2020 年湖南省研究生数学建模竞赛

竞赛承诺书

我们仔细阅读了湖南省研究生数学建模竞赛的竞赛规则。

我们完全明白，在竞赛开始后参赛队员不能以任何方式（包括电话、电子邮件、网上咨询等）与队外的任何人（包括指导教师）研究、讨论与赛题有关的问题。

我们知道，抄袭别人的成果是违反竞赛规则的，如果引用别人的成果或其他公开的资料（包括网上查到的资料），必须按照规定的参考文献的表述方式在正文引用处和参考文献中明确列出。

我们郑重承诺，严格遵守竞赛规则，以保证竞赛的公正、公平性。如有违反竞赛规则的行为，我们将受到严肃处理。

我们授权湖南省研究生数学建模竞赛组委会，可将我们的论文以任何形式进行公开展示（包括进行网上公示，在书籍、期刊和其他媒体进行正式或非正式发表等）。

我们参赛选择的题号是（从组委会提供的试题中选择一项填写）：B

我们的参赛报名号为（如果组委会设置报名号的话）：202018001005

所属学校（请填写完整的全名）：中国人民解放军国防科技大学

参赛队员（打印并签名）：1. 省瑞靖

2. 陈刚

3. 杨从林

指导教师或指导教师组负责人 （打印并签名）：杨忠伟

日期： 2020 年 11 月 30 日

评阅编号（由组委会评阅前进行编号）：

评阅编号：
（由组委会填写）

2020年湖南省高校第五届研究生数学建模竞赛

编 号 专 用 页

评阅记录：

评阅人						
备注						

（请勿改动此页内容和格式。此编号专用页仅供评阅使用。）

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：本文在对题目所给的定向越野问题背景介绍和附件数据的基础上，对定向越野中的路线规划设计进行深入分析，建立了求解问题的多目标优化模型并采用 NSGA-III 为主要求解算法，TOPSIS、爬山算法，提出的双原则排序等为主要辅助算法进行约束优化问题的求解分析，具体包含：

针对问题一，将路线以可变长度向量化表示，向量中的每个数值表示该路线所经过的检查点的序号。约束优化问题的建模过程以完整的路线向量为计算求解单元或对象。并据此依次分析考虑定向越野路线设计的约束条件和目标函数，将约束条件和目标函数利用路线向量进行数学语言描述，最后得到多目标优化模型。利用 NSGA-III 主算法，加爬山算法作为辅助，提出“双原则”排序法（即成为当前点的下一个点需遵循(1)该点距离当前点尽可能近；(2)距离终点尽可能远）进行多目标优化问题的求解。最终得到参赛选手的总数量为 269 人，路线的总条数为 23 条，两个必经检查点的序号为 55 和 10。问题一的详细结果见附件 1。

针对问题二，将每个检查点的爬高量纳入目标函数，每条路线的爬高量计算为所有点“上坡量”的和，建立同问题相似的约束优化模型。求解过程与问题一类似，最终得到参赛选手的总数量为 426 人，路线的总条数为 39 条，问题二的详细结果见附件 2，两个必经检查点的序号为 134 和 80。

针对问题三，增加考虑题目所给的人数 120 人、完成时间下限和上限等约束条件和路线尽可能少的目标函数。首先根据路线长度的均匀程度、路线难度均匀程度、路线的爬高均匀程度（每种均匀性以最大值与最小值之差来衡量）三个指标结合终点的难度等其他约束条件利用 TOPSIS 进行终点的初步筛选，将出发时间间隔设置为前后两个人完成时间差的最大值以初步保证解的可行性，然后将 120 人和已知的路线转化为人员分配问题，编码时考虑种群中每个个体包含路线和时间间隔两类属性，以 NSGA-III-DE 算法进行求解。最终得到路线上参赛选手的的编号，路线检查点等结果数据见附件 3。

综上所述，本文在对实际问题合理简化的基础上，对定向越野路线规划设计进行了深入分析，建立了约束优化求解模型，并利用 NSGA-III 为主算法进行求解，得到合理科学的定向越野路线，计算过程数据详实，可追溯前馈。

关键词：定向越野，检查点，路线向量，路线爬高量，约束优化，NSGA-III，双排序原则，爬山算法，TOPSIS，交叉变异

1 问题重述

1.1 问题背景

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，从竞技比赛到团建活动，已受到越来越多的人们的喜爱。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。一次比赛安排若干条从起点到终点的比赛路线，所有路线的起点和终点相同。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不一样，通常与打卡点附近的地形和打卡要求有关。路线中设置每一条路线都必须经过的检查点，为参赛选手提供医疗或物资保障，以便掌握参赛选手的状况。该竞技性运动受诸多的规则限制。

1.2 问题提出

根据以上背景及规则，结合附件所给各个检查点的坐标及难度值等数据，需要建模解决以下问题：

（1）不考虑不同打卡点之间的高程信息，已知数据为给出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（时间度量），参赛选手的速度都为 6km/h, 假设期望本次活动中，参赛选手完成任务的平均时间不低于 2 小时，竞赛的总完成时间最长不超过 3 个小时，需要为本次定向越野比赛设计比赛路线、路线的终点、必经检查点，使得尽可能多参赛选手能够加入到此次竞赛中。计算给出总参赛选手数量、各条路线的参赛选手数量、检查点、总长度及总难度。

（2）已知定向越野赛的起点、候选检查点的三维坐标和检查点的难度，进一步要求每条比赛路线的爬高量尽可能均匀，在此条件下重新考虑问题 1。建模求解总参赛选手数量、各条路线的参赛选手数量、检查点、总长度、总难度及爬高量等重要信息。

（3）已知某次定向越野比赛的起点和候选检查点三维坐标和检查点难度数据，此次活动有 120 名参赛选手参赛，通过事先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，不同参赛选手的速度数据已知。本次比赛期望参赛选手平均完成时间不低于 2 小时，比赛总完成时间不超过 3.5 个小时，建模设计最佳比赛路线，使得使用的路线尽可能少，完成的时间尽可能短。需要建模计算各条路线的参赛选手数量、检查点、总长度、总难度及爬高量等数据信息。

2 问题分析

2.1 问题一的分析

问题 1 简化了实际问题，考虑整个比赛场地为二维平面，即不考虑检查点的 Z 坐标，并且任意两检查点之间都可达。参赛队员在两点之间按直线行进。待优化目标有：不同路线的长度尽可能均匀；不同路线总打卡难度尽可能均匀；尽量避免重复路段；终点的难度尽可能低；避免参赛选手在完成任务过程中距离过近；参赛队员数尽可能多。需要求解计算总参赛选手数量、各条路线的参赛选手数量、检查点、总长度及总难度等数据。属于多约束多目标的优化问题。

2.1 问题二的分析

问题二在问题一的基础上增加了需要考虑的高程因素。也因此增加了一个可优化的目标：不同路线的爬高量应尽可能均匀。需要求解计算总参赛选手数量、各条路线的参赛选手数量、检查点、总长度及总难度等数据。

2.1 问题三的分析

问题三给定此次活动的参赛选手是 120 人，并且每个参赛队员的行进速度存在各不相同。路线规划设计的约束条件因此增加了：所有参赛选手平均完成时间不低于 2 小时，比赛总完成时间不超过 3.5 个小时。需要优化的目标也因此增加了：建模设计最佳比赛路线，使得使用的路线尽可能少，完成的时间尽可能短。需要建模计算各条路线的参赛选手数量、检查点、总长度、总难度及爬高量等数据信息。

3 符号说明

3.1 符号说明与使用

表 1　主要变量符号说明

符号	含义
X_i	第 i 条路线
N	路线的总条数
x_{ij}	第 i 条路线中的第 j 个检查点的序号
$D(X_i)$	路线 X_i 的总长度
$d^i_{j(j+1)}$	第 i 条路线中第 j 和第 $j+1$ 个点之间的距离
$t(x_{ij})$	点 x_{ij} 的难度值(单位：分钟)
$T_i(X_i)$	路线 X_i 总的打卡难度
$\lvert x_{ij}\rvert$	点 x_{ij} 的序号值
v	问题一的队员速度 6km/h
v_{ij}	第 i 条路线中第 j 个队员的行进速度
Δt_i	路线 X_i 的出发时间间隔
$P(X_i)$	路线 X_i 的参赛选手数量
$H(X_i)$	路线 X_i 的爬高量

4 模型的建立与求解

4.1 问题一模型的建立与求解

约束优化问题的建模过程以完整的路线为对象入手。设第 i 条路线是上的第 j 个点的序号为 x_{ij}，共 N 条路线，则第 i 条参赛路线 X_i 表示为：

$$X_i = (x_{i1}, x_{i2}, ..., x_{ij}, ..., x_{in_i}) \tag{1}$$

其中 n_i 表示第 i 条路线上所有检查点的总个数。由任意两点 A、B 之间的距离公式

$$d = \sqrt{(x_A - x_B)^2 + (y_A - y_B)^2} \tag{2}$$

可以计算路线中任意两点之间的距离。则 X_i 的路线总长度 $D(X_i)$：

$$D(X_i) = \sum_{j=1}^{n_i - 1} d^i_{j(j+1)} \tag{3}$$

其中 $d^i_{j(j+1)}$ 表示第 i 条路线中第 j 和第 $j+1$ 个点之间的距离。设点 x_{ij} 的难度值（以时间度

量）为$t(x_{ij})$，有：

$$t(x_{ij}) \in \{1,2,3,4\} \tag{4}$$

则路线X_i总的打卡难度$T_i(X_i)$：

$$T_i(X_i)=\sum_{j=1}^{n_i} t(x_{ij}) \tag{5}$$

比赛线路的设计需要遵守系列的规则和可优化的目标，依次考虑每一个规则。不同路线的长度应尽可能均匀，则有如下优化目标函数：

$$\min \ \{\max_i(D(X_i)) - \min_i(D(X_i))\} \tag{6}$$

不同路线的总打卡难度应尽可能均匀，则有如下优化目标函数：

$$\min \ \{\max_i(T(X_i)) - \min_i(T(X_i))\} \tag{7}$$

不同的参赛路线中可能存在重复路径，重复路径示意图如下图所示：

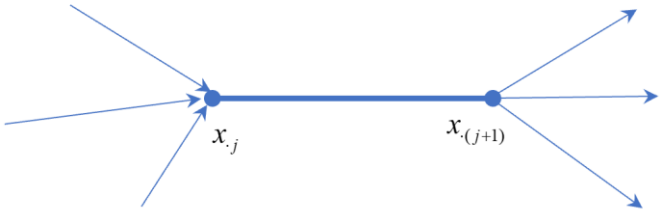

图 1　重复路径示意图

其中$x_{\cdot j}, x_{\cdot(j+1)}$分别表示重复路径线段的起点和终点，路线$X_i$中的第$j$条路径可以表示为：

$$E_{ij} =< x_{ij}, x_{i(j+1)} > \tag{8}$$

令包含检查点最多的路线为第一条路线，则对于其余路线X_i中，令：

$$E_{1j}(X_i) = \begin{cases} 1, E_{ij} \in X_i \\ 0, else \end{cases}, i = (2,3,...,N) \tag{9}$$

于是，为了尽量避免重复路径，考虑所有路线总的重复路径数越少越好，得到以下优化目标函数：

$$\min \sum_{i=2}^{N} \sum_{j=1}^{n_1} E_{1j}(X_i) \tag{10}$$

每条路线中检查点的数量不低于 20 个，检查点的总数为 100 个：

$$\min_i(n_i) \geq 20 \ and \ \max_i(n_i) \leq 100 \tag{11}$$

所有线路都必须经过其中 2 个检查点，必经打卡点通常用于补给水和医疗保障，以便及时了解各参赛选手的情况，设必经点分别为 A、B，表示为x_{im}, x_{in}，令$|x_{ij}|$表示点x_{ij}的序号值，则：

$$|x_{im}| \in \forall X_i \ and \ |x_{in}| \in \forall X_i \tag{12}$$

问题一中所给数据 100 个候选检查点的二维平面示意图如图 2 所示。

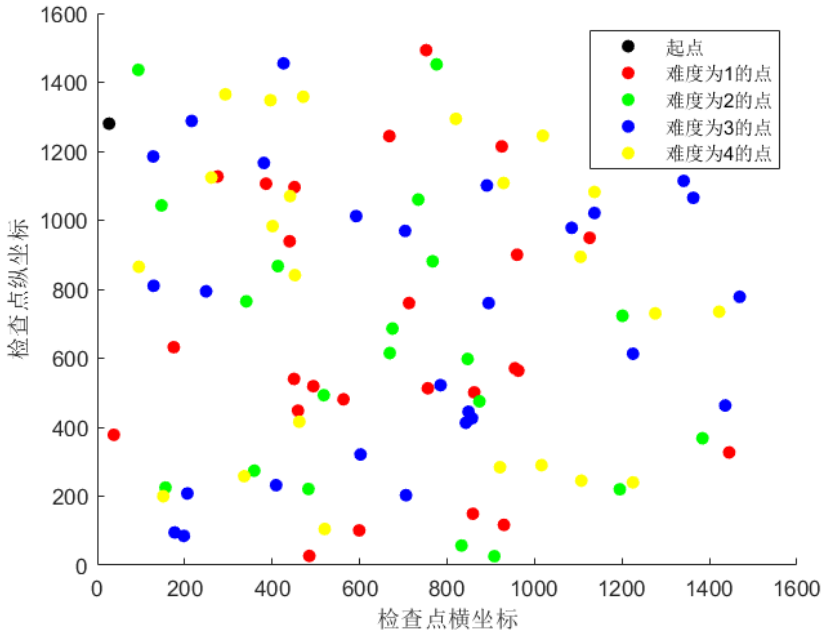

图 2　问题一 100 个检查点的二维平面示意图

由图可知候选检查点的分布具有很强的随机性，考虑以下两条原则：(1)比赛全局中应尽可能均衡的掌握队员的状况以便更好的实施医疗等保障，即必经检查点 A，B 应避免距离过近；(2)尽可能避免不同路线的重复路段。综合分析，A、B 两必经检查点的位置示意图应如图 3 中方案(a)所示。其中 A、B 为必经检查点，O 为起点，P 为终点。

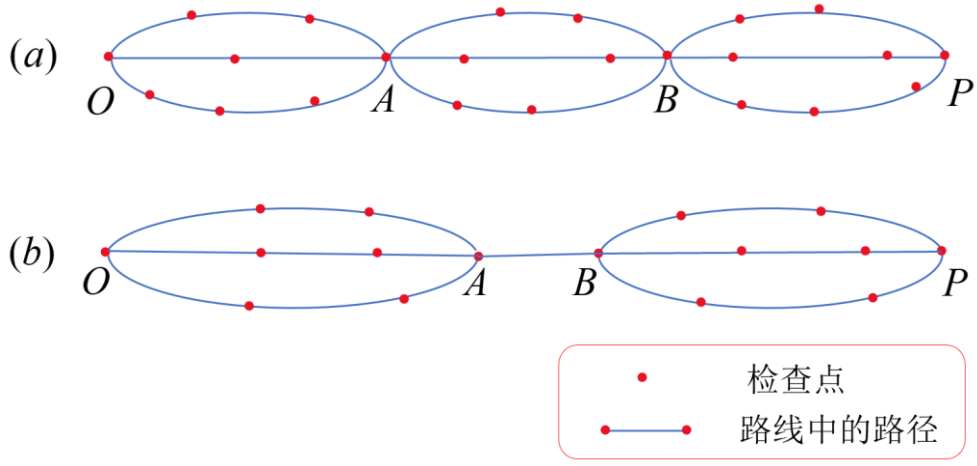

图 3　必经检查点 A、B 选择方案(a)较为合理(仅示意，实际路径为直线)

即 A 点的位置尽量接近路线总时间的$\frac{1}{3}$位置处，B 点尽量位于总时间的$\frac{2}{3}$处：

$$\begin{aligned}&\min abs((\frac{1}{1000v}(\sum_{j=1}^{m-1}d^{i}_{j(j+1)})+\sum_{j=1}^{m}t(x_{ij}))-N\cdot\frac{1}{3}\cdot\overline{T}(X_i))\\&\min abs((\frac{1}{1000v}(\sum_{j=1}^{n-1}d^{i}_{j(j+1)})+\sum_{j=1}^{n}t(x_{ij}))-N\cdot\frac{2}{3}\cdot\overline{T}(X_i))\end{aligned}\tag{13}$$

其中，$v=6\,\mathrm{km/h}$，$\overline{T}(X_i)$表示参赛选手完成任务的平均时间，$abs(\cdot)$表示绝对值。终点 P 设置在难度较低的检查点：

$$\min_i t(x_{in_i})\tag{14}$$

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

$$\min_i(\max_i D(X_i)) \tag{15}$$

$D(X_i)$用(3)式计算。

设到达路线X_i的第j个点的时间为$t_i(x_{ij})$，该时间由路线中路径的时间以及打卡点的难度（时间度量）两部分组成，其计算表达式为：

$$t_i(x_{ij})=(\frac{1}{1000v}(\sum_{k=1}^{j-1}d^i_{k(k+1)})+\sum_{k=1}^{j}t(x_{ik}) \tag{16}$$

其中$k=(1,2,...,j)$，到达路线$X_{i'}$的第j'个点的时间记为$t_{i'}(x_{i'j'})$。当点x_{ij}和点$x_{i'j'}$为同一个点时，则两点的序号须相同：

$$\left|x_{ij}\right|=\left|x_{i'j'}\right| \tag{17}$$

此时，要求1分钟之内到达该点的人数不超过5人，记该点的人数属性$tag(x_{ij})$记为：

$$tag(x_{ij})=\begin{cases}1, |t_i(x_{ij})-t_{i'}(x_{i'j'})|\le 1\\ 0, else\end{cases} \tag{18}$$

则到达该点的人数应满足如下约束：

$$\sum_{i=1}^{N}tag(x)\le 5 \tag{19}$$

完成任务的平均时间应大于两小时:

$$\bar{T}(X_i)=\frac{1}{N}\sum_{i=1}^{N}(\frac{1}{1000v}(\sum_{j=1}^{n_i}d^i_{j(j+1)})+\sum_{j=1}^{n_i}t(x_{ij}))\ge 2 \tag{20}$$

竞赛的总完成时间最长不超过3个小时，转化为完成时间最长的线路须不超过3小时，即：

$$\max_i T(X_i)\le 3 \tag{21}$$

为了使参与的选手人数尽可能多，我们充分利用(21)式的约束，令$T=3\text{h}$，路线X_i的出发时间间隔Δt_i，则路线X_i的参赛选手数量$P(X_i)$最多可计算为：

$$P(X_i)=\frac{1}{\Delta t_i}(T-T(X_i)) \tag{22}$$

比赛希望参赛选手的总数量越多越好，则：

$$\max\sum_{i=1}^{N}P(X_i) \tag{23}$$

此外，所有路线$\{X_i\}$的起点和终点都相同，故起点的序号都为1，终点的序号都相等，有：

$$|x_{i1}|=1\ and\ |x_{1n_1}|=|x_{2n_2}|=...=|x_{in_i}|=...=|x_{Nn_N}| \tag{24}$$

综上，得到如下约束优化模型：

$$
\begin{aligned}
&\max \sum_{i=1}^{N} P(X_i) \\
&\min \ \{\max_i(D(X_i)) - \min_i(D(X_i))\} \\
&\min \ \{\max_i(T(X_i)) - \min_i(T(X_i))\} \\
&\min \sum_{i=2}^{N} \sum_{j=1}^{n_i} E_{1j}(X_i) \\
&\min abs((\frac{1}{1000v}(\sum_{j=1}^{m-1} d^i_{j(j+1)}) + \sum_{j=1}^{m} t(x_{ij})) - N \cdot \frac{1}{3} \cdot \bar{T}(X_i)) \\
&\min abs((\frac{1}{1000v}(\sum_{j=1}^{n-1} d^i_{j(j+1)}) + \sum_{j=1}^{n} t(x_{ij})) - N \cdot \frac{2}{3} \cdot \bar{T}(X_i)) \\
&\min_i t(x_{in_i}) \\
&\min_i(\max_i D(X_i)) \\
s.t.\begin{cases}
D(X_i) = \sum_{j=1}^{n_i-1} d^i_{j(j+1)} \\
d = \sqrt{(x_A - x_B)^2 + (y_A - y_B)^2} \\
t(x_{ij}) \in \{1,2,3,4\} \\
T_i(X_i) = \sum_{j=1}^{n_i} t(x_{ij}) \\
\min_i(n_i) \geq 20 \\
\max_i(n_i) \leq 100 \\
|x_{im}| \in \forall X_i \ \ and \ \ |x_{in}| \in \forall X_i \\
t_i(x_{ij}) = (\frac{1}{1000v}(\sum_{k=1}^{j-1} d^i_{k(k+1)}) + \sum_{k=1}^{j} t(x_{ik}) \\
|x_{ij}| = |x_{i'j'}| \\
\sum_{i=1}^{N} tag(x) \leq 5 \\
\bar{T}(X_i) = \frac{1}{N} \sum_{i=1}^{N} (\frac{1}{1000v}(\sum_{j=1}^{n_i} d^i_{j(j+1)}) + \sum_{j=1}^{n_i} t(x_{ij})) \geq 2 \\
\max_i T(X_i) \leq 3 \\
P(X_i) = \frac{1}{\Delta t_i}(T - T(X_i)) \\
|x_{i1}| = 1 \ and \ \ |x_{1n_1}| = |x_{2n_2}| = ... = |x_{in_i}| = ... = |x_{Nn_N}|
\end{cases}
\end{aligned}
\tag{25}
$$

根据问题和模型的特点，问题求解采用基于参考点的非支配排序方法的进化多目标优化算法NSGA-III算法，NSGA-III是基于遗传算法的多目标优化算法，基于pareto最优解来讨论的多目标的优化过程[1][2]。

求解过程算法逻辑流程及子方法如图4所示。

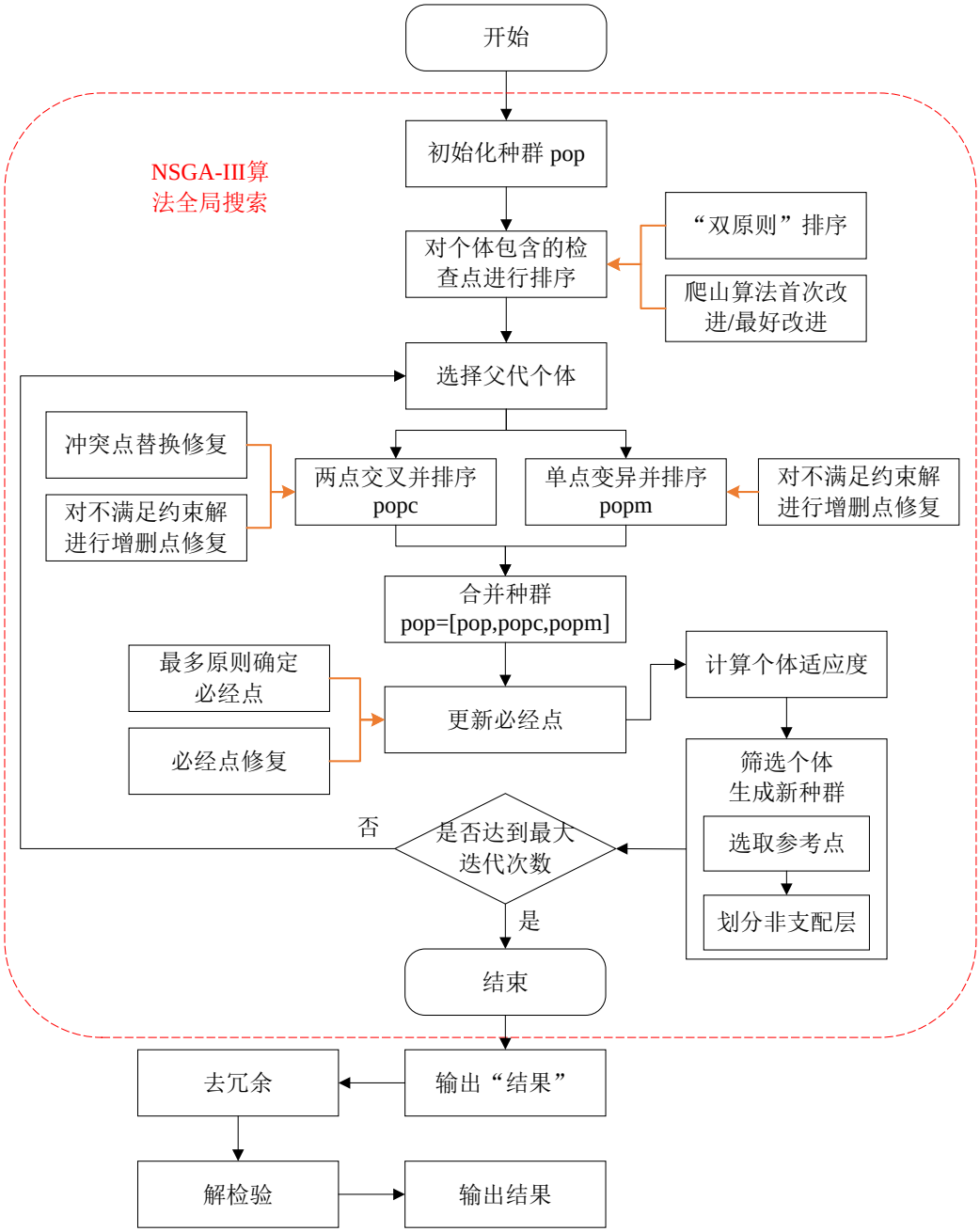

图 4　基于 NSGA-III 算法的定向越野路线规划流程图

算法求解逻辑如下：

Step1：初始化种群并对个体中的检查点进行排序。

排序是指确定个体(路线)中检查点的打卡先后顺序。种群表示一组解，其中每个个体由一维向量表示，即一个个体表示一条路线 X_i。个体中包含的检查点进行排序时考虑“双原则”，即成为当前点的下一个点需遵循(1)该点距离当前点尽可能近；(2)距离终点尽可能远。计算中我们也采用爬山算法来实现，算法策略包含首次改进（首次找到比当前解优的解即更新）和最好改进（遍历找到最好的解再更新当前解），结果发现“双排序”原则更优。

Step2：选择交叉变异以更新种群。

从父代种群中选择个体进行交叉变异。交叉变异过程易使得个体中包含重复点等非可行解，以及不包含必经点等情况。因此，需要进行消解以及修复不满足约束条件的解。对于重复点采用交叉映射的方法去除。

每次迭代过程中，必经点的选取策略如下：根据(13)式计算出现在路线中$\frac{1}{3}$和$\frac{2}{3}$位置（时间）处的检查点，选择频数最高的点作为所有线路的必经点。对于当前路线，若其包含必经点，不论它处在路线的哪个位置，不必调整。若当前路线不包含必经点，修改当前路线的$\frac{1}{3}$和$\frac{2}{3}$位置处的点为必经点。直至所有被选择出的路线都包含了必经点。

将被选择出的路线（个体）与原始种群合并得到当前种群，至此算法完成了一次更新过程。算法的部分参数设置如表2所示：

表 2　NSGA-III 算法部分关键参数

种群大小	迭代次数	交叉概率	变异概率
150	200	0.9	0.1

交叉和变异概率保证了合并之后的当前种群的规模是原始种群规模$N(pop)$的2倍。种群大小和迭代次数能够有效保证算法的多样性和收敛性。

Step3：计算个体的适应度并从2$N(pop)$个个体中筛选出$N(pop)$个个体。

新种群的选择过程建立在2$N(pop)$条路线的基础上，适应度函数为路线的总长度尽可能均匀(6)式、路线的总难度尽可能均匀(7)式、和尽可能避免重复路径(10)式，实际计算时，先计算种群所有个体路线总长度的均值、路线总难度的均值，再考虑单个个体距离均值的远近程度来确定个体的适应度值。利用NSGA-III算法基于参考点的非支配排序选择方法进行种群筛选。

Step4：迭代计算直至满足结束条件后输出初步结果。**算法结束**。

由于算法收敛和约束条件不全的因素，求解结束后发现得到的种群中个体存在大量重复解和不可行解，因此还需要进行后续计算。具体包含去冗余和解的检验及部分未知量的计算等。

首先，去除种群中重复的个体，其次利用式(20)和式(21)进行解筛选和检验。最后利用式(22)考虑“剩余时间”计算参赛总人数，“剩余时间”是指路线的最长时间(包含行进时间和打卡点的难度时间)与路线最短时间的差，令该时间差作为出发时间间隔，即：

$$\Delta t_i = \max_i T(X_i) - \min_i T(X_i) \tag{26}$$

其中$T(X_i)$表示路线X_i的完成时间，公式同时表示所有路线的出发间隔时间相同。

综上，目标函数的处理与算法过程的对应关系如下图所示：

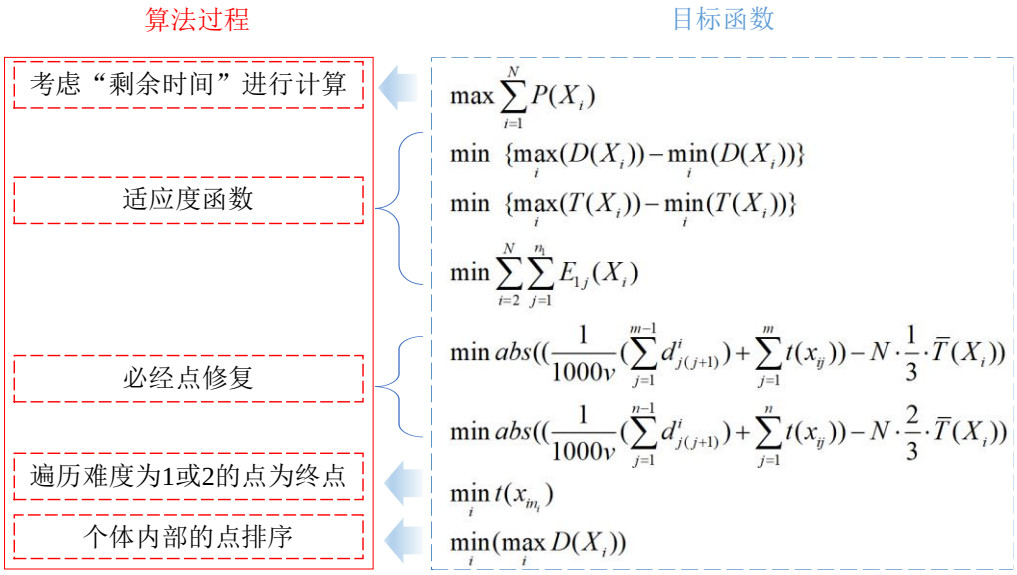

图 5　目标函数处理过程与算法过程在图 4 中的对应关系

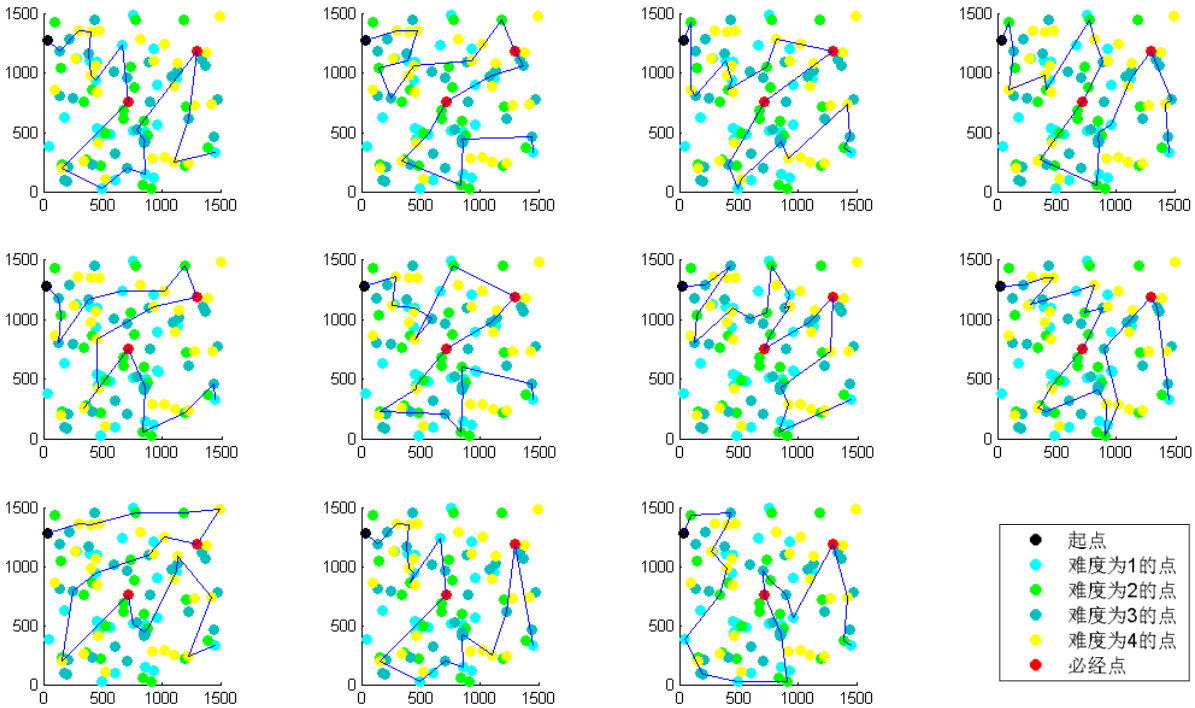

图 6　问题一 11 条路线及其必经打卡点(起点和终点相同)

4.2 问题二模型的建立与求解

问题二在问题一的基础上增加了每个检查点的Z坐标，即需要考虑每一条线路的爬

高量，并将检查点的总数量增加至150个，其余已知条件不变。

则每一条路线的爬高量总和为所有“上坡量”之和，若后一检查点比前一检查点低则累加量记为0，否则记为后一检查点与前一检查点Z坐标之差。故爬高量总和的计算公式为：

$$H(X_i)=\sum_{j=1}^{n_i-1}(\max((Z(x_{i(j+1)})-Z(x_{ij})),0) \tag{27}$$

其中 $Z(x_{ij})$ 表示点 x_{ij} 的Z坐标值。需要使得所有路线的爬高量尽可能均匀，则有：

$$\min(\max_i H(X_i)-\min_i H(X_i)) \tag{28}$$

该目标函数的处理过程与问题一类似：在迭代过程中计算种群中个体爬高量的均值，然后计算每个个体路线的爬高量与均值之差的绝对值作为选择个体的依据之一。

其余求解过程同问题一。

4.3 问题三模型的建立与求解

问题三给定此次活动有参赛选手的总数量为120人，并且每个人的行进速度不相同。参赛队员的总数约束，有：

$$\sum_{i=1}^{N}P(X_i)=120 \tag{29}$$

设第 i 条路线 X_i 上的第 j 个队员的速度为 v_{ij}，要求满足参赛选手平均完成时间不低于2小时的约束，即：

$$\frac{1}{120}(\sum_{i=1}^{N}\sum_{j}^{P(X_i)}\frac{D(X_i)}{v_{ij}}+\sum_{i=1}^{N}\sum_{j=1}^{n_i}t(x_{ij}))\geq 2 \tag{30}$$

比赛总完成时间不超过3.5个小时，即耗时最长的参赛队员完成任务的时间小于3.5h。对于第 i 条路线 X_i 上的第 j 个队员，他完成任务的时间由三部分组成：(1)出发的间隔时间 Δt_i；(2)路线中所有路径上行进时间的总和；(3)所有检查点难度(时间度量)之和。因此，有：

$$\max_i((j-1)\Delta t_i+\frac{D(X_i)}{v_{ij}}+\sum_{j=1}^{n_i}t(x_{ij}))\leq 3.5 \tag{31}$$

同时，为了使完成的时间尽可能短，有目标函数：

$$\min(\max((j-1)\Delta t_i+\frac{D(X_i)}{v_{ij}}+\sum_{j=1}^{n_i}t(x_{ij}))) \tag{32}$$

使用的路线尽可能少：

$$\min N \tag{33}$$

在实际的求解过程中，初步模糊筛选终点时，我们使用120名参赛队员的行进速度平均值 $\bar{v}_{ij}=6\,\text{km/h}$，但在路线时间和难度等参数的实际计算时则使用每名参赛队员的实际速度 v_{ij}。求解步骤如下：

Step1：初步筛选择终点。

在问题二算法的基础上，更新并计算每个检查点的属性信息(检查点的个数，坐标，难度值，检查点之间的距离等参数)。考虑(14)式，遍历每个难度值为1和2的点依次作为终点使用算法计算得到相应终点下的一组解(一组路线)$\{X_i\}$，分别计算每一组解的路线长度的均匀程度、路线难度均匀程度、路线的爬高均匀程度。均匀程度用最大值与最小

值的差值来作为衡量指标。即：

$$D(\{X_i\})=\max_i(D(X_i))-\min_i(D(X_i))$$
$$T(\{X_i\})=\max_i(T(X_i))-\min_i(T(X_i)) \quad (34)$$
$$H(\{X_i\})=\max_i(H(X_i))-\min_i(H(X_i))$$

因此，终点的选择与该三个指标有关，故而我们利用TOPSIS方法来对所有终点进行“方案”排序，选择贴近度值最大的点作为终点。TOPSIS方法的输入为多个方案（候选终点）不同指标下的数据矩阵和指标的权重（此处指标权重都相等），输出为方案按照贴近度的优劣排序。

Step2：不同线路出发时间间隔的确定。

根据第一和第二问的计算结果数据，考虑到每个人行进速度不同，我们令同一条路线上行进速度慢的人先出发，速度较快的人后出发，则出发时间间隔设置为前后两个人完成时间差的最大值：

$$\Delta t_i=\max(\frac{D(X_i)}{v_{ij}}-\frac{D(X_i)}{v_{i(j+1)}}) \quad (35)$$

以此先保证同一个点不会同时到达超过5人的约束。

Step3：根据约束条件和目标函数将120人分配至不同的路线。

考虑(29)-(30)式，沿用问题一和问题二的NSGA-III算法并改进求解。过程如下图所示：

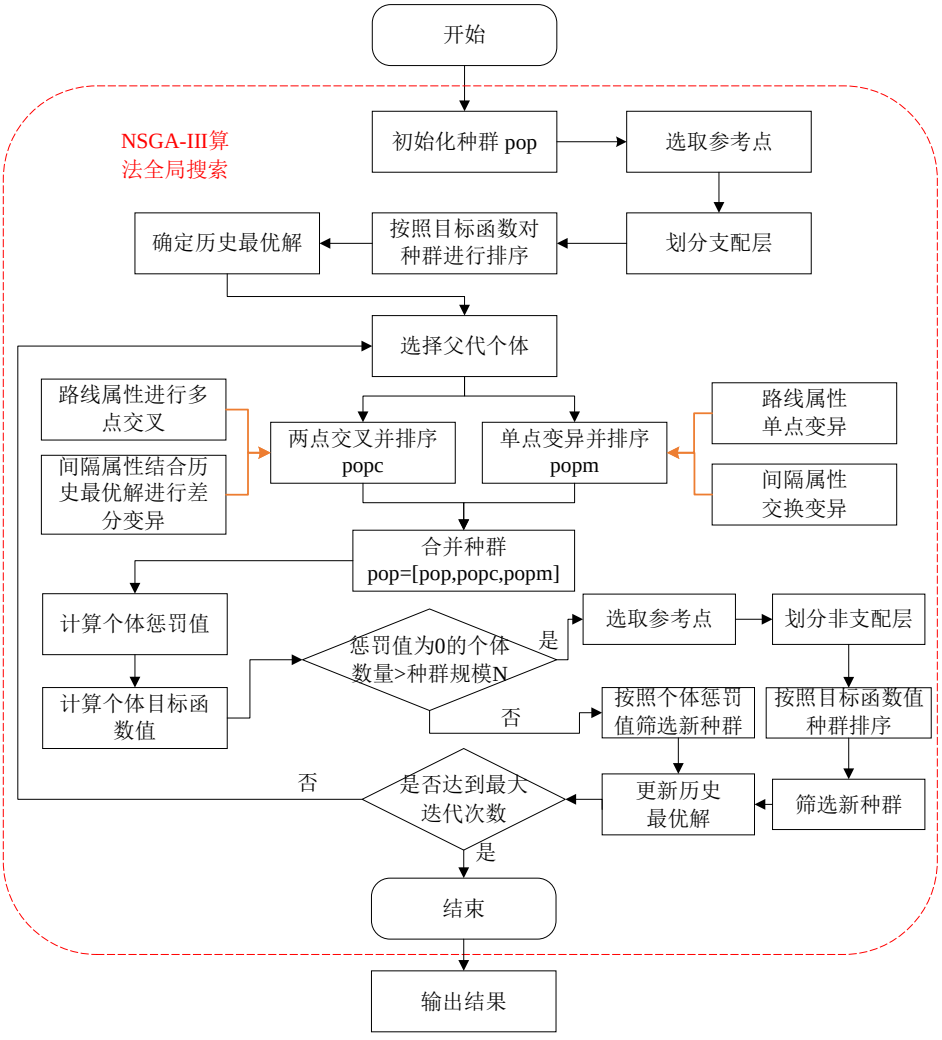

图 7　基于 NSGA-III-DE 算法“为路线分配队员”的规划流程图

算法的主逻辑与问题一相似，结合问题三的要求，主要有以下不同之处：

个体的编码不同。种群中包含两类个体：第一类个体为维度为1×120 的向量，向量中的每个数值表示对应队员被分配到的路线的序号，第二类个体为维度为1× N 的向量，向量中的每个数值表示对应路线的出发时间间隔。即初始的种群中每个个体包含路线和时间间隔两类属性。每一条路线的终点相同，终点的难度考虑式(14)的约束，附件3所给数据除起点外，其余点的难度值均大于1，故初始化种群的计算中我们令：

$$t(x_{in_i}) \in \{2,3,4\} \tag{36}$$

对于前序计算中没考虑的约束(19)式和(31)式作为惩罚函数和本题的目标函数路线最短和时间最短来对迭代过程中的种群进行分类筛选。

重复迭代直至最终输出结果。算法流程的输出并结算得到的结果为每条路线的参赛人数、每条路线每个人的出发先后顺序、各条路线的出发时间间隔。

5 模型评价

5.1 模型优点

(1) 模型既对现实情况做了部分简化，又考虑了定向越野的实际情况，例如必经点A和B的设置未知尽量让参赛举办方能够全局把控队员情况。

(2) 建立了严谨细致的问题求解模型，考虑了各种不同的实际情况，对问题简化较少，求解较为合理。

(3) 求解过程对问题进行了分析的基础上进行合理简化，模型求解计算采用了NSGA-III为主算法，期间融入了TOPSIS，爬山算法等方法考虑约束条件，加快了问题求解的速度。

(4) 计算过程数据详实，可追溯前馈。

5.2 模型缺陷

(1) 编程求解过程中由于运算量庞大，约束条件众多而导致求最优解缓慢，故对求解过程进行合理的简化，导致结果可能有偏差。

(2) 模型考虑了不同路线之间应尽可能避免包含较多重复路段，但问题简化故未考虑应避免参赛选手在完成任务过程中距离过近的问题，考虑了同时到达同一个点的人数不超过5人来代替该约束条件。

(3) 计算求解主要基于优化算法，但也有人为主观的约束条件检验和筛选解，求解效率较低，算法复杂度较高，运行时间长。

参考文献

[1] Deb K , Jain H . An Evolutionary Many-Objective Optimization Algorithm Using Reference-Point-Based Nondominated Sorting Approach, Part I: Solving Problems With Box Constraints[J]. IEEE Transactions on Evolutionary Computation, 2014, 18(4):577-601.

[2] Fellow, IEEE, Jain H , et al. An Evolutionary Many-Objective Optimization Algorithm Using Reference-Point Based Nondominated Sorting Approach, Part II: Handling Constraints and Extending to an Adaptive Approach[J]. IEEE Transactions on Evolutionary Computation, 2014, 18(4):602-622.

附录

附录 1 详细计算过程数据见附件 Excel 表

附录 2 MATLAB 代码过多，此处仅附上主程序代码，全部代码见附件.m 文件。

第一问

```
clc;
clear;
close all;

%% Problem Definition

CostFunction = @(x,distance,end_point,route)
MOP2(x,distance,end_point,route);    % Cost Function
nObj = 3;

%% NSGA-II Parameters
nDivision = 5;
Zr = GenerateReferencePoints(nObj, nDivision);
MaxIt =200;    % Maximum Number of Iterations
nPop = 150;    % Population Size
pCrossover = 0.9;              % Crossover Percentage
nCrossover = 2*round(pCrossover*nPop/2); % Number of Parnets (Offsprings)
pMutation = 0.1;               % Mutation Percentage
nMutation = round(pMutation*nPop);    % Number of Mutants
mu = 0.02;          % Mutation Rat
%¾àÀë
route = xlsread('¸½¼þ1.xlsx');
distance = zeros(100,100);
for i =1:100
      for j = 1:100
            distance(i,j)=norm(route(i,2:3)-route(j,2:3),2);
      end
end

% end_point = 84;
%% ½øÈëÑ-»·
rank1 = route(route(2:25,4)==1,1)+1;
% rank1 = route(route(26:50,4)==1,1)+25;
% rank1 = route(route(51:75,4)==1,1)+50;
```

```
% rank1 = route(route(76:100,4)==1,1)+75;

% rank2 = route(route(1:25,4)==2,1);
% rank2 = route(route(26:50,4)==2,1)+25;
% rank2 = route(route(51:75,4)==2,1)+50;
% rank2 = route(route(76:100,4)==2,1)+75;

end_points = rank1;
% end_points = rank2;
% end_points = [rank1;rank2];
for num_end_point = 1:length(end_points)
    end_point = end_points(num_end_point);

    %% Colect Parameters
    params.nPop = nPop;
    params.Zr = Zr;
    params.nZr = size(Zr,2);
    params.zmin = [];
    params.zmax = [];
    params.smin = [];

    %% Initialization

    disp('Staring NSGA-III ...');

    empty_individual.Position = [];
    empty_individual.Cost = [];
    empty_individual.Rank = [];
    empty_individual.shuxing = [];
    empty_individual.DominationSet = [];
    empty_individual.DominatedCount = [];
    empty_individual.NormalizedCost = [];
    empty_individual.AssociatedRef = [];
    empty_individual.DistanceToAssociatedRef = [];
    %% Éú³É³õÊ¼½â
    pop = repmat(empty_individual, nPop, 1);
    for i = 1:nPop
        pop(i).Position = initial(end_point);
        pop(i).Position = paixu(pop(i).Position,distance,end_point);
        %       pop(i).shuxing =
CostFunction(pop(i).Position,distance,end_point,route(:,4));
    end
    %%
    % Sort Population and Perform Selection
    %[pop, F, params] = SortAndSelectPopulation(pop, params);
```

```
%% NSGA-II Main Loop
% num_f1 = zeros(1,MaxIt);
for it = 1:MaxIt

    % Crossover
    popc = repmat(empty_individual, nCrossover/2, 2);
    for k = 1:nCrossover/2

        i1 = randi([1 nPop]);
        p1 = pop(i1);

        i2 = randi([1 nPop]);
        p2 = pop(i2);

        [popc(k, 1).Position, popc(k, 2).Position] = Crossover(p1.Position,
p2.Position,end_point);
        popc(k, 1).Position= paixu(popc(k, 1).Position,distance,end_point);
        popc(k, 2).Position= paixu(popc(k, 2).Position,distance,end_point);

        % ½âµÄÐ¸´ solution_repair =
repair(solution,distance,end_point,route)
        popc(k, 1).Position= repair(popc(k,
1).Position,distance,end_point,route(:,4));
        popc(k, 2).Position= repair(popc(k,
2).Position,distance,end_point,route(:,4));

        popc(k, 1).Position= paixu(popc(k, 1).Position,distance,end_point);
        popc(k, 2).Position= paixu(popc(k, 2).Position,distance,end_point);

    end
    popc = popc(:);


    % Mutation
    popm = repmat(empty_individual, nMutation, 1);
    for k = 1:nMutation

        i = randi([1 nPop]);
        p = pop(i);

        popm(k).Position = Mutate(p.Position,end_point);
        popm(k).Position = paixu(popm(k).Position,distance,end_point);
        % ½âµÄÐ¸´ solution_repair =
repair(solution,distance,end_point,route)
        popm(k).Position=
repair(popm(k).Position,distance,end_point,route(:,4));
        popm(k).Position = paixu(popm(k).Position,distance,end_point);
```

```
            %               popm(k).shuxing =
CostFunction(popm(k).Position,distance,end_point,route(:,4));

        end

        % Merge
        pop = [pop
            popc
            popm]; %#ok
        [pop,front_value, back_value] = updatebjd(pop,distance,end_point);

        for m =1: numel(pop)
            pop(m).Position = paixu(pop(m).Position,distance,end_point);
            pop(m).shuxing =
CostFunction(pop(m).Position,distance,end_point,route(:,4));
        end
        shuxings = [pop.shuxing];
        mean_shuxing = mean(transpose(shuxings));
        for m =1: numel(pop)
            pop(m).Cost = abs(pop(m).shuxing - transpose(mean_shuxing));
        end
        edgs = [pop(:).Position];
        for m1 = 1:numel(pop)
            num_chongfu = 0;
            for m2 = 1:length(pop(m1).Position)-1
                chongfu_place = strfind(edgs,
[pop(m1).Position(m2),pop(m1).Position(m2+1)]);
                num_chongfu =    num_chongfu + length(chongfu_place);
            end
            pop(m1).Cost = [pop(m1).Cost;num_chongfu];
        end
        % Sort Population and Perform Selection
        [pop, F, params] = SortAndSelectPopulation(pop, params);

        % Store F1
        F1 = pop(F{1});
        F12 = F1;
        %       if numel(F1) < nPop
        %           F2 = pop(F{2});
        %           F12 = [F12;F2];
        %       end
        % Show Iteration Information
        %       disp(['Iteration ' num2str(it) ': Number of F1 Members = '
num2str(numel(F1))]);
        disp(['Iteration ' num2str(it) ': Number of F12 Members = '
num2str(numel(F12))]);
        %       num_f1(it) = numel(F1);
```

```
            % Plot F1 Costs
%               figure(1);
%               PlotCosts(F12);
%               pause(0.01);

        end
        % plot(1:MaxIt,num_f1);
        %% Results

        % disp(['Final Iteration: Number of F1 Members = ' num2str(numel(F1))]);
        disp(['Final Iteration: Number of F12 Members = '    num2str(numel(F12))]);
        disp('Optimization Terminated.');

        %%   ´æ´¢
        cost = [F12.Cost];
        cost_single = singlecost(cost);
        index = zeros(size(cost_single,2),1);
        for j =1:size(cost_single,2)
            for i = 1:length(F12)
                if cost(1,i)==cost_single(1,j) && cost(2,i)==cost_single(2,j)
                    index(j)=i;
                    break
                end
            end
        end
        %F1 = F1(index);
        bjd_points = [front_value,back_value];
        xlswrite('output1_1_2_25.xlsx',bjd_points,['bjd_points',num2str(end_point)]);
        for i = 1:length(index)
            position = [1,F12(index(i)).Position,end_point];

xlswrite('output1_1_2_25.xlsx',position,['endpoint',num2str(end_point)],['A',num2str(i
)]);
        end
%
%         clearvars -except num_end_point
end
```

第二问

```
clc;
clear;
close all;

%% Problem Definition
```

```
CostFunction = @(x,distance,end_point,route)
MOP2(x,distance,end_point,route);    % Cost Function
nObj = 3;

%% NSGA-II Parameters
nDivision = 5;
Zr = GenerateReferencePoints(nObj, nDivision);
MaxIt =200;    % Maximum Number of Iterations
nPop = 150;    % Population Size
pCrossover = 0.9;             % Crossover Percentage
nCrossover = 2*round(pCrossover*nPop/2); % Number of Parnets (Offsprings)
pMutation = 0.1;              % Mutation Percentage
nMutation = round(pMutation*nPop);    % Number of Mutants
mu = 0.02;          % Mutation Rat
%¾àÀë
route = xlsread(',½¼þ1.xlsx');
distance = zeros(100,100);
for i =1:100
      for j = 1:100
            distance(i,j)=norm(route(i,2:3)-route(j,2:3),2);
      end
end

% end_point = 84;
%% ½øÈëÑ-»·
rank1 = route(route(2:25,4)==1,1)+1;
% rank1 = route(route(26:50,4)==1,1)+25;
% rank1 = route(route(51:75,4)==1,1)+50;
% rank1 = route(route(76:100,4)==1,1)+75;

% rank2 = route(route(1:25,4)==2,1);
% rank2 = route(route(26:50,4)==2,1)+25;
% rank2 = route(route(51:75,4)==2,1)+50;
% rank2 = route(route(76:100,4)==2,1)+75;

end_points = rank1;
% end_points = rank2;
% end_points = [rank1;rank2];
for num_end_point = 1:length(end_points)
      end_point = end_points(num_end_point);

      %% Colect Parameters
      params.nPop = nPop;
      params.Zr = Zr;
      params.nZr = size(Zr,2);
      params.zmin = [];
```

```
params.zmax = [];
params.smin = [];

%% Initialization

disp('Staring NSGA-III ...');

empty_individual.Position = [];
empty_individual.Cost = [];
empty_individual.Rank = [];
empty_individual.shuxing = [];
empty_individual.DominationSet = [];
empty_individual.DominatedCount = [];
empty_individual.NormalizedCost = [];
empty_individual.AssociatedRef = [];
empty_individual.DistanceToAssociatedRef = [];
%% Éú³É³õÊ¼½â
pop = repmat(empty_individual, nPop, 1);
for i = 1:nPop
    pop(i).Position = initial(end_point);
    pop(i).Position = paixu(pop(i).Position,distance,end_point);
    %       pop(i).shuxing =
CostFunction(pop(i).Position,distance,end_point,route(:,4));
end
%%
% Sort Population and Perform Selection
%[pop, F, params] = SortAndSelectPopulation(pop, params);


%% NSGA-II Main Loop
% num_f1 = zeros(1,MaxIt);
for it = 1:MaxIt

    % Crossover
    popc = repmat(empty_individual, nCrossover/2, 2);
    for k = 1:nCrossover/2

        i1 = randi([1 nPop]);
        p1 = pop(i1);

        i2 = randi([1 nPop]);
        p2 = pop(i2);

        [popc(k, 1).Position, popc(k, 2).Position] = Crossover(p1.Position,
p2.Position,end_point);
        popc(k, 1).Position= paixu(popc(k, 1).Position,distance,end_point);
        popc(k, 2).Position= paixu(popc(k, 2).Position,distance,end_point);
```

```
                % ½âµÄÐÞ¸´ solution_repair =
repair(solution,distance,end_point,route)
                popc(k, 1).Position= repair(popc(k,
1).Position,distance,end_point,route(:,4));
                popc(k, 2).Position= repair(popc(k,
2).Position,distance,end_point,route(:,4));

                popc(k, 1).Position= paixu(popc(k, 1).Position,distance,end_point);
                popc(k, 2).Position= paixu(popc(k, 2).Position,distance,end_point);

            end
            popc = popc(:);


            % Mutation
            popm = repmat(empty_individual, nMutation, 1);
            for k = 1:nMutation

                i = randi([1 nPop]);
                p = pop(i);

                popm(k).Position = Mutate(p.Position,end_point);
                popm(k).Position = paixu(popm(k).Position,distance,end_point);
                % ½âµÄÐÞ¸´ solution_repair =
repair(solution,distance,end_point,route)
                popm(k).Position=
repair(popm(k).Position,distance,end_point,route(:,4));
                popm(k).Position = paixu(popm(k).Position,distance,end_point);
                %               popm(k).shuxing =
CostFunction(popm(k).Position,distance,end_point,route(:,4));

            end

            % Merge
            pop = [pop
                popc
                popm]; %#ok
            [pop,front_value, back_value] = updatebjd(pop,distance,end_point);

            for m =1: numel(pop)
                pop(m).Position = paixu(pop(m).Position,distance,end_point);
                pop(m).shuxing =
CostFunction(pop(m).Position,distance,end_point,route(:,4));
            end
            shuxings = [pop.shuxing];
            mean_shuxing = mean(transpose(shuxings));
```

```
            for m =1: numel(pop)
                pop(m).Cost = abs(pop(m).shuxing - transpose(mean_shuxing));
            end
            edgs = [pop(:).Position];
            for m1 = 1:numel(pop)
                num_chongfu = 0;
                for m2 = 1:length(pop(m1).Position)-1
                    chongfu_place = strfind(edgs,
[pop(m1).Position(m2),pop(m1).Position(m2+1)]);
                    num_chongfu =    num_chongfu + length(chongfu_place);
                end
                pop(m1).Cost = [pop(m1).Cost;num_chongfu];
            end
            % Sort Population and Perform Selection
            [pop, F, params] = SortAndSelectPopulation(pop, params);

            % Store F1
            F1 = pop(F{1});
            F12 = F1;
            %         if numel(F1) < nPop
            %             F2 = pop(F{2});
            %             F12 = [F12;F2];
            %         end
            % Show Iteration Information
            %         disp(['Iteration ' num2str(it) ': Number of F1 Members = '
num2str(numel(F1))]);
            disp(['Iteration ' num2str(it) ': Number of F12 Members = '
num2str(numel(F12))]);
            %         num_f1(it) = numel(F1);
            % Plot F1 Costs
%             figure(1);
%             PlotCosts(F12);
%             pause(0.01);

        end
        % plot(1:MaxIt,num_f1);
        %% Results

        % disp(['Final Iteration: Number of F1 Members = ' num2str(numel(F1))]);
        disp(['Final Iteration: Number of F12 Members = '    num2str(numel(F12))]);
        disp('Optimization Terminated.');

        %%   ´æ´¢
        cost = [F12.Cost];
        cost_single = singlecost(cost);
        index = zeros(size(cost_single,2),1);
        for j =1:size(cost_single,2)
```

```
            for i = 1:length(F12)
                if cost(1,i)==cost_single(1,j) && cost(2,i)==cost_single(2,j)
                    index(j)=i;
                    break
                end
            end
        end
        %F1 = F1(index);
        bjd_points = [front_value,back_value];
        xlswrite('output1_1_2_25.xlsx',bjd_points,['bjd_points',num2str(end_point)]);
        for i = 1:length(index)
            position = [1,F12(index(i)).Position,end_point];

xlswrite('output1_1_2_25.xlsx',position,['endpoint',num2str(end_point)],['A',num2str(i
)]);
        end
%
%         clearvars -except num_end_point
end
```

第三问

```
clc;
clear;
close all;

%% Problem Definition

%                 CostFunction                =                @(x,distance,end_point,pagao,route)
MOP2(x,distance,end_point,pagao,route);    % Cost Function
nObj = 2;

%% NSGA-II Parameters
nDivision = 5;
Zr = GenerateReferencePoints(nObj, nDivision);
MaxIt =100;    % Maximum Number of Iterations
nPop = 150;    % Population Size
pCrossover = 0.9;             % Crossover Percentage
nCrossover = 2*round(pCrossover*nPop/2); % Number of Parnets (Offsprings)
pMutation = 0.1;              % Mutation Percentage
nMutation = round(pMutation*nPop);    % Number of Mutants
mu = 0.02;          % Mutation Rat
%距离
% best_route = xlsread('best_ending.xlsx');
best_route = xlsread('output3.xlsx','endpoint23');
num_route = length(best_route(:,1));
route = xlsread('附件 3.xlsx','候选检查点');
```

```
data_people = xlsread('附件 3.xlsx','参赛队平均速度');
people_v = data_people(:,2);
distance = zeros(200,200);
for i =1:length(distance)
    for j = 1:length(distance)
        distance(i,j)=norm(route(i,2:end-1)-route(j,2:end-1),2);
    end
end
end_point = 2;

%% Colect Parameters
params.nPop = nPop;
params.Zr = Zr;
params.nZr = size(Zr,2);
params.zmin = [];
params.zmax = [];
params.smin = [];

%% Initialization

disp('Staring NSGA-III ...');

empty_individual.Position = [];
empty_individual.Cost = [];
empty_individual.Punish = [];
empty_individual.interval = [];
empty_individual.Rank = [];
empty_individual.DominationSet = [];
empty_individual.DominatedCount = [];
empty_individual.NormalizedCost = [];
empty_individual.AssociatedRef = [];
empty_individual.DistanceToAssociatedRef = [];
%%  生成初始解
pop = repmat(empty_individual, nPop, 1);
for i = 1:nPop
    pop(i).Position = initial2(num_route);
    pop(i).interval = rand(1,num_route) * 10; %rzeros(1,num_route)
    pop(i).Cost                                                                                    =
CostFunction2(pop(i).Position,pop(i).interval,best_route,people_v,distance,route(:,end));
end
%%
% Sort Population and Perform Selection
[pop, F, params] = SortAndSelectPopulation(pop, params);
%  寻找最优解
ranks = [pop.Rank];
first_place = find(ranks == 1);
len_f = length(first_place);
r_place = randperm(len_f,1);
best_solution = pop(first_place(r_place)).Position;
best_solution_punish = pop(first_place(r_place)).Punish;
```

```
best_solution_interval = pop(first_place(r_place)).interval;

%% NSGA-II Main Loop
for it = 1:MaxIt

    % Crossover
    popc = repmat(empty_individual, nCrossover/2, 2);
    for k = 1:nCrossover/2

        i1 = randi([1 nPop]);
        p1 = pop(i1);

        i2 = randi([1 nPop]);
        p2 = pop(i2);

        [popc(k, 1).Position, popc(k, 2).Position] = Crossover2(p1.Position, p2.Position);

        %  间隔
        i3 = randi([1 nPop]);
        p3 = pop(i3);
        popc(k, 1).interval = p1.interval + rand() * (best_solution_interval - p3.interval);
        popc(k, 1).interval(popc(k, 1).interval < 0) = 0;
        popc(k, 1).interval(popc(k, 1).interval > 10) = 10;
        i4 = randi([1 nPop]);
        p4 = pop(i4);
        popc(k, 2).interval = p2.interval + rand() * (best_solution_interval - p4.interval);
        popc(k, 2).interval(popc(k, 2).interval < 0) = 0;
        popc(k, 2).interval(popc(k, 1).interval > 10) = 10;
        popc(k, 1).Cost= CostFunction2(popc(k, 1).Position,popc(k,
1).interval,best_route,people_v,distance,route(:,end));
        popc(k, 2).Cost= CostFunction2(popc(k, 2).Position,popc(k,
2).interval,best_route,people_v,distance,route(:,end));
    end
    popc = popc(:);


    % Mutation
    popm = repmat(empty_individual, nMutation, 1);
    for k = 1:nMutation

        i = randi([1 nPop]);
        p = pop(i);

        popm(k).Position = Mutate2(p.Position,num_route);
        popm(k).interval = p.interval;
        r = randperm(num_route,2);
        tmp_interval = popm(k).interval(r(1));
        popm(k).interval(r(1)) = popm(k).interval(r(2));
        popm(k).interval(r(2)) = tmp_interval;
        popm(k).Cost =
```

```
CostFunction2(popm(k).Position,popm(k).interval,best_route,people_v,distance,route(:,end));

    end

    % Merge
    pop = [pop
        popc
        popm]; %#ok

    % 计算惩罚
%       pop_Punish = getpunish();
    for k = 1:length(pop)
        pop(k).Punish = max(0,pop(k).Cost(2) - 210);
    end
    pop_Punish = [pop.Punish];
    newpop = pop(pop_Punish == min(pop_Punish));
    len_pop = numel(newpop);
    if len_pop < nPop || min(pop_Punish) > 0
%         [~,rank] = sort(pop_Punish);
%         newpop = pop(rank(1:nPop));
        newpop2 = repmat(empty_individual, nPop, 1);
        for k = 1:nPop
            r3 = randperm(nPop,3);
            [~,ran] = sort(pop_Punish(r3));
            newpop2(k) = pop(r3(ran(1)));
        end
        newpop2_Punish = [newpop2.Punish];
        [~,place_min_punish] = sort(newpop2_Punish);

        if newpop2(place_min_punish(1)).Punish < best_solution_punish
            best_solution = newpop2(place_min_punish(1)).Position;
            best_solution_punish = newpop2(place_min_punish(1)).Punish;
            best_solution_interval = newpop2(place_min_punish(1)).interval;
        end
        pop = newpop2(:);
    else
        pop = newpop;
        % Sort Population and Perform Selection
        [pop, F, params] = SortAndSelectPopulation(pop, params);

        % Store F1
        F1 = pop(F{1});
        F12 = F1;
        %     if numel(F1) < nPop
        %         F2 = pop(F{2});
        %         F12 = [F12;F2];
        %     end
        % Show Iteration Information
        %         disp(['Iteration ' num2str(it) ': Number of F1 Members = '
num2str(numel(F1))]);
```

```
            disp(['Iteration ' num2str(it) ': Number of F12 Members = ' num2str(numel(F12))]);
            %         num_f1(it) = numel(F1);
            % Plot F1 Costs
            figure(1);
            PlotCosts(F12);
            pause(0.01);
        end
        disp(['Final Punish: = '    num2str(min([pop.Punish]))]);

end
% plot(1:MaxIt,num_f1);
%% Results

% disp(['Final Iteration: Number of F1 Members = ' num2str(numel(F1))]);
% disp(['Final Iteration: Number of F12 Members = '    num2str(numel(F12))]);
disp('Optimization Terminated.');


%%   存储
cost = [F12.Cost];
cost_single = singlecost(cost);

index = zeros(size(cost_single,2),1);
for j =1:size(cost_single,2)
        for i = 1:length(F12)
            if cost(1,i)==cost_single(1,j) && cost(2,i)==cost_single(2,j)
                index(j)=i;
                break
            end
        end
end
F_final = F12(index);
% bjd_points = [front_value,back_value];
% xlswrite('output3_1_1_25.xlsx',bjd_points,['bjd_points',num2str(end_point)]);
%
% for i = 1:length(index)
%         position = [1,F12(index(i)),end_point];
%         xlswrite('333.xlsx',position,['endpoint',num2str(end_point)],['A',num2str(i)]);
%          %CostFunction2(F12(index(i)).position,F12(index(i)).interval,best_route,people_v,di
stance,route(:,end))
% end
% w = length(cost(1,:));
% w = ones(1,length(cost(1,:))) / w;
% topsis(cost,w)
% last = F12(8).position;
%
% CostFunction2(F12(8).position,F12(8).interval,best_route,people_v,distance,route(:,end))
```