A 题　分拣系统优化问题

摘要

随着电商的不断发展，提高电商公司分拣环节的效率对配送中心整体性能有十分显著的影响。为提高电商公司分拣环节的效率，本文从订单的运送批次、货物的摆放位置及为分拣工指派分拣任务三个方面分别建立模型，对该问题进行了研究。

在问题一中，要求求得最少的运送批次。我们根据约束条件建立了**离散规划模型**，采用**种子算法**进行求解，选用包含品相数量最多的拣选订单作为初始订单，将共同货品种类数作为订单相似度，并根据相似度顺序更新种子订单，不断进行迭代分批以得到最优分批方案。最终得到，当货架数量为 200 时，总批次为 34。在所有批次中订单数不超过 45，不低于 10。结果所得批次较少且货品数量较为平均，有利于提高分拣效率。

在问题二中，题目要求设计货品摆放算法，使得对于任意一批分拣任务中所有订单的拣选距离总和尽量最小。根据题中所给条件建立**离散优化模型**，结合**遗传算法**，使使一个订单中所含商品的位置尽量集中。最终得到所有批次全部订单的拣选距离总和为 120776。

在问题三中，题目要求设计订单指派算法，将分拣任务分配给多个分拣工，使得能够尽快完成任务，且运动距离尽可能均衡。这里我们选择使用**遗传算法**对分拣距离进行分配，是拣选距离尽可能少。该题是一个 **TSP 的变种问题**，要使其能尽快完成任务，就要求每个分拣工所走路程最短，我们采用**蚁群算法**对其优化分配；要使得每个分拣工的运动距离尽可能均衡，就要对其再进行合理分配，从而得到在满足两个条件下的最优解。

关键词：　离散规划、TSP 问题、种子算法、遗传算法、蚁群算法

目录

1 问题重述

1.1 问题背景

某电商分配公司的分拣工需要按任务单分批次分拣出每一个订单中的货品。但是由于成本和场地的限制，货架很难增加。在货架数量保持不变的前提下，设计分批算法，使批次尽可能的小；优化货品摆放位置，提高拣选效率；按批次给多名分拣工分配任务，并使得各分拣工得运动距离尽可能均衡。

分拣环节示意图

图 1: 分拣环节示意图

1.2 问题提出

在问题一中，在每个批次的订单所含货品种类数均不超过 N 的前提下，设计算法为附件中的订单分批次，并使得批次最小。并利用该算法计算当 $N = 200$ 时最少的批次数，给出每批订单数量、货品种类数、分批方案等结果。

在问题二中，基于问题一的批次划分方法，自己设计算法，优化给定批次中所有订单的拣选距离总和。

在问题三中，基于前两问以及题目所给假设，为分拣工分配任务。并给出当 $n = 5$ 时各订单指派结果和每一位分拣工处理订单的顺序。

2 模型假设

- 货架数量有限，并且在不同批次中保持不变。
- 不考虑货架的容积和载重限制，即每个货架能够放置的货品件数没有限制。
- 一个货架中仅放置同一种货品，并且一个货品只能放在一个货架上。
- 任意相邻货架距离相同。
- 分拣工在分拣过程中互相不会干扰，即运动路线不会冲突，也可以同时拣选同一货架上的货品。一位分拣工同一时间只能处理一个订单，即一位分拣工完成一个订单的分拣工作之后，才能开始下一个订单的分拣工作。每位分拣工在自己的分拣任务完成之前，始终匀速运动，忽略从货架上取货所需的时间，忽略订单之间的切换时间，忽略休息时间，且所有分拣工的运动速度相同。结合前一个假设，一位分拣工在处理当前订单时，只要经过该订单中某一货品所在货架至少一次，就视为已经拣选到该货品。拣选到的货品放入拣货筐中，同一订单的货品放入同一个拣货筐中，不

考虑拣货筐容量限制。一个订单的货品全部拣齐后，分拣工将该筐放上传送带，自动转移至核对打包环节，之后可立即原地取用一个新的空的拣货筐。忽略放置拣货筐、取用新筐的时间。在每一批分拣任务开始前，每位分拣工领取到自己当前批次的任务单，需要按照任务单上的订单顺序，依次完成所有订单的分拣工作。同一订单中的不同货品不区分拣货顺序，由分拣工按照最优方式决定。在每一批分拣任务开始时，所有分拣工位于 1 号货架处，同时由 1 号货架出发，开始当前批次的分拣工作。每位分拣工完成一个订单之后，可以立即开始下一个订单的分拣工作，不需要回到 1 号货架。分拣工在拣选完当前批次自己任务单上所有订单的货品后，回到 1 号货架，完成当前批次任务，休息，等待下一批分拣任务开始。

- 某一分拣工分拣完成任务单上所有订单所需时间与其运动总距离呈正比。

3 符号说明

符号	意义
N	货架数量
SR_t	第 t 个货架
n_P	订单总数
O_k	第 k 订单
$S(i)$	货品 i 所在货架的序号
$d(O_k)$	序号为 O_k 的订单的拣选距离
n	总批次数量
O_{seed}	种子订单
n_p	总货品种类数
d_{jk}	第 j 个货架与第 k 个货架之间的距离

4 问题一模型的建立与求解

4.1 问题分析

根据题目要求，每批批次的货品种类数不得大于货架数量，并且批次数量要求尽量小。该问题是一个典型的优化问题。在所有订单的分批过程中，最重要的就是根据某一初始订单来给其他订单分批。有关论文已经指出，订单分批问题可以看作离散优化问题来解决, 并使用种子算法进行优化，可以采用种子算法进行适当简化[1]。通过把相似程度高的订单放在同一个批次中从而用较少的货架完成更多的订单，并且算法简便高效。由于本问中数据量较大，精确算法和启发式算法都很难求得最优解。相比之

下，使用种子算法是很合适的。

4.2 模型建立

根据题目中给出的约束条件和附件中的数据信息建立模型，模型具体如下：

决策变量 我们可以调整一个批次中的订单号，使总批次数尽可能地小。设决策变量为 X_{ik}，意义如下：

$$X_{jk} = \begin{cases} 1, & \text{订单}j\text{分给批次}k \\ 0, & \text{订单}j\text{不分给批次}k \end{cases}$$

B_k 是由第 k 批次的订单组成的集合，P_j 是第 j 个订单，可表示为：

$$B_k = \bigcup_{j=1}^{n_P} \{P_j X_{jk}\}$$

目标函数 设 n 为总批次数量，$|\bigcup B_k|_{B_k}$ 表示所有批次组成的集合中，B_k 子集的个数。要使总批次数最小，则 B_k 子集的个数要最小。目标函数为

$$\min\ n = \left|\bigcup B_k\right|_{B_k}$$

约束条件

货架数量约束 对于任意一批次 B_k ，至少要有一种货品并且货品种类数不超过货架数量 N。定义函数 $s(B_k)$，表示订单 B_k 中的货品品种数：

$$1 \leqslant s(B_k) \leqslant N$$

订单限制 一个订单都必须对应一个批次：

$$\left|\bigcup_{k=1}^{n} B_k\right| = n_P$$

4.2.1 模型总述

以下为订单分批模型

$$\min\ n = \left|\bigcup B_k\right|_{B_k}$$

$$s.t.\begin{cases} 1 \leqslant s(B_k) \leqslant N \\ \left|\bigcup_{k=1}^{f} B_k\right| = n_P \end{cases}$$

4.3 利用种子算法求解模型

我们可以将尽可能多的订单共用一个货架，这样就可以使用有限的货架，来满足尽可能多的订单。

种子算法中“种子”的含义就是第一个被选出来的订单，该订单作为每一批次的初始订单。之后再根据一定的相似度规则选取其他订单加入到当前批次，并与原来的种子订单一起构成新的种子订单。

种子算法的主要步骤如下：

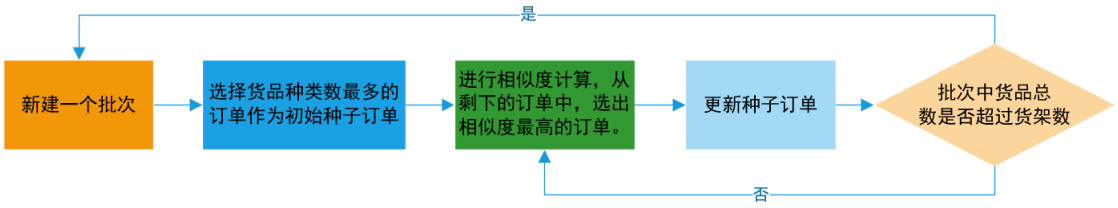
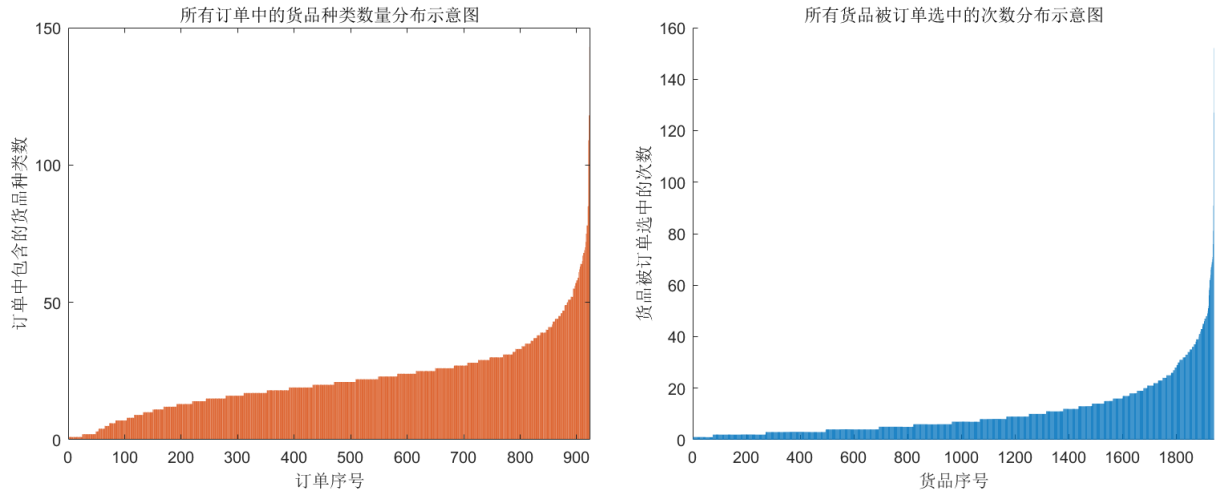


图 2: 种子算法思路

采用种子算法最重要的环节就是初始种子的选取规则和相似度的度量方式。图 3(a)说明订单之间的数目存在差别，图 3(b)说明订单之间存在相似性（有一部分货品被重复选择）我们要选取一个种子订单，并以批次中货物种类不超过 N 为限制，不断地将剩余满足相似度条件的订单加入该批次中，直到不满足分批条件或者全部订单已经完成分批。



(a) (b)

图 3

4.3.1 初始种子的选取

初始种子的选取方式有如下几种：

1. 包含品项数量最多的拣选订单;

2. 采用随机方法，从所有订单中任意选取一个;

3. 包含品项数量最少的拣选订单;

考虑到题目要求使批次总是尽量小，一个订单有更多种类的货品时，满足相似度度量的订单可能就越多，货架的利用率就越高。我们决定将包含最多货品种类数的订单作为初始订单。选择完成后将种子订单从现有订单中移除。

4.3.2 相似度度量的选取

相似度度量是衡量订单相似度的指标。在本问题中，我们决定将共同货品种类数与不同的货品数的比值作为订单相似度 Sim。显然共同货品种类数占该订单总货品种类数越多，相似度越高。设 t_k 为第 k 个订单（不包括种子订单）中的货品种类数，t_{sk} 为种子订单和第 k 个订单具有的相同货品种类数。为了综合考虑不同和相似的订单，对订单相似度作一定的改变：

$$Sim = \frac{t_{sk} + \alpha}{t_k + \beta} \times 100\%$$

上式中的 α,β 是可变参数，分别是衡量相同的货品和不同的货品数对相似度的影响权重。α 越大不同货品数对相似度影响程度更大，β 越大相同货品数对相似度的影响更大。

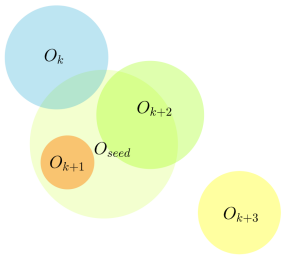


图 4: 种子订单的选取

根据上述求解的订单相似度，将剩余订单按照相似度从大到小的顺序排序。

4.3.3 更新种子订单

按照相似度顺序，依次将剩余订单加入到当前批次中。每加入一个订单，就将该订单和原来的种子订单作为新的种子订单，重复种子订单的选取、相似度计算及排序、加入当前批次和更新种子订单操作，直到不满足分批条件为止，该批次的分批完成，进入下一批次的分批操作。

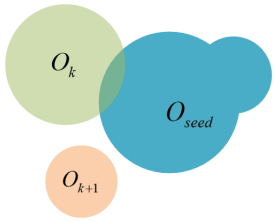


图 5: 更新种子订单

根据以上操作不断地进行迭代分批之后，就可以原始分批方案。

4.3.4 求解结果

根据上述算法，我们求得当货架数量 $N = 200$ 时总批次数 n 为 38，具体分批结果如下图所示。每批订单数量、货品品种数等结果见附件 (result1.csv)。

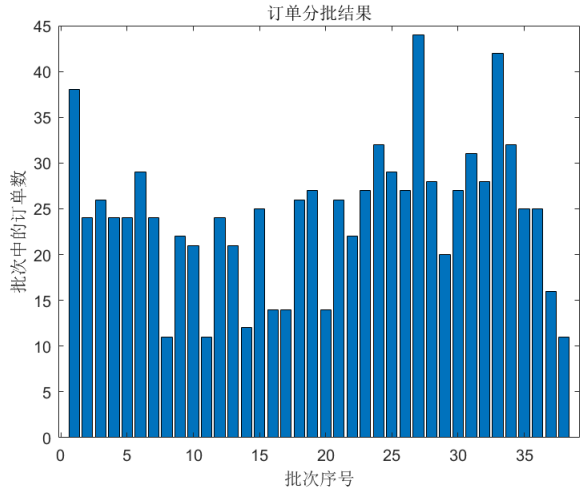


图 6: 分批方案结果

货架利用率如下图所示，各个批次中的货架数接近货架数量 N，说明货架的利用的很充分。

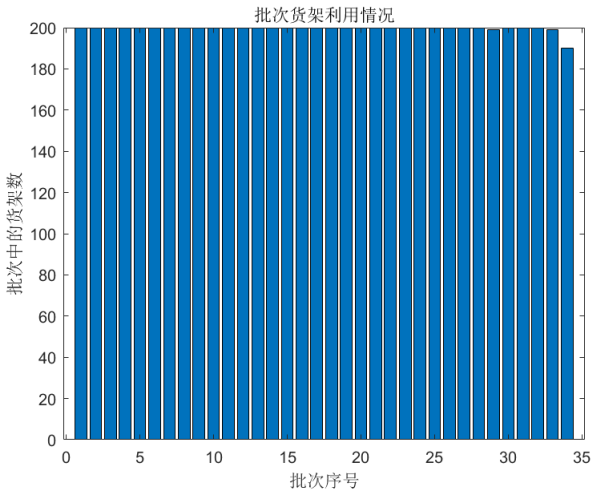


图 7: 货架利用率

4.4 结果分析

根据图 6，在所有批次中，订单数不超过 45，不低于 10。订单数超过 35 和订单数低于 15 的批次数量之和为 10，约占总批次数的 25%。其余批次的订单数分布较为均匀，大致在区间 $[20, 30]$ 内。批次数量较少，并且批次中货品数量较为平均，可以提高分拣效率，为问题二和问题三的求解打下基础。

5 问题二模型的建立与求解

5.1 问题分析

根据题目要求，我们需要设计货品摆放算法，使得某一批次所有订单的货品摆放尽量集中，货架序号已经确定，一个货架只能摆放同一种货品。在处理本问题的过程中，我们只考虑拣选距离，不考虑分拣工具体操作情况。对于问题的求解，我们决定结合遗传算法来求解。在本问题中，决策变量是某一品种的货物，目标函数是使给定批次的所有订单的拣选距离之和最小，并且存在货品摆放位置约束。

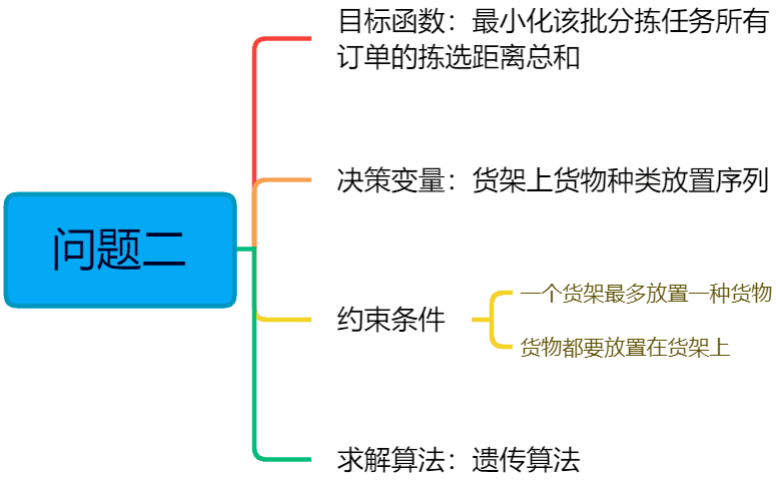


图 8: 问题二思路图

5.2 模型建立

决策变量 我们需要设计一个批次中的每种货品对应的货架序号，减小给定批次中所有订单的拣选距离之和。在本问题中，决策变量是货品 i 所在货架的序号 $S(i)$。

目标函数 定义序号为 O_{jk} 订单的拣选距离为 $d(O_{jk})$，即该订单中货品所在货架序号的最大值与最小值之差：

$$d(O_{jk}) = \max_{i \in O_{jk}} S(i) - \min_{i \in O_{jk}} S(i)$$

根据题目要求，该批分拣任务 B_j 所有批次的拣选距离总和要尽量小。

$$\min\ f = \sum_{k=1}^{n_j} d(O_{jk})$$

其中 n_j 为第 j 批次中的订单个数。

约束条件

货架约束 对于任意一个货架 SR_t ，其摆放的货品只能为同一类。设 P_{it} 表示 SR_t 上第 i 种货品，其中 $1 \leqslant i \leqslant n_p$，且

$$P_{it} = \begin{cases} 1, & \text{if 货物在}SR_t\text{上} \\ 0, & \text{if 货物不在}SR_t\text{上} \end{cases}$$

则约束条件可以表示为：

$$\sum_{i=1}^{n_p} P_{it} = 1$$

货品约束 每一种货品只能摆放到一个货架上，即：

$$\sum_{t=1}^{N} P_{it} = 1$$

货架数量约束 货品 i 对应的货架号 $S(i)$ 不能大于货架数量 N，即

$$1 \leqslant S(i) \leqslant N$$

模型总述 以下为货品摆放位置模型：

$$\min\ f = \sum_{1}^{n_j} d(O_k)$$

$$s.t. \begin{cases} \sum_{i=1}^{n_p} P_{it} = 1 \\ \sum_{t=1}^{N} P_{it} = 1 \\ 1 \leqslant S(i) \leqslant N \end{cases}$$

5.3 遗传算法求解模型的步骤

利用遗传算法求解模型的步骤如下图所示：

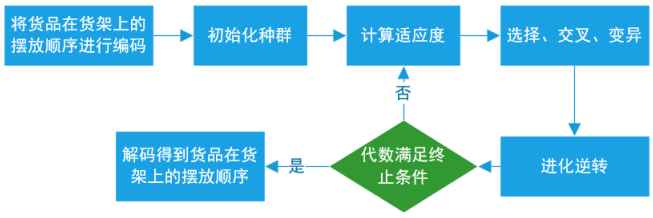


图 9: 遗传算法流程图

步骤 1 产生初始种群，每个染色体由 N 个基因构成，其中每一段对应货架的编号，评价各个染色体的适应度；

步骤 2 判断是否达到最大迭代次数，若达到则搜索结束，输出结果；否则，继续执行；

步骤 3 根据适应度值使用轮盘赌法从旧群体选择个体到新群体中；

步骤 4 染色体以概率 P_c 进行交叉操作；

步骤 5 染色体以概率 P_m 进行变异操作；

步骤 6 在染色体中随机选取两个位置，将其基因即货架的编号进行逆转操作，接受能使适应度值提高的逆转操作；

步骤 7 返回步骤 2 并进行判断。

5.3.1 求解结果

各批分拣任务的拣选距离总和如下图所示：

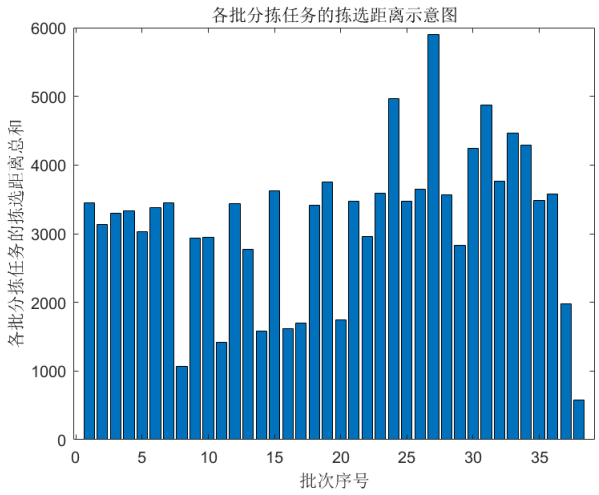


图 10: 各批分拣任务的拣选距离示意图

各批分拣任务的拣选距离总和，以及所有批次全部订单的拣选距离总和，完整原始货品摆放方案见附件 (result2.csv)。

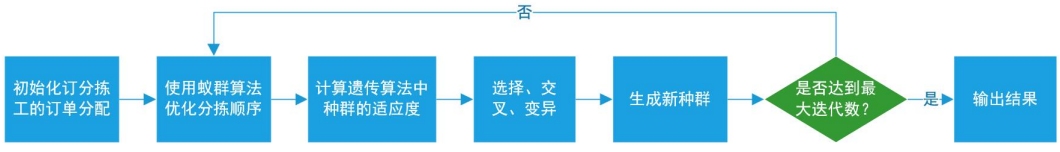
5.4 结果分析

根据图 10，大部分批次的拣选距离总和在区间 $[3000, 4000]$ 内，分布较为均匀，间接说明了问题一的分批方式有合理性。

6 问题三模型的建立与求解

6.1 问题分析

根据题目要求和假设，对于每批分拣任务，使用遗传算法将其分配给 n 位分拣工，使得尽快完成任务并且使运动距离尽可能均衡。在问题求解过程中，我们采用蚁群算法对已经分配好的订单优化分拣顺序[2]。



遗传算法的适应的函数使目标函数，其余步骤在问题二中已经给出，这里不在赘述。

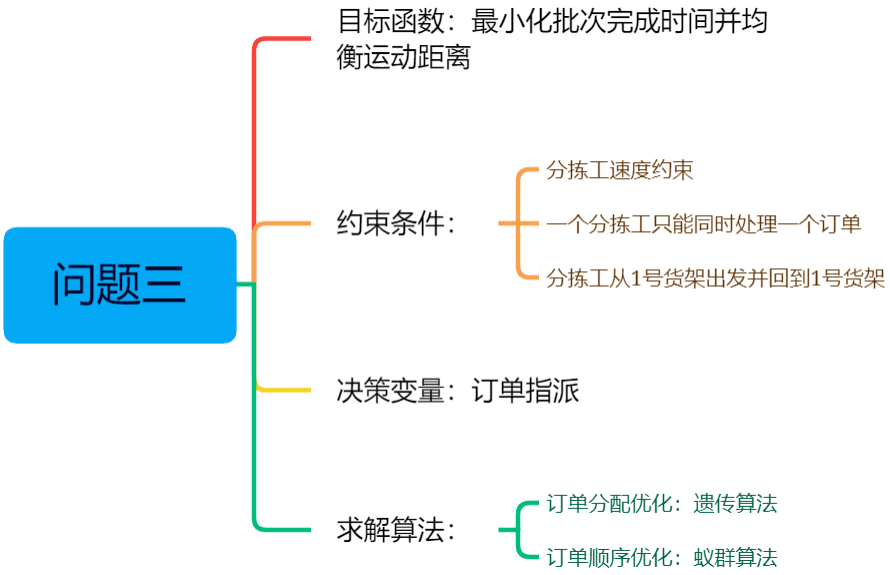


图 11: 问题三思路图

6.2 模型建立

这一问要使得各分拣工在各批次中能尽快完成任务，我们不仅要考虑到挑选每个订单时分拣工所要走的距离，也要考虑到两个订单之间的最短距离。为此，我们建立如下模型：

决策变量 我们需要将各个批次的订单合理分配给每个分拣工，以达到每个分拣工都能尽快完成任务的目的。在本问题中，我们仍选用货品 i 所在的货架序号 $S(i)$ 为决策变量。

目标函数 定义序号为 O_k 的订单与序号为 O_{k+q} 订单的最短距离为 $L(O_k, O_{k+q})$，

$$L(O_k, O_{k+q}) = \min\left\{\left[\max_{i\in O_k} S(i) - \max_{i\in O_{k+q}} S(i)\right], \left[\max_{i\in O_k} S(i) - \min_{i\in O_{k+q}} S(i)\right]\right\}$$

由第二问得序号为 O_k 订单得拣选距离为 $d(O_k)$，根据题目要求，要使分拣工在 B_j 批次中能尽快完成任务，则第 i 个分拣工的分拣距离

$$\min f_i = \sum_{i=1}^{n_j} [d(O_k) + \min\ L(O_k, O_{k+q})]$$

n_j 为第 j 批次地订单个数。

定义一个函数 $SQ(\sum_1^{n_w} f_i)$，计算所有分拣工的分拣距离的方差，并使其最小：

$$\min\ SQ(\sum_{i=1}^{n_w} f_i)$$

约束条件

货架数量约束 本约束为问题二中所有约束。

订单约束 对于任意一个订单 O_k，其只能分配给一个分拣工，分拣工总数为 n_w。设 $G_{ki}(1 \leqslant i \leqslant n_w)$ 是表示订单 O_k 分配给第 i 个分拣工，

$$G_{ki} = \begin{cases} 1, & \text{订单}O_k\text{分配给第}i\text{个分拣工} \\ 0, & \text{订单没有分配给第}i\text{个分拣工} \end{cases}$$

则约束条件可以表示为：

$$\sum_{i=1}^{n_w} G_{ki} = 1$$

模型总述

$$\min\ f_i = \sum_{i=1}^{n_j} [d(O_k) + \min\ L(O_k, O_{k+q})]$$

$$\min\ SQ(\sum_{1}^{n_w} f_i)$$

$$s.t. \begin{cases} \sum_{i=1}^{n_w} G_{ki} = 1 \\ \sum_{i=1}^{n_p} P_{it} = 1 \\ \sum_{t=1}^{N} P_{it} = 1 \\ 1 \leqslant S(i) \leqslant N \end{cases}$$

6.3 利用蚁群算法优化模型

以下为利用蚁群算法求解模型的大致思路。首先，对相关参数进行初始化。我们根据遗传算法得到分拣工分配的订单，之后用蚁群算法优化分拣路线。我们保证分拣工能尽快完成任务即所走路程最短。

6.3.1 算法步骤

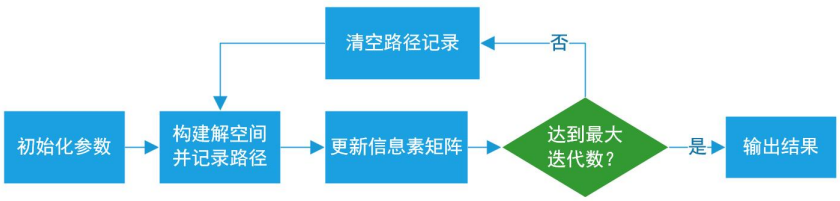


图 12: 蚁群算法解决 TSP 问题基本步骤

初始化参数 在计算之初，需要对相关参数进行初始化，记蚁群规模，也就是蚁群数量为 m，信息素重要程度因子 α，启发函数重要程度因子 β，信息素挥发因子 ρ，信息素释放总量 Q，最大迭代次数 $iter\_max$。

构建解空间 我们将蚂蚁行走路径作为待优化问题的可行解，整个蚂蚁群体的所有路径构成优化问题的解空间。根据订单首端及末端货品所在的货架序号，计算两个订单之间的相互距离，从而得到对称

的距离矩阵。

更新信息素 在整个算法运行的过程中，路径较短的蚂蚁释放的信息素更多。随着时间的推移，较短路径的信息素逐渐变多，选择这条路径的蚂蚁也会变多。在计算各个蚂蚁经过的路径长度之后，实时更新每两个订单间的信息素浓度，并记录当前迭代次数中的最优解。经过循环迭代，记录下最优的分拣工处理订单的顺序及每个分拣工要走的路程。

7 模型的总结与评价

7.1 模型优点

对于问题一 此问是一个典型的优化问题，该问题解空间大，使用精确算法和遗传算法都很难在短时间内求解，而种子算法实现简单，效果良好，可以较好解决该问题。

对于问题二 此问根据题目中给出“同一订单所含货品的位置越集中，移动距离越短，拣选效率越高”的这一条件。此问题规模较小，遗传算法可以有效地求解本问题。根据问题一所得的结果，结合遗传算法来求解，尽量使一个订单中所含商品的位置集中，给求解带来优化。

对于问题三 此问不仅要求对于每批任务分建工可以尽快完成，而且要求了使每个分拣工运动距离尽可能相等，我们在采用指派订单使每个分拣工走的路程尽量均衡的情况下又采用了优化拣选路径，使每个分拣工所运动距离尽可能短，以得到最符合题意的最终结果。

7.2 模型缺点

对于问题一 在初始种子订单的选取方面具有随机性，且初始种子订单对分批的影响较大，算法独立性较差。

对于问题二 此问采用遗传算法进行求解，算法的搜索速度比较慢，要得到精确的解需要较多的训练时间，且算法对初始种群的选择有一定的依赖性。

对于问题三 此问因有要使每个分拣工的运动距离尽可能相等和尽快完成任务两个要求，难以确定其系数。

7.3 模型推广

本文为解决电商公司的订单分批环节的效率问题，基于种子算法、遗传算法及蚁群算法等建立了订单分批及分拣的离散优化模型，为电商公司提供了合理的方案。

本文巧妙运用多种智能算法，充分利用其高效的优化性能及易于与其他方法结合的优点，在解决难以用精确算法计算的目标优化问题方面有较好的前景。

参考文献

[1] 王占磊. 配送中心订单分批及拣选路径优化问题研究 [D]. 吉林大学, 2013.

[2] 宁春林, 田国会, 尹建芹, 等. Max-Min 蚁群算法在固定货架拣选路径优化中的应用 [J]. 山东大学学报：工学版, 2003, 33(6): 5.

附录

ACO.m

```
function [Dis,Seq] = ACO(Node_Work)
% 此函数用于优化分拣工处理订单的顺序
% 输入订单集合的左右位置
% 输出分拣订单的序列和移动距离， Seq不是真正的订单序列，外部调用时需要进行映射
Order_Number = size(Node_Work,1);
%% 特殊情况处理
if Order_Number==1
    Dis = (Node_Work(1,2)-1)*2;
    Seq = 1;
    return;
end
%% 蚁群算法
%% 初始化参数
n = Order_Number;                    % 解向量长度
m = 20;                              % 蚂蚁数量
alpha = 1;                           % 信息素重要程度因子
beta = 5;                            % 启发函数重要程度因子
rho = 0.1;                           % 信息素挥发因子
Q = 1;                               % 常系数
% Eta = 1./D;                          % 启发函数
Tau = ones(n,n);                     % 信息素矩阵 HINT
Table = zeros(m,n);                  % 路径记录表,(Ant,route)
iter = 1;                            % 迭代次数初值
iter_max = 50;                      % 最大迭代次数
Route_best = zeros(iter_max,n);      % 各代最佳路径
Length_best = zeros(iter_max,1);     % 各代最佳路径的长度
Length_ave = zeros(iter_max,1);      % 各代路径的平均长度
Ant_Now = ones(m,1);                 % 记录蚂蚁在订单的哪一端,1左2右
Length = zeros(m,1);

%% 迭代寻找最佳路径
while iter <= iter_max
    % 随机产生各个蚂蚁的起点城市
    start = zeros(m,1);
    for i = 1:m
        temp = randperm(n);
        start(i) = temp(1);
        Ant_Now(i) = 2; %开始都应在右边
        Length(i) = Node_Work(start(i),2)-1;
    end
    Table(:,1) = start;
```

```
    % 构建解空间
    citys_index = 1:n;
    % 逐个蚂蚁路径选择
    for i = 1:m
        % 逐个城市路径选择
        for j = 2:n
            tabu = Table(i,1:(j - 1));           % 已访问的城市集合, 1:(j-1)是已经到过
                的城市
            allow_index =  ismember(citys_index,tabu);
            allow = citys_index(allow_index);  % 待访问的城市集合
            P = allow;
            Ant_temp = ones(length(allow),1);
            leng = zeros(length(allow),1);
            % 计算城市间转移概率, 距离计算方式需要修改
            for k = 1:length(allow)
                left = Node_Work(allow(k),1)-Node_Work(tabu(end),Ant_Now(i));
                left = abs(left);
                right = Node_Work(allow(k),2)-Node_Work(tabu(end),Ant_Now(i));
                right = abs(right);
                leng(k) = min(left,right);   %缺少一个分类!!!!!!!!!!!!!!!
                if leng(k)==right
                    Ant_temp(k) = 2;
                end
                Eta = 1/(leng(k)+1); % 新定义的启发函数,+1是为了防止为0
                P(k) = Tau(tabu(end),allow(k))^alpha * Eta^beta;
            end
            P = P/sum(P);
            % 轮盘赌法选择下一个访问城市
            Pc = cumsum(P);
            target_index = find(Pc >= rand);
            target = allow(target_index(1));
            Table(i,j) = target;
            Ant_Now(i) = Ant_temp(target_index(1));
            if Ant_Now(i)==1
                Ant_Now(i)=2;
            else
                Ant_Now(i) = 1;
            end
            Length(i) = Length(i)+leng(target_index(1)); %在过程中计算距离
        end
        if j==n
            Length(i) = Length(i)+Node_Work(target,Ant_Now(i))-1;
        end
    end
    %% 计算最短路径距离及平均距离
    if iter == 1
```

```
87          [min_Length,min_index] = min(Length);
88          Length_best(iter) = min_Length;
89          Length_ave(iter) = mean(Length);
90          Route_best(iter,:) = Table(min_index,:);
91      else
92          [min_Length,min_index] = min(Length);
93          Length_best(iter) = min(Length_best(iter - 1),min_Length);
94          Length_ave(iter) = mean(Length);
95          if Length_best(iter) == min_Length
96              Route_best(iter,:) = Table(min_index,:);
97          else
98              Route_best(iter,:) = Route_best((iter-1),:);
99          end
100     end
101     % 更新信息素
102     Delta_Tau = zeros(n,n);
103     % 逐个蚂蚁计算
104     for i = 1:m
105         % 逐个城市计算
106         for j = 1:(n - 1)
107             Delta_Tau(Table(i,j),Table(i,j+1)) = Delta_Tau(Table(i,j),Table(i,j+1)) + Q
                    /Length(i);
108         end
109     end
110     Tau = (1-rho) * Tau + Delta_Tau;
111     % 迭代次数加1，清空路径记录表
112     iter = iter + 1;
113     Table = zeros(m,n);
114 end
115
116 %% 结果显示
117 [Shortest_Length,index] = min(Length_best);
118 Shortest_Route = Route_best(index,:);
119 % disp(['最短距离:' num2str(Shortest_Length)]);
120 % disp(['最短路径:' num2str(Shortest_Route)]);
121
122 % Dis需要加上固定距离
123 Dis = Shortest_Length;
124 Dis = Dis + sum(Node_Work(:,2)-Node_Work(:,1));
125 Seq = Shortest_Route;
126
127 %% 绘图
128 % plot(1:iter_max,Length_best,'b',1:iter_max,Length_ave,'r:')
129 % legend('最短距离','平均距离')
130 % xlabel('迭代次数')
131 % ylabel('距离')
```

```
% title('各代最短距离与平均距离对比')
end
```

Data_Progress.m

```
    clear;
clc;
% 将订单转化为0-1向量
% 导入数据
load ItemNo.mat
load OrderNo.mat
load Quantity.mat
% 货品类别数
Item = max(ItemNo);
% 订单总数
Order = OrderNo(end);
% 建立存储向量
Order_Item = zeros(Order,Item); %存储种类
Order_Item_Quantity = zeros(Order,Item); %存储数量
% 数据集长度
len = length(ItemNo);
% 开始进行分类
row = 1;
col = 0;
for i=1:len
    row = OrderNo(i);
    col = ItemNo(i);
    Order_Item(row,col) = 1;
    Order_Item_Quantity(row,col) = Quantity(i);
end

% 记录每个订单的货品总数
Number = sum(Order_Item,2);
hold on;
xlabel('订单序号');
ylabel('订单中包含的货品种类数');
title('所有订单中的货品种类数量分布示意图');
% 展示订单货品数的分布特征
New = sort(Number);
bar(New);


% % 记录每个货品被选中的次数
% Number = sum(Order_Item,1);
```

```
% hold on;
% xlabel('货品序号');
% ylabel('货品被订单选中的次数');
% title('所有货品被订单选中的次数分布示意图');
% % 展示订单货品数的分布特征
% [New,temp] = sort(Number);
% bar(New);
% temp(end)


save('Order_Item.mat','Order_Item');
save('Order_Item_Quantity.mat','Order_Item_Quantity');
save('Number.mat','Number');
```

Data_Upload1.m

```
    clear;
clc;
% 加载数据
load Order_Item.mat;
load Batch_total.mat;

[Order, ]=size(Order_Item);
filename='result1.csv';%.csv可以更改为.txt等
fid=fopen(filename,'w');
fprintf(fid,'OrderNo ,GroupNo\n');%   ','是分隔符
for i=1:Order
    % 寻找对应的批次
    temp = Batch_total(i,:);
    Batch_Order = find(temp==1);
    % 判断Order的位数
    if i<10
        fprintf(fid,'D%d%d%d%d,%s\n',0,0,0,i,num2str(Batch_Order));%
    elseif i<100
        fprintf(fid,'D%d%d%s,%s\n',0,0,num2str(i),num2str(Batch_Order));%
    else
        fprintf(fid,'D%d%s,%s\n',0,num2str(i),num2str(Batch_Order));%
    end
end
```

Data_Upload2.m

```
clear;
clc;
load Order_Item.mat
load Batch_total.mat
load Seq_total.mat

% ItemNo,GroupNo,ShelfNo

[Order,Item]=size(Order_Item);
Batch =size(Batch_total,2);
filename='result2.csv';
fid=fopen(filename,'w');
fprintf(fid,'ItemNo, GroupNo, ShelfNo\n');
for i=1:Item
    % 寻找货品对应的所有批次
    for j=1:Batch
        if find(Seq_total(j,:)==i)
            Group = j;
            Shelf = find(Seq_total(j,:)==i);
            % 判断Item的位数
            if i<10
                fprintf(fid,'P%d%d%d%d,%s,%s\n',0,0,0,i,num2str(Group),num2str(Shelf));
                    %
            elseif i<100
                fprintf(fid,'P%d%d%s,%s,%s\n',0,0,num2str(i),num2str(Group),num2str(
                    Shelf));%
            elseif i<1000
                fprintf(fid,'P%d%s,%s,%s\n',0,num2str(i),num2str(Group),num2str(Shelf
                    ));%
            else
                fprintf(fid,'P%s,%s,%s\n',num2str(i),num2str(Group),num2str(Shelf));%
            end
        else
            continue;
        end
    end
end
```

Data_Upload3.m

```
    clear;
clc;
load Order_Group_Work_Time.mat

```

```
% OrderNo,GroupNo,WorkerNo,TaskNo

[Order, ]=size(Order_Group_Work_Time);
filename='result3.csv';
fid=fopen(filename,'w');
fprintf(fid,'OrderNo, GroupNo, WorkerNo, TaskNo\n');
for i=1:Order
    % 判断Order的位数
    GroupN = Order_Group_Work_Time(i,1);
    WorkerN = Order_Group_Work_Time(i,2);
    TaskN = Order_Group_Work_Time(i,3);
    if i<10
        fprintf(fid,'D%d%d%d%d,%s,%s,%s\n',0,0,0,i,num2str(GroupN),num2str(WorkerN),
            num2str(TaskN) ) ;%
    elseif i<100
        fprintf(fid,'D%d%d%s,%s,%s,%s\n',0,0,num2str(i),num2str(GroupN),num2str(WorkerN
            ),num2str(TaskN));%
    else
        fprintf(fid,'D%d%s,%s,%s,%s\n',0,num2str(i),num2str(GroupN),num2str(WorkerN),
            num2str(TaskN));%
    end
end
```

GA.m

```
    function [Seq,Dis] = GA(Order_Index,Order_Item,Goods,Item_Real)
    NIND=60;        %种群大小
    MAXGEN=400;
    Pc=0.9;          %交叉概率
    Pm=0.05;         %变异概率
    GGAP=0.9;        %代沟(Generation gap)
    N=Goods;        %解向量的长度
    %% 初始化种群
    Chrom=InitPop(NIND,N);
    %% 优化
    % 返回值
    Seq = [];
    Dis = Inf;
    gen=0;
    ObjV=PathLength(Chrom,Order_Index,Order_Item,Item_Real);  %计算路线长度
    [preObjV,Min_index]=min(ObjV);
    Seq = Chrom(Min_index,:);
    Dis = preObjV;
    while gen<MAXGEN
```

```
        %% 计算适应度
        ObjV=PathLength(Chrom,Order_Index,Order_Item,Item_Real);  %计算路线长度
        preObjV=min(ObjV);
        if preObjV<Dis
            Seq = Chrom(Min_index,:);
            Dis = preObjV;
        end
        FitnV=Fitness(ObjV);
        %% 选择
        SelCh=Select(Chrom,FitnV,GGAP);
        %% 交叉操作
        SelCh=Recombin(SelCh,Pc);
        %% 变异
        SelCh=Mutate(SelCh,Pm);
        %% 逆转操作
        SelCh=Reverse(SelCh,Order_Index,Order_Item,Item_Real);
        %% 重插入子代的新种群
        Chrom=Reins(Chrom,SelCh,ObjV);
        %% 更新迭代次数
        gen=gen+1 ;
    end
    disp(['最短距离:' num2str(Dis)]);
    % disp(['最短路径:' num2str([Seq])]);
    end

    %% 初始化种群
    %输入:
    % NIND: 种群大小
    % N:    个体染色体长度（这里为城市的个数）
    %输出:
    %初始种群
    function Chrom=InitPop(NIND,N)
    Chrom=zeros(NIND,N);%用于存储种群
    for i=1:NIND
        Chrom(i,:)=randperm(N);%随机生成初始种群
    end
    end

    %% 适配值函数
    %输入:
    %个体的长度（TSP的距离）
    %输出:
    %个体的适应度值
    function FitnV=Fitness(len)
    FitnV=1./len;
    end
```

```
66
67      function SelCh = Cross(SelCh,Pc)
68      %% 交叉操作
69      % 输入
70      %SelCh  被选择的个体
71      %Pc     交叉概率
72      %输出:
73      % SelCh 交叉后的个体
74      NSel=size(SelCh,1);
75      for i=1:2:NSel-mod(NSel,2)
76          if Pc>=rand %交叉概率Pc
77              [SelCh(i,:),SelCh(i+1,:)]=intercross(SelCh(i,:),SelCh(i+1,:));
78          end
79      end
80
81      %输入:
82      %a和b为两个待交叉的个体
83      %输出:
84      %a和b为交叉后得到的两个个体
85      end
86
87      function [a1,b1]=intercross(a,b)
88      % 随机选取两个端点
89      len = length(a);
90      temp = randperm(len);
91      m1 = temp(1);
92      m2 = temp(2);
93      Min = min(m1,m2);
94      Max = max(m1,m2);
95      % 对两个个体进行交叉
96      temp_a = a(Min:Max);
97      temp_b = b(Min:Max);
98      a(Min:Max) = temp_b;
99      b(Min:Max) = temp_a;
100     a1 = a;
101     b1 = b;
102     end
103
104     %% 变异操作
105     %输入:
106     %SelCh  被选择的个体
107     %Pm     变异概率
108     %输出:
109     % SelCh 变异后的个体
110     function SelCh=Mutate(SelCh,Pm)
111     [NSel,L]=size(SelCh);
```

```
    for i=1:NSel
        if Pm>=rand
            R=randperm(L);
            SelCh(i,R(1:2))=SelCh(i,R(2:-1:1));
        end
    end
    end

    %% 计算各个体的路径长度
    % 输入:
    % Order_Index 批次中真正的订单号
    % Order_Item 订单信息
    % Item_Real 批次中所含的货号
    % Chrom 个体的轨迹
    function len=PathLength(Chrom,Order_Index,Order_Item,Item_Real)
    NIND=size(Chrom,1);
    len=zeros(NIND,1);

    for i = 1:NIND
        p=Chrom(i,:);
        % 解码计算
        Goods_Seq = Item_Real(p);     %真正的货架上的货号，解向量
        for k=1:length(Order_Index)
            % 查找一个订单中的移动距离
            Order = Order_Index(k);  %真正的订单号
            temp = Order_Item(Order,:);
            Index_temp = find(temp =0); %订单中的真正货号
            % 寻找相同的订单
            minmax = ismember(Goods_Seq,Index_temp); %logical数组
            leng = sum(minmax);
            if leng==1
                Min = 1;
                Max = 1;
            else
                Min = find(minmax==1,1,'first');
                Max = find(minmax==1,1,'last');
            end
            len(i) = len(i) + Max-Min;
        end
    end
    end

    %% 交叉操作
    % 输入
    %SelCh  被选择的个体
    %Pc     交叉概率
```

```
    %输出:
    % SelCh 交叉后的个体
    function SelCh=Recombin(SelCh,Pc)
    NSel=size(SelCh,1);
    for i=1:2:NSel-mod(NSel,2)
        if Pc>=rand %交叉概率Pc
            [SelCh(i,:),SelCh(i+1,:)]=intercross(SelCh(i,:),SelCh(i+1,:));
        end
    end

    %输入:
    %a和b为两个待交叉的个体
    %输出:
    %a和b为交叉后得到的两个个体
    end

    function [a,b]=intercross(a,b)
    L=length(a);
    r1=randsrc(1,1,[1:L]);
    r2=randsrc(1,1,[1:L]);
    if r1 =r2
        a0=a;b0=b;
        s=min([r1,r2]);
        e=max([r1,r2]);
        for i=s:e
            a1=a;b1=b;
            a(i)=b0(i);
            b(i)=a0(i);
            x=find(a==a(i));
            y=find(b==b(i));
            i1=x(x =i);
            i2=y(y =i);
            if  isempty(i1)
                a(i1)=a1(i);
            end
            if  isempty(i2)
                b(i2)=b1(i);
            end
        end
    end
    end

    %% 重插入子代的新种群
    %输入:
    %Chrom  父代的种群
    %SelCh  子代种群
```

```
    %ObjV   父代适应度
    %输出
    % Chrom  组合父代与子代后得到的新种群
    function Chrom=Reins(Chrom,SelCh,ObjV)
    NIND=size(Chrom,1);
    NSel=size(SelCh,1);
    [ ,index]=sort(ObjV);
    Chrom=[Chrom(index(1:NIND-NSel),:);SelCh];
    end

    %% 进化逆转函数
    %输入
    %SelCh 被选择的个体
    %输出
    %SelCh  进化逆转后的个体
    function SelCh=Reverse(SelCh,Batch1,Order_Item,Item_Real)
    [row,col]=size(SelCh);
    ObjV=PathLength(SelCh,Batch1,Order_Item,Item_Real);  %计算路径长度
    SelCh1=SelCh;
    for i=1:row
        r1=randsrc(1,1,[1:col]);
        r2=randsrc(1,1,[1:col]);
        mininverse=min([r1 r2]);
        maxinverse=max([r1 r2]);
        SelCh1(i,mininverse:maxinverse)=SelCh1(i,maxinverse:-1:mininverse);
    end
    ObjV1=PathLength(SelCh1,Batch1,Order_Item,Item_Real);  %计算路径长度
    index=ObjV1<ObjV;
    SelCh(index,:)=SelCh1(index,:);
    end

    %% 选择操作
    %输入
    %Chrom 种群
    %FitnV 适应度值
    %GGAP: 选择概率
    %输出
    %SelCh  被选择的个体
    function SelCh=Select(Chrom,FitnV,GGAP)
    NIND=size(Chrom,1);
    NSel=max(floor(NIND*GGAP+.5),2);
    ChrIx=Sus(FitnV,NSel);
    SelCh=Chrom(ChrIx,:);
    end

    % 输入:
```

```
    %FitnV   个体的适应度值
    %Nsel    被选择个体的数目
    % 输出:
    %NewChrIx  被选择个体的索引号
    function NewChrIx = Sus(FitnV,Nsel)

    % Identify the population size (Nind)
    [Nind, ] = size(FitnV);

    % Perform stochastic universal sampling
    cumfit = cumsum(FitnV);
    trials = cumfit(Nind) / Nsel * (rand + (0:Nsel-1)');
    Mf = cumfit(:, ones(1, Nsel));
    Mt = trials(:, ones(1, Nind))';
    [NewChrIx,  ] = find(Mt < Mf & [ zeros(1, Nsel); Mf(1:Nind-1, :) ] <= Mt);

    % Shuffle new population
    [ , shuf] = sort(rand(Nsel, 1));
    NewChrIx = NewChrIx(shuf);


    % End of function
    end
```

Question1.m

```
    %
% 此文件用于订单分拣
%
clear;
clc;
% 加载数据
load Order_Item.mat
load Order_Item_Quantity.mat
load Number.mat
% 初始变量设置
[Order,Item] = size(Order_Item);
N = 200; %货架数
% 订单分拣状态
Order_State = ones(Order,1); %1代表未分拣，0代表已分拣
Batch_total = [];
Batch_goods = [];
% 开始进行分拣
flag = 1;
```

```
B = 1;%批次编号使用
while flag
    %% 初始设置
    % 建立批次变量:批次中的订单集合，批次中货品种类数
    Batch = [];
    Item_State = zeros(1,Item);
    %% 开始选取
    %% 选取种子订单
    % 建立未分拣的集合
    Index = find(Order_State =0);   %Index是真实的订单号
    Number_sort = Number(Index);
    %     种子选择方式1：最大订单选择
    [ ,Seed] = max(Number_sort); %注意此处Seed的映射集并不是Number
    % 种子选择方式2：随机选择订单
    %     temp = randperm(length(Index));
    %     Seed = temp(1);
    Seed = Index(Seed); %重映射
    % 将种子订单对应的货品置1
    Order_State(Seed) = 0;%代表已经挑选过了
    Item_State = Item_State + Order_Item(Seed,:);
    Item_State = Item_State =0; % 将变量置1
    temp_number = sum(Item_State);
    Batch = [Batch;Seed];
    %% 选取其他订单
    Batch_flag = 1;
    while Batch_flag
        % % 跟随方式1：不同个数最多的
        %         % 计算相似度
        %         Dislikeness_best = Inf;
        %         Like_index = 0;
        %         for i=Index'
        %             % 不能是已经选过的
        %             if  Order_State(i)
        %                 continue;
        %             end
        %             Dislikeness = sum((Item_State-Order_Item(i,:))==-1);
        %             if Dislikeness<Dislikeness_best
        %                 % 检查是否超过200N
        %                 Item_State_temp = Item_State + Order_Item(i,:);
        %                 if sum(Item_State_temp =0)>N
        %                     continue;
        %                 end
        %                 Dislikeness_best = Dislikeness;
        %                 Like_index = i;
        %             end

```

```
%         % 跟随方式2：相同个数最多的
%         % 计算相似度
%         Likeness_best = -1;
%         Like_index = 0;
%         for i=Index'
%             % 不能是已经选过的
%             if  Order_State(i)
%                 continue;
%             end
%             Likeness = sum((Item_State+Order_Item(i,:))==2);
%             if Likeness>Likeness_best
%                 % 检查是否超过200N
%                 Item_State_temp = Item_State + Order_Item(i,:);
%                 if sum(Item_State_temp =0)>N
%                     continue;
%                 end
%                 Likeness_best = Likeness;
%                 Like_index = i;
%             end
%         end
        % 跟随方式3：相似程度最大的
        % 计算相似度
        Likeratio_best = -1;
        Likeness_best = 0;
        DisLikeness_best = 0;
        Like_index = 0;
        for i=Index'
            % 不能是已经选过的
            if  Order_State(i)
                continue;
            end
            Likeness = sum((Item_State+Order_Item(i,:))==2);
            DisLikeness = sum((Item_State-Order_Item(i,:))==-1);
            Likeratio = (Likeness)/(DisLikeness+1e1);  %加一防止为,这个定义为相似系数
            if Likeratio>Likeratio_best
                % 检查是否超过200N
                Item_State_temp = Item_State + Order_Item(i,:);
                if sum(Item_State_temp =0)>N
                    continue;
                end
                Likeratio_best = Likeratio;
                Like_index = i;
                Likeness_best = Likeness;
                DisLikeness_best = DisLikeness;
            end
        end
```

```
        % 检查是否有可行的相似订单
        if Like_index==0
            Batch_flag = 0;
        else
            % 将相应订单加入Seed
            Order_State(Like_index) = 0;%代表已经挑选过了
            Item_State = Item_State + Order_Item(Like_index,:);
            Item_State = Item_State =0; % 将变量置1
            Batch = [Batch;Like_index];
            temp_number = sum(Item_State);
        end
    end
    %% 数据存储
    % 将Batch转化为0-1向量进行存储
    Batch_temp = zeros(Order,1);
    for i=Batch'
        Batch_temp(i)=1;
    end
    Batch_goods = [Batch_goods,sum(Item_State)];
    Batch_total = [Batch_total,Batch_temp];
    %     eval(['Batch',num2str(B),'=','Batch',';']);
    %     B = B+1;
    %     Batch_total = {Batch_total,Batch};
    % 判断是否跳出循环
    if  sum(Order_State)
        flag = 0;
    end
end
% 显示批次信息
figure(1);
bar(sum(Batch_total));
hold on;
xlabel('批次序号');
ylabel('批次中的订单数');
title('订单分批结果');

figure(2);
bar(Batch_goods);
hold on;
xlabel('批次序号');
ylabel('批次中的货架数');
title('批次货架利用情况');

save('Batch_total.mat','Batch_total');
```

Question2.m

```
    % 此文件是问题2的主函数
clear;
clc;
% 载入数据
load Batch_total.mat
load Order_Item.mat

% 参数设置
N = 200;

% 载入初始数据
Batch_Number = size(Batch_total,2);
Batch_Order = sum(Batch_total);
[Order,Item] = size(Order_Item);

% 重要变量设置
Seq_total = zeros(Batch_Number,N); %注意其中一部分可能是0，Goods不等于N时
Dis_total = zeros(Batch_Number,1);

% 分批开始优化
for i=1:Batch_Number
    %% get货品数量
    Batch_now = Batch_total(:,i);
    Order_Index = find(Batch_now =0); %Order的序号
    Item_State = zeros(1,Item);
    for j=Order_Index'
        Item_State = Item_State + Order_Item(j,:);
    end
    Item_Now = Item_State =0;
    Item_Real = find(Item_Now =0);
    Goods = sum(Item_Now); % 货品数量
    %% 调用优化函数
    [Seq,Dis] = GA(Order_Index,Order_Item,Goods,Item_Real);
    %% 记录数据
    % 将Seq解码
    Seq = Item_Real(Seq);
    len = length(Seq);
    if len =200
        Seq = [Seq,zeros(1,N-len)];
    end
    Seq_total(i,:) = Seq;
    Dis_total(i) = Dis;
end
```

```
save('Seq_total.mat','Seq_total');
save('Dis_total.mat','Dis_total');
bar(Dis_total);
hold on;
xlabel('批次序号');
ylabel('各批分拣任务的拣选距离总和');
title('各批分拣任务的拣选距离示意图');
```

Question3.m

```
    clear;
    clc;
    % 加载数据
    load Order_Item.mat
    load Batch_total.mat
    load Seq_total.mat
    %参数设置
    Worker = 5; %代表分拣工有5人

    %% 分配订单
    Batch_Number = size(Batch_total,2);
    Order_Group_Work_Time = zeros(size(Order_Item,1),3);
    Load_total = [];
    for i=1:Batch_Number
        % 建立初始变量
        Batchi = Batch_total(:,i);
        Seqi = Seq_total(i,:); %此处是货架上的序号与真实货号的映射关系
        Order_Index = find(Batchi =0);
        Order = length(Order_Index);
        Order_State = ones(Order,1);  %用于记录订单是否已经分配
        Value = zeros(Worker,1); %效用函数,用于评价一个订单到低分给谁
        Load = zeros(Worker,1); %负载函数用于衡量worker的任务量，即距离
        Dis = zeros(Worker,1); %用于衡量订单分配给某一Worker增加的任务量

        % 将订单信息转化成坐标信息
        Goods_Number = sum(Seqi =0); %货架总数,可能会少于200
        %% 信息预处理，将批次中的订单信息转化成端点位置
        Node = zeros(Order,2); %用于记录端点位置的数组每一行包括（leftnode,
            rightnode），Node中的序号并非真正订单序号
        for j=1:Order
            temp = Order_Item(Order_Index(j),:);
            temp_Item = find(temp =0); %订单中的货号
            minmax = ismember(Seqi,temp_Item);
            leftnode = find(minmax==1,1,'first');
```

```
            rightnode = find(minmax==1,1,'last');
            Node(j,1) = leftnode;
            Node(j,2) = rightnode;
        end

        for j=1:Worker
            eval(['Work',num2str(j),'=','[]',';']); %建立工人的工作表
        end
        flag = 1; % 用于记录是否出现了任务数少于工人数的情况
        % 初始随机分配任务
        if Order<=Worker
            % 任务数少于工人数
            ran = randperm(Order);
            for j=1:Order
                eval(['Work',num2str(j),'=',num2str(ran(j)),';']); %记录的不是真正的订
                    单序号
            end
            % 计算负载
            for j=1:Worker
                eval(['temp','=','Work',num2str(j),';']);
                Load(j) = (Node(temp,2)-1)*2;
            end
            flag = 0;% 特殊处理
        else
            % 任务数多于工人数
            ran = randperm(Order);
            k=1;
            for j=1:Order
                eval(['Work',num2str(k),'=','[','Work',num2str(k),',',num2str(ran(j)),'
                    ]',';']); %记录的不是真正的订单序号
                if k==Worker
                    k=1;
                else
                    k=k+1;
                end
            end
            % 优化任务序列
            % 对每个工人进行优化
            disp('-----------------');
            disp('-----------------');
            disp(['第',num2str(i),'批次： ']);
            for j=1:Worker
                eval(['temp','=','Work',num2str(j),';']);
                Node_Worker = Node(temp,:);
                [Load(j),Seq_temp] = ACO(Node_Worker);
                eval(['Seq',num2str(j),'=Seq_temp;']);    % 此处的Seq是指工人处理的订单
```

```
                号顺序
78              disp(['最短距离:' num2str(Load(j))]);
79              disp(['最短路径:' num2str(Seq_temp)]);
80          end
81          % 负载均衡计算
82          % FEs为平衡轮数
83          FEs = 20;
84          for j=1:FEs
85              % 找出负载差异最大的两个工人
86              [Load_Max,Worker_Max] = max(Load);
87              [Load_Min,Worker_Min] = min(Load);
88              Delta_Load = abs(Load_Max - Load_Min);
89              % 重新分配两个人的任务
90              % 将负载重的随机分一个给负载轻的
91              eval(['tempWork_Max=Work',num2str(Worker_Max),';']);
92              eval(['tempWork_Min=Work',num2str(Worker_Min),';']);
93              len = length(tempWork_Max);
94              if len==1
95                  continue;
96                  % 不足够分的结束
97              end
98              temp = randperm(len);
99              temp = temp(1);
100             Node_give = tempWork_Max(temp);
101             temp = ismember(tempWork_Max,Node_give);
102             temp =  temp;
103             tempWork_Max = tempWork_Max(temp);
104             tempWork_Min = [tempWork_Min,Node_give];
105             Node_Worker = Node(tempWork_Max,:);
106             [tempLoad_Max,tempSeq_Max] = ACO(Node_Worker);
107             Node_Worker = Node(tempWork_Min,:);
108             [tempLoad_Min,tempSeq_Min] = ACO(Node_Worker);
109             Delta_temp = abs(tempLoad_Max-tempLoad_Min);
110             if Delta_Load>Delta_temp   %均衡成功
111                 % 修改原有分工
112                 eval(['Work',num2str(Worker_Max),'=tempWork_Max;']);
113                 Load(Worker_Max) = tempLoad_Max;
114                 eval(['Seq',num2str(Worker_Max),'=tempSeq_Max;']);
115                 eval(['Work',num2str(Worker_Min),'=tempWork_Min;']);
116                 Load(Worker_Min) = tempLoad_Min;
117                 eval(['Seq',num2str(Worker_Min),'=tempSeq_Min;']);
118             end
119         end
120         % 均衡以后再次显示
121         disp(['                    ']);
122         for j=1:Worker
```

```
                eval(['Seq_temp=Seq',num2str(j),';']);
                disp(['最短距离:' num2str(Load(j))]);
                disp(['最短路径:' num2str(Seq_temp)]);
            end
        end
        %% 数据保存 存储是第几个工人在第几次做的
        Load_total = [Load_total,Load];
        for j=1:5 %按照work的标注去找的
            eval(['temp=Work',num2str(j),';']);
            temp_Order = Order_Index(temp);%将Work中的序号转化为订单号
            for k=1:length(temp_Order)
                 Order_Group_Work_Time(temp_Order(k),1) = i;
                 Order_Group_Work_Time(temp_Order(k),2) = j;
                 eval(['temp_Order_Seq=Seq',num2str(j),'(k);']);
                 Order_Group_Work_Time(temp_Order(k),3) = temp_Order_Seq;
            end
        end
    end
    save Order_Group_Work_Time.mat Order_Group_Work_Time;
```