队伍编号	206281
题号	C

基于蚁群算法和遗传算法优化的仓内拣货问题

摘 要

在现在物流应用广泛的社会中，物流的任何一个环节的成本能够得到合理优化将节省巨大的物流经济成本，其中拣货成本是物流成本的重要部分。大部分仓储管理在相当大数量的货架中寻找需要的货物，在送往指定的复核台进行检验。如果路线和时间不进行规划的话，会耗费大量的时间成本寻找物资。因此，本文对于仓内拣货问题进行了初步分析后，尝试了蚁群算法、遗传算法等规划算法对其进行模拟路线与时间安排规划，最后对货架商品摆放提出了建议，且主要解决了如下问题：

问题 1 构建较为理想的仓库货柜与复核台共 3013 个模型，并计算任意两个模型之间的距离。根据已知模型构造将货柜分为左边朝向和右边朝向，将复核台分为向上朝向和向右朝向，对应坐标规定成取货/复核结束后离开 750mm 的坐标位置，通过一定的讨论，直接使用简化成坐标计算曼哈顿距离得到。

问题 2 考虑第一种情况，规划拣货员在固定复核台领取一个任务单后理想的拣货路线。在考虑了模拟退火和蚁群算法之后，选择了稳定性和可靠性较高的蚁群算法，模拟 200 和 400 只蚂蚁以及多次迭代得到较为准确的路线规划。

问题 3 基于问题 2，考虑第二种情况，规定拣货员在任选 2 个正常工作的复核台之一领取 5 个任务单或送还货物的路线与时间规划，延续了问题 2 的蚁群算法，分析了 4x4x5 种拣货路线，在贪心下得到了两个较好结果，其中一个没有复核台和拣货交付的时间冲突，得到一种最好的结果。

问题 4 基于问题 2，3，考虑第三种情况，规定 9 个拣货员在 4 个工作复核台之一任意领取 49 个任务单或送还货物的路线与时间分配规划，延续了问题 3 算法，分析了 4x4x49 种拣货路线，并且使用遗传算法规划时间分配问题。满足时间成本最小化。

问题 5 基于问题 4，考虑复核台从只有 1 个增加到有 5 个的情况，继续使用问题 2，3，4 的蚁群算法和遗传算法得到不同情况下的时间成本最小值，拟合出库时间与复核台数量的函数关系，评估增加一个正常工作的复核台对出库时间的影响。

问题 6 基于实际情况中拣货员步行时间和复核台等待排队之间的考量，设计一种基于打分机制的商品货架分区的仓内存储模型。

关键词：仓库拣货；优化调度；蚁群算法；遗传算法

目录

一、问题重述

1.1 问题背景

物流体系越来越完善丰富的现在，物流已经作为“第三利润源泉”越来越受重视，仓库是物流不可或缺的一部分，而拣货又是仓库作业中重要的一环。据统计拣货成本占到了整个仓库物流成本的 40%，拣货作业的效率大大影响了公司的服务质量，如何有效提升作业效率是拣货作业的最大课题。此外，对于优化算法后的时间成本的节省，评估与改进，对于建立客观有效的模型也十分重要。

1.2 问题要求与分析

问题 1. 根据已给货格和复核台的坐标设计计算 3000 个货格和 13 个复核台总共 3013 个元素之间距离的方法。

分析：该问题为路线距离计算问题，需要在具体运动路线模型的基础上，简化路线计算的关系，最终转化为比较简单的坐标直接计算

问题 2. 假设所有复核台正常工作，有一个任务单，拣货员在指定复核台领取了任务单 后，尝试规划理想的拣货路线。

分析：该问题属于固定经过节点后的最短路线规划问题，通过建立蚁群算法模型，计算出在完成任务基础上总行走路线最短，并建立数学模型用于实际应用。通过该问题可以解决从任意复核台出发或回来，结合任意任务单的具体内容，可以规划处需要的最短路线。

问题 3. 假设 2 个复核台正常工作，5 个任务单等待拣货，继续由一个拣货员负责拣货，初始位置固定，给拣货员指定任务领取顺序，规划理想的拣货路线，使得这些任务尽快出库并完成出库需要花费的时间和每个复核台利用率。

分析：该问题在最短路线规划思路上与问题二类似，区别在于该问题要考虑复核台复核任务清单上的货物和拣货员在各个货架之间拣货于返回的时间是同步进行的，可能存在拣货员拣货回来后复核台还没复核好上一批货物的情况，存在等待时间的可能，在规划好最短路线后，需要进一步考虑时间方面的问题。

问题 4. 假设 4 个复核台正常工作，49 个任务单等待拣货，9 个拣货员负责拣货，给每个拣货员分配任务单、起始拣货复核台，并分别规划理想的拣货路线，使得 49 个任务单尽快完成出库，并计算完成出库需要花费的时间和每个复核台的利用率。

分析：该问题在最短路线规划思路上于问题二类似，在时间规划上与问题三类似，区别在于情况更加复杂。在问题三中因为情况较少可以使用贪心找到最好结果，而问题四由于情况多样，最终采用了遗传算法通过优化目标函数的方式建模解决。

问题 5. 在问题 4 中，有 4 个复核台正常工作，请评估增加一个正常工作的复核台对出库时间的影响。

分析：该问题在整体规划思路上与问题四一致，我们考虑增加一个正常工作的复核台或者逐个减少正常工作的复核台，探究复核台个数与全部完成时间之间的函数关系。

问题 6. 商品在货架中的摆放位置，会影响拣货效率。若将畅销品放置在离复核台较近的位置，拣货员行走距离相应减少，但畅销品所在货架可能拥挤，反而降低拣货效率。对于仓内商品摆放问题，你有什么建议？

分析：该问题是一个与现实紧密联系，需要考虑实际问题中拣货步行时间与排队等待时间之间的权衡，结合畅销品订单数量多，订单内商品数量较多的特点，我们可以建立分级评分模型，进行定性的评估，进而给出建议。

1.3 问题相关信息

1. 附件 1 给出了货架货格复核台具体编号、坐标、大小等信息，以及任务单中的任务单号、订单号、商品货格、商品件数等信息。
2. 附件 2 给出了仓库图的示意图
3. 附件 3 给出了仓库左下角的距离路线示意图

二、模型假设

1. 假设拣货员领取任务单的时间不记，且到达复核台后无需等待，继续领取拣货车和任务单。
2. 假设拣货员负责多个任务单时，每次只能拣一个任务单的商品。
3. 假设拣货员的行走速度为 1.5m/s，对任意一个货格，若下架商品数量小于 3 件，每件完成下架花费 5 秒，否则每件花费 4 秒。
4. 假设多人同时在一个货格拣货，不需要考虑等待时间。
5. 假设每个订单复核和打包花费 30 秒。
6. 假设绕障碍物折现行走时横向和竖向偏移都取 d=750mm。
7. 假设复核台之间的距离简化为两复核台坐标差的绝对值之和。
8. 假设复核台和货格的距离简化为货格中点到复核台最近一条边中点的距离。

三、符号说明

符号	符号定义	单位或备注
TW(k, i, j)	任务单 k 从复核台 i 到复核台 j 所需步行时间	秒
TU(k)	任务单 k 中所有商品下架所需时间	秒
TP(k)	任务单 k 中所有订单复核和打包所需时间	秒
DR(k, i, j)	任务 k 从复核台 i 到复核台 j 所需步行路程	米
BS(k, m)	01 矩阵，若等于 1 则表示任务 k 需要到达货架 m	
Plan(p, i)	拣货员 p 所选择的第 i 个任务编号	
C(p)	拣货员 p 所选择的任务总数	
DM(m1, m2)	货架 m1 与货架 m2 之间的距离矩阵	毫米
DR(r1, r2)	复核台 r1 与复核台 r2 之间的距离矩阵	毫米
DMR(m, r)	货架 m 与复核台 r 之间的距离矩阵	毫米
BT(k, p)	01 矩阵，若等于 1 则表示任务单 k 为拣货员 p 负责	
BTP(k, p, i)	01 矩阵，若等于 1 则表示任务单 k 为拣货员 p 的第 i 个任务	

四、模型建立与求解

4.1 问题 1 的模型建立与求解

4.1.1 问题 1 的分析与求解思路

为了计算出 3000 个货格和 13 个复核台总共 3013 个元素之间的距离，解读题目大意：

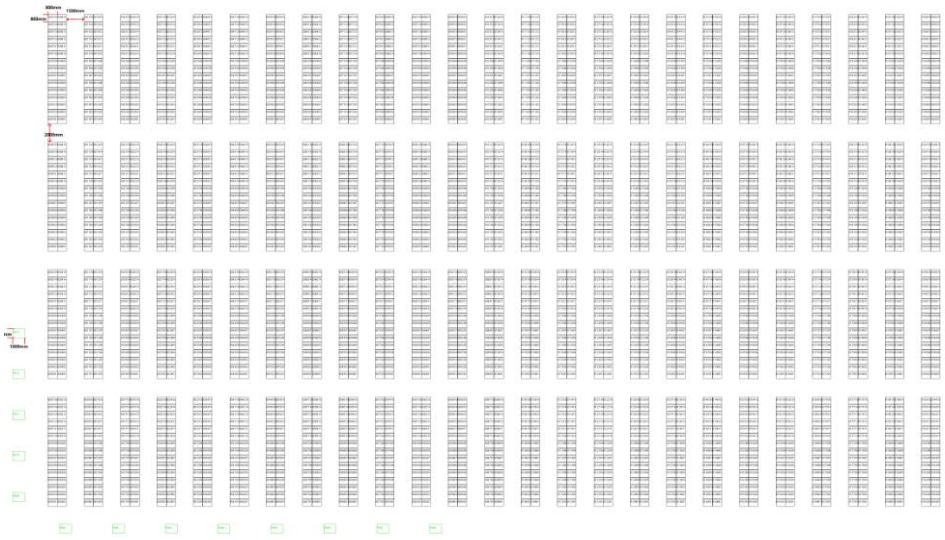


图 1.1 仓库示意图

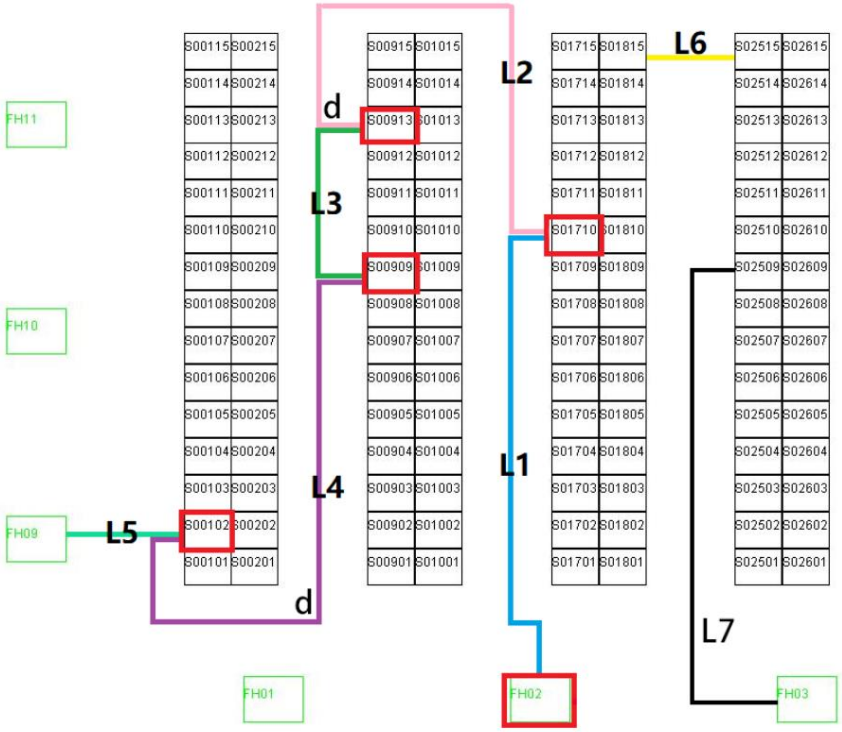


图 1.2 仓库左下角示意图

根据两个示意图我们可以看出，取货的位置在货格的中点，复核台复核的位置在每一个复核台的一边的中点，并且在路线行走中，必须时刻保持着与货格或者复核台 x 或 y 坐标间距范围大于 750mm。

4.1.2 问题 1 坐标简化模型的建立

4.1.2.1 几个事实

通过分析不难发现几个事实：

1. 货格到复核台间的距离：因为复核台的长和宽都是 1000mm，不论是在复核台的任何一边中点交付货物，其距离都是相等的。

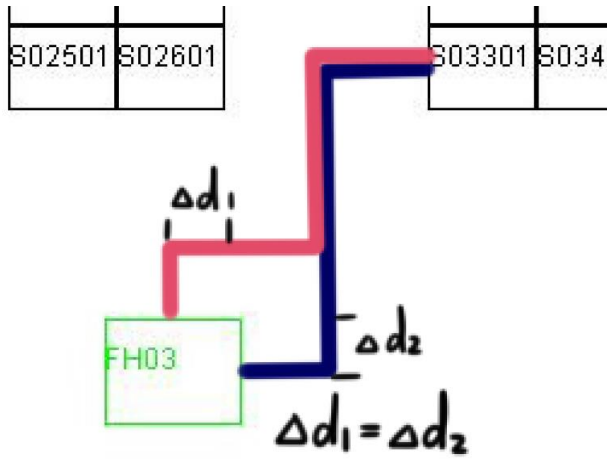


图 1.3 货格与复核台的不同行走路线对比

2. 货格间距离以及货格与复核台之间距离：每次距离货格/复核台折线行走时，都要保持 750mm 的横向和竖向偏移，因此，可以直接将坐标等效于取货完后离开 750mm 的位置

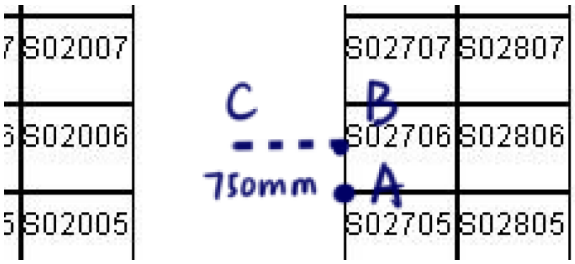


图 1.4 三个位置坐标信息

A：附件 1 给出的货格具体位置编号——对应每个货格的左下角

B：实际取货的位置

C：实际行走/取货完毕/复核完毕后与货格/复核台之间的距离

直接将货格/复核台的坐标都取成 C 点的计算坐标后，通过观察不难发现：

情况 1 货格与货格(复核台)不在同一行、或在同一列的情况：

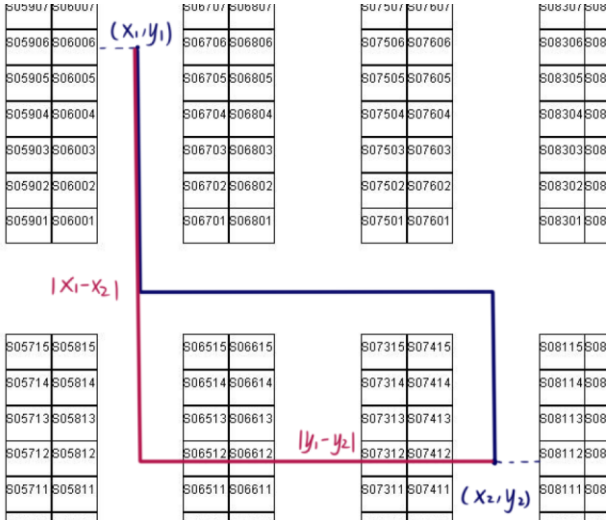


图 1.5 情况 1 货格与货格不在同一行的情况

如图 1.5 所示，取货格 S06006 的 C 位置坐标为(x1,y1)，取货格 S08112 的 C 位置坐标为(x2,y2)，可以直观看出不论是保证了横向或者竖向偏移 750mm，还是直接用坐标计算曼哈顿距离，得到的结果都是相同的。

得到情况 1 的计算公式：

$$\Delta L = |x_1 - x_2| + |y_1 - y_2| + 1500 \qquad (1.1)$$

情况 2 货格与货格(复核台)在同一行的情况：

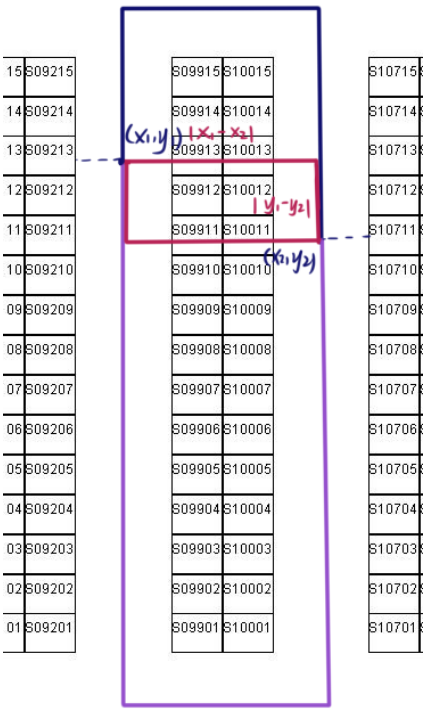


图 1.6 情况 2 货格与货格在同一行的情况

如图 1.6 所示，取货格 S09213 的 C 位置坐标为(x1,y1)，取货格 S08112 的 C 位置坐标为(x2,y2)，可以直观看出直接计算曼哈顿距离是不可行的，并且要考虑从上方走距离短还是从下方走距离短的情况。在纵坐标的考虑上会进行分类讨论，但在横坐标的考虑上依然保持不变。

所以得到情况 2 的计算公式：

$$\Delta L_x = |x_1 - x_2| \quad (1.2)$$

$$\Delta L_y = min\{\Delta L_{y\uparrow}, \Delta L_{y\downarrow}\} \quad (1.3)$$

$$\Delta L = \Delta L_x + \Delta L_y \quad (1.4)$$

情况 3 复核台与复核台

在模型假设中，提到了假设复核台之间的距离简化为两复核台坐标差的绝对值之和，故通过复核台的 A 坐标直接计算曼哈顿距离得出。

4.1.3 坐标简化模型的求解与 3013 个元素之间距离计算的求解

根据附件 1 中的每个元素的位置坐标，进一步推导出了每个元素的 A 位置、B 位置和 C 位置坐标。依照坐标简化模型的建立分别讨论四种情况下的位置计算公式，通过 Python 编码，计算出了 3013 个元素的 C 位置坐标。并根据上述规则分别计算出 3000*3000 的距离矩阵 DM，3000*13 的距离矩阵 DR，以及 13*13 的距离矩阵 DMR，其中部分信息如图 1.7

0	2300	3100	3900	4700	5500	6300
2300	0	2300	3100	3900	4700	5500
3100	2300	0	2300	3100	3900	4700
3900	3100	2300	0	2300	3100	3900
4700	3900	3100	2300	0	2300	3100
5500	4700	3900	3100	2300	0	2300
6300	5500	4700	3900	3100	2300	0
7100	6300	5500	4700	3900	3100	2300
7900	7100	6300	5500	4700	3900	3100
8700	7900	7100	6300	5500	4700	3900
9500	8700	7900	7100	6300	5500	4700
10300	9500	8700	7900	7100	6300	5500
11100	10300	9500	8700	7900	7100	6300
11900	11100	10300	9500	8700	7900	7100
7700	8500	9300	10100	10900	11700	12500
8500	9300	10100	10900	11700	12500	13300

图 1.7 货格、复核台间的距离矩阵

实现详见附录 1。

3013 个元素之间的距离表格参考详见附件 4。

4.2 问题 2 的模型建立与求解

4.2.1 问题 2 的分析与求解思路

分析题目，不难发现问题 2 主要是求解在必须经过指定位置的情况下，选择耗时最短的路线，也就是移动路程最小的路径。常见的最短路径求法不一定经过指定位置，所以需要对算法进行优化。

在满足上述条件的情况下，选出了几个合适的模型：枚举法、模拟退火算法、蚁群算法。

1. 枚举法适用于较少条件的情况下，显然该问有 23 个指定位置以及 13 个返回复核台的情况，数量过大，故排除。

2. 模拟退火法结合概率突跳特性在解空间随机寻找目标函数的全局最优解，即在局部最优解能概率性地跳出并最终趋于全局最优。但是由于其参数难控制，不能保证一次就收敛到最优值，一般需要尝试多次，且大部分情况还是会陷入局部最优值，故舍弃。

3 . 蚁群算法是基于蚂蚁生物的特点建立的，其基本思路是利用蚂蚁的行走路径表示优化问题的可行解，整个蚂蚁群体的所有路径构成待优化问题的解空间。路径较短的蚂蚁释放的信息素量较多，随着蚂蚁数量的增加以及迭代次数增加，最终整个蚂蚁群会在正反馈的作用下集中到最佳路径上，此时对应的便是待优化问题的最优解。

蚁群算法和模拟退火算法的区别在于，蚁群算法是通过正反馈机制使得搜索过程不断逼近至最优解，而模拟退火算法是在较大的范围随机寻找合适解，很可能陷入局部最优解，故在移动路程最小路径的算法中选择了蚁群算法。

继续分析题目，蚁群算法本质设定是为了解决旅行商问题，即要求蚂蚁从原点触发，经过若干个给定的需求点，最终返回原点的最短路径。为了完成整个闭环，我们把复核台之间的距离设定成 0.000001 以完成闭环并忽略两个复核台之间的距离，事实上在此题中，只需要蚂蚁完成闭环，就等效于拣货员从指定复核台领取任务后，回到某一个复核台交付。题目未指定返回的复核台，所以需要讨论返回 13 中复核台的情况，再取其中的时间最小值，至此为第二问的基本思路。

4.2.2 问题 2 的蚁群算法模型的建立

第二题要求我们求解拣货员在复核台 FH10 领取了任务单 T0001 的情况下的理想拣货路线，也就是完成拣货所需时间的最小值。从题目描述中我们已知拣货员的速度恒定，对于单一订单而言其从货架上拿货物的时间总和一定，因此我们只需最小化拣货员 P 行走的路程，并可以将其简化为一个固定了起点而不确定终点的旅行商问题。

变量说明:

$R=\{R_1,R_2,\dots,R_{13}\}$代表所有的复核台所在位置

$V_1=\{v_1,v_2,\cdots,v_{23}\}$代表 T001 任务单中 23 个订单的货架

目标函数:

$$minZ=\Sigma_k\Sigma_i\Sigma_j\left(DM_{i,j}XM_{i,j}+DR_{G1,j}XR_{G1,i}+DMR_{j,k}RX_{j,k}\right)$$

约束条件:

(1) $\Sigma_i\left(XM_{i,j}\right)=1 \quad i\in V_1$

(2) $\Sigma_j\left(XM_{i,j}\right)=1 \quad j\in V_1$

(3) $\Sigma_{i\epsilon S}\Sigma_{j\epsilon S}\left(XM_{i,j}\right)\le|S|-1 \quad \forall S\in V_1,2\le|S|\le 22$

(4) $\Sigma_i\left(XR_{R1,j}\right)=1 \quad i\in V_1$

(5) $\Sigma_j\Sigma_k\left(RX_{j,k}\right)=1 \quad j\in V_1,k\in R$

(6) $XM_{i,j}\in\{0,1\}$

(7) $XR_{R1,j}\in\{0,1\}$

(8) $RX_{j,k}\in\{0,1\}$

其中，约束条件（1）确保对于每一个货架，都只有一条路线进入其所在的位置。约束条件（2）确保对于每一个货架，都只有一条路线从其所在的位置出发。约束条件（3）保证了没有子回路解的产生。约束条件（4）表示对于初始点 R1 只有一条路径从中出发。约束条件（5）表示对于所有货架，有且只有一个货架会有从其出发抵达复核台的路径。

对于该路径规划问题，我们按照以下蚁群算法的方式进行求解：

1. 初始化变量，蚁群规模为 400，轨迹相对重要性为 1，能见度相对重要性为 1，最大迭代次数为 400，初始化信息素浓度，以及每只蚂蚁的出发点。
2. 每只蚂蚁根据轮盘选择法，选择下一个抵达的货格位置，同时按照局部信息素的更新规则进行信息素更新。
3. 在所有蚂蚁的路径表填满后，计算每只蚂蚁走过的路程总距离，并更新最短距离和路径信息。
4. 更新信息素矩阵，迭代次数加一，若迭代次数小于最大迭代次数，返回 Step2
5. 输出最终结果和最短路径信息

4.2.3 问题 2 的蚁群算法模型求解并规划最短路线

根据 13 次蚁群算法求解，可以得到由复核台 FH05 出发，最终返回各复核台的最短距离的集合。我们通过不同种群数量的蚂蚁和不同迭代次数进行多次求解，最终均收敛到一组解中，见表 2.1。

表 2.1 不同复核台作为终点的最短距离

复核台编号	**FH01**	**FH02**	**FH03**	**FH04**	**FH05**	**FH06**	**FH07**
最短距离(米)	**392.5**	**388.0**	**383.5**	**379.0**	**374.5**	**370.0**	**368.2**
复核台编号	**FH08**	**FH09**	**FH10**	**FH11**	**FH12**	**FH13**	
最短距离(米)	**372.7**	**393.0**	**392.3**	**390.6**	**389.6**	**385.1**	

由表 2.1 可知，复核台 FH07 的最短距离最小，总长为 392.5 米，该路线为：

FH10 > S00107 > S01713 > S01308 > S08532 > S07515 > S06213 > S07212 > S10115 > S11106 > S11205 > S12608 > S13509 > S15911 > S14401 > S14908 > S13812 > S14510 > S13809 > S13004 > S12103 > S10501 > S10508 > S07305 > FH07

其路线如下图所示，我们发现该路线在空间和时间的规划上的确是一个较优的选择。

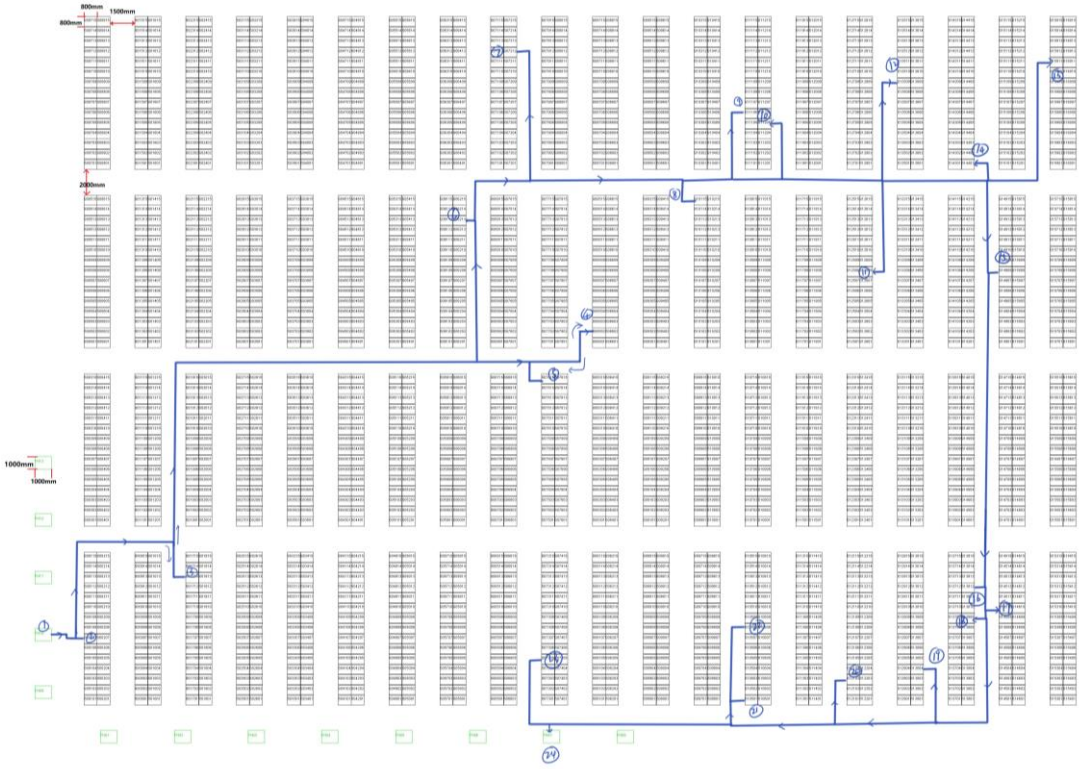


图 2.1 问题 2 答案可视化描述

已知完成出库需要的时间为拣货步行时间、下架商品时间和复核打包时间之和，即：

$$T = TW + TU + TP \quad (2.1)$$

拣货员步行时间：245.47s

总下架商品时间：1*5*13+2*5*4+3*4*6 = 177s

复核和打包时间：10*30 = 300s

总计出库花费时间为 722.47s

4.3 问题 3 的模型建立与求解

4.3.1 问题 3 的分析与求解思路

分析题目，此问在基于问题 2 基础上，设定了只有两个复核台正常工作，并且有五个任务单等待拣货，由于拣货员拣货和复核台复核货物是同步进行的，故可能存在当拣货员完成了一次任务返回复核台时，复核台仍未复核完上一次复核单上的所有货物的情况。

首先，延续问题 2 继续使用蚁群算法，讨论每个任务单在四种领取任务复核台和返回复核台的最短距离。初步讨论 2x2x5 共 20 种可能。

接下来优先选取五个任务单中路程最短情况，由于 20 种可能的排列组合关系较少，可能尝试是否可以存在耗时最短的最优解，事实上经过验证，恰有一种情况满足五个任务单路程最短，且拣货员拣货完返回时，复核台都处于空闲状态的情况，至此为第三问的基本思路。

4.3.2 问题 3 的数学描述与模型建立

问题三的求解思路与问题二类似。问题三在问题二的基础上取消了固定起点，并将任务单总数从 1 个增加到五个。因为仍只有一个拣货员，任务单确定，所以在问题三中等待的时间和从货架上取货物的时间不会影响优化结果，我们仍只需要最小化拣货员 P 走过的总路程即可。

变量描述:

$R_1 = \{R_3, R_{11}\}$代表所有的复核台所在位置

$Ts = \{ts_2, \dots, ts_6\}$代表 T002 至 T006 任务单

$V_k = \{v_1, v_2, \dots, v_n(k)\}$代表第 k 个任务单中 n(k)个订单的货架

目标函数:

$$minZ = \Sigma_{ts}[\Sigma_p \Sigma_q \Sigma_i \Sigma_j (DM_{i,j} XM_{i,j} + DR_{p,i} XR_{p,i} Y_{ts[p,q-1]} + DMR_{j,q} RX_{j,q})]$$

约束条件:

（1）$\Sigma_i (XM_{i,j}) = 1 \quad i \in V_k, k \in TS$

（2）$\Sigma_j (XM_{i,j}) = 1 \quad j \in V_k, k \in TS$

（3）$\Sigma_{i \epsilon S} \Sigma_{j \epsilon S} (XM_{i,j}) \leq |S| - 1 \quad \forall S \in V_1, 2 \leq |S| \leq 22$

（4）$\Sigma_p \Sigma_i (RX_{p,i}) = 1 \quad i \in V_k, p \in R_1, k \in TS$

（5）$\Sigma_q \Sigma_j (RX_{j,q}) = 1 \quad j \in V_k, q \in R_1, k \in TS$

（6）$\Sigma_p (Y_{ts[p,q-1]}) = 1 \quad p \in R_1$

（7）$XM_{i,j} \in \{0,1\}$

（8）$XR_{p,i} \in \{0,1\}$

（9）$RX_{j,q} \in \{0,1\}$

（10）$Y_{ts[p,q-1]} \in \{0,1\}$

其中，约束条件（1）（2）确保在每个任务单中，对于每一个货架，都只有一条路线进入其所在的位置且都只有一条路线从其所在的位置出发。约束条件（3）保证了对于每个订单没有子回路解的产生。约束条件（4）表示对于初始点只有一条路径从中出发。约束条件（5）表示对于所有货架，有且只有一个货架会有从其出发抵达复核台的路径。约束条件（6）表示，在每个订单中，有且只有一个出发点与上一个订单中的结束点是同一个复核台。

4.3.3 问题 3 的贪心任务领取规划

根据与问题二类似的方法，我们可以求出任务单 T0002-T0006 关于 FH03 和 FH10 间的最小路径矩阵，如表 3.1：

表 3.1 问题三 任务单-路径距离

任务单编号	FH03 – FH03 路径距离（米）	FH03 – FH10 路径距离（米）	FH10 – FH03 路径距离（米）	FH10 – FH10 路径距离（米）
T0002	370.2	351.0	351.0	359.5
T0003	375.5	376.8	376.8	396.5
T0004	355.9	349.6	349.6	364.5
T0005	352.5	361.2	361.2	364.9
T0006	430.3	423.2	423.2	423.7

通过贪心算法，我们首先取每个任务单的最短路径（如表 3.1 中黄色标注），计算拣货员所需要的步行时间和商品下架时间。其次我们计算复核台的复核与打包时间，发现该序列顺序并未发生排队现象。所有任务单都能在抵达对应的复核台后开始复核打包操作。但考虑复核打包时间后，我们需要对复核打包时间也进行贪心，此时，最后一个任务将选择 T0003，若将 T0003 作为最后一个任务，则会与上述贪心策略形成矛盾。但是，我们发现，将 T0006 的策略换成绿色格子中，也可以实现任务分配，因为 423.7-423.2 = 0.5 是表格中最小的非零差值，因此该替换后仍满足贪心条件，并且其步行时间和商品下架时间之和恰好大于上一个复核台的复核和打包时间 450s 因此最终分配情况如下表。

表 3.2 问题三 最优时间对应的任务序列

任务编号	步行时间(秒)	商品下架时间(秒)	复核打包时间(秒)	复核台编号
T0005	235	165	395	FH03
T0002	234	182	450	FH11
T0006	282.4	168	390	FH11
T0004	233.1	186	390	FH03
T0003	250.3	171	330	FH03

4.3.4 计算完成出库需要花费的时间和每个复核台的利用率

拣货员行走时间合计：1234.8s

商品下架时间合计：872s

最后复核台复核打包时间：330s

复核台 FH03 工作时间合计：1115s

复核台 FH11 工作时间合计：840s

TOTAL_TIME = 1234.8s+872s+330s = 2437.8s

复核台 FH03 利用率：45.74%

复核台 FH11 利用率：34.46%

4.4 问题 4 的模型建立与求解

4.4.1 问题 4 的分析与求解思路

分析题目，此问题在基于问题 2 的基础上，设定了有九个拣货员负责拣货，四个复核台正常工作和五个任务单等待拣货。我们基于问题 3 可以看出存在同步进行的时候拣货员等待复核台处于空闲状态的时间差。需要寻找其它算法对时间安排进行优化。

首先，延续问题 2 继续使用蚁群算法，讨论每个任务单在十六种领取任务复核台和返回复核台的最短距离。初步讨论 4x4x49 共 20 种可能。

接下来选取了解决耗时最短问题的模型，遗传算法。遗传算法是模拟达尔文生物进化论的自然选择和遗传学机理的生物进化过程的计算模型，通过模拟自然进化过程寻找最优解。

我们将遗传算法的几个需要的生物模型与实际问题对应：

1) DNA 序列：对应任务单的领取顺序，出发返回的复核台，每个任务单的拣货员

2) 适应性函数：最后一个完成任务结束复核时，总用时尽可能小

通过增加种群数量和选取代数，可以使 DNA 序列趋于最优解，通过反复增加，优化选取了多种群数量和多迭代次数下的最优解，至此为第四问的基本思路。

4.4.2 问题 4 的遗传算法模型建立

已知当每名拣货员的起始复核台确定、所对应的任务单及完成顺序固定、且每个任务到达复核台的编号确定时，所需要优化的函数（适应性函数）有确定解。设计如下算法，计算该确定解:

Algorithm 1 Calculate The Reviewer's MaxTime

Require:

The walking time required in task sheet k from review desk i to j, $TW(k,i,j)$;
The unshelve time required in task sheet k, $TU(k)$;
The packaging time required in task sheet k, $TP(k)$;
The No. of the ith task selected by picker p, $Plan(p,i)$;
The capicity of the tasks selected by picker p, $C(p)$;
The initial review desk where the picker p is located, $IR(p)$
The destination review desk in Task order k,$DR(k)$

$Q = \emptyset$
// ReviewDesk is recorded the working time in Review Desk
$Reviewer = [0,0,0,0]$
for each i $\in [1,9]$ **do**
  task $= Plan(i,0)$
  time $= TU(task) + TW(task, IR(i), DR(task))$
  $ENQUEUE(Q,(time,i,0,DR(task)))$
end for
while $Q \neq \emptyset$ **do**
  $time1, pid, cnt, rid \Leftarrow DEQUEUE(Q)$
  // time2 is the next time the picker will depart
  if $ReviewDesk[rid-1] > time1$ **then**
    $time2 \Leftarrow ReviewDesk[rid-1]$
    $ReviewDesk[rid-1] \Leftarrow ReviewDesk[rid-1] + TP(Plan(pid\ , cnt))$
  else
    $time2 = time1$
    $ReviewDesk[rid-1] \Leftarrow time1 + TP(Plan(pid, cnt))$
  end if
  $cnt \Leftarrow cnt + 1$
  if $cnt < C(pid)$ **then**
    $task \Leftarrow Plan(pid, cnt)$
    $time \Leftarrow time2 + TU(task) + TW(task, rid, DR(task))$
    $ENQUEUE(Q,(time,pid,cnt,DR(task)))$
  end if
end whilereturn $\max(ReviewDesk)$

4.4.3 问题 4 的遗传算法模型求解并规划人员与任务分配调度

a. 问题 4 的遗传算法模型求解并规划人员与任务分配调度

遗传算法采用开源的进化算法工具箱 Geatpy，在种群数量为 8000，最大代数为 1000 代的情况下利用增强精英保留的遗传算法模板进行数值模拟。最终得到一组任务分配方式，最后一项任务复核打包结束后总用时为 5556.0 秒。进化代数与最优个体目标函数值的结果如下

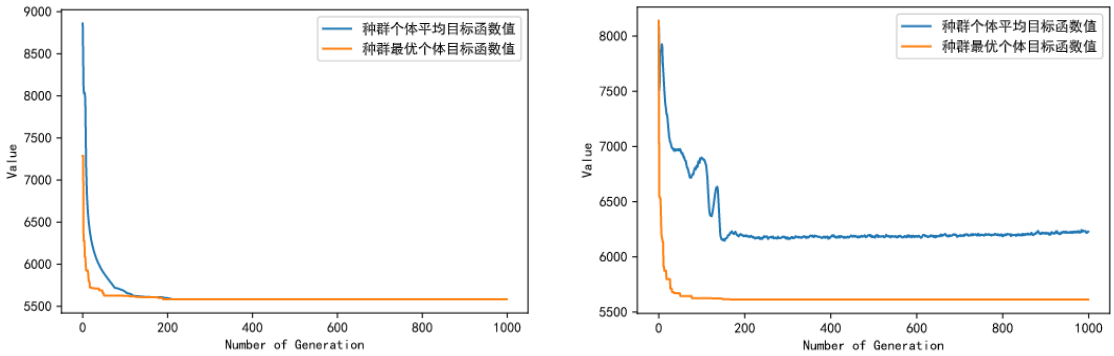


图 4.1 进化代数-最优个体目标函数值

b. 计算完成给每个拣货员分配的任务单、起始拣货复核台，规划理想的拣货路线，以及计算完成出库花费的时间和每个复核台的利用率

表 4.1 理想拣货路线的人员安排

人员标号	任务单及顺序
P1	T0045, T0031, T0040, T0010, T0029, T0026
P2	T0039, T0036, T0008, T0020, T0014
P3	T0009, T0043, T0012, T0041, T0028
P4	T0006, T0037, T0035, T0046
P5	T0033, T0049, T0001, T0013, T0025,
P6	T0015, T0027, T0007, T0047, T0038, T0004
P7	T0044, T0032, T0024, T0017, T0003, T0022
P8	T0034, T0023, T0018, T0016, T0002, T0019
P9	T0011, T0030, T0021, T0005, T0048

其中，每个任务抵达的复核台详见附件 4 和支撑文件

设计算法模拟九个拣货员和四个复核台的工作状态来计算复核台的工作时间，进而计算复核台利用率。

复核台 FH01 的工作总时间为 5130 秒，利用率为 91.03%

复核台 FH03 的工作总时间为 5130 秒，利用率为 91.03%

复核台 FH10 的工作总时间为 5100 秒，利用率为 90.50%

复核台 FH12 的工作总时间为 5160 秒，利用率为 91.56%

4.5 问题 5 的模型建立与求解

4.5.1 问题 4 的解法基础拓展

分析题目，此问题模型仍然可以沿用问题 4 的遗传算法进行优化来评估增加一个正常工作的复核台对出库时间的影响。据此我们选取 1-5 个复核台正常工作的情况下的对比，复核台的选择尽量满足下方复核台和左边复核台选取数量对称，这样的位置对于出库时间的影响占比较小，因此我们选择分别增加 FH02 和 FH11，计算出库时间。

获取复核台数量与出库时间的散点图后，尝试通过函数拟合描述复核台个数与出库时间的关系，至此为第五问的基本思路。

4.5.2 评估增加一个复核台对出库时间的影响

首先通过蚁群算法计算对于每个任务单在新增复核台的不同条件下的最短路径，并通过控制遗传算法自变量的取值范围来限制正常工作复核台的个数。绘制遗传算法得到的理想最短时间（见表 5.1）和复核台个数之间的关系。

复核台个数	1	2	3	4	5
理想最短时间 (秒)	20874	10638	7248	5556	4686

根据这五组数据，我们利用双曲线进行拟合，拟合结果如图 5.1，其决定系数 R^2 为 0.9999，效果较好。

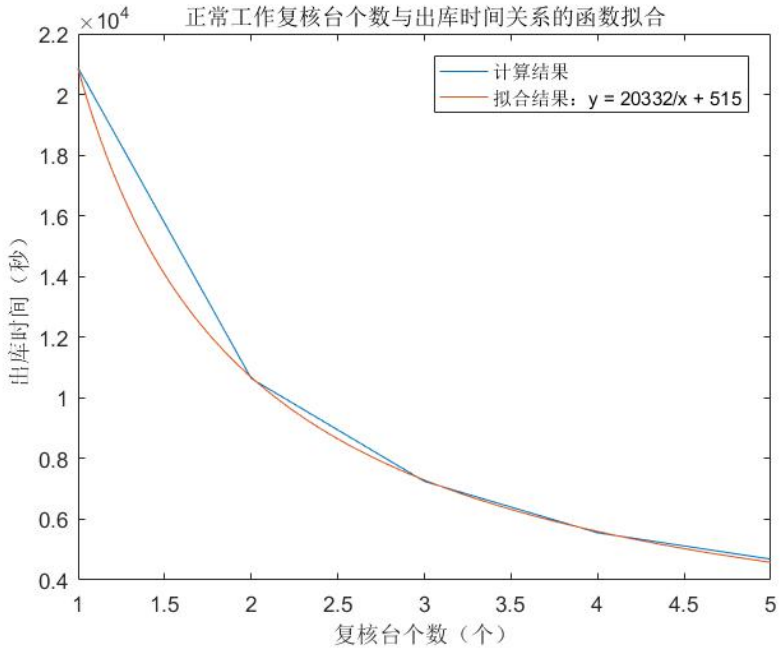


图 5.1 正常工作的复核台个数与出库时间关系的函数拟合

4.6 问题 6 的仓内商品的摆放建议

4.6.1 常见仓库的货物摆放规则

1. 最接近物流出口原则。 在规定固定货位和机动货位的基础上，要求物料摆放在离物流出口最近的位置上。

2. 以库存周转率为排序的依据的原则。 经常性的出入库频次高且出入量比较大的品种放在离物流出口最近的固定货位上。

3. 关联原则。 由于表或习惯，两个或两个以上相关联的物料被经常同时使用，如果放在相邻的位置，就可以缩短分拣人员的移动距离，提高工作效率。

4. 唯一原则。 （合格的）同一物料要求集中保管在唯一货位区域内，便于统一管理，避免多货位提货； 当然，自动化立体仓库不用严格遵守这个原则。

5. 隔离易混物料原则。 外观相近，用肉眼难以识别的物料，在标示清晰的基础上，要间隔 2 个以上的货位，防止混在一起，难以区分。

6. 通过先进先出，进行严格管理，同一批号的商品如果检验不合格或者早产不允许放行，要设立红牌警示，避免混乱。

7. 合理搭配原则。 要考虑物料的形状大小，根据实际仓库的条件，合理搭配空间； 避免空间不足多货位放货，避免空间太大使用不充分。

8. 上轻下重原则。 楼上或上层货位摆放重量轻的物料，楼下或者下层货位摆放重量大的物料，这样可以减轻搬运强度，保证货架、建筑与人员的安全。

9. 目视化看板原则。 绘制《货位平面图》，标明商品明确的货位，即使是临时人员，也能准确无误的分拣出正确的商品。

10. 面对通道原则。 即把商品的标示面对通道，不仅是把外面的一层面对通道，而且要把所有的商品标示都要面对通道，面对同一方向，使分拣人员能够始终流畅地进行工作，不用中断工作去确认标示。 不围不堵。

4.6.2 对仓内商品摆放问题的建议

1. 对于出入量比较大且出入库频次高的畅销商品，优先摆放在离复核台的最近的货架。

2. 对于同一个任务单的商品，尽量保证其对应的货架相对集中；对于不同任务单的商品，尽量使其避免空间拥挤。

我们设计了一个基于打分的商品存放分区模型。首先，我们计算出各个货架到每个复核台之间的平均距离，对于分数较高的商品选择平均距离复核台较近的货架。

我们探讨设计了两种区域，一种为短期爆火型商品区，另一种为长期需求型商品区。根据不同的分区有如下打分标准：

1. 短期爆火型商品（季节性、节日性、网红产品）：

   A．销售程度：对于短期商品，由于商品更换频繁，我们可以选取周销量作为评价标准进行打分。

   B．关联效应：同款产品或类似产品销量较高时，该产品也会获得更高的打分。

   C．订单数量：考虑到拥挤程度对拣货效率的影响，对于订单取货较多的产品，会给予一定的惩罚的分值。

   D．预期产品：对于周期性、时令性等预估在短时间内会爆火的产品，该类产品也会得到更多的分值。

2. 长期需求型商品：

   A. 销售程度：对于短期商品，由于商品更换较不频繁，我们可以选取月销量作为评价标准进行打分。

   B. 关联效应：同款产品或类似产品销量较高时，该产品也会获得更高的打分。

   C. 订单数量：考虑到拥挤程度对拣货效率的影响，对于订单取货较多的产品，会给予一定的惩罚的分值。

五、模型的优缺点分析

5.1 模型的优点分析

我们设计了一个基于蚁群算法和遗传算法优化的仓内拣货最佳路径规划的模型，巧妙地利用了货格与复核台的空间关系和取货、复核、打包等流程的时间关系。

蚁群算法采用正反馈机制，随着迭代次数的增加，蚁群会趋于选择残留信息素最多，即路径长度最短的结果。遗传算法通过优势基因的保留和遗忘策略，随着进化代数的增加，种群将会逐渐符合筛选标准。蚁群算法和遗传算法都较难陷入局部的最优解。

5.2 模型的缺点分析

蚁群算法和遗传算法均作为元启发式算法的代表，其运算效率和结果与其迭代次数有较大关系，当迭代次数较少时可能会落入局部最优解与全局最优解可能有较大的差距。此外，该算法的时间成本和实际效益之间的权衡也是在实际应用中需要考虑的。在我们的计算中，很多结果都是通过较长的时间迭代得到的，其中以第四题的遗传算法为例，在 50000 个种群，700 代遗传的情况下，计算用时约为一个半小时，得到最优结果为 5559.7 秒。但这个结果仅比运行 10 秒的 500 种群，100 代遗传的 5672.7 秒提高了 13 秒。

参考文献

[1] Geatpy development team, Geatpy 教程, http://geatpy.com/index.php/details/, 访问于 2020/5/23

[2] YisuZhou, 遗传算法、禁忌搜索、模拟退火、蚁群算法, https://github.com/YisuZhou/TSP , 访问于 2020/5/22

[3] Sebastian Henn, Verena Schmid, Metaheuristics for order batching and sequencing in manual order picking systems, Computers & Industrial Engineering, Volume 66, Issue 2, Pages 338-351, 2013, https://doi.org/10.1016/j.cie.2013.07.003.

[4] Tzu-Li Chen, Chen-Yang Cheng, Yin-Yann Chen, Li-Kai Chan, An efficient hybrid algorithm for integrated order batching, sequencing and routing problem, International Journal of Production Economics, Volume 159, Pages 158-167, 2015, https://doi.org/10.1016/j.ijpe.2014.09.029.

[5] 易库电商仓储, 电商仓库设计原则 http://www.yikucangchu.com/content/warehouse-design, 访问于 2020/5/24

附录

使用到的软件名称：Matlab, Jupyter lab, JetBrains IDEA, Notepad++

计算结果

问题 4 理想规划结果

任务单,起始复核台,
抵达复核台,拣货时间
T0001,FH01,FH01,266.933
T0001,FH01,FH03,260.933
T0001,FH01,FH10,261.667
T0001,FH01,FH12,264.733
T0001,FH03,FH01,260.933
T0001,FH03,FH03,266.933
T0001,FH03,FH10,255.667
T0001,FH03,FH12,261.667
T0001,FH10,FH01,261.667
T0001,FH10,FH03,255.667
T0001,FH10,FH10,261.533
T0001,FH10,FH12,259.733
T0001,FH12,FH01,264.733
T0001,FH12,FH03,261.667
T0001,FH12,FH10,259.733
T0001,FH12,FH12,265.733
T0002,FH01,FH01,266.933
T0002,FH01,FH03,260.933
T0002,FH01,FH10,261.667
T0002,FH01,FH12,264.733
T0002,FH03,FH01,260.933
T0002,FH03,FH03,266.933
T0002,FH03,FH10,255.667
T0002,FH03,FH12,261.667
T0002,FH10,FH01,261.667
T0002,FH10,FH03,255.667
T0002,FH10,FH10,261.533
T0002,FH10,FH12,259.733
T0002,FH12,FH01,264.733
T0002,FH12,FH03,261.667
T0002,FH12,FH10,259.733
T0002,FH12,FH12,265.733
T0003,FH01,FH01,266.933
T0003,FH01,FH03,260.933
T0003,FH01,FH10,261.667
T0003,FH01,FH12,264.733
T0003,FH03,FH01,260.933
T0003,FH03,FH03,266.933
T0003,FH03,FH10,255.667
T0003,FH03,FH12,261.667
T0003,FH10,FH01,261.667
T0003,FH10,FH03,255.667
T0003,FH10,FH10,261.533
T0003,FH10,FH12,259.733
T0003,FH12,FH01,264.733
T0003,FH12,FH03,261.667
T0003,FH12,FH10,259.733
T0003,FH12,FH12,265.733
T0004,FH01,FH01,266.933
T0004,FH01,FH03,260.933
T0004,FH01,FH10,261.667
T0004,FH01,FH12,264.733
T0004,FH03,FH01,260.933
T0004,FH03,FH03,266.933
T0004,FH03,FH10,255.667
T0004,FH03,FH12,261.667
T0004,FH10,FH01,261.667
T0004,FH10,FH03,255.667
T0004,FH10,FH10,261.533
T0004,FH10,FH12,259.733
T0004,FH12,FH01,264.733
T0004,FH12,FH03,261.667
T0004,FH12,FH10,259.733
T0004,FH12,FH12,265.733
T0005,FH01,FH01,266.933
T0005,FH01,FH03,260.933
T0005,FH01,FH10,261.667
T0005,FH01,FH12,264.733
T0005,FH03,FH01,260.933
T0005,FH03,FH03,266.933
T0005,FH03,FH10,255.667
T0005,FH03,FH12,261.667
T0005,FH10,FH01,261.667
T0005,FH10,FH03,255.667
T0005,FH10,FH10,261.533
T0005,FH10,FH12,259.733
T0005,FH12,FH01,264.733
T0005,FH12,FH03,261.667
T0005,FH12,FH10,259.733
T0005,FH12,FH12,265.733
T0006,FH01,FH01,266.933
T0006,FH01,FH03,260.933
T0006,FH01,FH10,261.667
T0006,FH01,FH12,264.733
T0006,FH03,FH01,260.933
T0006,FH03,FH03,266.933
T0006,FH03,FH10,255.667
T0006,FH03,FH12,261.667
T0006,FH10,FH01,261.667
T0006,FH10,FH03,255.667
T0006,FH10,FH10,261.533
T0006,FH10,FH12,259.733
T0006,FH12,FH01,264.733
T0006,FH12,FH03,261.667
T0006,FH12,FH10,259.733
T0006,FH12,FH12,265.733
T0007,FH01,FH01,266.933
T0007,FH01,FH03,260.933
T0007,FH01,FH10,261.667
T0007,FH01,FH12,264.733
T0007,FH03,FH01,260.933
T0007,FH03,FH03,266.933
T0007,FH03,FH10,255.667
T0007,FH03,FH12,261.667
T0007,FH10,FH01,261.667
T0007,FH10,FH03,255.667
T0007,FH10,FH10,261.533
T0007,FH10,FH12,259.733
T0007,FH12,FH01,264.733
T0007,FH12,FH03,261.667
T0007,FH12,FH10,259.733
T0007,FH12,FH12,265.733
T0008,FH01,FH01,266.933
T0008,FH01,FH03,260.933
T0008,FH01,FH10,261.667

T0008,FH01,FH12,264.733
T0008,FH03,FH01,260.933
T0008,FH03,FH03,266.933
T0008,FH03,FH10,255.667
T0008,FH03,FH12,261.667
T0008,FH10,FH01,261.667
T0008,FH10,FH03,255.667
T0008,FH10,FH10,261.533
T0008,FH10,FH12,259.733
T0008,FH12,FH01,264.733
T0008,FH12,FH03,261.667
T0008,FH12,FH10,259.733
T0008,FH12,FH12,265.733
T0009,FH01,FH01,266.933
T0009,FH01,FH03,260.933
T0009,FH01,FH10,261.667
T0009,FH01,FH12,264.733
T0009,FH03,FH01,260.933
T0009,FH03,FH03,266.933
T0009,FH03,FH10,255.667
T0009,FH03,FH12,261.667
T0009,FH10,FH01,261.667
T0009,FH10,FH03,255.667
T0009,FH10,FH10,261.533
T0009,FH10,FH12,259.733
T0009,FH12,FH01,264.733
T0009,FH12,FH03,261.667
T0009,FH12,FH10,259.733
T0009,FH12,FH12,265.733
T0010,FH01,FH01,266.933
T0010,FH01,FH03,260.933
T0010,FH01,FH10,261.667
T0010,FH01,FH12,264.733
T0010,FH03,FH01,260.933
T0010,FH03,FH03,266.933
T0010,FH03,FH10,255.667
T0010,FH03,FH12,261.667
T0010,FH10,FH01,261.667
T0010,FH10,FH03,255.667
T0010,FH10,FH10,261.533
T0010,FH10,FH12,259.733
T0010,FH12,FH01,264.733
T0010,FH12,FH03,261.667
T0010,FH12,FH10,259.733
T0010,FH12,FH12,265.733
T0011,FH01,FH01,266.933
T0011,FH01,FH03,260.933
T0011,FH01,FH10,261.667
T0011,FH01,FH12,264.733
T0011,FH03,FH01,260.933
T0011,FH03,FH03,266.933
T0011,FH03,FH10,255.667
T0011,FH03,FH12,261.667
T0011,FH10,FH01,261.667
T0011,FH10,FH03,255.667
T0011,FH10,FH10,261.533
T0011,FH10,FH12,259.733
T0011,FH12,FH01,264.733
T0011,FH12,FH03,261.667
T0011,FH12,FH10,259.733
T0011,FH12,FH12,265.733
T0012,FH01,FH01,266.933
T0012,FH01,FH03,260.933
T0012,FH01,FH10,261.667
T0012,FH01,FH12,264.733
T0012,FH03,FH01,260.933
T0012,FH03,FH03,266.933
T0012,FH03,FH10,255.667
T0012,FH03,FH12,261.667
T0012,FH10,FH01,261.667
T0012,FH10,FH03,255.667
T0012,FH10,FH10,261.533
T0012,FH10,FH12,259.733
T0012,FH12,FH01,264.733
T0012,FH12,FH03,261.667
T0012,FH12,FH10,259.733
T0012,FH12,FH12,265.733
T0013,FH01,FH01,266.933
T0013,FH01,FH03,260.933
T0013,FH01,FH10,261.667
T0013,FH01,FH12,264.733
T0013,FH03,FH01,260.933
T0013,FH03,FH03,266.933
T0013,FH03,FH10,255.667
T0013,FH03,FH12,261.667
T0013,FH10,FH01,261.667
T0013,FH10,FH03,255.667
T0013,FH10,FH10,261.533
T0013,FH10,FH12,259.733
T0013,FH12,FH01,264.733
T0013,FH12,FH03,261.667
T0013,FH12,FH10,259.733
T0013,FH12,FH12,265.733
T0014,FH01,FH01,266.933
T0014,FH01,FH03,260.933
T0014,FH01,FH10,261.667
T0014,FH01,FH12,264.733
T0014,FH03,FH01,260.933
T0014,FH03,FH03,266.933
T0014,FH03,FH10,255.667
T0014,FH03,FH12,261.667
T0014,FH10,FH01,261.667
T0014,FH10,FH03,255.667
T0014,FH10,FH10,261.533
T0014,FH10,FH12,259.733
T0014,FH12,FH01,264.733
T0014,FH12,FH03,261.667
T0014,FH12,FH10,259.733
T0014,FH12,FH12,265.733
T0015,FH01,FH01,266.933
T0015,FH01,FH03,260.933
T0015,FH01,FH10,261.667
T0015,FH01,FH12,264.733
T0015,FH03,FH01,260.933
T0015,FH03,FH03,266.933
T0015,FH03,FH10,255.667
T0015,FH03,FH12,261.667
T0015,FH10,FH01,261.667
T0015,FH10,FH03,255.667
T0015,FH10,FH10,261.533
T0015,FH10,FH12,259.733
T0015,FH12,FH01,264.733
T0015,FH12,FH03,261.667
T0015,FH12,FH10,259.733
T0015,FH12,FH12,265.733
T0016,FH01,FH01,266.933
T0016,FH01,FH03,260.933
T0016,FH01,FH10,261.667
T0016,FH01,FH12,264.733
T0016,FH03,FH01,260.933
T0016,FH03,FH03,266.933
T0016,FH03,FH10,255.667
T0016,FH03,FH12,261.667
T0016,FH10,FH01,261.667
T0016,FH10,FH03,255.667
T0016,FH10,FH10,261.533
T0016,FH10,FH12,259.733
T0016,FH12,FH01,264.733

T0016,FH12,FH03,261.667
T0016,FH12,FH10,259.733
T0016,FH12,FH12,265.733
T0017,FH01,FH01,266.933
T0017,FH01,FH03,260.933
T0017,FH01,FH10,261.667
T0017,FH01,FH12,264.733
T0017,FH03,FH01,260.933
T0017,FH03,FH03,266.933
T0017,FH03,FH10,255.667
T0017,FH03,FH12,261.667
T0017,FH10,FH01,261.667
T0017,FH10,FH03,255.667
T0017,FH10,FH10,261.533
T0017,FH10,FH12,259.733
T0017,FH12,FH01,264.733
T0017,FH12,FH03,261.667
T0017,FH12,FH10,259.733
T0017,FH12,FH12,265.733
T0018,FH01,FH01,266.933
T0018,FH01,FH03,260.933
T0018,FH01,FH10,261.667
T0018,FH01,FH12,264.733
T0018,FH03,FH01,260.933
T0018,FH03,FH03,266.933
T0018,FH03,FH10,255.667
T0018,FH03,FH12,261.667
T0018,FH10,FH01,261.667
T0018,FH10,FH03,255.667
T0018,FH10,FH10,261.533
T0018,FH10,FH12,259.733
T0018,FH12,FH01,264.733
T0018,FH12,FH03,261.667
T0018,FH12,FH10,259.733
T0018,FH12,FH12,265.733
T0019,FH01,FH01,266.933
T0019,FH01,FH03,260.933
T0019,FH01,FH10,261.667
T0019,FH01,FH12,264.733
T0019,FH03,FH01,260.933
T0019,FH03,FH03,266.933
T0019,FH03,FH10,255.667
T0019,FH03,FH12,261.667
T0019,FH10,FH01,261.667
T0019,FH10,FH03,255.667
T0019,FH10,FH10,261.533
T0019,FH10,FH12,259.733
T0019,FH12,FH01,264.733
T0019,FH12,FH03,261.667
T0019,FH12,FH10,259.733
T0019,FH12,FH12,265.733
T0020,FH01,FH01,266.933
T0020,FH01,FH03,260.933
T0020,FH01,FH10,261.667
T0020,FH01,FH12,264.733
T0020,FH03,FH01,260.933
T0020,FH03,FH03,266.933
T0020,FH03,FH10,255.667
T0020,FH03,FH12,261.667
T0020,FH10,FH01,261.667
T0020,FH10,FH03,255.667
T0020,FH10,FH10,261.533
T0020,FH10,FH12,259.733
T0020,FH12,FH01,264.733
T0020,FH12,FH03,261.667
T0020,FH12,FH10,259.733
T0020,FH12,FH12,265.733
T0021,FH01,FH01,266.933
T0021,FH01,FH03,260.933
T0021,FH01,FH10,261.667
T0021,FH01,FH12,264.733
T0021,FH03,FH01,260.933
T0021,FH03,FH03,266.933
T0021,FH03,FH10,255.667
T0021,FH03,FH12,261.667
T0021,FH10,FH01,261.667
T0021,FH10,FH03,255.667
T0021,FH10,FH10,261.533
T0021,FH10,FH12,259.733
T0021,FH12,FH01,264.733
T0021,FH12,FH03,261.667
T0021,FH12,FH10,259.733
T0021,FH12,FH12,265.733
T0022,FH01,FH01,266.933
T0022,FH01,FH03,260.933
T0022,FH01,FH10,261.667
T0022,FH01,FH12,264.733
T0022,FH03,FH01,260.933
T0022,FH03,FH03,266.933
T0022,FH03,FH10,255.667
T0022,FH03,FH12,261.667
T0022,FH10,FH01,261.667
T0022,FH10,FH03,255.667
T0022,FH10,FH10,261.533
T0022,FH10,FH12,259.733
T0022,FH12,FH01,264.733
T0022,FH12,FH03,261.667
T0022,FH12,FH10,259.733
T0022,FH12,FH12,265.733
T0023,FH01,FH01,266.933
T0023,FH01,FH03,260.933
T0023,FH01,FH10,261.667
T0023,FH01,FH12,264.733
T0023,FH03,FH01,260.933
T0023,FH03,FH03,266.933
T0023,FH03,FH10,255.667
T0023,FH03,FH12,261.667
T0023,FH10,FH01,261.667
T0023,FH10,FH03,255.667
T0023,FH10,FH10,261.533
T0023,FH10,FH12,259.733
T0023,FH12,FH01,264.733
T0023,FH12,FH03,261.667
T0023,FH12,FH10,259.733
T0023,FH12,FH12,265.733
T0024,FH01,FH01,266.933
T0024,FH01,FH03,260.933
T0024,FH01,FH10,261.667
T0024,FH01,FH12,264.733
T0024,FH03,FH01,260.933
T0024,FH03,FH03,266.933
T0024,FH03,FH10,255.667
T0024,FH03,FH12,261.667
T0024,FH10,FH01,261.667
T0024,FH10,FH03,255.667
T0024,FH10,FH10,261.533
T0024,FH10,FH12,259.733
T0024,FH12,FH01,264.733
T0024,FH12,FH03,261.667
T0024,FH12,FH10,259.733
T0024,FH12,FH12,265.733
T0025,FH01,FH01,266.933
T0025,FH01,FH03,260.933
T0025,FH01,FH10,261.667
T0025,FH01,FH12,264.733
T0025,FH03,FH01,260.933
T0025,FH03,FH03,266.933
T0025,FH03,FH10,255.667

T0025,FH03,FH12,261.667
T0025,FH10,FH01,261.667
T0025,FH10,FH03,255.667
T0025,FH10,FH10,261.533
T0025,FH10,FH12,259.733
T0025,FH12,FH01,264.733
T0025,FH12,FH03,261.667
T0025,FH12,FH10,259.733
T0025,FH12,FH12,265.733
T0026,FH01,FH01,266.933
T0026,FH01,FH03,260.933
T0026,FH01,FH10,261.667
T0026,FH01,FH12,264.733
T0026,FH03,FH01,260.933
T0026,FH03,FH03,266.933
T0026,FH03,FH10,255.667
T0026,FH03,FH12,261.667
T0026,FH10,FH01,261.667
T0026,FH10,FH03,255.667
T0026,FH10,FH10,261.533
T0026,FH10,FH12,259.733
T0026,FH12,FH01,264.733
T0026,FH12,FH03,261.667
T0026,FH12,FH10,259.733
T0026,FH12,FH12,265.733
T0027,FH01,FH01,266.933
T0027,FH01,FH03,260.933
T0027,FH01,FH10,261.667
T0027,FH01,FH12,264.733
T0027,FH03,FH01,260.933
T0027,FH03,FH03,266.933
T0027,FH03,FH10,255.667
T0027,FH03,FH12,261.667
T0027,FH10,FH01,261.667
T0027,FH10,FH03,255.667
T0027,FH10,FH10,261.533
T0027,FH10,FH12,259.733
T0027,FH12,FH01,264.733
T0027,FH12,FH03,261.667
T0027,FH12,FH10,259.733
T0027,FH12,FH12,265.733
T0028,FH01,FH01,266.933
T0028,FH01,FH03,260.933
T0028,FH01,FH10,261.667
T0028,FH01,FH12,264.733
T0028,FH03,FH01,260.933
T0028,FH03,FH03,266.933
T0028,FH03,FH10,255.667
T0028,FH03,FH12,261.667
T0028,FH10,FH01,261.667
T0028,FH10,FH03,255.667
T0028,FH10,FH10,261.533
T0028,FH10,FH12,259.733
T0028,FH12,FH01,264.733
T0028,FH12,FH03,261.667
T0028,FH12,FH10,259.733
T0028,FH12,FH12,265.733
T0029,FH01,FH01,266.933
T0029,FH01,FH03,260.933
T0029,FH01,FH10,261.667
T0029,FH01,FH12,264.733
T0029,FH03,FH01,260.933
T0029,FH03,FH03,266.933
T0029,FH03,FH10,255.667
T0029,FH03,FH12,261.667
T0029,FH10,FH01,261.667
T0029,FH10,FH03,255.667
T0029,FH10,FH10,261.533
T0029,FH10,FH12,259.733
T0029,FH12,FH01,264.733
T0029,FH12,FH03,261.667
T0029,FH12,FH10,259.733
T0029,FH12,FH12,265.733
T0030,FH01,FH01,266.933
T0030,FH01,FH03,260.933
T0030,FH01,FH10,261.667
T0030,FH01,FH12,264.733
T0030,FH03,FH01,260.933
T0030,FH03,FH03,266.933
T0030,FH03,FH10,255.667
T0030,FH03,FH12,261.667
T0030,FH10,FH01,261.667
T0030,FH10,FH03,255.667
T0030,FH10,FH10,261.533
T0030,FH10,FH12,259.733
T0030,FH12,FH01,264.733
T0030,FH12,FH03,261.667
T0030,FH12,FH10,259.733
T0030,FH12,FH12,265.733
T0031,FH01,FH01,266.933
T0031,FH01,FH03,260.933
T0031,FH01,FH10,261.667
T0031,FH01,FH12,264.733
T0031,FH03,FH01,260.933
T0031,FH03,FH03,266.933
T0031,FH03,FH10,255.667
T0031,FH03,FH12,261.667
T0031,FH10,FH01,261.667
T0031,FH10,FH03,255.667
T0031,FH10,FH10,261.533
T0031,FH10,FH12,259.733
T0031,FH12,FH01,264.733
T0031,FH12,FH03,261.667
T0031,FH12,FH10,259.733
T0031,FH12,FH12,265.733
T0032,FH01,FH01,266.933
T0032,FH01,FH03,260.933
T0032,FH01,FH10,261.667
T0032,FH01,FH12,264.733
T0032,FH03,FH01,260.933
T0032,FH03,FH03,266.933
T0032,FH03,FH10,255.667
T0032,FH03,FH12,261.667
T0032,FH10,FH01,261.667
T0032,FH10,FH03,255.667
T0032,FH10,FH10,261.533
T0032,FH10,FH12,259.733
T0032,FH12,FH01,264.733
T0032,FH12,FH03,261.667
T0032,FH12,FH10,259.733
T0032,FH12,FH12,265.733
T0033,FH01,FH01,266.933
T0033,FH01,FH03,260.933
T0033,FH01,FH10,261.667
T0033,FH01,FH12,264.733
T0033,FH03,FH01,260.933
T0033,FH03,FH03,266.933
T0033,FH03,FH10,255.667
T0033,FH03,FH12,261.667
T0033,FH10,FH01,261.667
T0033,FH10,FH03,255.667
T0033,FH10,FH10,261.533
T0033,FH10,FH12,259.733
T0033,FH12,FH01,264.733
T0033,FH12,FH03,261.667
T0033,FH12,FH10,259.733
T0033,FH12,FH12,265.733
T0034,FH01,FH01,266.933

T0034,FH01,FH03,260.933
T0034,FH01,FH10,261.667
T0034,FH01,FH12,264.733
T0034,FH03,FH01,260.933
T0034,FH03,FH03,266.933
T0034,FH03,FH10,255.667
T0034,FH03,FH12,261.667
T0034,FH10,FH01,261.667
T0034,FH10,FH03,255.667
T0034,FH10,FH10,261.533
T0034,FH10,FH12,259.733
T0034,FH12,FH01,264.733
T0034,FH12,FH03,261.667
T0034,FH12,FH10,259.733
T0034,FH12,FH12,265.733
T0035,FH01,FH01,266.933
T0035,FH01,FH03,260.933
T0035,FH01,FH10,261.667
T0035,FH01,FH12,264.733
T0035,FH03,FH01,260.933
T0035,FH03,FH03,266.933
T0035,FH03,FH10,255.667
T0035,FH03,FH12,261.667
T0035,FH10,FH01,261.667
T0035,FH10,FH03,255.667
T0035,FH10,FH10,261.533
T0035,FH10,FH12,259.733
T0035,FH12,FH01,264.733
T0035,FH12,FH03,261.667
T0035,FH12,FH10,259.733
T0035,FH12,FH12,265.733
T0036,FH01,FH01,266.933
T0036,FH01,FH03,260.933
T0036,FH01,FH10,261.667
T0036,FH01,FH12,264.733
T0036,FH03,FH01,260.933
T0036,FH03,FH03,266.933
T0036,FH03,FH10,255.667
T0036,FH03,FH12,261.667
T0036,FH10,FH01,261.667
T0036,FH10,FH03,255.667
T0036,FH10,FH10,261.533
T0036,FH10,FH12,259.733
T0036,FH12,FH01,264.733
T0036,FH12,FH03,261.667
T0036,FH12,FH10,259.733
T0036,FH12,FH12,265.733
T0037,FH01,FH01,266.933
T0037,FH01,FH03,260.933
T0037,FH01,FH10,261.667
T0037,FH01,FH12,264.733
T0037,FH03,FH01,260.933
T0037,FH03,FH03,266.933
T0037,FH03,FH10,255.667
T0037,FH03,FH12,261.667
T0037,FH10,FH01,261.667
T0037,FH10,FH03,255.667
T0037,FH10,FH10,261.533
T0037,FH10,FH12,259.733
T0037,FH12,FH01,264.733
T0037,FH12,FH03,261.667
T0037,FH12,FH10,259.733
T0037,FH12,FH12,265.733
T0038,FH01,FH01,266.933
T0038,FH01,FH03,260.933
T0038,FH01,FH10,261.667
T0038,FH01,FH12,264.733
T0038,FH03,FH01,260.933
T0038,FH03,FH03,266.933
T0038,FH03,FH10,255.667
T0038,FH03,FH12,261.667
T0038,FH10,FH01,261.667
T0038,FH10,FH03,255.667
T0038,FH10,FH10,261.533
T0038,FH10,FH12,259.733
T0038,FH12,FH01,264.733
T0038,FH12,FH03,261.667
T0038,FH12,FH10,259.733
T0038,FH12,FH12,265.733
T0039,FH01,FH01,266.933
T0039,FH01,FH03,260.933
T0039,FH01,FH10,261.667
T0039,FH01,FH12,264.733
T0039,FH03,FH01,260.933
T0039,FH03,FH03,266.933
T0039,FH03,FH10,255.667
T0039,FH03,FH12,261.667
T0039,FH10,FH01,261.667
T0039,FH10,FH03,255.667
T0039,FH10,FH10,261.533
T0039,FH10,FH12,259.733
T0039,FH12,FH01,264.733
T0039,FH12,FH03,261.667
T0039,FH12,FH10,259.733
T0039,FH12,FH12,265.733
T0040,FH01,FH01,266.933
T0040,FH01,FH03,260.933
T0040,FH01,FH10,261.667
T0040,FH01,FH12,264.733
T0040,FH03,FH01,260.933
T0040,FH03,FH03,266.933
T0040,FH03,FH10,255.667
T0040,FH03,FH12,261.667
T0040,FH10,FH01,261.667
T0040,FH10,FH03,255.667
T0040,FH10,FH10,261.533
T0040,FH10,FH12,259.733
T0040,FH12,FH01,264.733
T0040,FH12,FH03,261.667
T0040,FH12,FH10,259.733
T0040,FH12,FH12,265.733
T0041,FH01,FH01,266.933
T0041,FH01,FH03,260.933
T0041,FH01,FH10,261.667
T0041,FH01,FH12,264.733
T0041,FH03,FH01,260.933
T0041,FH03,FH03,266.933
T0041,FH03,FH10,255.667
T0041,FH03,FH12,261.667
T0041,FH10,FH01,261.667
T0041,FH10,FH03,255.667
T0041,FH10,FH10,261.533
T0041,FH10,FH12,259.733
T0041,FH12,FH01,264.733
T0041,FH12,FH03,261.667
T0041,FH12,FH10,259.733
T0041,FH12,FH12,265.733
T0042,FH01,FH01,266.933
T0042,FH01,FH03,260.933
T0042,FH01,FH10,261.667
T0042,FH01,FH12,264.733
T0042,FH03,FH01,260.933
T0042,FH03,FH03,266.933
T0042,FH03,FH10,255.667
T0042,FH03,FH12,261.667
T0042,FH10,FH01,261.667
T0042,FH10,FH03,255.667
T0042,FH10,FH10,261.533

T0042,FH10,FH12,259.733
T0042,FH12,FH01,264.733
T0042,FH12,FH03,261.667
T0042,FH12,FH10,259.733
T0042,FH12,FH12,265.733
T0043,FH01,FH01,266.933
T0043,FH01,FH03,260.933
T0043,FH01,FH10,261.667
T0043,FH01,FH12,264.733
T0043,FH03,FH01,260.933
T0043,FH03,FH03,266.933
T0043,FH03,FH10,255.667
T0043,FH03,FH12,261.667
T0043,FH10,FH01,261.667
T0043,FH10,FH03,255.667
T0043,FH10,FH10,261.533
T0043,FH10,FH12,259.733
T0043,FH12,FH01,264.733
T0043,FH12,FH03,261.667
T0043,FH12,FH10,259.733
T0043,FH12,FH12,265.733
T0044,FH01,FH01,266.933
T0044,FH01,FH03,260.933
T0044,FH01,FH10,261.667
T0044,FH01,FH12,264.733
T0044,FH03,FH01,260.933
T0044,FH03,FH03,266.933
T0044,FH03,FH10,255.667
T0044,FH03,FH12,261.667
T0044,FH10,FH01,261.667
T0044,FH10,FH03,255.667
T0044,FH10,FH10,261.533
T0044,FH10,FH12,259.733
T0044,FH12,FH01,264.733
T0044,FH12,FH03,261.667
T0044,FH12,FH10,259.733
T0044,FH12,FH12,265.733
T0045,FH01,FH01,266.933
T0045,FH01,FH03,260.933
T0045,FH01,FH10,261.667
T0045,FH01,FH12,264.733
T0045,FH03,FH01,260.933
T0045,FH03,FH03,266.933
T0045,FH03,FH10,255.667
T0045,FH03,FH12,261.667
T0045,FH10,FH01,261.667
T0045,FH10,FH03,255.667
T0045,FH10,FH10,261.533
T0045,FH10,FH12,259.733
T0045,FH12,FH01,264.733
T0045,FH12,FH03,261.667
T0045,FH12,FH10,259.733
T0045,FH12,FH12,265.733
T0046,FH01,FH01,266.933
T0046,FH01,FH03,260.933
T0046,FH01,FH10,261.667
T0046,FH01,FH12,264.733
T0046,FH03,FH01,260.933
T0046,FH03,FH03,266.933
T0046,FH03,FH10,255.667
T0046,FH03,FH12,261.667
T0046,FH10,FH01,261.667
T0046,FH10,FH03,255.667
T0046,FH10,FH10,261.533
T0046,FH10,FH12,259.733
T0046,FH12,FH01,264.733
T0046,FH12,FH03,261.667
T0046,FH12,FH10,259.733
T0046,FH12,FH12,265.733
T0047,FH01,FH01,266.933
T0047,FH01,FH03,260.933
T0047,FH01,FH10,261.667
T0047,FH01,FH12,264.733
T0047,FH03,FH01,260.933
T0047,FH03,FH03,266.933
T0047,FH03,FH10,255.667
T0047,FH03,FH12,261.667
T0047,FH10,FH01,261.667
T0047,FH10,FH03,255.667
T0047,FH10,FH10,261.533
T0047,FH10,FH12,259.733
T0047,FH12,FH01,264.733
T0047,FH12,FH03,261.667
T0047,FH12,FH10,259.733
T0047,FH12,FH12,265.733
T0048,FH01,FH01,266.933
T0048,FH01,FH03,260.933
T0048,FH01,FH10,261.667
T0048,FH01,FH12,264.733
T0048,FH03,FH01,260.933
T0048,FH03,FH03,266.933
T0048,FH03,FH10,255.667
T0048,FH03,FH12,261.667
T0048,FH10,FH01,261.667
T0048,FH10,FH03,255.667
T0048,FH10,FH10,261.533
T0048,FH10,FH12,259.733
T0048,FH12,FH01,264.733
T0048,FH12,FH03,261.667
T0048,FH12,FH10,259.733
T0048,FH12,FH12,265.733
T0049,FH01,FH01,266.933
T0049,FH01,FH03,260.933
T0049,FH01,FH10,261.667
T0049,FH01,FH12,264.733
T0049,FH03,FH01,260.933
T0049,FH03,FH03,266.933
T0049,FH03,FH10,255.667
T0049,FH03,FH12,261.667
T0049,FH10,FH01,261.667
T0049,FH10,FH03,255.667
T0049,FH10,FH10,261.533
T0049,FH10,FH12,259.733
T0049,FH12,FH01,264.733
T0049,FH12,FH03,261.667
T0049,FH12,FH10,259.733
T0049,FH12,FH12,265.733

问题四各任务所需取货和复核时间

任务单,取货时间,复核时间

T0001,172.0,270.0
T0002,182.0,450.0
T0003,171.0,330.0
T0004,186.0,390.0
T0005,165.0,390.0
T0006,168.0,390.0
T0007,182.0,420.0
T0008,185.0,540.0
T0009,171.0,330.0
T0010,179.0,360.0
T0011,174.0,480.0
T0012,170.0,420.0
T0013,182.0,390.0
T0014,183.0,450.0
T0015,176.0,360.0
T0016,194.0,540.0
T0017,179.0,480.0
T0018,180.0,390.0
T0019,167.0,360.0
T0020,186.0,450.0
T0021,174.0,390.0
T0022,178.0,420.0
T0023,179.0,480.0
T0024,178.0,450.0
T0025,188.0,480.0
T0026,184.0,570.0
T0027,172.0,390.0
T0028,189.0,480.0
T0029,174.0,450.0
T0030,175.0,330.0
T0031,185.0,450.0
T0032,177.0,330.0
T0033,176.0,240.0
T0034,186.0,390.0
T0035,189.0,540.0
T0036,173.0,480.0
T0037,165.0,330.0
T0038,183.0,540.0
T0039,173.0,420.0
T0040,178.0,510.0
T0041,183.0,450.0
T0042,165.0,450.0
T0043,171.0,420.0
T0044,181.0,450.0
T0045,183.0,390.0
T0046,180.0,360.0
T0047,182.0,420.0
T0048,185.0,420.0
T0049,153.0,300.0

问题 4 任务安排和货物使用规划

人员编号,任务编号,起始复核台,抵达复核台,货物访问顺序

P4,T0049,3000,3000,0 14 15 9 6 13 19 8 17 2 12 10 20 1 7 4 16 3 18 5 11 22 21

P8,T0005,3009,3000,0 2 4 17 1 12 10 21 9 11 7 18 3 15 14 16 5 20 8 6 13 22 23 19

P9,T0001,3009,3011,0 5 16 13 4 12 17 3 1 6 22 2 9 11 8 10 21 15 18 7 19 14 23 24 20

P2,T0023,3011,3002,0 9 14 20 23 22 7 21 4 18 5 8 13 12 2 10 11 3 6 19 16 17 15 1

P7,T0029,3000,3000,0 24 5 10 25 23 3 11 6 7 13 9 1 16 15 12 27 28 19 8 2 22 26 20 18 14 21 4 17

P5,T0004,3009,3002,0 25 2 18 21 9 12 17 26 27 10 5 6 14 8 19 20 4 13 23 16 22 7 11 1 3 15 24

P1,T0038,3009,3011,0 26 8 24 11 19 13 22 25 21 20 9 14 18 23 1 27 28 6 16 10 12 4 5 7 2 15 3 17

P6,T0016,3002,3009,0 18 4 13 25 12 2 28 1 5 8 20 3 16 17 27 30 29 19 14 23 24 10 21 6 26 15 22 7 9 11

P3,T0020,3011,3000,0 21 5 14 17 8 20 3 15 12 18 24 2 16 13 9 23 4 7 25 19 11 6 22 10 1 27 28 26

P4,T0032,3002,3000,0 7 12 5 25 24 2 14 9 23 20 21 17 16 4 22 10 15 8 6 1 18 3 13 11 19

P9,T0026,3000,3009,0 6 13 14 11 21 2 28 4 9 3 12 27 1 5 16 22 7 20 15 23 19 18 25 17 26 29 30 10 24 8

P6,T0018,3000,3002,0 15 22 18 17 21 7 12 2 3 8 1 5 19 16 14 24 23 9 10 6 13 4 11 20

P2,T0048,3011,3000,0 20 22 3 13 12 27 28 24 5 26 10 23 11 9 14 2 15 25 6 1 4 18 8 19 21 7 17 16

P1,T0042,3011,3011,0 18 20 15 16 19 2 11 6 5 14 22 23 4 3 21 9 1 7 8 12 10 17 13

P8,T0008,3011,3002,0 4 18 10 15 3 22 8 12 7 14 9 21 19 2 17 20 24 1 5 11 25 6 27 28 13 16 26 23

P5,T0040,3009,3009,0 22 5 25 15 17 9 1 11 10 21 23 28 27 3 7 14 8 4 16 24 18 12 19 26 2 13 20 6

P9,T0036,3000,3000,0 2 20 22 7 15 10 19 16 1 13 12 14 3 18 9 5 17 21 23 11 4 8 6 25 24

P7,T0046,3002,3011,0 1 9 5 11 21 22 15 16 17 2 18 7 10 20 19 13 8 6 12 3 4 14

P1,T0017,3011,3011,0 24 9 12 10 7 26 13 19 4 16 5 8 6 22 29 30 18 1 17 23 15 25 14 20 2 11 21 3 27 28

P6,T0034,3009,3011,0 24 25 10 21 13 1 8 6 23 14 9 3 4 11 16 2 15 20 17 12 22 7 18 19 5

P5,T0039,3002,3009,0 15 11 23 18 1 24 3 20 8 9 7 5 16 10 17 2 14 21 22 6 25 26 13 12 4 19

P3,T0035,3009,3011,0 20 2 13 25 9 29 10 12 26 32 31 4 21 3 18 6 24 5 22 28 14 1 19 16 30 8 7 23 17 15 11 27

P7,T0014,3002,3011,0 26 27 1 13 9 10 4 19 8 2 7 24 17 12 20 18 23 21 5 11 16 14 25 3 22 15 6

P8,T0015,3011,3002,0 14 16 20 10 7 1 9 13 17 12 18 4 6 23 22 19 15 21 3 5 2 11 8

P4,T0009,3000,3002,0 17 4 16 22 21 12 19 8 1 18 14 15 2 20 10 3 9 13 11 6 5 7

P1,T0021,3000,3009,0 23 24 2 15 16 8 4 1 22 12 19 6 7 13 21 11 10 18 20 17 5 14 3 9

P5,T0043,3002,3009,0 19 7 9 20 21 4 5 8 18 15 2 22 11 23 24 12 1 6 17 3 13 14 16 10

P8,T0033,3009,3002,0 3 17 9 15 20 21 7 19 10 6 4 18 8 13 5 14 12 2 1 11 16

P2,T0030,3000,3000,0 15 14 5 9 4 23 22 21 1 8 19 3 11 17 7 20 10 6 18 2 13 16 12

P1,T0041,3000,3000,0 3 10 8 19 18 1 22 23 6 13 4 7 15 20 17 9 5 11 2 21 16 12 14

P6,T0027,3011,3009,0 9 24 23 16 10 6 8 1 17 5 18 22 4 14 13 12 20 3 19 15 2 7 21 11

P4,T0037,3000,3011,0 4 2 13 17 7 20 21 1 6 5 11 18 12 3 19 23 22 10 9 16 14 15 8

P9,T0002,3009,3002,0 2 3 19 5 13 1 21 12 17 11 15 20 22 26 27 14 7 4 6 8 16 24 25 23 18 9 10

P5,T0045,3009,3002,0 16 5 8 3 14 23 18 4 2 24 7 21 6 19 26 25 11 10 9 1 20 17 12 22 13 15

P8,T0011,3000,3009,0 17 24 25 10 27 28 3 13 23 9 8 16 5 19 26 4 11 18 2 6 12 7 15 20 22 21 1 14

P3,T0022,3009,3000,0 7 2 11 17 13 3 14 10 25 20 23 9 27 26 12 1 18 6 22 15 19 5 21 16 4 8 24

P9,T0012,3002,3009,0 9 3 12 4 21 20 16 19 2 23 10 7 6 1 14 8 18 17 24 25 15 11 5 13 22

P2,T0003,3011,3000,0 19 10 5 17 21 20 9 1 14 15 4 13 12 8 6 7 16 2 3 11 18

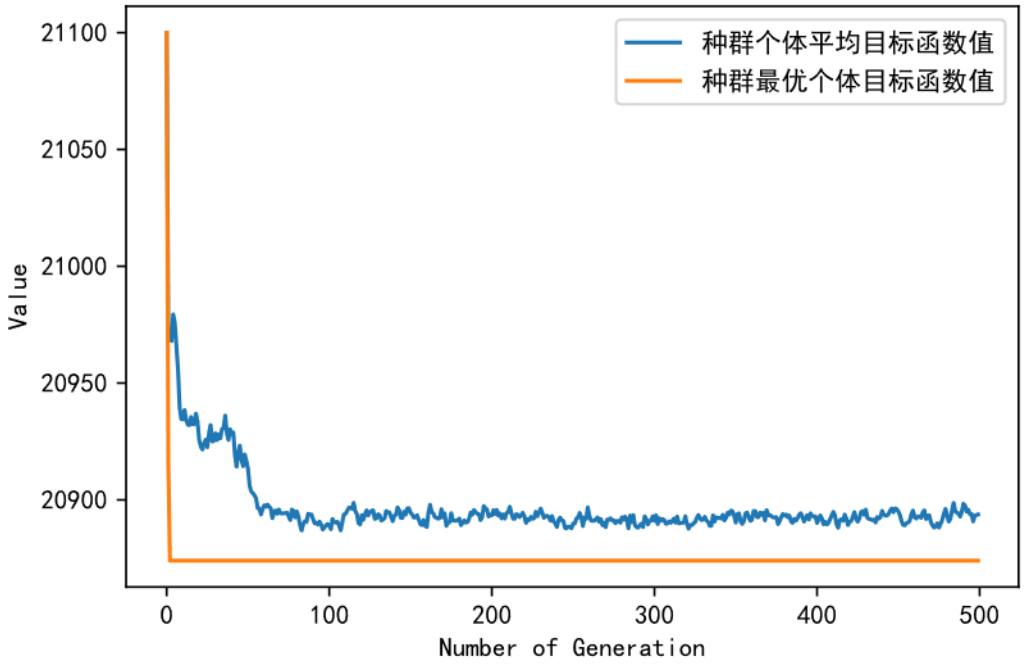
P6,T0006,3009,3002,0 7 13 19 5 10 18 9 2 22 3 12 14 11 8 21 24 17 16 4 1 6 25 26 15 23 20

P7,T0010,3009,3011,0 23 24 15 10 22 14 11 8 12 20 13 18 3 7 1 19 9 6 4 5 16 21 17 2

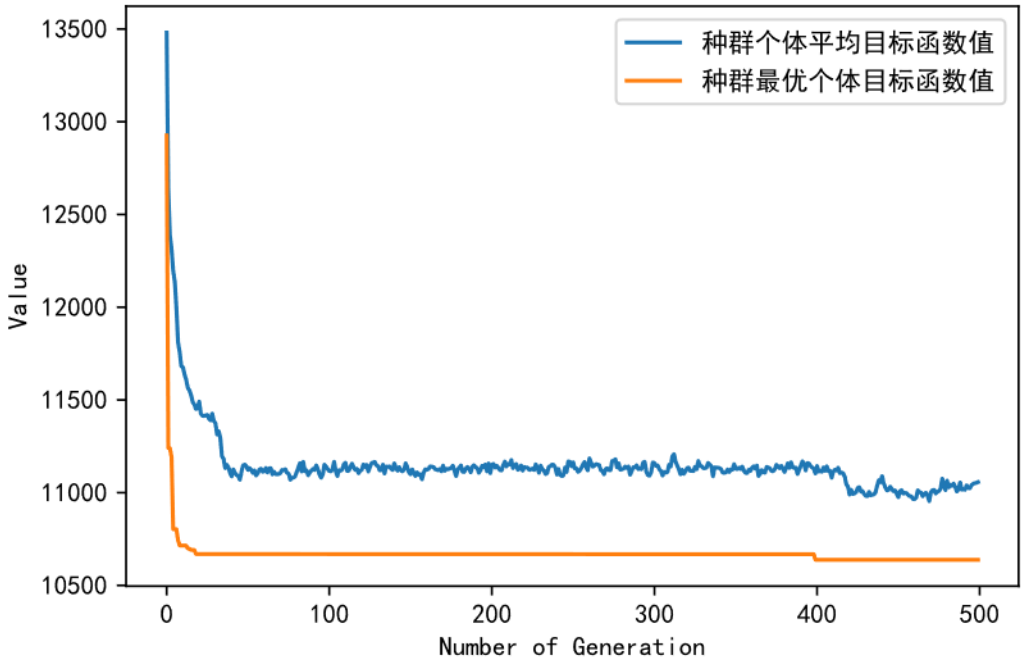
P1,T0013,3000,3000,0 24 23 8 20 11 22 15 19 10 12 5 7 14 13 4 16 1 18 6 2 17 9 21 3
P5,T0031,3002,3011,0 21 22 2 10 7 6 18 12 17 19 15 14 4 20 8 24 25 5 3 9 23 11 16 1 13
P4,T0047,3002,3011,0 17 15 25 1 22 10 8 20 19 5 13 24 6 23 12 2 4 14 18 11 27 26 21 7 9 3 16
P3,T0024,3011,3002,0 1 23 18 10 24 15 12 8 6 7 4 14 5 21 26 25 11 13 16 2 9 22 3 19 17 20
P6,T0025,3011,3000,0 18 17 20 9 4 10 23 16 7 6 14 11 25 26 27 19 13 21 5 3 22 24 15 8 1 2 12
P9,T0044,3002,3009,0 10 13 25 26 19 12 11 3 17 14 21 5 16 9 24 4 15 7 22 8 20 6 23 1 2 18
P2,T0019,3009,3002,0 4 23 22 3 18 6 12 10 2 8 5 19 1 9 21 16 20 15 14 7 17 13 11
P3,T0028,3000,3002,0 15 23 17 10 13 9 2 11 25 26 27 22 4 8 6 3 28 29 1 19 21 5 20 7 16 24 14 12 18
P5,T0007,3011,3009,0 19 5 15 4 17 1 6 23 20 9 7 12 2 13 21 22 8 10 11 16 14 18 3 24 25

问题 5 的不同复核台个数，进化代数和种群最优值和平均目标值的关系

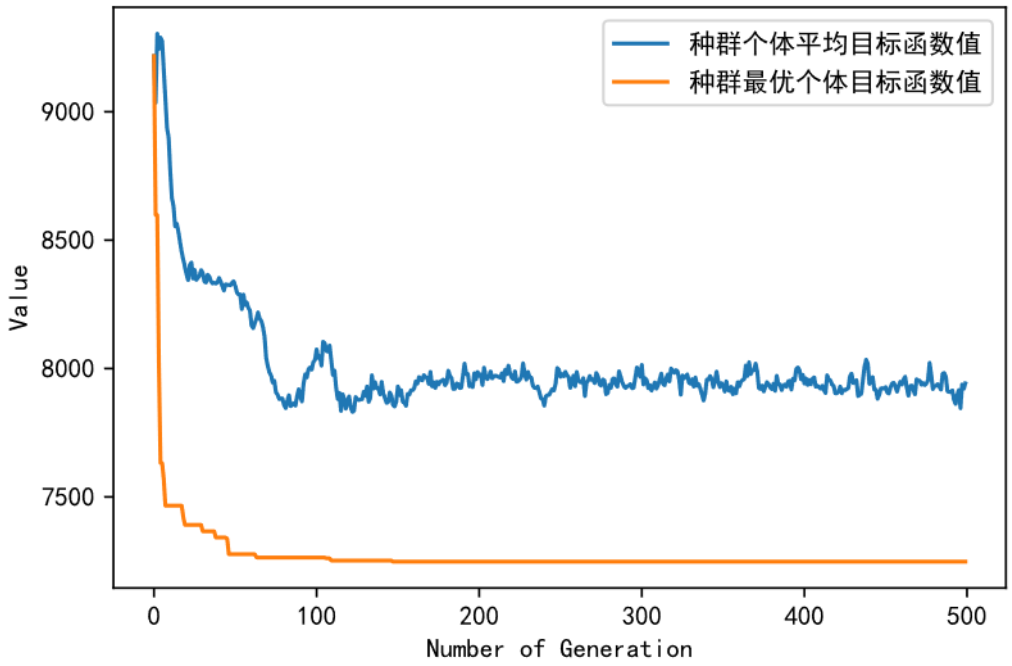
一个复核台:



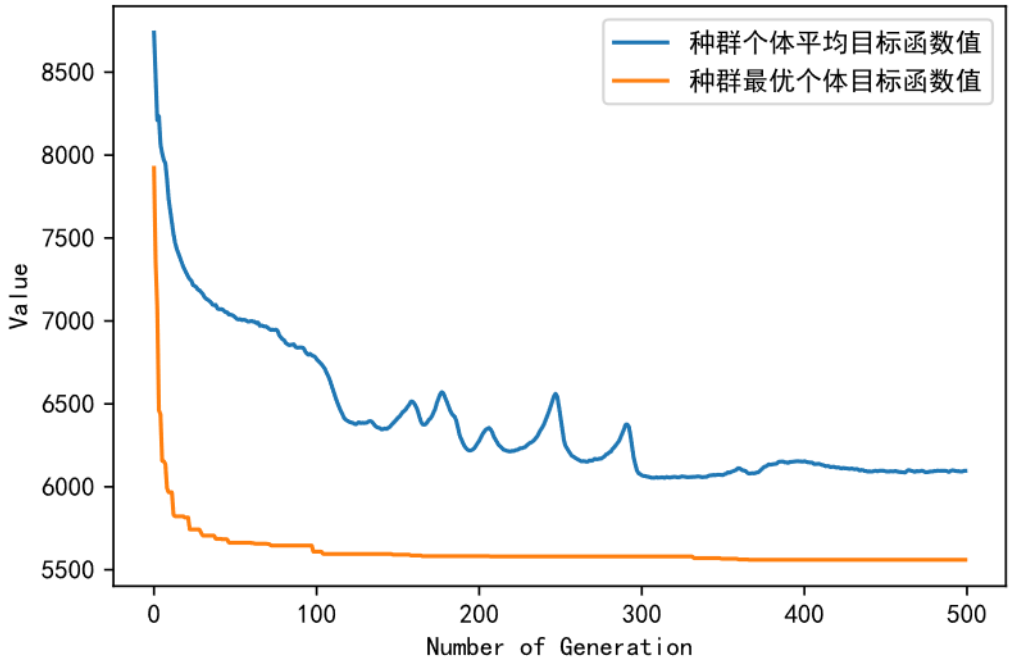
两个复核台:


种群个体平均目标函数值
种群最优个体目标函数值
13500
13000
12500
12000
11500
11000
10500
Value
0
100
200
300
400
500
Number of Generation

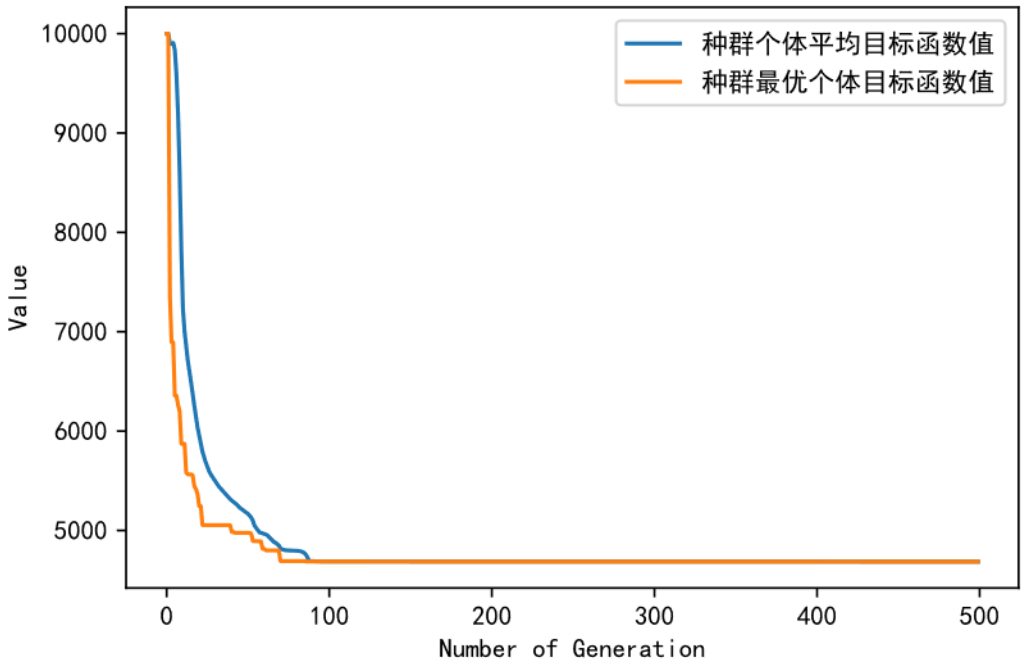
三个复核台:


种群个体平均目标函数值
种群最优个体目标函数值
9000
8500
8000
7500
Value
0
100
200
300
400
500
Number of Generation

四个复核台：


种群个体平均目标函数值
种群最优个体目标函数值
Value
Number of Generation
8500
8000
7500
7000
6500
6000
5500
0
100
200
300
400
500

五个复核台:


种群个体平均目标函数值
种群最优个体目标函数值
Value
Number of Generation
10000
9000
8000
7000
6000
5000
0
100
200
300
400
500

代码部分

CalculateTime.java

```
package sxjm;

import java.io.*;
import java.util.ArrayList;
import java.util.HashSet;

public class CalculateTime {

    private static ArrayList<TestList> test = new ArrayList<>();

    public static void main(String[] args) throws IOException {
        String path = "task6_3.csv";
       // CalculateTime ct = new CalculateTime(path, "T0001");
        for (int i = 75; i < 87; i++) {
            CalculateTime ct = new CalculateTime(path, "T00" + i);
            test.clear();
        }

    }
    private CalculateTime(String path, String orderListNumber) throws IOException
{
        File file = new File(path);
        BufferedReader br = new BufferedReader(new FileReader(file));
        String s;
        while ((s = br.readLine()) != null) {
            String[] as = s.split(",");//分割方式，这个看看要不要改
            String testNumber = as[0];
            String orderNumber = as[1];
            int piece = Integer.parseInt(as[3]);
            TestList test1 = new TestList(testNumber, orderNumber, piece);
            test.add(test1);
        }
        br.close();
        System.out.println(orderListNumber + "," + timeToClaimGoodsCalculate(test,
orderListNumber) + "," +  reCheckGoodsCalculate(test, orderListNumber));
        /*System.out.println("单号[" + orderListNumber + "]的取货时间为：" +
timeToClaimGoodsCalculate(test, orderListNumber) + "s");
        System.out.println("单号[" + orderListNumber + "]的复核时间为：" +
reCheckGoodsCalculate(test, orderListNumber) + "s");*/
    }

    private static double timeToClaimGoodsCalculate(ArrayList<TestList> testLists,
```

```
String orderListNumber) {

        ArrayList<TestList> orderList = new ArrayList<>();//包括了所有指定任务单号
上的任务
        for (TestList list : testLists) {
            if (list.getTestNumber().equals(orderListNumber)) {
                orderList.add(list);
            }
        }

        //取货时间
        double timeToClaimGoods = 0.0;
        for (TestList testList : orderList) {
            if (testList.getPiece() < 3) {
                timeToClaimGoods += testList.getPiece() * 5;
                continue;
            }
            if (testList.getPiece() >= 3) {
                timeToClaimGoods += testList.getPiece() * 4;
            }
        }

        return timeToClaimGoods;
    }

    private static double reCheckGoodsCalculate(ArrayList<TestList> testLists,
String orderListNumber) {

        HashSet<String> hashSet = new HashSet<>();
        for (TestList testList : testLists) {
            if (testList.getTestNumber().equals(orderListNumber))
hashSet.add(testList.getOrderNumber());
        }

        return hashSet.size() * 30;
    }
}
```

CalculateDistance.java

```
package sxjm;

import au.com.bytecode.opencsv.CSVReader;

import java.io.*;
```

```
public class CalculateDistance {



    public static void main(String[] args) throws IOException {
        int[][] yBorder = new int[4][2];
        yBorder[0] = new int[]{ 2250, 14750 };
        for (int i = 1; i < 4; i++) {
            yBorder[i][0] = yBorder[i - 1][0] + 14000;
            yBorder[i][1] = yBorder[i - 1][1] + 14000;
        }

        String goodsPath = "E:\\FourthDesk\\Projects\\adaa\\src\\sxjm\\new.csv";
        String reviewerPath =
"E:\\FourthDesk\\Projects\\adaa\\src\\sxjm\\new2.csv";
        String reviewerOriginPath = "E:\\FourthDesk\\Projects\\adaa\\src\\sxjm\\
复核台.csv";

        CSVReader csvReader = new CSVReader(new FileReader(new File(goodsPath)));
        csvReader.readNext(); // header()
        int n = 3000;
        int[][] goods = new int[n][4];
        String[] next;
        for (int i = 0; i < n; i++) {
            next = csvReader.readNext();
            for (int j = 0; j < 4; j++) goods[i][j] = Integer.parseInt(next[j + 1]);
        }
        int[][] matrix = new int[n + 13][n + 13];
        int dis;
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < i; j++) {
                if (goods[i][3] == goods[j][3] && goods[i][2] != goods[j][2]) {
                    int row = goods[i][3];
                    dis = Math.abs(goods[i][0] - goods[j][0]) +
                            Math.min(yBorder[row][1] * 2 - goods[i][1] - goods[j][1],
goods[i][1] + goods[j][1] - 2 * yBorder[row][0])
                            + 1500;
                    if (dis < 0) {
                        System.out.println(i + "  " + j);
                        System.out.println(row);
                        System.out.println(goods[i][0] + "  " + goods[i][1]);
                        System.out.println(goods[j][0] + "  " + goods[j][1]);
                        System.out.println(yBorder[row][1] * 2);
                        System.out.println(Math.min(yBorder[row][1] * 2 -
goods[i][1] - goods[j][1], goods[i][1] + goods[j][1] - 2 * yBorder[row][0]));
```

```
                    return;
                }
            } else {
                dis = Math.abs(goods[i][0] - goods[j][0]) + Math.abs(goods[i][1]
- goods[j][1]) + 1500;
            }
            matrix[i][j] = dis;
            matrix[j][i] = dis;
        }
    }

    csvReader = new CSVReader(new FileReader(new File(reviewerPath)));
    csvReader.readNext(); // header()
    int[][] reviewer = new int[13][3];
    for (int i = 0; i < 13; i++) {
        next = csvReader.readNext();
        for (int j = 0; j < 3; j++) reviewer[i][j] = Integer.parseInt(next[j
+ 1]);
    }
    for (int i = 0; i < 8; i++) {
        for (int j = 0; j < n; j++) {
            dis = Math.abs(reviewer[i][0] - goods[j][0]) +
Math.abs(reviewer[i][1] - goods[j][1]) + 1500;
            matrix[i + n][j] = dis;
            matrix[j][i + n] = dis;
        }
    }
    for (int i = 8; i < 13; i++) {
        for (int j = 0; j < n; j++) {
            if (reviewer[i][2] == goods[j][3] && goods[i][2] != 0) {
                int row = goods[i][3];
                dis = Math.abs(reviewer[i][0] - goods[j][0]) +
                        Math.min(yBorder[row][1] * 2 - reviewer[i][1] -
goods[j][1], reviewer[i][1] + goods[j][1] - 2 * yBorder[row][0])
                        + 1500;
            } else {
                dis = Math.abs(reviewer[i][0] - goods[j][0]) +
Math.abs(reviewer[i][1] - goods[j][1]) + 1500;
            }
            matrix[i + n][j] = dis;
            matrix[j][i + n] = dis;
        }
    }

    csvReader = new CSVReader(new FileReader(new File(reviewerOriginPath)));
    csvReader.readNext(); // header()
```

```
        reviewer = new int[13][3];
        for (int i = 0; i < 13; i++) {
            next = csvReader.readNext();
            for (int j = 0; j < 2; j++) reviewer[i][j] = Integer.parseInt(next[j
+ 1]);
        }

        for (int i = 0; i < 13; i++) {
            for (int j = 0; j < 13; j++) {
                matrix[i + n][j + n] = Math.abs(reviewer[i][0] - reviewer[j][0]) +
Math.abs(reviewer[i][1] - reviewer[j][1]);
            }
        }

        String fileName = "out.csv";
        FileWriter writer = new FileWriter(fileName);
        for (int i = 0; i < 3013; i++) {
            StringBuilder sb = new StringBuilder();
            sb.append(matrix[i][0]);
            for (int j = 1; j < 3013; j++) {
                sb.append(",");
                sb.append(matrix[i][j]);
            }
            sb.append("\n");
            writer.write(sb.toString());
        }
        writer.close();
    }
}
```

Task2.ipynb

```
def AntColony(dis_mat):
    num_ant=200
    num_city=25
    alpha=1
    beta=1
    info=0.1
    Q=1
    count_iter = 0
    iter_max = 200
    #期望矩阵
    e_mat_init=1.0/(dis_mat+np.diag([10000]*num_city))#加对角阵是因为除数不能是 0
    diag=np.diag([1.0/10000]*num_city)
    e_mat=e_mat_init-diag#还是把对角元素变成 0
```

```
    #初始化每条边的信息素浓度，全 1 矩阵
    pheromone_mat=np.ones((num_city,num_city))
    #初始化每只蚂蚁路径，都从 0 城市出发
    path_mat=np.zeros((num_ant,num_city)).astype(int)

    #while dis_new>400:
    while count_iter < iter_max:
        for ant in range(num_ant):
            visit=0#都从 0 城市出发
            unvisit_list=list(range(1,num_city))
            for j in range(1,num_city):
            #轮盘法选择下一个城市
                trans_list=[]
                tran_sum=0
                trans=0
                for k in range(len(unvisit_list)):
                    trans
+=np.power(pheromone_mat[visit][unvisit_list[k]],alpha)*np.power(e_mat[visit][unvisit_list[k]],
beta)
                    trans_list.append(trans)
                    tran_sum =trans

                rand=random.uniform(0,tran_sum)#产生随机数

                for t in range(len(trans_list)):
                    if(rand <= trans_list[t]):
                        visit_next=unvisit_list[t]
                        break
                    else:
                        continue
                path_mat[ant,j]=visit_next#填路径矩阵
                unvisit_list.remove(visit_next)#更新
                visit=visit_next#更新

        #所有蚂蚁的路径表填满之后，算每只蚂蚁的总距离
        dis_allant_list=cal_newpath(dis_mat,path_mat)

        #每次迭代更新最短距离和最短路径
        if count_iter == 0:
            dis_new=min(dis_allant_list)
            path_new=path_mat[dis_allant_list.index(dis_new)].copy()
        else:
            if min(dis_allant_list) < dis_new:
                dis_new=min(dis_allant_list)
                path_new=path_mat[dis_allant_list.index(dis_new)].copy()
```

```
    # 更新信息素矩阵
        pheromone_change=np.zeros((num_city,num_city))
        for i in range(num_ant):
            for j in range(num_city-1):
                pheromone_change[path_mat[i,j]][path_mat[i,j+1]] +=
Q/dis_mat[path_mat[i,j]][path_mat[i,j+1]]
            pheromone_change[path_mat[i,num_city-1]][path_mat[i,0]] +=
Q/dis_mat[path_mat[i,num_city-1]][path_mat[i,0]]
        pheromone_mat=(1-info)*pheromone_mat+pheromone_change
        count_iter += 1 #迭代计数+1，进入下一次

    print('最短距离：',dis_new)
     print('最短路径：',path_new)
```

```
    num_ant=100 #蚂蚁个数
    num_city=25 #城市个数
    alpha=1 #信息素影响因子
    beta=1   #期望影响因子
    info=0.1 #信息素的挥发率
    Q=1 #常数

    count_iter = 0
    iter_max = 400
    #dis_new=1000
    #=========================================
    #对称矩阵，两个城市之间的距离

    #计算所有路径对应的距离

    #=========================================
    #点对点距离矩阵
    #期望矩阵
    e_mat_init=1.0/(dis_mat+np.diag([10000]*num_city))#加对角阵是因为除数不能是 0
    diag=np.diag([1.0/10000]*num_city)
    e_mat=e_mat_init-diag#还是把对角元素变成 0
    #初始化每条边的信息素浓度，全 1 矩阵
```

```
    pheromone_mat=np.ones((num_city,num_city))
    #初始化每只蚂蚁路径，都从 0 城市出发
    path_mat=np.zeros((num_ant,num_city)).astype(int)


    #while dis_new>400:
    while count_iter < iter_max:
        for ant in range(num_ant):
            visit=0#都从 0 城市出发
            unvisit_list=list(range(1,25))#未访问的城市
            for j in range(1,num_city):
                #轮盘法选择下一个城市
                trans_list=[]
                tran_sum=0
                trans=0
                for k in range(len(unvisit_list)):
                    trans
+=np.power(pheromone_mat[visit][unvisit_list[k]],alpha)*np.power(e_mat[visit][unvisit_list[k]],
beta)
                    trans_list.append(trans)
                    tran_sum =trans

                rand=random.uniform(0,tran_sum)#产生随机数

                for t in range(len(trans_list)):
                    if(rand <= trans_list[t]):
                        visit_next=unvisit_list[t]



                        break
                    else:
                        continue
                path_mat[ant,j]=visit_next#填路径矩阵

                unvisit_list.remove(visit_next)#更新
                visit=visit_next#更新

        #所有蚂蚁的路径表填满之后，算每只蚂蚁的总距离
        dis_allant_list=cal_newpath(dis_mat,path_mat)

        #每次迭代更新最短距离和最短路径
        if count_iter == 0:
            dis_new=min(dis_allant_list)
            path_new=path_mat[dis_allant_list.index(dis_new)].copy()
        else:
            if min(dis_allant_list) < dis_new:
```

```
                dis_new=min(dis_allant_list)
                path_new=path_mat[dis_allant_list.index(dis_new)].copy()

        # 更新信息素矩阵
        pheromone_change=np.zeros((num_city,num_city))
        for i in range(num_ant):
            for j in range(num_city-1):
                pheromone_change[path_mat[i,j]][path_mat[i,j+1]] +=
Q/dis_mat[path_mat[i,j]][path_mat[i,j+1]]
            pheromone_change[path_mat[i,num_city-1]][path_mat[i,0]] +=
Q/dis_mat[path_mat[i,num_city-1]][path_mat[i,0]]
        pheromone_mat=(1-info)*pheromone_mat+pheromone_change
        count_iter += 1 #迭代计数+1，进入下一次

    print('最短距离：',dis_new)
    print('最短路径：',path_new)
```

Task4.ipynb

```
    def calculate(a, travelTime, Time): #, traveltime, takeTime, reviewTime
        for i in range(len(a)):
            a[i] = int(a[i])
        # print(a)
        all_a = a[:49]
        nextTo = a[49:98]
        divide = a[98:107]
        oringin = a[107:]
        if(sum(divide)!=49):
            return 10000
        c = 0
        plan = []
        for i in range(0,9):
            plan.append(all_a[c: c+divide[i]])
            c += divide[i]
        # print(plan)
        people = [] #time, pid, cnt,
        for i in range(0,9):
            id_number = plan[i][0]
            heapq.heappush(people,(Time[id_number][0] + travelTime[(id_number, oringin[i],
nextTo[id_number-1])], i, 0, nextTo[id_number-1]))
        # print(people)
        reviewer = [0,0,0,0]
        while(len(people)>0):
            time1,pid,cnt,rid = heapq.heappop(people)
            if (reviewer[rid-1] <= time1):
                time2 = time1
                reviewer[rid-1] = time1 + Time[plan[pid][cnt]][1]
```

```
                else:
                    time2 = reviewer[rid-1]
                    reviewer[rid-1] += Time[plan[pid][cnt]][1]
                cnt+=1
                if (cnt < len(plan[pid])):
                    id_number = plan[pid][cnt]
                    heapq.heappush(people,(time2+ Time[id_number][0] +
travelTime[(id_number, rid, nextTo[id_number-1])], pid, cnt, nextTo[id_number-1]))
        return max(reviewer)

    travelTime = dict()
    task_time = pd.read_csv("task_time.csv")
    task_travel_time = pd.read_csv("data4plus.csv")
    review = {'FH01':1, 'FH03':2, 'FH10':3, 'FH12':4}
    task_time["任务单"]=task_time[["任务单"]].apply(lambda x: int(x["任务单"][1:]),axis=1)
    Time = task_time.set_index('任务单').T.to_dict('list')
    task_travel_time["任务单"]=task_travel_time[["任务单"]].apply(lambda x: int(x["任务单
"][1:]),axis=1)
    task_travel_time["起始复核台"]=task_travel_time[["起始复核台"]].apply(lambda x:
review[x["起始复核台"]],axis=1)
    task_travel_time["抵达复核台"]=task_travel_time[["抵达复核台"]].apply(lambda x:
review[x["抵达复核台"]],axis=1)
    merge_result_tuples = [tuple(xi) for xi in task_travel_time.values]
    for item in merge_result_tuples:
        travelTime[(item[0],item[1],item[2])]=round(item[3],3)


    import numpy as np
    import geatpy as ea

    class MyProblem(ea.Problem): # 继承 Problem 父类
        def __init__(self):
            name = 'MyProblem' # 初始化 name（函数名称，可以随意设置）
            M = 1 # 初始化 M（目标维数）
            maxormins = [1] # 初始化 maxormins（目标最小最大化标记列表，1：最小化该目
标；-1：最大化该目标）
            Dim = 116 # 初始化 Dim（决策变量维数）
            varTypes = [1]*116 # 初始化 varTypes（决策变量的类型，元素为 0 表示对应的变量
是连续的；1 表示是离散的）
            lb = [1]*49+[1]*49+[4]*9+[1]*9    # 决策变量下界
            ub = [49]*49+[1]*49+[17]*9+[1]*9 # 决策变量上界
            lbin = [1] * Dim # 决策变量下边界（0 表示不包含该变量的下边界，1 表示包含）
            ubin = [1] * Dim # 决策变量上边界（0 表示不包含该变量的上边界，1 表示包含）
            # 调用父类构造方法完成实例化
            ea.Problem.__init__(self, name, M, maxormins, Dim, varTypes, lb, ub, lbin, ubin)
```

```
        def aimFunc(self, pop): # 目标函数
            X = pop.Phen # 得到决策变量矩阵
            x1 = X[:,[98]]
            x2 = X[:,[99]]
            x3 = X[:,[100]]
            x4 = X[:,[101]]
            x5 = X[:,[102]]
            x6 = X[:,[103]]
            x7 = X[:,[104]]
            x8 = X[:,[105]]
            x9 = X[:,[106]]
            ObjV = [] # 存储所有种群个体对应的总路程
            for i in range(X.shape[0]):
                distance = calculate(X[i].tolist(), travelTime, Time)
                ObjV.append(np.array(distance))
            pop.ObjV = np.array([ObjV]).T # 把求得的目标函数值赋值给种群 pop 的 ObjV
            pop.CV = np.hstack([np.abs(x1+x2+x3+x4+x5+x6+x7+x8+x9-49)])

    if __name__ == '__main__':
        """==========================实例化问题对象==========================="""
        problem = MyProblem() # 生成问题对象
        """=============================种群设置============================="""
        NIND = 500                  # 种群规模
        # 创建区域描述器，这里需要创建两个，前 2 个变量用 RI 编码，剩余变量用排列编码
        Encodings = ['P', 'RI']
        Field1 = ea.crtfld('P', problem.varTypes[:49], problem.ranges[:,:49],
problem.borders[:,:49])
        Field2 = ea.crtfld('RI', problem.varTypes[49:], problem.ranges[:,49:],
problem.borders[:,49:])
        Fields = [Field1, Field2]
        population = ea.PsyPopulation(Encodings, Fields, NIND) # 实例化种群对象（此时种群
还没被初始化，仅仅是完成种群对象的实例化）
        """=============================算法参数设置============================="""
        myAlgorithm = ea.soea_psy_GGAP_SGA_templet(problem, population) # 实例化一个
算法模板对象
        myAlgorithm.MAXGEN = 500 # 最大进化代数
        myAlgorithm.drawing = 1
        """==========================调用算法模板进行种群进化======================="""
        [population, obj_trace, var_trace] = myAlgorithm.run() # 执行算法模板
        population.save() # 把最后一代种群的信息保存到文件中
        # 输出结果
        best_gen = np.argmin(problem.maxormins * obj_trace[:, 1]) # 记录最优种群个体是在
```

哪一代

```
best_ObjV = obj_trace[best_gen, 1]
print('最优的目标函数值为：%s'%(best_ObjV))
print('最优的控制变量值为：')
for i in range(var_trace.shape[1]):
    print(var_trace[best_gen, i])
print('有效进化代数：%s'%(obj_trace.shape[0]))
print('最优的一代是第 %s 代'%(best_gen + 1))
print('评价次数：%s'%(myAlgorithm.evalsNum))
print('时间已过 %s 秒'%(myAlgorithm.passTime))
```