**基于贪心-MCTS对出库方式及排程方案的研究**

摘要

在如今纺织业竞争生态格局中，不同种类SKU的仓储模式有着不可替代的作用。通过研究优化布料的高效出库模式及排程方案，对企业实现多年发展有着重要的战略意义。本问通过建立起0-1规划模型，运用了TOPSIS模型、蒙特卡洛树搜索、K-means++聚类算法等多种模型与算法，对仓储的方式及订单的排班进行优化，从而达到尽可能提升订单数量的同时降低企业员工的工作负荷。

**针对任务一：**本问通过建立**0-1规划模型**，研究一种合理的出库方式及排程方案来使得出库的订单尽可能多。在数据预处理阶段，借助题目中所给信息建立起整体仓库的位置图，对每个货品位置进行定位；并采用了**K-means++聚类分析**算法，按照下单的不同时间进行归类分析，以达到局部最优效果。在模型建立过程中，使用0-1规划模型，构建起相关的目标函数和约束条件，并创新性地提出了使用状态函数进行更好的规划。利用**贪心思想**,通过对不同叉车不同时间段的状态分析，更好地追踪任务的完成情况并分配下一个任务。在最后通过**蒙特卡洛树搜素(MCTS)**，确定各天完成订单数量：**1月1日**：**90/110**；**1月5日：89/105；1月10日：106/115；1月15日：100/108**。总体上完成**385**个订单。

**针对任务二：**本问通过研究不同商品的销售情况，对商品进行重新排序并规划排程方案。总体上先使用**TOPSIS模型**，对不同商品的销售情况通过总订单数量、顺季节订单数量、反季节订单数量进行逐一**评分**，从而完成对不同商品销售评价。随后，通过销售热度和货品位置一一对应，完成商品的重新排布。基于0-1规划模型，利用贪心-MCTS算法对新的货架分布采用重新安排出库方案。最后确定销售评分前三名SKU编号，**春夏：2613、3522、5077；秋冬：6135、5837、5618。**同时，确定了在重排后各天完成的订单数量：**1月1日**：**104/110**；**1月5日：102/105；1月10日：109/115；1月15日：102/108**。总体上完成**417**个订单。

关键词: **0-1规划 TOPSIS法 K-means++聚类分析 蒙特卡洛树搜索 贪心思想**

# 问题重述

## 问题背景

在全球零售行业数字化转型的进程中，连锁超市作为现代流通体系的关键节点，其运营模式正经历从经验驱动向数据驱动的深刻变革。随着零售交易数据采集技术的精细化发展，企业已积累海量包含商品属性、消费者行为轨迹、支付模式等多维信息的交易数据，如何从这些数据中提炼商业洞察并构建科学决策体系，成为提升零售企业核心竞争力的战略课题。

某国外连锁超市在三大核心城市构建商业网络，依托信息化管理系统对每笔交易进行全维度数据记录，累计形成具有深度分析价值的数据集。当前企业管理层面临双重运营挑战：一方面，采购计划制定仍依赖传统经验判断，导致库存配置与区域消费需求的动态匹配度不足，造成供应链效率损耗；另一方面，营销策略的精准性缺乏量化支撑，难以针对不同客群的消费特征实现价值转化优化。这种数据资源与决策需求的结构性脱节，使得建立基于数据挖掘的收入预测体系成为企业突破经营瓶颈的必然选择。

由此可见，构建数据驱动的库存调度与智能排程方案，已成为布料企业突破运营瓶颈、提升综合竞争力的战略关键。

## 问题重述

通过建立数学模型，结合超市的各商品的数据统计，建立起相关科学的收入预测模型，以达到最小化成本，最大化利润的目标。

问题一：基于该连锁超市2025年1月7日至4月5日的1000条交易数据，需构建系统化的数据特征解析体系。首先，运用统计学方法，精确计算各城市在销售额、商品单价、商品偏好等核心指标上的统计分布，通过交叉分析与时间序列建模，深入识别不同城市在早、中、晚消费高峰时段及付款方式选择上的显著差异；其次，采用对比分析法与聚类算法，细致剖析超市会员群体的消费特征，从消费频次、客单价、购买品类等维度量化会员与非会员群体的差异，并科学测算会员群体对超市总收入的贡献度；再者，利用Apriori等关联规则算法，挖掘商品类别、付款方式、会员属性等多维度组合下的高频购买模式；最后，通过周期图分析与季节性分解，系统性探究六类商品销售数据的周期性特征，识别周末、节假日等特殊时段的销量波动规律，为后续经营决策筑牢数据根基。

问题二：针对超市毛收入与毛利率的预测需求，需搭建完整的数据分析与建模框架。第一步，运用描述性统计、方差分析等方法，系统探究毛收入与毛利率在不同城市、消费者类别、产品类别、付款方式等维度的分布规律，借助热力图、箱线图等可视化手段直观呈现分析结果，并通过相关性检验与显著性分析，精准识别对毛收入与毛利率具有显著影响的关键变量；第二步，综合运用时间序列模型、回归分析及机器学习算法构建预测模型，对2025年4月6日至15日超市日均毛收入与毛利率进行预测。通过均方误差（MSE）、平均绝对误差（MAE）等指标对比不同模型的预测精度，结合数据特征与业务场景，合理阐释模型选择依据；同时，结合该国节假日日历，评估特殊节假日对销售数据的影响，在模型中科学引入虚拟变量或季节性调整因子，确保预测结果的准确性与可靠性。

问题三：基于上述数据分析与模型结果，从战略决策与运营优化层面，为超市管理层提供具有实操性的量化经营策略优化建议。其一，通过需求预测与边际收益分析，精准确定需优先提升库存水平或加大促销力度的商品类别、城市区域及客户群体；其二，建立客户满意度评分与经营指标的关联模型，提出针对性的服务流程优化与产品改进方案；其三，运用组合优化算法，设计提升毛利率的商品组合策略，并通过模拟分析验证策略有效性；其四，结合各城市销售数据特征与需求预测结果，制定差异化的库存优化方案；其五，构建会员扩容对收入影响的弹性模型，预测会员占比提升10%时的收入增长幅度；其六，建立多目标决策模型，综合考量收入损失、客户流失率、区域竞争态势等因素，为超市关店决策提供科学的量化评估框架，助力企业实现资源的高效配置与竞争力的全面提升。

# 问题分析

## 问题一的分析

首先，小组通过K-means++聚类分析方法以及对题目中的图片距离分析，计算出各排各列SKU相对于订单合并区的距离，从而方便综合考虑后续订单相关数据处理。其次，在模型的选择上，小组采用的是0-1规划模型的方式，通过建立任务完成总数为目标函数，依照题目所给出条件确定约束条件，最后小组创新性地提出了状态函数以补全约束条件，其中包括任务完成的前期、中期和后期的任务状态，以递推的形式形成闭环逻辑，从而使得最后目标函数最大。

## 问题二的分析

首先，小组对问题二总体上进行了分析，即大致上分成三个部分：寻找到热销较好的SKU，调整现有的SKU存储位置，将调整后的位置带入到问题一中进行求解。因此，在寻找热销较好的SKU时，小组决定使用TOPSIS模型进行分析，根据总销售量、顺季节销售量、反季节销售量三个指标，对每一个商品进行评分，根据高低系数依次排序，得到对应的热销水平。其次，利用第一问中的数据预处理数据，得到每一个货架上的格数位置，从而将商品按照距离远近及热销水平依次放入。最后，同第一问，带入到一月份进行求解。

# 模型假设

为了构建更加精准的数学模型，本文根据实际情况以及任务所给要求做出以下合理的假设或条件约束：

* 假设订单合并区域覆盖长度为从第一排到第八排；
* 假设单次登上货架只取单一SKU；
* 假设叉车运行轨迹均在所有道路的中轴线上；
* 假设单次取货仅取一种SKU；
* 假设3月到8月为春夏季度，9月到次年2月为秋冬季度；
* 假设运输过程中无损坏；
* 假设不同工人、工作效率相同不同叉车

# 符号说明

表1 本文的符号说明

|  |  |  |
| --- | --- | --- |
| **符号** | **说明** | **单位** |
|  | 第个订单第个SKU运送总时长 | 秒 |
|  | 第个订单内第个SKU | 无 |
|  | 第个和个订单下单时间的间隔 | 秒 |
|  | 号叉车在第个订单开始前的任务量 | 秒 |
|  | 号叉车第个订单进行时的任务量 | 秒 |
|  | 完成号叉车第个订单之后的任务量 | 秒 |
|  | 第个SKU在TOPSIS中的得分 | 分 |

# 问题一模型的建立与求解

## 数据预处理

在制定合理的出库方案和排程方式前，对任务内部的已知数据进行全面预处理是非常重要的。数据预处理应该包括分析数据的内部之间的联系、分析数据的组成及有效性、无量纲化处理、数量级的归一化处理以能够保证数据的质量，提高分析结果的可靠性。

首先，小组对附件1的数据进行了有效性的筛查。由于问题一中要求筛查一月份的订单进行处理规划，小组提取出一月份的订单，删去了其余月份的订单。同时，为了方便后续计算，小组对所有下单时间进行了无量纲化处理，将其转换成以秒为单位的数据。例如：题目中下单时间为10：15，在下文中转换成了36900。

同时，小组通过观察订单中SKU的下单数目，发现其中**不存在异常值和缺失值**。

其次，小组在阅读题目发现仓位示意图中缺少货架之间距离以及货架与订单区的距离、同一排货架的间距等多个数据。小组设定直角坐标系，以订单合并区域的左下角为坐标原点，横向为轴，纵向为轴。在关于轴距离分配上， 设货架宽度为米，过道宽度为米，通过春夏和秋冬款面料存放区距离建立以下方程组：

|  |  |
| --- | --- |
|  | () |

由(1)式可得，，。由此可得，货架宽度为1.8米，过道宽度为2米。

在题目所给图1中，订单合并区域存在一定的长度，小组认为这段长度不可以被忽略，并根据题目中图2信息假设该区域覆盖长度为从第一排到第八排，即假设叉车在y轴行进距离从第10排开始计算，则可以得到如下表2：

表2 各排面料相对订单合并区的间隔距离

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 1--9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
| x | 0 | 1 | 3 | 3 | 5 | 5 | 6 | 8 | 8 | 10 | 10 | 11 |
| y | 0 | 1 | 2 | 2 | 3 | 4 | 5 | 6 | 6 | 7 | 7 | 8 |
| D | 0 | 3.8 | 9.4 | 9.4 | 15 | 17 | 20.8 | 26.4 | 26.4 | 32 | 32 | 35.8 |

上表中，行代表各排面料相对订单合并区的货架排数，代表相对订单合并区的过道排数，代表各排相对订单合并区的总距离。

然后，由于不同位置具体的横坐标在题目中并没有表达很清楚，因此小组通过等比例测绘的形式，规定了各个区域之间的距离，如下图所示：



图1 各个区域间距离示意图

最后，需要分析单个SKU的运配时间。根据题目中所给信息，即单个SKU时间包括查询仓位时间、行进时间、登上货架时间、取货时间、扫码出库时间以及卸货时间，同时在订单内部所有SKU完成后，加上在客服区域打印销售单据及返回时间。即满足以下数学公式：

|  |  |
| --- | --- |
|  | () |

其中，表示第个订单(按照时间顺序依次向下排序，即第1个订单为101001)第个SKU运送总时长，表示第个订单内第个SKU。

通过(2)式，小组得到如下时间（以101001订单为例）：

表3 101001订单各个SKU的运送时间与截止时间

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **下单日期** | **下单时间** | **订单**  **编号** | **面料出库量** | **仓库编码** | **订单类别** | **截止**  **时间** | **运送**  **时间** |
| **0** | 45658 | 36900 | 101001 | 14 | 112101 | 镇外订单 | 44100 | 251.056 |
| **1** | 45658 | 36900 | 101001 | 5 | 023003 | 镇外订单 | 44100 | 138.424 |
| **2** | 45658 | 36900 | 101001 | 14 | 042802 | 镇外订单 | 44100 | 239.6 |
| **3** | 45658 | 36900 | 101001 | 5 | 080102 | 镇外订单 | 44100 | 121.496 |
| **4** | 45658 | 36900 | 101001 | 12 | 072103 | 镇外订单 | 44100 | 217.056 |

其中，运送时间不包括总体订单在客服区域打印销售单据及返回时间。

### K-means++聚类分析[1]

K-means算法是一种迭代求解的聚类分析算法，其核心思想是将数据集中的个对象划分为个聚类，使得每个对象到其所属聚类的中心（或称为均值点、质心）的距离之和最小。本题中，=7。

本次建模采用的是K-means++聚类算法，即将不同时间分配的任务进行分类，其选择初始化簇中心之间的相互距离要尽可能里的更远。步骤如下：

**Step1**在数据集中，小组随机选择一个样本点作为第一个初始聚类中心;

**Step2**选择出其余的聚类中心：

计算样本中的每一个样本点与已经初始化的聚类中心之间的距离，并选择其中最短的距离，记为;

计算每个样本点被选为下一个聚类中心的概率，最后选择最大概率值（或者概率分布）所对应的样本点作为下一个簇中心。起计算公式如下：

|  |  |
| --- | --- |
|  | () |

**Step3**重复上述步骤，直到7个聚类中心都被确定。

**Step4** 分配：将每个数据点分配给最近的聚类中心。通过计算样本与各均值向量的距离，根据样本距离最近的均值向量，将样本划入相应的簇。其计算公式如下：

|  |  |
| --- | --- |
|  | () |

其中，是数据点，是第个聚类中心，是数据的维度，分别是在维上的值。

**Step3** 更新：重新计算每个聚类的中心。其计算公式如下：

|  |  |
| --- | --- |
|  | () |

其中，是第个聚类的数据点集合，是该集合中数据点的数量

**Step4** 迭代：重复分配和更新步骤，直到满足终止条件。

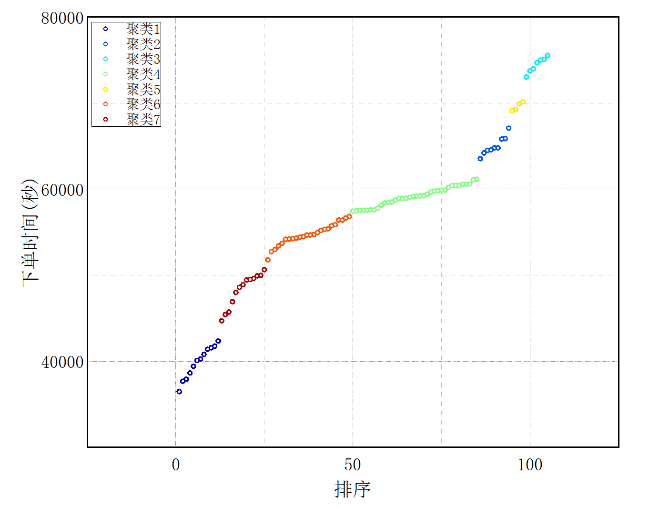
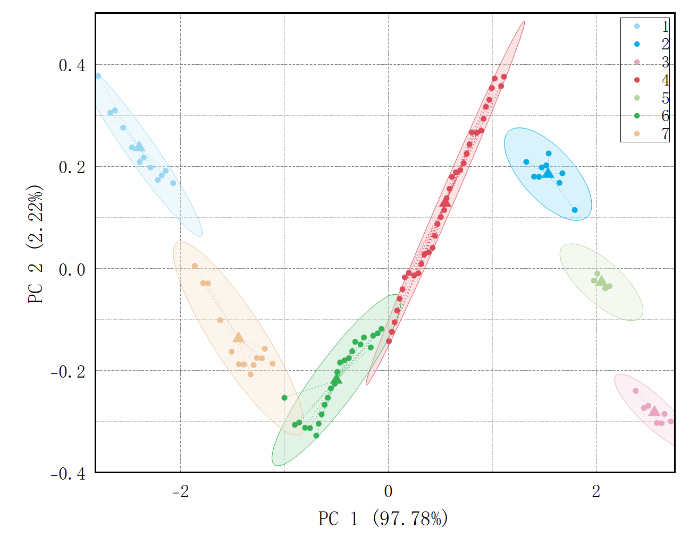
**Step5**  代入数据得出结论最终聚类中心之间的距离以及各组数据到聚类中心的距离。可以得到以下两张图片：

图2 不同下单时间的散点图 图3 不同下单时间的聚类分析图

上图中，图2是在聚类分析前，每个时间段的不同散点分布；图3是在聚类分析后，通过不同订单的不同时间和不同订单次序，对每个时间段分割开来综合考虑从，使后续在模型建立时可以减少算法的复杂度，提升算法的效率。

## 0-1规划模型的建立[2]

在本题中，最终目标为出库订单数量尽可能增加。为达成此目标，即使得所有订单按照订单先后次序以及订单内所有SKU的处理总时间进行排序，从而达到所有订单完成的最优解。而事实上，小组在粗略观察后发现，不可能使得所有订单都能够完成，即需要对目标进行规划以达到最优化目标。

小组在设计模型时，注意到使用0-1规划模型可以很大程度上帮助建立起相关的目标函数。在求解整数规划问题上，若可行域是有界的情况下，可以使用枚举法进行求解，但是这样大多是基于串行计算的方式。而本问中，受限于数据量较大，以此类方式计算会带来巨大计算成本。因此需要对以上思路做进一步的优化。

在附件1中，1月份的订单分为4天，由于订单之间间隔时间远远超过运送时限，因此，小组讨论后认为可以将4天的运送单独考虑，以达到减少串行计算的难度，利用并行计算提升计算速率。以下，为相关规划模型的建立过程：

### 目标函数的建立

由问题一的要求可知，最终目标为使得运输订单尽可能多，则由此建立以下目标函数：

|  |  |
| --- | --- |
|  | () |

在上式（3）中，目标函数为，其中是一个指示函数，即当该订单被采用时则指示为1，当不采用时为0。

### 构建约束条件

由于每个订单的最后在客服区域打印销售单据及返回时间是固定为4分钟，且同一订单不同SKU在这个时间上是相互统一的，因此我们约定，在计算单个SKU的时候，减去其最后的固定时间从而方便计算。因此，我们可以得到下面式子：

|  |  |
| --- | --- |
|  | () |

其中，表示单个SKU最久配送时间，表示不包括在客服区域打印销售单据及返回时间的单个SKU最久配送时间。

同时，为了方便计算和表达，我们规定了一个变量，表达不同订单下单时间的间隔，则有下面这个表达式：

|  |  |
| --- | --- |
|  | () |

其中，表示第个订单下单时间（换算成秒为单位）。

由上5.2.0可知，不能保证所有订单均被处理，因此，为方便后续解答问题，小组规定，使得满足：

|  |  |
| --- | --- |
|  | () |

因此，所有均能保证其对应的订单在配送的订单之内。

综上可得，本题的总约束条件为：

|  |  |
| --- | --- |
|  | () |

### 构建状态函数

由于不同订单之间的SKU数量存在巨大差异，并且在同一个订单中完成不同SKU的时间差异巨大，因此小组决定通过分析叉车的任务完成情况，从而更好的反映不同订单任务完成度。在构建有关叉车配送时间问题时，小组决定构建状态函数来反映不同叉车在完成分配SKU的空闲情况。

在构建状态函数时，小组将叉车完成任务划分成三个步骤，分别是：在第个订单开始前，第个订单进行时，和完成第个订单之后。我们定义，不同叉车任务量为分配到SKU所需要的工作时间。由此，我们引入了状态函数的主要变量，其表达的含义为第个订单开始时，第个叉车剩余的任务量。表示号叉车在第个订单开始前的任务量，表示号叉车第个订单进行时的任务量，表示完成号叉车第个订单之后的任务量。

在订单任务开始前，不同叉车会有两种不同情况，分别是完全空闲状态以及任务进行阶段，且在下一个订单下单时，有：

|  |  |
| --- | --- |
|  | () |

同时，为满足最大化订单完成，小组认为需要在完成任务时对不同叉车剩余的任务量做一个升序排序，从而在下一个订单分配SKU时能将较大任务量的SKU分配给剩余任务量较小的叉车。因此，我们规定一个函数，能满足升序排序，即：

|  |  |
| --- | --- |
|  | () |

在对应的SKU任务完成后，叉车需要分配下一个订单任务。其中，每次结算任务为不同订单下单时间。同时，对应上述升序排序后的需要对应的降序排序后的分配任务，即：

|  |  |
| --- | --- |
|  | () |

其中，表示第个叉车在完成第个任务后分配到的任务，表示降序排列后的。

综上，结合5.2.1的目标函数和5.2.2的约束条件，我们可以得到总体上的目标-约束状态函数，即：

|  |  |
| --- | --- |
|  | () |

## 蒙特卡洛树搜索求解[3]

小组认为，通过蒙特卡洛树搜索可以进一步提升模型质量。[3]作为一种人工智能领域内解决复杂决策问题的算法，此类算法在博弈等路径规划问题上发挥着重要作用，即可以构建搜索树来寻找最优策略。

### 订单优化建模转化成MCTS决策树

小组定义了在初始状态下的状态为蒙特卡洛树的根节点，即未处理任何订单。设置了节点S，包括、和,分别表示已处理订单集合、剩余可处理时间和当前处理订单的累计时间。

在此基础上，我们设计了奖励函数，其核心目标为最大化出库订单数量，并且兼顾时间效率。如下：

|  |  |
| --- | --- |
|  | () |

其中：=1,即表示每个订单得分为1分；作为超时惩罚系数，超时则为扣分。由于本题中超时订单失效，则小组将此数设置为无穷大。

### MCTS分天并行运算

按4天数据将问题拆解为4个子问题，每天单独构建MCTS树。其优点时每天订单数量减少，可以并行运行4个MCTS的实例。

在此基础上，小组设计改进了UCT节点选择策略，以适应订单优化问题：

|  |  |
| --- | --- |
|  | () |

其中，为订单的平均奖励，即该订单处理后的收益预期；为启发式权重，处理时间和订单之间的平衡系数。

并且在随机模拟的基础上，小组采用“贪心+随机”的策略，即按照时间升序排序形式待选订单，并且通过大概率形式选择时间最短订单，小概率形式随机选择订单。

### 算法性能评估

在使用MCTS搜索完成后，小组使用python对算法性能进行评估，得到：

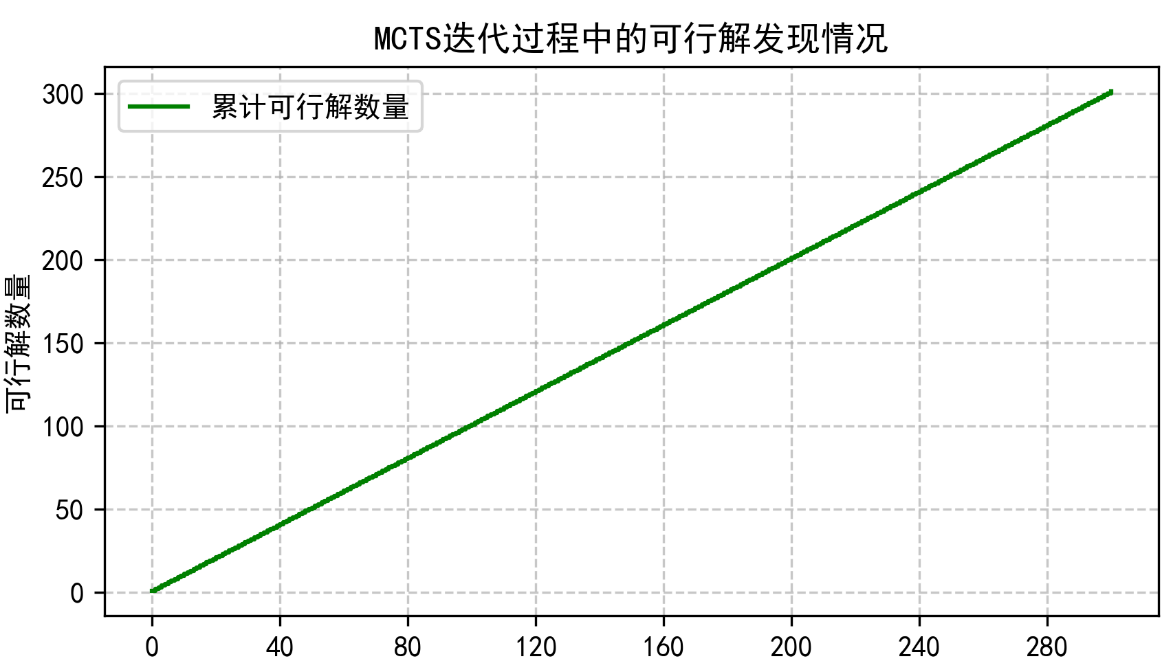
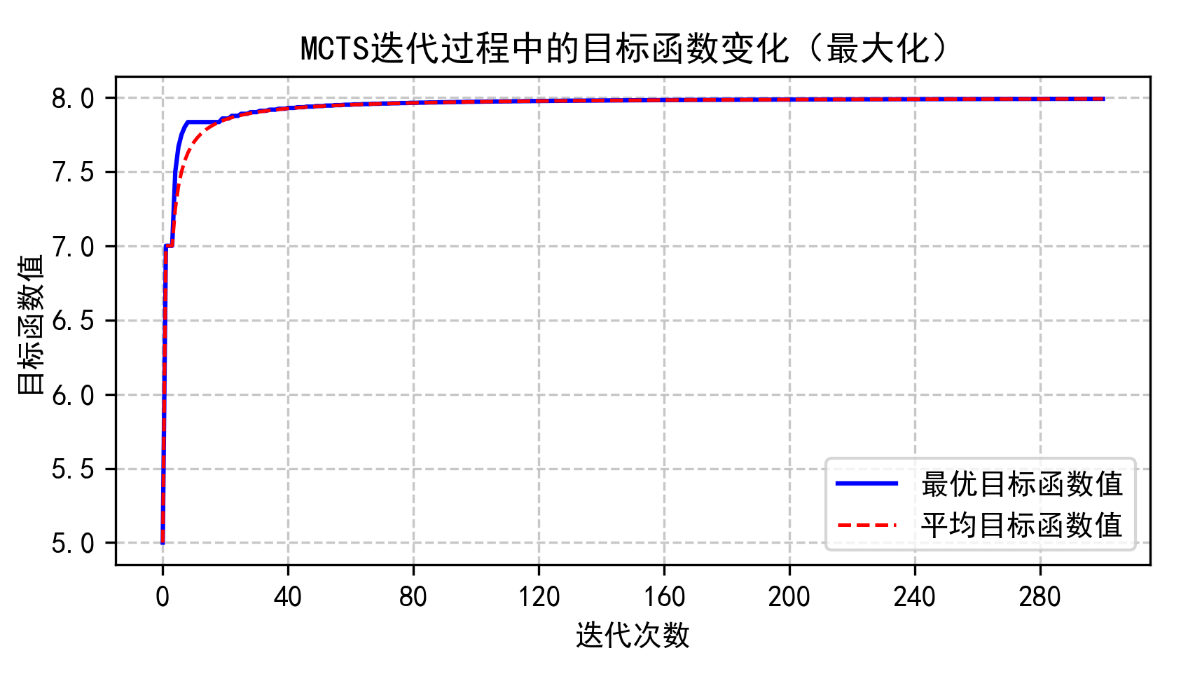


图4（上）迭代次数与累计可行解数量关系

图5（下）迭代次数与目标函数值关系

由上图可知，随着迭代次数增加，可行解数量大幅提升，且相较之前直接运算的方式，使用MCTS搜索后的算法复杂度大幅降低，有助于减少后续运算成本。

## 模型求解

以下，我们截取了两张图作为最后的出库模式及排程方案：

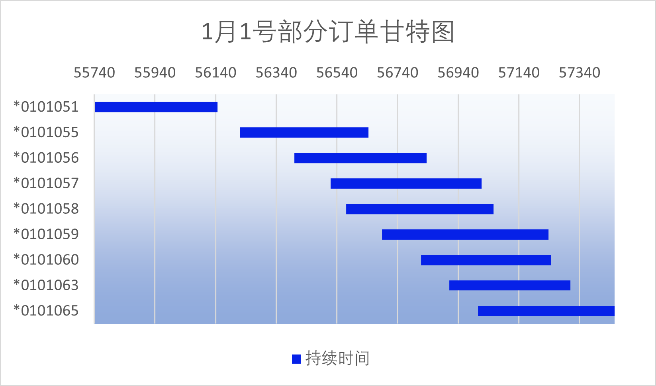
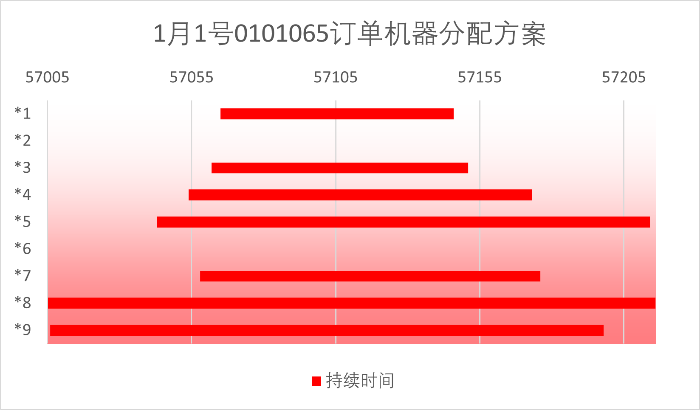
 

图6 1月1日部分订单甘特图 图7 0101065订单叉车分配方案

上图中，左图为订单安排的甘特图（以1月1日部分为示例），右图为不同叉车安排的甘特图（以1月1日0101065订单叉车分配为示例）。下表中，左表展示了订单完成情况（以1月1日部分为例），右表展示了叉车分配情况（以1月1日部分为例）。解答详见附件。

表4 订单完成情况（部分） 表5 叉车分配情况（部分）

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **订单编号** | **起始时间** | **结束时间** |  | **订单编号** | **SKU** | **处理机器号** | **起始时间** | **结束时间** |
| 101001 | 10:15:00 | 10:22:52 |  | 101001 | 1510 | 0 | 10:15:00 | 10:18:52 |
| 101002 | 10:46:00 | 10:52:22 |  | 101001 | 4231 | 1 | 10:15:00 | 10:18:51 |
| 101003 | 10:59:00 | 11:07:37 |  | 101001 | 2633 | 2 | 10:15:00 | 10:18:22 |
| 101004 | 11:03:00 | 11:10:09 |  | 101001 | 0719 | 3 | 10:15:00 | 10:17:15 |
| 101005 | 11:32:00 | 11:39:37 |  | 101001 | 2902 | 4 | 10:15:00 | 10:16:41 |
| 101006 | 11:39:00 | 11:46:32 |  | 101002 | 3016 | 0 | 10:46:00 | 10:48:22 |
| 101007 | 11:44:00 | 11:53:04 |  | 101002 | 5150 | 1 | 10:46:00 | 10:48:19 |
| 101008 | 11:51:00 | 11:58:38 |  | 101002 | 2508 | 2 | 10:46:00 | 10:48:05 |
| 101009 | 11:59:00 | 12:05:48 |  | 101002 | 2748 | 3 | 10:46:00 | 10:47:55 |
| 101010 | 12:01:00 | 12:08:04 |  | 101002 | 3910 | 4 | 10:46:00 | 10:47:43 |
| 101011 | 12:02:48 | 12:10:14 |  | 101002 | 2633 | 5 | 10:46:00 | 10:47:32 |
| 101012 | 12:04:04 | 12:12:39 |  | 101002 | 3501 | 6 | 10:46:00 | 10:47:26 |
| 101013 | 12:07:00 | 12:15:20 |  | 101002 | 0536 | 7 | 10:46:00 | 10:47:18 |
| 101014 | 12:19:00 | 12:28:44 |  | 101003 | 4437 | 0 | 10:59:00 | 11:03:37 |
| 101015 | 12:21:00 | 12:29:35 |  | 101003 | 5203 | 1 | 10:59:00 | 11:02:26 |
| 101016 | 12:21:00 | 12:30:37 |  | 101003 | 1941 | 2 | 10:59:00 | 11:01:38 |
| 101017 | 12:39:00 | 12:47:06 |  | 101003 | 3808 | 3 | 10:59:00 | 11:00:50 |
| 101018 | 12:42:00 | 12:48:39 |  | 101003 | 1124 | 4 | 10:59:00 | 11:00:43 |
| 101019 | 12:47:00 | 12:54:14 |  | 101003 | 4622 | 5 | 10:59:00 | 11:00:36 |
| 101020 | 12:51:00 | 12:59:03 |  | 101003 | 1328 | 6 | 10:59:00 | 11:00:04 |
| 101021 | 13:09:00 | 13:16:41 |  | 101003 | 1707 | 7 | 10:59:00 | 10:59:59 |

# 问题二模型的建立与求解

## TOPSIS模型的建立[4]

TOPSIS模型是一种常用的多属性决策性模型。其核心思路是基于归一化决策矩阵，通过计算各个评价对象和最优方案、最劣方案的相对接近程度，从而评价对象排序。

团队通过对以上多个指标的粗略估计，决定采用TOPSIS模型，以达到尽可能减少人为的外来干预，更加客观地量化不同SKU的热销系数和反季节销量、顺季节销量和总销量之间的关系。

**Step1** 正向化处理表格数据

本题中，由于数据均为正向化数据，因此小组跳过了本步骤。

**Step2** 标准化处理表格数据

标准化处理表格数据最大目的是在于其能够消除不同指标的量纲带来的影响。

在本题中，有1400+2600个评价对象即不同季节（春夏和秋冬）的SKU，3个评价指标（即总订单数量、顺季节订单数量、反季节订单数量），以此构建出标准化矩阵如下：

|  |  |
| --- | --- |
|  | () |

上述矩阵中。表达为春夏矩阵，表达为秋冬矩阵。

然后，对标准化后的矩阵中的每一个元素进行如下操作：

|  |  |
| --- | --- |
|  | () |

**S**tep3 计算得分并归一化处理

这些指标的标准化矩阵𝑍表示为：

|  |  |
| --- | --- |
|  | () |

定义最大值:

|  |  |
| --- | --- |
|  | () |

定义最小值:

|  |  |
| --- | --- |
|  | () |

计算距离：

1. 第个评价对象与最大值的距离：

|  |  |
| --- | --- |
|  | () |

2.第个评价对象与最小值的距离：

|  |  |
| --- | --- |
|  | () |

**Step4 计算得分**

|  |  |
| --- | --- |
|  | () |

## TOPSIS模型得分

我们将销售数据带入到上述TOPSIS模型中，可以得到以下得分：

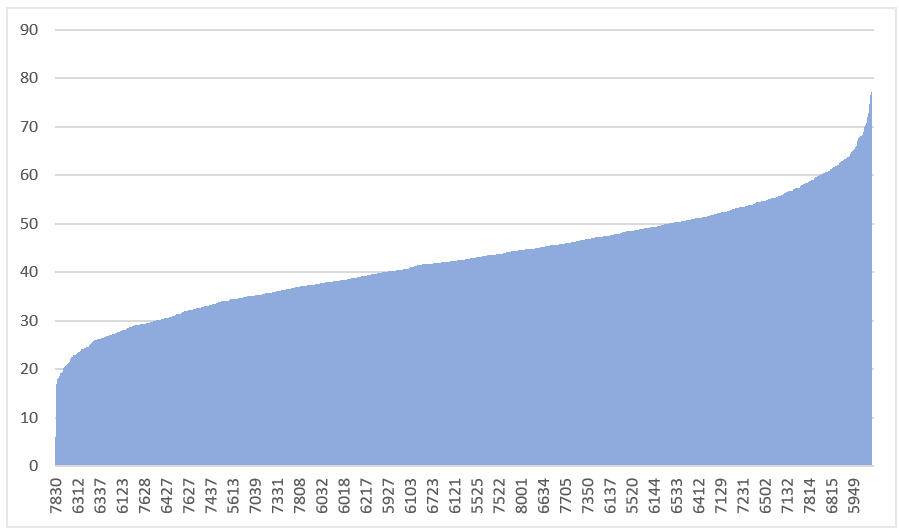
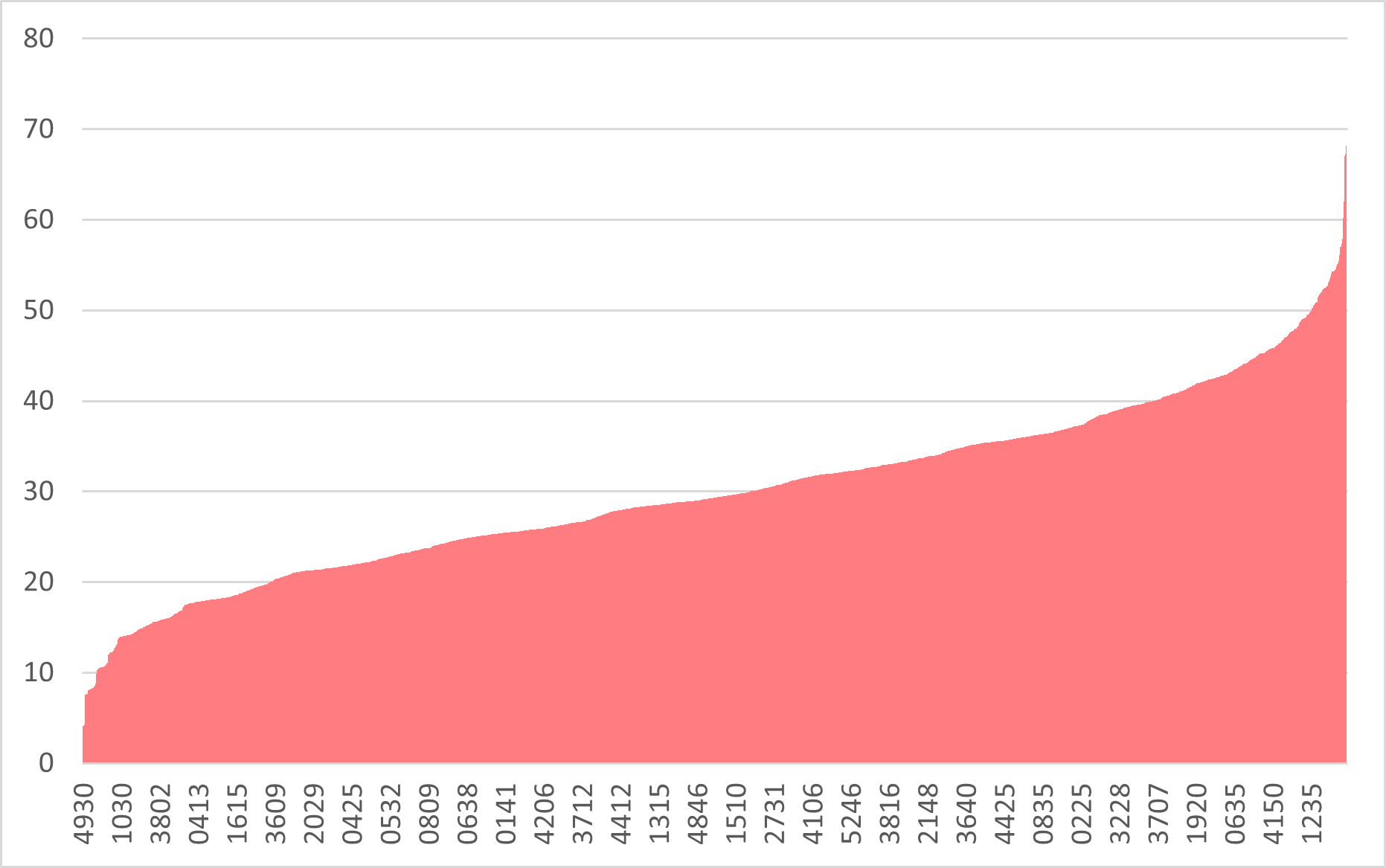
 

图8 秋冬季SKU得分模型 图9 春夏季SKU得分模型

从上图中不难得出，春夏销售评分前三名SKU编号是**：2613、3522、5077；**秋冬销售评分前三名SKU编号是**：6135、5837、5618。**

## 调整热销位置

根据6.3中的不同SKU对应的得分，我们将不同SKU得分按照降序排序，并依次将SKU按照得分排序与相对距离排序相互对应，即距离越近，SKU得分越高。

下图中，左图为没有调整过位置的SKU在不同货架上的位置，右图为根据TOPSIS评分体系后调整的热力图：

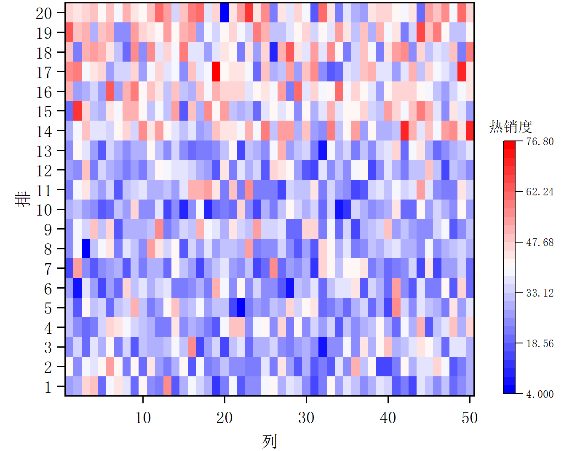
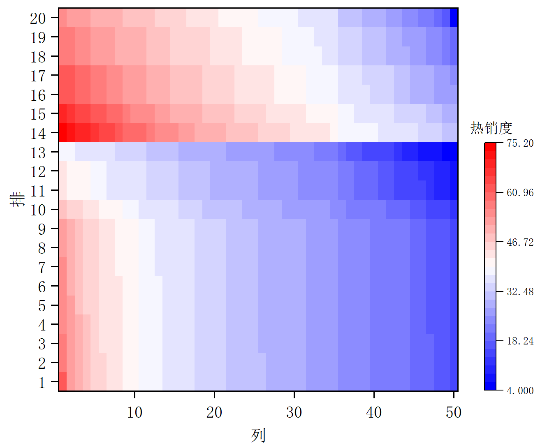
 

图10 调整前热力图 图11 调整后热力图

下表展示了部分调配后商品的位置：

表6 调配后商品的位置（部分）

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 仓位编码 | SKU编码 | 仓位编码 | SKU编码 | 仓位编码 | SKU编码 | 仓位编码 | SKU编码 |
| 010101 | 2613 | 010403 | 3628 | 010801 | 1827 | 011103 | 0644 |
| 010102 | 4303 | 010404 | 2825 | 010802 | 4649 | 011104 | 4405 |
| 010103 | 3522 | 010501 | 3736 | 010803 | 4934 | 011201 | 4335 |
| 010104 | 0346 | 010502 | 1239 | 010804 | 1115 | 011202 | 1847 |
| 010201 | 2508 | 010503 | 0112 | 010901 | 2811 | 011203 | 0537 |
| 010202 | 5227 | 010504 | 4417 | 010902 | 1137 | 011204 | 2908 |
| 010203 | 3042 | 010601 | 0315 | 010903 | 0304 | 011301 | 2903 |
| 010204 | 3446 | 010602 | 2832 | 010904 | 0735 | 011302 | 2225 |
| 010301 | 1323 | 010603 | 2106 | 011001 | 4748 | 011303 | 0150 |
| 010302 | 0546 | 010604 | 2318 | 011002 | 0437 | 011304 | 3214 |
| 010303 | 0946 | 010701 | 1541 | 011003 | 4620 | 011401 | 1612 |
| 010304 | 0439 | 010702 | 2830 | 011004 | 1332 | 011402 | 0849 |
| 010401 | 3215 | 010703 | 1923 | 011101 | 0707 | 011403 | 2518 |
| 010402 | 4609 | 010704 | 3039 | 011102 | 4023 | 011404 | 2026 |

## 调配方案

将6.2中TOPSIS模型带入问题一中的0-1规划模型，即带入公式（14），得到调整后的位置，即：

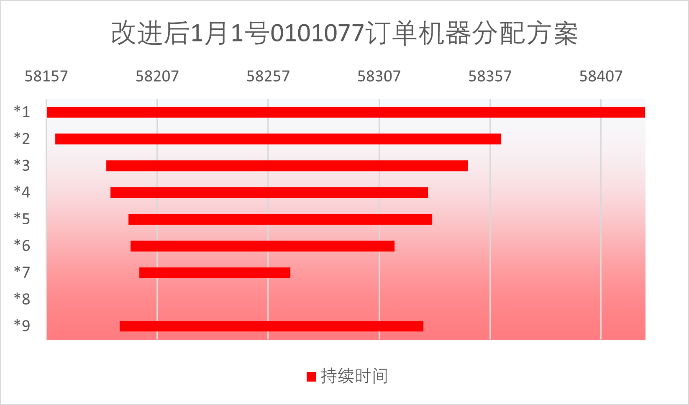
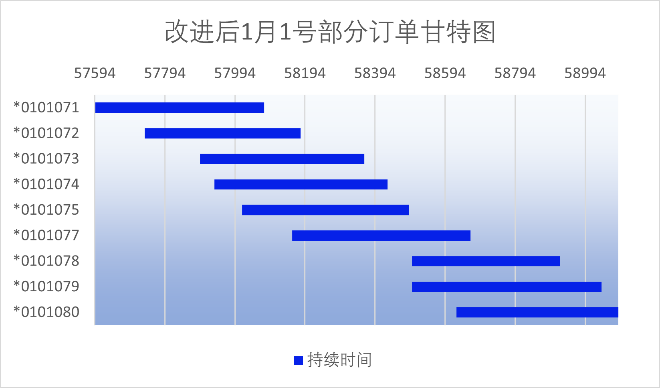


图12 改进后1月1日部分订单甘特图 图13 改进后0101077订单叉车分配方案

上图中，左图为订单安排的甘特图（以改进后1月1日部分订单为示例），右图为不同叉车安排的甘特图（以改进后1月1日0101077订单叉车分配为示例）。下表中，左表展示了订单完成情况（以1月1日部分为例），右表展示了叉车分配情况（以1月1日部分为例）。解答详见附件。

表7 订单完成情况（部分） 表8 叉车分配情况（部分）

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **订单编号** | **起始时间** | **结束时间** |  | **订单编号** | **SKU** | **处理机器号** | **起始时间** | **结束时间** |
| 101001 | 10:15:00 | 10:22:50 |  | 101001 | 1510 | 0 | 10:15:00 | 10:18:50 |
| 101002 | 10:46:00 | 10:52:30 |  | 101001 | 4231 | 1 | 10:15:00 | 10:18:29 |
| 101003 | 10:59:00 | 11:06:55 |  | 101001 | 2633 | 2 | 10:15:00 | 10:18:02 |
| 101004 | 11:03:00 | 11:10:26 |  | 101001 | 0719 | 3 | 10:15:00 | 10:17:17 |
| 101005 | 11:32:00 | 11:39:21 |  | 101001 | 2902 | 4 | 10:15:00 | 10:17:10 |
| 101006 | 11:39:00 | 11:46:45 |  | 101002 | 3016 | 0 | 10:46:00 | 10:48:30 |
| 101007 | 11:44:00 | 11:52:28 |  | 101002 | 3910 | 1 | 10:46:00 | 10:48:07 |
| 101008 | 11:51:00 | 11:59:07 |  | 101002 | 2508 | 2 | 10:46:00 | 10:48:05 |
| 101009 | 11:59:00 | 12:05:44 |  | 101002 | 3501 | 3 | 10:46:00 | 10:47:40 |
| 101010 | 12:01:00 | 12:08:22 |  | 101002 | 5150 | 4 | 10:46:00 | 10:47:24 |
| 101012 | 12:04:22 | 12:12:27 |  | 101002 | 2748 | 6 | 10:46:00 | 10:47:14 |
| 101013 | 12:07:00 | 12:15:26 |  | 101002 | 2633 | 7 | 10:46:00 | 10:47:12 |
| 101014 | 12:19:00 | 12:28:23 |  | 101003 | 4437 | 0 | 10:59:00 | 11:02:55 |
| 101015 | 12:21:00 | 12:29:49 |  | 101003 | 5203 | 1 | 10:59:00 | 11:01:47 |
| 101016 | 12:21:00 | 12:29:39 |  | 101003 | 1941 | 2 | 10:59:00 | 11:01:18 |
| 101017 | 12:39:00 | 12:46:46 |  | 101003 | 3808 | 3 | 10:59:00 | 11:00:51 |
| 101018 | 12:42:00 | 12:49:12 |  | 101003 | 1124 | 4 | 10:59:00 | 11:00:48 |
| 101019 | 12:47:00 | 12:54:16 |  | 101003 | 4622 | 5 | 10:59:00 | 11:00:45 |
| 101020 | 12:51:00 | 12:59:12 |  | 101003 | 1707 | 6 | 10:59:00 | 11:00:27 |
| 101021 | 13:09:00 | 13:16:16 |  | 101003 | 1328 | 7 | 10:59:00 | 11:00:02 |

# 模型优缺点评价

**7.1 模型的优点**

本文创新点在于小组在两个问题求解过程中，创新性地使用蒙特卡洛树搜索（MCTS）算法进行并行计算从而实现最终算法优化。该算法可以极大提升算法效率，其记忆化模拟与优先级节点拓展极大减少了运算的次数；同时在策略改进方面，提升了订单完成数地核心设计，借助动态将来函数，减少了树的深度，避免了状态的爆炸。

同时TOPSIS法具有易懂性、计算简易性、考虑方面周到性、结果直观性等优点,同时对数据分布不敏感，鲁棒性较好。

最后，本文在数据预处理阶段使用了K-means++聚类分析算法，使得初始化聚类中心选择更加科学，收敛效果显著提升；同时聚类结果稳定性相对较强，抗噪声能力较强，适用于大规模的数据。

**7.2 模型的缺点**

1. 在TOPSIS模型中，数据有关权重均为小组主观赋值，可能受到个人偏好的影响。并且有关销量的部分可能会有一定程度上的权重重叠，对结果产生轻微影响。
2. 在K-means++聚类分析算法中，我们预设的聚类数可能存在一定的不合理性，从而导致后续数据样本中心冗余、簇内样本稀疏。

参考文献

1. 曾艳. 0-1规划中并行隐枚举法的实现方式[J]. 计算机应用与软件,2010,27(7):268-269,289.
2. 林雯琦,李薇.基于0-1整数规划的高校实验排课方案的研究[J].福建电脑,2018,34(12):46-47.DOI:10.16707/j.cnki.fjpc.2018.12.023.
3. Browne C B, Powley E, Whitehouse D, et al. A survey of monte carlo tree search methods[J]. IEEE Transactions on Computational Intelligence and AI in games, 2012, 4(1): 1-43.
4. 高钰晶.基于 DPSIR-TOPSIS 模型的兰州市大气污染治理绩效评估研究[D].兰州大学,2024.DOI:10.27204/d.cnki.glzhu.2023.002284.

附录

|  |
| --- |
| 附录1 |
| 介绍：问题1的python代码 |
| import numpy as np  import pandas as pd  import random  from collections import defaultdict  from openpyxl.utils.dataframe import dataframe\_to\_rows  class MCTSNode:  """蒙特卡洛树搜索的节点"""  def \_\_init\_\_(self, state, parent=None):  self.state = state # 当前部分解 (0-1向量)  self.parent = parent  self.children = []  self.visits = 0  self.value = 0.0 # 越小越好 (最小化问题)  self.fully\_expanded = False  class BinaryProgrammingSolver:  """求解0-1规划问题的求解器"""  def \_\_init\_\_(self, n, objective, constraints, max\_iter=1000, exploration\_weight=1.0, weights=None):  self.n = n # 变量数量  self.objective = objective # 目标函数  self.constraints = constraints # 约束条件列表  self.max\_iter = max\_iter # 最大迭代次数  self.exploration\_weight = exploration\_weight # UCT探索权重  self.weights = weights # 变量权重，用于启发式  def solve(self, warm\_start=None):  """执行MCTS算法求解问题，支持预热解"""  # 使用预热解或初始化为全零向量  initial\_state = warm\_start.copy() if warm\_start is not None else np.ones(self.n, dtype=int)  # 验证预热解的可行性  if warm\_start is not None and not self.\_check\_constraints(initial\_state):  print("警告：提供的预热解不可行，将使用默认初始状态")  initial\_state = np.ones(self.n, dtype=int)  # 初始化根节点  root = MCTSNode(state=initial\_state)  for \_ in range(self.max\_iter):  # 选择  node = self.\_select(root)  # 扩展  if not self.\_is\_terminal(node.state):  node = self.\_expand(node)  # 模拟  reward = self.\_simulate(node.state)  # 反向传播  self.\_backpropagate(node, reward)  # 返回最优解  best\_node = self.\_get\_best\_child(root, exploration\_weight=0, warm\_start=warm\_start)  return {  'success': best\_node is not None,  'x': best\_node.state if best\_node else None,  'fun': best\_node.value if best\_node else None  }  def \_select(self, node):  """选择要扩展的节点"""  while not self.\_is\_terminal(node.state):  if not node.fully\_expanded:  return self.\_expand(node)  else:  node = self.\_get\_best\_child(node, self.exploration\_weight)  return node  def \_expand(self, node):  """优化扩展策略，只生成有潜力的子节点"""  # 计算每个未设置变量的"价值分数"  scores = []  for i in range(self.n):  if node.state[i] == 0:  # 使用目标函数权重作为启发式（如果有）  score = self.weights[i] if self.weights is not None else 1.0  scores.append((i, score))  # 按分数排序，优先扩展高分变量  scores.sort(key=lambda x: x[1], reverse=True)  # 只扩展前k个变量，避免生成过多子节点  k = min(10, len(scores)) # 限制每次扩展的数量  unexpanded\_actions = []  for i, \_ in scores[:k]:  new\_state = node.state.copy()  new\_state[i] = 1  # 提前检查约束（如果可能）  if self.\_check\_constraints(new\_state):  unexpanded\_actions.append(new\_state)  if not unexpanded\_actions:  node.fully\_expanded = True  return node  # 随机选择一个动作进行扩展  action = random.choice(unexpanded\_actions)  child\_node = MCTSNode(state=action, parent=node)  node.children.append(child\_node)  # 检查是否已完全扩展  if len(node.children) >= k or len(node.children) == self.n - np.sum(node.state):  node.fully\_expanded = True  return child\_node  def \_simulate(self, state):  """基于启发式的模拟，而非纯随机"""  current\_state = state.copy()  # 按权重降序排列未设置的变量  unset\_indices = np.where(current\_state == 0)[0]  if self.weights is not None:  # 使用目标函数的权重作为启发式  weights = self.weights[unset\_indices]  sorted\_indices = unset\_indices[np.argsort(-weights)]  else:  sorted\_indices = unset\_indices  # 贪心补全剩余变量  for i in sorted\_indices:  # 尝试设置为1  current\_state[i] = 1  if not self.\_check\_constraints(current\_state):  # 如果不可行，则撤销  current\_state[i] = 0  # 检查最终解的可行性  if not self.\_check\_constraints(current\_state):  return float('inf')  return self.objective(current\_state)  def \_backpropagate(self, node, reward):  """反向传播结果"""  while node is not None:  node.visits += 1  # 更新平均值 (越小越好)  if node.value == 0.0:  node.value = reward  else:  node.value = (node.value \* (node.visits - 1) + reward) / node.visits  node = node.parent  def \_get\_best\_child(self, node, exploration\_weight, warm\_start=None):  """使用UCT公式选择最优子节点，支持预热解引导"""  best\_value = -float('inf')  best\_child = None  for child in node.children:  # 基础UCT值  if child.visits == 0:  uct\_value = float('inf')  else:  exploitation = -child.value  exploration = np.sqrt(np.log(node.visits) / child.visits)  uct\_value = exploitation + exploration\_weight \* exploration  # 添加预热解的倾向性（如果提供）  if warm\_start is not None:  # 计算与预热解的相似度（汉明距离的倒数）  similarity = 1.0 - np.sum(np.abs(child.state - warm\_start)) / self.n  # 增加相似度得分  uct\_value += 0.5 \* similarity # 权重0.5可调整  if uct\_value > best\_value:  best\_value = uct\_value  best\_child = child  return best\_child  def \_is\_terminal(self, state):  """检查状态是否为终态"""  return np.sum(state) == self.n # 所有变量都已设置为1  def \_check\_constraints(self, state):  """增量式约束检查，提前终止无效解"""  # 跟踪已选中的变量索引和数量  selected\_indices = np.where(state == 1)[0]  selected\_count = len(selected\_indices)  # 对每个约束进行快速检查  for constraint in self.constraints:  # 完整检查（保留原始逻辑）  if not constraint(state):  return False  return True  # 生成预热解的辅助函数  def generate\_warm\_start(n, constraints, max\_attempts=1000):  """生成一个简单的可行解作为预热解"""  for \_ in range(max\_attempts):  # 随机生成候选解  x = np.random.choice(2, p=[0.3,0.7],size=n)  # 检查约束  if all(constraint(x) for constraint in constraints):  return x  # 如果尝试多次仍无法生成可行解，返回全零解  return np.ones(n, dtype=int)  # 示例使用  if \_\_name\_\_ == "\_\_main\_\_":  df = pd.read\_excel('output1.xlsx')  n = 0  m = []  a = []  s = []  k = []  for i in df.index:  o = df["订单编号"][i] % 1000  j = df["编号"][i]  if j == 0:  a.append(df["截止时间"][i] - df["下单时间"][i] - 240)  s.append(df["下单时间"][i])  n += 1  if i > 0:  u.sort(reverse=True)  m.append(u)  k.append(df["编号"][i - 1] + 1)  u = []  u.append(df["时间"][i])  u.sort(reverse=True)  m.append(u)  k.append(df["编号"][len(df) - 1] + 1)  # 定义目标函数 (示例: 最小化选中变量的权重和)  hh=np.ones(n)  weights = np.array(hh)  def objective(x):  return np.sum(x)  # 定义黑盒约束条件  def black\_box\_constraint(x):  t=[]  tt=[]  ss=[]  cc=[]  aa=[]  mm=[]  kk=[]  nn=0  for i in range(n):  if x[i]>0.5:  ss.append(s[i])  aa.append(a[i])  mm.append(m[i])  kk.append(k[i])  nn+=1  for i in range(nn-1):  cc.append(ss[i+1]-ss[i])  for i in range(nn):  t.append([])  tt.append([])  for j in range(9):  if i==0 or tt[i-1][j]-cc[i-1]<0:  t[i].append(0)  else:  t[i].append(tt[i-1][j]-cc[i-1])  t[i].sort()  for j in range(9):  if j<kk[i]:  tt[i].append(t[i][j]+mm[i][j]\*1.7)  else:  tt[i].append(t[i][j])  if tt[i][j]>aa[i]:  return False  return True  def another\_constraint(x):  """另一个约束: 至少选中两个变量"""  return np.sum(x) >= 2  constraints = [  black\_box\_constraint,  another\_constraint  ]  # 生成预热解  warm\_start = generate\_warm\_start(n, constraints)  print(f"生成的预热解: {warm\_start}")  # 创建求解器并求解  solver = BinaryProgrammingSolver(  n=n,  objective=objective,  constraints=constraints,  max\_iter=10000,  exploration\_weight=1.0,  weights=weights # 传递权重用于启发式  )  result = solver.solve(warm\_start=warm\_start)  # 输出结果  if result['success']:  print("最优解找到!")  print(f"最优值: {result['fun']}")  print(f"最优解: {result['x']}")  print(sum(result['x']))  dd=pd.DataFrame(result['x'])  #print(dd)  dd.to\_excel("out.xlsx")  print(n)  else:  print("未找到可行解") |

|  |
| --- |
| 附录2 |
| 介绍：问题2TOPSIS的python代码 |
| import pandas as pd  df=pd.read\_excel('附件1-订单数据.xlsx')  df1=pd.read\_excel('附件2：SKU仓位信息.xlsx')  dd=pd.read\_excel('outmode1.xlsx')  dic={}  ifo={}  print(dd)  for i in df1.index:      s1=str(int(df1["SKU编码"][i])+10000)[1:5]      s2=str(int(df1["仓位编码"][i])+1000000)[1:7]      dic[s1]=s2  for i in df.index:      ss=df["SKU编码"][i]      s = df["每个SKU对应的出库量（单位：卷）"][i]      m=0      j=1      num=[]      while (s[j] != ']'):          if '0' <= s[j] <= '9':              m = m \* 10 + int(s[j])          else:              num.append(m)              m = 0          j += 1      num.append(m)      print(num)      for k in range(0,len(ss),6):          tt=ss[k:k+4]          if not tt in ifo:              ff=int(dic[tt][0:2])              f=1              if ff<=13:                 f=0              ifo[tt]=[f,0,0,0]          if 3<=int(df["订单编号"][i]//100000)<=8:              if ifo[tt][0]:                  ifo[tt][2]+=1              else:                  ifo[tt][1]+=1          else:              if ifo[tt][0]:                  ifo[tt][1]+=1              else:                  ifo[tt][2]+=1          ifo[tt][3]+=num[k//6]  for i in ifo.keys():      dd.loc[len(dd)]=[i]+ifo[i]  print(dd)  dd.to\_excel("out.xlsx") |

|  |
| --- |
| 附录3 |
| 介绍：方案转调度的python代码 |
| from idlelib.pyparse import trans  import pandas as pd  from decorator import append  dic={}  df=pd.read\_excel('output1.xlsx')  df1=pd.read\_excel('outg1.xlsx')  df11=pd.read\_excel('调整后SKU仓位信息对应表.xlsx')  dd=pd.read\_excel('outmode11.xlsx')  dc=pd.read\_excel('outmode12.xlsx')  def ttt(n):      if n<10:          return '0'+str(n)      return str(n)  def trans(t):      t=int(t)      return ttt(t//3600)+":"+ttt(t//60%60)+":"+ttt(t%60)  for i in df11.index:      s2=str(int(df11["SKU编码"][i])+10000)[1:5]      s1=str(int(df11["仓位编码"][i])+1000000)[1:7]      dic[s1]=s2  class jiqi:      def \_\_init\_\_(self, id, num):          self.id = id          self.num = num      def \_\_lt\_\_(self, other):          # 定义小于运算符，按年龄比较          return self.num < other.num  # 直接排序  n = 0  m = []  a = []  s = []  k = []  z = []  for i in df.index:      o = df["订单编号"][i] % 1000      j = df["编号"][i]      if j == 0:          a.append(df["截止时间"][i] - df["下单时间"][i] - 240)          s.append(df["下单时间"][i])          n += 1          if i > 0:              u.sort(reverse=True)              m.append(u)              k.append(df["编号"][i - 1] + 1)              z.append(df["订单编号"][i - 1])          u = []      u.append(jiqi(dic[str(int(df["仓库编码"][i])+1000000)[1:7]],df["时间"][i]))  u.sort(reverse=True)  m.append(u)  k.append(df["编号"][len(df) - 1] + 1)  z.append(df["订单编号"][len(df) - 1])  t = []  tt = []  ss = []  cc = []  aa = []  mm = []  kk = []  zz = []  x=[]  nn = 0  for i in df1.index:      x.append(df1[0][i])  for i in range(n):      if x[i] > 0.5:          ss.append(s[i])          aa.append(a[i])          mm.append(m[i])          kk.append(k[i])          zz.append(z[i])          nn += 1  for i in range(nn - 1):      cc.append(ss[i + 1] - ss[i])  for i in range(nn):      minn=9999999999999      maxn=0      bh=zz[i]      t.append([])      tt.append([])      for j in range(9):          for k in range(9):              if i!=0 and tt[i-1][k].id==j:                  qq=tt[i-1][k].num          if i == 0 or qq - cc[i - 1] < 0:              pp=jiqi(j,0)              t[i].append(pp)          else:              t[i].append(jiqi(j,qq - cc[i - 1]))      t[i].sort()      ifo1=[]      for j in range(kk[i]):          ifo1.append([zz[i],mm[i][j].id,0,0,0])      for j in range(9):          if j < kk[i]:              tt[i].append(jiqi(j,t[i][j].num + mm[i][j].num))              ifo1[j][2]=t[i][j].id              ifo1[j][3]=int(ss[i]+t[i][j].num)              ifo1[j][4]=int(ss[i]+t[i][j].num+mm[i][j].num)              minn=min(ss[i]+t[i][j].num,minn)              maxn=max(ss[i]+t[i][j].num+mm[i][j].num,maxn)          else:              tt[i].append(jiqi(j,t[i][j].num))          if tt[i][j].num > aa[i]:              print("?????????????????????")      dc.loc[len(dc)]=[bh,int(minn),int(maxn+240)]      for j in range(kk[i]):          dd.loc[len(dd)]=ifo1[j]  print(dc)  print(dd)  dd.to\_excel("改进机器分配.xlsx")  dc.to\_excel("改进订单完成情况.xlsx") |

|  |
| --- |
| 附录4 |
| 介绍：时间计算的python代码 |
| import numpy as np  import pandas as pd  def tim(s,m):  ans=40+11\*m  x=[0,0,0,0,0,0,0,0,0,1,3,3,5,5,6,8,8,10,10,11]  y=[0,0,0,0,0,0,0,0,0,1,2,2,3,4,5,6,6,7,7,8]  lc=8-0.72  p=int(s[0:2])  l=int(s[2:4])  lc+=l\*1.44  if l>35:  lc+=6  lc+=x[p-1]\*1.8+y[p-1]\*2  ans+=lc\*2/2.5  return ans  df=pd.read\_excel('附件1-订单数据.xlsx')  df1=pd.read\_excel('调整后SKU仓位信息对应表.xlsx')  print(df1)  dd=pd.read\_excel('outmode.xlsx')  dic={}  print(dd)  for i in df1.index:  s1=str(int(df1["SKU编码"][i])+10000)[1:5]  s2=str(int(df1["仓位编码"][i])+1000000)[1:7]  dic[s1]=s2  for i in df.index:  ss=df["SKU编码"][i]  s=df["每个SKU对应的出库量（单位：卷）"][i]  num=[]  name=[]  j=1  tt=0  ssum=0  while(s[j]!=']'):  if '0'<=s[j]<='9':  tt=tt\*10+int(s[j])  else:  num.append(tt)  ssum+=tt  tt=0  j+=1  num.append(tt)  ssum += tt  for k in range(0,len(ss),6):  name.append(dic[ss[k:k+4]])  if sum(num)!=df["面料总出库量（单位：卷）"][i] or len(num)!=len(name):  print("!!!!!!!!!!!!!!!")  sttm=int(df["下单时间"][i][0:2])\*3600+int(df["下单时间"][i][3:5])\*60  edtm=sttm+3600  if(df["订单类别"][i]=="镇外订单"):  edtm+=3600  n=len(name)  for j in range(n):  dd.loc[len(dd)]=[df["下单日期"][i],sttm,df["订单编号"][i],num[j],name[j],df["订单类别"][i],edtm,tim(name[j],num[j]),j]  print(dd)  dd.to\_excel("output1.xlsx") |