穿越沙漠的优化策略探索

摘要

本文从规划角度出发，基于图论的模型研究了“穿越沙漠”小游戏在不同给定条件下的策略选择。由于初始地图信息较多，本文首先将附件中地图抽象由成关键点（起点、矿山、村庄、终点）构成的的无向图 $\mathcal{G}$，关键点的距离由在原始包含全部区域的无向图中使用多源最短路径算法得到，从起点到终点的路径可以使用深度优先算法全部获得。

针对问题一的一般情况，本文在不同的到达时间 T 的情况下，以总挖矿时间 d_m 为决策变量，以在终点时的总资金 G 为目标函数，以每个点的负载都不能超过负载上限、终点的物资（水、食物）不剩余、在特定的天气的影响下到达时间仍为 T 为约束条件。根据此规划模型，本文将 $\mathcal{G}$ 简化为仅包含起点、抽象的矿山-村庄集合体、终点构成的有向图 $\mathcal{G}'$ 。此模型最终得到了基础收益 g、估测的日均成本 $p_wC_w + p_fC_f$ 和到达时间 T 对 d_m 的决定性限制，获得了一般情况下的最优策略。

针对第一关和第二关，本文通过计算机模拟的方法来获取全局最优解，并给出算法流程图与分析。经检验，两关的最优解均符合问题一规划得到的一般情况下的最优策略，最终资金数分别为 10430 和 12730 元。

针对问题二的一般情况，本文从问题一的全局规划模型中获得对决策描述的启发，并结合天气未知的题设，建立状态机 $\{t, w_t, f_t, info(D)\}$ 来描述玩家在游戏中的时间、自身状态和地图信息。以剩余行程中的挖矿时间 d_{m_t} 为决策变量，以剩余行程中的总成本期望为目标函数，以状态机中的信息及假设作为约束条件，建立规划模型。该模型最后获得了基础收益、日均成本期望和前往村庄的路径增量对 d_{m_t} 的决定性限制，籍此指导玩家在一般情况下的最优策略。另外，本文利用概率统计模型对天气分布进行了描绘，进而获得成本期望，综合问题一的全局优化模型决定初始购买量的最优决策。

针对第三关和第四关，本文通过对问题二模型的推演和定性分析，对于第三关给出了玩家最佳策略：不挖矿直接前往终点；针对第四关，给出玩家最优策略的指导变量 ΔD_t，即地图信息，根据玩家状态的不同带来的 ΔD_t 的不同情况，本文给出了玩家不同的优化策略。

针对问题三，我们采用了纳什均衡的模型，利用问题一、二的模型对策略空间进行了缩小，最终可以获得纳什均衡的指导优化策略组合，并对关卡五、六进行了具体的分析和讨论。

最后，本文分析了模型的优缺点，讨论了通过引入动态规划的方法来改进计算机模拟和模型建立的方案，并就此对模型进行了简单推广。

关键字： 图论　整数规划　计算机模拟　纳什均衡

一、 问题重述

1.1 问题背景

“穿越沙漠”是一类有丰富策略性的游戏。在游戏中，每一关有给定的地图。玩家首先要利用初始资金购买一定数量的水和食物。然后从起点出发，在沙漠中行走。途中会遇到不同的天气，这会影响水与食物的消耗量。矿山、村庄是玩家可以补充资金和资源的地点。玩家的目标是在规定时间内到达终点，并保留尽可能多的资金。

1.2 游戏规则

问题中，游戏的基本规则如下：

1. 以天为基本时间单位，第 0 天为游戏的开始时间，此时玩家位于起点。玩家必须在截止日期或这之前到达终点，到达终点后该玩家的游戏结束。
2. 穿越沙漠需水和食物两种资源，它们的最小计量单位均为箱。每天玩家拥有的水和食物质量之和不能超过负重上限。若水或食物在到达终点前耗尽，视为游戏失败。
3. 每天的天气共有三种状况：“晴朗”、“高温”、“沙暴”，沙漠中所有区域的天气相同。
4. 每天玩家可从地图中的某个区域到达与之相邻的另一个区域（地图中有公共边界的两个区域称为相邻，仅有公共顶点而没有公共边界的两个区域为不相邻），也可在原地停留。沙暴日必须在原地停留。
5. 玩家在原地停留一天消耗的资源数量称为基础消耗量，行走一天消耗的资源数量为基础消耗量的 2 倍。
6. 玩家第 0 天可在起点处用初始资金以基准价格购买水和食物。玩家可在起点停留或回到起点，但不能多次在起点购买资源。玩家到达终点后可退回剩余的水和食物，每箱退回价格为基准价格的一半。
7. 玩家在矿山停留时，可通过挖矿获得资金，挖矿一天获得的资金量称为基础收益。如果挖矿，消耗的资源数量为基础消耗量的倍；如果不挖矿，消耗的资源数量为基础消耗量。到达矿山当天不能挖矿。沙暴日也可挖矿。
8. 玩家经过或在村庄停留时可用剩余的初始资金或挖矿获得的资金随时购买水和食物，每箱价格为基准价格的 2 倍。

1.3 问题提出

根据游戏的背景与基本规则，需要建立数学模型解决如下问题：

1. 假设只有一名玩家，在整个游戏时段内每天天气状况事先全部已知，试给出一般情况下玩家的最优策略。求解附件中的“第一关”和“第二关”，并将相应结果分别填入附件中的 Result.xlsx。
2. 假设只有一名玩家，玩家仅知道当天的天气状况，可据此决定当天的行动方案，试给出一般情况下玩家的最佳策略，并对附件中的“第三关”和“第四关”进行具体讨论。
3. 现有 n 名玩家，他们初始资金相同，且同时从起点出发。若某天其中的任意 k 名玩家均从区域 A 行走到区域 B($2 \leqslant k \leqslant n$)，则他们中的每一位消耗的资源数量均为基础消耗量的 $2k$ 倍；若某天其中的任意 $k(2 \leqslant k \leqslant n)$ 名玩家在同一矿山挖矿，则他们中的每一位消耗的资源数量均为基础消耗量的 3 倍，且每名玩家一天可通过挖矿获得的资金是基础收益的 $\frac{1}{k}$；若某天其中的任意 $k(2 \leqslant k \leqslant n)$ 名玩家在同一村庄购买资源，每箱价格均为基准价格的 4 倍。其他情况下消耗资源数量与资源价格与单人游戏相同。

   (a) 假设在整个游戏时段内每天天气状况事先全部已知，每名玩家的行动方案需在第 0 天确定且此后不能更改。试给出一般情况下玩家应采取的策略，并对附件中的“第五关”进行具体讨论。

   (b) 假设所有玩家仅知道当天的天气状况，从第 1 天起，每名玩家在当天行动结束后均知道其余玩家当天的行动方案和剩余的资源数量，随后确定各自第二天的行动方案。试给出一般情况下玩家应采取的策略，并对附件中的“第六关”进行具体讨论。

二、 问题分析

2.1 问题一的分析

问题一要求给出一名玩家在游戏时段内已知每天天气状况的情况下，在一般情况下的最优策略，并进一步对“第一关”和“第二关”求解。为找出影响玩家策略的决定因素，分析一般情况下玩家的最优策略，首先需要将具体的游戏地图简化为抽象的图论模型。以此为基础分析在参数设定及天气状况全部已知的情况下最优策略的依赖因素，籍此建立优化模型最大化玩家收益。并将模型分别带入“第一关”和“第二关”的具体情境求解每一关的最优行为策略。

2.2 问题二的分析

问题二将天气分布设为未知变量，使静态的全局优化模型转变为动态的局部优化模型，若寻求玩家的一般性最优策略，须考虑玩家在不同状态下对天气的预测以及筛选众

多影响决策的因素中最主要的成分，建立合适的图论模型，将动态模型转化为静态模型，便于分析最优策略的决定因子，并将普适性模型代入具体关卡中进行讨论。

2.3 问题三的分析

问题三引入多人博弈的影响，(1) 小题对应问题一的静态全局最优问题，可以视作完全信息博弈问题，而 (2) 小题对应问题二的动态局部规划，需要对各种情形的不同博弈条件和策略空间进行讨论和求解。

三、 模型假设

1. 对于给定游戏设置，必定存在一条能够到达重点的路径
2. 在所有一般性最优策略的决定因素给出后，玩家可以根据这些因素以最优策略完成游戏

四、 符号说明

符号	说明	单位
$\mathcal{G}$	根据原始地图构建的无向图	无
V_0	$\mathcal{G}_0$ 中的顶点集合	无
v_i	$\mathcal{G}_0$ 中的一个顶点	无
v_j	$\mathcal{G}_0$ 中与 v_i 不同的一个顶点	无
E_0	$\mathcal{G}_0$ 中全部公共边界的集合	无
e_{ij}	v_i 与 v_j 连成的边	无
e	无向图 $\mathcal{G}_0$ 中的公共边界	无
$\mathcal{G}$	从 $\mathcal{G}_0$ 中抽象出关键点（起点、终点、村庄、矿山）的无向图	无
V	$\mathcal{G}$ 中关键点（起点、终点、矿山、村庄）的集合	无
$\mathcal{G}'$	从 $\mathcal{G}_0$ 中抽象出“工厂-村庄”集合体的有向图	无

符号	说明	单位
E_0	$\mathcal{G}_0$ 中全部公共边界的集合	无
e_{ij}	v_i 与 v_j 连成的边	无
e	无向图 $\mathcal{G}_0$ 中的公共边界	无
$\mathcal{G}$	从 $\mathcal{G}_0$ 中抽象出关键点（起点、终点、村庄、矿山）的无向图	无
V	$\mathcal{G}$ 中关键点（起点、终点、矿山、村庄）的集合	无
$\mathcal{G}'$	从 $\mathcal{G}_0$ 中抽象出“工厂-村庄”集合体的有向图	无
d_1	从起点到距起点最近的矿山所需的最短时间	天
d_2	从矿山到某个矿山（可能为本身）期间运动所需的最短时间	天
$d_2^{'}$	从矿山到某个矿山（可能为本身）期间停驻所需的最短时间	天
d_3	从距终点最近的矿山到终点所需的最短时间	天
d_m	挖矿总时间	天
G	玩家的总资金	元
p_0	玩家在游戏开始时的本金	元
p_w	水的基准价格	元/箱
p_f	食物的基准价格	元/箱
w_0	在起点处购买水的数量	箱
w_v	在村庄处购买水的总数量	箱
f_0	在起点处购买食物的数量	箱
f_v	在村庄处购买食物的总数量	箱
g	挖矿的基础收益	元
m_w	一箱水的重量	千克
m_f	一箱食物的重量	千克
m_m	玩家的负载上限	千克

符号	说明	单位
C_w	在 T 天内水的平均基础消耗量	箱
C_{w_1}	在 d_1 天内水的平均基础消耗量	箱
C_{w_2}	在 d_2 天内水的平均基础消耗量	箱
C_{w_3}	在 d_3 天内水的平均基础消耗量	箱
C_f	在 T 天内食物的平均基础消耗量	箱
C_{f_1}	在 d_1 天内食物的平均基础消耗量	箱
C_{f_2}	在 d_2 天内食物的平均基础消耗量	箱
C_{f_3}	在 d_3 天内食物的平均基础消耗量	箱
cw_i	水在第 i 天的基础消耗量	箱
cf_i	食物在第 i 天的基础消耗量	箱
T	预计到达的时间	天
T_m	截止日期	天
s_i	沙暴天气的指示函数	无
S	预计到达时间 T 内沙暴天气的总天数	天
p_s	沙暴天气发生的概率	无
p_f	晴朗天气发生的概率	无
p_h	高温天气发生的概率	无
D_{v_t}	第 t 天从当前位置到达终末村庄的最短距离	天
D_{m_t}	第 t 天从当前位置到达终末矿山的最短距离	天
D_{E_t}	第 t 天从当前位置到达终点的最短距离	天
D_{mE_t}	第 t 天从终末矿山到达终点的最短距离	天
D_{mv_t}	第 t 天从终末村庄到达终点的最短距离	天
D_{vE_t}	第 t 天从终末村庄到达终点的最短距离	天

五、 模型的建立与求解

5.1 问题一模型的建立与求解

5.1.1 问题一的分析

问题一要求我们给出一名玩家进行游戏且已知全部天气情况时，该玩家在一般情况下的最优策略，并根据“第一关”与“第二关”的具体情况带入讨论。这就要求我们首先将具体的关卡地图抽象成一般化的图模型，并根据该图的性质进行进一步的线性规划。我们根据模型假设和题目条件可以分析出，假设存在一个终点处玩家资源剩余量大于 0 的最优解，那么我们总是可以通过适当减少在起点或村庄处的资源购买来达到更优解，故此“最优解”不为最优。因此，我们得到结论：

最优策略下玩家到终点的资源剩余量均为 0

我们称其为分析 (a)。同时，基于模型假设 1：

最优策略下玩家的最大负重可以满足任意两个补给点之间的最短路线

我们称其为分析 (b)。

5.1.2 问题一模型的建立

1. 地图模型的简化

我们首先可以根据原始地图构建一个无向图 $\mathcal{G}_0$, 其顶点集合 V_0 中的元素为对应原始地图中的每一个区域（抽象为数字标号）；其公共边界集合 E_0 中的元素为对应原始地图中的每一条公共边界，即对于任两个顶点 v_i, v_j，当且仅当原始地图中两个对应的区域有公共边界时，边$e_{ij} \in E_0$；对于每条 $eij \in E_0$，$|e_{ij}| = 1$。在对 $\mathcal{G}_0$ 进行分析后，我们发现：仅图中的起点、终点、矿山和村庄为决策中需要考虑的点，除此之外的格子并不需要被额外考虑；这些非关键点对最优解的贡献仅仅为定义了这些关键点之间的最短距离。因此，我们可以将 $\mathcal{G}_0$ 进一步抽象为无向图 $\mathcal{G}$，其顶点集合 $V = \{\text{村庄}_1, \text{村庄}_2, \ldots, \text{村庄}_{n_v}, \text{矿山}_1, \text{矿山}_2, \text{矿山}_{n_m}, \text{起点}, \text{终点}\}$，$V$ 中各顶点中的距离通过对 $\mathcal{G}_0$ 应用多元最短路径算法得到，例如 Floyd-Warshall 算法。这里我们使用 Matlab 中 Network Analysis and Visualization 工具箱求解。以关卡一的地图为例，它对应的 $\mathcal{G}$ 可以被简化为如图 1 所示。

同时，分析得出，挖矿是玩家唯一在游戏中获取资金的方式，在村庄购买资源是玩家唯一在游戏过程中获得资源的方式。为了最有效率地获得资金，玩家需要尽快到达矿山，同时在资源不足时需要尽快到达村庄。所以我们将矿山和村庄作为一个集合体“矿山-村庄”提取出来，以此将玩家完成游戏的整个过程分为三部分：起

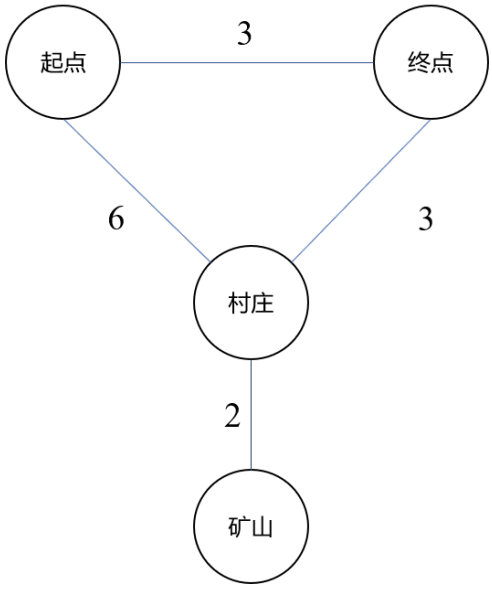

图 **1**　关卡一的无向图

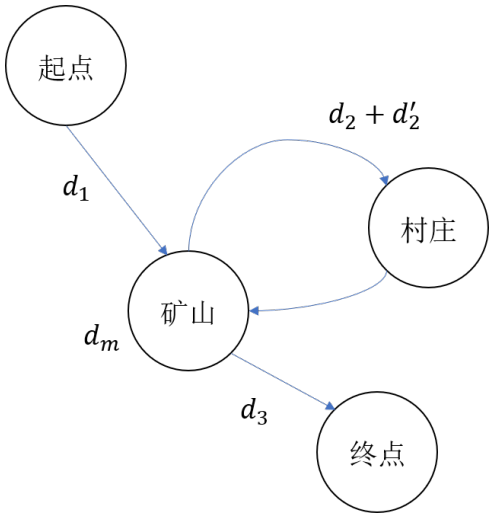

图 **2**　简化后的有向图模型

点与“矿山-村庄”之间，“矿山-村庄”内部，“矿山-村庄”与终点之间。进而将无向图进一步抽象成起点、“矿山-村庄”及终点三部分的有向图，即图 2：

图中，矿山指所有矿山所在格的集合，村庄则为从第一次挖矿到最后一次挖矿途中经过村庄的集合。d_1 指从起点到距起点最近的矿山所需的最短时间，d_m 指挖矿总时间，d_2 指从该矿山到某个矿山（可能为本身）期间运动所需的时间，d_2' 指这一期间停驻的时间，d_3 则表示从距终点最近的矿山到终点所需的最短时间。也就是说，$d_m + d_2 + d_2'$ 为从第一次挖矿到最后一次挖矿需经历的最短时间。

2. **优化模型的建立**

(a) **优化函数的确定**

由题目可知，玩家的目标是在终点时最大化总资金 G。根据图 1 并引入附件

中变量，可以将玩家资金 G 表示如下：

$$G = p_0 - p_w(w_0 + 2w_v) - p_f(f_0 + 2f_v) + gd_m \tag{1}$$

其中，p_0 表示玩家游戏开始时的本金，p_w 表示水的基准价格，p_f 表示食物的基准价格，w_0 表示在起点处购买水的数量，f_0 表示在起点处购买的食物数量，w_v 表示在村庄购买的水的总量，f_v 表示在村庄购买的食物总量，g 表示挖矿的基础收益，d_m 表示挖矿的时间。因此，优化函数可表示为

$$\arg\max_{d_m} \quad p_0 - p_w(w_0 + 2w_v) - p_f(f_0 + 2f_v) + gd_m \tag{2}$$

(b) **约束条件的确定**

根据题目中条件的限制及假设中条件的限制，我们可以进行约束条件的确定。

首先，起步时的负载不能超过负载上限，即：

$$0 \le m_w w_0 + m_f f_0 \le m_m \tag{3}$$

其中，m_w, m_f 分别代表一箱水与一箱食物的重量，m_m 代表玩家的负载上限。对于其他过程中的负重，由分析 (a) 和分析 (b) 可知，一定存在某种方案使玩家所需负重不会超出上限，且玩家的理性操作可保证局部的购买策略达到最优，故不在此设约束条件。

其次，根据分析 (a)，水与食物应当在终点时恰好消耗殆尽，因此在起点与村庄处购买水与食物的总量应当等于游戏过程中消耗的水与食物的总量，即：

$$w_0 + w_v = 2d_1 C_{w_1} + (3d_m + d_2^{'} + 2d_2)C_{w_2} + 2d_3 C_{w_3} \tag{4}$$

$$f_0 + f_v = 2d_1 C_{f_1} + (3d_m + d_2^{'} + 2d_2)C_{f_2} + 2d_3 C_{f_3} \tag{5}$$

其中，C_{w_1} 表示在 d_1 天内水的平均基础消耗量，可表示为：

$$C_{w_1} = \frac{\sum_{i=1}^{d_1} cw_i}{d_1} \tag{6}$$

其中 cw_i 为水在第 i 天时的基础消耗量，其值可由已知每天的天气确定（若沙暴天气停滞则使基础消耗量减半以获得统一数量级的均值）。类似地我们也可以得到从 d_1+1 天到第 d_1+d_m+d_2+$d_2^{'}$ 天水的平均基础消耗量 C_{w_2} 和第 d_1+d_m+d_2+$d_2^{'}$+1 天到 $d_1 + d_m + d_2 + d_2^{'} + d_3$ 天水的平均基础消耗量：

$$C_{w_2} = \frac{\sum_{i=d_1+1}^{d_1+d_m+d_2+d_2^{'}} cw_i}{d_m + d_2 + d_2^{'}} \tag{7}$$

$$C_{w_3} = \frac{\sum_{i=d_1+d_m+d_2+d_2^{'}+1}^{d_1+d_m+d_2+d_2^{'}+d_3} cw_i}{d_3} \tag{8}$$

同样，我们设 cf_i 为食物在第 i 天的基础消耗量，我们可以分别得到食物在以上三个时间阶段中的平均基础消耗量 $C_{f_1}, C_{f_2}, C_{f_3}$:

$$C_{f_1} = \frac{\sum_{i=1}^{d_1} cf_i}{d_1} \tag{9}$$

$$C_{f_2} = \frac{\sum_{i=d_1+1}^{d_1+d_m+d_2+d_2^{'}} cf_i}{d_m + d_2 + d_2^{'}} \tag{10}$$

$$C_{f_3} = \frac{\sum_{i=d_1+d_m+d_2+d_2^{'}+1}^{d_1+d_m+d_2+d_2^{'}+d_3} cf_i}{d_3} \tag{11}$$

显然，上述提到各阶段的时间总和应当等于预计到达的时间 T, 同时各阶段时间应当不小于 0，即

$$d_1 + d_m + d_2^{'} + d_2 + d_3 = T \tag{12}$$

$$d_1, d_m, d_2^{'}, d_2, d_3 \geq 0 \tag{13}$$

(c) **策略选择**

根据以上模型，设截止日期为 T_m，我们分别取 $T = 1, \ldots, T_m$，对于特定的游戏关卡，将已知参数代入规划问题中，利用线性规划求得玩家的最大资金 G 与此时对应的挖矿天数 d_m，最后比较不同 T 取值下 G 的大小，选择 G 最大时的挖矿天数 d_m 作为最优挖矿天数指导游戏策略的抉择。d_m 越大，指征挖矿的收益越高，玩家应该安排自己的行程使挖矿的天数尽可能向上逼近 d_m，相反，d_m 越小甚至为 0 时，玩家应该合理安排较少的挖矿天数，甚至避免挖矿选择最短路径直达终点。

5.1.3 问题一模型的求解

1. **一般策略的求解**

对于问题一建立的优化模型，由于其包含参数过多，复杂度太高，难以求出解析解，我们对其进行定性分析，通过对不等式约束的推导演化出一般性情况下玩家对 d_m 的选择策略。

将公式 (4) 和公式 (5) 代入式 (1)：

$$\begin{aligned} G = gd_m - p_w(4C_{w1}d_1 + 2C_{w2}(3d_m + d_2^{'} + 2d_2) + 4C_{w3}d_3 - w_0) - \\ p_f(4C_{f1}d_1 + 2C_{f2}(3d_m + d_2^{'} + 2d_2) + 4C_{f3}d_3 - f_0) + p_0 \end{aligned} \tag{14}$$

为了计算方便，在定性分析时我们假设三段过程的水和食物的平均日消耗量相等，即令：

$$C_{w1} = C_{w2} = C_{w3} = \bar{C_w} \tag{15}$$

$$C_{f1} = C_{f2} = C_{f3} = \bar{C_f} \tag{16}$$

则可由式 (12) 进一步化简：

$$G = gd_m - (p_wC_w + p_fC_f)(4T + 2d_m - 2d_2^{'}) + p_ww_0 + p_ff_0 + p_0 \tag{17}$$

对于 $d_2^{'}$, 考虑其实际意义可知，在“矿山-村庄”阶段，玩家停驻的时间越少对获得更大的利润更有利，于是 $d_2^{'}$ 应尽可能小以获得较优决策，故将其在等式中的影响暂且忽略，同时只保留 d_m 相关项，可得：

$$G \sim (g - 2(p_wC_w + p_fC_f))d_m \tag{18}$$

即当

$$g > 2(p_wC_w + p_fC_f) \tag{19}$$

时，挖矿天数 d_m 和资金 G 具有线性正相关，对应策略为尽可能多挖矿。值得注意的是，以上不等式的放缩十分松弛，导致挖矿收益 g 拥有极低的下界。而实际上，不等式给出了对于挖矿收益与水和食物的基准价格和基础消耗量的定性关系，即当挖矿收益大于挖矿导致对应的水和食物的额外消耗时，应更多的考虑挖矿以逼近最优策略。同时，我们在定性分析中引入新的约束条件，即在“矿山-村庄”中的行走时长 d_2 须不小于去村庄补给次数和往返最短可能路径（2 天）的乘积，即：

$$d_2 \geqslant 2\left\lceil \frac{m_ww_v + m_ff_v}{m_m} \right\rceil \geqslant 2\frac{m_ww_v + m_ff_v}{m_m} \tag{20}$$

由式 (4) 和式 (5) 代入, 并继承式 (15)、式 (16) 的假设，可得：

$$m_md_2 \geqslant 2(m_wC_w + m_fC_f)(2d_1 + 3d_m + d_2^{'} + 2d_2 + 2d_3) - 2(m_ww_0 + m_ff_0) \tag{21}$$

由式 (12) 并继承忽略 $d_2^{'}$ 的影响这一假设，可化简得：

$$m_md_2 \geqslant 2(m_wC_w + m_fC_f)(3T - d_1 - d_3 - d_2) - 2(m_ww_0 + m_ff_0) \tag{22}$$

将 d_2 代换为 $T - d_1 - d_3 - d_m$ 以关注 T 与 d_m 的关系：

$$(T - d_1 - d_3) - d_m \geqslant \frac{2(m_wC_w + m_fC_f)(3T - d_1 - d_3) - 2(m_ww_0 + m_ff_0)}{m_m + 2(m_wC_w + m_fC_f)} \tag{23}$$

可得不等式：

$$\frac{(m_m - 2(m_wC_w + m_fC_f))T - m_m(d_1 + d_3) - 2(m_ww_0 + m_ff_0)}{(m_m + 2(m_wC_w + m_fC_f))} \geqslant d_m \tag{24}$$

令

$$\frac{(m_m - 2(m_wC_w + m_fC_f))T - m_m(d_1 + d_3) - 2(m_ww_0 + m_ff_0)}{(m_m + 2(m_wC_w + m_fC_f))} \leqslant 0 \tag{25}$$

即

$$T \leqslant \frac{m_m(d_1 + d_3) + 2(m_ww_0 + m_ff_0)}{m_m - 2(m_wC_w + m_fC_f)} \tag{26}$$

由式 (3), 上式可放松为：

$$T \leqslant \frac{m_m(d_1+d_3+2)}{m_m-2(m_wC_w+m_fC_f)} \tag{27}$$

由分析 (b), 日均基础消耗的水和食物的重量 $(m_wC_w+m_fC_f)$ 远小于负重上限 m_m, 则

$$T \leqslant d_1+d_3+2+\epsilon \tag{28}$$

这里 ϵ 取较小整数。此时有 $d_m \leqslant 0$，由式 (13) 可知，$d_m=0$

由式 (28) 所得结果可知，当位于第二阶段“矿山-村庄”区域的时长较短（不超过 $2+\epsilon$）时，$d_m=0$，即应该尽量避免挖矿，直接前往终点。

综上所述，策略决定量 d_m 与挖矿收益 g、日均水和食物损耗 C_w, C_f、水和食物基准价格 p_w, p_f、地图信息 d_1, d_3 以及预计到达时间 T 都有较强的关系，在制定策略时，应综合考虑各个因素。其中，若满足 $g \gg 2(p_wC_w+p_f)$ 则尽量增大挖矿天数，多挖矿以获得更高收益，但若 $T \leqslant d_1+d_3+2+\epsilon$, 则应减少挖矿天数，少挖矿而取距离终点更短的路径直接前往。

2. **关卡一、二的求解**

(a) **计算机模拟的求解算法流程**

由于对于全局最优解需要确定的参数较多、难以直接通过上文所建立的模型求出全局最优解，我们考虑通过使用计算机模拟的方法求得全局最优解。求解算法的流程图如图 3 所示，左侧为决策空间，右侧为决策变量。

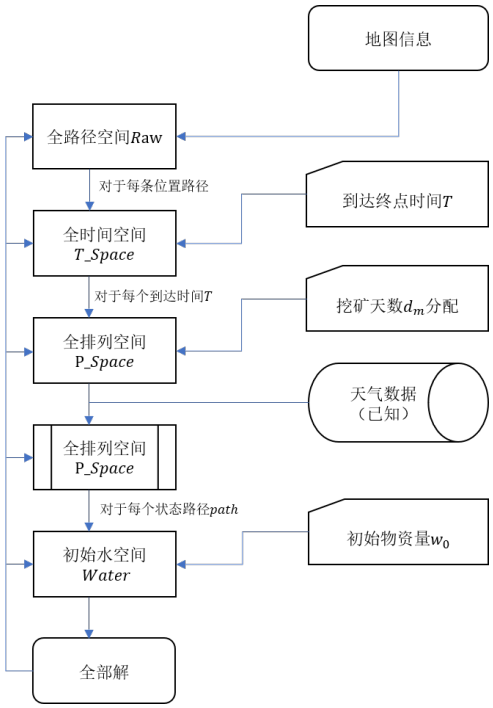

图 **3**　计算机模拟算法流程图

i. **图模型的建立与转化**

与模型一建立时的思想类似，我们首先将初始地图简化为无向图 G 。出于计算机模拟的需要，我们需要将无向图 G 进一步转化。

为了更好的刻画任意路径的位置信息，我们将 G 中的除起点与终点外的全部关键点（村庄、矿场）进行拆点处理，并且将无向图 G 转化为有向图 G_d，关卡一、二的 G_d 如图 4 所示。在这一转化中，矿场被拆的次数由在给定情况

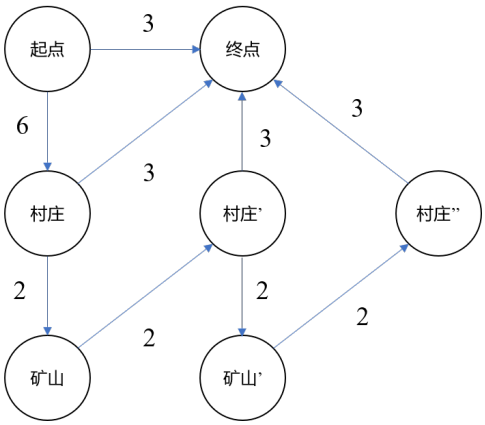

图 **4**　关卡一拆点处理后的有向图

下最多回到矿场挖矿的次数进行刻画，对应的村庄由附近的矿场的决定。以关卡一为例，我们将矿场拆成了三个点。这是因为基于前文“最优解到达终点刚好水和食物全部耗尽”的分析和最优解能够在三十天内在每个矿点挖至少一天去反推出来的一个十分松弛的假设；然而这保证了我们不会误筛掉极端的最优解。关卡二也采取了相同的策略。得到有向图 G_d 后，我们用邻接矩阵对其进行描述。

ii. **全路径空间、全时间空间的生成与最终到达时间的模拟**

利用深度优先搜索算法，可生成出从起点到终点的全路径空间 Raw，注意到这里的路径为位置路径，即仅有位置而无时间或行为的关键点路径。紧接着我们引入最终到达终点的时间的参数，对于 Raw 中每一条路径生成全时间空间 T_Space。

iii. **挖矿天数的模拟和全排列空间的生成**

对于每条位置路径的 T_Space 中的每个元素，我们以在每一矿山的挖矿天数为参数进行模拟，生成全排列空间 P_Space 。我们可以对 P_Space 进行初次筛选，用较松的约束条件筛掉一些一定不能到达终点的路径，即通过剪枝来提高算法性能。

iv. **状态路径的生成**

对于每一 P_Space 中的每一排列，我们结合已知的天气状态从起点开始生成到达终点的状态路径 $path$ ——一个每个代表当天行为（挖矿、等待或行

走）的有序数组。结合对应 P_Space 的位置路径，我们可以唯一的描述该情况下的玩家行为。根据该 P_Space 对应的到达终点的时间 T，我们可以判断这条路径是否能够在规定时间到达终点，从而继续剪枝来提高算法性能。

v. **初始物资量的模拟和每条的初始水空间的生成和筛选**

对于每一条 $path$，我们以不同的初始水购买量为参数进行模拟，生成可行的初始水空间 $Water$。$Water$ 中元素的可行性由对路径的分段消耗计算决定。我们将路径按照村庄进行分段，再基于“最优解到达终点刚好水和食物全部耗尽”的分析，从终点向起点求出每一段的水和食物的消耗值。每段消耗的水和食物对应的负担都应当小于玩家的负重上限，否则应当将其从 $Water$ 中删除。

此外，基于“初始应当尽可能的多买物资”的分析，我们可以确定出对应的食物的初始值。同样将路径按照村庄进行分段，然而从起点向终点遍历。以初始购买较多的物资为基础，在每个村庄检查在该村庄的较多的物资是否占用了较少的物资到下一个村庄所必须的量的重量空间。若出现这种情况，应当立刻停止遍历并将其从 $Water$ 中删除。

vi. **最优解的获得**

经历上述的步骤，我们获得的 $Water$ 空间不为空的每条 $path$ 都可行，这条 $path$ 的最优解对应的初始物资购买量通过在 $Water$ 空间中遍历获得。类似的，遍历比较这些嵌套的空间中的最优解，我们可以得到最终的最优策略。

(b) **关卡一、二的计算机模拟结果与检验**

i. **关卡一计算机模拟结果**

通过计算机模拟的方法，我们求得最优策略的路径为经过村庄补足物资后在矿山挖矿七天，再返回终点，$G = 10430$。这条最优策略对应的到达终点时间 $T = 23$，初始水购买量 $w_0 = 180$，挖矿天数 $d_m = 7$。结果如表 1 所示。

ii. **关卡一计算机模拟结果的检验**

首先，我们对模拟结果进行可行性检验：在以水与食物在第 0 天和在第 23 天的已知剩余量为基础正确填写表格后，计算每天物资剩余量对应的重量，如表 1 的第六列 m 所示，发现路径中没有超过负重上限的日期，故该路径可行。其次，我们对模拟结果进行最优性检验：这条路径对应的参数 (T, d_m, w_0) 通过在每一个空间中遍历全部可能值来确定，故在选择这条路径的情况下，此参数的组合为最优。同时我们也遍历了其他全部可能路径，故得到的此路径也应当是最优解对应的路径。

iii. **关卡二最优策略的计算机模拟结果**

通过计算机模拟的方法，我们求得最优策略的路径为前往村庄 1 补足物资后返回矿山 1 挖矿六天，再经过村庄 1 补足物资到达矿山 2 挖矿七天后返

表 **1**　第一关玩家策略

日期	所在区域	G	w	f	m
0	1	5800	180	330	1200
1	25	5800	164	318	1128
2	26	5800	148	306	1056
3	23	5800	138	292	998
4	23	5800	128	282	948
5	22	5800	118	268	890
6	9	5800	102	256	818
7	9	5800	92	246	768
8	15	4170	245	232	1199
9	13	4170	229	220	1127
10	12	4170	213	208	1055
11	12	5170	183	178	905
12	12	6170	159	160	797
13	12	7170	144	139	710
14	12	8170	120	121	602
15	12	9170	96	103	494
16	12	10170	72	85	386
17	12	11170	42	55	236
18	12	11170	32	45	186
19	14	11170	16	33	114
20	15	10430	36	40	188
21	9	10430	26	26	130
22	21	10430	16	12	72
23	27	10430	0	0	0

回终点，$G = 12730$。这条最优策略对应的到达终点时间 $T = 30$，初始水购买量 $w_0 = 130$，挖矿天数 $d_m = 13$。结果如表 2 所示。

表 **2**　第二关玩家策略

日期	所在区域	剩余资金数 (元)	剩余水量 (箱)	剩余食物量 (箱)	玩家负载 (千克)
0	1	5300	130	405	1200
1	2	5800	114	393	1128
2	3	5300	98	381	1056
3	4	5300	88	367	998
4	4	5300	78	357	948
5	5	5300	68	343	890
6	13	5300	52	331	818
7	13	5300	42	311	748
8	22	5300	32	304	704
9	30	5300	16	292	632
10	39	5200	10	280	590
11	39	3410	179	270	1077
12	30	3410	163	258	1005
13	30	4410	148	237	918
14	30	5410	124	219	810
15	30	6410	100	201	702
16	30	7410	76	183	594
17	30	8410	46	153	444
18	30	9410	16	123	294
19	39	5730	199	199	995
20	47	5730	183	187	923

表 **3**　第二关玩家策略

日期	所在区域	剩余资金数 (元)	剩余水量 (箱)	剩余食物量 (箱)	玩家负载 (千克)
21	55	5730	173	173	865
22	55	6730	163	159	807
23	55	7730	139	141	699
24	55	8730	124	120	612
25	55	9730	94	90	462
26	55	10730	70	72	354
27	55	11730	55	51	267
28	55	12730	40	30	180
29	63	12730	16	12	72
30	64	12730	0	0	0

iv. **关卡二最优策略计算机模拟结果的检验**

首先，我们对模拟结果进行可行性检验：与关卡一类似，在以水与食物在第 0 天和在第 30 天的已知剩余量为基础正确填写表格后，计算每天物资剩余量对应的重量，如表 2 的第六列 m 所示，发现路径中没有超过负重上限的日期，故该路径可行。其次，我们对模拟结果进行最优性检验：与关卡一相同，这条路径对应的参数 (T, d_m, w_0) 通过在每一个空间中遍历全部可能值来确定，故在选择这条路径的情况下，此参数的组合为最优。同时我们也遍历了其他全部可能路径，故得到的此路径也应当是最优解对应的路径。

5.1.4 问题一模型结果的分析

将模型一中优化模型的推导策略结果与实际游戏过程结合分析：玩家挖矿的策略由众多游戏参数共同影响，其中，挖矿天数与基础收益成正相关：对于挖矿的取舍，基础收益是否显著大于估测的日均成本增加是重要影响因素之一，若满足此条件，有极大概率保证倾向于挖矿的策略具有优势地位，与我们的经验所知相符。这里估测的日均成本，可以用第二阶段“矿山-村庄”的日均成本来估计，因为一个选择尽可能多挖矿的最优解的第二阶段的日均成本大于全过程中日均成本。

从第一关与第二关的最优解来看，关卡一最优策略的计算机模拟结果在图 4 模型下，

$d_1 = 10, d_m = 7, d_2 = 2, d_2' = 1, d_3 = 3, T = 23$。经计算，第二阶段的日均成本为312.5，显著小于单日的挖矿收益1000；类似的，关卡二最优策略的计算机模拟结果在图5模型下，$d_1 = 9, d_m = 13, d_2 = 5, d_2' = 1, d_3 = 2, T = 30$。经计算，第二阶段的日均成本为295.5，显著小于单日的挖矿收益1000。这佐证了前文的分析。

另外，挖矿的天数极大的限制于总天数的多少，由于抽象后玩家第一阶段和第三阶段的行动较为固定，第二阶段“矿山-村庄”的路径决策也尤为关键，而若总天数的缩短限制了第二阶段的总时间乃至较小上界，挖矿的收益便失去优势地位，此时倾向于放弃挖矿的策略应该被多加考虑，此结果与游戏的规则相对应，符合游戏本身特性。

5.2 问题二模型的建立与求解

5.2.1 问题二的假设

1. 考虑玩家无法确定未来的天气，除了选择挖矿之外，假设玩家下一步总是选择行走（若因沙暴天气停驻视为一半消耗的行走）；
2. 每天的天气是独立的，即后一天的天气不受之前天气的影响；
3. 非沙暴天气下晴天发生的概率与沙暴天气发生的概率相互独立。

5.2.2 问题二的分析

问题二要求我们讨论一名玩家在仅知道当前天气状况的情况时，该玩家在一般情况下的最优策略，并根据“第三关”与“第四关”的具体情况带入讨论。因此，将问题二的一般模型分为两步，第一步是决策出发时携带资源的数量，第二步是决策出发后的挖矿天数，以最小化未来预期的成本。

5.2.3 问题二模型的建立

1. **天气概率模型的建立**

定义沙暴天气的指示函数 s_i 如式 (29)：

$$s_i = \begin{cases} 0 & \text{第 i 天为非沙暴天气,} \\ 1 & \text{第 i 天为沙暴天气.} \end{cases} \tag{29}$$

由于每天的天气是独立的，因此可以得到预计到达时间 T 内沙暴天气的总天数 S 满足

$$S = \sum_{i=1}^{T} s_i \sim B(T, \hat{p}_s) \tag{30}$$

其中 $\hat{p}_s$ 代表沙暴天气发生的估计概率，可由经验获得。

注意到这是一个二项分布，根据经验公式，在该分布均值 $T\hat{p_s} > 5$ 时，将其近似为正态分布进行分析的效果更好。为了方便，这里不对 $T\hat{p_s}$ 进行限制，即无论 $T\hat{p_s}$ 的值为多少，我们都可将该二项分布近似为正态分布进行分析，即

$$S \sim N(T\hat{p}_s, T\hat{p}_s(1-\hat{p}_s)) \tag{31}$$

所以，我们可以表示出沙暴天气发生的概率分布为

$$p_s = \frac{S}{T} \sim N(\hat{p}_s, \frac{\hat{p}_s(1-\hat{p}_s)}{T}) \tag{32}$$

2. **决策一模型的建立**

决策一要求我们确定在起点处购买水和食物的数量。根据模型一，我们可以利用先验知识（即过去的游戏经验）得到特定地图下最优的估计挖矿天数 $\hat{d}_m$ 与最优预计到达时间 $\hat{T}$。同时，我们可以再利用问题一中的线性规划模型，通过最大化在终点时的预计资金 $\hat{G}$ 来得到在起点处购买的水和食物的最优数量 $\hat{w}_0, \hat{f}_0$，即：

$$\arg\max_{\hat{w}_0, \hat{f}_0} \quad \hat{G} = p_0 - p_w(\hat{w}_0 + 2w_v) - p_f(\hat{f}_0 + 2f_v) + g\hat{d}_m, \tag{33}$$

$$\text{s.t.} \quad m_w\hat{w}_0 + m_f\hat{f}_0 \le m_m, \tag{34}$$

$$\hat{w}_0 + w_v \ge [2d_1 + (3\hat{d}_m + d_2^{'} + 2d_2) + 2d_3]E[C_w], \tag{35}$$

$$\hat{f}_0 + f_v \ge [2d_1 + (3\hat{d}_m + d_2^{'} + 2d_2) + 2d_3]E[C_f], \tag{36}$$

$$d_1 + \hat{d}_m + d_2^{'} + d_2 + d_3 = \hat{T} - d_s \tag{37}$$

$$d_1, \hat{d}_m, d_2^{'}, d_2, d_3 \ge 0 \tag{38}$$

其中，$E[C_w], E[C_f]$ 表示水与食物的期望基础消耗量，其表达式分别如式 (39)(40)：

$$E[C_w] = p_s c_{w_s} + p_f c_{w_f} + p_h c_{w_h} \tag{39}$$

$$E[C_f] = p_s c_{f_s} + p_f c_{f_f} + p_h c_{f_h} \tag{40}$$

其中，$c_{w_s}, c_{w_f}, c_{w_h}$ 分别表示在沙暴、晴天和高温天气下水的基础消耗量，$f_{w_s}, c_{f_f}, c_{f_h}$ 分别表示在沙暴、晴天和高温天气下食物的基础消耗量，p_s, p_f, p_h 分别为天气为沙暴、晴天和高温的概率。由于在天气概率模型中可以得到沙暴天气发生的概率满足正态分布，我们可以设定置信度 α，这样在 $(1-\alpha)\times 100\%$ 的把握下，沙暴发生的概率一定小于

$$p_{se} = z_{\alpha/2}\frac{\hat{p}_s(1-\hat{p}_s)}{\hat{T}} + \hat{p_s} \tag{41}$$

其中 $z_{\alpha/2}$ 为标准正态分布下置信度为 α 时的 z 值（可查表得到），将此概率作为沙暴发生的最大概率，再根据经验获得的非沙暴天气下晴天与高温发生的概率 $\hat{p}_{f|\bar{s}}, \hat{p}_{h|\bar{s}}$，得到沙暴发生概率最大时晴天与高温发生的概率 p_{fe}, p_{he}

$$p_{fe} = (1-p_{se})\hat{p}_{f|\bar{s}} \tag{42}$$

$$p_{he} = (1 - p_{se})\hat{p}_{h|\bar{s}} \tag{43}$$

而式 (37) 中的 d_s 表示沙暴预计发生的最大天数，即

$$d_s = \lceil p_{se}\hat{T} \rceil \tag{44}$$

这样，优化模型中的参量都得以确定，可以以 $\hat{w}_0, \hat{f}_0$ 为变量，其他值为参量，对玩家总资金 G 进行线性规划。

3. **决策二模型的建立**

(a) **游戏地图模型的抽象**

考虑到在未知天气做决策时，玩家下一步必定会选择更加靠近矿山、村庄或者终点（若已在矿山区域中，则会选择继续挖矿或者前往下一个矿山、村庄或者终点）。在地图中，会存在这样的矿山（村庄）如定义 1：

定义 1 *与经过其他矿山（村庄）相比，从当前位置经过该矿山（村庄）到达终点的距离最短的村庄叫做终末村庄。*

在决策过程中，若要决定预期挖矿天数的行动，经过终末矿山到达村庄的距离是关键因素（若资源消耗不足以支撑经过终末村庄到达终点的预期时间，则应当选择不挖矿），因此我们将天气条件未知的地图简化为以当前位置、终点、终末矿山及终末村庄四个关键点的图模型如图 5：

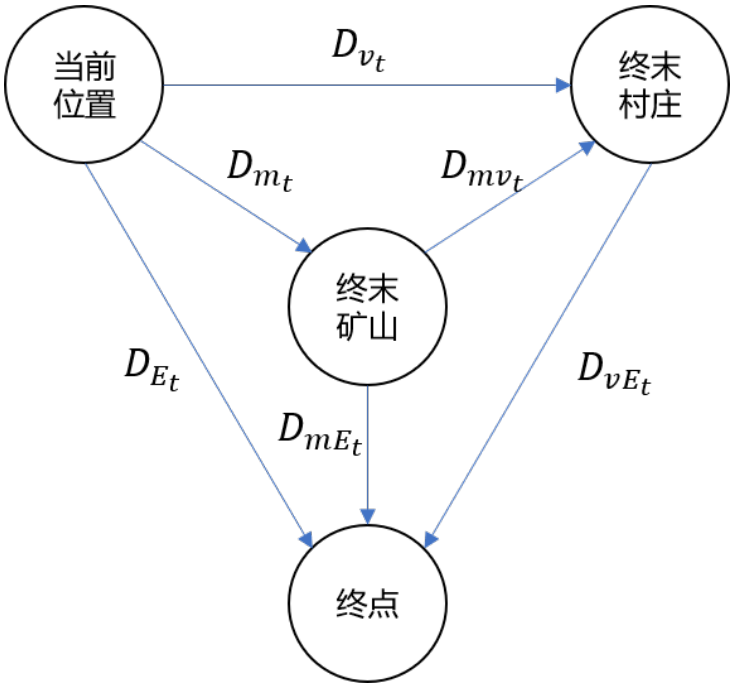

图 **5**　未来天气状况未知时的简化沙漠图模型

图中，t 指玩家已出发的时间（天），D_{v_t} 指第 t 天从当前位置到达终末村庄的最短距离；D_{m_t} 指第 t 天从当前位置到达终末矿山的最短距离；D_{E_t} 指第 t 天从

当前位置到达终点的最短距离；D_{mE_t} 指第 t 天从终末矿山到达终点的最短距离；D_{mv_t} 指第 t 天从终末矿山到达终末村庄的最短距离；D_{vE_t} 指第 t 天从终末村庄到达终点的最短距离。

(b) **优化函数的确定**

在未知未来天气的情况下，可以将最大化玩家在终点时的期望总资金等价为最小化玩家从当前时间 t 开始预计的最终总成本，即：

$$\min \quad E[C_t] \tag{45}$$

(c) **约束条件的确定**

首先，本题的优化目标关注于玩家在任一局部状态 $\{t, w_t, f_t, info(\mathcal{G})\}$ 下对其之后行程的规划。其中 t 为时间信息;w_t, f_t 分别代表玩家在时间 t 时携带水与食物的数量，为玩家自身状态信息;$info(\mathcal{G}) = \{D_{v_t}, D_{m_t}, D_{E_t}, D_{mE_t}, D_{vE_t}, D_{mv_t}\}$ 为地图信息。玩家所作出的优化决策应基于以上信息进行分析求解。注意到当关注于局部状态时，正在携带的水和食物的成本于购买时的单价差异无关，而是统一由其当前价值（即到达终点时卖出价格）决定，而计划中将在之后购买的水和食物的单价皆应为在村庄购买的单价。于是可以给出时间 t 后的总成本期望表达式 $E[C_t]$ 如式 (46)

$$E[C_t] = 2p_w N_{w_t} + 2p_f N_{f_t} - (\frac{3}{2}p_w w_t + \frac{3}{2}p_f f_t) - g d_{m_t} \tag{46}$$

其中,d_{m_t} 表示从时间 t 起预计挖矿的天数，N_{w_t} 表示从时间 t 起预计水的总消耗量，有上界关系：

$$N_{w_t} \leqslant E[C_w](2(T-t) + d_{m_t}) \tag{47}$$

类似地，从时间 t 起预计食物的总消耗量为

$$N_{f_t} \leqslant E[C_f](2(T-t) + d_{m_t}) \tag{48}$$

注意到与之前的变量讨论不同，此时由假设 1，玩家在尽量避免行走的前提下可以根据自身选择线路估算出到达终点的时间 T，因此 T 作为 d_m 的表达式参与模型讨论，即：

$$T - t = d_m + D_{m_t} + D_{mE_t} + \Delta D_t \left\lceil \frac{m_w(N_{w_t} - w_t) + m_f(N_{f_t} - f_t)}{m_m} \right\rceil \tag{49}$$

其中 ΔD_t 表示与不前往村庄的路径相比因为前往村庄补给一次而带来的时间增量，极大程度上依赖于地图信息，难以以统一的数学表达式进行表达，但若已知地图，则可以对其进行较精确的推算。在该优化问题的条件下，可以依据于地图信息 $info(D)$ 对其同 d_m 的关系进行估测, 进而近似求解。

约束条件还包含：在当前时间 t 时，玩家的负载不能大于最大负载 m_m，即

$$0 \leq m_w w_t + m_f f_t \leq m_m \tag{50}$$

其中 w_t, f_t 分别代表玩家在时间 t 时携带水与食物的数量。
同时，根据游戏地图模型的抽象，关键点之间的距离有如下约束：

$$D_{m_t} + D_{mE_t} \geq D_{E_t} \tag{51}$$

$$D_{v_t} + D_{vE_t} \geq D_{E_t} \tag{52}$$

$$D_{m_t} + D_{mv_t} \geq D_{v_t} \tag{53}$$

$$D_{v_t} + D_{mv_t} \geq D_{m_t} \tag{54}$$

$$D_{vE_t} + D_{mv_t} \geq D_{mE_t} \tag{55}$$

$$D_{mE_t} + D_{mv_t} \geq D_{vE_t} \tag{56}$$

显然，对于各个时间参量都必须满足

$$0 < t, \Delta D_t, d_{m_t}, D_{v_t}, D_{m_t}, D_{E_t}, D_{mE_t}, D_{vE_t}, D_{mv_t} < T \tag{57}$$

4. 决策二模型的求解

(a) 一般策略的求解

首先，由式 (47)、式 (48)，优化函数具有上界

$$\begin{aligned} E[C_t] \leqslant 2(p_w E[C_w] + p_f E[C_f])(2(T-t) + d_{m_t}) \\ -(\frac{3}{2} p_w w_t + \frac{3}{2} p_f f_t) - g d_{m_t} \end{aligned} \tag{58}$$

通过假设 1 可以分析知式 (47)、式 (48) 的不等式十分紧致，因此优化问题可以转化为对优化函数的上界考虑最小化。即：

$$\begin{aligned} \arg\min_{d_{m_t}} \quad E^{'} = 2(p_w E[C_w] + p_f E[C_f])(2(T-t) + d_{m_t}) \\ -(\frac{3}{2} p_w w_t + \frac{3}{2} p_f f_t) - g d_{m_t} \end{aligned} \tag{59}$$

将式 (48) 代入 $E^{'}$:

$$\begin{aligned} E^{'} = 2(p_w E[C_w] + p_f E[C_f])(3 d_{m_t} + 2(D_{m_t} + D_{mE_t} + \\ \Delta D_t \left\lceil \frac{m_w(N_{w_t} - w_t) + m_f(N_{f_t} - f_t)}{m_m} \right\rceil)) \\ -(\frac{3}{2} p_w w_t + \frac{3}{2} p_f f_t) - g d_{m_t} \end{aligned} \tag{60}$$

在对 ΔD_t 的估测中，将玩家前往村庄的行走模式划分为三类：当前位置-终末村庄-终末矿山；终末矿山-终末村庄-终末矿山；终末矿山-终末村庄-终点。为了计

算方便，将三种行走模式的出现频率简化为相等，则可估测出 ΔD_t 的值为：

$$\begin{aligned}\Delta D_t &= \frac{1}{3}(D_{v_t} + D_{mv_t} - D_{m_t}) + \frac{1}{3}(2D_{mv_t})) + \frac{1}{3}(D_{vE_t} + D_{mv_t} - D_{mE_t}) \\ &= \frac{1}{3}(D_{v_t} + D_{vE_t} + 4D_{mv_t} - D_{m_t} - D_{mE_t})\end{aligned} \tag{61}$$

其值与 d_{m_t} 无关，故不做优化考虑。对于 ΔD 项的系数，其实际意义为前往村庄补给的次数，根据对实际游戏情况的观察和模拟，给出一下关系：

$$\left\lceil \frac{m_w(N_{w_t} - w_t) + m_f(N_{f_t} - f_t)}{m_m} \right\rceil \approx \frac{1}{e^{-0.3\sqrt{d_{m_t}}}} - 1 \tag{62}$$

则代入式 (59) 得：

$$\begin{aligned}E^{'} = (6(p_w E[C_w] + p_f E[C_f]) - g)d_{m_t} + \frac{2\Delta D_t(p_w E[C_w] + p_f E[C_f])}{e^{-0.3\sqrt{d_{m_t}}}} \\ +4(p_w E[C_w] + p_f E[C_f])(D_{m_t} + D_{mE_t} - \Delta D) - (\frac{3}{2}p_w w_t + \frac{3}{2}p_f f_t)\end{aligned} \tag{63}$$

即：

$$E^{'} \sim (6(p_w E[C_w] + p_f E[C_f]) - g)d_{m_t} + \frac{2\Delta D_t(p_w E[C_w] + p_f E[C_f])}{e^{-0.3\sqrt{d_{m_t}}}} \tag{64}$$

可以观察到，d_{m_t} 对 $E^{'}$ 的影响是由参数基础收益 g、用时增量 ΔD、水和食物的基准价格 p_w, p_f 以及期望基础消耗量 $E[C_w], E[C_f]$ 决定的，对于不同的参数取值，分别可以得到可行域内使 $E^{'}$ 最小化的 d_{m_t}。将其作为指导指数决定玩家在状态 $\{t, w_t, f_t, info(\mathcal{G})\}$ 时的最优策略，例如，若 d_{m_t} 较大，则应该向终末矿山移动，反之则向终末村庄移动，若 d_{m_t} 为零则直接向终点移动。

(b) **关卡三的具体讨论**

与关卡一、二的策略求解中类似地，首先将关卡的无向图进行拆点处理得到有向图如图 6：

由图 6 不难看出，根据第二题上述模型，玩家在任意状态下的策略由于不存在村庄而变为是否前往矿山，即为求解优化函数的最小值 d_{m_t} 是否大于 0，若大于 0 则做出前往矿山挖矿的策略，否则即直接前往终点。根据题目所给信息可以获得 $g = 200$ 元，$p_w = 5$ 元，$p_f = 10$ 元。则根据天气概率模型的假设，无沙暴天气的条件下晴朗天气和高温天气出现的概率各占 50%，可得 $p_w E[C_w] + p_f E[C_f] = 95$ 元。即式 (64) 第一项 d_{m_t} 的线性系数 $6(p_w E[C_w] + p_f E[C_f]) - g = 370 > 0$，即第一项与 d_{m_t} 成正相关，而第二项总与 d_{m_t} 成正相关，故对于第三关的任意状态下，$E^{'}$ 总与 d_{m_t} 呈正相关，说明随着挖矿天数增加成本的期望总会增大，对应收益的期望减小，故应尽量避免挖矿，即选择起点直接前往终点为最优策略。

(c) **关卡四的具体讨论**

同样给出处理后的有向图如图 7：与关卡三的计算方式类似，此时假定沙暴

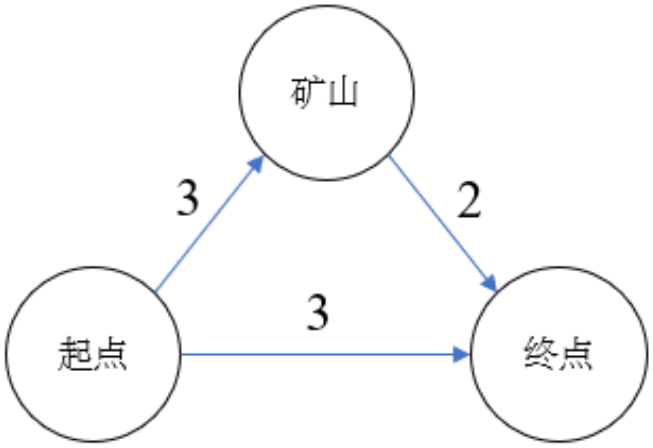

图 **6**　关卡三拆点处理后的有向图

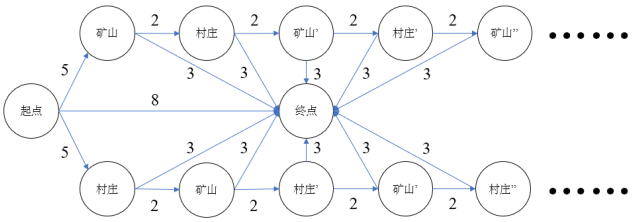

图 **7**　关卡四拆点处理后的有向图

的出现概率为 10%（出现较少），而晴朗和高温出现的概率分别为 45% 和 45%，故计算出 $6(p_w E[C_w] + p_f E[C_f]) - g = -397 < 0$, 即第一项与 d_{m_t} 成负相关，而对于第二项 $\frac{201\Delta D_t}{\exp(-0.3\sqrt{d_{m_t}})}$，其对于 E' 的影响是由 ΔD_t 决定的，即 ΔD_t 越大，玩家挖矿的成本越大，收益越小，倾向于多挖矿的策略优势地位减弱。即玩家在每次位于某个状态时需要根据当前状态下去村庄补给的路程增量抉择是否加长挖矿时间，若路程增量较小则可以多挖矿，反之则少挖矿甚至不挖，以此指导玩家的优化策略。

5.3 问题三模型的建立与求解

5.3.1 问题三的分析

1. **问题三第一问的分析**

在问题三第一问中，游戏时段内的天气情况全部已知，玩家需要在起点确定自己的策略，每个玩家没有私人信息，因此这是一个完全信息静态博弈的模型。在这种情况下，玩家需要考虑所有可能的情况，找到博弈中的均衡点并以此确定自己的

策略。

2. 问题三第二问的分析

在问题三第二问中，游戏时段内的天气情况未知且玩家的策略变为动态策略，由每一天玩家的相关状态有关，我们尝试通过对状态机的描述将这个动态博弈模型转化为完全信息静态博弈模型，然后利用第一问的模型进行求解。考虑问题二对状态机的设置，我们描绘时间信息、玩家自身信息和地图信息对玩家成本的影响来完成动态至静态的转化。

5.3.2 问题三模型的建立

博弈论中的策略有两种形式，纯策略和混合策略，根据纳什定理 [1] 对于任意有限人参与，有限策略空间的博弈，一定存在一个纳什均衡，但该纳什均衡可能包含混合策略。

由于纯策略是混合策略的一种特殊情况，因此我们探究混合策略的纳什均衡 [1] 情况。对于任意有限人参与，有限策略空间的博弈，一定存在一个纳什均衡，但该纳什均衡可能包含混合策略。由于纯策略是混合策略的一种特殊情况，我们可以通过建立混合策略模型，并探究该模型的纳什均衡情况。

首先，记 P_i 为第 i 位玩家，N 为所有参与者 P_i 的集合，且有

$$|N| = n \tag{65}$$

$$N_{-i} = N - \{P_i\} \tag{66}$$

记参与者 P_i 的纯策略为 $\mathbf{s}_i$,P_i 的所有策略组成一个人的策略空间 $\mathbf{S}_i$, 设其大小为

$$|\mathbf{S}_i| = k^i \tag{67}$$

同时，$\mathbf{s}_i$ 可表示为

$$\mathbf{s}_i = (\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_m) \tag{68}$$

其中，$\mathbf{p}_i, i \in \{1, \ldots, m\}$ 表示参与人第 i 天的起点，可定义 $|\{\mathbf{p}_i\}|$ 为策略 $\mathbf{s}_i$ 的点长度 $l(\mathbf{s}_i)$。根据单个人的纯策略，我们可以得到所有人的纯策略元组

$$\mathbf{s} = (\mathbf{s}_1, \ldots, \mathbf{s}_n) = (\mathbf{s}_i, \mathbf{s}_{-i}) \tag{69}$$

同时，可以得到所有人的纯策略元组 $\mathbf{s}$ 的全集

$$\mathbf{S} \equiv \times_{i \in N}(\mathbf{S}_i) \tag{70}$$

类似地，可以得到除 P_i 以外所有人的纯策略元组 s_{-i} 的全集为

$$\mathbf{S}_{-i} \equiv \times_{i \in N_{-i}}(\mathbf{S_i}) \tag{71}$$

其中 s_{-i} 表示除了玩家 P_i 的策略之外所有人的一个策略组成的元组。

混合策略是纯策略上的一个概率分布，因此 P_i 的混合策略可以表示成

$$\sigma_i = (\sigma_i(\mathbf{s}_i^1), \sigma_i(\mathbf{s}_i^2), \ldots, \sigma_i(\mathbf{s}_i^{k^i})) \tag{72}$$

其中 $\mathbf{s}_i^j, j \in \{1, \ldots, k^i\}$ 为 P_i 策略空间里的每一个纯策略，$\sigma(\mathbf{s}_i^j), j \in \{1, \ldots, k^i\}$ 表示该混合策略中纯策略 $\mathbf{s}_i^j$ 被选择的概率。记所有混合策略组成的集合为 $\Delta(\mathbf{S_i})$, 混合策略 σ_i 的支撑为

$$\delta(\sigma_i) = \{\mathbf{s}_i \in \mathbf{S}_i : \sigma_i(\mathbf{s}_i) > 0\} \tag{73}$$

根据单个人的混合策略，可以得到所有人的混合策略合成一个元组

$$\sigma = (\sigma_1, \ldots, \sigma_n) \tag{74}$$

为了方便，也可以表示成

$$\sigma = (\sigma_i, \sigma_{-i}) \tag{75}$$

其中 σ_{-i} 为除 P_i 之外所有人的一个混合策略组成的元组同时，可以得到所有人混合策略元组的全集

$$\Delta(\mathbf{S}) \equiv \times_{i \in N}(\Delta(\mathbf{S}_i)) \tag{76}$$

而除 P_i 之外所有人的混合策略元组全集为

$$\Delta(\mathbf{S}_{-i}) \equiv \times_{i \in N_{-i}}(\Delta(\mathbf{S}_i)) \tag{77}$$

给出所有人策略元组 σ 的收益函数 $u_i : \Delta(\mathbf{S}) \to \mathbb{R}$, 可记为

$$u_i(\sigma) = u_i(\sigma_i, \sigma_{-i}) \tag{78}$$

因此，我们给出了有 n 名玩家的情况下游戏的混合策略模型，在不同关卡，需要给出不同的收益函数对其纳什均衡进行求解，以得出不同玩家的策略。其中，收益函数的描绘由问题一和问题二的对应模型给出，分别对应静态全局的优化和动态局部的优化。

5.3.3 问题三模型的求解

1. **一般策略的求解**

首先，给出在混合策略下纳什均衡的定义：给定一个所有人的混合策略元组

$$\sigma' = (\sigma'_1, \sigma'_2, \cdots, \sigma'_n) = (\sigma'_i, \sigma'_{-i}) \tag{79}$$

若:

$$u_i(\sigma'_i, \sigma'_{-i}) \geq u_i(\sigma_i, \sigma'_{-i}), \quad \forall \sigma_i \in \Delta(\mathbf{S}_i), \quad \forall i \in n \tag{80}$$

那么 σ' 为一个混合策略纳什均衡。注意到纳什均衡中有一种特殊情况称为优势均衡，在混合策略下，优势均衡的定义如下：给定一个所有人的混合策略元组

$$\sigma' = (\sigma'_1, \sigma'_2, \cdots, \sigma'_n) = (\sigma'_i, \sigma'_{-i}) \tag{81}$$

若：

$$u_i(\sigma'_i, \sigma_{-i}) \geq u_i(\sigma_i, \sigma_{-i}), \quad \forall \sigma_i \in \Delta(\mathbf{S}_i), \quad \forall \sigma_{-i} \in \Delta(\mathbf{S}_{-i}), \forall i \in N \tag{82}$$

那么 σ' 为一个混合策略优势均衡。

特别地，作为混合策略的特殊情况，我们也给出纯策略的纳什均衡和最优均衡定义：给定一个所有人的纯策略元组

$$\mathbf{s}' = (\mathbf{s}'_1, \mathbf{s}'_2, \cdots, \mathbf{s}'_n) = (\mathbf{s}'_i, \mathbf{s}'_{-i}) \tag{83}$$

若:

$$u_i(\mathbf{s}'_i, \mathbf{s}'_{-i}) \geq u_i(\mathbf{s}_i, \mathbf{s}'_{-i}), \quad \forall \mathbf{s}_i \in \mathbf{S}_i, \quad \forall i \in N \tag{84}$$

那么 $\mathbf{s}'$ 为一个纯策略纳什均衡。

若:

$$u_i(\mathbf{s}'_i, \mathbf{s}_{-i}) \geq u_i(\mathbf{s}_i, \mathbf{s}_{-i}), \quad \forall \mathbf{s}_i \in \mathbf{S}_i, \quad \forall \mathbf{s}_{-i} \in \mathbf{S}_{-i}, \forall i \in N \tag{85}$$

那么 s' 为一个纯策略优势均衡。

在玩家计算出纳什均衡后，他们应根据均衡策略选择自己的路线。但对于混合策略均衡，玩家的策略是对几条路线的概率分布，但事实上，对于玩家均衡点混合策略的任意选择概率不为 0 的纯策略，玩家选择他们得到的收益相同，且都等于选择混合策略时的收益，以下引理给出了证明。

引理 1 $\sigma^* = (\sigma_1^*, \sigma_2^*, \cdots, \sigma_n^*) = (\sigma_i^*, \sigma_{-i}^*)$ 是一个混合策略纳什均衡，那么对于 $\forall \mathbf{s}_i \in \delta(\sigma_i^*)$，$u_i(\mathbf{s}_i, \sigma_{-i}^*)$ 都是相同的并且等于 $u_i(\sigma_i^*, \sigma_{-i}^*)$

证明 1

$$u_i(\sigma_i^*, \sigma_{-i}^*) = \max_{\sigma_i \in \Delta(\mathbf{S}_i)} u_i(\sigma_i, \sigma_{-i}^*) = \max_{\mathbf{s}_i \in \mathbf{S}_i} u_i(\mathbf{s}_i, \sigma_{-i}^*)$$

$$\sum_{\mathbf{s}_i \in \mathbf{S}_i} \sigma_i^*(\mathbf{s}_i) u_i(\mathbf{s}_i, \sigma_{-i}^*) = \max_{\mathbf{s}_i \in \mathbf{S}_i} u_i(\mathbf{s}_i, \sigma_{-i}^*)$$

$$\sum_{\mathbf{s}_i \in \delta(\sigma_i^*)} \sigma_i^*(\mathbf{s}_i) u_i(\mathbf{s}_i, \sigma_{-i}^*) = \max_{\mathbf{s}_i \in \mathbf{S}_i} u_i(\mathbf{s}_i, \sigma_{-i}^*)$$

$$u_i(\mathbf{s}_i, \sigma_{-i}^*) = \max_{\mathbf{s}_i \in S_i} u_i(\mathbf{s}_i, \sigma_{-i}^*) = u_i(\sigma_i^*, \sigma_{-i}^*) \quad \forall \mathbf{s}_i \in \delta(\sigma_i^*)$$

对于完全信息的静态博弈，计算混合策略的纳什均衡有很多已有算法 [2][3]。

基于以上分析，我们可以得出求解问题三的一般方法：

(a) 找到从起点到终点的所有可能路径作为每个参与人 i 的策略空间 S_i

(b) 对于 $\forall\sigma \in \Delta(\mathbf{S}), \forall i \in N$ 算出 $u_i(\sigma_i, \sigma_{-i})$

(c) 通过上述文献中的算法找到该博弈的均衡

(d) 对于有多个均衡点的情况，优先选择优势策略均衡

(e) 经过第四步的筛选，再选择其中所有人收益和最大的均衡点

(f) 经过第四步和第五步的筛选，如果存在，选择剩余均衡点中的纯策略均衡

(g) 根据最终得到均衡确定策略, 对于混合策略均衡选定其支撑中的任意纯策略即可

但由于以上算法复杂度过高，处理具体问题时应尽可能进行简化，其中最关键的是缩小每个人的策略空间。

对于缩小策略空间，我们可以利用问题一中建立的优化模型，当前条件与问题一中的题目参数条件完全相同，同时玩家的最优策略都是从全局做参考，故可以利用问题一的模型求解结果 d_m 指征策略空间的缩小，即从每位玩家的全部行程组合缩小为满足 d_m 最优条件的行程组合，大大减小了策略空间的复杂度。

2. **关卡五的求解**

对于第五关，可以通过参数计算出晴朗天气基础消耗为 55 元，高温天气基础消耗为 135 元。

考虑一个人的情况，在去矿山的情况下，玩家在晴朗天的收益为 $200-55*3=35$, 在高温天的收益为 $200-135*3=-205$, 故玩家在高温天挖矿只能获得负收益，而在人更多的情况下矿山收益会减少，因此玩家的策略中不可能存在高温天在矿山挖矿的情况。

考虑一个人的情况，在到矿山的途中如果遇到高温，则此时权衡的量为在矿山多挖一天的最高收益和高温天在路上多消耗的钱，前者为 35, 后者为 135。因此在去矿山的途中，如遇到高温则必定停留。在这种情况下玩家如果要挖矿，则最早在第四天到达矿山，第五天开始挖矿。由于在高温天挖矿会得到负收益，因此玩家只会在第五天，第六天挖矿。考虑玩家只经过矿山不挖矿，则玩家将在晴朗的第五，第六天前往终点，一共消耗 $55*2*2=220$。由于第七到第十天均为高温，因此相比于经过矿山直接到终点的情况，玩家每在矿山多挖一天矿就意味着多一天在高温天气行走，此时的收益差为 $35-(55*2-135*2)=-125$。因此玩家与其挖矿不如经过矿山不挖矿直接到达终点。

对于两个人的情况，唯一不同的情况在于两人挖矿的时间或路线有交集，假如两人都挖矿，有交集不会比一个人的情况更好，因此此时一个玩家单方面选择策略 $(1,4,4,6,13)$ 会获得更高收益,而另一个玩家选择经过矿山不挖矿的策略 $(1,2,2,3,9,11,13)$ 也会获得更高收益。因此存在挖矿的策略永远不可能成为均衡解，将挖矿策略从策略集中剔除。

综上所述，对于两个人的情况，最优解必定存在于从起点直接到终点的情况。

考虑最佳情况，若只有一个玩家，则其收益最高的路线应为最短路径，并考虑是

否应该在高温天停留。通过该方法可找到最优策略为 $(1,4,4,6,13)$ 或 $(1,5,5,6,13)$, 因此 $\min(l(s_i))=4$。此时玩家的收益 $u_i^{max}=10000-55*2-135-55*2*2=9535$

根据对关卡五体系的具体分析，给出以下性质及证明：

性质 1 该博弈不存在优势策略均衡。

证明如证明 2：

证明 2 当 $\mathbf{s}_2=(1,4,4,6,6,13)$ 时，参与人 1 仅在选择 $\mathbf{s}_1=(1,5,5,6,13)$ 可以得到最大收益 $u_i^{max}=9935$。当 $\mathbf{s}_2=(1,5,5,6,13)$ 时，参与人 1 仅在选择 $\mathbf{s}_1=(1,4,4,6,6,13)$ 可以得到最大收益 $u_i^{max}=9935$，$\mathbf{s}_1=(1,5,5,6,3)$ 此时并非得到收益最大的策略，因此该博弈不存在优势策略均衡

性质 2 设该博弈所有纳什均衡点组成的集合为 E

$e_1=((1,5,5,6,13),(1,4,4,6,6,13))$，$e_2=((1,4,4,6,6,13),(1,5,5,6,13))$,

$e_3=((1,4,4,6,13),(1,5,5,6,6,13))$，$e_4=((1,5,5,6,6,13),(1,4,4,6,13))\in E$, 且 $\forall e\in E-\{e_1,e_2,e_3,e_4\}$, $u_1(e_1)+u_2(e_1)=u_1(e_2)+u_2(e_2)=u_1(e_3)+u_2(e_3)=u_1(e_4)+u_2(e_4)>u_1(e)+u_2(e)$

证明如证明 3

证明 3 在此仅证明 $e_1\in E$，其他三个点是纳什均衡点的证明方法类似。

设 $\mathbf{s}_1^*=(1,5,5,6,13),\mathbf{s}_2^*=(1,4,4,6,6,13)$, $u_1(\mathbf{s}_1^*,\mathbf{s}_2^*)=u_1^{max}=9535$, 所以 $\forall\mathbf{s}_1\in\mathbf{S}_1,u_1(\mathbf{s}_1,\mathbf{s}_2^*)\le u_1(\mathbf{s}_1^*,\mathbf{s}_2^*)$。$u_2(\mathbf{s}_1^*,\mathbf{s}_2^*)=9480$。

在仅从起点到终点的情况下，次优解 $u_i^{max_2}$ 最大为最优解多减一天晴朗的基础消耗，即为 9480。因此只需证明 $\forall\mathbf{s}_2\in\mathbf{S}_2,u_i(\mathbf{s}_1^*,\mathbf{s}_2)\neq u_2^{max}=9535$。对于 $l(\mathbf{s}_2)>4$ 的情况，由于走过了更多的路径不可能达到 $l(\mathbf{s}_2)=4$ 的最优解。对于 $l(\mathbf{s}_2)=4$ 的情况，一旦在第二个点以外的停留就不可能达到最优解，但若仅在第二天停留，则这样的 $\mathbf{s}_2$ 和 $\mathbf{s}_1^*$ 必定有一段时间在同一段路径上行走，此时 $u_i(\mathbf{s}_1^*,\mathbf{s}_2)\neq u_2^{max}$。综上所述，$(\mathbf{s}_1^*,\mathbf{s}_2^*)\in E$。

最后，我们证明上述四个纳什均衡点为两人总收益最大的四个均衡点

证明 4 由上述证明可知，$u_1(e_1)+u_2(e_1)=u_1(e_2)+u_2(e_2)=u_1(e_3)+u_2(e_3)=u_1(e_4)+u_2(e_4)=u_i^{max}+u_i^{max_2}$，且 $\nexists e\in E,u_1(e)+u_2(e)=2u_i^{max}$。因此 $\max(u_1(e)+u_2(e))=u_i^{max}+u_i^{max_2}$。

由于这四个均衡点为包含最优解策略 $(1,4,4,6,13),(1,5,5,6,13)$ 的全部情况，因此 $\nexists e\in E-\{e_1,e_2,e_3,e_4\},u_1(e)=u_i^{max}$ ***or*** $u_2(e)=u_i^{max}$。故 $\forall e\in E-\{e_1,e_2,e_3,e_4\}$, $u_1(e)+u_2(e)<u_i^{max}+u_i^{max_2}$

除此之外，e_1,e_2,e_3,e_4 为四个纯策略均衡点，根据一般分析中的方法，这四个均衡点为最终用来选择策略的均衡点。最终两个玩家的选择结果将会是这四个均衡点中的某一个。

3. **关卡六的求解**

通过第二问的模型，我们可以将动态博弈的模型转化为静态博弈的模型，即对应所有 N 位玩家的状态机集合为 $\{\{t_i, w_{t_i}, f_{t_i}, info(D_t)_i\}\}, i = 1, 2, ..., N$，根据第二问的模型求解可以解出策略指导量集合 $d_{m_{t_i}}, i = 1, 2, ..., N$，基于指导量求得独立相对优化路径构建策略空间集合 $\mathbf{S}_{t_i}$，类似于第一问，可以求解出纳什均衡点，遍历所有纳什均衡点可以获得最优的玩家策略组合。

六、 模型的评价

6.1 模型的优点

首先，本文对原始地图进行多次简化，抽象出了关键决策信息，如模型一中的挖矿天数 d_m 和模型二中 d_t 天后的挖矿天数，简化了模型求解。其次，本文通过计算机模拟的方法遍历全部可能性，从而计算出关卡一和关卡二的最优路径，并验证了模拟的结果符合问题一中建立的规划模型得出的最优策略。最后，问题三基于纳什均衡的模型的同时，也利用了问题一和问题二的模型建立的思路和结果，全文的思路具有连贯性。

6.2 模型的缺点

玩家的决策受到多个非理性因素的影响，导致做出的决策并非一定理性；第一题的计算机模拟程序时间复杂度较高。

七、 模型的改进与推广

7.1 通过动态规划对第一题计算机模拟程序进行优化

根据上文对于模型缺点的分析，我们可以使用动态规划模型来求解给定参数条件下的最优路径，此算法的时间、空间复杂度均优于文中的计算机模拟算法；又由于动态规划算法本身考虑状态转移的特点，我们可以将此算法推广，与状态转移模型结合求解局部优化问题。动态规划算法如下：

基于上图，求解最优线路转化为求解该无向权图从起点到终点的最短路径，采用动态规划模型尝试模拟最短路径。定义玩家在第 t 天从 pos 位置出发行走到任一其他地区 pos' 的这一时刻为该玩家当前状态 $\{t, pos\}$，则动态规划状态量可以写为这一时刻到终点的最短路径：$S_{\{t,pos\}}$, 状态转移方程可以从图中得到：

$$S_{\{t,pos\}} = S_{\{t',pos'\}} + D_{(\{t,pos\},\{t',pos'\})}, \quad t' > t \tag{86}$$

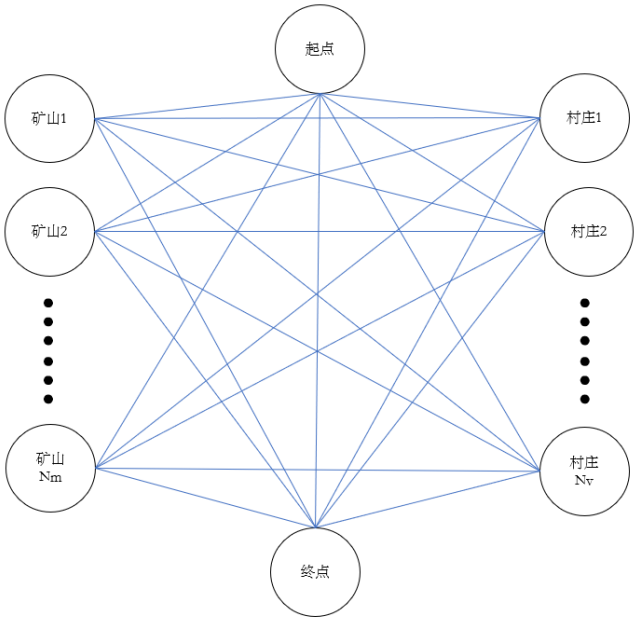

图 **8**　动态规划模型图

其中 $D_{(\{t,pos\},\{t',pos'\})}$ 指从 pos 前往 pos' 的最低消耗，可以从已知信息中计算得到：

$$\begin{aligned} D_{(\{t,pos\},\{t',pos'\})} = &\sum_{i=t}^{t'-1} 2\left(p_w c w_i + p_f c f_i\right) + \\ &\sum_{i=t}^{t'-t-d_{(pos,pos')}-1} \left(p_w c w_i + p_f c f_i\right) - g(t' - t - d_{(pos,pos')}) \end{aligned} \tag{87}$$

其中 $d_{(pos,pos')}$ 表示从 pos 行走至 pos' 所需的最短时间（单位为天），可以通过对地图的查看和对已知天气是否存在沙暴进行测算。

于是，求解最优解的路径就转化为求解 $S_{\{1,start\}}$。最小子问题即为 $S_{\{T,end\}} = 0, T = 1, 2, ..., T_m$。

参考文献

[1] John F. *Nash* Jr. PNAS . January 1, *1950* 36 (1) 48-49.

[2] Lemke C, Howson J. Equibrium Points of Bimatrix Games[J]. Journal of Society for Industrial and Applied Mathematics, 1964, 12: 413-423.

[3] McKelvey R, McLennan A. Computation of equilibria in finite games[A]. In: Handbook of computational Economics, Elsevier Science, 1996, 1: 87-142.

[4] 基于 0-1 规划的单 RGV 动态调度模型, 2018 年全国大学生数学建模竞赛论文展示 (B203).

附录 A　支撑文件说明

表 **4**　支撑文件的说明

支撑文件	文件说明	附录位置
Result.xlsx	记录第一关与第二关的最优解	附录 F
ShortestPath_1.m	刻画第一关中每个点之间的最短距离	附录 B
findAllPath.py	生成第一关和第二关从起点到终点的全部路径	附录 C
1and2_simulation.py	生成第一关和第二关的最优解	附录 D
1and2_analysis.py	问题一的验证程序	附录 E

表 **5**　支撑文件的运行环境

支撑文件	运行环境
Result.xlsx	略
ShortestPath_1.m	Matlab 2018b, 需要安装 Network Analysis and Visualization 工具箱
findAllPath.py	Python 3.7.0
1and2_simulation.py	Python 3.7.0, 需要外部模块 NumPy
1and2_analysis.py	Python 3.7.0, 需要外部模块 NumPy

附录 B

```
clc, clear
%给出图中各点相对位置
am(1,2)=1;am(1,25)=1;
am(2,1)=1;am(2,3)=1;
am(3,2)=1;am(3,4)=1;am(3,25)=1;
am(4,3)=1;am(4,5)=1;am(4,24)=1;am(4,25)=1;
am(5,4)=1;am(5,24)=1;am(5,6)=1;
am(6,5)=1;am(6,24)=1;am(6,7)=1;am(6,23)=1;
am(7,6)=1;am(7,8)=1;am(7,22)=1;
```

```
am(8,7)=1;am(8,9)=1;am(8,22)=1;
am(9,8)=1;am(9,22)=1;am(9,17)=1;am(9,10)=1;am(9,15)=1;am(9,16)=1;am(9,21)=1;
am(10,9)=1;am(10,15)=1;am(10,11)=1;am(10,13)=1;
am(11,10)=1;am(11,12)=1;am(11,13)=1;
am(12,11)=1;am(12,13)=1;am(12,14)=1;
am(13,10)=1;am(13,11)=1;am(13,12)=1;am(13,14)=1;am(13,15)=1;
am(14,12)=1;am(14,13)=1;am(14,15)=1;am(14,16)=1;
am(15,9)=1;am(15,10)=1;am(15,13)=1;am(15,14)=1;am(15,16)=1;
am(16,9)=1;am(16,14)=1;am(16,15)=1;am(16,17)=1;am(16,18)=1;
am(17,9)=1;am(17,16)=1;am(17,18)=1;am(17,21)=1;
am(18,17)=1;am(18,19)=1;am(18,16)=1;am(18,20)=1;
am(19,20)=1;am(19,18)=1;
am(20,19)=1;am(20,18)=1;am(20,21)=1;
am(21,17)=1;am(21,9)=1;am(21,22)=1;am(21,23)=1;am(21,27)=1;am(21,20)=1;
am(22,7)=1;am(22,8)=1;am(22,9)=1;am(22,21)=1;am(22,23)=1;
am(23,6)=1;am(23,22)=1;am(23,21)=1;am(23,24)=1;am(23,26)=1;
am(24,4)=1;am(24,6)=1;am(24,5)=1;am(24,23)=1;am(24,26)=1;am(24,25)=1;
am(25,1)=1;am(25,4)=1;am(25,3)=1;am(25,24)=1;am(25,26)=1;
am(26,27)=1;am(26,23)=1;am(26,24)=1;am(26,25)=1;
am(27,26)=1;am(26,21)=1;
am=am';
[i,j,v]=find(am);
b=sparse(i,j,v,27,27);%改变参量求最小路径
[x,y,z]=graphshortestpath(b,12,27,'Directed',false);
```

附录 C

```
adjMisson1 = [[0,6,0,0,0,0,0,0,3],[0,0,2,0,0,0,0,0,3],\
[0,0,0,2,0,0,0,0,0],[0,0,0,0,2,0,0,0,3],\
[0,0,0,0,0,2,0,0,0],[0,0,0,0,0,0,2,0,3],\
[0,0,0,0,0,0,0,2,0],[0,0,0,0,0,0,0,0,3]]
adjMisson2 = []

for i in range(16):
adjChild = []
for j in range(16):
adjChild.append(0)
adjMisson2.append(adjChild)

adjMisson2[0][1] = 7
adjMisson2[1][2] = 1
adjMisson2[2][3] = 1
adjMisson2[2][9] = 2
adjMisson2[2][15] = 3
adjMisson2[3][4] = 1
```

```
adjMisson2[4][5] = 1
adjMisson2[4][9] = 2
adjMisson2[4][15] = 3
adjMisson2[5][6] = 1
adjMisson2[6][7] = 1
adjMisson2[6][9] = 2
adjMisson2[6][15] = 3
adjMisson2[7][8] = 1
adjMisson2[8][9] = 2
adjMisson2[8][15] = 3
adjMisson2[9][10] = 2
adjMisson2[9][15] = 2
adjMisson2[10][11] = 1
adjMisson2[10][15] = 2
adjMisson2[11][12] = 1
adjMisson2[11][15] = 2
adjMisson2[12][13] = 1
adjMisson2[12][15] = 2
adjMisson2[13][14] = 1
adjMisson2[13][15] = 2
adjMisson2[14][15] = 2

def findAllPath(adjMatrix, start, end, path=[]):
path = path + [start]
if start == end:
return [path]

paths = []
for i in range(end + 1):
if (adjMatrix[start][i] > 0 and i not in path):
newPaths = findAllPath(adjMatrix, i, end, path)
for newPath in newPaths:
paths.append(newPath)

return paths

print(findAllPath(adjMisson1, 0, 8))
print(findAllPath(adjMisson2, 0, 15))
print(adjMisson1)
print(adjMisson2)
```

附录 D

```
from itertools import permutations, compress
import numpy as np
```

```
from math import floor

# global constant
WAIT,WALK,WORK = 1,2,3
SUN,HOT,SAND = 5,8,10
START,FACTORY,SHOP,END = 0,1,2,3
WEIGHT_BOUND = 1200

# global constant arrays: weather, pathS, graph
WATER_COST = [8, 8, 5, 10, 5, 8, 10, 5, 8, 8, 10, 8, 5, 8, 8, 8, 10, 10, 8, 8, 5, 5, 8, 5,
    10, 8, 5, 5, 8, 8]
FOOD_COST = [6, 6, 7, 10, 7, 6, 10, 7, 6, 6, 10, 6, 7, 6, 6, 6, 10, 10, 6, 6, 7, 7, 6, 7,
    10, 6, 7, 7, 6, 6]
weather = WATER_COST

# all possible paths
paths1 = [[0, 1, 2, 3, 4, 5, 6, 7, 8], [0, 1, 2, 3, 4, 5, 8], [0, 1, 2, 3, 8], [0, 1, 8],
    [0, 8]]
paths2 = [[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15], [0, 1, 2, 3, 4, 5, 6, 7,
    8, 9, 10, 11, 12, 13, 15], [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 15], [0, 1, 2, 3,
    4, 5, 6, 7, 8, 9, 10, 11, 15], [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 15], [0, 1, 2, 3, 4,
    5, 6, 7, 8, 9, 15], [0, 1, 2, 3, 4, 5, 6, 7, 8, 15], [0, 1, 2, 3, 4, 5, 6, 9, 10, 11,
    12, 13, 14, 15], [0, 1, 2, 3, 4, 5, 6, 9, 10, 11, 12, 13, 15], [0, 1, 2, 3, 4, 5, 6, 9,
    10, 11, 12, 15], [0, 1, 2, 3, 4, 5, 6, 9, 10, 11, 15], [0, 1, 2, 3, 4, 5, 6, 9, 10, 15],
    [0, 1, 2, 3, 4, 5, 6, 9, 15], [0, 1, 2, 3, 4, 5, 6, 15], [0, 1, 2, 3, 4, 9, 10, 11, 12,
    13, 14, 15], [0, 1, 2, 3, 4, 9, 10, 11, 12, 13, 15], [0, 1, 2, 3, 4, 9, 10, 11, 12, 15],
    [0, 1, 2, 3, 4, 9, 10, 11, 15], [0, 1, 2, 3, 4, 9, 10, 15], [0, 1, 2, 3, 4, 9, 15], [0,
    1, 2, 3, 4, 15], [0, 1, 2, 9, 10, 11, 12, 13, 14, 15], [0, 1, 2, 9, 10, 11, 12, 13, 15],
    [0, 1, 2, 9, 10, 11, 12, 15], [0, 1, 2, 9, 10, 11, 15], [0, 1, 2, 9, 10, 15], [0, 1, 2,
    9, 15], [0, 1, 2, 15]]
paths = [paths1, paths2]

# adjacent matrices
graph1 = [[0, 6, 0, 0, 0, 0, 0, 0, 3], [0, 0, 2, 0, 0, 0, 0, 0, 3], [0, 0, 0, 2, 0, 0, 0, 0,
    0], [0, 0, 0, 0, 2, 0, 0, 0, 3], [0, 0, 0, 0, 0, 2, 0, 0, 0], [0, 0, 0, 0, 0, 0, 2, 0,
    3], [0, 0, 0, 0, 0, 0, 0, 2, 0], [0, 0, 0, 0, 0, 0, 0, 0, 3]]
graph2 = [[0, 7, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0], [0, 0, 0, 1, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 3], [0, 0, 0, 0, 1, 0,
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 1, 0, 0, 0, 2, 0, 0, 0, 0, 0, 3], [0, 0,
    0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 1, 0, 2, 0, 0, 0, 0, 0,
    3], [0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0,
    0, 0, 0, 0, 3], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 2], [0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 1, 0, 0, 0, 2], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 2], [0, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 2], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
    2], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0]]
graphs = [graph1, graph2]
```

```
# key_vertices
hash1 = {0:START, 1:SHOP, 2:FACTORY, 3:SHOP, 4:FACTORY, 5:SHOP, 6:FACTORY, 7:SHOP, 8:END}
hash2 = {0:START, 1:FACTORY, 2:SHOP, 3:FACTORY, 4:SHOP, 5:FACTORY, 6:SHOP,7:FACTORY, 8:SHOP,
    9:FACTORY, 10:SHOP, 11:FACTORY, 12:SHOP, 13:FACTORY, 14:SHOP, 15:END}


# some helper functions
def raw_path_length(path, graph):
sum = 0
if graph == 1:
for i in range(len(path)-1):
sum += graph1[path[i]][path[i+1]]
else:
for i in range(len(path)-1):
sum += graph2[path[i]][path[i+1]]
return sum

def count_key_vertices(key,graph,path):
count = 0
if graph == 1:
for i in path:
if hash1[i] == key:
count += 1
else:
for i in path:
if hash2[i] == key:
count += 1
return count


class exhaust_k_days:
def __init__(self,k,path,graph):
self.k = k
self.path = path
self.graph = graph
self.basic_length = raw_path_length(self.path,self.graph)
# can be reduced by considering sand
self.space0 = list(permutations(list(range(k - self.basic_length)),
    count_key_vertices(FACTORY,graph,path)))


def rough_cleaning(self):
permutations_sum = [sum(i) for i in self.space0]
total_sand = weather[:self.k].count(SAND)
low, high = self.k - total_sand - self.basic_length, self.k - total_sand
mask = [True if i >= low and i <= high else False for i in permutations_sum]
```

```
self.space0 = list(compress(self.space0,mask))


class permutation:
def __init__(self,mine_days,k,raw_path,graph):
self.k = k
self.raw_path = raw_path
self.graph = graph
self.mine_days = mine_days
self.shop = []
self.end = []
self.cost = 0

def translate(self):
# _path: compute from the first day
_path = []
ptr_factory = 0
if self.graph == 1:
for i in range(1,len(self.raw_path)):
_path += [2]*graph1[self.raw_path[i-1]][self.raw_path[i]]
self.shop += [0]*graph1[self.raw_path[i-1]][self.raw_path[i]]
self.end += [0]*graph1[self.raw_path[i-1]][self.raw_path[i]]
if hash1[self.raw_path[i]] == SHOP:
self.shop[-1] = 1
elif hash1[self.raw_path[i]] == FACTORY:
_path += [3]*self.mine_days[ptr_factory]
self.shop += [0]*self.mine_days[ptr_factory]
self.end += [0]*self.mine_days[ptr_factory]
ptr_factory += 1
else:
for i in range(1,len(self.raw_path)):
_path += [2]*graph2[self.raw_path[i-1]][self.raw_path[i]]
self.shop += [0]*graph2[self.raw_path[i-1]][self.raw_path[i]]
self.end += [0]*graph2[self.raw_path[i-1]][self.raw_path[i]]
if hash2[self.raw_path[i]] == SHOP:
self.shop[-1] = 1
elif hash2[self.raw_path[i]] == FACTORY:
_path += [3]*self.mine_days[ptr_factory]
self.shop += [0]*self.mine_days[ptr_factory]
self.end += [0]*self.mine_days[ptr_factory]
ptr_factory += 1
self.days = _path
self.end[-1] = 1

def wait_insert(self):
flag = 1
for k in range(6):
```

```
if not flag or len(self.days) > self.k:
break
for i,state in enumerate(self.days):
flag = 0
try:
if state == WALK and weather[i] == SAND:
self.days.insert(i,1)
self.shop.insert(i,0)
self.end.insert(i,0)
flag = 1
break
except:
print(i,len(self.days))

def length_check(self):
return self.k == len(self.days)

def weight_check(self,x):
np_days = np.array(self.days)
r_water = (np_days * np.array(WATER_COST[:self.k]))[::-1]
r_food = (np_days * np.array(FOOD_COST[ :self.k]))[::-1]

r_shop = self.shop[::-1]
r_weight = r_water * 3 + r_food * 2

shop_num = sum(self.shop)

water_block = []
food_block = []
low = 0
for c in range(shop_num):
high = r_shop.index(1)
r_shop[high] = 0
tmp_water = r_water[low:high].sum()
tmp_food = r_food[low:high].sum()
tmp = tmp_water*3 + tmp_food*2

if tmp > WEIGHT_BOUND:
return False
else:
water_block.append(tmp_water)
food_block.append(tmp_food)
low = high

tmp = r_weight[low:].sum()
tmp_water = r_water[low:].sum()
tmp_food = r_food[low:].sum()
```

```
y = floor((1200 - 3*x)/2)
if tmp > WEIGHT_BOUND or x < tmp_water or y < tmp_food:
return False
else:
water_block.append(tmp_water)
food_block.append(tmp_food)

water_block = water_block[::-1]
food_block = food_block[::-1]

if x < y:
for i in range(len(food_block)-1):
if y - food_block[i] > food_block[i+1]:
if (y - food_block[i])*2 + water_block[i+1]*3 > 1200:

return False
y = y - food_block[i]
y = floor((1200 - 3*x)/2)
else:
for i in range(len(water_block)-1):
if x - water_block[i] > water_block[i+1]:
if (x - water_block[i])*2 + food_block[i+1]*3 > 1200:
return False
x = x - water_block[i]

return True



def total_cost(self,x):
y = floor((1200 - 3*x)/2)
cost_0 = x*5 + y*10
np_days = np.array(self.days)
return cost_0 + ((np_days*np.array(WATER_COST[:self.k])).sum()-x)*10 +
    ((np_days*np.array(FOOD_COST[:self.k])).sum()-y)*20

def total_gain(self):
return sum([1 if i == 3 else 0 for i in self.days])*1000


if __name__ == "__main__":
print("================================================")
print("Graph 1")
print("================================================")
for path in paths1:
print("Finding optimal solution for one path......")
path_clean_gain = 0
```

```
path_final_path = []
path_x = 0

for k in range(20,31):
a = exhaust_k_days(k,path,1)
a.rough_cleaning()
k_clean_gain = 0
k_final_path = []
k_x = 0

for perm in a.space0:
b = permutation(perm,k,path,1)
b.translate()
b.wait_insert()
perm_clean_gain = -100000
perm_final_path = []
perm_x = 0

if b.length_check() and len(b.end) == k and count_key_vertices(FACTORY,1,path):
perm_gain = b.total_gain()
perm_min_cost = 1000000
for x in range(98,241):
if b.weight_check(x):
x_min_cost = b.total_cost(x)
if x_min_cost < perm_min_cost:
perm_min_cost = x_min_cost
perm_clean_gain = perm_gain - x_min_cost
perm_final_path = b.days
perm_x = x

if perm_clean_gain > k_clean_gain:
k_clean_gain = perm_clean_gain
k_final_path = perm_final_path
k_x = perm_x



if k_clean_gain > path_clean_gain:
path_clean_gain = k_clean_gain
path_final_path = k_final_path
path_x = k_x


if (path_clean_gain != 0):
print("For path",path,":")
print("Total gain = ",path_clean_gain+10000,",Total day = ", len(path_final_path),",Initial
    x = ", path_x)
```

```
print()
else:
print("NO SOLUTION")
print()


print("==================================================")
print("Graph 2")
print("==================================================")
for path in paths2:
print("Finding optimal solution for one path......")
path_clean_gain = 0
path_final_path = []
path_x = 0

for k in range(18,31):
a = exhaust_k_days(k,path,2)
a.rough_cleaning()
k_clean_gain = 0
k_final_path = []
k_x = 0
k_perm = 0

for perm in a.space0:
# print("============================================================")
b = permutation(perm,k,path,2)
b.translate()
b.wait_insert()
perm_clean_gain = -100000
perm_final_path = []
perm_x = 0

if b.length_check() and len(b.end) == k and count_key_vertices(FACTORY,2,path):
perm_gain = b.total_gain()
perm_min_cost = 1000000
for x in range(130,318):
if b.weight_check(x):
x_min_cost = b.total_cost(x)
if x_min_cost < perm_min_cost:
perm_min_cost = x_min_cost
perm_clean_gain = perm_gain - x_min_cost
perm_final_path = b.days
perm_x = x



if perm_clean_gain > k_clean_gain:
```

```
k_clean_gain = perm_clean_gain
k_final_path = perm_final_path
k_x = perm_x
k_perm = perm
# print(k_clean_gain,k_final_path,len(k_final_path),k_x)



if k_clean_gain > path_clean_gain:
path_clean_gain = k_clean_gain
path_final_path = k_final_path
path_x = k_x
path_perm = k_perm
# print(path_clean_gain,path_final_path,len(path_final_path),path_x)


if (path_clean_gain != 0):
print("For path",path,":")
print("Total gain = ",path_clean_gain+10000,",Total day = ", len(path_final_path),",Initial
    x = ", path_x)
print()
else:
print("NO SOLUTION")
print()
```

附录 E

```
import numpy as np
WATER_COST = [8, 8, 5, 10, 5, 8, 10, 5, 8, 8, 10, 8, 5, 8, 8, 8, 10, 10, 8, 8, 5, 5, 8, 5,
    10, 8, 5, 5, 8, 8]
FOOD_COST = [6, 6, 7, 10, 7, 6, 10, 7, 6, 6, 10, 6, 7, 6, 6, 6, 10, 10, 6, 6, 7, 7, 6, 7,
    10, 6, 7, 7, 6, 6]
np_water = np.array(WATER_COST)
np_food = np.array(FOOD_COST)

def second_average_cost_compute(path,d1,dm,d2,d2_,d3,T):
np_path = np.array(path)
second_cost = (5*np_path*np_water[:T] + 10*np_path*np_food[:T])[d1:T-d3+1].sum()
second_average_cost = second_cost/((T-d3+1)-d1-1)
print(second_cost,(T-d3+1)-d1-1,second_average_cost)

# 关卡一
path = [2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 3, 3, 3, 3, 3, 3, 3, 1, 2, 2, 2, 2, 2]
d1,dm,d2,d2_,d3,T = 10,7,2,1,3,23
second_average_cost_compute(path,d1,dm,d2,d2_,d3,T)
```

```
# 关卡二
path = [2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 1, 2, 3, 3, 3, 3, 3, 3, 2, 2, 2, 3, 3, 3, 3, 3, 3, 3,
    2, 2]
d1,dm,d2,d2_,d3,T = 9,13,5,1,2,30
second_average_cost_compute(path,d1,dm,d2,d2_,d3,T)
```

附录 F　关卡一、二的结果

表 **6**　第一关玩家策略

日期	所在区域	G	w	f	m
0	1	5800	180	330	1200
1	25	5800	164	318	1128
2	26	5800	148	306	1056
3	23	5800	138	292	998
4	23	5800	128	282	948
5	22	5800	118	268	890
6	9	5800	102	256	818
7	9	5800	92	246	768
8	15	4170	245	232	1199
9	13	4170	229	220	1127
10	12	4170	213	208	1055
11	12	5170	183	178	905
12	12	6170	159	160	797
13	12	7170	144	139	710
14	12	8170	120	121	602
15	12	9170	96	103	494
16	12	10170	72	85	386
17	12	11170	42	55	236
18	12	11170	32	45	186
19	14	11170	16	33	114
20	15	10430	36	40	188
21	9	10430	26	26	130
22	21	10430	16	12	72
23	27	10430	0	0	0

表 **7**　第二关玩家策略

日期	所在区域	剩余资金数 (元)	剩余水量 (箱)	剩余食物量 (箱)	玩家负载 (千克)
0	1	5300	130	405	1200
1	2	5800	114	393	1128
2	3	5300	98	381	1056
3	4	5300	88	367	998
4	4	5300	78	357	948
5	5	5300	68	343	890
6	13	5300	52	331	818
7	13	5300	42	311	748
8	22	5300	32	304	704
9	30	5300	16	292	632
10	39	5200	10	280	590
11	39	3410	179	270	1077
12	30	3410	163	258	1005
13	30	4410	148	237	918
14	30	5410	124	219	810
15	30	6410	100	201	702
16	30	7410	76	183	594
17	30	8410	46	153	444
18	30	9410	16	123	294
19	39	5730	199	199	995
20	47	5730	183	187	923
21	55	5730	173	173	865
22	55	6730	163	159	807
23	55	7730	139	141	699

表 **8**　第二关玩家策略

日期	所在区域	剩余资金数 (元)	剩余水量 (箱)	剩余食物量 (箱)	玩家负载 (千克)
24	55	8730	124	120	612
25	55	9730	94	90	462
26	55	10730	70	72	354
27	55	11730	55	51	267
28	55	12730	40	30	180
29	63	12730	16	12	72
30	64	12730	0	0	0