《高级运筹学》课程大作业

ICAPS 2021-The Dynamic Pickup and Delivery Problem

动态取送货问题研究

组员：

纪子阳　2024214062

谢奕飞　2024215060

王勇　2024215061

王涛　2024215065

杨惠如　2024215066

李婧　2024215068

康修瑜　2024215080

授课教师：　张灿荣 教授

2024 年 12 月 25 日

目录

一、 绪论 1
1.1 问题描述 1
1.1.1 问题背景 1
1.1.2 输入 1
1.1.3 输出 2
1.1.4 约束条件 2
1.1.5 目标 3
1.2 算法介绍 3
二、 模型建立 5
2.1 符号说明 5
2.2 建立数学模型 5
2.3 模型解释 7
目标函数 7
约束条件 7
三、 算法设计 10
3.1 遗传算法 10
3.1.1 算法流程 10
3.1.2 约束实现 11
3.2 基于贪心的仿真方法 13
3.2.1 算法流程 13
3.2.2 调用框架 13
四、 结果分析 15
参考文献 18

一、绪论

1.1 问题描述

1.1.1 问题背景

随着经济的高速发展以及信息技术的进步，人们对物流调度的成本以及效率提出了更高的要求，如何提高优化运输路径，提高客户的满意度成为当下的研究热点。动态取送货问题(Dynamic Pickup and Delivery Problem, DPDP)综合考虑了取货和送货双向物流过程，并且能够实时处理订单，在物流优化领域具有显著的优越性，能够为物流行业带来多方面的改进和创新。

华为是全球最大的通信设备供应商之一，每年在数百家工厂生产数十亿种产品。在生产过程中，大量货物(包括原材料、成品和半成品)需要在工厂之间进行运输。由于客户需求和生产过程的不确定性，大部分交付要求不能完全提前确定。交付订单，包括提货工厂、交货工厂、货物数量和时间要求等信息，它是随机产生的，并定期安排一个同质车辆车队为这些订单提供服务。由于交付的请求量巨大，即使是物流效率的微小提高，也可以带来显著的财务效益。因此，开发一种有效的优化算法来调度订单和规划车辆路线具有重要意义。

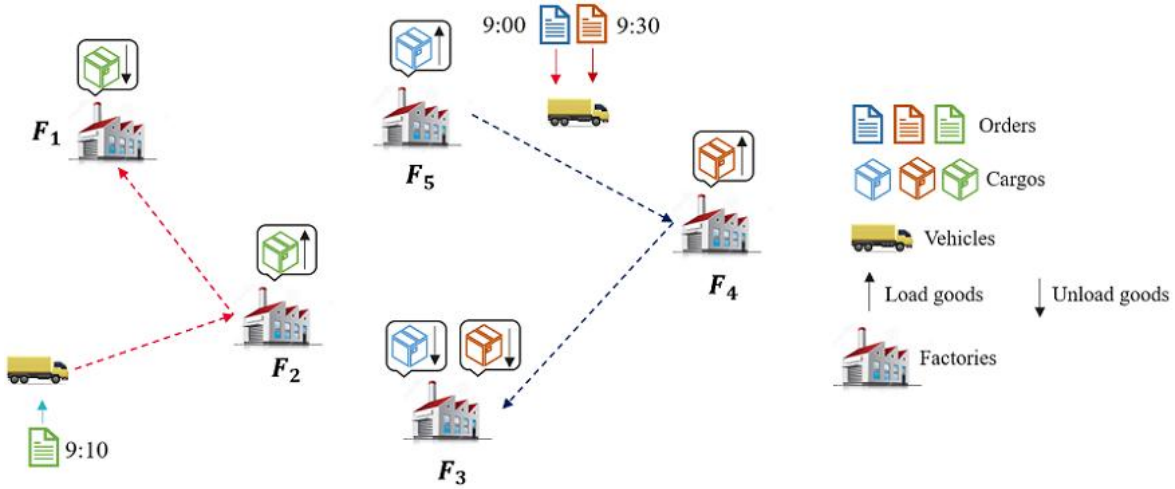

图 1.1 DPDP 的例子

DPDP 的目标是调度车辆服务动态生成的订单，通过将所需货物从起点运输到目的地。其目标是最小化订单的超时和车辆的平均行驶距离。以图 1 为例，一个订单在上午 9:00 生成，包含需要从 F5 运送到 F4 的货物。

1.1.2 输入

- 一张道路网络$\mathrm{G}=(F,A)$，其中$\mathrm{F}=\{F_i|\mathrm{i}=1,\ldots,\mathrm{M}\}$为工厂的点集合，$\mathrm{A}=\{(i,j)\mid i,\mathrm{j}\in M\}$是弧集。每条弧$(i,j)$关联一个非负的运输距离$di$和从节点$i$到节点$j$的运输时间$t_{ij}$。
- 一个订单集$\mathrm{O}=\{o_i\mid i=1,\cdots,\mathrm{N}\}$，其中每个订单$o_i=\left(F_p^i,F_d^i,q^i,t_e^i,t_l^i\right)$（这些订单是随机到达的），其中$F_p^i$和$F_d^i$分别表示取货节点和送货节点，$q^i=\left(q_{standard}^i,q_{small}^i,q_{box}^i\right)$是需要交付的货物，$q_{standard}^i$是标准托盘的数量，$q_{small}^i$是小托盘的数量，$q_{box}^i$是箱子的数量。$t_e^i$是订单的创建时间，$t_l^i$承诺完成时间（承诺完成时间包括车辆行驶时间、码头靠近时间、码头排队时间以及装卸时间）。1 个标准托盘等于 2 个小型托盘，1 个小型托盘等于 2 个箱子。
- 一个同质车辆车队$\mathrm{V}=\{\mathrm{v_k}|\mathrm{k}=1,\cdots,\mathrm{K}\}$，每辆车$v_k$都与装载能力和司机工作班次相关联。车辆的初始位置被随机分配给工厂。
- M个节点（工厂）$\{F_i|i=1,\cdots,M\}$，每个节点只有有限数量的货物码头供装卸，以及工作班次约束。如果所有码头都在使用，则新到的车辆必须等待放行。而且装卸作业只能在工作班次时进行。
- 码头靠近时间，即不考虑排队情况下的车辆到达工厂到分配码头之间的时间。
- 装卸时间，如车上有q个标准托盘，则装货时间$t_p=w\times q$，卸货时间$t_d=w\times q$，,w=180 秒/个标准托盘。

为了简化这个问题，我们假设车辆和工厂没有工作班次限制，也就是说他们一天 24 小时工作。

1.1.3 输出

模型的输出主要分为以下两个方面：1、订单的分配方案，即把订单分配给哪辆车承担运输任务；2、车辆的路径规划，即规划每辆车的运行路径。

1.1.4 约束条件

1. 订单履行约束：所有订单都要实现。
2. 承诺完成时间约束：订单必须在承诺的完成时间内完成，例如 4 小时。否则将发生罚款。
3. 订单拆分约束：只有当订单超过车辆的装载能力时，才允许拆分。订单的最小单位不允许拆分。例如，如果一个订单包含 13 个标准货盘、7 个小货盘和 1 个箱子，那么最小的单位是 1 个标准货盘、1 个小货盘或 1 个箱子。
4. 车辆装载能力约束：对于每辆车 v_k，装载货物的总容量不能超过最大装载能力 Q，例如每辆车最多可装载 12 个标准货盘。
5. 司机排班约束：例如，8:30-12:00，13:30-18:00。
6. 后进先出装载约束：例如，如果订单 o_1 和 o_2 被分配给同一辆车 v_k，其中 o_1 的提货

和交货节点分别为F_p^1和F_d^1，而o_2的提货和交货节点分别为F_p^2和F_d^2，那么路线计划$\{F_p^1, F_p^2, F_d^1, F_d^2\}$违反了 LIFO 约束，而$\{F_p^1, F_p^2, F_d^2, F_d^1\}$不违反。

7. 每个节点的货物码头数量有限：例如，如果节点有 3 个货物码头，4 辆车到达，则最后一辆车必须等待直到有一个码头空闲。
8. 装卸遵循先到先服务规则：如果多个车辆同时到达同一节点，而该节点只有一个空闲货物码头则节点将随机选择一辆车进行装卸。注意，以上情况在现实中不会发生，但在模拟环境中可能会出现。

1.1.5 目标

该问题包含两个目标。第一个目标是最小化订单的总超时，记为f_1。如果订单o_i到达目的地的时间为a_d^i，则

$$f_1 = \sum_{i=1}^{N} max(0, a_d^i - t_l^i)$$

其中t_l^i是承诺完成时间，N是订单总数。

第二个目标是最小化车辆的平均行驶距离，记为f_2。如果车辆v_k的路线计划为$\prod_k = \{n_1^k, n_2^k, \dots, n_{l_k}^k\}$，其中$n_i^k$代表第$i$个节点，即车辆的路线中的工厂，$l_k$是车辆$k$行驶的总节点数。那么，

$$f_2 = \frac{1}{K} \sum_{k=1}^{K} \sum_{i=1}^{l_k - 1} d_{n_i^k, n_{i+1}^k}$$

其中$d_{n_i^k, n_{i+1}^k}$是从节点n_i^k到节点n_{i+1}^k的距离。

整体目标为：

$$minf = \lambda \times f_1 + f_2$$

其中λ是一个较大的正数常量。

1.2 算法介绍

DPDP 是车辆路径问题（VRP）的动态扩展，其核心任务是通过动态调度车辆以满足实时生成的取送订单需求，同时最小化运输成本和服务时间。未来订单的不可预测性使得传统优化方法难以应对 DPDP 问题，时间窗、容量限制、回仓约束和装载原则限值以及工业规模的应用场景中订单和车辆数量的激增都进一步加剧了求解的难度。

DPDP 的研究方法主要分为传统优化方法、启发式方法和基于强化学习的方法。

传统方法主要基于运筹学原理，通常通过将动态问题分解为多个静态问题求解。Savelsbergh 等提出的动态路径规划算法，将动态问题分解为一系列静态问题，在滚动时间窗内优化每个子问题[1]。这类方法通过混合整数规划（MIP）等手段求解每个子问

题。由于动态订单的不确定性以及子问题间可能的相互影响，这类方法难以适应实时需求。

启发式方法因其计算速度快、适应性强而被广泛应用于 DPDP。Gendreau 等设计了基于禁忌搜索的算法，通过使用逐步优化的邻域结构改善解的质量[2]。Minic 等提出了两阶段启发式算法，第一阶段采用最便宜插入策略将订单插入到现有路径中，第二阶段利用禁忌搜索进一步优化路径[3]。Saez 等结合模糊聚类和遗传算法开发了自适应预测控制策略[4]。该方法通过提前预测需求，优化调度策略。Pureza 和 Laporte 提出了一种结合等待和缓冲策略的启发式方法，可以有效减少车辆使用数量或行驶距离[5]。启发式方法易陷入局部最优，且难以获得全局最优解。特别是当问题规模较大或动态性增强时，其效果可能不理想。

随着人工智能技术的兴起，强化学习（RL）方法因其数据驱动特性和动态适应能力而逐渐受到关注。Khalil 等通过强化学习框架解决旅行商问题（TSP）和车辆路径问题（VRP），验证了 RL 方法在动态环境中的潜力[6]。Zambaldi 等提出了基于图卷积网络的强化学习方法，用于捕捉系统中实体之间的关系[7]。Jiang 等将这一框架扩展到多智能体强化学习，用于学习智能体间的交互关系[8]。Li 等提出了 ST-DDGN 模型结合了空间时间分布预测和注意力机制，显著提高了 DPDP 的求解性能[9]。RL 方法可以动态调整策略以适应实时变化，但其训练时间较长，对计算资源需求较高。此外，算法的泛化性能在复杂工业场景中仍需进一步验证。

二、模型建立

2.1 符号说明

变量类型	符号	描述
决策变量	x_{ij}^{ok}	0-1 变量，1：车辆k搭载订单o经过边ij
	y_{ij}^{k}	0-1 变量，1：车辆k经过边ij
	a_{ok}	0-1 变量，1：订单o被分配给车辆k
图信息	F_i	工厂i的码头集合
	V_1	当前车辆所在的位置集合
	V_2	有需求的点（取货或送货）的集合
	V	$V_1 \cup V_2$
辅助变量	t_{ij}	节点i与节点j之间的行驶时间
	c_{ij}	节点i与节点j之间的行驶距离
	T_i^k	车辆k到达工厂i的时间（已经包含 approaching time）
	W_i^k	车辆k在工厂i的等待时间
	$f_m(k)$	车辆k到达工厂时，码头m的前序车辆
	$l(o)$	不可重新规划的订单o被分配的车辆
	q_i^k	车辆k离开节点i时的载重

2.2 建立数学模型

动态建模思路如下：

每隔十分钟调用该优化算法进行规划。

调用算法时输入：

① 车辆信息：

当前位置的状态（行驶中、工厂中）

② 订单信息：订单总集合O，将订单分为可被规划（新产生的订单以及可被重规划的订单，用集合O_s表示）与不可被规划（已经装车的订单与还未装车的订单，针对已经装车的订单，将其订单起点更新为当前车辆的位置）。针对订单o，其五元组信息$(F_p^o, F_d^o, d_o, t_e^o, t_l^o)$分别表示该订单的起始节点、终止节点、需求量、订单产生时间、订单承诺完成时间。s_o表示订单的装载时间与卸载时间。

建立数学模型如下：

$$\min\left(\lambda\sum_{k\in K}\sum_{i\in V_2}\xi_i^k+\frac{1}{k}\sum_{k\in K}\sum_{i,j\in V}c_{ij}y_{ij}^k\right)$$

$$\sum_{\substack{j\in V\\ j\neq F_p^o}} x_{F_p^o j}^{o\,l(o)} = 1, \qquad \forall o \in O_f \tag{1}$$

$$\sum_{\substack{i\in V\\ i\neq F_d^o}} x_{iF_d^o}^{o\,l(o)} = 1, \qquad \forall o \in O_f \tag{2}$$

$$\sum_{\substack{i\in V\\ i\neq F_p^o}} x_{iF_p^o}^{ok} = \sum_{\substack{j\in V\\ j\neq F_p^o}} x_{F_p^o j}^{ok} - a_{ok}, \qquad \forall o \in O_s, k \in K \tag{3}$$

$$\sum_{\substack{i\in V\\ i\neq F_d^o}} x_{iF_d^o}^{ok} = \sum_{\substack{j\in V\\ j\neq F_d^o}} x_{F_d^o j}^{ok} + a_{ok}, \qquad \forall o \in O_s, k \in K \tag{4}$$

$$\sum_{k\in K}\sum_{\substack{j\in V\\ j\neq F_p^o}} x_{F_p^o j}^{ok} \leq 1, \qquad \forall o \in O_s \tag{5}$$

$$\sum_{k\in K}\sum_{\substack{i\in V\\ i\neq F_d^o}} x_{iF_d^o}^{ok} \leq 1, \qquad \forall o \in O_s \tag{6}$$

$$\sum_{o\in O} x_{ij}^{ok}\, d_o \leq C y_{ij}^k, \quad \forall i,j \in V, k \in K \tag{7}$$

$$T_j^k \geq T_i^k + W_i^k + \sum_{o\in\left\{o\middle|F_p^o = i \text{ or } F_d^o = i\right\}} a_{ok}s_o + t_{ij} - M\left(1 - y_{ij}^k\right), \forall i,j \in V, k \in K \tag{8}$$

$$T_i^k + W_i^k + M\left(1 - \sum_{j\in V} y_{ji}^k\right) \geq \max_{o\in\left\{o\middle|F_p^o = i, a_{ok} = 1\right\}} t_e^o, \quad \forall i \in V_2, k \in K \tag{9}$$

$$W_i^k \geq \max\left\{\min_{m\in F_i}\left\{T_i^{f_m(k)} + \sum_{o\in\left\{o\middle|F_p^o = i \text{ or } F_d^o = i\right\}} a_{of_m(k)}s_o\right\} - T_i^k, 0\right\}, \forall i \in V_2, k \in K \tag{10}$$

$$\xi_i^k \geq \max\left\{0, T_i^k + W_i^k + \sum_{o\in\left\{o\middle|F_d^o = i\right\}} a_{ok}s_o - \min_{o\in\left\{o\middle|F_d^o = i, a_{ok} = 1\right\}} t_l^o\right\}, \forall i \in V_2, k \in K \tag{11}$$

$$q_{F_p^o}^k - q_{F_d^o}^k \geq d_o, \quad \forall o \in O, k \in K \tag{12}$$

$$\sum_{j\in V} y_{ij}^k \leq 1, \ \ \forall i \in V, K \in K \tag{13}$$

$$\sum_{j\in V} y_{ji}^k \leq 1, \ \ \forall i \in V, K \in K \tag{14}$$

$$\sum_{k\in K} a_{ok} = 1, \ \ \forall o \in O \tag{15}$$

$$\sum_{j\in V} y_{ij}^k \leq \sum_{j\in V} y_{ji}^k, \ \ \forall i \in V, K \in K \tag{16}$$

$$\sum_{j\in V} x_{ij}^{ok} = \sum_{j\in V} x_{ji}^{ok}, \ \ \forall i \in V - F_p^o - F_d^o \tag{17}$$

2.3 模型解释

目标函数

目标函数：最小化订单总超时及车辆的平均行驶距离。

$$\min\left(\lambda\sum_{k\in K}\sum_{i\in V_2}\xi_i^k+\frac{1}{k}\sum_{k\in K}\sum_{i,j\in V}c_{ij}y_{ij}^k\right)$$

约束条件

针对已经不可被规划的订单：

$$\sum_{\substack{j\in V\\ j\neq F_p^o}}x_{F_p^o j}^{o\,l(o)}=1,\qquad \forall o\in O_f \tag{1}$$

$$\sum_{\substack{i\in V\\ i\neq F_d^o}}x_{iF_d^o}^{o\,l(o)}=1,\qquad \forall o\in O_f \tag{2}$$

（1）保证订单能够跟随相应的车辆进行移动；（2）保证订单均能搭载相应的车辆到达终点。

针对可被规划的订单：

$$\sum_{\substack{i\in V\\ i\neq F_p^o}}x_{iF_p^o}^{ok}=\sum_{\substack{j\in V\\ j\neq F_p^o}}x_{F_p^o j}^{ok}-a_{ok},\qquad \forall o\in O_s,k\in K \tag{3}$$

$$\sum_{\substack{i\in V\\ i\neq F_d^o}}x_{iF_d^o}^{ok}=\sum_{\substack{j\in V\\ j\neq F_d^o}}x_{F_d^o j}^{ok}+a_{ok},\qquad \forall o\in O_s,k\in K \tag{4}$$

$$\sum_{k\in K}\sum_{\substack{j\in V\\ j\neq F_p^o}}x_{F_p^o j}^{ok}\leq 1,\qquad \forall o\in O_s \tag{5}$$

$$\sum_{k\in K}\sum_{\substack{i\in V\\ i\neq F_d^o}}x_{iF_d^o}^{ok}\leq 1,\qquad \forall o\in O_s \tag{6}$$

（3）保证订单能够在起点被分配的车辆搭载。

例如，若$a_{ok}=1$，表示订单o被分配给车辆k，x_{ij}^{ok}均为 0-1 变量，要使等式成立，则针对车辆k，$\sum_{\substack{j\in V\\ j\neq F_p^o}}x_{F_p^o j}^{ok}=1$，$\sum_{\substack{i\in V\\ i\neq F_p^o}}x_{iF_p^o}^{ok}=0$，表示订单$o$将从起点开始跟随分配的车辆$k$一起移动。

（4）保证订单能够被分配的车辆送往终点，其逻辑与（3）相同。

（5）、（6）保证订单只能跟随一辆车移动。

容量约束：

$$\sum_{o\in O} x_{ij}^{ok}\, d_o \le C y_{ij}^{k}, \quad \forall i,j \in V, k \in K \tag{7}$$

（7）保证车辆在任意一条道路上行驶时，装载的订单量总和小于其最大载重。且该条约束包含了x_{ij}^{ok}与y_{ij}^{k}之间的逻辑关系：若$y_{ij}^{k}=0$，则x_{ij}^{ok}必须为 0。即只有车辆k会经过边(i,j)时，订单才能被搭载在车辆k上经过边(i,j)。

时间窗约束：（假设旅行时间中包含对应的 approaching time）

$$T_j^k \ge T_i^k + W_i^k + \sum_{o\in\{o|F_p^o = i \text{ or } F_d^o = i\}} a_{ok}s_o + t_{ij} - M\left(1 - y_{ij}^k\right), \forall i,j \in V, k \in K \tag{8}$$

$$T_i^k + W_i^k + M\left(1 - \sum_{j\in V} y_{ji}^k\right) \ge \max_{o\in\{o|F_p^o = i, a_{ok}=1\}} t_e^o, \quad \forall i \in V_2, k \in K \tag{9}$$

$$W_i^k \ge \max\left\{\min_{m\in F_i}\left\{T_i^{f_m(k)} + \sum_{o\in\{o|F_p^o = i \text{ or } F_d^o = i\}} a_{of_m(k)}s_o\right\} - T_i^k, 0\right\}, \forall i \in V_2, k \in K \tag{10}$$

$$\xi_i^k \ge \max\left\{0, T_i^k + W_i^k + \sum_{o\in\{o|F_d^o = i\}} a_{ok}s_o - \min_{o\in\{o|F_d^o = i, a_{ok}=1\}} t_l^o\right\}, \forall i \in V_2, k \in K \tag{11}$$

（8）为到达时间约束，当$y_{ij}^k = 1$时，即表示车辆k将从节点i前往节点j，则其到达j的时间应当大于等于车辆k到达i的时间+在节点i的等待时间W_i^k+在节点i的总服务时间$\sum_{o\in\{o|F_p^o = i \text{ or } F_d^o = i\}} a_{ok}s_o$。

（9）表示车辆k到达节点i后开始服务的时间应当晚于订单的产生时间

（10）是对等待时间的核算。车辆k到达节点i后，将会在其码头集合F_i中选择最早空闲的码头开始服务。

（11）是对车辆k在节点i的订单总超时核算。为简化模型，此处对准确的订单总超时进行了鲁棒计算。

后进先出约束：

$$q_{F_p^o}^k - q_{F_d^o}^k \ge d_o, \quad \forall o \in O, k \in K \tag{12}$$

式（12）为后进先出约束。

逻辑约束：

$$\sum_{j\in V} y_{ij}^{k} \le 1,\ \ \forall i \in V, K \in K \tag{13}$$

$$\sum_{j\in V} y_{ji}^{k} \le 1,\ \ \forall i \in V, K \in K \tag{14}$$

$$\sum_{k\in K} a_{ok} = 1,\ \ \forall o \in O \tag{15}$$

式（13）、（14）表示针对每个节点，每辆车最多只能访问一次且只能前往一个节点；

式（15）表示订单o必须且只能被分配给某一辆车。

流平衡约束：

$$\sum_{j\in V} y_{ij}^{k} \le \sum_{j\in V} y_{ji}^{k},\ \ \forall i \in V, K \in K \tag{16}$$

$$\sum_{j\in V} x_{ij}^{ok} = \sum_{j\in V} x_{ji}^{ok},\ \ \forall i \in V - F_p^o - F_d^o \tag{17}$$

式（16）表示到达节点i的车辆可被发出或停止；式（17）表示订单被搭载在车辆上时，必须跟随车辆移动直至订单终点。

三、算法设计

3.1 遗传算法

3.1.1 算法流程

1. 初始化种群:

使用 initialize_population 函数随机生成初始种群，每个个体代表一种车辆分配方案。

2. 预计算与数据准备:

提取订单的取货点和送货点 ID，并将其转换为对应的索引。

初始化工厂端口状态矩阵，假设每个工厂有 6 个端口。

3. 适应度函数准备:

使用 functools.partial 固定适应度函数的部分参数，简化多进程调用。

4. 多进程池创建:

使用 multiprocessing.Pool 创建一个进程池，利用所有可用 CPU 核心加速适应度评估。

5. 遗传算法迭代:

- 适应度评估: 并行计算种群中每个个体的适应度。
- 统计信息输出: 打印当前代的最佳适应度、平均适应度和最少晚点订单数。
- 选择操作: 使用 select_population 函数选择优秀个体，保留精英。
- 交叉与变异:
  - ✓ 通过随机选择父代，进行交叉生成子代。
  - ✓ 对生成的子代应用变异操作，增加种群多样性。
  - ✓ 确保生成的子代数量满足种群规模要求。
- 种群更新: 将子代加入种群，准备进入下一代迭代。

6. 最终适应度评估与种群关闭:

- 在所有代迭代完成后，进行最终的适应度评估。
- 关闭多进程池，释放资源。

7. 获取最佳解:

- 从最终种群中选择适应度最优的个体作为最佳解决方案。
- 输出最优方案的总路程、总惩罚成本和晚点订单数。

8. 获取计算环境信息:

使用 psutil 和 platform 库获取当前运行环境的详细信息，包括编程语言版本、操作系统、CPU 信息、内存等。

3.1.2 约束实现

表 3-2 约束清单总结

约束名称	描述	代码逻辑
车辆载重限制	每辆车的总载重量不能超过车辆的最大载重。	如果超载, 返回无效解。
订单的承诺完成时间限制	每个订单必须在承诺完成时间前完成, 否则产生超时惩罚。	超时部分乘以惩罚系数累加到总惩罚中。
车辆等待订单生成时间	如果车辆到达取货点早于订单生成时间, 则车辆必须等待。	当前时间小于订单生成时间时累加等待时间。
工厂装卸货港口数量限制	每个工厂同时最多允许 6 辆车装卸货操作, 超过时车辆需等待空闲港口。	使用 factory_ports_state 跟踪每个工厂的港口状态。
装卸货时间限制	每个订单有固定的装货和卸货时间, 需要计入车辆总时间。	累加 load_time 和 unload_time 到车辆的当前时间。
工厂之间的运输时间限制	工厂之间的运输时间由时间矩阵决定, 需要计入车辆总时间。	累加 time_matrix 中的运输时间到车辆总时间。
距离成本限制	每辆车的运输距离会被累加到总距离中, 作为优化目标之一。	从当前位置到取货点、从取货点到送货点的距离被累加。
每个订单必须被分配	每个订单必须被分配给一辆车, 并且按分配顺序完成。	在种群初始化时,确保每个订单都有车辆分配。

1. 时间窗约束

描述：

每个订单都有两个关键时间点：

- 创建时间 (creation_time)：订单被创建的时间，即车辆可以开始处理该订单的最早时间。
- 承诺完成时间 (committed_completion_time)：订单必须完成配送的最晚时间。

实现方式：

在适应度函数 calculate_fitness 中，通过以下逻辑处理时间窗约束：

1) 针对不能提前处理订单：

车辆在订单创建时间之前不能开始处理该订单。如果车辆的当前时间小于订单的创建时间，车辆需要等待，等待时间被累加到 total_waiting_time 中，并更新车辆的当前时间。

```
if vehicle_current_time[vehicle_id] < creation_time:
    total_waiting_time += creation_time - vehicle_current_time[vehicle_id]
    vehicle_current_time[vehicle_id] = creation_time
```

2) 在完成订单配送后，检查车辆的当前时间是否超过了订单的承诺完成时间。

3) 如果超时，计算延迟时间 (delay)，并根据每秒延迟的惩罚系数(penalty_per_second)计算延迟成本，将其累加到总惩罚 (penalty) 中。同时，晚点订单计数 (late_orders) 也会增加。

```
if vehicle_current_time[vehicle_id] > committed_completion_time:
    delay = vehicle_current_time[vehicle_id] - committed_completion_time
    penalty += delay * penalty_per_second
    late_orders += 1
```

2. 工厂端口可用性

描述：

每个工厂拥有多个端口（在代码中设定为 6 个），每个端口一次只能处理一辆车。车辆到达工厂时需要等待端口可用。

```
factory_ports_state_init = np.zeros((dist_matrix.shape[0], 6))
```

实现方式：

1) 在适应度函数中处理端口可用性：
当车辆到达取货或送货工厂时，检查工厂端口的最早可用时(earliest_port_time)

```
port_times = factory_ports_state[pickup_id]
earliest_port_time = np.min(port_times)

if vehicle_current_time[vehicle_id] < earliest_port_time:
    total_waiting_time += earliest_port_time - vehicle_current_time[vehicle_id]
    vehicle_current_time[vehicle_id] = earliest_port_time
```

2) 选择最早可用的端口：
使用 np.argmin(port_times) 找到最早可用的端口，并更新该端口的可用时间。

```
port_index = np.argmin(port_times)
factory_ports_state[pickup_id, port_index] = vehicle_current_time[vehicle_id]
```

3) 计算行驶距离：
在适应度函数中，如果车辆之前没有执行任何订单（即 vehicle_current_location [vehicle_id] 为 None），则假设车辆从仓库出发。

```
# 计算行驶距离
if vehicle_current_location[vehicle_id] is not None:
    travel_distance = dist_matrix[vehicle_current_location[vehicle_id], pickup_id] + dist_matrix[pickup_id, delivery_id]
else:
    # 假设车辆从仓库出发，仓库ID为0（需要根据实际情况调整）
    travel_distance = dist_matrix[pickup_id, delivery_id]
total_distance += travel_distance
vehicle_current_location[vehicle_id] = delivery_id
```

3.2 基于贪心的仿真方法

3.2.1 算法流程

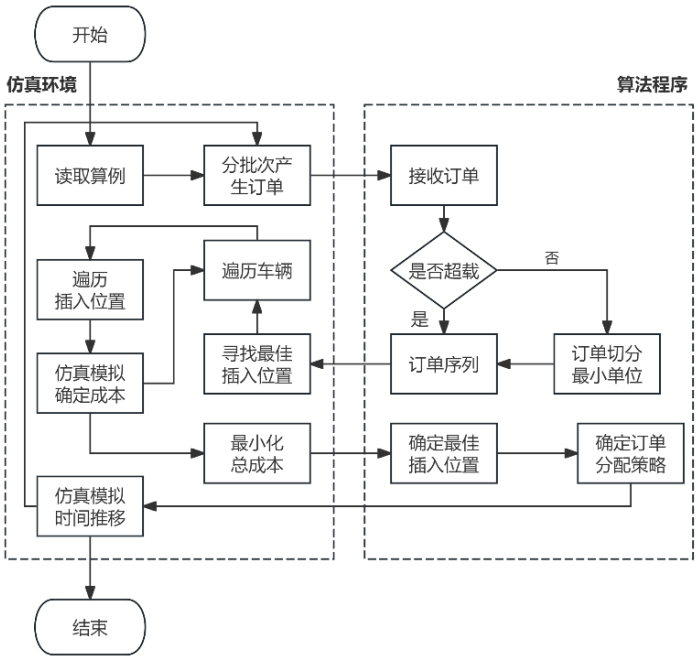

图 3-1 贪心算法流程图

3.2.2 调用框架

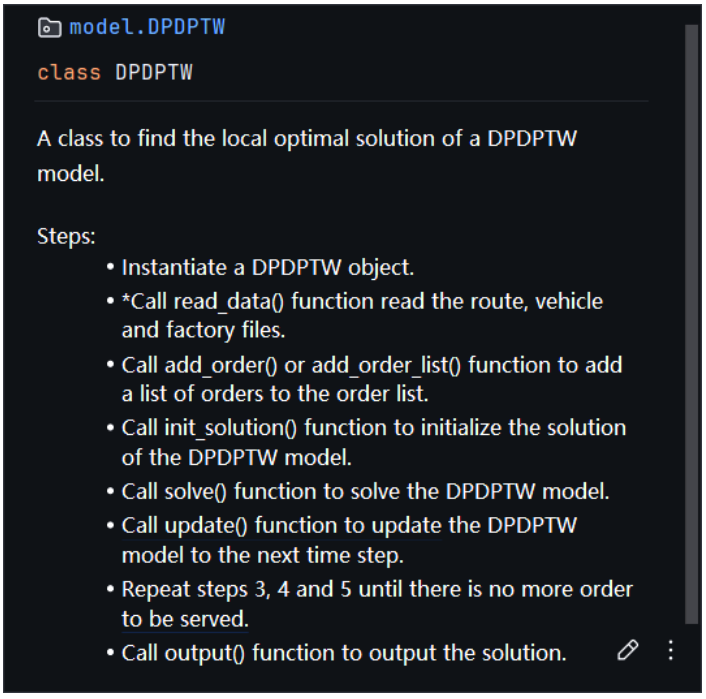

图 3-2 算法调用步骤

- **实例化对象：**创建一个 DPDPTW 对象，用于管理数据和方法调用。
- **数据读取 (read_data)：**调用 read_data()方法，加载包含路线、车辆、工厂等信息的输入文件。
- **添加订单 (add_order)：**使用 add_order()或 add_order_list()方法，将订单添加到系统的订单列表中。
- **初始化解 (init_solution)：**调用 init_solution()方法，初始化模型的求解过程。
- **求解 (solve)：**通过 solve()方法，利用内置算法生成当前时间步的优化解。
- **更新模型 (update)：**调用 update()方法，将模型推进到下一个时间步，同时考虑新订单或其他动态变化。
- **循环执行：**重复调用 solve()和 update()直至没有订单需要处理。
- **输出结果 (output)：**最后通过 output()方法导出优化结果。

表 3-1 更新信息总结

状态	情况	更新货物情况	更新订单总超时	更新任务列表	记录历史任务、路径、时间信息	更新到达下一状态时间	备注
初始状态	已被分配任务	×	×	√	×	√	
	未被分配任务	--	--	--	--	--	进入空闲
中间状态	到达节点（工厂）	×	×	×	√	√	
	开始服务	×	×	×	√	√	
	离开节点（仍有任务）	√	√×	√	√	√	
	离开节点（已无任务）	√	√	√	√	×	进入空闲

四、结果分析

针对前 20 个算例，分别用遗传算法、基于贪心的仿真方法以及比赛银奖代码进行求解，并记录各个算例中车辆行驶的总距离、超时时间和算法的得分。

其中，得分=车辆平均行驶距离+超时时间*10000/3600。得分越小，模型的效果越好。遗传算法的求解结果如下：

表 4-1 遗传算法得分表

算例	总距离(km)	超时时间(s)	得分	算例	总距离(km)	超时时间(s)	得分
1	746.30	0	149.26	11	1307.40	0	261.48
2	528.10	0	105.62	12	1248.00	0	249.60
3	619.40	0	123.88	13	1349.00	0	269.80
4	673.70	0	134.74	14	1467.90	0	293.58
5	624.80	0	124.96	15	1562.30	0	312.46
6	732.70	0	146.54	16	1177.60	0	235.52
7	660.70	0	132.14	17	3969.1	0	198.46
8	474.10	0	94.82	18	4021.5	0	201.08
9	1319.20	0	263.84	19	4376.30	0	218.82
10	1383.80	0	276.76	20	2008.3	0	200.83

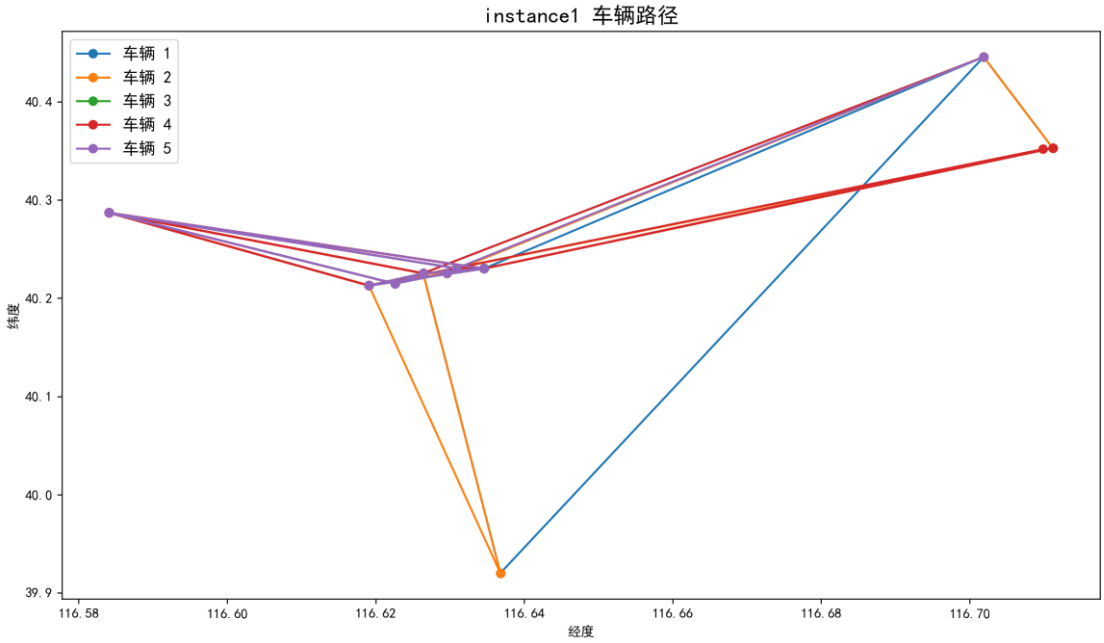

图 4-1 案例 1 车辆路径图

基于贪心的仿真方法求解结果如下：

表 4-2 基于贪心的仿真方法得分表

算例	总距离(km)	超时时间(s)	得分	算例	总距离(km)	超时时间(s)	得分
1	368.9	0	73.78	11	1043.9	0	208.78
2	428.1	0	85.62	12	1148.8	0	229.76
3	438.5	0	87.7	13	978.6	0	195.72
4	518.4	0	103.68	14	1242.9	0	248.58
5	488.2	0	97.64	15	1450.2	0	290.04
6	585.2	0	117.04	16	1069.3	0	213.86
7	354.9	0	70.98	17	2726.4	0	136.32
8	233.1	0	46.62	18	2846.8	0	142.34
9	952.55	0	190.51	19	3115.7	0	155.785
10	1173.4	0	234.68	20	1744.6	0	87.23

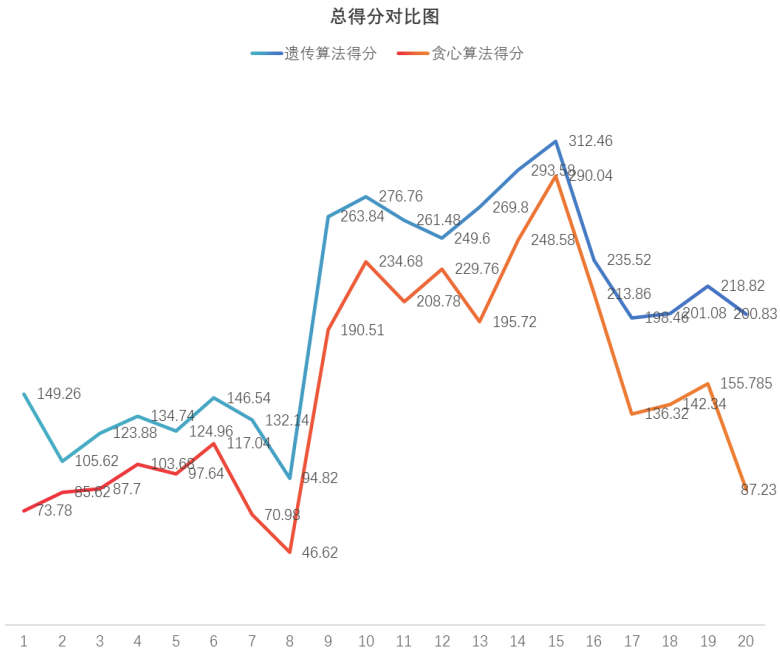

图 4-2 总得分对比图

由于前八个算例的订单数仅有 50 个，车辆为 5 辆，各个算法的得分均较低；第九至第十六个算例，订单增加至 100 个，而车辆数不变，车辆平均行驶距离迅速增长，因而得分有明显增大；第十七到二十个算例，订单数为 300，车辆增加到 20 辆，得分有所下降。

银奖代码的得分如下：

表 4-3 银奖代码得分表

算例	总距离 (km)	超时时间 (s)	得分	算例	总距离 (km)	超时时间 (s)	得分
1	795.3	0	159.06	11	1188.8	1494	4387.76
2	556.9	0	111.38	12	1177.2	0	235.44
3	748.2	291	957.973	13	1243.1	0	248.62
4	651.9	0	130.38	14	1104	0	220.8
5	607.6	0	121.52	15	1444.9	12300	6677.869
6	691.9	0	138.38	16	1310	1707	5003.667
7	800.1	6	160.02	17	1893.7	0	94.685
8	473.9	0	94.78	18	2216.8	0	110.84
9	1109.7	0	221.94	19	2669.4	0	133.47
10	1519.6	66	487.253	20	2598	0	129.9

对于部分算例，如当订单数增加至 100 而车辆有限的情况下，银奖代码会存在部分订单超时的情况，表现较差。相较而言，本文的两个算例均没有超时，鲁棒性更强。

表 4-4 平均得分对比表

	遗传算法	基于贪心的仿真方法	银奖代码
平均得分	199.710	150.833	991.287
得分中位数	200.955	139.33	159.54

由表 4-4，本文设计的基于贪心的仿真方法整体表现优于银奖代码设计的算法，平均得分和得分中位数都更低。虽然遗传算法的得分中位数较高，但是由于算法设计中对于超时的惩罚的设计，各个算例都没有超时，表现较优。

参考文献

[1] Savelsbergh M, Sol M. Drive: Dynamic routing of independent vehicles[J]. Operations Research, 1998, 46(4): 474-490.

[2] Gendreau M, Guertin F, Potvin J Y, et al. Neighborhood search heuristics for a dynamic vehicle dispatching problem with pick-ups and deliveries[J]. Transportation Research Part C: Emerging Technologies, 2006, 14(3): 157-174.

[3] Mitrović-Minić S, Laporte G. Waiting strategies for the dynamic pickup and delivery problem with time windows[J]. Transportation Research Part B: Methodological, 2004, 38(7): 635-655.

[4] Sáez D, Cortés C E, Núñez A. Hybrid adaptive predictive control for the multi-vehicle dynamic pick-up and delivery problem based on genetic algorithms and fuzzy clustering[J]. Computers & Operations Research, 2008, 35(11): 3412-3438.

[5] Pureza V, Laporte G. Waiting and buffering strategies for the dynamic pickup and delivery problem with time windows[J]. INFOR: Information Systems and Operational Research, 2008, 46(3): 165-175.

[6] Khalil E, Dai H, Zhang Y, et al. Learning combinatorial optimization algorithms over graphs[J]. Advances in neural information processing systems, 2017, 30.

[7] Zambaldi V, Raposo D, Santoro A, et al. Relational deep reinforcement learning[J]. arXiv preprint arXiv:1806.01830, 2018.

[8] Jiang J, Dun C, Huang T, et al. Graph convolutional reinforcement learning[J]. arXiv preprint arXiv:1810.09202, 2018.

[9] Li X, Luo W, Yuan M, et al. Learning to optimize industry-scale dynamic pickup and delivery problems[C]//2021 IEEE 37th international conference on data engineering (ICDE). IEEE, 2021: 2511-2522.