

汽车组装车间流水线物料配送问题

摘要

在充分理解题意以及合理假设的基础上，通过对问题的深入分析，我们建立了相关优化目标和非线性规划模型解决问题，并利用 Python 编程实现并进行计算。

针对问题一：通过对最优距离与最优时间的关系进行分析，建立了相应的优化目标及相关约束，对各个工位与其对应存储点的距离关系进行统计分析，设计了工位分配算法，使用 python 编程计算，求得的最优配送方案中的最长路径的距离为 1066 米。

针对问题二：为了在减少拖车数量的情况下尽可能高效，我们建立了相应的优化目标，同时为了使得停线风险尽可能小，我们在优化目标中设置了关于超时与早到的惩罚系数，并根据存储点与工位对应的关系、不同拖车路径工位之间的关系设置了相关约束。基于软时间窗的单向配送车辆路径问题的建模思想，对配送问题进行取货与送货的拆分，建立了非线性规划模型。我们使用遗传算法对模型进行求解，首先根据需求列表中的工位点建立初始种群，然后使用子路径交换交叉、基因换位算子、基因突变算子生成新的方案，最后使用最优保存选择策略更新当前种群，从而对任务进行优化。我们分别针对附件 2 中任务 1、任务 2 和任务 3 生成了最优配送方案，并对方案中部分路径进行了分析。

针对问题三：问题三是在问题二方法的基础上，对时间进行了严格的限制。因此，我们很自然地想到了将第二问的软时间窗的单向配送车辆路径问题改为硬时间窗的单向配送车辆路径问题。目标函数去掉了小车提前到达和延后到达的惩罚系数，约束条件大部分不变，只增加一项对送达时间的严格约束。但是通过实验分析我们发现，我们所采集的样本并不能满足新增的关于时间需求的约束，基于此我们提出了对于采样的改进思路，设想了多次巡回的数学模型。

关键词：物流配送管理 带时间窗的单向配送车辆路径问题 软时间窗 硬时间窗 遗传算法

1 问题重述

汽车制造有四大关键工艺——冲压、焊接、涂装和总装，其中总装车间的占地面积最大、工人数量最多，迫切需要数学建模技术帮助降本增效。总装是指将发动机等全部内外饰件装配到车身上的工艺，通常围绕流水线展开，线上的每个工位负责装配固定的物料。为防止停线影响生产效率，需要及时补充工位上的物料，此物料补充工作称为运输，一般由拖车完成。拖车接收到任务指令后，需要进行如下操作：1) 取料：前往目标物料的存储位，将物料装载至拖车；2) 配送：将物料运送至目标工位和卸载。需要注意的是，在不超过小车容量的情况下，多个任务可以同时进行。

问题一：某总装车间内有两条并行的流水线，其布局如图 1 所示，其道路关键节点和工作点见附件 1。该车间共有 10 辆拖车，目前采用承包制，即一个拖车负责若干个工位（称为“承包区”）的配送任务，且承包区互不交叠。请根据上述信息为拖车安排承包区（每个承包区不超过 5 个工位）。

问题二：为了降本增效，该车间计划通过打破承包制，减少拖车数量。在某一时间点，系统收到任务需求和当前可用车辆，通过计算生成派送方案并下发至拖车执行。请设计一种方法，在尽量减少停线风险的前提下，使得派送方案可以自动生成。图 2 和图 3 所示为系统收到需求信息示例和生成的派送方案。假设小车行进速度为 5 米/秒，装/卸零件耗时均为 3 秒，一次最大可装载 4 个任务。附件 2 为若干用于验证的任务数据，亦鼓励参赛者设计更复杂的场景和数据验证其技术方案。

问题三：问题 2 中的方案能否杜绝停线情况？若能，请解释原因；若不能，请改进或设计一种新的机制，并验证其效果。

2 条件假设

1. 运输拖车车型一致，车辆的负载能力一定。
2. 道路的宽度足够，拖车之间不会发生碰撞。
3. 仓库到工位，以及工位之间的拖车行驶时间只与拖车的速度有关。

3 符号说明

符号	含义
----	----

t_{a1}	拖车装载时间
t_{a2}	拖车卸货时间
v	拖车速度
T	各拖车所消耗总时间的集合
T_i	各拖车所消耗的总时间
P_i	拖车 <i>i</i> 负责工位的集合
p_j	工位点
r_j	存储点(位)
Z	拖车运完存储点所需的总次数
$ P_i $	P_i 集合中的元素数目

4 问题分析

4.1 问题一分析

问题一中未进行成本的限制，因此我们可以在不计成本的情况下让效率最大化。对于拖车数目，基于假设2的条件，毫无疑问在拖车数目取最大时效率是最高的；而优化效率很自然地可以转换为对运输时间的优化，也就是说我们得到最短的运输时间即可获取得到最高的运输效率。

4.2 问题二分析

问题二的目的是控制成本，因此对拖车数目进行了限制，即将拖车数目变成一个输入变量进行控制。减小停线风险意味着我们要尽可能在需求时间结束前将零件送到相应工位上，因此我们可以在这里设置一个超时惩罚系数；而早到的话要在工位等待到需求时间，因此我们可以设置一个等待惩罚系数。

4.3 问题三分析

问题三是基于问题二方案的基础上，将“减小停线风险”这一条件改为“杜绝停线风险”。在问题二中，我们引入了惩罚系数，这可以使得停线风险尽可能小，但是由于我们的目标函数是使配送总距离，也就是各条配送路径的总长度之和最短，所以在进行计算时，为了使目标函数有最优解，仍会有拖车在需求时间之后送达的可能性。因此，我们需要对问题二的模型进行改进，完全杜绝停线状况。

5 问题一模型的建立与求解

5.1 模型的建立

给定拖车装载时间 t_{a1} ，拖车卸货时间 t_{a2} ，拖车速度为 v ，各拖车所消耗总时

间的集合 $T = \{T_1, T_2, \dots, T_{10}\}$, 所有工作均完成所需的时间可以写为:

$$\min_P \max\{T_i\} \quad (5.1)$$

$$s.t. \quad id(p_j) = id(r_j)$$

其中 $T_i \in T$ 。记拖车 i 负责工位的集合为 P_i , $p_j \in P_i$, $0 < j \leq 5$, p_j 所对应的存储点为 r_j , 假定存储点货物有限且各个存储点的货量相等, 拖车 i 运输完成所需的总时间 T_i 可以表示为:

$$T_i = \left[\frac{\sum_1^{|P_i|} dist(p_j, r_j)}{v} + (t_{a1} + t_{a2})|P_i| \right] \cdot Z \quad (5.2)$$

其中, $|P_i|$ 为 P_i 集合中的元素数目, $dist(p_j, r_j)$ 输出为 p_j 与 r_j 两点间的路径距离, $id(\cdot)$ 为节点序号如存储点 III-2 的节点序号为 2, 工位 II-1 的节点序号为 1, Z 为拖车运完存储点所需的总次数, v 为拖车行进速度。

我们接下来对公式(2)进行变换:

$$T_i = \frac{Z}{v} \cdot \sum_1^{|P_i|} dist(p_j, r_j) + (t_1 + t_2) \cdot Z \cdot \sum_1^{|P_i|} |P_i| \quad (5.3)$$

不难发现, $(t_1 + t_2) \cdot Z \cdot \sum_1^{|P_i|} |P_i|$ 为一个常数项, $\frac{Z}{v}$ 同样为常数, 由于对 T_i 进行缩放并不会影响最终的求解结果, 我们可以将公式(3)变换为:

$$T_i = \sum_1^{|P_i|} dist(p_j, r_j) \quad (5.4)$$

从公式(3)我们可以看出, 对时间 T_i 的优化实际上就是对行进距离的优化。

5.2 模型的求解

由于这里对目标函数的求解我们将 $|P_i|$ 设置为 5, 再根据约束条件将所有可能的承包方案都代入来计算消耗时间, 从而获取满足目标方程的解。但是在实际进行求解时我们发现, 如果使用暴力求解的方法对所有可能的结果进行逐次遍历, 将消耗大量算力, 浪费大量时间, 因此我们提出以下方案来进行求解的优化。

由于总共的拖车数目为 10, 而两个装配区的工位数目总共为 48, 因此一定有一台或者两台拖车所负责的工位数目不足 5, 我们将这类拖车称为**二类拖车**, 其他拖车称为**一类拖车**。因为这小拖车所负责的工位数目较少, 所以小拖车应当承担与对应存储点距离更大的工位。

5.2.1 算法介绍

对于各拖车所消耗的总时间 $T = \{T_1, T_2, \dots, T_{10}\}$, 令

$$T_{avg} = \frac{1}{10} \sum_i^{10} T_i \quad (5.5)$$

对于最终的优化结果 T_{best} , 一定有 $T_{best} \geq T_{avg}$, 也就是说我们最优解的结果越接近 T_{avg} 则结果越佳。对下图 1 为对数据进行统计后的分布图, 其中横坐标为工位与其对应存储点的距离。

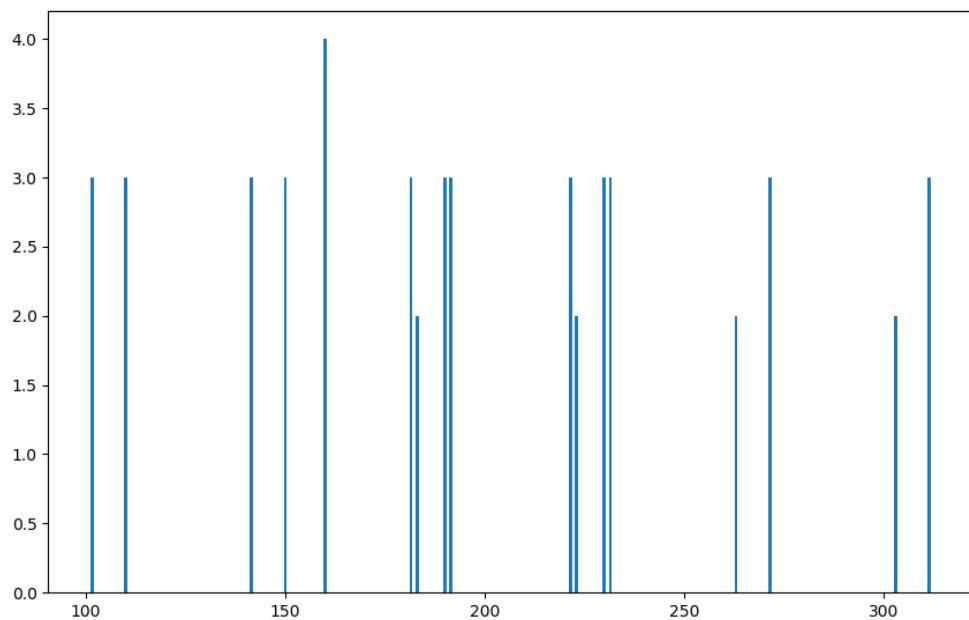


图 1 数据分布图

通过观察图中数据分布, 我们发现距离的分布比较均匀, 我们根据这一结果并结合取平均的思想设计了如下算法来进行求解。

算法流程图如下图 2 所示, 首先计算出存储位到对应工位的距离。然后按照计算出的距离进行从小到大排序。一开始设置两个指针 `left` 和 `right`, 分别指向最小的距离和最大的距离。每次分别从左指针和右指针所指向的位置各拿一个元素放到第 i 辆车的任务列表中, 并且左右指针各自向中间移动, 直到左右指针在中间相遇, 说明所有任务均已分配给对应的车。

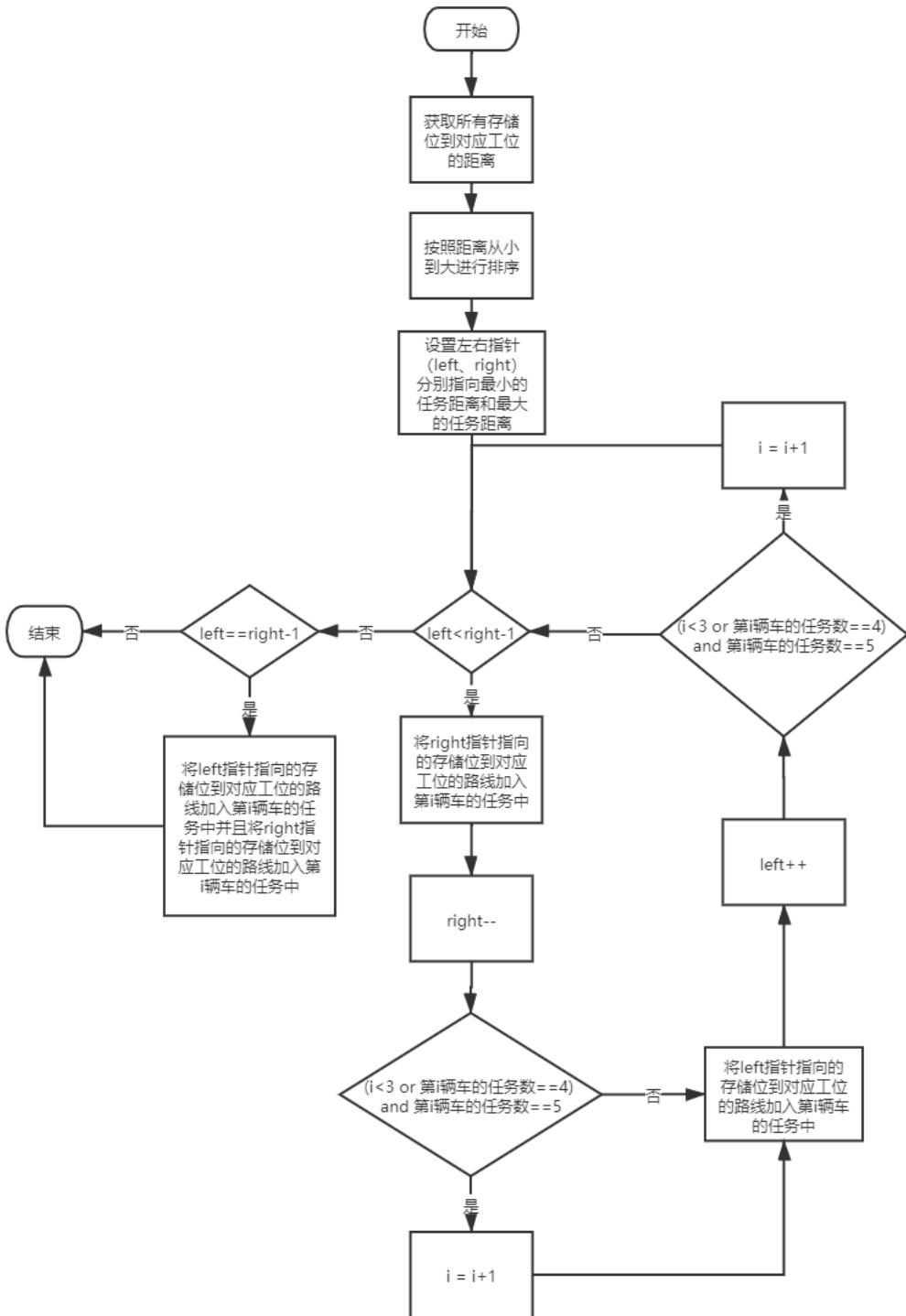


图 2 求解最优路径的算法流程图

5.2.2 求解结果

我们根据拖车的分配情况，分别计算以下两种情况的解：

- (1) 一类拖车的数量为 9，二类拖车的数量为 1；
- (2) 一类拖车的数量为 8，二类拖车的数量为 2。

计算后得到的最优配送方案中的最长路径的距离分别为 1137 米和 1066 米，为使效率最大，我们选择第二种分配情况，即一类拖车的数量为 8，二类拖车的数量为 2 的情况。具体承包方案如下表 1 所示。

表 1 拖车的承包情况表（完整结果见附件）

拖车编号	配送物料	存储点位	送达工位	配送距离 (米)	每辆拖车的总行驶距离 (米)
拖车 1	1	III24	I 24	311.5	
	2	III4	I 4	101.5	825
	3	III16	I 16	311.5	
	4	III12	I 12	101.5	
拖车 2	1	III8	I 8	311.5	
	2	III20	I 20	101.5	826
	3	III17	II 17	303	
	4	III5	II 5	110	
拖车 3	1	III9	II 9	303	
	2	III13	II 13	110	
	3	III23	I 23	271.5	1066
	4	III21	II 21	110	
	5	III15	I 15	271.5	
...					
拖车 10	1	III7	II 7	190	
	2	III13	I 13	191.5	
	3	III15	II 15	190	953
	4	III23	II 23	190	
	5	III5	I 5	191.5	

由表可以看出，拖车 1 和拖车 2 为二类拖车，拖车 3-10 为一类拖车，每辆拖车出发后先到达存储点位再到达配送点位，车上每件物料的配送距离之和为总行驶距离，最大的总行驶距离代表了整个配送方案的行驶距离。

6 问题二模型的建立与求解

6.1 动态配送优化调度问题的分析

经分析我们确定此问为带时间约束条件的有限单向车辆路径问题，此问题需要我们在制定配送路线时，不仅考虑工位的物料需求（或供应）数量约束和配送车辆一次配送的最大行驶距离约束，而且要考虑工位（或仓库）对物料送到（或取走）的需求时间。

本问属于软时间窗有限单向车辆路径问题，即工位要求将物料尽量在时间窗内送到，但也允许提前或拖后，只不过提前或拖后时，要实施一定的惩罚。在本问中，要求尽量减少停线风险，即为软时间窗有限单向车辆路径问题。

问题二的方案可以描述为：从 B 点用若干台拖车，根据多个工位的需求，先前往工位对应仓库点位取货，再向多个工位送货。每个工位的位置和需求量一定，要求将货物送达的需求时间一定，每台配送车辆的载重量一定，其一次配送的最大行驶距离一定，要求合理安排车辆配送路线和车辆行车时间，使目标函数得到优化。这里的优化目标有两个：一是要使配送的总距离最短，二是尽量减少停线风险，即尽量在需求时间之前送达，也就是说使拖车因未能将货物在需求时间内送到而遭受的惩罚最小，同时，还要满足以下约束条件：

- (1) 每条配送路径上各工位的物料需求量之和不超过配送拖车的载重量；
- (2) 每个工位的需求必须满足，且一次只能由一台配送拖车送货。
- (3) 车辆将货物送到工位后将返回配送中心，并可以多次巡回使用。

6.2 模型的建立

设 B 点有 K 台配送车辆，车辆记为 k ，由题意，小车一次最大可装载 4 个任务，则其载重量为 4，需要向 L 个工位（库房）送货（取货），注意这 L 个工位中可能存在重复的工位，且要求货物在 b_i 这一时刻运到，工位 i 到 j 的运动距离为 d_{ij} ，从 B 点到各工位的距离为 d_{0j} ($i, j = 1, 2, \dots, L$)；设 s_i 表示车辆到达工位（库房） i 的时刻， t_i 表示车辆在工位 i 的等待时间， t_{ij} 表示配送车辆由工位 i 行驶到工位 j 的旅行时间， t_a 表示装/卸物料的时间；我们将问题分解为从存储点取货和往工位送货两部分来处理，如图 3 所示：

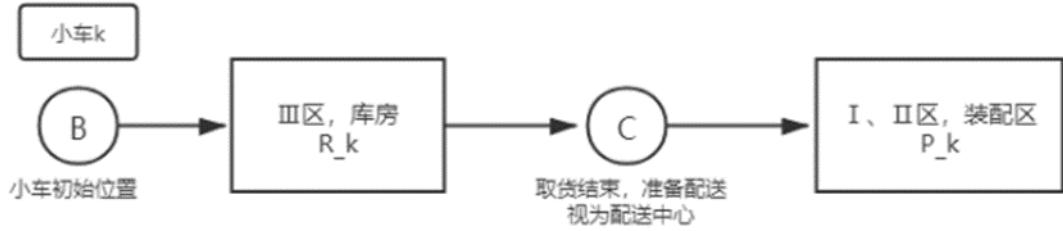


图 3 小车配送过程示意图

图 3 中小车在初始时间均位于 B 点，收到某一工位的需求后，首先要从 B 点进入 III 区的库房内取货，由于所有小车在取到货物之后都必会经过 C 点向 I 区和 II 区运货，因此我们将 C 点视为配送中心，即所有取到物料的小车开始送货的起点。

设 n_k 为第 k 辆车配送的工位数（工位数应当与库房数相等）， $n_k = 0$ 表示未使用第 k 台小车，用集合 R_k 表示第 k 条路径即第 k 辆拖车的行驶路径上所有的库房，其中的元素 r_{ki} 表示库房， r_{ki} 在路径 k 中的顺序为 i (r_{kl} 表示在路径 k 中顺序为 i 的库房)，令 r_{k0} 表示 B 点， $s_0 = 0$ 表示配送车辆从 B 点出发的时刻为 0；用集合 P_k 表示第 k 辆拖车的行驶路径上所有的工位，其中的元素 p_{ki} 表示工位， p_{ki} 表示在路径 k 中顺序为 i 的工位，令 p_{k0} 表示 C 点；取目标函数为配送总里程最短。

在取货过程中，拖车从 B 点（即 r_{k0} ）出发，取货后最终到达 C 点（即 p_{k0} ），令第 k 辆拖车取货的路程为 $D_k^{(R)}$ ，我们给定其表达式如下：

$$D_k^{(R)} = \sum_{i=1}^{n_k} d_{r_{k(i-1)} r_{ki}} + d_{r_{kn_k} p_{k0}} \cdot sign(n_k) \quad (6.1)$$

其中 C 代表点 C， $sign(n_k)$ 的取值为 0 或 1，若该拖车参加了配送任务，则取 $sign(n_k) = 1$ ，若当第 k 辆车服务的工位数小于 1 时，表示未使用该台车辆，因此取 $sign(n_k) = 0$ ；在送货过程中，拖车从 C 点（即 p_{k0} ）出发，将车上的物料送往指定工位后，最后回到 C 点（即 p_{k0} ）。令所有拖车运送物料的路程累加和为 D_P ，我们给定其表达式如下：

$$D_k^{(P)} = \sum_{i=1}^{n_k} d_{p_{k(i-1)} p_{ki}} + d_{p_{kn_k} p_{k0}} \cdot sign(n_k) \quad (6.2)$$

在该问题中，我们还需要规划配送拖车的行车时间，即拖车何时从 B 点出发，何时到达工位，在工位处是否需要等待，需要等待多长时间，何时从前一工位离开前往下一工位等等。由于工位存储空间有限，拖车必须在零件消耗完的时刻补充零件，故即使拖车提前到达工位，它开始卸载零件的时刻亦为该任务的需求时间。也就是说，车辆如果在需求时间之前到达工位，它会在客户处等待，一直等

到需求时刻再进卸货。针对以上这些问题，我们设置约束条件来逐一进行解决。

对于每条配送路径，经过的工位（存储点）数不超过需求列表中总工位（存储点）数，即

$$0 \leq n_k \leq L \quad k = 1, 2, \dots, K \quad (6.3)$$

对于需求列表中的所有需求都应该得到配送服务：

$$\sum_{k=1}^K n_k = L \quad (6.4)$$

需求列表中的配送需求不应该被重复配送，因此对于每条配送路径所包含的需求，取货过程中各个拖车负责的需求的集合交集为空，即：

$$M_i \cap M_j = \emptyset \quad \forall i \neq j \quad (6.5)$$

其中 M_i 和 M_j 分别表示第 i 条路径与第 j 条路径中所需要完成的配送需求集合。

在配送过程中，第 k 辆拖车到达工位 p_{ki} 的时刻 = 拖车到达工位 $p_{k(i-1)}$ 的时刻 + 拖车在 $p_{k(i-1)}$ 处的等待卸货时间 + 拖车在 $p_{k(i-1)}$ 的卸货时间 + 从 $p_{k(i-1)}$ 到 p_{ki} 所需要的时间，关系表达式为：

$$s_{p_{k(i-1)}} + t_{p_{k(i-1)}} + t_a + t_{p_{k(i-1)}p_{ki}} = s_{p_{ki}} \quad i = 1, 2, \dots, n_k \quad (6.6)$$

对于等待时间 t_i ，我们可以写出其表达式：

$$t_i = \max\{b_i - s_i, 0\} \quad i = 1, 2, \dots, L \quad (6.7)$$

最后，在配送过程所包含的工位序号，取货过程中一定要有与之对应序号的存储点存在，也就是说工位的序号需要与存储点的序号一一对应，如存储点 III-2 的序号为 2，工位 II-2 的序号也为 2，III-2 和 II-2 就是对应的，

$$\exists r_{ki} \in R_k, p_{ki} \in P_k, s.t. id(r_{ki}) = id(p_{ki}) \quad (6.8)$$

其中 $id(\cdot)$ 表示该工位（或存储点）的序号。

根据以上分析，我们可以建模：

$$\min_{R, P} \{w_1 \cdot \sum_{k=1}^K [D_k^{(R)} + D_k^{(P)}] + w_2 \cdot \sum_{j=1}^K \max[(b_j - s_j), 0] + w_3 \cdot \sum_{j=1}^L \max[(s_j - b_j), 0]\} \quad (6.9)$$

$$s.t. \quad 0 \leq n_k \leq L \quad k = 1, 2, \dots, K$$

$$\sum_{k=1}^K n_k = L$$

$$M_i \cap M_j = \emptyset \quad \forall i \neq j$$

$$\exists r_{ki} \in R_k, p_{ki} \in P_k, s.t. id(r_{ki}) = id(p_{ki})$$

式(6.9)所述模型中,目标函数要求各拖车取货和配送的总距离最短,由于最小化总距离和总时间其实是等价的(详见5.1),因此该式也可理解为要求取货和配送所花总时间最短。模型中 w_1 为配送总路程的权重, w_2 为拖车早于需求时间将物料卸下的累计时间的惩罚权重, w_3 为车辆晚于需求时间将物料送到的累计时间的惩罚权重, w_1 、 w_2 、 w_3 的取值与对上述各目标的重视程度有关, w_3 直接关系到停线风险,为了尽可能减小停线风险,我们设定提前的惩罚系数小于拖后的惩罚系数,在本题中,我们设置 w_1 为10, w_2 为0.2, w_3 为1。

考虑到很可能出现到所有拖车仅仅只出动一次不够完成所有需求这种情况,因此此处提出多次巡回假设,即拖车进行二次巡回可以满足需求列表中的所有需求。我们将车辆编号进行改变,即 $k \in \{1, 2, \dots, K, K+1, K+2, \dots, 2K\}$, $K+1$ 到 $2K$ 这一部分即为二次巡回作业拖车的虚拟编号,虚拟编号拖车的路径即为二次巡回作业路径,第 k 台拖车的二次巡回作业虚拟编号为 $k+K$ 。我们对式(6.3)和式(6.4)进行修改,从而得到改进后的模型:

$$\min_{R,P} \{w_1 \cdot \sum_{k=1}^{2K} [D_k^{(R)} + D_k^{(P)}] + w_2 \cdot \sum_{j=1}^K \max[(b_j - s_j), 0] + w_3 \cdot \sum_{j=1}^L \max[(s_j - b_j), 0]\} \quad (6.10)$$

$$s.t. \quad 0 \leq n_k \leq L \quad k = 1, 2, \dots, 2K$$

$$\sum_{k=1}^{2K} n_k = L$$

$$M_i \cap M_j = \emptyset \quad \forall i \neq j$$

$$\exists r_{ki} \in R_k, p_{ki} \in P_k, s.t. id(r_{ki}) = id(p_{ki})$$

6.3 模型的求解

模型的求解主要包括有采样和迭代求解两部分。在采样过程中,我们根据已给的约束条件来生成符合条件的样本,在获取到这些样本后通过使用遗传算法来寻找最优的路径。

6.3.1 遗传算法的流程

遗传算法是一种模仿生物进化自然选择的原理和基因遗传机制的启发式搜索算法。这一算法的基本思想是,在最初基因编码的基础上,一组随机产生的解决方案按照染色体的选择、交叉、变异,逐代遗传,使得后代产生最优解的概率增加,从而展示父母的遗传基因的特性。

遗传算法的步骤为:

- Step1. 编码，确定表示可行解的染色体编码方法；
 Step2. 确定解码方法；
 Step3. 确定个体适应度的评价方法；
 Step4. 设计遗传因子，确定选择、交叉、变异运算等具体操作方法；
 Step5. 设置运行参数，如种族规模、交叉概率、变异概率、迭代数等。
 它的算法流程如图 4 所示

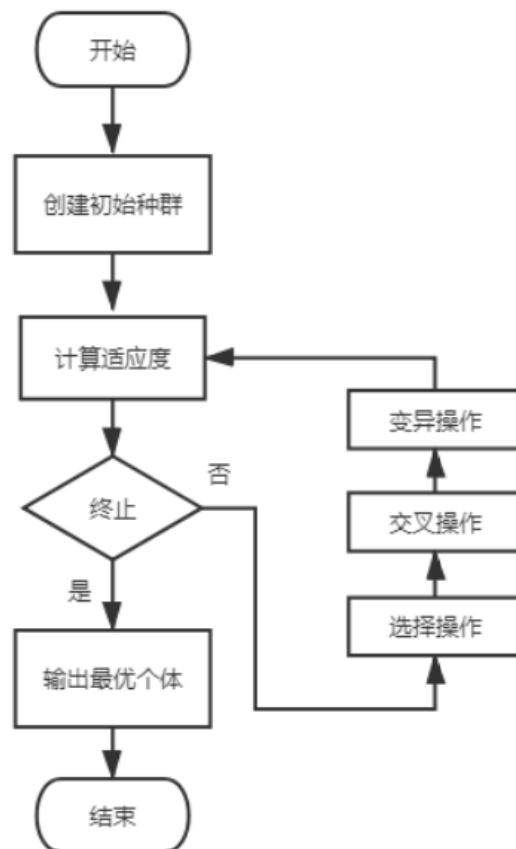


图 4 遗传算法的流程图

6.3.2 遗传算法在问题二中的应用

(1). 编码和解码

根据前文提到的遗传算法，本文对染色体采用自然数编码的方式，且采用矩阵作为染色体的表现形式，具体编码方法为：

拖车行驶的每一条可行的路径可以编写为：出发点 B 点 r_{k0} -存储点 r_{ki} -配送中心点 p_{k0} -工位点 p_{ki} -终点 p_{kn_k} ($r_{k0} - r_{ki} - p_{k0} - p_{ki} - p_{kn_k} - r_{k0} \dots \dots$)，其中每一段 $r_{k0} - r_{ki} - p_{k0} - p_{ki} - p_{kn_k}$ 代表一条可行路径。拖车数量 k 为可变数值，每一行代表一辆拖车要经过的工位点（存储点），不同的行代表不同的拖车所经过

的点位；第一列代表车辆的起点，最后一列的点代表车辆的终点。当列的顺序不同，其输出的结果也会不同。为了更清晰地说明具体编码方式，我们举例如式（6.11）。

$$\begin{bmatrix} 0 & 1 & 2 & 3 & 0 \\ 0 & 4 & 5 & 6 & 0 \\ 0 & 7 & 8 & 9 & 0 \end{bmatrix} \quad (6.11)$$

我们称式(6.11)的矩阵为方案矩阵，代表有3辆拖车（指3行）的执行运输方案，其中，第一辆车经过点位1、2、3。第二辆车经过点位4、5、6，第三辆车经过点位7、8、9。

(2). 初始群体

本文所使用的初始种群产生方式的具体操作为：

- ①从需求列表中的工位点取出若干个进行随机排列，产生随机数列，并根据该随机数列找到对应的存储点数列，按照先取货再运输的原则生成一条完整的配送路线。重复以上步骤直到需求列表中所有的工位点都被分配完，从而生成一个完整的配送方案。
- ②对生成的配送方案进行约束条件判断。若该配送方案符合约束条件，则将该方案加入种群。
- ③重复①, ②中操作，直到初始种群中的方案数目达到预先设定的阈值。

(3). 适应度函数

在遗传算法中，适应度是进行种群优化的主要依据和评价指标，它是评估染色体优劣的关键指标，也是选择算子运行的重要依据。

群体的进化程度由个体的适应度决定，适应度的值必须是一个大于零的数，适值越大，个体就越有可能遗传到下一代。

本问题采用的适应度函数为：

$$Cost = w_1 \cdot \sum_{k=1}^{2K} [D_k^{(R)} + D_k^{(P)}] + w_2 \cdot \sum_{j=1}^K \max[(b_j - s_j), 0] + w_3 \cdot \sum_{j=1}^L \max[(s_j - b_j), 0] \quad (6.12)$$

$$Fitness = \frac{C_0}{C_1} \quad (6.13)$$

其中，式(6.12)和式(6.10)中的目标函数相同，此处不重复解释，式(6.13)中 C_0 为初始种群中最优个体所对应方案的成本， C_1 为染色体个体所对应的方案的成本。当染色体进化过程中出现成本降低的情况时，适应度便会增大，以实现遗传算法的遗传过程中追求高适应度的目标。

(4). 选择算子

选择算子是依据一些特殊的方法或原则，以适应度值作为运算依据，从父

代种群选择出表现优良的个体，没有被选中的个体就会被淘汰。常用的选择策略有轮盘赌法、随机的无回放选择、最优保存选择、排序法选择等。本论文采取的选择策略为最优选择保存，具体流程如下：

Step1：采用最优保存策略，每一代均对染色体进行适应度函数的计算并进行排序。

Step2：适应度最优的染色体与上一代适应度最优的染色体进行对比，若更优则替代，否则保留上一代适应度最优的染色体。

Step3：当代适应度最优的染色体替换当代适应度最劣的染色体。

通过采用此策略，能更好的保护优秀的个体进入下一代，且能够替代适应度较低的个体，降低优秀的个体被破坏的情况，有效提高算法的收敛速度。

(5). 染色体重组

传统遗传算法中，染色体的重组主要依赖于交叉算子和变异算子。本文使用的算子分别为交叉算子、基因换位算子、基因突变算子。

①交叉算子

本问题中的交叉算子根据子路径交换交叉 (Subtour Exchange Crossover, SE) 的思想来设计，具体思路如下：从方案 A 中随机选择一辆拖车的路径，随机地从中确定一个位置 P，并确定其右边的若干个位置来构成一个序列 S，最后从另外的方案 B 中找到序列 S 中各元素所对应的位置，并将其与 A 中序列 S 的位置进行替换。

②基因换位算子

遗传算法的基因换位操作是按一定的概率 P_e 交换一条染色体中某两个（些）位置的基因值的过程，被交换基因值的位置是随机的。

基因换位可分为单点换位和多点换位。前者是交换一对位置的基因值；而后者是先确定一个正整数 e，再取随机数 i ($1 \leq i \leq s$)，然后交换 i 对位置的基因值。

本文中采用单点换位操作，具体操作如下：

Step1：从方案矩阵中确认两辆拖车（两条路线）路径中各取 1 个点（工位点以及对应存储点）

Step2：对两条路径中选定的两个点进行交换，完成操作

③基因突变算子

先从一个方案中随机确定两辆拖车的路径，从包含更多需求工位的路径 A 中随机删除一个需求工位 P，并将删除的需求工位 P 加入到包含更少需求工位的路径 B 中。

6. 3. 3 求解结果

表 2 问题二遗传算法求解结果

任务 1

拖车编号	取件顺序	配送顺序
1	III11-III12-III15-III16	II 11- II 12- II 15- II 16
2	III4-III3-III8-III7	I 4- I 3- I 8- I 7

任务 2

拖车编号	取件顺序	配送顺序
1	III3-III8-III17-III19	II 3- II 8- II 17- II 19
2	III15-III13-III16-III16	II 15- I 13- I 16 I 16
3	III7-III22-III12-III9	I 7- I 22- I 12- I 9
4	III5-III5-III21-III18	II 5- II 5- I 21- I 18
1	III24-III22-III11	II 24- II 22- II 11

任务 3

拖车编号	取件顺序	配送顺序
1	III19-III3-III12-III13	II 19- I 3- II 12- I 13
2	III2-III7-III14-III22	II 2- I 7- I 14- II 22
3	III8-III17-III17-III13	II 8- II 17- I 17- II 13
4	III18-III19-III22-III24	I 18- I 19- I 22- II 24
5	III9-III6-III5-III21	II 0- I 6- I 5- I 21
6	III1-III12-III1	II 1- I 12- I 1

6.3.4 结果分析

通过对表 2 遗传算法给出结果进行验证，我们可以发现任务 1，任务 2 和任务 3 均能满足需求列表的所有需求。在任务 3 中拖车编号为 3 的取件顺序为 III 8-III17-III17-III13，可以看出从 III17 进行了两次取件操作；而任务 3 中拖车编号为 6 的取件顺序为 III1-III12-III1，易知这并非最优的取件顺序，这也说明遗传算法的局限性，它并不能全局寻找最优解，找到的很大可能只是次优解。

7 问题三模型的建立与求解

7.1 硬时间窗有限单向车辆路径问题的分析

如图 6 所示，根据时间窗约束的严格与否，带时间窗约束条件的车辆路径问题，可以被分为硬时间窗车辆路径问题和软时间窗车辆路径问题。硬时间窗有

限单向车辆路径问题是指每个工位都要求将物料在指定时间内送到，既不能提前，也不能拖后。

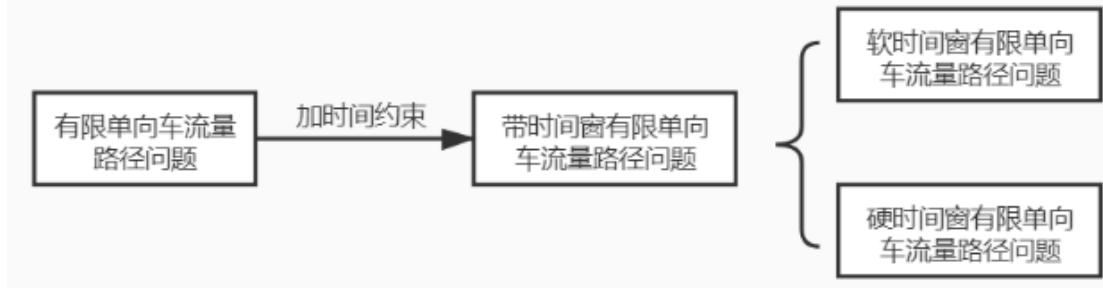


图 5 模型的分类

在第二问中，我们使用的是软时间窗有限单向车辆路径问题。而本问要求杜绝停线状况，即必须在需求时间之前将物料送到，可知为硬时间窗有限单向车辆路径问题。

硬时间窗有限单向车辆路径问题可以描述为：从 B 点起用多台拖车向多个工位运送物料，每个工位的位置和需求量一定，要求将货物送到的需求时间一定，每台拖车的载重量一定，其一次配送的最大行驶距离一定，要求合理安排车辆的派送方案，使目标函数得到优化，并满足以下约束条件：

- (1) 每条配送路径上各工位的物料需求量之和不超过配送拖车的载重量；
- (2) 每个工位的需求必须满足，且一次只能由一台配送拖车送货。
- (3) 车辆将货物送到工位后将返回配送中心，并可以多次巡回使用。
- (4) 货物必须在需求时间内送到

即与第二问相比，多了一个约束条件 (4)，此模型可杜绝停线情况。

7.2 模型的建立

本问中，各指标的含义与问题二的完全相同，此处不再重述。为了保证在拖车在给定的需求时间前到达，我们加入新的约束：

$$s_i \leq b_i, i = 1, 2, \dots, L \quad (7.1)$$

结合 6.2 中所提出的模型与新约束式(7.1)，我们建立新模型如下：

$$\min_{R,P} \{ \sum_{k=1}^{2K} [D_k^{(R)} + D_k^{(P)}] \} \quad (7.2)$$

$$s.t. \quad 0 \leq n_k \leq L \quad k = 1, 2, \dots, 2K \quad (7.3)$$

$$\sum_{k=1}^{2K} n_k = L \quad (7.4)$$

$$M_i \cap M_j = \emptyset \quad \forall i \neq j \quad (7.5)$$

$$\exists r_{ki} \in R_k, p_{ki} \in P_k, s.t. id(r_{ki}) = id(p_{ki}) \quad (7.6)$$

$$s_i \leq b_i, i = 1, 2, \dots, L \quad (7.7)$$

式 (7.1) 为目地函数, 即要求配送总路径最短, 不再考虑早到时间与晚到时间的损失。式 (7.2) – (7.6) 的含义与第二问模型中的含义相同, 此处不再重述。

7.3 模型的求解

本问同样采用问题二中的遗传算法进行求解, 由于模型的目地函数发生了变化, 因此我们对适应度函数进行了修改, 即:

$$Cost = \sum_{k=1}^{2K} [D_k^{(R)} + D_k^{(P)}] \quad (7.8)$$

$$Fitness = \frac{C_0}{C_1} \quad (7.9)$$

公式 (7.8) 即公式 (7.2) 所述的目地函数, 公式 (7.9) 的解释和公式 (6.13) 相同。

求解算法的剩余部分与 6.3 节相同, 此处不再重述。

我们使用问题二中的初始群体生成 (即采样与约束检验) 时, 发现不论我们怎样进行样本量的扩充, 公式 (7.7) 所给定的约束我们总是无法满足, 即无法让拖车同时满足需求列表中所有需求的需求时间, 结果如图 6 所示, “Min fitness of current pop: inf” 代表无法满足约束。这一方面是遗传算法的局限性造成 (采样方面), 难以进行全局优化, 另一方面, 问题三相比于问题二可用样本的范围进行了大幅度缩减, 导致了原有的采样方法不能找到任何可用方案。

```
Best individual found is [[['B', 21, 12, 22, 22], ['B', 16, 5, 18, 7], ['B', 17, 16, 9, 24], ['B', 8, 3, 11, 5], ['B', 13, 15], ['B', 19]], [['D', 45, 36, 70, 46], ['D', 40, 53, 42, 31], ['D', 65, 40, 33, 72], ['D', 56, 51, 59, 53], ['D', 37, 63], ['D', 67]]], inf
Min fitness of current pop: inf
```

图 6 失败结果

7.4 机制修改

我们对任务数据进行分析发现, 对原有的采样方法进行分析发现, 任务 3 中有些需求例如需求编号 1 的需求时间为 51 秒, 需求 2、3、15 等多个需求和他有

着极为相近的需求时间，这就大大降低了采样结果能够通过诸多约束的成功率。因此我们需要考虑对采样方法进行修改。

在原本的方案采样方法中，我们更倾向于将拖车的载重拉满后再进行配送，但是对于时间需求间距小的需求列表来说这无疑减小了采样成功率，因此我们去除优先考虑将拖车拉满的情况，也就是不再只是二次巡回假设，我们使用更多次的巡回来解决，即考虑 $k+nK$ 个虚拟编号，这样使得最终结果更加平均。

8 模型的推广

前面的问题中，研究的配送车辆优化调度问题是基于给出的信息，如配送中心信息、存储点和工位信息、小车信息、运输网络信息都是确定的，我们称上述配送车辆优化调度问题为静态问题。

但是，客观世界中，存在很多的不确定的因素，如不确定货物需求量、不确定需求时间、道路拥堵、车辆之间的碰撞、小车故障等等。这些因素是随着时间的增加而有一定概率出现，这就需要我们对模型和路径及时进行调整。此时，静态的配送车辆优化调度问题的模型无法处理这些问题，需要对模型进行改进。由于一些问题是随时间的推移而出现的，所以我们可以建立动态优化模型。

随着目前的通信和信息处理技术的发展，实时处理随机和模糊的信息、动态调控车间配送情况有了实现的可能。目前关于动态配送车辆优化调度问题的研究多集中在需求不确定的配送车辆优化调度问题。动态配送车辆优化掉地问题比静态配送车辆调度问题更能反映实际状况，更有利于解决现实问题。所以，动态配送车辆优化调度问题的研究有着更强的理论和现实意义。

9 参考文献

- [1] 郎茂祥. 配送车辆优化调度模型与算法[M]. 电子工业出版社, 2009.
- [2] 符卓. 开放式车辆路径问题及其应用研究[D]. 中南大学, 2003.
- [3] 梁思媚. 带时间窗和送取货的开放式备件物流车辆路径规划问题研究[D]. 暨南大学, 2019.
- [4] 聂靖. 带装载能力限制的开放式车辆路径问题及其遗传算法研究[D]. 长沙: 中南大学, 2007.
- [5] 蒋波. 基于遗传算法的带时间窗车辆路径优化问题研究[D]. 北京交通大学, 2010.
- [6] 孙国华. 带时间窗的开放式满载车辆路径问题建模及其求解算法[J]. 系统工程理论与实践, 2012, (08): 1801–1807.

[7] 聂靖. 带装载能力限制的开放式车辆路径问题及其遗传算法研究[D]. 长沙: 中南大学, 2007.

附录

```
# GenA 类中 evaluate 的形参 is_hard 为 False 时为问题二, 为 True 且 w2、w3=0  
为问题三  
  
import random  
from operator import itemgetter  
import numpy as np  
import seaborn as sns  
import matplotlib.pyplot as plt  
  
random.seed(10)  
node = {'A': (83, 131.5), 'B': (83, 111.5), 'C': (83, 91.5), 'D': (83., 61.5),  
        'E': (1.5, 51.5), 'F': (83, 51.5), 'G': (83, 41.5), 'H': (164.5, 41.5),  
        'I': (1.5, 31.5), 'J': (83, 31.5), 'K': (83, 21.5), 'L': (164.5, 21.5),  
  
        'M': (1.5, 11.5), 'N': (83, 11.5), 'O': (1.5, 1.5), 'P': (1.5, 61.5), 'Q': (164.5,  
        61.5), 'R': (83, 1.5),  
        'S': (164.5, 1.5),  
        1: (3., 131.5), 2: (23., 131.5), 3: (43., 131.5), 4: (63., 131.5),  
        5: (103., 131.5), 6: (123., 131.5), 7: (143., 131.5), 8: (163., 131.5),  
        9: (3., 111.5), 10: (23., 111.5), 11: (43., 111.5), 12: (63., 111.5),  
  
        13: (103., 111.5), 14: (123., 111.5), 15: (143., 111.5), 16: (163., 111.5),  
        17: (3., 91.5), 18: (23., 91.5), 19: (43., 91.5), 20: (63., 91.5),  
        21: (103., 91.5), 22: (123., 91.5), 23: (143, 91.5), 24: (163, 91.5),  
        25: (11.5, 61.5),  
        26: (31.5, 61.5),  
        27: (51.5, 61.5),  
        28: (71.5, 61.5),  
        29: (11.5, 51.5),  
        30: (31.5, 51.5),  
        31: (51.5, 51.5),  
        32: (71.5, 51.5),  
        33: (11.5, 41.5),
```

34: (31. 5, 41. 5),
35: (51. 5, 41. 5),
36: (71. 5, 41. 5),
37: (11. 5, 31. 5),
38: (31. 5, 31. 5),
39: (51. 5, 31. 5),
40: (71. 5, 31. 5),
41: (11. 5, 21. 5),
42: (31. 5, 21. 5),
43: (51. 5, 21. 5),
44: (71. 5, 21. 5),
45: (11. 5, 11. 5),
46: (31. 5, 11. 5),
47: (51. 5, 11. 5),
48: (71. 5, 11. 5),
49: (93, 61. 5), 50: (113, 61. 5), 51: (133, 61. 5), 52: (153, 61. 5),
53: (93, 51. 5), 54: (113, 51. 5), 55: (133, 51. 5), 56: (153, 51. 5),
57: (93, 41. 5),
58: (113, 41. 5),
59: (133, 41. 5),
60: (153, 41. 5),
61: (93, 31. 5),
62: (113, 31. 5),
63: (133, 31. 5),
64: (153, 31. 5),
65: (93, 21. 5),
66: (113, 21. 5),
67: (133, 21. 5),
68: (153, 21. 5),
69: (93, 11. 5),
70: (113, 11. 5),
71: (133, 11. 5),
72: (153, 11. 5)

}

```

def node_dis(node1, node2) :
    if node1[0] == node2[0] or node1[1]==node2[1]:
        diss = abs(node1[0]-node2[0]) + abs(node1[1]-node2[1])
        return diss
    else:
        print('not at a line:', node1, ', ', node2)
        return 0

def cal_dis3(nodes):
    diss = np.zeros(73)
    matrix = np.zeros((73, 73))
    for i in range(1, 25):
        tmp = (i-1) // 8
        if tmp == 0:
            d = node_dis(nodes[i], nodes['A'])
            d = d + node_dis(nodes['A'], nodes['D'])
            diss[i] = d
        elif tmp == 1:
            d = node_dis(nodes[i], nodes['B'])
            d = d + node_dis(nodes['B'], nodes['D'])
            diss[i] = d
        elif tmp == 2:
            d = node_dis(nodes[i], nodes['C'])
            d = d + node_dis(nodes['C'], nodes['D'])
            diss[i] = d

    for i in range(1, 25):
        for j in range(i+1, 25):
            if i == 1 and j == 12:
                debug = 0
                tmp_i = (i-1)//8
                tmp_j = (j-1)//8

```

```

        if abs(tmp_i-tmp_j) == 0:
            matrix[i][j]= matrix[j][i] = node_dis(node[i], node[j])
        elif abs(tmp_i - tmp_j) == 1:
            d1, d2 = 0., 0.
            for tmp in ['A', 'B']:
                d1 += node_dis(node[i], node[tmp])
            for tmp in ['B', 'C']:
                d2 += node_dis(node[j], node[tmp])
            matrix[i][j]= matrix[j][i] = d1+d2+node_dis(node['A'], node['B'])

        elif abs(tmp_i -tmp_j) == 2:
            d1 = node_dis(node[i], node['A'])
            d2 = node_dis(node[j], node['C'])
            matrix[i][j] = matrix[j][i] = d1+d2+node_dis(node['A'], node['C'])

    return diss, matrix

def route_dis(road):
    global node
    ans = 0.
    for i in range(1, len(road)):
        ans += node_dis(node[road[i-1]], node[road[i]])
    return ans

def cal_dis1(nodes):
    diss = np.zeros(73)
    matrix = np.zeros((73, 73))
    for i in range(25, 49):
        if (i - 24 - 1) // 4 == 0:
            d = node_dis(nodes[i], nodes['D'])
            diss[i] = d
        elif (i-24-1)//4 == 1:
            d = node_dis(nodes[i], nodes['E'])

```

```

        d      =      d      +      node_dis(nodes['E'], nodes['P'])      +
node_dis(node['P'], node['D'])
        diss[i] = d
    elif (i-24-1)//4 == 2:
        d = node_dis(nodes[i], nodes['G'])
        d = d + node_dis(nodes['G'], nodes['D'])
        diss[i] = d
    elif (i-24-1)//4 == 3:
        d = node_dis(nodes[i], nodes['I'])
        d      =      d      +      node_dis(nodes['I'], nodes['P'])      +
node_dis(node['P'], node['D'])
        diss[i] = d
    elif (i-24-1)//4 == 4:
        d = node_dis(nodes[i], nodes['K'])
        d = d + node_dis(nodes['K'], nodes['D'])
        diss[i] = d
    elif (i-24-1)//4 == 5:
        d = node_dis(nodes[i], nodes['M'])
        d      =      d      +      node_dis(nodes['M'], nodes['P']) +
node_dis(node['P'], node['D'])
        diss[i] = d

for i in range(25, 49):
    for j in range(i+1, 49):
        tmp_i = (i-24-1)//4
        tmp_j = (j-24-1)//4
        if i == 25 and j == 29:
            debug = 0
        if tmp_i == 0:
            if tmp_j == 1 or tmp_j == 3 or tmp_j == 5:
                d1 = node_dis(nodes[i], nodes['P'])
                d2, d3 = 0., 0.
                for tmp in ['E', 'I', 'M']:
                    d_tmp = node_dis(nodes[j], nodes[tmp])
                    if d3 == 0:

```

```

d3 = node_dis(nodes[tmp], nodes['P']) if
d_tmp != 0 else 0
d2 += d_tmp
matrix[i][j] = matrix[j][i] = d1 + d2 + d3
elif tmp_j == 2 or tmp_j == 4:
    d1 = node_dis(nodes[i], nodes['D'])
    d2, d3 = 0., 0.
    for tmp in ['G', 'K']:
        d_tmp = node_dis(nodes[j], nodes[tmp])
        if d3 == 0:
            d3 = node_dis(nodes[tmp], nodes['D']) if
d_tmp != 0 else 0
        d2 += d_tmp
        matrix[i][j] = matrix[j][i] = d1+d2+d3
    elif tmp_j == 0:
        matrix[i][j] = matrix[j][i] =
node_dis(nodes[i], nodes[j])
    elif tmp_i == 1:
        if tmp_j == 1:
            matrix[i][j] = matrix[j][i] =
node_dis(nodes[i], nodes[j])
        elif tmp_j == 3 or tmp_j == 5:
            d1 = node_dis(nodes[i], nodes['E'])
            d2 = node_dis(nodes[j], nodes['I']) if tmp_j == 3
else node_dis(nodes[j], nodes['M'])
            d3 = node_dis(nodes['E'], nodes['I']) if tmp_j ==
3 else node_dis(nodes['E'], nodes['M'])
            matrix[i][j] = matrix[j][i] = d1+d2+d3
        elif tmp_j == 2 or tmp_j == 4:
            d1 = node_dis(nodes[i], nodes['E'])
            d2 = node_dis(nodes[j], nodes['G'])
            d3 = node_dis(nodes['G'], nodes['D']) +
node_dis(nodes['D'], nodes['P']) + node_dis(nodes['P'], nodes['E'])
            matrix[i][j] = matrix[j][i] = d1 + d2 + d3
    elif tmp_i == 2:

```

```

if tmp_j == tmp_i:
    matrix[i][j] = matrix[i][j] = node_dis(nodes[i],
nodes[j])

elif tmp_j == 3:
    d1      = node_dis(nodes[i], nodes['G'])      +
node_dis(nodes['G'], nodes['D']) + node_dis(nodes['D'], nodes['P']) +
node_dis(nodes['P'], nodes['I'])

    d2 = node_dis(nodes[j], nodes['I'])
    matrix[i][j] = matrix[j][i] = d1 + d2

elif tmp_j == 4:
    d1 = node_dis(nodes[i], nodes['G'])
    d2 = node_dis(nodes[j], nodes['K'])
    matrix[i][j] = matrix[j][i] = d1 + d2 +
node_dis(nodes['G'], nodes['K'])

elif tmp_j == 5:
    d1 = node_dis(nodes[i], nodes['G'])
    d2 = node_dis(nodes[j], nodes['M'])
    matrix[i][j] = matrix[j][i] = d1 + d2 +
node_dis(nodes['M'], nodes['O']) + node_dis(nodes['O'], nodes['R']) +
node_dis(nodes['R'], nodes['G'])

elif tmp_i == 3:
    if tmp_j == 3:
        matrix[i][j] = matrix[j][i] = node_dis(nodes[i],
nodes[j])

    elif tmp_j == 4:
        d1 = node_dis(nodes[i], nodes['I'])
        d2 = node_dis(nodes[j], nodes['K'])
        d3      = node_dis(nodes['I'], nodes['O'])      +
node_dis(nodes['O'], nodes['R']) + node_dis(nodes['R'], nodes['K'])

        matrix[i][j] = matrix[j][i] = d1 + d2 + d3

    elif tmp_j == 5:
        d1 = node_dis(nodes[i], nodes['I'])
        d2 = node_dis(nodes[j], nodes['M'])
        d3 = node_dis(nodes['M'], nodes['I'])

        matrix[i][j] = matrix[j][i] = d1 + d2 + d3

```

```

        elif tmp_i == 4:
            if tmp_j == 4:
                matrix[i][j] = matrix[i][j] = node_dis(nodes[i],
nodes[j])
            elif tmp_j == 5:
                d1 = node_dis(nodes[i], nodes['K'])
                d2 = node_dis(nodes[j], nodes['M'])
                d3      =      node_dis(nodes['M'], nodes['O'])      +
node_dis(nodes['O'], nodes['K'])
                matrix[i][j] = matrix[j][i] = d1 + d2 + d3
            elif tmp_i == 5:
                matrix[i][j] = matrix[j][i] = node_dis(nodes[i],
nodes[j])

    return diss, matrix

def cal_dis2(nodes):
    diss = np.zeros(73)
    matrix = np.zeros((73, 73))
    for i in range(49, 73):
        if (i - 48 - 1) // 4 == 0:
            d = node_dis(nodes[i], nodes['D'])
            diss[i] = d
        elif (i-48-1)//4 == 1:
            d = node_dis(nodes[i], nodes['F'])
            d = d + node_dis(nodes['F'], nodes['D'])
            diss[i] = d
        elif (i-48-1)//4 == 2:
            d = node_dis(nodes[i], nodes['H'])
            d      =      d      +      node_dis(nodes['H'], nodes['Q'])      +
node_dis(nodes['Q'], nodes['D'])
            diss[i] = d
        elif (i-48-1)//4 == 3:
            d = node_dis(nodes[i], nodes['J'])
            d = d + node_dis(nodes['J'], nodes['D'])

```

```

        diss[i] = d
    elif (i-48-1)//4 == 4:
        d = node_dis(nodes[i], nodes['L'])
        d = d + node_dis(nodes['L'], nodes['Q']) +
node_dis(nodes['Q'], nodes['D'])
        diss[i] = d
    elif (i-48-1)//4 == 5:
        d = node_dis(nodes[i], nodes['N'])
        d = d + node_dis(nodes['N'], nodes['D'])
        diss[i] = d

for i in range(49, 73):
    for j in range(i+1, 73):
        tmp_i = (i-48-1)//4
        tmp_j = (j-48-1)//4
        if tmp_i==tmp_j:
            matrix[i][j] = matrix[j][i] =
node_dis(nodes[i], nodes[j])
        elif tmp_i == 0:
            if tmp_j == 1:
                matrix[i][j] = matrix[j][i] =
route_dis([i, 'D', 'F', j])
            elif tmp_j == 2:
                matrix[i][j] = matrix[j][i] = route_dis([i, 'Q',
'H', j])
            elif tmp_j == 3:
                matrix[i][j] = matrix[j][i] = route_dis([i, 'D',
'J', j])
            elif tmp_j == 4:
                matrix[i][j] = matrix[j][i] = route_dis([i, 'Q',
'L', j])
            elif tmp_j == 5:
                matrix[i][j] = matrix[j][i] = route_dis([i, 'D',
'N', j])
        elif tmp_i == 1:

```

```

    if tmp_j == 2:
        matrix[i][j] = matrix[j][i] = route_dis([i, 'F',
'D', 'Q', 'H', j])
    elif tmp_j == 3:
        matrix[i][j] = matrix[j][i] = route_dis([i, 'F',
'J', j])
    elif tmp_j == 4:
        matrix[i][j] = matrix[j][i] = route_dis([i, 'F',
'D', 'Q', 'L', j])
    elif tmp_j == 5:
        matrix[i][j] = matrix[j][i] = route_dis([i, 'F',
'N', j])
    elif tmp_i == 2:
        if tmp_j == 3:
            matrix[i][j] = matrix[j][i] = route_dis([i, 'H',
'Q', 'D', 'J', j])
        elif tmp_j == 4:
            matrix[i][j] = matrix[j][i] = route_dis([i, 'H',
'L', j])
        elif tmp_j == 5:
            matrix[i][j] = matrix[j][i] = route_dis([i, 'H',
'S', 'R', 'N', j])
    elif tmp_i == 3:
        if tmp_j == 4:
            matrix[i][j] = matrix[j][i] = route_dis([i, 'J',
'R', 'S', 'L', j])
        elif tmp_j == 5:
            matrix[i][j] = matrix[j][i] = route_dis([i, 'J',
'N', j])
    elif tmp_i == 4:
        if tmp_j == 5:
            matrix[i][j] = matrix[j][i] = route_dis([i, 'L',
'S', 'R', 'N', j])

```

```

    return diss_matrix

def I2II(nodes):
    matrix = np.zeros((73, 73))
    for i in range(25, 49):
        for j in range(49, 73):
            tmp_i = (i-24-1) //4
            tmp_j = (j-48-1)//4
            if tmp_i == 0:
                if tmp_j == 0:
                    matrix[i][j] = matrix[j][i] = route_dis([i, 'D',
j])
                elif tmp_j == 1:
                    matrix[i][j] = matrix[j][i] = route_dis([i, 'D',
'F', j])
                elif tmp_j == 2:
                    matrix[i][j] = matrix[j][i] = route_dis([i, 'Q',
'H', j])
                elif tmp_j == 3:
                    matrix[i][j] = matrix[j][i] = route_dis([i, 'D',
'J', j])
                elif tmp_j == 4:
                    matrix[i][j] = matrix[j][i] = route_dis([i, 'Q',
'L', j])
                elif tmp_j == 5:
                    matrix[i][j] = matrix[j][i] = route_dis([i, 'D',
'N', j])
            elif tmp_i == 1:
                if tmp_j == 0:
                    matrix[i][j] = matrix[j][i] = route_dis([i,
'E', 'P', j])
                elif tmp_j == 1:
                    matrix[i][j] = matrix[j][i] = route_dis([i, 'E',
'P', 'D', 'F', j])
                elif tmp_j == 2:

```

```

        matrix[i][j] = matrix[j][i] = route_dis([i, 'E',
'P', 'Q', 'H', j])
    elif tmp_j == 3:
        matrix[i][j] = matrix[j][i] = route_dis([i, 'E',
'P', 'D', 'J', j])
    elif tmp_j == 4:
        matrix[i][j] = matrix[j][i] = min(route_dis([i,
'E', 'P', 'Q', 'L', j]), route_dis([i, 'E', 'O', 'S', 'L', j]))
    elif tmp_j == 5:
        matrix[i][j] = matrix[j][i] = min(route_dis([i,
'E', 'P', 'D', 'N', j]), route_dis([i, 'E', 'O', 'R', 'N', j]))
    elif tmp_i == 2:
        if tmp_j == 0:
            matrix[i][j] = matrix[j][i] = route_dis([i, 'G',
'D', j])
        elif tmp_j == 1:
            matrix[i][j] = matrix[j][i] = route_dis([i, 'G',
'F', j])
        elif tmp_j == 2:
            matrix[i][j] = matrix[j][i] = route_dis([i, 'G',
'D', 'Q', 'H', j])
        elif tmp_j == 3:
            matrix[i][j] = matrix[j][i] = route_dis([i, 'G',
'J', j])
        elif tmp_j == 4:
            matrix[i][j] = matrix[j][i] = min(route_dis([i,
'G', 'D', 'Q', 'L', j]), route_dis([i, 'G', 'R', 'S', 'L', j]))
        elif tmp_j == 5:
            matrix[i][j] = matrix[j][i] = route_dis([i, 'G',
'N', j])
    elif tmp_i == 3:
        if tmp_j == 0:
            matrix[i][j] = matrix[j][i] = route_dis([i, 'I',
'P', j])
        elif tmp_j == 1:

```

```

        matrix[i][j] = matrix[j][i] = route_dis([i, 'I',
'P', 'D', 'F', j])
        elif tmp_j == 2:
            matrix[i][j] = matrix[j][i] = route_dis([i, 'I',
'P', 'Q', 'H', j])
        elif tmp_j == 3:
            matrix[i][j] = matrix[j][i] = route_dis([i, 'I',
'P', 'D', 'J', j])
        elif tmp_j == 4:
            matrix[i][j] = matrix[j][i] = min(route_dis([i,
'I', 'P', 'Q', 'L', j]), route_dis([i, 'I', 'O', 'S', 'L', j]))
        elif tmp_j == 5:
            matrix[i][j] = matrix[j][i] = route_dis([i, 'I',
'O', 'R', 'N', j])
        elif tmp_i == 4:
            if tmp_j == 0:
                matrix[i][j] = matrix[j][i] = route_dis([i, 'K',
'D', j])
            elif tmp_j == 1:
                matrix[i][j] = matrix[j][i] = route_dis([i, 'K',
'F', j])
            elif tmp_j == 2:
                matrix[i][j] = matrix[j][i] = route_dis([i, 'K',
'D', 'Q', 'H', j])
            elif tmp_j == 3:
                matrix[i][j] = matrix[j][i] = route_dis([i, 'K',
'J', j])
            elif tmp_j == 4:
                matrix[i][j] = matrix[j][i] = min(route_dis([i,
'K', 'D', 'Q', 'L', j]), route_dis([i, 'K', 'R', 'S', 'L', j]))
            elif tmp_j == 5:
                matrix[i][j] = matrix[j][i] = route_dis([i, 'K',
'N', j])
        elif tmp_i == 5:
            if tmp_j == 0:

```

```

        matrix[i][j] = matrix[j][i] = route_dis([i, 'M',
'P', j])
        elif tmp_j == 1:
            matrix[i][j] = matrix[j][i] = route_dis([i, 'M',
'O', 'R', 'F', j])
        elif tmp_j == 2:
            matrix[i][j] = matrix[j][i] = min(route_dis([i,
'M', 'P', 'Q', 'H', j]), route_dis([i, 'M', 'O', 'S', 'H', j]))
        elif tmp_j == 3:
            matrix[i][j] = matrix[j][i] = route_dis([i, 'M',
'O', 'R', 'J', j])
        elif tmp_j == 4:
            matrix[i][j] = matrix[j][i] = route_dis([i, 'M',
'O', 'S', 'L', j])
        elif tmp_j == 5:
            matrix[i][j] = matrix[j][i] = route_dis([i, 'M',
'O', 'R', 'N', j])
    return matrix

def cal_distance(matrix):
    task = []
    distance = []
    for i in range(1, 25):
        j = 24+i
        jj = 48 + i
        distance.append((matrix[i][j], i, j))
        distance.append((matrix[i][jj], i, jj))
    sort_dist = sorted(distance, key=lambda x:x[0])
    #print(sort_dist)

    left = 0
    right = len(sort_dist) - 1
    tmp_task = []
    while(left < right-1):
        # tmp_task.append(sort_dist[left])

```

```

# if (len(task) <2 and len(tmp_task) == 4) or len(tmp_task) ==
5:
    #     task.append(tmp_task)
    #     tmp_task = []
    # left += 1

tmp_task.append(sort_dist[right])
if (len(task) <2 and len(tmp_task) == 4) or len(tmp_task) ==
5:
    task.append(tmp_task)
    tmp_task = []
right -= 1

tmp_task.append(sort_dist[left])
if (len(task) < 2 and len(tmp_task) == 4) or len(tmp_task) ==
5:
    task.append(tmp_task)
    tmp_task = []
left += 1

# tmp_task.append(sort_dist[left])
# if len(tmp_task) == 5:
#     task.append(tmp_task)
#     tmp_task = []
# left += 1

if left == right - 1:
    tmp_task.append(sort_dist[left])
    tmp_task.append(sort_dist[right])
if len(tmp_task) > 0:
    task.append(tmp_task)
print(task)

cost_list = []

```

```

for car_task in task:
    cost = 0.
    for ct, _, _ in car_task:
        cost += ct
    cost_list.append(cost)
print('max cost: ', max(cost_list))
print(cost_list)

def calB(nodes):
    disB = np.zeros(25)
    for i in range(1, 25):
        tmp_i = (i-1) //8
        if tmp_i == 0:
            d1 = route_dis([i, 'A', 'B'])
            disB[i] = d1
        elif tmp_i == 1:
            disB[i] = route_dis([i, 'B'])
        elif tmp_i == 2:
            disB[i] = route_dis([i, 'C', 'B'])
    return disB

```

```

class Gene:
    def __init__(self, data, use, not_use):
        store, work = self.create_store(data)
        self.data = [store, work]
        self.use = use
        self.not_use = not_use
        self.d_backup = data

    def create_store(self, data):
        store = []
        work = []
        for i in range(len(data)):
            s_t = []

```

```

        s_w = []
        for j in range(len(data[i])):
            tmp = data[i][j]
            if j != 0:
                store_tmp = 24 if tmp % 24 == 0 else tmp % 24
                s_t.append(store_tmp)
                s_w.append(tmp)
            else:
                s_t.append('B')
                s_w.append('D')
        store.append(s_t)
        work.append(s_w)
    return store, work

def copy(self):
    gen = Gene(self.d_backup, self.use, self.not_use)
    return gen

class TaskNode:
    def __init__(self, time_list):
        self.time_list = time_list
        self.cur = 0

    def step(self):
        val = self.time_list[self.cur] if len(self.time_list) - 1 >=
self.cur else -1
        self.cur += 1
        return val

    def restore(self):
        self.cur = 0

class GenA:
    def
    __init__(self, parameter, matrix, disB, task, car_num, disD, w1=100, w2=0.2, w

```

```

3=1, v=5, t1=3, t2=3) :
    ...
    :param parameter:
    :param matrix:
    :param disB:
    :param task: [tuple(point, time)]
    :param car_num:
    ...
    self.pop_size = parameter[0]
    self.epoch = parameter[1]
    self.CXPB = parameter[2]
    self.MUTPB = parameter[3]
    self.matrix = matrix
    self.disB = disB
    self.task = task
    self.car_num = car_num
    self.disD = disD
    self.task_dic = {}
    self.w1 = w1
    self.w2 = w2
    self.w3 = w3
    self.v = v # 拖车速度
    self.t1 = t1 # 装货时间
    self.t2 = t2 # 卸货时间

    cycle_num = len(task) // (self.car_num * 4)
    num_rest = car_num if (len(task) % (4*self.car_num)) > 0 else
0
    self.car_num = self.car_num * (len(task)//(4*self.car_num)) +
num_rest

    pop = []

    for p, t in task:

```

```

#points.append(p)
work_li = self.task_dic.get(p, [])
work_li.append(t)
self.task_dic[p] = work_li
for k in list(self.task_dic.keys()):
    self.task_dic[k] = sorted(self.task_dic[k])
    self.task_dic[k] = TaskNode(self.task_dic[k])

while len(pop) < self.pop_size:
    plan = []
    task_num = len(task)
    points = []
    for p, t in task:
        points.append(p)
    random.shuffle(points)
    all_s = points
    use = []
    for j in range(self.car_num):
        car_tmp = ['B']
        #num = random.randint(0, min(len(points), 4))
        car_tmp.extend(points[0:4])
        #use = use | set(points[0:num])
        use.extend(points[0:4])
        points = points[4:len(points)]
        plan.append(car_tmp)
    not_use = points
    gene = Gene(plan, use, not_use)
    if not self.check(gene):
        print(self.check(gene))
        continue
    score = self.evaluate(gene)
    pop.append({'Gene':gene, 'score':score})
self.pop = pop
self.bestindividual = self.selectBest(self.pop)

```

```

def evaluate(self, gene, is_hard=False):
    D_R = 0 # 计算总取货路程
    D_P = 0 # 计算总送货路程
    T_w = 0 # 计算总早到时间
    T_p = 0 # 计算总晚到时间
    # 对拖车遍历
    # print(gene)
    for k in range(len(gene.data[0])):
        # 获取经过的仓储位
        k_r = gene.data[0][k]
        if len(k_r) == 1:
            continue
        # 该拖车取货路程
        d_r = 0
        # 对该车仓储位点遍历
        for r in range(len(k_r) - 1):
            # 如果是 B 点
            if isinstance(k_r[r], str):
                d_r += self.disB[k_r[r + 1]]
            # 如果非 B 点
            else:
                d_r += self.matrix[k_r[r]][k_r[r + 1]]
        # 最后的点与 D 点的距离
        d_r += self.disD[k_r[-1]]
        D_R += d_r
        # 取货时间
        t_current = d_r / self.v + len(k_r) * self.t1
        # 获取经过的工位
        k_p = gene.data[1][k]
        # 对点遍历
        for p in range(len(k_p) - 1):
            # 如果是 D 点
            if isinstance(k_p[p], str):
                D_P += self.disD[k_p[p + 1]]
                t_current += self.disD[k_p[p + 1]] / self.v

```

```

        # 如果是工位
    else:
        D_P += self.matrix[k_p[p]][k_p[p + 1]]
        t_current += self.matrix[k_p[p]][k_p[p + 1]] /
self.v

        # 获取任务需求时间
        req_time = self.task_dic[k_p[p + 1]].step()
        # 早到
        if t_current < req_time:
            T_w += (req_time - t_current)
            t_current = req_time + self.t2
        # 晚到
        elif t_current > req_time:
            T_p += (t_current - req_time)
            t_current = t_current + self.t2
            if is_hard:
                return float("inf")
        # 最后的点与 D 的距离
        D_P += self.disD[k_p[-1]]
for key in self.task_dic.keys():
    self.task_dic[key].restore()
cost = self.w1 * (D_R + D_P) + self.w2 * T_w + self.w3 * T_p
return cost

def selectBest(self, pop):
    s_inds = sorted(pop, key=itemgetter("score")) # from large to
small, return a pop
    return s_inds[0]

def check(self, gene: Gene):
    ...
    验证是否满足约束条件
:param gene:
:return:
...

```

```

# 6. 4
for k_data in gene.data[0]:
    if len(k_data) - 1 > len(task) or len(k_data) - 1 < 0:
        print('6. 4')
        return False

# 6. 5
N = 0
for k_data in gene.data[0]:
    N += (len(k_data) - 1)
if N != len(task):
    print('6. 5')
    return False

# 6. 6 and 6. 7
# for k in range(len(gene.data[0])):
#     # 获取 k 经过的仓储位
#     k_r = gene.data[0][k]
#     # 获取 k 经过的工位
#     k_p = gene.data[1][k]
#     for o_k in range(len(gene.data[0])):
#         # 获取 o_k 经过的仓储位
#         o_r = gene.data[0][o_k]
#         # 获取 o_k 经过的工位
#         o_p = gene.data[1][o_k]
#         set(o_r)

# 6. 10
def id(no):
    return no % 24
for k in range(len(gene.data[0])):
    # 获取 k 经过的仓储位
    k_r = gene.data[0][k]
    # 获取 k 经过的工位
    k_p = gene.data[1][k]
    for i in range(1, len(k_r)):
        ok = False
        for j in range(1, len(k_p)):

```

```

        if id(k_r[i]) == id(k_p[j]):
            ok = True
            break
        if not ok:
            print('6.10')
            return False
    return True

def selection(self, individuals, k):
    s_inds = sorted(individuals, key=itemgetter('score'))

    sum_fits = sum(ind['score'] for ind in individuals)

    chosen = []
    for i in range(k):
        u = random.random() * sum_fits
        sum_ = 0
        for ind in s_inds:
            if not self.check(ind['Gene']):
                continue
            sum_ += ind['score']
            if sum_ <= u:
                chosen.append(ind)
                s_inds.pop(0)
                break
    chosen = sorted(chosen, key=itemgetter('score'))
    return chosen

def crossoperate(self, offspring):
    spring1 = offspring[0]['Gene']
    spring2 = offspring[1]['Gene']

    car_num1 = len(spring1.data[0])
    car_num2 = len(spring2.data[0])

```

```

car1 = random.randint(0, car_num1-1)
car1_num = len(spring1.data[0][car1])

if car1_num >=3:
    pos = random.randint(1, car1_num-2)
    s_task_id = spring1.data[0][car1][pos:pos+2]
    w_task_id = spring1.data[1][car1][pos:pos+2]

    loc = []
    s = set()
    for i in range(car_num2):
        work_list = spring2.data[1][i]
        for j,w_i in enumerate(work_list):
            if w_i in w_task_id and w_i not in s:
                loc.append((i, j))
                s.add(w_i)

    cur1 = pos
    task_cur = 0
    for car_id, work_id in loc:
        spring1.data[0][car1][cur1] =
spring2.data[0][car_id][work_id]
        spring1.data[1][car1][cur1] =
spring2.data[1][car_id][work_id]

        spring2.data[0][car_id][work_id] = s_task_id[task_cur]
        spring2.data[1][car_id][work_id] = w_task_id[task_cur]

    cur1 += 1
    task_cur += 1

return spring1, spring2

def mutation(self, crossover):
    muteoff = crossover

```

```

car_num = len(crossoff.data[0])

car1 = random.randint(0, car_num-1)
car2 = random.randint(0, car_num-1)

if      len(muteoff.data[0][car1])      >      1      and
len(muteoff.data[0][car2]) > 1:#两辆车之间的交换
    pos1 = random.randint(1, len(muteoff.data[0][car1])-1)
    pos2 = random.randint(1, len(muteoff.data[0][car2])-1)

    tmp_s = muteoff.data[0][car1][pos1]
    tmp_w = muteoff.data[1][car1][pos1]

    muteoff.data[0][car1][pos1] = muteoff.data[0][car2][pos2]
    muteoff.data[1][car1][pos1] = muteoff.data[1][car2][pos2]

    muteoff.data[0][car2][pos2] = tmp_s
    muteoff.data[1][car2][pos2] = tmp_w

elif      len(muteoff.data[0][car1])      >      1      or
len(muteoff.data[0][car2]) > 1: # 多的车向少的车变换
    car_more = car1 if len(muteoff.data[0][car1]) > 1 else car2
    car_less = car1 + car2 - car_more

    pos = random.randint(1, len(muteoff.data[0][car_more])-1)
    tmp_s = muteoff.data[0][car_more][pos]
    tmp_w = muteoff.data[1][car_more][pos]

    muteoff.data[0][car_more].pop(pos)
    muteoff.data[1][car_more].pop(pos)

    muteoff.data[0][car_less].append(tmp_s)
    muteoff.data[1][car_less].append(tmp_w)

if 0<=random.random() <= 0.5:

```

```

        car3 = random.randint(0, car_num-1)
        if len(muteoff.data[0][car3])>=3:
            pos1 = random.randint(1, len(muteoff.data[0][car3])-1)
            pos2 = random.randint(1, len(muteoff.data[0][car3])-1)

            tmp_s = muteoff.data[0][car3][pos1]
            tmp_w = muteoff.data[1][car3][pos1]

            muteoff.data[0][car3][pos1] =
muteoff.data[0][car3][pos2]
            muteoff.data[1][car3][pos1] =
muteoff.data[1][car3][pos2]

            muteoff.data[0][car3][pos2] = tmp_s
            muteoff.data[1][car3][pos2] = tmp_w

        return muteoff
    
```

```

def GA_main(self, select_num = 100):

    popsize = self.pop_size
    for step in range(self.epoch):

        selectpop = self.selection(self.pop, select_num)

        nextoff = []
        while len(nextoff) != popsize:
            if len(selectpop) <2:
                break
            offspring = [selectpop.pop() for _ in range(2)]

            if random.random() < self.CXPB:
                crossoff1, crossoff2 =

```

```

        self.crossover(offspring)

        if random.random() < self.MUTPB:
            muteoff1 = self.mutation(crossover1)
            muteoff2 = self.mutation(crossover2)
            fit_muteoff1 = self.evaluate(muteoff1)
            fit_muteoff2 = self.evaluate(muteoff2)

            nextoff.append({'Gene':muteoff1, 'score':fit_muteoff1})
            nextoff.append({'Gene': muteoff2, 'score':
            fit_muteoff2})

        else:
            fit_crossover1 = self.evaluate(crossover1)
            fit_crossover2 = self.evaluate(crossover2)
            nextoff.append({'Gene': crossover1, 'score':
            fit_crossover1})
            nextoff.append({'Gene': crossover2, 'score':
            fit_crossover2})

        else:
            nextoff.extend(offspring)

        self.pop = nextoff

    fits = [ind['score'] for ind in self.pop]

    best_ind = self.selectBest(self.pop)

    if best_ind['score'] < self.bestindividual['score']:
        self.bestindividual['score'] = best_ind['score']
        self.bestindividual['Gene'] = best_ind['Gene'].copy()

    print("Best individual found is {},\n{} ".format(self.bestindividual['Gene'].data,
    self.bestindividual['score']))

    print(" Min fitness of current pop: {} ".format(min(fits)))

```

```

if __name__ == '__main__':
    diss3, matrix3 = cal_dis3(node)
    diss2, matrix2 = cal_dis2(node)
    diss1, matrix1 = cal_dis1(node)
    matrix1to2 = l2l1(node)
    diss_total = np.zeros(72)
    diss_total = diss1+diss2+diss3

    matrix = np.zeros((73, 73))
    for i in range(1, 25):
        for j in range(25, 73):
            matrix[i][j]=matrix[j][i] = diss_total[i] + diss_total[j]
    matrix = matrix + matrix3 + matrix1 + matrix2 + matrix1to2
    print(diss3,diss2,diss1)
    print(diss_total)
    print(matrix)
    print(matrix[30][47],matrix[47][30],diss_total[1],diss_total[12])

#ax = sns.heatmap(matrix,center=0)
#plt.show()
dic = {}
for i in range(1, 25):
    j = 24 + i
    jj = 48 + i
    li = dic.get(matrix[i][j], 0)
    li += 1
    dic[matrix[i][j]] = li
    li = dic.get(matrix[i][jj], 0)
    li += 1
    dic[matrix[i][jj]] = li

```

```

print(i, j, ':', matrix[i][j])
print(i, jj, ':', matrix[i][jj])

#plt.bar(list(dic.keys()), list(dic.values()))
#plt.show()
cal_distance(matrix)

disB = calB(node)
print(disB)

# task = [(27, 112), (28, 130), (32, 150), (31, 167),
#          (59, 122), (60, 144), (63, 158), (64, 171)]

# task = [(37, 60), (31, 62), (45, 65), (36, 72),
#          (40, 80), (33, 82), (46, 85), (42, 100),
#          (40, 110), (53, 55), (63, 58), (51, 94),
#          (65, 96), (67, 100), (56, 120), (70, 121),
#          (72, 123), (53, 140), (59, 150)
#          ]

task = [(46, 51),
         (41, 53),
         (38, 55),
         (45, 62),
         (36, 67),
         (43, 69),
         (29, 74),
         (25, 79),
         (37, 80),
         (42, 85),
         (27, 89),
         (31, 93),
         (30, 97),
         (56, 48),
         ]

```

```
(57, 50),  
(67, 55),  
(49, 58),  
(70, 63),  
(65, 67),  
(60, 68),  
(50, 69),  
(72, 72),  
(61, 78)]  
  
parameter = [10000, 10000, 0.8, 0.1]  
  
ga = GenA(parameter, matrix, disB, task, 6, diss_total)  
ga.GA_main(select_num=1000)
```