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）：

我们的参赛报名号为（如果组委会设置报名号的话）：

所属学校（请填写完整的全名）：

参赛队员（打印并签名）：1. 吴天昊

2. 赵铁华

3. 赵晓晖

指导教师或指导教师组负责人 （打印并签名）：

日期：　　年　　月　　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。）

最优化定向越野路线设计

摘　　要

定向越野路线设计对比赛结果有着根本性的影响，为了秉持公平性的原则，不同的路线之间应该尽量保证难度相近。路线的难易程度往往受到路线的总长度、行进路线地形（即爬高量）、打卡点难度等等影响。为了防止选手们相互交流并保证选手们的比赛体验，路线的设计还要满足其它约束。如何在有限的比赛时间和有限的比赛场地内制定多条符合条件的比赛路线，是比赛组织方需要考虑的重要问题。本组就这个问题建立了数学模型并求解，最终进行仿真验证了设计方案的合理性，为组织方提供行之有效的路线设计方案。

针对问题一和二，引入路线总长相对极差、总打卡难度相对极差、总爬高量相对极差、路线空间隔离度、路段相对重复率等概念，将题目中的宽松性约束条件转化为含有可调参量的严格约束条件，建立有约束的单目标寻优模型；对初始解空间不断增加约束条件，逐步缩小解空间范围，得到可求路线候选解空间，结合问题具体而言，在解决含有必经点的路线规g 线设计方案；进行模型的敏感性分析，根据解空间状态，利用单一变量法改变阈值参数，得到多组不同的路线设计方案；建立评价函数，选取最优路线设计方案：问题一方案 16 条线路，总容纳人数为 215 人，各路线用时在 2.1 小时波动，；问题二方案 17 条线路，总容纳人数为 268，各线路用时在 2.1 小时波动。最后进行仿真实验，检验了严格约束条件的符合程度，验证了本路线设计方案的合理性；

针对问题三，在前两问模型的基础上，更改目标函数为路线数最少与完成任务时间最短，建立有优先级的双目标寻优的模型。模型的求解方法本质上与前两问相同。但是考虑到所有参赛选手的行进速度各不相同，不再能够通过仿真实验遍历所有解空间，去验证所选路线是否满足所有的严格约束条件，因此我们转而使用蒙特卡洛仿真的方法去检验。对于每一种路线规划方案进行 1000 次蒙特卡洛仿真，观察仿真的 5 人同时到达次数的概率大小和比赛平均总用时，若两指标若说明该路线数量的设计方案的鲁棒性好，受到参赛选手速度变化带来的波动小。最终得到我们选择 4 路线的设计方案，比赛平均用时为 2.951 小时，同时满足尽可能少的路线和尽可能少的时间的题设条件。

关键词： 多目标寻优 单目标寻优 含必经点路线规划

1 问题重述与分析

1.1 问题重述

定向越野是一项要求各参赛选手依据规定好的路线依次打卡的户外运动。通常，受到比赛时间和比赛场地的限制，无法保证所有参赛选手的比赛路线相同，否则总比赛时间过长；也无法保证不同打卡路线的所有打卡点之间的间隔很远，或是所有参赛选手在比赛的过程中都不会与与别的参赛选手靠近，否则要求比赛场地的空间足够大。因此主办方需要考虑如何在有限的比赛时间和空间内制定多条比赛路线，且这些比赛路线要满足以下共同约束条件：

（1）为了保证比赛的公平性，不同的比赛路线的总路程、打卡总难度、爬高总量要尽可能相同；

（2）为了避免“跟跑”现象的发生，不同路线之间的重复路段要尽可能少；

由于同一条路线不止一位参赛选手，如果不同路线有共同路段，后面的选手可能跟着前面的选手跑。可以适当强化这个约束条件，尽可能地减少重复路段，从根本上杜绝跟跑的可能。

（3）为了保证不同路线在空间上的隔离度，避免选手交流，尽可能减少不同路线间的共用打卡点，且选手的距离不能过近；

计算选手相遇的可能性。在实际比赛中，每一位选手的行进速度不同，无法仿真模拟出所有的情况，通过路线中共同检查点数量和交叉点数量表征总的相遇的概率。

（4）为了保证时间的隔离度，避免同时到达同一打卡点的人数超过 5 人；

（5）每条路线都要经过两个必经的检查点。

（6）在选取哪些检查点确定后，使得相临近的点尽可能连续打卡，使总的路线最短。

1.2 问题分析

针对问题一：

在不考虑各打卡点高程的前提下，在二维平面内设计比赛路线，使得尽可能多的参赛的选手可以加入。新增选手完成任务的平均时间不少于 2 小时，竞赛总时长不超过 3 小时这两个约束条件。

分析：目标函数是参赛选手人数尽可能多。竞赛总时长可以理解为：从第一批比赛选手出发至最后一名比赛选手到达终点的时间。“不超过 3 小时”规定了竞赛总时长的上限，对于既定的路线设计方案，可容纳的人数上限即取决于同一条路线上的参赛人数，亦即竞赛总时长；

要求完成任务的平均时间不少于 2 小时，这与“检测点的数量不少于 20 个”这个约束条件一同规定了比赛难度的下界，对打卡点数、路线总路程的下限提出了要求。

针对问题二：

在考虑各打卡点高程的前提下重新解决问题一。

分析：打卡点高程的引入会给问题一的模型带来两点变化

①新增了各条路线的总爬高量近似相等这一约束条件；

②各打卡点之间的距离不再是平面中的直线距离，而是三维空间中的距离，因而每两个检查点之间路程的距离计算发生了变化，各路线的总距离会发生变化。

本质上问题一是问题二在各检查点的高程为零时退化得到的，因此在建立模型并求解的过程中，不妨先考虑包含高各点程信息的完整模型，再分别对问题一和问题二求解。

针对问题三：

在总参赛人数 120 人的前提下，设计最少的比赛路线，且完成时间尽可能短。与问题一类似，完成任务的平均时间不少于 2 小时，竞赛总时长不超过 3.5 小时。 另外提供了所有参赛选手各自的预估平均速度。

分析：与前两问相比，问题三改变的主要有两个方面：

①目标函数与约束条件发生改变。问题三中期望的是比赛路线少且路线用时短，而总人数此时变成了约束条件；

②两个目标函数相互制约。在总人数一定的情况下，路线越少，分配到每条路线的人数即越多，如果极限的出发间隔固定，那么总的完成时间就会增加，与完成时间尽可能短相悖；

③各参赛选手的速度预估速度不再相同。在前两个问题中默认所有参赛选手平均速度为 6 km/h，那么对于确定的路线设计方案，任意调换参赛选手的顺序都不会改变比赛结果；当每名参赛选手速度不同时，即使在同一条参赛路线上的选手在比赛中也有可能相遇，相当于强化了约束条件；

2 模型假设：

(1)假设所有参赛队的行进速度不会因为地形的起伏而变化。因为每条路线的总爬高量会在模型中单独约束，使其近似相等，所以爬高给所有参赛队带来的总体能消耗是近似相同的，故统一不考虑奔跑中上坡带来的额外体能消耗。

(2)假设两个参赛选手交流是在瞬时发生的，两个参赛选手只要相遇，就可以交流。

(3)假设每个参赛选手在检查点处的实际停留时间等于该检查点的打卡难度数值，且不存在放弃打卡行为。

3 符号说明

表 1　符号说明

符号	符号含义
N	附件给出的打卡点集合
M_r	第 r 条路线所包含所有点的点集
$\tilde{\mathrm{M}}_r$	依照打卡顺序映射生成新的点集合
k_{r}、k_{L}、k_H、k_{D}	引入的相对极差,用于衡量数据均匀程度
P_r	代表第 r 条路线上的第 P_r 个出发的选手
$h_{m_j m_{j+1}}$	相邻两个打卡点间高程差
L_r	第 r 条路线的总长度
C_r	第 r 条路线的所有打卡点打卡难度总和

4 模型建立

4.1 模型整体思路

4.1.1 约束条件分类

对于题中所有的约束条件，按照约束条件的容忍度可以将其分为两类：严格约束条件和宽松约束条件；按照约束条件作用对象也可以分成两类：针对单条路线的约束条件和针对多条路线的约束条件。具体阐述如下：

（1）严格约束条件和宽松约束条件

1.严格约束条件

设计所得到的比赛路线必须满足的条件。例如“所有比赛路线的起点和终点必须相同”，不满足这个条件的比赛路线的设计方案直接抛弃。

2.宽松约束条件

在设计比赛路线的过程中要尽可能满足这些宽松性的约束条件，但这些条件在描述时是一个概述、含糊的情况，可以通过人为限定阈值，使得宽松性约束条件转变为可调参量的严格约束条件。比如：“不同路线总打卡难度应尽可能均匀”，可以规定所有路线总打卡难度的极差小于 5 分钟，所有不满足条件的路线设计方案都被抛弃。这里的“5 分钟”即人为限定的阈值，可根据解空间的收缩程度进行调整。

(2) 针对单条/多条路线的约束条件

1.针对单条路线的约束条件

对于某种路线设计方案，只考虑方案中每一条路线是否满足约束条件（宽松性约束条件可以通过设定阈值转化成严格约束条件），不满足即抛弃该方案；例如“每条路线的检查点数量要大于 20”。

2.针对多条路线的约束条件

某些约束条件需要同时考察多条路线的相互关系；例如“不同的比赛路线总长度要近似相同”，要横向比较多条的路线长度的差异，设定可以允许的差异的阈值，超出阈值的方案即抛弃。

4.1.2 多目标问题转化为单目标问题

仔细思考题目中约束条件与与目标函数之间的关系可以发现，宽松性约束条件本质上也是目标函数，例如“路线长度尽可能相同”，就可以认为路线长度之间的差异越小这一组设计方案越优；但是题中这样的宽松性约束条件有很多，倘若都将其看成目标函数，那么设计比赛路线则是一个有约束的多目标优化模型，除了一个主目标（参赛人数最多或是比赛线路最小）以外，还有若干的次目标。

解决这一类多目标优化的问题常用方法是对多个目标函数进行分析，定权重，将其转化为单目标函数。在定权重的过程中，各目标函数的量纲有较大差异，相互之间相对的重要程度的衡量具有较大的主观性，最后很有可能影响到最优方案的设计；另外考虑到本题巨大的可行解空间，约束条件越少，在全部解空间中寻优的时间代价也就越高，不利于降低计算难度。因此本组在建立模型的过程中，采用将宽松性约束条件转化为可调参量的严格约束条件这种方式，将带约束条件的多目标优化问题转化为带约束条件的单目标优化问题。

4.1.3 模型建立与求解流程

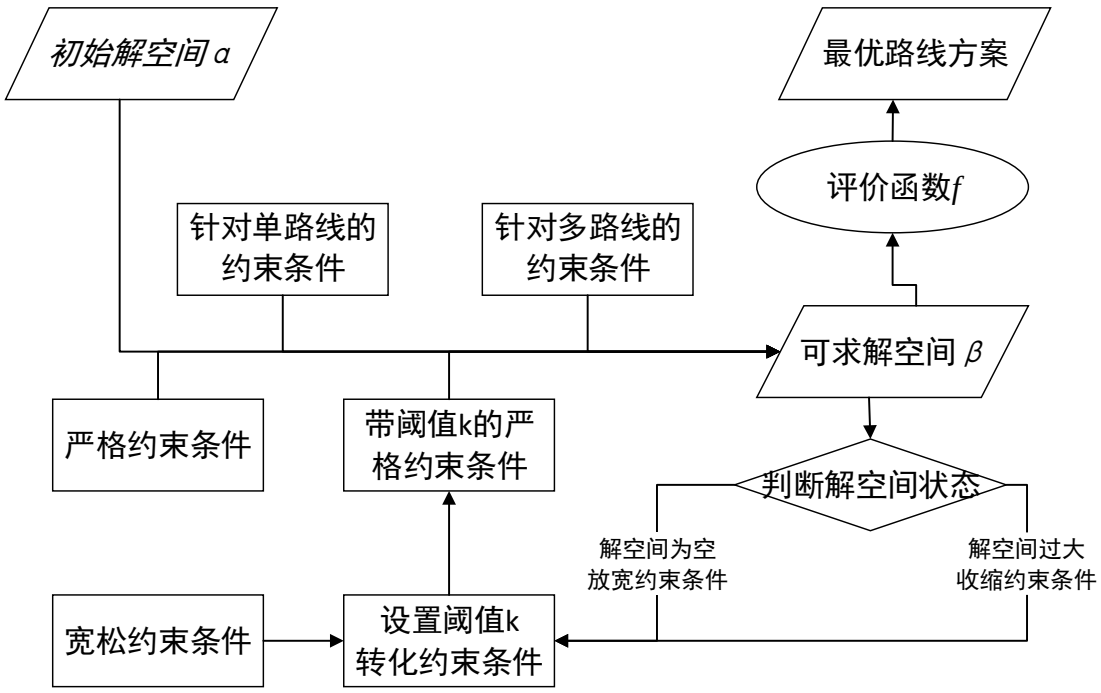


图 1　模型建立与求解流程

步骤一：将所有可能存在的路线设计方案纳入总的解空间，记为解空间 α，由于计算量巨大无法直接对其进行求解；

步骤二：对所有的宽松性约束条件人为限定阈值，使其转变为可调参量的严格约束条件；将多目标优化问题转化为单目标优化问题；

步骤三：通过众多约束条件不断地对解空间 α 进行约束，得到可求解空间 β；

步骤四：遍历求解出解空间 β 内所有路线规划方案；

步骤五：做敏感度分析，判断当前解空间状体，使用单一变量法改变前面每一个约束条件的阈值，然后重复步骤四；

步骤六：建立评价函数 f，对所有路线规划方案进行评价，分析这些方案的优劣变化得到最终的路线规划方案。

4.2 模型基本参数定义

4.2.1 路线中的检查点集

根据附件中提供数据，将所有 N_0 个候选检查点表示为点的集合 N，

$$\mathrm{N}=\{n_i\} \qquad i=1,2,...,N_0$$

定义所有路线的起点为 S，终点为 E，必经打卡点 M_{check_1}、M_{check_2}，其中 E、M_{check_1}、M_{check_2} 是所有路线的必经检查点，在集合 N 中分别记为 n_{end}、n_{check1}、n_{check2}。

定向越野比赛的路线一般由一个起点，若干个检查点和终点构成，定向越野比赛路线的基本形式如下图所示：

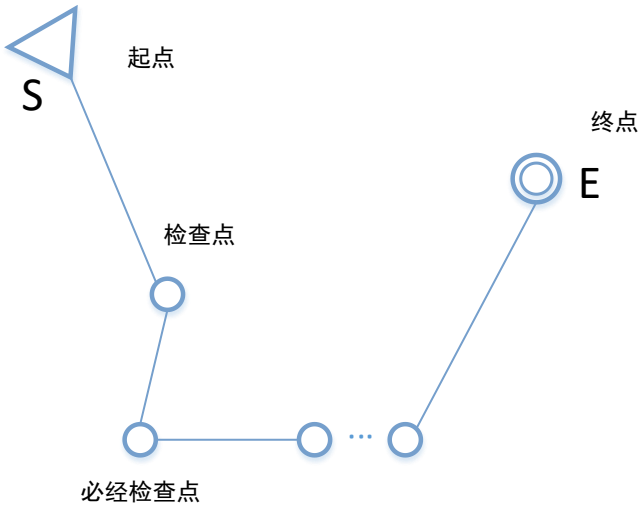


图 2　比赛路线示意图

M_r 代表第 r 条定向越野路线中包含的所有检查点构成的点集，所有检查点均从点集 N 中选取， M_r 中包含元素个数不少于 20，且一定包含 n_{end} 、 n_{check1} 、 n_{check2} 这三个特殊检查点，见公式

$$\mathrm{M}_r = \{n_i\} \qquad i \in [1, N_0] \cap R \tag{1}$$

$$\mathrm{M}_r \subseteq \mathrm{N} \tag{2}$$

$$\mathrm{crad}(\mathrm{M}_r) \geq 20 \tag{3}$$

$$\{n_{end}, n_{end}, n_{end}\} \subseteq \mathrm{M}_r \tag{4}$$

在下文建立模型的过程中，由于 M_r 中点元素下标不连续，不便于模型的求解计算，所以首先对 M_r 中的元素有以下预处理：

建立映射关系 g_r

$$g_r : \mathrm{M}_r \to \tilde{\mathrm{M}}_r \tag{5}$$

在已知某条比赛路线且制定了打卡顺序后，对于 M_r 中的每一个点都有打卡相应的顺序，按照这样的顺序对 M_r 中的点重新排序，得到新的集合 $\tilde{\mathrm{M}}_r$

$$\tilde{\mathrm{M}}_r = \{m_j\} \qquad j = 1, 2, ..., N_r \tag{6}$$

集合 $\tilde{\mathrm{M}}_r$ 中元素的顺序即为在实际比赛中选手的打卡顺序。

每一个 m_j 对应一个 c_j ，以 c_j 代表每个打卡点对应打卡难度，不同的检查点受到打卡点附近地形和打卡要求影响，题中采用打卡所需时间作为量化指标， $c_j \in \{1, 2, 3, 4\}$ 。

4.2.2 选手的出发时刻与打卡时刻

假设本次定向越野共设计 m 条比赛路线，任一比赛路线参赛选手有一个或多个。不同路线的首名参赛选手同时从起点出发，记录为 t=0 时刻。对于第 r 条路线：

假设第 r 条路线上共 P_r 名参赛选手， p_r 表示第 r 条路线上第 p 名参赛选手， $p_r = 1, 2, ..., P_r$ 。同一条参赛路线的多名选手按照一定的时间间隔 Δt_r 依次出发，时间间隔 Δt 与路线 r 有关。

（1）选手 p_r 的出发时刻可以表示为：

$$t_{p_{r\_0}} = (p_r - 1) \bullet \Delta t_r \tag{7}$$

当 $p_r=1$时， $t=0$，即为初始时刻。

（2）记选手 p_r 在 t_{p_r} 时刻到达 m_j 打卡点，那么该选手离开 m_j 打卡点的时刻为 $t_{p_r}+c_j$。

4.2.3 路线总长度

实际参赛过程中，各选手打卡的路线受地形影响较为复杂，为了简化模型结构，按照文中提示，采取任意两打卡点间的连线作为选手的行进路线。点 m_j 与点 m_{j+1} 之间的距离 $l_{m_jm_{j+1}}$ 为：

$$l_{m_jm_{j+1}}=\sqrt{(x_{j+1}-x_j)^2+(y_{j+1}-y_j)^2+(z_{j+1}-z_j)^2} \tag{8}$$

其中， $l_{m_0m_1}$ 表示参赛选手从路线起点到第一个打卡点的直线距离； x_j， y_j， z_j 分别表示点 m_j 三个维度的坐标。

针对问题一，点 m_j 与点 m_{j+1} 之间的距离 $l_{m_jm_{j+1}}$ 退化为

$$l_{m_jm_{j+1}}=\sqrt{(x_{j+1}-x_j)^2+(y_{j+1}-y_j)^2} \tag{9}$$

比赛路线的总距离是从起点经各打卡点至终点的折线段距离之和，对于第 r 条路线，路线总长度 L_r 可表示为：

$$L_r=\sum_{j=0}^{N_r-1}l_{m_jm_{j+1}} \tag{10}$$

4.2.4 路线总爬高量

按照题中定义，当后一检查点高于当前检查点时，两点之间的高度差即为该路段爬高量，换言之当后一检查点低于当前检查点时，该路段爬高量为 0。点 m_j 与点 m_{j+1} 之间的爬高量 $h_{m_jm_{j+1}}$ 为：

$$h_{m_jm_{j+1}}=\begin{cases}0 & ,z_{j+1}\le z_j\\ z_{j+1}-z_j & ,z_{j+1}>z_j\end{cases} \tag{11}$$

其中有 $h_{m_0m_1}$ 代表由起点与第一个打卡点之间路段的爬高量； z_j 表示点 m_j 的高度信息。

将路线上所有路段的爬高量累加即为该路线的总爬高量，对于第 r 条路线，路线总爬高量 H_r 可表示为：

$$H_r=\sum_{j=0}^{N_r-1}h_{m_{j+1}}h_{m_j} \tag{12}$$

4.3 建立约束条件

4.3.1 路线总长度近似相等

对于 m 条路线而言，根据上式。。可得一系列的路线长度值 $L_1,L_2,\cdots L_m$，题目中要求不同路线长度应尽量彼此均匀，此处引入路线总长的相对极差 k_L 来表征不同路线长度之间的差异：

$$k_L=\frac{L_{\max}-L_{\min}}{L_{\text{mean}}} \tag{13}$$

$$L_{\text{mean}}=\frac{1}{m}\sum_{r=1}^{m}L_r \tag{14}$$

$$\begin{cases}L_{\max}=\max\{L_r\}\\ L_{\min}=\min\{L_r\}\end{cases},i=1,2\cdots,m \tag{15}$$

其中 L_{mean} 表示某设计方案中所有路线长度的均值， $L_{\max}$ 、 $L_{\min}$ 分别代表路线长度的最大值和最小值；

设定阈值 K_L，故路线总长度近似相等的约束条件为

$$k_L \le K_L \tag{16}$$

选取的K_L值越大，能够容忍的路线长度之间的差异值即越大。

4.3.2 路线总打卡难度近似相等

由上文定义容易得出第 r 条路线的总打卡难度C_r：

$$C_r = \sum_{j=1}^{N_r} c_j \tag{17}$$

c_j 代表在m_j检查点选手打卡难度，用时间衡量。

引入路线总打卡时间的相对极差 k_C:

$$k_C = \frac{C_{\max} - C_{\min}}{C_{\text{mean}}} \tag{18}$$

$$C_{\text{mean}} = \frac{1}{m}\sum_{r=1}^{m} C_r \tag{19}$$

$$\begin{cases} C_{\max} = \max\{C_r\} \\ C_{\min} = \min\{C_r\} \end{cases}, r = 1, 2\cdots, m \tag{20}$$

其中C_{mean}表示所有路线总打卡难度的均值，$C_{\max}$、$C_{\min}$分别代表路线总打卡难度的最大值和最小值；

同样地，设定阈值K_C，故路线总打卡难度近似相等的约束条件为

$$k_C \le K_C \tag{21}$$

选取的K_C值越大，能够容忍的路线打卡难度之间的差异值即越大。

4.3.3 路线总爬高量近似相等

为了控制各路线受地形起伏影响的差异，引入路线总爬高量的相对极差k_H：

$$k_H = \frac{H_{\max} - H_{\min}}{H_{\text{mean}}} \tag{22}$$

$$H_{\text{mean}} = \frac{1}{m}\sum_{r=1}^{m} H_r \tag{23}$$

$$\begin{cases} H_{\max} = \max\{H_r\} \\ H_{\min} = \min\{H_r\} \end{cases}, r = 1, 2\cdots, m \tag{24}$$

其中H_{mean}表示所有路线爬高量的均值，$H_{\max}$、$H_{\min}$分别代表路线爬高量总和的最大值和最小值。

4.3.4 提高路线空间隔离度

为了保证各参赛选手独立完成比赛，要减少选手在比赛中距离较近的情况，避免选手“互相帮助”。作为比赛路线的制定者，自然无法预知所有参赛选手在实际比赛过程中的行进速度，因此为了降低选手发生距离过近的可能性，较容易实现的方法是使每一条路线都尽可能分散。路线完全的分散是无法实现的，各路线之间不可避免地会共用某些检查点。

本组引入空间隔离度 k_D 来衡量各路线之间空间上的分散程度。空间隔离度 k_D 与路线的两个方面有关：路线的共用检查点数量n_{share}和路线的交叉点数量n_{cross}。

分别求出n_{share}和n_{cross}后，得到空间隔离度 k_D：

$$k_D = \frac{c}{k_1 N_{cross} + k_2 N_{cross}} \tag{25}$$

其中c为比例系数；k_1，k_2分别为相应的权重系数。

本质上空间隔离度 k_D 描述的是一个概率，即 k_D 越大空间隔离越高，选手比赛过程中发生跟

跑、相互交流的可能性就越低。显然因为选手在每一个检查点要停留 1-4 分钟不等，选手间距离较近的概率要远大于路线中存在交叉点所带来的相互靠近的概率，设定权重系数 k_1 要远大于 k_2。

设定阈值 K_D,要求

$$k_D \geq K_D \tag{26}$$

下面具体表示路线的共用检查点数量 N_{share} 和路线的交叉点数量 N_{cross}。

（1）路线共用检查点数量 N_{share}

以第 i 条和第 j 条路线为例，两条路线包含的检查点构成的集合分别为 $\tilde{\mathbf{M}}_i$ 和 $\tilde{\mathbf{M}}_j$，这里为了寻求公共元素，使用相应的未作映射的原点集 $\mathbf{M}_i$ 和 $\mathbf{M}_j$。两条路线共有的检查点表示为集合 $\mathbf{M}_{ij}$，从而得到第 i 条和第 j 条路线共有的检查点数量 N_{ij_share}：

$$\mathbf{M}_{ij} = \mathbf{M}_i \cap \mathbf{M}_j \tag{27}$$

$$N_{ij\_share} = crad(\mathbf{M}_{ij}) \tag{28}$$

根据握手原理，容易求得 N_{share}：

$$N_{share} = \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m} N_{ij\_share} \tag{29}$$

（2）路线的交叉点数量 N_{cross}

以第 i 条和第 j 条路线为例，两条路线包含的检查点数量分别为 N_i 和 N_j；定义一个 $(N_i-1)\times(N_j-1)$ 维的矩阵 $\mathbf{N}_{ij\_cross}$ 存放第 i 条和第 j 条路线交叉点的数量。第 i 条路线的第 1 段路线与第 j 条路线的第 1 段路线存在交点的数量（0 或 1）存入矩阵 $\mathbf{N}_{ij\_cross}$ 中的第 1 行第 1 列，相应地，第 i 条路线的第 p 段路线与第 j 条路线的第 q 段路线存在交点的数量（0 或 1）存入矩阵 $\mathbf{N}_{ij\_cross}$ 中的第 p 行第 q 列。从而易得 N_{cross}：

$$N_{cross} = \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m}\left\|\mathbf{N}_{ij\_cross}\right\|_{m_1} \tag{30}$$

4.3.4 减少重复路段

为了避免参赛选手在跑动过程中后面跟着前面选手行进，要减少路线之间相同的路段下图中绿色、蓝色和黄色分别表示三条路线，记为路线 a,b,c，其中路线 a 与 b、b 与 c、a 与 c 分别重复了 2、2、1 段路线。

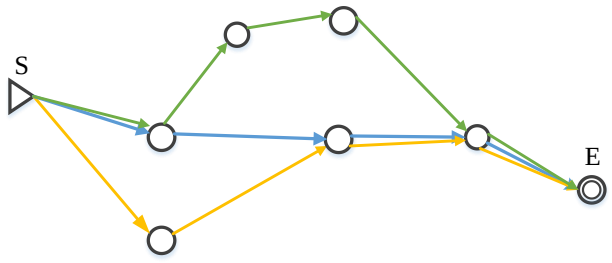


图 3 重复路段示意图

下面计算一个路线方案中总的路线重复路段数量 $N_{overlap}$。

以第 i 条和第 j 条路线为例，由上文式得到由两条路线共有的检查点构成的集合 $\mathbf{M}_{ij}$。当 $N_{ij} \geq 2$ 时，分别作映射 g_i 和 g_j，得到公共检查点在各自集合 $\tilde{\mathbf{M}}_i$ 和 $\tilde{\mathbf{M}}_j$ 中的排序，判断其是否连续,累计所有连续的公共打卡点所构成的路段，即为以第 i 条和第 j 条路线的重复路段数量 $\mathbf{N}_{ij\_overlap}$，易得 $N_{overlap}$：

$$N_{overlap} = \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m} N_{ij\_overlap} \tag{31}$$

容易求得 m 条路线的总路段数 N:

$$N=\sum_{r=1}^{m}\left[crad(\mathrm{M}_r)\right] \tag{32}$$

引入路段相对重复率 $k_{overlap}$，设定阈值 $K_{overlap}$，从而有约束条件：

$$k_{overlap}=\frac{N_{overlap}}{N} \tag{33}$$

$$k_{overlap}\geq K_{overlap} \tag{34}$$

4.3.5 限制同时到达检查点人数

现考虑检查点 A，它位于第 r_1 条路线上的第 s_1 个节点，那么第 r_1 条路线上的第 p 个人 p_{r_1} 到达点 A 的时刻 $T_{pr_1s_1}$ 结合式（1）-（1）易求得为

$$T_{pr_1s_1}=\frac{L_{r_1-s_1}}{v_{p_{r_1}}}+C_{r_1-s_1}+t_{p_{r_1-0}} \tag{35}$$

$$L_{r_1-s_1}=\sum_{j=0}^{s_1-1}l_{m_jm_{j+1}} \tag{36}$$

$$C_{r_1-s_1}=\sum_{j=1}^{s_1-1}c_j \tag{37}$$

$$t_{p_{r_1-0}}=(p_{r_1}-1)\bullet\Delta t_{r_1} \tag{38}$$

其中 $L_{r_1-s_1}$ 表示第 r_1 条路线从起点开始到的第 s_1 个节点的总路线长，$C_{r_1-s_1}$ 表示第 r_1 条路线从起点开始到的第 s_1 个节点之前的总打卡时间，$t_{p_{r_1-0}}$ 表示第 r_1 条路线第 p 个人的出发时刻，$v_{p_{r_1}}$ 表示第 r_1 条路线第 p 个人的行进速度。

从而能够得到所有第 r_1 条路线上选手到达的第 s_1 个节点的时刻序列 $\mathbf{T}_{r_1s_1}$

$$\mathbf{T}_{r_1s_1}=\left\{T_{pr_1s_1}\right\}\qquad p=1,2,...,P_{r_1} \tag{39}$$

当检查点 A 同时也位于第 r_2 条路线上的第 s_2 个节点，相应地可以得到另一组时刻序列 $\mathbf{T}_{r_2s_2}$

现将检查点 A 所对应的所有时刻序列取并集得到 $\mathbf{T}_{\mathrm{A}}$，将所有时刻按照从小到大排序，在时间轴上得到了所有参赛选手到达检查点 A 的时刻序列，要求满足下式(单位取分钟)：

$$\forall T_i,T_{i+4}\in\mathbf{T}_{\mathrm{A}},\ T_{i+4}-T_i\geq 1 \tag{40}$$

对规划的路线中的所有节点都要类似地满足上式，便能够保证一分钟内到达任一节点的人数不超过 5 人。

4.4 建立目标函数

4.4.1 问题一&问题二目标函数

由前文的问题分析知，问题一和问题二的目标函数相同，仅约束条件有所不同。在满足 4.3 中所有约束条件的前提下，要使得总参赛选手可能多。假设第 r 条路线上共 P_r 名参赛选手，总参赛人数 P_{all}

$$P_{all}=\sum_{r=1}^{m}P_{r\_\max} \tag{41}$$

其中 $P_{r\_\max}$ 是满足了约束条件的最大值。

那么目标函数为

$$J_1=\max_{k=1}^{\mathrm{K}}\left\{J_{1\_k}\right\} \tag{42}$$

$$J_{1\_k}\left\{\tilde{\mathrm{M}}_r\right\}_{r=1,2,...,k}=\max\left\{P_{\mathrm{all}}\right\} \tag{43}$$

其中 k 为路线规划方案中路线的条数,K 为路线的条数可能存在的最大值。

目标函数 J 为广义函数，在实际模型求解中用程序仿真来替代该函数的解析形式。

4.4.2 问题三目标函数

问题三中总人数不再是目标函数，而是严格约束条件，此时的目标函数变为两条，且这两条约束条件是相互制约的。由于此时仍然有每个检查点同时到达人数小于 5 这一约束条件，意味着且参赛选手的速度各不相同，意味着每条路线上选手出发的间隔总有一个下限，那么未来在 3.5 小时内结束比赛，就不可能无限制地减少路线条数。因此，本组认为不妨优先考虑减少使用路线的数量，再使完成时间尽可能短；

（1）使用路线数量尽可能少

$$J_2 = \min(m) \tag{44}$$

（2）完成时间尽可能短

设第 j 条路线的最后一个人 $(P_r)_j$ 恰为所有比赛选手中最后一名完成比赛的，结合前文式（）可求 $(P_r)_j$ 到达终点的时刻 $T_{(P_r)rn_{end}}$，$T_{(P_r)rn_{end}} \le 3.5$ (单位取小时)

$$J_2 = \min(T_{(P_r)rn_{end}}) \tag{45}$$

4.5 建立评价函数

在经过多层约束条件限制后，得到一个可求解空间。此时要通过评价函数选取一个最优解。评价函数由所有宽松性约束条件及目标函数联合加权求得。

5 模型求解

5.1 确定终点与必经检查点的选取空间

如果路线中的所有打卡点的位置不定，在所有的候选检查点中自由选择，穷举所得的解空间将十分巨大，且存在很多实际上与题中路线规划原则相悖的解。由题目知，存在两个所有比赛路线必经的检查点，且所有路线的终点相同。对于这一类特殊的点，倘若能够根据实际比赛路线的约束条件和目标函数求取这几个特殊点合理的空间分布，优先在这样的空间分布中选取，便可以将大大减小解空间的规模。

5.1.1 确定终点的分布空间

从题中的约束条件和目标函数出发，为了提高不同路线之间的空间隔离度，使路线分散，要尽可能用到所有候选的检查点；确定终点位置有以下三条原则：

①终点要选取打卡难度较低的点

打卡难度按照时间来衡量一共有 1、2、3、4 分钟这四种，在这里认为打卡难度为 1 分钟和 2 分钟的都属于打卡难度较低的。

②终点与起点的直线距离尽可能大

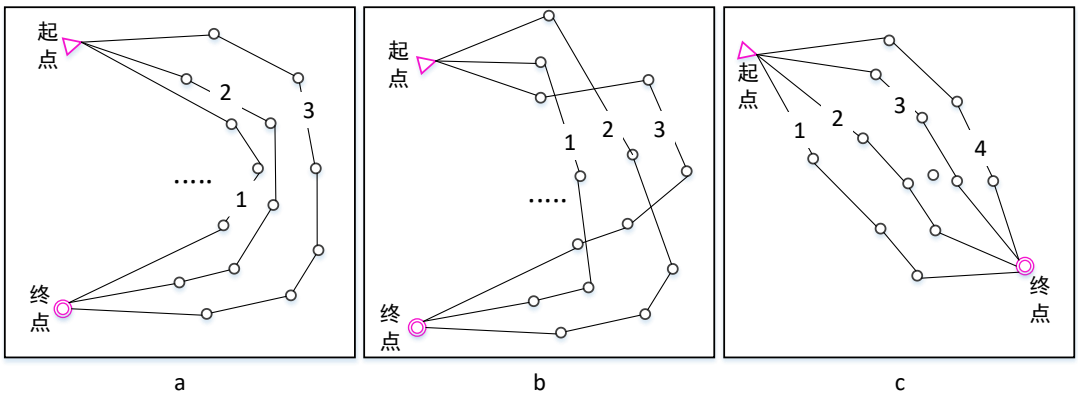
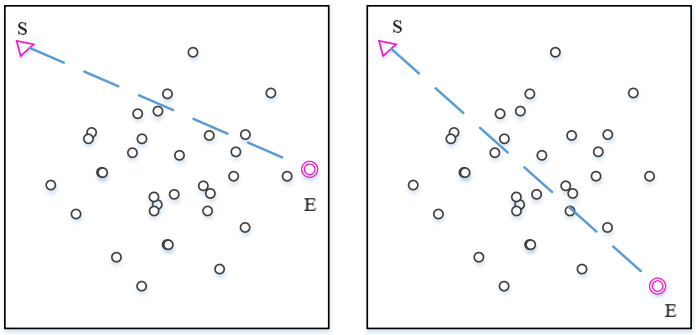


图 4 起点与终点相对位置分布示意图

如果起点与终点距离较近，那么路线的分布可以归纳为可以如上图 a，b 两种形式。对于图 a 的情况，路线 1、2、3 的行进路线距离相差较大，存在一定的不公平现象。对于图 b，

1、2、3 三条路线还存在一定的路线交叉的问题，不满足题中要求的参赛选手距离较近的约束，一定概率出现运动员跟跑或者相互交流等现象。因此，按照本条路线设计的原则，使得起点与终点的直线距离尽可能大，如图 c 所示，可能位于打卡点地图区域的对角位置。该情况下，多条路线可以较为平均的分布在起点与终点连线两侧，最大限度的利用全地图区域内的检查点。

③起点与终点的连线尽可能将整个比赛场地分为两个对称的部分



a　　b

图 5　起点终点连线划分区域示意图

若采用起始点位置关系如如图 a 所示，起点与终点连线将比赛场地分成的两个区域内所包含候选打卡点数相差较大，路线设计过程中由于受到路线长度均匀的条件约束，大部分左下方范围的打卡点无法纳入设计路线的可选打卡点集；若将起始点按照图 b 所示进行路线规划，则可以尽可能多的运用到所有的打卡点，为布置必经打卡点位置提供更多的选择。最终问题一和问题二中候选终点分布如图 6、7，问题三的终点分布类似于问题二，在此不再单独展示。

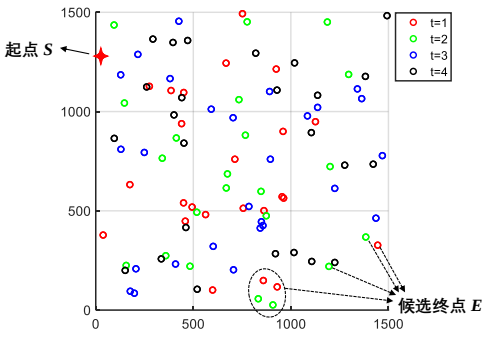


图 6　问题一候选终点分布

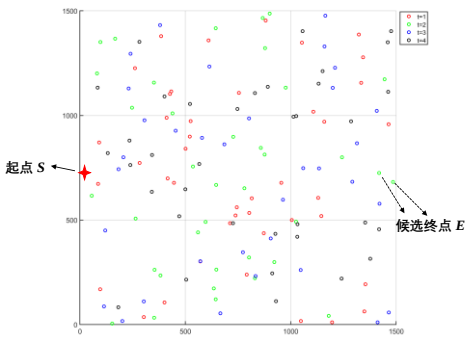


图 7　问题二候选终点分布

5.1.2 确定必经打卡的分布空间

题目中要求本次比赛所有路线应包含两处必经打卡点，必经打卡点的引入是为了对竞赛过程加以管控，同时也是为竞赛过程选手可能发生的突发情况提供基本的安全与医疗保障。

确定必经检查点的原则（两个必经检查点中靠近起点的为点 M_{check1}，另一个记为点 M_{check2}）：

①尽可能使得两个必经打卡点将所有路线分成近似相等的三段

②考虑 M_{check1}、M_{check2} 和起点 S 终点 E 的相对位置关系；

③使其尽可能位于整个区域的中心位置

结合问题一、二、三附件中所提供的候选检查点分布，现有两种布置方案如下：

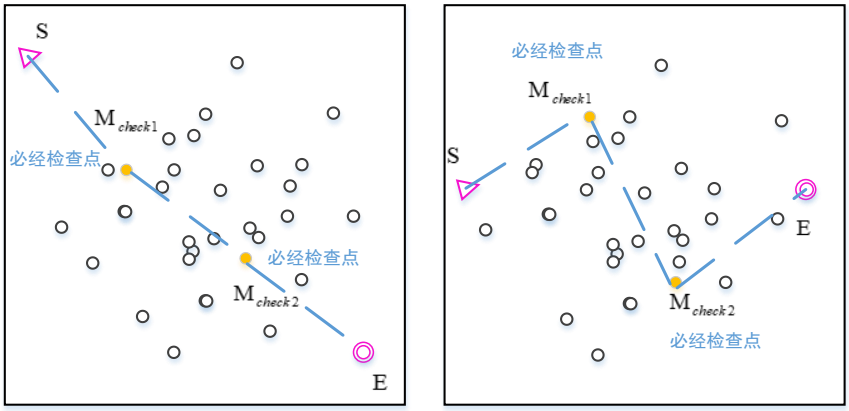


图 8　必经检查点分布示意图

图 8（a）所示方案，起点终点位于整个比赛区域的两个对角。必经检查点大致分布在起始点与终点连线上，两个必经检查点将起点与终点连线分成三段，将区域内所有候选检查点向起点终点连线投影，投影点落在三段连线内的数量大致一致。

图 8（b）所示方案，起点终点位于整个比赛区域的两条对边中点。此时如果使必经检查点大致分布在起始点与终点连线上，会造成靠近上、下边缘的众多候选打卡点出现闲置，因此做出如图所示的调整。

最终得到候选终点以及相应的候选必经打卡点的情况见附录中的表格（分别使用附件 1 和 2 中点的序号表示相应的检查点），其分布示意图如下

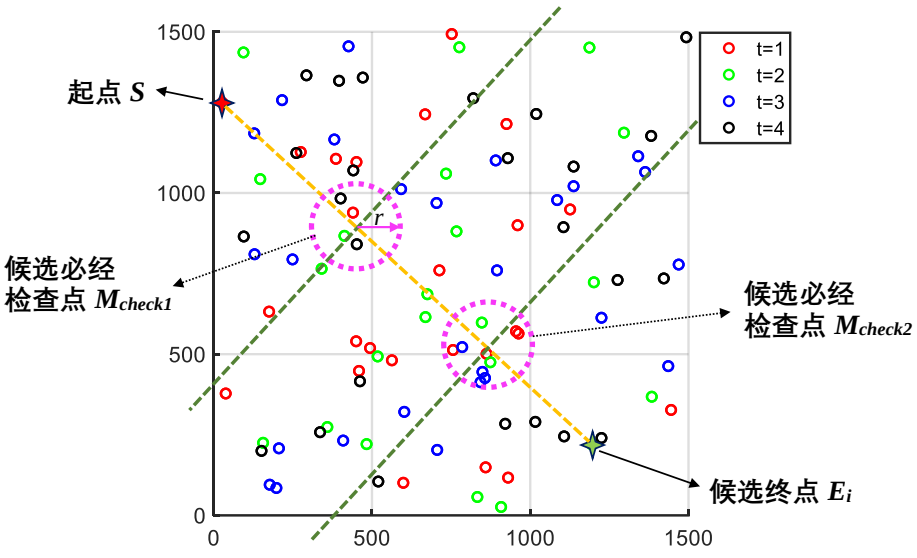


图 9　问题一候选必经检查点分布

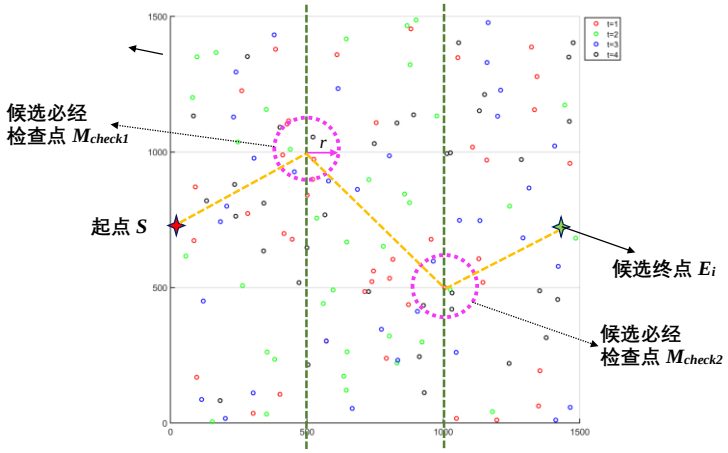


图 10　问题二候选必经检查点分布

5.2 单个区域寻找候选路线

在确立了终点及两个必经检查点后，以两个必经打卡点为中心，按照合适的方式将整个比赛区域划分成了点分布相对均匀的三个区域。在全局范围内寻找过必经点路线的问题被转化为三个局部区域内寻找路线并将其组合，且在每一个区域内的路线除起点终点外不存在必经点。下面就详细考虑如何在单个区域内挑选出候选路线。

5.2.1 确定检查点路线规划模型

针对已知若干检查点的路线规划，利用辅助量 $L_{\min}$，它表示从起点 S 到终点 E 的最短长度路线的数值。

$$L_{\min}=\sum_{i=0}^{N_r-1} l_{m_i m_{i+1}}=\sum_{i=0}^{N_r-1}\sqrt{(x_{i+1}-x_i)^2+(y_{i+1}-y_i)^2+(z_{i+1}-z_i)^2} \tag{46}$$

依据**贪婪算法**的思想，从起点 S 出发，依照就近原则寻找路线节点，使路线不断拓展，从而得到符合路线尽可能短、临近的几个点打卡顺序尽可能连续的要求的较优的局部最优解。具体的算法寻点思想可由下图示意：

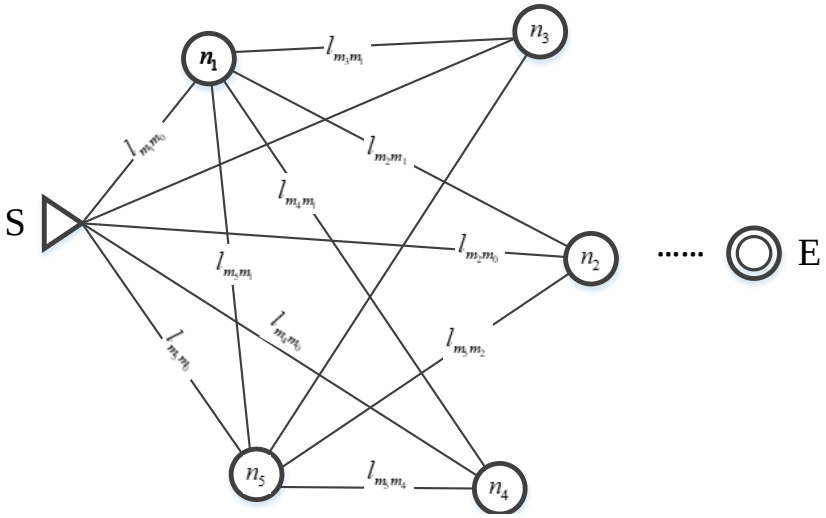


图 11　路线规划模型示意图

首先我们选择辅助量 $l_{m_i m_j}$ 作为算法的寻优参考，表示由节点 n_i 到达节点 n_j 的路线距离，如上图路线由起点 S 出发，存在 $l_{m_1 m_0}, l_{m_2 m_0}, \cdots\cdots l_{m_i m_j}$ 若干的路线选择，选取其中

$l_{\min}(i)=\min(l_{m_i m_j})$，其中 i,j 是比赛区域两独立打卡点。照此方法，由起点依次寻找符合条件的最佳节点连接后，最终可得到将起点、终点和若干打卡点联接，构成一条带有顺序的路线。

5.2.2 候选路线树生长模型

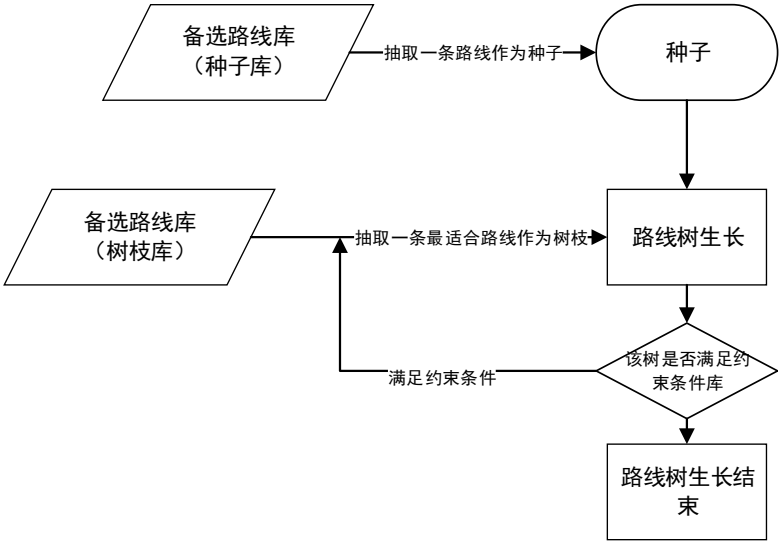


图 12　路线生长树模型示意图

5.3 分段路线匹配寻找全局路线

上文指出，在路线设计规划时，利用两必经打卡点 M_{check1}、M_{check2} 作为节点，将打卡点区域沿起点与终点方向截分为三段。在具体解题过程中，我们容易得出如下图所示的路线预选空间。

对于总的定向越野区域来说，从图中容易得出设计路线的最大数量为 $m_1 \cdot m_2 \cdot m_3$。由于题目中给出了路线长度、爬高量、打卡难度等若干的设计约束原则，我们将不符合条件的设计路线从中删除。

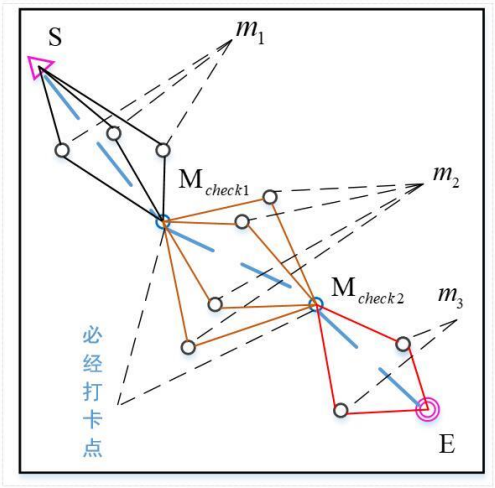


图 13　路线预选空间示意图

5.4 容纳人数上限求解

以上的模型已经将最优的路线方案匹配得出，对于已经确定的路线方案，还需要确定路线容纳的人数上限，这里本文使用如下模型迭代寻找上限。

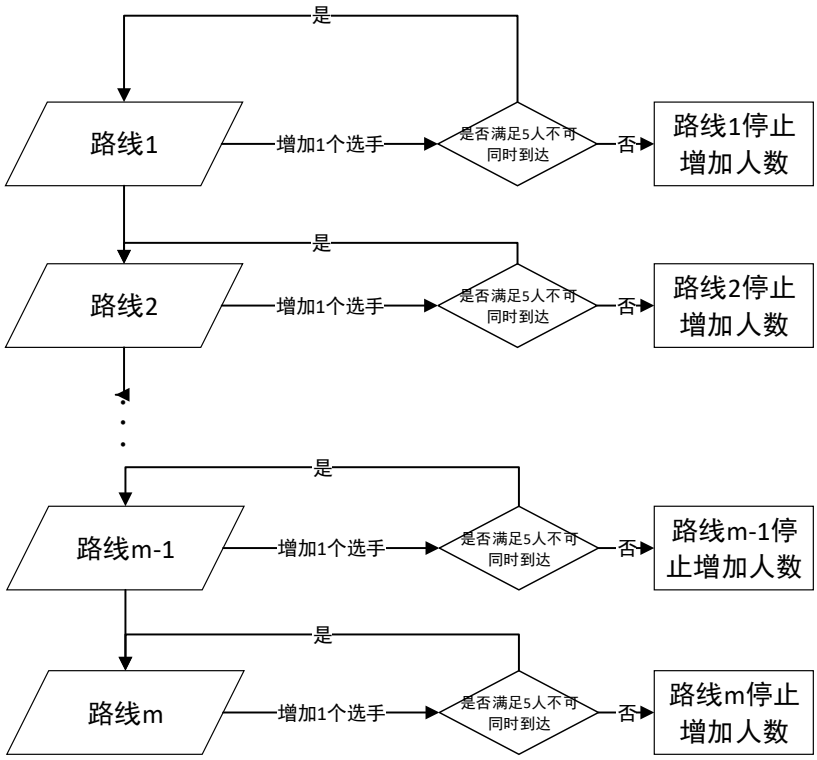


图 14　容纳人数上限迭代求解流程图

核心是循环增加路线的人数，直到该路线不再满足严格条件和宽松条件，例如：同时到达检查点的人数不超过 5 人。最终当所有路线均稳定收敛无法增加人数时，得到当前人数作为路线的人数上限。

5.5 蒙特卡洛仿真

针对**问题三，**首先由问题一和问题二使用的模型，针对问题三的检查点分布，在人数指定情况下，可以得到指定路线数目的最优路线设计（时间最短）。所以，问题三转化为如何找到最优的路线数目作为求解目标，这里我们使用 1000 次随机指派选手参赛进入路线进行蒙特卡洛实验。一个优秀的鲁棒性好的路线设计应当具有适应不同跑速选手的能力，因此，路线设计方案从 2 条路线开始逐步增加，分别进行 1000 次蒙特卡洛实验，并且统计分析，路线设计方案如下所表示：

表 2　蒙特卡洛仿真路线方案设计表

路线条数	路线 1		路线 2		路线 3		路线 4		路线 5	
	人数	时间间隔	人数	时间间隔	人数	时间间隔	人数	时间间隔	人数	时间间隔
2	58	1	62	1						
3	42	1.2	38	1.2	40	1.2				
4	31	1.5	31	1.5	32	1.5	26	15		
5	25	1.6	24	2	26	2	23	16	24	2

对上表中 4 种方案进行蒙特卡洛实验，随机生成 1000 种随机选手参赛分布，进入比赛

路线，统计得到的检查点同时到达 5 人的概率和 1000 次活动完成的平均用时。

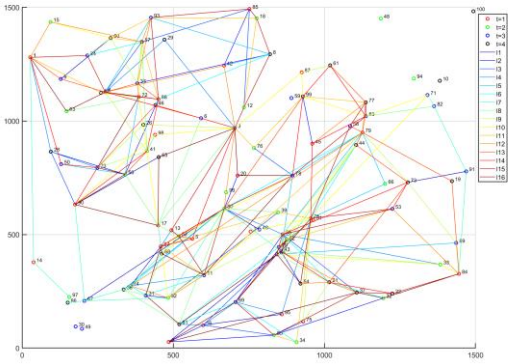
表 3　蒙特卡洛仿真路线方案结果表

路线条数	某检查点蒙特卡洛仿真同时出现 5 人的概率	蒙特卡洛仿真平均活动完成用时
2	0.167	3.535
3	0.109	3.367
4	0.048	2.951
5	0.041	2.876

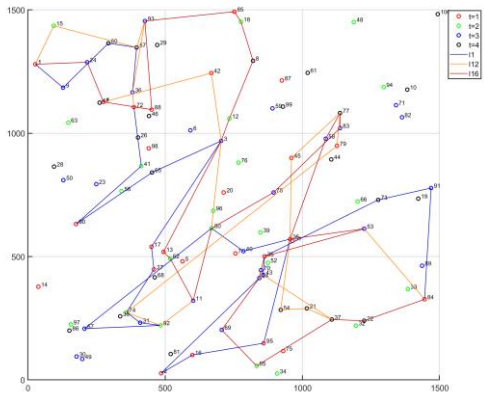
由表 3 直观地得出，随着路线条数的增多，某检查点蒙特卡洛仿真结果中同时出现 5 人的概率和平均活动完成用时都显著降低。问题三要求使用的线路尽可能少，因此本组从路线条数等于 2 向上迭代增加。由表 3 当选取蒙特卡洛仿真定向越野路线为 4 条时，在同时五名参赛者在同一节点出现概率、比赛路线地参赛容纳量及仿真活动平均用时三个方面，得到一个较好的妥协值。

6 结果分析

6.1 问题一结果分析



a.问题一路线设计方案全集



b.问题一路线设计方案部分展示

图 15　问题一路线结果

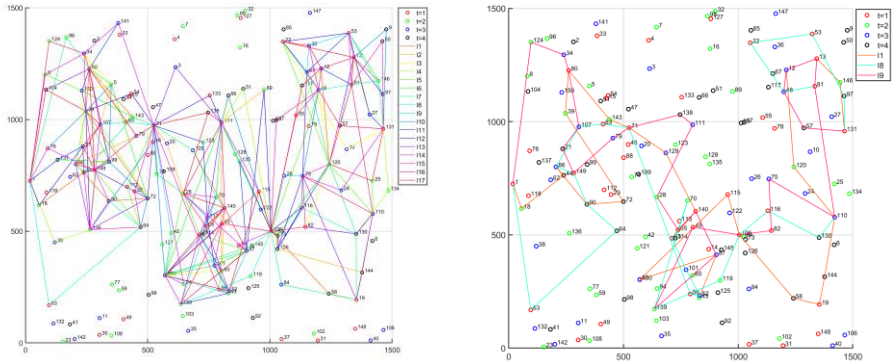
根据上文给出的初始分布状态和路线的粗略设计策略，可以得到路线设计全集，由图 15 可以明显地看出路线在整个定向越野地域仅仅是简单地节点遍历后进行路线构建，路线在节点选取、路线分布方面未加约束，大部分路线之间存在交叉点和重复路段，影响实际比赛中的寻点打卡与赛事公平独立原则。同时，由于贪婪算法的条件局限性，使得节点选取较为集中，无法拉开参赛选手间间隔，影响比赛成绩的权威参考性。图 15（a）（b）基于上述考虑，对路线全集进行重新评价，利用评价函数优化设计方案，最后得出 16 条较为理想的比赛路线，见表 4。

表 4　问题一路线设计结果

序号	总路程	总难度	总时间
1	6663.454	62	2.143909
2	6311.786	60	2.051964
3	6263.143	59	2.02719
4	6134.588	60	2.022431

5	6142.404	63	2.073734
6	6578.559	66	2.196426
7	6260.087	64	2.110014
8	6453.86	66	2.175643
9	5992.958	70	2.165493
10	6831.829	62	2.171972
11	5992.64	68	2.132107
12	6210.072	64	2.101679
13	5848.454	70	2.141409
14	6069.836	64	2.078306
15	5912.557	67	2.102093
16	6342.153	65	2.140359

6.2 问题二结果分析



a.问题二路线设计方案全集　　b.问题二路线设计方案局部展示

图 16　问题二路线结果

对问题二的求解算法与问题一大致相似，对于新的打卡点数据进行相同的处理算法，利用文中给出的相关约束条件对所有初始设计方案进行筛选后得到十七条较为符合条件的设计路线，如图（a）所示。对其中几条进行展示，更易看出路线的设计中覆盖了定向越野比赛区域的尽可能大的范围，可以很好的检验参赛者定向水平，进一步提升参赛体验。

同时，根据实际数据测量值也不难得到，设计得到的路线方案基本满足参赛者彼此路线距离、比赛爬高量、打卡点总打卡难度相持平，保证比赛成绩客观、可靠。如下表数据值可知，17 条路线的总录车工、总难度、总爬高量、跑完全程的总用时处于均匀水平。

表 5 问题二路线设计结果

序号	总路程/m	总难度/min	总时间/h	总爬高量/m
1	6082.821	62	2.03047	58
2	6452.271	60	2.058712	50
3	6557.666	57	2.026278	50
4	6758.167	56	2.043028	62
5	6759.62	54	2.009937	60
6	5764.204	61	2.022943	55
7	5963.992	59	2.04646	52

8	6469.494	54	2.041869	53
9	6203.552	59	2.000592	53
10	6221.42	60	2.020237	54
11	5910.179	63	2.018363	43
12	6418.459	56	2.013602	52
13	6257.035	60	2.026172	44
14	6681.398	55	2.013566	43
15	6287.366	60	2.031228	62
16	5787.097	57	2.001142	43
17	6179.684	56	2.008724	51

6.3 问题三结果分析

根据蒙特卡洛仿真结果得出结论，选取路线数为 4，根据题中最佳路线设计原则，最终对打卡区域的路线设计如下图所示，可以达到使用路线尽可能少，路线完成时间尽可能短的要求。

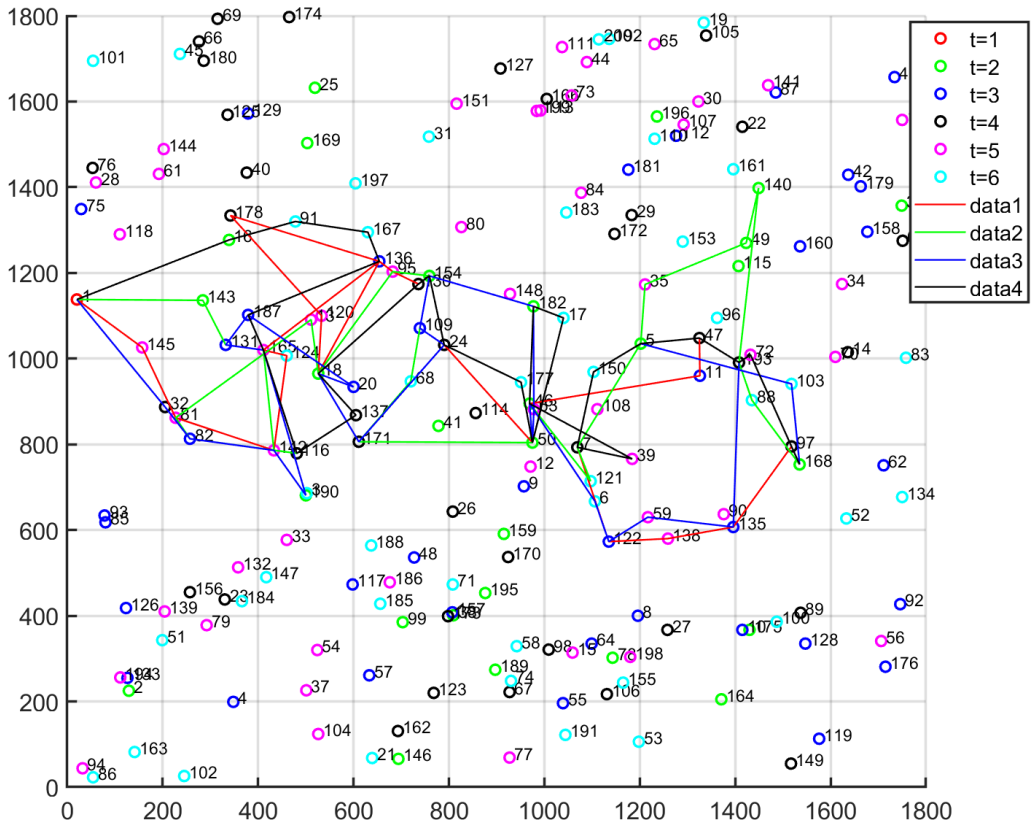


图 17　问题三路线结果

根据对 4 条路线的实际数据解算，对 120 名参赛选手进行路线的分配，得出路线的选手数量、出发间隔时间、总长度、总打卡难度、总爬高量的具体数值如下。

表 6　问题三路线设计结果

路线序号	选手数量	间隔（分钟）	总长度/m	总难度/min	爬高量/m
1	31	1.52258499	4545.55	96	67
2	31	1.52548575	4537.01	97	75

3	32	1.52849414	4649.60	90	71
4	26	1.50120091	4251.13	100	67

从表 6 中数据可以判断出，本文的求解结果中四条路线人员分配均匀不存在拥堵现象，间隔时间和路线总长度相近，参赛选手关门时间能保证相差不大，对于打卡点的难度衡量值来说基本稳定在 90-100 附近，对最终的成绩影响较小，爬高量数值相差属于可容范围，因此本次设计路线设计利用蒙特卡洛仿真确定设计数目，更加贴近路线最佳设计要求，同是最终的数据看出路线对竞技中可能存在的主观影响因素进行了科学限制，最终的成绩具有公平性和可参考价值。

7 敏感性分析

对于本题系列问题的求解，主要是针对必经点的路线规划，而节点的选择过程采取贪婪思想寻优建立可行解空间，通过对算法和模型的改进处理，可以克服由于定向越野打卡点的不规则分布和打卡难度的不统一性。

另外，问题的路线求解具有普适性，算法针对不同的打卡点基础数据，可以自适应的针对新的节点位置做出新的路线调整，同时调整算法的约束函数，可以得到不同使用场景的路线。假设约束条件门限阈值高时，路线数量就会变大，路线的重复率和跟跑现象升高；反之，路线数量减少，具有较好的公平性和独立性原则。

对问题一模型求解过程进行重新求解，为了凸显两次模型求解中对重复节点和重复路段参量变化值得敏感程度，严格约束第二次求解中无重复路段，完全规避定向越野比赛中的跟跑现象。求解模型后得到满足严苛条件的路线有 7 条，如下图所示。

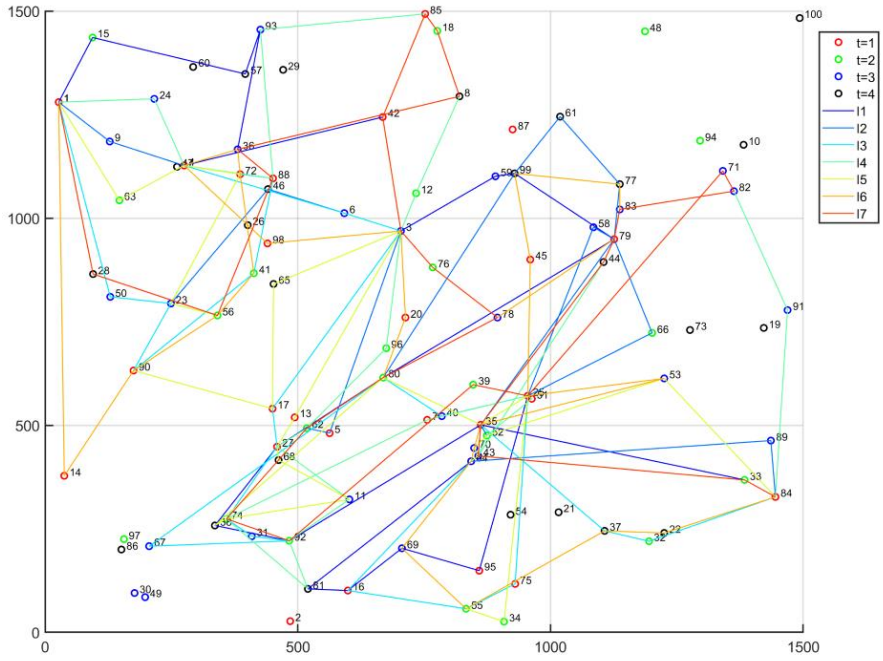


图 18　严格无重复的七条路线

对比两次模型求解结果，严格约束和非严格约束条件下路线数量和容纳参赛

选手数量对比如下。

表 7 敏感性分析结果

	严格约束	非严格约束
路线数量	7	16
参赛选手最大容量	213	215

可以看出，重复路段的存在与否，基本没有影响到路线的总容纳参赛选手人数，但是，两次设计路线存在较大差异，而在参赛容量相同时，较多的路线更符合路线设计中的公平、独立原则，也有利于为参赛者提供可信的比赛成绩。

参考文献

[1] 吴海峰. 最短路线算法——Dijkstra 及 Floyd 算法[J]. 中国新通信, 2019, 21(02):37-38.

[2] 李宁宁, 刘玉树. 改进的 Dijkstra 算法在 GIS 路线规划中的应用[J]. 计算机与现代化, 2004, 000(009):12-14.

[3] 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.

[4] 王书勤, 黄茜. 军事定向越野路线优化问题建模及混合蚁群算法求解[J]. 运筹与管理, 2018, 027(004):105-111.

附录

```
%  数据初始化%%%%%%%%%%%%%%%%%%%%
clear all;
load Q1_pre.mat;
此矩阵包含所有检查点和终点的检查点列表,序号,x,y,t
global start; %起点的序号 1,x,y
start=[start,1];
P_xyt=[start;P_xyt];
global P_xyt; %起点的序号 1,x,y
endP = [84,1445,327,1]; %设定终点的候补
% 计算起点到终点的连线，根据所有点投影到连线上的三分位置点，作为补给点,并且输出具有灵活度的划分方式
r_range=120; % 设置 10 米作为备选补给点的选择范围
[buji1,buji2,Part1,Part2,Part3]=cal_buji(start,endP(1,:),P_xyt(2:end,:),r_range);
%
P_t1 = P_xyt(find(P_xyt(:,4)==1),:);
P_t2 = P_xyt(find(P_xyt(:,4)==2),:);
P_t3 = P_xyt(find(P_xyt(:,4)==3),:);
P_t4 = P_xyt(find(P_xyt(:,4)==4),:);
% 选取三等分点和终点，对检查点进行划分，划分成 3 个区间，将问题分解成 3 个子问题
%遍历求解第一部分,计算三种的上下界，首先下界肯定是 6，因为不少于 20 个节点，已经有两个作为补给点了
%关键要求上界，其余两个部分取下界最小，剩下的部分按 3h 能取到上界
Part1(find(Part1(:,1)==buji1(1,1)),:)=[];
Part1(find(Part1(:,1)==start(1,1)),:)=[];
t1_lb=cal_lb(Part1,start,buji1(1,:));%t1_lb

Part2(find(Part2(:,1)==buji1(1,1)),:)=[];
Part2(find(Part2(:,1)==buji2(1,1)),:)=[];
t2_lb=cal_lb(Part2,buji1(1,:),buji2(1,:));

Part3(find(Part3(:,1)==buji2(1,1)),:)=[];
Part3(find(Part3(:,1)==endP(1,1)),:)=[];
t3_lb=cal_lb(Part3,buji2(1,:),endP(1,:));


%以上计算出下界全是 6，6，6，上界为 14，14，14，根据上下界，遍历所有的路线
%% 接下来，根据上下界，从 Part1 中挑选 6 个选 7 个。。。一直到上界 14，对这些路线就是备选的解空间，首先设定至少 5 条线，每条路至少 16 人，出发时间间隔至少 1
分钟
% 去除打卡点和终点
Part1(find(Part1(:,1)==buji1(1,1)),:)=[];
Part1(find(Part1(:,3)<=59),:)=[];
Part1(find(Part1(:,1)==buji1(1,1)),:)=[];
Part2(find(Part2(:,1)==buji1(1,1)),:)=[];
Part2(find(Part2(:,3)>=1294),:)=[];
Part2(find(Part2(:,3)<=57),:)=[];
Part2(find(Part2(:,2)<=200),:)=[];
Part2(find(Part2(:,1)==buji2(1,1)),:)=[];
Part3(find(Part3(:,1)==buji2(1,1)),:)=[];
Part3(find(Part3(:,1)==endP(1,1)),:)=[];
Part3(find(Part3(:,3)>=1177),:)=[];
选择 5 个出发点
start_out=[15,24,9,28,14,63,50,47];
for i=1:length(start_out)
    Part1(find(Part1(:,1)==start_out(i)),:)=[];
end
buji1_in=[42,12,6,98,65,26,46,29];
for i=1:length(buji1_in)
    Part1(find(Part1(:,1)==buji1_in(i)),:)=[];
end
buji1_out=[87,59,76,17,13,5,20,96];
for i=1:length(buji1_out)
    Part2(find(Part2(:,1)==buji1_out(i)),:)=[];
end
buji2_in=[43,35,39,44,45,40,58,7];
for i=1:length(buji2_in)
    Part2(find(Part2(:,1)==buji2_in(i)),:)=[];
end
buji2_out=[54,66,53,71,51,95,75,34];
for i=1:length(buji2_out)
    Part3(find(Part3(:,1)==buji2_out(i)),:)=[];
end
endP_in=[91,89,31,22,53,33,32,19];
for i=1:length(endP_in)
    Part3(find(Part3(:,1)==endP_in(i)),:)=[];
end


%% 第一部分
rout_all1=[];
for k=1:8%遍历出发 5 个点
    for m=1:8%遍历进入之后的 5 个点
        % k=1;m=1;
        for j=6:7
```

```
                Part_S=nchoosek(Part1(:,1),j);
                Part_S_=Part_S;
                p=0;
                for i=1:size(Part_S,1)
                    %                       for i=1:1
                    rout=[start(1,1),start_out(k),Part_S(i,:),buji1_in(1,m),buji1(1,1)];
                    [rout1p,d]=number_dist(rout,rout(1));
                    if rout1p(1,23)<0.75&&rout1p(1,23)>0.63
                        rout_all1=[rout_all1;rout1p];
                    end
                end
            end
        end
end
save 'Q1_1_3.mat' rout_all1


%% 第二部分
rout_all2=[];
for k=1:8%遍历出发 5 个点
    for m=1:8%遍历进入之后的 5 个点
        % k=1;m=1;
        for j=6:7
            Part_S=nchoosek(Part2(:,1),j);
            Part_S_=Part_S;
            p=0;
            for i=1:size(Part_S,1)
                %                       for i=1:1
                rout=[buji1(1,1),buji1_out(1,k),Part_S(i,:),buji2_in(1,m),buji2(1,1)];
                [rout1p,d]=number_dist(rout,rout(1));
                if rout1p(1,23)<0.75&&rout1p(1,23)>0.63
                    rout_all2=[rout_all2;rout1p];
                end
            end
        end
    end
end
save 'Q1_2_3.mat' rout_all2

%% 第三部分
rout_all3=[];
for k=1:8%遍历出发 5 个点
    for m=1:8%遍历进入之后的 5 个点
        % k=1;m=1;
        for j=6:7
            Part_S=nchoosek(Part3(:,1),j);
            Part_S_=Part_S;
            p=0;
            for i=1:size(Part_S,1)
                %for i=1:1
                rout=[buji2(1,1),buji2_out(1,k),Part_S(i,:),endP_in(1,m),endP(1,1)];
                [rout1p,d]=number_dist(rout,rout(1));
                if rout1p(1,23)<0.75&&rout1p(1,23)>0.63
                    rout_all3=[rout_all3;rout1p];
                end
            end
        end
    end
end
save 'Q1_3_3.mat' rout_all3
%% 挑选第一部分的路线
load Q1_1_3.mat
% n=10;%设定的路线最大数量
final_result=[];
%选择最优的第一条路线
rout_all1=rout_all1;
rout_all1_=rout_all1;
final_result=[final_result;rout_all1(find(rout_all1(:,23)==min(rout_all1(:,23))),:)];
%将该路线从路线中剔除
rout_all1_(find(rout_all1(:,23)==min(rout_all1(:,23))),:)=[];
rout_all1=rout_all1_;

rout_all1=rout_all1_;
for j=1:size(rout_all1,1)
    rout1=rout_all1(j,:);
    repeat=0;
    for k=1:size(final_result,1)
        repeat = repeat+cal_repeat(final_result(k,:),rout1);%计算该线与已经有的选定路线的重复度
    end
    if repeat>1
        %将该路线从路线中剔除
        rout_all1_(find(rout_all1_(:,23)==rout_all1(j,23)),:)=[];
    end
end
save 'Q1_1_3_repeat01.mat' rout_all1_ rout_all1
%这样挑选出了所有与第一条路线不重合或者重合一次的线路，记为 rout_all2_,这就是接下来的解空间
```

```
%% 挑选第二部分的路线
load Q1_2_3.mat
% n=10;%设定的路线最大数量
final_result=[];
%选择最优的第一条路线
rout_all2_=rout_all2;
final_result=[final_result;rout_all2(find(rout_all2(:,23)==min(rout_all2(:,23))),:)];
%将该路线从路线中剔除
rout_all2_(find(rout_all2(:,23)==min(rout_all2(:,23))),:)=[];
rout_all2=rout_all2_;

rout_all2=rout_all2_;
for j=1:size(rout_all2,1)
    rout2=rout_all2(j,:);
    repeat=0;
    for k=1:size(final_result,1)
        repeat = repeat+cal_repeat(final_result(k,:),rout2);%计算该线与已经有的选定路线的重复度
    end
    if repeat>1
        %将该路线从路线中剔除
        rout_all2_(find(rout_all2_(:,23)==rout_all2(j,23)),:)=[];
    end
end
save 'Q1_2_3_repeat01.mat' rout_all2_ rout_all2
%%这样挑选出了所有与第一条路线不重合或者重合一次的线路，记为 rout_all2_,这就是接下来的解空间

%% 挑选第三部分的路线
load Q1_3_3.mat
% n=10;%设定的路线最大数量
final_result=[];
%选择最优的第一条路线
rout_all3=rout_all3;
rout_all3_=rout_all3;
final_result=[final_result;rout_all3(find(rout_all3(:,23)==min(rout_all3(:,23))),:)];
%将该路线从路线中剔除
rout_all3_(find(rout_all3(:,23)==min(rout_all3(:,23))),:)=[];
rout_all3=rout_all3_;

rout_all3=rout_all3_;
for j=1:size(rout_all3,1)
    rout1=rout_all3(j,:);
    repeat=0;
    for k=1:size(final_result,1)
        repeat = repeat+cal_repeat(final_result(k,:),rout1);%计算该线与已经有的选定路线的重复度
    end
    if repeat>1
        %将该路线从路线中剔除
        rout_all3_(find(rout_all3_(:,23)==rout_all3(j,23)),:)=[];
    end
end
save 'Q1_3_3_repeat01.mat' rout_all3_ rout_all3
%%这样挑选出了所有与第一条路线不重合或者重合一次的线路，记为 rout_all2_,这就是接下来的解空间
% 在挑选第一部分的解空间里
load Q1_1_3_repeat01.mat
final_result1=[];
先选择不重合的路线

final_result1=[final_result1;rout_all1(find(rout_all1(:,23)==min(rout_all1(:,23))),:)];
rout_all1=rout_all1_;
for i=1:10
    rout_all1=rout_all1_;
    rout_all1_=[];
    for j=1:size(rout_all1,1)
        rout1=rout_all1(j,:);
        repeat=0;
        for k=1:size(final_result1,1)
            repeat = repeat+cal_repeat(final_result1(k,:),rout1);%计算该线与已经有的选定路线的重复度
        end
        if repeat<1
            将该路线从路线中剔除
            rout_all1_=[rout_all1_;rout1];
        end
    end
    if size(rout_all1_,1)>0
        final_result1=[final_result1;rout_all1_(find(rout_all1_(:,23)==min(rout_all1_(:,23))),:)];
        rout_all1_(find(rout_all1_(:,23)==min(rout_all1_(:,23))),:)=[];
    end
end
for r=2
    load Q1_1_3_repeat01.mat
    for i=1:10
        rout_all1=rout_all1_;
        rout_all1_=[];
        for j=1:size(rout_all1,1)
            rout1=rout_all1(j,:);
```

```
                repeat=0;
                for k=1:size(final_result1,1)
                    repeat = repeat+cal_repeat(final_result1(k,:),rout1);%计算该线与已经有的选定路线的重复度
                end
                if repeat<r
                    将该路线从路线中剔除
                    rout_all1_=[rout_all1_;rout1];
                end
            end
            if size(rout_all1_,1)>0
                final_result1=[final_result1;rout_all1_(find(rout_all1_(:,23)==min(rout_all1_(:,23))),:)];
                rout_all1_(find(rout_all1_(:,23)==min(rout_all1_(:,23))),:)=[];
            end
        end
end
% 在挑选第 2 部分的解空间里
load Q1_2_3_repeat01.mat
final_result2=[];
先选择不重合的路线

final_result2=[final_result2;rout_all2(find(rout_all2(:,23)==min(rout_all2(:,23))),:)];
rout_all2=rout_all2_;
for i=1:10
    rout_all2=rout_all2_;
    rout_all2_=[];
    for j=1:size(rout_all2,1)
        rout1=rout_all2(j,:);
        repeat=0;
        for k=1:size(final_result2,1)
            repeat = repeat+cal_repeat(final_result2(k,:),rout1);%计算该线与已经有的选定路线的重复度
        end
        if repeat<1
            将该路线从路线中剔除
            rout_all2_=[rout_all2_;rout1];
        end

    end
    if size(rout_all2_,1)>0
        final_result2=[final_result2;rout_all2_(find(rout_all2_(:,23)==min(rout_all2_(:,23))),:)];
        rout_all2_(find(rout_all2_(:,23)==min(rout_all2_(:,23))),:)=[];
    end

end

for r=2
    load Q1_2_3_repeat01.mat
    for i=1:10
        rout_all2=rout_all2_;
        rout_all2_=[];
        for j=1:size(rout_all2,1)
            rout1=rout_all2(j,:);


            repeat=0;
            for k=1:size(final_result2,1)
                repeat = repeat+cal_repeat(final_result2(k,:),rout1);%计算该线与已经有的选定路线的重复度
            end
            if repeat<r
                将该路线从路线中剔除
                rout_all2_=[rout_all2_;rout1];
            end
        end
        if size(rout_all2_,1)>0
            final_result2=[final_result2;rout_all2_(find(rout_all2_(:,23)==min(rout_all2_(:,23))),:)];
            rout_all2_(find(rout_all2_(:,23)==min(rout_all2_(:,23))),:)=[];
        end
    end
end
% 在挑选第 3 部分的解空间里
load Q1_3_3_repeat01.mat
final_result3=[];
先选择不重合的路线

final_result3=[final_result3;rout_all3(find(rout_all3(:,23)==min(rout_all3(:,23))),:)];
rout_all3=rout_all3_;
for i=1:10
    rout_all3=rout_all3_;
    rout_all3_=[];
    for j=1:size(rout_all3,1)
        rout1=rout_all3(j,:);
        repeat=0;
        for k=1:size(final_result3,1)
            repeat = repeat+cal_repeat(final_result3(k,:),rout1);%计算该线与已经有的选定路线的重复度
        end
        if repeat<1
```

```
                将该路线从路线中剔除
                rout_all3_=[rout_all3_;rout1];
            end
        end
        if size(rout_all3_,1)>0
            final_result3=[final_result3;rout_all3_(find(rout_all3_(:,23)==min(rout_all3_(:,23))),:)];
            rout_all3_(find(rout_all3_(:,23)==min(rout_all3_(:,23))),:)=[];
        end

end

for r=2
    load Q1_3_3_repeat01.mat
    for i=1:10
        rout_all3=rout_all3_;
        rout_all3_=[];
        for j=1:size(rout_all3,1)
            rout1=rout_all3(j,:);


            repeat=0;
            for k=1:size(final_result3,1)
                repeat = repeat+cal_repeat(final_result3(k,:),rout1);%计算该线与已经有的选定路线的重复度
            end
            if repeat<r
                将该路线从路线中剔除
                rout_all3_=[rout_all3_;rout1];
            end
        end
        if size(rout_all3_,1)>0
            final_result3=[final_result3;rout_all3_(find(rout_all3_(:,23)==min(rout_all3_(:,23))),:)];
            rout_all3_(find(rout_all3_(:,23)==min(rout_all3_(:,23))),:)=[];
        end
    end
end
%  进一步扩充解空间
%  允许重复更多的解
load Q1_2_3.mat%导入没有删选过的解空间
load final_01.mat%导入目前的解路线
% n=10;%设定的路线最大数量


rout_all2_=rout_all2;

% for j=1:size(rout_all2,1)
%         rout2=rout_all2(j,:);
%         repeat=0;
%         for k=1:size(final_result2,1)
%             repeat = repeat+cal_repeat(final_result2(k,:),rout2);%计算该线与已经有的选定路线的重复度
%         end
%         if repeat>10
%             %将该路线从路线中剔除
%             rout_all2_(find(rout_all2_(:,23)==rout_all2(j,23)),:)=[];
%         end
% end

for r=2:10
    for i=1:10
        rout_all2=rout_all2_;
        rout_all2_=[];
        for j=1:size(rout_all2,1)
            rout1=rout_all2(j,:);


            repeat=0;
            for k=1:size(final_result2,1)
                repeat = repeat+cal_repeat(final_result2(k,:),rout1);%计算该线与已经有的选定路线的重复度
            end
            if repeat<r
                %将该路线从路线中剔除
                rout_all2_=[rout_all2_;rout1];
            end
        end
        if size(rout_all2_,1)>0
            final_result2=[final_result2;rout_all2_(find(rout_all2_(:,23)==min(rout_all2_(:,23))),:)];
            rout_all2_(find(rout_all2_(:,23)==min(rout_all2_(:,23))),:)=[];
        end
    end
end

%%这样挑选出了所有与第一条路线不重合或者重合一次的线路，记为 rout_all2_.这就是接下来的解空间
```