基于风险与博弈思想的沙漠掘金策略

摘要

本文通过研究穿越沙漠游戏，分析规则和游戏特性，从已知未来天气和未知未来天气两种情况出发建立数学模型，针对分别不同模型设计相对应的求解算法。接着从**单玩家模型**加入新的指标进行更新，延伸到**多玩家模型**，并在此过程中完成了附件中对第一关到第五关的具体讨论、模型求解、模型检验等，为广大玩家提供了**低风险下可行可靠且收益高的最优策略**。

针对问题一：在点出单玩家地图无人争夺最佳路线的特点后，为减少寻优空间将**地图抽象化**，且利用 **Floyd 算法**求出各个关键点之间的最短路径。接着围绕两个方向分类讨论，进而建立以单人玩家以获取最多资金为目的的**线性规划模型**。为求解模型，在讨论模型中决策变量的特性后将其转化为二进制编码，把问题转化为**在已知未来天气下求固定期限内决策组合的最优化问题**，以此为基础建立求解对应模型的**遗传算法**，并完成大量细节修改。在经过算法计算后得出**第一关和第二关在最佳策略下的最优收益**分别为 **10460 元**和 **11170 元**。

针对问题二：在分析和讨论后，发现问题实质为**未知天气情况下沙漠探险游戏背景的风险决策问题**。本文在第一阶段分别建立时间风险指标、物质风险指标、收益风险指标，并将其综合成**生存风险函数**。在到达矿山后的第二阶段继续建立**挖矿决策函数、返回决策函数、补给决策函数**。综合两个阶段得出了**风险决策模型**，并据此设计了模拟仿真的流程图以便算法求解。

针对问题三：对于单人玩家变化到多人玩家后可能的变化进行阐述并指出了一些注意的要点，进而引出用**博弈论**来决策玩家自己的行动方案的思想。然后从玩家的角度出发，量化玩家之间的**相互干扰度**，将问题一的模型进行了更新，建立在**博弈论约束下的最佳策略模型**。接着新问题制定求解策略，建立**初始路线集合**来减少寻优空间，利用**格雷码和二进制码相结合的编码方式**设计基于问题一的**改进遗传算法**，将 n 名玩家到达终点后的平均资金作为衡量整体的新标准。最后对于**第五关两种情景**求解出最优策略，**此时整体最优的平均资金为 9495 元**。

针对问题四：由于多人游戏的规则限制，玩家之间的动态决策影响至深。建立所有玩家的**收益矩阵**后，根据**混合策略 Nash 均衡理论**将每名玩家的策略向量构成**混合策略空间**，并以此得到混合政策下的所有玩家的**效用函数**。将效用函数以及约束条件与静态模型结合后成功建立**以最大收益为目标的动态博弈模型**。最后采用**蒙特卡洛模拟法**对沙漠探险过程进行仿真，并在运行多次后得出**第六关下蒙特卡洛法下历史最优策略**以及各个玩家的最终资金，**此时三名玩家的结束游戏时整体平均资金为 10639.17。**

综合来讲，本文研究了沙漠探险游戏下各种情况的最佳策略，并对不同的模型进行相关**灵敏性分析**。模型的优势在于求解方法全面可靠，覆盖情况多种多样，每个模型环环相套，为广大喜爱沙漠探险游戏的玩家**提供了不同风险承受能力下尽可能取得收益的策略服务**。

关键词：最佳策略模型　遗传算法　风险决策模型　动态博弈

一、问题重述

1.1 问题背景

设想一种沙漠穿越游戏，其流程大体如下：在遵守八条游戏规则的前提下，玩家开局有一张全局沙漠地图，凭着初始资金购买一定的水和食物。从起点出发，在沙漠中行走，到终点结束。路途中会有三种不同天气，玩家也可以在村庄、矿山补充资源或资金。目标是保证能在规定时间内到达终点，获得尽可能多的资金。

1.2 问题要求

因此本文依据游戏的不同设定，建立数学模型，解决以下问题：

问题 1：假如仅有一名玩家，并事先知道在整个游戏时间段内的每日天气状况，建立数学模型，试着给出在一般情况下玩家的最优策略。

问题 2：假如仅有一名玩家，玩家依据仅有的当日天气状况对当天的行动方案进行决策，建立数学模型，试着给出在一般情况下玩家的最优策略。

问题 3：假如现有 n 名玩家，游戏规则有一定的变化，据此条件解决以下问题：

（1）假如所有玩家事先知道在整个游戏时间段内的每日天气状况，在游戏开始时需确定行动方案且在游戏过程中不能更改。建立数学模型，试着给出在一般情况下玩家应采取的策略。

（2）假如所有玩家仅知道当日的天气状况，从第1天起，各位玩家在当日行动结束后均知道其他玩家剩余的资源数量以及当日的行动方案，从而确定各自在第二天的行动方案。建立数学模型，试着给出在一般情况下玩家应采取的策略。

二、问题分析

作为游戏玩家，希望在游戏中可以在保障到达终点的情况下尽可能多的挣到钱，因此建立适当的数学模型可以为玩家提供更好的游戏建议与游戏体验。

2.1 问题一分析

问题一要求我们求解在玩家已知所有天气情况下的最优策略。在本游戏中不仅有天数限制，而且还有生存限制，因此在建立模型时应充分考虑这两点。由于天气状况已知，则玩家可以避免到达不了以及生活必需品不够等危险情况。在保证安全前提下，建立路径规划的规划方程组。由于直接计算运算量太大，因此求解本题结果可使用相关智能算法进行求解。

2.2 问题二分析

问题二要求我们在仅知道当天天气情况下，给出玩家的策略。考虑到未来不确定因素较大，因此建议玩家在游戏刚开始尽可能多的购买生活必需品。对于一般玩家可以选择直接去终点或者选择先挖矿后去终点。如果选择了先挖矿后去终点则可以将整个游戏分为两个阶段：起点到矿山，矿山到终点。在起到到矿山过程中玩家只需要决策当天走还是停，因此本阶段可以设立风险度指标针对不同玩家的心理阈值进行决策。在矿山到终点过程中，玩家需要考虑是否挖矿、是否补给、是否去终点等问题，因此可以建立多个决策变量来决策玩家行动。

2.3 问题三分析

问题三要求我们多人游戏中在已知天气情况下的游戏策略。由于多人游戏中改变了游戏规则，则玩家应该尽量避免与其他玩家相遇，从而减少损失。因为天气已知，则可以通过模型一求解出游戏关卡的最优策略，但是为了尽量避免与其他玩家接触，因此本题可以引入静态博弈论模型。在保证所有玩家最终受益均衡的状态下，通过博弈论选出不同的路线，从而最大程度上减少了多个玩家相遇的情况。

2.4 问题四分析

问题四要求我们在只知道当前天气状态下的多人游戏策略。由于未来天气未知，因此玩家可以根据模型二进行每一步的决策，但为了避免玩家相遇情况，本题可以引入动态博弈论进行求解。首先每名玩家都可以根据当前情况去决定第二天的行动，但考虑到存在其他玩家，因此在每一个做决定的过程中都引入动态博弈环节，从而保证了所有玩家收益均衡。

三、模型假设与约定

1）假设在负重上限之下，负重的差异不影响资源消耗的差异；

2）假设在有多名玩家的游戏设定上，所有玩家都会前往矿山；

3）假设在天气模拟时不出现几乎所有天数天气相同的反常现象；

4）假设在多人游戏中不存在玩家恶意破坏游戏规则的现象；

四、符号说明及名词定义

符号	符号说明	单位
$M^{(K)}$	游戏结束时的资金总量	元
n_0	水的箱数	箱
m_0	食物的箱数	箱
w_i	天气向量	

$C(w_i)$	天气消耗的向量	
E_i	玩家收益	元
B_k	消耗的资金	元
D_S	时间风险指标	
F_S	物质风险指标	
E_s	收益风险指标	
S	生存风险函数	
ν_i	挖矿决策函数	
v_i	返回决策函数	
V_i	补给决策函数	
$p_i(v_i)$	一名玩家的策略	
M	收益矩阵	
p_i	玩家选取i策略的概率	
S_i	混合策略空间	
$u_j(p_j)$	效用函数	

五、问题一模型建立与求解

5.1 模型准备

问题一要求单个玩家在知道 30 天每一天的天气下求出最优策略，那么首先应该了解到单人玩家进行游戏有个非常重要的特征：无人争抢最佳路线。那么在研究问题和建立模型可以先将地图抽象化成几个主要点的链接地图，大大减少求解时的寻路空间。如下图以第一关为例，将地图抽象化：

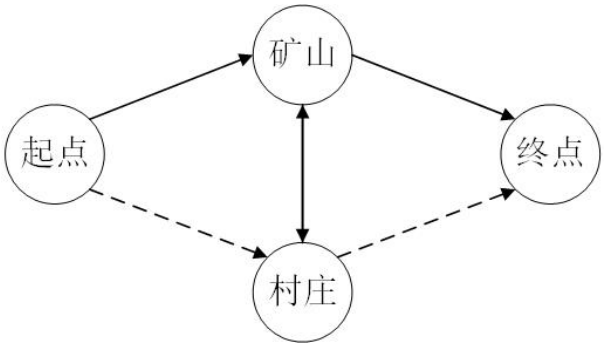

图 5.1：关卡抽象化地图

抽象化后几个关键点，起点、村庄、矿山和终点之间的最短路径可以利用 Floyd 算法代入地图的邻接矩阵求出。这样就成功针对问题本质把地图冒险游戏暂且变成简单无向图上的策略问题。

在地图中，玩家的选择有很多。玩家可能会从起点直奔矿山；也可能直奔村庄；更有甚者可以直接前往终点。但不管玩家的选择多么多样化，根据目的可以大致上分为两种：

其一为前往矿山或者村庄然后开始“挖矿-补给”环节。这种行为的目的在于保证可以安全回到终点的前提下尽量去获取更多来自挖矿带来的收益，但也有可能会出现中途补给不到或者食物短缺的情况，遇到这样的情况时就应该提前选择补给或者撤离。所以很有必要针对这个方向建立模型。

其二为直接奔往终点。当终点离起点较近，矿山却离起点很远的时候，直接前往终点反而是更好的选择。为了考虑周全，在特定情况下有时候也有必要讨论。

综上，在抽象化地图后，接下来主要围绕第一种方向建立模型。

5.2 模型建立

保证在规定时间内到达终点的前提下，建立获得尽可能多资金的单目标规划模型如下：

在沙漠探险的过程中，如果以挖矿为目标进行决策，那么必然会经历前往矿山、在矿山作业/在村庄补给途中、返回终点三个阶段。如果量化每个阶段的决策，可以采用 0-1 变量[1]的形式：

决策变量：

$$X_i=\begin{cases}0 & \text{停留}\\ 1 & \text{行走}\end{cases}\ i\in(0,k_i),\quad Y_i=\begin{cases}0 & \text{停留}\\ 1 & \text{挖矿}\end{cases}\ i\in(k_1,k_2),\quad Z_i=\begin{cases}0 & \text{停留}\\ 1 & \text{行走}\end{cases}\ i\in(k_2,k_3)$$

其中 $0\sim k_1$天前往矿山或村庄，$k_1\sim k_2$天进行挖矿或补给资源，$k_2\sim k_3$天返回终点。

目标函数：

$$\max\ Z\ =\ \max\ M^{(K)} \tag{1}$$

其中$M^{(K)}$为游戏结束时的资金总量。

约束条件：

（1）水和食物重量不能超过负重上限，即

$$3n_0+2m_0\leqslant 1200 \qquad n_0>0,m_0>0 \tag{2}$$

其中n_0为水的箱数，m_0为食物的箱数。

（2）水和食物的初始消耗资金不能超过初始资金，即

$$5n_0+10m_0\leqslant 10000 \qquad n_0>0,m_0>0 \tag{3}$$

（3）为研究决策向量与天气的关系，设计与天气向量高度相匹配，仅有沙暴天为 0 的向量T（w），且$\mathrm{T(w)}=\{1,1,\cdots,0,\cdots\}$。对于 $0\sim k_1$天前往矿山或村庄，$k_1\sim k_2$天进行挖矿或补给资源，$k_2\sim k_3$天返回终点。考虑到天气因素，数学表达式有：

$$\sum_{i=0}^{k_1} X_i{}^{*}T_i(w)\geqslant S_{起-矿} \tag{4}$$

$$\sum_{i=0}^{k_1} X_i{}^{*}T_i(w)\geqslant S_{起-村} \tag{5}$$

$$S_{村-终}\leqslant \sum_{i=k_2}^{k_3} Z_i{}^{*}T_i(w)\leqslant S_{矿-终} \tag{6}$$

其中w_i为天气向量，C（$\mathrm{w_i}$）为天气消耗的向量。

（4）玩家在游戏过程中的收益，即

$$E_i=E_{i-1}+y_i{}^{*}1000 \tag{7}$$

（5）玩家在村庄补充资源消耗的资金，即

$$B_k=B_{k-1}+6n_k+4m_k \tag{8}$$

（6）资金总量$M^{(K)}$，即

$$M^{(K)}=10000-5n_0-10m_0-B_k+E_i \tag{9}$$

综上所述，得出的最佳策略模型为：

$$\max\ Z\ =\ \max\ M^{(K)}$$

$$
s.t.\begin{cases}
n_0>0,m_0>0\\
3n_0+2m_0\leqslant 1200\\
5n_0+10m_0\leqslant 10000\\
T(w_i)=\{1,1,\cdots,0,\cdots\}\\
\sum_{i=0}^{k_1}X_i{*}T(w_i)\geqslant S_{起-矿}\\
\sum_{i=0}^{k_1}X_i{*}T(w_i)\geqslant S_{起-村}\\
S_{村-终}\leqslant\sum_{i=k_2}^{k_3}Z_i{*}T(w_i)\leqslant S_{矿-终}\\
3n_i+2m_i\leqslant 1200\\
E_i=E_{i-1}+y_i{*}1000\\
B_k=B_{k-1}+6n_k+4m_k\\
M^{(K)}=10000-5n_0-10m_0-B_k+E_i
\end{cases}
\tag{10}
$$

5.3 模型求解

5.3.1 遗传算法模型

为了求解模型，针对问题的特性我们决定采用了遗传算法[2]。遗传算法简称GA，是将大自然中的淘汰机制和自身的优化过程相结合一种高效的全局寻优搜索算法。尤其是在本题中可能会有走不到终点的情况出现，此时具有“优胜劣汰”特性的遗传算法就非常契合本题的解题思路。并且在使用编码解决问题这点上和30 天进行决策有相似之处，因此可以利用二进制编码去代替决策向量。

5.3.2 针对问题和模型的改进遗传算法的细节

遗传算法包括三个基本操作：选择、交叉和变异。我们分开从这些细节进行修改。

1）产生初始种群

种群的每个个体的编码代表的是 30 天内玩家可能会使用的决策向量，根据目标模型可知虽然整个过程分为三段，但决策变量一直是 0-1 变量。因此在产生种群上我们直接采用长度为 30 的二进制编码即可。并且因为以赚钱和不在路上浪费过多时间为目标，我们将 1 的生成概率设为 0.8,而 0 的生成概率为 0.2。

2）适应度函数

根据模型和种群内容，最优目标应该是成功完成游戏并且剩余金钱最高的个体。因此我们修改为以输出的剩余金钱变量为主要的衡量对象。并且保存并输出最好情况的个体编码。

3）选择

针对本题的特点来设置，选择将使适应度较大(即结束游戏后金钱最多)个体有较大的存在机会，反之个体继续存在的机会较小，在选择上采用赌轮选择机制。

4）交叉

根据个体编码的“1 多 0 少”的特点，使用一次交叉，随机选取两个点，将两个父个体中两个交叉点之间的中间段直接交叉，并将其余部分组合。这样的好处是避免有规律的交叉可能会造成少数地位的“0”规律出现某几个位置，进而导致容易收敛到局部解。

5）变异

由于基于二进值编码的变异操作不能够由简单的变量的翻转来实现，可以利用“1 多 0 少”，随机的在这个编码序列选取两个位置，然后交换他们的位置。这样就实现了个体编码的变异。

5.3.3 算法流程图

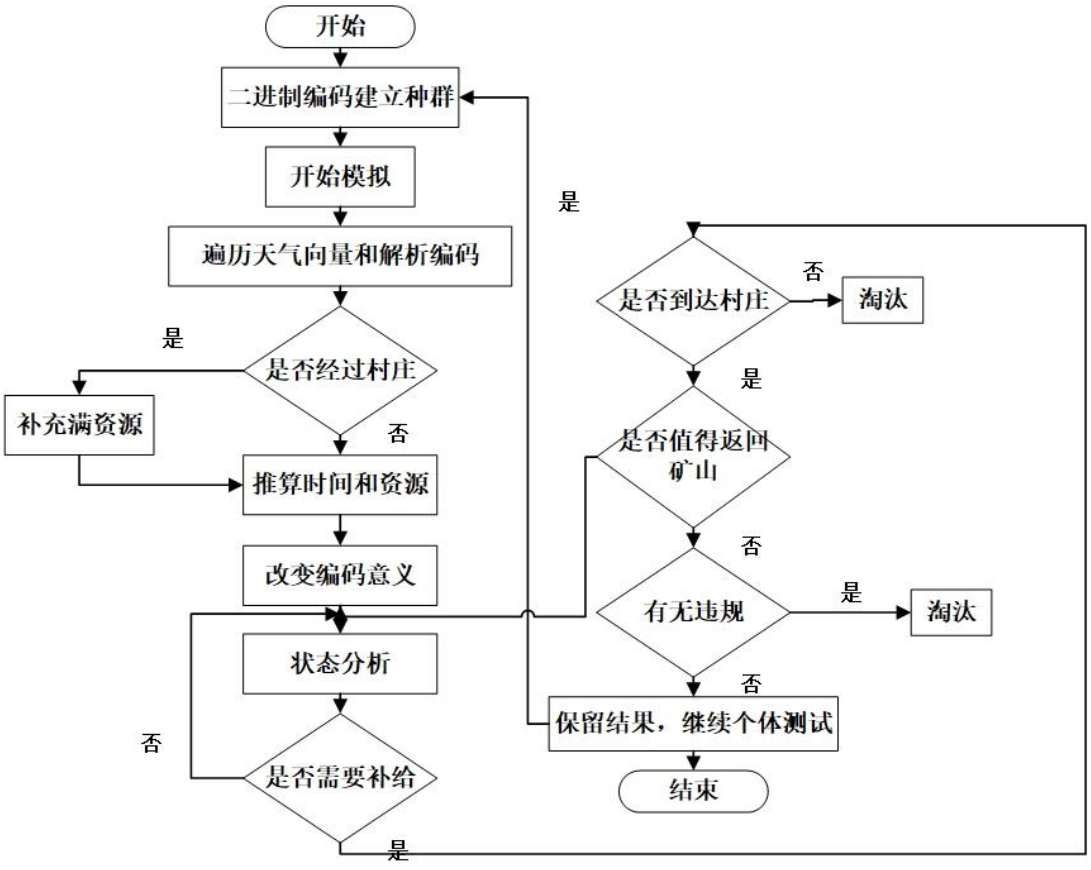

图 5.3.3：问题一算法流程图

5.3.4 求解结果

通过算法去求解模型，以迭代次数 20，种群数量 50，在外层通过不同的初始水量和食物量的组合，求得的第一关及第二关最优策略部分如下表：

表 5.3.4-1：第一关最优策略（部分）

日期	所在区域	剩余资金数	剩余水量	剩余食物量
0	1	5790	178	332
1	25	5790	162	320
2	24	5790	146	308
3	23	5790	136	294
…	…	…	…	…
24	27	10460	0	0

表 5.3.4-2：第二关最优策略（部分）

日期	所在区域	剩余资金数	剩余水量	剩余食物量
0	1	5640	164	354
1	2	5640	148	342
2	3	5640	132	330
3	4	5640	122	316
…	…	…	…	…
28	64	11170	0	0

5.3.5 模型检验

由于该模型的目标函数即为所剩余额最大化，因此在涉及到影响模型目标函数的因素中应该注意研究该因素对模型的影响程度。生活必需品的价格对所剩余额有一定的影响力，为了确保模型的稳健性，需要对模型的稳健性进行检验。

表 5.3.5：模型检验

价格改变量	水的费用	变化率	食物的费用	变化率
0	10250	0%	10250	0
0.1	10197.8	0.51%	10198.6	0.50%
0.2	10145.6	1.02%	10147.2	1.00%
0.3	10093.4	1.53%	10098.8	1.48%
0.4	10041.2	2.04%	10044.4	2.01%

通过上表可观察到当水费与食物的费用变动时，只对水或食物最终的费用产生不到 3%的影响，可以判断出本模型可以很好的适应生活必需品价格的变化，从而保证了模型的稳健性。

六、问题二模型建立与求解

6.1 模型准备

在第三关和第四关中不考虑天气极端的情况下，玩家顺利从起点到终点的天数远小于整个游戏时间段的天数，因此矿山是个必经之地。在这个动态问题中，宏观上可以将整个游戏流程大体分为两个时间段[3]：第一阶段是从起点到矿山的时间段，第二阶段是到了矿山之后的时间段。不仅如此，两个阶段玩家考虑的问题也不一样。玩家在第一阶段只要考虑是否行走等问题，而在第二阶段玩家要考虑是否要挖矿，是否要补给以及是否要去终点等问题。

6.2 模型建立

6.2.1 第一阶段

第一阶段是从起点到矿山的时间段，在这个时间段玩家要考虑时间风险因素，物质风险因素以及收益风险因素。由此我们建立时间风险指标D_S、物质风险指标F_S以及收益风险指标E_S三个风险指标，助于玩家更好地决策自己的行动方案。

$$\textbf{时间风险指标：}\ D_S = \frac{D_L^{(i)}}{D_F^{(i)}} \tag{11}$$

其中$D_L^{(i)} = D - i$代表玩家剩余的天数，$D_F^{(i)}$代表在当前位置玩家到达终点的最快天数。时间风险指标D_S越接近数值 1，说明玩家剩余时间的可操作性变小，代表玩家游戏失败的风险性越高。

$$\textbf{物质风险指标：}\ F_S = \frac{F_L^{(i)}}{F_Q^{(i)}} \tag{12}$$

其中$F_L^{(i)} = \min\{nL^{(i)}, mL^{(i)}\}$代表玩家剩余的生活必需品，$F_Q^{(i)} = \sum_{i=0}^{D_F^{(i)}} C(w_i)$代表玩家期望的生活必需品量。物质风险指标$F_S$越接近数值 1，说明玩家剩余的生活必需品越不足以支撑到达终点，代表玩家游戏失败的风险性越高。

$$\textbf{收益风险指标：}\ E_S = \frac{E_Q}{E_W} \tag{13}$$

其中E_Q代表玩家的期望收益，$E_W = C_{nw_i} * P_w + C_{mw_i} * P_f$代表玩家的等待收益。

综合三个风险指标，最终得到生存风险函数：$S=\alpha D_S+\beta F_S+\gamma E_S$。由此玩家可以借助生存风险函数得到生存风险值，从而根据玩家的心理风险阈值$\mu^*S_{\max}$对自己的行动方案进行决策。其中，决策函数为：

$$D_i=\begin{cases}0,\ S>\mu^*S_{\max}\\1,\ S\leqslant\mu^*S_{\max}\end{cases} \tag{14}$$

6.2.2 第二阶段

第二阶段是玩家到了矿山之后的时间段，玩家在这个时间段内要考虑是否要挖矿，是否要补给以及是否要去终点等问题，由此我们需要建立三个决策函数对相应状态进行决策，从而求解得一般情况下玩家的最佳策略。

$$\textbf{挖矿决策函数：}\nu_i=\begin{cases}0,\ \dfrac{E_Q}{(C_{nw_i}{}^*P_w+C_{mw_i}{}^*P_f)^*3}<1\\1,\ \dfrac{E_Q}{(C_{nw_i}{}^*P_w+C_{mw_i}{}^*P_f)^*3}>1\end{cases} \tag{15}$$

其中当玩家一天的期待收益与挖矿的消耗资源比值小于 1 则玩家应该放弃挖矿；当比值大于 1，玩家应该继续挖矿，从而扩大资源优势。

$$\textbf{返回决策函数：}\upsilon_i=\begin{cases}0,\ \dfrac{D_d}{D_F^{(i)}}>1\\1,\ \dfrac{D_d}{D_F^{(i)}}<1\end{cases} \tag{16}$$

其中D_d代表玩家从矿山到村庄再返回矿山所需要的天数。当折返所需要的天数与玩家剩余的天数比值大于 1 则玩家应该考虑返回终点；当比值小于 1，说明玩家的时间充盈，则玩家需要考虑是否可以去挖矿或者去村庄补给。

$$\textbf{补给决策函数：}V_{i(1)}=\begin{cases}0,\ \dfrac{F_L^{(i)}}{F_C}>1\\1,\ \dfrac{F_L^{(i)}}{F_C}<1\end{cases} \tag{17}$$

$$V_{i(2)}=\begin{cases}0,\ \dfrac{C_B+C_R}{(D_F^{(i)}-D_L^{(i)}-1)^*(E_Q-C_{nw_i}{}^*P_w{}^*3-C_{mw_i}{}^*P_f{}^*3)}>1\\1,\ \dfrac{C_B+C_R}{(D_F^{(i)}-D_L^{(i)}-1)^*(E_Q-C_{nw_i}{}^*P_w{}^*3-C_{mw_i}{}^*P_f{}^*3)}<1\end{cases} \tag{18}$$

其中F_C代表玩家从矿山到村庄所需要的补给，C_B代表玩家在村庄补给所需要的资金，C_R代表玩家在矿山与村庄折返所需要补给消耗的资金。当玩家剩余补给与需要补给的比值大于 1 则不需要补给，当小于 1 需要补给且玩家需要考虑

补给得值不值问题。当玩家在村庄购买补给消耗的资金与在折返路上消耗的资金之和与玩家在剩余天数所期望收益的比值大于 1，则玩家不应该折返矿山而应返回终点。

综合第一和第二阶段，最终可以得到资金总量M的目标函数：

$$M = M_{初} - 5n - 10m - \sum_{k=0}^{v} V_{i(2)} * B_k + \sum_{k=0}^{\nu} \nu_i * E_i \tag{19}$$

其中v代表玩家补给的次数，ν代表玩家挖矿的天数。

6.3 模型求解

6.3.1 求解策略

针对第三关，通过程序随机生成十天的天气状况，然后将天气带入模型中进行解算。由于第三关没有村庄补给因此在计算第三关时应修改涉及到村庄的表达式，同时在没有补给的状态下也会给玩家一定心理压力。

表 6.3-1：随机决策方案

日期	1	2	3	4	5	6	7	8	9	10
天气	晴朗	晴朗	高温	高温	高温	晴朗	高温	高温	晴朗	高温
决策	行走	行走	停步	停步	行走	终点				

注：假设取 30%的心理风险阈值。

在玩家走到第三天的时候遇到高温天气，此时根据模型将匹配到游戏的第二阶段并判断今天是否继续行走，因为第三天是第一天到达金矿，所以玩家只能选择等待。在第四天时仍然为高温，于是玩家通过ν_i决策函数求得$\frac{E_Q}{(C_{nw_i} * P_w + C_{mw_i} * P_f) * 3} < 1$，于是玩家选择不挖。在返回决策函数$\upsilon_i$和$\mathrm{V_i}$没有触发时，玩家则一直根据$\nu_i$决策函数进行判断，一旦当$\upsilon_i$处罚，玩家则立即前往终点。

针对第四关，相比于第三关，第四关设有村庄，可以进行补给，同时游戏天数也相对较长。通过蒙特卡洛模拟随机生成晴朗与高温天气，并以 10%的概率随机生成沙暴天气。不同于第三关，第四关的矿山附近有村庄，方便玩家进行补给。当补给决策函数触发时，玩家便可提前计算与预测去村庄的时间以及来回消耗从而进行下一步的决策。

6.3.2 求解算法流程图

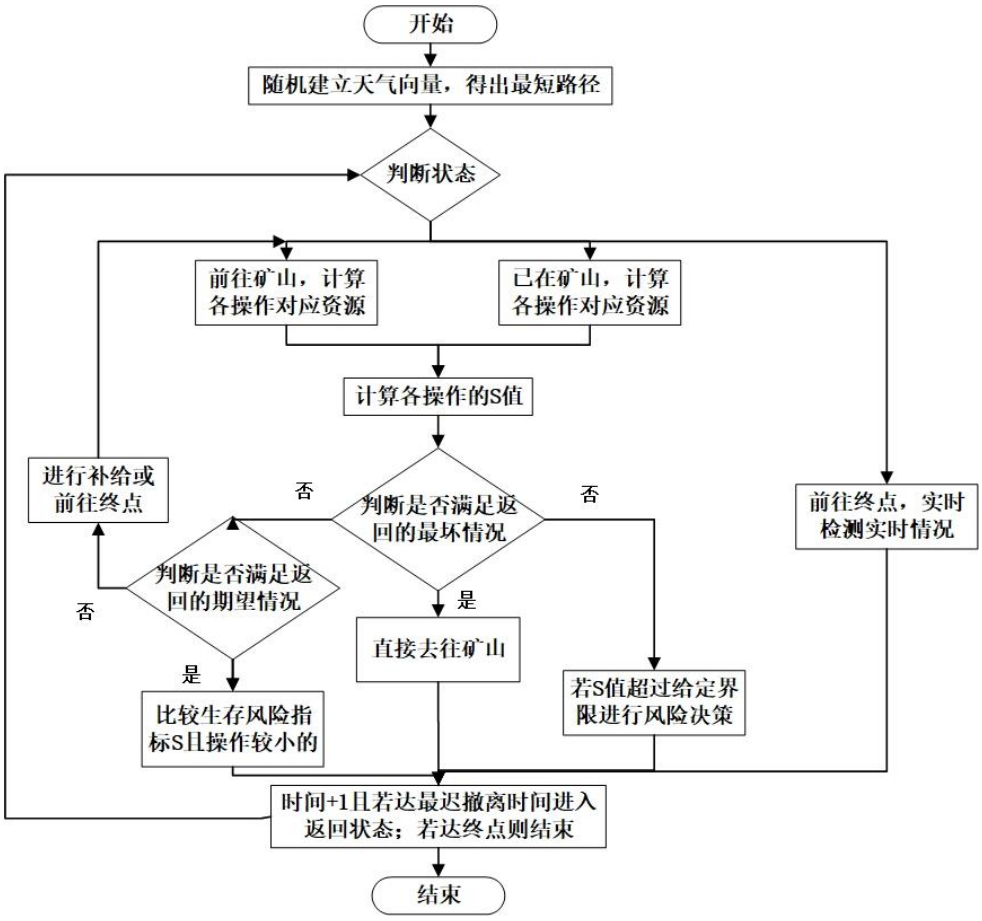

图 6.3.2：问题二算法流程图

6.3.3 求解结果

综上所述，在以晴天为 60%,高温为 40%的概率下，通过大量的蒙特卡洛模拟可得出该模型的具体特征：

表 6.3.3：模型的具体特征

关卡	心理风险决策阈值	平均金额	失败率
第三关	0-0.3	8950	2.3%
	0.3-0.6	9344	3.5%
	0.6-1	9526	5.1%
第四关	0-0.3	10342	2.6%
	0.3-0.6	11431	3.8%
	0.6-1	12046	5.7%

从结果上来看，收益往往伴随着风险。在该模型中，心理风险决策阈值的选取越高，可能获得的收益越高，但失败率也会随之升高；心理风险阈值的选取越低，可能获得的收益越低，失败率也会越低，也会越安全。这为不同性格的玩家、不同风险承受能力的玩家提供了多样化选择。

6.3.4 模型检验

由于模型二作用于未来天气未知的情况下，因此为了检验本模型的稳健性，以第四关为例，通过修改不同天气比例来检测平均金额与失败率的变化情况。通过采用蒙特卡洛模拟法在经过大量的仿真试验后得到以下结果：

表 6.3.4：模型检验

天气比例（晴/高）	平均金额	金额变化率	失败率	失败率变化率
70/30	11823	0%	2.3%	0
60/40	11687	1.15%	2.5%	8%
50/50	11376	2.66%	2.6%	3.85%
40/60	11249	1.12%	2.8%	8%

注：不考虑沙暴情况

通过表 6.3.4 可以发现，在模拟天气比例变化时，该模型得出的平均金额、失败率的变化率均小于 10%，表明在未来天气未知的情况下，通过本模型的选择策略可以保证玩家收入的稳定性与生存的安全性，表明该模型的稳健性较好。

七、问题三模型建立与求解

7.1 模型准备

在第五关中事先知道在整个游戏时段内每天天气状况，因此可以在问题一建立的模型基础之上加以改进。现有多名玩家，根据游戏的不同设定，为了保证每名玩家的平均资金总量最大化，每名玩家要减少与其他玩家的接触时间。由此，每名玩家要采用博弈论来决策自己的行动方案，从而使个人的资金总量合理化。

7.2 模型建立

记其中一名玩家的策略为$p_i(v_i)$，即当这名玩家认为其他玩家造成的干扰度为v_i，则这名玩家采取策略$p_i(v_i)$。对于任意给定的$v_i \epsilon [0,1]$，玩家的策略$p_i(v_i)$应该使个人的资金总量最大化，所以$p_i(v_i)$满足：

$$\max_{p_i} \left\{ \frac{p_i + E[p_j(v_j)|p_j(v_j)] \geq p_i}{n} \right\} P\{p_j(v_j) \geq p_i\} \qquad (20)$$

其中，$i,j=1,\cdots,n\ i\neq j$，$E[\ ]$表示条件$p_j(v_j)\geqslant p_i$下$p_j(v_j)$的条件期望，$p\{\ \}$表示事件的概率。

假设每名玩家的相互干扰度分别是资金总量对每名玩家的线性函数，最终可以表示为：

$$p_i(v_i)=M_{初}-C_B^{(i)}+E_Q^{(i)}+\frac{nL^{(i)}*p_w}{2}+\frac{mL^{(i)}*p_f}{2} \tag{21}$$

在 Nash 均衡状态下，所有策略组合$(p_i(v_i),p_j(v_j))$满足公式（max），可以使多名玩家的资金总量达到一个均衡的状态。因此可以将问题一的模型改进为：

$$\max\ Z\ =\ \max\ M^{(K)}$$

$$s.t.\begin{cases} n_0>0,m_0>0 \\ 3n_0+2m_0\leqslant 1200 \\ 5n_0+10m_0\leqslant 10000 \\ T(w_i)=\{1,1,\cdots,0,\cdots\} \\ \sum_{i=0}^{k_1}X_iT(w_i)\geqslant S_{起-矿} \\ \sum_{i=0}^{k_1}X_iT(w_i)\geqslant S_{起-村} \\ S_{村-终}\leqslant\sum_{i=k_2}^{k_3}Z_iT(w_i)\leqslant S_{矿-终} \\ 3n_i+2m_i\leqslant 1200 \\ E_i=E_{i-1}+y_i*1000 \\ B_k=B_{k-1}+6n_k+4m_k \\ M=10000-5n_0-10m_0-B_k+E_i \\ \max\limits_{p_i}\left\{\dfrac{p_i+E[p_j(v_j)|p_j(v_j)]\geqslant p_i}{n}\right\}P\{p_j(v_j)\geqslant p_i\} \\ p_i(v_i)=M_{初}-C_B^{(i)}+E_Q^{(i)}+\dfrac{nL^{(i)}*p_w}{2}+\dfrac{mL^{(i)}*p_f}{2} \end{cases} \tag{22}$$

7.3 模型求解——改进遗传算法

7.3.1 求解策略分析

该模型的优化求解背景仍然与问题一相似，知道未来的天气，并且玩家人数增加到 n 人，游戏日期也缩短到 10 天。虽然当不同的玩家做相同的某些事的时候收益和损益会彼此相互影响，但知道未来天气即确定了天气发生的概率，该问

题的本质上仍然是 n 名玩家的决策向量组合之间所产生的综合效益的优化问题。因此可以考虑改进问题一中的遗传算法，使其可以适应新模型的求解[4]。

7.3.2 针对问题的模型改进

具体改进主要体现的三个方面：

第一个方面是在种群建立之前，通过对地图的预处理，选取从起点到矿山的多条路线建立初始路线集合。因为问题中当玩家从一个区域一起向另一个区域前进的时候消耗会变大，因此有必要整理初始路线以方便初始路线的建立和筛选。在具体实现的时候，模拟的玩家先根据自身在初始路线集合中选择一条，这样可以大大减少寻优空间大小。

第二个方面是改进编码，使得可以在一段编码中可以同时体现未来的决策指令向量和初始路线的选择。所以我采用了格雷码和二进制编码相结合的方式，同时也发挥了格雷码的优势，即：交叉变异的截断点如果在格雷码范围内，之后个体的结构和选择也不会因此发生剧烈变化。编码长度调整为 13 位，格雷码为前三位。

第三个方面为适应度值的计算。将 n 位玩家在一个地图到达终点后的平均资金作为新的标准。因为不是每一个玩家场场都能成为“领头羊”，所以将平均资金作为衡量整体的标准很合理。

7.3.3 算法流程图

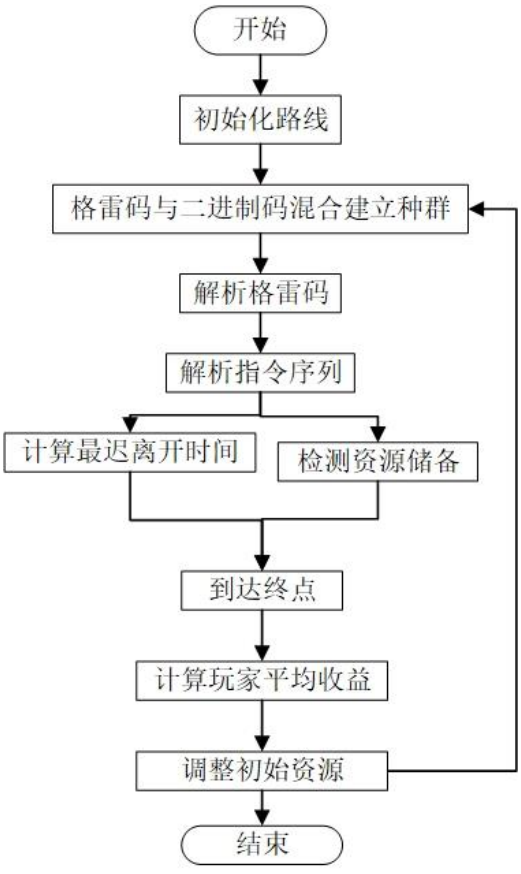

图 7.3.3：算法流程图

7.4 第五关求解结果及模型检验

7.4.1 求解结果

针对第五关，我们通过两种情景，以迭代次数 30，种群数量 40 求解出两种情境下的解如下：

1）以前往矿山赚钱为目的

表 7.4.1-1：以赚钱为目的的线路表

玩家	路线	余额
玩家一	1->4->4->3->9->9->9(挖)->11->13	8975
玩家二	1->2->3->9->9(挖)->9(挖)->11->	9200

2）以直接前往终点为目的

表 7.4.1-2：以前往终点为目的的线路表

玩家	路线	余额
玩家一	1->5->5->6->13	9565
玩家二	1->4->4->7->12-13	9425

从表中可以发现，前往终点为目的的线路比前往矿山赚钱为目的的线路更短而剩余的资金总量也更多，因此直接前往终点为目的去求解第五关，比前往矿山更好。

7.4.2 模型检验

在问题三中引入了多人游戏模式，因此为了保证玩家个人所剩余额较多，如何避免与其他玩家接触是个非常重要的问题。通过以上表格的结果可以看出两位在关卡三中的总余额均值达到了 9495 元，这个结果与单人通过关卡三所剩的余额非常接近，表明该模型在多人游戏规划中效果较好。

八、问题四模型建立与求解

8.1 模型准备

由于多人游戏规则的限制，在未来天气未知的情况下需要尽量避免与其他玩家接触，因此不能单一基于模型二进行求解，还需考虑多名玩家决策情况。由于玩家每天均需要确定下一天的行动方案，因此可以在模型二的基础上引入动态博弈论模型进行求解。

8.2 模型建立

通过对模型二分析发现，玩家在每次做出判断时都有不同的可选项，同时选择每个选项的概率也不相同。因此可以建立所有玩家的收益矩阵：

$$\mathrm{M}=(\mathrm{m_{ij}})_{\mathrm{n*k}}=\begin{bmatrix}\mathrm{p_{11}} & \cdots & \mathrm{p_{1k}}\\ \cdots & \cdots & \cdots\\ \mathrm{p_{n1}} & \cdots & \mathrm{p_{nk}}\end{bmatrix} \tag{23}$$

因为每名玩家都可以随机的采取行动，因此每名玩家的决策形成了混合策略。根据混合策略 Nash 均衡理论，设每名玩家选取i策略的概率为$\mathrm{p_i(i=1,2...k)}$，满足：

$$0\leqslant \mathrm{p_{ji}}\leqslant 1,\sum_{\mathrm{i}=1}^{\mathrm{k}}\mathrm{p_{ji}}=1 \tag{24}$$

所有的概率向量构成动态博弈模型的混合策略空间，记为$\mathrm{S_i}$。同时在混合策略下，所有玩家的效用函数用期望定义为：

$$\mathrm{U_j(p_j)}=\sum_{\mathrm{j}=1}^{\mathrm{n}}\left(\sum_{\mathrm{i}=1}^{\mathrm{k}}\mathrm{p_j}\right)\mathrm{m_{ji}} \tag{25}$$

综上所述，可以得出动态博弈模型为：

$$\max\ Z=\max\ M^{(K)} \tag{26}$$

$$s$$

$$s.t.\begin{cases}0\leqslant \mathrm{p_{ji}}\leqslant 1,\sum_{\mathrm{i}=1}^{\mathrm{k}}\mathrm{p_{ji}}=1\\ \mathrm{U_j(p_j)}=\sum_{\mathrm{j}=1}^{\mathrm{n}}\left(\sum_{\mathrm{i}=1}^{\mathrm{k}}\mathrm{p_j}\right)\mathrm{m_{ji}}\\ \dfrac{E_Q}{(C_{nw_i}*P_w+C_{mw_i}*P_f)*3}>1\\ \dfrac{D_d}{D_F^{(i)}}>1\\ \dfrac{F_L^{(i)}}{F_C}>1\\ \dfrac{C_B+C_R}{(D_F^{(i)}-D_L^{(i)}-1)*(E_Q-C_{nw_i}*P_w*3-C_{mw_i}*P_f*3)}>1\\ M=M_{初}-5n-10m-\sum_{k=0}^{v}V_{i(2)}*B_k+\sum_{k=0}^{\nu}\nu_i*E_i\end{cases} \tag{27}$$

8.3 模型求解

在计算机仿真过程中，依次判断每个决策函数的值，然后给出相关决策，玩家再通过动态博弈确定下一天的安排。

针对关卡六，通过蒙特卡洛模拟随机模拟出 30 天的天气情况进行仿真。关卡六设有村庄与矿山，本局有三名玩家。首先开局每名玩家均不知道对方的行动计划，但根据第六关的地图必有两个人会走到同一个地图上。当天结束各个玩家即可知道其他玩家的信息。对于两个在一起的玩家通过效用函数$U_j(p_j)$即可在下一天选择不同的路，而另一位玩家则可以不受影响继续按原定计划前进。对于在砂矿附近的决策方案，玩家可以不断判断决策函数状态，并根据期望效用进行博弈，从而确定使所有玩家收益均衡的行动方案。下表为多名玩家收益均衡的行动方案示例：

表 8.3：多名玩家线路表

玩家	路线	余额（元）
玩家一	1->6->11->12->17->18->18(挖)->18(挖)->18(挖)->18(挖)->18(挖)->18(挖)->18(挖)->18(挖)->18(挖)->19->24->25	10952.5
玩家二	1->3->4->9->14->13->13->18->18(挖)->18(挖)->18(挖)->18(挖)->18(挖)->18->18(挖)->18(挖)->18(挖)->18(挖)->18->19->20->25	9475
玩家三	2->7->8->13->13->14->19->19->18->18(挖)->18(挖)->18(挖)->18(挖)->18(挖)->18(挖)->18(挖)->18(挖)->18(挖)->18(挖)->18->19->19->14->19->24->25	11490

8.4 模型检验

问题四模型引入了混合策略 Nash 均衡理论，玩家每天可以通过动态博弈以及相关决策标准确定第二天的行动方案。相比于单一的决策模型，动态博弈可以根据每天情况的不同给出不同的决策方案。在上表中可以观察到在矿区已有两名玩家挖矿时，另一名玩家为了不亏损一直在矿山附近等待，当有玩家离去时该名玩家立刻进入矿山。表明该模型具有较好的动态规划能力，模型检验效果较好混合策略 Nash 均衡理论。

九、模型评价与推广

9.1 模型优点

1）模型环环相扣。从已知天气情况的单玩家的线性规划求最优，到未来天气未知时单玩家的风险决策模型；从已知天气情况的多玩家博弈论约束下的最佳策略模型，到基于混合策略 Nash 均衡理论的动态博弈模型，四个模型从浅入深，从已知到未知，从单人到多人。这样的模型系统考虑全面，具有很好的推广性。

2）求解方法多样可靠。每种求解方法都是针对模型和沙漠探险游戏规则而制定的求解算法，分别采取了遗传算法、模拟仿真、格雷码-二进制混合编码的改进遗传算法、蒙特卡洛模拟法等方法来求解，力求方便快捷，真实可靠。
3）模型考虑周全，且经过检验稳定性好。四个模型解出来的的最优策略详细前面，在不同参数取值解出来的结果都体现一定的趋势和特征。不仅考虑到了游戏中的行走决策、补给决策，更是在开局购置物品上进行优化，尽量取得最优解。
4）模型受益人群覆盖范围广，为广大玩家提供可行的建议。模型不仅在已知天气时从无风险方面考虑最大收益（优先考虑安全），更会在不知道未来天气的前提下给出最佳建议。通过选取不同的心理风险决策阈值，不同风险承受能力的玩家可以选取属于自己的最佳方案。

9.2 模型欠缺

1）题目在未来天气不知道的前提下，没有提供天气数据和天气比例，因此采用的模拟仿真和蒙特卡洛模拟法可能会出现一定程度的结果偏差。
2）因为时间欠缺，没有在模型求解部分采取更多的优化算法去求解和比较。未来可以尝试不同算法去找到更好的解法。

9.3 模型推广

1）对于单人模型，可以采用更多的地图进行实际验证，并且单人模型在形式上并没有复杂的结构，如果沙漠探险游戏有更多好玩的机制可以在原模型上进行修改、更新和推广。
2）对于多人模型，如果未来有新的机制或者如果扩充决策的选择，可以直接进行修改更新。并且多人模型在博弈上不一定采取单单几个指标，还可以在未来组合更多的模型，使模型更加完善。

十、参考文献

[1]姜启源，谢金星，叶俊.数学模型(第五版）[M].北京：高等教育出版社，2018.369-398

[2]陈晓艳,张东洋,苏学斌, 等.基于改进遗传算法和多目标决策的货位优化策略[J].天津科技大学学报,2020,35(4):75-80. DOI:10.13364/j.issn.1672-6510.20190109.

[3]钱小冬.多阶段不确定最优控制的微分动态规划法[D].江苏:南京理工大学,2014. DOI:10.7666/d.Y2521823.

[4]陈晓艳,张东洋,苏学斌, 等.基于改进遗传算法和多目标决策的货位优化策略[J].天津科技大学学报,2020,35(4):75-80. DOI:10.13364/j.issn.1672-6510.20190109.

附录

支撑材料的文件列表：

1） Result.xlsx

2） MATLAB 程序及数据

MATLAB 程序

yichuan.m

```
%cities = cities';
code = 30;
max_gEN = 100;
pop_p_size = 100; % 遗传算法种群大小
crossover_probabilty = 0.9; %交叉概率
mutation_probabilty = 0.1; %变异概率
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
gbest = Inf;
% 随机生成城市位置
%cities = rand(2,code) * 100;%100 是最远距离
% 计算上述生成的城市距离
D_istances = maptest;
% 生成种群 为 30 长度的二进制码
pop_p = zeros(pop_p_size, code);
for i=1:pop_p_size
    for j = 1:30
        pop_p(i,:) = randsrc(1,30,[1 0;0.9 0.1]);
    end
end
off_spring = zeros(pop_p_size,code);
%保存每代的最小路劲便于画图
min_pathes = zeros(max_gEN,1);
% GA 算法
Money_start = 10000;
for  gen=1:max_gEN
% 计算适应度的值，即路径总距离
[fval, sumDistance, minPath, maxPath,best] = fitness(D_istances, pop_p,Money_start,maptest,weather,cost);
% 轮盘赌选择
tournamentSize=4; %设置大小
for k=1:pop_p_size
```

```
% 选择父代进行交叉
tourpop_pD_istances=zeros( tournamentSize,1);
for i=1:tournamentSize
randomRow = randi(pop_p_size);
tourpop_pD_istances(i,1) = sumDistance(randomRow,1);
end
% 选择最好的，即距离最小的
parent1  = min(tourpop_pD_istances);
[parent1X,parent1Y] = find(sumDistance==parent1,1, 'first');
parent1Path = pop_p(parent1X(1,1),:);
for i=1:tournamentSize
randomRow = randi(pop_p_size);
tourpop_pD_istances(i,1) = sumDistance(randomRow,1);
end
parent2  = min(tourpop_pD_istances);
[parent2X,parent2Y] = find(sumDistance==parent2,1, 'first');
parent2Path = pop_p(parent2X(1,1),:);
subPath = crossover(parent1Path, parent2Path, crossover_probabilty);%交叉
subPath = mutate(subPath, mutation_probabilty);%变异
off_spring(k,:) = subPath(1,:);
min_pathes(gen,1) = minPath;
end
fprintf('代数:%d   最短路径:%.2fM \n', gen,minPath);
% 更新
pop_p = off_spring;
gbest = minPath;

end

function [childPath] = crossover(parent1Path, parent2Path, prob)
% 交叉
random = rand();
if prob >= random
    [l, length] = size(parent1Path);
    childPath = zeros(l,length);
    for i = 1:length
        childPath(1,i) = -1;
    end
    setSize = floor(length/2) -1;
    offset = randi(setSize);
    for i=offset:setSize+offset-1
        childPath(1,i) = parent1Path(1,i);
    end
    iterator = i+1;
```

```
    j = iterator;
    while any(childPath == -1)
        if j > length
            j = 1;
        end
        if iterator > length
            iterator = 1;
        end
        %交叉采用随机交叉的方法 其余的位置直接交叉
        childPath(1,iterator) = parent2Path(1,j);
        iterator = iterator+1;
        j = j + 1;
    end
else
        childPath = parent1Path;
end
end

function [ fitnessvar, sumD_istances,minPath, maxPath, best ] = fitness( D_istances, pop_p,Money_start,
maptest,weather,cost )
% 计算整个种群的适应度值 即最后的资金大小 然后取最优
[pop_p_size, col] = size(pop_p);
%col 长度为 30
bool_success = zeros(1,pop_p_size);
sumD_istances = zeros(pop_p_size,1);
fitnessvar = zeros(pop_p_size,1);
for i=1:pop_p_size
  % pop_p(1,:) = [1,1,1,1,1,1,1,1,1,1,0,1,1,1,1,1,1,0,1,1,1,1,1,1,1,0,1,1,0,1,1];
   water_now = 133;
   food_now = 400;
   money_now = Money_start - water_now*5 - food_now*10;
   %针对每一个种群的个体进行计算
   %在给定初始购买后进行第一阶段 行走阶段的计算
%当二进制码累加超过最短路距离时即为到达目的地
tempnum1 = 0;%记录路程数
tempnum2 = 1;%记录累加数
[dis1,p1] = floyd(maptest,1,12);
while tempnum1 < dis1
   if tempnum1 == dis1-2
      %到达矿山前经过村庄 此时补充补给到满
      if food_now < 234
         ff1 = 20*(234-food_now);

         food_now = 234;
```

```
118.        else
119.            ff1 = 0;
120.        end
121.        if water_now < 244
122.            ww1 = 10*(244-water_now);
123.            water_now = 244;
124.        else
125.            ww1 = 0;
126.        end
127.        money_now = money_now - ff1 - ww1;
128.        if money_now <0%防止没有资金购买的保险措施
129.            money_now = money_now + 20*5 - 10*5;
130.            food_now = food_now-5;
131.            water_now = water_now-5;
132.        end
133.    end
134.    if weather(1,tempnum2) == 3
135.        %如果是沙暴天 直接停靠
136.        food_now = food_now - cost(2,weather(1,tempnum2));
137.        water_now = water_now - cost(1,weather(1,tempnum2));
138.    else
139.        tempnum1 = tempnum1 + pop_p(i,tempnum2);
140.        if pop_p(i,tempnum2) == 1
141.            %说明当天选择行进 此时消耗水和食物
142.            food_now = food_now - 2*cost(2,weather(1,tempnum2));
143.            water_now = water_now - 2*cost(1,weather(1,tempnum2));
144.        else
145.            %说明当天选择停靠
146.            food_now = food_now - cost(2,weather(1,tempnum2));
147.            water_now = water_now - cost(1,weather(1,tempnum2));
148.        end
149.    end
150.    tempnum2 = tempnum2 + 1;
151. end
152. cf = 0;%记录是否从村庄返程
153. %为了撤离 提前计算该个体的临界逃离时间和所需要的物资
154. [dis2,p1] = floyd(maptest,12,27);
155. templeast = 0;%用来计数
156. tempnum4 = 30;
157. least_time = 30;%记录最迟时间
158. while templeast < dis2
159.    if pop_p(i,tempnum4) == 1 &&weather(1,tempnum4)~=3
160.        templeast = templeast + pop_p(i,tempnum4);
161.    end
```

```
    least_time = least_time - 1;
    tempnum4 = tempnum4 - 1;
end
food_least = 0;%记录最少需要多少食物
water_least = 0;%记录最少需要多少水
%已知该个体的最迟撤离时间 就可以算出最少需要多少资源
for l = least_time : 30
    if weather(1,l) == 3
        food_least = food_least + cost(2,weather(1,l));
        water_least = water_least + cost(1,weather(1,l));
    else
        food_least = food_least + 2*cost(2,weather(1,l));
        water_least = water_least + 2*cost(1,weather(1,l));
    end
end
%到达矿山后开始改变编码意义 此时的 0-1 表示是否挖矿以及当需要 补给的时候是否补给
tempnum3 = 1;%记录进入第二阶段的时间
tempnum5 = 0;%记录完成补给的走路路程
befalse = 0;
while tempnum2 < least_time
    %说明此时还可以挖矿
    if food_now <= 2*cost(2,weather(1,tempnum2+1))+2*cost(2,weather(1,tempnum2+2))+ 3*cost(2,wea
    ther(1,tempnum2))|| water_now <= 2*cost(1,weather(1,tempnum2+1))+2*cost(1,weather(1,tempnum2+2
    ))+3*cost(1,weather(1,tempnum2))
        %说明此时需要补给 因为今天如果选择工作后不能及时去补给
        if food_now <= 2*cost(2,weather(1,tempnum2+1))+2*cost(2,weather(1,tempnum2+2))|| water_now
    <= 2*cost(1,weather(1,tempnum2+1))+2*cost(1,weather(1,tempnum2+2))
            %如果此时仍小于 这说明已经失败
            bool_success(1,i) = -1;
            break;
        end
        %但是此时如果未来有沙暴 则会多承受一轮移动 浪费时间在多余的路程也视为失败
        if weather(1,tempnum2+1) == 3 || weather(1,tempnum2+2) == 3
            bool_success(1,i) = -1;
            break;
        end
        %因为该情况下必定可以到达村庄 所以直接进行扣除
        tempwalk = 0;
        while tempwalk <2
            if weather(1,tempnum2) == 3
                %此时停留
                food_now = food_now - cost(2,weather(1,tempnum2));
                water_now = water_now - cost(1,weather(1,tempnum2));
```

```
        else%此时行走一个单位
            food_now = food_now - 2*cost(2,weather(1,tempnum2));
            water_now = water_now- 2*cost(1,weather(1,tempnum2));
            tempwalk = tempwalk +1;
        end
        if food_now<0||water_now < 0
            bool_success(1,i) = -1;
        end
         tempnum2 = tempnum2+1;%时间流逝
    end

    %如果可以进行补给，则进入补给阶段
    %在已知未来的天气情况下 补给的数量应该是固定的
    %此时应该有两种情况 第一种是此时如果离最后离开期限小于 3 天的话建议直接离开 因为无法挖矿
    %如果返回路上同样要出现沙暴 也建议不要浪费时间 直接前往终点
    if least_time - tempnum2 < 3 || weather(1,tempnum2) == 3|| weather(1,tempnum2+1) == 3
        %此时补给的数量为补到正好可以回去 并且开始前往终点
        %首先计算此时回去需要多少资源
        %先计算在当前决策序列上需要多少天回去 需要的资源则是需要补充的资源
        back_need = 0;%记录需要多少天回去
        tempnum6 = 0;%记录里程数
        food_least2 = 0;
        water_least2 = 0;
        [dis3,p1] = floyd(maptest,15,27);
        while tempnum6 < dis3
            if tempnum2+back_need >30
                befalse = 1;
                bool_success(1,i) = 0;
                break;
            end
            if weather(1,tempnum2+back_need) == 3
                %此时停留
                food_least2 = food_least2 + cost(2,weather(1,tempnum2+back_need));
                water_least2 = water_least2+cost(1,weather(1,tempnum2+back_need));
            else%此时行走一个单位
                food_least2 = food_least2 + 2*cost(2,weather(1,tempnum2+back_need));
                water_least2 = water_least2+ 2*cost(1,weather(1,tempnum2+back_need));
                tempnum6 = tempnum6 +1;
            end
            back_need = back_need+1;
        end
        %然后根据计算结果进行补给
        if befalse == 1
```

```
            break;
        end
        if food_least2 - food_now >0 && water_least2 - water_now >0
            %此时都缺
            money_now = money_now - 20*(food_least2 - food_now) - 10*(water_least2 - water_now);
            food_now = food_least2;
            water_now = water_least2;
        elseif food_least2 - food_now >0 && water_least2 - water_now <=0
            money_now = money_now - 20*(food_least2 - food_now);
            food_now = food_least2;
        else

            money_now = money_now - 20*(water_least2 - water_now);
        water_now = water_least2;
        end
        %补给结束后直接跳出 进行返程
        cf = 1;
        break;
    end
    if befalse == 1
            break;
    end
    %如果还可以返回挖金 那么补给的数量要计算上最迟返回
    earnmore_time = least_time - tempnum2 - 2;
    money_now = money_now - 20*(food_least - food_now + 2*(cost(2,weather(1,tempnum2))+cost(2,
weather(1,tempnum2+1))));
    money_now = money_now - 10*(water_least - water_now + 2*(cost(1,weather(1,tempnum2))+cost(
1,weather(1,tempnum2+1))));
    food_now = food_least+ 2*(cost(2,weather(1,tempnum2))+cost(2,weather(1,tempnum2+1)));
    water_now = water_least+ 2*(cost(1,weather(1,tempnum2))+cost(1,weather(1,tempnum2+1)));
    %接着是补充在矿上还要工作的分量

    for ll = 1:earnmore_time
        money_now = money_now - 3*20*(cost(2,weather(1,tempnum2+2+ll)));
        food_now = food_now+3*cost(2,weather(1,tempnum2+ll+2));
        money_now = money_now - 3*10*(cost(1,weather(1,tempnum2+2+ll)));
        water_now = water_now+3*cost(1,weather(1,tempnum2+ll+2));
    end
    tempnum2 = tempnum2+2;%时间流逝
    %然后是返回之后开始挖矿，此时已经不需要二次补给了
    money_now = money_now + 1000*earnmore_time;
    food_now = food_least;
    water_now = water_least;
    tempnum2 = tempnum2+earnmore_time;
```

```
287.       continue;
288.     end
289.     if tempnum3== 1
290. %         %说明此时第一次进入矿山 无法挖矿
291. %         %此时只能停留
292.         %否则根据编码选择是否挖矿
293.       if pop_p(i,tempnum2) == 1
294.           %说明要选择挖矿
295.           money_now = money_now + 1000;
296.           food_now = food_now - 3*cost(2,weather(1,tempnum2));
297.           water_now = water_now - 3*cost(1,weather(1,tempnum2));
298.           tempnum2 = tempnum2+1;%时间流逝
299.       else%否则停留
300.           food_now = food_now - cost(2,weather(1,tempnum2));
301.           water_now = water_now - cost(1,weather(1,tempnum2));
302.           tempnum2 = tempnum2+1;%时间流逝
303.       end
304.     end
305.
306. end
307. %此时结束循环之后检查记录失败的数组 如果失败直接淘汰
308. %如果没有失败 就计算最后的行程 并将最后的金额作为适应度值
309. if bool_success(1,i) == -1
310.     sumD_istances(i) = 0;
311. else
312.     %如果物资不够可以购买
313.
314.     %进行最终进程 根据指令序列进行返程
315.     if cf ==1%此时从村庄返程
316.       for kk = 1 : back_need
317.           if weather(1,tempnum2+kk-1) == 3
318.               food_now = food_now - cost(2,weather(1,tempnum2+kk-1));
319.               water_now = water_now - cost(1,weather(1,tempnum2+kk-1));
320.           else
321.               food_now = food_now - 2*cost(2,weather(1,tempnum2+kk-1));
322.               water_now = water_now - 2*cost(1,weather(1,tempnum2+kk-1));
323.           end
324. %           if food_now <0||water_now<0
325. %               bool_success(1,i) = -1;
326. %               break;
327. %           end
328.       end
329.     else%此时从矿山返程
330.       if food_least - food_now >0 && water_least - water_now >0
```

```
            %此时都缺
            money_now = money_now - 20*(food_least - food_now) - 10*(water_least - water_now);
            food_now = food_least;
            water_now = water_least;
        elseif food_least - food_now >0 && water_least - water_now <0
            food_now = food_least;
            money_now = money_now - 20*(food_least - food_now);
        elseif food_least - food_now <0 && water_least - water_now >0
            water_now = water_least;
            money_now = money_now - 20*(water_least - water_now);
        end
        for kk = tempnum2 : col
            if weather == 3
                food_now = food_now - cost(2,weather(1,kk));
                water_now = water_now - cost(1,weather(1,kk));
            else
                food_now = food_now - 2*cost(2,weather(1,kk));
                water_now = water_now - 2*cost(1,weather(1,kk));
            end
%             if food_now <0||water_now<0
%                 bool_success(1,i) = -1;
%                 break;
%             end
        end
    end
end
if bool_success(1,i) == -1
    sumD_istances(i) = 0;
else
    sumD_istances(i) = money_now + food_now*2.5+ water_now*5;
end
end
mmax = 0;
mmax_pos = 1;
for i = 1:pop_p_size
    if sumD_istances(i) > mmax
        mmax = sumD_istances(i);
        mmax_pos = i;
    end
end
best = zeros(1,30);
best = pop_p(mmax_pos,:);
sumD_istances = -1*sumD_istances;
minPath = min(sumD_istances);
```

```
maxPath = max(sumD_istances);
for i=1:length(sumD_istances)
fitnessvar(i,1)=(maxPath - sumD_istances(i,1)+0.000001) / (maxPath-minPath+0.00000001);
end
end

function [ mutatedPath ] = mutate( path, prob )
%对指定的路径利用指定的概率进行更新
random = rand();
if random <= prob
[l,length] = size(path);
index1 = randi(length);
index2 = randi(length);
%交换
temp = path(l,index1);
path(l,index1) = path(l,index2);
path(l,index2)=temp;
end
mutatedPath = path;
end

function [n_citys,city_position] = Read(filename)
fid = fopen(filename,'rt');
location=[];
A = [1 2];
tline = fgetl(fid);
while ischar(tline)
if(strcmp(tline,'NODE_COORD_SECTION'))
while ~isempty(A)
A=fscanf(fid,'%f',[3,1]);
if isempty(A)
break;
end
location=[location;A(2:3)'];
end
end
tline = fgetl(fid);
if strcmp(tline,'EOF')
break;
end
end
[m,n]=size(location);
n_citys = m;
city_position=location;
```

```
fclose(fid);
end
```

floyd.m

```
%利用 Floyd 算法求出最短路
function [distance,minpath] = floyd(A,firstpoint,lastpoint)
n = size(A,1);
path = zeros(n);
for k = 1:n
    for i = 1:n
        for j = 1:n
            if A(i,j) > A(i,k) + A(k,j)
                A(i,j) = A(i,k) + A(k,j);
                path(i,j) = k;
            end
        end
    end
end
distance = A(firstpoint,lastpoint);
parent = path(firstpoint,:);
parent(parent == 0) = firstpoint;
minpath = [lastpoint];
temp = lastpoint;
while temp ~= firstpoint
    p = parent(temp);
    minpath = [p,minpath];
    temp = p;
end
end
```