摘要

本文基于多背包优化模型，设计**沙石算法**，优化部队的装载方案。再通过**遗传算法**与**泊口仿真模型**，选择出装载方案中装载时间最少，需要船支数量最少的装载方案。最后在**泊口仿真模型**中添加港口摧毁事件，再次使用使用遗传算法求解出港口被摧毁对应的最佳装载方案。

针对问题一，将原问题转化成多背包优化问题，设计**沙石算法**，优化部队装载方案。根据题目要求，首先对兵力、装备与船载的面积进行量化，得到相关的有效面积值。然后建立船舰数量的决策向量，以调用船支数量作为目标函数，题中船舰装载限制作为约束条件，建立**部队装载优化模型**。设计**沙石算法**，进行装备均分和部队装载，求解得到各旅级单位装备人口的装载方案和各类舰船的使用数量。各类船舰的有效面积利用率基本达到 **90%** 以上，共使用船舰数 **236 艘**。

针对问题二，调整问题一中的**沙石算法**作为验证算法，并搭建**泊口仿真模型**计算每种装载方案需要的装载时间作为适应度函数，代入**遗传算法**求得最佳装载方案。本组首先基于问题一中设计的**沙石算法**，编译验证算法，以验证每种装载方案是否能装载全体部队。然后设计**泊口仿真模型**，使每个港口以最大工作效率工作，计算用时最长泊口装载所用时间作为总装载时间。再利用遗传算法，将每种民用船调用量作为染色体，搜素总装载时间最少，使用总船支数量最少的调用方案。最优方案为军用登陆舰**全部调用**，民用船调用方案为 2 万吨级滚装船 **15 艘**，3 万吨级滚装船 **5 艘**，集装箱船 **1 艘**，杂货船 **1 艘**和客船 **1 艘**。装载时间为 **20 小时**，共使用船舰 **364 艘**。

针对问题三，基于问题二中的模型，延用遗传算法，并调整**泊口仿真模型**，增加摧毁泊口事件，求解出泊口被摧毁对应的最佳装载方案。首先调整**泊口仿真模型**，当运行时间等于攻击时间时，停止被摧毁港口正在进行的任务，并将任务进度清零，并在之后的检索列表中剔除被摧毁泊口。重新运行遗传算法以寻找港口被摧毁对应的最佳装载方案。最优方案为军用登陆舰**全部调用**，民用船调用方案为 2 万吨级滚装船 **18 艘**，3 万吨级滚装船 **3 艘**，集装箱船 **1 艘**，杂货船 **1 艘**和客船 **1 艘**。装载时间为 **20 小时**，共使用船舰 **365 艘**。

本文中所提到的模型优点主要有两点：一、使用沙石算法与泊口仿真模型，使得军舰与泊口的利用率达到较高水平，最终方案转载时间少，使用船支数量少；二、利用遗传算法优化装载方案，鲁棒性强，全局搜素能力强。

关键词： 沙石算法　遗传算法　泊口仿真模型　部队装载优化模型

目录

一、 问题重述

1.1 问题背景

随着全球化的进程，人类活动范围日益扩大，人群流动频繁，传染病可在大范围内迅速传播，是对人类社会存在威胁的公共卫生问题。在疾病控制实际工作中，疾病的发病与流行趋势分析是极其重要的一环，科学、准确的分析能对卫生行政部分制定疾病预防与控制策略产生重要的影响，传染病早期预警将大大降低传染病的社会经济危害。

为了提高某传染病疫情和突发公共卫生事件报告的质量和时效，加强对全国感染病人的诊断、治疗和督导管理，卫生部建立了全国监管机制, 及时通报相关病情和相关数据, 并通过对疫情数据的动态分析，建立该传染病防治工作督导检查、防治效果评价和制定防治对策和策略，控制并逐渐消灭该传染病。构建预测模型从早期探测到传染病的爆发并及时预警，采取应对措施，是目前传染病防控的重要手段，具有重要的实际意义。

1.2 问题概述

围绕相关附件和条件要求，研究海运装载行动输送兵力任务的合理安排，依次提出以下问题:

问题一：根据合适的指标建立模型，分析流行病在 2004-2016 年的变化趋势，并预测 2019 年全国感染该病的发病数和死亡数。

问题二：基于 2004-2016 年每隔三年的不同地区的和职业分类的数据，建立疾病传播模型，并预测 2019 年传染病重点防控前 3 名的区域和职业人群。

问题三：结合地区经济发展的相关数据，选择一个角度建立传染病与经济发展相关的模型，并分析结论。

问题四: 综合模型结果及分析，给卫生健康委员会相关部门写一封公开信，谈谈对传染病疫情防治的看法和建议。

二、 模型假设

(1) 为保证预测结果精确性，假设题目所给出数据真实可信。
(2) 为了优化运算结果，假设所有全副武装的士兵都保持坐姿休息。
(3) 假设在战争中，装载消耗更少时间的优先度高于使用更少的民用船。
(4) 假设在装载过程中，同一泊口的船舰装载交替时间可以忽略不记。
(5) 假设港口被摧毁时，由于提前得到信息，港口上的船只与兵力没有损失，只是正在进行的装载工作停止，且进度完全损失。

三、 符号说明

符号	说明
X_i	第 i 型装备
s_{xi}	装备 X_i 的占用面积
l_i	装备 X_i 的长
w_i	装备 X_i 的宽
ε_i	装备 X_i 的面积修正系数
p_i	全副武装人员数
eta_1	海军船舰的有效面积率
eta_2	民用船只的有效面积率
y_k	登陆舰数量
z_n	民用船数量
D	决策向量
S_D	舰载面积系数向量
S_z	民用船有效面积向量
S_y	登陆舰有效面积向量
S_p	每营人口面积向量
A^k	面积规划向量
T_n	颖编制总部对数列
P_n	船舰面积数列
η	平均有效面积利用率

四、 问题一模型的建立与求解

4.1 问题描述与分析

问题一要求，根据附件中 2004 年至 2016 年的流行病相关数据，预测 2019 年全国感染该疾病的发病人数和死亡人数。本组首先基于灰色预测，预测分析 2004-2019 年该流行病的发病人数和死亡人数。再通过马尔科夫模型，由 2004-2016 年的数据模拟残差在各个区间的分布，计算 2017-2019 年预测残差的期望值。最后将预测结果与残差期望做差，校正传统灰色预测的固有偏差。

4.2 模型的建立

4.2.1 灰度预测 GM(1,1)

设 2004-2016 年总发病人数为时间序列：

$$X^{(0)} = [x^{(0)}(1), x^{(0)}(2), \cdots, x^{(0)}(13)] \tag{1}$$

通过一次累加生成 1-AGO 序列:

$$X^{(1)} = [x^{(1)}(1), x^{(1)}(2), \cdots, x^{(1)}(13)] \tag{2}$$

式中：$x^{(1)}(k) = \sum_{i=1}^{k} x^{(1)}(i), k = 1, 2, \cdots, 13$。根据 1-AGO 序列建立微分方程为：

$$\frac{dX^{(1)}}{dt} + aX^{(1)} = u \tag{3}$$

其中，a 称为发展灰度，u 称为内生控制灰度。设 $\widehat{\alpha}$ 为待估参数向量，且 $\widehat{\alpha} = [a, u]^T$, 利用最小二乘法求出：

$$\widehat{\alpha} = (B^T B)^{-1} B^T Y_n \tag{4}$$

求解方程 (3), 可得第 $k+1$ 年发病人数预测为：

$$\widehat{X}(k+1) = [X^{(0)}(1) - \frac{u}{a}]e^{-ak} + \frac{u}{a}, k = 1, 2, \cdots, 16 \tag{5}$$

同理将死亡数作为向量 $X^{(0)} = [x^{(0)}(1), x^{(0)}(2), \cdots, x^{(0)}(13)]$ 带入模型可求得 2017-2019 年死亡数灰度预测值。

4.2.2 马尔科夫模型校正

利用马尔科夫模型对 GM(1,1) 预测误差项的状态及状态概率进行预估，并利用预测状态的期望值对 GM(1,1) 预测值进行修正。用 2004-2016 年预测数据与真实数据残差进行状态划分，设残差序列为：

$$\varepsilon = [\varepsilon(1), \varepsilon(2), \cdots, \varepsilon(13)] \tag{6}$$

最大残差绝对值为 $\delta_{max} = \max\limits_{1\leqslant i\leqslant 13} |\varepsilon(i)|$，将预测误差化均分为三个状态。令 $\lambda = \frac{\delta_{max}}{6}$。状态分别为 $E_1 : (-3\lambda, -\lambda)$、$E_2 : (-\lambda, \lambda)$ 和 $E_1 : (\lambda, 3\lambda)$。其中初始状态概率向量计算公式为：

$$t_0 = [p_{E1}, p_{E2}, p_{E3}] \tag{7}$$

$$p_{Ek} = \frac{n_{Ek}}{13} \tag{8}$$

式中 n_{Ek} 是状态 E_k 在 2004-2016 年内出现的次数，以状态 E_k 出现的频率代替其出现的概率 p_{Ek}。且构建状态转移矩阵为：

$$P = \begin{pmatrix} P_{11} & P_{12} & P_{13} \\ P_{21} & P_{22} & P_{23} \\ P_{31} & P_{32} & P_{33} \end{pmatrix} \tag{9}$$

其中 P_{ij} 是由状态 E_i 经过一个时期转移到 E_j 的转移概率。即马尔科夫模型可表示为：

$$t_{k+1} = t_k \cdot p \tag{10}$$

设状态区间的中间值分别为 $\overline{E}_1$、$\overline{E}_2$ 和 $\overline{E}_3$，即第 k 年 GM(1,1) 的误差期望为：

$$\eta = \begin{bmatrix} p_{E1} & p_{E2} & p_{E3} \end{bmatrix} \cdot \begin{bmatrix} \overline{E}_1 \\ \overline{E}_2 \\ \overline{E}_3 \end{bmatrix} \tag{11}$$

设第 k 年的患病人数的 GM(1,1) 预测值为 $\widehat{x}(k)$, 修正后的组合预测值 $\overline{x}(k)$ 可以记作：

$$\overline{x}(k) = \widehat{x}(k) - \eta \tag{12}$$

4.2.3 预测结果评价指标

均方根误差 (RMSE)、平均相位误差绝对值 (MAPE) 和纳什效率系数 (NSE) 三者是常用来衡量预测结果的指标。RMSE 能评价患病人数和死亡人数中高值的预测结果，其计算公式为：

均方根误差越小，表明模型可靠性越高，结果越准确。MAPE 用来评价预测数据中平稳部分的预测结果，其计算公式为：

MAPE 所求值为绝对值，是一个相对指标，当两个 MAPE 值进行比较时，值越小的说明模型可靠性越高。NSE 可以用来评价模型的预测能力，其计算公式如下：

求得 NSE 值越接近 1，表示模型质量越好，模型可信度越高。接近 0，表示模拟结果接近观测值的平均水平，即总体结果可信，但模拟误差较大。远远小于 0，则模型是不可信的。

4.3 模型的求解

通过 GM(1,1) 计算 2004-2016 年发病人数预测值得到灰度预测解如下：

$$\widehat{X}(k+1) = -2527359e^{-0.037k} + 3497638, k = 1, 2, \cdots, 16 \tag{13}$$

其误差状态区间如表 1 所示：

表 **1**　发病人数状态区间划分

状态	E_1	E_2	E_3
相对误差区间	$(-66389, -22130]$	$(-22130, 22130)$	$(22130, 66389]$

根据误差区间范围，将 2004-2016 年发病人数预测值归类于误差区间如表 2 所示：

表 **2**　发病人数误差状态区间

年份	2004	2005	2006	2007	2008	2009	2010	2011	2012	2013	2014	2015	2016
所处状态区间	E_2	E_2	E_1	E_2	E_3	E_2	E_1	E_1	E_2	E_2	E_2	E_2	E_2

由此求得初始状态概率向量 t_0, 转移矩阵 P 为:

$$t_0' = [3/13, 9/13, 1/13] \tag{14}$$

$$P' = \begin{pmatrix} 1/3 & 2/3 & 0 \\ 1/4 & 5/8 & 1/8 \\ 0 & 1 & 0 \end{pmatrix} \tag{15}$$

得到由灰色预测与马尔科夫校正后预测解如图 1 所示

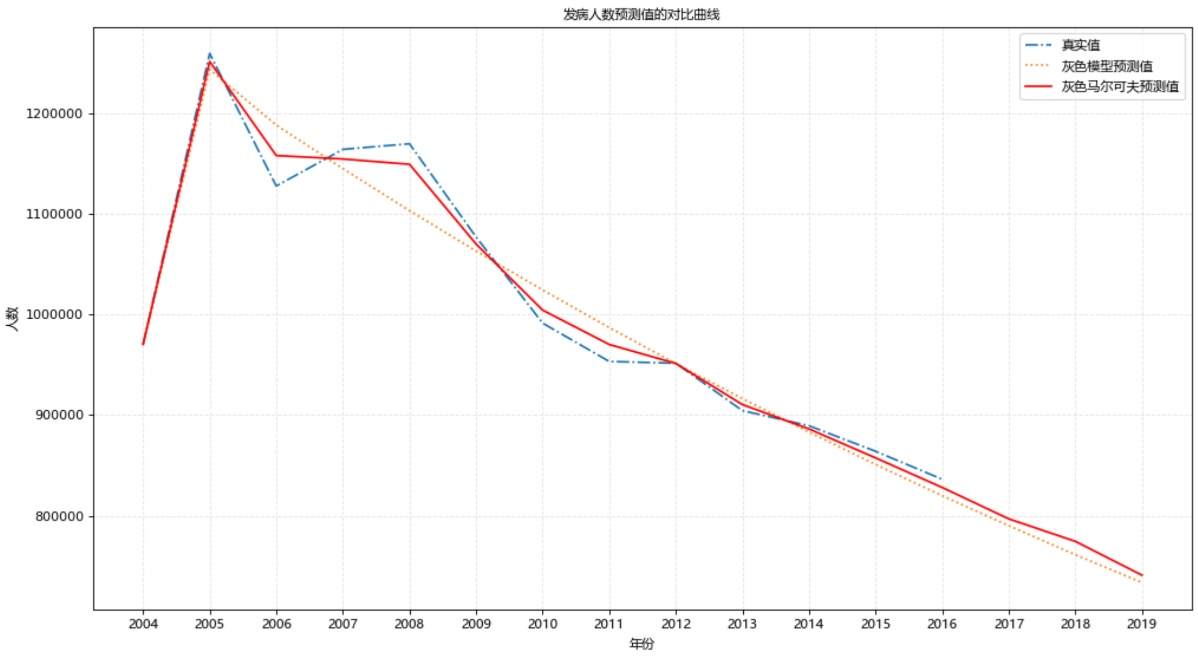

图 **1**　发病人数预测对比曲线

同理，计算 2004-2016 年死亡人数预测值得到灰度预测解如下：

$$\widehat{X}(k+1) = -92315e^{-ak} + 93750, k = 1, 2, \cdots, 16 \tag{16}$$

其误差状态区间如表 3 所示：

表 **3**　死亡数状态区间划分

状态	E_1	E_2	E_3
相对误差区间	$(-684, -228]$	$(-228, 228)$	$(228, 684]$

将发病人数预测值归类于误差区间如表 4 所示：

表 **4**　死亡人数误差状态区间

年份	2004	2005	2006	2007	2008	2009	2010	2011	2012	2013	2014	2015	2016
所处状态区间	E_2	E_2	E_2	E_3	E_1	E_3	E_2	E_2	E_2	E_2	E_1	E_2	E_2

求得初始状态概率向量 t_0', 转移矩阵 P' 为:

$$t_0' = [2/13, 9/13, 2/13] \tag{17}$$

$$P' = \begin{pmatrix} 0 & 1/2 & 1/2 \\ 1/8 & 3/4 & 1/8 \\ 1/2 & 1/2 & 0 \end{pmatrix} \tag{18}$$

得到由灰色预测与马尔科夫校正后预测解如图 2 所示

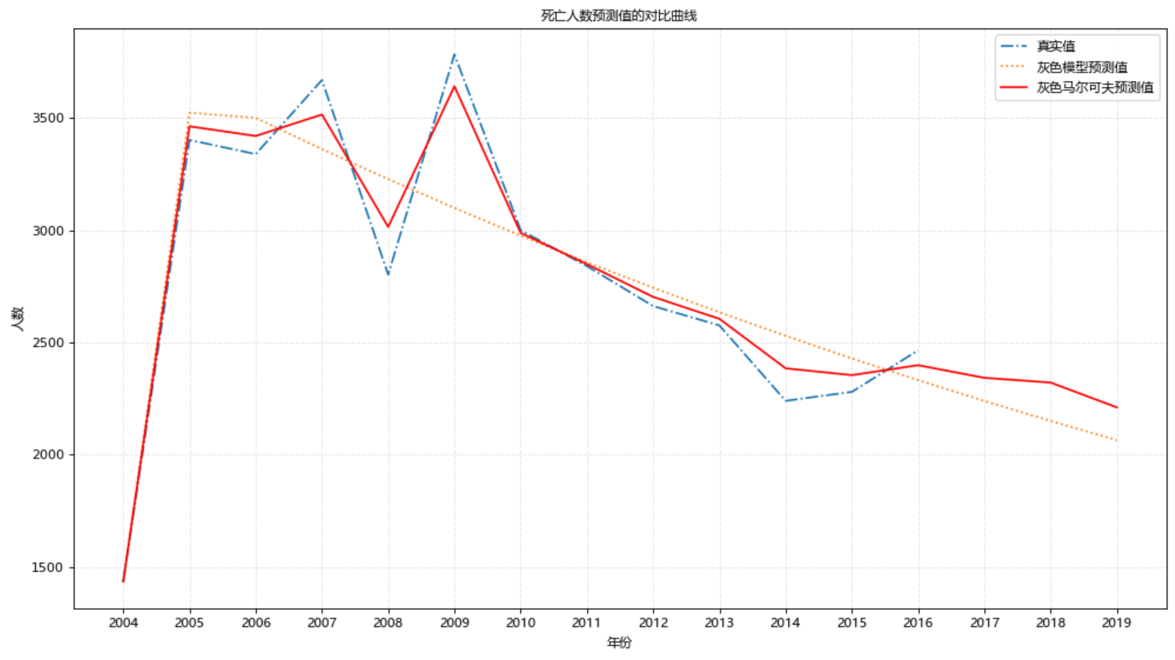

图 **2**　死亡人数预测对比曲线

4.4 结果分析

由图像直观对比可知由马尔科夫模型校正后的预测值相较于普通灰色预测值更加贴近实际值，且两种模型预测指标如表 **??** 所示：

五、 模型的评价

5.1 模型的优点

(1) 将多目标优化转换成背包优化问题，自主设计沙石算法，使得船舰的面积利用率高，对船只数量的需求量少。
(2) 使用遗传算法，具有很强的全局搜索能力和鲁棒性，运算时间远小于全遍历算法。

5.2 模型的缺点

遗传算法初始解由卡特蒙洛法随机生成，每次搜索结果不完全相同，可能引起结果的偏差，需要多搜索几次选择才能得到最优结果。

5.3 模型改进

可使用改进的生命遗传算法，加强算法的局部搜索能力，解决算法早熟的问题。

参考文献

[1] Febri Liantoni,Rifki Indra Perwira,Syahri Muharom,Riza Agung Firmansyah,Akhmad Fahruzi. Leaf classification with improved image feature based on the seven moment invariant[J]. Journal of Physics: Conference Series,2019,1175(1).

附录 A 代码

A.1 问题一沙石算法–python 源代码

```
import math
import numpy as np
import pandas as pd


def append(list,x):
'''numpy数组中插入一个元素在末尾'''
return np.append(list,x)


def Merge(dict1, dict2):
'''字典合并'''
res = {**dict1, **dict2}
return res


def shell_sort(lists):
'''
按从大到小排序，时间复杂度O(n)
:param lists:传进来的数组，比如船只或者包裹
:return: 返回从大到小的包裹
'''
# 希尔排序
lists = sorted(lists,reverse=True)
return lists


def kv_sort(kv):
'''
键值对由大到小排序
eg:
d = {'d1':2, 'd2':4, 'd4':1,'d3':3,}
res = sorted(d.items(),key=lambda d:d[1],reverse=True)
[('d2', 4), ('d3', 3), ('d1', 2), ('d4', 1)]
:param kv:键值对
:return:元组
'''
res = sorted(kv.items(), key=lambda d: d[1], reverse=True)
return res


def min_index(lists):
```

```
'''
返回最穷连的下标
:param lists: 各连的装备面积数目
:return: 最穷的那个连的下标
'''
index_min = 0
for i in range(1, len(lists)):
if lists[i] < lists[index_min]:
index_min = i
return index_min


def ship_list2dict(sum_ship):
'''
船变字典
:param sum_ship:船的列表
:return: 船的字典
'''
# 船只变成列表
# 列表转字典
dict_ship = {}
for i in range(len(sum_ship)):
if i < 5:
dict_ship["Y1_" + str(i)] = sum_ship[i]
elif i < 8:
dict_ship["Y2_" + str(i - 5)] = sum_ship[i]
elif i < 17:
dict_ship["Y3_" + str(i - 8)] = sum_ship[i]
elif i < 21:
dict_ship["Y4_" + str(i - 17)] = sum_ship[i]
elif i < 32:
dict_ship["Y5_" + str(i - 21)] = sum_ship[i]
elif i < 42:
dict_ship["Y6_" + str(i - 32)] = sum_ship[i]
elif i < 43:
dict_ship["Y7_" + str(i - 42)] = sum_ship[i]
elif i < 44:
dict_ship["Y8_" + str(i - 43)] = sum_ship[i]
elif i < 75:
dict_ship["Y9_" + str(i - 44)] = sum_ship[i]
elif i < 87:
dict_ship["Y10_" + str(i - 75)] = sum_ship[i]
elif i < 97:
dict_ship["Y11_" + str(i - 87)] = sum_ship[i]
elif i < 297:
dict_ship["Y12_" + str(i - 97)] = sum_ship[i]
elif i < 333:
```

```
dict_ship["Y13_" + str(i - 297)] = sum_ship[i]
elif i < 341:
dict_ship["Y14_" + str(i - 333)] = sum_ship[i]
return dict_ship


def sum_ship_list(ship_S, ship_num):
'''
计算所有的船只数目/装备数目
:param ship_S: 船只面积/装备面积
:param ship_num: 各类船只数目/装备数目
:return: 所有用有船只面积/装备面积
'''
sum_ship = []
if(len(ship_num)==len(ship_S)):
for i in range(len(ship_num)):
for j in range(ship_num[i]):
sum_ship.append(ship_S[i])
else:
print('船数与船面积的两个列表不相等，可能是打错了\n')
return sum_ship


def zhuang_baoguo(zhuangbei_num_list, zhuangbei_S_list, lian_num, lian_mianji, type):
'''
给出装备数目，装备面积，要求*均分*给每个连
均分打包思想：谁穷我给谁装备：
装备我从小到大排序；
for 装备的数目：
装备先给连j；
判断哪个连最穷
给穷的连装备

:param zhuangbei_num_list: 各个装备的数目
:param zhuangbei_S_list: 不同装备的面积
:param lian_num: 连的数量
:param lian_mianji: 连中人的面积
:param type: 旅的类型，键
:return: 装备打包好了后，下一步就是上船
'''

#初始化连的数目
lian = [0.0]*lian_num

#c初始化装备数目
zhuangbei = sum_ship_list(zhuangbei_S_list, zhuangbei_num_list)
#装备排序
```

```
zhuangbei = shell_sort(zhuangbei)
# print("总的装备的排序\n"+str(zhuangbei)+'\n')

for i in range(len(zhuangbei)):
# 最穷的下标
index_min = min_index(lian)
lian[index_min] += zhuangbei[i]

for i in range(len(lian)):
lian[i] += lian_mianji
# print(type+str(lian))

#列表转字典
dict_lian = {}
for i in range(len(lian)):
dict_lian[type+str(i)]=lian[i]
return dict_lian



def recode_zhuangzai(baoguo_k, ship_k, baoguo_v,zhuangzai_Map):
'''记录每一次装载到哪里了'''
if baoguo_k.startswith('一'):
if ship_k not in zhuangzai_Map[0]:
zhuangzai_Map[0][ship_k] = baoguo_v
else:
zhuangzai_Map[0][ship_k] += baoguo_v
elif baoguo_k.startswith('二'):
if ship_k not in zhuangzai_Map[1]:
zhuangzai_Map[1][ship_k] = baoguo_v
else:
zhuangzai_Map[1][ship_k] += baoguo_v
elif baoguo_k.startswith('三'):
if ship_k not in zhuangzai_Map[2]:
zhuangzai_Map[2][ship_k] = baoguo_v
else:
zhuangzai_Map[2][ship_k] += baoguo_v
elif baoguo_k.startswith('四'):
if ship_k not in zhuangzai_Map[3]:
zhuangzai_Map[3][ship_k] = baoguo_v
else:
zhuangzai_Map[3][ship_k] += baoguo_v
elif baoguo_k.startswith('五'):
if ship_k not in zhuangzai_Map[4]:
zhuangzai_Map[4][ship_k] = baoguo_v
else:
zhuangzai_Map[4][ship_k] += baoguo_v
elif baoguo_k.startswith('六'):
```

```
if ship_k not in zhuangzai_Map[5]:
zhuangzai_Map[5][ship_k] = baoguo_v
else:
zhuangzai_Map[5][ship_k] += baoguo_v
elif baoguo_k.startswith('七'):
if ship_k not in zhuangzai_Map[6]:
zhuangzai_Map[6][ship_k] = baoguo_v
else:
zhuangzai_Map[6][ship_k] += baoguo_v
elif baoguo_k.startswith('八'):
if ship_k not in zhuangzai_Map[7]:
zhuangzai_Map[7][ship_k] = baoguo_v
else:
zhuangzai_Map[7][ship_k] += baoguo_v
elif baoguo_k.startswith('九'):
if ship_k not in zhuangzai_Map[8]:
zhuangzai_Map[8][ship_k] = baoguo_v
else:
zhuangzai_Map[8][ship_k] += baoguo_v
elif baoguo_k.startswith('十'):
if ship_k not in zhuangzai_Map[9]:
zhuangzai_Map[9][ship_k] = baoguo_v
else:
zhuangzai_Map[9][ship_k] += baoguo_v
elif baoguo_k.startswith('eleven'):
if ship_k not in zhuangzai_Map[10]:
zhuangzai_Map[10][ship_k] = baoguo_v
else:
zhuangzai_Map[10][ship_k] += baoguo_v
elif baoguo_k.startswith('twelve'):
if ship_k not in zhuangzai_Map[11]:
zhuangzai_Map[11][ship_k] = baoguo_v
else:
zhuangzai_Map[11][ship_k] += baoguo_v
return zhuangzai_Map


def zhuangzai_ship(sum_baoguo, sum_ship,zhuangzai_Map):
'''
第一问核心思想
装船的操作，按石头装满装沙子的思想
:param sum_baoguo: 总的包裹
:param sum_ship: 总的船只
:param zhuangzai_Map:装载图
:return: 装载后的船只,装载图
'''
mod_dist = {}
```

```
for baoguo_k,baoguo_v in sum_baoguo.items():
flag = False
x=0
for ship_k,ship_v in sum_ship.items():
#如果包裹量小于等于第j艘船的量，装入第j艘船中
x+=1
# if sum_baoguo[i] <= sum_ship[j]:
if baoguo_v <= ship_v:
# sum_ship[j] -= sum_baoguo[i] #第j量船的载容量减少
flag=True
zhuangzai_Map = recode_zhuangzai(baoguo_k, ship_k , baoguo_v, zhuangzai_Map) #记录map中的位置
sum_ship[ship_k] -= baoguo_v
# print(baoguo_k+"装到"+ship_k)
break  #开始装第二个包裹
else:
pass   #否则开启第二只船
if len(sum_ship.items())==x and flag==False:
print(baoguo_k+"没有船可装"+str(baoguo_v))
#收集起来
mod_dist[baoguo_k] = baoguo_v
return sum_ship,zhuangzai_Map,mod_dist

def loss_baoguo(sum_baoguo):
sum_ship = {"2万吨级滚装船":38000*0.7}
mod_dist = {}
a = []
b = []
c = []
for baoguo_k, baoguo_v in sum_baoguo.items():
flag = False
x = 0
for ship_k, ship_v in sum_ship.items():
# 如果包裹量小于等于第j艘船的量，装入第j艘船中
x += 1
# if sum_baoguo[i] <= sum_ship[j]:
if baoguo_v <= ship_v:
# sum_ship[j] -= sum_baoguo[i] #第j量船的载容量减少
flag = True
sum_ship[ship_k] -= baoguo_v
print(baoguo_k+"装到"+ship_k)
if baoguo_k.startswith("二"):
a.append(baoguo_v)
elif baoguo_k.startswith("三"):
b.append(baoguo_v)
else:
c.append(baoguo_v)
break # 开始装第二个包裹
```

```
else:
pass # 否则开启第二只船
if len(sum_ship.items()) == x and flag == False:
print(baoguo_k + "没有船可装" + str(baoguo_v))
# 收集起来
mod_dist[baoguo_k] = baoguo_v
print(sum(a),sum(b),sum(c))
return sum_ship, mod_dist


if __name__ == '__main__':

'''每个装备对应的面积'''
x_len = np.array([11,10,8,7.5,6.7,6,6.7,7.5,13,9,5.2,5.6,7,8.5])
x_kunan = np.array([3.5,3.2,3.5,3.3,3.3,3,3.2,3.3,12,2.5,3.5,3.5,3.5,2.4])
x_weight = np.array([1.24,1.24,1.24,1.24,1.24,1.24,1,1,1.14,1.1,1.1,1.05,1.24,1.05])
x_S = x_kunan*x_len*x_weight
print("每个装备对应的面积\n"+str(x_S)+'\n')

'''船只融载面积'''
Y = np.array([4000,750,1500,750,750,600,600,1200,300,200,200,200,70,70])
Y = Y*0.75
num_Y = np.array([5,3,9,4,11,10,1,1,31,12,10,200,36,8])
print("船只融载面积\n"+str(shell_sort(Y))+'\n')
# print("所有船总面积sum(Y) = " + str(sum(Y*num_Y)))

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

#一个旅对应的包裹
one_sum_baoguo =
    zhuang_baoguo([130,100,20,5,8,5,50],[47.74,34.72,27.4164,22.32,21.44,24.75,21.42],54,43.51851852,'一')
two_sum_baoguo =
    zhuang_baoguo([220,20,5,8,5,50],[34.72,27.4164,22.32,21.44,24.75,21.42],54,43.51851852,'二')
three_sum_baoguo =
    zhuang_baoguo([120,10,5,8,4,50],[34.72,27.4164,22.32,21.44,24.75,21.42],54,41.6666667,'三')
four_sum_baoguo = zhuang_baoguo([120,50],[27.4164,21.42],35,37.14285714,'四')
five_sum_baoguo = zhuang_baoguo([20,25,30],[21.44,24.75,30.38],35,28.5714285,'五')
six_sum_baoguo_feiji =
    zhuang_baoguo([40,30],[177.84,21.42],35,28.57142857,'六')#带飞机的旅，先装！！！！！！
seven_sum_baoguo = zhuang_baoguo([30,15],[34.72,24.75],35,27.57142857,'七')
eight_sum_baoguo = zhuang_baoguo([150],[20.02],35,35.71428571,'八')
night_sum_baoguo = zhuang_baoguo([100],[20.58],35,32.85714286,'九')
ten_sum_baoguo = zhuang_baoguo([100],[21.42],35,28.57142857,'十')
elevn_sum_baoguo =
    zhuang_baoguo([100,105,18,12,50],[39.68,30.69,27.4164,22.32,21.42],35,71.42857143,'eleven')
twelve_sum_baoguo = zhuang_baoguo([],[],35,42.85714286,'twelve')
# sum_baoguo = [2000,140,1611,41,780,145,17,2510,1,148] #测试数据1
# sum_baoguo = one_sum_baoguo #测试装包数据
# print("第一旅的包裹\n"+str(sum_baoguo)+'\n')

# '''这里写几个旅的包裹相加'''
sum_baoguo =
    Merge(Merge(Merge(Merge(Merge(Merge(Merge(Merge(Merge(Merge(one_sum_baoguo,two_sum_baoguo),three_sum_baoguo
    +.........
# print(sum_baoguo)
first_baoguo = six_sum_baoguo_feiji
# # 总船只
sum_ship = sum_ship_list(Y,num_Y)
dict_ship = ship_list2dict(sum_ship)#船变字典
# print(dict_ship)


# '''包裹与船只分别从大到小排序'''
sum_baoguo = dict(kv_sort(sum_baoguo))
first_baoguo = dict(kv_sort(first_baoguo))
sum_ship = dict(kv_sort(dict_ship))

# zhuangzai_Map = np.array([[0.0]*12]*len(sum_ship))
zhuangzai_Map = [{},{},{},{},{},{},{},{},{},{},{},{}]
sum_zhuangzai_ship,zhuangzai_Map2,mod_dist =
    zhuangzai_ship(first_baoguo,sum_ship,zhuangzai_Map)
print('总船只装载飞机后的船只装载量\n'+str(sum_zhuangzai_ship)+'\n')
# print("装载图\n"+str(zhuangzai_Map2)+'\n')
# print("总船只\n"+str(sum_ship)+'\n')
# print("总包裹\n"+str(sum_baoguo)+'\n')
print("剩余的包裹\n"+str(mod_dist)+'\n')

sum_zhuangzai_ship,zhuangzai_Map3,mod_dist =
    zhuangzai_ship(sum_baoguo,sum_zhuangzai_ship,zhuangzai_Map2)
print('总船只装载后的装载量\n'+str(sum_zhuangzai_ship)+'\n')
print("装载图\n"+str(zhuangzai_Map3)+'\n')
print("剩余的包裹\n"+str(mod_dist)+'\n')
sum_ship,mod_dist = loss_baoguo(mod_dist)
print("总船只\n"+str(sum_ship)+'\n')

#输出结果并存储
zhuangzai_Map_df = pd.DataFrame(zhuangzai_Map3)
zhuangzai_Map_df.to_csv('zhuangzai_Map.csv')
```

A.2 问题二沙石遗传与时间轴仿真模型–python 源代码

```
import math
import numpy as np
import pandas as pd
import random
import matplotlib.pyplot as plt
from matplotlib import font_manager

my_font = font_manager.FontProperties(fname="C:\Windows\Fonts\msyh.ttc")#微软雅黑字体位置


def append(list,x):
'''numpy数组中插入一个元素在末尾'''
return np.append(list,x)


def max_iii(a):
return a.index(max(a))


def Merge(dict1, dict2):
'''字典合并'''
res = {**dict1, **dict2}
return res


def kv_deng(dic1):
modddd = {}
for k, v in dic1.items():
modddd[k] = v
return modddd


def shell_sort(lists):
'''
按从大到小排序，时间复杂度O(n)
:param lists:传进来的数组，比如船只或者包裹
:return: 返回从大到小的包裹
'''
# 希尔排序
lists = sorted(lists,reverse=True)
return lists


def kv_sort(kv):
'''
键值对由大到小排序
eg:
d = {'d1':2, 'd2':4, 'd4':1,'d3':3,}
```

```
res = sorted(d.items(),key=lambda d:d[1],reverse=True)
[('d2', 4), ('d3', 3), ('d1', 2), ('d4', 1)]
:param kv:键值对
:return:元组
'''
res = sorted(kv.items(), key=lambda d: d[1], reverse=True)
return res


def min_index(lists):
'''
返回最穷连的下标
:param lists: 各连的装备面积数目
:return: 最穷的那个连的下标
'''
index_min = 0
for i in range(1, len(lists)):
if lists[i] < lists[index_min]:
index_min = i
return index_min


def jiaopei_1(live1,live2):
'''
遗传交配的第一种方法
:param live1: 父代
:param live2: 父代
:return: 子代
'''
son = [0,0,0,0,0]
for i in range(5):
son[i] = random.randint(min(live1[i],live2[i]),max(live1[i],live2[i]))
return son

def jiaopei_2(live1,live2):
'''
遗传交配的第二种方法
:param live1: 父代
:param live2: 父代
:return: 子代两个
'''
son1 = [0,0,0,0,0]
son2 = [0,0,0,0,0]
x = random.randint(1,4)
for i in range(x):
son1[i] = live1[i]
son2[i] = live2[i]
```

```
for i in range(5-x):
son1[4-i] = live2[4-i]
son2[4-i] = live1[4-i]

return son1,son2

def bianyi(geti , gailv):
'''
遗传操作中变异的操作
:param geti 个体
:param gailv 变异概率
:return: 变异后的个体
'''
is_bianyi = False
bianyihou = geti
if gailv > random.random():
# print("变异")
x = random.randint(0, 4)
y = random.randint(0, 4)
if geti[2]==1 and y==2:
geti[2]+=1
is_bianyi=True
elif geti[y] == 0:
geti[y]+=1
is_bianyi=True
else:
if random.random()>0.5:
geti[y]+=1
is_bianyi = True
else:
geti[y]-=1
is_bianyi = True

if geti[2]==1 and x==2:
geti[2]+=1
is_bianyi=True
elif geti[x] == 0:
geti[x]+=1
is_bianyi=True
else:
if random.random()>0.5:
geti[x]+=1
is_bianyi = True
else:
geti[x]-=1
is_bianyi = True
```

```
return bianyihou,is_bianyi


def ship_list2dict(sum_ship):
'''
船变字典
:param sum_ship:船的列表
:return: 船的字典
'''
# 船只变成列表
# 列表转字典
dict_ship = {}
for i in range(len(sum_ship)):
if i < 5:
dict_ship["Y1_" + str(i)] = sum_ship[i]
elif i < 8:
dict_ship["Y2_" + str(i - 5)] = sum_ship[i]
elif i < 17:
dict_ship["Y3_" + str(i - 8)] = sum_ship[i]
elif i < 21:
dict_ship["Y4_" + str(i - 17)] = sum_ship[i]
elif i < 32:
dict_ship["Y5_" + str(i - 21)] = sum_ship[i]
elif i < 42:
dict_ship["Y6_" + str(i - 32)] = sum_ship[i]
elif i < 43:
dict_ship["Y7_" + str(i - 42)] = sum_ship[i]
elif i < 44:
dict_ship["Y8_" + str(i - 43)] = sum_ship[i]
elif i < 75:
dict_ship["Y9_" + str(i - 44)] = sum_ship[i]
elif i < 87:
dict_ship["Y10_" + str(i - 75)] = sum_ship[i]
elif i < 97:
dict_ship["Y11_" + str(i - 87)] = sum_ship[i]
elif i < 297:
dict_ship["Y12_" + str(i - 97)] = sum_ship[i]
elif i < 333:
dict_ship["Y13_" + str(i - 297)] = sum_ship[i]
elif i < 341:
dict_ship["Y14_" + str(i - 333)] = sum_ship[i]
return dict_ship


def ship_list2dict_z(sum_ship, z1, z2, z3, z4, z5):
'''
船变字典，带有一个参数的
```

```
:param sum_ship:船的列表
:return: 船的字典
'''
# 船只变成列表
# 列表转字典
dict_ship = {}
for i in range(len(sum_ship)):
if i < z1:
dict_ship["Z1_"+str(i)] = sum_ship[i]
elif i < z1+z2:
dict_ship["Z2_"+str(i-z1)] = sum_ship[i]
elif i < z1+z2+z3:
dict_ship["Z3_"+str(i-z1-z2)] = sum_ship[i]
elif i < z1+z2+z3+z4:
dict_ship["Z4_"+str(i-z1-z2-z3)] = sum_ship[i]
elif i < z1+z2+z3+z4+z5:
dict_ship["Z5_"+str(i-z1-z2-z3-z4)] = sum_ship[i]
return dict_ship


def sum_ship_list(ship_S, ship_num):
'''
计算所有的船只数目/装备数目
:param ship_S: 船只面积/装备面积
:param ship_num: 各类船只数目/装备数目
:return: 所有用有船只面积/装备面积
'''
sum_ship = []
if(len(ship_num)==len(ship_S)):
for i in range(len(ship_num)):
for j in range(ship_num[i]):
sum_ship.append(ship_S[i])
else:
print('船数与船面积的两个列表不相等，可能是打错了\n')
return sum_ship


def zhuang_baoguo(zhuangbei_num_list, zhuangbei_S_list, lian_num, lian_mianji, type, lv_num):
'''
给出装备数目，装备面积，要求*均分*给每个连
均分打包思想：谁穷我给谁装备：
装备我从小到大排序；
for 装备的数目：
装备先给连j;
判断哪个连最穷
给穷的连装备
```

```
:param zhuangbei_num_list: 各个装备的数目
:param zhuangbei_S_list: 不同装备的面积
:param lian_num: 连的数量
:param lian_mianji: 连中人的面积
:param type: 旅的类型, 键
:param lv_num: 旅的数目
:return: 装备打包好了后, 下一步就是上船
'''

#初始化连的数目
lian = [0.0]*lian_num

#c初始化装备数目
zhuangbei = sum_ship_list(zhuangbei_S_list, zhuangbei_num_list)
#装备排序
zhuangbei = shell_sort(zhuangbei)
# print("总的装备的排序\n"+str(zhuangbei)+'\n')

for i in range(len(zhuangbei)):
# 最穷的下标
index_min = min_index(lian)
lian[index_min] += zhuangbei[i]

for i in range(len(lian)):
lian[i] += lian_mianji
# print(type+str(lian))

lian2 = []
for i in range(lv_num):
lian2 += lian

#列表转字典
dict_lian = {}
for i in range(len(lian2)):
dict_lian[type+str(i)] = lian2[i]
return dict_lian


def zhuangzai_ship(sum_baoguo, sum_ship):
'''
第一问核心思想
装船的操作, 按石头装满装沙子的思想
:param sum_baoguo: 总的包裹
:param sum_ship: 总的船只
:return: 装载后的船只,装载图
'''
mod_dist = {}
```

```
for baoguo_k,baoguo_v in sum_baoguo.items():
flag = False
x=0
for ship_k,ship_v in sum_ship.items():
#如果包裹量小于等于第j艘船的量，装入第j艘船中
x+=1
# if sum_baoguo[i] <= sum_ship[j]:
if baoguo_v <= ship_v:
# sum_ship[j] -= sum_baoguo[i] #第j量船的载容量减少
flag=True
sum_ship[ship_k] -= baoguo_v
# print(baoguo_k+"装到"+ship_k)
break  #开始装第二个包裹
else:
pass   #否则开启第二只船
if len(sum_ship.items()) == x and flag == False:
# print(baoguo_k+"没有船可装"+str(baoguo_v))
#收集起来
mod_dist[baoguo_k] = baoguo_v
return sum_ship,mod_dist


def zhuangzai_zhishengji_ship(sum_baoguo, sum_ship):
'''
装六型旅的操作，按石头装满装沙子的思想
:param sum_baoguo: 总的包裹
:param sum_ship: 总的船只
:return: 装载后的船只,装载图
'''
mod_dist = {}
for baoguo_k,baoguo_v in sum_baoguo.items():
flag = False
x=0
for ship_k,ship_v in sum_ship.items():
#如果包裹量小于等于第j艘船的量，装入第j艘船中
x += 1
# if sum_baoguo[i] <= sum_ship[j]:
if baoguo_v <= ship_v:
if ship_k.startswith("Y1") or ship_k.startswith("Y5"):
# sum_ship[j] -= sum_baoguo[i] #第j量船的载容量减少
flag=True
sum_ship[ship_k] -= baoguo_v
# print(baoguo_k+"装到"+ship_k)
break  #开始装第二个包裹
else:
pass   #否则开启第二只船
if len(sum_ship.items())==x and flag==False:
```

```
# print(baoguo_k+"没有船可装"+str(baoguo_v))
#收集起来
mod_dist[baoguo_k] = baoguo_v
return sum_ship,mod_dist


def zhuangzai_xuanze_ship(sum_baoguo, sum_Z_ship):
'''
剩余包裹装民船
:param sum_baoguo: 总的包裹
:param sum_ship: 总的船只
:return: 装载后的船只,装载图
'''
# 一开始是不适应的
flag = True
for baoguo_k,baoguo_v in sum_baoguo.items():
for ship_k,ship_v in sum_Z_ship.items():
# 白包裹
if baoguo_k.startswith("一") or baoguo_k.startswith("二") or baoguo_k.startswith("三") or
    baoguo_k.startswith("七") or baoguo_k.startswith("eleven"):
if ship_k.startswith("Z3") or ship_k.startswith("Z4") or ship_k.startswith("Z5"):
#如果白包裹碰到z345, 换下一艘船
pass
else:
if baoguo_v <= ship_v:
# 装上船, 换下一个包裹
sum_Z_ship[ship_k] -= baoguo_v
sum_baoguo[baoguo_k] = 0
break
#黄包裹
elif baoguo_k.startswith("四") or baoguo_k.startswith("五") or baoguo_k.startswith("六") or
    baoguo_k.startswith("八") or baoguo_k.startswith("九") or baoguo_k.startswith("十"):
if ship_k.startswith("Z4") or ship_k.startswith("Z5"):
pass
else:
if baoguo_v <= ship_v:
sum_Z_ship[ship_k] -= baoguo_v
sum_baoguo[baoguo_k] = 0
break
#人
else:
if baoguo_v <= ship_v:

sum_Z_ship[ship_k] -= baoguo_v
sum_baoguo[baoguo_k] = 0
break
```

```
for baoguo_k, baoguo_v in sum_baoguo.items():
if sum_baoguo[baoguo_k] != 0:
flag=False

return sum_Z_ship,flag


def jizhuangxaing_zhishengji_ship(sum_baoguo, sum_ship):
'''
民船Z3装在直升机
:param sum_baoguo: 总的包裹
:param sum_ship: 总的船只
:return: 装载后的船只,装载图
'''
mod_dist = {}
for baoguo_k,baoguo_v in sum_baoguo.items():
flag = False
x=0
for ship_k,ship_v in sum_ship.items():
#如果包裹量小于等于第j艘船的量, 装入第j艘船中
x += 1
# if sum_baoguo[i] <= sum_ship[j]:
if baoguo_v <= ship_v:
if ship_k.startswith("Y1") or ship_k.startswith("Y5"):
# sum_ship[j] -= sum_baoguo[i] #第j量船的载容量减少
flag=True
sum_ship[ship_k] -= baoguo_v
# print(baoguo_k+"装到"+ship_k)
break  #开始装第二个包裹
else:
pass   #否则开启第二只船
if len(sum_ship.items())==x and flag==False:
# print(baoguo_k+"没有船可装"+str(baoguo_v))
#收集起来
mod_dist[baoguo_k] = baoguo_v
return sum_ship,mod_dist


def zhuangmingchuan(mod_dist, zx):
'''
剩余的包裹装一种类型船的数量
:param mod_dist: 剩余的包裹
:param zx: 船的容积
:return: 船的数目
'''
zx_num = 1
zxy=zx
```

```
for baoguo_k, baoguo_v in mod_dist.items():
if zxy>=0 and zxy>=baoguo_v:
zxy-=baoguo_v
else:
zxy=zx
zx_num+=1
return zx_num


def surt_time(live):
'''
计算适应度的函数，适应度为时间
:param live: 存活个体[z1,z2...,z5]
:return: 适应时间
'''
num_matou = [10, 5, 1, 3, 2, 10, 1, 4, 3, 6, 5, 3, 5, 3, 3, 7]
w_mat = [30000, 20000, 1000, 30000, 1000, 1000, 1000, 30000, 1000, 30000, 1000, 1000, 500,
    1000, 30000, 1000]
t_matou = [0] * 71
W_matou = sum_ship_list(w_mat,num_matou)
numZ = np.array(live)
# 第一行向量，船的数目
num_ship = np.hstack((num_Y, numZ))
# print("输入船的数量"+str(num_ship))
# print(num_ship)
# 第二行向量，船的吨位
w_ship = [18500, 4170, 4800, 4170, 4800, 2000, 1650, 1800, 850, 850, 800, 600, 128, 85, 20000,
    30000, 30000, 10000,
5000]
# 第三行向量，船的时间
t_ship = [5, 5, 5, 5, 5, 4, 4, 4, 4, 3, 3, 3, 1, 1, 10, 15, 18, 5, 4]

'''适应度的计算：时间t0'''
t0 = -1
while True:
t0+=1

if len(num_ship) != len(w_ship) or len(num_ship) != len(t_ship) or len(W_matou) !=
    len(t_matou):
print("输入错误，请查看船或码头个数")
break

#检索71次
for i in range(len(t_matou)):
#如果有码头是空闲的
if t_matou[i]==t0:
#检索对应的船
```

```
manzu_w_ship = []
manzu_j_ship = []
for j in range(len(w_ship)):
#如果满足条件
if W_matou[i]>w_ship[j]:
manzu_w_ship.append(w_ship[j])#记录重量和对应重量的位置
manzu_j_ship.append(j)
#取出最大满足吨位的船的下标
j_max = max_iii(manzu_w_ship)
if num_ship[j_max]>0:
#船的数量减一
num_ship[j_max] -= 1
#码头对应时间加上tk
# print(t_matou)
t_matou[i] += t_ship[j_max]
if num_ship[j_max]==0:
w_ship[j_max]=0
# print(t_matou)
#跳出循环
# print(num_ship)

if t0>max(t_matou) and sum(num_ship)==0:
break
# print(t0)
return t0


def is_surt_manzai(Z_X,mod_dist):
'''
判断某一个个体是否满足能装载
:return: 是否能装满
'''
Zx_w = [19000 * 0.7, 26000 * 0.7, 38000 * 0.7, 6000 * 0.7, 2000 * 0.7]
sum_Z_ship = sum_ship_list(Zx_w, Z_X)
dict_ship_Z = ship_list2dict_z(sum_Z_ship, Z_X[0], Z_X[1], Z_X[2], Z_X[3], Z_X[4]) # 船变字典
dict_ship_Z['Z3_' + str(Z_X[2])] = 2494.836677158
sum_Z_ship_dict = dict(kv_sort(dict_ship_Z)) # 排序
moddd = kv_deng(mod_dist)
shiying_ship, is_sure = zhuangzai_xuanze_ship(moddd, sum_Z_ship_dict)
# print("是否适应\n" + str(is_sure) + '\n')

return is_sure


if __name__ == '__main__':

'''每个装备对应的面积'''
```

```
x_len = np.array([11,10,8,7.5,6.7,6,6.7,7.5,13,9,5.2,5.6,7,8.5])
x_kunan = np.array([3.5,3.2,3.5,3.3,3.3,3,3.2,3.3,12,2.5,3.5,3.5,3.5,2.4])
x_weight = np.array([1.24,1.24,1.24,1.24,1.24,1.24,1,1,1.14,1.1,1.1,1.05,1.24,1.05])
x_S = x_kunan*x_len*x_weight
# print("每个装备对应的面积\n"+str(x_S)+'\n')

'''船只融载面积'''
Y = np.array([4000,750,1500,750,750,600,600,1200,300,200,200,200,70,70])
Y = Y*0.75
num_Y = np.array([5,3,9,4,11,10,1,1,31,12,10,200,36,8])
# print("船只融载面积\n"+str(shell_sort(Y))+'\n')
# print("所有船总面积sum(Y) = " + str(sum(Y*num_Y)))

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''


#一个旅对应的包裹
one_sum_baoguo =
    zhuang_baoguo([130,100,20,5,8,5,50],[47.74,34.72,27.4164,22.32,21.44,24.75,21.42],54,43.51851852,'一',12)
two_sum_baoguo =
    zhuang_baoguo([220,20,5,8,5,50],[34.72,27.4164,22.32,21.44,24.75,21.42],54,43.51851852,'二',3)
three_sum_baoguo =
    zhuang_baoguo([120,10,5,8,4,50],[34.72,27.4164,22.32,21.44,24.75,21.42],54,41.6666667,'三',3)
four_sum_baoguo = zhuang_baoguo([120,50],[27.4164,21.42],35,37.14285714,'四',3)
five_sum_baoguo = zhuang_baoguo([20,25,30],[21.44,24.75,30.38],35,28.5714285,'五',3)
six_sum_baoguo_feiji =
    zhuang_baoguo([40,30],[177.84,21.42],35,28.57142857,'六',5)#带飞机的旅，先装！！！！！！！
seven_sum_baoguo = zhuang_baoguo([30,15],[34.72,24.75],35,27.57142857,'七',5)
eight_sum_baoguo = zhuang_baoguo([150],[20.02],35,35.71428571,'八',3)
night_sum_baoguo = zhuang_baoguo([100],[20.58],35,32.85714286,'九',3)
ten_sum_baoguo = zhuang_baoguo([100],[21.42],35,28.57142857,'十',3)
elevn_sum_baoguo =
    zhuang_baoguo([100,105,18,12,50],[39.68,30.69,27.4164,22.32,21.42],35,71.42857143,'eleven',6)
twelve_sum_baoguo = zhuang_baoguo([],[],35,42.85714286,'twelve',10)
# sum_baoguo = [2000,140,1611,41,780,145,17,2510,1,148] #测试数据1
# sum_baoguo = one_sum_baoguo #测试装包数据
# print("第一旅的包裹\n"+str(sum_baoguo)+'\n')

# '''这里写几个旅的包裹相加'''
sum_baoguo =
    Merge(Merge(Merge(Merge(Merge(Merge(Merge(Merge(Merge(Merge(one_sum_baoguo,two_sum_baoguo),three_sum_baoguo
    +.........
# print(sum_baoguo)
first_baoguo = six_sum_baoguo_feiji
# # 总船只
sum_ship = sum_ship_list(Y,num_Y)
dict_ship = ship_list2dict(sum_ship)#船变字典
print(dict_ship)
```

```
'''包裹与船只分别从大到小排序'''
sum_baoguo = dict(kv_sort(sum_baoguo))
first_baoguo = dict(kv_sort(first_baoguo))
sum_ship = dict(kv_sort(dict_ship))
# print("总船只\n"+str(sum_ship)+'\n')

# '''装直升机的操作'''
sum_zhuangzai_ship, mod_dist = zhuangzai_zhishengji_ship(first_baoguo, sum_ship)
print('总船只装载飞机后的船只装载量\n'+str(sum_zhuangzai_ship)+'\n')
# print("总船只\n"+str(sum_ship)+'\n')
print("总包裹\n"+str(sum_baoguo)+'\n')
# '''106个直升机的包裹
#  总面积24105.163322842
#   开一个Z3装得下13300*2
#   剩下2494.836677158
#   '''
# print("剩余的包裹\n"+str(mod_dist)+'\n')
# z3 = 1
#
# '''装人的操作'''
sum_zhuangzai_ship,mod_dist = zhuangzai_ship(sum_baoguo,sum_zhuangzai_ship)
print('总船只装载后的装载量\n'+str(sum_zhuangzai_ship)+'\n')
print("装民船的面积的面积\n"+str(sum(mod_dist.values())))
mod_dist = dict(kv_sort(mod_dist))
# print("剩余的包裹\n"+str(mod_dist)+'\n')
#
#
# z1 = zhuangmingchuan(mod_dist,19000*0.7)
# z2 = zhuangmingchuan(mod_dist,26000*0.7)
# z3 += zhuangmingchuan(mod_dist,38000*0.7)
#Z的最大上限
# print(z1,z2,z3) #[25, 18, 14, 4, 11]

##########################################################################################################################
#存活个体数
w = 50
live = []
Zx_w = [19000 * 0.7, 26000 * 0.7, 38000 * 0.7, 6000 * 0.7, 2000 * 0.7]
Zx_num = [25, 18, 6, 4, 11]

#初始化适应个体
while len(live) != w:
# 总民用船的数目
random_W =
    np.array([random.random(),random.random(),random.random(),random.random(),random.random()])
```

```
# 初始个体数目
num_Zx = np.trunc(random_W*Zx_num).astype(np.int8)#取整

# # 总民用船船只
sum_Z_ship = sum_ship_list(Zx_w, num_Zx)
# print("总民用船船只\n" + str(sum_Z_ship) + '\n')
dict_ship_Z = ship_list2dict_z(sum_Z_ship,num_Zx[0],num_Zx[1],num_Zx[2],num_Zx[3],num_Zx[4])#
    船变字典
dict_ship_Z['Z3_'+str(num_Zx[2])] = 2494.836677158
sum_Z_ship_dict = dict(kv_sort(dict_ship_Z))# 排序
# print("民用船船只字典和个数\n" + str(sum_Z_ship_dict) + '\n')
# print("剩余的包裹\n" + str(mod_dist) + '\n')
# print(mod_dist)
moddd = kv_deng(mod_dist)

shiying_ship, is_sure = zhuangzai_xuanze_ship(moddd, sum_Z_ship_dict)
# print("是否适应\n" + str(is_sure) + '\n')
moddd = kv_deng(mod_dist)

if is_sure:
num_Zx[2] += 1
live.append(num_Zx.tolist())
# print("装完物体后民用船适应度\n" + str(shiying_ship) + '\n')

# break
# print("适应个体: "+str(live)+'\n')

###############################################################################################################
# 计算适应度
# t0 = surt_time(live[0])
# print("计算适应度: : "+str(t0))
# 测试交配
# print(live[0],live[1])
# print(jiaopei_1(live[0],live[1]))
# print(jiaopei_2(live[0],live[1]))
#
# #测试变异
# print(live[0])
# print(bianyi(live[0],0.8))
#
# 测试是否满足适应
# print(is_surt_manzai(live[4],mod_dist))
#
###############################################################################################################
# 遗传的实现
min_t = []
duiying_num = []
```

```
for i in range(100):
the_all_son = live
bianyilv = 0.2
for i in range(w):
#变异
bianyi_son,is_bianyi = bianyi(live[i],bianyilv)
if is_bianyi:
the_all_son.append(bianyi_son)
# print(len(the_all_son))

#交配
random.shuffle(live)
for i in range(0,w-1,2):
son = jiaopei_1(live[i],live[i+1])
the_all_son.append(son)
son1,son2 = jiaopei_2(live[i],live[i+1])
the_all_son.append(son1)
the_all_son.append(son2)
# the_all_son.append()
# print(len(the_all_son))


t0_all = []
surt_son = []
for i in range(len(the_all_son)):
if is_surt_manzai(the_all_son[i],mod_dist):
t0_all.append(surt_time(the_all_son[i]))
surt_son.append(the_all_son[i])

chongfu = []
for i in range(len(surt_son)):
for j in range(i,len(surt_son)):
if i!=j and surt_son[i]==surt_son[j]:
# print(j)
chongfu.append(i)
# print(t0_all)
# print(surt_son)

for i in range(len(chongfu),0,-1):
surt_son.pop(i)
t0_all.pop(i)

# print("变异后个体: "+str(len(the_all_son)))
num_ship = []
index = []
for i in range(len(surt_son)):
# if t0_all[i] == 21:
```

```
#    print(surt_son[i])
num_ship.append(sum(surt_son[i]))
index.append(i)
# print(t0_all)
# print(num_ship)

x = np.vstack((np.array(t0_all),np.array(num_ship),np.array(index)))
# 第一行排序
x = x.T[np.lexsort(x[::-1,:])].T
min_t.append(x[0][0])
duiying_num.append(x[1][0])

youxiu_geti = []
for i in range(w):
youxiu_geti.append(surt_son[x[2][i]])
print(youxiu_geti)
print("时间: "+str(x[0]))
print("船数: "+str(x[1]))
live = youxiu_geti

plt.plot(duiying_num, label='种群中对应船只数量', )

plt.plot(min_t, label='种群中最小装载时间', color="r")
plt.xlabel("迭代次数", fontproperties=my_font)
plt.title("适应度变化趋势", fontproperties=my_font)
plt.legend(prop=my_font, loc=0)
plt.grid(alpha=0.3, linestyle="--") # alpha为透明度 0-1

plt.show()
##############################################################################################################
######遍历Zx_num = [25, 18, 14, 4, 11]
# file = 'test.txt'
# with open(file, 'a+') as f:
#    for i_a in range(25):
#        for i_b in range(18):
#            for i_c in range(6):
#                for i_d in range(4):
#                    for i_e in range(11):
#                        # print(i_a,i_b,i_c,i_d,i_e)
#                        if is_surt_manzai([i_a,i_b,i_c,i_d,i_e], mod_dist):
#                            tqqq = surt_time([i_a,i_b,i_c,i_d,i_e])
#
    f.write(str(tqqq)+","+str([i_a,i_b,i_c,i_d,i_e])+","+str(sum([i_a,i_b,i_c,i_d,i_e]))+"\n")
#                            print("*"*20+str(tqqq))
##############################################################################################################
```

A.3 问题三改进时间轴仿真模型–python 源代码

```
import math
import numpy as np
import pandas as pd
import random
import matplotlib.pyplot as plt
from matplotlib import font_manager

my_font = font_manager.FontProperties(fname="C:\Windows\Fonts\msyh.ttc")#微软雅黑字体位置


def append(list,x):
'''numpy数组中插入一个元素在末尾'''
return np.append(list,x)


def max_iii(a):
return a.index(max(a))


def Merge(dict1, dict2):
'''字典合并'''
res = {**dict1, **dict2}
return res


def kv_deng(dic1):
modddd = {}
for k, v in dic1.items():
modddd[k] = v
return modddd


def shell_sort(lists):
'''
按从大到小排序, 时间复杂度O(n)
:param lists:传进来的数组, 比如船只或者包裹
:return: 返回从大到小的包裹
'''
# 希尔排序
lists = sorted(lists,reverse=True)
return lists


def kv_sort(kv):
'''
```

```
键值对由大到小排序
eg:
d = {'d1':2, 'd2':4, 'd4':1,'d3':3,}
res = sorted(d.items(),key=lambda d:d[1],reverse=True)
[('d2', 4), ('d3', 3), ('d1', 2), ('d4', 1)]
:param kv:键值对
:return:元组
'''
res = sorted(kv.items(), key=lambda d: d[1], reverse=True)
return res


def min_index(lists):
'''
返回最穷连的下标
:param lists: 各连的装备面积数目
:return: 最穷的那个连的下标
'''
index_min = 0
for i in range(1, len(lists)):
if lists[i] < lists[index_min]:
index_min = i
return index_min


def jiaopei_1(live1,live2):
'''
遗传交配的第一种方法
:param live1: 父代
:param live2: 父代
:return: 子代
'''
son = [0,0,0,0,0]
for i in range(5):
son[i] = random.randint(min(live1[i],live2[i]),max(live1[i],live2[i]))
return son

def jiaopei_2(live1,live2):
'''
遗传交配的第二种方法
:param live1: 父代
:param live2: 父代
:return: 子代两个
'''
son1 = [0,0,0,0,0]
son2 = [0,0,0,0,0]
x = random.randint(1,4)
```

```python
for i in range(x):
son1[i] = live1[i]
son2[i] = live2[i]
for i in range(5-x):
son1[4-i] = live2[4-i]
son2[4-i] = live1[4-i]

return son1,son2

def bianyi(geti , gailv):
'''
遗传操作中变异的操作
:param geti 个体
:param gailv 变异概率
:return: 变异后的个体
'''
is_bianyi = False
bianyihou = geti
if gailv > random.random():
# print("变异")
x = random.randint(0, 4)
y = random.randint(0, 4)
if geti[2]==1 and y==2:
geti[2]+=1
is_bianyi=True
elif geti[y] == 0:
geti[y]+=1
is_bianyi=True
else:
if random.random()>0.5:
geti[y]+=1
is_bianyi = True
else:
geti[y]-=1
is_bianyi = True

if geti[2]==1 and x==2:
geti[2]+=1
is_bianyi=True
elif geti[x] == 0:
geti[x]+=1
is_bianyi=True
else:
if random.random()>0.5:
geti[x]+=1
is_bianyi = True
else:
```

```
geti[x]-=1
is_bianyi = True

return bianyihou,is_bianyi


def ship_list2dict(sum_ship):
'''
船变字典
:param sum_ship:船的列表
:return: 船的字典
'''
# 船只变成列表
# 列表转字典
dict_ship = {}
for i in range(len(sum_ship)):
if i < 5:
dict_ship["Y1_" + str(i)] = sum_ship[i]
elif i < 8:
dict_ship["Y2_" + str(i - 5)] = sum_ship[i]
elif i < 17:
dict_ship["Y3_" + str(i - 8)] = sum_ship[i]
elif i < 21:
dict_ship["Y4_" + str(i - 17)] = sum_ship[i]
elif i < 32:
dict_ship["Y5_" + str(i - 21)] = sum_ship[i]
elif i < 42:
dict_ship["Y6_" + str(i - 32)] = sum_ship[i]
elif i < 43:
dict_ship["Y7_" + str(i - 42)] = sum_ship[i]
elif i < 44:
dict_ship["Y8_" + str(i - 43)] = sum_ship[i]
elif i < 75:
dict_ship["Y9_" + str(i - 44)] = sum_ship[i]
elif i < 87:
dict_ship["Y10_" + str(i - 75)] = sum_ship[i]
elif i < 97:
dict_ship["Y11_" + str(i - 87)] = sum_ship[i]
elif i < 297:
dict_ship["Y12_" + str(i - 97)] = sum_ship[i]
elif i < 333:
dict_ship["Y13_" + str(i - 297)] = sum_ship[i]
elif i < 341:
dict_ship["Y14_" + str(i - 333)] = sum_ship[i]
return dict_ship
```

```
def ship_list2dict_z(sum_ship, z1, z2, z3, z4, z5):
'''
船变字典，带有一个参数的
:param sum_ship:船的列表
:return: 船的字典
'''
# 船只变成列表
# 列表转字典
dict_ship = {}
for i in range(len(sum_ship)):
if i < z1:
dict_ship["Z1_"+str(i)] = sum_ship[i]
elif i < z1+z2:
dict_ship["Z2_"+str(i-z1)] = sum_ship[i]
elif i < z1+z2+z3:
dict_ship["Z3_"+str(i-z1-z2)] = sum_ship[i]
elif i < z1+z2+z3+z4:
dict_ship["Z4_"+str(i-z1-z2-z3)] = sum_ship[i]
elif i < z1+z2+z3+z4+z5:
dict_ship["Z5_"+str(i-z1-z2-z3-z4)] = sum_ship[i]
return dict_ship


def sum_ship_list(ship_S, ship_num):
'''
计算所有的船只数目/装备数目
:param ship_S: 船只面积/装备面积
:param ship_num: 各类船只数目/装备数目
:return: 所有用有船只面积/装备面积
'''
sum_ship = []
if(len(ship_num)==len(ship_S)):
for i in range(len(ship_num)):
for j in range(ship_num[i]):
sum_ship.append(ship_S[i])
else:
print('船数与船面积的两个列表不相等，可能是打错了\n')
return sum_ship


def zhuang_baoguo(zhuangbei_num_list, zhuangbei_S_list, lian_num, lian_mianji, type, lv_num):
'''
给出装备数目，装备面积，要求*均分*给每个连
均分打包思想：谁穷我给谁装备：
装备我从小到大排序；
for 装备的数目：
装备先给连j；
```

```
判断哪个连最穷
给穷的连装备

:param zhuangbei_num_list: 各个装备的数目
:param zhuangbei_S_list: 不同装备的面积
:param lian_num: 连的数量
:param lian_mianji: 连中人的面积
:param type: 旅的类型，键
:param lv_num: 旅的数目
:return: 装备打包好了后，下一步就是上船
'''

#初始化连的数目
lian = [0.0]*lian_num

#c初始化装备数目
zhuangbei = sum_ship_list(zhuangbei_S_list, zhuangbei_num_list)
#装备排序
zhuangbei = shell_sort(zhuangbei)
# print("总的装备的排序\n"+str(zhuangbei)+'\n')

for i in range(len(zhuangbei)):
# 最穷的下标
index_min = min_index(lian)
lian[index_min] += zhuangbei[i]

for i in range(len(lian)):
lian[i] += lian_mianji
# print(type+str(lian))

lian2 = []
for i in range(lv_num):
lian2 += lian

#列表转字典
dict_lian = {}
for i in range(len(lian2)):
dict_lian[type+str(i)] = lian2[i]
return dict_lian


def zhuangzai_ship(sum_baoguo, sum_ship):
'''
第一问核心思想
装船的操作，按石头装满装沙子的思想
:param sum_baoguo: 总的包裹
:param sum_ship: 总的船只
```

```
:return: 装载后的船只,装载图
'''
mod_dist = {}
for baoguo_k,baoguo_v in sum_baoguo.items():
flag = False
x=0
for ship_k,ship_v in sum_ship.items():
#如果包裹量小于等于第j艘船的量, 装入第j艘船中
x+=1
# if sum_baoguo[i] <= sum_ship[j]:
if baoguo_v <= ship_v:
# sum_ship[j] -= sum_baoguo[i] #第j量船的载容量减少
flag=True
sum_ship[ship_k] -= baoguo_v
# print(baoguo_k+"装到"+ship_k)
break  #开始装第二个包裹
else:
pass   #否则开启第二只船
if len(sum_ship.items()) == x and flag == False:
# print(baoguo_k+"没有船可装"+str(baoguo_v))
#收集起来
mod_dist[baoguo_k] = baoguo_v
return sum_ship,mod_dist


def zhuangzai_zhishengji_ship(sum_baoguo, sum_ship):
'''
装六型旅的操作, 按石头装满装沙子的思想
:param sum_baoguo: 总的包裹
:param sum_ship: 总的船只
:return: 装载后的船只,装载图
'''
mod_dist = {}
for baoguo_k,baoguo_v in sum_baoguo.items():
flag = False
x=0
for ship_k,ship_v in sum_ship.items():
#如果包裹量小于等于第j艘船的量, 装入第j艘船中
x += 1
# if sum_baoguo[i] <= sum_ship[j]:
if baoguo_v <= ship_v:
if ship_k.startswith("Y1") or ship_k.startswith("Y5"):
# sum_ship[j] -= sum_baoguo[i] #第j量船的载容量减少
flag=True
sum_ship[ship_k] -= baoguo_v
# print(baoguo_k+"装到"+ship_k)
break  #开始装第二个包裹
```

```
else:
pass   #否则开启第二只船
if len(sum_ship.items())==x and flag==False:
# print(baoguo_k+"没有船可装"+str(baoguo_v))
#收集起来
mod_dist[baoguo_k] = baoguo_v
return sum_ship,mod_dist


def zhuangzai_xuanze_ship(sum_baoguo, sum_Z_ship):
'''
剩余包裹装民船
:param sum_baoguo: 总的包裹
:param sum_ship: 总的船只
:return: 装载后的船只,装载图
'''
# 一开始是不适应的
flag = True
for baoguo_k,baoguo_v in sum_baoguo.items():
for ship_k,ship_v in sum_Z_ship.items():
# 白包裹
if baoguo_k.startswith("一") or baoguo_k.startswith("二") or baoguo_k.startswith("三") or
    baoguo_k.startswith("七") or baoguo_k.startswith("eleven"):
if ship_k.startswith("Z3") or ship_k.startswith("Z4") or ship_k.startswith("Z5"):
#如果白包裹碰到z345, 换下一艘船
pass
else:
if baoguo_v <= ship_v:
# 装上船, 换下一个包裹
sum_Z_ship[ship_k] -= baoguo_v
sum_baoguo[baoguo_k] = 0
break
#黄包裹
elif baoguo_k.startswith("四") or baoguo_k.startswith("五") or baoguo_k.startswith("六") or
    baoguo_k.startswith("八") or baoguo_k.startswith("九") or baoguo_k.startswith("十"):
if ship_k.startswith("Z4") or ship_k.startswith("Z5"):
pass
else:
if baoguo_v <= ship_v:
sum_Z_ship[ship_k] -= baoguo_v
sum_baoguo[baoguo_k] = 0
break
#人
else:
if baoguo_v <= ship_v:

sum_Z_ship[ship_k] -= baoguo_v
```

```
sum_baoguo[baoguo_k] = 0
break

for baoguo_k, baoguo_v in sum_baoguo.items():
if sum_baoguo[baoguo_k] != 0:
flag=False

return sum_Z_ship,flag



def jizhuangxaing_zhishengji_ship(sum_baoguo, sum_ship):
'''
民船Z3装在直升机
:param sum_baoguo: 总的包裹
:param sum_ship: 总的船只
:return: 装载后的船只,装载图
'''
mod_dist = {}
for baoguo_k,baoguo_v in sum_baoguo.items():
flag = False
x=0
for ship_k,ship_v in sum_ship.items():
#如果包裹量小于等于第j艘船的量, 装入第j艘船中
x += 1
# if sum_baoguo[i] <= sum_ship[j]:
if baoguo_v <= ship_v:
if ship_k.startswith("Y1") or ship_k.startswith("Y5"):
# sum_ship[j] -= sum_baoguo[i] #第j量船的载容量减少
flag=True
sum_ship[ship_k] -= baoguo_v
# print(baoguo_k+"装到"+ship_k)
break  #开始装第二个包裹
else:
pass   #否则开启第二只船
if len(sum_ship.items())==x and flag==False:
# print(baoguo_k+"没有船可装"+str(baoguo_v))
#收集起来
mod_dist[baoguo_k] = baoguo_v
return sum_ship,mod_dist



def zhuangmingchuan(mod_dist, zx):
'''
剩余的包裹装一种类型船的数量
:param mod_dist: 剩余的包裹
:param zx: 船的容积
:return: 船的数目
```

```
'''
zx_num = 1
zxy=zx
for baoguo_k, baoguo_v in mod_dist.items():
if zxy>=0 and zxy>=baoguo_v:
zxy-=baoguo_v
else:
zxy=zx
zx_num+=1
return zx_num


def surt_time(live):
'''
计算适应度的函数，适应度为时间
:param live: 存活个体[z1,z2...,z5]
:return: 适应时间
'''
num_matou = [10, 5, 1, 3, 2, 10, 1, 4, 3, 6, 5, 3, 5, 3, 3, 7]
w_mat = [30000, 20000, 1000, 30000, 1000, 1000, 1000, 30000, 1000, 30000, 1000, 1000, 500,
    1000, 30000, 1000]
t_matou = [0] * 71
W_matou = sum_ship_list(w_mat,num_matou)
numZ = np.array(live)
# 第一行向量，船的数目
num_ship = np.hstack((num_Y, numZ))
# print("输入船的数量"+str(num_ship))
# print(num_ship)
# 第二行向量，船的吨位
w_ship = [18500, 4170, 4800, 4170, 4800, 2000, 1650, 1800, 850, 850, 800, 600, 128, 85, 20000,
    30000, 30000, 10000,
5000]
# 第三行向量，船的时间
t_ship = [5, 5, 5, 5, 5, 4, 4, 4, 4, 3, 3, 3, 1, 1, 10, 15, 18, 5, 4]

'''适应度的计算：时间t0'''
# 销毁时间
rtime = 18
t0 = -1
while True:
t0+=1

if len(num_ship) != len(w_ship) or len(num_ship) != len(t_ship) or len(W_matou) !=
    len(t_matou):
print("输入错误，请查看船或码头个数")
break
```

```
if t0 == rtime:
# print("码头被摧毁！！！")
pass

#检索71次
for i in range(len(t_matou)):
if t0 == rtime:
#摧毁对应码头
t_matou[0]=-1
t_matou[1]=-1
t_matou[10]=-1
t_matou[40]=-1
t_matou[46]=-1
t_matou[47]=-1
pass
#如果有码头是空闲的
if t_matou[i]==t0:
#检索对应的船
manzu_w_ship = []
manzu_j_ship = []
for j in range(len(w_ship)):
#如果满足条件
if W_matou[i]>w_ship[j]:
manzu_w_ship.append(w_ship[j])#记录重量和对应重量的位置
manzu_j_ship.append(j)
#取出最大满足吨位的船的下标
j_max = max_iii(manzu_w_ship)
if num_ship[j_max]>0:
#船的数量减一
num_ship[j_max] -= 1
#码头对应时间加上tk
# print(t_matou)
t_matou[i] += t_ship[j_max]
if i==0 or i==1 or i==40 or i==46 or i== 47 or i== 10:
# print(str(j_max) + "装载到" + str(i) + "码头里")
if t_matou[i]>rtime:
num_ship[j_max] += 1
if num_ship[j_max]==0:
w_ship[j_max]=0
# print(t_matou)
#跳出循环
# print(num_ship)

if t0>max(t_matou) and sum(num_ship)==0:
break
# print(t0)
return t0
```

```
def is_surt_manzai(Z_X,mod_dist):
'''
判断某一个个体是否满足能装载
:return: 是否能装满
'''
Zx_w = [19000 * 0.7, 26000 * 0.7, 38000 * 0.7, 6000 * 0.7, 2000 * 0.7]
sum_Z_ship = sum_ship_list(Zx_w, Z_X)
dict_ship_Z = ship_list2dict_z(sum_Z_ship, Z_X[0], Z_X[1], Z_X[2], Z_X[3], Z_X[4]) # 船变字典
dict_ship_Z['Z3_' + str(Z_X[2])] = 2494.836677158
sum_Z_ship_dict = dict(kv_sort(dict_ship_Z)) # 排序
moddd = kv_deng(mod_dist)
shiying_ship, is_sure = zhuangzai_xuanze_ship(moddd, sum_Z_ship_dict)
# print("是否适应\n" + str(is_sure) + '\n')

return is_sure


if __name__ == '__main__':

'''每个装备对应的面积'''
x_len = np.array([11,10,8,7.5,6.7,6,6.7,7.5,13,9,5.2,5.6,7,8.5])
x_kunan = np.array([3.5,3.2,3.5,3.3,3.3,3,3.2,3.3,12,2.5,3.5,3.5,3.5,2.4])
x_weight = np.array([1.24,1.24,1.24,1.24,1.24,1.24,1,1,1.14,1.1,1.1,1.05,1.24,1.05])
x_S = x_kunan*x_len*x_weight
# print("每个装备对应的面积\n"+str(x_S)+'\n')

'''船只融载面积'''
Y = np.array([4000,750,1500,750,750,600,600,1200,300,200,200,200,70,70])
Y = Y*0.75
num_Y = np.array([5,3,9,4,11,10,1,1,31,12,10,200,36,8])
# print("船只融载面积\n"+str(shell_sort(Y))+'\n')
# print("所有船总面积sum(Y) = " + str(sum(Y*num_Y)))

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

#一个旅对应的包裹
one_sum_baoguo =
    zhuang_baoguo([130,100,20,5,8,5,50],[47.74,34.72,27.4164,22.32,21.44,24.75,21.42],54,43.51851852,'一',12)
two_sum_baoguo =
    zhuang_baoguo([220,20,5,8,5,50],[34.72,27.4164,22.32,21.44,24.75,21.42],54,43.51851852,'二',3)
three_sum_baoguo =
    zhuang_baoguo([120,10,5,8,4,50],[34.72,27.4164,22.32,21.44,24.75,21.42],54,41.6666667,'三',3)
four_sum_baoguo = zhuang_baoguo([120,50],[27.4164,21.42],35,37.14285714,'四',3)
five_sum_baoguo = zhuang_baoguo([20,25,30],[21.44,24.75,30.38],35,28.5714285,'五',3)
six_sum_baoguo_feiji =
    zhuang_baoguo([40,30],[177.84,21.42],35,28.57142857,'六',5)#带飞机的旅，先装！！！！！！！
seven_sum_baoguo = zhuang_baoguo([30,15],[34.72,24.75],35,27.57142857,'七',5)
eight_sum_baoguo = zhuang_baoguo([150],[20.02],35,35.71428571,'八',3)
night_sum_baoguo = zhuang_baoguo([100],[20.58],35,32.85714286,'九',3)
ten_sum_baoguo = zhuang_baoguo([100],[21.42],35,28.57142857,'十',3)
elevn_sum_baoguo =
    zhuang_baoguo([100,105,18,12,50],[39.68,30.69,27.4164,22.32,21.42],35,71.42857143,'eleven',6)
twelve_sum_baoguo = zhuang_baoguo([],[],35,42.85714286,'twelve',10)
# sum_baoguo = [2000,140,1611,41,780,145,17,2510,1,148] #测试数据1
# sum_baoguo = one_sum_baoguo #测试装包数据
# print("第一旅的包裹\n"+str(sum_baoguo)+'\n')

# '''这里写几个旅的包裹相加'''
sum_baoguo =
    Merge(Merge(Merge(Merge(Merge(Merge(Merge(Merge(Merge(Merge(one_sum_baoguo,two_sum_baoguo),three_sum_baoguo
    +.........
# print(sum_baoguo)
first_baoguo = six_sum_baoguo_feiji
# # 总船只
sum_ship = sum_ship_list(Y,num_Y)
dict_ship = ship_list2dict(sum_ship)#船变字典
print(dict_ship)


'''包裹与船只分别从大到小排序'''
sum_baoguo = dict(kv_sort(sum_baoguo))
first_baoguo = dict(kv_sort(first_baoguo))
sum_ship = dict(kv_sort(dict_ship))
# print("总船只\n"+str(sum_ship)+'\n')

# '''装直升机的操作'''
sum_zhuangzai_ship, mod_dist = zhuangzai_zhishengji_ship(first_baoguo, sum_ship)
print('总船只装载飞机后的船只装载量\n'+str(sum_zhuangzai_ship)+'\n')
# print("总船只\n"+str(sum_ship)+'\n')
print("总包裹\n"+str(sum_baoguo)+'\n')
# '''106个直升机的包裹
#  总面积24105.163322842
#   开一个Z3装得下13300*2
#   剩下2494.836677158
#   '''
# print("剩余的包裹\n"+str(mod_dist)+'\n')
# z3 = 1
#
# '''装人的操作'''
sum_zhuangzai_ship,mod_dist = zhuangzai_ship(sum_baoguo,sum_zhuangzai_ship)
print('总船只装载后的装载量\n'+str(sum_zhuangzai_ship)+'\n')
print("装民船的面积的面积\n"+str(sum(mod_dist.values())))
```

```
mod_dist = dict(kv_sort(mod_dist))
# print("剩余的包裹\n"+str(mod_dist)+'\n')
#
#
# z1 = zhuangmingchuan(mod_dist,19000*0.7)
# z2 = zhuangmingchuan(mod_dist,26000*0.7)
# z3 += zhuangmingchuan(mod_dist,38000*0.7)
#Z的最大上限
# print(z1,z2,z3) #[25, 18, 14, 4, 11]

##################################################################################################################
#存活个体数
w = 50
live = []
Zx_w = [19000 * 0.7, 26000 * 0.7, 38000 * 0.7, 6000 * 0.7, 2000 * 0.7]
Zx_num = [25, 18, 6, 4, 11]

#初始化适应个体
while len(live) != w:
# 总民用船的数目
random_W =
    np.array([random.random(),random.random(),random.random(),random.random(),random.random()])
# 初始个体数目
num_Zx = np.trunc(random_W*Zx_num).astype(np.int8)#取整

# # 总民用船船只
sum_Z_ship = sum_ship_list(Zx_w, num_Zx)
# print("总民用船船只\n" + str(sum_Z_ship) + '\n')
dict_ship_Z = ship_list2dict_z(sum_Z_ship,num_Zx[0],num_Zx[1],num_Zx[2],num_Zx[3],num_Zx[4])#
    船变字典
dict_ship_Z['Z3_'+str(num_Zx[2])] = 2494.836677158
sum_Z_ship_dict = dict(kv_sort(dict_ship_Z))# 排序
# print("民用船船只字典和个数\n" + str(sum_Z_ship_dict) + '\n')
# print("剩余的包裹\n" + str(mod_dist) + '\n')
# print(mod_dist)
moddd = kv_deng(mod_dist)

shiying_ship, is_sure = zhuangzai_xuanze_ship(moddd, sum_Z_ship_dict)
# print("是否适应\n" + str(is_sure) + '\n')
moddd = kv_deng(mod_dist)

if is_sure:
num_Zx[2] += 1
live.append(num_Zx.tolist())
# print("装完物体后民用船适应度\n" + str(shiying_ship) + '\n')

# break
# print("适应个体: "+str(live)+'\n')

###############################################################################################################
# 计算适应度
t0 = surt_time(live[0])
print("计算适应度: : "+str(t0))
# 测试交配
# print(live[0],live[1])
# print(jiaopei_1(live[0],live[1]))
# print(jiaopei_2(live[0],live[1]))
#
# #测试变异
# print(live[0])
# print(bianyi(live[0],0.8))
#
# 测试是否满足适应
# print(is_surt_manzai(live[4],mod_dist))
#
###############################################################################################################
# 遗传的实现
 min_t = []
 duiying_num = []
 for i in range(100):
    the_all_son = live
    bianyilv = 0.2
    for i in range(w):
        #变异
        bianyi_son,is_bianyi = bianyi(live[i],bianyilv)
        if is_bianyi:
          the_all_son.append(bianyi_son)
   # print(len(the_all_son))

    #交配
    random.shuffle(live)
    for i in range(0,w-1,2):
        son = jiaopei_1(live[i],live[i+1])
        the_all_son.append(son)
        son1,son2 = jiaopei_2(live[i],live[i+1])
        the_all_son.append(son1)
        the_all_son.append(son2)
        # the_all_son.append()
   # print(len(the_all_son))


    t0_all = []
    surt_son = []
    for i in range(len(the_all_son)):
        if is_surt_manzai(the_all_son[i],mod_dist):
            t0_all.append(surt_time(the_all_son[i]))
            surt_son.append(the_all_son[i])
    #改进遗传算法种的实现
    # chongfu = []
    # for i in range(len(surt_son)):
    #    for j in range(i,len(surt_son)):
    #        if i!=j and surt_son[i]==surt_son[j]:
    #            # print(j)
    #            chongfu.append(i)
    # # print(t0_all)
    # # print(surt_son)
    #
    # for i in range(len(chongfu),0,-1):
    #    surt_son.pop(i)
   #    t0_all.pop(i)

    # print("变异后个体: "+str(len(the_all_son)))
    num_ship = []
    index = []
    for i in range(len(surt_son)):
        # if t0_all[i] == 21:
        #    print(surt_son[i])
        num_ship.append(sum(surt_son[i]))
        index.append(i)
    # print(t0_all)
    # print(num_ship)

    x = np.vstack((np.array(t0_all),np.array(num_ship),np.array(index)))
    # 第一行排序
    x = x.T[np.lexsort(x[::-1,:])].T
    min_t.append(x[0][0])
    duiying_num.append(x[1][0])

    youxiu_geti = []
    for i in range(w):
        youxiu_geti.append(surt_son[x[2][i]])
    print(youxiu_geti)
    print("时间: "+str(x[0]))
    print("船数: "+str(x[1]))
    live = youxiu_geti

plt.plot(duiying_num, label='种群中对应船只数量', )

plt.plot(min_t, label='种群中最小装载时间', color="r")
plt.xlabel("迭代次数", fontproperties=my_font)
plt.title("适应度变化趋势", fontproperties=my_font)
plt.legend(prop=my_font, loc=0)
plt.grid(alpha=0.3, linestyle="--") # alpha为透明度 0-1

plt.show()
###############################################################################################################
#####遍历Zx_num = [25, 18, 14, 4, 11]
# file = 'testaaaa.txt'
# with open(file, 'a+') as f:
#    for i_a in range(25):
#        for i_b in range(18):
#            for i_c in range(6):
#                for i_d in range(4):
#                    for i_e in range(11):
#                        # print(i_a,i_b,i_c,i_d,i_e)
#                        if is_surt_manzai([i_a,i_b,i_c,i_d,i_e], mod_dist):
#                            tqqq = surt_time([i_a,i_b,i_c,i_d,i_e])
#
    f.write(str(tqqq)+","+str([i_a,i_b,i_c,i_d,i_e])+","+str(sum([i_a,i_b,i_c,i_d,i_e]))+"\n")
#                            print("*"*20+str(tqqq))
###############################################################################################################
```