基于 0-1 整数规划模型的订单配送优化问题

摘要

物流服务的优质与否是决定电商、零售企业的成败关键因素。本文针对如何节约货物配送成本，提高服务效率，提出了完整的模型于求解，以助于企业进行配送的优化。

针对问题 1：在本问中，优化目标是极小化所有运输车辆的总停车次数。为了使得求解过程更为顺利，我们建立了 **0-1 整数规划模型**，将车辆是否停车，是否搬运货物等信息统一用 0-1 矩阵表示，并以此建立单目标优化模型，根据题目要求设置约束条件。在求解过程中，我们使用**贪心策略**，使每辆车停车卸货时尽可能携带其他的订单。求解得出四个实例的总停车次数最小值分别为：82、159、162、636。

针对问题 2：相对于问题 1，本问中每辆运输车最多同时携带的订单件数从变成了 2 件，因此需要修改约束条件，重新建立单目标优化模型。此时求解的复杂度大大增加，使用贪心算法无法准确求解，于是使用**模拟退火算法**进行解的优化，通过不断迭代产生新的最优解。求解得出四个实例的总停车次数最小值分别为： 62、131、139、583。

针对问题 3：本问中的最优化目标改为极小化所使用运输车辆总个数，此外设置了新的约束条件，规定车辆不能空载。因此需要根据新的优化目标和约束条件进行求解。继续使用贪心策略求得初始解，之后使用模拟退火算法进行优化，求解得出四个实例的使用运输车辆最小值分别为：16、28、29、104。

针对问题 4：本问当中需要考虑运输车辆总个数和极小化所有运输车辆的总停车次数的两个目标。因此需要建立双目标优化模型。为了便于求解，我们引入**时间损耗**的概念来描述两个目标的重要程度，并通过**线性加权和法**赋予两个目标不同的权重，来将双目标优化改为单目标优化，从而进行求解。通过贪心策略和模拟退火算法求解得出了综合考虑使用运输车辆和停车次数最小化的解。

最后，我们对模型进行了**优缺点分析以及推广**，期望得到模型的优化以及进一步应用。

关键词： 0-1 整数规划模型；贪心策略；模拟退火算法；线性加权和法

目录

一、问题重述 3
二、问题分析 4
三、问题假设 4
四、符号说明 5
五、模型的建立与求解 5
5.1 问题一模型的建立与求解 5
5.1.1 模型的准备 5
5.1.2 0-1 整数规划模型的建立 6
5.1.3 模型的求解 8
5.2 问题二模型的建立与求解 9
5.2.1 模型的建立 9
5.2.1 模拟退火算法对模型的求解 10
5.3 问题三模型的建立与求解 13
5.3.1 模型的建立 13
5.3.2 模型的求解 14
5.4 问题四模型的建立与求解 14
5.2.1 模型的建立 14
5.2.2 线性加权和法转化为单目标优化的求解 15
六 模型评价与推广 15
6.1 模型的优点分析 15
6.2 模型的缺点分析 16
6.3 模型的改进与推广 16
参考文献 17
附录 18

一、问题重述

客户订单通常由到达城市 H 和从城市 H 寄出的两类订单组成，到达城市 H 的订单首先会被分配到某一个站点，然后再经快递公司运输车配送到目的地所属的配送站点；而寄出城市 H 的订单首先要经快递公司运输车从出发地所属站点配送到城市 H 的某一个站点，再经该站点被配送到市内或市外目的地。即我们可以假定每一个订单在城市 H 都有一个初始站点和结束站点，允许多个订单具有相同的初始站点和结束站点，为简化问题，我们假定所有站点都排列在一条直线上并依次从小到大进行标号，且每个订单的初始站点标号小于其结束站点标号。假设快递公司 K 在城市 H 共有 n 个站点，某时段内共有 m 件订单需要在 n 个站点之间进行运输。每一个订单 d1 从其初始站点出发通过运输车配送到其结束站点，一旦订单 d1 到达其结束站点将立即被搬运下车（运输车辆停车），同时若该结束站点也是另一订单 d2 的初始站点，则运输车可以把订单 d2 搬运到车辆上继续进行配送。但每辆运输车的容量相同且有限，既每辆运输车最多同时携带的订单件数不超过 C。请你们为快递公司 K 制定订单协同配送方案，回答以下四个问题：

问题 1：假设快递公司 K 的运输车辆无限多，优化目标是极小化所有运输车辆的总停车次数。针对 C=1 的情形，建立此问题的一般数学模型，设计高效的求解算法给出最优订单协同配送方案，并利用数据中的四个实例进行验证。分别给出四种情况下的配送方案。

问题 2：假设快递公司 K 的运输车辆无限多，优化目标是极小化所有运输车辆的总停车次数。针对 C=2 的情形，建立此问题的一般数学模型，设计高效的求解算法给出最优订单协同配送方案，并利用数据中的四个实例进行验证。分别给出四种情况下的配送方案。

问题 3：假设快递公司 K 的运输车辆有限，而且当运输车把它上面的最后一个订单配送到结束站点后，该运输车即结束其配送运输服务（所有车辆运行方向都是从左往右，不允许调头）。优化目标是极小化所使用运输车辆总个数。针对 C=2 的情形，建立此问题的一般数学模型，设计高效的求解算法给出最优订单协同配送方案，并利用数据中的四个实例进行验证。分别给出四种情况下的配送方案。

问题 4：假设快递公司 K 的运输车辆有限，同时考虑极小化所使用运输车辆总个数和极小化所有运输车辆的总停车次数的两个目标。针对 C=2 的情形，建立此问题的一般数学模型，设计高效的求解算法给出最优订单协同配送方案，要求同时考虑停车次数和运输车辆的最优解，车辆不能存在空载的状况，即并利用数据中的四个实例进行验证。分别给出四种情况下的配送方案。

二、问题分析

对于问题 1：要求建立极小化所有运输车辆的总停车次数，并求出最优订单协同配送方案。其中，每辆车最多只能携带一件订单数，每个订单也只能被运送一次。可以考虑建立 0-1 整数规划模型，将订单信息，车辆信息都变为 0，1 矩阵，对此进行计算求解，因为每辆车最多只能携带一件订单数，问题较为简单，因此考虑直接使用贪心算法进行求解，所求解即为最优解。

对于问题 2：要求建立极小化所有运输车辆的总停车次数，并求出最优订单协同配送方案。其中，每辆车最多只能携带两件订单，因为携带订单的最大值改变，其解的规模也大幅度增大，因此弹性算法所求得的解不一定最最优解，考虑使用遗传算法进行局部最优解的优化，重新规划约束条件，进行目标方程的最优化求解。

对于问题 3：要求建立极小化所有运输车辆的总停车次数，并求出最优订单协同配送方案。其中，每辆车最多只能携带两件订单。此外需要新增约束条件，当运输车把它上面的最后一个订单配送到结束站点后，该运输车即结束其配送运输服务，即运输车不能空载。重新规划约束条件，进行目标方程的最优化求解。

对于问题 4：要求建立极小化所有运输车辆的总停车次数，并求出最优订单协同配送方案。本问新增了一个优化目标，为多目标优化模型，因此考虑采用分别给两个目标赋予不同的权重，从而将其转化为单目标优化模型，使用和第三问相同的约束条件，进行目标方程的最优化求解。

三、问题假设

1. 每件货物只能由一辆运输车来运输
2. 系统有一个配送中心，该中心拥有固定数量的多台配送车辆，而且每台配送车辆的最大装载已知且完全相同。
3. 所有车辆运行方向都是从左往右，不允许调头。
4. 在第三、四问中，当运输车把它上面的最后一个订单配送到结束站点后，该运输车即结束其配送运输服务，即运输车辆不能空载。
5. 配送中心有足够数量的车辆满足中心每天的配送需求。
6. 订单货物在配送中心滞留期间不考虑库存成本。

四、符号说明

符号	含义
i	第 i 辆车
j	第 j 个订单
t	第 t 个站点
N	车辆总数
M	停车总数
n	站点总数
m	订单总数
A_{jt}	订单运输矩阵
B_{jt}	订单信息矩阵
C_{ij}	车辆订单矩阵
X_{it}	车辆停车矩阵
Y_{it}	车辆站点货物矩阵

五、模型的建立与求解

5.1 问题一模型的建立与求解

5.1.1 模型的准备

快递是快递公司快速收集、运输和递送客户文件、物品或货物的一种服务。在“门到门”的递送方式下,快递公司从发件人委托发件开始,以收发站为中心派车上门揽收快件;然后,将快件收集到快递中心,以航空运输的方式中转递运到收件人所在的收发站;最后,派车将快件递送到收件人手中。

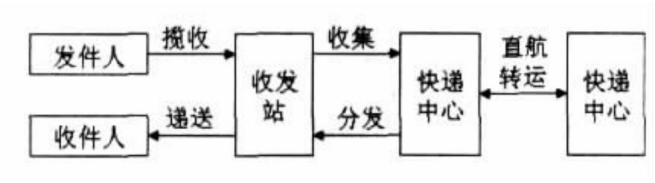

图 1：快递的一般业务流程

由于快件通常是样品、小件货物、文件、单据等小件物品,而快递公司所面对的是分散的众多社会客户,因此,快件的单位运输成本要远高于普通货物。快递公司调度车辆从发件人到收发站和从收发站到收件人的快递过程,关键决策在于

优化求解多点线路的车辆路径问题,即以收发站为中心,在满足调度资源的限制和各类时间约束的条件下,以路径成本最低和等待时间最短的目标编排行车路径来完成多点的快件揽收或递送。接下来,用数学模型对该过程中的车辆路径问题进行描述分析[1]。

综上，本文研究的快递配送优化问题为一类特殊的车辆路径问题,即考虑运输车辆，停车次数动态变化的车辆路径问题。

5.1.2 0-1 整数规划模型的建立

在问题一中，假设运输车辆是无限多的，即不需要担心运输车辆个数的限制，我们希望得到的是运输车辆的总停车次数的最小值，不需要考虑车辆总数，只要总的停车次数最少，便可作为最优解。

0-1 规划是决策变量仅取值 0 或 1 的一类特殊的整数规划。在处理经济管理中某些规划问题时，若决策变量采用 0-1 变量即逻辑变量，可把本来需要分别各种情况加以讨论的问题统一在一个问题中讨论。

在本文中，可以将车辆是否承担货物运输订单任务，车辆是否停车,车辆是否使用等多变量统一采用 0-1 逻辑变量继续统一，从而便于运输，增加求解的可能性。

问题 1 方案的选取，要保证每一个订单都被配送到目的地，也要使得运输车辆的总停车次数最少，同时还要考虑一辆车一次只能配送一个货物，我们建立 0-1 整数规划模型。

定义一（车辆订单矩阵）

为了表示车辆和订单之间的关系，我们假设第 i 辆车承担第 j 个订单，记为车辆订单矩阵，记为C_{ij}，即

$$C_{ij}=\begin{cases}1 & \text{承担}\\ 0 & \text{不承担}\end{cases} \tag{1}$$

（1）目标函数的确定

定义二（订单信息矩阵）

为了表示订单的起始站点和结束站点，我们假设第 j 个订单起始或终止于第 t 个站点，记为订单信息矩阵，记B_{jt}，即

$$B_{jt}=\begin{cases}1 & \text{起始或终止}\\ 0 & \text{其他}\end{cases} \tag{2}$$

定义三（车辆停车矩阵）

为了表示停车次数，我们假设第 i 辆车在第 t 个站点停车，记为站点停车矩阵，记为X_{ij}，即

$$X_{it}=\begin{cases}1 & \text{停车}\\ 0 & \text{不停车}\end{cases} \tag{3}$$

如果第 i 辆车承担第 j 个订单，同时第 t 个站点是第 j 个订单的初始站点或结束站点，那么第 i 辆车就要在 t 个站点停车，同时部分站点可能既是一个货物初始站点，也是另一个货物的结束站点，为了防止重复计算，我们要将X_{it}中大于 1 的数据变为 1，即与 0 进行异或。

$$X_{it}=\left(\sum_{j=1}^{m}C_{ij}\cdot B_{jt}\right)\oplus 0 \qquad (i=1,2,\dots,N,t=1,2,\dots,n) \tag{4}$$

记运输车数量为 N，订单数量为 m，要使得运输车辆的总停车次数尽可能少，则目标函数建立为：

$$min\ \ M=\sum_{i=1}^{N}\sum_{t=1}^{n}X_{it} \tag{5}$$

（2）约束条件的确定

定义四（订单运输矩阵）

为了表示订单所经过的站点，我们假设第 j 个订单运输中经过第 t 个和第 t+1 个站点，记为订单运输矩阵，记A_{jt}，即

$$A_{jt}=\begin{cases}1 & \text{经过}\\ 0 & \text{不经过}\end{cases} \tag{6}$$

定义五（车辆站点货物矩阵）

记运输车数量为 N，站点数量为 n,为了表示车辆在站点运输的货物量，我们假设第 i 辆车经过第 t 个站点时运输的货物量为Y_{it}，记为车辆站点货物矩阵，即

$$Y_{it}=\sum_{j=1}^{m}C_{ij}\cdot A_{jt} \qquad (i=1,2,\dots,N,t=1,2,\dots,n) \tag{7}$$

由于每辆车一次只能运送一个货物，故每辆车在每个站点中最多只能承担一个货物，即

$$0\le Y_{it}\le 1 \qquad (i=1,2,\dots,N,t=1,2,\dots,n) \tag{8}$$

同时每个订单只能被配送 1 次：

$$\sum_{j=1}^{m}C_{ij}=1 \qquad (i=1,2,\dots,N) \tag{9}$$

(3)建立单目标优化模型

$$min \quad M = \sum_{i=1}^{N} \sum_{t=1}^{n} X_{it} \tag{10}$$

$$s.t. \begin{cases} X_{it} = \left(\sum_{j=1}^{m} C_{ij} \cdot B_{jt} \right) \oplus 0 \quad (i = 1,2,\dots,N, t = 1,2,\dots,n) \\ Y_{it} = \sum_{j=1}^{m} C_{ij} \cdot A_{jt} \quad (i = 1,2,\dots,N, t = 1,2,\dots,n) \\ \sum_{j=1}^{m} C_{ij} = 1 \quad (i = 1,2,\dots,N, t = 1,2,\dots,n) \\ 0 \le Y_{it} \le 1 \quad (i = 1,2,\dots,N, t = 1,2,\dots,n) \\ C_{ij} = 0 \text{ 或 } 1 \quad (i = 1,2,\dots,N, j = 1,2,\dots m) \end{cases} \tag{11}$$

根据上述目标方程以及约束条件，可以建立 0-1 整数规划模型，进行对所有运输车辆的总停车次数的最小值求解。

5.1.3 模型的求解

完成一个运营周期内快件扡收的车辆路径问题,具有不确定性和动态性的显著特征。实际中,通常采用具有贪婪性特点的求解算法, 一方面寻求短期内达到最优, 其求解的时间复杂度经可能的少[2]。

另一方面则满足快速求解的要求在动态且不确定的决策环境中, 根据最优化模型公式给出的最小化目标, 对末揽收的快件进行当前约束下的路径寻优, 可快速确定揽收新快件 u 的车辆及其揽收规划。

给定车辆 k,由上述最优化公式描述的数学模型，该题目是优化停车次数的路线优化模型。它是 NP 难问题,需要在解的质量和计算时间之间的权衡考虑下选择合适的算法：

当站点以及快件不多时,可采用分支定界法、基于动态规划的优化方法等最优化算法，其计算复杂性通常很大，只适合于**求解小规模问题**，在工程中每每不实用[3]。

当站点以及快件很多时,可采用遗传算法、禁忌搜索法、模拟退货算好等启发式算法。这类算法虽然求解的不一定是全局最优解，但是时间复杂度较小，是考虑求解的复杂性和解的准确性权衡下最佳的选择[4]。

在第一问中，我们考虑使用贪心算法求解，当$C = 1$时，只需要车辆在放置货物时，将当前站点尚未运输的货物一并带上，继续运输，便可以保障该解为最优解，因此，我们使用如下算法进行求解。

按照贪心算法的思想，按改进的规则进行遍历，步骤如下：

Step1：选择第一辆车，从第一个需要运输货物的站点开始装货，当到达卸货站点时进行卸货。

Step2：当卸货完成后，如果有其他可以装载的货物，则选择装载，继续运行，到达卸货地点

Step3：重复步骤 2。当此辆车到达终点后，结束该辆车使用。

Step4：使用新的车辆从站点 1 重新进行货物运输，重复步骤 1~3

Step5：当所有货物都到达运输目的地时，结束运输流程。

Step6：计算车辆总数和停车总次数，得出最终结果。

经过 python 软件编程进行求解后，对题目中的四个实例进行计算，得到结果如下表:

实例	总停车次数
实例 1	82
实例 2	159
实例 3	162
实例 4	636

表 1：问题 1 总停车次数的最优解

5.2 问题二模型的建立与求解

5.2.1 模型的建立

由于一辆车能够同时运送两个货物，故每辆车在每个站点最多都可以运送两个货物，即

$$0 \le Y_{it} \le 2 \qquad (i = 1,2,\dots,N, t = 1,2,\dots,n) \tag{12}$$

建立单目标优化模型：

$$min \;\; M = \sum_{i=1}^{N}\sum_{t=1}^{n} X_{it} \tag{13}$$

$$s.t.\begin{cases} X_{it}=\left(\sum_{j=1}^{m}C_{ij}\cdot B_{jt}\right)\oplus 0 \quad (i=1,2,\dots,N,t=1,2,\dots,n) \\ Y_{it}=\sum_{j=1}^{m}C_{ij}\cdot A_{jt} \quad (i=1,2,\dots,N,t=1,2,\dots,n) \\ \sum_{j=1}^{m}C_{ij}=1 \quad (i=1,2,\dots,N,t=1,2,\dots,n) \\ 0\le Y_{it}\le 2 \quad (i=1,2,\dots,N,t=1,2,\dots,n) \\ C_{ij}=0 \text{ 或 } 1 \quad (i=1,2,\dots,N,j=1,2,\dots m) \end{cases} \tag{14}$$

5.2.1 模拟退火算法对模型的求解

与问题一相同，首先采用贪心算法进行局部最优解的求解，但由于 C 的取值有所改变，因此按照第一问当中的贪心策略所求得解偏离最优解的距离会很大，因此需要采用新的贪心策略，具体算法如下：

step1:选择第一辆车，从第一个站点开始

step2:每一辆车到达每一个站点，首先判断是否要卸货，若要卸货就将可以卸下的货全部卸下；然后判断是否有货可以装上车，若有可以装上的货就全部装上

step3:完成 step2 后，前往下一个站点，继续执行 step2，直到车上的货为零。

step4:选择下一辆车，继续执行 step2，直到全部任务完成。

但是，根据贪心算法所求得的解任然不为全局最优解，因此考虑采用优化进行进一步求解。由于实例 3、4 的数据量庞大，因此考虑采用启发式算法进行优化，在这里，我们使用模拟退火算法。

模拟退火算法最早的思想是由 N.Metropolis 等人于 1953 年提出[6]。它是基于迭代求解策略的一种随机寻优算法，其出发点是基于物理中固体物质的退火过程与一般组合优化问题之间的相似性。该算法的优点为求解速度快，可达到全局最优解。

模拟退火算法做为局部搜索算法的扩展，在每一次修改模型的过程当中，随机产生一个新的状态模型，而后以必定的几率选择邻域中能量值大的状态．这种接受新模型的方式使其成为一种全局最优算法，并获得理论证实和实际应用的验证．SA 虽然在寻优能力上无可置疑，但它是以严密的退火计划为保证的，具体地讲，就是足够高的初始温度、缓慢的退火速度、大量的迭代次数及同一温度下足够的扰动次数

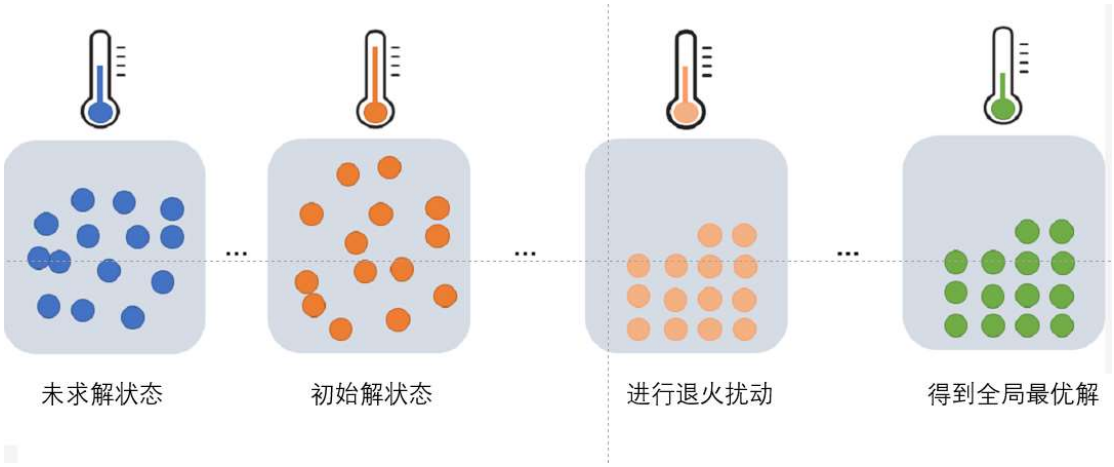

图 2:退火算法示意图[5]

模拟退火算法包括三函数两准则，即状态产生函数，状态接受函数，温度更新函数；内循环终止和外循环终止准则[6]。这些环节之间影响模拟退火算法的优化性能。

求解模拟退火算法描述如下：

(1)解空间。解空间 S 可表示为数据的所有分组形式的排列组合。

(2)目标函数。目标函数为总停车次数最少。要求

$$min \quad M = \sum_{i=1}^{N} \sum_{t=1}^{n} X_{it} \tag{15}$$

(3)新解的产生。利用上一步迭代分好的组，每次选取三个组，每组内各挑选一个数据进行交换，得到新的分组。

选组要求为：1.每次选取三个不同的组；2.三个小组两两之间隐匿的个数不能完全相同。

(4)代价函数差。假设原先的分组数据隐匿量为 T1，新的分组数据隐匿量为 T2。隐匿量差可表示为：

$$\Delta T = T2 - T1$$

(5)接受准则

$$P = \begin{cases} 1, \Delta T < 0 \\ exp(-\Delta T/T), \Delta T \geq 0 \end{cases} \tag{16}$$

如果$\Delta T < 0$，则接受新的分组；否则，以概率$exp(-\Delta T/T)$接受新的分组，即用计算机产生一个[0,1]区间上均匀分布的随机数 rand，若 rand <= $exp(-\Delta T/T)$则接受。

(6)降温。利用选定的降温系数 α 进行降温，取新的温度 T 为 AT(这里 T 为上一步迭代的温度)，这里选 α=0.999。

(7)结束条件。用选定的终止温度$e = 10^{-30}$，判断退火过程是否结束。若 T<e，则算发结束，输出当前状态。

其算法的步骤具体为：

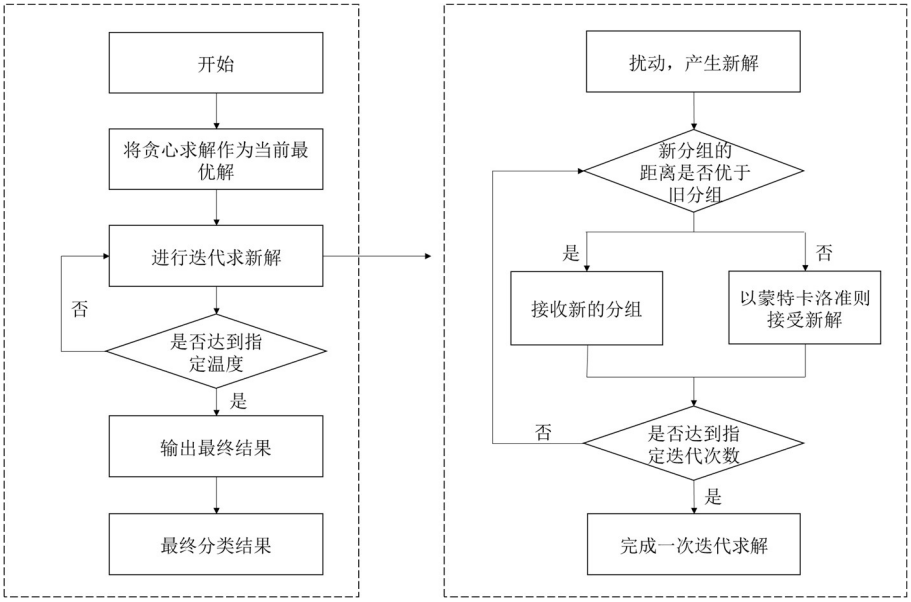

图 3： 模拟**退火算法流程图**

在利用模拟退火算法对该优化问题之前，我们先对该优化算法进行简述。模拟退火算法是--种基于蒙特卡洛迭代求解策略的随机寻优算法,出发点是基于固体物质的退火过程与一般组合优化问题之间的相似性。它是局部搜索算法的扩展，以一定的概率选择领域中目标值较大的状态，从而达到求解全局优化问题的目的。新解的迭代过程如下：

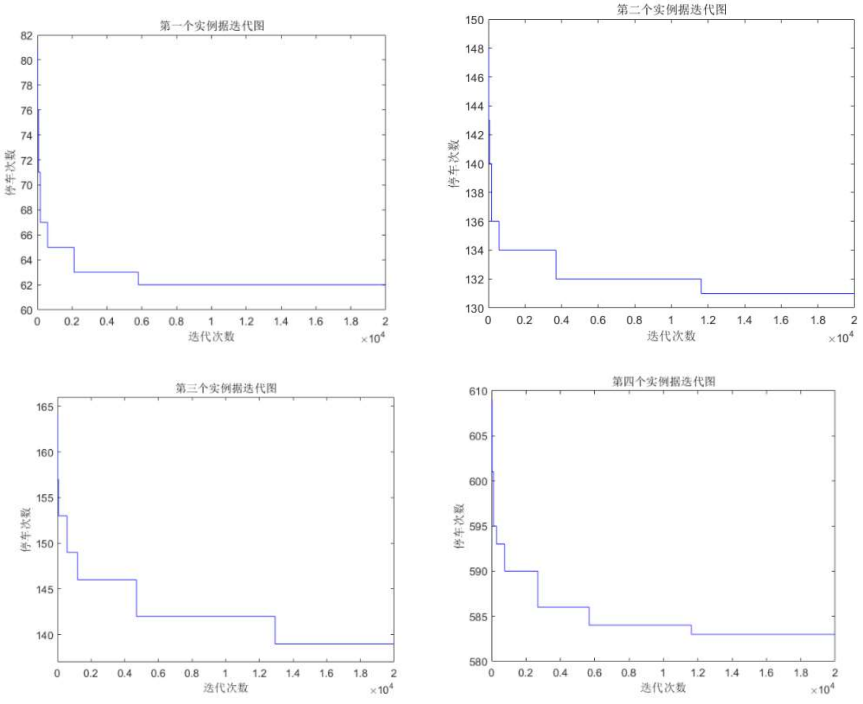

图 4：模拟退火算法新解的迭代图

经过 python 软件编程进行求解后，对题目中的四个实例进行计算，得到结果如下表:

实例	总停车次数
实例 1	62
实例 2	131
实例 3	139
实例 4	583

表 2：问题 2 总停车次数的最优解

需要指出的是:模拟退火算法中不同的参数取 值会引起不同的结果。在求解具体问题时，需通过 大量的数值模拟计算找出最佳的参数搭配，以求得 比较好的结果。

5.3 问题三模型的建立与求解

5.3.1 模型的建立

要求运送的车辆最少，停车次数没有限制，那么我们就可以将单目标优化的目标改为使运输车辆最小。由于有些车辆没有承担任何订单，我们只需计算承担货物的有效车辆最小，即

$$\sum_{i=1}^{N}\left(\sum_{j=1}^{m} C_{ij} \oplus 0\right)$$

考虑到车辆不能空载，因此需要修改上一问当中的约束条件，即每个车辆在每个车站只能携带 1 或 2 个货物：

$$1 \le Y_{it} \le 2 \quad (i = 1,2,\dots,N, t = 1,2,\dots,n)$$

因此，重新建立最优化模型为目标方程为：

$$min \ \sum_{i=1}^{N}\left(\sum_{j=1}^{m} C_{ij} \oplus 0\right)$$

约束条件为：

$$s.t.\begin{cases} Y_{it} = \sum_{j=1}^{m} C_{ij} \cdot A_{jt} \quad (i = 1,2,\dots,N, t = 1,2,\dots,n) \\ \sum_{j=1}^{m} C_{ij} = 1 \quad (i = 1,2,\dots,N, t = 1,2,\dots,n) \\ 1 \le Y_{it} \le 2 \quad (i = 1,2,\dots,N, t = 1,2,\dots,n) \\ C_{ij} = 0 \text{ 或 } 1 \quad (i = 1,2,\dots,N, j = 1,2,\dots m) \end{cases} \tag{17}$$

5.3.2 模型的求解

经过 python 软件编程进行求解后，对题目中的四个实例进行计算，得到结果如下表:

实例	所需车辆
实例 1	16
实例 2	28
实例 3	29
实例 4	104

表 3：问题 3 总停车次数的最优解

5.4 问题四模型的建立与求解

5.2.1 模型的建立

在问题四中，不仅需要考虑前几问中所有运输车辆的总停车次数最小化，还需要考虑所使用运输车辆总个数最小化。因此，需要建立双目标优化模型进行求解。

继续沿用上一问的约束条件，使用极小化所使用运输车辆总个数和极小化所有运输车辆的总停车次数的两个目标为最优化目标函数，建立双目标最优化模型：

$$\begin{cases} min \quad = M \\ min \quad \sum_{i=1}^{N}\left(\sum_{j=1}^{m} C_{ij} \oplus 0\right) \end{cases} \tag{17}$$

$$\boldsymbol{s.t.}\begin{cases} X_{it} = \left(\sum_{j=1}^{m} C_{ij} \cdot B_{jt}\right) \oplus 0 \quad (i = 1,2,\dots,N, t = 1,2,\dots,n) \\ Y_{it} = \sum_{j=1}^{m} C_{ij} \cdot A_{jt} \quad (i = 1,2,\dots,N, t = 1,2,\dots,n) \\ M = \sum_{i=1}^{N}\sum_{t=1}^{n} X_{it} \\ \sum_{j=1}^{m} C_{ij} = 1 \quad (i = 1,2,\dots,N, t = 1,2,\dots,n) \\ 0 < Y_{it} \le 2 \quad (i = 1,2,\dots,N, t = 1,2,\dots,n) \\ C_{ij} = 0 \text{ 或 } 1 \quad (i = 1,2,\dots,N, j = 1,2,\dots m) \end{cases} \tag{18}$$

5.2.2 线性加权和法转化为单目标优化的求解

在具有多个指标的优化问题中，可以给予那些相对重要的指标较大的权系数，于是可将多目标优化问题转化为所有目标加权求和的单目标优化问题。

因此，我们使用线性加权和法。其基本思想是: 按照多目标 $f_i(x)(i = 1,2,\cdots,m)$ 的重要程度, 分别乘以一组权系数 $\lambda_j(j = 1,2,\cdots,m)$ 然后相加作为目标函数而构成单目标规划问题[6]。即

$$m\ f = \sum_{j=1}^{m} \lambda_j f_j(x), \text{其中 } \lambda_j \geq 0 \text{ 且 } \sum_{j=1}^{m} \lambda_j = 1$$

为了求解总停车次数和使用运输车辆总个数在最优化模型的权重，我们引入时间损耗的概念，我们认为在每辆车发车时，停车运送货物时，都会造成时间损耗，根据相关资料的查阅，考虑每辆车发车时的时间存好为 51 分钟，停车运送货物的时间损耗为 9 分钟。

故在本题中对于总车辆数$\lambda_1 = 0.85$，对于车辆停车次数的权重$\lambda_2 = 0.15$，即优化目标可以改为：

$$min\ \ \lambda_1 \cdot \sum_{i=1}^{N} \left(\sum_{j=1}^{m} C_{ij} \oplus 0 \right) + \lambda_2 \cdot M$$

经过 python 软件编程进行求解后，对题目中的四个实例进行计算，得到结果如下表:

实例	所需车辆	总停车次数
实例 1	16	62
实例 2	28	131
实例 3	29	139
实例 4	104	583

表 4：问题 4 总停车次数的最优解

六 模型评价与推广

6.1 模型的优点分析

1) 本文针对配送路径优化问题通过 0-1 整数规划方法建立贪心模型来进行求解，并通过实例计算得出了比较优的结果。

2) 在实例的计算中，具有较小的时间复杂度和空间复杂度，综合考虑了解的准确性和求解过程的复杂性。
3) 使用启发式算法当中的模拟退火算法进行求解，使得求解为全局最优解的可能性大大增加，优化解的求解过程。
4) 在问题 4 中，将双目标最优化模型转化为了单目标最优化模型，使得模型的求解更可能实现。

6.2 模型的缺点分析

1) 当实例较少时，该模型可以效率较高的得出最优解，但随着送货地点数目的增加，用此方法得到的则不一定是最优解，同时计算量也相当大。
2) 在第四问中，在使用线性加权和法进行权重的确定时，虽然进行了相关资料和文献的查询，但是仍然具有一定的主观性和不确定性，需要进行后续的改进。

6.3 模型的改进与推广

可以考虑采用蚁群算法或者智能粒子群算法求解，以期望获得更小的时间复杂度，此外，还可以查找资料确定更加的权重确定算法，以此来获得更好的线性加权，从而改进解空间。

该模型可以很好的用于不同站点之间的车辆优化和配送优化问题，有助于快递和货物的实际配送。该模型的实用性很强，它可以很好地用于中小城市城内的对于送货地点不多的情况，同时通过对该模型的延伸，还可以实现多个城市之间的物流配送路径的优化[7]。

参考文献

[1] 姜大立, 杨西龙, 杜文, 等. 车辆路径问题的遗传算法研究[J]. 系统工程理论与实践, 1999, 19(6): 40-45.

[2] 孙丽君, 胡祥培, 王征. 车辆路径规划问题及其求解方法研究进展[J]. 系统工程, 2006, 24(11): 31-37.

[3] XING Wen-xun, XIE Jin-xing.Modern Optimization Algorithm[M].Beijing: Tsinghua University Press, 2003.

[4] 祝崇隽, 刘 民, 吴 澄.供应链中车辆路径问题的研究进展及前景[J].计算机集成制造系统, 2001, 7(11): 1-6.

[5] POTVIN J Y, ROUSSEAU J M.An Exchange Heuristic for Routing Problems with Time Windows[J].Journal of the Operational Research Society, 1995, 46(12): 1 433-1 446

[6] 胡大伟, 朱志强, 胡勇. 车辆路径问题的模拟退火算法[J]. 中国公路学报, 2006, 19(4): 123.

[7] 田昀, 梁海龙. 基于模拟退火算法的物流配送路径优化问题分析[J]. 交通科技与经济, 2013, 15(5): 85-88.

附录

```
1. import pandas as pd
2. import random
3. import copy
4. import math
5. import csv
6.
7. C = 2
8. stationlen = 40
9. w1 = 0.85
10.w2 = 0.15
11.
12.a = 0.99
13.T = 1
14.targetmin = 0
15.
16.data = []
17.dataA = []
18.dataB = []
19.datastart = []
20.dataend = []
21.dataC = []
22.dataX = []
23.datalen = 0
24.carlen = 0
25.
26.tempdataA=[]
27.
28.def readData():
29.    global data
30.    global datalen
31.    global dataA
32.    global dataB
33.    global dataC
34.    global datastart
35.    global dataend
36.    global carlen
37.
38.    df = pd.read_excel(r'D:\Desktop\one.xls', sheet_name='实
   例 2')
39.    for i in df.values:
40.        data.append(i.tolist())
```

```
41.     datalen = len(data)
42.
43.     dataB = [[0 for col in range(stationlen)] for row in ra
    nge(datalen)]
44.     for i in range(datalen):
45.         dataB[i][data[i][0]] = 1
46.         dataB[i][data[i][1]] = 1
47.
48.     dataA = [[0 for col in range(stationlen)] for row in ra
    nge(datalen)]
49.     for i in range(datalen):
50.         j = data[i][0]
51.         while j < data[i][1] :
52.             dataA[i][j] = 1
53.             j += 1
54.
55.
56.     carlen = round(datalen / C)
57.
58.     datastart = [[] for i in range(stationlen)]
59.     dataend = [[] for i in range(stationlen)]
60.     for i in range(datalen):
61.         datastart[data[i][0]].append(i)
62.         dataend[data[i][1]].append(i)
63.
64.     dataC = [[0 for j in range(datalen)] for i in range(car
    len)]
65.
66. def greedy_algorithm():
67.     result = []
68.     tempdatastart = copy.deepcopy(datastart)
69.     for i in range(carlen):
70.         # c = 0
71.         list1 = []
72.         list2 = []
73.         key = 0
74.         for t in range(stationlen):
75.
76.             if key == 1 and len(list1) == 0:
77.                 break
78.
79.             for j in list1:
80.                 if dataB[j][t] == 1:
81.                     list1.remove(j)
```

```
82.
83.             while len(list1) < C:
84.                 le = len(tempdatastart[t])
85.                 if le != 0:
86.                     r = random.randint(0, len(tempdatastart
   [t]) - 1)
87.                     list1.append(tempdatastart[t][r])
88.                     list2.append(tempdatastart[t][r])
89.                     tempdatastart[t].pop(r)
90.                     key = 1
91.                 else:
92.                     break
93.
94.         result.append(list2)
95.     return result
96.
97.def write():
98.     from xlutils.copy import copy
99.     import xlrd as xr
100.
101.      file = "D:\\Desktop\\MatrixA.xls"
102.      oldwb = xr.open_workbook(file)
103.      newwb = copy(oldwb)
104.      newws = newwb.get_sheet(-1 + 4)
105.      for j in range(datalen):
106.          for t in range(stationlen):
107.              newws.write(j + 1, t + 1, dataA[j][t])  #  行，
   列，数据
108.      newwb.save(file)
109.
110.      file = "D:\\Desktop\\MatrixB.xls"
111.      oldwb = xr.open_workbook(file)
112.      newwb = copy(oldwb)
113.      newws = newwb.get_sheet(-1 + 4)
114.      for j in range(datalen):
115.          for t in range(stationlen):
116.              newws.write(j + 1, t + 1, dataB[j][t])  #  行，
   列，数据
117.      newwb.save(file)
118.
119.  def getMatrixC(list):
120.      result = [[0 for j in range(datalen)] for i in range(
   carlen)]
121.      for i in range(carlen):
```

```
122.             for j in list[i]:
123.                 result[i][j] = 1
124.         return result
125.
126.     def getMatriX(list):
127.         result = [[0 for t in range(stationlen) ] for i in ra
    nge(carlen)]
128.         for i in range(carlen):
129.             for t in range(stationlen):
130.                 s = sum([list[i][j] * dataB[j][t] for j in ra
    nge(datalen)])
131.                 # print(s)
132.                 if s != 0:
133.                     result[i][t] = 1
134.         return result
135.
136.     def ifMatriY(list):
137.         key = 1
138.         for i in range(carlen):
139.             for t in range(stationlen):
140.                 s = sum([list[i][j] * dataA[j][t] for j in ra
    nge(datalen)])
141.                 if s > C:
142.                     key = 0
143.                     break
144.             if key == 0:
145.                 break
146.         return key
147.
148.     def sumMatriX(list):
149.         num = 0
150.         for i in list:
151.             num += sum(i)
152.             # print(sum(i))
153.         return num
154.
155.     def sumCarnumber(list):
156.         num = 0
157.         for i in list:
158.             s = sum(i)
159.             if s != 0:
160.                 num += 1
161.         return num
162.
```

```
163. def acceptrule(df):
164.     if df < 0:
165.         return 1
166.     else:
167.         r = random.randint(0,1000000000000)/1000000000000
168.         if r < math.exp(-1*df/T):
169.             return 1
170.         else :
171.             return 0
172.
173. def writescv(x):
174.     with open("D:\Desktop\MatrixC.csv", "w") as csvfile:
175.         writer = csv.writer(csvfile)
176.         writer.writerows(dataC)
177.     with open("D:\Desktop\MatrixX.csv", "w") as csvfile:
178.         writer = csv.writer(csvfile)
179.         writer.writerows(x)
180.
181.
182.
183. if __name__ == "__main__":
184.
185.     readData()
186.
187.     list = greedy_algorithm()
188.     dataC = getMatrixC(list)
189.     x = getMatriX(dataC)
190.     sC = sumCarnumber(x)
191.     sP = sumMatriX(x)
192.     targetmin = w1 * sC+ w2 * sP
193.     # print(list)
194.     # print(dataC)
195.     # print(x)
196.     writescv(x)
197.     print(targetmin,sC,sP)
198.     while T > math.pow(10,-90):
199.         templist = greedy_algorithm()
200.         tempdataC = getMatrixC(templist)
201.         tempx = getMatriX(tempdataC)
202.         tempsC = sumCarnumber(tempx)
203.         tempsP = sumMatriX(tempx)
204.         temptargetmin = w1 * tempsC+ w2 * tempsP
205.         # print(temptargetmin,targetmin)
206.         if acceptrule(temptargetmin - targetmin) == 1:
```

```
207.                 list = copy.deepcopy(templist)
208.                 dataC = copy.deepcopy(tempdataC)
209.                 x = tempx
210.                 targetmin = temptargetmin
211.                 sC = tempsC
212.                 sP = tempsP
213.                 # print(list)
214.                 # print(dataC)
215.                 # print(x)
216.                 writescv(x)
217.                 print(targetmin,sC,sP)
218.             T = a * T
```