在线零售商的订单履行建模

摘要

多品订单包含多个品类的 SKU，往往需要通过订单拆分进行处理，对于在线零售商，如何在综合考虑总成本与客户满意度的情况下采取合适的订单履行方案至关重要。本文建立了基于订单的非线性 0-1 规划模型对仓库的 SKU 种类进行合理分配，利用基于成本的订单履行方案决策模型给出了仅考虑总成本时的最优配送方案，最后基于客户倾向收到更少包裹，在模型基础上添加由包裹数决定的惩罚成本，构建了基于成本与满意度的订单履行方案决策模型。

针对问题一，首先对订单数据的 SKU 进行相关性分析，得出 SKU 之间基本没有相关性，客户不存在对商品的倾向性，每次购买的商品数也呈均匀分布。以每个仓库中是否储存某一类商品为 0-1 变量，以各仓库最大储存量 70 种和每类商品至少存储在 1 个仓库中为约束条件，派送的包裹总数为目标函数，构建**基于订单的非线性 0-1 规划模型**。本模型中有 300 个变量且为非线性，故采用**遗传算法**求型，得到最终派送的最少总包裹数为 **75540**，并给出了仓库存储 SKU 的种类，部分结果如下：

SKU 种类	1	2	3	4	5	6	7	8	9	10	.....	95	96	97	98	99	100
仓库 1	1	0	1	0	1	1	1	1	1	0	.....	0	1	1	1	0	1
仓库 2	1	1	1	1	1	1	0	0	1	1	.....	1	1	0	1	1	0
仓库 3	0	1	0	1	1	0	1	1	0	1	.....	1	1	1	1	1	1

针对问题二，基于 SKU 转运费用与包裹运输成本的表达式，提出 7 种备选的订单履行方案，分别计算不同方案的成本，以履行单个订单成本为目标函数构建**基于成本的订单履行方案决策模型**。因为订单之间基本没有关联，故对每个订单分别求出最优方案，最后进行汇总。代入问题二的参数对 50000 个订单数据的履行方案求解，得到派送的包裹总数为 **50000**，订单履行总成本为 **581926.9 元**。

针对问题三，基于问题 2 仓库储存 SKU 的分布情况，计算不考虑转运时的订单履行方案最低成本和总订单数。在问题 2 构建的基于成本的决策模型基础上，考虑客户倾向于接收较少的包裹，增加由包裹数量决定的惩罚成本，构建**基于成本与满意度的订单履行方案决策模型**。重新求解问题二并对比了转运与不考虑转运两种条件下的总成本与满意度，得出**不考虑转运时订单履行总成本显著增加，而客户满意度随着派送包裹数的增多而显著降低**。此外，随机生成均匀分布的 100000 条 SKU 订单与客户位置坐标，求解并分析结果，所得最优方案与原数据相同，说明模型的稳健性。

关键字：非线性 0-1 规划　遗传算法　决策模型　订单转运

目录

一、 问题重述

1.1 问题背景

中国是世界上网络零售交易最活跃的国家，每天都将产生大量的网络订单数据。网络交易中涉及种类繁多的商品，一个订单通常包含多种商品 (SKU)，这种订单称为多品订单。目前在线零售商主要采用基于拆分的方式来完成多品订单，即根据订单中 SKU 在多个仓库的仓储情况拆分成若干个子订单，每个子订单在对应的仓库进行作业并派送给顾客。此外，为尽量降低订单的配送成本，零售商可能会将分散在不同仓库的 SKU 集中转运至某个仓库后再合并打包，派送给顾客。

为解决订单拆分带来的成本高、污染性强、顾客满意度低等问题，合理分配仓库的 SKU 种类与订单转运是重要的方法。本文以履行订单的总成本和顾客满意度为出发点，讨论在线零售商履行订单过程中的优化决策问题。

1.2 问题提出

某在线零售商负责销售 100 种 SKU，且在某个地区内有三个仓库，每个仓库的存储能力一定。现根据提供的订单数据与仓库数据，分配 50000 个客户的订单，使得履行订单的总成本与顾客满意度满足要求。本文根据题目条件，对数据进行分析，并运用数学思想建立模型讨论如下问题：

- **问题一**　根据订单数据，在不考虑订单转运的条件下确定每个仓库中 SKU 的种类，使得履行订单时产生的包裹总数最少。
- **问题二**　根据订单数据与仓库数据，在考虑订单转运的条件下，得出履行订单总成本的最小值。
- **问题三**　在考虑顾客满意度和履行订单总成本的条件下给出订单的分配方案，并且对是否进行订单转运进行讨论。

二、 模型假设

- 在可预见的一段时间内，在线零售商面临的销售环节不会产生较大变化。
- 不考虑标记拆单，即 SKU 因为自有属性而导致拆单的情况。
- 仓库中储存的 SKU 数量充足，不会发生缺货。
- 所有的转运都能在时效要求内完成，因为仓库之间距离较近。

三、 符号说明

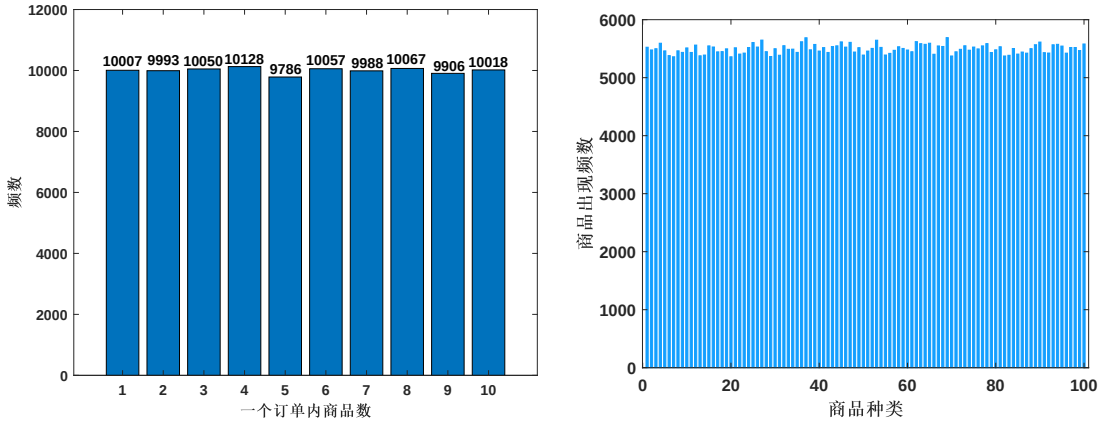
符号	含义	单位
W_i	仓库 i	
W_i^p	仓库 i 缺货时，优先给其补货的仓库	
W_k^o	仓库 i 和仓库 j 缺货时，仓库 k 优先补货的对象	
n	订单包含的 SKU 品类个数	
n_i	仓库 i 提供的 SKU 品类个数	
$P(x)$	包裹运输成本	元
t_i^p	仓库 i 与其优先补货仓库之间的转运成本	元
t_k^o	仓库 k 与其优先补货对象之间的转运成本	元
C	订单的总成本	元
s_i	决策方案，取 1 表示采取该方案，取 0 表示不采取该方案	
t_{ij}	仓库之间的转运费用	元
p_0	基础包裹运输费用	元
a	基础包裹运输距离	公里
p_1	额外运输费用增长系数	元/公里

四、 问题一模型的建立及求解

根据附件一给出的 50000 个订单数据，对 3 个仓库所储存的 SKU 商品种类进行合理的分配，使得履行订单时产生的包裹总数最少。首先对订单的数据进行相关性分析，统计结果说明各 SKU 商品之间几乎没有相关性。又考虑到本问题为非线性 0-1 规划问题，变量多且情况复杂，因此选用以二进制编码方式工作的遗传算法求解本问，得到三个仓库的 SKU 储存种类和履行 50000 个订单后产生的最少的包裹总数。

4.1 各订单间相关性分析

根据附件一给出的订单数据，统计每个订单内 SKU 个数的频数和 100 种 SKU 在全部订单中的频数：



(a) 订单内 SKU 个数的频数统计

(b) 订单内 SKU 品类数的频数统计

图 1　问题一订单 SKU 频数统计

由图可知，每个订单包含的 SKU 个数在 5000 左右，可认为在 [1,10] 内均匀分布；SKU 在所有订单中出现的次数在 2700 左右，可认为在 [1,100] 内均匀分布。不同 SKU 个数的订单数量相互接近，且不同 SKU 商品种类在订单中出现的次数也近似相等，由此说明订单数据的 SKU 之间基本没有关联。

为进一步说明 SKU 之间的相关性，利用皮尔逊相关系数刻画 SKU 间的相关关系。

皮尔逊相关系数的定义为：对于矩阵 X 中的 X_a 列和矩阵 Y 中的 Y_b 列，均值的含义分别为 $\overline{X_a}=\frac{1}{n}\sum_{i=1}^{n}(X_{a,i})$ 和 $\overline{Y_b}=\frac{1}{n}\sum_{j=1}^{n}(Y_{b,j})$，则皮尔逊相关系数 $\rho(a,b)$ 表示为

$$\rho(a,b)=\frac{\sum_{i=1}^{n}(X_{a,i}-\overline{X_a})(Y_{b,i}-\overline{Y_b})}{\sqrt{\sum_{i=1}^{n}(X_{a,i}-\overline{X_a})^2\sum_{j=1}^{n}(Y_{b,j}-\overline{Y_b})^2}}$$

其中 n 是列的长度，此处为 100 种 SKU，故 $n=100$。根据皮尔逊相关系数画出 100 种 SKU 间相关性图 2。在皮尔逊相关系数图2中，颜色越浅表示相关性越弱。由图可知，除对角线外，两两 SKU 之间相关系数几乎为 0，由此说明订单数据的 SKU 之间基本没有相关性，与订单 SKU 频数统计结果一致。

由以上相关性分析结果可知各 SKU 商品种类之间没有相关性，不能用常规的商品热销商算法求解本问题；又考虑到本问题是非线性 0-1 规划问题，共有 300 个变量，2^300 种情况，计算量大，没有固定的结构与参数，先验辅助信息较少，而遗传算法以二进制编码的方式工作，适合于求解复杂的优化问题，因此舍弃传统优化算法而选用遗传算法求解。

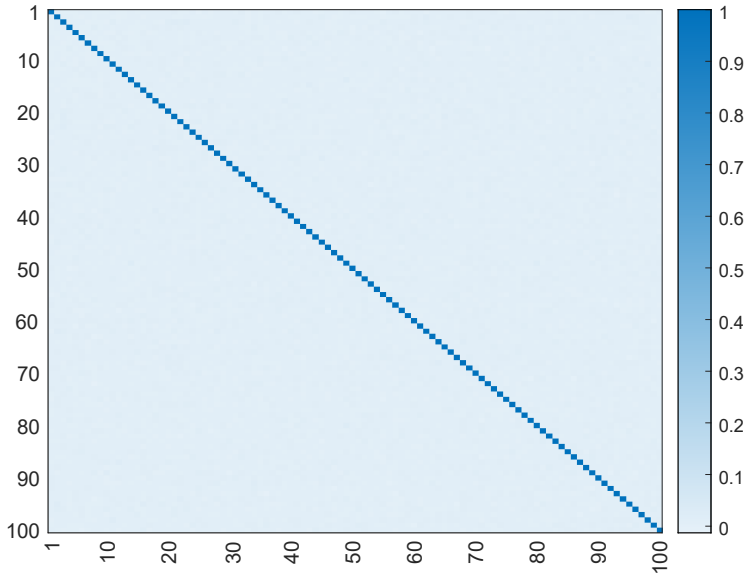


图 2　SKU 间相关关系

4.2 基于订单的非线性 0-1 规划模型

附件一中给出的订单数据，包括 50000 个订单的 SKU 信息，求解每个仓库中 SKU 的种类分布，使得履行订单时产生的包裹数最少。设变量 $x_{i,j}$，$x_{i,j}=1$ 表示第 i 个仓库中有第 j 种商品，而 $x_{i,j}=0$ 则表示第 i 个仓库中没有第 j 种商品。对于第 k 个订单中有 N_k 种 SKU，分别为 $u_{k,1},u_{k,2},...,u_{k,N_k}$。设第 k 个订单需要的最少包裹数为 y_k。

对于第 k 个订单，当 $\sum_{j=u_{k,1}}^{u_{k,N_k}} x_{i,j}=N_k$ 时，第 i 个仓库的 SKU 可以满足该订单，此时仅需要 1 个包裹就可以完成该订单，即 $y_k=1$。当 $\sum_{j=u_{k,1}}^{u_{k,N_k}} x_{i,j}<N_k, i=1,2,3$ 时，每一个仓库单独的 SKU 库存都不能满足该订单，则考虑两个仓库的 SKU 是否能满足。若 $\prod_{j=u_{k,1}}^{u_{k,N_k}}(x_{1,j}+x_{2,j})\neq 0$ 则第 $1,2$ 仓库的 SKU 库存可以满足第 k 个订单。同理，当 $\prod_{j=u_{k,1}}^{u_{k,N_k}}(x_{1,j}+x_{3,j})\neq 0$ 时仓库 $1,3$ 可以满足第 k 个订单；当 $\prod_{j=u_{k,1}}^{u_{k,N_k}}(x_{2,j}+x_{3,j})\neq 0$ 时仓库 $2,3$ 可以满足第 k 个订单。当需要两个仓库才可以满足第 k 个订单时，$y_k=2$。均不满足上述条件时，即 $\prod_{j=u_{k,1}}^{u_{k,N_k}}(x_{1,j}+x_{2,j})+\prod_{j=u_{k,1}}^{u_{k,N_k}}(x_{1,j}+x_{3,j})+\prod_{j=u_{k,1}}^{u_{k,N_k}}(x_{2,j}+x_{3,j})=0$ 时，需要三个仓库的 SKU 才能够满足，此时 $y_k=3$。所以第 k 个订单需要的最少包裹数满足：

$$y_k=\begin{cases}1 & ,\sum_{j=u_{k,1}}^{u_{k,N_k}} x_{i,j}=N_k\\ 2 & ,\prod_{j=u_{k,1}}^{u_{k,N_k}}(x_{1,j}+x_{2,j})+\prod_{j=u_{k,1}}^{u_{k,N_k}}(x_{1,j}+x_{3,j})+\prod_{j=u_{k,1}}^{u_{k,N_k}}(x_{2,j}+x_{3,j})>0\\ 3 & ,\prod_{j=u_{k,1}}^{u_{k,N_k}}(x_{1,j}+x_{2,j})+\prod_{j=u_{k,1}}^{u_{k,N_k}}(x_{1,j}+x_{3,j})+\prod_{j=u_{k,1}}^{u_{k,N_k}}(x_{2,j}+x_{3,j})=0\end{cases}$$

对于附件一共有 $M=50000$ 个订单，所以包裹数 Y 满足：

$$Y=\sum_{k=1}^{M} y_k \tag{4.2.1}$$

题中要求每一个仓库储存的 SKU 不超过 70 种，因此有约束条件:

$$\sum_{i=1}^{100} x_{i,j} \leq 70, i=1,2,3 \tag{4.2.2}$$

每一种商品至少需要储存在一个仓库中，因此有约束条件：

$$\sum_{i=1}^{3} x_{i,j} \geq 1, j=1,2,...,100 \tag{4.2.3}$$

由上述4.2.1、4.2.2、4.2.3式可以得到如下非线性 0-1 规划模型：

$$\text{min} \quad Y=\sum_{k=1}^{M} y_k$$

$$\text{s.t.} \quad \begin{cases} \sum\limits_{i=1}^{100} x_{i,j} \leq 70 & ,i=1,2,3 \\ \sum\limits_{i=1}^{3} x_{i,j} \geq 1 & ,j=1,2,...,100 \\ x_{i,j}=0\text{或}1 & ,i=1,2,3 \quad j=1,2,...,100 \end{cases}$$

$$\text{其中 } y_k=\begin{cases} 1 & ,\sum\limits_{j=u_{k,1}}^{u_{k,N_k}} x_{i,j}=N_k \\ 2 & ,\prod\limits_{j=u_{k,1}}^{u_{k,N_k}} (x_{1,j}+x_{2,j})+\prod\limits_{j=u_{k,1}}^{u_{k,N_k}} (x_{1,j}+x_{3,j})+\prod\limits_{j=u_{k,1}}^{u_{k,N_k}} (x_{2,j}+x_{3,j})>0 \\ 3 & ,\prod\limits_{j=u_{k,1}}^{u_{k,N_k}} (x_{1,j}+x_{2,j})+\prod\limits_{j=u_{k,1}}^{u_{k,N_k}} (x_{1,j}+x_{3,j})+\prod\limits_{j=u_{k,1}}^{u_{k,N_k}} (x_{2,j}+x_{3,j})=0 \end{cases}$$

4.3 求解非线性 0-1 规划模型的遗传算法

遗传算法是一种计算数学中用于求解最优化问题的智能进化算法，基于进化生物学中如遗传、变异、杂交、自然选择等经典现象发展而来。遗传算法从一个较大的种群出发开始搜索，覆盖范围大，选择其中函数值较大的染色体作为亲代，经过交叉重组、基因突变后作为子代再次计算函数值，在保证选择出函数最大值的情况下扩大搜索范围，简单通用且具有很强的鲁棒性。遗传算法中变量采用二进制编码，完全吻合本文中求解的非线性 0-1 规划模型中自变量的要求，因此是求解该模型的良好方法。

问题 1 中总包裹数小于等于 150000 个，而遗传算法对于求解最大化问题有着优异的性能，所以令适应度函数为 $Z=150000-Y$。

- **初始化种群**

  取遗传算法的种群规模 $Pop = 500$, 随机生成 500 组 0-1 随机变量 $x_{i,j}, i = 1, 2, 3 \quad j = 1, 2, ..., 100$，并使其满足约束关系：

$$\begin{cases} \sum_{i=1}^{100} x_{i,j} \leq 70 & , i = 1, 2, 3 \\ \sum_{i=1}^{3} x_{i,j} \geq 1 & , j = 1, 2, ..., 100 \\ x_{i,j} = 0\text{或}1 & , i = 1, 2, 3 \quad j = 1, 2, ..., 100 \end{cases}$$

- **自然选择**

  给定的种群中每一条染色体都是一个组解，求解给定条件下的适应度函数 $Z = 150000 - Y = 150000 - \sum_{k=1}^{M} y_k$，并对其排序。设亲代和子代间的代沟为 Gap，即选择亲代形成子代的比例，选择个数为 $Nsel = Pop \cdot Gap$ 。而第 l 个染色体的适应度为 Z_l 则其被选中的概率为 $Select_l = \frac{Z_l}{\sum_{l=1}^{Pop} Z_l}$。采用随机数的方式挑选其中前 $Pop - Nsel$ 个染色体直接作为子代，前 $Nsel$ 个染色体进行交叉互换、变异处理。

- **交叉重组**

  模仿真核生物有丝分裂过程中染色体的联会、交叉互换过程，对选择出的 $Nsel$ 个染色体随机两两配对，随机选择自变量片段成对交换，构成交叉重组后的染色体。

- **基因突变**

  模仿生物中基因突变过程，对交叉重组后的 $Nsel$ 个染色体，随机选取若干染色体上的若干位点，对 0-1 变量取进行取非操作，即将 $x_{i,j}$ 赋值为 $\neg x_{i,j}$。

- **适者生存**

  判断经过交叉重组、基因突变两过程的 $Nsel$ 个染色体是否满足约束条件。对于不满足约束条件的染色体进行随机变异操作，即对于仓库储存商品种类超过 70 的随机剔除超过部分，对于不满足至少存在一个仓库的商品种类，随机挑选一个仓库储存。

- **构成新种群**

  将自然选择中挑选出的前 $Pop - Nsel$ 个染色体与经过交叉重组、基因突变和适者生存过程的 $Nsel$ 个染色体重新组合成新的种群，进行新一轮的遗传迭代。

- **退出条件**

  当前的全局最优解已经稳定存在 30 代以上时，可以认为这是本轮遗传算法找到的最优解，可以退出遗传迭代，输出最优解和最大值，否则认为还未收敛，将继续进行遗传迭代。

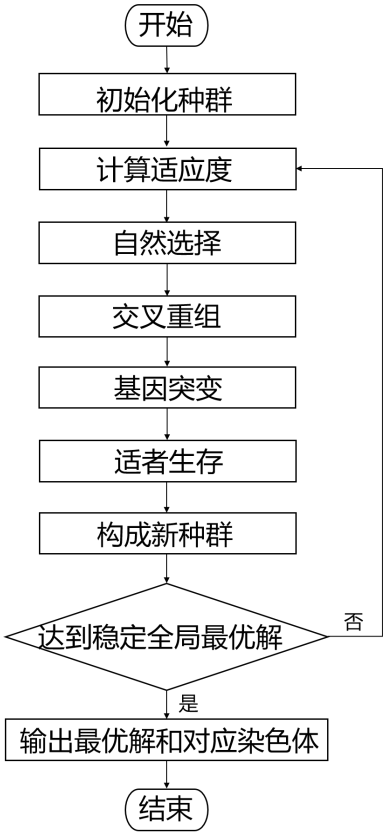


图 3　遗传算法流程图

4.4 问题一结果分析

如图4所示，由遗传算法经过 395 次遗传迭代，维持稳定的当前最优解 30 代，认为已经收敛到全局最优解，故退出循环。此时的包裹数为 75540，部分仓库存储商品的种类 $x_{i,j}$ 如表 1所示：

表 1　问题 1 最优解

SKU 种类	1	2	3	4	5	6	7	8	9	10	.....	95	96	97	98	99	100
仓库 1	1	0	1	0	1	1	1	1	1	0	.....	0	1	1	1	0	1
仓库 2	1	1	1	1	1	1	0	0	1	1	.....	1	1	0	1	1	0
仓库 3	0	1	0	1	1	0	1	1	0	1	.....	1	1	1	1	1	1

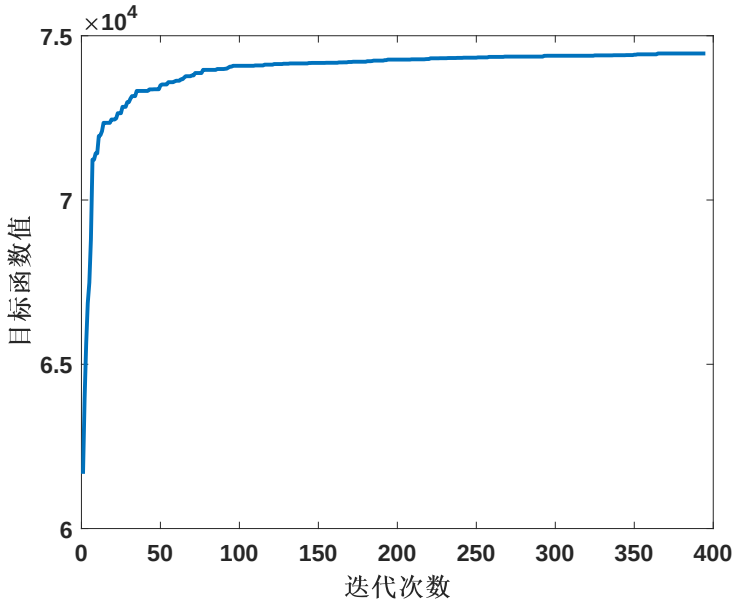


图 4　遗传算法迭代过程

五、 问题二模型的建立及求解

图5为基于拆分与转运的订单履行流程，本问题主要讨论订单拆分、订单转运与包裹派送。首先给出 SKU 转运成本与包裹运输成本的表达式，再给出了 7 种备选的订单履行方案，并分别计算出每个方案的费用支出，最后建立基于成本的订单履行方案决策模型，给出每个订单的最优履行方案。此外，还对决策结果进行分析，说明模型的准确性。

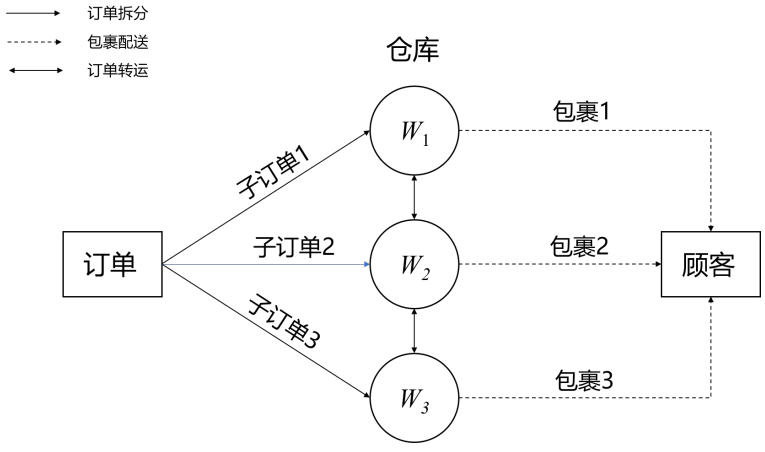


图 5　基于拆分与转运的订单分配流程

5.1 订单成本

零售商履行订单的总成本由两部分组成，一部分是 SKU 的转运成本，由仓库间转运 SKU 的个数决定；另一部分是包裹运输成本，由运输路程决定。

5.1.1 SKU 转运成本

仓库之间转运 SKU 需要花费一定资金，仓库 1 与仓库 2 之间每个 SKU 的转运费用为 t_{12}，仓库 1 与仓库 3 之间为 t_{13}，仓库 2 和仓库 3 之间为 t_{23}，如下图所示。

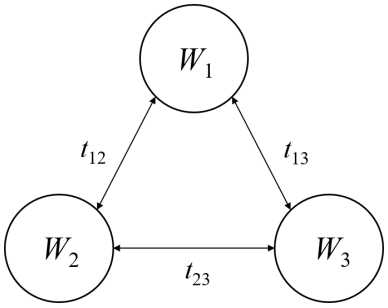


图 6　仓库之间的转运费用

根据转运费用，当仓库 i 缺货时，其优先补货仓库 W_i^p 应选择转运成本较小的，即

$$W_i^p = \begin{cases} W_j & ,t_{ij} \le t_{ik} \\ W_k & ,t_{ij} > t_{ik} \end{cases}$$

当仓库 i 和仓库 j 缺货时，仓库 k 优先补货的对象 W_k^o 同样选择转运成本较小的，即

$$W_k^o = \begin{cases} W_i & ,t_{ik} \le t_{jk} \\ W_j & ,t_{ik} > t_{jk} \end{cases}$$

5.1.2 包裹运输成本

包裹运输的计价规则可表示为以距离为自变量，价格为因变量的分段函数形式。客户与仓库间的距离在 a 公里及以内的包裹运输费用为 p_0，距离超过 a 公里后每一公里运输费用增加 p_1(不满一公里时按一公里计算)，具体的价格分段函数为:

$$P(x) = \begin{cases} p_0 & ,x \le a \\ p_0 + p_1(\lceil x \rceil - a) & ,x > a \end{cases}$$

5.2 订单履行方案成本确定

考虑 7 种订单履行方案作为备选，分别为由仓库 1 派送、由仓库 2 派送、由仓库 3 派送、由仓库 1 和仓库 2 派送、由仓库 1 和仓库 3 派送、由仓库 2 和仓库 3 派送与由三个仓库共同派送。通过讨论每种备选方案的总成本，比较之后选择成本最低的最优方案作为实际订单履行方案。

5.2.1 单仓库派送

单仓库派送包含由仓库 1 派送、由仓库 2 派送与由仓库 3 派送共三种情况，其履行思路为:

1. 提交订单，选择一个仓库派送。
2. 判断该仓库是否能提供订单要求的所有 SKU 种类，若能提供，则直接由该仓库派送包裹给顾客；若不能提供，则添加该仓库的优先补货仓库。
3. 判断该仓库与优先补货仓库是否能提供订单要求的所有 SKU 种类，若能提供，则需要优先补货仓库转运；若不能提供，则需要其余两个仓库转运。
4. 计算不同履行方式的费用。

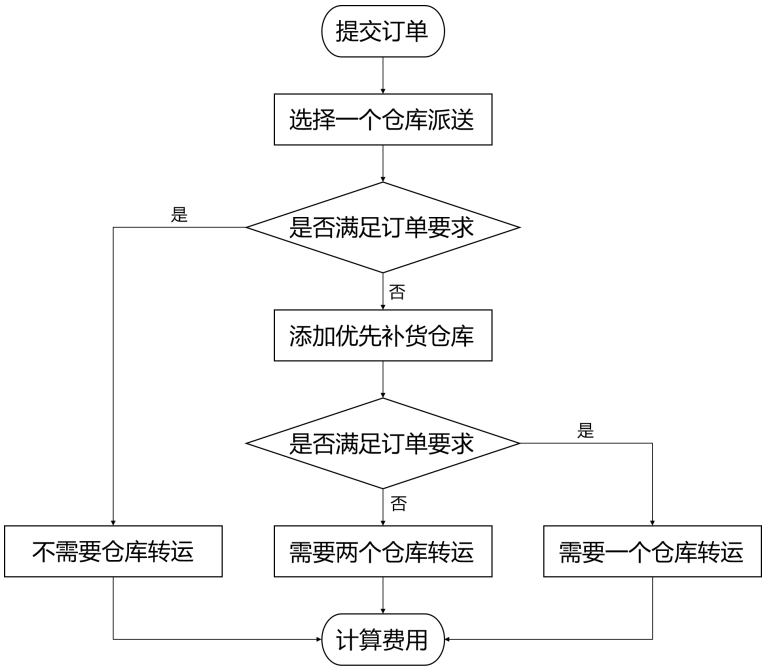


图 7　单仓库派送的履行思路流程图

其中不需要仓库转运时，所需费用仅为包裹运输成本；需要一个仓库转运时，所需费用为包裹运输成本和一个仓库的 SKU 转运成本；需要两个仓库转运时，所需费用为

包裹运输成本和两个仓库的 SKU 转运成本。选择仓库 $i(i \in \{1,2,3\})$ 派送时订单成本表示为

$$C_1(x_i) = \begin{cases} P(x_i) & ,\text{不需要仓库转运} \\ P(x_i) + (n - n_i)t_i^p & ,\text{需要一个仓库转运} \\ P(x_i) + n_j t_{ij} + n_k t_{ik} & ,\text{需要两个仓库转运} \end{cases}$$

x_i 表示仓库 i 与顾客之间的距离，n 表示订单包含的 SKU 品类个数，n_i 表示仓库 i 提供的 SKU 品类个数，n_j 表示仓库 j 提供的 SKU 品类个数,n_k 表示仓库 k 提供的 SKU 品类个数,t_i^p 表示仓库 i 与其优先补货仓库之间的转运成本。

5.2.2 双仓库派送

双仓库派送包含由仓库 1 和仓库 2 派送，由仓库 1 和仓库 3 派送与由仓库 2 和仓库 3 派送共三种情况，其履行思路为:

1. 提交订单，选择两个仓库派送。
2. 判断这两个仓库是否能提供订单要求的所有 SKU 种类，若能提供，则直接由这两个仓库派送包裹给顾客；若不能提供，则由剩下的仓库选择优先补货对象转运。
3. 计算不同履行方式的费用

其中由两个仓库派送时，所需费用仅为两个包裹的运输成本；需要剩下的仓库转运时，所需费用为两个包裹的运输成本和一个仓库的 SKU 转运成本。选择仓库 i 和仓库 $j(i,j \in \{1,2,3\})$ 派送时订单成本表示为

$$C_2(x_i, x_j) = \begin{cases} P(x_i) + P(x_j) & ,\text{不需要仓库转运} \\ P(x_i) + P(x_j) + (n - n_i - n_j)t_k^o & ,\text{需要一个仓库转运} \end{cases}$$

n 表示订单包含的 SKU 品类个数，n_i 表示仓库 i 提供的 SKU 品类个数，n_j 表示仓库 j 提供的 SKU 品类个数,t_k^o 表示仓库 k 与其优先补货对象之间的转运成本。

5.2.3 三仓库派送

三仓库派送仅包含由三个仓库共同派送共一种情况，其所需费用仅为三个包裹的运输成本，派送时订单成本表示为

$$C_3(x_1, x_2, x_3) = P(x_1) + P(x_2) + P(x_3)$$

5.3 基于成本的订单履行方案决策模型

本问题主要研究在允许转运的情况下履行订单的总成本最低的方案，故分别求出 7 种备选方案的成本，通过比较得出成本最小的方案作为在线零售商应采取的订单分配方案。因为订单与订单之间没有较大关联，因此将所有订单分解为单个订单进行求解，分别得出每个订单的最佳履行方案，最后再进行汇总。

5.3.1 决策变量

对于决策问题，采用 7 个 0-1 变量 $s_1, s_2, \ldots, s_7$ 表征在线零售商的决策方案，若变量取 1，表示在线零售商采取该方案；变量取 0，则代表在线零售商不采取该方案。

5.3.2 目标函数

已知 7 个方案的成本，故可将某种决策下的履行订单成本表示为:

$$C = s_1C_1(x_1)+s_2C_1(x_2)+s_3C_1(x_3)+s_4C_2(x_1,x_2)+s_5C_2(x_1,x_3)+s_6C_2(x_2,x_3)+s_7C_3(x_1,x_2,x_3)$$

其中 s 为决策变量，$C(x)$ 代表方案的成本，目标函数对成本 C 求取最小值即可。

5.3.3 约束条件

约束条件主要保证零售商只采取 7 个履行方案中的一种，表示为:

$$\sum_{i=1}^{7} s_i = 1$$

5.3.4 决策模型的建立

综上可给出决策模型:

$$\min \quad C$$

$$\text{s.t.} \quad \begin{cases} \sum\limits_{i=1}^{7} s_i = 1 \\ s_i \in \{0,1\} \quad i = 1,2,\ldots,7 \end{cases}$$

5.4 基于成本的订单履行方案决策模型的求解与结果分析

代入问题二的题目条件，确定决策模型中参数的取值，从而得出题中在线零售商采取的最优订单履行方案，并对结果的合理性进行分析。

5.4.1 求解结果

根据题目条件，部分模型参数具体见下表。

表 2　问题二的模型参数

符号	含义	取值
t_{12}	仓库 1 与仓库 2 之间的转运费用	1.95 元
t_{13}	仓库 1 与仓库 3 之间的转运费用	1.4 元
t_{23}	仓库 2 与仓库 3 之间的转运费用	2.1 元
p_0	基础包裹运输费用	3 元
a	基础包裹运输距离	5 公里
p_1	额外运输费用增长系数	0.2 元/公里

本文以订单 O00001 为例，介绍基于成本的订单履行方案决策模型的具体实现过程，其订单信息如下:

表 3　O00001 订单信息

订单编号	SKU 品类	顾客位置坐标
O00001	[77,63,94,1,36,10,70,86,14,67]	[71.0270441124638,33.8644796735834]

由订单信息可知，该顾客位置与仓库 1 的距离 x_1=86.385 公里，与仓库 2 的距离 x_2=52.394 公里，与仓库 3 的距离 x_3=57.342 公里；对应包裹运输成本为 $P(x_1)$ =19.4 元，$P(x_2)$=12.6 元，$P(x_3)$=13.6 元。

故 7 种方案的订单总成本如下表所示

方案	1	2	3	4	5	6	7
总成本	25.55 元	16.65 元	18.5 元	33.4 元	34.95 元	27.6 元	45.6 元

因此最优订单履行方案为总成本最低的方案 2，即从仓库 1 转运编号为 S036 的 SKU 到仓库 2，从仓库 3 转运编号为 S063 的 SKU 到仓库 2，最后由仓库 2 派送一个包裹到顾客。

对 50000 个订单按照上述方式采用基于成本的订单履行方案决策模型，得出每个订单的最优方案，下图给出采取每种方案的订单数。由图可知，所有订单均只需要由一个仓库合并打包派送，即履行订单时产生的包裹总数为 50000 个，最终得出最低订单履行总成本为 581926.9 元。

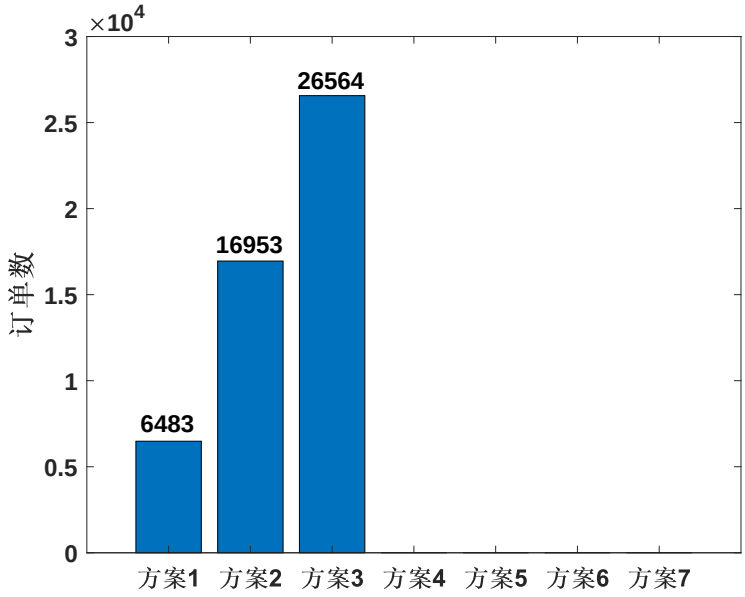


图 8　采取每种方案的订单数

5.4.2 结果合理性分析

为说明结果的合理性，进而体现模型的准确性，本文将分析决策结果产生的原因。

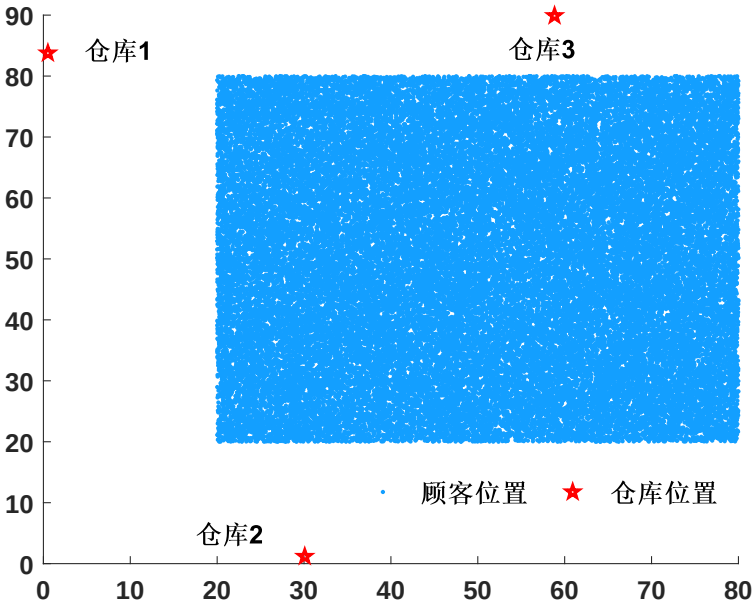


图 9　客户与仓库的位置分布

首先根据订单中的客户位置坐标数据与仓库位置坐标数据，画出客户与仓库在坐标系中的分布情况如9。顾客的区域主要集中在以 (20.082,20.305)、(20.102,70.903)、(79.850,79.961)、(79.618,20.059) 四个点围成的矩形范围内，而仓库 1 与仓库 2 靠近坐标轴，仓库 3 在矩心区域上方。

仓库与客户的最短距离为仓库 3 与 O16608 订单客户的距离 x_3=9.9072 公里，对应包裹的运输成本 $P(x_3)$ =4 元，大于仓库之间的转运费用。这说明仓库与顾客之间的距离较长，这将带来高昂的包裹运输成本，因此对于一个 SKU 来说，采取的最优订单履行方案应该以 SKU 转运为主，即选择方案 1、方案 2 或方案 3。

而基于成本的订单履行方案决策模型给出的最优方案均为前三个方案之一，这说明该决策模型很好地反映了实际情况，并针对已知条件给出了十分合理的结果，进一步证明了模型的准确性。

六、 问题三模型的建立及求解

首先确定不考虑转运时的订单履行成本，给出方案成本的表达式，再在基于成本的订单履行方案决策模型的基础上，向目标函数中添加与客户满意度有关的惩罚成本，从而构建基于成本与满意度的方案决策模型。利用该模型对问题二重新进行分析，并比较转运与不考虑转运两种情况下履行订单总成本与满意度。再利用均匀分布生成全新的订单数据，并利用决策模型对其进行分析，进一步说明模型的普适性。

6.1 不考虑转运的订单履行方案成本确定

在不考虑转运的情况下，同样有 7 种备选的订单履行方案，各个方案的成本确定思路为:

1. 提交订单，设置初始成本矩阵 $A = [L, L, L, L, L, L, L]$，其中 L 为很大的数，要求显著大于单个方案的履行订单成本。

2. 判断仓库 W_i 能否提供订单要求的所有 SKU 种类，若能提供，则令成本矩阵中第 i 项为 $C_1(x_i)$。

3. 再判断仓库 W_i 与仓库 W_j 能否提供订单要求的所有 SKU 种类，若能提供，则令成本集合中第 $(i + j + 1)$ 项为 $C_2(x_i, x_j)$；若不能提供，则令成本矩阵最后一项为 $C_3(x_1, x_2, x_3)$。

4. 输出成本矩阵 A。

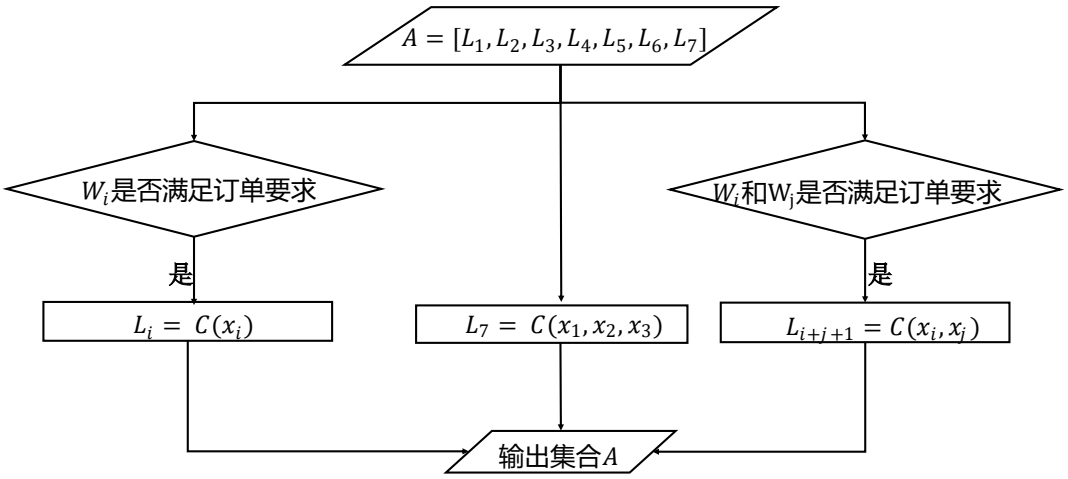


图 10　不考虑转运时的方案成本确定

以分段函数形式表示各个方案的成本:

- 单仓库派送

$$C_1(x_i) = \begin{cases} P(x_i) & ,W_i \text{ 能够满足要求} \\ L & ,W_i \text{ 不能满足要求} \end{cases}$$

- 双仓库派送

$$C_2(x_i, x_j) = \begin{cases} P(x_i) + P(x_j) & ,W_i \text{ 与 } W_j \text{ 能够满足要求} \\ L & ,W_i \text{ 与 } W_j \text{ 不能满足要求} \end{cases}$$

- 三仓库派送

$$C_3(x_1, x_2, x_3) = P(x_1) + P(x_2) + P(x_3)$$

6.2 基于成本与满意度的订单履行方案决策模型

本问题主要研究在同时考虑顾客满意度和履行订单总成本的情况下，如何履行订单。因此在问题二决策模型的基础上，在目标函数中派送包裹的总数量，从而构建多目标优化模型。

6.2.1 决策变量

采用 7 个 0-1 变量 $s_1, s_2, \ldots, s_7$ 表征在线零售商对每一个订单的决策方案，若变量取 1，表示在线零售商采取该方案；否则表示在线零售商不采取该方案。

6.2.2 目标函数

履行订单的费用为可变成本，由某个决策对应的支出表示，即

$$C_v = s_1C_1(x_1)+s_2C_1(x_2)+s_3C_1(x_3)+s_4C_2(x_1,x_2)+s_5C_2(x_1,x_3)+s_6C_2(x_2,x_3)+s_7C_3(x_1,x_2,x_3)$$

其中 s 为决策变量，$C(x)$ 代表方案的成本。

客户对服务的满意度为惩罚成本，由派送给客户的包裹总数量表示，即

$$C_p = [s_1 + s_2 + s_3] + 2[s_4 + s_5 + s_6] + 3s_7$$

目标函数由可变成本与惩罚成本组成，表示为:

$$C = \alpha_v C_v + \alpha_p C_p$$

其中 α_v 为可变成本系数，α_p 为惩罚成本系数，成本叠加前已完成归一化。对于在线零售商，履行订单总成本相对于客户满意度更加重要，因为履行订单总成本直接影响收益，而客户满意度间接影响收益，且影响程度取决于客户，对包裹数量的要求因人而异。因此令 α_v=0.7，α_p=0.3。

6.2.3 约束条件

约束条件主要保证零售商只采取 7 个履行方案中的一种，表示为:

$$\sum_{i=1}^{7} s_i = 1$$

6.2.4 决策模型的建立

综上可给出决策模型:

$$\min \quad C = \alpha_v C_v + \alpha_p C_p$$

$$\text{s.t.} \quad \begin{cases} \sum\limits_{i=1}^{7} s_i = 1 \\ s_i \in \{0,1\} \quad i = 1,2,\ldots,7 \end{cases}$$

6.3 基于成本与满意度的决策模型对问题二的求解与分析

利用基于成本与满意度的订单履行方案决策模型，分别在考虑转运与不考虑转运的条件下进行求解，并对所得结果进行分析，最后比较两种条件下的订单履行总成本与客户满意度，给出优选方案。

6.3.1 考虑转运时的结果与分析

代入问题二的题目条件，确定决策模型中参数的取值，令极大数 L=100。然后对每一个订单利用基于成本与满意度的订单履行方案决策模型进行分析，得出最优决策方案。所得最优策略方案的结果与基于成本的订单履行方案决策模型得出结果相同。

在问题二中，SKU 转运成本明显小于其运输成本，因此订单转运后在某个仓库内合并打包并以一个包裹派送给客户的方案为优势方案，该方案对应的派送包裹数量也最少，进而客户对零售商的满意度最高。因此基于成本与满意度的决策模型和基于成本的决策模型给出的最优方案一致，这也进一步说明了模型的合理性。

6.3.2 不考虑转运时的结果与分析

在问题二条件的基础上，不允许订单转运，再对每一个订单利用基于成本与满意度的订单履行方案决策模型进行分析，得出最优决策方案。最终履行订单时产生的包裹总数为 79718 个，订单履行总成本为 968389.6 元，所得最优方案的分布如图所示:

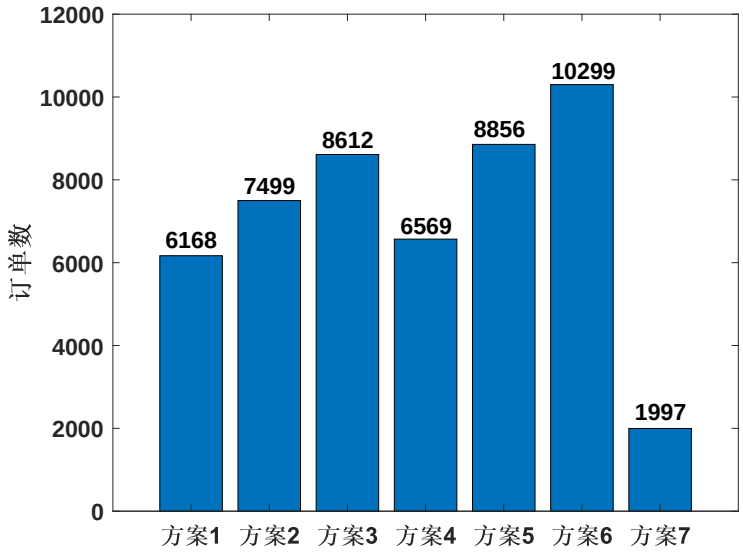


图 11　不考虑转运时的最优方案分布

由图可知，当不允许转运时，共有 27721 个订单的最优履行方案从方案 1、方案 2 与方案 3 变为多仓库方案，这是因为单个仓库无法满足这些订单的 SKU 品类需求，而订单转运无法进行，因此需要两个甚至三个仓库完成派送任务。

6.3.3 两种情况下的对比与分析

与考虑转运的情况相比，不考虑转运时选择多仓库履行方案的订单显著增多，故所需派送的包裹数量也相应增多，最终派送的总包裹数增加了 29718 个，这将大大降低用户的满意度。此外，虽然仓库间的转运费用降至零，但由之前对转运成本与包裹运输成

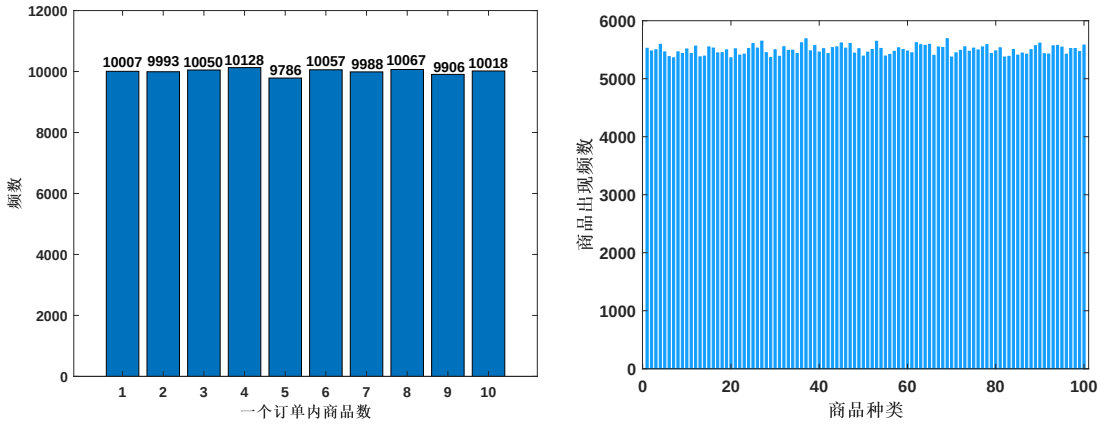
本的比较可得，增加的包裹运输费用显著高于转运费用的减少量，最终订单履行总成本增加了 386462.7 元。这些结果说明了订单转运能够显著降低成本与增加用户满意度，因此在条件允许的情况下应尽量采取订单转运的方式派送包裹。

6.4 基于成本与满意度的决策模型对模拟数据的求解与分析

为说明模型的普适性与完整性，利用均匀分布随机生成全新的 100000 条订单数据，然后利用决策模型在不同条件下求解，并对结果进行对比与分析。

6.4.1 模拟数据的生成

根据附件一的订单数据特点，采取均匀分布生成全新的 100000 条订单数据，其中每个订单包含的 SKU 个数在 [1,10] 内均匀分布，每个订单包含的 SKU 品类数在 [1,100] 内均匀分布，具体频数统计为:



(a) 订单内 SKU 个数的频数统计

(b) 订单内 SKU 品类数的频数统计

图 12　订单模拟数据频数统计

由图可知，不同 SKU 个数的订单数量相互接近，且不同 SKU 品类数的订单数量也近似相等。订单数据的 SKU 之间基本没有关联，与附件一类似，因此该模拟数据是有效的。

客户位置在 [-10,100]×[-10,100] 的区域内以均匀分布的方式随机生成，如图13所示。此时订单数据特点与问题二中不同，仓库分布在客户矩形区域内，这与现实情况更加接近，部分包裹的运输成本明显小于仓库转运成本甚至为 0。故仓库转运不再占据主导地位，在这种条件下利用决策模型讨论每个订单的最优履行方案。

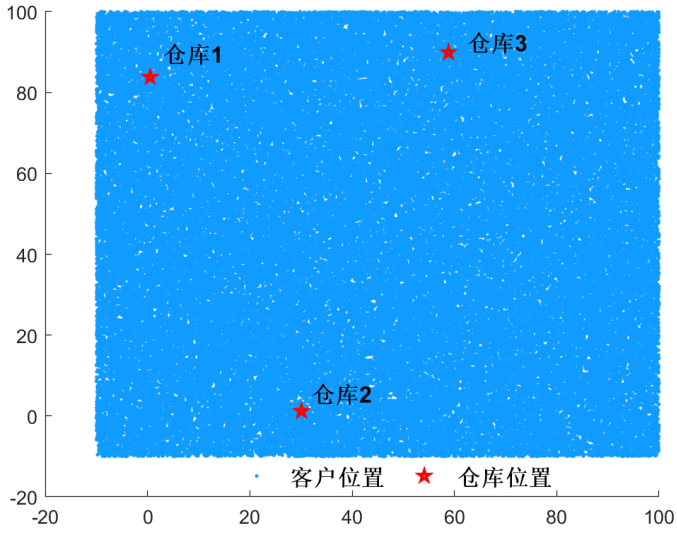


图 13　客户与仓库的位置分布

6.4.2 两种情况下的对比与分析

将 100000 条订单数据输入到基于成本与满意度的订单履行方案决策模型中，分别在考虑转运与不考虑转运的情况下给出每个订单的最优履行方案，最终统计采取各个方案的订单总数如下图所示。

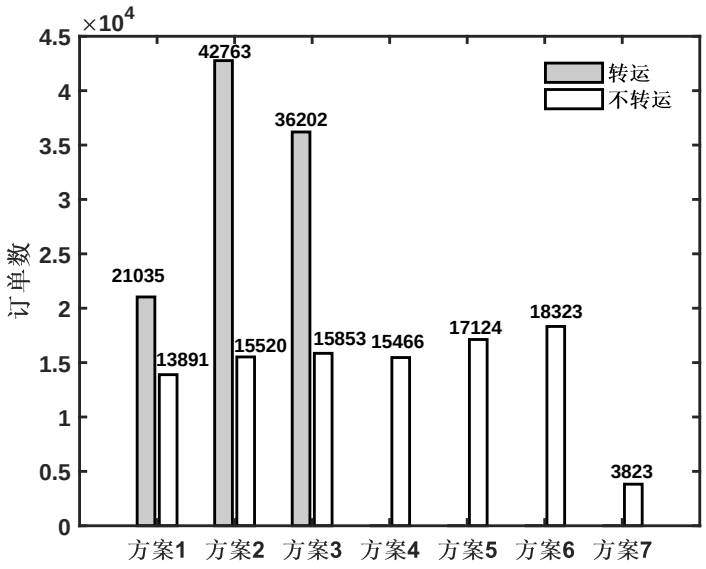


图 14　采取各方案的订单个数

根据统计结果，当考虑转运时，最优履行方案依然是前三个方案，由一个仓库派送包裹履行订单，最终共派送 100000 个包裹，订单履行总成本为 1119662.6 元，客户满意度达到最大值。这是因为三个仓库两两之间的距离较大，分别为 87.727 公里、58.639 公里与 93.263 公里，对应的运输成本远大于仓库转运费用。

对于靠近某个仓库的客户，若仓库能够满足订单需求则由该仓库派送货物；若无法满足，由其他仓库派送包裹的费用明显多于仓库间的转运费用，故采取订单转运后由该仓库派送货物；对于不靠近仓库的用户，如三个仓库围成三角形的外心，该用户离三个仓库均有一定距离，因此转运后派送一个包裹比派送多个包裹的成本更低，故也选择前三个方案。综上，考虑转运时每个订单的最优履行方案依然是方案 1、方案 2 或方案 3。

当不考虑转运时，选择多仓库履行方案的订单数同样显著增多，进而使得派送的包裹数增加 58559 个，降低了用户的满意度。此外，派送包裹数的增多也带来了包裹运输成本的显著增加，使得履行订单总成本增加至 2094082.6 元。综上，相对于考虑转运的情况，不转运时履行订单的总成本明显增加，而顾客满意度明显降低，这再次证明了订单转运的优越性与决策模型的准确性。

七、 模型的评价

7.1 模型的优点

1. 针对问题一采用遗传算法，与 0-1 规划问题的决策变量相匹配，能够得出复杂优化问题的全局最优解，具有良好的可操作性。

3. 在求解问题二与问题三时，将订单数据分解，针对每一个订单求出最优解，进而得到所有订单的最优解，最终结果较为准确。

2. 本文基于附件一和附件二中的数据分布，随机生成全新的十万个订单数据，并利用决策模型对其进行分析，进一步说明了模型的普适性。

7.2 模型的缺点

1. 由于本文的前提假设较为理想，如每个订单中所含的不同 SKU 数量均为一个、不考虑 SKU 的重量和体积、忽略车辆容量等，因此本文的讨论是在一定的前提条件下进行的，现实意义较小。

2. 问题一中非线性 0-1 规划问题的决策变量较多，情况复杂，使用遗传算法迭代求解的收敛速度比较慢。

八、 模型的改进

在实际的物流系统中，仓库间的 SKU 存储量并不完全充足，可能存在缺货的情况，同时还有物流中心等设施的存在用于打包并派送包裹，这都说明本文提出的模型条件较为理想化，因此需要对模型进行改进。在本文构建的非线性 0-1 规划模型与决策模型基础上，针对实际情况可添加 SKU 的重量和体积、车辆容量、运输路径与时间等约束条件进一步完善模型，使结果更具有现实指导意义。

附录 A　第一问求解程序

订单、变量间的相关性分析

```
clc
clear
load('附件一.mat')
M=50000;

relation    = zeros(50000,100);
for  i=1:M
    for  j=1:10
        if  SKU(i,j)~=0
            relation ( i , SKU(i,j) )=1;
        end
    end
end
correlation  = corr( relation );
sum(relation)

num_SKU = zeros(100,1);

for  i=1:M
    for  j  =1:10
        if  SKU(i,j)~=0
            num_SKU(SKU(i,j)) = num_SKU(SKU(i,j))+1;
        end
    end
end

bar(num_SKU)

figure(1)
heatmap(correlation,'GridVisible','off')

num = sum(relation,2);
```

```
ind_m_SKU = find(num==10);
m_correlation = corr( relation (ind_m_SKU,:));
heatmap(m_correlation)

figure(2)
histogram(num)
num_num = [4918,5025,5034,4927,4921,5186,5045,4945,5095,4904];
bar(num_num)
```

遗传算法主程序

```
tic
clear
clc

%% 创建数据
load('SKU.mat')
global SKU

n=300;        % 三百个 0-1 变量
%% 参数设置
NIND=500;                                   %种群大小
MAXGEN=500;                                 %迭代次数
Pc=0.9;                                     %交叉概率
Pm=0.08;                                     %变异概率
GGAP=0.9;                                   %代沟

%% 初始化种群
Chrom=InitPop(NIND,n);
%% 优化
gen=1;
bestIndividual=Chrom(1,:);
    %初始将初始种群中一个个体赋值给全局最优个体
bestObj=Obj_Fun(bestIndividual);  %计算初始bestIndividual的物品总价值
BestObj=zeros(MAXGEN,1);
    %记录每次迭代过程中的最优适应度值
```

```
while gen<=MAXGEN
    %% 计算适应度
    Obj=Obj_Fun(Chrom);
        %计算每个染色体的包裹总数量的线性函数 150000-包裹总数
    FitnV=Obj;                                  %适应度值=目标函数值
    %% 选择
    SelCh=Select(Chrom,FitnV,GGAP);
    %% 交叉操作
    SelCh=Crossover(SelCh,Pc);
    %% 变异
    SelCh=Mutate(SelCh,Pm);
    %% 重插入子代的新种群
    Chrom=Reins(Chrom,SelCh,Obj);
    %% 将种群中不满足约束的个体进行约束处理
    for  i=1:NIND
        Individual=Chrom(i,:);
        flag=judge_individual(Individual);

        while flag~=1
            Individual=repair_individual(Individual,flag);
                %修复个体Individual
            flag=judge_individual(Individual);
        end
        Chrom(i,:) = Individual;
    end

    %% 记录每次迭代过程中最优目标函数值
    [cur_bestObj,cur_bestIndex]=max(Obj);
        %在当前迭代中最优目标函数值以及对应个体的编号
    cur_bestIndividual=Chrom(cur_bestIndex,:); %当前迭代中最优个体
    %如果当前迭代中最优目标函数值大于等于全局最优目标函数值，则进行更新
    if  cur_bestObj>=bestObj
        bestObj=cur_bestObj;
        bestIndividual=cur_bestIndividual;
    end
```

```
    BestObj(gen,1)=bestObj;
        %记录每次迭代过程中最优目标函数值
    %% 打印每次迭代过程中的全局最优解
    disp(['第',num2str(gen),'次迭代的全局最优解为：',num2str(bestObj)]);

    %% 当最优解稳定超过30代则认为达到最优
    if gen>50
        if BestObj(gen)==BestObj(gen-30)
            disp('最优解稳定超过30代，退出迭代');
            break;
        end
    end

    gen=gen+1 ; % 更新迭代次数
end

%% 画出迭代过程图
figure;
plot(BestObj,'o-','LineWidth',1);
xlabel('迭代次数');
ylabel('目标函数值');
% %% 最终装进包中的物品序号
% pack_item=find( bestIndividual ==1);
% %% 计算最优装包方案的物品总价值和总重量
% [bestP,bestW]=Individual_P_W(n,bestIndividual ,p,w);
toc

xx = zeros(100,3);
xx(:,1) = bestIndividual(1:100)';
xx(:,2) = bestIndividual(101:200)';
xx(:,3) = bestIndividual(201:300)';
```

初始化种群函数

```
%% 初始化种群
%输入NIND:                种群大小
```

```
%输出Chrom:                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,:)=encode(N);       %编码，生成满足约束的个体
    end
end
```

编码函数

```
%% 编码，生成满足约束的个体
%输入n:
%输出Individual:
function Individual=encode(n)
    Individual=round(rand(1,n));
        %随机生成n个数字（每个数字是0或1）
    flag=judge_individual(Individual);
        %判断Individual是否满足约束，1表示满足，0表示不满足
    %% 如果flag为0，则需要修复个体Individual。否则，不需要修复
    while flag~=1
        Individual=repair_individual(Individual,flag);          %修复个体Individual
        flag=judge_individual(Individual);
    end
end
```

判断个体是否满足约束函数

```
%% 判断一个个体是否满足约束
% 1表示满足
% -1表示仓库1库存大于 70种
% -2表示仓库2库存大于 70种
% -3表示仓库3库存大于 70种
% -4表示仓库所有库存不满足总共100种
% 输入Individual:              个体
%输出flag:
function flag=judge_individual(Individual)
    flag =1;
```

```
    if sum(Individual(1:100))>70
        flag=-1;
    elseif sum(Individual(101:200))>70
        flag=-2;
    elseif sum(Individual(201:300))>70
        flag=-3;
    elseif sum((Individual(1:100)+Individual(101:200)+Individual(201:300))>0)<100
        flag=-4;
    end
end
```

修复违反约束的个体函数

```
%% 对违反约束的个体进行修复
% 对多的元素随机剔除
% 缺少的随机多添到任意仓库去

function Individual=repair_individual(Individual,flag)
    switch flag
        case -1
            ind = find(Individual(1:100)==1);
            if size(ind,2)>70
                r = randi([1,size(ind,2)],1,size(ind,2)-70);
                Individual(ind(r))=0;
            end

        case -2
            ind = find(Individual(101:200)==1);
            if size(ind,2)>70
                r = randi([1,size(ind,2)],1,size(ind,2)-70);
                Individual(100+ind(r))=0;
            end

        case -3
            ind = find(Individual(201:300)==1);
            if size(ind,2)>70
```

```
                r = randi([1, size(ind,2)],1, size(ind,2)-70);
                Individual(200+ind(r))=0;
            end

        case -4
            ind =
                find(((Individual(1:100)+Individual(101:200)+Individual(201:300))>0)==0);
            for i=1:size(ind,2)
                r = randi([0,2],1,1);
                Individual(ind(i)+r*100)=1;
            end
    end

end
```

计算目标函数

```
%% 计算种群中每个染色体的物品总价值
%输入Chrom:                种群
%输出Obj:                  种群中每个个体的物品总价值
function Obj=Obj_Fun(Chrom)
global SKU
    Obj = zeros(size(Chrom,1),1);
    for i=1:size(Chrom,1)
        xx = Chrom(i,:);
        num = 0;
        for j=1:size(SKU,1)
            len =sum(SKU(j,:)>0);
            goods = zeros(len,3);
            for a = 1:len
                goods(a,1) = xx(SKU(j,a));
                goods(a,2) = xx(100+SKU(j,a));
                goods(a,3) = xx(200+SKU(j,a));
            end

            if  max(sum(goods,1))==len
```

```
                num = num+1;
            elseif
                (sum((goods(:,1)+goods(:,2))>0)==len)||(sum((goods(:,1)+goods(:,3))>0)==len)||(s
                num = num+2;
            else
                num = num+3;
            end
        end
        Obj(i) = 150000-num;
    end
end
```

选择子代

```
%% 选择操作
%输入Chrom:          种群
%输入FitnV:          适应度值
%输入GGAP:           代沟
%输出SelCh:          被选择的个体
function SelCh=Select(Chrom,FitnV,GGAP)
    NIND=size(Chrom,1);                %种群数目
    Nsel=NIND*GGAP;
    total_FitnV=sum(FitnV);            %所有个体的适应度之和
    select_p=FitnV./total_FitnV;       %计算每个个体被选中的概率
    select_index=zeros(Nsel,1);        %储存被选中的个体序号
    %对select_p进行累加操作，c(i)=sum(select_p(1:i))
    %如果select_p =[0.1,0.2,0.3,0.4] ，则c =[0.1,0.3,0.6,1]
    c=cumsum(select_p);
    %% 循环NIND次，选出NIND个个体
    for i=1:Nsel
        r=rand;                        %0~1之间的随机数
        index=find(r<=c,1,'first');    %每次被选择出的个体序号
        select_index(i,1)=index;
    end
    SelCh=Chrom(select_index,:);       %被选中的个体
end
```

交叉重组

```
%% 交叉操作
%输入SelCh:              被选择的个体
%输入Pc:                 交叉概率
%输出SelCh:              交叉后的个体
function SelCh=Crossover(SelCh,Pc)
    [NSel,n]=size(SelCh);                              %n为染色体长度
    for  i=1:2:NSel-mod(NSel,2)
        if  Pc>=rand %交叉概率Pc
            cross_pos=unidrnd(n);
                %随机生成一个1~N之间的交叉位置
            cross_Selch1=SelCh(i,:);                   %第i个进行交叉操作的个体
            cross_Selch2=SelCh(i+1,:);
                %第i+1个进行交叉操作的个体
            cross_part1=cross_Selch1(1:cross_pos);
                %第i个进行交叉操作个体的交叉片段
            cross_part2=cross_Selch2(1:cross_pos);
                %第i+1个进行交叉操作个体的交叉片段
            cross_Selch1(1:cross_pos)=cross_part2;
                %用第i+1个个体的交叉片段替换掉第i个个体交叉片段
            cross_Selch2(1:cross_pos)=cross_part1;
                %用第i个个体的交叉片段替换掉第i+1个个体交叉片段
            SelCh(i,:)=cross_Selch1;                   %更新第i个个体
            SelCh(i+1,:)=cross_Selch2;                 %更新第i+1个个体
        end
    end
end
```

基因突变函数

```
%% 变异操作
%输入SelCh:              被选择的个体
%输入Pm:                 变异概率
%输出SelCh:              变异后的个体
function SelCh=Mutate(SelCh,Pm)
    [NSel,n]=size(SelCh);                              %n为染色体长度
```

```
    for i=1:NSel
        if Pm>=rand
            R=randperm(n);                          %随机生成1~n的随机排列
            pos1=R(1);                              %第1个变异位置
            pos2=R(2);                              %第2个变异位置

            left =min([pos1,pos2]);
                %更小的那个值作为变异起点
            right =max([pos1,pos2]);
                %更大的那个值作为变异终点

            mutate_Selch=SelCh(i,:);                %第i个进行变异操作的个体
            mutate_part=mutate_Selch(right:-1:left);
                %进行变异操作后的变异片段
            mutate_Selch(left: right )=mutate_part;
                %将mutate_Selch上的第left至right位上的片段进行替换

            SelCh(i ,:) =mutate_Selch;
                %更新第i个进行变异操作的个体
        end
    end
end
```

构成新种群函数

```
%% 重插入子代的新种群
%输入Chrom:              父代种群
%输入SelCh:              子代种群
%输入Obj:                父代适应度
%输出Chrom:              重组后得到的新种群
function Chrom=Reins(Chrom,SelCh,Obj)
    NIND=size(Chrom,1);
    NSel=size(SelCh,1);
    [~,index]=sort(Obj,'descend');
    Chrom=[Chrom(index(1:NIND-NSel),:);SelCh];
end
```

附录 B　第二问求解程序

```
1
2  P_ware=zeros(3,2);
3  P_ware(1,:) = [0.516205320509211,83.7701245337600];
4  P_ware(2,:) = [30.0817907887535,1.17493342829637];
5  P_ware(3,:) = [58.8349268443702,89.8949407626723];
6
7  DD = zeros(M,3);
8
9  DD(:,1) = sqrt((position(:,1)-P_ware(1,1)).^2+(position(:,2)-P_ware(1,2)).^2);
10 DD(:,2) = sqrt((position(:,1)-P_ware(2,1)).^2+(position(:,2)-P_ware(2,2)).^2);
11 DD(:,3) = sqrt((position(:,1)-P_ware(3,1)).^2+(position(:,2)-P_ware(3,2)).^2);
12
13 sqrt((P_ware(3,1)-P_ware(2,1)).^2+(P_ware(3,2)-P_ware(2,2)).^2);
14
15
16 scatter(position(:,1),position(:,2),'b.')
17 hold on
18 scatter(P_ware(:,1),P_ware(:,2),'rp','LineWidth',1.5)
19
20 n_W = ones(M,1);
21 for i=1:M
22     goods = zeros(num(i),3);
23     for j=1:num(i)
24         goods(j,:) = Ware(SKU(j),:);
25     end
26
27     if max(sum(goods,1))==num(i)
28         n_W(i)=1;
29     elseif
       (sum((goods(:,1)+goods(:,2))>0)==num(i))||(sum((goods(:,1)+goods(:,3))>0)==num(i))||(su
30         n_W(i)=2;
31     else
32         n_W(i)=3;
```

```
    end
end

MM = 3+0.2*(ceil(DD)-5); % 各仓库到各客户的路费
choose = zeros(M,1);
Money = zeros(M,7);

for  i=1:M
    goods = zeros(num(i),3);
    for  j=1:num(i)
        goods(j,:)  = Ware(SKU(i,j),:);
    end

    % 全从1运出
    Money(i,1)=MM(i,1)+1.4*sum((ones(num(i),1)-goods(:,1)).*goods(:,3))+1.95*(num(i)-sum(goo

    % 全从2运出
   Money(i,2)=MM(i,2)+1.95*sum((ones(num(i),1)-goods(:,2)).*goods(:,1))+2.1*(num(i)-sum(goo

    % 全从3运出
   Money(i,3)=MM(i,3)+1.4*sum((ones(num(i),1)-goods(:,3)).*goods(:,1))+2.1*(num(i)-sum(good

    % 从1,2 运出,3肯定运到1去
   Money(i,4)=MM(i,1)+MM(i,2)+1.4*sum((goods(:,1)+goods(:,2))==0);

    % 从1,3 运出,2肯定运到1去
    Money(i,5)=MM(i,1)+MM(i,3)+1.95*sum((goods(:,1)+goods(:,3))==0);

    % 从2,3 运出,1肯定运到3去
    Money(i,6)=MM(i,2)+MM(i,3)+1.4*sum((goods(:,2)+goods(:,3))==0);
    % 从1,2,3 都运
    Money(i,7)=sum(MM(i,:));

    choose(i)=find(Money(i,:)==min(Money(i,:)),1);
end
```

```
68
69  %记录能够满足的商品数量
70  need = zeros(M,3);
71
72  for  i=1:M
73  %      i=15;
74      goods = zeros(num(i),3);
75      for  j=1:num(i)
76          goods(j,:)  = Ware(SKU(i,j),:);
77      end
78      need(i,:)    = sum(goods,1);
79  end
80
81  for  i=1:M
82      choose(i)=find(Money(i,:)==min(Money(i,:)),1);
83  end
84
85  choose = find(Money==min(Money,2));
86
87  nearest = zeros(M,1);
88  for  i=1:M
89      nearest(i)=find(MM(i,:)==min(MM(i,:)),1);
90  end
91
92  max(choose)
93  lowest_money = min(Money,[],2);
94  sum(lowest_money)
95
96  case_number = [6483;16953;26564;0;0;0;0];
97  bar(case_number)
98  set(gca, 'xTick', 1:7);
99  set(gca,'XTickLabel',{'方案1','方案2','方案3','方案4','方案5','方案6','方案7'})
100 set(gca, 'FontSize', 12);
101 xlim ([0,   8])
102 ylabel('订单数');
```

附录 C　第三问求解程序

```
%% 第三问 基于第二问表格的情况下不转运时
Money_notransport=zeros(M,2);

for  i=1:M
    goods = zeros(num(i),3);
    for  j=1:num(i)
        goods(j ,:)  = Ware(SKU(i,j),:) ;
    end

    mm1 = 1000*ones(7,1);

    if  sum(goods(:,1))==num(i)
        mm1(1)=MM(i,1);
    end

    if  sum(goods(:,2))==num(i)
        mm1(2)=MM(i,2);
    end

    if  sum(goods(:,3))==num(i)
        mm1(3)=MM(i,3);
    end

    if  sum((goods(:,1)+goods(:,2))>0)==num(i)
        mm1(4)=MM(i,1)+MM(i,2);
    end

    if  sum((goods(:,1)+goods(:,3))>0)==num(i)
        mm1(5)=MM(i,1)+MM(i,3);
    end

    if  sum((goods(:,2)+goods(:,3))>0)==num(i)
        mm1(6)=MM(i,2)+MM(i,3);
```

```
    end

    mm1(7)=MM(i,1)+MM(i,2)+MM(i,3);
    Money_notransport(i,1)=min(mm1);
    Money_notransport(i,2)=find(mm1==Money_notransport(i,1),1);
end

sum(Money_notransport(:,1))
sum((Money_notransport(:,2)>0)+(Money_notransport(:,2)>3)+(Money_notransport(:,2)>6));

case_number = [6168;7499;8612;6569;8856;10299;1997];
bar(case_number)
set(gca, 'xTick', 1:7);
set(gca,'XTickLabel',{'方案1','方案2','方案3','方案4','方案5','方案6','方案7'})
set(gca, 'FontSize', 12);
xlim ([0,  8])
ylabel('订单数');
```

```
%% 生成随机订单
clc
clear
M =100000;
num = randi([1,10],M,1);
SKU = zeros(M,10);

for  i  = 1:M
    r  = randi ([1,100],1, num(i));
    SKU(i,1:num(i)) = r;
end
num_SKU = zeros(100,1);

for  i=1:M
    for  j  =1:10
        if  SKU(i,j)~=0
            num_SKU(SKU(i,j)) = num_SKU(SKU(i,j))+1;
```

```
        end
    end
end

position  = random('uniform',-10,100,M,2);
scatter(position (:,1) , position (:,2) , 'b.')
hold on
scatter(P_ware(:,1),P_ware(:,2),'rp','LineWidth',1.5)

n_W = ones(M,1);
for  i=1:M
    goods = zeros(num(i),3);
    for  j=1:num(i)
        goods(j ,:)  = Ware(SKU(j),:);
    end

    if  max(sum(goods,1))==num(i)
        n_W(i)=1;
    elseif
        (sum((goods(:,1)+goods(:,2))>0)==num(i))||(sum((goods(:,1)+goods(:,3))>0)==num(i))||(su
        n_W(i)=2;
    else
        n_W(i)=3;
    end
end

num_1 = [10007,9993,10050,10128,9786,10057,9988,10067,9906,10018]
bar(num_1)

%% 计算随机订单
P_ware=zeros(3,2);
P_ware(1,:) = [0.516205320509211,83.7701245337600];
P_ware(2,:) = [30.0817907887535,1.17493342829637];
P_ware(3,:) = [58.8349268443702,89.8949407626723];
DD = zeros(M,3);
```

```
52
53 DD(:,1) = sqrt((position (:,1) -P_ware(1,1)).^2+(position(:,2)-P_ware(1,2)).^2);
54 DD(:,2) = sqrt((position (:,1) -P_ware(2,1)).^2+(position(:,2)-P_ware(2,2)).^2);
55 DD(:,3) = sqrt((position (:,1) -P_ware(3,1)).^2+(position(:,2)-P_ware(3,2)).^2);
56
57 D_r = DD;
58 D_r(find(DD<5)) = 5;
59 MM = 3+0.2*(ceil(DD)-5); % 各仓库到各客户的路费
60
61 %% 转运的情况
62 choose = zeros(M,1);
63 Money = zeros(M,7);
64
65 for  i=1:M
66     goods = zeros(num(i),3);
67     for  j=1:num(i)
68         goods(j ,:)  = Ware(SKU(i,j),:) ;
69     end
70     % 全从1运出
71     Money(i,1)=MM(i,1)+1.4*sum((ones(num(i),1)-goods(:,1)).*goods(:,3))+1.95*(num(i)-sum(goo
72     % 全从2运出
73    Money(i,2)=MM(i,2)+1.95*sum((ones(num(i),1)-goods(:,2)).*goods(:,1))+2.1*(num(i)-sum(goo
74     % 全从3运出
75    Money(i,3)=MM(i,3)+1.4*sum((ones(num(i),1)-goods(:,3)).*goods(:,1))+2.1*(num(i)-sum(goods
76     % 从1,2 运出,3肯定运到1去
77    Money(i,4)=MM(i,1)+MM(i,2)+1.4*sum((goods(:,1)+goods(:,2))==0);
78     % 从1,3 运出,2肯定运到1去
79     Money(i,5)=MM(i,1)+MM(i,3)+1.95*sum((goods(:,1)+goods(:,3))==0);
80     % 从2,3 运出,1肯定运到3去
81     Money(i,6)=MM(i,2)+MM(i,3)+1.4*sum((goods(:,2)+goods(:,3))==0);
82     % 从1,2,3 都运
83     Money(i,7)=sum(MM(i,:));
84     choose(i)=find(Money(i,:)==min(Money(i,:)),1);
85 end
86
```

```
%记录能够满足的商品数量
need = zeros(M,3);
for  i=1:M
%      i=15;
     goods = zeros(num(i),3);
     for  j=1:num(i)
          goods(j,:)  = Ware(SKU(i,j),:);
     end
     need(i,:)    = sum(goods,1);
end

for  i=1:M
     choose(i)=find(Money(i,:)==min(Money(i,:)),1);
end

lowest_money = min(Money,[],2);
sum(lowest_money)

nearest = zeros(M,1);
for  i=1:M
     nearest(i)=find(MM(i,:)==min(MM(i,:)),1);
end

case_number1 = [21035;42763;36202;0;0;0;0];
case_number2 = [13891;15520;15853;15466;17124;18323;3823];
case_number = [case_number1,case_number2];

bar(case_number)
set(gca, 'xTick', 1:7);
set(gca,'XTickLabel',{'方案1','方案2','方案3','方案4','方案5','方案6','方案7'})
set(gca, 'FontSize', 12);
xlim ([0,   8])
ylabel('订单数');

%% 不转运的情况
```

```
Money_notransport=zeros(M,2);

for  i=1:M
     goods = zeros(num(i),3);
     for  j=1:num(i)
          goods(j,:)  = Ware(SKU(i,j),:);
     end

     mm1 = 1000*ones(7,1);

     if  sum(goods(:,1))==num(i)
          mm1(1)=MM(i,1);
     end

     if  sum(goods(:,2))==num(i)
          mm1(2)=MM(i,2);
     end

     if  sum(goods(:,3))==num(i)
          mm1(3)=MM(i,3);
     end

     if  sum((goods(:,1)+goods(:,2))>0)==num(i)
          mm1(4)=MM(i,1)+MM(i,2);
     end

     if  sum((goods(:,1)+goods(:,3))>0)==num(i)
          mm1(5)=MM(i,1)+MM(i,3);
     end

     if  sum((goods(:,2)+goods(:,3))>0)==num(i)
          mm1(6)=MM(i,2)+MM(i,3);
     end

     mm1(7)=MM(i,1)+MM(i,2)+MM(i,3);
```

```
    Money_notransport(i,1)=min(mm1);
    Money_notransport(i,2)=find(mm1==Money_notransport(i,1),1);
end

sum(Money_notransport(:,1))
sum((Money_notransport(:,2)>0)+(Money_notransport(:,2)>3)+(Money_notransport(:,2)>6));

histogram(Money_notransport(:,2))
case_number2 = [13891;15520;15853;15466;17124;18323;3823];
bar(case_number)
set(gca, 'xTick', 1:7);
set(gca,'XTickLabel',{'方案1','方案2','方案3','方案4','方案5','方案6','方案7'})
set(gca, 'FontSize', 12);
xlim ([0,  8])
ylabel('订单数');
```