2020 年湖南省高校第五届研究生数学建模竞赛

题目： 定向越野比赛的路线设计

摘要：

本文主要研究在**定向越野比赛线路**的各项**设计规则**下，如何灵活设计**比赛线路**、参加比赛的**参赛选手**、参赛选手**出发顺序和出发时间间隔**等要素，使之满足题目给出的各个不同的比赛情况。在**达到赛事方主要目标**的同时，兼顾各项设计准则，使之**“尽可能”**达到其最优结果，并给出最终**比赛设计方案**，为定向越野的赛事组织者**提供较好的线路**参考。

首先经过对题目的整体理解，本文确立了比赛题目模型当中的**决策变量**，并**基于比赛线路的基本设计规则添加约束或目标函数项**，通过建立的数学模型实现了对原题目的准确描述。该模型**满足题目给出的比赛线路设计规则**，在该模型的基础上根据不同题目的差异性，对基本模型进行小幅度的修改即可满足题目要求。通过 cplex **商业软件精确求解**建立模型，由于计算能力的限制，我们获得了**减去模型中某些约束后的松弛解**，可将其作为**启发式求解模型方法的初始解**使用。基于**人的知识**建立了**启发式初始化路线生成规则**，生成了一些**满足比赛线路基本约束的初始解**，仅对部分复杂约束部分尚不满足。本文考虑**启发式变邻域搜索策略**，通过建立的**评价机制**返回的评分设置了与之相对应的启发式**邻域操作规则**，以此确定了其在邻域中的**搜索方向**，**提高**了搜索到最优解的**可能性**。经过一系列启发式变邻域的搜索流程，最终找到了各问题的**近似最优解**。

问题一的求解结果表明，**选取 15 条线路**的结果**相比 20 条更加均衡**，且**最大负载人数分别为 313 和 379 人**，结果相差并不算太大，并且 20 条线路的差异过大，且有过多重复路段，以此再**提高线路条数对参赛人员的增加意义也不是太大**。

问题二的求解结果表明，**5 条、15 条和 25 条线路**的**最大参赛人数分别为 140、346 和 367 人**，说明**选取的参赛路线越多那么能够参加比赛的选手数量也越多**。但是从上图对比中可以发现，**25 条的线路方案的两个必经点距离较近**，且线路的**总路径长度也相差较大**，而 **15 条线路**的情况效果相比之下**更好一些**，且参赛人员数量相差不大，因此在本问题当中**选取 15 条线路的情况作为最优线路**。

问题三的求解结果表明，通过变邻域搜索得到基本线路之间的爬高量和线路长度差别较小，**四条路线的爬高分别为 72m、64m、77m、49m**，线路上**能够承载的最大人数分别为 91、107、104、24 人**，总体爬高量还算比较均匀说明通过此算法求得的路径能够在**很大程度上满足比赛线路设计的基本规则**。经过**遗传算法**求解，得到**参赛人员的出发顺序与人的知识判断大致相同**，发现**行进速度快的人一般均会被分派到距离较长的路线，并且有很大概率会被安排到出发顺序较为靠后的位置**。

关键词：定向越野，线路设计，整数规划，变邻域搜索，启发式算法

1 问题重述

1.1 任务背景

定向越野是一项借助地形图和指北针，按规定的顺序独立地打卡若干个检查点，并以最短的时间完成任务的一项户外运动。在定向越野比赛中，比赛路线一般由一个起点，若干个检查点和一个终点构成且一次比赛有若干条从起点到终点的比赛路线。每一条比赛路线可安排一个或多个参赛选手，选择同一条路线的参赛选手按照一定的时间间隔错时出发，不同路线的首发参赛选手同时出发。所有参赛选手须按照所选路线中检查点的顺序依次打卡，不同检查点的打卡难度也不一样。

在某次定向越野的比赛中，希望通过设计合理线路达到活动目标，并满足参赛人员情况以及场地限制，需要考虑如下规则：

（1） 不同路线的长度应尽可能均匀。

（2）不同路线的总打卡难度应尽可能均匀，检查点的难度用参赛选手需要在该点打卡花费的时间来衡量，其值越高，难度越大。

（3）不同路线的爬高量应尽可能均匀，爬高量由检查点之间大于 0 的高度差定义，路线上所有前后检查点的爬高量之和即称为该条路线的爬高量。

（4）不同路线之间应尽可能避免包含较多重复路段，也应避免参赛选手在完成任务过程中距离过近。

（5）每条路线中检查点数量不低于 20 个，设置 2 个检查点为所有路线都必经的打卡点，终点设置在难度较低的检查点。

（6）同一路线中距离相近的检查点的打卡顺序应尽可能连续，制定检查点打卡顺序时要考虑路线的长度尽可能短。

（7）同时到达同一个检查点的参赛选手数量应控制在不超过 5 人，“同时到达”指的是不同参赛选手到达同一检查点的时间差不超过 1 分钟。

（8）假定所有参赛选手的行进速度均为 6km/h,，起点、终点和不同检查点之间均存在路径可直线连接，参赛选手按照任意两点之间的直线行进。

现需要根据题目给定的比赛起点、候选检查点的位置及其难度，参照上述规则设计每条线路的途径检查点、终点、两个必经检查点以及每条线路上的人数和出发时间间隔。

1.2 问题重述

问题一：二维平面中的线路设计

在该题中无需考虑不同打卡点之间的高程差，附件 1 给出了了某次定向越野赛的起点、候选检查点的平面坐标和检查点的难度，如图 1 所示，点的颜色表示其难度的不同，绿色表示停留时间为 1 分钟的检查点，蓝色表示停留时间为 2 分钟的检查点，黄色表示停留时间为 3 分钟的检查点，红色表示停留时间为 4 分钟的检查点，编号 1 的检查点即为起点。在问题一中，期望通过设计线路使得参赛选手完成线路的平均时间不低于 2 小时，且竞赛的总完成时间最长不超过 3 小时，即最后一人完成其路线的时间不能超过 3 小时，并且使得尽可能多的参赛选手能够参与本次的竞赛。最后需给出设计线路方案中的总参赛选手数量、各条路线的参赛选手数量、检查点、总长度及总难度。

问题二：考虑三维情况的线路设计

在本题中须在问题一的基础上进一步考虑爬高量对比赛线路的影响，使每条线路的总爬高量尽可能均匀。附件 2 给出了该问题中的起点、候选检查点的三维坐标和检查点难度，具体分布如图 2 所示，其颜色代表了该点的难易程度，颜色的设定与图 1 相同。最后需给出总参赛选手数量、各条路线的参赛选手数量、检查点、总长度、总难度及爬高量。

问题三：参赛人员非等速的线路设计

该问题仍为三维问题，设定了参赛人数为 120 人，并且通过对不同参赛选手的识图能力、体能等的预估，其平均行进速度存在差异，附件 3 给出了该问题的起点和候选检查点三维坐标、检查点难度以及不同选手的速度数据，具体检查点分布如图 3 所示。在问题三中，期望通过设计线路使得参赛选手完成线路的平均时间不低于 2 小时，竞赛的总完成时间最长不超过 3.5 小时，并且使用尽可能少的线路，完成时间也尽可能短。最后需给出各条路线的参赛选手数量、检查点、总长度、总难度及爬高量。

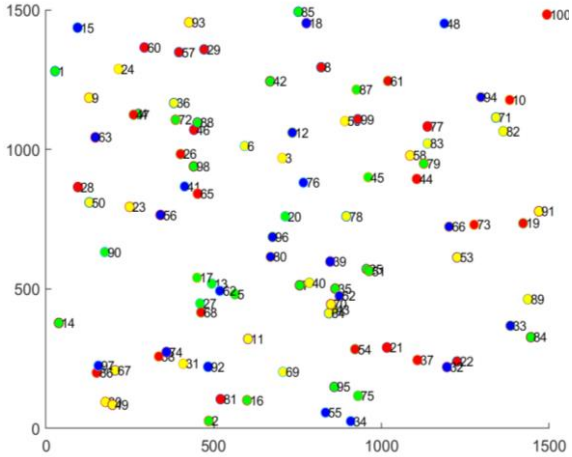

图 1 问题一检查点分布情况

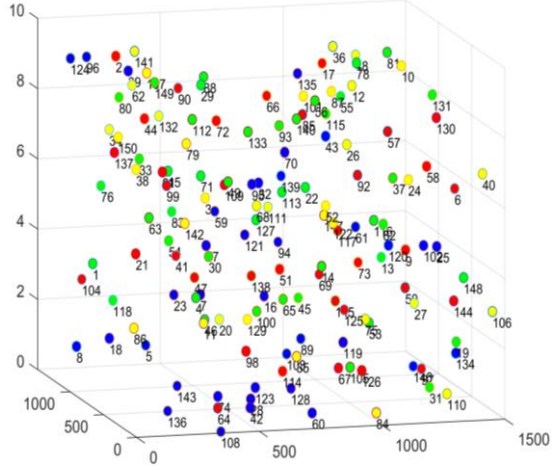

图 2 问题二检查点分布情况

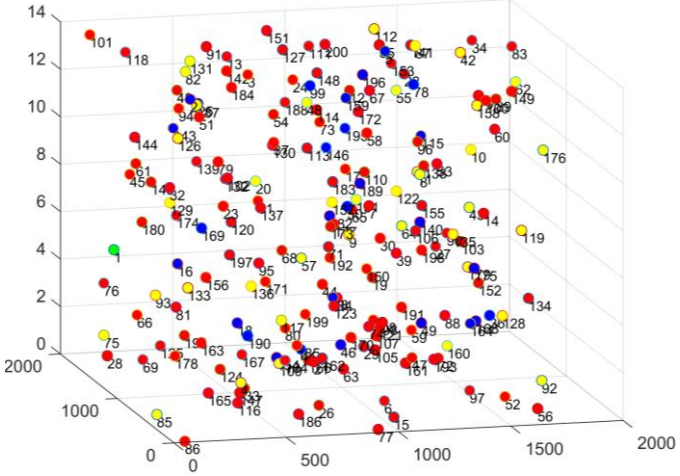

图 3 问题三检查点分布情况

2 问题假设及符号说明

2.1 问题假设

1.假设所有比赛线路都只有一个起点，若干个检查点和一个终点；
2.假设所有比赛线路的起点相同，终点也相同；
3.假设终点设在难度较低的检查点；
4.假设所有参赛选手的行进速度均为 6km/h；
5.假设路线中的存在路径均由直线连接，参赛选手也按照直线行进完成比赛；
6.假设定向越野赛的开始时间均为上午 8：00；
7.假设不考虑同时到达终点处的参赛选手数量约束。

2.2 符号说明

符号	含义	单位
r	比赛线路中的第 r 条线路	无
$r_{\max}$	比赛最大总线路数（人为设定上界）	条
R	比赛线路的集合	无
N	比赛中所有路径点编号的集合	无
e_i	终点判断变量	无
c_i	起点判断变量	无
d_i	第 i 个路径点的难度	分钟
d_1	起点的难度	分钟
$h_{i,j}$	第 i 个路径点到第 j 个路径点路径的爬高量	米
$x_{i,j}^{r}$	第 r 条路径包括 i 路径点到 j 路径点路径的判断变量	无
$S_{\max}$	最大参赛人数（人为设定上界）	人
K	参与者编号的集合	无
m_k^r	第 k 个人分配到第 r 条路线判断变量	无
Q	出发时间间隔的取值集合	无
$q_{\max}$	最大出发时间间隔（人为设定上界）	分钟
q_r	第 r 条线路的出发时间间隔	分钟
l^r	路线 r 总长度	米
d^r	路线 r 总难度	分钟
h^r	路线 r 总爬高量	米
a_r	总重复路线	条
T_i	到达 i 路径点的时间集合	无
$T_i^{'}$	到达 i 路径点的时间排序集合	无
t_j	时间排序集合中第 j 个到达路径点的时间	分钟
L	所有线路长度的集合	无
DT	所有线路难度的集合	无
H	所有线路爬高量的集合	无
v	参赛队员平均行进速度	米/分钟
v_k	第 k 个队员的平均行进速度	米/分钟
R_S	有效线路的集合	无
r_s	有效线路集合的数目	条

3 问题一模型建立与求解

3.1 问题一分析

问题一是一个二维平面中的路线设计问题（即不考虑不同打卡点之间的高程信息），通过对问题规则的分析，发现存在多数规则均包含“尽可能”的字样，因此在对这些影响线路设计的因素进行考虑时，可将其作为次要因素进行设计。但是问题一提出的线路设计目标是使得尽可能多参赛选手能够加入到此次竞赛中，因此参赛选手数目应该是我们建立模型和求解问题时关注的重中之中。对于参赛选手平均完成时间不低于 2 小时、竞赛总完成时间不超过 3 小时以及路线检查点设置的个数等明确给出具体要求的因素，需要将其放到首要位置考虑。

首先可以通过建模尝试精确求解的方法解决该问题，但是可能会由于约束过多造成求解困难，因此遇到无法求解的情况按照影响要素关键程度由“弱”到“强”的顺序依次减去获得问题的一些初始解，进而可以采用启发式邻域搜索方法对初始解进行修复，寻找满足约束条件的可行解，再对可行解进行分析找到尽可能好的设计路线。

3.2 定向越野二维平面线路设计模型

3.2.1 目标函数和决策变量的确立

通过上面对问题一的分析，线路设计的目标实际上是使得尽可能多参赛选手能够加入到此次竞赛中，考虑对参赛人数的影响因素，在竞赛总完成时间不超过 3 个小时的约束下，线路设计的好坏直接影响着最后的最大参赛人数，而在线路的设计当中决定线路的因素是终点的选取、必经点的选取和每条线路上路径点的选取，这些点的顺序确定了竞赛当中的一条条线路。因此，在设立决策变量时，首先设定三个决策变量

$$e_i=\begin{cases}1,\ i\text{为终点}\\ 0,\ otherwise\end{cases},\forall i\in N \tag{1}$$

和

$$c_i=\begin{cases}1,\ i\text{为必经点}\\ 0,\ otherwise\end{cases},\forall i\in N \tag{2}$$

以及

$$x_{i,j}^{r}=\begin{cases}1,\ r\text{路线包括路径}i\to j\\ 0,\quad otherwise\end{cases},\forall i,j\in N,r\in R \tag{3}$$

其中 $N=\{1,2,\ldots,n\}$,$R=\{1,2,\ldots,r_{\max}\}$,$r_{\max}$ 为最大线路数，N 和 R 分别表示路径点和路线的集合；e_i 为判断路径点是否为终点的决策变量；c_i 为判断路径点是否为必经点的决策变量；$x_{i,j}^{r}$ 为判断线路 r 上面是否存在路径 $i\to j$ 的决策变量。

通过上面三个决策变量，我们可以实现对路线的定义，但实际上影响最终完赛时间的影响因素还有每条线路上面分配的人数和每条线路上出发的时间间隔，因此还需设置 m_k^r 和 q^r 两个决策变量分别确定线路上参赛人员的分配和线路的时间间隔，具体表示为

$$m_k^r=\begin{cases}1,\text{第k个人分到第}r\text{条线路}\\ 0,\quad otherwise\end{cases},\forall r\in R,\forall k\in K \tag{4}$$

其中，$K=\{1,2,\ldots,S_{\max}\}$,$S_{\max}$ 为最大参加比赛人数，在该问题中可以设定一个较大的数作为上界。设定 q^r 表示第 r 条线路上的时间间隔，且 $q^r\in Q$，$Q=\{1,2,\ldots,q_{\max}\}$，$q_{\max}$ 为最大时间间隔，也可通过人为设定较大值作为上界。至此，我们完成了对决策变量的设定，选取了 5 个决定优化目标的重要影响因素，接下来考虑目标函数的具体形式。

在问题的规则当中包含了许多“尽可能”的线路设计准则，为了在具体求解过程中

能够较大可能地求得一组解，本文不将其作为硬约束进行考虑，这一点会在接下来约束条件的确立部分详细讨论。因此，将这些“尽可能”的影响因素放到目标函数部分进行综合考虑，既保证了求解的可能性，同时统筹规划了各方面影响因素。为此，我们考虑问题一中涉及到的不同路径长度应尽可能均匀、不同路线的总打卡难度应尽肯能均匀以及应尽可能避免包含较多重复路段。但是需要注意的是，由于我们人为设定了一个较大的比赛线路数目上界，这将导致可能对于有些线路并不会为其分配参赛选手，因此只需考虑有效的线路，即分配参赛选手大于等于 1 人的比赛线路，当满足下式的情况时，建立有效线路的集合 *Rs*。

$$\sum_{k\in K} m_k^r \geq 1, \forall r \in R_s \tag{5}$$

建立如下数组和变量：

$$L = \left(l^1, \cdots, l^r, \cdots l^{r_s}\right), r \in R_s \tag{6}$$

和

$$DT = \left(d^1, \cdots, d^r, \cdots d^{r_s}\right), r \in R_s \tag{7}$$

以及

$$a^r = \sum_{r_1 \in R_s} \sum_{\substack{r_2 \in R_s \\ r_2 \neq r_1}} \sum_{i\in N} \sum_{j\in N} x_{i,j}^{r_1} \cdot x_{i,j}^{r_2} \cdot l_{i,j} \tag{8}$$

其中 L 为每条线路总长度组成的数组，l^r 为 R_s 中第 r 条线路的总长度；DT 为 R_s 中每条线路总难度组成的数组，d^r 为 Rs 中第 r 条线路的总难度；a^r 为 Rs 中所有路径中重复的路径长度，$l_{i,j}$ 为路径 $i \rightarrow j$ 的长度，r_s 即为 R_s 的最大有效线路数目。

对于“尽可能均匀”的要求，可以通过方差刻画其均匀程度，对于设计路线的长度和难度的方差可以表示为 $D(L)$和 $D(DT)$，因此将其放入目标函数后的具体形式为

$$\max \omega_1 \sum_{r\in R} \sum_{k\in K} m_k^r - \omega_2 D\left(L\right) - \omega_3 D(DT) - \omega_4 a^r \tag{9}$$

其中 ω_1、ω_2、ω_3 和 ω_4 为大于 0 的权重，第一项的求和实际为有效参与比赛人数，由于第二项、第三项和第四项均为求最小值，因此前面需加上负号，根据目标的重要程度对权重进行赋值可实现多目标的优化。

3.2.2 约束条件的确立

1) 保证所有路径点中有且仅有一个终点；

$$\sum_{i\in N} e_i = 1 \tag{10}$$

2) 保证所有路径点中仅有两个必经点；

$$\sum_{i\in N} c_i = 2 \tag{11}$$

3) 保证起点不是终点；

$$e_1 = 0 \tag{12}$$

4) 保证起点不是必经点；

$$c_1 = 0 \tag{13}$$

5) 保证终点不是必经点；

$$e_i \cdot c_i = 0, \forall i \in N \tag{14}$$

6) 保证每条线路一定经过两个必经点；

$$\sum_{i\in N} \sum_{j\in N} x_{i,j}^r \cdot c_j = 2, \forall r \in R \tag{15}$$

7) 保证每条线路一定有超过 20 个检查点；

$$\sum_{i\in N}\sum_{j\in N}x_{i,j}^{r}\geq 21,\forall r\in R \tag{16}$$

8) 保证每条线路一定经过起点和终点；

$$\sum_{i\in N}x_{i,j}^{r}-\sum_{i\in N}x_{j,i}^{r}=\begin{cases}-1, & j=1\\ 1, & e_j=1\\ 0, & otherwise\end{cases},\forall j\in N,\forall r\in R \tag{17}$$

9) 保证每个参赛选手不会被分到两个及以上的线路；

$$\sum_{r\in R}m_k^r\leq 1,\forall k\in K \tag{18}$$

10) 保证每个参赛选手的平均完成时间大于等于两个小时；

$$\frac{1}{\sum_{r\in R}\sum_{k\in K}m_k^r}\sum_{r\in R}\left[\left(\frac{l^r}{v}+d^r\right)\cdot\sum_{k\in K}m_k^r+\frac{\sum_{k\in K}m_k^r\cdot\left(\sum_{k\in K}m_k^r-1\right)}{2}\cdot q^r\right]\geq 120 \tag{19}$$

其中，v 为参赛选手的平均行进速度；

11) 保证竞赛的总完成时间最长不超过 3 个小时；

$$\frac{l^r}{v}+d^r+\left(\sum_{k\in K}m_k^r-1\right)\cdot q^r\leq 180,\forall r\in R_s \tag{20}$$

12) 保证同时到达每个检查点的参赛选手数量不超过 5 人；

$$T_i=\left[t_i^{r_1},t_i^{r_1}+q^{r_1},\cdots,t_i^{r_1}+\left(\sum_{k\in K}m_k^{r_1}-1\right)\cdot q^{r_1},t_i^{r_2},t_i^{r_2}+q^{r_2},\cdots,t_i^{r_a},t_i^{r_a}+q^{r_a},\cdots,t_i^{r_a}+\left(\sum_{k\in K}m_k^{r_a}-1\right)\cdot q^{r_a}\right]$$

$$\forall i\in N,e_i=0\ or\ i\neq 1 \tag{21}$$

$$T_i^{'}=\left[t_1,t_2,\cdots,t_j,\cdots,t_b\right],\forall t_j\in T_i^{'},t_j\in T_i,t_{j+1}\geq t_j,\forall j\in\{1,2,\cdots,b-1\} \tag{22}$$

$$t_{j+4}-t_j>1,\forall j\in\{1,2,\cdots,b-4\} \tag{23}$$

其中，T_i 为所有参赛选手到达第 i 个路径点的时间的集合，$T_i^{'}$ 为 T_i 集合内到达时间由早到晚的排序集合。

3.2.4 定向越野二维平面线路设计模型建立

综上所述，定向越野二维平面线路设计模型建立为：

$$\max\ \omega_1\sum_{r\in R}\sum_{k\in K}m_k^r-\omega_2 D(L)-\omega_3 D(DT)-\omega_4 a^r$$

$$S.t.\ T_i=\left[t_i^{r_1},t_i^{r_1}+q^{r_1},\cdots,t_i^{r_1}+\left(\sum_{k\in K}m_k^{r_1}-1\right)\cdot q^{r_1},t_i^{r_2},t_i^{r_2}+q^{r_2},\cdots,t_i^{r_a},t_i^{r_a}+q^{r_a},\cdots,t_i^{r_a}+\left(\sum_{k\in K}m_k^{r_a}-1\right)\cdot q^{r_a}\right]$$

$$\forall i\in N,e_i=0\ or\ i\neq 1$$

$$T_i^{'}=\left[t_1,t_2,\cdots,t_j,\cdots,t_b\right],\forall t_j\in T_i^{'},t_j\in T_i,t_{j+1}\geq t_j,\forall j\in\{1,2,\cdots,b-1\}$$

$$t_{j+4}-t_j>1,\forall j\in\{1,2,\cdots,b-4\}$$

$$\frac{l^r}{v}+d^r+\left(\sum_{k\in K}m_k^r-1\right)\cdot q^r\leq 180,\forall r\in R_s$$

$$\frac{1}{\sum_{r\in R}\sum_{k\in K} m_k^r}\sum_{r\in R}\left[\left(\frac{l^r}{v}+d^r\right)\cdot\sum_{k\in K} m_k^r+\frac{\sum_{k\in K} m_k^r\cdot\left(\sum_{k\in K} m_k^r-1\right)}{2}\cdot q^r\right]\geq 120$$

$$\sum_{i\in N} x_{i,j}^r-\sum_{i\in N} x_{j,i}^r=\begin{cases}-1, & j=1\\ 1, & e_j=1\\ 0, & otherwise\end{cases},\ \forall j\in N,\forall r\in R$$

$$\sum_{i\in N}\sum_{j\in N} x_{i,j}^r\cdot c_j=2,\ \forall r\in R$$

$$\sum_{i\in N}\sum_{j\in N} x_{i,j}^r\geq 21,\ \forall r\in R$$

$$\sum_{r\in R} m_k^r\leq 1,\ \forall k\in K$$

$$e_i\cdot c_i=0,\ \forall i\in N$$

$$\sum_{i\in N} e_i=1$$

$$\sum_{i\in N} c_i=2$$

$$e_1=c_1=0$$

$$x_{i,j}^r\in\{0,1\},\ \forall i,j\in N,\forall r\in R$$

$$m_k^r\in\{0,1\},\ \forall k\in K,\forall r\in R$$

$$e_i\in\{0,1\},\ \forall i\in N$$

$$c_i\in\{0,1\},\ \forall i\in N$$

$$\omega_1,\omega_2,\omega_3,\omega_4\geq 0$$

3.3 定向越野二维平面线路设计模型求解

建立上述全局规划后，首先尝试对模型进行精确求解，利用 cplex 软件进行编程求解，发现无法求解得到满足所有约束的可行解，因此通过松弛掉一些约束之后得到了一些松弛解，并结合人的经验知识建立了基于启发式变邻域搜索方法的求解算法，下面介绍模型的详细求解过程。

3.3.1 生成初始解

在一般的算法设计当中，生成初始解的方法有很多，但不外乎可以归类于随机生成方法，启发式的生成方法以及通过精确求解得到复杂模型松弛解的方法。在本问题当中，一共存在 99 个备选点作为比赛线路的路径选择，且要求所有线路均存在一个相同的起点、一个相同的终点和两个相同的必经点，因此该问题实际上包括了一些启发式规则在里面，而且如果采用随机生成的方法，无法保证短时间内在如此庞大的解空间中找到一组近似可行的初始化路线作为进一步搜索最优解的基础，将造成求解困难等诸多问题。

因此，本文选择的生成初始解的策略为采用启发式生成的初始解生成和基于 cplex 商业软件求得的松弛解，这样生成的解会满足模型的一些基本约束，仅需考虑路径选择对较为不易满足的时间以及同时到达检查点人数的约束。

在启发式生成初始解的过程当中，由于起点是固定的，因此每条线路的起始点是一

样的，但是终点和中间点的选择对整个线路设计的影响是很大的。在这里我们要结合人的直观感受和经验，通过这种知识来确定终点和中间点的选择区域。在题目要求中，对于同时到达同一检查点的人数不能超过 5 人这一约束是一个有明确量化标准的硬约束，其实通过人的直观理解可以认为在检查点的停留时间越短，即检查点的难度越低，那么同时到达检查点的人数就会越少，而在所有路线中所有路线经过次数最多的检查点为设定的那两个必经点，因此在选择必经点时考虑这种启发式规则，只让必经点在难度较低的点里面选取，降低了违背此约束的不可行解的数目，提高了初始解的质量。同时，题目要求终点应选在难度较低的检查点，这样的要求将终点的选取范围进一步的缩小。考虑题目中多个“均匀性”的要求，且每条路线由于必经点的存在会被分成三个区段，通过分析图 1 中的检查点分布情况，认为终点大概率会出现在与起点相据较远的右下角区域，而两个中间点很可能会在终点与起点连线的中段，大致分布范围如图 4 所示，这样在初始解生成的步骤中，会在十分有限的几个点中选取相应的必经点和终点，而其修正过程将在下面变邻域搜索部分详细介绍。

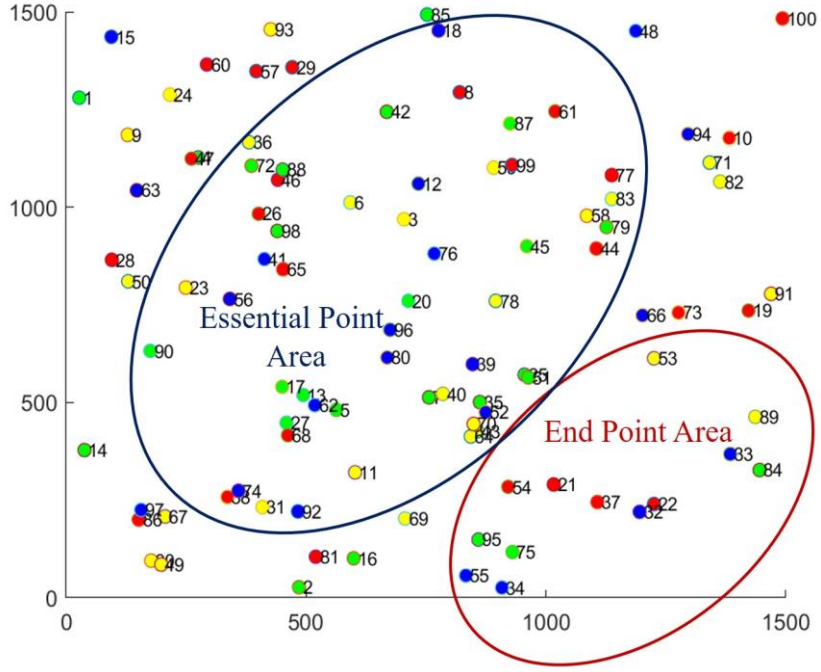

图 4 启发式初始解终点及必经点生成区域

通过 cplex 求解发现，由于一些较难约束的存在使得约束中出现非线性，无法通过求解获得可行路径，仍需通过进一步搜索寻找可行解。因为 cplex 的求解规模不能过大，因此也要求对于一些较难满足的约束先不做考虑，此时发现可以求得一些松弛解，只会不满足某一约束。此时，我们同样获得了一组较好的初始解。

3.3.2 启发式变邻域搜索策略

由于影响目标的要素，即影响线路上人数的主要因素有路径中非关键点（非终点、必经点和起点）、分配到路径的人数、参赛选手的出发间隔和路径的终点和必经点。分析这些要素对总参赛选手的影响程度，实际上改变路径选择的终点对整个线路设计的影响最大，其次是必经点，然后依次为路径、时间间隔和路径分配人数。改变除了路径分配人数之外的任何要素都会对路径的分配人数产生较大改变，因为限制路径分配人数上限的主要约束实际上是竞赛完成时间不超过 3 小时，而我们为了使参加比赛的选手数量更多，初始可将每条线路的分配人数设置为最大限制人数，接下来在启发式变邻域搜索过程中结合约束评价确定邻域变换的操作方式，关于评价指标的详细介绍将在评价指标

章节给出。

无论是启发式生成初始解方法还是基于 cplex 求解得到的初始解均满足对线路的最基本的要求，即具有相同起点、必经点和终点，检查点个数大于等于 20 个，终点不是起点等，但是对于一些复杂的约束，如平均完赛时长、最后完赛时长和同时到达人数的约束不一定满足。首先为了邻域操作的便捷，我们先通过尽可能小地改变线路设计操作进行邻域搜索，同时对于所有邻域操作，我们只通过简单的增加、减少和替换操作实现邻域搜素。邻域操作的顺序与之前给出的重要性排序恰恰相反，当发现该解不满足那三个复杂约束时，先通过给出的评价启发式地判断接下来需进行的是增加还是减少或者替换，如果该项操作不足以满足约束条件，接下来跳入影响更强的操作当中搜寻解，如果直到改变终点还不能满足，就改变终点，然后回到最初操作按设定邻域操作顺序继续搜索，具体邻域搜索的示意图如图 5 所示。

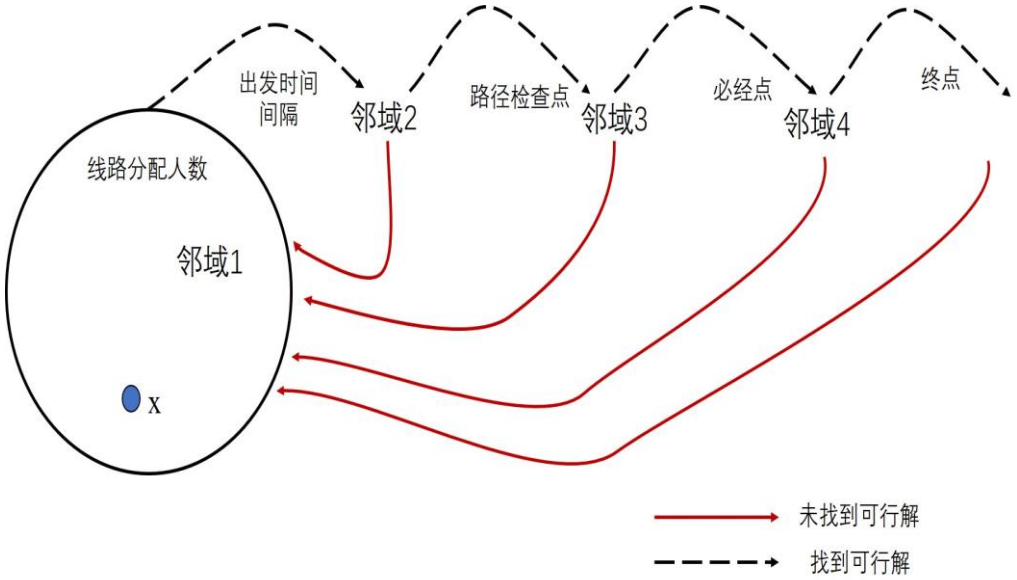

图 5 邻域搜索流程示意图

邻域搜索的具体操作只有增加、减少和替换，增加操作实际上就是通过一种随即机制确定增加某一条线路上的分配人数、某一条线路的出发时间间隔、某一条路线上的某一个路径点，而减少与替换的操作和增加机制大致相同。更改必经点和终点的操作当中，也包含了一种随机性，即从候选区域内随机选取一个点替换为原有点。在这些操作中加入这些偶然因素，会增大发现更优解的可能性。

当求得路线已经满足三个复杂约束，评价指标会给出一个评判该路线好坏的评分，根据评分确定下一步的邻域操作，直至找不到更好的解，输出搜寻过程中评分最高的路线设计方案，即完成对原问题的求解。

3.3.3 评价机制

邻域搜索的评价函数决定了启发式算法在邻域中搜索的策略，通过评价函数返回的评分可以实现对解优劣的判断，进而为搜索提供方向。我们在设计评价函数时，首先是为了满足寻找可行解，进而沿着提升方向逐步搜寻更优的的解。因此在可行解已经满足最基本约束的情况中，首先对解是否为可行解建立评价机制，考虑剩下三个不易满足的复杂约束，其涉及多个决策变量，同时这三个约束的重要性是等价的，因为如果解不能同时满足就不可能是可行解。但是我们又不能完全一棒子打死，因为同样的不可行解之间也存在很大的差异，因此在设计评价机制的时候也需要有接受劣解的能力。

我们通过分别对解是否满足上述三个约束进行打分，如若满足该约束则该项评分即为 100，但若是不满足则按照其与约束之间的差距进行打分，差异越小则评分越高。对

于不可行解，由于这三个约束之间实际上是相互影响的，因此可以随机选取一项评分不为 100 的约束项，并将其作为启发式规则代入到邻域搜素中，如随机选取最终完赛时间超过三小时，则根据人的知识进行判断应该会减少路线上的分配参赛人员，如若不满足则需增加线路的出发间隔，如若仍不满足则需进一步改变路径，甚至改变必经点和终点。但是对于可行解，我们还需考虑解的优劣，并对其进行综合的判断，这个就和我们建立模型中目标函数相一致，结合我们之前在目标函数中考虑的“尽可能”目标的权重对解的好坏进行评分。需要注意的是，后面的评分是建立在之前三个约束项评分全部为 100 的情况上的，同样地，为了接受劣解，即总评分不高但某一项评分较高的解，随机选择单项较低项进行启发式邻域搜索，针对上升方向寻找更优解，最后经过一定次数的搜寻可以将最优解输出，即完成求解任务。而建立的评价本质上通过启发式的方法决定了邻域中的操作，即搜寻最优解的方向。

3.3.4 启发式邻域搜索流程

上述的求解简化流程如图 6 所示，在图中简化了具体的启发式邻域搜索流程，详细搜索策略可参见上文中对搜索策略和评价机制的相关说明。

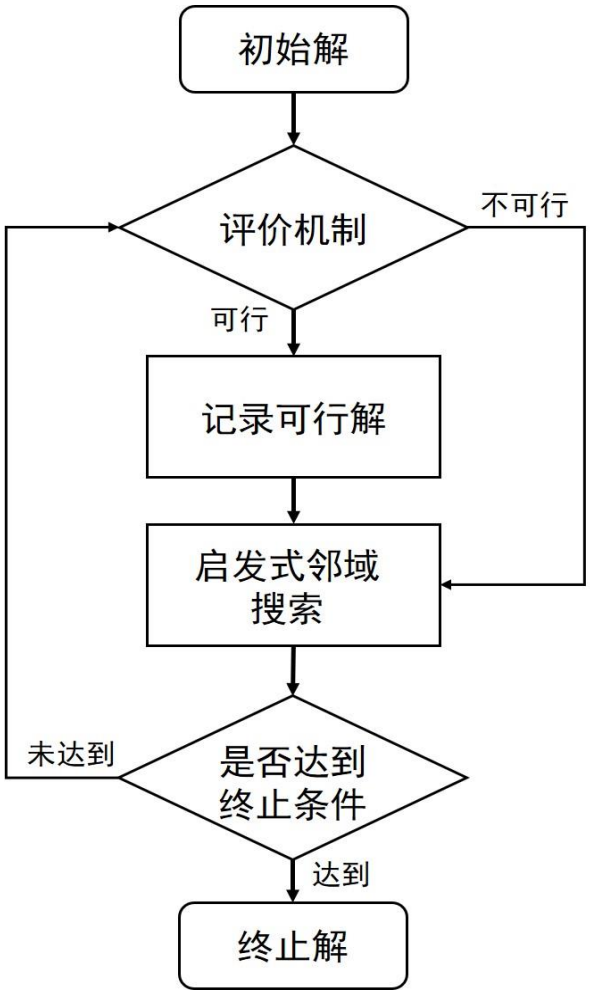

图 6 启发式邻域搜索流程简图

3.4 定向越野二维平面线路设计模型求解结果展示

我们分别对 15 条和 20 条线路进行求解，路线详情如图 7 和图 8 所示。

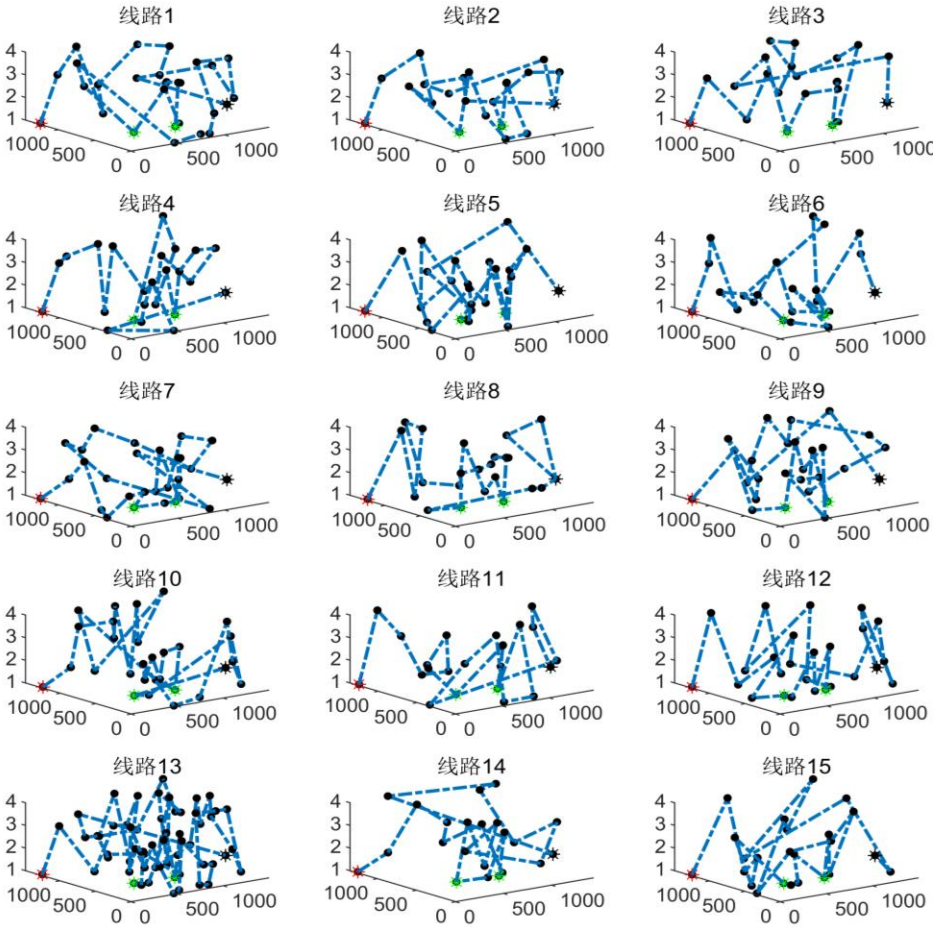

图 7 选取 15 条线路的结果展示

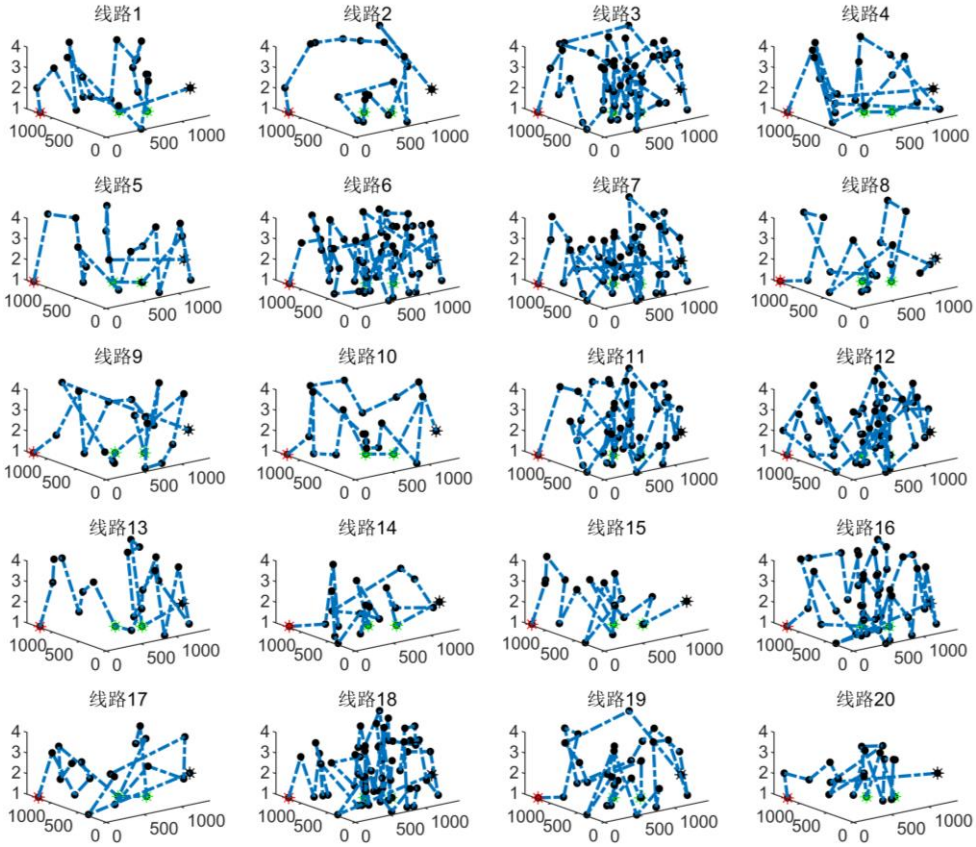

图 8 选取 20 条线路的结果展示

选取 15 条线路的结果相比 20 条更加均衡，且最大负载人数分别为 313 和 379 人，结果相差并不算太大，并且 20 条线路的差异过大，且有过多重复路段，以此再提高线路条数对参赛人员的增加意义也不是太大。

4 问题二模型建立与求解

4.1 问题二的分析

在问题一的基础上，问题二加入了检查点的三维信息，增加了对各线路爬高量的“尽可能”均匀的要求，主要目标仍然是要设计线路方案使得参赛选手尽可能的多，而且兼顾比赛设计的诸多准则，因此与问题一的求解方法类似，仅需再加入爬高量这一影响目标的因素，可参照问题一目标函数的形式，将其赋予相关权重实现对多目标问题的求解。

4.2 定向越野三维情况的线路设计模型

4.2.1 目标函数和决策变量的确立

考虑到线路的优劣形和分配到各条线路上参赛人员数量仍然是显著影响参赛人数的主要因素，因此对于问题二，选取与问题一相同的五个决策变量，分别为 e_i、c_i、$x_{i,j}^r$、m_k^r 和 q^r，其代表的含义仍然不变，表征了线路和参赛人员的分配。

对于目标函数，由于问题二变成了三维问题，还需充分考虑爬高量对线路设计的影响，因此我们先依据原题中对线路爬高量的定义，将其表示为

$$h_{i,j}=\begin{cases} z_j-z_i\ ,\ if\ \ z_j>z_i \\ 0\quad ,otherwise \end{cases} \tag{24}$$

接着可以通过下式得到每条有效线路上的总爬高量为

$$h^r=\sum_{i\in N}\sum_{j\in N}x_{i,j}^r\cdot h_{i,j}\ ,\forall r\in R_s \tag{25}$$

构建所有有效线路总爬高量的集合

$$H=\left(h^1,\cdots,h^r,\cdots h^{r_s}\right) \tag{26}$$

则可以用 $D(H)$表示所有有效线路总爬高量的方差，将其加上一个权重放入目标函数中，可将目标函数表示为

$$\max\ \omega_1\sum_{r\in R}\sum_{k\in K}m_k^r-\omega_2 D\left(L\right)-\omega_3 D(DT)-\omega_4 a^r-\omega_5 D\left(H\right) \tag{27}$$

由于要求同样是使得每条线路的总爬高量尽可能的均匀，因此在该目标函数中仍需要在非负权重 ω_5 前加上负号，通过对该目标函数的求解即可实现对问题中多目标的优化。

4.2.2 约束条件的确立

问题二中的约束条件与问题一完全相同，对于二维模型和三维模型的差异只体现在对爬高量的考虑中，因此原有约束不做改变，将只在下面给出具体形式，不再详细展开说明每条约束的具体意义，如需查看请参照问题一中的约束条件的确立部分。

4.2.3 定向越野三维情况的线路设计模型建立

综上所述，三维情况的线路设计模型建立为：

$$\max\ \omega_1\sum_{r\in R}\sum_{k\in K}m_k^r-\omega_2 D(L)-\omega_3 D(DT)-\omega_4 a^r-\omega_5 D(H)$$

$$S.t.\ T_i=\left[t_i^{r_1},t_i^{r_1}+q^{r_1},\cdots,t_i^{r_1}+\left(\sum_{k\in K}m_k^{r_1}-1\right)\cdot q^{r_1},t_i^{r_2},t_i^{r_2}+q^{r_2},\cdots,t_i^{r_a},t_i^{r_a}+q^{r_a},\cdots,t_i^{r_a}+\left(\sum_{k\in K}m_k^{r_a}-1\right)\cdot q^{r_a}\right]$$

$$\forall i\in N,\ e_i=0\ or\ i\neq 1$$

$$T_i^{'}=\left[t_1,t_2,\cdots,t_j,\cdots,t_b\right],\forall t_j\in T_i^{'},t_j\in T_i,t_{j+1}\geq t_j,\forall j\in\{1,2,\cdots,b-1\}$$

$$t_{j+4}-t_j>1,\forall j\in\{1,2,\cdots,b-4\}$$

$$\frac{l^r}{v}+d^r+\left(\sum_{k\in K}m_k^r-1\right)\cdot q^r\leq 180,\forall r\in R_s$$

$$\sum_{i\in N}x_{i,j}^r-\sum_{i\in N}x_{j,i}^r=\begin{cases}-1, & j=1\\ 1, & e_j=1\\ 0, & otherwise\end{cases},\forall j\in N,\forall r\in R$$

$$\sum_{i\in N}\sum_{j\in N}x_{i,j}^r\cdot c_j=2,\forall r\in R$$

$$\sum_{i\in N}\sum_{j\in N}x_{i,j}^r\geq 21,\forall r\in R$$

$$e_i\cdot c_i=0,\forall i\in N$$

$$\sum_{i\in N}e_i=1$$

$$\sum_{i\in N}c_i=2$$

$$e_1=c_1=0$$

$$x_{i,j}^r\in\{0,1\},\forall i,j\in N,\forall r\in R$$

$$m_k^r\in\{0,1\},\forall k\in K,\forall r\in R$$

$$e_i\in\{0,1\},\forall i\in N$$

$$\omega_1,\omega_2,\omega_3,\omega_4,\omega_5\geq 0$$

4.3 定向越野三维情况的线路设计模型求解

由于问题二与问题一的模型相比差别不大，只有目标函数部分多了有效路线爬高量的方差项，因此问题二的模型求解流程与上一问相同。区别主要体现在评价函数当中需要考虑爬高量的影响，并且在相应的邻域搜索过程当中为其增加相应的启发式规则，即规定了其搜索方向，具体的策略和方法见问题一的模型求解部分，在该部分不再赘述。

4.4 定向越野三维情况的线路设计模型求解结果展示

我们分别对 5 条、15 条和 25 条线路进行求解，路线详情如图 7、图 8 和图 10 所示

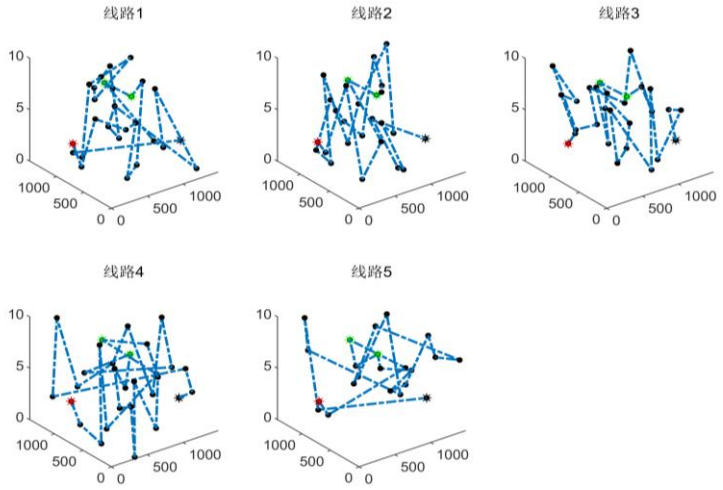

图 9 选取 5 条线路的结果展示

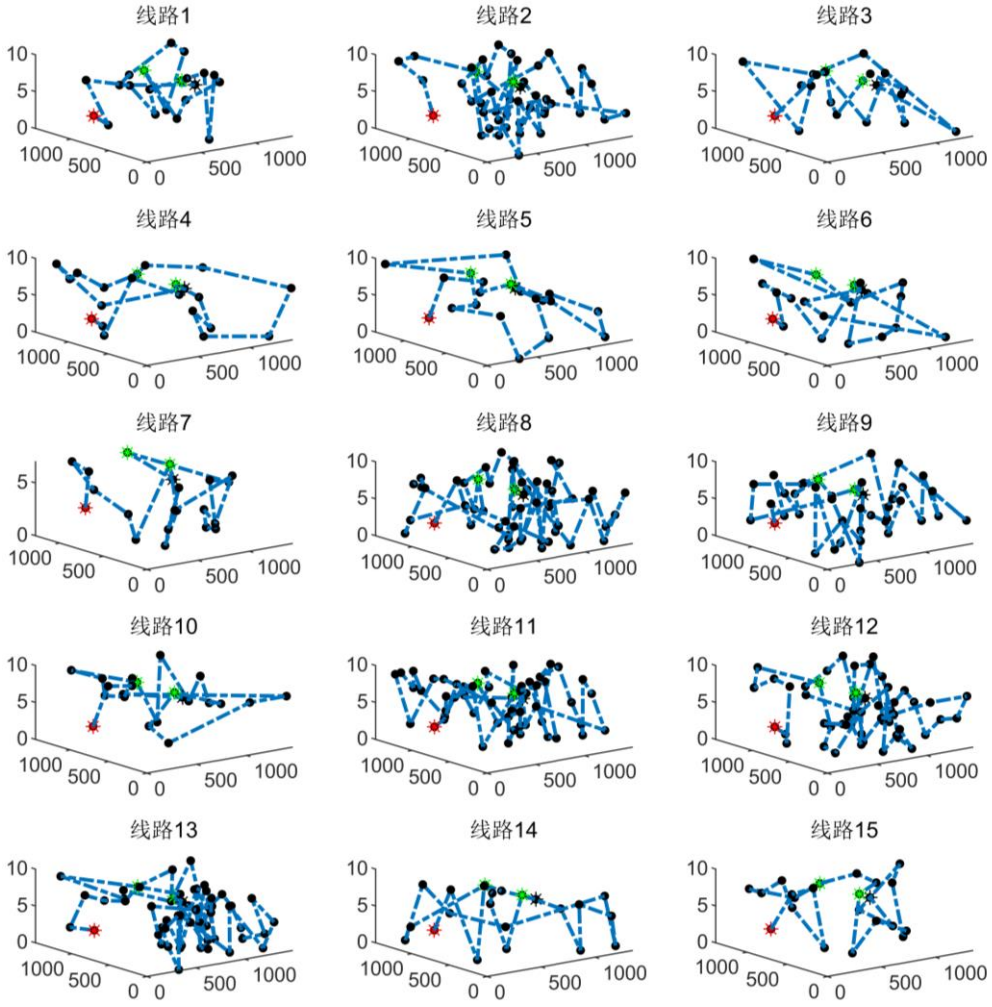

图 10 选取 15 条线路的结果展示

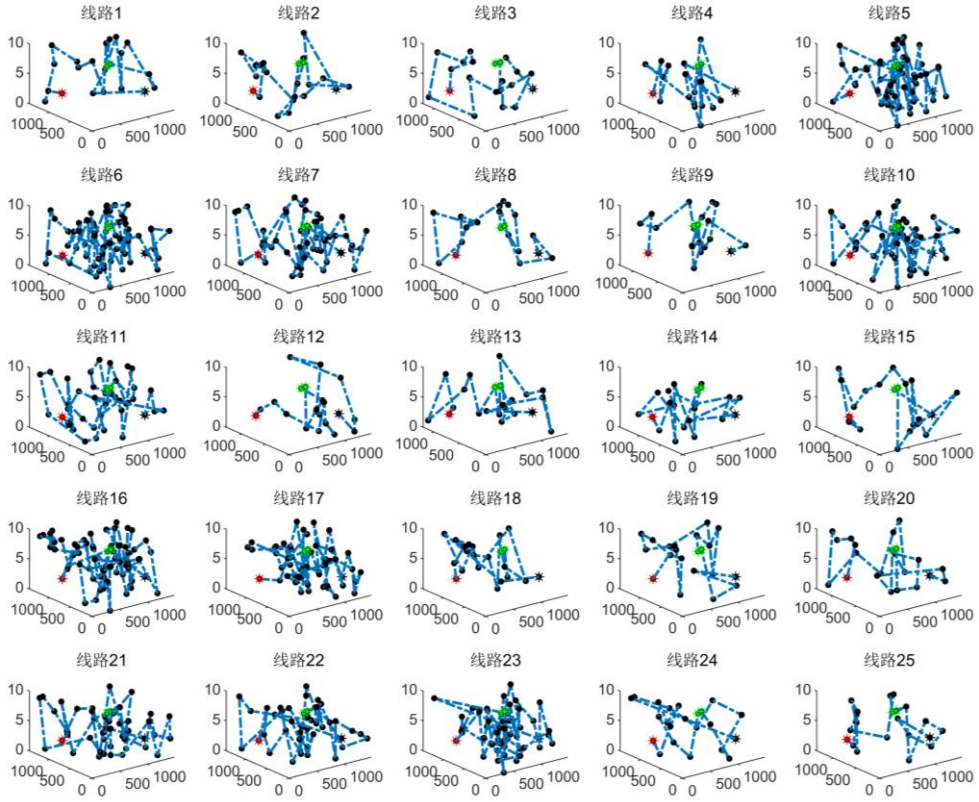

图 11 选取 25 条线路的结果展示

计算结果表明 5 条、15 条和 25 条线路的最大参赛人数分别为 140、346 和 367 人，说明选取的参赛路线越多那么能够参加比赛的选手数量也越多。但是从上图对比中可以发现，25 条的线路方案的两个必经点距离较近，且线路的总路径也相差较大，而 15 条线路的情况效果相比之下更好一些，且参赛人员数量相差不大，因此在本问题当中选取 15 条线路的情况作为最优线路。

5 问题三模型建立与求解

5.1 问题三的分析

在问题三当中，题目给出了 120 名参赛选手的行进速度差异，在该问题中，参赛人员的数目与之前问题当中作为优化目标不同，线路设计的目的变为通过设计线路使得此次竞赛用到尽可能少的比赛线路数和尽可能短的完成时间。同时，在该问题当中的参赛选手的总完成时间限制变为 3.5 个小时。在本问题当中，除了在前两问当中着重考虑的比赛设计准则，由于参赛选手间识图能力和体能等方面差异，还需充分考虑每一条比赛线路上参赛选手的出发顺序，否则可能较难满足问题的要求。

5.2 考虑体能差异的线路设计模型

5.2.1 目标函数和决策变量的确立

考虑到问题三中涉及到每条参赛路线上参赛人员出发顺序的问题，之前建立模型的决策变量已经不足以支撑描述该问题，因此我们通过新增决策变量 $Z^r_{k_1,k_2}$ 表征第 r 条路径上 k_1 在 k_2 出发顺序的前一位，具体表示如下

$$Z^r_{k_1,k_2}=\begin{cases}1, \text{第}r\text{条路径上}k_1\text{在}k_2\text{的出发顺序前一位}\\ 0, \qquad otherwise\end{cases}, \forall k_1,k_2\in K \qquad (28)$$

由于在之前的问题当中，我们优化的目标为参赛人员数量，因此我们人为地设定了一个较大的最大比赛人数 $S_{\max}$ 作为参赛选手数量的上界，在该问题中 $S_{\max}$=120，因此此

时参赛选手编号集合 K 实际上应为{1,2,…,120}。

通过上面对问题三的分析，我们已经发现此次模型优化的目标已经改变，我们需要对使用的比赛路线和完成时间进行优化，使之尽可能的少和短，因此我们在问题二的目标函数式（27）的基础上应该首先去除原先定义的有效参赛人数的目标，并且将比赛完成时间和比赛线路的条数写入到目标函数中去。

我们在之前的模型中定义过一个有效线路集合 R_s，其包含的有效路线数目为 r_s，因此可将 r_s 作为一个目标项。同样地，在之前的模型定义中我们通过使得每条路径的最后一名参赛选手的完成时间小于规定完赛时间作为必要约束，但是之前我们考虑的每名参赛选手都是相同速度的，但是对于问题三来说，每条路线的最后一名参赛选手不一定是最后完赛的，因此考虑到每名选手各自的速度，我们将下式设为目标项

$$\max_{\substack{k\in K_r\\ r\in R_s}} \frac{l^r}{v_k}+d^r+\left(w_k-1\right)\cdot q^r \tag{29}$$

式中，K_r 表示了第 r 条路线上的参赛选手编号集合；v_k 表示编号为 k 的参赛选手预估平均速度；w_k 表示了编号为 k 的参赛选手在其分配线路上的出发顺序，可通过决策变量 m_k^r 和 Z_{k_1,k_2}^r 表示出来。

因此，问题三的目标函数可以写作

$$\min\ \omega_1\left(\max_{\substack{k\in K_r\\ r\in R_s}} \frac{l^r}{v_k}+d^r+\left(w_k-1\right)\cdot q^r\right)+\omega_2 r_s+\omega_3 D\left(L\right)+\omega_4 D(DT)+\omega_5 a^r+\omega_6 D\left(H\right) \tag{30}$$

由于本问题是一个求极小值问题，这与目标函数靠后项的“尽可能”目标相吻合，因此式（30）中权重前为正号，其他未作说明的变量与已有模型保持一致，如有疑问请查阅之前说明。

5.2.2 约束条件的确立

1) 保证每名参赛选手只被分配到一条路线上；

$$\sum_{r\in R} m_k^r=1,\forall k\in K \tag{31}$$

2) 保证每条路线排序的数目等于路线上分配的人数；

$$\sum_{k\in K} m_k^r-\sum_{k_1\in K}\sum_{k_2\in K} Z_{k_1,k_2}^r=0,\forall r\in R \tag{32}$$

3) 保证最终完成竞赛时间不超过 3.5 个小时；

$$\max_{\substack{k\in K_r\\ r\in R_s}} \frac{l^r}{v_k}+d^r+\left(w_k-1\right)\cdot q^r\le 210 \tag{33}$$

4) 保证平均完成比赛时间不低于 2 个小时；

$$\frac{1}{120}\sum_{r\in R_s}\sum_{k\in K_r}\left(\frac{l^r}{v_k}+d^r+\left(w_k-1\right)\cdot q^r\right)\ge 120 \tag{34}$$

5) 保证同时到达同一检查点的参赛选手不超过 5 个；

$$T_i^{'}=[t_1,t_2,\cdots,t_j,\cdots,t_b] \tag{35}$$

$$t_{j+4}-t_j>1,\forall j\in\{1,2,\cdots,b-4\} \tag{36}$$

这里需要注意的是，虽然该约束形式并没有进行改变，$T_i^{'}$ 仍然是到达路径点 i 的参赛者时间排序，但是实际上到达该检查点的时间实际上是由每个选手沿路径计算时间得到的，而不是基于一个等差数列得到的。

上述确立约束仅为与之前模型中约束存在差异，或经过更新以及增加的约束条件，

具体所有模型约束可见下部分内容。

5.2.3 考虑体能差异的线路设计模型建立

综上所述，考虑体能差异的线路设计模型建立为：

$$\min\ \omega_1\left(\max_{\substack{k\in K_r\\ r\in R_s}}\frac{l^r}{v_k}+d^r+(w_k-1)\cdot q^r\right)+\omega_2 r_s+\omega_3 D(L)+\omega_4 D(DT)+\omega_5 a^r+\omega_6 D(H)$$

$$S.t.\ T_i^{'}=[t_1,t_2,\cdots,t_j,\cdots,t_b],\ t_{j+1}\geq t_j,\ \forall j\in\{1,2,\cdots,b-1\}$$

$$t_{j+4}-t_j>1,\ \forall j\in\{1,2,\cdots,b-4\}$$

$$\max_{\substack{k\in K_r\\ r\in R_s}}\frac{l^r}{v_k}+d^r+(w_k-1)\cdot q^r\leq 210$$

$$\frac{1}{120}\sum_{r\in R_s}\sum_{k\in K_r}\left(\frac{l^r}{v_k}+d^r+(w_k-1)\cdot q^r\right)\geq 120$$

$$\sum_{k\in K}m_k^r-\sum_{k_1\in K}\sum_{k_2\in K}Z_{k_1,k_2}^r=0,\ \forall r\in R$$

$$\sum_{i\in N}x_{i,j}^r-\sum_{i\in N}x_{j,i}^r=\begin{cases}-1, & j=1\\ 1, & e_j=1\\ 0, & otherwise\end{cases},\ \forall j\in N,\forall r\in R$$

$$\sum_{i\in N}\sum_{j\in N}x_{i,j}^r\cdot c_j=2,\ \forall r\in R$$

$$\sum_{i\in N}\sum_{j\in N}x_{i,j}^r\geq 21,\ \forall r\in R$$

$$\sum_{r\in R}m_k^r=1,\ \forall k\in K$$

$$e_i\cdot c_i=0,\ \forall i\in N$$

$$\sum_{i\in N}e_i=1$$

$$\sum_{i\in N}c_i=2$$

$$e_1=c_1=0$$

$$x_{i,j}^r\in\{0,1\},\ \forall i,j\in N,\forall r\in R$$

$$m_k^r\in\{0,1\},\ \forall k\in K,\forall r\in R$$

$$e_i\in\{0,1\},\ \forall i\in N$$

$$\omega_1,\omega_2,\omega_3,\omega_4,\omega_5,\omega_6\geq 0$$

5.3 考虑体能差异的线路设计模型求解

5.3.1 生成初始解

问题三中生成初始解的过程相比问题 1 和问题 2 还有所区别，因此在这里特意指出。在该问题当中，参赛的人数是固定的为 120 人，因此我们可以通过给每名选手进行从 1 到 120 的编号。在之前的初始解当中，由于我们是认为的设置比赛路线的最大数量，我们是希望能够达到该数量的，但实际上可能存在某些路径并没有分配参赛人员，因此这些路径实际上也不在我们考虑的范围当中。

但是本题条件下的每名参赛选手均应该能够分配到某一路线并完成比赛的。同时基

于对比赛的基本认识，可以知道跑得快的选手安排在行进速度较慢的选手后面，可能会造成追赶上并最终在某个路径点造成多余 5 人同时到达的情况出现。但是将跑得慢的选手安排在后面，可能由于等待出发的时间间隔导致无法在规定完成时间内结束比赛。因此我们在生成初始解的时候，采取两种启发式生成规则，一种是将较快选手放置到每条路线的第一个出发位置，另一种则将较快选手和较慢选手分别归在不同路线上，让快的走路线长的和慢的走路线较短的策略，而路线的生成原则与原有方法相比较基本没有变动。

5.3.2 求解方法

对问题三的求解方法，我们采用启发式变邻域搜索加遗传算法，对于启发式变邻域搜索方法的策略与之前模型基本一致，因为通过分析发现给出的所有 120 名参赛人员的平均行进速度的均值为 6km/h，这一值与前两问中计算用到的平均行进速度一致，因此即可认为通过变邻域搜索得到的比赛设计路线、参赛选手分配和各线路出发时间间隔，可以使得行进速度存在差异的相同数量参赛人员通过更改其在各条线路上出发顺序同样满足所有约束。

但是在问题三中，我们的优化目标变为了尽量少的比赛路线和尽量短的比赛完成时间，而我们之前求解模型中的主要目标是使得参赛选手尽量得多。因此，我们可以仍采用原有求解模型求得最大参加比赛人员数量在 120 人左右的比赛线路条数，这样我们即可得到满足 120 名参赛选手所需尽量少的比赛线路条数。

进一步，根据求得的比赛线路信息，我们获得了各条线路的路径、出发时间间隔以及线路分配的人数，同样可以利用启发式初始解生成方法得到初始的参赛人员分配及线路上的出发顺序，基于遗传算法实现参赛选手分配问题的优化。我们已经通过变邻域搜索实现了对尽可能少的线路数的确定，因此在设置遗传算法的适应度函数时应该考虑对“尽可能短”完赛时间的优化，通过适应度函数对最终完成比赛时间进行打分，使其趋向于完赛时间减少的顺序遗传下来，遗传算法简化流程如图 7 所示。

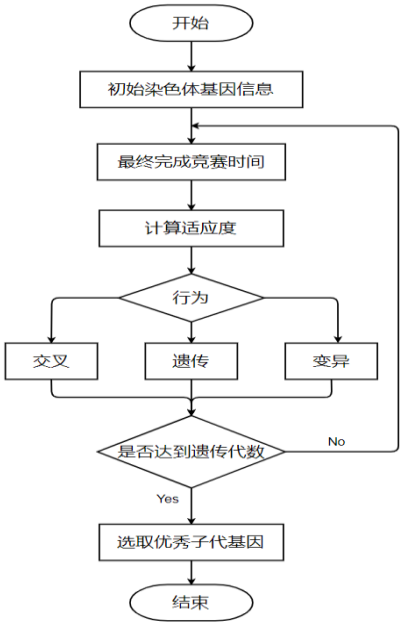

图 7 遗传算法计算流程简图

5.4 考虑体能差异的线路设计模型求解结果展示

经过启发式变邻域搜索，我们得到了线路条数为 4 条时即可满足最基本的 120 人参赛选手，具体路线图像如图 8 所示，图中红色星号和黑色星号分别表示了线路的起点和终点，黑色点为路线上的检查点。

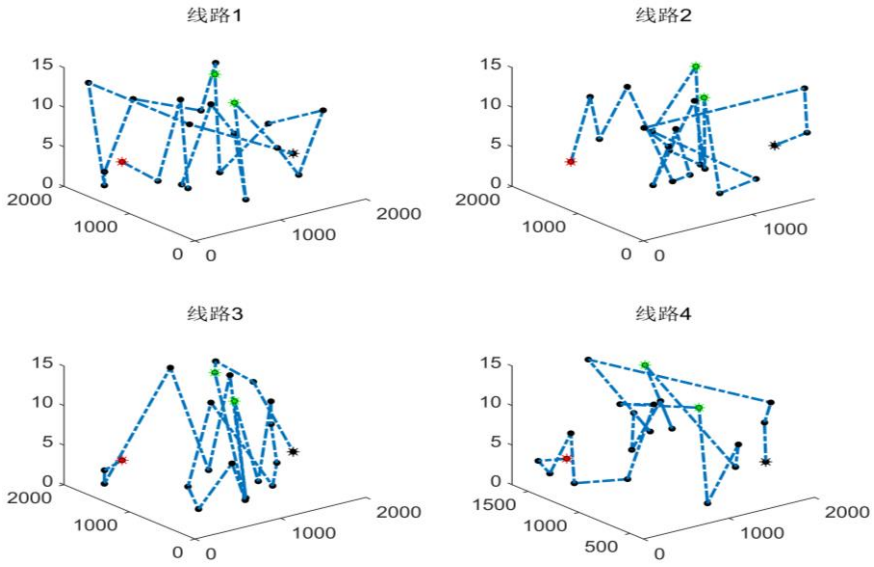

图 8 问题三比赛线路展示

由图 8 不难看出，通过变邻域搜索得到基本线路之间的爬高量和线路长度差别较小，四条路线的爬高分别为 72m、64m、77m、49m，线路上能够承载的最大人数分别为 91、107、104、24 人，总体爬高量还算比较均匀说明通过此算法求得的路径能够在很大程度上满足比赛线路设计的基本规则。

经过遗传算法求解，得到参赛人员的出发顺序与人的知识判断大致相同，发现行进速度快的人一般均会被分派到距离较长的路线，并且有很大概率会被安排到出发顺序较为靠后的位置。

参考文献

[1] 董红宇，黄敏，王兴伟,等. 变邻域搜索算法综述[J]. 控制工程, 2009(S2):1-5.
[2] 贺国光，胡学年，刘豹. 用于公交线路优化设计的四步启发式算法[J]. 系统工程学报, 1988(02):60-68.
[3] 董伟. 变邻域搜索算法研究及在组合优化中的应用[D]. 辽宁工程技术大学.
[4] 贾普俊. 山地定向越野比赛场地选择与路线图设计规律研究[D]. 山西师范大学, 2012.
[5] 张铁良，邢启明，张洁. 定向越野路线设计原则与实践[J]. 国防科技, 2018(2):123-127.

附录

附录 1 代码

附录：代码（PYTHON）

```
import numpy as np
import math
import random
import copy
import xlrd

#变邻域搜索算法
def get_route_people_number(people_number): #  求路线上有人的路线序号集合
    route_people_number=[]
    for i in range(r_max + 1):
        if people_number[i] != 0:
            route_people_number.append(i)
    return route_people_number

def get_route_number_legal(people_number):   #  求有人的路线数量
    route_people_number=get_route_people_number(people_number)
    route_number_legal=len(route_people_number)
    return route_number_legal

def get_people_number_legal(people_number):#求一个解的合法参与人数
    people_sum_legal=0
    for i in range(1,r_max+1):
        people_sum_legal=people_sum_legal+people_number[i]
    return people_sum_legal

def get_route_lenth_difficulty_legal(path,route_people_number):#求每条有人路线的总长度和总难度
-分别是一个列表
    route_lenth=[]              #求每条有人路线的长度列表(第几个数代表路线编号)
    route_difficulty = []   #求每条有人路线的难度列表
    for i in route_people_number:
        sum_lenth=0
        route_path=path[i]
        j=0
        while j>=(len(route_path)-1):
            sum_lenth=sum_lenth+distance_matrix[route_path[j]-1][route_path[j+1]-1]     #查距离
矩阵两点间的距离,编号要减一
            j=j+1
        route_lenth.append(sum_lenth)

        sum_difficulty=0
        for k in route_path:
            sum_difficulty =sum_difficulty+diff[k]                                      #查距
难度列表中各点的难度
        route_difficulty.append(sum_difficulty)
    return route_lenth,route_difficulty
```

```
def get_route_height_legal(path,route_people_number):#求每条有人路线的总爬高量-是一个列表
    route_height = []   #  求每条有人路线的爬高量列表
    for i in route_people_number:
        sum_height=0
        route_path = path[i]
        for j in range(len(route_path)-1):
            if hight[route_path[j+1]] >= hight[route_path[j]]:#  查 z 轴坐标列表中各点的 z 轴坐标
                sum_height = sum_height + hight[route_path[j+1]]- hight[route_path[j]]
        route_height.append(sum_height)
    return route_height

def get_people_time_average_latest(route_people_number,people_sum_legal,
people_number,interval,route_lenth,route_difficulty): #求合法参赛者的平均用时和最长用时
    '''
    people_time_average     #合法参赛者的平均用时
    people_time_late          #某一路线中最晚完成比赛的参与者花费时长
    people_time_latest       #合法参赛者的最长用时
    '''

    people_time_sum=0        #某条路线参赛者的总用时
    route_time_late=[]       #全部路线中最晚完成比赛的参与者花费时长
    for i in range(len(route_people_number)): #某一条有人路线
        people_time_sum=people_time_sum+(route_lenth[i] / v + route_difficulty[i]) *
people_number[route_people_number[i]]+0.5 * (people_number[route_people_number[i]]) *
(people_number[route_people_number[i]]-1) * interval[route_people_number[i]]
        people_time_late=route_lenth[i] / v + route_difficulty[i] +
(people_number[route_people_number[i]]-1) * interval[route_people_number[i]]
        route_time_late.append(people_time_late)
    people_time_average=people_time_sum/people_sum_legal
    people_time_latest=max(route_time_late)
    return people_time_average,people_time_latest

def get_checking_node_people_number(route_people_number,path,ending_node): #求有人的路线上
的有效检查点编号
    checking_node_people_number=[]
    ending_node_same=ending_node[1]                  #终点取路线编号 1 的终点
    for i in range(len(route_people_number)): #某一条有人路线
        for j in range(len(path[route_people_number[i]])):
            checking_node_people_number.append(path[route_people_number[i]][j])
    checking_node_people_number.sort()   #去除重复点,使得相同编号的点只有一个
    a=checking_node_people_number[-1]
    for i in range(len(checking_node_people_number) - 2, -1, -1):
        if a == checking_node_people_number[i]:
            checking_node_people_number.remove(checking_node_people_number[i])
        else:
            a=checking_node_people_number[i]
    checking_node_people_number.remove(1)                          #  去除起点
    checking_node_people_number.remove(ending_node_same)   #  去除终点
```

```
    return checking_node_people_number

def get_route_from_node_number(node_number,path,route_people_number):        #根据检查点编号求包含该编号的有效路线
    route_from_node_number=[]
    for i in route_people_number:
        for j in path[i]:
            if j == node_number:
                route_from_node_number.append(i)
                break
    return route_from_node_number

def time_arrive_node(node_number,route_path):#求一条路径上的参赛者最快到达某指定点的时间
    sum_lenth=0
    sum_difficulty=0
    for i in range(len(route_path)-1):
        sum_lenth=sum_lenth+distance_matrix[route_path[i]-1][route_path[i+1]-1]    #查距离矩阵两点间的距离,编号要减一
        sum_difficulty = sum_difficulty + diff[route_path[i]]                      #查距难度列表中各点的难度
        if route_path[i+1] == node_number:
            break
    time_arrive_node=sum_lenth/v+sum_difficulty
    return time_arrive_node

def get_node_illegal_number_arrived_same_time(checking_node_people_number, route_people_number,path,people_number,interval): #求不满足同时到达人数不能超过 5 个人约束条件的检查点个数
    node_illegal=0
    for i in checking_node_people_number: #某一检查点
        time_checking_node=[]
        route_from_node_number=get_route_from_node_number(i, path,route_people_number) #包含某一检查点的有效路线集合
        for j in route_from_node_number:   #包含某一检查点的有效路线序号
            for k in range(people_number[j]):
                time=time_arrive_node(i, path[j])+k*interval[j]
                time_checking_node.append(time)
        time_checking_node.sort()
        if len(time_checking_node) > 5:
            for l in range(len(time_checking_node)-4):
                if (time_checking_node[l+4]-time_checking_node[l]) <=1:
                    node_illegal=node_illegal+1
    return node_illegal

def get_people_number_maxmin_bound(route_number,people_number,path,interval): #在给定路线编号下，求该路线人数的上下界
    lenth=0
    difficulty=0
    route_people_number=get_route_people_number(people_number)
```

```
    route_lenth,route_difficulty=get_route_lenth_difficulty_legal(path, route_people_number)
    for i in range(len(route_people_number)):
        if i == route_number:
            index=route_people_number.index(i)
            lenth =route_lenth[index]
            difficulty=route_difficulty[index]
            break
    people_number_max=1 +(180 - (lenth / v + difficulty)) / interval[route_number]
    people_number_max= math.floor(people_number_max)
    people_number_min=1 +(120 - (lenth / v + difficulty)) / interval[route_number]
    people_number_min = math.ceil(people_number_min)
    return people_number_max,people_number_min

def lenth_difficulty_var_mean(route_lenth,route_difficulty):#求路线长度、难度的期望和方差
    var_lenth=np.var(route_lenth)
    var_difficulty=np.var(route_difficulty)
    mean_lenth=np.mean(route_lenth)
    return var_lenth,var_difficulty,mean_lenth

def height_var(route_height):#求路线爬高量的方差
    var_height = np.var(route_height)
    return var_height

def get_evaluate_score(solution):#求一个解的评价函数
    people_number=solution[0]
    interval=solution[1]
    path=solution[2]
    ending_node=solution[3]
    #不满足至多 5 人同时到达约束的检查点数量
    route_people_number=get_route_people_number(people_number)

checking_node_people_number=get_checking_node_people_number(route_people_number,path,en
ding_node)

node_illegal=get_node_illegal_number_arrived_same_time(checking_node_people_number,route_pe
ople_number,path,people_number,interval)
    #平均时长、最晚完成时长
    people_sum_legal=get_people_number_legal(people_number)
    route_lenth, route_difficulty=get_route_lenth_difficulty_legal(path,route_people_number)
    people_time_average,
people_time_latest=get_people_time_average_latest(route_people_number, people_sum_legal,
people_number, interval, route_lenth,route_difficulty)
    #路线长度方差、路线难度方差、路线长度期望
    var_lenth, var_difficulty, mean_lenth=lenth_difficulty_var_mean(route_lenth,route_difficulty)
    # 路线爬高量方差
    route_height=get_route_height_legal(path,route_people_number)
    var_height=height_var(route_height)
    if node_illegal==0: #5 人同时到达约束
        score_1 = 100
```

```
    else:
        score_1 = 100-5*node_illegal
    if people_time_average >= 120: #平均时长超过 120min 约束
        score_2 = 100
    else:
        score_2 = 100-2*(120-people_time_average)
    if people_time_latest <= 180:  #最晚完成时长不超过 180min 约束
        score_3 = 100
    else:
        score_3 = 100-2*(people_time_latest-180)
    score_4=score_1+score_2+score_3                                    #不合法解评价值，最大值为 300，评价值为 300 时表示解为合法解
    if score_4 <300:  #解为不合法解
        score_5 =0
        score_6 =0
    else:                  #解为合法解
        score_5 = people_sum_legal
# 总有效参与人数评价值
        score_6 = 1 / 10000 * var_lenth + var_difficulty + 1 / 1000 * mean_lenth + var_height # 多目标评价值
    return score_4,score_5,score_6

def update_solution(solution_1,solution_2):#解的更新规则
    #解的更新规则:solution_1 为原来的解,solution_2 为新产生的解
    # not_update_legal_solution_frequency 一直没有更新合法解的次数
    solution_1_score_4, solution_1_score_5, solution_1_score_6=get_evaluate_score(solution_1)
    solution_2_score_4, solution_2_score_5, solution_2_score_6=get_evaluate_score(solution_2)
    if solution_1_score_4 <300: #原来的解为不合法解
        if solution_2_score_4 >= solution_1_score_4:    #原来的解不合法，新解改善了不合法程度,此时更新解
            solution_1 = solution_2
            return solution_1, solution_2_score_4, solution_2_score_5, solution_2_score_6
        else:
            return solution_1, solution_1_score_4, solution_1_score_5, solution_1_score_6
    else:  #原来的解为合法解
        if solution_2_score_5 > solution_1_score_5:    #原来的解合法，新解增加了有效比赛人数,此时更新解（新解不合法时，有效比赛人数为 0）
            solution_1 = solution_2
            return solution_1, solution_2_score_4, solution_2_score_5, solution_2_score_6

        elif solution_2_score_5 == solution_1_score_5:#原来的解合法,新解与原来解的有效比赛人数相同
            if solution_2_score_6>=solution_1_score_6:#原来的解合法,新解与原来解的有效比赛人数相同,但新解提高多目标评价值，更新解
                solution_1 = solution_2
                return solution_1, solution_2_score_4, solution_2_score_5, solution_2_score_6
            else:
                return solution_1, solution_1_score_4, solution_1_score_5, solution_1_score_6
        else:
```

```
            return solution_1, solution_1_score_4, solution_1_score_5, solution_1_score_6
def read_xsls_hight(xlsx_path):
    """
        basic function for reading the location and difficulty from xls file.
        :param xlsx_path: .xls path
        :return: a list
        """
    data_xls = xlrd.open_workbook(xlsx_path)  # 打开此地址下的 exl 文档
    sheet_name = data_xls.sheets()[0]    # 进入第一张表
    count_nrows = sheet_name.nrows    # 获取总行数
    count_nocls = sheet_name.ncols  # 获得总列数
    line_value = sheet_name.row_values(0)
    data = []
    diff = [0]
    hight = [0]
    for i in range(1,count_nrows):
        data_1 = {}
        for j in range(1,count_nocls-1):
                cell = sheet_name.cell(i, j)
                if cell.ctype == 2 and cell.value % 1 == 0.0:  # 如果是整形
                    cell_value = int(cell.value)
                    data_1[line_value[j]] = cell_value
                else:
                    cell_value = float(cell.value)
                    data_1[line_value[j]] = cell_value
        data.append(data_1)
        diff_cell = sheet_name.cell(i,count_nocls-2)
        diff.append(int(diff_cell.value))
        hight_cell = sheet_name.cell(i,count_nocls-3)
        hight.append(int(hight_cell.value))
    return data, diff, hight

def three_dimension_distance(data,i,j):
    distance = (data[i]['X 坐标(米)']-data[j]['X 坐标(米)'])**2 + (data[i]['Y 坐标(米)']-data[j]['Y 坐标
(米)'])**2 +\
                (data[i]['Z 坐标(米）']-data[j]['Z 坐标(米）'])**2
    return round(distance**(1/2))

def distance_matrix_1(data):
    dist = []
    distance_matrix = []
    for i in range(200):
        cur_distance_matrix = []
        for j in range(200):
            dist.append(three_dimension_distance(data, i, j))
            # cur_time_matrix.append(two_dimension_distance(data,i,j)/100)
            cur_distance_matrix.append(three_dimension_distance(data, i, j))
        distance_matrix.append(cur_distance_matrix)
    return distance_matrix
```

```
def local_search_2_opt(Xdata, city_tour):
    new = [1]
    re_path = copy.deepcopy(city_tour)
    re_path.remove(1)
    re_path.remove(city_tour[-1])
    for i in range(len(city_tour)-2):
        d = []
        for j in range(len(re_path)):
            d.append(three_dimension_distance(Xdata,new[-1]-1,re_path[j]-1))
        temp = d.index(min(d))
        new.append(re_path[temp])
        re_path.remove(re_path[temp])
    new.append(city_tour[-1])
    return new

def add_node(data,tour):
    path = copy.deepcopy(tour)
    r = random.randint(1,200)
    while r in path:
        r = random.randint(1, 200)
    path.insert(random.randint(1,len(path)-1),r)
    new_p = local_search_2_opt(data,path)
    new_path = []
    for i in range(len(new_p)):
        new_path.append(new_p[i])
    return new_path

def del_node(tour,cur_must_goes):
    path = copy.deepcopy(tour)
    r = random.randint(1,len(tour)-1)
    while tour[r] in cur_must_goes:
        r = random.randint(1, len(tour) - 1)
    path.remove(path[r])
    return path

def add_or_del(data,tour,necessity_node,pos=0.4):
    r = random.random()
    if r > pos:
        path = add_node(data,tour)
    elif r <= pos and len(tour)>=22:
        path = del_node(tour,necessity_node)
    else:
        path = tour
    return path

def
update_must_go(sol,must_go_candidate=[41,46,18,190,20,136,117,54,109,38,157,195,159,50,9,182,2
2]):
```

```
    new_sol = copy.deepcopy(sol)
    path = copy.deepcopy(new_sol[2])
    cur = copy.deepcopy(new_sol[4])
    temp1 = random.choice(must_go_candidate)
    temp2 = random.choice(cur)
    while temp1 in cur:
        temp1 = random.choice(must_go_candidate)
    for i in range(1,len(path)):
        if temp1 not in path[i]:
            path[i][path[i].index(temp2)] = temp1
    new_sol[2] = path
    cur[new_sol[4].index(temp2)] = temp1
    new_sol[4] = copy.deepcopy(cur)
    return new_sol

def update_destination(sol,destination_candidate = [158,36,179,168,62,92,128,175,115,140,87,42]):
    new_sol = copy.deepcopy(sol)
    path = copy.deepcopy(new_sol[2])
    cur_destination = new_sol[3][3]
    temp1 = random.choice(destination_candidate)
    while temp1 == cur_destination:
        temp1 = random.choice(destination_candidate)
    for i in range(1,len(path)):
        path[i][-1] = temp1
        new_sol[3][i] = temp1
    new_sol[2] = copy.deepcopy(path)
    return new_sol

def update_route(sol,data):
    new_sol = copy.deepcopy(sol)
    must_goes = copy.deepcopy(new_sol[4])
    path = copy.deepcopy(new_sol[2])
    change_route = random.randint(1,len(new_sol[0])-1)
    new_route = add_or_del(data,path[change_route],must_goes)
    path[change_route] = new_route
    new_sol[2] = copy.deepcopy(path)
    return new_sol

def update_people_number(sol):
    new_sol = copy.deepcopy(sol)
    people_num = copy.deepcopy(new_sol[0])
    change_route = random.randint(1,len(new_sol[0])-1)
    ub,lb = get_people_number_maxmin_bound(change_route,people_num,new_sol[2],new_sol[1])
    new_num = random.randint(lb,ub)
    while new_num == people_num[change_route]:
        new_num = random.randint(lb, ub)
    people_num[change_route] = new_num
    new_sol[0] = copy.deepcopy(people_num)
    return new_sol
```

```
def update_interval(sol,time_slot_candidate = [1,2,3,4,5,6,7,8,9,10]):
    new_sol = copy.deepcopy(sol)
    cur_inter = copy.deepcopy(new_sol[1])
    change_inter = random.randint(1, len(new_sol[0])-1)
    new_inter = random.choice(time_slot_candidate)
    while new_inter == cur_inter[change_inter]:
        new_inter = random.choice(time_slot_candidate)
    cur_inter[change_inter] = new_inter
    new_sol[1] = copy.deepcopy(cur_inter)
    return new_sol

global data
global diff
global distance_matrix
global hight
data, diff,hight=read_xsls_hight('附件 3.xlsx')
distance_matrix=distance_matrix_1(data)

#  必经点、终点和时间间隔
must_go_point_candidate = [41,46,18,190,20,136,117,54,109,38,157,195,159,50,9,182,22]
destination_candidate = [158,36,179,168,62,92,128,175,115,140,87,42]
time_slot_candidate = [1,2,3,4,5,6,7,8,9,10]
# cur_must_go_point = [46,133]
#cur_destination = [84]

n=200        #路径点数量
global v
v=100        #速度为 6 km/h=100 m/min
global r_max
r_max=10   #最大路线设置
t_max=10     #最大时间间隔

#生成初始解
people_number=[0 for i in range(r_max+1)]     #解的选手数量取值
interval=[0 for i in range(r_max+1)]              #解的出发时间间隔取值
necessity_node=[[] for i in range(r_max+1)] #解的必经点取值
ending_node=[0 for i in range(r_max+1)]        #解的终点取值
path=[[] for i in range(r_max+1)]                  #解的路径点顺序取值
route_people_number=[] #有人的路线集合编号

must_goes = []
for i in range(2):
    r = must_go_point_candidate[random.randint(0, len(must_go_point_candidate) - 1)]
    while r in must_goes:
        r = must_go_point_candidate[random.randint(0, len(must_go_point_candidate) - 1)]
    must_goes.append(r)
dest = destination_candidate[random.randint(0,len(destination_candidate)-1)]
sol = []
```

```
for i in range(1,r_max+1):
    people_number[i] = random.randint(20,30)
    interval[i] = time_slot_candidate[random.randint(0,len(time_slot_candidate)-1)] # 随机确定一个时间间隔
    necessity_node[i] = must_goes
    ending_node[i] = dest
    path[i].append(1)
    path[i].extend(must_goes)
    for j in range(18):
        r = random.randint(2,200)
        while r in must_goes or r == dest or r in path[i]:
            r = random.randint(2,200)
        path[i].append(r)
    path[i].append(ending_node[i])
    path[i] = local_search_2_opt(data,path[i])
solution_1 = [people_number,interval,path,ending_node,must_goes]

Neighborhood_number=[1,2,3,4,5]                          #邻域动作编号
Neighborhood_search_frequency=[50,20,500,80,10]      #邻域动作执行最大次数
i=1
while i<=5:
    if i == 1:  # 人数邻域动作
        for j in range(Neighborhood_search_frequency[i-1]):
            solution_2=update_people_number(solution_1)
            # solution_3 = update_solution(solution_1, solution_2)
            solution_3, solution_3_score_4, solution_3_score_5, solution_3_score_6=update_solution(solution_1, solution_2)
            route_number_legal=get_route_number_legal(solution_3[0])
            if solution_3 == solution_2:  # 更新解
                solution_1 = solution_3
                # print(solution_1)
                i = 1
                break
            else:  # 不更新解
                # print(solution_1)
                if j == Neighborhood_search_frequency[i-1] - 1:
                    i = 2
            print(route_number_legal,solution_3_score_4, solution_3_score_5, solution_3_score_6)
    elif i == 2:  # 时间间隔邻域动作
        for j in range(Neighborhood_search_frequency[i-1]):
            solution_2=update_interval(solution_1)
            # solution_3 = update_solution(solution_1, solution_2)
            solution_3, solution_3_score_4, solution_3_score_5, solution_3_score_6 = update_solution(solution_1,solution_2)
            route_number_legal = get_route_number_legal(solution_3[0])
            if solution_3 == solution_2:  # 更新解
                solution_1 = solution_3
                # print(solution_1)
                i = 1
```

```
                    break
                else:  # 不更新解
                    # print(solution_1)
                    if j == Neighborhood_search_frequency[i-1] - 1:
                        i = 3
                print(route_number_legal,solution_3_score_4, solution_3_score_5, solution_3_score_6)

        elif i == 3:  # 路径顺序邻域动作
            for j in range(Neighborhood_search_frequency[i-1]):
                solution_2=update_route(solution_1,data)
                # solution_3 = update_solution(solution_1, solution_2)
                solution_3, solution_3_score_4, solution_3_score_5, solution_3_score_6 =
update_solution(solution_1,solution_2)
                route_number_legal = get_route_number_legal(solution_3[0])
                if solution_3 == solution_2:  # 更新解
                    solution_1 = solution_3
                    # print(solution_1)
                    i = 1
                    break
                else:  # 不更新解
                    # print(solution_1)
                    if j == Neighborhood_search_frequency[i-1] - 1:
                        i = 4
                print(route_number_legal,solution_3_score_4, solution_3_score_5, solution_3_score_6)
        elif i == 4:  # 必经点邻域动作
            for j in range(Neighborhood_search_frequency[i-1]):
                solution_2 = update_must_go(solution_1)
                # solution_3 = update_solution(solution_1, solution_2)
                solution_3, solution_3_score_4, solution_3_score_5, solution_3_score_6 =
update_solution(solution_1,solution_2)
                route_number_legal = get_route_number_legal(solution_3[0])
                if solution_3 == solution_2:  # 更新解
                    solution_1 = solution_3
                    # print(solution_1)
                    i = 1
                    break
                else:  # 不更新解
                    # print(solution_1)
                    if j == Neighborhood_search_frequency[i-1] - 1:
                        i = 5
                print(route_number_legal,solution_3_score_4, solution_3_score_5, solution_3_score_6)

        else:  # 终点点邻域动作
            for j in range(Neighborhood_search_frequency[i-1]):
                solution_2 = update_destination(solution_1)
                # solution_3 = update_solution(solution_1, solution_2)
                solution_3, solution_3_score_4, solution_3_score_5, solution_3_score_6 =
update_solution(solution_1,solution_2)
                route_number_legal = get_route_number_legal(solution_3[0])
```

```
                if solution_3 == solution_2:   #  更新解
                    solution_1 = solution_3
                    # print(solution_1)
                    i = 1
                    break
                else:   #  不更新解
                    # print(solution_1)
                    if j == Neighborhood_search_frequency[i-1] - 1:
                        i =6
                print(route_number_legal,solution_3_score_4, solution_3_score_5, solution_3_score_6)
print(solution_1)

#遗传算法
import random
import numpy as np
from numpy.linalg import solve
import math
import xlrd
# import cluster
import copy
import xlrd

def read_xsls_speed(xlsx_path):
    """
        basic function for reading the location and difficulty from xls file.
        :param xlsx_path: .xls path
        :return: a list
        """
    data_xls = xlrd.open_workbook(xlsx_path)   #  打开此地址下的 exl 文档
    sheet_name = data_xls.sheets()[1]    #  进入第一张表
    print(sheet_name)
    count_nrows = sheet_name.nrows    #  获取总行数
    count_nocls = sheet_name.ncols   #  获得总列数
    line_value = sheet_name.row_values(0)
    speed = [0]
    for i in range(1,count_nrows):
        speed_cell = sheet_name.cell(i,count_nocls-1)
        speed.append(float(speed_cell.value))
    return speed

global speed_list
speed_list = read_xsls_speed('附件 3.xlsx')

# dataset = np.array(loadDataSet('data.xls', splitChar=','))

class psyGeneticAlorithm():
    def __init__(self, popsize, crosspro, mutationpro, iternumber, pathnum, size):

        self.popsize = popsize
```

```
        self.crosspro = crosspro
        self.mutationpro = mutationpro
        self.iternumber = iternumber
        # self.chromosome1 = []
        self.chromosome2 = []
        self.pathnumber = pathnum
        self.size = size
        self.bestfit = 0

    def initial_population(self):
        """
        种群初始化 grefenstette 编码
        :return:
        """
        for i in range(self.popsize):
            temp = [k for k in range(1,121)]
            temp1 = []
            for k in range(self.pathnumber-1):
                cur_people = []
                for j in range(int(120/self.pathnumber)):
                    can_p = random.randint(0,len(temp)-1)
                    cur_people.append(temp[can_p])
                    temp.remove(temp[can_p])
                temp1.append(cur_people)
            temp1.append(temp)
            self.chromosome2.append(temp1)

    def crossover(self):
        crossnumber = int(self.popsize * self.crosspro / 2) * 2
        crossoverindexs = random.sample(range(1, self.popsize), crossnumber)
        for i in range(int(crossnumber / 2)):
            self.singlepointcross(crossoverindexs[i], crossoverindexs[crossnumber - i - 1])

    def singlepointcross(self, index1, index2):
        pathindex = random.randint(1,self.pathnumber - 1)
        list1 = copy.deepcopy(self.chromosome2[index1][pathindex])
        list2 = copy.deepcopy(self.chromosome2[index2][pathindex])
        # list1 = self.chromosome2[index1][uavindex].copy()
        # list2 = self.chromosome2[index2][uavindex].copy()
        min_len = min(len(list1) - 1, len(list2) - 1)
        k1 = random.randint(0, min_len)
        k2 = random.randint(0, min_len)
        if k2 < k1:   # todo 一定要记得改!!!
            t = k2
            k2 = k1
            k1 = t
        k11 = k1
        k22 = k1
        fragment1 = list1[k1: k2]
```

```
fragment2 = list2[k1: k2]
new1 = []
new2 = []
for pos in fragment1:
    if pos not in list2:
        if fragment2[fragment1.index(pos)] in new2:
            new2.append(fragment2[new2.index(fragment2[fragment1.index(pos)])])
        else:
            new2.append(fragment2[fragment1.index(pos)])
    else:
        if fragment2[fragment1.index(pos)] not in new2:
            new2.append(fragment2[fragment1.index(pos)])
        else:
            new2.append(fragment2[new2.index(fragment2[fragment1.index(pos)])])
for pos in fragment2:
    if pos not in list1:
        if fragment1[fragment2.index(pos)] in new1:
            new1.append(fragment1[new1.index(fragment1[fragment2.index(pos)])])
        else:
            new1.append(fragment1[fragment2.index(pos)])
    else:
        if fragment1[fragment2.index(pos)] not in new1:
            new1.append(fragment1[fragment2.index(pos)])
        else:
            new1.append(fragment1[new1.index(fragment1[fragment2.index(pos)])])

list1[k1: k2] = new2
list2[k1: k2] = new1
del list1[k1: k2]
left1 = list1
offspring1 = []
for pos in left1:
    if pos in new1:
        pos = fragment1[new1.index(pos)]
        while pos in new1:
            pos = fragment1[new1.index(pos)]
        offspring1.append(pos)
        continue
    offspring1.append(pos)
for i in range(0, len(new1)):
    offspring1.insert(k11, new1[i])
    k11 += 1
# self.chromosome2[index1][uavindex] = offspring1.copy()
self.chromosome2[index1][pathindex] = copy.deepcopy(offspring1)
del list2[k1: k2]
left2 = list2
offspring2 = []
for pos in left2:
    if pos in new2:
```

```
                pos = fragment2[new2.index(pos)]
                while pos in new2:
                    pos = fragment2[new2.index(pos)]
                offspring2.append(pos)
                continue
            offspring2.append(pos)
        for i in range(0, len(new2)):
            offspring2.insert(k22, new2[i])
            k22 += 1
        # self.chromosome2[index2][uavindex] = offspring2.copy()
        self.chromosome2[index2][pathindex] = copy.deepcopy(offspring2)

    def mutation(self):
        mutationnumber = int(self.popsize * self.mutationpro)
        crossoverindexs = random.sample(range(1, self.popsize), mutationnumber)
        for i in range(mutationnumber):
            self.addmutation(crossoverindexs[i])

    def addmutation(self,index):
        length = []
        for i in range(self.pathnumber):
            length.append(len(self.chromosome2[index][i]))
        pathindex = np.argmin(length)
        # offspring = self.chromosome2[index].copy()
        offspring = copy.deepcopy(self.chromosome2[index])
        # cur_uav = self.chromosome2[index][uavindex].copy()
        cur_path = copy.deepcopy(self.chromosome2[index][pathindex])
        temp = [k for k in range(1, 121)]
        for i in cur_path:
            temp.remove(i)
        can = temp[random.randint(0,len(temp)-1)]
        for i in range(self.pathnumber):
            if can in self.chromosome2[index][i]:
                self.chromosome2[index][i].remove(can)
        in_index = random.randint(0,len(cur_path)-1)
        cur_path.insert(in_index,can)
        self.chromosome2[index][pathindex] = copy.deepcopy(cur_path)

    def decode(self, index):
        """
        :param index:个体索引
        :return:  剩余时间
        """
        print("------------------------------")
        print("开始计算新的个体适应度")
        fitness = random.randint(-10,20)
        return fitness

    def select(self):
```

```
        """
        选择：采用锦标赛
        :return:
        """
        fitness = np.array([self.decode(i) for i in range(0, self.popsize)])

        # 精英保留
        bestIndex = np.argmax(fitness)
        self.bestfit = fitness[bestIndex]
        print("精英个体的适应度：",self.bestfit)
        print(self.chromosome2[bestIndex])
        newpopulation = []
        newpopulation.append(self.chromosome2[bestIndex].copy())
        for i in range(self.popsize - 1):
            indexset = random.sample(range(self.popsize), self.size)
            bestIndex = indexset[np.argmax(fitness[indexset])]
            newpopulation.append(self.chromosome2[bestIndex][:])

        self.chromosome2 = newpopulation


    def run(self):
        self.initial_population()
        for i in range(self.iternumber):
            print("------------------------------------------------------------")
            print("第",i,"轮迭代:")
            self.crossover()
            self.mutation()
            self.select()
        bestIndex = 0
        for i in range(0, self.popsize):
            temp = self.decode(i)
            if self.bestfit < temp:
                self.bestfit = temp
                bestIndex = i
        print('当前最优适应度：', str(self.bestfit))
        print(self.chromosome2[bestIndex])
        return self.chromosome2[bestIndex]

psyGa = psyGeneticAlorithm(60,0.2,0.1,100,4,4)
psyGa.run()

import numpy as np
import xlrd



def read_xsls_hight(xlsx_path):
    """
```

```
        basic function for reading the location and difficulty from xls file.
        :param xlsx_path: .xls path
        :return: a list
        """
    data_xls = xlrd.open_workbook(xlsx_path)  # 打开此地址下的 exl 文档
    sheet_name = data_xls.sheets()[0]    # 进入第一张表
    count_nrows = sheet_name.nrows    # 获取总行数
    count_nocls = sheet_name.ncols  # 获得总列数
    line_value = sheet_name.row_values(0)
    data = []
    diff = [0]
    hight = [0]
    for i in range(1,count_nrows):
        data_1 = {}
        for j in range(1,count_nocls-1):
                cell = sheet_name.cell(i, j)
                if cell.ctype == 2 and cell.value % 1 == 0.0:  # 如果是整形
                    cell_value = int(cell.value)
                    data_1[line_value[j]] = cell_value
                else:
                    cell_value = float(cell.value)
                    data_1[line_value[j]] = cell_value
        data.append(data_1)
        diff_cell = sheet_name.cell(i,count_nocls-2)
        diff.append(int(diff_cell.value))
        hight_cell = sheet_name.cell(i,count_nocls-3)
        hight.append(int(hight_cell.value))
    return data, diff, hight

def three_dimension_distance(data,i,j):
    distance = (data[i]['X 坐标(米)']-data[j]['X 坐标(米)'])**2 + (data[i]['Y 坐标(米)']-data[j]['Y 坐标
(米)'])**2 +\
                (data[i]['Z 坐标(米）']-data[j]['Z 坐标(米）'])**2
    return round(distance**(1/2))

def distance_matrix_1(data):
    dist = []
    distance_matrix = []
    for i in range(200):
        cur_distance_matrix = []
        for j in range(200):
            dist.append(three_dimension_distance(data, i, j))
            # cur_time_matrix.append(two_dimension_distance(data,i,j)/100)
            cur_distance_matrix.append(three_dimension_distance(data, i, j))
        distance_matrix.append(cur_distance_matrix)
    return distance_matrix

def get_route_people_number(): # 求路线上有人的路线序号集合
    route_people_number=[i for i in range(1, r_max + 1)]
```

```
    return route_people_number

def get_route_lenth_difficulty_legal(path,route_people_number):#求每条有人路线的总长度和总难度-分别是一个列表
    route_lenth=[]              #求每条有人路线的长度列表(第几个数代表路线编号)
    route_difficulty = []   #求每条有人路线的难度列表
    for i in route_people_number:
        sum_lenth=0
        route_path=path[i]
        j=0
        while j>=(len(route_path)-1):
            sum_lenth=sum_lenth+distance_matrix[route_path[j]-1][route_path[j+1]-1]    #查距离矩阵两点间的距离,编号要减一
            j=j+1
        route_lenth.append(sum_lenth)

        sum_difficulty=0
        for k in route_path:
            sum_difficulty =sum_difficulty+diff[k]                                    #查距难度列表中各点的难度
        route_difficulty.append(sum_difficulty)
    return route_lenth,route_difficulty

def get_route_height_legal(path,route_people_number):#求每条有人路线的总爬高量-是一个列表
    route_height = []   #  求每条有人路线的爬高量列表
    for i in route_people_number:
        sum_height=0
        route_path = path[i]
        for j in range(len(route_path)-1):
            if hight[route_path[j+1]] >= hight[route_path[j]]:#  查 z 轴坐标列表中各点的 z 轴坐标
                sum_height = sum_height + hight[route_path[j+1]]- hight[route_path[j]]
        route_height.append(sum_height)
    return route_height

def get_people_number_legal():#求一个解的合法参与人数
    people_sum_legal=120
    return people_sum_legal

def get_people_time_average_latest(route_people_number,people_sum_legal,solution,interval,route_lenth,route_difficulty): #求合法参赛者的平均用时和最长用时
    '''
    people_time_average    #合法参赛者的平均用时
    people_time_late       #某一路线中最晚完成比赛的参与者花费时长
    people_time_latest     #合法参赛者的最长用时
    '''
    people_time_sum=0       #某条路线参赛者的总用时
    time_per_people=[]      #全部路线中每个参赛者花费时长
    for i in range(len(route_people_number)): #某一条有人路线
```

```
            route_people_sum=len(solution[i])
            route_people_sequence=solution[i]
            for j in route_people_sequence:
                time_per_people.append(route_lenth[i] / speed_list[j] + route_difficulty[i] +
(route_people_sum-1)*interval[route_people_number[i]])
        for k in range(len(time_per_people)):
            people_time_sum=people_time_sum+time_per_people[i]
        people_time_average=people_time_sum/people_sum_legal
        people_time_latest=max(time_per_people)
        return people_time_average,people_time_latest

def lenth_difficulty_var_mean(route_lenth,route_difficulty):#求路线长度、难度的期望和方差
        var_lenth=np.var(route_lenth)
        var_difficulty=np.var(route_difficulty)
        mean_lenth=np.mean(route_lenth)
        return var_lenth,var_difficulty,mean_lenth

def height_var(route_height):#求路线爬高量的方差
        var_height = np.var(route_height)
        return var_height


def get_route_from_node_number(node_number,path,route_people_number):        #根据检查点编
号求包含该编号的有效路线
        route_from_node_number=[]
        for i in route_people_number:
            for j in path[i-1]:
                if j == node_number:
                    route_from_node_number.append(i)
                    break
        return route_from_node_number

def get_checking_node_people_number(route_people_number,path,ending_node): #求有人的路线上
的有效检查点编号
        checking_node_people_number=[]
        ending_node_same=ending_node[1]                #终点取路线编号 1 的终点
        print(route_people_number)
        for i in range(len(route_people_number)-1): #某一条有人路线
            # print(i)
            for j in range(len(path[route_people_number[i]])):
                # print(path[route_people_number[i]])
                # print(j)
                checking_node_people_number.append(path[route_people_number[i]][j])
                # print(checking_node_people_number)
        checking_node_people_number.sort()  #去除重复点,使得相同编号的点只有一个
        a=checking_node_people_number[-1]
        for i in range(len(checking_node_people_number) - 2, -1, -1):
            if a == checking_node_people_number[i]:
                checking_node_people_number.remove(checking_node_people_number[i])
```

```
            else:
                a=checking_node_people_number[i]
        checking_node_people_number.remove(1)                          # 去除起点
        checking_node_people_number.remove(ending_node_same)   # 去除终点
        return checking_node_people_number

def lenth_difficulty_arrive_node(node_number,route_path):#求一条路径上的到达指定点的距离和长度
        sum_lenth=0
        sum_difficulty=0
        for i in range(len(route_path)-1):
            sum_lenth=sum_lenth+distance_matrix[route_path[i]-1][route_path[i+1]-1]    #查距离矩阵两点间的距离,编号要减一
            sum_difficulty = sum_difficulty + diff[route_path[i]]                                #查距难度列表中各点的难度
            if route_path[i+1] == node_number:
                break
        return sum_lenth,sum_difficulty

def get_node_illegal_number_arrived_same_time(solution,checking_node_people_number, route_people_number,path,interval): #求不满足同时到达人数不能超过5个人约束条件的检查点个数
        node_illegal=0
        for i in checking_node_people_number: #某一检查点
            time_checking_node=[]
            route_from_node_number=get_route_from_node_number(i, path,route_people_number) #包含某一检查点的有效路线集合
            for j in route_from_node_number:   #包含某一检查点的有效路线序号
                for k in solution[j-1]: #人的编号
                    route_people_sum = len(solution[j-1])
                    sum_lenth, sum_difficulty=lenth_difficulty_arrive_node(i,path[j])
                    time_checking_node.append(sum_lenth / speed_list[k] + sum_difficulty + (route_people_sum - 1) * interval[route_people_number[i]])
            time_checking_node.sort()
            if len(time_checking_node) > 5:
                for l in range(len(time_checking_node)-4):
                    if (time_checking_node[l+4]-time_checking_node[l]) <=1:
                        node_illegal=node_illegal+1
        return node_illegal

def get_fitness(solution):#求一个解的评价函数
        #不满足至多 5 人同时到达约束的检查点数量
        route_people_number=get_route_people_number()

checking_node_people_number=get_checking_node_people_number(route_people_number,path,ending_node)
        node_illegal=get_node_illegal_number_arrived_same_time(solution, checking_node_people_number, route_people_number, path,interval)

        #平均时长、最晚完成时长
```

```
    people_sum_legal = get_people_number_legal()
    route_lenth, route_difficulty = get_route_lenth_difficulty_legal(path, route_people_number)
    people_time_average,
people_time_latest=get_people_time_average_latest(route_people_number,people_sum_legal,solutio
n,interval,route_lenth,route_difficulty)

    #路线长度方差、路线难度方差、路线长度期望
    var_lenth, var_difficulty, mean_lenth=lenth_difficulty_var_mean(route_lenth,route_difficulty)

    # 路线爬高量方差
    route_height=get_route_height_legal(path,route_people_number)
    var_height=height_var(route_height)

    if node_illegal==0: #5 人同时到达约束
        score_1 = 10000
    else:
        score_1 = 10000-5*node_illegal

    if people_time_average >= 120: #平均时长超过 120min 约束
        score_2 = 10000
    else:
        score_2 = 10000-2*(120-people_time_average)

    if people_time_latest <= 210:   #最晚完成时长不超过 210min 约束
        score_3 = 10000
    else:
        score_3 = 10000-2*(people_time_latest-210)
    score_4=score_1+score_2+score_3   #不合法解评价值，最大值为 30000，评价值为 30000 时表
示解为合法解
    if score_4 <30000:   #解为不合法解
        score_5 =0
        score_6 =0
    else:                    #解为合法解
        score_5 = 500-people_time_latest
#  最晚完成时间
        score_6 = 500-(1 / 10000 * var_lenth + var_difficulty + 1 / 1000 * mean_lenth + var_height)
#  多目标评价值
    fitness=score_1+score_2+score_3+score_4+score_5+score_6
    return fitness

def read_xsls_speed(xlsx_path):
    """
        basic function for reading the location and difficulty from xls file.
        :param xlsx_path: .xls path
        :return: a list
        """
    data_xls = xlrd.open_workbook(xlsx_path)   # 打开此地址下的 exl 文档
    sheet_name = data_xls.sheets()[1]     # 进入第一张表
    print(sheet_name)
```

```
    count_nrows = sheet_name.nrows     # 获取总行数
    count_nocls = sheet_name.ncols   # 获得总列数
    line_value = sheet_name.row_values(0)
    speed = [0]
    for i in range(1,count_nrows):
        speed_cell = sheet_name.cell(i,count_nocls-1)
        speed.append(float(speed_cell.value))
    return speed

global speed_list
speed_list = read_xsls_speed('附件 3.xlsx')

global data
global diff
global distance_matrix
global hight

data, diff,hight=read_xsls_hight('附件 3.xlsx')
distance_matrix=distance_matrix_1(data)

global r_max
r_max=4

global path
path=[[1,81,142,109,154,114,7,6,159,106,164,176,14,161,22,87,110,40,66,69,101,51,175],
      [1,40,169,13,41,46,182,84,161,22,192,97,6,159,38,99,190,102,94,149,175],
      [1,66,69,111,29,5,6,159,74,55,8,168,97,114,109,147,57,21,22,87,175],
      [1,76,125,169,178,18,3,82,159,6,135,168,22,152,43,84,183,182,151,89,175]]

global interval
interval=[0,1,1,1,1]

global ending_node
ending_node=[0,175,175,175,175]
solution=[[62, 96, 55, 46, 15, 34, 47, 14, 85, 86, 92, 1, 16, 93, 108], [80, 20, 107, 33, 101, 43, 17, 94, 57,
118, 103, 27, 2, 32, 71, 81], [36, 22, 8, 113, 87, 82, 90, 42, 3, 56, 61, 69, 53, 48, 7, 37], [11, 19, 29, 45, 49,
50, 54, 70, 74, 75, 77, 100, 102, 110, 115]]
fitness=get_fitness(solution)
print(fitness)
```