基于相似度网络与相关度网络的动态模拟分拣方案的研究

摘 要

为探究电商公司货品分拣系统的优化问题，我们设计**基于贪心策略的分批算法**，建立**基于相似度网络的订单分批模型**，来研究给定货架数量时的最少转运批次数。然后设计出**基于贪心策略的货品摆放算法**，建立**基于相关度网络的货物摆放模型**，使所有批次的拣选距离总和最小。最后**设计订单指派算法**，建立**动态模拟指派模型**，使分拣工能够尽快完成分拣任务且运动距离尽可能平均。

针对问题一，我们**基于贪心策略设计出分批算法**，建立**基于相似度网络的订单分批模型**。贪心策略为定义**订单相似度计算公式**，计算任意两个订单之间的货品种类相似度，将计算结果构建**订单相似度网络**。然后基于相似度网络进行订单分批，优先考虑相似度较高的订单，将其安排在同一批次。相比直接暴力分批，基于贪心策略的分批算法通过高相似度**减缓了货架占用量的增长**，可以满足批次量尽可能少的要求。最终计算得出**订单分为 53 个批次**，并给出每批次订单的具体数量、货品种类数量以及**订单分批方案**。

针对问题二，需要对问题一分批的结果继续深入优化货品摆放位置，设计合理的算法使得各批次的拣选距离总和达到最小。因为所有货架等距排列在一条直线上，因此同样采用**贪心策略设计货品摆放算法**，建立**基于相关度网络的货品摆放模型**。在每一批次的货品摆放中，定义不同货品之间的**货品相关度计算公式**，计算同一批次中任意两个货品之间的联系程度，将计算结果构建**货品相关度网络**。基于货品相关度网络进行货品摆放，优先考虑将相关度较高的货品安排在较近的位置。相比于直接暴力摆放货品，基于贪心策略的摆放算法通过将关系较大的货品摆放在较近的位置，综合权衡了单一订单和所有订单的货品之间的关系，可以很好地使每个订单的拣选距离总和较小。最终计算得出**所有批次的订单拣选距离总和为 140172**，并给出各批次货品分拣任务的拣选距离总和与**货品摆放方案**。

针对问题三，需要在解决问题一和问题二得到的订单分批以及货品摆放之后的数据的基础上，指派分拣工进行订单的分拣任务。为了能满足完成任务的时间尽可能短且分拣工运动距离尽可能均衡，我们设计出**动态模拟指派算法**，建立**动态模拟指派模型**。算法的主要策略是在分拣工完成当前任务之后，立即寻找距该分拣工最近的一个订单进行指派，等待下一个分拣工完成时继续指派，直至所有任务都完成为止。针对寻找距离当前分拣工最近的订单，我们定义**员工完成某一订单所需的运动距离的计算公式**，找出所需的运动距离最小的一个订单作为距离当前分拣工最近的订单。最后通过比对完成任务的具体时间以及每位分拣工的运动距离决定是否采纳该指派方案。最终计算得出**所有批次分拣任务的消耗时间总和为 47711，所有批次五名分拣工在每批次的运动距离的均值的平均值为 755.62，所有批次五名分拣工在每批次运动距离的标准差均值的平均值为 103.79**，并给出各批次的具体**分拣工指派方案**。通过计算机模拟可以准确预测方案的效果，且动态指派可以基于当前任务完成情况给出一个最近的下一个订单，这一点解决了静态分配容易出现的订单重复分拣以及运动距离不均衡问题。

关键词：**贪心策略 相似度网络 相关度网络 动态模拟**

目录

一、 问题重述

1.1 问题背景

随着人民生活水平的普遍提高，电商与物流行业发展较为迅速，配送分拣效率逐渐成为影响行业发展趋势的重要因素。其工作总体流程可以分为：汇总统计、转运上架、根据订单分拣、核对出库。由于业务量的不断增加与场地成本的制约，考虑到将来当日订单所包含的货品种类数量会大于货架数量，所以深究并建立分拣系统优化模型对于电商公司和物流企业的整体性能具有极为重大的经济价值与实际意义。

2.1 问题要求

基于上述问题背景我们需要设计如下数学算法解决具体问题：

1) 问题一设计**分批算法**，即将当天所有订单分为各个分拣批次，且各批次订单所包含的货品种类数量不超过货架数量，批次越少越好。并计算货架数量 N =200 时最少的批次数量、每批次订单数量、货品种类数以及分批方案。
2) 问题二设计**货品摆放位置优化算法**，使得各批次分拣任务的所有订单拣选货物距离总和尽可能较小。在问题一分批算法的结果数据基础上，考虑并计算各批次订单拣选距离总和与相应的货物摆放位置以及所有批次订单拣选距离总和。
3) 问题三设计**最优订单指派算法**，即按批次将所有订单指派给各个分拣工同时完成分拣任务，使得完成时间较短，且运输距离尽可能平均。基于问题一与问题二的算法，当分拣工人数 n =5 时，计算给出各批次订单的分拣工指派方案以及处理订单的顺序。

二、 问题分析

2.1 问题分析

1） 问题一的分析

针对问题一，由于货架的容积和载重不受限制，因此无需考虑每个订单的总货物量，只需考虑每个批次订单所包含的货物种类数量这一因素。首先我们**基于贪心策略设计出分批算法，建立基于相似度网络的订单分批模型**。贪心策略为定义任意两个订单之间的相似度计算公式，建立相似度网络。然后基于相似度网络进行订单分批，优先考虑相似度较高的订单，将其安排在同一批次。相比于直接暴力分批，基于贪心策略的分批算法通过高相似度减缓了货架占用量的增长，可以满足批次量越少越好的要求，从而达到减少相应转运次数来提高分拣效率的目标。

2） 问题二的分析

针对问题二，需要对问题一分批的结果继续深入优化货品摆放位置以提高优化效率，设计合理的算法使得各批次的拣选距离总和达到最小。因为所有货架等距排列在一条直线上，因此我们采用同样的**贪心策略设计货品摆放算法，建立基于相关度网络的货品摆放模型**。贪心策略为：在每一批次的货品摆放中，定义不同货品之间的货品相关度计算公式，建立货品相关度网络，基于货品相关度网络进行货品摆放，优先考虑将相关度较高的货品安排在较劲的位置。相比于直接暴力算法摆放货品，基于贪心策略的摆放算法通过将关系较大的货品摆放在较近的位置，综合权衡了单一订单和所有订单的货品之间

的关系，可以很好地使每个订单的拣选距离总和较小，从而满足所有批次的拣选距离总和达到最小这一目标。

3） 问题三的分析

针对问题三，需要在解决问题一和问题二得到的订单分批以及货品摆放之后的数据的基础上，指派分拣工进行订单的分拣任务。为了能满足完成任务的时间尽可能短且分拣工运动距离尽可能均衡，我们设计出**动态模拟指派算法，建立动态模拟指派模型**。即通过程序模拟分配过程，根据模拟结果的可靠性决定是否作为指派方案。

算法的主要策略是在分拣工完成当前任务之后，立即寻找距该分拣工最近的一个订单进行指派，等待下一个分拣工完成时继续指派，直至所有任务都完成为止。最后通过比对完成任务的具体时间以及每位分拣工的运动距离决定是否采纳该指派方案。通过计算机模拟可以准确预测方案的效果，且动态指派可以基于当前任务完成情况给出一个最近的下一个订单，这一点解决了静态分配容易出现的订单重复分拣以及运动距离不均衡问题。

三、 模型假设

3.1 假设

1) 假设每名分拣工工作时不会放错货品的位置。
2) 假设每名分拣工工作时体力充足。
3) 假设每名分拣工移动单位为 1 的距离所需的时间为 1。

四、 符号说明

4.1 符号的说明

符号	符号解释	符号单位
R_{ij}	订单相似度	
IN_i	一个订单所包含的货品种类数量	个
G_p	所有批次	
V_{ij}	货品相关度	
O_k	某一订单	
N	货架数量	个
d	拣选距离	
S	货品所在货架序号	
DP_{ij}	分拣工 j 完成订单 i 所需的运动距离	

WP_j	分拣工 j 当前位置
AP_i	订单 i 最大货架号所在位置
IP_i	订单 i 最小货架号所在位置

五、 模型建立与求解

5.1 问题一对应的模型建立与求解

针对问题一，首先我们根据订单信息数据设计出基于贪心策略的分批算法，建立基于相似度网络的订单分批模型。贪心策略为在确保各个批次的订单所包含的货品种类数量均不超过货架数量 N 的基础上，定义任意两个订单之间的相似度计算公式，建立相似度网络。然后基于相似度网络进行订单分批，优先考虑若干个相似度较高的订单，将其安排在同一批次，使其批次尽可能减少从而提高效率。

5.1.1 相似度计算公式

根据任意两个订单间所包含货品种类数量的交集与并集元素个数的比例，进行任意两个订单之间的相似度计算，并给出如下相似度计算公式：

$$R_{ij}=\frac{len\left(IN_i\cap IN_j\right)}{len\left(IN_i\cup IN_j\right)} \tag{1}$$

其中 R_{ij} 为任意两个订单间的相似度大小，IN_i 表示一个订单所包含的货品种类集合，len 函数用来统计集合元素的个数。

5.1.2 建立相似度网络

根据任意两个订单间所包含货品种类数量的交集与并集元素个数的比例，进行任意两个订单之间的相似度计算，并给出如下相似度计算公式：

$$\begin{bmatrix} R_{11} & R_{12} & \cdots & R_{1n} \\ R_{21} & R_{22} & \cdots & R_{2n} \\ \cdots & \cdots & \cdots & \cdots \\ R_{m1} & R_{m2} & \cdots & R_{mn} \end{bmatrix} \tag{2}$$

将相似度计算结果构建成一个订单相似度网络，用于描述任意两个订单之间的彼此相似性，并将普通聚类问题转化为图的划分问题，有助于分批算法中的订单分批操作。

5.1.3 设计分批算法

我们根据订单相似度数据设计出基于贪心策略的分批算法，以当前订单情况为基础根据订单相似度作最优选择，而尽可能减少考虑各种可能的整体情况，节省了为计算最优解而穷尽所有可能所必须耗费的大量时间，降低算法时间复杂度，并建立基于相似度网络的订单分批模型。以下是**基于贪心策略的订单分批算法**流程图：

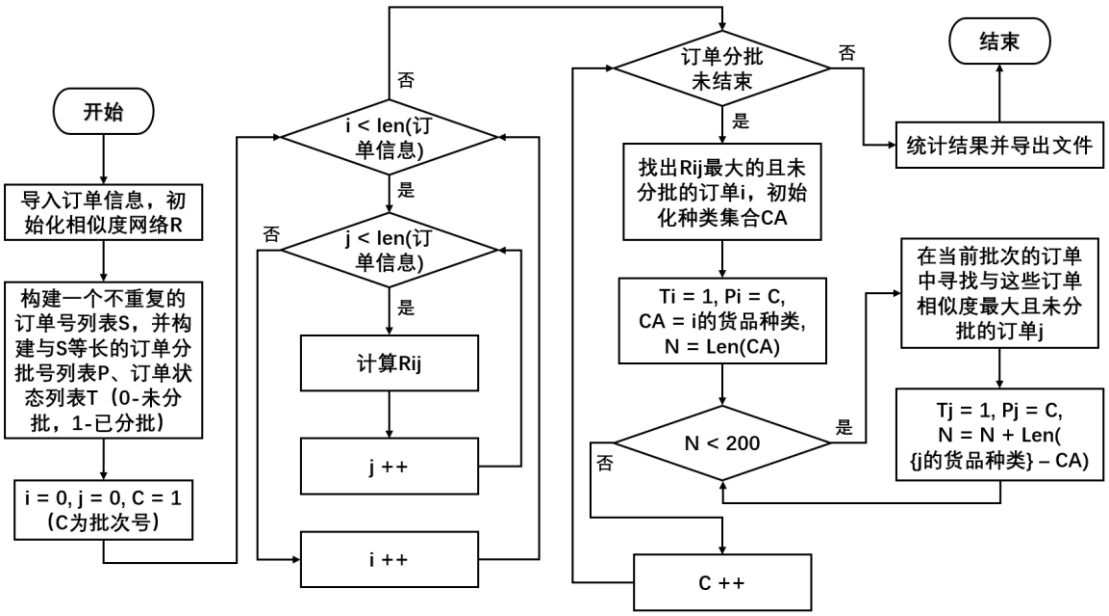

图 1 订单分批算法流程图

该算法的主要思想是通过将相似度较高的几个订单作为同一批次，这一能够让相似的货品种类尽可能的多，从而把尽可能多的订单数量安排在同一批次下达到缓慢增长 N 的目的，实现分批数量越少越好的目标。其主要步骤如下：

Step1：导入数据，初始化变量。

首先导入附件中所给的订单数据信息，因为订单数据信息中的订单号存在重复，所以需要定义三个列表，列表 S 为去重后的订单号信息，列表 T 记录订单的状态信息（已分批为 1，未分批为 0），列表 P 记录订单的分批号，列表 T 和列表 P 的内容初始化为空。

Step2：建立相似度网络。

然后根据订单相似度计算公式，以变量 i 和变量 j 进行双重循环，计算出任意两个订单之间的相似度 R，将结果构建成一个订单相似度网络，用于后面的分批操作。

Step3：进行订单分批。

然后以 N = 200 为一轮，进行订单的分批。每进行一轮，就将批次号 C 加 1。在每一轮中，我们首先获取当前相似度最大且未分批的订单，分配给该订单批次号，并修改其状态，记录其货品种类；然后我们每次都寻找在当前同一批次中与已分配的订单相似度最大的订单，对该订单进行分配批次号操作，并修改其状态，将其不重复的货品种类记录；直至当前剩余货架数量无法满足当前找到的订单的货架需求为止。在每一轮结束后更新批次号。

Step4：统计模型结果并导出。

最后我们统计出每个分批的订单数量、货品种类数量等信息导出文件，并单独构建分批方案写入 result1.csv 文件导出，同时输出最终所用的分批数。

5.1.4 模型结果

最后我们通过以上**基于贪心策略的订单分批算法**，建立**基于相似度网络的订单分批模型**，计算得出以下订单批次模型统计结果：

表 1 订单批次模型结果统计表

分批号	订单数量	货品种类数量	分批号	订单数量	货品种类数量
1	35	196	28	19	198
2	11	199	29	14	197
3	20	194	30	13	193
4	20	194	31	13	198
5	23	190	32	7	155
6	54	196	33	18	200
7	21	199	34	16	174
8	12	195	35	17	196
9	17	189	36	20	198
10	11	194	37	13	194
11	19	192	38	18	190
12	12	190	39	14	193
13	10	193	40	20	190
14	14	198	41	13	199
15	25	195	42	16	194
16	13	184	43	15	187
17	19	190	44	21	199
18	14	186	45	23	200
19	24	199	46	16	187
20	23	198	47	12	182
21	12	180	48	13	197
22	22	200	49	17	200
23	22	192	50	14	191
24	20	193	51	12	199
25	21	196	52	12	172
26	23	200	53	9	132
27	11	197			

通过观察发现，将当日所有订单分为以上 53 个批次，且保证每批订单所包含的货品种类数量不超过货架数量 N，并给出相对应的订单数量以及货品种类数量，以达到较小的转运次数与较高的转运效率。

然后我们列举出以下如下 12 条数据作为订单分批方案展示，详细数据作为结果文件 result1.csv 提交。

表 2 订单分批结果展示表

序号	订单号	分批号
1	D0001	18
2	D0002	3

3	D0003	1
4	D0004	42
5	D0005	6
6	D0006	45
7	D0007	7
8	D0008	18
9	D0009	3
10	D0010	2
11	D0011	24
12	D0012	43

相比于直接暴力分批求解，基于贪心策略的分批算法通过高相似度减缓了批次量 N 的增长，可以满足批次量越少越好的要求，从而达到减少相应转运次数来提高分拣效率的目标。

接着我们为了将结果数据更为直观的可视化，根据计算得出的分批号、订单数量以及货品种类数量，绘制如下各批次订单数量与货品种类数量结果柱形图：

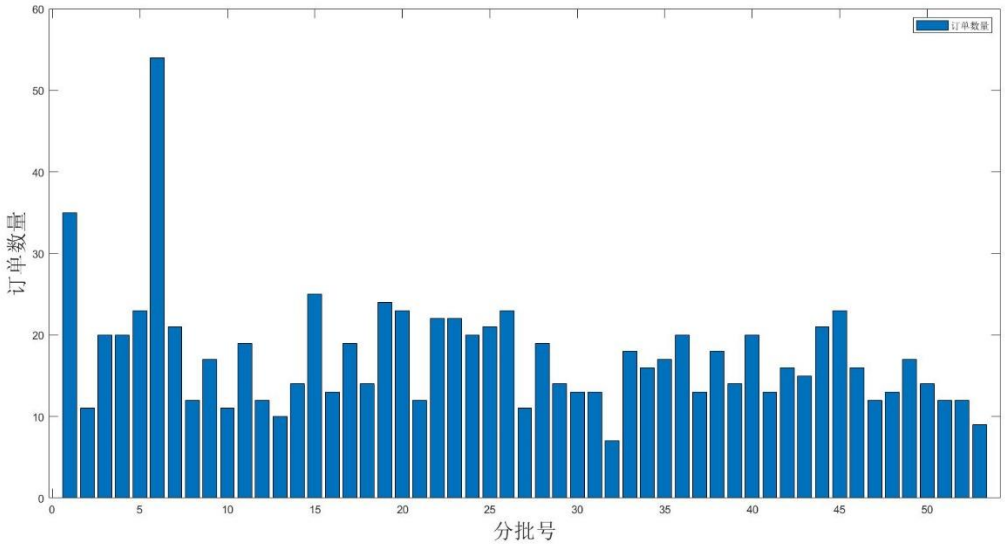

图 2 各批次订单数量结果柱形图

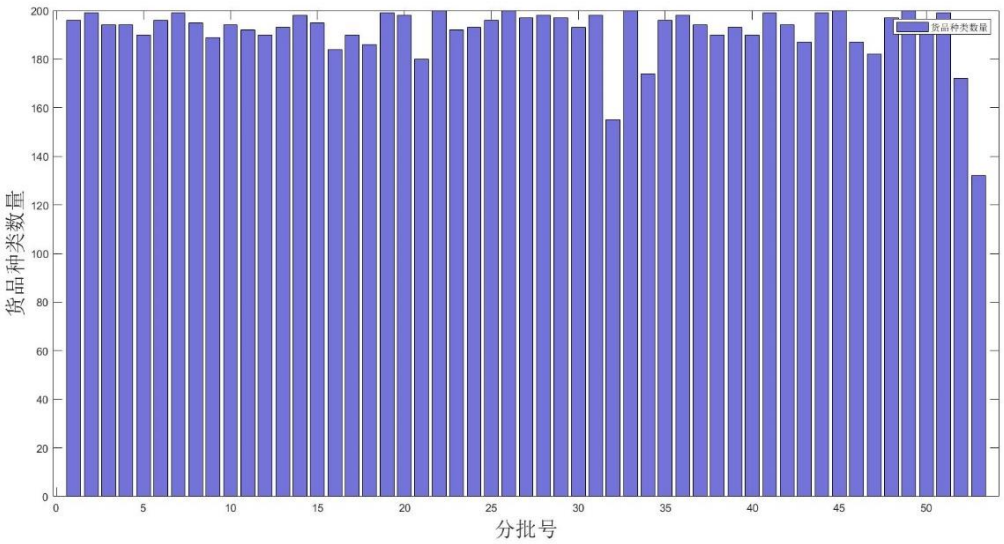

图 3 各批次货品种类数量结果柱形图

通过观察可以发现，各批次订单数量变化趋势相较于货品种类数量较为明显，变化幅度较大，说明基于贪心策略的订单分批算法会根据当日所有货品中不同订单的相似度，动态调整各批次订单数量以及货品种类数量，且货品种类数量在满足不超过的最大货架数量的条件下尽可能较大接近 200，以达到较少的转运批次，从而提高转运效率。

5.2 问题二对应的模型建立与求解

针对问题二，需要对问题一分批的结果继续深入优化货品摆放位置以提高优化效率，设计合理的算法使得各批次的拣选距离总和达到最小。因为所有货架等距排列在一条直线上，所以我们采用与问题一算法核心相似的**贪心策略设计货品摆放算法**，建立**基于相关度网络的货品摆放模型**。

贪心策略为：在每一批次的货品摆放中，定义不同货品之间的货品相关度计算公式。然后建立货品相关度网络，基于货品相关度网络进行货品摆放，优先考虑将相关度较高的两种货品集中在距离较近的位置以减少拣选距离总和，缩短拣选时间，从而提高拣选效率。因此我们设立如下目标函数与约束条件：

目标函数：

$$\min\ d\left(G_p\right)=\sum_{O_k\in G_p} d\left(O_k\right) \tag{3}$$

约束条件：

$$s.t.\ \begin{cases} d\left(O_k\right)=\max_{i\in O_k} S\left(i\right)-\min_{i\in O_k} S\left(i\right) \\ 1\le S\left(i\right)\le 200 \end{cases} \tag{4}$$

其中 G_p 代表所有批次，O_k 代表任意一个订单，目标函数是求解所有批次全部订单所包含货品拣选距离总和的最小值，约束条件是任意订单中货品所在货架序号的最大值与最小值的差值最小以及货架序号最大值不超过货架数量 $(N=200)$ 且最小值不小于 1。

5.2.1 相关度计算公式

首先我们基于贪心策略的货品摆放算法，给出如下相关度计算公式：

$$V_{ij}=\begin{cases}V_{ij}+1, \exists O_k \in G_p, i, j \in O_k \\ V_{ij}, \text{其他}\end{cases} \tag{5}$$

其中V_{ij}表示任意两个订单间的相关度大小，计算时首先初始化V_{ij}全部为 0，然后再判断两种货品是否属于该批次任意订单之内。若都属于，则将这两个订单间的相关度加 1，否则不变。

5.2.2 建立相关度网络

然后利用相关度计算公式计算出同一批次中任意两个货品之间的联系程度，利用计算结果构建以下网络：

$$\begin{bmatrix} V_{11} & V_{12} & \cdots & V_{1n} \\ V_{21} & V_{22} & \cdots & V_{2n} \\ \cdots & \cdots & \cdots & \cdots \\ V_{m1} & V_{m2} & \cdots & V_{mn} \end{bmatrix} \tag{6}$$

将相关度计算结果构建成一个货品相关度网络，用于描述任意两个货品之间的彼此相关度，并将普通聚类问题转化为图的划分问题，有助于货品摆放位置算法中的货品位置选择操作。

5.2.3 设计摆放算法

我们根据同一订单内任意两件货品之间的相关度数据设计出基于贪心策略的货品摆放算法，以当前货品情况为基础根据货品相关度作最优选择，而尽可能减少考虑各种可能的整体情况，节省了为计算最优解而穷尽所有可能所必须耗费的大量时间，降低算法时间复杂度，并建立**基于相关度网络的货品摆放模型**。以下是**基于贪心策略的货品摆放算法**流程图：

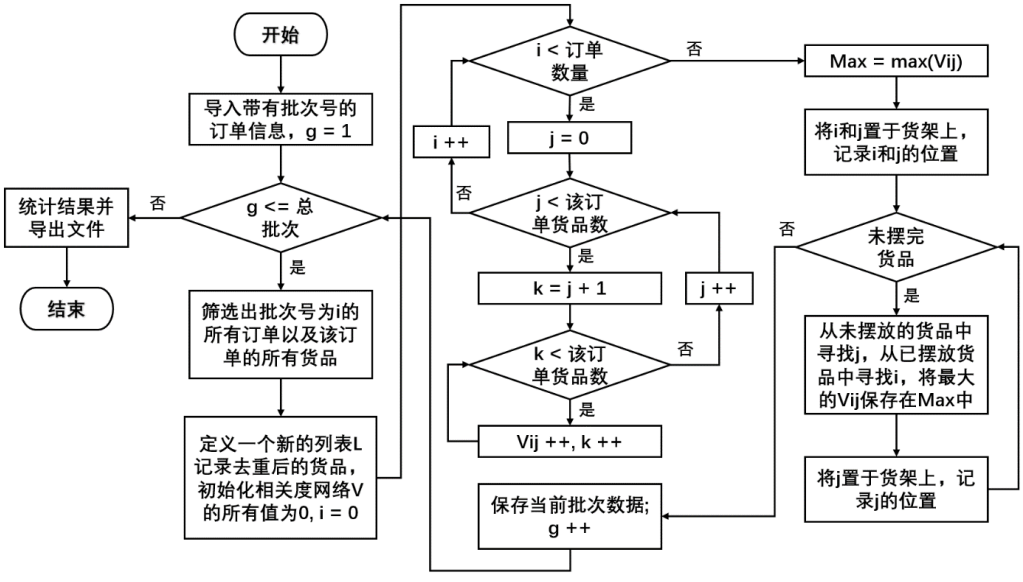

图 4 货品摆放算法流程图

该算法的主要思想是突破单个订单的局限，针对所有订单的货品建立一个货品相关度网络，将相关度较大的几个货品放在较近的位置，从而使每个订单的拣选距离总和较小，实现所有批次的拣选距离总和达到最小这一目标。其主要步骤如下：

Step1：导入数据，初始化变量。

首先根据问题一的运行结果，将附件中所给的订单数据信息进行完善，增加分批号这一列，然后导入该数据，并以批次号作为循环变量来统计各批次的拣选距离总和数据。

Step2：建立相关度网络。

然后根据货品相关度计算公式，以变量 i 循环遍历每一个订单。针对每一个订单，都通过循环遍历 j 和 k 对该订单中任意两个货品按相关度计算公式进行相关度的计算。因为不同订单可能存在同样的货品，所以货品之间的相关度越高说明同时包含两者的订单数量越多，因此将他们放置在较近的位置可以很好的同时让多个订单满足拣选距离较小。最终将结果构建成一个货品相关度网络，用于后面的货品摆放操作。

Step3：进行货品摆放。

针对每一批次的货品摆放操作，首先从该批次中选出相关度最大的两个货品，将其摆放在货架上，然后针对未摆放的货品进行循环。每轮循环中都从未摆放的货品中寻找货品 j，从已摆放货品中寻找货品 i，找出货品 j 与货品 i 之间相关度最大的两者，然后摆放货品 j，更新货品状态与货架信息。

Step4：统计模型结果并导出。

最后我们统计出每个分批的订单拣选距离总和信息导出文件，并单独构建货品摆放方案写入 result2.csv 文件导出，同时输出所有批次的订单拣选距离总和。

5.2.4 模型结果

最后我们通过以上基于贪心策略的货品摆放算法，建立基于相关度网络的货品摆放模型，计算得出以下货品摆放位置模型统计结果：

表 3 货品摆放位置模型结果统计表

分批号	订单拣选距离总和	分批号	订单拣选距离总和
1	3255	28	3151
2	1605	29	2325
3	2708	30	2051
4	3088	31	2188
5	3395	32	935
6	6037	33	2875
7	3525	34	2434
8	2040	35	3007
9	2707	36	3364
10	1390	37	1858
11	3307	38	2957
12	1986	39	2408
13	1471	40	3119

14	2283	41	2057
15	4267	42	2342
16	1985	43	2086
17	2776	44	3630
18	2185	45	4001
19	4265	46	2556
20	3723	47	1560
21	1911	48	2176
22	3092	49	2802
23	3490	50	1627
24	3434	51	1376
25	3626	52	1412
26	3947	53	487
27	1890		

相比于直接暴力算法摆放货品，基于贪心策略的摆放算法通过将关系较大的货品摆放在较近的位置，综合权衡了单一订单和所有订单的货品之间的关系，可以很好地使每个订单的拣选距离总和较小，从而满足所有批次的拣选距离总和达到最小这一目标。

通过观察发现，**所有批次的订单拣选距离总和为 140172**，且保证每批订单所包含的货品种类数量不超过货架数量 ，并给出各批次中每一种货品的摆放位置、各批次分拣任务的拣选距离总和的最小值以及所有批次全部订单的拣选距离总和的最小值，以达到较小的转运距离与较高的转运效率。

然后我们列举出以下如下 12 条数据作为货品摆放位置方案展示，详细数据作为结果文件 result2.csv 提交。

表 4 货品摆放位置结果展示表

序号	货品编号	分批号	货架号
1	P0007	3	170
2	P0002	3	185
3	P0006	3	189
4	P0004	4	189
5	P0002	4	193
6	P0007	5	180
7	P0008	8	183
8	P0006	12	172
9	P0009	12	184
10	P0006	13	187
11	P0007	13	188
12	P0008	13	189

接着我们为了将结果数据更为直观的可视化，根据计算得出的所有批次全部订单拣选距离总和，绘制如下分批号与订单拣选距离总和结果柱形图：

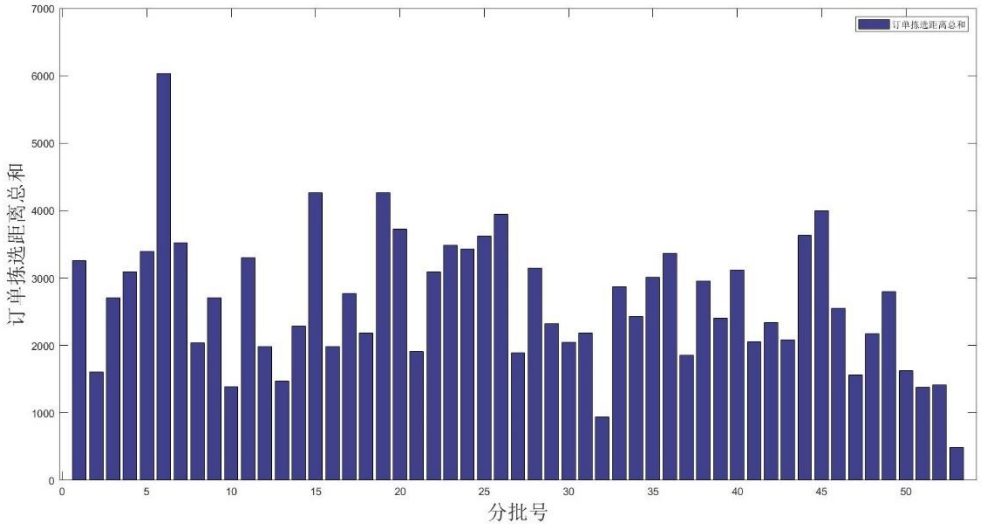

图 5 各批次订单拣选距离总和结果柱形图

通过观察柱形图我们可以发现，各批次订单拣选距离总和普遍较小，变化趋势较为明显，说明基于贪心策略的货品摆放算法随着不同分批号中货品种类的相关度变化，动态调整相关度高的货品放置在相近货架，从而减少订单拣选距离总和，结果数据良好，对于转运效率具有较为明显的提升。

5.3 问题三对应的模型建立与求解

针对问题三，我们需要在解决问题一和问题二得到的订单分批以及货品摆放的数据的基础上，指派分拣工进行订单的分拣任务。为了能够满足完成任务的时间尽可能短且分拣工运动距离尽可能均衡的要求，设计出**动态模拟指派算法，建立动态模拟指派模型**。即通过程序模拟分配过程，根据模拟结果的可靠性决定是否作为指派方案。

算法的主要策略是在分拣工完成当前任务之后，立即寻找距该分拣工最近的一个订单进行指派，等待下一个分拣工完成时继续指派，直至所有任务都完成为止。最后通过比对完成任务的具体时间以及每位分拣工的运动距离判断是否采纳该指派方案。

5.3.1 算法名词解释

首先我们引入时间轴 T 的概念，在假设中我们已经假设每名分拣工移动单位 1 的距离所需的时间为 1，因此时间轴 T 的作用就是记录当前已工作的时间，其初始值为 0。每当有分拣工完成一个订单任务时，更新该时间轴。

然后需要设计分拣工类以及其属性，属性包括：分拣数、任务结束时间以及任务结束位置。如下为分拣工属性说明表：

表 5 分拣工属性说明表

属性	作用
分拣数	该分拣工当前已完成分拣的订单数
任务结束时间	该分拣工执行完当前任务的时间
任务结束位置	该分拣工执行完当前任务的位置（货架）

针对分拣数的更新，分拣工每完成一次任务该值则增加 1；针对任务结束时间的更新，当时间轴 T 有变动时，时间轴 T 增加的值则任务结束时间减去对应的值（时间轴 T 的增加代表经过了多少时间），因为时间轴T的增加严格取决于任务结束时间的最小值，所以不会出现任务结束时间为负的情况；针对任务结束位置的更新，当分拣工完成当前任务且得到新的指派任务时候，更新任务结束时间以及任务结束位置的值。

5.3.2 设计动态模拟指派算法

然后我们设计**动态模拟指派算法**，使预先模拟和计划可能发生的任何种类的指派事件成为可能，动态模拟所有可能发生的任务指派过程，**建立动态模拟指派模型**。以下是动态模拟指派算法流程图：

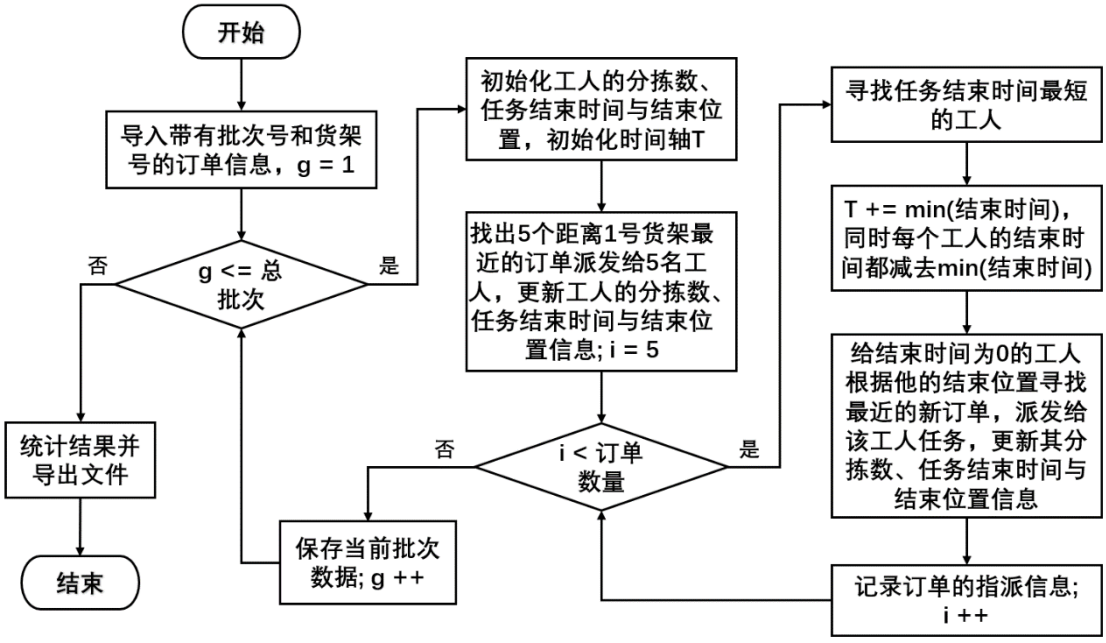

图 6 动态模拟指派算法流程图

该算法的主要思想是通过动态指派避免出现重复派单以及分拣工无任务执行的情况，通过模拟指派确保指派方案的可行性，从而在分拣工尽快完成指派任务的情况下使得各分拣工运动距离尽可能均衡。其主要步骤如下：

Step1：导入数据，初始化变量。

首先根据问题一和问题二的运行结果，将附件中所给的订单数据信息进行完善，增加分批号和货架号这两列，然后导入该数据，并以批次号作为循环变量来统计各批次的总耗时、分拣工的运动距离以及具体的指派方案。

Step2：初始化分拣任务。

然后针对各批次分拣任务的指派，初始化分拣工的分拣数、任务结束时间与任务结束位置，并初始化时间轴 T 为 0。之后选择出 5 个距离 1 号货架（可以是 1 号货架）最近的订单分别派发给 5 名分拣工，将他们的分拣数设置为 1，任务结束位置为该订单所需分拣的货架中距离该分拣工最远的货架，任务结束时间为分拣工到达任务结束位置所需的时间。

Step3：模拟指派过程。

然后开始等待分拣工工作，这一技术的模拟体现在时间轴 T 的更新。选择任务结束时间最短的分拣工，然后让时间轴 T 增加，增加的值为该分拣工的任务结束时间，然后让每名分拣工的任务结束时间减去时间轴 T 的增加量。针对任务结束时间为 T 的分拣工（至少 1 个），根据他们的任务结束位置，找到与他们距离最近的订单作为他们的下一个任务，此时需要更新分拣工的任务结束时间和任务结束位置，并让分拣工的分拣数增加 1。直至所有的订单都完成了分拣之后结束模拟指派。

针对寻找距离分拣工最近的订单算法的实现，我们定义了计算分拣工 j 完成订单 i 所用的运动距离的公式：

$$DP_{ij} = \begin{cases} AP_i - WP_j, \ WP_j \le IP_i \\ WP_j - IP_i, \ WP_j \ge AP_i \\ AP_i + WP_j - 2IP_i, \ IP_i \le WP_j \le AP_i 且 AP_i - WP_j \ge WP_j - IP_i \\ 2AP_i - WP_j - IP_i, \ 其他 \end{cases} \tag{7}$$

以某一订单最大的货架号所在位置与最小的货架号所在位置作为该订单的工作区间 $[IP_i, AP_i]$。如果分拣工在区间外，则运动距离为该分拣工位置到最大位置或者最小位置的差值；否则说明分拣工在工作区间内部，需要让分拣工先到达距离最近的一个端点，然后再到达另一端点完成该订单。根据以上算法找出分拣工花费运动距离最短的订单指派给该分拣工。

Step4：统计模型结果并导出。

最后我们统计出每个分批的指派方案的总耗时、分拣工的运动距离信息并导出文件，然后单独构建指派方案写入 result3.csv 文件导出。如果模拟的结果合理，则为该问题的指派方案，否则重新模拟。

5.3.3 模型结果

最后我们通过**动态模拟指派算法**，**建立动态模拟指派模型**，计算得出以下分拣工运动距离模型统计结果：

表 6 运动距离模型结果统计表

分批号	总耗时	运动距离					距离均值	距离标准差
		分拣工1	分拣工2	分拣工3	分拣工4	分拣工5		
1	1367	1008	1307	1367	1048	1174	1180.8	140.1
2	554	396	486	499	549	554	496.8	57.06
3	1089	872	967	1059	1089	845	966.4	97.22
4	1166	942	897	1102	1166	806	982.6	132.71
5	918	850	886	918	744	822	844	59.6
6	1978	1653	1872	1918	1685	1978	1821.2	129.13
7	1124	894	1124	929	1028	865	968	95.44
8	732	732	395	542	546	557	554.4	106.94
9	886	886	798	631	672	698	737	92.63
10	445	445	231	398	411	419	380.8	76.46
11	897	820	867	874	897	716	834.8	64.46

12	820	766	820	477	517	528	621.6	142
13	530	286	293	501	520	530	426	111.86
14	833	699	604	833	536	557	645.8	109.14
15	1198	1114	1173	1198	1180	1017	1136.4	66.02
16	766	709	737	766	432	472	623.2	141.51
17	916	916	685	691	775	639	741.2	97.8
18	827	634	567	827	474	486	597.6	128.54
19	1298	1182	1239	1278	1298	922	1183.8	136.75
20	1255	887	958	1255	876	885	972.2	144.43
21	789	740	789	485	497	501	602.4	133.36
22	1071	723	850	971	945	1071	912	117.83
23	1149	893	1134	796	1149	823	959	152.41
24	1180	971	1180	851	886	915	960.6	116.53
25	1267	1095	1174	1242	1040	1267	1163.6	85.94
26	1217	1138	994	1217	856	879	1016.8	141.52
27	714	714	425	562	566	569	567.2	91.44
28	868	720	763	776	775	868	780.4	48.32
29	873	563	779	870	873	527	722.4	149.17
30	828	697	565	828	489	498	615.4	129.74
31	858	787	858	700	540	551	687.2	126.11
32	430	385	430	146	146	149	251.2	128.41
33	868	868	738	755	838	800	799.8	48.86
34	792	659	612	784	635	792	696.4	76.3
35	924	786	827	908	919	924	872.8	55.9
36	1209	1089	1209	840	836	881	971	151.03
37	701	473	672	701	374	381	520.2	140.51
38	833	750	795	825	657	833	772	64.45
39	898	630	648	827	898	526	705.8	136.41
40	877	877	744	775	788	849	806.6	49.02
41	836	662	624	836	471	514	621.4	127.96
42	834	491	641	538	655	834	631.8	118.43
43	774	585	584	774	636	682	652.2	70.93
44	1214	1170	1214	845	863	874	993.2	163.18
45	1267	1058	1154	1267	911	927	1063.4	135.29
46	751	751	607	643	668	687	671.2	48.05
47	529	529	432	415	432	494	460.4	43.61
48	869	687	846	869	513	523	687.6	152.04
49	872	772	806	652	872	708	762	76.36
50	512	355	412	480	512	342	420.2	66.99
51	500	470	500	266	301	409	389.2	91.82
52	495	495	340	393	403	435	413.2	51.06
53	313	115	131	250	313	113	184.4	81.98

通过统计发现，所有批次分拣任务的消耗时间总和为 47711，批次中所有分拣工的运动距离每批次均值的平均值为 755.62 以及所有运动距离标准差的均值为 103.79，距离标准差较小说明分拣工运动距离做到尽可能平均，且保证每批订单所包含的货品种类

数量不超过货架数量，并给出各批次中每个分拣工的运动距离，通过动态模拟以达到较小的转运距离与较高的转运效率。

通过计算机模拟可以准确预测方案的效果，且动态指派可以基于当前任务完成情况给出一个最近的下一个订单，这一点解决了静态分配容易出现的订单重复分拣以及运动距离不均衡问题。

然后我们列举出以下如下 12 条数据作为货品摆放方案展示，详细数据作为结果文件 result3.csv 提交。

表 7 运动距离模型结果展示表

货品编号	分批号	货架号	分拣工号	处理次序
D0070	1	1	5	D0070
D0026	2	5	1	D0026
D0097	3	2	3	D0097
D0054	4	1	4	D0054
D0007	7	1	1	D0007
D0084	17	1	2	D0084
D0090	18	3	1	D0090
D0033	36	3	1	D0033
D0040	40	2	1	D0040
D0077	40	4	1	D0077
D0063	45	5	4	D0063
D0047	46	3	1	D0047

接着我们为了将结果数据更为直观的可视化，根据计算得出的所有批次各订单拣选总耗时与分拣工运动距离标准差，绘制如下分批号与总耗时以及分拣工运动距离标准差结果柱形图：

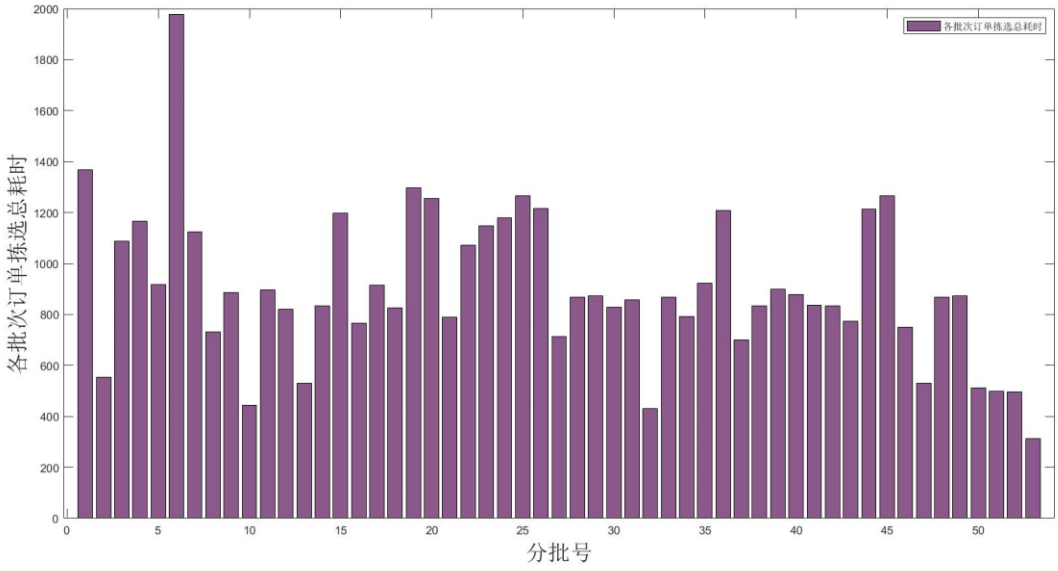

图 7 各批次订单拣选总耗时结果柱形图

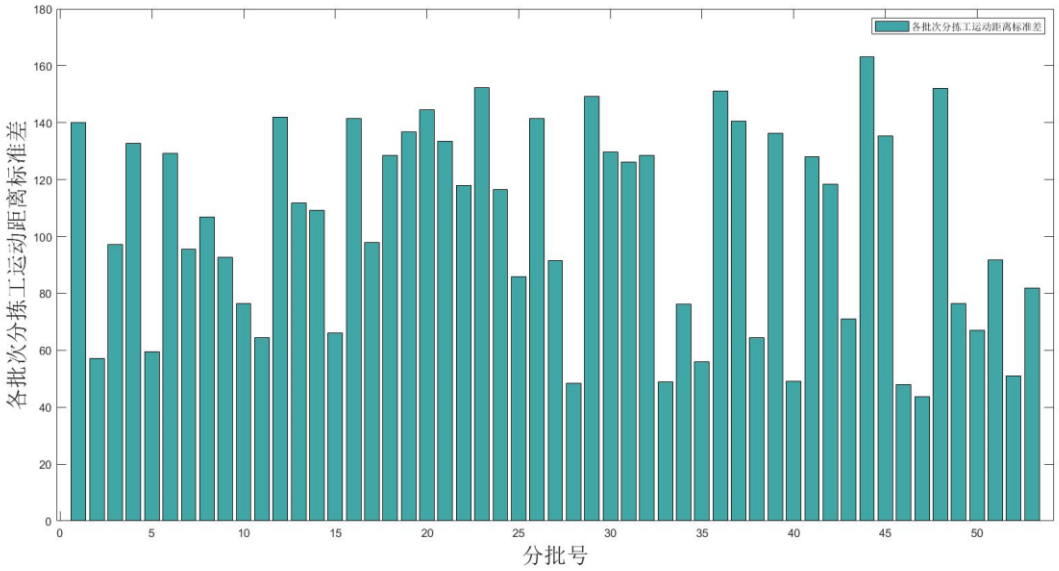

图 8 各批次分拣工运动距离标准差结果柱形图

通过观察柱形图我们可以发现，各批次订单拣选总耗时普遍较小，变化趋势较为平缓，说明该算法通过模拟指派分拣工执行任务，对完成分拣任务的分拣工选择距离他最近（即所需的运动距离最短）的订单进行指派达到了良好的效果，满足了尽快完成任务这一条件。同时各批次分拣工的运动距离标准差普遍在 200 以内，说明各批次内 5 名分拣工的运动距离较为均衡，对于转运效率具有较为明显的提升。

六、 模型评价与推广

6.1 模型的评价

6.1.1 模型优点

1) 基于相似度网络的订单分批模型较为准确地量化了任意两个订单之间的相似度，有利于分批操作的进行。
2) 基于相关度网络的货物摆放模型较为准确地量化了同一批次中任意两个货品之间的联系程度，从而达到一个整体拣选距离总和较小的结果。
3) 动态模拟指派模型通过模拟分配工作任务确保给出的指派方案耗时较小且每名分拣工的运动距离较为均衡。
4) 动态模拟指派模型可以基于当前任务完成情况给出一个最近的下一个订单，解决了静态分配容易出现的订单重复分拣以及运动距离不均衡问题。

6.1.2 模型缺点

1) 基于相似度网络的订单分批模型易受单个批次中订单数量过少影响。
2) 动态模拟指派模型在实际应用中会受到多方面因素制约。

6.2 模型的推广

本文主要设计了基于相似度网络的订单分批模型、基于相关度网络的货物摆放模型以及动态模拟指派模型，适用于电子商务、货仓管理、交通物流、企业管理等许多相关“互联网+”领域。同时动态模拟指派模型能够在尽快完成分拣任务的情况下合理均衡地指派任务，在企业规模化管理领域具有一定的参考价值。

七、 参考文献

[1]赵铁军,王玲.基于改进贪心算法的 Delta 机器人分拣路径优化[J].组合机床与自动化加工技术,2021(12):58-61+66.
[2]陈林,毕树生,李大寨,林闯,欧阳铜.基于贪心策略的直角坐标机器人动态分拣规划[J/OL].北京航空航天大学学报:1-14[2022-05-01].
[3]汪荣贵,杨娟,薛丽霞.算法设计与应用[M].机械工业出版社.2016.12.
[4]杨克昌,严权峰.算法设计与分析实用教程[M].中国水利水电出版社.2013.6.
[5]和欣,毕濯玺,侯东含.基于复合相似度的订单分批优化方法[J].信息与电脑(理论版),2021,33(08):81-83.

八、 附录

本文中除了绘图使用 MATLAB 以外，其余代码均使用 python 编写，采用 Windows 命令提示符 CMD 运行，输入”python 文件名.py”命令运行。在附录中给出的代码均在支撑材料中存放了源文件，数据文件因数据流过大全部存放于支撑材料中。下表为支撑材料说明表：

表 8 支撑材料说明表

文件名	说明
sort_order.py	订单分批模型代码，给出分批方案
total_order.py	订单分批统计代码，统计每批次的订单数量与货品种类数量
pose_good.py	货品摆放模型代码，给出货品摆放方案
total_good.py	货品摆放统计代码，给出各批次分拣任务的拣选距离总和
distribute_worker.py	分拣工指派模型代码，给出指派方案
total_worker.py	完成最终数据集（info_data.csv）的统计
result1.csv	问题一结果数据（备用）
result2.csv	问题二结果数据（备用）
result3.csv	问题三结果数据（备用）
info_data.csv	增加分批号、货架号、分拣工号以及分拣次序数据的数据集
A202203600964.docx	论文的 word 版本（备用）
A202203600964.pdf	论文的 pdf 版本（备用）
承诺书.jpg	承诺书（备用）

问题一（基于相似度网络的订单分批模型）：

代码：

sort_order.py

```
import plotly as py
from plotly import figure_factory as FF
import pandas as pd
import plotly.graph_objs as go
from plotly.offline import plot
from pandas import Series, DataFrame
pyplt = py.offline.plot
infodata = pd.read_csv('data//info_data.csv')
OrderNoLst = infodata['OrderNo'].tolist()
ItemNoLst = infodata['ItemNo'].tolist()
PC = []#批次
R = []#矩阵
C = []#解决 1
OrderNoSet = []
```

```
def EvaRi(x, y):
    xLst = []
    yLst = []
    n = 0
    for i in range(0, len(ItemNoLst)):
        if OrderNoLst[i] == x:
            xLst.append(ItemNoLst[i])
        if OrderNoLst[i] == y:
            yLst.append(ItemNoLst[i])
    for i in xLst:
        if i in yLst:
            n = n + 1
    return float(int((float(n) / float(len(xLst) + len(yLst) - n)) * 100)) /
100

def PK(OrderNumber, PosLst):
    flag = True
    for i in range(0, len(ItemNoLst)):
        if OrderNoLst[i] == OrderNumber and ItemNoLst[i] not in PosLst:
            flag = False
            break
    return flag


for x in OrderNoLst:
    if x not in OrderNoSet:
        OrderNoSet.append(x)
for i in range(0, len(OrderNoSet)):
    R.append([])
    C.append(0)
    PC.append(0)
    for j in range(0, len(OrderNoSet)):
        R[i].append(-1)


num = 0
Max = -1
MaxI = 0
MaxJ = 0
for x in OrderNoSet:
    i = int(x[2:5]) - 1
    for y in OrderNoSet:
        j = int(y[2:5]) - 1
        if i == j:
            R[i][j] = 1
        elif R[i][j] == -1:
```

```
            R[i][j] = EvaRi(x, y)
            R[j][i] = R[i][j]
        if R[i][j] > Max and R[i][j] != 1:
            Max = R[i][j]
            MaxI = i
            MaxJ = j
    num = num + 1
    if num % 10 == 0:
        print(str(num) + '已完成')


SocLst = [MaxI, MaxJ]#已决订单
PosLst = []#已放货架
C[MaxI] = 1
C[MaxJ] = 1
PC[MaxI] = 1
PC[MaxJ] = 1
N = 0#货架
P = 1#批次
pn = 2#解决订单数
xLst = []
yLst = []
if 0 <= MaxI + 1 and MaxI + 1 <= 9:
    x = 'D000' + str(MaxI + 1)
elif 10 <= MaxI + 1 and MaxI + 1 <= 99:
    x = 'D00' + str(MaxI + 1)
else:
    x = 'D0' + str(MaxI + 1)
if 0 <= MaxJ + 1 and MaxJ + 1 <= 9:
    y = 'D000' + str(MaxJ + 1)
elif 10 <= MaxJ + 1 and MaxJ + 1 <= 99:
    y = 'D00' + str(MaxJ + 1)
else:
    y = 'D0' + str(MaxJ + 1)
for i in range(0, len(ItemNoLst)):
    if OrderNoLst[i] == x:
        xLst.append(ItemNoLst[i])
    if OrderNoLst[i] == y:
        yLst.append(ItemNoLst[i])
for i in xLst:
    if i not in yLst and N < 200:
        N = N + 1
        PosLst.append(i)
for i in yLst:
    if N < 200:
        N = N + 1
```

```
        PosLst.append(i)

while pn < len(OrderNoSet):
    Max = -1
    MaxI = 0
    MaxJ = 0
    for x in SocLst:
        for y in range(0, len(OrderNoSet)):
            if (R[x][y] > Max or Max == -1) and R[x][y] != 1 and C[y] == 0:
                Max = R[x][y]
                MaxI = x
                MaxJ = y
    if 1 <= MaxJ + 1 and MaxJ + 1 <= 9:
        y = 'D000' + str(MaxJ + 1)
    elif 10 <= MaxJ + 1 and MaxJ + 1 <= 99:
        y = 'D00' + str(MaxJ + 1)
    else:
        y = 'D0' + str(MaxJ + 1)
    e = -1
    for i in range(0, len(ItemNoLst)):
        if OrderNoLst[i] == y and ItemNoLst[i] not in PosLst:
            if N >= 200:
                e = 0
                break
            PosLst.append(ItemNoLst[i])
            N = N + 1
    if N < 200 or (N == 200 and e == -1):
        SocLst.append(MaxJ)
        pn = pn + 1
        C[MaxJ] = 1
        PC[MaxJ] = P
    else:
        for i in range(0, len(OrderNoSet)):
            if  C[i]  ==  0  and  PK(OrderNoSet[i],  PosLst)  and  pn  <
len(OrderNoSet):
                pn = pn + 1
                C[i] = 1
                PC[i] = P
        N = 0#货架
        PosLst = []#已放货架
        SocLst = [MaxJ]
        C[MaxJ] = 1
        pn = pn + 1
        P = P + 1#批次
        PC[MaxJ] = P
        if 0 <= MaxJ + 1 and MaxJ + 1 <= 9:
```

```
            y = 'D000' + str(MaxJ + 1)
        elif 10 <= MaxJ + 1 and MaxJ + 1 <= 99:
            y = 'D00' + str(MaxJ + 1)
        else:
            y = 'D0' + str(MaxJ + 1)
        for i in range(0, len(ItemNoLst)):
            if OrderNoLst[i] == y and N < 200:
                PosLst.append(ItemNoLst[i])
                N = N + 1
    if pn % 100 == 0:
        print(str(pn) + '已完成')
print(P)


#导出文件
outdata = {'OrderNo': OrderNoSet, 'GroupNo': PC}
df = DataFrame(outdata)
df.to_csv('result_data1.csv')
```

total_order.py

```
import plotly as py
from plotly import figure_factory as FF
import pandas as pd
import plotly.graph_objs as go
from plotly.offline import plot
from pandas import Series, DataFrame
pyplt = py.offline.plot
infodata = pd.read_csv('data//info_data.csv')
OrderNoLst = infodata['OrderNo'].tolist()
ItemNoLst = infodata['ItemNo'].tolist()
QuantityLst = infodata['Quantity'].tolist()
result1 = pd.read_csv('data//result1.csv')
OrderNoSet = result1['OrderNo'].tolist()
GroupNoLst = result1['GroupNo'].tolist()

PC = []
orderNum = []
itemNum = []

for i in range(1, 54):
    orderTemp = []
    itemTemp = []
    PC.append(i)
    for j in range(0, len(GroupNoLst)):
        if GroupNoLst[j] == i:
            orderTemp.append(OrderNoSet[j])
```

```
            for k in range(0, len(OrderNoLst)):
                if OrderNoSet[j] == OrderNoLst[k]:
                    if ItemNoLst[k] not in itemTemp:
                        itemTemp.append(ItemNoLst[k])
    orderNum.append(len(orderTemp))
    itemNum.append(len(itemTemp))

outdata = {'GroupNo': PC, 'OrderNum': orderNum, 'ItemNum': itemNum}
df = DataFrame(outdata)
df.to_csv('total_data1.csv')


Group = []
for i in range(0, len(OrderNoLst)):
    k = -1
    for j in range(0, len(OrderNoSet)):
        if OrderNoSet[j] == OrderNoLst[i]:
            k = GroupNoLst[j]
            break
    Group.append(k)

outdata = {'OrderNo': OrderNoLst, 'ItemNo': ItemNoLst, 'Quantity':
QuantityLst, 'GroupNo': Group}
df = DataFrame(outdata)
df.to_csv('info_data1.csv')
```

问题二（基于相关度网络的货物摆放模型）：

代码：

pose_good.py

```
import plotly as py
from plotly import figure_factory as FF
import pandas as pd
import plotly.graph_objs as go
from plotly.offline import plot
from pandas import Series, DataFrame
pyplt = py.offline.plot
infodata = pd.read_csv('data//info_data1.csv')
OrderNoLst = infodata['OrderNo'].tolist()
ItemNoLst = infodata['ItemNo'].tolist()
GroupNoLst = infodata['GroupNo'].tolist()
GroupItemSet_out = []
GroupNoData = []
PC_out = []
```

```
def FindOrder(group, DataLst):
    for i in range(0, len(GroupNoLst)):
        if GroupNoLst[i] == group and OrderNoLst[i] not in DataLst:
            DataLst.append(OrderNoLst[i])

def FindItem(GroupOrderLst, DataLst, DataSet):
    for i in range(0, len(GroupOrderLst)):
        DataLst.append([])
        for j in range(0, len(OrderNoLst)):
            if OrderNoLst[j] == GroupOrderLst[i]:
                DataLst[i].append(ItemNoLst[j])
                if ItemNoLst[j] not in DataSet:
                    DataSet.append(ItemNoLst[j])

def GetPos(GroupItemSet, key):
    p = 0
    for i in range(0, len(GroupItemSet)):
        if GroupItemSet[i] == key:
            p = i
            break
    return p


for g in range(1, 54):
    Max = -1
    MaxI = 0
    MaxJ = 0
    GroupOrderLst = []
    GroupItemLst = []
    GroupItemSet = []
    C = []
    PC = []
    FindOrder(g, GroupOrderLst)
    FindItem(GroupOrderLst, GroupItemLst, GroupItemSet)
    V = []
    for i in range(0, len(GroupItemSet)):
        V.append([])
        C.append(0)
        PC.append(0)
        for j in range(0, len(GroupItemSet)):
            V[i].append(0)
    for i in range(0, len(GroupItemLst)):
        for j in range(0, len(GroupItemLst[i])):
            x = GetPos(GroupItemSet, GroupItemLst[i][j])
            for k in range(j + 1, len(GroupItemLst[i])):
                y = GetPos(GroupItemSet, GroupItemLst[i][k])
```

```
                V[x][y] += 1
                if V[x][y] > Max or Max == -1:
                    Max = V[x][y]
                    MaxI = x
                    MaxJ = y
    GroupNoData.append(g)
    GroupNoData.append(g)
    PosLst = [MaxI, MaxJ]
    pn = 2
    C[MaxI] = 1
    C[MaxJ] = 1
    PC[MaxI] = 1
    PC[MaxJ] = 2
    PS = 3
    while pn < len(GroupItemSet):
        Max = -1
        MaxI = 0
        MaxJ = 0
        for i in PosLst:
            for j in range(0, len(GroupItemSet)):
                if (V[i][j] > Max or Max == -1) and C[j] == 0:
                    Max = V[i][j]
                    MaxI = i
                    MaxJ = j
        pn = pn + 1
        PosLst.append(MaxJ)
        C[MaxJ] = 1
        PC[MaxJ] = PS
        PS = PS + 1
        GroupNoData.append(g)

    for x in GroupItemSet:
        GroupItemSet_out.append(x)
    for x in PC:
        PC_out.append(x)
    print('批次' + str(g) + '已完成')

#导出文件
outdata = {'ItemNo': GroupItemSet_out, 'GroupNo': GroupNoData, 'ShelfNo':
PC_out}
df = DataFrame(outdata)
df.to_csv('result_data2.csv')
```

total_good.py

```
import plotly as py
from plotly import figure_factory as FF
```

```
import pandas as pd
import plotly.graph_objs as go
from plotly.offline import plot
from pandas import Series, DataFrame
pyplt = py.offline.plot
infodata = pd.read_csv('data//info_data2.csv')
OrderNoLst = infodata['OrderNo'].tolist()
ItemNoLst = infodata['ItemNo'].tolist()
GroupNoLst = infodata['GroupNo'].tolist()
ShelfNoLst = infodata['ShelfNo'].tolist()

def FindGroup(OrderNo):
    group = 1
    for i in range(0, len(OrderNoLst)):
        if OrderNoLst[i] == OrderNo:
            group = GroupNoLst[i]
            break
    return group

def EvaGroup(OrderNo):
    Max = -1
    Min = -1
    for i in range(0, len(OrderNoLst)):
        if OrderNoLst[i] == OrderNo:
            if Max == -1 or ShelfNoLst[i] > Max:
                Max = ShelfNoLst[i]
            if Min == -1 or ShelfNoLst[i] < Min:
                Min = ShelfNoLst[i]
    return Max - Min

OrderNoSet = []
GroupSumLst = []
GroupNoLst_out = []
for i in range(0, 53):
    GroupSumLst.append(0)
    GroupNoLst_out.append(i + 1)
for i in range(0, len(OrderNoLst)):
    if OrderNoLst[i] not in OrderNoSet:
        OrderNoSet.append(OrderNoLst[i])

#各批次的订单拣选距离总和
for i in range(0, len(OrderNoSet)):
    g = FindGroup(OrderNoSet[i])
    GroupSumLst[g - 1] += EvaGroup(OrderNoSet[i])
Sum = 0
for x in GroupSumLst:
```

```
    Sum += x
print(Sum)

outdata = {'GroupNo': GroupNoLst_out, 'GroupSum': GroupSumLst}
df = DataFrame(outdata)
df.to_csv('total_data2.csv')


'''
#带货架号的数据集
infodata = pd.read_csv('data//info_data1.csv')
OrderNoLst = infodata['OrderNo'].tolist()
ItemNoLst1 = infodata['ItemNo'].tolist()
GroupNoLst1 = infodata['GroupNo'].tolist()
QuantityLst = infodata['Quantity'].tolist()
result1 = pd.read_csv('data//result2.csv')
ItemNoLst2 = result1['ItemNo'].tolist()
GroupNoLst2 = result1['GroupNo'].tolist()
ShelfNoLst = result1['ShelfNo'].tolist()

ShelfNo = []
for i in range(0, len(OrderNoLst)):
    for j in range(0, len(ShelfNoLst)):
        if ItemNoLst1[i] == ItemNoLst2[j] and GroupNoLst1[i] ==
GroupNoLst2[j]:
            ShelfNo.append(ShelfNoLst[j])
            break

outdata = {'OrderNo': OrderNoLst, 'ItemNo': ItemNoLst1, 'Quantity':
QuantityLst, 'GroupNo': GroupNoLst1, 'ShelfNo': ShelfNo}
df = DataFrame(outdata)
df.to_csv('info_data2.csv')
'''
```

问题三（动态模拟指派模型）：

代码：

distribute_worker.py

```
import plotly as py
from plotly import figure_factory as FF
import pandas as pd
import plotly.graph_objs as go
from plotly.offline import plot
from pandas import Series, DataFrame
pyplt = py.offline.plot
```

```
infodata = pd.read_csv('data//info_data2.csv')
OrderNoLst = infodata['OrderNo'].tolist()
ItemNoLst = infodata['ItemNo'].tolist()
GroupNoLst = infodata['GroupNo'].tolist()
ShelfNoLst = infodata['ShelfNo'].tolist()
GroupNo = []
OrderNo = []
WorkNo = []
RangeNo = []
GroupNo_out = []
T_out = []
Worker1 = []
Worker2 = []
Worker3 = []
Worker4 = []
Worker5 = []

def NumOrder(g, OrderNoSet, OrderCilck):
    for i in range(0, len(OrderNoLst)):
        if GroupNoLst[i] == g:
            if OrderNoLst[i] not in OrderNoSet:
                OrderNoSet.append(OrderNoLst[i])
                OrderCilck.append(0)
    return len(OrderNoSet)

def RangeOrder(order,  OrderNoSet):
    r = 0
    for i in range(0, len(OrderNoSet)):
        if OrderNoSet[i] == order:
            r = i
            break
    return r

def FindMinTime(w_time):
    Min = w_time[0]
    for i in range(1, 5):
        if w_time[i] < Min:
            Min = w_time[i]
    return Min

def FindMaxTime(w_time):
    Max = w_time[0]
    for i in range(1, 5):
        if w_time[i] > Max:
            Max = w_time[i]
    return Max
```

```
def GetTime(g, pose, order):
    MaxP = -1
    MinP = -1
    time = 0
    pp = 0
    for i in range(0, len(OrderNoLst)):
        if OrderNoLst[i] == order:
            if ShelfNoLst[i] > MaxP or MaxP == -1:
                MaxP = ShelfNoLst[i]
            if ShelfNoLst[i] < MinP or MinP == -1:
                MinP = ShelfNoLst[i]
    if MinP >= pose:
        pp = MaxP
        time = MaxP - pose
    elif MaxP <= pose:
        pp = MinP
        time = pose - MinP
    else:
        if MaxP - pose >= MinP - pose:
            pp = MaxP
            time = MaxP - MinP + pose - MinP
        else:
            pp = MinP
            time = MaxP - MinP + MaxP - pose
    return time, pp

def FindOrder(g, pose, OrderNoSet, OrderCilck):
    MinTime = -1
    wp = 0
    order = OrderNoSet[0]
    for i in range(0, len(OrderNoLst)):
        if GroupNoLst[i] == g and OrderNoLst[i] in OrderNoSet and
OrderCilck[RangeOrder(OrderNoLst[i], OrderNoSet)] == 0:
            tmp, pmp = GetTime(g, pose, OrderNoLst[i])
            if tmp < MinTime or MinTime == -1:
                MinTime = tmp
                wp = pmp
                order = OrderNoLst[i]
    return MinTime, wp, order


for g in range(1, 54):
    T = 0
    w_num = [0, 0, 0, 0, 0]
    w_time = [0, 0, 0, 0, 0]
```

```
    w_pose = [1, 1, 1, 1, 1]
    w_numtime = [0, 0, 0, 0, 0]
    OrderNoSet = []
    OrderCilck = []
    Num = NumOrder(g, OrderNoSet, OrderCilck)
    key = 0
    while key < Num:
        MinTime = FindMinTime(w_time)
        T += MinTime
        for i in range(0, 5):
            w_time[i] -= MinTime
            if w_time[i] == 0:
                w_time[i], w_pose[i], order = FindOrder(g, w_pose[i],
OrderNoSet, OrderCilck)
                w_num[i] += 1
                w_numtime[i] += w_time[i]
                OrderCilck[RangeOrder(order, OrderNoSet)] = 1
                key += 1
                GroupNo.append(g)
                OrderNo.append(order)
                WorkNo.append(i + 1)
                RangeNo.append(w_num[i])
    MaxTime = FindMaxTime(w_time)
    T += MaxTime
    GroupNo_out.append(g)
    T_out.append(T)
    Worker1.append(w_numtime[0])
    Worker2.append(w_numtime[1])
    Worker3.append(w_numtime[2])
    Worker4.append(w_numtime[3])
    Worker5.append(w_numtime[4])
    print('批次' + str(g) + '已完成')

#导出文件
outdata = {'OrderNo': OrderNo, 'GroupNo': GroupNo, 'WorkerNo': WorkNo,
'TaskNo': RangeNo}
df = DataFrame(outdata)
df.to_csv('result_data3.csv')
outdata = {'GroupNo': GroupNo_out, 'TotalTime': T_out, 'Worker1': Worker1,
'Worker2': Worker2, 'Worker3': Worker3, 'Worker4': Worker4, 'Worker5':
Worker5}
df = DataFrame(outdata)
df.to_csv('tatal_data3.csv')
```

total_worker.py

```
import plotly as py
from plotly import figure_factory as FF
import pandas as pd
import plotly.graph_objs as go
from plotly.offline import plot
from pandas import Series, DataFrame
pyplt = py.offline.plot
infodata = pd.read_csv('data//info_data2.csv')
OrderNoLst1 = infodata['OrderNo'].tolist()
ItemNoLst = infodata['ItemNo'].tolist()
GroupNoLst1 = infodata['GroupNo'].tolist()
QuantityLst = infodata['Quantity'].tolist()
ShelfNoLst = infodata['ShelfNo'].tolist()
WorkerNo = []
TaskNo = []
result3 = pd.read_csv('data//result3.csv')
OrderNoLst2 = result3['OrderNo'].tolist()
GroupNoLst2 = result3['GroupNo'].tolist()
WorkerNoLst = result3['WorkerNo'].tolist()
TaskNoLst = result3['TaskNo'].tolist()


for i in range(0, len(OrderNoLst1)):
    for j in range(0, len(WorkerNoLst)):
        if OrderNoLst1[i] == OrderNoLst2[j] and GroupNoLst1[i] ==
GroupNoLst2[j]:
            WorkerNo.append(WorkerNoLst[j])
            TaskNo.append(TaskNoLst[j])
            break

outdata = {'OrderNo': OrderNoLst1, 'ItemNo': ItemNoLst, 'Quantity':
QuantityLst, 'GroupNo': GroupNoLst1, 'WorkerNo': WorkerNo, 'TaskNo': TaskNo}
df = DataFrame(outdata)
df.to_csv('info_data3.csv')
```