6-4}$$

其中 d_{ij} 表示从 i 点飞行至 j 点经过的路程。

约束条件：（1）飞行器垂直校正条件：垂直定位误差 ε_v 小于 α_1，水平定位误差 ε_h 小于 α_2；（2）飞行器水平校正条件：垂直定位误差 ε_v 小于 β_1，水平定位误差 ε_h 小于 β_2；（3）飞行器到达目的地前，垂直误差和水平误差都应小于 θ；（4）飞行器转向轨迹为圆弧，最小转弯半径为 200 米。

由于飞行器转向过程受到限制，在三维空间内转向情况复杂，分析转向情况如下：首先分析飞行器飞行时从空间中某点 P 飞向空间中另一点 D 的过程。由于问题中限定了飞行器转向的轨迹为圆弧，最小半径为 r，可以借助 Dubins 曲线[7]分析各种可能出现的情况。

Dubins 曲线是在满足曲率约束和规定起点、终点的速度方向（切线）的条件下，连接两个二位平面的最短路径，且限制目标只能向前行进。如图 6-1 所示，物体从点 S 有一个

初速度$\vec{v}_S$，且要求物体到达 E 时以速度$\vec{v}_E$驶离，Dubins 曲线即为一条满足要求的最短路径。Dubins 曲线在空间中需要经过平面P_1和P_2中的一段圆弧完成转向，其曲线由圆弧和直线段构成。

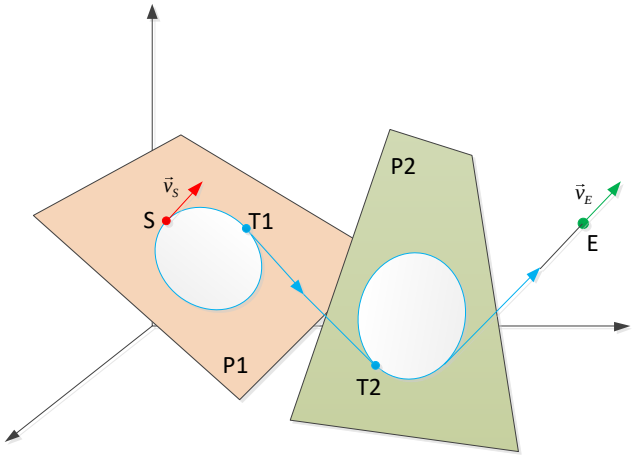

图 6-1 Dubins 曲线空间示意图

本文参考了 Dubins 曲线的 4 种情况，绘制了这 4 种情况的示意图，参考图 6-2：

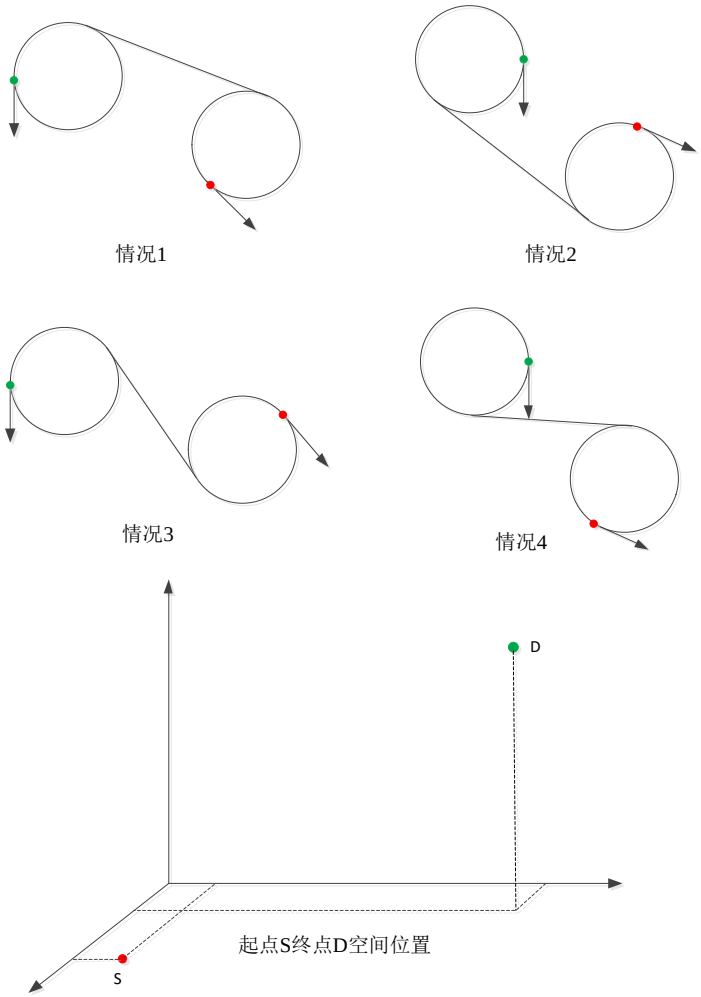

图 6-2 Dubins 曲线四种情况示意图

考虑本问题，由于问题中限定了飞行器转向的轨迹为圆弧，最小半径为 r，我们希望飞行器飞行方向始终是前进的（飞行方向与位置至终点方向夹角小于 90 度），因此设定每次进行转向的弧度小于 π（不会向后飞行）。分析可得，在上述条件下，飞行器从空间某点 S 飞向空间中另一点 D 的轨迹情况为 Dubins 曲线，可抽象为以下两种情况，如下图 6-3 和图 6-4 所示：

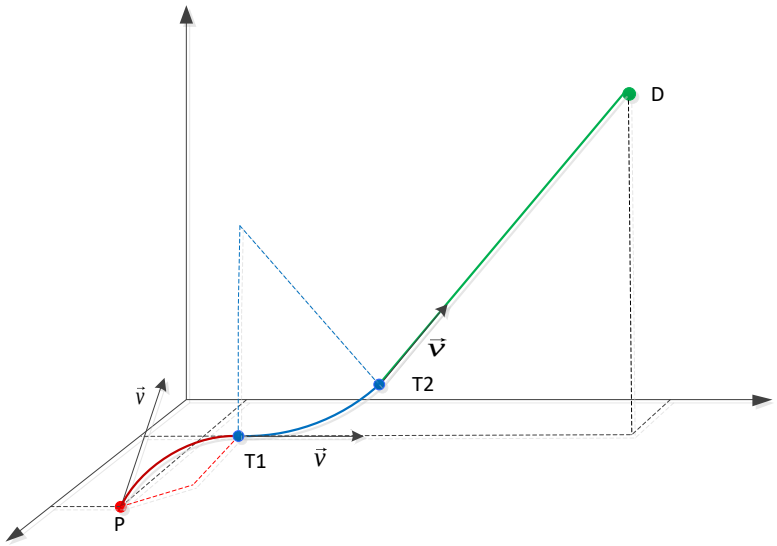

图 6-3 飞行器转向轨迹情况 1

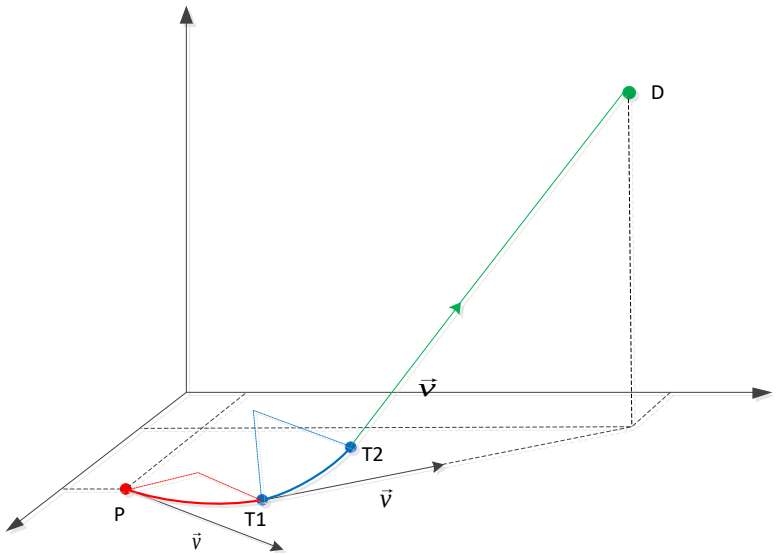

图 6-4 飞行器转向轨迹情况 2

首先本文假定飞行器在非转向状态时沿直线飞行，以飞行平面建立 X-Y 坐标轴，Z 轴垂直 X-Y 坐标轴。飞行器首先需要在 X-Y 平面上转向至 D 点在 X-Y 方向的投影点，然后再向 Z 轴方向转向，直到速度方向与停止转向点至目标点方向一致，即可完成转向。即任何一次从出发点 P 至空间中另一点 D，经历的路程为：$PT_1 + T_1T_2 + \|T_2D\|$。根据出发点 P 和终点 D 在空间种不同的位置情况，PT_1，T_1T_2，$\|T_2D\|$ 的取值存在以下几种情况：

(1) $\overrightarrow{PD}$ 与 $\vec{v}$ 共线，则飞行器不需要转向，因此 PT_1 ， T_1T_2 为 0；

(2) $\overrightarrow{PD}$ 与 $\vec{v}$ 共面，则飞行器仅需要完成一次转向，因此 T_1T_2 为 0；

(3) $\overrightarrow{PD}$ 与 $\vec{v}$ 异面，则飞行器需要完成二次转向，$\|T_2D\|$ 可能为 0。

一般而言飞行器在三维空间中从第 i 校正点飞行至第 j 校正点，其飞行路程总是可以用两段圆弧和一段直线来描述。因此可以将 d_{ij} 表述为：

$$d_{ij}=l_1+l_2+y \tag{6-5}$$

其中 l_1 和 l_2 表示 PT_1 ， T_1T_2 所代表的圆弧，y 表示 $\|T_2D\|$ 所代表的直线。

分析完转向情况，在探讨约束条件前，定义校正器类型变量 T_i：

$$T_i=\begin{cases}0 & \text{第}i\text{个校正点是水平校正点}\\ 1 & \text{第}i\text{个校正点是垂直校正点}\end{cases} \tag{6-6}$$

约束条件分析如下：

1）当飞行器从起点出发时，即 k=1 时，会选择一个合适的水平校正点或垂直校正点。两种情况的约束条件汇总如下表 6-1：

表 6-1 飞行器从起点选择校正点约束条件

情况	到达校正点时的误差	约束条件
选择水平校正点	$\varepsilon_h=\delta\times d_{0j}\times x_{0j}$	$\varepsilon_h\le\beta_2$
	$\varepsilon_v=\delta\times d_{0j}\times x_{0j}$	$\varepsilon_v\le\beta_1$
选择垂直校正点	$\varepsilon_h=\delta\times d_{0j}\times x_{0j}$	$\varepsilon_h\le\alpha_2$
	$\varepsilon_v=\delta\times d_{0j}\times x_{0j}$	$\varepsilon_v\le\alpha_1$

因此，选择合适的校正器类型变量 T_i，当 k=1 时的约束条件可以表述为：

$$\begin{cases}k=1\\ \delta\times d_{0j}\times x_{0j}\times(1-T_i)\le\beta_1\\ \delta\times d_{0j}\times x_{0j}\times(1-T_i)\le\beta_2\\ \delta\times d_{0j}\times x_{0j}\times T_i\le\alpha_1\\ \delta\times d_{0j}\times x_{0j}\times T_i\le\alpha_2\end{cases} \tag{6-7}$$

2）当飞行器从第 i 个校正点出发时，即 k>1 时，会选择一个合适的水平校正点或垂直校正点。两种情况的约束条件汇总如下表 6-2：

表 6-2 飞行器从校正点选择下一校正点约束条件

情况		到达校正点时的误差	约束条件
选择水平校正点	上一次经过水平校正点	$\varepsilon_h=\delta\times\left(d_{ij}^{(k)}\times x_{ij}^{(k)}\right)$	$\varepsilon_h\le\beta_2,\theta$
		$\varepsilon_v=\delta\times\left(d_{ij}^{(k)}\times x_{ij}^{(k)}+d_{ij}^{(k-1)}\times x_{ij}^{(k-1)}\right)$	$\varepsilon_v\le\beta_1,\theta$
	上一次经过垂直校正点	$\varepsilon_h=\delta\times\left(d_{ij}^{(k)}\times x_{ij}^{(k)}+d_{ij}^{(k-1)}\times x_{ij}^{(k-1)}\right)$	$\varepsilon_h\le\beta_2,\theta$
		$\varepsilon_v=\delta\times\left(d_{ij}^{(k)}\times x_{ij}^{(k)}\right)$	$\varepsilon_v\le\beta_1,\theta$
选择垂直校正点	上一次经过水平校正点	$\varepsilon_h=\delta\times\left(d_{ij}^{(k)}\times x_{ij}^{(k)}+d_{ij}^{(k-1)}\times x_{ij}^{(k-1)}\right)$	$\varepsilon_h\le\alpha_2,\theta$
		$\varepsilon_v=\delta\times\left(d_{ij}^{(k)}\times x_{ij}^{(k)}\right)$	$\varepsilon_v\le\alpha_1,\theta$
	上一次经过垂直校正点	$\varepsilon_h=\delta\times\left(d_{ij}^{(k)}\times x_{ij}^{(k)}\right)$	$\varepsilon_h\le\alpha_2,\theta$
		$\varepsilon_v=\delta\times\left(d_{ij}^{(k)}\times x_{ij}^{(k)}+d_{ij}^{(k-1)}\times x_{ij}^{(k-1)}\right)$	$\varepsilon_v\le\alpha_1,\theta$

因此，选择合适的校正器类型变量T_i，当k>1时的约束条件可以表述为：

$$\begin{cases}k>1\\ \delta\times\left(d_{ij}^{(k)}\times x_{ij}^{(k)}\times(1-T_i)+d_{ij}^{(k-1)}\times x_{ij}^{(k-1)}\times T_i\right)\le\beta_1,\theta\\ \delta\times\left(d_{ij}^{(k)}\times x_{ij}^{(k)}\times(1-T_i)+d_{ij}^{(k-1)}\times x_{ij}^{(k-1)}\times(1-T_i)\right)\le\beta_2,\theta\\ \delta\times\left(d_{ij}^{(k)}\times x_{ij}^{(k)}\times T_i+d_{ij}^{(k-1)}\times x_{ij}^{(k-1)}\times T_i\right)\le\alpha_1,\theta\\ \delta\times\left(d_{ij}^{(k)}\times x_{ij}^{(k)}\times T_i+d_{ij}^{(k-1)}\times x_{ij}^{(k-1)}\times(1-T_i)\right)\le\alpha_2,\theta\end{cases}\tag{6-8}$$

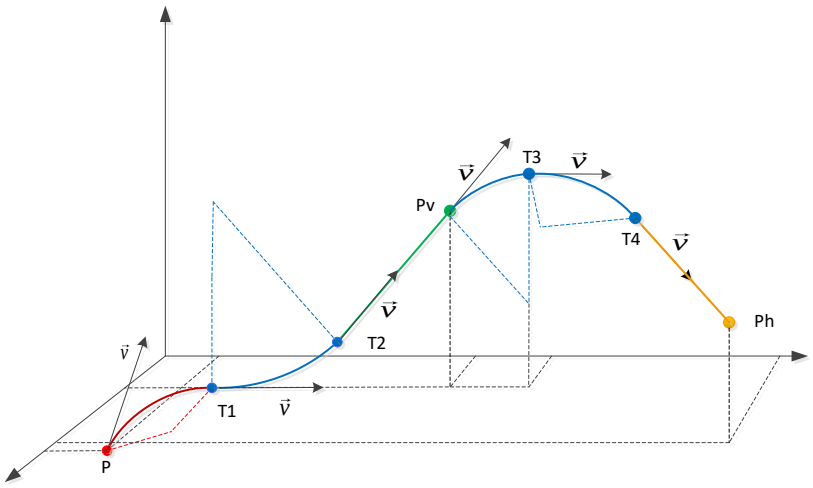

图 6-5 飞行器经过两个校正点示意图

3)决策变量的约束条件。由于定义的决策变量x_{ij}的目的是寻找一条最优路径，即x_{ij}构成的邻接矩阵是一个有向无环图，且不是一种树结构。这意味着每个校正点最多只能经过

一次，且必须保证从起点到终点是一条通路。

首先应当约束邻接矩阵每一列的和小于等于 1，这样可以确保每个校正点最多经过一次；

其次需要约束起点到终点是一条通路。本文使用向量 $\mathbf{x}=[x_{ij}^{(1)},x_{ij}^{(2)},...x_{ij}^{(k)}...]$ 记录每一次的决策，其中 $x_{ij}^{(k)}$ 表示第 k 次选择连接的 i，j 两点，展开为 $x_{ij}^{(k)}(i_k,j_k)$。应当使第 k 次选择的点 $x_{ij}^{(k)}(i_k,j_k)$ 纵坐标与第 k+1 次选择的点 $x_{ij}^{(k)}(i_k,j_k)$ 横坐标相等，即 $j_k=i_{k+1}$。

因此，决策变量的约束条件可以表述为：

$$\begin{cases}\sum_{j=0}^{N}x_{ij}\le 1\\ j_k=i_{k+1}\\ i,j\in(0,1,2,...N)\end{cases}\tag{6-9}$$

综上所述，所建立的双目标优化模型可以表述为：

$$\min z_1=Distance=\sum_{i=0}^{N}\sum_{j=0}^{N}x_{ij}\times d_{ij}$$

$$\min\ z_2=Times=\sum_{i=0}^{N}\sum_{j=0}^{N}x_{ij}$$

$$s.t.\begin{cases}\begin{cases}k=1\\ \delta\times d_{0j}\times x_{0j}\times(1-T_i)\le\beta_1\\ \delta\times d_{0j}\times x_{0j}\times(1-T_i)\le\beta_2\\ \delta\times d_{0j}\times x_{0j}\times T_i\le\alpha_1\\ \delta\times d_{0j}\times x_{0j}\times T_i\le\alpha_2\end{cases}\\ \begin{cases}k>1\\ \delta\times\left(d_{ij}^{(k)}\times x_{ij}^{(k)}\times(1-T_i)+d_{ij}^{(k-1)}\times x_{ij}^{(k-1)}\times T_i\right)\le\beta_1,\theta\\ \delta\times\left(d_{ij}^{(k)}\times x_{ij}^{(k)}\times(1-T_i)+d_{ij}^{(k-1)}\times x_{ij}^{(k-1)}\times(1-T_i)\right)\le\beta_2,\theta\\ \delta\times\left(d_{ij}^{(k)}\times x_{ij}^{(k)}\times T_i+d_{ij}^{(k-1)}\times x_{ij}^{(k-1)}\times T_i\right)\le\alpha_1,\theta\\ \delta\times\left(d_{ij}^{(k)}\times x_{ij}^{(k)}\times T_i+d_{ij}^{(k-1)}\times x_{ij}^{(k-1)}\times(1-T_i)\right)\le\alpha_2,\theta\end{cases}\\ \begin{cases}\sum_{j=0}^{N}x_{ij}\le 1\\ j_k=i_{k+1}\\ i,j\in(0,1,2,...N)\end{cases}\\ \alpha_1,\alpha_2,\beta_1,\beta_2,\delta,\theta>0\\ d_{ij}^{(k)}=l_1^{(k)}+l_2^{(k)}+y^{(k)}\end{cases}\tag{6-10}$$

6.2 模型求解

6.2.1 双目标转化为单目标

在本问题中，由最小化航迹变量 *Distance* 和最小化经过矫正区域进行校正的次数 *Times* 建立了两个目标函数，选取两个值 $0<\gamma_i<1(i=1,2)$，作为这两个目标的线性加权系数，并且 $\sum_{i=1}^{2}\gamma_i$。由于本问题中两个目标函数的量纲不同，因此需要归一化目标函数，对其进行无量纲化处理。此时目标函数分别通过函数 $\alpha_{z_3}=\dfrac{z_3-\min z_3}{\max z_3-\min z_3}$，$\alpha_{z_4}=\dfrac{z_4-\min z_4}{\max z_4-\min z_4}$ 规范。此时，由于目标函数归一化导致整个目标函数缩小至变化细微，因此需要乘以 100，这样迭代时目标函数区分明显。该问题就转化为了单目标数学规划问题，转化后的单目标函数可以写为：

$$\min z_{z_2}=(\gamma_1\alpha_{z_3}+\gamma_2\alpha_{z_4})\times 100 \tag{6-11}$$

因此，模型可以表述为：

$$\min z_{z_2}=(\gamma_1\alpha_{z_3}+\gamma_2\alpha_{z_4})\times 100$$

$$s.t.\begin{cases}\begin{cases}k=1\\ \delta\times d_{0j}\times x_{0j}\times(1-T_i)\le\beta_1\\ \delta\times d_{0j}\times x_{0j}\times(1-T_i)\le\beta_2\\ \delta\times d_{0j}\times x_{0j}\times T_i\le\alpha_1\\ \delta\times d_{0j}\times x_{0j}\times T_i\le\alpha_2\end{cases}\\ \begin{cases}k>1\\ \delta\times\left(d_{ij}^{(k)}\times x_{ij}^{(k)}\times(1-T_i)+d_{ij}^{(k-1)}\times x_{ij}^{(k-1)}\times T_i\right)\le\beta_1,\theta\\ \delta\times\left(d_{ij}^{(k)}\times x_{ij}^{(k)}\times(1-T_i)+d_{ij}^{(k-1)}\times x_{ij}^{(k-1)}\times(1-T_i)\right)\le\beta_2,\theta\\ \delta\times\left(d_{ij}^{(k)}\times x_{ij}^{(k)}\times T_i+d_{ij}^{(k-1)}\times x_{ij}^{(k-1)}\times T_i\right)\le\alpha_1,\theta\\ \delta\times\left(d_{ij}^{(k)}\times x_{ij}^{(k)}\times T_i+d_{ij}^{(k-1)}\times x_{ij}^{(k-1)}\times(1-T_i)\right)\le\alpha_2,\theta\end{cases}\\ \begin{cases}\sum_{j=0}^{N}x_{ij}\le 1\\ j_k=i_{k+1}\\ i,j\in(0,1,2,\ldots N)\end{cases}\\ \alpha_1,\alpha_2,\beta_1,\beta_2,\delta,\theta>0\\ d_{ij}^{(k)}=l_1^{(k)}+l_2^{(k)}+y^{(k)}\end{cases} \tag{6-12}$$

接下来便转化为单目标的禁忌搜索算法求解目标函数。

6.2.2 基于单目标规划的改进型禁忌搜索算法

（1）藐视准则

在禁忌搜索算法中，可能会出现候选集全部被禁忌，或者存在一个优于“best so far”状态的禁忌候选解，此时可以通过特赦准则将某些状态解禁来实现更高效率的优化。特赦准则一般有两种原则：

1）基于目标函数值的原则：某个禁忌候选解的目标函数值优于“best so far”则解禁当前状态和新的“best so far”状态。

2）基于搜索方向上的准则：若禁忌对象上次被禁忌时目标函数值有所改善，并且当前该禁忌对象对应的候选集的目标函数值优于当前解，则对其解禁。

在本问题中，我们选择基于目标函数值的特赦准则，除此之外，我们还引入了另外的法则：如果一个互换操作在相当长的一段时间内没有被选中，那么不管它当前的禁忌属性和对目标函数值的影响，都会执行这个互换操作。

（2）改进禁忌长度

禁忌长度是禁忌算法中的一个关键因素：如果禁忌长度值取小了，则在搜索过程中容易出现重复搜索；如果禁忌长度值太大，又可能会禁止一些有益的移动操作，导致求解质量下降。

在该算法中禁忌长度采用动态变化的方式，在区间$\left(0.9\sqrt{n},1.1\sqrt{n}\right)$（$n$ 为误差校正点总数）中随机取值。

（3）改进禁忌搜索算法（Improved Tabu Search, ITS）

与其他启发式算法类似，禁忌搜索算法在求解优化问题时，存在着停滞现象，即所求解问题的目标函数在很长一段时间内得不到进一步优化。在开始阶段，目标函数最优值值会急剧下降，算法收敛速度非常快；然而随着搜索过程的进行，目标函数最优值的更新会越来越慢，并逐步停滞在某个状态，算法效率非常低；又经过相当长的时间，目标函数最优值再一次改变。

为了进一步提高禁忌搜索算法的搜索质量和收敛速度，引入了集中和和分散机制，提出了一种改进禁忌搜索算法(ITS)。ITS 的主要思想是，当搜索过程停滞时，及时对保留的局部优化解中的优良个体（成为精英）进行重组，构造出新的可行解，在此基础上再开展进一步搜索。具体而言，ITS 由集中搜索和精英重组两个步骤经过多次迭代完成。在集中搜索阶段，采用问题一的禁忌搜索算法，实现对重点区域的集中搜索；在精英重组阶段，对精英个体采用交叉操作，获得新的可行解，以加大全局搜索范围。该算法流程图如图 6-6 所示。

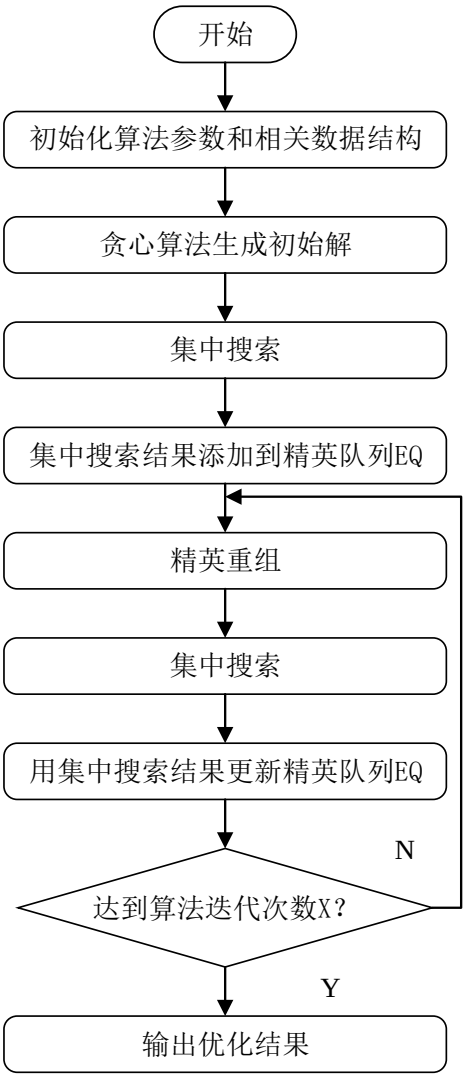

图 6-6 ITS 算法流程图

精英队列 EQ 的长度为 s，保留了历次集中搜索获得的 s 个最佳局部优化解，并根据个体解的优劣程度排序。在主循环的每一次迭代中，集中搜索输出集中优化结果 φ_1，并根据 φ_1 更新 EQ。若 EQ 中的个体数已达到 s，只有当 φ_1 优于队尾的个体 EQ[s]时，才用 φ_1 替换 EQ[s]，并对 EQ 重新排序。

集中搜索

集中搜索采用问题一的禁忌搜索算法，但禁忌长度在区间 $\left(0.9\sqrt{n},1.1\sqrt{n}\right)$（$n$ 为误差校正点总数）中随机取值。另外，当目前的禁忌对象所对应状态的目标函数值优于“best so far”状态时，使用特赦准则。

精英重组

交叉操作源于生物的自然进化过程，是遗传算法中的一个重要概念。交叉操作即从两个已有的可行解（父个体）中通过重组产生一个新的可行解。在交叉过程中，父个体的基本特征会被新的个体继承，同时子代又会产生异于父代的新变化。在 ITS 中，要求分散策

略既要保留已有的搜索状态信息，又要跳出发生“停滞”现象的搜索区域，因而对集中搜索得到的解集通过交叉操作产生新的可行解。

交叉操作采用整数交叉法，首先从解集中随机选取两个可行解，然后随机选择可交叉位置进行交叉。

（4）目标解的迭代

通过分析表 6-1 中数据，可以看到迭代次数在 120 次以上时，解的改善情况变化不大，可以认为在 120 次以后能得到较优解。

表 6-3 数据集 1 各迭代次数解的情况

迭代次数	运行时间（s）	目标函数值	航迹距离（m）	校正次数	总体改善情况
30	16.74	299	112274	9	~
60	35.35	287	107130	8	4.18%
90	55.52	281	105621	8	2.14%
120	74.41	280	104960	8	0.357%

6.2.3 求解结果及分析

由于问题二增加了约束条件，限制了飞行器转向过程。为此本文分析了飞行器转向的各种情况，修改了问题一的模型，并在此基础上设计使用贪心算法求解初始化解，然后引入了集中和分散机制，提出并使用改进禁忌搜索算法优化初始解，求解结果汇总如下。

数据集 1 航迹规划路线如下图 6-9 所示，具体数据如下：

A→503→69→237→233→598→561→448→485→B，航迹长度（折线）为 104935m，航迹长度（添加转弯半径约束后）为 104960m，仅增加 0.02%；仅经过 8 个误差校正点。

表 6-4 数据集 1：飞行器经过的误差校正点编号与校正前后误差

校正点编号	校正前垂直误差	校正前水平误差	校正点类型	校正后垂直误差	校正后水平误差
0	0	0	出发点 A	~	~
503	13.397941415	13.397941415	1	0	13.397941415
69	8.808777111	22.206718527	0	8.808777111	0
237	12.500134919	21.308912031	1	0	12.500134919
233	10.825611465	23.325746385	0	10.825611465	0
598	24.832987479	14.007376014	1	0	14.007376014
561	10.956418231	24.963794246	0	10.956418231	0
448	16.112385647	5.155967415	1	0	16.11238564
485	21.848607361	5.736221714	0	5.736221714	0
621	23.571902802	29.308124516	B	23.571902802	29.308124516

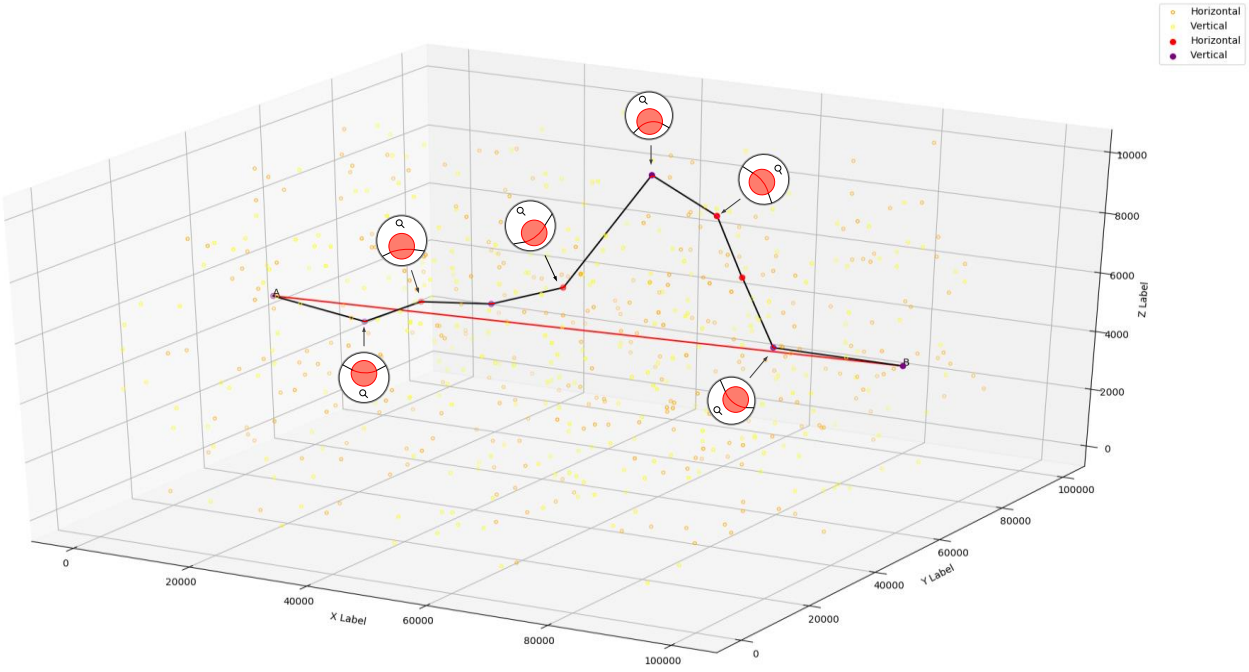

图 6-7 数据集 1 航迹 3D 图

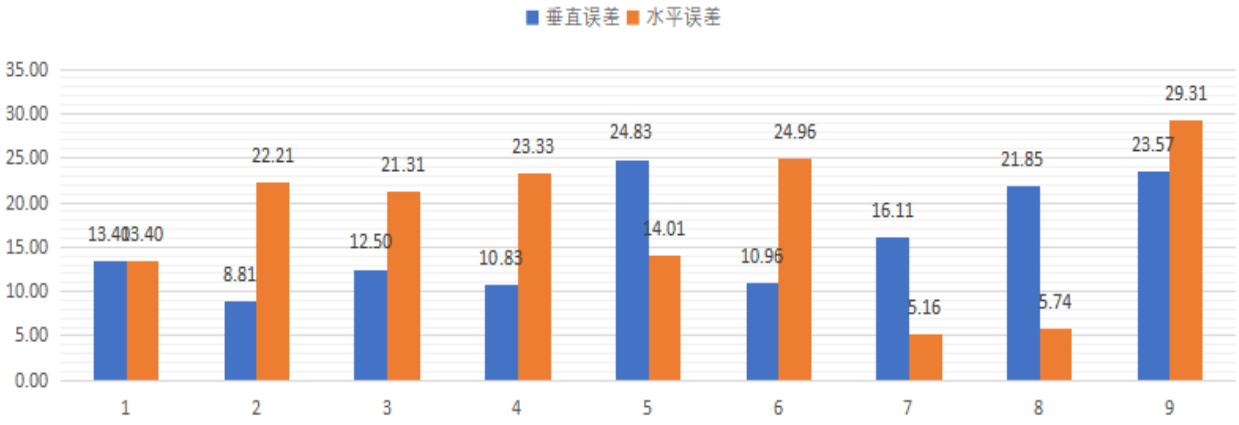

图 6-8 数据集 1：各个校正点误差分布图

结合数据表 6-4，由上柱状图表 6-10 可以直观地看出，飞行器到达垂直校正点时，垂直误差较大；飞行器到达水平校正点时，水平误差较大。误差都在约束条件之内尽可能大，且安排的校正点交替出现，校正点位置安排合理，能够在误差即将超出限制条件时及时校正。

航行轨迹沿着 AB 方向上下穿梭，安排较为合理，航行轨迹长度为算法最优解，航迹长度（添加转弯半径约束后）为 104960m，添加转弯半径约束后，较问题一折线方法的航迹长度（折线，104935m）增加了 0.02%；同时仅使用 8 个误差校正点，AB 直线距离为 100465m，航迹长度仅比 AB 直线距离增加了 4.47%。

数据集 2 航迹规划路线如下图 6-11 所示，具体数据如下：

A→163→114→8→309→305→123→45→160→92→93→61→292→B，航迹长度（折线）为 109342m，航迹长度（添加转弯半径约束后）为 109411m，仅增加 0.06%；同时，仅经过 12 个误差校正点。

表 6-5 数据集 2：飞行器经过的误差校正点编号与校正前后误差

校正点编号	校正前垂直误差	校正前水平误差	校正点类型	校正后垂直误差	校正后水平误差
0	0	0	出发点 A	~	~
163	13.288776817	13.288776817	0	13.288776817	0
114	18.625658222	5.336881405	1	0	5.336881405
8	13.947280827	19.284162232	0	13.947280827	0
309	19.474485409	5.527204582	1	0	5.527204582
305	5.971446855	11.498651438	0	5.971446855	0
123	15.181679994	9.210233138	1	0	9.210233138
45	10.006140911	19.216374049	0	10.006140911	0
160	17.492582326	7.486441415	1	0	7.486441415
92	5.780211496	13.266652911	0	5.780211496	0
93	15.265549745	9.485338249	1	0	9.485338249
61	9.839395830	19.324734080	0	19.324734080	0
292	16.410273686	6.570877855	1	0	6.570877855
326	6.961604102	13.532481958	B	6.9616041021	13.532481958

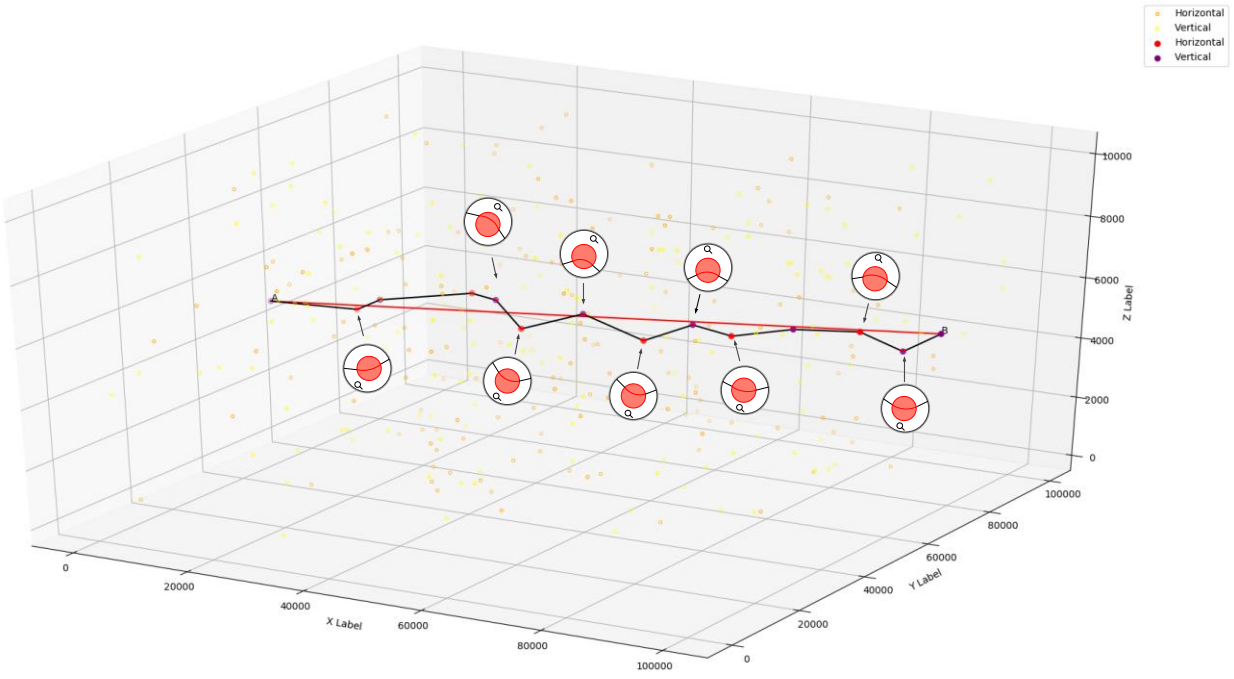

图 6-9 数据集 2 航迹 3D 图

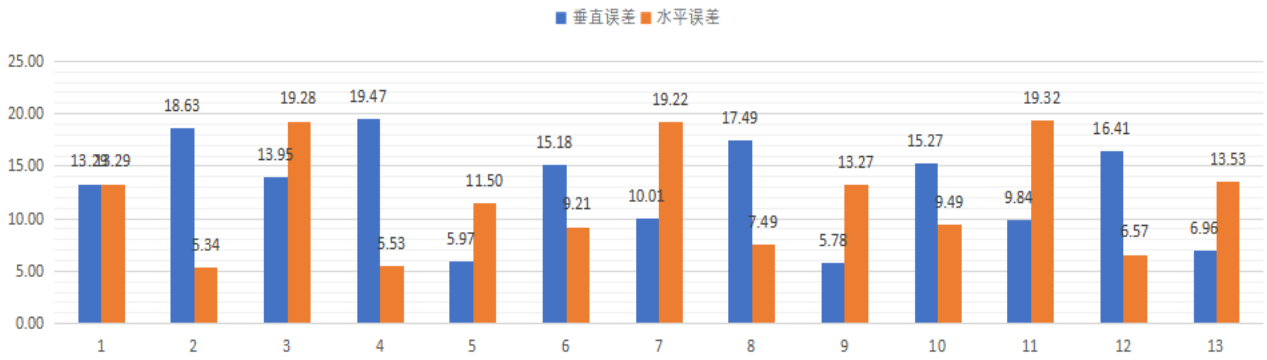

图 6-10 数据集 2：各个校正点误差分布图

结合数据表 6-5，由上柱状图表 6-10 可以直观地看出，飞行器到达垂直校正点时，垂直误差较大；飞行器到达水平校正点时，水平误差较大。误差都在约束条件之内尽可能大，且安排的校正点交替出现，校正点位置安排合理，能够在误差即将超出限制条件时及时校正。

航行轨迹沿着 AB 方向上下穿梭，安排较为合理，航行轨迹长度为算法最优解，航迹长度（添加转弯半径约束后）为 109411m，添加转弯半径约束后，较问题一折线方法的航迹长度（折线，109342m）增加了 0.06%；同时仅使用 12 个误差校正点，AB 直线距离为 103045m，航迹长度仅比 AB 直线距离增加了 6.17%。

6.3 模型评估

6.3.1 算法的有效性和复杂度

本问题仍是双目标规划问题，在本模型中，我们以航迹路线最小化、误差校正点总数最小化的优先级顺序作为主线，将双目标规划模型通过无量纲化处理和参考优先级顺序，引入线性加权系数，使复杂的多目标规划模型求解转换为单目标规划模型，极大降低了问题的求解难度。该算法在问题一算法的基础上加入了藐视准则，使一些禁忌区域中的一些优良状态被释放，保证了搜索的多样性，从而更能实现全局最优；采用动态变化的方式改进了禁忌长度，使禁忌长度的设置更加合理；引入了集中和分散机制，提出了一种改进禁忌搜索算法（ITS），一定程度上解决了禁忌搜索算法在求解优化问题时存在着的停滞现象，进一步提高了禁忌搜索算法的搜索质量和收敛速度。结果表明改进禁忌搜索算法在飞行器航迹规划这一问题的求解过程中展现了更好的寻优能力，具有更好的有效性。

改进禁忌搜索算法的时间复杂度低于 $O(\text{Max\_GEN}\times(n^2+n+l))$，在程序运行过程中，找到最优解的时间数据集 1 为 63.69s，数据集 2 为 50.42s。

6.3.2 灵敏度分析

本文针对问题二设计使用贪心算法求解初始化解，然后引入了集中和分散机制，提出并使用改进禁忌搜索算法优化初始解。在设计算法时，引入精英队列长度 s 参数。精英队列 EQ 的长度为 s，长度太小会导致解集太小，不利于跳出发生停滞现象的搜索区域；长度太大会导致程序占用的内存较大，运行时间较长，容易引起内部混淆，不利于算法收敛。因此，选取合理的精英队列长度 s 能够尽可能提升算法的收敛性能和有效性。故对进行精英队列长度 s 灵敏度分析。

分别取 $s\in[10,15,20,25,30,35,40]$，设置迭代次数为 60，通过改进禁忌搜索算法得出目标函数值及程序运行时间，汇总于下表 6-6：

表 6-6 不同精英队列长度 *s* 取值对目标函数和运行时间的影响

迭代次数	*s* 取值	目标函数值	运行时间
60	10	368	27.83s
	15	337	30.11s
	20	310	32.16s
	25	287	35.35s
	30	292	43.19s
	35	299	56.63s
	40	306	78.87s

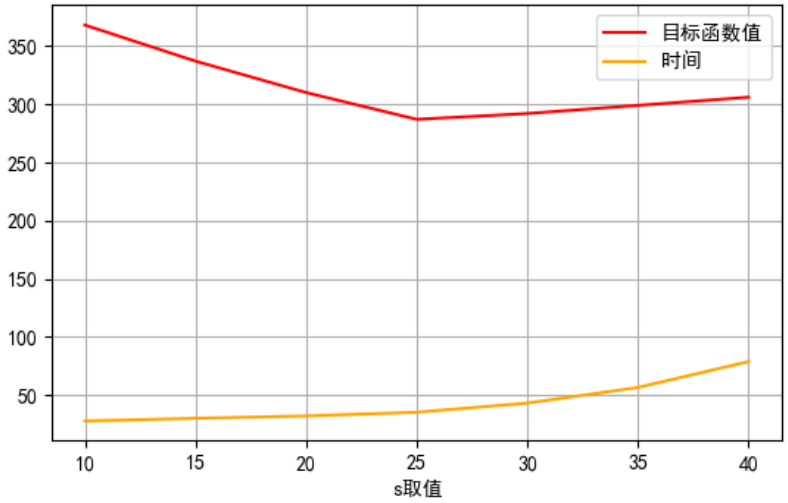

图 6-11 目标函数值和时间随 s 变化曲线

观察图 6-11 和表 6-6 可以得出，迭代 60 次后，*s* 取值在 25 附近，整体效果较为良好；若 *s* 取值过大，虽然目标函数整体较小，但时间开销较大；若 *s* 取值过校，时间开销大，但目标函数收敛性能较差。因此，本文选定 *s* 取值为 25。

7. 问题三：模型建立与求解

7.1 模型建立

7.1.1 低失败概率-短航迹-少校正次数多目标规划模型

优化目标：（1）飞行器从出发地 A 航行至目的地 B 的成功概率尽可能大，即可转化为失败概率尽可能小；（2）飞行器从出发地 A 航行至目的地 B 的航线距离尽可能短；（3）经过校正区域进行校正的次数尽可能少。

因此，定义飞行器无法成功到达目的地的概率为 p；定义航迹长度变量 *Distance*，即飞行器从出发地 A 航行至终点 B 所经过的总路程；定义经过矫正区域进行校正的次数 *Times*。

目标函数：采用少失败率-短航迹长度-少校正次数多目标优化模型，即同时优化失败概率，航迹长度和校正次三变量。目标函数可定义为：

$$\min z_5 = p \tag{7-1}$$

$$\min z_6 = Distance \tag{7-2}$$

$$\min z_7 = Times \tag{7-3}$$

决策变量：（1）定义决策变量 x_{ij}，当前校正点（或起点）i 的下一个最优校正点（或终点）j 是否连通，用以确定最优路径。

$$x_{ij} = \begin{cases} 0 & i点和j点没有连接 \\ 1 & i点和j点存在连接 \end{cases} \tag{7-4}$$

同时，使用向量 $\mathbf{x} = [x_{ij}^{(1)}, x_{ij}^{(2)}, ...x_{ij}^{(k)}...]$ 记录每一次的决策，其中 $x_{ij}^{(k)}$ 表示第 k 次选择连接的 i，j 两点，用向量 $\mathbf{x}$ 来记录最佳路径。

目标函数中的 *Distance* 和 *Times* 可以通过决策变量表示为：

$$\begin{aligned} Distance &= \sum_{i=0}^{N}\sum_{j=0}^{N} x_{ij} \times d_{ij} \\ Times &= \sum_{i=0}^{N}\sum_{j=0}^{N} x_{ij} \end{aligned} \tag{7-5}$$

其中 d_{ij} 表示 i 点与 j 点的欧式距离。

约束条件：（1）飞行器垂直校正条件：垂直定位误差 ε_v 小于 α_1，水平定位误差 ε_h 小于 α_2；（2）飞行器水平校正条件：垂直定位误差 ε_v 小于 β_1，水平定位误差 ε_h 小于 β_2；（3）飞行器到达目的地前，垂直误差和水平误差都应小于 θ；（4）飞行器在经过非理想校验点有概率会校验失败。

在探讨约束条件前，定义校正器类型变量T_i：

$$T_i=\begin{cases}0 & \text{第}i\text{个校正点是水平校正点}\\ 1 & \text{第}i\text{个校正点是垂直校正点}\end{cases} \tag{7-6}$$

定义理想校正器判别变量Q_i

$$Q_i=\begin{cases}0 & \text{第}i\text{个校正点是理想校正点}\\ 1 & \text{第}i\text{个校正点是理想校正点}\end{cases} \tag{7-7}$$

约束条件分析如下：

1）当飞行器从起点出发时，即 k=1 时，会选择一个合适的水平校正点或垂直校正点。两种情况的约束条件汇总如下表 7-1：

表 7-1 飞行器从起点选择校正点约束条件

情况	到达校正点时的误差	约束条件
选择水平校正点	$\varepsilon_h=\delta\times d_{0j}\times x_{0j}$	$\varepsilon_h\le\beta_2$
	$\varepsilon_v=\delta\times d_{0j}\times x_{0j}$	$\varepsilon_v\le\beta_1$
选择垂直校正点	$\varepsilon_h=\delta\times d_{0j}\times x_{0j}$	$\varepsilon_h\le\alpha_2$
	$\varepsilon_v=\delta\times d_{0j}\times x_{0j}$	$\varepsilon_v\le\alpha_1$

因此，选择合适的校正器类型变量T_i，当 k=1 时的约束条件可以表述为：

$$\begin{cases}k=1\\ \delta\times d_{0j}\times x_{0j}\times(1-T_i)\le\beta_1\\ \delta\times d_{0j}\times x_{0j}\times(1-T_i)\le\beta_2\\ \delta\times d_{0j}\times x_{0j}\times T_i\le\alpha_1\\ \delta\times d_{0j}\times x_{0j}\times T_i\le\alpha_2\end{cases} \tag{7-8}$$

2）当飞行器从第 i 个校正点出发时，即 k>1 时，会选择一个合适的水平校正点或垂直校正点。这里需要探讨可能出现的两种情况，即到达当前校正点时，上一次经过的校正点是否为理想校正点。若飞行器能在当前经过的校正点进行校正，其约束条件分别汇总如下表 7-2 和表 7-3：

表 7-2 飞行器从理想校正点选择下一校正点约束条件

情况		到达校正点时的误差	约束条件
选择水平校正点	上一次经过理想水平校正点	$\varepsilon_h=\delta\times\left(d_{ij}^{(k)}\times x_{ij}^{(k)}\right)$	$\varepsilon_h\le\beta_2,\theta$
		$\varepsilon_v=\delta\times\left(d_{ij}^{(k)}\times x_{ij}^{(k)}+d_{ij}^{(k-1)}\times x_{ij}^{(k-1)}\right)$	$\varepsilon_v\le\beta_1,\theta$

	上一次经过理想垂直校正点	$\varepsilon_h=\delta\times\left(d_{ij}^{(k)}\times x_{ij}^{(k)}+d_{ij}^{(k-1)}\times x_{ij}^{(k-1)}\right)$	$\varepsilon_h\le\beta_2,\theta$
		$\varepsilon_v=\delta\times\left(d_{ij}^{(k)}\times x_{ij}^{(k)}\right)$	$\varepsilon_v\le\beta_1,\theta$
选择垂直校正点	上一次经过理想水平校正点	$\varepsilon_h=\delta\times\left(d_{ij}^{(k)}\times x_{ij}^{(k)}+d_{ij}^{(k-1)}\times x_{ij}^{(k-1)}\right)$	$\varepsilon_h\le\alpha_2,\theta$
		$\varepsilon_v=\delta\times\left(d_{ij}^{(k)}\times x_{ij}^{(k)}\right)$	$\varepsilon_v\le\alpha_1,\theta$
	上一次经过理想垂直校正点	$\varepsilon_h=\delta\times\left(d_{ij}^{(k)}\times x_{ij}^{(k)}\right)$	$\varepsilon_h\le\alpha_2,\theta$
		$\varepsilon_v=\delta\times\left(d_{ij}^{(k)}\times x_{ij}^{(k)}+d_{ij}^{(k-1)}\times x_{ij}^{(k-1)}\right)$	$\varepsilon_v\le\alpha_1,\theta$

表 7-3　飞行器从非理想校正点选择下一校正点约束条件

情况		到达校正点时的误差	约束条件
选择水平校正点	上一次经过非理想水平校正点	$\varepsilon_h=\delta\times\left(d_{ij}^{(k)}\times x_{ij}^{(k)}\right)+5$	$\varepsilon_h\le\beta_2,\theta$
		$\varepsilon_v=\delta\times\left(d_{ij}^{(k)}\times x_{ij}^{(k)}+d_{ij}^{(k-1)}\times x_{ij}^{(k-1)}\right)$	$\varepsilon_v\le\beta_1,\theta$
	上一次经过非理想垂直校正点	$\varepsilon_h=\delta\times\left(d_{ij}^{(k)}\times x_{ij}^{(k)}+d_{ij}^{(k-1)}\times x_{ij}^{(k-1)}\right)$	$\varepsilon_h\le\beta_2,\theta$
		$\varepsilon_v=\delta\times\left(d_{ij}^{(k)}\times x_{ij}^{(k)}\right)+5$	$\varepsilon_v\le\beta_1,\theta$
选择垂直校正点	上一次经过非理想水平校正点	$\varepsilon_h=\delta\times\left(d_{ij}^{(k)}\times x_{ij}^{(k)}+d_{ij}^{(k-1)}\times x_{ij}^{(k-1)}\right)+5$	$\varepsilon_h\le\alpha_2,\theta$
		$\varepsilon_v=\delta\times\left(d_{ij}^{(k)}\times x_{ij}^{(k)}\right)$	$\varepsilon_v\le\alpha_1,\theta$
	上一次经过非理想垂直校正点	$\varepsilon_h=\delta\times\left(d_{ij}^{(k)}\times x_{ij}^{(k)}\right)$	$\varepsilon_h\le\alpha_2,\theta$
		$\varepsilon_v=\delta\times\left(d_{ij}^{(k)}\times x_{ij}^{(k)}+d_{ij}^{(k-1)}\times x_{ij}^{(k-1)}\right)+5$	$\varepsilon_v\le\alpha_1,\theta$

因此，选择合适的校正器类型变量T_i，当 $k>1$ 时的约束条件可以表述为：

$$
\begin{cases}
k>1 \\
\delta\left(\left(1-Q_i\right)\left(d_{ij}^{(k)}\times x_{ij}^{(k)}\times\left(1-T_i\right)+d_{ij}^{(k-1)}\times x_{ij}^{(k-1)}\times T_i\right)+Q_i\left(d_{ij}^{(k)}\times x_{ij}^{(k)}\times\left(1-T_i\right)+d_{ij}^{(k-1)}\times x_{ij}^{(k-1)}\times T+5_i\right)\right)\le\beta_1,\theta \\
\delta\left(\left(1-Q_i\right)\left(d_{ij}^{(k)}\times x_{ij}^{(k)}\times\left(1-T_i\right)+d_{ij}^{(k-1)}\times x_{ij}^{(k-1)}\times\left(1-T_i\right)\right)+Q_i\left(d_{ij}^{(k)}\times x_{ij}^{(k)}\times\left(1-T_i\right)+d_{ij}^{(k-1)}\times x_{ij}^{(k-1)}\times\left(1-T_i\right)+5\right)\right)\le\beta_2,\theta \\
\delta\left(\left(1-Q_i\right)\left(d_{ij}^{(k)}\times x_{ij}^{(k)}\times T_i+d_{ij}^{(k-1)}\times x_{ij}^{(k-1)}\times T_i\right)+Q_i\left(d_{ij}^{(k)}\times x_{ij}^{(k)}\times T_i+d_{ij}^{(k-1)}\times x_{ij}^{(k-1)}\times T+5_i\right)\right)\le\alpha_1,\theta \\
\delta\left(\left(1-Q_i\right)\left(d_{ij}^{(k)}\times x_{ij}^{(k)}\times T_i+d_{ij}^{(k-1)}\times x_{ij}^{(k-1)}\times\left(1-T_i\right)\right)+Q_i\left(d_{ij}^{(k)}\times x_{ij}^{(k)}\times T_i+d_{ij}^{(k-1)}\times x_{ij}^{(k-1)}\times\left(1-T_i\right)+5\right)\right)\le\alpha_2,\theta
\end{cases}
\tag{7-9}
$$

3）决策变量的约束条件。由于定义的决策变量 x_{ij} 的目的是寻找一条最优路径，即 x_{ij} 构成的邻接矩阵是一个有向无环图，且不是一种树结构。这意味着每个校正点最多只能经过一次，且必须保证从起点到终点是一条通路。

首先应当约束邻接矩阵每一列的和小于等于 1，这样可以确保每个校正点最多经过一次；

其次需要约束起点到终点是一条通路。本文使用向量 $\mathbf{x}=[x_{ij}^{(1)},x_{ij}^{(2)},...x_{ij}^{(k)}...]$ 记录每一次的决策，其中 $x_{ij}^{(k)}$ 表示第 k 次选择连接的 i，j 两点，展开为 $x_{ij}^{(k)}(i_k,j_k)$。应当使第 k 次选择的点 $x_{ij}^{(k)}(i_k,j_k)$ 纵坐标与第 k+1 次选择的点 $x_{ij}^{(k)}(i_k,j_k)$ 横坐标相等，即 $j_k=i_{k+1}$。

因此，决策变量的约束条件可以表述为：

$$\begin{cases}\sum_{j=0}^{N}x_{ij}\le 1\\ j_k=i_{k+1}\\ i,j\in(0,1,2,...N)\end{cases}\tag{7-10}$$

4）概率约束条件。分析飞行器成功到达概率 p 情况：飞行器在经过非理想校正点时，可能出现校正失败的情况，其分布律如下表 7-1：

表 7-1 非理想校正点校正情况分布律

X	**校正成功**	**校正失败**
概率	p_s	1-p_s

一旦校正失败，飞行器航行仍不一定会失败。根据条件，误差会变化为 5 个单位，即飞行器仍然有机会航行至下一个校正点。

设定判决变量 η_i，判决飞行器能够航行至下一校正点：

$$\eta_i=\begin{cases}0 & \text{飞行器能够航行至下一校正点}\\ 1 & \text{飞行器不能航行至下一校正点}\end{cases}\tag{7-11}$$

飞行器在非理想校正点校正失败后，影响飞行器成功航行至下一个校正点的因素受限于距离，根据上文分析，假定垂直校验失败，首先计算到达下一个垂直校验点所经过的路程，通过飞行器到达校正点时误差阈值判别决定判决变量 η_i 的取值。所有情况汇总如表 7-4 所示：

表 7-4 各种情况 η_i 的取值分析

校验失败点类型	**后续校验点安排顺序**	**到达下一个同类校验点时的误差**		**判决条件**	**η_i 取值**
水平	先水平	水平误差	$\varepsilon_h=\delta\times\left(d_{ij}^{(k)}\times x_{ij}^{(k)}\right)+5$	$\le\beta_2$	1（否则为 0）
		垂直误差	$\varepsilon_v=\delta\times\left(d_{ij}^{(k)}\times x_{ij}^{(k)}+d_{ij}^{(k-1)}\times x_{ij}^{(k-1)}\right)$	$\le\beta_1$	

	先垂直后水平	水平误差	$\varepsilon_h=\delta\times\left(d_{ij}^{(k)}\times x_{ij}^{(k)}+d_{ij}^{(k-1)}\times x_{ij}^{(k-1)}\right)$	$\le\beta_2$	1（否则为0）
		垂直误差	$\varepsilon_v=\delta\times\left(d_{ij}^{(k)}\times x_{ij}^{(k)}\right)+5$	$\le\beta_1$	
垂直	先水平	水平误差	$\varepsilon_h=\delta\times\left(d_{ij}^{(k)}\times x_{ij}^{(k)}+d_{ij}^{(k-1)}\times x_{ij}^{(k-1)}\right)+5$	$\le\alpha_2$	1（否则为0）
		垂直误差	$\varepsilon_v=\delta\times\left(d_{ij}^{(k)}\times x_{ij}^{(k)}\right)$	$\le\alpha_1$	
	先垂直后水平	水平误差	$\varepsilon_h=\delta\times\left(d_{ij}^{(k)}\times x_{ij}^{(k)}\right)$	$\le\alpha_2$	1（否则为0）
		垂直误差	$\varepsilon_v=\delta\times\left(d_{ij}^{(k)}\times x_{ij}^{(k)}+d_{ij}^{(k-1)}\times x_{ij}^{(k-1)}\right)+5$	$\le\alpha_1$	

根据表 7-4 可知，η_i 的取值情况如下：

$$\eta_i=\begin{cases}0 & \text{不满足下述条件}\\ 1 & \begin{cases}k>1\\ \delta\left((1-Q_i)\left(d_{ij}^{(k)}\times x_{ij}^{(k)}\times(1-\mathrm{T}_i)+d_{ij}^{(k-1)}\times x_{ij}^{(k-1)}\times\mathrm{T}_i\right)+Q_i\left(d_{ij}^{(k)}\times x_{ij}^{(k)}\times(1-\mathrm{T}_i)+d_{ij}^{(k-1)}\times x_{ij}^{(k-1)}\times\mathrm{T}+5_i\right)\right)\le\beta_1,\theta\\ \delta\left((1-Q_i)\left(d_{ij}^{(k)}\times x_{ij}^{(k)}\times(1-\mathrm{T}_i)+d_{ij}^{(k-1)}\times x_{ij}^{(k-1)}\times(1-\mathrm{T}_i)\right)+Q_i\left(d_{ij}^{(k)}\times x_{ij}^{(k)}\times(1-\mathrm{T}_i)+d_{ij}^{(k-1)}\times x_{ij}^{(k-1)}\times(1-\mathrm{T}_i)+5\right)\right)\le\beta_2,\theta\\ \delta\left((1-Q_i)\left(d_{ij}^{(k)}\times x_{ij}^{(k)}\times\mathrm{T}_i+d_{ij}^{(k-1)}\times x_{ij}^{(k-1)}\times\mathrm{T}_i\right)+Q_i\left(d_{ij}^{(k)}\times x_{ij}^{(k)}\times\mathrm{T}_i+d_{ij}^{(k-1)}\times x_{ij}^{(k-1)}\times\mathrm{T}+5_i\right)\right)\le\alpha_1,\theta\\ \delta\left((1-Q_i)\left(d_{ij}^{(k)}\times x_{ij}^{(k)}\times\mathrm{T}_i+d_{ij}^{(k-1)}\times x_{ij}^{(k-1)}\times(1-\mathrm{T}_i)\right)+Q_i\left(d_{ij}^{(k)}\times x_{ij}^{(k)}\times\mathrm{T}_i+d_{ij}^{(k-1)}\times x_{ij}^{(k-1)}\times(1-\mathrm{T}_i)+5\right)\right)\le\alpha_2,\theta\end{cases}\end{cases}$$

(7-12)

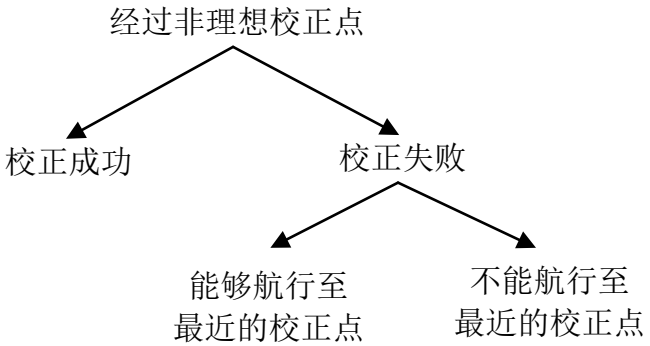

图 7-1 飞行器经过非理想校正点后各类情况分析

由图 7-1 可以得到，飞行器经过 1 个非理想校正点，导致航行失败的概率为：

$$(1-p_s)\times(1-\eta_i) \tag{7-13}$$

由此可以推导出，飞行器经过 n 个非理想校正点，导致航行失败的概率为：

$$p=\left(1-\prod_{i=1}^{n}(1-(1-p_s)\times(1-\eta_i))\right) \tag{7-14}$$

综上所述，所建立的双目标优化模型可以表述为：

$$\min z_5 = p$$

$$\min z_6 = Distance = \sum_{i=0}^{N}\sum_{j=0}^{N} x_{ij} \times d_{ij}$$

$$\min z_7 = Times = \sum_{i=0}^{N}\sum_{j=0}^{N} x_{ij}$$

$$
s.t.\begin{cases}
\begin{cases} k=1 \\ \delta \times d_{0j} \times x_{0j} \times (1-T_i) \le \beta_1 \\ \delta \times d_{0j} \times x_{0j} \times (1-T_i) \le \beta_2 \\ \delta \times d_{0j} \times x_{0j} \times T_i \le \alpha_1 \\ \delta \times d_{0j} \times x_{0j} \times T_i \le \alpha_2 \end{cases} \\
\begin{cases} k>1 \\ \delta\left((1-Q_i)\left(d_{ij}^{(k)} \times x_{ij}^{(k)} \times (1-T_i) + d_{ij}^{(k-1)} \times x_{ij}^{(k-1)} \times T_i\right) + Q_i\left(d_{ij}^{(k)} \times x_{ij}^{(k)} \times (1-T_i) + d_{ij}^{(k-1)} \times x_{ij}^{(k-1)} \times T + 5_i\right)\right) \le \beta_1, \theta \\ \delta\left((1-Q_i)\left(d_{ij}^{(k)} \times x_{ij}^{(k)} \times (1-T_i) + d_{ij}^{(k-1)} \times x_{ij}^{(k-1)} \times (1-T_i)\right) + Q_i\left(d_{ij}^{(k)} \times x_{ij}^{(k)} \times (1-T_i) + d_{ij}^{(k-1)} \times x_{ij}^{(k-1)} \times (1-T_i) + 5\right)\right) \le \beta_2, \theta \\ \delta\left((1-Q_i)\left(d_{ij}^{(k)} \times x_{ij}^{(k)} \times T_i + d_{ij}^{(k-1)} \times x_{ij}^{(k-1)} \times T_i\right) + Q_i\left(d_{ij}^{(k)} \times x_{ij}^{(k)} \times T_i + d_{ij}^{(k-1)} \times x_{ij}^{(k-1)} \times T + 5_i\right)\right) \le \alpha_1, \theta \\ \delta\left((1-Q_i)\left(d_{ij}^{(k)} \times x_{ij}^{(k)} \times T_i + d_{ij}^{(k-1)} \times x_{ij}^{(k-1)} \times (1-T_i)\right) + Q_i\left(d_{ij}^{(k)} \times x_{ij}^{(k)} \times T_i + d_{ij}^{(k-1)} \times x_{ij}^{(k-1)} \times (1-T_i) + 5\right)\right) \le \alpha_2, \theta \end{cases} \\
\begin{cases} \sum_{j=0}^{N} x_{ij} \le 1 \\ j_k = i_{k+1} \\ i, j \in (0,1,2,\ldots N) \end{cases} \\
\alpha_1, \alpha_2, \beta_1, \beta_2, \delta > 0 \\
d_{ij}^{(k)} = \left\| \overline{i^{(k)} j^{(k)}} \right\| \\
p = \left(1 - \prod_{i=1}^{n} (1 - (1-p_s) \times (1-\eta_i))\right) \\
\eta_i = \begin{cases} 0 & \text{不满足下述条件} \\ 1 & \begin{cases} k>1 \\ \delta\left((1-Q_i)\left(d_{ij}^{(k)} \times x_{ij}^{(k)} \times (1-T_i) + d_{ij}^{(k-1)} \times x_{ij}^{(k-1)} \times T_i\right) + Q_i\left(d_{ij}^{(k)} \times x_{ij}^{(k)} \times (1-T_i) + d_{ij}^{(k-1)} \times x_{ij}^{(k-1)} \times T + 5_i\right)\right) \le \beta_1, \theta \\ \delta\left((1-Q_i)\left(d_{ij}^{(k)} \times x_{ij}^{(k)} \times (1-T_i) + d_{ij}^{(k-1)} \times x_{ij}^{(k-1)} \times (1-T_i)\right) + Q_i\left(d_{ij}^{(k)} \times x_{ij}^{(k)} \times (1-T_i) + d_{ij}^{(k-1)} \times x_{ij}^{(k-1)} \times (1-T_i) + 5\right)\right) \le \beta_2, \theta \\ \delta\left((1-Q_i)\left(d_{ij}^{(k)} \times x_{ij}^{(k)} \times T_i + d_{ij}^{(k-1)} \times x_{ij}^{(k-1)} \times T_i\right) + Q_i\left(d_{ij}^{(k)} \times x_{ij}^{(k)} \times T_i + d_{ij}^{(k-1)} \times x_{ij}^{(k-1)} \times T + 5_i\right)\right) \le \alpha_1, \theta \\ \delta\left((1-Q_i)\left(d_{ij}^{(k)} \times x_{ij}^{(k)} \times T_i + d_{ij}^{(k-1)} \times x_{ij}^{(k-1)} \times (1-T_i)\right) + Q_i\left(d_{ij}^{(k)} \times x_{ij}^{(k)} \times T_i + d_{ij}^{(k-1)} \times x_{ij}^{(k-1)} \times (1-T_i) + 5\right)\right) \le \alpha_2, \theta \end{cases} \end{cases}
\end{cases}
$$

(7-15)

7.2 模型求解

7.2.1 多目标转化为单目标

在本问题中，由航行失败概率 p，最小化航迹变量 *Distance* 和最小化经过矫正区域进行校正的次数 *Times* 建立了三个目标函数，选取三个值 $0<\zeta_i<1 \quad (i=1,2,3)$，作为这三个

目标的线性加权系数，并且$\sum_i \zeta_i = 1$。由于本问题中三个目标函数的量纲不同，因此需要归一化目标函数，对其进行无量纲化处理。此时目标函数分别通过函数$\alpha_{z_6} = \dfrac{z_6 - \min z_6}{\max z_6 - \min z_6}$，$\alpha_{z_7} = \dfrac{z_7 - \min z_7}{\max z_7 - \min z_7}$，规范目标函数$z_5$本身无单位无需处理。此时，由于目标函数归一化导致整个目标函数缩小至变化细微，因此需要乘以 100，这样迭代时目标函数区分明显。该问题可转化为单目标数学规划问题，转化后的单目标函数可以写为：

$$\min z_{z_3} = (\zeta_1 p + \zeta_2 \alpha_{z_6} + \zeta_3 \alpha_{z_7}) \times 100 \tag{7-16}$$

因此，模型可以表述为：

$$\min z_{z_3} = (\zeta_1 p + \zeta_2 \alpha_{z_6} + \zeta_3 \alpha_{z_7}) \times 100$$

$$
\begin{cases}
\begin{cases}
k = 1 \\
\delta \times d_{0j} \times x_{0j} \times (1 - T_i) \le \beta_1 \\
\delta \times d_{0j} \times x_{0j} \times (1 - T_i) \le \beta_2 \\
\delta \times d_{0j} \times x_{0j} \times T_i \le \alpha_1 \\
\delta \times d_{0j} \times x_{0j} \times T_i \le \alpha_2
\end{cases} \\
\begin{cases}
k > 1 \\
\delta\left((1-Q_i)\left(d_{ij}^{(k)} \times x_{ij}^{(k)} \times (1-T_i) + d_{ij}^{(k-1)} \times x_{ij}^{(k-1)} \times T_i\right) + Q_i\left(d_{ij}^{(k)} \times x_{ij}^{(k)} \times (1-T_i) + d_{ij}^{(k-1)} \times x_{ij}^{(k-1)} \times T + 5_i\right)\right) \le \beta_1, \theta \\
\delta\left((1-Q_i)\left(d_{ij}^{(k)} \times x_{ij}^{(k)} \times (1-T_i) + d_{ij}^{(k-1)} \times x_{ij}^{(k-1)} \times (1-T_i)\right) + Q_i\left(d_{ij}^{(k)} \times x_{ij}^{(k)} \times (1-T_i) + d_{ij}^{(k-1)} \times x_{ij}^{(k-1)} \times (1-T_i) + 5\right)\right) \le \beta_2, \theta \\
\delta\left((1-Q_i)\left(d_{ij}^{(k)} \times x_{ij}^{(k)} \times T_i + d_{ij}^{(k-1)} \times x_{ij}^{(k-1)} \times T_i\right) + Q_i\left(d_{ij}^{(k)} \times x_{ij}^{(k)} \times T_i + d_{ij}^{(k-1)} \times x_{ij}^{(k-1)} \times T + 5_i\right)\right) \le \alpha_1, \theta \\
\delta\left((1-Q_i)\left(d_{ij}^{(k)} \times x_{ij}^{(k)} \times T_i + d_{ij}^{(k-1)} \times x_{ij}^{(k-1)} \times (1-T_i)\right) + Q_i\left(d_{ij}^{(k)} \times x_{ij}^{(k)} \times T_i + d_{ij}^{(k-1)} \times x_{ij}^{(k-1)} \times (1-T_i) + 5\right)\right) \le \alpha_2, \theta
\end{cases} \\
s.t. \begin{cases}
\sum_{j=0}^{N} x_{ij} \le 1 \\
j_k = i_{k+1} \\
i, j \in (0,1,2,\ldots N)
\end{cases} \\
\alpha_1, \alpha_2, \beta_1, \beta_2, \delta > 0 \\
d_{ij}^{(k)} = \left\| \overrightarrow{i^{(k)} j^{(k)}} \right\| \\
p = \left(1 - \prod_{i=1}^{n} (1 - (1 - p_s) \times (1 - \eta_i))\right) \\
\eta_i = \begin{cases}
0 & \text{不满足下述条件} \\
1 & \begin{cases}
k > 1 \\
\delta\left((1-Q_i)\left(d_{ij}^{(k)} \times x_{ij}^{(k)} \times (1-T_i) + d_{ij}^{(k-1)} \times x_{ij}^{(k-1)} \times T_i\right) + Q_i\left(d_{ij}^{(k)} \times x_{ij}^{(k)} \times (1-T_i) + d_{ij}^{(k-1)} \times x_{ij}^{(k-1)} \times T + 5_i\right)\right) \le \beta_1, \theta \\
\delta\left((1-Q_i)\left(d_{ij}^{(k)} \times x_{ij}^{(k)} \times (1-T_i) + d_{ij}^{(k-1)} \times x_{ij}^{(k-1)} \times (1-T_i)\right) + Q_i\left(d_{ij}^{(k)} \times x_{ij}^{(k)} \times (1-T_i) + d_{ij}^{(k-1)} \times x_{ij}^{(k-1)} \times (1-T_i) + 5\right)\right) \le \beta_2, \theta \\
\delta\left((1-Q_i)\left(d_{ij}^{(k)} \times x_{ij}^{(k)} \times T_i + d_{ij}^{(k-1)} \times x_{ij}^{(k-1)} \times T_i\right) + Q_i\left(d_{ij}^{(k)} \times x_{ij}^{(k)} \times T_i + d_{ij}^{(k-1)} \times x_{ij}^{(k-1)} \times T + 5_i\right)\right) \le \alpha_1, \theta \\
\delta\left((1-Q_i)\left(d_{ij}^{(k)} \times x_{ij}^{(k)} \times T_i + d_{ij}^{(k-1)} \times x_{ij}^{(k-1)} \times (1-T_i)\right) + Q_i\left(d_{ij}^{(k)} \times x_{ij}^{(k)} \times T_i + d_{ij}^{(k-1)} \times x_{ij}^{(k-1)} \times (1-T_i) + 5\right)\right) \le \alpha_2, \theta
\end{cases}
\end{cases}
\end{cases}
$$

(7-17)

接下来便转化为单目标的禁忌搜索算法求解目标函数。

7.2.2 基于单目标规划的遗传禁忌搜索算法（GATS）

传统遗传算法是由选择、交叉和变异 3 个操作算子来进行的，由于局部搜索能力差等缺陷使传统遗传算法难以收敛到全局最优解。禁忌搜索算法是从局部搜索发展而来的，是一种全局逐步寻优算法。禁忌搜索法的基本思想是在搜索过程中将近期的历史搜索存放在禁忌表中，有效避免了算法陷入循环和局部最优解。禁忌搜索算法中的蔑视准则可以使算法在候选集中搜索到更优解，并以此替代历史最优解。

基于上述说明，在本问题中将局部搜索能力强的禁忌搜索算法与遗传算法结合，得到遗传禁忌搜索算法。禁忌搜索作为遗传算法的选择操作可以得到每一代的最优个体，并且通过遗传算法中的变异算子可以提高禁忌算法中解空间的多样性。下面为遗传禁忌搜索算法步骤，该流程图见图 7-2：

遗传禁忌搜索算法步骤

Step1：通过贪心算法得到初始解；
Step2：判断适应值，如果满足终止条件则终止。否则通过变异操作生成下一代种群（此处的变异操作指交叉变异和插入变异）；
Step3：用禁忌搜索操作作为选择算子，通过精英选择得到候选集，从初始解邻域中选出最优的 m 个个体构成禁忌搜索候选集；
Step4：选出候选集中的最优解 χ，进行禁忌表判断；
Step5：重复 Step2 中操作，直到满足终止条件（算法终止条件：算法达到了最大迭代次数）。

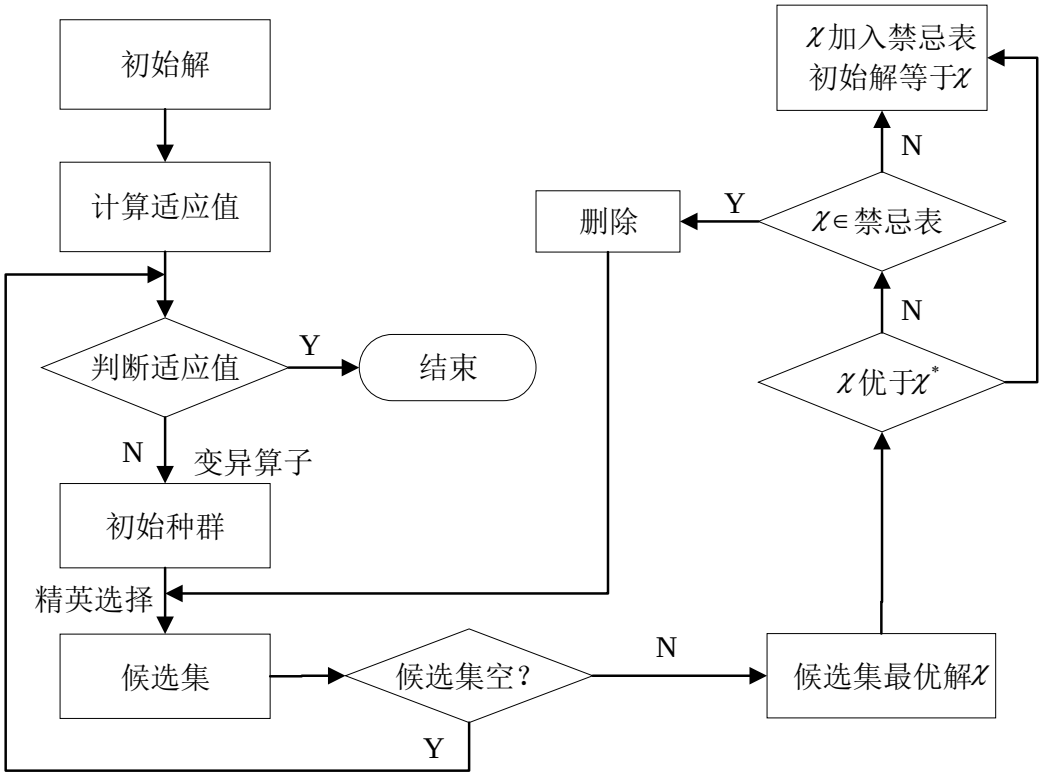

图 7-2 遗传禁忌搜索算法流程图

目标解的迭代

通过分析表 7-5 中数据，可以看到迭代次数在 120 次以上时，解的改善情况变化不大，可以认为在 120 次以后能得到较优解。

表 7-5 数据集 1 各迭代次数解的情况

迭代次数	运行时间（s）	目标函数值	航迹距离（m）	校正次数	总体改善情况
30	25.52	329	112887	13	~
60	46.32	318	109420	12	3.46%
90	65.71	312	108726	12	1.92%
120	87.86	310	108439	12	0.645%

7.2.3 求解结果及分析

问题三的难点在于某些校正点存在失效可能，但如果需要寻找一条路程较短的航行路径，不得不经过这些非理想校正点。为此本文根据非理想校正点的失效概率和整体失效概率，结合航迹长度、校正点个数建立了多目标优化模型，并设计了遗传禁忌搜索算法，求解出给定数据集 1 和数据集 2 的最优航行轨迹及成功抵达终点的概率。

数据集 1 最终得到的航迹规划路线为：

$$A \to 578 \to 417 \to 80 \to 237 \to 607 \to 33 \to 194 \to 450 \to 448 \to 485 \to 302 \to 612 \to B$$

上述航迹规划路线参考图 7-3，其航迹长度为 108439m，误差校正点 12 点，成功抵达终点概率 100%。

由于航迹规划路线中存在部分非理想校正点，会导致出现多种情况（校正成功或校正失败），因此本文选取了其中一种校正情况，汇总于下表 7-6 中：

表 7-6 数据集 1 结果（选取一种情况）

校正点编号	校正前垂直误差	校正前水平误差	校正点类型	校正后垂直误差	校正后水平误差
0	0	0	出发点 A	~	~
578	12.010738484	12.010738484	02	12.010738484	5
417	19.975355029	12.964616544	12	5	12.964616544
80	17.003730055	24.968346599	02	17.003730055	5
237	21.631474624	9.627744569	12	5	9.6277445696
607	10.187559566	14.815304136	01	10.187559566	0
33	22.109964838	11.922405271	11	0	11.922405271
194	10.502849793	22.425255064	01	10.502849793	0
450	16.479994043	5.977144250	11	0	5.977144250
448	8.710476166	14.687620416	01	8.7104761664	0
485	14.442679870	5.732203704	11	0	5.732203704
302	17.556085657	23.288289361	02	17.556085657	5
612	23.800045131	11.243959473	B 点	23.800045131	11.243959473

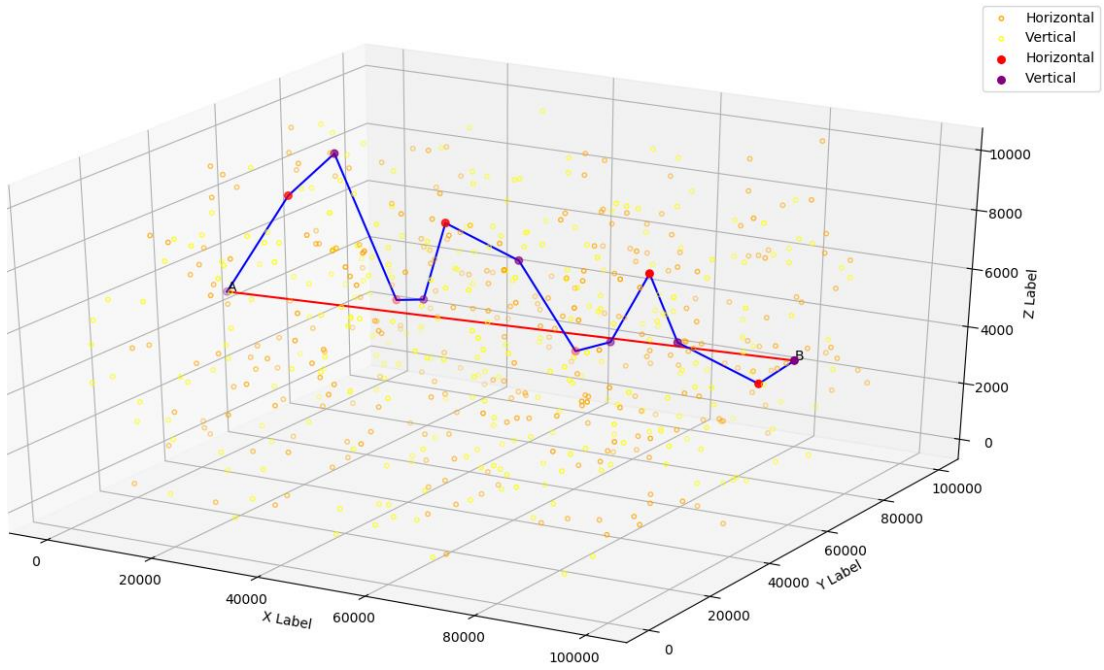

图 7-3 问题三数据集 1 航迹规划路线

结合数据表 7-6，由下柱状图表 7-4 和饼图 7-5 可以直观地看出，飞行器到达垂直校正点时，垂直误差较大；飞行器到达水平校正点时，水平误差较大。同时，考虑到航迹规划路线中存在非理想校正点，飞行器到达非理想校正点后会保证限制条件内的误差有一定的冗余，以防止出现校正失败的情况。所有误差都在约束条件之内尽可能大，且安排的校正点交替出现，校正点位置安排合理，能够在误差即将超出限制条件时及时校正。

航行轨迹沿着 AB 方向上下穿梭，安排较为合理，航行轨迹长度为算法最优解，航迹长度 108439m，使用 12 个误差校正点，且成功抵达终点概率 100%。AB 直线距离为 100465m，航迹长度仅比 AB 直线距离增加了 7.93%。

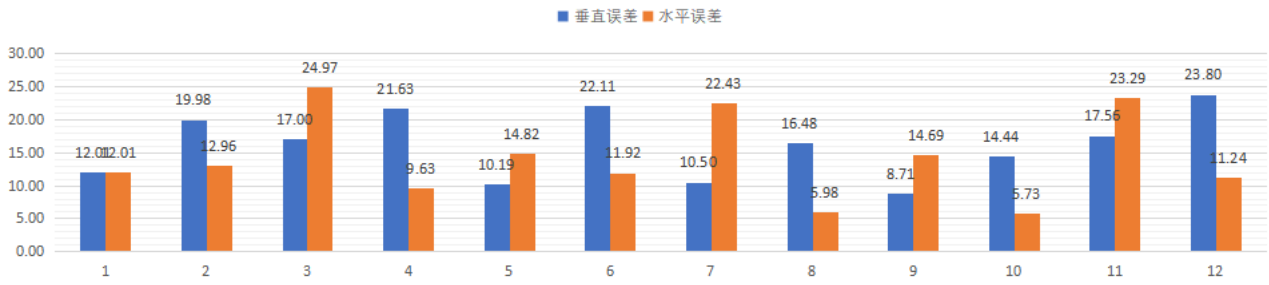

图 7-4 问题三数据集 1 垂直/水平误差变化图

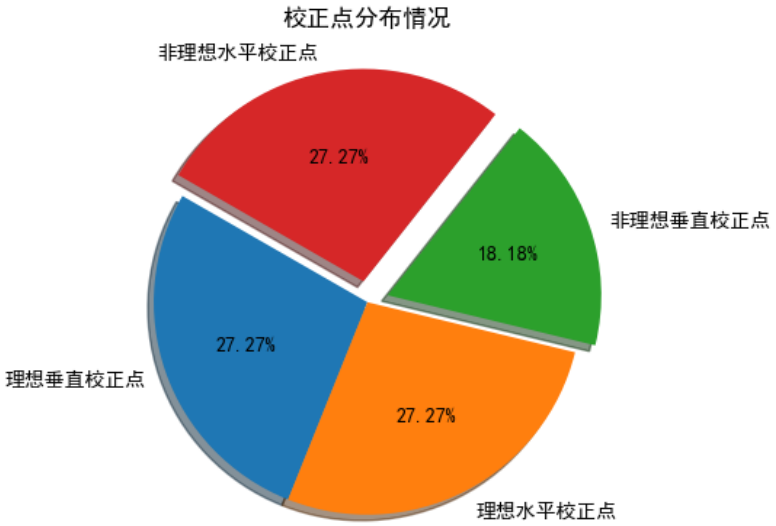

图 7-5 问题三数据集 1 航迹规划路线中校正点分布情况

为了进一步说明本文针对问题三所设计的算法求解出的结果是最优解，我们选取了部分数据集 1 其他解果，参考下表 7-7：

表 7-7 数据集 1 中不同校正点个数下的结果

校正点个数	航迹路径	航迹长度	失败概率
11	[0, 285, 162, 55, 234, 42, 316, 312, 379, 284, 273, 370, 612]	121975m	0.00%
	[0, 285, 162, 55, 234, 192, 70, 171, 312, 198, 513, 277, 612]	119272m	5.02%
12	[0, 578, 417, 80, 237, 607, 33, 194, 450, 448, 397, 612]	108819m	0.00%
	[0, 578, 417, 80, 237, 607, 89, 11, 403, 3, 501, 612]	108894m	0.00%
13	[0, 285, 162, 55, 234, 42, 316, 312, 538, 556, 424, 555, 436, 18, 612]	120765m	3.21%
	[0, 285, 162, 55, 234, 42, 316, 499, 312, 96, 476, 520, 59, 425, 612]	122443m	0.00%

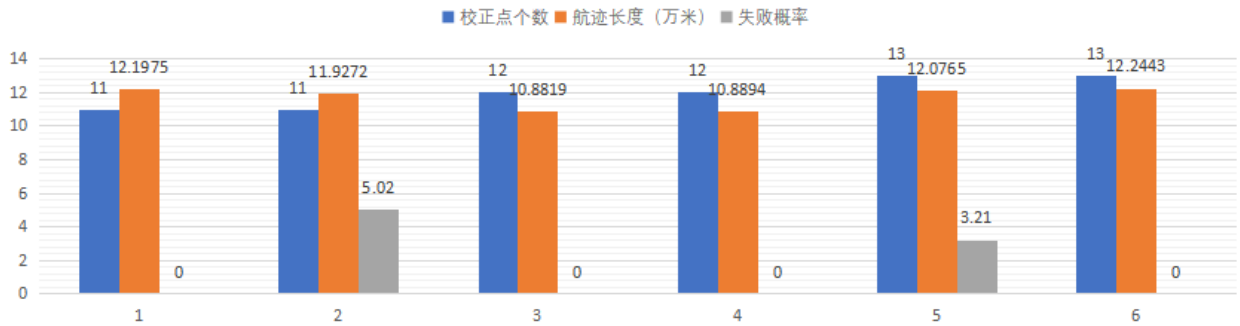

图 7-6 校正点个数及航迹长度对比柱状图

由表 7-7 和图 7-6 可以观察到，在我们选取的部分结果中，校正点个数为 12 时，航迹长度明显小于其他校正点数的结果，失败概率明显小于其他校正点数的结果。可以从这个

角度说明本文求解出的 12 个校正点，航迹长度 108439m 为算法的最优解。

数据集 2 最终得到的航迹规划路线为：

$$A \to 169 \to 322 \to 100 \to 137 \to 194 \to 190 \to 296 \to 250 \to 243 \to 73 \to 82 \to 44 \to 211 \to 321 \to 279 \to 301 \to 38 \to 287 \to 99 \to 326 \to B$$

上述航迹规划路线参考图 7-7，其航迹长度为 147271m，误差校正点 19 点，成功抵达终点概率 65.01%。

由于航迹规划路线中存在部分非理想校正点，会导致出现多种情况（校正成功或校正失败），因此本文选取了其中一种校正情况，汇总于下表 7-8 中：

表 7-8 数据集 2 结果（选取一种情况）

校正点编号	校正前垂直误差	校正前水平误差	校正点类型	校正后垂直误差	校正后水平误差
0	0	0	出发点 A	~	~
169	9.270541364	9.270541364	01	9.270541364	0
322	13.41862841	4.148087043	11	0	4.148087043
100	11.18279525	15.33088229	01	11.18279525	0
137	17.01660766	5.833812419	11	0	5.833812419
194	9.881899182	15.7157116	01	9.881899182	0
190	16.85883149	6.976932307	12	5	6.976932307
296	12.1672607	14.14419301	01	12.1672607	0
250	19.25832871	7.091068004	11	0	7.091068005
243	6.958944328	14.05001233	01	6.958944328	0
73	10.50177678	3.542832454	11	0	3.542832454
82	5.731596685	9.274429139	01	5.731596685	0
44	14.67397936	8.942382674	11	0	8.942382674
211	5.49743914	14.43982181	01	5.49743914	0
321	13.24784132	7.750402177	11	0	7.750402177
279	10.35893177	18.10933395	02	10.35893177	5
301	15.3120715	9.95313973	11	0	9.95313973
38	9.872129547	19.82526928	01	9.872129547	0
287	13.83741332	3.965283777	11	0	3.965283777
99	9.496407885	13.46169166	01	9.496407885	0
326	18.1460027	8.649594817	B 点	18.1460027	8.649594817

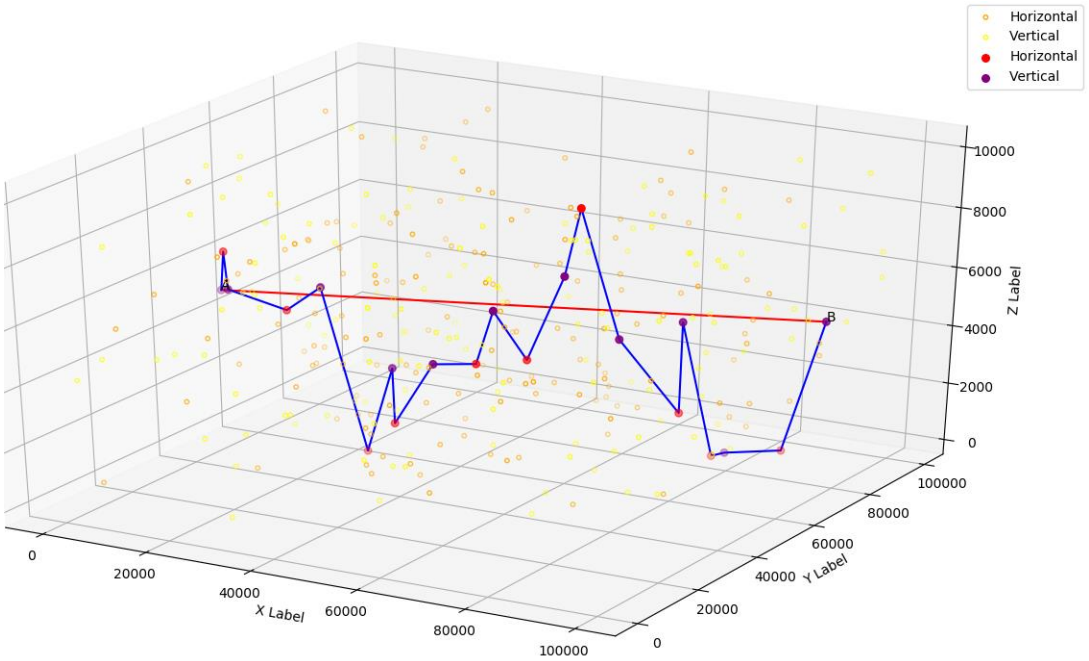

图 7-7 问题三数据集 2 航迹规划路线

结合数据表 7-8，由下柱状图表 7-8 和饼图 7-9 可以直观地看出，飞行器到达垂直校正点时，垂直误差较大；飞行器到达水平校正点时，水平误差较大。同时，考虑到航迹规划路线中存在非理想校正点，飞行器到达非理想校正点后会保证限制条件内的误差有一定的冗余，以防止出现校正失败的情况。所有误差都在约束条件之内尽可能大，且安排的校正点交替出现，校正点位置安排合理，能够在误差即将超出限制条件时及时校正。

航行轨迹沿着 AB 方向上下穿梭，安排较为合理，航行轨迹长度为算法最优解，航迹长度 147271m，使用 19 个误差校正点，且成功抵达终点概率 65.01%。AB 直线距离为 103045m，航迹长度比 AB 直线距离增加了 42.91%。

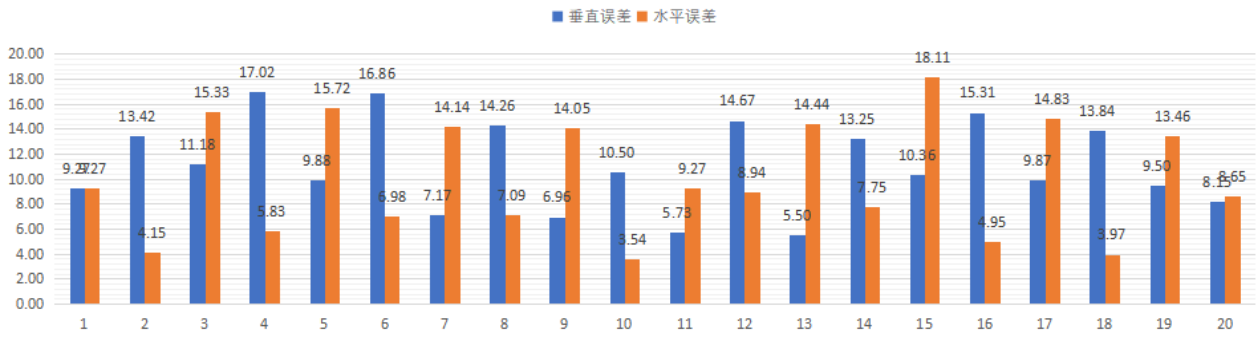

图 7-8 问题三数据集 2 垂直/水平误差变化图

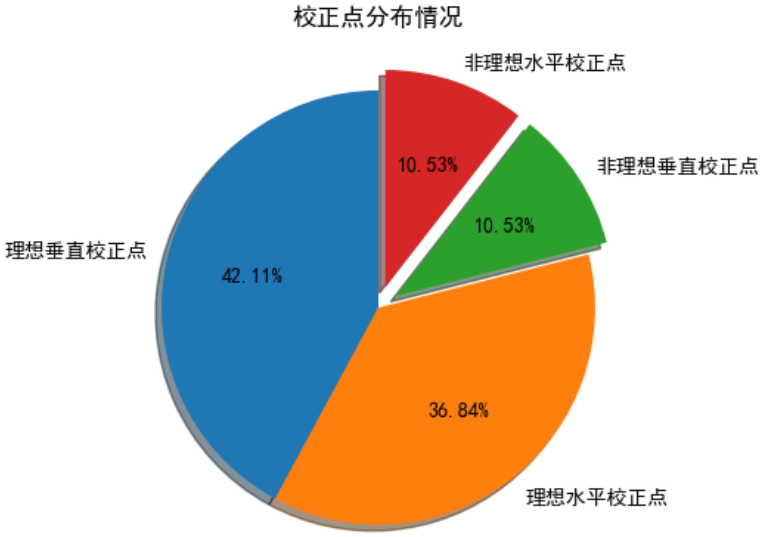

图 7-9 问题三数据集 2 航迹规划路线中校正点分布情况

为了进一步说明本文针对问题三所设计的算法求解出的结果是最优解，我们选取了部分数据集 2 其他解果，参考下表 7-9：

表 7-9 数据集 1 中不同校正点个数下的结果

校正点个数	航迹路径	航迹长度	失败概率
18	无	无	
19	[0, 285, 162, 55, 234, 42, 316, 312, 379, 284, 273, 370, 612]	121975m	45.17%
	[0, 285, 162, 55, 234, 192, 70, 171, 312, 198, 513, 277, 612]	119272m	48.32%
20	[0, 184, 163, 114, 251, 137, 194, 190, 296, 250, 243, 73, 82, 44, 211, 321, 279, 301, 38, 110, 99, 326]	149008m	56.21%
	[0, 184, 163, 114, 251, 137, 194, 190, 296, 250, 243, 73, 249, 274, 12, 216, 279, 301, 38, 110, 99, 326]	149871m	54.79%
21	[0, 184, 163, 114, 251, 137, 194, 190, 296, 250, 243, 167, 82, 207, 44, 211, 321, 279, 301, 38, 287, 99, 326]	151302m	54.13%
	[0, 285, 162, 55, 234, 42, 316, 499, 312, 96, 476, 520, 59, 425, 612]	150967m	49.72%

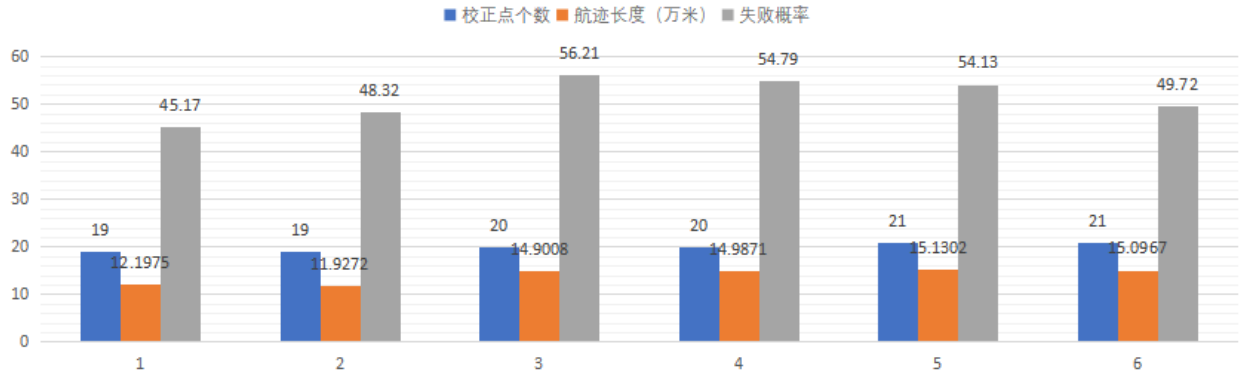

图 7-10 校正点个数及航迹长度对比柱状图

由表 7-9 和图 7-10 可以观察到，在我们选取的部分结果中，校正点个数为 19 时，航迹长度明显小于其他校正点数的结果，失败概率明显小于其他校正点数的结果。可以从这个角度说明本文求解出的 19 个校正点，航迹长度 147271m 为算法的最优解。

7.3 模型评估

7.3.1 算法的有效性和复杂度

本问题是多目标规划问题，在本模型中，我们以失败概率最小化、航迹路线最小化、误差校正点总数最小化的优先级顺序作为主线，将多目标规划模型通过无量纲化处理和参考优先级顺序，引入线性加权系数，使复杂的多目标规划模型求解转换为单目标规划模型，极大降低了问题的求解难度。本问题将局部搜索能力强的禁忌搜索算法与遗传算法结合，提出了遗传禁忌搜索算法。禁忌搜索作为遗传算法的选择操作可以得到每一代的最优个体，并且通过遗传算法中的变异算子可以提高禁忌算法中解空间的多样性。应用马尔科夫链模型可以证明遗传禁忌搜索算法是以概率 1 收敛到全局最优解的。结果表明遗传禁忌搜索算法在飞行器航迹规划这一问题的求解过程中展现了更好的寻优能力，具有更好的有效性。

应用求解随机算法时间复杂度的方法，即求解算法的期望收敛时间，得到该算法的时间复杂度为 $T(\mathrm{n})=O(n^4/\lambda)$，在程序运行过程中，找到最优解的时间数据集 1 为 70.69s，数据集 2 为 54.42s。

7.3.2 灵敏度分析

在部署遗传禁忌搜索算法时，交叉率 P_c 和变异率 P_m 是需要手动设置的参数，且交叉率 P_c 和变异率 P_m 是影响算法效率的重要因素，但是到目前为止仍然没有一种合理选择交叉率 P_c 和变异率 P_m 的依据，只是给出了一个选择它们的推荐范围，即交叉率 $P_c \sim [0.4,0.8]$ 和变异率 $P_m \sim [10^{-4},10^{-1}]$。应用实践表明，交叉率 P_c 和变异率 P_m 选择不当，会导致算法收敛效率降低甚至失败的情况。

为此，本文在算法设计时，研究了 (P_c,P_m) 对结果的影响。取交叉率

$P_c \in (0.4,0.5,0.6,0.7,0.8)$，取变异率 $P_e \in (10^{-4},10^{-3},10^{-2},10^{-1})$。根据交叉率和变异率的不同取值，对于数据集 1，进行 100 次迭代，求得不同的目标函数值，如下图 7-9 所示：

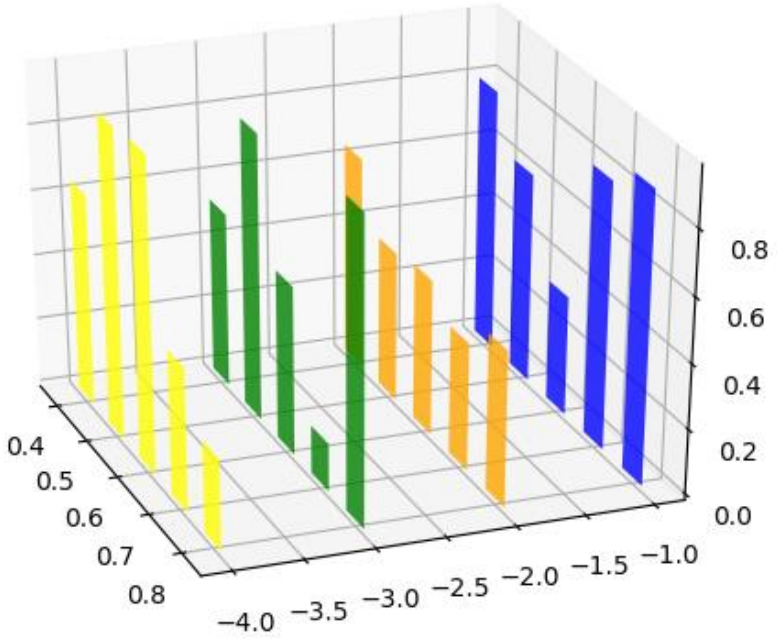

图 7-9 不同交叉率和变异率下的目标函数值

可以观察到，(P_c,P_m) 取 $(0.7,10^{-3})$ $(0.6,10^{-1})$ 时，100 次迭代后得到的目标函数值较小，算法收敛快，可作为备选参数使用。

对于数据集 2，进行 100 次迭代，可以求得不同的目标函数值，如下图 7-10 所示：

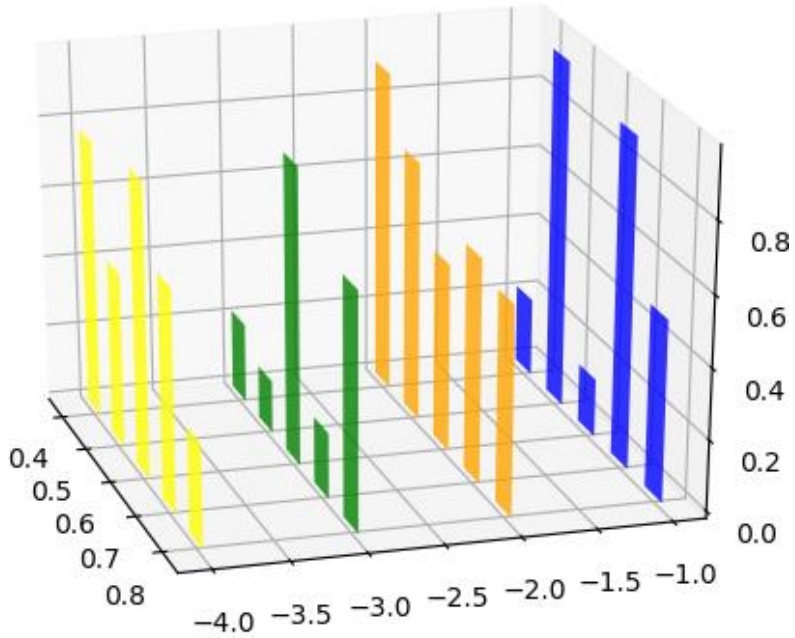

图 7-10 不同交叉率和变异率下的目标函数值

可以观察到，(P_c,P_m) 取 $(0.6,10^{-1})$，$(0.7,10^{-3})$，$(0.5,10^{-3})$ 时，100 次迭代后得到的目标函数值较小，算法收敛快，可作为备选参数使用。

8. 模型的评价

8.1 模型的优点

（1）本文在建模时充分考虑了三个问题的现实条件，对模型的多种可能情况进行了详细的分析论证。在问题一中，分析了飞行器在进行水平或垂直校正器时的约束条件，并抽象成数学语言描述这若干个约束条件；在问题二中，本文着重分析了飞行器在三维空间中转向的各类情况，参考 Dubins 曲线，建立方程对飞行器转向方式进行约束；在问题三中，本文对目标函数引入飞行器飞行失败概率和判决变量，分析了导致飞行器飞行失败的原因，并在问题一的基础上，将判决变量转化为模型的约束条件。

（2）模型充分考虑到需要优化的变量。本文在建模时将多目标规划模型通过无量纲化处理和参考优先级顺序，引入了线性加权系数，使复杂的多目标规划模型求解转换为单目标规划模型，极大降低了问题的求解难度，可以使需要优化的变量向全局最优逼近。

（3）模型的扩展性和可移植性比较强。本文所建立的模型在两个不同场景下均能给出最优解。

（4）模型的实用性较强，模型易于理解，符合实际，计算简便。

8.2 模型的缺点

（1）模型中多目标优化转化为单目标优化时，权系数根据经验定义，计算结果与全局最优解存在一定偏差。

（2）模型建立过程简化了一些条件，对这些条件采用理想化处理方案，在实际过程中应更加细致讨论。

（3）模型在一些条件上做出假设，可能导致对部分信息考虑不足，不够完善。

9. 参考文献

[1] Z. Li and R. Han, "Unmanned Aerial Vehicle Three-dimensional Trajectory Planning Based on Ant Colony Algorithm," 2018 37th Chinese Control Conference (CCC), Wuhan, 2018, pp. 9992-9995.

[2] Sun, Liguo, Li, Shidan, Wang, Desheng . ”Flight route planning forterrain navigation using multi-fractal theory”. Journal of Tsinghua University, 2011.

[3] Qu, Dongcai, Yu, Dehai, Feng, Yuguang. ”On flight route planning method of the terrain avoidance system”. 2014 IEEE Chinese Guidance,Navigation and Control Conference, CGNCC 2014, 2015, pp. 492-496.

[4] Sun, Tsung-Ying; Huo, Chih-Li; Tsai, Shang-Jeng. ”Optimal UAVflight path planning using skeletonization and Particle Swarm Optimizer”.2008 IEEE Congress on Evolutionary Computation, CEC 2008, 2008, pp. 1183-1188

[5] 彭建亮，孙秀霞，朱凡, et al. 基于遗传算法的多约束三维航迹规划方法研究[C]// 第二十七届中国控制会议论文集. 2008.

[6] 季敏惠. 禁忌搜索算法[J]. 电脑知识与技术, 2009, 5(27):7748-7750.

[7] Classification of the Dubins set. Shkel, A.M. 1 ; Lumelsky, V. 1

10. 附录

10.1 问题一代码

10.1.1 main.py

路径搜索脚本：

```
import openpyxl
import sys
import random
random.seed(1)
sys.setrecursionlimit(100000)
alpha1=25
alpha2=15
beta1=20
beta2=25
theta=30
sigma=0.001
import os
data_path = os.path.join('..', '..', '2019 年中国研究生数学建模竞赛 F 题\附件 1：数据集 1-终稿.xlsx')
# 打开 excel 文件,获取工作簿对象
wb = openpyxl.load_workbook(data_path)
# 获取指定的表单
sheet = wb['data1']
point_list = []
for row in sheet[3:sheet.max_row]:
    point_list.append([row[1].value, row[2].value, row[3].value, row[4].value, row[5].value])
point_num = len(point_list)

def get_distance(start_index, end_index):
    x1, y1, z1 = point_list[start_index][0:3]
    x2, y2, z2 = point_list[end_index][0:3]
    return ((x1-x2)**2+(y1-y2)**2+(z1-z2)**2)**0.5
def judge_z(start_index, end_index=point_num-1):
    x1, y1, z1 = point_list[start_index][0:3]
    x2, y2, z2 = point_list[end_index][0:3]
    if abs(z1-z2)>5000:
        return False
    else:
        return True
def judge(start_index, end_index, horizontal_error,vertical_error):
    end_point_type = point_list[end_index][3]
    distance = get_distance(start_index, end_index)
```

```
    delta_error = distance * sigma
    end_point_horizontal_error = horizontal_error + delta_error
    end_point_vertical_error = vertical_error + delta_error
    if judge_z(start_index):
        if end_index != point_num - 1:#下一个点不是终点
            if end_point_type == 0: #  水平
                if end_point_horizontal_error<=beta2 and end_point_vertical_error<=beta1:
                    is_pass = True
                else:
                    is_pass = False
            elif end_point_type == 1:#垂直
                if end_point_horizontal_error<=alpha2 and end_point_vertical_error<=alpha1:
                    is_pass = True
                else:
                    is_pass = False
            else:
                is_pass = False
        else:#下一个点是终点
            if end_point_horizontal_error <= theta and end_point_vertical_error <= theta:
                is_pass = True
            else:
                is_pass = False
    else:
        is_pass=False
    after_end_point_horizontal_error = end_point_horizontal_error
    after_end_point_vertical_error = end_point_vertical_error
    if is_pass:
        if end_point_type == 0:
            after_end_point_horizontal_error = 0
            after_end_point_vertical_error = end_point_vertical_error
        elif end_point_type ==1:
            after_end_point_horizontal_error = end_point_horizontal_error
            after_end_point_vertical_error = 0
    return is_pass, end_point_horizontal_error, end_point_vertical_error, after_end_point_horizontal_error, after_end_point_vertical_error
def rank_distance(point_index_list):
    ranked_list = sorted(point_index_list, key=lambda index_list: get_distance(index_list[0], point_num-1))
    return ranked_list

def get_all_distance(index_list):
    distance = 0
    for i in range(len(index_list)-1):
```

```
            start_index = index_list[i]
            end_index = index_list[i+1]
            distance = distance + get_distance(start_index, end_index)
        return distance
vis = point_num*[0]
order = []
temp_all_distance = 1000000
temp_order = []
def find_path(start_index=0, horizontal_error=0, vertical_error=0):
        global temp_all_distance
        global order, temp_order
        order.append(start_index)
        candidate_list = []
        for index in range(point_num):
            if index != start_index:
                is_pass, end_point_horizontal_error, end_point_vertical_error, after_end_point_horizontal_error, after_end_point_vertical_error = judge(start_index, index, horizontal_error, vertical_error)
                if is_pass and get_distance(start_index, point_num-1) > get_distance(index, point_num-1):
                    candidate_list.append(index)
        if len(candidate_list) == 0:
            order.pop()
            return
        candidate_list.sort(key=lambda index_list: get_distance(index_list, point_num-1))
        # random.shuffle(candidate_list)
        if len(candidate_list) >= 5:
            candidate_list = candidate_list[0:5]
        for candidate in candidate_list:
            if candidate == point_num - 1:
                order.append(candidate)
                all_distance = get_all_distance(order)
                if all_distance < temp_all_distance:
                    temp_all_distance = all_distance
                    temp_order = order.copy()
                    print('\n' + 'small', order, all_distance)
                order.pop()
                break
            if len(order) > 13:
                break
            is_pass, end_point_horizontal_error, end_point_vertical_error, after_end_point_horizontal_error, after_end_point_vertical_error = judge(
                start_index, candidate, horizontal_error, vertical_error)
            find_path(candidate, after_end_point_horizontal_error, after_end_point_vertical_error)
```

```
        order.pop()
        return
find_path()
print(temp_order)
```

10.1.2 compute_error.py

计算误差脚本：

```
import openpyxl
import sys
sys.setrecursionlimit(100000)
alpha1=25
alpha2=15
beta1=20
beta2=25
theta=30
sigma=0.001
import os
data_path = os.path.join('..', '..', '2019 年中国研究生数学建模竞赛 F 题\附件 1：数据集 1-终稿.xlsx')
# 打开 excel 文件,获取工作簿对象
wb = openpyxl.load_workbook(data_path)
# 获取指定的表单
sheet = wb['data1']
point_list = []
print(sheet.max_row)
for row in sheet[3:sheet.max_row]:
        point_list.append([row[1].value, row[2].value, row[3].value, row[4].value, row[5].value])
print(point_list)
point_num = len(point_list)
def judge_z(start_index, end_index=point_num-1):
        x1, y1, z1 = point_list[start_index][0:3]
        x2, y2, z2 = point_list[end_index][0:3]
        if abs(z1-z2)>5000:
                return False
        else:
                return True
def get_distance(start_index, end_index):
        x1, y1, z1 = point_list[start_index][0:3]
        x2, y2, z2 = point_list[end_index][0:3]
        return ((x1-x2)**2+(y1-y2)**2+(z1-z2)**2)**0.5
def judge(start_index, end_index, horizontal_error,vertical_error):
        end_point_type = point_list[end_index][3]
        distance = get_distance(start_index, end_index)
```

```
    delta_error = distance * sigma
    end_point_horizontal_error = horizontal_error + delta_error
    end_point_vertical_error = vertical_error + delta_error
    if judge_z(start_index):
        if end_index != point_num - 1:#下一个点不是终点
            if end_point_type == 0: #  水平
                if end_point_horizontal_error<=beta2 and end_point_vertical_error<=beta1:
                    is_pass = True
                else:
                    is_pass = False
            elif end_point_type == 1:#垂直
                if end_point_horizontal_error<=alpha2 and end_point_vertical_error<=alpha1:
                    is_pass = True
                else:
                    is_pass = False
            else:
                is_pass = False
        else:#下一个点是终点
            if end_point_horizontal_error <= theta and end_point_vertical_error <= theta:
                is_pass = True
            else:
                is_pass = False
    else:
        is_pass=False
    after_end_point_horizontal_error = end_point_horizontal_error
    after_end_point_vertical_error = end_point_vertical_error
    if is_pass:
        if end_point_type == 0:
            after_end_point_horizontal_error = 0
            after_end_point_vertical_error = end_point_vertical_error
        elif end_point_type == 1:
            after_end_point_horizontal_error = end_point_horizontal_error
            after_end_point_vertical_error = 0
    return is_pass, end_point_horizontal_error, end_point_vertical_error, after_end_point_horizontal_error, after_end_point_vertical_error


matlab_path_list = [0, 503, 294, 91, 282, 33, 315, 403, 594, 501, 612]
path_list = [[i, 0, 0, 0, 0] for i in matlab_path_list]
distance = 0
for i in range(len(path_list) - 1):
    start_index = path_list[i][0]
    end_index = path_list[i + 1][0]
```

```
    is_pass, end_point_horizontal_error, end_point_vertical_error, after_end_point_horizontal_error, after_end_point_vertical_error = judge(start_index, end_index, path_list[i][3], path_list[i][4])
    if is_pass:
        path_list[i + 1][1:] = end_point_horizontal_error, end_point_vertical_error, after_end_point_horizontal_error, after_end_point_vertical_error
    else:
        print('error', start_index, end_index)
    distance = distance + get_distance(start_index, end_index)
for i in path_list:
print(i, point_list[i[0]][3])
```

10.2 问题二代码

10.2.1 main.py

路径搜索脚本：

```
import openpyxl
import sys
import random
random.seed(1)
sys.setrecursionlimit(100000)
alpha1=25
alpha2=15
beta1=20
beta2=25
theta=30
sigma=0.001
import os
data_path = os.path.join('..', '..', '2019 年中国研究生数学建模竞赛 F 题\附件 1：数据集 1-终稿.xlsx')
# 打开 excel 文件,获取工作簿对象
wb = openpyxl.load_workbook(data_path)
# 获取指定的表单
sheet = wb['data1']
point_list = []
for row in sheet[3:sheet.max_row]:
    point_list.append([row[1].value, row[2].value, row[3].value, row[4].value, row[5].value])
point_num = len(point_list)
def get_distance(start_index, end_index):
    x1, y1, z1 = point_list[start_index][0:3]
    x2, y2, z2 = point_list[end_index][0:3]
    return ((x1-x2)**2+(y1-y2)**2+(z1-z2)**2)**0.5
def judge_z(start_index, end_index=point_num-1):
```

```
    x1, y1, z1 = point_list[start_index][0:3]
    x2, y2, z2 = point_list[end_index][0:3]
    if abs(z1-z2)>5000:
        return False
    else:
        return True
def judge(start_index, end_index, horizontal_error,vertical_error):
    end_point_type = point_list[end_index][3]
    distance = get_distance(start_index, end_index)
    delta_error = distance * sigma
    end_point_horizontal_error = horizontal_error + delta_error
    end_point_vertical_error = vertical_error + delta_error
    if judge_z(start_index):
        if end_index != point_num - 1:#下一个点不是终点
            if end_point_type == 0: #  水平
                if end_point_horizontal_error<=beta2 and end_point_vertical_error<=beta1:
                    is_pass = True
                else:
                    is_pass = False
            elif end_point_type == 1:#垂直
                if end_point_horizontal_error<=alpha2 and
end_point_vertical_error<=alpha1:
                    is_pass = True
                else:
                    is_pass = False
            else:
                is_pass = False
        else:#下一个点是终点
            if end_point_horizontal_error <= theta and end_point_vertical_error <= theta:
                is_pass = True
            else:
                is_pass = False
    else:
        is_pass=False
    after_end_point_horizontal_error = end_point_horizontal_error
    after_end_point_vertical_error = end_point_vertical_error
    if is_pass:
        if end_point_type == 0:
            after_end_point_horizontal_error = 0
            after_end_point_vertical_error = end_point_vertical_error
        elif end_point_type ==1:
            after_end_point_horizontal_error = end_point_horizontal_error
            after_end_point_vertical_error = 0
    return is_pass, end_point_horizontal_error, end_point_vertical_error,
```

```
after_end_point_horizontal_error, after_end_point_vertical_error
def rank_distance(point_index_list):
    ranked_list = sorted(point_index_list, key=lambda index_list: get_distance(index_list[0], point_num-1))
    return ranked_list
def get_all_distance(index_list):
    distance = 0
    for i in range(len(index_list)-1):
        start_index = index_list[i]
        end_index = index_list[i+1]
        distance = distance + get_distance(start_index, end_index)
    return distance
vis = point_num*[0]
order = []
temp_all_distance = 104898
temp_order = []

def find_path(start_index=0, horizontal_error=0, vertical_error=0):
    global temp_all_distance
    global order, temp_order
    order.append(start_index)
    candidate_list = []
    for index in range(point_num):
        if index != start_index:
            is_pass, end_point_horizontal_error, end_point_vertical_error, after_end_point_horizontal_error, after_end_point_vertical_error = judge(start_index, index, horizontal_error, vertical_error)
            if is_pass and get_distance(start_index, point_num-1) > get_distance(index, point_num-1):
                candidate_list.append(index)
    if len(candidate_list) == 0:
        order.pop()
        return
    # candidate_list.sort(key=lambda index_list: get_distance(index_list, point_num-1))
    random.shuffle(candidate_list)
    if len(candidate_list) >= 5:
        candidate_list = candidate_list[0:5]
    for candidate in candidate_list:
        if candidate == point_num - 1:
            order.append(candidate)
            all_distance = get_all_distance(order)
            if all_distance < temp_all_distance:
                temp_all_distance = all_distance
                temp_order = order.copy()
```

```
                print('\n' + 'small', order, all_distance)
            order.pop()
            break
        if len(order) > 13:
            break
        is_pass, end_point_horizontal_error, end_point_vertical_error,
after_end_point_horizontal_error, after_end_point_vertical_error = judge(
            start_index, candidate, horizontal_error, vertical_error)
        find_path(candidate, after_end_point_horizontal_error, after_end_point_vertical_error)
    order.pop()
    return
find_path()
print(temp_order)
```

10.2.2 compute_error.py

计算误差脚本：

```
# import threading
import openpyxl
import sys
sys.setrecursionlimit(100000)

alpha1=25
alpha2=15
beta1=20
beta2=25
theta=30
sigma=0.001

import os
data_path = os.path.join('..', '..', '2019 年中国研究生数学建模竞赛 F 题\附件 1：数据集 1-终稿.xlsx')

# 打开 excel 文件,获取工作簿对象
wb = openpyxl.load_workbook(data_path)
# 获取指定的表单
sheet = wb['data1']
point_list = []
for row in sheet[3:sheet.max_row]:
    point_list.append([row[1].value, row[2].value, row[3].value, row[4].value, row[5].value])
point_num = len(point_list)
matlab_path_list = [0, 503, 69, 237, 233, 598, 561, 448, 485, 612]
path_list = [[i, 0, 0, 0, 0] for i in matlab_path_list]
all_distance_list = [0, 13397.94141591609, 22206.718527477846, 34706.853446975045,
45532.46491249756, 59539.840926734374, 70496.2591586171, 75652.22657391406,
```

```
81388.44828831236, 104960.35109086095]


def judge_z(start_index, end_index=point_num-1):
    x1, y1, z1 = point_list[start_index][0:3]
    x2, y2, z2 = point_list[end_index][0:3]
    if abs(z1-z2)>5000:
        return False
    else:
        return True


def get_distance(start_index, end_index):
    return all_distance_list[matlab_path_list.index(end_index)] - all_distance_list[matlab_path_list.index(start_index)]


def judge(start_index, end_index, horizontal_error,vertical_error):
    end_point_type = point_list[end_index][3]
    distance = get_distance(start_index, end_index)
    delta_error = distance * sigma
    end_point_horizontal_error = horizontal_error + delta_error
    end_point_vertical_error = vertical_error + delta_error
    if judge_z(start_index):
        if end_index != point_num - 1:#下一个点不是终点
            if end_point_type == 0: #  水平
                if end_point_horizontal_error<=beta2 and end_point_vertical_error<=beta1:
                    is_pass = True
                else:
                    is_pass = False
            elif end_point_type == 1:#垂直
                if end_point_horizontal_error<=alpha2 and end_point_vertical_error<=alpha1:
                    is_pass = True
                else:
                    is_pass = False
            else:
                is_pass = False
        else:#下一个点是终点
            if end_point_horizontal_error <= theta and end_point_vertical_error <= theta:
                is_pass = True
            else:
                is_pass = False
    else:
```

```
            is_pass=False
        after_end_point_horizontal_error = end_point_horizontal_error
        after_end_point_vertical_error = end_point_vertical_error
        if is_pass:
            if end_point_type == 0:
                after_end_point_horizontal_error = 0
                after_end_point_vertical_error = end_point_vertical_error
            elif end_point_type == 1:
                after_end_point_horizontal_error = end_point_horizontal_error
                after_end_point_vertical_error = 0
        return is_pass, end_point_horizontal_error, end_point_vertical_error,
after_end_point_horizontal_error, after_end_point_vertical_error


distance = 0
for i in range(len(path_list) - 1):
        start_index = path_list[i][0]
        end_index = path_list[i + 1][0]
        is_pass, end_point_horizontal_error, end_point_vertical_error,
after_end_point_horizontal_error, after_end_point_vertical_error = judge(start_index, end_index,
path_list[i][3], path_list[i][4])
        if is_pass:
            path_list[i + 1][1:] = end_point_horizontal_error, end_point_vertical_error,
after_end_point_horizontal_error, after_end_point_vertical_error
        else:
            print('error',start_index,end_index)
        distance = distance + get_distance(start_index, end_index)
for i in path_list:
        print(i, point_list[i[0]][3])
```

10.2.3 dubins.py

计算 dubins 曲线脚本：

```
import dubins
import openpyxl
import sys
from scipy import interpolate
import numpy as np
sys.setrecursionlimit(100000)
import math
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import os
data_path = os.path.join('..', '..', '2019 年中国研究生数学建模竞赛 F 题\附件 1：数据集 1-终
稿.xlsx')
```

```
# 打开 excel 文件,获取工作簿对象
wb = openpyxl.load_workbook(data_path)
# 获取指定的表单
sheet = wb['data1']
point_list = []
for row in sheet[3:sheet.max_row]:
    point_list.append([row[1].value, row[2].value, row[3].value, row[4].value, row[5].value])
point_num = len(point_list)
matlab_path_list = [0, 503, 69, 237, 233, 598, 561, 448, 485, 612]
path_list = [point_list[i][:3] for i in matlab_path_list]
x_min = min([path_list[i][0] for i in range(len(path_list))])
x_max = max([path_list[i][0] for i in range(len(path_list))])
x_new = list(range(x_min+10, x_max-10))
x = []
y = []
for i in range(len(path_list)-1):
    q0 = (path_list[i][0], path_list[i][1], 0)
    q1 = (path_list[i+1][0], path_list[i+1][1], 0)
    turning_radius = 200
    step_size = 10
    path = dubins.shortest_path(q0, q1, turning_radius)
    configurations, _ = path.sample_many(step_size)
    for point in configurations:
        x.append(point[0])
        y.append(point[1])
f=interpolate.interp1d(x,y,kind='slinear')
y_new = f(x_new)
plt.plot(x,y)
plt.plot(x_new,y_new)
plt.show()

# xz
x_1 = []
z = []
for i in range(len(path_list)-1):
    q0 = (path_list[i][0], path_list[i][2], 0)
    q1 = (path_list[i+1][0], path_list[i+1][2], 0)
    turning_radius = 200
    step_size = 10
    path = dubins.shortest_path(q0, q1, turning_radius)
    configurations, _ = path.sample_many(step_size)
    for point in configurations:
        x_1.append(point[0])
        z.append(point[1])
```

```
f=interpolate.interp1d(x_1,z,kind='slinear')
z_new = f(x_new)

plt.plot(x_1,z)
plt.plot(x_new,z_new)
plt.show()

ax = plt.figure().add_subplot(111, projection='3d')
ax.plot(x_new,y_new, z_new)
plt.show()

y_new = y_new.tolist()
z_new = z_new.tolist()

x_new.insert(0,int(path_list[0][0]))
y_new.insert(0,int(path_list[0][1]))
z_new.insert(0,int(path_list[0][2]))

x_new.append(int(path_list[-1][0]))
y_new.append(int(path_list[-1][1]))
z_new.append(int(path_list[-1][2]))

distance_list = []
total_distance = 0
for i in range(len(x_new)-1):
    x1, y1, z1 = x_new[i], y_new[i], z_new[i]
    x2, y2, z2 = x_new[i+1], y_new[i+1], z_new[i+1]
    distance = ((x1-x2)**2+(y1-y2)**2+(z1-z2)**2)**0.5
    total_distance = distance + total_distance
    distance_list.append(total_distance)

distance_list.insert(0, 0)
final_distance = []
for point in path_list:
    final_distance.append(distance_list[x_new.index(int(point[0]))])

print(final_distance)

all_point = np.array([x_new,y_new,z_new])
np.save(os.path.join('.', 'xyz_point_1.npy'), all_point)
```

10.3 问题三代码

10.3.1 main.py

路径搜索脚本：

```
import openpyxl
import sys
import random
random.seed(1)
import numpy as np
sys.setrecursionlimit(100000)
alpha1=25
alpha2=15
beta1=20
beta2=25
theta=30
sigma=0.001
import os
data_path = os.path.join('..', '..', '2019 年中国研究生数学建模竞赛 F 题\附件 1：数据集 1-终稿.xlsx')
# 打开 excel 文件,获取工作簿对象
wb = openpyxl.load_workbook(data_path)
# 获取指定的表单
sheet = wb['data1']
point_list = []
for row in sheet[3:sheet.max_row]:
    point_list.append([row[1].value, row[2].value, row[3].value, row[4].value, row[5].value])
point_num = len(point_list)

def get_distance(start_index, end_index):
    x1, y1, z1 = point_list[start_index][0:3]
    x2, y2, z2 = point_list[end_index][0:3]
    return ((x1-x2)**2+(y1-y2)**2+(z1-z2)**2)**0.5

def judge_z(start_index, end_index=point_num-1):
    x1, y1, z1 = point_list[start_index][0:3]
    x2, y2, z2 = point_list[end_index][0:3]
    if abs(z1-z2)>5000:
        return False
    else:
        return True

def judge(start_index, end_index, horizontal_error,vertical_error):
    end_point_type = point_list[end_index][3]
    end_point_safe = point_list[end_index][4]
    distance = get_distance(start_index, end_index)
    delta_error = distance * sigma
```

```
    end_point_horizontal_error = horizontal_error + delta_error
    end_point_vertical_error = vertical_error + delta_error
    if judge_z(start_index):
        if end_index != point_num - 1:#下一个点不是终点
            if end_point_type == 0: #  水平
                if end_point_horizontal_error<=beta2 and end_point_vertical_error<=beta1:
                    is_pass = True
                else:
                    is_pass = False
            elif end_point_type == 1:#垂直
                if end_point_horizontal_error<=alpha2 and end_point_vertical_error<=alpha1:
                    is_pass = True
                else:
                    is_pass = False
            else:
                is_pass = False
        else:#下一个点是终点
            if end_point_horizontal_error <= theta and end_point_vertical_error <= theta:
                is_pass = True
            else:
                is_pass = False
    else:
        is_pass=False
    after_end_point_horizontal_error = end_point_horizontal_error
    after_end_point_vertical_error = end_point_vertical_error
    if is_pass:
        if end_point_safe == 0:
            if end_point_type == 0:
                after_end_point_horizontal_error = 0
                after_end_point_vertical_error = end_point_vertical_error
            elif end_point_type == 1:
                after_end_point_horizontal_error = end_point_horizontal_error
                after_end_point_vertical_error = 0
        else:
            if end_point_type == 0:
                after_end_point_horizontal_error = min(end_point_horizontal_error, 5)
                after_end_point_vertical_error = end_point_vertical_error
            elif end_point_type == 1:
                after_end_point_horizontal_error = end_point_horizontal_error
                after_end_point_vertical_error = min(end_point_vertical_error, 5)
    return is_pass, end_point_horizontal_error, end_point_vertical_error, after_end_point_horizontal_error, after_end_point_vertical_error
```

```
def rank_distance(point_index_list):
    ranked_list = sorted(point_index_list, key=lambda index_list: get_distance(index_list[0], point_num-1))
    return ranked_list


def get_all_distance(index_list):
    distance = 0
    for i in range(len(index_list)-1):
        start_index = index_list[i]
        end_index = index_list[i+1]
        distance = distance + get_distance(start_index, end_index)
    return distance

def judge_P(start_index, end_index, horizontal_error,vertical_error):
    end_point_type = point_list[end_index][3]
    end_point_safe = point_list[end_index][4]
    if end_point_safe == 1:
        end_point_safe = random.random()
    distance = get_distance(start_index, end_index)
    delta_error = distance * sigma
    end_point_horizontal_error = horizontal_error + delta_error
    end_point_vertical_error = vertical_error + delta_error
    if judge_z(start_index):
        if end_index != point_num - 1:#下一个点不是终点
            if end_point_type == 0: #  水平
                if end_point_horizontal_error<=beta2 and end_point_vertical_error<=beta1:
                    is_pass = True
                else:
                    is_pass = False
            elif end_point_type == 1:#垂直
                if end_point_horizontal_error<=alpha2 and end_point_vertical_error<=alpha1:
                    is_pass = True
                else:
                    is_pass = False
            else:
                is_pass = False
        else:#下一个点是终点
            if end_point_horizontal_error <= theta and end_point_vertical_error <= theta:
                is_pass = True
            else:
                is_pass = False
    else:
```

```
            is_pass=False
        after_end_point_horizontal_error = end_point_horizontal_error
        after_end_point_vertical_error = end_point_vertical_error
        if is_pass:
            if end_point_safe <= 0.8:
                if end_point_type == 0:
                    after_end_point_horizontal_error = 0
                    after_end_point_vertical_error = end_point_vertical_error
                elif end_point_type == 1:
                    after_end_point_horizontal_error = end_point_horizontal_error
                    after_end_point_vertical_error = 0
            else:
                if end_point_type == 0:
                    after_end_point_horizontal_error = min(end_point_horizontal_error, 5)
                    after_end_point_vertical_error = end_point_vertical_error
                elif end_point_type == 1:
                    after_end_point_horizontal_error = end_point_horizontal_error
                    after_end_point_vertical_error = min(end_point_vertical_error, 5)
        return is_pass, end_point_horizontal_error, end_point_vertical_error,
after_end_point_horizontal_error, after_end_point_vertical_error, end_point_safe


def get_type(point_index, end_point_safe):
        if point_list[point_index][3] == 0 and end_point_safe < 0.8: #  水平
            return '01'
        elif point_list[point_index][3] == 0 and end_point_safe > 0.8: #  水平
            return '02'
        elif point_list[point_index][3] == 1 and end_point_safe < 0.8: #  水平
            return '11'
        elif point_list[point_index][3] == 1 and end_point_safe > 0.8: #  水平
            return '12'
        elif point_index == len(point_list) - 1:
            return 'B 点'

def get_P(matlab_path_list):
        path_list = [[i, 0, 0, 0, 0, '55'] for i in matlab_path_list]
        path_list[0][-1] = 'A 点'
        path_list[-1][-1] = 'B 点'
        distance = 0
        success = 0
        for j in range(10000):
            is_ok = True
            for i in range(len(path_list) - 1):
                start_index = path_list[i][0]
```

```
                end_index = path_list[i + 1][0]
                is_pass, end_point_horizontal_error, end_point_vertical_error, after_end_point_horizontal_error, after_end_point_vertical_error, end_point_safe = judge_P(start_index, end_index, path_list[i][3], path_list[i][4])
                type = get_type(path_list[i + 1][0], end_point_safe)
                if is_pass:
                    path_list[i + 1][1:] = end_point_horizontal_error, end_point_vertical_error, after_end_point_horizontal_error, after_end_point_vertical_error, type
                else:
                    is_ok = False
                    break
                distance = distance + get_distance(start_index, end_index)
            if is_ok == True:
                success = success + 1
        return success/10000

vis = point_num*[0]
order = []
temp_all_distance = 10000000
temp_order = []
all_solution = []
def find_path(start_index=0, horizontal_error=0, vertical_error=0):
        global temp_all_distance, all_solution
        global order, temp_order
        order.append(start_index)
        candidate_list = []
        for index in range(point_num):
            if index != start_index:
                is_pass, end_point_horizontal_error, end_point_vertical_error, after_end_point_horizontal_error, after_end_point_vertical_error = judge(start_index, index, horizontal_error, vertical_error)
                if is_pass and get_distance(start_index, point_num-1) > get_distance(index, point_num-1):
                    candidate_list.append(index)
        if len(candidate_list) == 0:
            order.pop()
            return
        candidate_list.sort(key=lambda index_list: get_distance(index_list, point_num-1))
        # random.shuffle(candidate_list)
        length = 10
        if len(candidate_list) >= length:
            candidate_list = candidate_list[0:length]
        for candidate in candidate_list:
            if candidate == point_num - 1:
```

```
                order.append(candidate)
                all_distance = get_all_distance(order)
                if all_distance < temp_all_distance:
                    temp_all_distance = all_distance
                    temp_order = order.copy()
                    P = get_P(order)
                    print('small', order, all_distance, P)
                    all_solution.append([order.copy(), all_distance, P])
                order.pop()
                break
            if len(order) > 13:
                break
            is_pass, end_point_horizontal_error, end_point_vertical_error, after_end_point_horizontal_error, after_end_point_vertical_error = judge(
                start_index, candidate, horizontal_error, vertical_error)
            find_path(candidate, after_end_point_horizontal_error, after_end_point_vertical_error)
        order.pop()
        return

find_path()
print(temp_order)
print(all_solution)
all_solution = np.array(all_solution)
np.save('all_solution_1.npy', all_solution)
```

10.3.2 compute_error.py

计算误差脚本：

```
import openpyxl
import sys
import random
from tqdm import tqdm
sys.setrecursionlimit(100000)
alpha1=25
alpha2=15
beta1=20
beta2=25
theta=30
sigma=0.001
import os
data_path = os.path.join('..', '..', '2019 年中国研究生数学建模竞赛 F 题\附件 1：数据集 1-终稿.xlsx')
# 打开 excel 文件,获取工作簿对象
wb = openpyxl.load_workbook(data_path)
```

```
# 获取指定的表单
sheet = wb['data1']
point_list = []
for row in sheet[3:sheet.max_row]:
    point_list.append([row[1].value, row[2].value, row[3].value, row[4].value, row[5].value])
point_num = len(point_list)


def judge_z(start_index, end_index=point_num-1):
    x1, y1, z1 = point_list[start_index][0:3]
    x2, y2, z2 = point_list[end_index][0:3]
    if abs(z1-z2)>5000:
        return False
    else:
        return True


def get_distance(start_index, end_index):
    x1, y1, z1 = point_list[start_index][0:3]
    x2, y2, z2 = point_list[end_index][0:3]
    return ((x1-x2)**2+(y1-y2)**2+(z1-z2)**2)**0.5


def judge(start_index, end_index, horizontal_error,vertical_error):
    end_point_type = point_list[end_index][3]
    end_point_safe = point_list[end_index][4]
    if end_point_safe == 1:
        end_point_safe = random.random()
    distance = get_distance(start_index, end_index)
    delta_error = distance * sigma
    end_point_horizontal_error = horizontal_error + delta_error
    end_point_vertical_error = vertical_error + delta_error
    if judge_z(start_index):
        if end_index != point_num - 1:#下一个点不是终点
            if end_point_type == 0: # 水平
                if end_point_horizontal_error<=beta2 and end_point_vertical_error<=beta1:
                    is_pass = True
                else:
                    is_pass = False
            elif end_point_type == 1:#垂直
                if end_point_horizontal_error<=alpha2 and
end_point_vertical_error<=alpha1:
                    is_pass = True
                else:
```

```
                        is_pass = False
                else:
                    is_pass = False
            else:#下一个点是终点
                if end_point_horizontal_error <= theta and end_point_vertical_error <= theta:
                    is_pass = True
                else:
                    is_pass = False
        else:
            is_pass=False
        after_end_point_horizontal_error = end_point_horizontal_error
        after_end_point_vertical_error = end_point_vertical_error
        if is_pass:
            if end_point_safe <= 0.8:
                if end_point_type == 0:
                    after_end_point_horizontal_error = 0
                    after_end_point_vertical_error = end_point_vertical_error
                elif end_point_type == 1:
                    after_end_point_horizontal_error = end_point_horizontal_error
                    after_end_point_vertical_error = 0
            else:
                if end_point_type == 0:
                    after_end_point_horizontal_error = min(end_point_horizontal_error, 5)
                    after_end_point_vertical_error = end_point_vertical_error
                elif end_point_type == 1:
                    after_end_point_horizontal_error = end_point_horizontal_error
                    after_end_point_vertical_error = min(end_point_vertical_error, 5)
        return is_pass, end_point_horizontal_error, end_point_vertical_error,
after_end_point_horizontal_error, after_end_point_vertical_error, end_point_safe


def get_type(point_index, end_point_safe):
        if point_list[point_index][3] == 0 and end_point_safe <= 0.8: #  水平
            return '01'
        elif point_list[point_index][3] == 0 and end_point_safe > 0.8:
            return '02'
        elif point_list[point_index][3] == 1 and end_point_safe <= 0.8:
            return '11'
        elif point_list[point_index][3] == 1 and end_point_safe > 0.8:
            return '12'
        elif point_index == len(point_list) - 1:
            return 'B 点'
```

```
def get_sum_of_unsafe(path_list):
    sum = 0
    for point in path_list:
        type = point[-1]
        if '2' in type:
            sum = sum + 1
    return sum


matlab_path_list = [0, 578, 417, 80, 237, 607, 33, 194, 450, 448, 485, 302, 612]
path_list = [[i, 0, 0, 0, 0, '55'] for i in matlab_path_list]
path_list[0][-1] = 'A 点'
path_list[-1][-1] = 'B 点'
distance = 0
success = 0
all_solution = []
for j in tqdm(range(100000)):
    is_ok = True
    for i in range(len(path_list) - 1):
        start_index = path_list[i][0]
        end_index = path_list[i + 1][0]
        is_pass, end_point_horizontal_error, end_point_vertical_error, after_end_point_horizontal_error, after_end_point_vertical_error, end_point_safe = judge(start_index, end_index, path_list[i][3], path_list[i][4])
        type = get_type(path_list[i + 1][0], end_point_safe)
        if is_pass:
            path_list[i + 1][1:] = end_point_horizontal_error, end_point_vertical_error, after_end_point_horizontal_error, after_end_point_vertical_error, type
        else:
            is_ok = False
            break
        distance = distance + get_distance(start_index, end_index)
    if is_ok == True:
        success = success + 1
        all_solution.append(path_list)
print('P={}'.format(success/100000))
temp_unsafe_sum = 0
temp_index = 0
for i, path in enumerate(all_solution):
    unsafe_sum = get_sum_of_unsafe(path)
    if unsafe_sum > temp_unsafe_sum:
        temp_unsafe_sum = unsafe_sum
        temp_index = i
for i in all_solution[temp_index]:
```

```
    print(i)

safe_list = []
for i in [0, 578, 417, 80, 237, 607, 33, 194, 450, 448, 485, 302, 612]:
    safe_list.append(point_list[i][4])
print(safe_list)


def get_P(matlab_path_list):
    path_list = [[i, 0, 0, 0, 0, '55'] for i in matlab_path_list]
    path_list[0][-1] = 'A 点'
    path_list[-1][-1] = 'B 点'
    distance = 0
    success = 0
    for j in tqdm(range(10000)):
        is_ok = True
        for i in range(len(path_list) - 1):
            start_index = path_list[i][0]
            end_index = path_list[i + 1][0]
            is_pass, end_point_horizontal_error, end_point_vertical_error, after_end_point_horizontal_error, after_end_point_vertical_error, end_point_safe = judge(start_index, end_index, path_list[i][3], path_list[i][4])
            type = get_type(path_list[i + 1][0], end_point_safe)
            if is_pass:
                path_list[i + 1][1:] = end_point_horizontal_error, end_point_vertical_error, after_end_point_horizontal_error, after_end_point_vertical_error, type
            else:
                is_ok = False
                break
            distance = distance + get_distance(start_index, end_index)
        if is_ok == True:
            success = success + 1
    return success/10000

print(get_P(matlab_path_list))
```