关于竞赛

动态取送货问题（Dynamic Pickup and Delivery Problem，DPDP）是物流领域的一个重要问题。目前该问题的研究主要基于人工数据集，而这无法真实地反映现实世界的复杂性和难度。为促进该问题的学术研究和实际应用，基于华为实际业务场景和脱敏后的真实数据集，我们在ICAPS 2021上组织本次竞赛。获奖者将获得现金和奖杯奖励，并被邀请在ICAPS 2021会议上在线发表演讲。ICAPS是智能和自动规划调度领域的顶会，ICAPS 2021将于2021年08月02日至13日在广州在线举行。

我们期待您的参与！

竞赛详情

问题简介

作为全球领先的信息和通信技术（ICT）基础设施和智能设备供应商，华为每年在数百家工厂/仓库生产加工数以亿计的产品。在制造过程中，大量货物（包括材料、产品和半成品）需要在工厂/仓库之间流通运输。由于客户需求和生产流程的不确定性，大多数运输需求无法事先确定。订单信息随机产生，该信息包括取货工厂、送货工厂、货物数量和时间要求。而我们需要安排车辆在一定时间周期内为这些订单提供服务。由于运输需求量大，即使物流效率的小幅度提升也能带来显著的经济和服务收益。因此，开发一种高效的动态调度算法具有重要的意义。

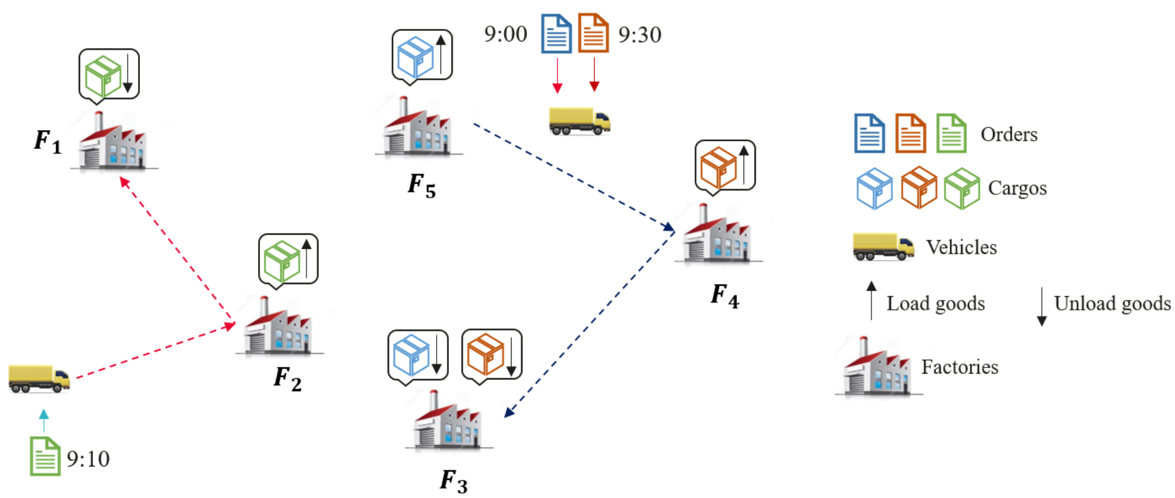

图1: DPDP问题示例

上述场景被抽象为动态取送货问题，指将所需货物从始发地运输到目的地，即通过车辆调度为动态生成的订单提供服务。该问题的目标是最大限度地减少订单的超时和车辆的平均行驶距离。以图1为例，一订单在上午9点生成，该订单包含需要从F_5出发送到F_4的货物。

问题描述

输入:

- 道路网络 $G=(F,A)$ 是一个完全有向图。 F 是由多个节点组成的集合 (即工厂集合): $\{F_i|i=1,\cdots,M\}$， $A=\{(i,j)|i,j\in M\}$ 是节点间边的集合。 每条边(i,j) 都对应着从节点i

到节点 j的非负运输距离d_{ij}和行驶时间t_{ij} 。
- 订单集合 $O=\{o_i|i=1,\cdots,N\}$，每个订单 $o_i=(F_p^i,F_d^i,q^i,t_e^i,t_l^i)$ 随机生成， 其中 F_p^i 和 F_d^i 分别代表取货和送货节点； q^i 是待运输货物的数量，表示为： $q^i=(q_{standard}^i,q_{small}^i,q_{box}^i)$， $q_{standard}^i$代表标准栈板的数量， q_{small}^i 代表小栈板的数量， q_{box}^i 代表箱子的数量； t_e^i 是订单创建时间； t_l^i是承诺完成时间 （注意该时间是指每一个订单的最终完成时间，包括车辆行驶时间，驶入垛口时间，垛口排队时间和装卸货时间）。1个标准栈板等于2个小托盘，1个小栈板等于2个箱子。
- 车辆集合 $V=\{v_k|k=1,\cdots,K\}$，其中每辆车 v_k 都有对应的装载容量和司机的工作班次。车辆的初始位置随机分配给各个节点。
- M 节点数（工厂数） $\{F_i|i=1,\cdots,M\}$ ，其中每个节点都有有限数量的垛口用于装卸作业，以及工作班次约束。如果所有垛口都在使用中，那么新到达的车辆必须排队等待。装卸操作只能在工作班次中进行。
- 驶入垛口时间是指在不考虑排队情况下，在车辆到达工厂和完成垛口分配之间的时间，如**30**分钟。
- 装卸时间，比如，如果车辆中有q个标准托盘，则装载时间$t_p=\omega\times q$和卸载时间$t_d=\omega\times q$, $\omega=240s/\text{每标准托盘}$。

输出:

- 订单分配计划及每辆车的行驶路线。

约束:

- 订单履行：所有订单都需要送达。
- 承诺完成时间：订单需要在承诺完成时间内完成，如4小时。否则，将会超时产生惩罚成本。
- 订单拆分：当一个订单无法被一辆空车完全装载时，可以拆分由多辆车装载运输，否则，不允许拆分订单。*比如车辆容量为15个标准栈板，如果一个订单为10个标准栈板，则不允许拆分，如另一个订单为20个标准栈板，则可以拆分且不限制拆分个数。*不允许拆分货物的最小单位，*例如，如果订单包含**13**个标准托盘、**7**个小托盘和**1**个箱子，则最小的单位是**1**标准栈板或**1**小栈板或**1**箱子，不能将栈板或箱子拆开。*
- 车辆的装载量：每辆车v_k，装载货物的总装载量不得超过最大装载量Q，例如每辆车最多装载**15**个标准栈板。
- 司机工作班次：例如，8:30-12:00, 13:30-18:00。为简化问题，此竞赛可忽略此约束。
- 后进先出（LIFO）装载约束：例如，如果订单o_1和o_2分配给同一车辆v_k，其中o_1的取货和送货节点为F_p^1和F_d^1，o_2的取货和送货节点为F_p^2和F_d^2，则路线方案$\{F_p^1、F_p^2、F_d^1、F_d^2\}$违反后进先出约束，$\{F_p^1、F_p^2,F_d^2,F_d^1\}$则不违反。
- 节点（工厂）的工作班次：例如，8:30-12:00, 13:30-19:00。为简化问题，此竞赛可忽略此约束。
- 每个节点的垛口有限：例如，如果节点F_1包含**3**个垛口，此时**4**辆车同时到达，则最后到达的车辆必须等到一个垛口空闲才可以装卸货。
- 装卸须遵守先到先得规则。如果多辆车辆同时到达同一节点，而该节点只有一个空闲垛口，则该节点将随机选择一辆车辆进行装卸。请注意，上述情况不会发生在实际情况中，但可能发生在模拟环境中。

目标:

该问题目标函数包括两项。第一项f_1表示最小化所有订单的总超时。如果订单o_i的到达时间表示为a_d^i，则

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

其中t_l^i是承诺的完成时间，N是订单总数。

第二项f_2表示最小化车辆的平均行驶距离。如果车辆v_k的路线计划为$\prod_k = \{n_1^k, n_2^k, \cdots, n_{l_k}^k\}$，其中$n_i^k$代表第$i$-个节点，即车辆$v_k$运输路线中的工厂，$l_k$是车辆$k$行驶的总节点数。则，

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

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

目标函数可以表示为：

$$\min f = \lambda \times f_1 + f_2$$

其中λ是一个很大的正数，保证目标f_1优先级大于f_2。

示例:

例如，如图1所示，有两辆车和三个订单：

$o_1 = (F_2, F_1, (13, 0, 0), 9:10, 13:10)$,

$o_2 = (F_5, F_3, (8, 0, 0), 9:00, 13:00)$,

$o_3 = (F_4, F_3, (4, 0, 0), 9:30, 13:30)$.

为了简化问题，我们假设订单中只有标准托盘，每个工厂有3个垛口。此外，假设车辆的初始位置位于这些订单的取货点。现在，参赛者将决定如何将这3个订单分配给2辆车，以及如何安排运输路线。

给定工厂间的距离和时间矩阵，如下所示：

	F_1	F_2	F_3	F_4	F_5
F_1	0 km, 0 min	5 km, 10 min	50 km, 80 min	55 km, 90 min	60 km, 100 min
F_2	5 km, 10 min	0 km, 0 min	60 km, 100 min	65 km, 110 min	70 km, 120 min
F_3	50 km, 80 min	60 km, 100 min	0 km, 0 min	20 km, 25 min	40 km, 50 min
F_4	55 km, 90 min	65 km, 110 min	20 km, 25 min	0 km, 0 min	25 km, 30 min
F_5	60 km, 100 min	70 km, 120 min	40 km, 50 min	25 km, 30 min	0 km, 0 min

此示例的一个可行解是：

车辆v_1为订单o_1服务，车辆v_2为订单o_2和o_3服务。

行驶路线是：

$v_1 : \{F_2, F_1\}$

$v_2 : \{F_5, F_4, F_3\}$

显然，我们很容易发现，车辆行驶时间、驶入码头时间和装卸时间的总和没有超过承诺的4小时交付要求，即这些订单没有超时，所以$f_1 = 0$。

此外，行驶距离f_2计算为：

$$f_2 = \frac{1}{2} \times (5 + 25 + 20) = 25$$

因此，目标函数f为：

$$f = 0 \times \lambda + 25 = 25$$

模拟器及结果检查

我们将提供一个带有结果检查功能的模拟器，以评估参赛者提交的算法在计划范围内（例如，一天）的性能。在模拟器中，一天将被分割为相同长度的T个时间间隔（例如，$T = 144$，则每个时间间隔Δt为10分钟）。模拟器和算法之间的交互如图2所示。

在每个时间节点t $(t = l \cdot \Delta t, l = 0, 1, 2, 3, \ldots)$，模拟器将更新所有环境信息，包括：

- 订单信息：在$[t - \Delta t, t]$中生成的新订单信息，以及在$[0, t - \Delta t]$中生成，但尚未完成订单的状态信息
- 车辆信息：所有车辆的当前状态信息，如位置，装载情况等

在每个时间节点t $(t = l \cdot \Delta t, l = 0, 1, 2, 3, \ldots)$，算法可以从模拟器中获取最新的环境信息，并开始将新的订单分配给车辆，已经分配但未开始的订单也可以重新分配。然后算法将调度结果返回给模拟器，模拟任务下发过程。此过程会重复执行，直到所有订单完成运输。

如图2所示，参赛者提交算法的运行时间大于或小于Δt模拟器都支持。例如，当$\Delta t = 10min$，如果算法的运行时间小于Δt，如3分钟，则模拟器将检查并获取调度结果，并跳过其余7分钟，直接进入下一个时间间隔。如果算法的运行时间大于Δt，如15分钟，模拟器仍将在后台更新车辆和既有订单的状态信息，且每10分钟接收新生成的订单，但此时模拟器不会调度这些新生成的订单和既有未完成订单，需要等待参赛者算法在第15分钟给出调度结果，从而调度上述订单。

模拟器和算法之间通过API实现交互。

模拟器和结果检查工具及其详细说明下载链接：https://competition.huaweicloud.com/information/1000041411/Download，需报名后方可下载。

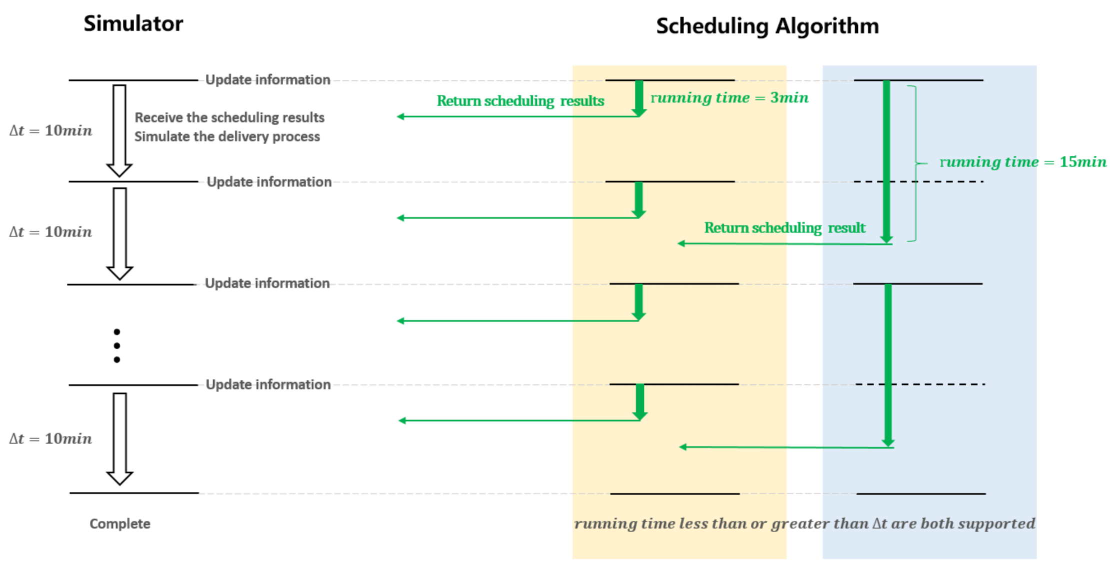

图2：模拟器与算法的交互

数据集

公开测试数据集包含n天的历史数据，隐藏的评估数据集包含x天的历史数据，其中每天包含**2000** ~**5000**个订单，**100** ~**200**辆车辆。当一个订单无法被一辆空车完全装载时，可以拆分由多辆车装载运输，否则，不允许拆分订单。*比如车辆容量为15个标准栈板，如果一个订单为10个标准栈板，则不允许拆分，如另一个订单为20个标准栈板，则可以拆分且不限制拆分个数。*

公开测试数据集及其详细说明下载链接：https://competition.huaweicloud.com/information/1000041411/Download ，需报名后方可下载。

算法提交

参赛者提交的算法将在一组隐藏数据集上进行评估。在**测试提交**阶段，每个团队最多可以提交**3**次，但不计入最终排名。在**正式提交**阶段，每个团队最多可以提交**5**次。参赛者的排名将在官网公布，并及时更新。

参赛者提交的算法将在Docker上以**单核**模式运行，相关配置如下：

- **OS** - Ubuntu 18.04.4 LTS
- **CPU** - Intel(R) Xeon(R) Gold 6151C CPU @ 3.00GHz
- **Memory** – 32G

注意：

1. 服务器支持的编程语言：Python/Java/C/C++。
2. 参赛者提交前请务必在我们提供的本地docker上测试无误，docker镜像将在测试提交阶段前公布。

时间节点

- 2021年05月01日：竞赛开始。
- 2021年05月21日：北京时间晚上11:59，测试提交开始。
- 2021年06月01日：北京时间晚上11:59，测试提交结束，正式提交开始。
- 2021年06月30日：北京时间晚上11:59，组队冻结。
- 2021年07月15日：北京时间晚上11:59，竞赛结束。
- 2021年08月01日：宣布获胜队伍。
- 2021年08月03日：ICAPS 2021将邀请获奖团队进行在线技术报告。
- 最终的获奖队伍排名，将由竞赛委员会根据竞赛成绩，确认提交方案有效性后评判。

奖项设置

获奖者将获得奖金和奖杯。奖金总额（美元）：10,000。

- **金奖**：5,000美元
- **银奖**：3,000美元
- **铜奖**：2,000美元

组委会(字母序)

- 郝建业（华为诺亚方舟实验室）
- 陆佳文（华为诺亚方舟实验室）
- 童夏良（华为诺亚方舟实验室）
- 袁明轩（华为诺亚方舟实验室）
- 项　翔（华为诺亚方舟实验室）
- 卓汉奎（中山大学）