基于改进的蚁群算法优化仓库拣货管理过程

摘 要

本题背景是电商运行中比较关键的仓库拣货管理环节的优化问题（Warehouse Management System）．本题着重探究同情况下货仓中货物出货最短时间的优化问题，在求解问题过程中我们运用特定启发式算法——蚁群算法（ACO），结合实际情况建立了对应的数学模型，并使用 MATLAB、Python 等软件编程解决这些问题．

针对问题 1，我们给出拣货员在拣货过程中可能遍历的所有路线的距离计算公式．对起点与终点分别为货格和货格、货格和复核台、复核台和复核台三种情况之间的距离计算进行分类讨论．根据该距离矩阵的对称性，我们先对距离矩阵进行了优化，仅计算一个上三角距离矩阵，以简化计算量；

针对问题 2，我们为拣货员规划合理的路线使得任务单出库时间达到最小．由题目给出的假设，求解一个任务单最小出库时间就是求解拣货员在拣货该任务单过程中行走的最短路径．又因为未固定拣货员返回的复核台，所以此题等价于求解带权图最优哈密顿通路问题，该问题是一个完全 NP 问题，精确求解十分困难．我们考虑了原问题的一个松弛问题，并针对不同情况将松弛后的最优解恢复为原问题的近似最优解，最终得到拣货员从复核台 FH10 出发完成任务单 T0001 的最优时间为 9 分 13 秒．

针对问题 3, 其与问题 2 相比增加了双复核台的约束与任务单数量．我们首先采用问题 2 的优化模型分别计算出各任务单完成出货的最优时间，再考虑任务单执行顺序不同带来的影响，继而找到出货耗时最短的路线．此题最后的优化计算结果为总出货时间 40 分钟 46 秒，复核台 FH03 的利用率 3.679%，复核台 FH11 的利用率 2.453%．

针对问题 4，题目给出了更多的可用复核台，拣货员与大量的任务单．基于问题 2 的优化模型，我们先计算一个任务单在以不同或相同复核台分别作为起点与终点的最短路径，从而计算出不同任务单对应最短出货时间．接着，我们考虑 9 个拣货员分别从这些任务单中挑选所能达到的最短完成时间．根据排队理论，我们提出了一个算法来进行该问题的优化，从而解得全局最短时间的分配方案．

针对问题 5，在问题 4 的优化基础上添加一个未使用的复核台进行求解，求解过程与问题 4 类似．最后比较添加复核台前后所有任务单的总出库时间的变化．

针对问题 6，我们根据建模背景、求解模型所得结果与分析结果所得结论，结合理论研究与推理，将仓库布局按照我们提出的衡量标准进行分块分类，提供了一种可行的货物摆放方案．

关键词：蚁群算法，仓库管理模型，最优路径求解，哈密顿通路

目录

一、问题的提出

电商公司客户下订单后，商品开始下架出库，出库包含：定位、组单、拣货、复核和打包五个流程．仓库有多个货架，每个货架有多个货格，商品摆放在货格中，且每个货格最多摆放一种商品，商品可以摆放在多个货格．订单下达仓库后，进行定位操作，确定商品下架的货格和每个货格下架的商品数量．单个客户订单商品数量少，对于中小件商品仓库，需要将多个客户的订单合并，构成任务单，这就是组单操作．拣货开始，拣货员在某个复核台领拣货车及任务单，领取时间不计，然后根据推荐顺序依次访问任务单中商品所在货格，并下架商品，将商品放在拣货车上．下架完毕，拣货员将拣货车送往某个复核台，到达复核台后拣货员无需等待，继续领取拣货车和任务单，开始下一个任务单拣货流程．拣货时，拣货员可能多拣或者漏拣商品．拣货车放到定位组单拣货复核打包复核台先对任务中商品复核，然后将商品按照订单打包．对这个背景我们需要解决如下六个问题：

1. 当拣货员在仓库中拣货时，需要在货格之间、货格与复核台之间、复核台与复核台之间行走．由于这些行走通常要绕过障碍物，不能直接采用坐标计算欧几里得距离．请你按照图中距离标示，设计一种计算 3000 个货格和 13 个复核台总共 3013 个元素之间距离的方法．（附件中货架坐标可理解为第一个货格左下角坐标，如 S001 的 (x,y) 表示货格 S00101 的坐标．其他相类似．复核台坐标也理解为左下角坐标．）

2. 假设所有复核台正常工作，任务单 T0001 等待拣货，拣货员 P 在复核台 FH10 领取了任务单 T0001．请给 P 规划理想的拣货路线，包括货格访问顺序、返回的复核台，计算完成出库花费的时间（拣货员拣货开始到所有任务复核打包完成花费的时间）.

3. 假设 2 个复核台 (FH03，FH11) 正常工作，5 个任务单（T0002-T0006）等待拣货，继续由拣货员 P 负责拣货，P 初始位置为 FH03．通过建模和优化，请给 P 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．请计算完成出库需要花费的时间和每个复核台利用率.

4. 假设 4 个复核台（FH01，FH03，FH10，FH12）正常工作，49 个任务单（T0001-T0049）等待拣货，9 个拣货员（P1-P9）负责拣货，请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利用率.

5. 在问题 4 中，有 4 个复核台（FH01，FH03，FH10，FH12）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.

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？

二、问题分析

问题 **1**　问题 1 要求我们基于给定情景，找到不同对象间合适的距离公式，这是进行下一步研究的基础．由于两物体之间的距离仅与物体本身有关（即物体 A 到物体 B 的距离和物体 B 到物体的距离一样），所以，对于同样的物体，我们仅指定一个路径作为计算距离的依据．在进行计算时，我们假设货格间距离计算仅由考虑编号小的物体指向编号较大的物体，这样简化了我们的求解步骤．

1. 一个货架左侧的货格到（可以是另一个）货架右侧的货格．

2. 一个货架右侧的货格到（可以是另一个）货架左侧的货格．

3. 一个货架左侧的货格到（可以是另一个）货架左侧的货格．

4. 一个货架右侧的货格到（可以是另一个）货架右侧的货格．

根据货仓的对称性，我们知道情况 1 和情况 4 的距离计算思路是类似的．相对的，由于我们指定了路径起始的方向，情况 2 和情况 3 之间的对称性被破坏了．因此，情况 2 与情况 3 的距离计算思路不相同．更深入地，对于情况 1 或情况 4，我们还可以根据货格是否处于同一个货架，进行距离计算的再分类．

对于货格到复核台之间的距离计算，我们按照复核台分布情况进行分类讨论．

1. 复核台位于仓库平面布局的左侧（FH09-FH13）

2. 复核台位于仓库平面布局的底部（FH01-FH08）

并讨论不同货格对于这两种分类下复核台之间距离的关系．最后关于任意两个不同复核台之间的距离，我们直接使用其对应横纵坐标绝对差值的和作为它们之间简化的距离．

问题 **2**　问题 2 要求我们在固定任务单以及出发复核台的情况下，为拣货员规划理想的路线以达到任务单出货时间最短．这个问题研究了货仓中基本工作对象的一个固定行为模式，这对于提高货仓整体效率有重要意义．由于对于每个任务单，其下货时间不变，因此问题变为求解带权图最大哈密顿通路问题．由于这个完全 NP 问题目前没有较优的精确算法，我们将其先松弛为求解带权图最大哈密顿回路问题即著名的旅行商问题（TSP）．对于规模较小的旅行商问题，目前其精确的求解可以采用分支定界法、动态规划法等算法．但目光转移到具体问题，由于涉及到的点数较多，且考虑到后续问题的模型调用，精确计算不切实际．为此，我们考虑遗传算法、模拟退火法、蚁群算法、禁忌搜索算法等近似与启发式求解算法．基于对题目的理解，以及降低计算的复杂程度的需要，我们将采用蚁群算法．得到原问题的松弛解后，我们对回路进行拆环操作，按照起点绕回路的不同走向进行分类讨论，取此时拆环后的最佳解作为原问题的最优解．

问题 **3**　问题 3 在问题 2 的基础上增加了双复核台的约束与更多任务单数，并且要求我们计算复核台的利用率．在问题 2 的思路上，我们考虑双复核台带来的变化．蚁群算法可以帮我们找到两个复核台作为拣货员一次拣货出发点与复核台四种不同的连接组合方式：

1. 拣货人从 FH03 领取货单出发，回到 FH03 进行货单的复查
2. 拣货人从 FH03 领取货单出发，回到 FH11 进行货单的复查
3. 拣货人从 FH11 领取货单出发，回到 FH11 进行货单的复查
4. 拣货人从 FH11 领取货单出发，回到 FH03 进行货单的打包与复查

随后分别计算五个任务单采用以上四种圈图分类进行移动的时间长短，指定四种图中最短的为相应任务单采用的路线．再根据每个任务单出发点和终结点路线之间的联系，合理规划任务单顺序，逐层优化，找到最优的拣货路线．最后根据确定的路线计算复核台利用率．

问题 **4**　相比较与问题 3，问题 4 加入了多个拣货员（P1-P9），更多的可选复核台（FH01，FH03，FH10，FH12）与附件中所有的任务单．此时，由于人数的增加，我们必需要考虑拣货员在复核台复核时等待时间以计算复核台的利用率．问题前期解决的思路与问题三类似，基于排队理论，利用蚁群算法与相关的数据处理，分别计算出 49 个货单耗时最小的路线并将其按照从小到达顺序排列．再将这些按照最短时间排列好的任务单，按顺序分配给不同的拣货员进行处理．值得注意的是，此时的模型中应当加入记时模块以准确计算不同拣货员在同一复核台的等待时间与所有组单挑拣完毕的总时间，进而计算出每个复核台的利用率.[5]

问题 **5**　问题 5 在问题 4 的基础上，拟增加一个不同于 F01、F03、F10、F12 的未知复核台，即有五个复核台正常工作．根据问题 4 的优化模型，我们只需要循环求解加入不同复核台后任务单出货最优总时间．并且将其与问题 4 中最优时间进行比较得出变化的结论．

问题 **6**　针对前五个问题，由于题干中给出：“多人同时在一个货格拣货，不考虑等待的时间”与“不用考虑拣货车尺寸，货架和复核台高度”的假设信息，我们只需要令不同的货物集中在离复核台较近的一片区域进行考虑即可．但是问题 6 考虑到实际情况下拣货员在同一货格取货需要排队且货架走廊间可能因为多辆拣货车通行而变得拥堵从而导致出货速度反而变慢的情况．针对这个问题，我们建立了一个衡量不同区域“拥挤程度”的计算公式，以判断我们所提出的复核台分布方案与原问题复核台分布方案之间的优劣．

三、模型假设

1. 假设题目所给数据真实可靠

2. 仓库中每个货格最多摆放一种商品，商品可以摆放在多个货格

3. 一个拣货员负责对多个任务单时，每次只能拣一个任务单的商品

4. 拣货员的行走速度为 1.5m/s

5. 多人同时在一个货格拣货，不考虑等待的时间

6. 可不考虑拣货车尺寸，货架和复核台高度

7. 当绕障碍物折线行走时横向和竖向偏移都取 d=750mm

8. 复核台之间距离简化为两复核台坐标差的绝对值之和

四、定义与符号

符号	符号说明
A	对象间的总距离
d	绕障碍物行走时的横向竖向偏移
D	对象间的欧氏距离
L	货格的长度
H	复核台边长
y_{imin}	第 i 列最小的 y 坐标
y_{imax}	第 i 列最大的 y 坐标
l_y	复核台长度
s	货格号
R_{FH}	复核台利用率
t_{check}	复核时间
n_a	拣货员到复核台次数
t_{sum}	所有任务单出货完毕用时
W	走廊拥挤度衡量标准
t_{min}	某个任务单出货的最小时间

五、模型的建立与求解

5.1 数据预处理

对随题目给出的“附件 1：仓库数据”中的“任务单”表进行预处理．此表分别展示了不同任务单号包含的商品订单号以及对应的商品货格以及订单要求的商品数量．我

们发现，在较后的任务单号中存在存储相同商品的货格出现多次的情况．而题目中给出规定“每个货格最多摆放一种商品，商品可以摆放在多个货格”，因此我们首先将这种情况进行合并，方便后续计算使用．

5.2 问题 1 的模型

5.2.1 模型的建立

我们需要给出货仓内任意两个对象之间距离计算的公式，我们需要根据题目给出的几个固定参数（货格长度 L、转折偏移 d 等），难点在于行走的路线基本是折线，不能用欧氏距离计算．为设计出完备的距离计算方法我们建立了如下模型：

1. 复核台之间距离简化为两复核台坐标差的绝对值之和，如复核台 A 坐标 x_1, y_1，复核台 B 坐标 x_2, y_2，则两复核台距离为 $|x_1 - x_2| + |y_1 - y_2|$；

2. 根据在计算货格间的距离时，由于距离的无序性（A 到 B 的距离与 B 到 A 的距离相同），我们规定：距离指的是从编号小的货格指向编号大的货格这样的有向线段的长度，依次规定我们减少了大量的计算量．并将距离划分为四类：

   (a) 一个货架左侧的货格到（可以是另一个）货架右侧的货格．

   (b) 一个货架右侧的货格到（可以是另一个）货架左侧的货格．

   (c) 一个货架左侧的货格到（可以是另一个）货架左侧的货格．

   (d) 一个货架右侧的货格到（可以是另一个）货架右侧的货格．

3. 复核台和货格之间的距离，复核台有横置 ($F_{01} - F_{08}$) 和竖置 ($F_{09} - F_{13}$) 两种，两种计算距离的表达式不同．

5.2.2 模型求解

1. 复核台之间距离简化为两复核台坐标差的绝对值之和，如复核台 A 坐标 x_1, y_1，复核台 B 坐标 x_2, y_2，则两复核台距离为 $D_1 = |x_1 - x_2| + |y_1 - y_2|$

2. 如图中 L4 与 L2 所示，在同一排货柜中，从货格出发可以向上走到达目标点，也可以向下到达目标点，其距离有所不同，这里我们寻求最短距离，由于向上走与向下走的决定与两个货格号相关，且有以下关系．

$$\begin{cases} f_1(x) & \text{当 } \mathrm{mod}\,(s_1/100) + \mathrm{mod}(s_2/100) <= 15 \\ f_2(x) & \text{当 } \mathrm{mod}\,(s_1/100) + \mathrm{mod}(s_2/100) > 15 \end{cases} \tag{1}$$

因此根据几何距离定义这两个函数：

$$f_1(x) = 2\epsilon(L/2 + d + min(y_1 - y_{1min}, y_2 - y_{1min})) \tag{2}$$

$$f_2(x) = 2\epsilon(L/2 + d + min(y_{1max} - y_1, y_{2max} - y_2)) \tag{3}$$

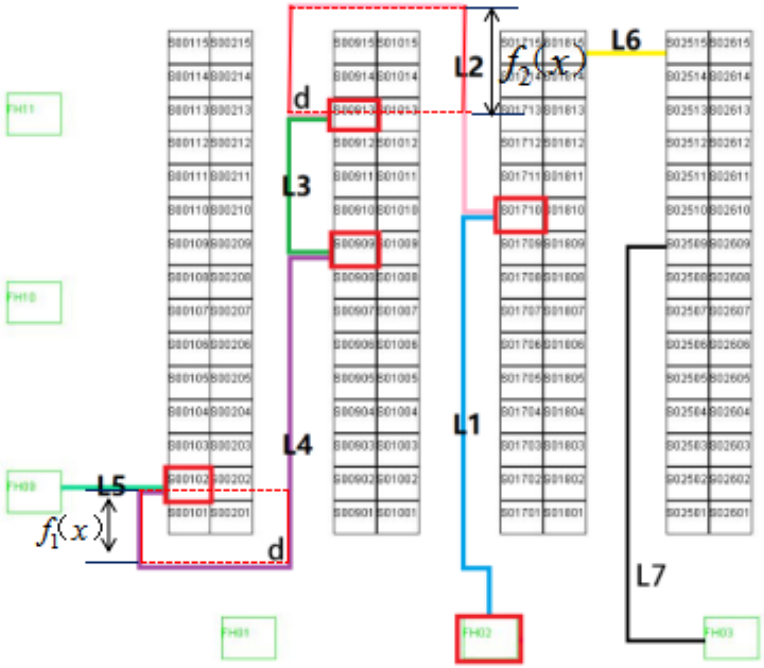

图 1 $f_1(x)$ 和 $f_2(x)$ 说明

对于货格和货格之间的距离，若两个货格在同一列或在不同排式中 ϵ=0，否则 ϵ=1. 对于竖置复核台和货格之间的距离，若货格在第一列，则取 ϵ=0，否则取 ϵ=1.

(a) 一个货架左侧的货格到（可以是另一个）货架右侧的货格，红色为出发货格，紫色为结束货格，深蓝色线段代表绕障碍物行走时的横向竖向偏移 d，紫色线段代表货格长度 L. 图 2 是关于 a 情况的图例，计算 A 坐标 x_1, y_1，B 坐标

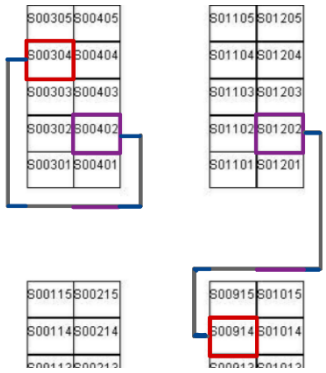

图 2 情况 a 图例

x_2, y_2 间距离的公式为：

$$A_1 = D + 4d + L + f_1(x) \tag{4}$$

$$A_2 = D + 4d + L + f_2(x) \tag{5}$$

(b) 一个货架右侧的货格到（可以是另一个）货架左侧的货格．图 3 是关于 b 情况的图例，计算 A 坐标 x_1, y_1，B 坐标 x_2, y_2 间距离的公式为：

$$A_3 = D - L + f_1(x) \tag{6}$$

$$A_4 = D - L + f_2(x) \tag{7}$$

(c) 一个货架左侧的货格到（可以是另一个）货架左侧的货格．图 4 是关于 c 情

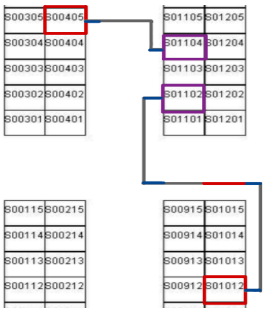

图 3 情况 b 图例

图 4 情况 c 图例

况的图例计算 A 坐标 x_1, y_1，B 坐标 x_2, y_2 间距离的公式为

$$A_5 = D + 2d + f_1(x) \tag{8}$$

$$A_6 = D + 2d + f_2(x) \tag{9}$$

(d) 一个货架右侧的货格到（可以是另一个）货架右侧的货格，与 c 情况完全一致，故不再列出．

3. 货格和复核台之间的距离：

(a) 货格与横置 $(F_{01} - F_{08})$ 复核台的距离：

i. 若左边货格到复核台右边：$A_7 = D - 3H/2 + L/2$

ii. 若左边货格到复核台左边：$A_8 = D - H/2 + L/2 + 2d$

iii. 若右边货格到复核台右边：$A_9 = D - H/2 - L/2$

iv. 若右边货格到复核台右边：$A_{10} = D - 3H/2 + 3L/2 + 2d$

(b) 货格与竖置 $(F_{09} - F_{13})$ 复核台的距离，不论是哪类货格，都是到复核台右边更近．因此分为两种情况:

i. 左边货格到复核台：$A_{11} = D - 3/2H + L/2 + f_1(x)\ D - 3/2H + L/2 + f_2(x)$

ii. 右边货格到复核台：$A_{12} = min(D + 2d - 3/2H + 3L/2 + f_1(x), D + 2d - 3/2H + 3L/2 + f_2(x))$

5.3 问题 2 的模型

5.3.1 模型的建立

问题 2 要求我们求解任务单 T0001 出货完成的最短时间．根据题目假设我们知道最短时间 t_{min} 计算公式包含任务单下货时间，拣货员行走时间与复核台打包复核时间三个部分，即表达为：

$$t_{min} = min_{i\in\mathcal{D}}(t^i_{pick} + t^i_{walk}) + t_{check} \tag{10}$$

这里 $\mathcal{D}$ 是完成某个任务单所有可能路径根据题目假设，给定任务单，其下货时间是固定的且拣货员工作时行走速度为 1.5m/s，复核台打包时间恒为 30s．因此，问题 2 数学模型可以建立为：求从复核台 F10 出发，经过所有的货柜拣货点后回到一个复核台最短路径的模型．此时，求得最短路径 (dis_{min}) 后得到的最短时间表述为:

$$t_{min} = \frac{dis_{min}}{v_{walk}} + t_{pick} + t_{check} = \frac{dis_{min}}{1.5m/s} + t_{pick} + 30s \tag{11}$$

其中 $t_{pick} = \sum_{k\in \mathrm{L}} n_k \cdot t_k$ ，$t_k = \begin{cases} 5s & n_k < 3 \\ 4s & n_k \geqslant 3 \end{cases}$，L 是任务单中商品的编号，$n_k$ 为商品的个数．

针对问题 2，模型可以抽象为求从复核台 F10 出发，经过所有的货柜拣货点后回到一个复核台的模型，其与著名的旅行推销员问题有些类似．旅行推销员问题（英语：Travelling salesman problem, TSP）也叫做 TSP 问题，即给定一系列城市和每对城市之间的距离，求解访问每一座城市一次并回到起始城市的最短回路．它是组合优化中的一个 NP 难问题，而规模较大的 NP 难问题的时间复杂度在多项式程度上无法解决．对于求解最短哈密顿回路问题，即旅行推销问题，这里我们采用蚁群算法求解 TSP 问题的最优近似解[4]．蚁群算法是一种用来寻找优化路径的概率型算法．这种算法具有分布计算、信息正反馈和启发式搜索的特征，本质上是进化算法中的一种启发式全局优化算法．蚁群可以在不同的环境下，寻找最短到达食物源的路径．这是因为蚁群内的蚂蚁可以通过某种信息机制实现信息的传递．蚂蚁会在其经过的路径上释放一种可以称之为“信息素”的物质，蚁群内的蚂蚁对“信息素”具有感知能力，它们会沿着“信息素”浓度较高路径行走，而每只路过的蚂蚁都会在路上留下“信息素”，这就形成一种类似正反馈的机制，这样经过一段时间后，整个蚁群就会沿着最短路径到达食物源．如图 5 所示，在蚁群无障碍物时如图 5（a）所示，在蚁群经过的路程中间放一个障碍物，将会阻断蚁群的行走，如图 5（b）所示，因此，蚁群会分成两部分分别向障碍物的上端与下端进行行走，并在沿途留下“信息素”，蚁群会在 Nest 与 Food 之间来回行走，假设蚁群在障碍物上端的路线路程较短，那么与下端相比，在相同时间内蚁群在上端走过的次数较多，留下的“信息素”浓度较高，而高浓度的“信息素”会使得更多的蚂蚁走上端的路程，如图 5（c）所示，到达一定长的时间后，所有的蚂蚁都走上端的路程，也就是在两条路中最短的一条路，如图 5（d）所示．将蚁群算法应用于解决优化问题的基本思路为：用蚂蚁的行走路径表示待优化问题的可行解，整个蚂蚁群体的所有路径构成待优化问题的解

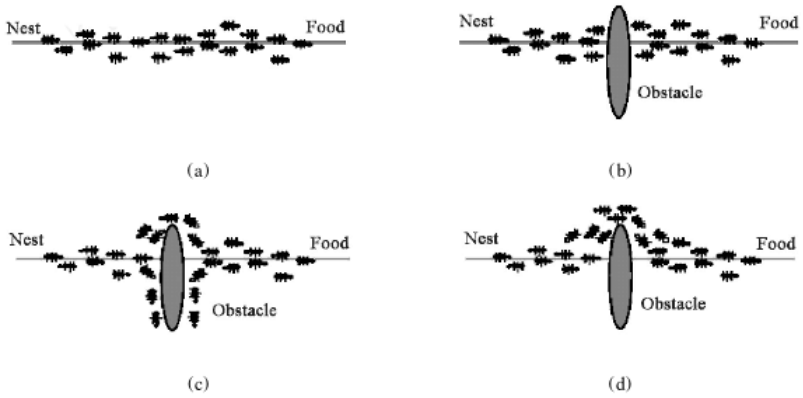

图 5 蚁群算法图例

空间．路径较短的蚂蚁释放的信息素量较多，随着时间的推进，较短的路径上累积的信息素浓度逐渐增高，选择该路径的蚂蚁个数也愈来愈多．最终，整个蚂蚁会在正反馈的作用下集中到最佳的路径上，此时对应的便是待优化问题的最优解．本题蚁群的参数[1]是：

参数名称	参数大小
蚂蚁数量 m	200
最大迭代次数 Nc_{max}	200
启发因子矩阵 η	距离的倒数
信息素重要程度 $alpha$	1
启发式因子重要程度 $Beta$	5
信息素蒸发系数 rho	0.5
信息素增加强度系数 Q	1

5.3.2 模型求解

对于问题 2 而言，其简单模型如图 6 所示．问题 2 要求从 $v1$ 出发，经过 $u1u2u3u4u5$，最后回到 $v2$ 的最短路径，其路径需要经过所有的 $u_i, (i = 1,2...)$ 点，构成一个哈密顿回路．由于 $v2$ 所决定的复核台未知，因此我们将在 $u5$ 点处，找到离其最近的复核台．由此问题变为找 $v1$ 到 $u5$ 的最短路问题，如图所示．这里如果我们求出图 7 中 6 个点的最短的哈密顿回路，再将多加的 $v1 \rightarrow u5$ 的路径长度减掉，就可以近似求解处 $v1 \rightarrow u5$ 的最短路径．公式如下：

$$L_{min} \approx L_{min} - d(v_1, u_5) \tag{12}$$

对于求解从 $v1$ 出发，经过一个哈密顿回路后，会在最后一个点回到 v1，但是由于与 $v1$ 临近的点有两个，如图 7 所示．

1. 从 $v1$ 出发，经过顺时钟行走后最后一个点到达 u1；

2. 从 $v1$ 出发，经过逆时钟行走后最后一个点到达 u5．

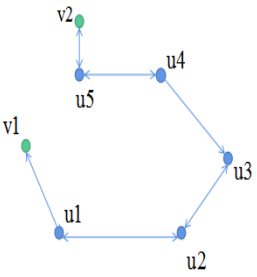

图 6 问题 2 简单模型

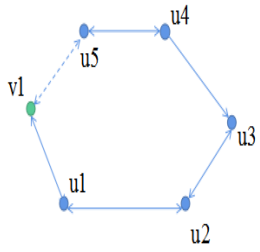

图 7

由此，最后点出发到达复核台的情况均有两种，这里我们选择两种情况中距离最小的情况．公式如下：

$$D_{min} = min(L_{min}(ac), L_{min}(c)) \tag{13}$$

这里 $L_{min}(ac)$ 代表逆时针距离，$L_{min}(c)$ 代表顺时针距离．

我们利用上述分析的模型进行求解，得到图所示的回路，其表示将组单中不同货物对应货格的横纵坐标映射到直角坐标系后的拣货员挑选货物的行进路线．随后为了达到题目中通路的要求，我们进行对路径最短回路的破圈处理，将与复核台相连两端的货格分别进行讨论得到通路情况下的近似最优解

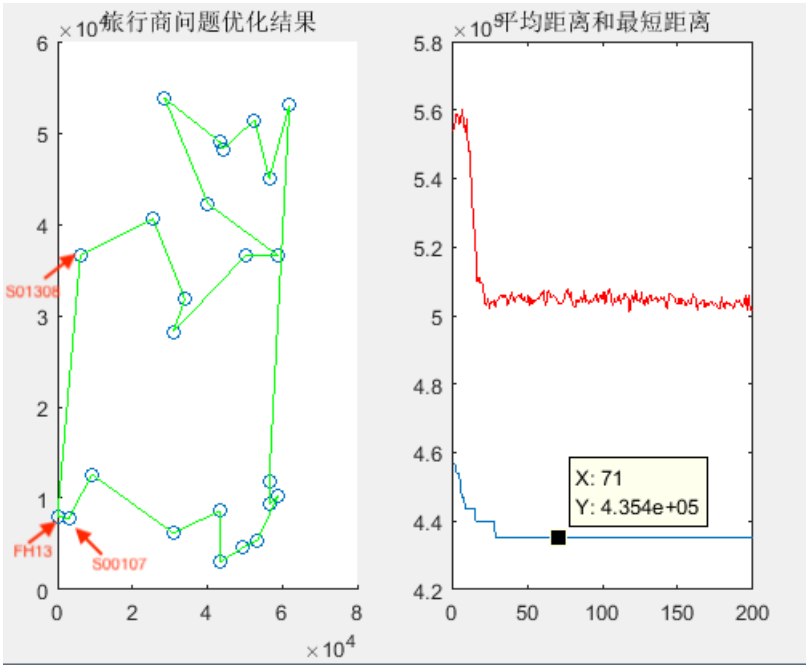

图 8 问题 2 的解

由图 8 可以看出蚁群算法在计算最短距离时，迭代次数在 70-200 次之间时，最短距离均无改变，这说明松弛解的最短路程十分稳定，即此最优解的可信度较高，模型拟合较好．为方便下列问题的求解，以及提高计算效率，将最大迭代次数改为 100 次，其对结果影响较小．

最终的路线为’FH10’, ’S00107’, ’S01713’, ’S07305’, ’S10508’, ’S10501’, ’S12103’, ’S13004’, ’S14510’, ’S13809’, ’S13812’, ’S15911’, ’S14401’, ’S13509’, ’S11205’, ’S11106’, ’S07212’, ’S10115’, ’S14908’, ’S12608’, ’S07515’, ’S08502’, ’S06213’, ’S01308’, ’FH13’ 根据问题 2 开始的分析，我们不难得到以下最优数据：对应拣货员从拿到货单到完成货单

T0001 出货最短距离	拣货员行走时间	T0001 下货时间	复核台打包时间
519.9m	5 分 46 秒	2 分 57 秒	30 秒

对应商品的复核打包时间为 553 秒即 9 分 13 秒.

5.4 问题 3 的模型

5.4.1 模型的建立

相比较问题 2，问题 3 加入了两个复核台（FH03、FH11）的限制，同时增多了订单数量．对于每个货单经过完整流程的起始点与终止点我们考虑四种情况：

1. 拣货人从 FH03 领取货单出发，回到 FH03 进行货单的复查
2. 拣货人从 FH03 领取货单出发，回到 FH11 进行货单的复查
3. 拣货人从 FH11 领取货单出发，回到 FH11 进行货单的复查
4. 拣货人从 FH11 领取货单出发，回到 FH03 进行货单的打包与复核

针对每个货单，我们基于这三种情况分别利用蚁群算法求得近似最优解，并找出每一个货单对应耗时最小拣货员行走路径．由于这时只存在一个拣货员进行拣货，因此不需要考虑在复核台等待的时间．

5.4.2 最优路径求解

对于全局最优路径的求解，我们考虑将所有货单对于其耗时最短的路径方式进行规划．对于上述分类的四种情形直观上我们不妨将情形 1 与情形 3 当作图中最优圈的情况进行讨论以方便中间计算．首先考虑路径中一个复核台成圈的情况，即情况 1 与情况 3．然后利用情况 2 中包含了两个不同复核台的最优路径将上述两种例子进行连接．在实际求解中，我们发现，对于每个任务单路径最优的情况只存在包含复核台 FH03 的最优圈以及拣货起始点与终止点为复核台 FH11 与 FH03 的通路．因此我们先安排包含 FH03 最优圈的任务单，随后利用起点终点不同的拣货路线交替连接复核台 FH11 与 FH03．

任务单执行顺序	出货最短距离	拣货员行走时间	任务单下货时间	起点复核台	终点复核台
T0003	401.1 米	4 分 27 秒	2 分 51 秒	FH03	FH03
T0004	438.8 米	4 分 52 秒	3 分 6 秒	FH03	FH03
T0005	376.0 米	4 分 11 秒	2 分 45 秒	FH03	FH03
T0006	469.9 米	5 分 13 秒	2 分 48 秒	FH03	FH11
T0002	451.6 米	5 分 1 秒	3 分 2 秒	FH11	FH03

5.4.3 复核台利用率的求解

基于 5.3.2 的讨论，由于不存在复核台等待时间，因此求解单个复核台的利用率我们只需要计算到达此复核台的次数即可．到达一次复核台代表拣货员在此复核台进行了一次时长 30s 钟的货物复核与打包，将此类时间累加，求解其与拣货员挑拣完毕货物总时间的比值即得到每一个复核台的利用率．

$$R_{FH} = \frac{t_{check} \cdot n_a}{t_{sum}} \tag{14}$$

其中 R_{FH} 是复核台利用率，t_{check} 是复核时间，n_a 是拣货员到复核台次数，t_{sum} 是所有任务单出货完毕的总时间最终运算得到的结果如下

项目	项目数值
任务完成用时	2446 秒
复核台 F03 利用率	3.679%
复核台 F11 利用率	2.453%

5.5 问题 4 的模型

5.5.1 模型的建立

与问题 3 相比，问题 4 将复核台的数量增加到了四个（F01、F03、F10、F12），且要求使用 9 个拣货员（P1-P9）对任务单中全部 49 个任务进行拣货．对于每一个拣货员领取任务单后所走的最优路径求解，我们延续使用问题 2 所建立的模型进行优化．对于拣货员从复核台领取任务单后，完成任务单的拣货即走遍了任务单中所有货物对应的货格，最终到达的复核台复核的最优路径计算我们调用问题 3 的优化模型．得到每个任务单通过一个复核台的最短距离后，由于同一个任务单拣货员拣货的时间一定，因此我们能够根据一个货单拣货最短路径求解出其出库的最短时间．最后通过建立拣货员拣货的时间模型得到最优的初始时刻的人员分配，以及在该最优的人员分配下的最短出库时间和人员任务单分配．问题 4 优化模型的建立流程图具体如图所示．在拣货时间模型中，我们设置了 4 个矩阵，具体参数如下，其中 n1 为处理后的任务单的行数，n2 取值 100．

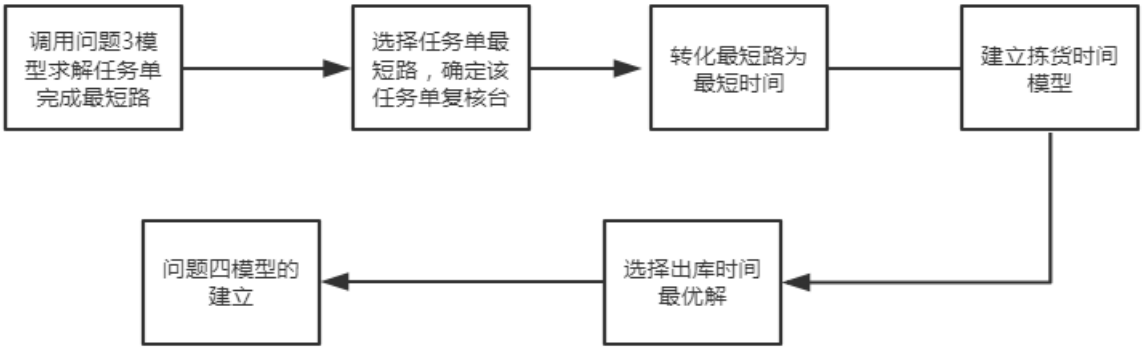

图 9 问题 4 流程图

矩阵	矩阵参数 $m \times n$
A	$n1 \times 5$
B	$n2 \times 18$
C	9×3
D	1×13

对于四个矩阵的具体行列意义，如下图所示，对于矩阵 A，第一列为任务单号；第二列为

1	2	3	4	5
48	0	t	F03	F01
48	0	t	F01	F03
49	0	t	F03	F03

图 10 A 矩阵实例

1	2	....	17	18
T	F03			

图 11 B 矩阵实例

1	2	3
$T_{起}$	F03	F10

图 12 C 矩阵实例

1		13
$t_{起}$		

图 13 D 矩阵实例

状态 0-1 变量，0 代表该任务单未完成，1 代表该任务单已完成；第三列为完成该任务单的最短时间 t；第四列与第五列为复核台的起点与终点，其中 $F01 \rightarrow F03$ 与 $F03 \rightarrow F01$ 为不同行数据，但它们第二列的状态变量同时改变，即完成 48 号任务单，从 $F01 \rightarrow F03$，状态变量变为 1，其 48 号任务单的 $F03 \rightarrow F01$ 的状态变量也变为 1.

- 对于矩阵 B，一共有 18 列，即列，每一个人分配两列，第一列（奇数列）为该拣货员在 T 时刻，第二列（偶数列）为该拣货员在 T 时刻所在的复核台号．其中 T 时刻为复核时刻或者打包后拣货员领取任务单出发时刻．

- 对于矩阵 C，大小为矩阵，每一行代表着一个拣货员，第一列代表拣货员完成任务单时的时刻，第二、三列代表该任务单的起点与终点的复核台．

- 对于矩阵 D，大小为矩阵，13 列代表了 13 个复核台号，其中的行数据为该复核台最后工作的总时间．

对于拣货时间模型的创建，原理与下图所示．先将前 9 个任务单分给拣货员，，此时 9 个任务单的完成时间由于路程 1，使得均在 C 矩阵内．在，C 矩阵中的完成时刻中选择最小的那个时刻，将其放入 B 矩阵对应拣货员中，如路径 2 所示．将 B 矩阵中对应的复核台号的时间加上 30s，为该复核台的完成任务单时刻，将其放入 D 矩阵中对应复核台下，如路径 3 所示，对应任务单完成，在 A 矩阵中修改任务单状态变量为 1．然后拣货员从 A 矩阵中按任务单完成时间从小到大搜索状态变量为 0 的任务单，选择从已完成任务单的复核台出发，可完成的最快任务单．重复递归，直至所有的任务单均完成，级状态变量均变为 1．具体计算代码由 Matlab 编程实现．然后拣货员从 A 矩阵中按

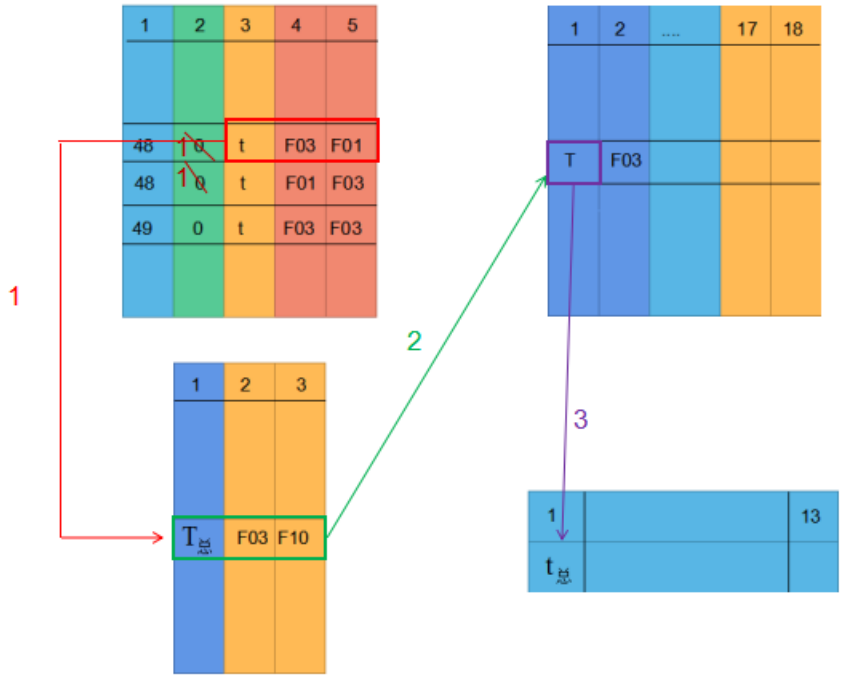

图 14 拣货时间模型示意图

任务单完成时间从小到大搜索状态变量为 0 的任务单，选择从已完成任务单的复核台出发，可完成的最快任务单．重复递归，直至所有的任务单均完成，级状态变量均变为 1．具体计算代码由 Matlab 编程实现．

5.5.2 模型求解

对于运用该模型进行计算时，预计在分配好初始时刻 9 个拣货员的起始复核台后，经过模型中的到达其他复核台，并在完成复核任务的复核台接受新的最短时间任务单，直到拣货员刚好完成所有的任务单.

但在实际情况中，不同的 9 人分配初态类型，所产生的结果也是不相同的，可能会导致有些任务单在最后并没有拣货员出现在该任务单的可发放任务的复核台，从而导致其没有被完成. 在对于不同的初态类型进行 Matlab 运算后，将迭代计算后无任务单残留标红点，有任务单残留标绿点，如下图所示；通过下图可以清晰的看出，其可以

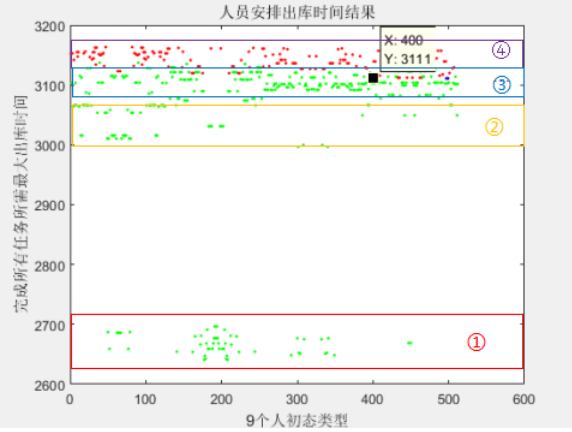

图 15 安排人员出库散点图

具体的从下到上分为 4 层，其中红点区域大部分集中在④区，大部分绿点集中在③区，有少部分绿点集中在①区和②区. 通过对所在点的任务清单完成情况表的检索，我们发现①区几乎在迭代计算后仍有 3 个任务单未完成，与之对应的，②区有 2 个任务单未完成，①区有一个任务单未完成. 由此，我们得到以上的分区状况反映了在不同的初态类型下，拣货员按照模型拣货后的任务单残留情况.

由于初态分布任务点情况，如下表中所示，因此新的任务单最短完成时间会大于 457.8s，对于有任务单残留的情况，需调动距离任务单所需复核台最近的拣货员前往领取任务单，而最小的无任务单残留情况与最小的有任务单残留情况的时间差为：

$$\Delta t \approx 3110s - 2700s = 410s < 457.8s \tag{15}$$

拣货员号	任务单号	完成时间	起始复核台号	收货复核台号
1	49	391.27	10	3
2	23	426.00	1	12
3	19	435.33	3	3
4	5	443.47	1	12
5	13	444.87	12	10
6	9	445.53	3	1
7	36	453.73	12	3
8	30	456.33	3	1
9	42	457.80	1	3

因此，最短出库时间的最优分配将在无任务单残留的初态类型中产生，在图中最优解共有 4 组，分别为 398,400,497,499，这里选择 398 类型的初态作为分析，上表显示即为 398 类型的初态人员分配情况．（这里 398 代表在表格 4_2_E_all.xls 中 1+(398-1)*9 到 9+(398-1)*9 行的初态数据）下图为前两个拣货员随时间其子复核台状态改变的行走结果图．

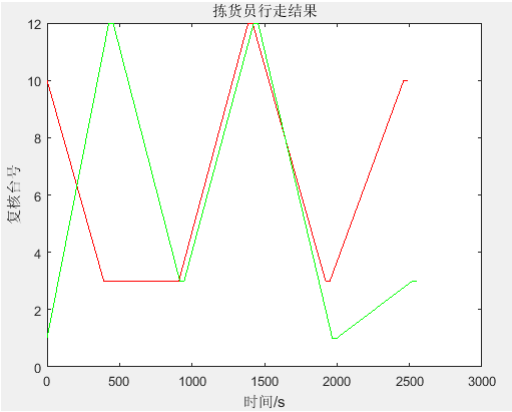

图 16 拣货员行走结果图

根据上述模型，能够轻易地得到以及分配给不同拣货员的任务单号以及复核台利用率．

拣货员编号	安排任务单
P1	T0049, T0024, T0046, T0002, T0040,T0016
P2	T0023, T0003, T0010, T0020, T0047,T0017
P3	T0019, T0033, T0018, T0027, T0026,T0025
P4	T0005, T0037, T0001, T0022,T0029
P5	T0013, T0015, T0034, T0045, T0011,T0028
P6	T0009, T0041, T0032, T0038,T0014
P7	T0036, T0039, T0007, T0008,T0035
P8	T0030, T0021, T0044, T0004,T0031
P9	T0042, T0043, T0006, T0012,T0048

复核台编号	复核台使用次数	复核台利用率
FH01	8	7.70%
FH03	22	21.22%
FH10	6	5.79%
FH12	13	12.54%

5.6 问题 5 的模型建立与分析

对于问题 5 而言，要想知道多增加一个复核台 FHi，(i=2,4,5,6,7,8,9,11,13)，对于出库时间的影响，可以先利用问题 3 的模型求解出每一个任务单在五个复核台上的最短时间（最短路程）共 15 种情形，在这 15 中情形中选择时间最短的路径作为该任务单的完成路径图．利用问题 4 的模型的拣货时间模型可求解出在不同初态类型的最优分配方案的出库时间散点图，在图中选择无任务单残留的最低点的初态类型，由此，可得出在新加的该复核台 FHi 后的最优分配方案与出库时间．对于每一个可能的 FHi 均类似求解，由此可得到关于新增复核台号的出库时间散点图，再将其与问题四中的最优出库时间比较，可得到新增复核台对于出库时间的影响．

5.7 问题 6 的模型建立与分析

前五题中，由题干中给出假设：“多人同时在一个货格拣货，不考虑等待的时间”与“不用考虑拣货车尺寸，货架和复核台高度”的信息，我们只需要令不同的货物集中在离复核台较近的一片区域进行考虑即可．但是问题 6 考虑到实际情况下“畅销品所在货架可能拥挤，反而降低拣货效率”，即拣货员在同一货格取货需要排队且货架走廊间可能因为多辆拣货车通行而变得拥堵从而导致出货速度反而变慢的情况．针对这个问题，我们建立了一个衡量货架间走廊拥挤程度的标准

$$W = \frac{N_p}{T} \tag{16}$$

其中 N_p 代表着每一次任务单分配执行后，在对应最优出货时间 T 内某条走廊上每一个拣货员到达次数的总和．直观上来说，在理想的情况下，如果将畅销商品选择摆放在 13 个复核台所围成矩形包围的货格中（如图所示的 A 区域），应当是最优的安排，因为 A 区域的货格离可使用复核台之间的路径较近．相应地，B，C 区域的货格被访问频率应当逐步下降．此时我们加入一个假设：A 区域全部摆放了畅销商品．我们以适当的比例

图 17 不同区域畅销品分布区域

将 B，C 区域货格中存储的商品与 A 区域中商品互换，每次交换后利用上面定义的拥挤度标准来衡量仓库效果的优劣．经过算法迭代最终能够找出收敛的商品交换比例，从而得到较佳的商品摆放规律．

六、模型优缺点

本文建立的模型大多以蚁群算法为中心基础．蚁群算法是一种分布式的并行算法，也是一种正反馈算法，其具有较强的鲁棒性，易于其他方法结合. 在解决 TSP 问题中具有较好的应用. 问题 3、4、5 建立的模型具有较强的适应性，对于同种问题的解决，快速且易于理解.

但蚁群算法也有局限性，带有明显的经验性，其收敛速度慢，计算时间长，易于过早陷入局部最优解. 在问题 2、3、4、5 模型的求解中都认为其解为近似最优解，但结果与真正最优解的距离尚不可知，对于规模较大的运算，模型计算速度较慢. 这导致本次竞赛的第五题我们没能来得及进行完整的数据分析，这是我们本次比赛的失虑之处．

参考文献

[1] 徐红梅，陈义保，刘家光，王燕涛. 蚁群算法中参数设置的研究. 山东理工大学，2008

[2] 蚁群算法的 Matlab 程序. 解放军信息工程大学,2008

[3] Flood M M. The Traveling-Salesman Problem[J]. Operations Research, 1956, 4(1):61-75.

[4] 陈文兰, 戴树贵. 旅行商问题算法研究综述 [J]. 滁州学院学报, 2006, 8(3):1-6

[5] 李文玉. 智能仓库系统多机器人任务分配问题研究. 北京物资学院，2016.

[6] 桑国珍，何小虎. 基于自适应蚁群算法的研究. 渭南师范学院计算机科学系，2013

[7] 扬德芹. 一种自适应蚁群算法及其应用 [J]. 软件导刊，2007,11.

[8] 李树嵩. 蚁群算法及应用研究. 哈尔滨学院，2017.

[9] 李国勇. 智能控制及其 MATLAB 实现 [M]. 电子工业出版社，2005.

附 录

附录 1：数据预处理代码

```
clear ;
B=xlsread('C:\Users\win\Desktop\建模国赛\问题2初值');%导入数据表
N=size(B,1);ssum=0;sssum=0;C=zeros(N,3);num=1;
for k=1:49
ssum=0;i=1;
for j=1:N
if B(j,1)==k;
ssum=ssum+1;
end
end
while i<=ssum
C(num,:)=B(sssum+i,:);
if B(sssum+i,2)==B(sssum+i+1,2)
C(num,3)=B(sssum+i,3)+B(sssum+i+1,3);
i=i+1;
end
num=num+1;i=i+1;
end
sssum=sssum+ssum;
end
xlswrite('C:\Users\win\Desktop\建模国赛\4_xlscheck.xls',C);%将处理后的数据表导出
```

附录 2：问题 1 代码

```
#计算货格之间的距离
import os
import pandas as pd
import numpy as np
import xlwt

root = 'C/附件/'
file = os.path.join(root, '附件1: 仓库数据.xlsx')
case = pd.read_excel(file, sheetname = '货格' , header = None, index = False)

def Judge(x, y):
if ((x&1 and y&1)or(not x&1 and not y&1)):
        if (int((x-1)/8) == int((y-1)/8)):
            return 1
```

```
        else:
            return 2
    elif( x&1 and not y&1):
        if (x%8+1 == y%8):
            return 3
        else:
            return 4
    elif(not x&1 and y&1):
        if (x%8-1 == y%8):
            return 5
        else:
            return 6

def cal(flag, a, b):
    G = 750#绕过障碍物的间隔
    L = 800#货格的边长
    #提取坐标
    x0 = int(case.loc[case[0] == a, 1])
    y0 = int(case.loc[case[0] == a, 2])
    x1 = int(case.loc[case[0] == b, 1])
    y1 = int(case.loc[case[0] == b, 2])
    #对应最底部的箱格y坐标
    y0_flagdown = int(case.loc[case[0] == a[:4]+str(0)+str(1), 2])
    y1_flagdown = int(case.loc[case[0] == b[:4]+str(0)+str(1), 2])
    y0_flagup = int(case.loc[case[0] == a[:4]+str(1)+str(5), 2])
    y1_flagup = int(case.loc[case[0] == b[:4]+str(1)+str(5), 2])
    D = abs(x0-x1)+abs(y0-y1)#欧式距离
    f1 = 2*(0.5*L+G+min(abs(y0-y0_flagdown), abs(y1-y1_flagdown)))
    f2 = 2*(0.5*L+G+min(abs(y0-y0_flagup), abs(y1-y1_flagup)))
    choose = int(a[4:])+int(b[4:])#判断从上走还是从下走
    if(flag == 1):
        d = D+2*G
    elif(flag == 2):
        if (choose<=15):
            d = 2*G+D+f1
        else:
            d = d = 2*G+D+f2
    elif(flag == 3):
        if(choose<=15):
            d = D+4*G+L+f1
        else:
            d = D+4*G+L+f2
    elif(flag == 4):
        d = D+4*G+L
```

```
    elif (flag == 5):
        if (choose<=15):
            d = D-L+f1
        else:
            d = D-L+f2
    elif(flag == 6):
        d = D+4*G
    return d

#求解上三角矩阵写入表格
def Dis(case):
    book = xlsxwriter.Workbook('result.xlsx', {'constant_memory': True})
    sheet = book.add_worksheet('test')
    for i in range(1, 3001):
        sheet.write(i, i, 0)
        for j in range(i+1, 3001):
            x = int(case[0][i][1:4])
            y = int(case[0][j][1:4])
            flag = Judge(x, y)
            sheet.write(i, j, cal(flag, case[0][i], case[0][j]))
            sheet.write(j, i, cal(flag, case[0][i], case[0][j]))

    book.close()

if __name__ == '__main__':
Dis(case)




#计算复核台与货格之间距离以及复核台与复核台之间距离
#的上三角矩阵
#输入为完整的货架号码
#x为货格号，y为复核台1-8
def Judge2(x, y):
    d0 = 1000
    L = 800
    G = 750
    x_x = int(case.loc[case[0] == x, 1])
    x_y = int(case.loc[case[0] == x, 2])
    y_x = int(case1.loc[case1[0] == y, 1])
    y_y = int(case1.loc[case1[0] == y, 2])
    D = abs(x_x-y_x)+abs(x_y-y_y)#欧式距离
    x1 = int(x[1:4])
```

```
    if (x_x<=y_x+100):
        if (x1&1):
            d = D+0.5*(L-d0)+2*G
        elif(not x1&1):
            d = D-0.5*(L+d0)
    else:
        if (x1&1):
            d = D-1.5*d0+0.5*L
        elif(not x1&1):
            d = D+1.5*(L-d0)+2*G
    return d

#复核台9-13与货格之间的距离
def Judge3(x, y):
    d0 = 1000
    L = 800
    G = 750
    x_x = int(case.loc[case[0] == x, 1])
    x_y = int(case.loc[case[0] == x, 2])
    y_x = int(case1.loc[case1[0] == y, 1])
    y_y = int(case1.loc[case1[0] == y, 2])
    x_flagdown = int(case.loc[case[0] == x[:4]+str(0)+str(1), 2])
    x_flagup = int(case.loc[case[0] == x[:4]+str(1)+str(5), 2])
    D = abs(x_x-y_x)+abs(x_y-y_y)#欧式距离
    f1 = 2*(0.5*L+G+abs(x_y-x_flagdown))
    f2 = 2*(0.5*L+G+abs(x_y-x_flagup))
    if (int(x[1:4])&1):
        if (int((int(x[1:4])-1)/8) == 0):
            d = D-1.5*d0+0.5*L
        else:
            d = D-1.5*d0+0.5*L+min(f1,f2)
    elif(not int(x[1:4])&1):
        d = D+1.5*(L-d0)+2*G+min(f1,f2)
    return d

#复合台之间的距离
def Judge4(x,y):
    x_x = int(case1.loc[case1[0] == x, 1])
    x_y = int(case1.loc[case1[0] == x, 2])
    y_x = int(case1.loc[case1[0] == y, 1])
    y_y = int(case1.loc[case1[0] == y, 2])
    if (x == y):
        d = 0
    else:
```

```
        d = abs(x_x-y_x)+abs(x_y-y_y)
    return d

#分别将矩阵写入表格
if __name__ == '__main()__':
    frame = pd.read_excel('result.xlsx', header = None, )
    for i in range(1, 3001):
        for j in range(3001, 3009):
            m = j-3000
            x = case[0][i]
            y = case1[0][m]
            frame[j][i] = Judge2(x, y)#计算1-8

    for i in range(1, 3001):
        for j in range(3009, 3014):
            m = j-3000
            x = case[0][i]
            y = case1[0][m]
            frame[j][i] = Judge3(x,y)#计算9-13

    for i in range(3001, 3014):
        for j in range(i, 3014):
            m = i-3000
            n = j-3000
            x = case1[0][m]
            y = case1[0][n]
            frame[j][i] = Judge4(x, y)#计算复核台距离
frame.to_excel('result.xlsx', index = None, header = None)
```

附录 3：问题 2 代码

```
    close all;
    clc;
    C=xlsread('C:\Users\win\Desktop\建模国赛\2_T01_P.xls');%T001货单数据导入
    NC_max=100;
    m=50;
    Alpha=1;
    Beta=5;
    Rho=0.5;
    Q=1;
    %%第一步：变量初始化
    n=size(C,1);%n表示问题的规模（城市个数）
```

```
D=xlsread('C:\Users\win\Desktop\建模国赛\2_T01_D.xls');%D表示距离矩阵导入
Eta=1./D;        %Eta为启发因子，这里设为距离的倒数
Tau=ones(n,n);   %Tau为信息素矩阵
Tabu=zeros(m,n); %存储并记录路径的生成
NC=1;             %迭代计数器，记录迭代次数
R_best=zeros(NC_max,n);   %各代最佳路线
L_best=inf.*ones(NC_max,1); %各代最佳路线的长度
L_ave=zeros(NC_max,1);    %各代路线的平均长度

while NC<=NC_max     %停止条件之一：达到最大迭代次数，停止
%%第二步：将m只蚂蚁放到n个城市上
Randpos=[]; %随机存取
for i=1:(ceil(m/n))
Randpos=[Randpos,randperm(n)];
end
Tabu(:,1)=(Randpos(1,1:m))'; %此句不太理解？

%%第三步：m只蚂蚁按概率函数选择下一座城市，完成各自的周游
for j=2:n    %所在城市不计算
for i=1:m
visited=Tabu(i,1:(j-1)); %记录已访问的城市，避免重复访问
J=zeros(1,(n-j+1));    %待访问的城市
P=J;                   %待访问城市的选择概率分布
Jc=1;
for k=1:n
if length(find(visited==k))==0 %开始时置0
J(Jc)=k;
Jc=Jc+1;                       %访问的城市个数自加1
end
end
%下面计算待选城市的概率分布
for k=1:length(J)
P(k)=(Tau(visited(end),J(k))^Alpha)*(Eta(visited(end),J(k))^Beta);
end
P=P/(sum(P));
%按概率原则选取下一个城市
Pcum=cumsum(P);  %cumsum，元素累加即求和
Select=find(Pcum>=rand); %若计算的概率大于原来的就选择这条路线
to_visit=J(Select(1));
Tabu(i,j)=to_visit;
end
end
if NC>=2
Tabu(1,:)=R_best(NC-1,:);
```

```
end
%%第四步：记录本次迭代最佳路线
L=zeros(m,1);   %开始距离为0，m*1的列向量
for i=1:m
R=Tabu(i,:);
for j=1:(n-1)
L(i)=L(i)+D(R(j),R(j+1)); %原距离加上第j个城市到第j+1个城市的距离
end
L(i)=L(i)+D(R(1),R(n));  %一轮下来后走过的距离
end
L_best(NC)=min(L);        %最佳距离取最小
pos=find(L==L_best(NC));
R_best(NC,:)=Tabu(pos(1),:); %此轮迭代后的最佳路线
L_ave(NC)=mean(L);        %此轮迭代后的平均距离
NC=NC+1;                   %迭代继续
%%第五步：更新信息素
Delta_Tau=zeros(n,n);     %开始时信息素为n*n的0矩阵
for i=1:m
for j=1:(n-1)
Delta_Tau(Tabu(i,j),Tabu(i,j+1))=Delta_Tau(Tabu(i,j),Tabu(i,j+1))+Q/L(i);
%此次循环在路径（i，j）上的信息素增量
end
Delta_Tau(Tabu(i,n),Tabu(i,1))=Delta_Tau(Tabu(i,n),Tabu(i,1))+Q/L(i);
%此次循环在整个路径上的信息素增量
end
Tau=(1-Rho).*Tau+Delta_Tau; %考虑信息素挥发，更新后的信息素
%%第六步：禁忌表清零
Tabu=zeros(m,n);          %%直到最大迭代次数
end
%%第七步：输出结果
Pos=find(L_best==min(L_best)); %找到最佳路径（非0为真）
Shortest_Route=R_best(Pos(1),:); %最大迭代次数后最佳路径
Shortest_Length=L_best(Pos(1)); %最大迭代次数后最短距离
subplot(1,2,1);               %绘制第一个子图形
%%=========================================================================
%% DrawRoute.m
%% 画路线图的子函数
%%-------------------------------------------------------------------------
%% C Coordinate 节点坐标，由一个N×2的矩阵存储
%% R Route 路线
%%=========================================================================

N=length(R);
scatter(C(:,1),C(:,2));
```

```
    hold on
    plot([C(R(1),1),C(R(N),1)],[C(R(1),2),C(R(N),2)],'g')
    hold on
    for ii=2:N
    plot([C(R(ii-1),1),C(R(ii),1)],[C(R(ii-1),2),C(R(ii),2)],'g')
    hold on
    end
    title('旅行商问题优化结果 ')

    subplot(1,2,2);               %绘制第二个子图形

    plot(0,8000,'r');
    plot(L_best);
    hold on ;                     %保持图形
    plot(L_ave,'r');
    title('平均距离和最短距离') ; %标题

    #根据利用蚁群算法计算出来的松弛问题解
    #再分情况进行讨论，得到原问题的近似解
import os
import pandas as pd
import numpy as np
import xlwt

root = '../Downloads/'
file = os.path.join(root, '问题2_visit.xls')
file0 = os.path.join(root, '2_5.xls')#要遍历图的点的坐标
file1 = os.path.join(root, '2_4.xls')#要遍历图的权重矩阵
lis = []
#读取表格为dataframe
case = pd.read_excel(file , header = None, index = False)
case0 = pd.read_excel(file0 , header = None, index = False)
case1 = pd.read_excel(file1 , header = None, index = False)
root2 = 'C/附件/'
file2 = os.path.join(root2, '附件1: 仓库数据.xlsx')
case2 = pd.read_excel(file2, sheetname = '任务单' , header = None, index = False)

#计算复合台左右索引
index = lis.index(24)#相邻index
left = lis[index-1]
right = lis[index+1]

#计算对应在距离最大矩阵中的行数
#case1是要遍历图的权重矩阵
se_left = case1[0][left]
```

```
se_right = case1[0][right]
#计算这两个货格在距离大矩阵(frame)中行数
y_left = 15*int(se_left[1:4])+(int(se_left[4:])-15)
y_right = 15*int(se_right[1:4])+(int(se_right[4:])-15)

#是否替换
def findmin(y, n):
    mi = 9999999#设置较大
    se = []
    for i in range(3001, 3014):
        if (frame[i][y]<mi):
            mi = frame[i][y]
            se.append(i-3000)
    diff = frame[n+3000][y] - mi
    if (diff<=0):
        print("不用换了，就是：%d\t"%n)
    else:
        print("要换哦，差值是：%d\t"%diff)
    return se[-1], mi

#计算修改后长度
def caldis(index, lis, mi):
    le = len(lis)
    dis = 0
    x = 0
    while(x<le-1):
        #print(case1[lis[index]][lis[index-1]])
        dis += case1[lis[index]][lis[index-1]]
        index -= 1
        x+=1
    dis += mi
    print(dis)
    return dis

#计算货单中商品的个数
def genmer(case2, T):
    mer = []
    for i in range(case2.shape[0]):
        if case2[0][i] == T:
            mer.append(case2[3][i])

    print(mer)
    return mer
```

```
#计算总时间
def caltime(mer, dis):
    time = 30
    time += (dis/1000)/1.5
    for m in mer:
        if(m<3):
            time += m*5
        elif(m>=3):
            time += m*4
    print(time)
    return time


if __name__ == '__main__':
    se1, mi1 = findmin(y_left ,10)
    se2, mi2 = findmin(y_right, 10)
    dis = caldis(index, lis, mi2) #最短长度
    mer = genmer(case2, 'T0001')#货单商品
time = caltime(mer, dis)#最终时间
```

附录 4：问题 3 代码

```
import os
import pandas as pd
import numpy as np
import xlwt

#根据处理过的数据计算两复核台不成圈的情况
file3 = '问题三所有解.xlsx'
lis = []
case3 = pd.read_excel(file3 , header = None, index = False)
root2 = 'C/附件/'
file2 = os.path.join(root2, '附件1: 仓库数据.xlsx')
case2 = pd.read_excel(file2, sheetname = '任务单' , header = None, index = False)

def substract(case3):
    minus = 25500#PH03和FH11之间的距离
    row = case3.shape[0]
    i = 1
    F03_F11_dis = []
    while (i<row):
        media = (case3[1][i]*10**5)-minus
        F03_F11_dis.append(round(media, 6))
```

```
        i += 4
    return F03_F11_dis
#读取生成货单中货物数量
def genmer(case2, T):
    mer = []
    for i in range(case2.shape[0]):
        if case2[0][i] == T:
            mer.append(case2[3][i])

    print(mer)
    return mer
#根据上述计算得到的不同货单时间
T2 = [451600.0, 464100.0, 454400.0]
T3 = [409500.0, 401100.0, 432100.0]
T4 = [439300.0, 438300.0, 463200.0]
T5 = [376000.0, 378100.0, 399000.0]
T6 = [469900.0, 476800.0, 475500.0]
#计算时间
def caltime(mer, dis):
    timeline = []
    for i in dis:
        time = 30
        time += (i/1000)/1.5
        for m in mer:
            if(m<3):
                time += m*5
            elif(m>=3):
                time += m*4
        print(time)
        timeline.append(time)
    mintime = min(timeline)
    index = timeline.index(mintime)
    return index, mintime, timeline
indexT2 , mintimeT2 , timelineT2= caltime(genmer(case2, 'T0002'), T2)
print(indexT2, mintimeT2)
indexT3, mintimeT3, timelineT3 = caltime(genmer(case2, 'T0003'), T3)
indexT4, mintimeT4, timelineT4 = caltime(genmer(case2, 'T0004'), T4)
indexT5, mintimeT5, timelineT5 = caltime(genmer(case2, 'T0005'), T5)
indexT6, mintimeT6, timelineT6 = caltime(genmer(case2, 'T0006'), T6)
index0 = []
for i in range(2, 7):
    index0.append(eval('indexT'+str(i)))
#计算利用率和最小时间mitime
mitime = 0
```

```
for i in range(2,7):
    mitime+= eval('mintimeT'+str(i))
#输入货格编号
def jisuan(azhe):
    return 15*int(azhe[1:4])+(int(azhe[4:])-15)
#主函数
if __name__ == '__main__':
    F03_F11_dis = substract(case3)
    n1 = index0.count(1)
    n2 = index0.count(0)
    F3ratio = ((n1+int(n2/2))*30/mitime)*100
    F11ratio = ((int(n2/2)+1)*30/mitime)*100

r = []
x = 0
while(x<5):
    r0 = 3*x+(1+x)+index0[x]
    r.append(r0)
    x += 1
case4 = pd.read_excel('3_T02_D.xls' , header = None, index = False)
t2p = []
for i in range(2, 30):
    t2p.append(case3[i][r[0]])
t2p0 = []
for i in t2p:
    t2p0.append(case4[i][0])

case5 = pd.read_excel('3_T03_D.xls' , header = None, index = False)
t3p = []
for i in range(2, 30):
    t3p.append(case3[i][r[1]])
print(t3p)
t3p0 = []
for i in t3p:
    if (np.isnan(i)):
        t3p0.append(0)
    else:
        t3p0.append(case5[i][0])
case6 = pd.read_excel('3_T04_D.xls' , header = None, index = False)
print(case6[0][2])
t4p = []
for i in range(2, 30):
    t4p.append(case3[i][r[2]])
print(t4p)
```

```
t4p0 = []
for i in t4p:
    if (np.isnan(i)):
        t4p0.append(0)
    else:
        t4p0.append(case6[i][0])

case7 = pd.read_excel('3_T05_D.xls' , header = None, index = False)
t5p = []
for i in range(2, 30):
    t5p.append(case3[i][r[3]])
print(t5p)
t5p0 = []
for i in t5p:
    if (np.isnan(i)):
        t5p0.append(0)
    else:
        t5p0.append(case7[i][0])

case8 = pd.read_excel('3_T06_D.xls' , header = None, index = False)
t6p = []
for i in range(2, 30):
    t6p.append(case3[i][r[4]])
print(t6p)
t6p0 = []
for i in t6p:
    if (np.isnan(i)):
        t6p0.append(0)
    else:
        t6p0.append(case8[i][0])
to_excel = pd.DataFrame({'T02':t2p0, 'T03':t3p0, 'T04':t4p0, 'T05':t5p0, 'T06':
    t6p0})
to_excel.to_excel('T3.xlsx', index = None, header = True)



    #存入附件表格
    nimd = pd.read_excel('C/附件/附件4: 计算结果.xlsx' ,sheetname = 'Ques3' ,header
         = None, index = False)
    print(nimd)
    book = xlsxwriter.Workbook('中转T30.xlsx', {'constant_memory': True})
    sheet = book.add_worksheet('test')
    for i in range(1, nimd.shape[0]):
        print(nimd[2][i])
```

```
        if (nimd[2][i][0] == 'S'):
            sheet.write(i, 0, jisuan(nimd[2][i]))
        else:
            sheet.write(i, 0, 0)
        sheet.write(0,i ,jisuan(nimd[i][0]))
        for j in range(case2.shape[0]):
            if case2[0][j] == nimd[1][i]:
                if (case2[2][j] == nimd[2][i]):
                    sheet.write(i, 2, case2[3][j])
                #print(case2[3][j])
    book.close()
```

附录 5：问题 4 代码

```
    #将数据写入附件表格
def jisuan(azhe):
    return 15*int(azhe[:-2])+(int(azhe[-2:])-15)
#定义结构体
class worker:
    def __init__(self, worklist):
        self.worklist = worklist
#寻找对应的任务单
def findjob(case4_2):
    worklist = []
    for j in range(9):
        num = []
        for i in range(int((case4_2.shape[0]-1)/2)):
            if (not np.isnan(case4_2[j*2][2*i])):
                com = case4_2[j*2][2*i+1]-case4_2[j*2][2*i]
                #print(com)
                for k in range(case4_3.shape[0]):
                    if (round(com,2) == round(case4_3[2][k],2) and case4_2[j*2+1][2*i
                        ]
                        == case4_3[3][k] and case4_2[j*2+1][2*i+1] == case4_3[4][k]):
                        num.append(case4_3[0][k])
                        #print(num)
        b = worker(num)
        worklist.append(b)
    return worklist
#特殊字符处理
def w(a):
    if int(a/10) == 0:
        return 'T000'+str(a)
    else:
```

```
        return 'T00'+str(a)

def w2(a):
    if int(a/10) == 0:
        return 'FH0'+str(int(a))
    else:
        return 'FH'+str(int(a))

def w3(a):
    if int(a/10) == 0:
        return 'S0000'+str(a)
    elif(int(a/100) == 0):
        return 'S000'+str(a)
    elif(int(a/1000) == 0):
        return 'S00'+str(a)
    elif(int(a/10000) == 0):
        return 'S0'+str(a)
    else:
        return 'S'+str(a)

#写入附件表格
def wi(shuru):
    leng = 0
    for i in range(shuru):
        l = len(worklist[i].worklist)
        for j in range(l):
            h = i+1
            sheet.write(leng, 0, 'P'+str(h))
            sheet.write(leng, 1, w(worklist[i].worklist[j]))
            sheet.write(leng, 2, w2(case4_2[2*i+1][2*j]))
            sheet.write(leng, 3, 3000+case4_2[2*i+1][2*j])
            sheet.write(leng, 4, 0)
            leng+=1
            for k in range(case4_1.shape[0]):
                if case4_1[0][k] == worklist[i].worklist[j]:
                    sheet.write(leng, 0, 'P'+str(h))
                    sheet.write(leng, 1, w(worklist[i].worklist[j]))
                    sheet.write(leng, 2, w3(int(case4_1[1][k])))
                    sheet.write(leng, 3, jisuan(str(case4_1[1][k])))
                    sheet.write(leng, 4, case4_1[2][k])
                    leng += 1
    book.close()

if __name__ == '__main__':
```

```
worklist = findjob(case4_2)
book = xlsxwriter.Workbook('T4/附件T4.xlsx', {'constant_memory': True})
sheet = book.add_worksheet('test')
wi(9)

#数据处理成状态表
import os
import pandas as pd
import numpy as np
import xlwt

file4_0 = 'T4/4_1.xlsx'
case4_0 = pd.read_excel(file4_0 , header = None, index = False)
case4_1 = pd.read_excel('T4/4_xlscheck.xls' , header = None, index = False)
case4_2 = pd.read_excel('T4/4_2_People.xls' , header = None, index = False)
case4_3 = pd.read_excel('T4/中转T4.xlsx' , header = None, index = False)
#计算两个复核台之间距离
def gap(a, b):
    return frame[3001+int(b)][3001+int(a)]
#判断函数
def J(a):
    i = a.index(max(a))
    if ((i<len(a)-1 and a[i+1]==max(a)-1) or a[i-1] == max(a)-1):
        return True
    elif(i == len(a) and a[0] == max[a]-1):
        return True
    else:
        return False

#预处理成表格进行算法实现
def substract0(cas, n):
    mini = 9999999
    re = []
    x = 3
    for k in range(cas.shape[0]):
        i = (n-1)*10+k
        s = (cas[1][i]).split(' ')
        #print(s)
        a = []
        for j in range(3, cas.shape[1]):
            if (cas[j][i]!=0):
                a.append(cas[j][i])
            else:
                break
        print(len(a))
```

```
        if ((cas[2][i] - gap(s[0], s[1]))<mini):
            if (s[0] != s[1] and J(a)):
                re.append(k)
                mini = cas[2][i] - gap(s[0], s[1])
            elif(s[0] == s[1]):
                re.append(k)
                mini = cas[2][i] - gap(s[0], s[1])
    #print(re)
    li = []
    for j in range(3, cas.shape[1]):
        if (cas[j][re[-1]+(n-1)*10]!=0):
            li.append(cas[j][re[-1]+(n-1)*10])
        else:
            break
    sm = (cas[1][re[-1]+(n-1)*10]).split(' ')
    return mini, re[-1], sm, li

#计算每个货单的挑货时间
def genmer(case, T):
    mer = []
    for i in range(case.shape[0]):
        if case[0][i] == T:
            mer.append(case[2][i])

    print(mer)
    return mer
mer = genmer(case4_1, 1)

def calptime(mer):
    time = 30
    for m in mer:
        if(m<3):
            time += m*5
        elif(m>=3):
            time += m*4
    print(time)
    return time
#定义结构体
class merc:
    def __init__(self, s, e, mintime, num, state, li):
        self.s = s#起始点
        self.e = e#终止点
        self.mintime = mintime#最小耗时
        self.num = num#所属货单号
```

```
        self.state = state#状态
        self.li = li#货物编号

#最终计算结果
def ultim(mertime, case4_0):
    merclist = []
    for n in range(1,50):
        m = (n-1)*10
        mini, rec, sm, li= substract0(case4_0[m:m+9], n)
        print(sm)
        mintime = ((mini/1000)/1.5)+mertime[n-1]
        #print((mini/1000)/1.5)
        a = merc(sm[0], sm[1], mintime, n, 0, li)
        #print(sm[0])
        merclist.append(a)
        if (sm[0] != sm[1]):
            b = merc(sm[1], sm[0], mintime, n, 0, li)
            merclist.append(b)
        #print(merclist[0].mintime)
    return merclist
#排序准则
def take(a):
    return a.mintime

if __name__ == '__main__':
    mertime = []
    for i in range(49):
        mertime.append(calptime(genmer(case4_1, i+1)))
    merclist = ultim(mertime, case4_0)
    merclist.sort(key = take)#按照耗时长短进行排序
    book = xlsxwriter.Workbook('T4/中转T4.xlsx', {'constant_memory': True})
    sheet = book.add_worksheet('test')
    for i in range(len(merclist)):
        sheet.write(i, 0, merclist[i].num)
        sheet.write(i, 1, merclist[i].state)
        sheet.write(i, 2, merclist[i].mintime)
        sheet.write(i, 3, merclist[i].s)
        sheet.write(i, 4, merclist[i].e)
        sheet.write(i, 5, merclist[i].li)
book.close()
```