基于图与矩阵的“穿越沙漠”决策模型

摘要

本文通过建立**图模型**，借助 Dijkstra **算法**，将游戏地图抽象并简化为数学模型。与此同时，借助**矩阵的运算**，描述游戏中的各项资源消耗以及玩家行动，以期构造帮助玩家获得游戏胜利的决策模型。

针对问题一，我们关注到，当已知所有的天气状况，理论上来说可以找到策略的全局最优解，首先使用基于**深度优先搜索算法**的程序对全局的情况进行仿真模拟，对得到的结果进行批量数据分析，并依靠数据分析的结果进行规律总结并辅助决策模型的搭建。本文基于决策论的知识，首先提供给玩家统领性的原则帮助玩家选择数个可以获得最大利益的策略，然后给出村庄和起点的购买策略，最后搭建基于**损益矩阵**的决策模型，帮助玩家优中择优，并最终找到最优的策略。

针对问题二，由于未来的天气因素未知，所以无法通过分析地图得到全局最优解，而只能保证每一次移动做出局部最优解。我们通过**马尔可夫模型**，对将来的天气进行预测，以便我们对下一步做出决策。经过分析，我们找到了几个影响未来收益的主要因素，并建立起各个功能性节点关于这些因素的**期望函数**，通过比较各个功能性节点的期望函数，我们就可以决策出下一步的理想目的地。

在模型检验中，我们对该模型所得的结果与已知全部天气序列时所得到的最优解作比较，证明了该模型的合理性。

针对问题三，本文基于**完全信息静态博弈**的理论，提出了一系列基于 Nash **原理**下的假设，并用线性模型简化了每名玩家的效用函数的计算。通过**线性规划**的方法求解模型，给出了一般情况下，多人游戏的最优策略。针对关卡五，进一步提出了**对等博弈模型**。针对关卡六，额外给出了**动态递推博弈模型**。

关键词：图论、博弈论、向量空间、决策论、深度优先搜索

1. 问题重述

1.1 问题一

游戏中规定单个玩家参与限时通关，通关后需使资金余量最大，故我们在保证生存的前提下应当尽可能谋取更多资金收益。单纯通关并非我们目的，故需要考虑远赴挖矿、甚至是边补给边挖矿等情形。问题一中，我们将在已知天气的情况下，根据游戏规则给出一般情况下的最优决策。

1.2 问题二

第二问相比于第一问，增加了天气的不确定性；这决定了我们难以从一开始就直接制定出绝对的行动路线。我们需要通过当天的天气情况以及自身的情况去预测未来，以决定下一步的路线。

1.3 问题三

问题三中多玩家的引入直接导致了博弈问题。多玩家在线游戏应避免同行、同挖、同买等情况，从而使得原来的最优路径因为损耗的增大和收益的减少而不再是最优路径。我们需要改进我们的模型，对这种博弈情况给出最优策略。

2. 问题分析

2.1 问题一

问题一属于上帝视角求解最优路径，首要问题是如何将地图抽象成数学语言，而图模型是解决这一问题的首要工具。其次，因为先知天气，理论上可以计算所有可行路线找到最优解，但这样的时间消耗将无法估计。为了能够找到更优方案，我们需要对一些已知可行路径进行分析比对，从中提取有用信息。为了评估每条路径经的优劣，需要建立一个损益评估的模型，来指导在一般情况下的决策。

2.2 问题二

天气未知带来的不确定性使得我们需要在每个功能性节点（起点、矿井、村庄和终点），根据之前决策所形成的状态做出决策，使得后续的决策构成最优策略，这就推动我们去预测未来的天气和行动。而这就带来了对于天气和行动的表示问题。我们需要找到一个直观、方便的数学模型来描述它们。

由于天气的未知，我们不可能制定一个“静态”的路线，我们需要走一步看一步。那么，问题就变成了在每一个节点的决策问题，而对于大多数节点，这些决策方法应该是相近的，因此我们只需讨论出一个通用的决策方案即可。

2.3 问题三

在问题三中，我们需要在前面两问的基础上，引入多人游戏的机制。并增添了新的规则，在新的规则下，当与别人同时同地挖矿，购买，行走时都需要付出更大的代价。这实质上是一个博弈的问题，此时的决策者是所有玩家，效用函数是决策者到达终点时所剩余的资金，决策的目的是使己方玩家到达终点时的效用函数最大。问题三中的两小问设置了不同的情形，分别对应了问题一二中所设置的决策背景，因此我们可以参照问题一二中的思想，在问题三（1）中，我们通过决策寻找全局最优解，在问题（2）中，我们通过决策寻找局部

最优解从而尽可能地逼近全局最优解。与此同时，我们求解本问题，需要基于博弈论的基本假设，寻找 Nash 均衡点。

3. 模型假设

1、负重不会对食物和水的消耗产生影响。
2、假设在高温天气下，玩家在地图上空白区域停留的收益小于前进的收益。

4. 符号说明

符号	说明
$\boldsymbol{e_n}$ $(n = 1,2,3)$	天气单位向量
$\boldsymbol{\alpha}$	天气向量
W	天气矩阵
k_n $(n = 1,2,3)$	天气概率系数
M	天气属性矩阵
$\boldsymbol{u_n}$ $(n = 1,2,3)$	行动单位向量
$\boldsymbol{\beta}$	行动向量
A	行动矩阵
$p_n(n = 1,2,3)$	行动概率系数
L	价格矩阵
N	物资消耗矩阵
n	物资消耗量
Ω	损益矩阵
G	负重上限
Q	初始金钱
B	挖矿的基础收益
P	状态转移概率矩阵
d	剩余天数
f_i	第 i 个玩家的效用函数

5. 基于图论对地图的初步建模与抽象

5.1 将地图抽象为图模型

为了方便后续的讨论和计算机编程计算，我们基于图论的知识，用一个具有有限的节点集和边集的无向图网络抽象化地代表地图。

$$G_0 = (V_0, E_0)$$

在无向图网络$G_0 = (V_0, E_0)$中，V_0是G_0中节点集合，节点类型一共有五种，分别是空白节点，起点节点，终点节点，村庄节点，矿山节点，分别代表地图中的空白块，起点块，终点块，村庄块和矿山块，E_0是G_0中的边集合，我们认为当地图中的两个块之间相邻，即有公共边时，其对应的节点之间有一条边连接。

以第一关为例，我们抽象出的无向图网络G_0如下图所示：

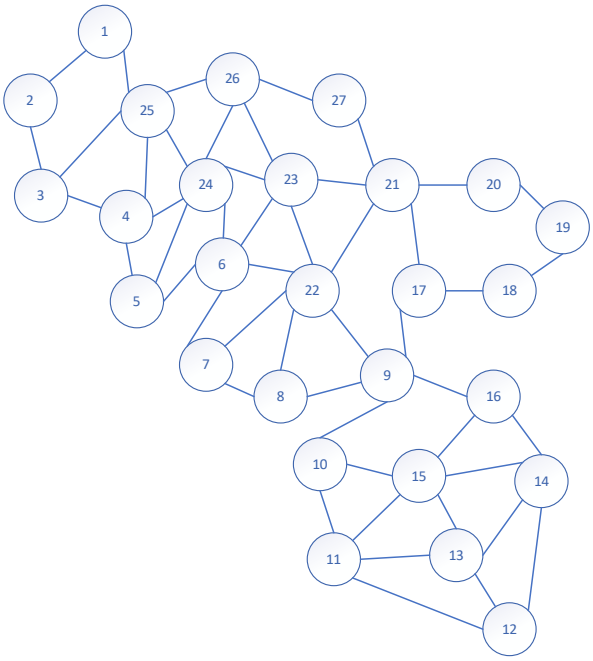

Fig.1 第一关无向图

5.2 对图模型的简化

我们将原有的图模型进一步抽象成边带权图

$$G = (V, E, f)$$

其中V是由起点节点，终点节点，村庄节点，矿山节点所组成的节点集合，E是G的边集合，我们认为当地图中的两个区域之间有一条可以从其中一个区域到达另一个区域的路时，其对应的节点有一条边连接，我们可以构造如下边集到正整数集的映射：

$$f: E \to N$$

$$\forall e \in E, f(e) = days$$

其中正整数集中的数代表了图上每一条边的权重，$days$是与该条边相邻接的两个节点之间移动所需要的最短天数，即不考虑沙暴天气的影响，下对该模型的合理性进行阐述。

首先，基于资源消耗最小原则，我们在空白节点上的动作分为以下两种:

(1) 在高温和晴朗天气下移动（基于在高温天气行走的收益大于停留的收益的假设）

(2) 在沙暴天气下停留。

因此在空白节点的操作完全由天气决定，所以我们无法通过决策来改变在空白节点的收益。

其次，由于我们需要在规定时间内到达终点并且尽量减少水和食物的消耗，因而要减少在空白顶点停留的时间，因此当从一个节点移动到另一个节点时，我们需要选择最短路径，因此，对于图中任意两个非空白顶点之间的移动，往往需要选择最短的路径。因此，我们将原有的无向图网络模型进一步简化成带权图网络模型不失一般性，并且为后续的进一步研究带来了便利。

除此之外，该带权图网络模型的构建还需要基于以下两条补充原则进行进一步的简化和修正：

1. 基于图论的知识，当G中A, B两个节点之间的边的权重与从A节点经过其他节点再到B节点的权重之和相等时，删去A, B之间的边不会影

响后续结果的分析。具体的演示如下图所示

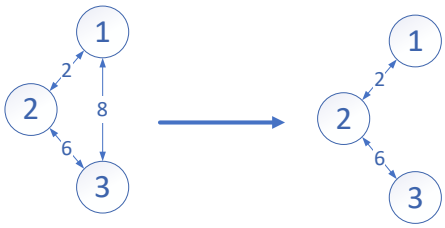

Fig.2 原则一演示说明

2. 在满足原则 1 的基础上，当在地图上的两个区域之间出现多条长度相同的路径时，选择经过更多功能点的那一条。首先从经济性的角度讲，我们在矿井点不会有负收益而只会有正收益。其次生存性的角度讲，当我们经过村庄时可以进行补给而使得生存的几率更大。因此，这样的修正是合理的。

上述图模型的构建，我们采用 Dijkstra 算法，借助计算机最终得到的带权图网络模型G如下：

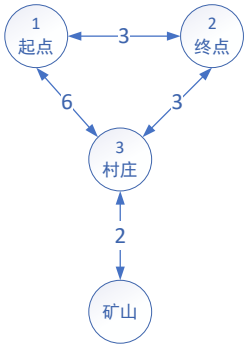

Fig.3 算法生成的图模型

6. 基于向量空间对天气和行动的描述

6.1 对于天气的描述

对于天气每天的天气情况，我们需要将它抽象成数学变量以便后续模型的建立。我们从游戏规则中知道，一共会出现三种天气，对此，我们将它们抽象成三个三维单位向量：

$$\boldsymbol{e_1} = (1,0,0)^T$$
$$\boldsymbol{e_2} = (0,1,0)^T$$
$$\boldsymbol{e_3} = (0,0,1)^T$$

这三个向量分别对应着晴朗、高温、沙暴三种天气。当某一天的天气情况已知时，则这天的天气就可以用以上三个向量的其中一个来表示，当我们不知道这一天的天气情况时，我们依然可以用向量来代表这一天的天气。我们令：

$$\boldsymbol{\alpha} = k_1\boldsymbol{e_1} + k_2\boldsymbol{e_2} + k_3\boldsymbol{e_3}$$
$$\text{其中，} k_1 + k_2 + k_3 = 1$$
$$\text{并且} k_1, k_2, k_3 \geq 0$$

通过三个单位向量的线性组合，我们就可以很好地抽象出未知天气的具体情况，其中三个线性组合系数的数学意义是三种天气在这一天的概率。这对我们后面在未知天气情况下的决策起到了很大的帮助。特别的，当我们整体考虑连续n

天的天气时，我们可以将每天的天气向量排列成一个$3*n$的天气矩阵：

$$W = [\boldsymbol{\alpha_1}, \boldsymbol{\alpha_2}, \boldsymbol{\alpha_3}, \dots\dots, \boldsymbol{\alpha_n}]$$

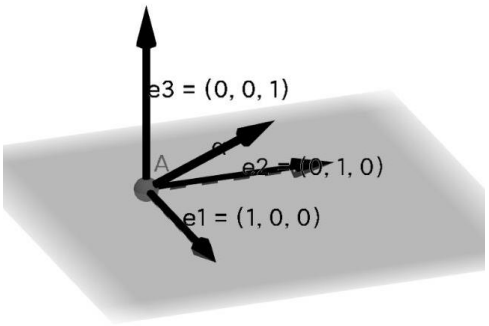

Fig.4 天气向量图示

6.2 对于行动的描述

参照天气的描述，我们一样可以利用向量和矩阵来描述我们在沙漠中每天的行动。与天气不同的是，在一天中我们采取不同的行动，我们所消耗物资的数量有很大的区别。根据游戏规则，具体物资消耗的规则如下：

- ✧ 当我们停一天，需要消耗一倍的基础消耗量
- ✧ 当我们选择前进，需要消耗两倍的基础消耗量
- ✧ 当我们选择挖矿，需要消耗三倍的基础消耗量

根据这些规则，我们选择以下三个三维向量作为行动的单位向量：

$$\boldsymbol{u_1} = (1,0,0)^T$$
$$\boldsymbol{u_2} = (0,2,0)^T$$
$$\boldsymbol{u_3} = (0,0,3)^T$$

这三个向量分别代表停留、前进、挖矿三种行动。同样在实际情况中，我们下一步采取的行动也是未知的，类比天气的描述方法，我们同样利用这三个向量的线性组合来描述下一步采取的行动：

$$\boldsymbol{\beta} = p_1\boldsymbol{u_1} + p_2\boldsymbol{u_2} + p_2\boldsymbol{u_3}$$

其中，$p_1 + p_2 + p_3 = 1$

且$p_1, p_2, p_3 \geq 0$

同样，三个线性组合系数代表下一步行动的预测概率。当我们考虑连续n天的行动时，我们也可以把每天的行动排列成一个$3*n$的行动矩阵：

$$A = [\boldsymbol{\beta_1}, \boldsymbol{\beta_2}, \boldsymbol{\beta_3}, \dots\dots, \boldsymbol{\beta_n}]$$

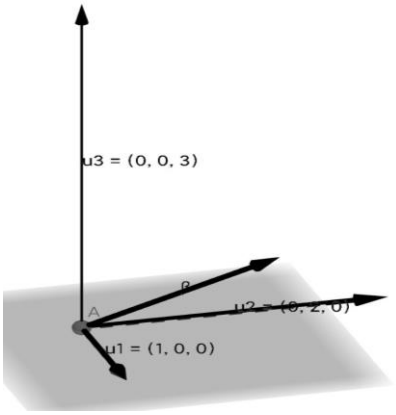

Fig.5 行动向量图示

7. 对于已知天气的决策模型

7.1 基于深度优先搜索的仿真模拟

由于游戏规则较为复杂，难以直接判断各种策略之间的优劣，因此我们决定从具体情况入手分析相对最优解，找出一般规律。我们随机生成了几个简单的地图，并按照一定的概率随机生成了几组可能的天气，在对其相对最优解进行求解。

7.1.1 最优解的求解——深度优先搜索

我们通过深度优先搜索法按照一定规律遍历绝大多数的解，找到其中的局部最优解。我们简化后的图上一共有四种节点：起点、村庄、矿山、终点。起点是整个图的入口，而终点是整张图的出口。当我们搜索到终点时，代表游戏结束，记录结果。游戏开始时，我们可以设定不同的物资购买比例，通过遍历得到最优解。

对于除了终点以外的节点，我们都可以选择走或者留，但是当遇到了沙暴天气就必须要留下。因此，对于每个节点，我们至少要遍历这些情况。

对于村庄，我们可以购买物资。由于我们知道每一天天气，因此当我们确定一条路径后，我们可以规划好购买的物资。所以，在深度优先搜索过程中，我们不需要考虑资金节省问题，只需要按照一定的比例去购买最多的食物和水。

对于矿山，我们可以选择挖矿与不挖矿，所以在矿山节点，我们需要增设一个选择——挖矿。

通过这种规律，借助计算机，我们就可以计算出最优解。这里我们利用 C 语言，以递归的形式实现了深度优先搜索的算法。（见附录七）

7.1.2 最优解的求解结果

由于结果较多，我们将选择其中一个的其中一部分进行展示：（其余结果见支撑材料）

起始水量	起始食物	购买比例				
		0	1	2	3	4
186	321	10650	12160	9920	9490	7750
188	318	10660	12150	9900	9470	7760
190	315	10670	12520	9990	9450	7770
192	312	10660	12500	10000	9430	8115
194	309	10090	12490	10010	9410	9350
196	306	10100	12470	10060	9390	9360
198	303	10110	12450	10200	9800	9370
200	300	10120	12430	10190	9780	9350
202	297	10130	12410	10410	9760	9330
204	294	10140	12390	10420	9740	9310
206	291	9550	12370	10430	9720	9290
208	288	9560	12350	10510	9700	9270
210	285	9570	12330	10730	9690	9250
212	282	9580	12310	10870	9940	9660
214	279	9590	12290	10850	9950	9640

216	276	9600	12270	11070	9960	9620
218	273	9570	12250	11050	10030	9600
220	270	9000	12230	11030	10790	9600
222	267	9010	12210	11170	10770	9610
224	264	9020	12190	11150	10750	9700
226	261	9030	12170	11130	10730	9710
228	258	9040	12150	11110	10710	9900
230	255	9050	12130	11090	10690	9880
232	252	8810	12110	11160	11070	9940

7.2 优选路径及策略

由于可选路径数量过于庞大，人很难在短时间内遍历计算，因此我们需要缩小路径范围。而我们基于该题的生存游戏背景：

- ✧ 获得足够的生存资源生存下来并在规定时间内通关
- ✧ 在通关后拥有尽可能多的资金余量

结合我们搜索得到的最佳路径，可以总结出以下规律性策略：

1) 奇货可居——在起点大量买进高价商品，即在满足生存条件的情况下尽可能多的以低价购进高价商品，以避免在村庄大量买进高价商品而造成的损失。
2) 物尽其用——尽量利用村庄的补给功能和矿井的收益功能：地图上于起点和终点之间存在若干的功能点，区别于路径中空白节点的负损耗，在规定时间内对功能点的尽量利用会使得我们有更长的生存距离，更大的资金余量，从而使得玩家占优。
3) 三是养精蓄锐——沙暴天气适当休息，即针对矿井和村庄相隔较远的情况下，多次补给带来的成本过高，我们考虑在极端恶劣的沙暴天气，适当休息以规避挖井三倍消耗对食物与水的致命性消耗。

根据这三条规律，针对不同具体的地图，我们就可以规划出多种可行且收益较大的路径方案，只需在这些路径中选择最优的一个即可。

7.3 物资购买方案决策

经过对于数据的分析和理论的推演，我们给出在起点和村庄购买资源的策略。**首先**，考察在起点的购买策略，对于已选定的路径，我们分为以下两种情况讨论：

1) 已选定的路径不经过村庄
2) 已选定的路径经过村庄

对于情况 1，在满足约束条件即生存性原则的情况下，由于在终点退回水和食物的价格仅有基准价格的一半，因此我们仅需要在起点购买整条路经所需要的水和食物的量即可使利益最大化。

对于情况 2，由于在村庄购买水和食物的价格是基准价格的二倍，因此所需要付出的代价更大，则我们需要在起始位置购买尽可能多的水和食物，**与此同时**，

$$2 * \max\{p_a, p_f\} - \max\{p_a, p_f\} > 2 * \min\{p_a, p_f\} - \min\{p_a, p_f\}$$

即在村庄购买水和食物之中价格更高的一方需要付出更高的代价，则从经济性的角度上来讲，在终点的收益也就会更小，因此我们需要在起点尽可能多地购买单价更高的物资。**而**考虑到生存性，我们在起点所购买的物资必须足以支撑我

们到达第一个村庄，因此给出以下三条原则：

a. 在起点购买的水和食物的量分别小于等于整条路经所需要的水和食物的量
b. 在起点购买的水和食物的量分别大于等于从起点到达第一个村庄所需要的水和食物的量
c. 在满足上述两条原则的基础上，尽可能多地购买水和食物中价格更高的一方

我们令起点购买的食物量为σ_f，购买水的量为σ_a，将他们写成向量形式：

$$\boldsymbol{\sigma} = [\sigma_a, \sigma_f]$$

我们需要利用天气来计算路径所需的水和食物。对于天气，30 天内的天气我们用天气矩阵W表示。由于不同的天气下的基础消耗量不同，因此，我们利用题目中的参数，构造出天气属性矩阵：(以关卡 1 为例)

$$M = \begin{bmatrix} 5 & 8 & 10 \\ 7 & 6 & 10 \end{bmatrix}$$

矩阵中的六个数分别是不同天气下水和食物的基础消耗量。由此我们可以通过矩阵乘法计算出每天所消耗的水和食物数量：

$$N = MW$$

则上述规则可以描述为：

$$MW_1 \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix} \leq \boldsymbol{\sigma} \leq MW \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix}$$

其次，考察在**村庄**的购买策略。由于在各个村庄购买资源所付出的代价相同，因此我们仅需按照以下两条原则进行补给：

a. 在村庄的补给量不超过总需求
b. 在满足上述原则的前提下，每次经过村庄尽可能多的购买水和食物。

7.4 损益矩阵的计算与行动决策

为了对不同的路径进行决策，我们需要对每条路径上的损益进行计算。我们假设一共有n条路径，当路径确定之后，每天的行动也随之确定，则这n调路径可以用n个行动矩阵$A_1 \sim A_n$来表示。

每天不同的行动也会导致消耗的变化，利用我们对行动矩阵的特殊定义，我们可以直接单条路径的消耗的水和食物：

$$\Omega_i = N_i A_i^T \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

我们令水的单价为p_a，食物的单价为p_f，我们可以构造价格向量：

$$L = [p_a \quad p_f]$$

令B为挖矿的基础收益，结合价格向量，加上挖矿所得，我们可以得到每条路径上的损益：

$$\varepsilon_i = A_i^T \begin{bmatrix} 0 \\ 0 \\ \frac{B}{3} \end{bmatrix} - 2L\Omega_i + L\boldsymbol{\sigma}$$

综合各路损益，损益矩阵：

$$\Omega = [\varepsilon_1, \varepsilon_2, \varepsilon_3, \dots\dots, \varepsilon_n]$$

7.5 不同路径的约束条件

本文中我们所有的决策建模均基于两条基本原则：

1. 生存性原则：保证在水和食物耗尽前以及在规定时间内到达终点
2. 利益最大化原则：尽可能使到达终点时保留更多的资金

对于给定的几条优选路径，我们需要对每条路径的生存性进行讨论，即该条路经是否可以在满足水和资源不消耗为零的前提下到达终点。我们分为以下两种情况进行考察：

1) 给定的路径经过村庄
2) 给定的路径不经过村庄

首先，对于情况 1，我们的生存物资全部来自于起点的购买，因此那么水和食物的总量受初始资金和背包负重的约束，我们约定水的单位重量为w_a，食物的单位重量为w_f，我们可以得到重量向量：

$$K = [w_a \quad w_f]$$

我们令G为最大负重，Q为初始金钱，那么需要满足约束条件：

$$\begin{cases} KMWA^T \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \leq G \\ LMWA^T \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \leq Q \end{cases}$$

只要满足上述约束条件即可到达终点，在不考虑利益最大化原则的前提下，该条路径是可行的。

其次，对于情况 2，当给定的路径经过村庄时，我们对于物资的补给不仅局限于起点，而在不考虑利益最大化的原则的前提下，起点和村庄对于补给物资的作用效果是类似的，因此我们将给定路径上所经过的起点节点，村庄节点，终点节点（地图上的同一村庄节点可以重复出现），如下图所示：

Fig.6 不定长链表模型

对于第i个区间，我们可以得出这个区间的天气矩阵W_i，行动矩阵为A_i到达第$i-1$个节点时，玩家剩余资金为q_i，则需要满足下列递推条件：

$$\text{对}\forall i\text{，有}\begin{cases} KMW_iA_i^T\begin{bmatrix}1\\1\\1\end{bmatrix} \le G \\ LMW_iA_i^T\begin{bmatrix}1\\1\\1\end{bmatrix} \le q_i \\ q_{i+1} = q_i - LMW_i\begin{bmatrix}1\\1\\1\end{bmatrix} + A_i\begin{bmatrix}0\\0\\\frac{B}{3}\end{bmatrix} \end{cases}$$

7.6 最优策略求解总结

1、根据所给优选路径的三条规则对路径进行筛选。
2、根据路径的约束条件，剔除一些不可行路径
3、根据购买方案决策、以及天气矩阵求出行动矩阵
4、根据天气矩阵、行动矩阵和所给参数计算损益矩阵
5、找到损益矩阵中元素的最大值所对应的路径，即为最佳路径

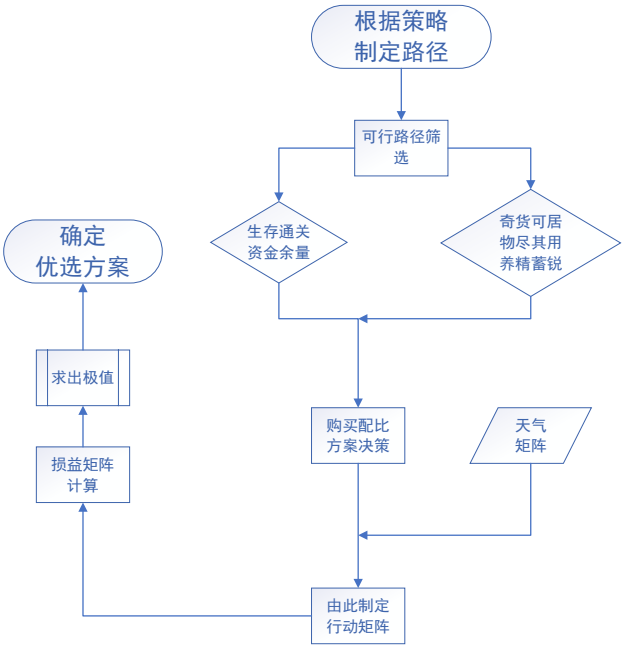

Fig.7 最优策略总结

7.7 对关卡一的讨论

首先对第一关的地图进行简化和抽象：

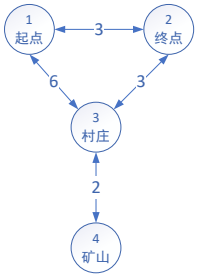

Fig.8 第一关简化图

我们根据总结的规律性策略和第一关简化后的图模型，利用程序，共找到了568 条路径，我们列出所有路径的行动矩阵，借助 Octave 进行矩阵运算，得到了关于这 568 调路径的损益矩阵。（详见支撑材料）。通过对损益矩阵中元素求解最大值，得到最优的解为 10430 元。最优路线如下：(答案见附录二)

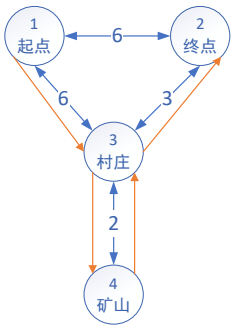

Fig.9 第一关路径展示

7.8 对关卡二的讨论

首先对第二关的地图进行简化和抽象：

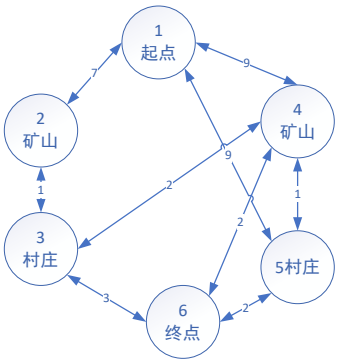

Fig.10 第二关简化图

根据总结的规律性策略和第二关简化后的图模型，共找到了 218452 条路径，我们列出所有路径的行动矩阵，借助 Octave 进行矩阵运算，得到了关于这 218452 调路径的损益矩阵。（详见支撑材料）。通过对损益矩阵中元素求解最大值，得到最优的解为 12570 元。最优路线如下：（答案见附录三）

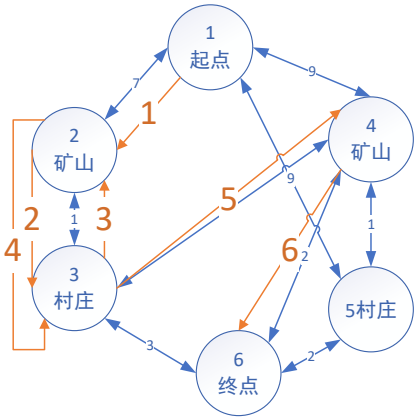

Fig.11 第二关路线展示

8. 对于未知天气的决策模型

8.1 未知天气的概率计算

为了方便对第二天的天气出现的概率进行判断，我们假定已知沙漠中各种天气出现的概率。这里我们利用马尔可夫模型进行对天气进行预测，其中一个重要的概念是状态转移概率矩阵[2]。

状态转移概率矩阵是由不同天气状态之间的转移规律的来的。以第一关的天气情况举例，通过统计，我们可以得出由晴天转为晴天、晴天转为高温、晴天转为沙暴等 9 种天气状态转变的次数，如下表所示：

	晴朗	高温	沙暴
晴朗	2	5	2
高温	5	6	3
沙暴	2	3	1

由统计表，其中 30 天内，有 5 天是由高温转化为晴朗，有 2 天由晴朗转化为沙暴。我们将数据按照行进行归一化，可以得到：

	晴朗	高温	沙暴
晴朗	0.222222	0.555556	0.222222
高温	0.357143	0.428571	0.214286
沙暴	0.333333	0.5	0.166667

于是我们即可得到这种情况下的状态转移概率矩阵：

$$P = \begin{bmatrix} 0.22 & 0.56 & 0.22 \\ 0.36 & 0.43 & 0.21 \\ 0.33 & 0.5 & 0.17 \end{bmatrix}$$

当我们已知天气发生的概率，我们可以通过随机生成相应概率的序列，通过统计，得到该概率下的状态转移概率矩阵。在我们的模型中，我们将基于状态转移概率矩阵，利用当天的天气对未来的天气进行预测。我们令当天的天气向量为α_t，第二天的天气向量为α_{t+1}，那么我们认为：

$$\boldsymbol{\alpha_{t+1}} = P\boldsymbol{\alpha_t}$$

8.2 地点的期望值

当我们不知到后续天气时，我们只能通过当天的状态来预判下一步的行动。我们找到了几个影响未来收益的主要因素，并建立起对于下一目的地期望值关于这些因素的函数。在这里我们将所有情况分为两类。一类是在空白节点，即在去往某一地点的路上，为决策下一理想目的地，我们需要利用期望函数对各个功能性节点的期望值进行评估，以选择最好的行动方案。第二类是为于某一功能性节点，如在矿山或者位于村庄，我们需要单独进行讨论。这里我们先对第一类情况进行讨论，我们需要得出各种地点的期望值的计算方法。

对于不同的地点，我们都需要对到该点的距离进行判断，结合天气和物资剩余情况，对到该点的代价进行评估。

我们通过图模型对地图的抽象，我们可以得出我们到各点的最短距离$\boldsymbol{x}$。因此，我们至少需要花费$\boldsymbol{x}$天到达该点，设当天天气向量为$\boldsymbol{\alpha}$，通过马尔可夫模型的计算，我们可以得到这一时间段的天气矩阵：

$$W_t = [P, P^2, P^3, \dots\dots, P^x]\boldsymbol{\alpha}$$

对于这几天的行动，由于路上不会停留挖矿，因此，每天的行动向量应该是

停留和前进的线性组合：

$$\beta_i = p_1 u_1 + p_2 u_2 = [u_1 \quad u_2] \begin{bmatrix} p_1 \\ p_2 \end{bmatrix}$$

而我们在路上停留和前进的概率会随天气变化而变化，由于我们只会在沙暴天气停留，因此停留的概率应和沙暴的概率形同。据此，我们可以发现：

$$\beta_i = [u_1 \quad u_2] \begin{bmatrix} p_1 \\ p_2 \end{bmatrix}$$

$$= [u_1 \quad u_2] \begin{bmatrix} 1 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \alpha_i$$

$$= [u_1 \quad u_1 \quad u_2] \alpha_i$$

于是我们可以得出这一段时间的行动矩阵：

$$A_t = [u_1 \quad u_1 \quad u_2] W_t$$

利用损益矩阵的算法，我们就可以预测路上行走所需花费：

$$w_t = LMW_t A_t^T \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

$$= LMW_t W_t^T [u_1 \ u_1 \ u_2]^T \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

$$= LMW_t W_t^T \begin{bmatrix} 1 \\ 1 \\ 2 \end{bmatrix}$$

同理，我们可以计算出物资消耗量：

$$n_t = MW_t W_t^T \begin{bmatrix} 1 \\ 1 \\ 2 \end{bmatrix}$$

对于矿山，当物资量充裕，剩余时间更多时，我们应该选择去矿山多赚钱。通过物资消耗量，我们可以预测出到达矿山后所剩的物资量：

$$n' = n - n_t$$

如果我们令剩余天数为d，那么对于矿山的期望值应为：

$$E_0(w_t, n', d) = j_{10} w_t + j_{20} n' + j_{30} d$$

其中$j_{20}, j_{30} > 0$ ，$j_{10} < 0$

对于村庄，只有当物资量紧缺，时间较为充裕时才会选则去，因此村庄的期望值应为：

$$E_1(w_t, n', d) = j_{11} w_t + j_{21} n' + j_{31} d$$

其中$j_{11}, j_{31} > 0$, $j_{21} < 0$

对于终点，当剩余时间严重不足时，必须向终点行进，因此，当d减小到一定的程度时，终点的期望必须上升非常快，因此终点的期望计算不能是线性的。我们选择反比例函数模型来刻画：

$$E_2(w_t, n', d) = j_{12} w_t + j_{22} n' + \frac{j_{32}}{d}$$

其中$j_{12},\ j_{22},\ j_{32} > 0$

8.3 挖矿的期望值

当我们正处于矿山时，我们有三种选择，走、挖矿、休息，我们需要对这三种选择进行评估。为了于前面形成统一的判断标准，对于挖矿，我们也需要计算出一个期望值。由于挖矿与赶路相比是由收入的，所以期望值中需要加上收入一项。根据对下一天的天气的预测，挖矿的收入为：

$$\omega_t = u_3 \begin{bmatrix} 0 \\ 0 \\ \frac{B}{3} \end{bmatrix} - LM\alpha_t u_3^T$$

$$= B - 3LMP\alpha$$

于是我们可以得到挖矿的期望值为：

$$E_3(\omega_t, n', d) = j_{13}\omega_t + j_{23}n' + j_{33}d$$

$$\text{其中} j_{13}, j_{23}, j_{33} > 0$$

考虑到我们在前一问中总结的一般规律，在矿山遇到沙暴天气，可以休息一天，以获取更长的挖矿时间，因此当满足：

$$E_3 > E_2, E_1, E_0 \text{ 且 } [0,0,1]P\alpha > 0.5$$

说明下一天挖矿的收益较大，但是遇到沙暴几率很大，那么下一天就停一天。

8.4 物资购买决策

当我们处于村庄节点时，我们不仅要判断下一步的行动，还需要决策买多少物资。由于后续天气状况未知，我们需要防备不时只需，因此我们倾向于多买。由于前文我们计算出了到每个关键地点的物资消耗，那么我们认为我们购买的物资需要尽量支撑我们到更远的地方。因此当我们可以购买物资时，我们购买的物资量为：

$$n_b = j_b \max\{w_t\}$$

$$\text{其中} j_b > 1$$

当这一值超出我们所能购买的最大量时，我们就按照所能购买的最大量来购买。

8.5 决策总结

玩家在每一点上，按照以下步骤做出决策：

1、通过状态转移概率矩阵，根据今天的天气，预测下一天的天气

2、如果自己的位置不在矿山，那么对地图上每个矿山和村庄计算期望值，找到期望最大的哪一个点，向期望最大的点运动。、

3、如果自己位于村庄，根据计算出来到下一个节点所需物资的最大值的倍数购买。

4、如果自己位于矿山，那么计算在矿山的期望值，与其他地点的期望值做对比，如果其他地点期望值更大，就向期望值更大的点移动，否则就停留挖矿。如果停留挖矿期望值更高，但是遇到了沙暴，就停留一天，不去挖矿。

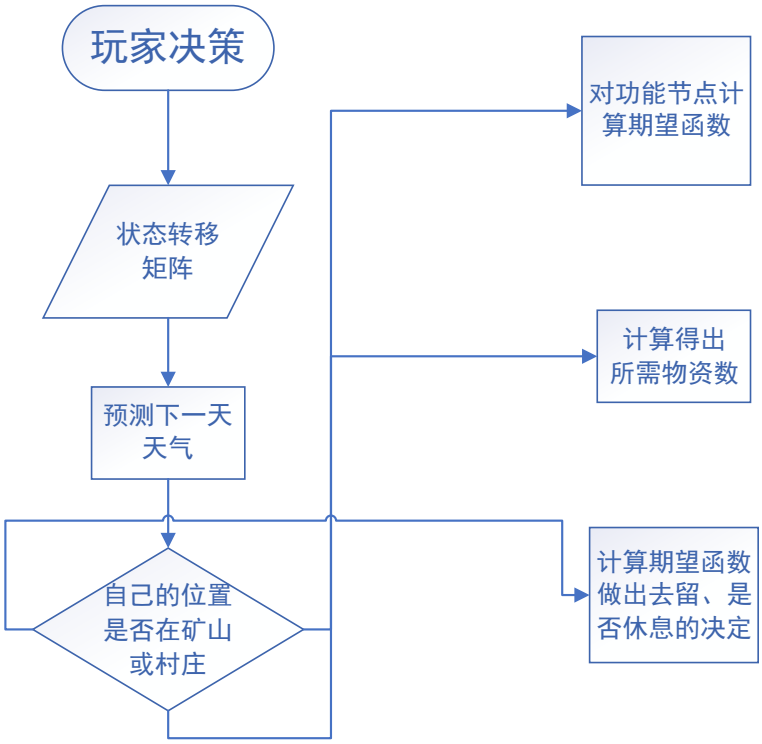

Fig.12 决策总结

8.6 对关卡三的讨论

我们将关卡三的地图简化：

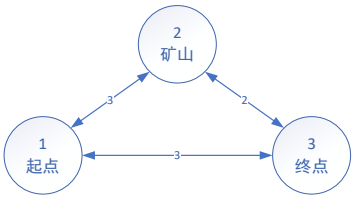

Fig.13 第三关简化图

由于这一关不会出现沙暴天气，我们假定高温、晴朗天气数量之比为 1：1，则我们给出相应的马尔可夫矩阵：

$$P = \begin{bmatrix} 0.61 & 0.39 & 0 \\ 0.59 & 0.41 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

我们根据高温晴朗天气的比例，生成了几组可能的天气条件，通过计算机模拟，计算出了按照我们给定的策略选择的结果。（见附录四）

8.7 对关卡四的讨论

我们将关卡四的地图简化：

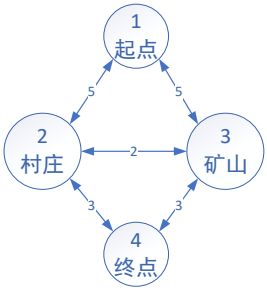

Fig.14 第四关简化图

由于较少出现沙暴天气，我们假定高温、晴朗、沙暴天气数量之比为 5：4：1，则我们给出相应的马尔可夫矩阵：

$$P = \begin{bmatrix} 0.42 & 0.49 & 0.09 \\ 0.45 & 0.41 & 0.14 \\ 0.32 & 0.34 & 0.33 \end{bmatrix}$$

我们根据高温晴朗天气的比例，生成了几组可能的天气条件，通过计算机模拟，计算出了按照我们给定的策略选择的结果。（见附录五）

9. 基于多人游戏的博弈模型

9.1 基于博弈论的假设

基于 Nash 均衡的原理，我们提出以下假设：

1. 假设每个参与者是理性的，是同质的，每个人都没有特殊的优缺点。在身份上没有什么差别。每个参与者选择的行为策略是为了实现特定状态下的收益（支付）最优的结果。
2. 假设一个玩家的策略选择不能被其他玩家预先知道或者猜到。
3. 假设所有的玩家都知道在解决问题一、二的过程中提出的最优化策略。

9.2 基于博弈论寻找最优策略

由于所有的玩家具有共同知识，同时做出决策，我们可以用完全信息的静态博弈模型来描述问题三中的过程。此时的决策者是所有玩家，每个参与人的效用函数是到达终点时所剩余的资金，决策的目的是使每个玩家到达终点时的效用函数最大。根据 Nash 均衡的思想，在达到 Nash 均衡点后，任一决策者单方面地偏离该策略均不能使自己的效用函数得到提升[6]。

我们约定Ω_i'是在新规则约束下第i个玩家的损益矩阵，则在k个玩家的情形下，我们可以形成一个k 个k维的张量，分别代表了k个玩家在每种情况下的损益张量。而针对静态博弈模型求解效用函数最大化的过程即是寻找 Nash 均衡点的过程，也就是在损益张量中寻找可以使各个玩家在不改变既定状态下可以得到最大利益的点。

由于在高维度上的 Nash 均衡点我们难以刻画，因此我们建立有k个玩家的情况下的效用函数的**线性模型**，然后用线性规划的方式进行求解

$$f_i = \sum_{j=0}^{k} a_{ij} * A_j + b$$

即可将我们所要求解的问题转化为

$$\max\ f_i$$

$$\begin{cases}\max f_0 \\ \cdots\cdots \\ \max f_k\end{cases}$$

即在保证其他人获得最大收益的情况下，使当前玩家获得更大的收益，而将上述求解过程推及其他人，最终得到的结果必然是 Nash 均衡点。

9.3 一般性最优策略

我们用单纯形法求解上式，通过对$k = 2,3,4,5$四种情况下的 Nash 均衡点数据进行观察比较，我们得出两条一般性的策略：

策略一：

$k = 2$时，每次移动均选择两个目标节点之间最短的路径。$k \geq 3$时，当两个目标节点之间有多条最短路径，则按照随机概率的方式进行选择,概率的计算方法在后文给出，下述其合理性。这是因为在只有两个玩家时，我们选择避开另一个玩家行走会降低我们的损耗，但是同样的也会减少另一个玩家的损耗，而且基于对问题一的讨论，另一个玩家所获得的利益显然更高，这对我们而言是不利的。当有多个玩家时，如果所有玩家均采用随机概率的方式进行选择，这样可以使单条路径的人数降到最少，因而所有玩家都可以获得更多的利益。下面我们根据博弈论给出概率的具体分配方法，首先我们假设在两个目标节点之间存在着r条路，第i条路对应的长度为l_i,$\boldsymbol{l} = [l_1, l_2, \cdots\cdots, l_r]$,

第i条路和第j条路有$t_{ij}(i \neq j)$个共同节点,则选择第j条道路的概率为：

$$p_j = t * \frac{\left(\max\{\boldsymbol{l}\} - l_j\right)}{r * \max\{\boldsymbol{l}\} - \sum_{i=1}^{r} l_i}$$

其中，t 是修正因子，由下式得出

$$t = \frac{\sum_{i=1}^{r}\sum_{j=1}^{r} t_{ij} - \sum_{i=1}^{r} t_{ij}}{\sum_{i=1}^{r}\sum_{j=1}^{r} t_{ij}}$$

策略二：

当地图上有矿山时，分别按照问题一中给出的方法计算从起点到终点和从起点经过矿山再到终点两种路径的损益，然后选择在问题一的情境下可以获得最大收益的那一条道路，下对于两人博弈的情况进行合理性阐述。因为基于 Nash 假设，我们可以得到双方的收益矩阵

	挖矿	不挖矿
挖矿	c/2 c/2	c v
不挖矿	v c	v/2 v/2

而通过观察收益矩阵，我们不难看出，当$c \geq v$时，如果我们选择挖矿而对方不挖矿，则我们可以获得更大的收益，而即便对方同样选择挖矿我们也可以使双方所获得利益相同，从竞争的角度来说可以获得最大的收益。$c \leq v$时同理。即在这种策略下，我们总能达到纳什均衡点。

9.4 对关卡五的讨论

由于所有玩家都已知在对于问题一的讨论中提出的最优化策略，那么基于 Nash 均衡的理论，所有玩家必定会在最优的路径之中选择一条。结合在上文一般性策略的讨论，我们将以下几条路径作为决策变量的范围，分别为：

路径1　1 -> 4 -> 6 -> 13
路径2　1 -> 5 -> 6 -> 13
路径3　1 -> 2 -> 3 -> 9 -> 9 -> 11 -> 13
路径4　1 -> 4 -> 3 -> 9 -> 9 -> 11 -> 13
路径5　1 -> 2 -> 3 -> 9 -> 9 -> 9 -> 11 -> 13
路径6　1 -> 4 -> 3 -> 9 -> 9 -> 9 -> 11 -> 13
路径7　1 -> 2 -> 3 -> 9 -> 9 -> 9 -> 9 -> 11 -> 13
路径8　1 -> 4 -> 3 -> 9 -> 9 -> 9 -> 9 -> 11 -> 13

分别对应了从起点直接到终点，从起点经过矿山挖矿（1，2，3 天）再到终点的情况。然后我们根据前文所述的损益矩阵的计算方法可以分别计算得出双方的收益矩阵：(源代码见附录六)

己方玩家：

	路径 1	路径 2	路径 3	路径 4	路径 5	路径 6	路径 7	路径 8
路径 1	9020	9400	9510	9400	9510	9400	9510	9400
路径 2	9400	9020	9510	9510	9510	9510	9510	9510
路径 3	9325	9325	8515	9115	8735	9115	8735	9115
路径 4	9215	9325	9115	8515	9115	8735	9115	8735
路径 5	9200	9200	8610	8990	8130	8890	8510	8890
路径 6	9090	9200	8990	8610	8890	8130	8890	8510
路径 7	9075	9075	8485	8865	8385	8765	7745	8665
路径 8	8965	9075	8865	8485	8765	8385	8665	7745

根据两名玩家的收益矩阵我们可以计算得出本次博弈过程的收益差值矩阵：

	路径 1	路径 2	路径 3	路径 4	路径 5	路径 6	路径 7	路径 8
路径 1	0	0	185	185	310	310	435	435
路径 2	0	0	185	185	310	310	435	435
路径 3	185	185	0	0	125	125	250	250
路径 4	185	185	0	0	125	125	250	250
路径 5	310	310	125	125	0	0	125	125
路径 6	310	310	125	125	0	0	125	125
路径 7	435	435	250	250	125	125	0	0
路径 8	435	435	250	250	125	125	0	0

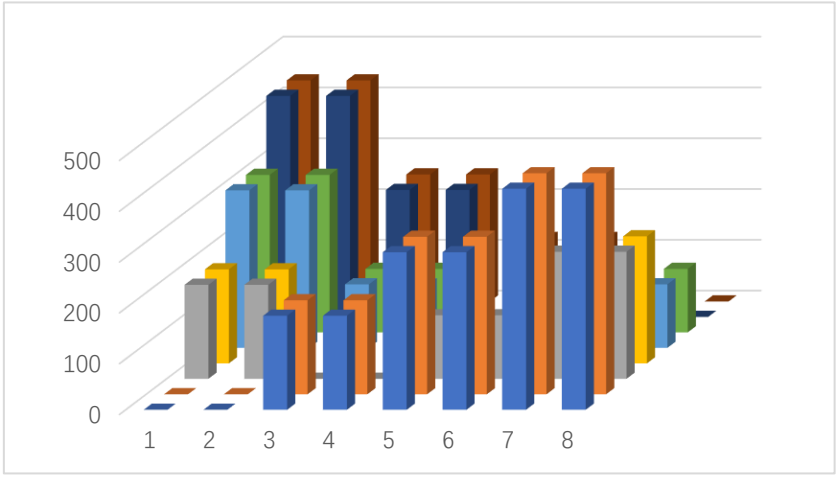

Fig.15 收益差值矩阵可视化

经过对数据的分析我们不难发现，纳什均衡点位于（路径 1，路径 2），（路径 2，路径 1），且总是在两名玩家选择相似的路径时差值最小。而在理论上我们无法通过纯策略达到这一纳什均衡点，因此需要借助混合策略来帮助分析。我们假设玩家 1 采用路径一，二的概率分别为c_1, c_2，玩家 2 采用路径一，二的概率分别为v_1, v_2,记$\boldsymbol{c} = (c_1, c_2), \boldsymbol{v} = (v_1, v_2)$，为两个行向量，且满足

$$0 \le c_i \le 1, \sum_{i=1}^{2} c_i = 1 \, , 0 \le v_i \le 1, \sum_{i=1}^{2} v_i = 1$$

则概率向量$\boldsymbol{c}$，$\boldsymbol{v}$分别构成了两名玩家的混合策略空间，记为S_1, S_2。我们用期望效用值来定义此时双方的效用函数：

$$U_1(\boldsymbol{c}, \boldsymbol{v}) = \sum_{i=1}^{2}\sum_{j=1}^{2} c_i m_{ij} q_j = \boldsymbol{c}\boldsymbol{M}\boldsymbol{v}^{\boldsymbol{T}}, U_2(\boldsymbol{c}, \boldsymbol{v}) = -U_1(\boldsymbol{c}, \boldsymbol{v})$$

则此时，双方面面临的决策问题分别为

$$\max_{\boldsymbol{c} \in S_1} U_1(\boldsymbol{c}, \boldsymbol{v})$$

$$\min_{\boldsymbol{v} \in S_2} U_1(\boldsymbol{c}, \boldsymbol{v})$$

由于双方都希望自己获得最大的效用，所以己方采用策略是得到的利益，往往是所有可能的利益中最小的那个，所以最优的策略应该是使得己方最小的效用达到最大。则我们可以将上述两式转化为

$$\max \min_{\boldsymbol{c} \in S_1} \boldsymbol{c}\boldsymbol{M}$$

$$\min \max_{\boldsymbol{v} \in S_2} \boldsymbol{M}\boldsymbol{v}^{\boldsymbol{T}}$$

求解得到

$$c = (0.5, 0.5)$$
$$v = (0.5, 0.5)$$

即玩家以等同的概率选择路径一、二中的任意一条均可以达到最大的期望效用。而这与我们的直观感受也是相符合的，虽然在玩家一二选择不同的路径时可以获得更高的收益，但是我们并无法提前获知对方的选择策略和选择偏好，因此只要双方按照相同且不设置偏好的概率进行选择，就有50%的概率达到最大的收益，这一数值在统计学上是要高于其他的情况的。

9.5 对关卡六的讨论

与（1）不同，在（2）中不需要提前设定行动方案，而是可以在每一天结束后都知道其他玩家的状态。这也就使整个博弈过程的共同知识的范围扩大。基于 Nash 原理，我们可以根据每天的实时数据来预测其他玩家的行为，而同样的，其他玩家也可以根据这部分信息来预测己方的行为。因此我们使用递推迭代的方式来动态地决策下一步的行为。

我们约定E_{ij}是第i天第j个玩家的期望收益函数，$\boldsymbol{\beta_{ij}}$是第i天第j个玩家基于问题二中给出的最优策略而预测选择的行动向量, 假设一共有 k 个玩家。则基于博弈论的基本原理，我们可以得到期望收益函数的递推式:

$$E_{ij} = E_{(i-1)j} + \Delta E - \sum_{p=1}^{k} \boldsymbol{\alpha_{ip}\beta_{ip}}$$

通过该表达式我们不难看出，在假设其余玩家均按照在问题二中给出的最优策略进行选择路径的假设下，己方在某一天的期望收益函数由ΔE唯一确定，而ΔE与己方玩家在当前局面的行动策略的选择有关，如果我们令己方玩家同样按照问题二中的最优策略进行选择，那么游戏的走向将趋于随机过程而无法逼近Nash 均衡点，那么我们就可以通过修正问题二中建立的模型来改变ΔE，从而改变己方玩家的决策，从而使全局游戏的走向逼近 Nash 均衡点。

我们在问题二中描述的矿山的期望值为：

$$E_0(w_t, n', d) = j_{10}w_t + j_{20}n' + j_{30}d$$

$$\text{其中} j_{20}, j_{30} > 0\ , j_{10} < 0$$

而由于加入问题三的规则后，我们在矿山的收入期望将会减少，而减少的幅度取决于矿山的数目以及距离因素，假设地图上有g座矿山。因此我们使用以下规则修正：

$$E_0'(w_t, n', d) = \frac{g}{k} E_0(w_t, n', d)$$

我们在问题二中描述的村庄的期望值为：

$$E_1(w_t, n', d) = j_{11}w_t + j_{21}n' + j_{31}d$$

$$\text{其中} j_{11}, j_{31} > 0\ , j_{21} < 0$$

由于在多人游戏的条件下，资源消耗的速度更快，所以基于生存性考虑，村庄的期望值应该增加，我们约定，在第i天之前，己方玩家分别与其他$(k-1)$名玩家有过γ次相遇因此我们用如下模型修正：

$$E_1'(w_t, n', d) = \gamma * j_{11}w_t + j_{21}n' + j_{31}d$$

我们在问题二中描述的终点的期望值为：

$$E_2(w_t, n', d) = j_{12}w_t + j_{22}n' + \frac{j_{32}}{d}$$

$$\text{其中} j_{12},\ j_{22},\ j_{32} > 0$$

在问题三的情形下，由于直接前往终点基本可以保证所消耗的资源最少，而同时由于矿山的收益减少，所以终点的期望值应该相应的变高，同时，由于前往终点与前往矿山是两个相反的路径选择，所以我们采用如下规则进行修正：

$$E_2'(w_t, n', d) = \frac{k}{g} E_2(w_t, n', d)$$

经过上述修正，我们在问题三（2）中做出正确的决策，只需要将期望收益函数进行修正，然后按照问题二中的流程进行决策即可。基于博弈论的理论，经过足够次数的迭代后，期望效益函数将会逼近于局面最优解。

10. 模型的分析检验

在问题二中，我们随机生成了一组天气去验证我们的模型。在这里，我们随机生成了 26 组天气，并将每组的天气情况放到第一问的条件中，计算得到了在已知天气情况时，我们能计算得到的全局最优解，并算出两者的差值：

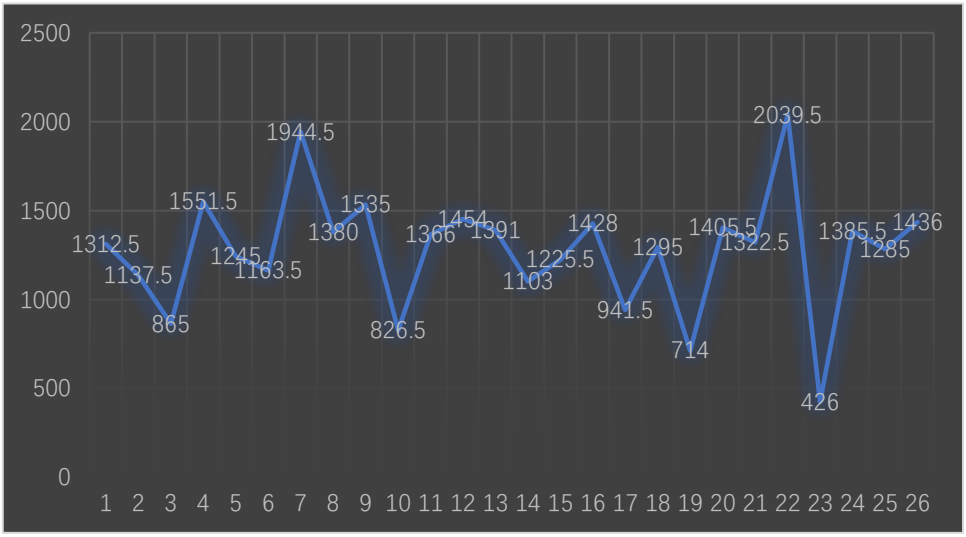

Fig.16 差值可视化

通过对比我们发现，基于我们的模型，在未知天气的情况下，与最优解相差1300左右，个别组别相差了将近2000元。这说明，在大多数的天气情况下，我们的模型能够较好地接近全局最优解，帮助玩家在游戏中进行正确的决策。而反常的几组数据，可能是由于天气排列的不均衡所导致的。由于在天气预测中，状态转移概率矩阵使用的是经验值，并没有将新的天气变化加入其中，因此会对模型的准确程度造成一定的影响。

11. 模型的评估与优化

11.1 模型的优缺点

我们认为本模型具有以下**优点**：

1. 借助矩阵这一数学工具简化了繁杂的数学运算。
2. 经过了大量的数据分析以后再进行决策模型的构建，使得决策模型更加全面。
3. 在对未知天气情况进行建模后将其与已知天气序列的情况做对比，说明了建模的合理性。
4. 将策略分为两部分，一个是玩家行走的路径，一个是在村庄和起点购买物资的配比，最大化的遵循了利益最大化和生存性原则。

同时，我们也认为该模型存在以下**不足**：

1. 在静态博弈模型中，没有针对参加游戏的人数大于10人的情况进行分析，并给出策略。
2. 问题二中的模型假设已知天气概率，而没有分析多种天气概率分布下的策略选择。
3. 模型受天气排列因素影响较大

11.2 模型的优化

我们后续还将通过这些工作进行模型改进：

1. 在问题二预测天气的过程中，我们使用马尔科夫链的模型进行预测，是基于今天发生的天气概率分布仅由昨天发生的天气决定，后续我们将改进模型，将过往天气的影响都考虑在内。

2. 本模型假设在高温天气下，行走的收益总是大于停留的收益，但事实上，存在这样一种情况，如果晴朗天气的食物消耗为f_1，高温天气的食物消耗为f_2,且有$(f_2 > 2 * f_1)$则在不考虑时间的因素的情况下，当高温天气的后一天是晴朗天气时，我们在高温天气选择行走将会造成更大的食物损耗，水的消耗同理。在本模型中没有将这种情况考虑在内，后续将会加入衡量当出现高温和晴朗天气连续出现时是否要选择行走的因子。

12. 参考文献

[1] 张长青. 运筹学的方法及应用[M]. 黑龙江科学技术出版社, 2010.
[2] 马尔可夫链在天气预测中的应用[J]. 梁爱凝. 价值工程. 2016(23)
[3] 刘克, 曹平. 马尔可夫决策过程理论与应用[M]. 科学出版社, 2015.
[4] FrankR.Giordano, 叶其孝, 姜启源. 数学建模:(原书第 3 版)[M]. 机械工业出版社, 2005.
[5] 纳什. 纳什博弈论论文集[M]. 首都经济贸易大学出版社, 2000.
[6] 吴广谋, 吕周洋. 博弈论基础与应用[M]. 东南大学出版社, 2009.

13. 附录

附录一 支撑材料文件列表

1、dfs.exe 深度优先搜索程序
2、problem3_1.exe 博弈过程求解源程序
3、第一关行动矩阵.xlsx
4、第一关损益矩阵.xlsx
5、第二关行动矩阵.xlsx
6、第二关损益矩阵.xlsx
7、深度优先搜索 第一组.xlsx
8、深度优先搜索 第二组.xlsx
9、深度优先搜索 第三组.xlsx
10、 深度优先搜索 第四组.xlsx
11、 Result.xlsx

附录二 第一关答案(result.xlsx)

第一关

日期	所在区域	剩余资金数	剩余水量	剩余食物量
0	1	5800	180	330
1	25	5800	164	318
2	24	5800	148	306
3	23	5800	138	292
4	23	5800	128	282
5	22	5800	118	268
6	9	5800	102	256
7	9	5800	92	246
8	15	4170	245	232
9	13	4170	229	220
10	12	4170	213	208
11	12	4170	203	198
12	12	5170	179	180
13	12	6170	164	159
14	12	7170	140	141
15	12	8170	116	123
16	12	9170	92	105
17	12	10170	62	75
18	12	11170	32	45
19	13	11170	16	33
20	15	10430	36	40
21	9	10430	26	26
22	21	10430	16	12

23	27	10430	0	0
24				
25				
26				
27				
28				
29				
30				

附录三 第二关答案(result.xlsx)

第二关

日期	所在区域	剩余资金数	剩余水量	剩余食物量
0	1	5840	184	324
1	2	5840	168	312
2	3	5840	152	300
3	4	5840	142	286
4	4	5840	132	276
5	5	5840	122	262
6	13	5840	106	250
7	13	5840	96	240
8	22	5840	86	226
9	30	5840	70	214
10	30	6840	46	196
11	30	7840	16	166
12	39	2660	186	320
13	30	2660	176	306
14	30	3660	152	288
15	30	4660	128	270
16	30	5660	104	252
17	30	6660	74	222
18	30	7660	44	192
19	30	8660	20	174
20	39	6570	175	181
21	46	6570	165	167
22	55	6570	155	153
23	55	7570	131	135
24	55	8570	116	114
25	55	9570	86	84
26	55	10570	62	66
27	55	11570	47	45
28	55	12570	32	24

29	63	12570	16	12
30	64	12570	0	0

附录四 第三关策略模拟结果

第一组：

日期	1	2	3	4	5	6	7	8	9	10
天气	晴朗	高温	晴朗	晴朗	晴朗	晴朗	高温	高温	高温	高温

结果：

天数	地点	资金	水	食物
0	1	7100	180	200
1	4	7100	170	186
2	3	7100	152	168
3	9	7100	142	154
4	9	7300	127	133
5	9	7500	112	112
6	9	7700	97	91
7	11	7700	79	73
8	13	8127.5	0	0

第二组：

日期	1	2	3	4	5	6	7	8	9	10
天气	晴朗	高温	晴朗	晴朗	高温	高温	晴朗	晴朗	高温	高温

结果：

天数	地点	资金	水	食物
0	1	7225	179	188
1	4	7225	169	174
2	3	7225	151	156
3	9	7225	141	142
4	9	7425	126	121
5	9	7625	99	94
6	11	7625	81	76
7	13	8112.5	0	0

第三组：

日期	1	2	3	4	5	6	7	8	9	10
天气	晴朗	高温	晴朗	高温	晴朗	高温	晴朗	高温	晴朗	高温

结果：

天数	地点	资金	水	食物
0	1	6350	230	250
1	4	6350	220	236
2	3	6350	202	218
3	9	6350	192	204
4	9	6550	165	177
5	9	6750	150	156
6	9	6950	123	129
7	9	7150	108	108

8	9	7350	81	81	
9	11	7350	71	67	
10	13	7727.5	0	0	

附录五 第四关策略模拟结果

第一组：

日期	1	2	3	4	5	6	7	8	9	10
天气	高温	高温	晴朗	晴朗	晴朗	高温	沙暴	晴朗	高温	高温
日期	11	12	13	14	15	16	17	18	19	20
天气	晴朗	高温	晴朗	高温	高温	高温	沙暴	沙暴	高温	高温
日期	21	22	23	24	25	26	27	28	29	30
天气	晴朗	晴朗	高温	晴朗	晴朗	高温	晴朗	晴朗	高温	高温

结果：

日期	地点	资金	水	食物
0	1	6400	240	240
1	2	6400	222	222
2	3	6400	204	204
3	4	6400	194	190
4	9	6400	184	176
5	14	4300	244	232
6	14	4300	226	214
7	14	4300	216	204
8	18	4300	206	190
9	18	5300	179	163
10	18	6300	152	136
11	18	7300	137	115
12	18	8300	110	88
13	18	9300	95	67
14	18	10300	68	40
15	13	10300	50	22
16	14	3730	251	223
17	14	3430	251	223
18	14	3130	251	223
19	13	3130	233	205
20	18	3130	215	187
21	18	4130	200	166
22	18	5130	185	145
23	18	6130	158	118
24	18	7130	143	97
25	18	8130	128	76
26	18	9130	101	49
27	19	9130	91	35
28	20	9130	81	21
29	25	9302.5	0	0

第二组：

日期	1	2	3	4	5	6	7	8	9	10
天气	晴朗	晴朗	高温	高温	晴朗	高温	沙暴	晴朗	高温	高温
日期	11	12	13	14	15	16	17	18	19	20
天气	晴朗	高温	高温	晴朗	高温	高温	沙暴	晴朗	高温	高温
日期	21	22	23	24	25	26	27	28	29	30
天气	晴朗	沙暴	高温	晴朗	晴朗	高温	高温	高温	晴朗	晴朗

结果：

天数	地点	资金	水	食物
0	1	6400	220	250
1	2	6400	210	236
2	3	6400	200	222
3	4	6400	182	204
4	9	6400	164	186
5	14	4300	224	242
6	19	4300	206	224
7	19	4300	196	214
8	18	4300	186	200
9	18	5300	159	173
10	18	6300	132	146
11	18	7300	117	125
12	18	8300	90	98
13	18	9300	63	71
14	23	9300	53	57
15	24	9300	35	39
16	25	9447.5	0	0

第三组：

日期	1	2	3	4	5	6	7	8	9	10
天气	高温	高温	沙暴	晴朗	高温	高温	沙暴	晴朗	高温	高温
日期	11	12	13	14	15	16	17	18	19	20
天气	晴朗	高温	高温	晴朗	晴朗	晴朗	高温	晴朗	高温	高温
日期	21	22	23	24	25	26	27	28	29	30
天气	晴朗	沙暴	晴朗	高温	晴朗	高温	高温	高温	晴朗	晴朗

结果：

天数	地点	资金	水	食物
0	1	6400	220	260
1	2	6400	202	242
2	3	6400	184	224
3	4	6400	174	214
4	9	6400	164	200
5	9	6400	146	182

6	14	3610	221	257
7	14	3310	221	257
8	13	3310	211	243
9	18	3310	193	225
10	18	4310	166	198
11	18	5310	151	177
12	18	6310	124	150
13	18	7310	97	123
14	18	8310	82	102
15	18	9310	67	81
16	18	10310	52	60
17	19	10310	34	42
18	14	4010	234	238
19	19	4010	216	220
20	18	4010	198	202
21	18	5010	183	181
22	18	6010	153	151
23	18	7010	138	130
24	18	8010	111	103
25	18	9010	96	82
26	23	9010	78	64
27	24	9010	60	46
28	25	9255	0	0

附录六 博弈过程求解源代码

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
int main()
{
    FILE *fp;
    fp = fopen("1.csv","w");
    int weather[10] = {0,1,0,0,0,0,1,1,1,1};
    int water[2] = {3,9};
    int food[2] = {4,9};
    int move[2][8][10] = {0};
    int area[2][8][10] = {0};
    int food_assume[2][8][8] = {0};
    int water_assume[2][8][8] = {0};
    int food_price = 10;
    int water_price = 5;
    int water_weight = 3;
    int food_weight = 2;
    int weight = 0;
```

```
    int cost[2][8][8] = {0};
    int earn[2][8][8] = {0};
    int money[2][8][8] = {0};
    //for(int i = 0; i < 2; i++)
    //{
        for(int j = 0; j < 8; j++)
        {
            for(int k = 0; k < 10; k++)
            {
                //int flag = 0;
                scanf("%d",&area[0][j][k]);
                area[1][j][k] = area[0][j][k] ;
                if(area[0][j][k] == -1)
                {
                    area[0][j][k] = 0;
                    area[1][j][k] = 0;
                    //flag = 1;
                    break;
                }
                /*if(flag)
                {
                    break;
                }*/
            }
        }
    //}
    //for(int i = 0; i < 2; i++)
    //{
        for(int j = 0; j < 8; j++)
        {
            for(int k = 0; k < 10; k++)
            {
                //int flag = 0;
                scanf("%d",&move[0][j][k]);
                move[1][j][k] = move[0][j][k];
                if(move[0][j][k] == -1)
                {
                    move[0][j][k] = 0;
                    move[1][j][k] = 0;
                    break;
                }
                /*if(flag)
                {
                    break;
```

```
                }*/
            }
        }
    //}
    for(int i = 0; i < 8; i++)
    {
        for(int j = 0; j < 8; j++)
        {
            for(int k = 0; k < 10; k++)
            {
                food_assume[0][i][j] += food[weather[k]]*move[0][i][k];

                food_assume[1][i][j] += food[weather[k]]*move[1][j][k];

                water_assume[1][i][j] += water[weather[k]]*move[1][j][k]
;
                water_assume[0][i][j] += water[weather[k]]*move[0][i][k]
;
                if(move[0][i][k] == 2&&move[1][j][k] == 2&&area[0][i][k]
 == area[1][j][k]&&area[0][i][k+1] == area[1][j][k+1])
                {
                    food_assume[0][i][j] += food[weather[k]]*move[0][i][
k];
                    food_assume[1][i][j] += food[weather[k]]*move[1][j][
k];
                    water_assume[1][i][j] += water[weather[k]]*move[1][j
][k];
                    water_assume[0][i][j] += water[weather[k]]*move[0][i
][k];
                }
            }
            for(int k = 0; k < 2; k++)
            {
                cost[k][i][j] = food_assume[k][i][j]*food_price + water_
assume[k][i][j]*water_price;
                weight = food_assume[k][i][j]*food_weight + water_assume
[k][i][j]*water_weight;
                if(cost[k][i][j] > 10000||weight > 1200)
                {
                    money[k][i][j] = -1;
                }
            }
        }
    }
```

```
97.     for(int i = 0; i < 8; i++)
98.     {
99.         for(int j = 0; j < 8; j++)
100.         {
101.             for(int k = 0; k < 10; k++)
102.             {
103.                 /*int flag1 = 0;
104.                 int flag2 = 0;*/
105.                 if(move[0][i][k] == 3&&move[1][j][k] == 3)
106.                 {
107.                     earn[0][i][j] += 100;
108.                     earn[1][i][j] += 100;
109.                 }
110.                 else if(move[0][i][k] == 3&&move[1][j][k] != 3)
111.                 {
112.                     earn[0][i][j] += 200;
113.                 }
114.                 else if(move[0][i][k] != 3&&move[1][j][k] == 3)
115.                 {
116.                     earn[1][i][j] += 200;
117.                 }
118.             }
119.         }
120.     }
121.     for(int k = 0;k < 2;k++)
122.     {
123.         for(int i = 0;i < 8;i++)
124.         {
125.             for(int j = 0;j < 8;j++)
126.             {
127.                 money[k][i][j] = 10000 + earn[k][i][j] - cost[k][i][j];

128.             }
129.         }
130.     }
131.     for(int k = 0;k < 2;k++)
132.     {
133.         for(int i = 0;i < 8;i++)
134.         {
135.             for(int j = 0;j < 8;j++)
136.             {
137.                 printf("%d ",money[k][i][j]);
138.                 fprintf(fp,"%d,",money[k][i][j]);
139.             }
```

```
            printf("\n");
            fprintf(fp,"\n");
        }
        printf("\n\n");
        fprintf(fp,"\n\n");
    }
    for(int i = 0;i < 8;i++)
        {
            for(int j = 0;j < 8;j++)
            {
                printf("%6d",abs(money[0][i][j]-money[1][i][j]));
                fprintf(fp,"%d,",abs(money[0][i][j]-money[1][i][j]));
            }
            printf("\n");
            fprintf(fp,"\n");
        }
return 0;
}
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
int main()
{
    FILE *fp;
    fp = fopen("1.csv","w");
    int weather[10] = {0,1,0,0,0,0,1,1,1,1};
    int water[2] = {3,9};
    int food[2] = {4,9};
    int move[2][8][10] = {0};
    int area[2][8][10] = {0};
    int food_assume[2][8][8] = {0};
    int water_assume[2][8][8] = {0};
    int food_price = 10;
    int water_price = 5;
    int water_weight = 3;
    int food_weight = 2;
    int weight = 0;
    int cost[2][8][8] = {0};
    int earn[2][8][8] = {0};
    int money[2][8][8] = {0};
    //for(int i = 0; i < 2; i++)
    //{
        for(int j = 0; j < 8; j++)
        {
```

```
            for(int k = 0; k < 10; k++)
            {
                //int flag = 0;
                scanf("%d",&area[0][j][k]);
                area[1][j][k] = area[0][j][k] ;
                if(area[0][j][k] == -1)
                {
                    area[0][j][k] = 0;
                    area[1][j][k] = 0;
                    //flag = 1;
                    break;
                }
                /*if(flag)
                {
                    break;
                }*/
            }
        }
    //}
    //for(int i = 0; i < 2; i++)
    //{
        for(int j = 0; j < 8; j++)
        {
            for(int k = 0; k < 10; k++)
            {
                //int flag = 0;
                scanf("%d",&move[0][j][k]);
                move[1][j][k] = move[0][j][k];
                if(move[0][j][k] == -1)
                {
                    move[0][j][k] = 0;
                    move[1][j][k] = 0;
                    break;
                }
                /*if(flag)
                {
                    break;
                }*/
            }
        }
    //}
    for(int i = 0; i < 8; i++)
    {
        for(int j = 0; j < 8; j++)
```

```
228.         {
229.             for(int k = 0; k < 10; k++)
230.             {
231.                 food_assume[0][i][j] += food[weather[k]]*move[0][i][k];
232.                 food_assume[1][i][j] += food[weather[k]]*move[1][j][k];
233.                 water_assume[1][i][j] += water[weather[k]]*move[1][j][k
    ];
234.                 water_assume[0][i][j] += water[weather[k]]*move[0][i][k
    ];
235.                 if(move[0][i][k] == 2&&move[1][j][k] == 2&&area[0][i][k
    ] == area[1][j][k]&&area[0][i][k+1] == area[1][j][k+1])
236.                 {
237.                     food_assume[0][i][j] += food[weather[k]]*move[0][i]
    [k];
238.                     food_assume[1][i][j] += food[weather[k]]*move[1][j]
    [k];
239.                     water_assume[1][i][j] += water[weather[k]]*move[1][
    j][k];
240.                     water_assume[0][i][j] += water[weather[k]]*move[0][
    i][k];
241.                 }
242.             }
243.             for(int k = 0; k < 2; k++)
244.             {
245.                 cost[k][i][j] = food_assume[k][i][j]*food_price + water
    _assume[k][i][j]*water_price;
246.                 weight = food_assume[k][i][j]*food_weight + water_assum
    e[k][i][j]*water_weight;
247.                 if(cost[k][i][j] > 10000||weight > 1200)
248.                 {
249.                     money[k][i][j] = -1;
250.                 }
251.             }
252.         }
253.     }
254.     for(int i = 0; i < 8; i++)
255.     {
256.         for(int j = 0; j < 8; j++)
257.         {
258.             for(int k = 0; k < 10; k++)
259.             {
260.                 /*int flag1 = 0;
```

```
                int flag2 = 0;*/
                if(move[0][i][k] == 3&&move[1][j][k] == 3)
                {
                    earn[0][i][j] += 100;
                    earn[1][i][j] += 100;
                }
                else if(move[0][i][k] == 3&&move[1][j][k] != 3)
                {
                    earn[0][i][j] += 200;
                }
                else if(move[0][i][k] != 3&&move[1][j][k] == 3)
                {
                    earn[1][i][j] += 200;
                }
            }
        }
    }
    for(int k = 0;k < 2;k++)
    {
        for(int i = 0;i < 8;i++)
        {
            for(int j = 0;j < 8;j++)
            {
                money[k][i][j] = 10000 + earn[k][i][j] - cost[k][i][j];

            }
        }
    }
    for(int k = 0;k < 2;k++)
    {
        for(int i = 0;i < 8;i++)
        {
            for(int j = 0;j < 8;j++)
            {
                printf("%d ",money[k][i][j]);
                fprintf(fp,"%d,",money[k][i][j]);
            }
            printf("\n");
            fprintf(fp,"\n");
        }
        printf("\n\n");
        fprintf(fp,"\n\n");
    }
    for(int i = 0;i < 8;i++)
```

```
        {
            for(int j = 0;j < 8;j++)
            {
                printf("%6d",abs(money[0][i][j]-money[1][i][j]));
                fprintf(fp,"%d,",abs(money[0][i][j]-money[1][i][j]));
            }
            printf("\n");
            fprintf(fp,"\n");
        }
return 0;
}
```

附录七 深度优先搜索 C 语言源代码

```
#include <iostream>

#define WATER_WEIGHT 3
#define FOOD_WEIGHT 2
#define WATER_PRICE 5
#define FOOD_PRICE 10

#define SUN_WATER 5
#define SUN_FOOD 7
#define HEIGH_WATER 8
#define HEIGH_FOOD 6
#define SEND_WATER 10
#define SEND_FOOD 10

#define BASIC_INCOME 1000
#define MAX_WEIGHT 1200

#define SUN 0
#define HEIGH 1
#define SEND 2

int map[100][100] = { 0 };
int book[100] = { 0 };
int isMine[100] = { 0 };
int isVillage[100] = { 0 };
int weather[40] = { 0 };
int max_points = 0;
int end = 0;
int way[100] = { 0 };
int status[100] = { 0 };
```

```
int count = 0;

int max_money = 0;
int k = 1;

int i_t, j_t;

FILE *pf;

void dp(int point, int day, int flag);
void minus(int mode, int times);
void add(int mode, int times);
void player_init();
int go(int start, int end, int day);
int back(int start, int end, int day);

void output(FILE *p, int sum, int day);
void minus_2(int mode, int times);
void add_2(int mode, int times);


struct Player
{
    int water, food;
    int money, weight;
    int villageInfo[100][10];
    int top;
    int income;
} player, player_2;

int main()
{
    int max = 0, max_index[2];

    fopen_s(&pf, "1.csv", "w");
    printf("weather:\n");
    for (int i = 1; i <= 30; i++)
    {
        scanf_s("%d", &weather[i]);
    }

    printf("n: \n");
    scanf_s("%d", &max_points);

```

```
75.     printf("map:\n");
76.     while (1)
77.     {
78.         int t1, t2, t3;
79.         scanf_s("%d%d%d", &t1, &t2, &t3);
80.         if (t1 == 0 && t2 == 0)
81.             break;
82.         else
83.         {
84.             map[t1][t2] = t3;
85.             map[t2][t1] = t3;
86.         }
87.     }
88.
89.     printf("out :\n");
90.     scanf_s("%d", &end);
91.
92.     printf("mine: \n");
93.     while (1)
94.     {
95.         int t;
96.         scanf_s("%d", &t);
97.         if (t != 0)
98.             isMine[t] = 1;
99.         else
100.             break;
101.     }
102.
103.     printf("village: \n");
104.     while (1)
105.     {
106.         int t;
107.         scanf_s("%d", &t);
108.         if (t != 0)
109.             isVillage[t] = 1;
110.         else
111.             break;
112.     }
113.
114.     player.money = 10000;
115.
116.     for (int i = 1; i <= player.money / WATER_PRICE; i++)
117.     {
118.         for (int j = 1; j <= player.money / FOOD_PRICE; j++)
```

```
119.         {
120.             i_t = i;
121.             j_t = j;
122.             player_init();
123.             if (i * WATER_PRICE + j * FOOD_PRICE <= player.money)
124.             {
125.                 max_money = 0;
126.                 player.water = i;
127.                 player.food = j;
128.                 player.money -= (WATER_PRICE * i + FOOD_PRICE * j);
129.                 player.weight = i * WATER_WEIGHT + j * FOOD_WEIGHT;
130.                 if (player.weight == MAX_WEIGHT)
131.                 {
132.                     for (k = 0; k < 10; k++)
133.                     {
134.                         max_money = 0;
135.                         dp(1, 0, 0);
136.                         printf("( i = %d , j = %d ,k = %d)    max = %d\
    n", i, j, k, max_money);
137.                         if (max_money > max)
138.                         {
139.                             max = max_money;
140.                             max_index[0] = i;
141.                             max_index[1] = j;
142.                         }
143.                     }
144.                 }
145.             }
146.         }
147.     }
148.     printf("\n\nmax = %d, i = %d, j = %d\n", max, max_index[0], max_ind
    ex[1]);
149.
150.
151.     /*
152.     int i = 192;
153.     int j = 312;
154.     i_t = i;
155.     j_t = j;
156.     player.top = 0;
157.     player.water = i;
158.     player.food = j;
159.     player.money -= (WATER_PRICE * i + FOOD_PRICE * j);
160.     player.weight = i * WATER_WEIGHT + j * FOOD_WEIGHT;
```

```
161.     dp(1, 0, 0);
162.     printf("\nmax = %d, count = %d", max_money, count);*/
163.     return 0;
164. }
165.
166. void dp(int point, int day, int flag)
167. {
168.     if (player.food < 0 || player.water < 0)
169.         return;
170.     if (point == end && day <= 30)
171.     {
172.         count++;
173.
174.         int sum_food = 0, sum_water = 0, sum_money = player.money;
175.         for (int i = 0; i < player.top; i++)
176.         {
177.             sum_water += player.villageInfo[player.top][8];
178.             sum_food += player.villageInfo[player.top][9];
179.         }
180.         if (player.food - sum_food > 0)
181.         {
182.             sum_money += sum_food * FOOD_PRICE * 2;
183.             sum_money += (player.food - sum_food) * FOOD_PRICE;
184.         }
185.         else
186.         {
187.             sum_money += player.food * FOOD_PRICE * 2;
188.         }
189.         if (player.water - sum_water > 0)
190.         {
191.             sum_money += sum_water * WATER_PRICE * 2;
192.             sum_money += (player.water - sum_water) * WATER_PRICE;
193.         }
194.         else
195.         {
196.             sum_money += player.water * WATER_PRICE * 2;
197.         }
198.
199.         if (sum_money > max_money)
200.             max_money = sum_money;
201.         way[day] = point;
202.         /*
203.         if (sum_money == max_money)
204.         {
```

```
205.
206.             printf("(%d) ", count);
207.             for (int i = 0; i <= day; i++)
208.             {
209.                 printf("%d -> ", way[i]);
210.             }
211.             printf("\n    ");
212.             for (int i = 0; i <= day; i++)
213.             {
214.                 printf("%d -> ", status[i]);
215.             }
216.             printf("\nwater = %d", player.water);
217.             printf("\nfood = %d", player.food);
218.             printf("\nmoney = %d     sum = %d", player.money, sum_money
    );
219.             printf("\nincome = %d", player.income);
220.             printf("\nday = %d", day);
221.             for (int i = 0; i < player.top; i++)
222.             {
223.                 printf("\nvillage %d : food %d    water %d     n = %d
     kn = %d", i, player.villageInfo[i][5], player.villageInfo[i][4], player
    .villageInfo[i][8], player.villageInfo[i][9]);
224.             }
225.             printf("\n");
226.         }
227.         if (count == 137)
228.         {
229.             fprintf(pf, "%d\n", count);
230.             output(pf, sum_money, day);
231.         }*/
232.         way[day] = 0;
233.         return;
234.     }
235.     if (day >= 30)
236.         return;
237.
238.     book[point] = 1;
239.     way[day] = point;
240.     //mine
241.
242.     if (isMine[point] == 1)
243.     {
244.         status[day] = 3;
245.         minus(weather[day + 1], 3);
```

```
246.         player.money += BASIC_INCOME;
247.         player.income += BASIC_INCOME;
248.         dp(point, day + 1, 1);
249.         player.money -= BASIC_INCOME;
250.         player.income -= BASIC_INCOME;
251.         add(weather[day + 1], 3);
252.         status[day] = 0;
253.     }
254.
255.     //village
256.     if (isVillage[point] == 1)
257.     {
258.         int n;
259.         player.villageInfo[player.top][0] = player.water;
260.         player.villageInfo[player.top][1] = player.food;
261.         player.villageInfo[player.top][2] = player.money;
262.         player.villageInfo[player.top][3] = player.weight;
263.         if (player.money / (WATER_PRICE + k * FOOD_PRICE) / 2 < (MAX_WE
    IGHT - player.weight) / (WATER_WEIGHT + k * FOOD_WEIGHT))
264.             n = player.money / (WATER_PRICE + k * FOOD_PRICE) / 2;
265.         else
266.             n = (MAX_WEIGHT - player.weight) / (WATER_WEIGHT + k * FOOD
    _WEIGHT);
267.         player.villageInfo[player.top][4] = player.water += n;
268.         player.villageInfo[player.top][5] = player.food += k * n;
269.         player.villageInfo[player.top][6] = player.money -
    = (WATER_PRICE + k * FOOD_PRICE) * n * 2;
270.         player.villageInfo[player.top][7] = player.weight += (WATER_WEI
    GHT + k * FOOD_WEIGHT) * n;
271.         player.villageInfo[player.top][8] = n;
272.         player.villageInfo[player.top][9] = k * n;
273.
274.         player.top++;
275.         if (weather[day + 1] == SEND)
276.         {
277.             //stop
278.             status[day] = 1;
279.             minus(weather[day + 1], 1);
280.             dp(point, day + 1, 1);
281.             add(weather[day + 1], 1);
282.             status[day] = 0;
283.         }
284.         else
285.         {
```

```
            //go
            for (int i = 1; i <= max_points; i++)
            {
                //if (book[i] == 0 && map[point][i] != 0)
                if (map[point][i] != 0)
                {
                    status[day] = -1;
                    //minus(weather[day+1], 2);
                    day = go(point, i, day);
                    dp(i, day, 0);
                    //add(weather[day+1], 2);
                    day = back(point, i, day);
                    status[day] = 0;
                }
            }
        }
        player.top--;
        player.water -= n;
        player.food -= k * n;
        player.money += (WATER_PRICE + k * FOOD_PRICE) * n * 2;
        player.weight -= (WATER_WEIGHT + k * FOOD_WEIGHT) * n;
    }
    else
    {
        if (weather[day + 1] == SEND)
        {
            //stop
            status[day] = 1;
            minus(weather[day + 1], 1);
            dp(point, day + 1, 1);
            add(weather[day + 1], 1);
            status[day] = 0;
        }
        else
        {
            //go
            for (int i = 1; i <= max_points; i++)
            {
                //if (book[i] == 0 && map[point][i] != 0)
                if (map[point][i] != 0)
                {
                    status[day] = -1;
                    //minus(weather[day + 1], 2);
                    day = go(point, i, day);
```

```
                    dp(i, day, 0);
                    //add(weather[day + 1], 2);
                    day = back(point, i, day);
                    status[day] = -1;
                }
            }
        }
    }
    way[day] = 0;

    if (flag == 0)
        book[point] = 0;
    return;
}

void minus(int mode, int times)
{
    if (mode == SUN)
    {
        player.food -= times * SUN_FOOD;
        player.water -= times * SUN_WATER;
        player.weight -= times * FOOD_WEIGHT * SUN_FOOD;
        player.weight -= times * WATER_WEIGHT * SUN_WATER;
    }
    else if (mode == HEIGH)
    {
        player.food -= times * HEIGH_FOOD;
        player.water -= times * HEIGH_WATER;
        player.weight -= times * FOOD_WEIGHT * HEIGH_FOOD;
        player.weight -= times * WATER_WEIGHT * HEIGH_WATER;
    }
    else if (mode == SEND)
    {
        player.food -= times * SEND_FOOD;
        player.water -= times * SEND_WATER;
        player.weight -= times * FOOD_WEIGHT * SEND_FOOD;
        player.weight -= times * WATER_WEIGHT * SEND_WATER;
    }
}

void add(int mode, int times)
{
    if (mode == SUN)
    {
```

```
374.         player.food += times * SUN_FOOD;
375.         player.water += times * SUN_WATER;
376.         player.weight += times * FOOD_WEIGHT * SUN_FOOD;
377.         player.weight += times * WATER_WEIGHT * SUN_WATER;
378.     }
379.     else if (mode == HEIGH)
380.     {
381.         player.food += times * HEIGH_FOOD;
382.         player.water += times * HEIGH_WATER;
383.         player.weight += times * FOOD_WEIGHT * HEIGH_FOOD;
384.         player.weight += times * WATER_WEIGHT * HEIGH_WATER;
385.     }
386.     else if (mode == SEND)
387.     {
388.         player.food += times * SEND_FOOD;
389.         player.water += times * SEND_WATER;
390.         player.weight += times * FOOD_WEIGHT * SEND_FOOD;
391.         player.weight += times * WATER_WEIGHT * SEND_WATER;
392.     }
393. }
394.
395. void player_init()
396. {
397.     player.water = 0;
398.     player.food = 0;
399.     player.money = 10000;
400.     player.weight = 0;
401.     player.income = 0;
402.     player.top = 0;
403. }
404.
405. int go(int start, int end, int day)
406. {
407.     for (int i = 0; i < map[start][end]; i++)
408.     {
409.         if (weather[day + 1] == SEND)
410.         {
411.             //stop
412.             minus(weather[day + 1], 1);
413.             i--;
414.         }
415.         else
416.         {
417.             //go
```

```
418.             minus(weather[day + 1], 2);
419.         }
420.         day++;
421.     }
422.     return day;
423. }
424.
425. int back(int start, int end, int day)
426. {
427.     for (int i = 0; i < map[start][end]; i++)
428.     {
429.         day--;
430.         if (weather[day + 1] == SEND)
431.         {
432.             //stop
433.             add(weather[day + 1], 1);
434.             i--;
435.         }
436.         else
437.         {
438.             //go
439.             add(weather[day + 1], 2);
440.         }
441.     }
442.     return day;
443. }
444.
445. void output(FILE *p, int sum, int day)
446. {
447.     player_2.money = 10000;
448.     int i = i_t;
449.     int j = j_t;
450.     player_2.top = 0;
451.     player_2.water = i;
452.     player_2.food = j;
453.     player_2.money -= (WATER_PRICE * i + FOOD_PRICE * j);
454.     player_2.weight = i * WATER_WEIGHT + j * FOOD_WEIGHT;
455.     for (int i = 0; i <= day; i++)
456.     {
457.         if (i != 0)
458.         {
459.             if (isMine[way[i]])
460.             {
461.                 if (way[i - 1] == way[i])
```

```
462.                 {
463.                     minus_2(weather[i], 3);
464.                     player_2.money += BASIC_INCOME;
465.                 }
466.                 else if (way[i + 1] == way[i])
467.                 {
468.                     minus_2(weather[i], 2);
469.                 }
470.                 else
471.                 {
472.                     if (weather[i] == SEND)
473.                         minus_2(SEND, 1);
474.                     else
475.                         minus_2(weather[i], 2);
476.                 }
477.
478.             }
479.             else if (isVillage[way[i]])
480.             {
481.                 if (weather[i] == SEND)
482.                 {
483.                     minus_2(SEND, 1);
484.                 }
485.                 else
486.                     minus_2(weather[i], 2);
487.                 int n;
488.                 player_2.villageInfo[player_2.top][0] = player_2.water;
489.                 player_2.villageInfo[player_2.top][1] = player_2.food;
490.                 player_2.villageInfo[player_2.top][2] = player_2.money;
491.                 player_2.villageInfo[player_2.top][3] = player_2.weight
    ;
492.                 if (player_2.money / (WATER_PRICE + k * FOOD_PRICE) / 2
     < (MAX_WEIGHT - player_2.weight) / (WATER_WEIGHT + k * FOOD_WEIGHT))
493.                     n = player_2.money / (WATER_PRICE + k * FOOD_PRICE)
     / 2;
494.                 else
495.                     n = (MAX_WEIGHT - player_2.weight) / (WATER_WEIGHT
    + k * FOOD_WEIGHT);
496.                 player_2.villageInfo[player_2.top][4] = player_2.water
    += n;
```

```
497.                 player_2.villageInfo[player_2.top][5] = player_2.food +
    = k * n;
498.                 player_2.villageInfo[player_2.top][6] = player_2.money
    -= (WATER_PRICE + k * FOOD_PRICE) * n * 2;
499.                 player_2.villageInfo[player_2.top][7] = player_2.weight
     += (WATER_WEIGHT + k * FOOD_WEIGHT) * n;
500.                 player_2.villageInfo[player_2.top][8] = n;
501.                 player_2.villageInfo[player_2.top][9] = k * n;
502.                 player_2.top++;
503.                 fprintf(p, "buy,water,%d,food,%d\n", n, k * n);
504.             }
505.             else
506.             {
507.                 if (weather[i] == SEND)
508.                 {
509.                     minus_2(SEND, 1);
510.                 }
511.                 else
512.                     minus_2(weather[i], 2);
513.             }
514.         }
515.         fprintf(p, "%d,", i);
516.         fprintf(p, "%d,", way[i]);
517.         fprintf(p, "%d,", status[i]);
518.         fprintf(p, "%d,", player_2.money);
519.         fprintf(p, "%d,", player_2.water);
520.         fprintf(p, "%d,", player_2.food);
521.         fprintf(p, "\n");
522.     }
523.     fprintf(p, ",,sum,%d", sum);
524.     fprintf(p, "\n");
525.     fprintf(p, "\n");
526.     fprintf(p, "\n");
527. }
528.
529. void minus_2(int mode, int times)
530. {
531.     if (mode == SUN)
532.     {
533.         player_2.food -= times * SUN_FOOD;
534.         player_2.water -= times * SUN_WATER;
535.         player_2.weight -= times * FOOD_WEIGHT * SUN_FOOD;
536.         player_2.weight -= times * WATER_WEIGHT * SUN_WATER;
537.     }
```

```
538.     else if (mode == HEIGH)
539.     {
540.         player_2.food -= times * HEIGH_FOOD;
541.         player_2.water -= times * HEIGH_WATER;
542.         player_2.weight -= times * FOOD_WEIGHT * HEIGH_FOOD;
543.         player_2.weight -= times * WATER_WEIGHT * HEIGH_WATER;
544.     }
545.     else if (mode == SEND)
546.     {
547.         player_2.food -= times * SEND_FOOD;
548.         player_2.water -= times * SEND_WATER;
549.         player_2.weight -= times * FOOD_WEIGHT * SEND_FOOD;
550.         player_2.weight -= times * WATER_WEIGHT * SEND_WATER;
551.     }
552. }
553.
554. void add_2(int mode, int times)
555. {
556.     if (mode == SUN)
557.     {
558.         player_2.food += times * SUN_FOOD;
559.         player_2.water += times * SUN_WATER;
560.         player_2.weight += times * FOOD_WEIGHT * SUN_FOOD;
561.         player_2.weight += times * WATER_WEIGHT * SUN_WATER;
562.     }
563.     else if (mode == HEIGH)
564.     {
565.         player_2.food += times * HEIGH_FOOD;
566.         player_2.water += times * HEIGH_WATER;
567.         player_2.weight += times * FOOD_WEIGHT * HEIGH_FOOD;
568.         player_2.weight += times * WATER_WEIGHT * HEIGH_WATER;
569.     }
570.     else if (mode == SEND)
571.     {
572.         player_2.food += times * SEND_FOOD;
573.         player_2.water += times * SEND_WATER;
574.         player_2.weight += times * FOOD_WEIGHT * SEND_FOOD;
575.         player_2.weight += times * WATER_WEIGHT * SEND_WATER;
576.     }
577. }
```