Multimodal Transportation 4 (2025) 100194

Contents lists available at ScienceDirect

Multimodal Transportation

journal homepage: www.elsevier.com/locate/multra

Full Length Article

A cost function approximation method for dynamic vehicle routing with docking and LIFO constraints

Markó Horváth, Tamás Kis*, Péter Györgyi

HUN-REN Institute for Computer Science and Control, H-1111 Budapest, Kende u. 13-17, Hungary

ARTICLE INFO

Keywords:
Dynamic pickup and delivery problem
Docking constraints
Cost function approximation
Variable neighborhood search

ABSTRACT

In this paper, we study a dynamic pickup and delivery problem with docking constraints. There is a homogeneous fleet of vehicles to serve pickup-and-delivery requests at given locations. The vehicles can be loaded up to their capacity, while unloading has to follow the last-in-first-out (LIFO) rule. The locations have a limited number of docking ports for loading and unloading, which may force the vehicles to wait. The problem is dynamic since the transportation requests arrive real-time, over the day. Accordingly, the routes of the vehicles are to be determined dynamically. The goal is to satisfy all the requests such that a combination of tardiness penalties and traveling costs is minimized. We propose a cost function approximation based solution method. In each decision epoch, we solve the respective optimization problem with a perturbed objective function to ensure the solutions remain adaptable to accommodate new requests. We penalize waiting times and idle vehicles. We propose a variable neighborhood search based method for solving the optimization problems, and we apply two existing local search operators, and we also introduce a new one. We evaluate our method using a widely adopted benchmark dataset, and the results demonstrate that our approach significantly surpasses the current state-of-the-art methods.

1. Introduction

Dynamic vehicle routing problems (DVRPs) constitute a rapidly developing field of transportation research, which is certified by a series of recent review papers, e.g., Berbeglia et al. (2010), Pillac et al. (2013), Bektaş et al. (2014), Psaraftis et al. (2016), Soeffker et al. (2022), Zhang and Van Woensel (2022). The growing interest is due to the wide range of real-world application areas such as transportation of goods and people, services, etc. (Rios et al., 2021). A related aspect of growing interest is environment friendly routing or eco-routing, which aims to optimize the control of a fleet of vehicles to reduce the energy consumption and the pollution (Dong et al., 2022; Pahwa and Jaller, 2024).

In this paper, we study a DVRP motivated by a real-life problem proposed by Huawei Technologies Co. Ltd. A fleet of homogeneous vehicles has to serve pickup-and-delivery requests which occur at given locations over the day. Each request is characterized by a size, a pickup and a delivery location, a release time, and a due date. The vehicles can be loaded up to their capacity, while unloading has to follow the last-in-first-out (LIFO) rule. The locations have a limited number of docking ports for loading and unloading, which may force the vehicles to wait. The problem is dynamic since the transportation requests arrive real-time, over the day. Accordingly, the routes of the vehicles are to be determined dynamically. The goal is to satisfy all the requests such that a combination of tardiness penalties and traveling costs is minimized.

* Corresponding author.
E-mail addresses: marko.horvath@sztaki.hu (M. Horváth), tamas.kis@sztaki.hu (T. Kis), peter.gyorgyi@sztaki.hu (P. Györgyi).

https://doi.org/10.1016/j.multra.2025.100194
Received 24 May 2024; Received in revised form 13 August 2024; Accepted 14 August 2024

An important feature of our problem is that although the pickup and delivery locations are known in advance, the characteristics of the transportation requests are unknown until their release times. Moreover, the distribution of the requests both in time and space may vary over the day. Consequently, a single problem instance does not provide exploitable stochastic information. However, statistical data may be collected over longer time periods that potentially could be used in solution approaches.

Our problem has two important constraints that frequently occur in practice. The first one limits the number of the docking ports at each location. The number of the docking ports is limited by the size of the buildings (warehouses or factories) and also, it may not be economical to have sufficient crew to serve (load or unload) any number of vehicles simultaneously. Usually, vehicles do not arrive evenly at a location over the day, which means that in peak periods, some of the vehicles have to wait until a docking port becomes free. In several applications the service time of a vehicle may be comparable to the average travel time between two locations, which may lead to high waiting times in case of superficial planning. Despite its high practical relevance, the literature on pickup and delivery problems with docking constraints at the locations is rather scarce, we refer to e.g., Cai et al. (2022a, 2023), Du et al. (2023). Most results are for outbound and inbound transportation problems with a depot, see Section 2.3 for references. The second side constraint is the LIFO rule. This policy means that the last loaded order must be unloaded first. This rule is necessary, for example, if the vehicles used for transportation have only a single access door for loading and unloading. Also, if the orders are hazardous, weighty, or fragile, the load rearrangement on the vehicle may consume much time and increase handling costs. The related literature is summarized in Section 2.3.

Main contributions. We propose a cost function approximation method for the problem at hand, where we manipulate the cost function by introducing two penalty terms. One of them is directly related to the waiting caused by the docking constraints. By explicitly penalizing waiting, we expect to make solutions flexible to accommodate new requests in the future. We model the problem as a sequential decision process, where in each decision epoch a variable neighborhood search (VNS) based method is used to solve the respective optimization problem. In the VNS method we use two old and a new neighborhood operator.

As a case study, we evaluate our solution approach on a dynamic pickup and delivery problem. This problem was introduced in *The Dynamic Pickup and Delivery Problem challenge* (Hao et al., 2022), hosted by the International Conference on Automated Planning and Scheduling in 2021[1] (ICAPS 2021).

To sum up, our contributions are as follows.

- We propose a cost function approximation based method for solving the respective optimization problem in each decision epoch. One of the penalty terms is directly related to the waiting times caused by the docking constraints.
- We evaluate our solution procedure on the benchmark instances of the ICAPS 2021 DPDP competition. The computational experiments show that our method significantly outperforms the state-of-the art methods on this dataset, especially on the large-size instances. The average improvement on the full dataset is more than 50% when compared to the best published methods.
- We demonstrate the benefit of using the suggested penalty terms in separate experiments. We also explain the mechanism how penalizing waiting improves the solution.

According to our best knowledge, our approach for avoiding waiting due to docking constraints at each station is new.

Structure of the paper. In Section 2, we overview the dynamic vehicle routing problems and the related literature. In Section 3, we give a formal description of the problem studied along with modeling as a sequential decision process. In Section 4, we describe our solution approach and in Section 5, we present our computational results. Finally, we conclude the paper in Section 6.

2. Literature review

In this section, we briefly overview the related literature. In Section 2.1, we narrow our scope to the problem class studied in this paper. Modeling with sequential decision process and related solution methods are summarized in Section 2.2. Finally, Section 2.3 is concerned with the LIFO loading rule, and problems with docking constraints.

2.1. Classification of our vehicle routing problem

The problem studied in this paper is *dynamic*, since the input of the problem is received and updated concurrently with the determination of the routes (Psaraftis, 1980). By Pillac et al. (2013), a dynamic problem is *stochastic*, if some exploitable stochastic knowledge is available on the dynamically revealed information, and *deterministic* otherwise. In a single instance of our problem, no explicit stochastic information is available, so in this sense our problem is deterministic.

In the *general pickup and delivery problem* a set of routes has to be constructed in order to satisfy transportation request by a fleet of vehicles (Savelsbergh and Sol, 1995). Each transportation request contains a set of pickup locations alongside load quantities and a set of delivery locations along with quantities to unload. Each request has to be fulfilled by a vehicle without transshipment and exceeding its capacity. A special case is the *pickup and delivery problem*, where each request has a single pickup location and a single delivery location. We refer to (Berbeglia et al., 2010) for a survey.

We conclude that the problem studied in this paper falls in the broad class of DVRPs, and in particular, it is a deterministic dynamic pickup and delivery problem.

[1] https://icaps21.icaps-conference.org/Competitions/.

2.2. Modeling and solving DVRPs

A dynamic vehicle routing problem can be formulated as a *sequential decision process* (SDP). An SDP goes through a sequence of *states* that describe the status of the system at distinct time points. When passing to the next state, the transition is determined by the dynamic information revealed and the decisions made since the occurrence of the last state. The transition may be deterministic or stochastic, in the latter case we have a *Markov decision process*. We refer the reader to (Powell, 2011) for a general introduction to SDPs.

When modeling a DVRP, decision epochs may occur regularly, or at some special occasions. The states encode all the information needed to make decisions, and evaluate solution alternatives. For instance, they may contain the position and current tasks of the vehicles, the set of open service requests, etc. The decisions are concerned with the assignment of new service requests and possibly routes to the vehicles. The transition to the next state determines the new position of the vehicles, or the progress of loading and unloading. The dynamic information may influence the transition to the new state, e.g., new service requests manifest.

A number of methods have been proposed for solving DVRPs. The simplest ones are *myopic*, also termed *rolling horizon re-optimization (RO)* methods, that focus only on the current state without any considerations of future uncertainties. Essentially, a series of static problems are solved with the assumption that the realizations will remain unchanged in the future, e.g., no new orders will be requested. Gendreau et al. (1999) solve a vehicle routing problem with time windows with this approach, while Ichoua et al. (2000) consider the same problem, but do not allow to change the destination of the moving vehicles. Lin et al. (2014) propose a MIP formulation and a heuristic method for processing offline and real-time service requests. An interesting aspect of the problem is that customers may cancel their requests. However, optimizing without acknowledging the future can be counterproductive as it often leads to inflexible solutions.

To alleviate the short-sightedness of the myopic strategies, Mitrović-Minić et al. (2004) propose waiting strategies for a dynamic problem with time windows to delay the dispatch of new orders, and to make decision together with the orders that may be requested in the near future. Van Hemert and La Poutré (2004) consider a dynamic problem of collecting loads, and inserted fictive, anticipated loads into the routes in order to encourage vehicles to explore fruitful regions (i.e., regions that have a high potential of generating loads).

Reinforcement learning. for solving DVRPs has gained an increasing attention, we refer to (Hildebrandt et al., 2023) for an overview. Below we summarize the most common techniques. *Policy function approximation (PFA)* is a function that assigns an action to any state, without any further optimization and without using any forecast of the future information. Ulmer and Streng (2019) apply this method for a problem where parcel pickup stations and autonomous vehicles are combined for same-day delivery. Ghiani et al. (2022) tackle a pickup and delivery problem using an anticipatory policy by creating priority classes for the requests and their algorithm utilizes a parametric policy function approximation. *Value function approximation (VFA)* is used to estimate the value function, which provides the expected cumulative reward of any given state. Ulmer et al. (2018) model a multi-period VRP by a Markov decision process, where the set of requests to be accepted in each period is determined by a suitable value function approximation computed offline. Van Heeswijk et al. (2019) consider a delivery dispatching problem and provided a method that can handle large instances. For further examples, we refer to the supplementary material of Zhang et al. (2022).

Beyond reinforcement learning, *cost function approximation (CFA)* is a further technique for solving DVRPs. The key idea is that the problem to be solved in each decision epoch is modified, that is, either the cost function or the problem constraints are slightly perturbed. For instance, Riley et al. (2019, 2020) consider a dial-a-ride problem with the aim of minimizing the total waiting time. The authors introduce an extra penalty term for unserved request to ensure that all riders are served in reasonable time. The penalty associated with a request is increased after each epoch in which the request is not served. Ulmer et al. (2020) study a retail distribution problem, where familiarity with the delivery location can save service time for the driver, and a related quantity is added to the original objective function. Hildebrandt et al. (2023) outline an enhancement of CFA methods by first modifying the state received before determining the next action, e.g., by reducing the due dates of some transportation requests, or by reducing vehicle capacities, etc.

Other approaches use the stochastic information internally, e.g., via sampling realizations. For example, *multiple scenario approaches* sample realizations to create a set of scenarios, which are then solved separately, and based on those individual solutions, a consensus decision is made (Bent and Van Hentenryck, 2004; Dayarian and Savelsbergh, 2020; Hvattum et al., 2006; Srour et al., 2018). For instance, Srour et al. (2018) study a stochastic and dynamic pickup and delivery problem with time windows, where the location of future requests is known, but only stochastic information is available about the time windows of future requests. The authors propose to sample future requests at each decision point and solve a VRP for each scenario, and finally synthesize the next action of the vehicles based on the set of solutions obtained for the different scenarios. An alternative approach to sampling scenarios is suggested by Györgyi and Kis (2019) for solving the same problem. The stochastic information is used to set up a minimum-cost network flow problem in which source-to-sink routes correspond to vehicle routes and the edges are weighted with conditional expected values. Fonseca-Galindo et al. (2022) use statistical data to assign packages to vehicle routes in a package delivery system, where a stream of incoming customer orders has to be delivered by a fleet of vehicles.

VNS is a widely used approach to solve a static problem or a dynamic problem at a decision point (Mladenović and Hansen, 1997). It is a metaheuristic which systematically performs the procedure of neighborhood change, both in descent to local minima and in escape from the valleys which contain them. We refer to (Hansen et al., 2019) for a general overview. Choosing the best neighborhood operators is a difficult problem, see e.g., (Chen et al., 2022; Liu et al., 2023; Wandelt et al., 2022) for recent developments.

2.3. Docking and LIFO constraints

The considered pickup and delivery problem has two important side constraints, namely, the docking constraints, and the LIFO constraints. They are of great practical importance and have been studied by the research community.

Docking constraints. Several papers deal with vehicle routing problems with service restrictions, however, these restrictions mostly apply to a single depot. In Dabia et al. (2019) shifts with limited loading capacities are considered. Hempsch and Irnich (2008) consider a problem with sorting capacity constraints at the depot and use a local search based method to solve it. Gromicho et al. (2012) apply a method that uses a decomposition scheme where columns are generated by a routine based on dynamic programming to solve a variant with limited number of loading docks or limited size of loading crew. Van der Zon (2017) considers also a vehicle routing problem with a limited number of loading docking ports and propose a method that shifts the start times of the routes in a smart way. To our best knowledge, the only papers that deal with docking constraints at the service locations are those related to the DPDP problem studied in this paper (Cai et al., 2022a; 2023; Du et al., 2023).

LIFO constraints in pickup and delivery problems. Cordeau et al. (2010) propose a branch-and-cut algorithm for a pickup and delivery problem with a single vehicle. Benavent et al. (2015) examine a variant of the problem with multiple vehicles and a special time constraint, and propose a branch-and-cut and a tabu search algorithm as solution methods. Xu and Wei (2023) construct a multi-objective mathematical model for a pickup and delivery problem with transshipments, and propose a method that generates an initial solution by Clarke-Wright saving algorithm and uses both neighborhood search and Q-learning to improve the solution. In (Carrabs et al., 2007), several local search operators (e.g., *couple-exchange, block-exchange, relocate-couple, relocate-block, multi-relocate, 2-opt-L, double-bridge*) are proposed to problems with LIFO constraints.

3. Problem statement

In the following, we define the problem in detail. We first define the basic data of the problem, then we present the dynamic problem as a sequential decision process.

Given a finite set of factories $\mathcal{F}$, a finite set of orders $\mathcal{O}$, and a fleet of homogeneous vehicles $\mathcal{V}$. The *distance* and the *travel time* between factories $f_i, f_j \in \mathcal{F}$ are denoted with $\mathrm{dist}(f_i, f_j)$ and $\mathrm{travel}(f_i, f_j)$, respectively. Each order $o_i \in \mathcal{O}$ is described by a tuple $(f_i^p, f_i^d, t_i^p, t_i^d, q_i, h_i^p, h_i^d)$, where f_i^p and f_i^d represent the *pickup factory* and the *delivery factory*, respectively, t_i^p is the *release time*, t_i^d is the *due date*, q_i is the order *quantity*, and h_i^p and h_i^d are the times required to *load* and to *unload* the order, respectively. Order o_i becomes known only at the release time t_i^p. The total quantity of those orders that can be carried by a vehicle at any given moment is limited by a constant Q. Initially, each vehicle is empty and parks at a given factory. Unloading the orders from a vehicle has to follow the LIFO rule, refer to Fig. 1 for an example.

Each factory has a given number of docking ports for loading and unloading. Vehicles are served on a first-come-first-served basis, that is, if a vehicle arrives at a factory and all ports are occupied, the service of the vehicle cannot begin immediately, but the vehicle has to wait until one of the docking ports becomes free, and no vehicle that arrived earlier is waiting for a port. This is illustrated in Fig. 2. The time elapsed between the arrival of the vehicle and the start of service is called the *waiting time*. Serving a vehicle decomposes to dock approaching, then unloading some carried orders, and finally loading some new orders. The *service time* of a vehicle is the sum of the factory-independent *dock approaching time*, h^{docking}, and the sum of the unloading and loading times of the corresponding orders. For more details we refer to Appendix A. After serving a vehicle, the port becomes free, and the vehicle may park at the factory, or depart to the next factory on its route.

The goal is to route the vehicles so that all orders are served, and the weighted sum of the total distance traveled and the total tardiness of the orders is minimized.

3.1. Modeling as a sequential decision process

We model our problem as an SDP (Powell, 2011; Soeffker et al., 2022). The process goes through a sequence of states $s_0, s_1, \ldots$, each corresponding to an instance of the problem's decision model at time points $\tau_0, \tau_1, \ldots$, as illustrated in Fig. 3. Between two consecutive time points τ_k and τ_{k+1}, an algorithm computes some actions based on the state s_k at τ_k, which comprises the new

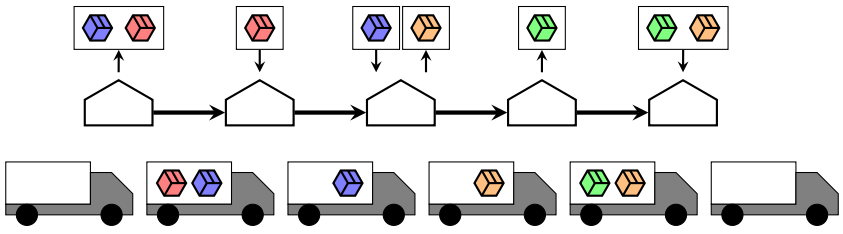


Fig. 1. An example for loading and unloading during a vehicle route. Orders are depicted as boxes, factories are depicted as pentagons. Unloading the orders from a vehicle has to follow the LIFO rule, see the order of orders above the factories, and also the position of the orders on the vehicle.

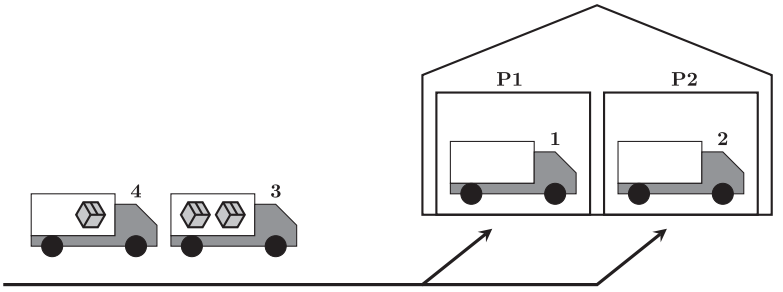


Fig. 2. An example for serving vehicles at a factory. The two docking ports, P1 and P2, of the factory are occupied by vehicles 1 and 2, respectively, thus the service of vehicles 3 and 4 is delayed until a docking port becomes free.

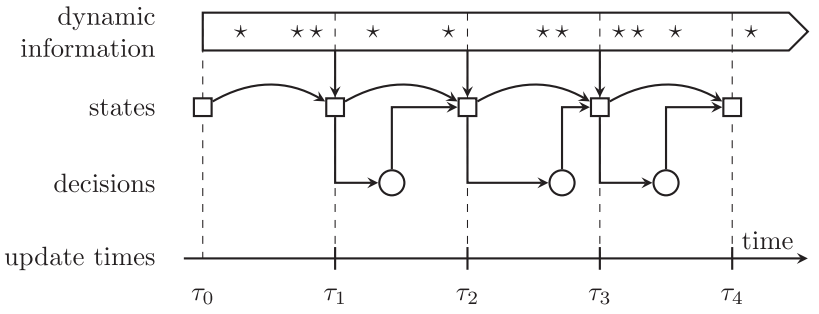


Fig. 3. Sequential decision process.

dynamic information revealed between τ_{k-1} and τ_k. The actions along with the dynamic information between τ_k and τ_{k+1} determine the transition of the system to the next state s_{k+1} at τ_{k+1}.

3.1.1. Decision epochs

The *update times* τ_k $(k = 0, 1, \dots, K)$ divide the operating horizon into *epochs* of length Δ each, that is, $\tau_0 = 0$ and $\tau_{k+1} = \tau_k + \Delta$. The orders are requested in a given finite planning horizon (e.g., one day), however, the operating horizon may be longer if we are not able to complete the orders within the planning horizon. The decision process ends if all the orders are delivered.

3.1.2. Dynamic information

The *realization* ω_k of the dynamic information at update time τ_k comprises the orders requested in the previous epoch, i.e., $\omega_k = \{o_i \in \mathcal{O} : \tau_{k-1} < t_i^p \leq \tau_k\}$.

3.1.3. States

A *state* s_k at update time τ_k is a tuple $(\tau_k, \Phi_k, \tilde{\omega}_k)$, where $\Phi_k = \{\Phi_{k,v} : v \in \mathcal{V}\}$ is the status of the vehicles, and $\tilde{\omega}_k$ is the set of unprocessed orders.

The set $\tilde{\omega}_k$ comprises the dynamic information ω_k, and those orders which are released not later than τ_{k-1}, but not picked up until τ_k.

At any time moment a vehicle is either located at a factory, which is then its *current factory* or is on the way to its *destination factory*. The *status of a vehicle* v is described by a tuple $\Phi_{k,v} = (\phi_{k,v}^{\text{curr}}, C_{k,v}, \theta_{k,v})$, where

- $\phi_{k,v}^{\text{curr}} = (f_{k,v}^{\text{curr}}, td_{k,v}^{\text{curr}})$ provides the current factory and the *earliest departure time* if the vehicle is located at the factory $f_{k,v}^{curr}$ at time τ_k, while $\phi_{k,v}^{\text{curr}} = \emptyset$ if the vehicle is on the way to a factory at time τ_k.
- $C_{k,v}$ is the list of orders carried by the vehicle, sorted in the order of loading. If $\phi_{k,v}^{\text{curr}} \neq \emptyset$, then $C_{k,v}$ contains all the orders that the vehicle had to pickup at $f_{k,v}^{curr}$, and does not contain those orders that were delivered to that factory.
- $\theta_{k,v}$ is the route plan of the vehicle consisting of a sequence of tuples each corresponding to a factory:

$$\theta_{k,v} = \left((f_{k,v}^j, ta_{k,v}^j, td_{k,v}^j, \mathcal{D}_{k,v}^j, \mathcal{P}_{k,v}^j) : 1 \leq j \leq \ell_{k,v} \right).$$

The jth tuple in this list corresponds to the jth factory $f_{k,v}^j$ of the route with arrival time $ta_{k,v}^j$, departure time $td_{k,v}^j$, and with the list of orders $\mathcal{D}_{k,v}^j$ to be unloded, and $\mathcal{P}_{k,v}^j$ to be loaded, respectively. The first factory visited in $\theta_{k,v}$ is the *destination factory* of the vehicle. Each route plan must be *feasible*, i.e., it has to fulfill the *fundamental routing constraints*, the *capacity constraints*, and the *LIFO constraints*, refer to Appendix B for full details.

In the initial state s_0 at τ_0, vehicles have only current factories, which coincide with their initial factories.

3.1.4. Actions

Actions are taken at update points. An action is merely a feasible route plan for each vehicle. That is, at update point τ_k, let $\Theta(s_k)$ be the set of all possible tuples of feasible and mutually compatible route plans for the vehicles from which exactly one tuple $x_k = (\theta^x_{k,v} : v \in \mathcal{V})$ must be chosen. Notice that the $\theta^x_{k,v}$ must be feasible route plans, and they must be *mutually compatible*, i.e., the same order cannot be served by route plans of distinct vehicles.

A further constraint is that if $\theta_{k,v}$ is non-empty, then the first factory visited in $\theta_{k,v}$ and $\theta^x_{k,v}$ must be the same, i.e., $f^1_{k,v} = f^{x,1}_{k,v}$. Moreover, $ta^1_{k,v} = ta^{x,1}_{k,v}$, and $\mathcal{D}^1_{k,v} = \mathcal{D}^{x,1}_{k,v}$ must hold. However, the set of orders to pickup and thus the departure time from $f^1_{k,v}$ may be different in $\theta_{k,v}$ and $\theta^x_{k,v}$.

3.1.5. Reward function

The reward function assigns a value to a (state, action) pair (s_k, x_k). Let $\mathbf{f}_1$ denote the total distance traveled by the vehicles, that is,

$$\mathbf{f}_1(s_k, x_k) = \sum_{v\in\mathcal{V}} \text{dist}\left(f^{\text{curr}}_{k,v}, f^{x,1}_{k,v}\right) + \sum_{v\in\mathcal{V}} \sum_{j=2}^{\ell^x_{k,v}} \text{dist}\left(f^{x,j-1}_{k,v}, f^{x,j}_{k,v}\right),$$

where $\text{dist}(f^{\text{curr}}_{k,v}, f^{x,1}_{k,v}) = 0$ if $\phi^{\text{curr}}_{k,v} = \emptyset$ or $\ell^x_{k,v} = 0$, and let $\mathbf{f}_2$ be the total tardiness anticipated:

$$\mathbf{f}_2(s_k, x_k) = \sum_{v\in\mathcal{V}} \sum_{j=1}^{\ell^x_{k,v}} \sum_{o_i\in\mathcal{D}^{x,j}_{k,v}} \max\left(0, ta^{x,j}_{k,v} - t^d_i\right).$$

Then, the reward function is

$$R_0(s_k, x_k) = \lambda_1\mathbf{f}_1(s_k, x_k) + \lambda_2\mathbf{f}_2(s_k, x_k), \tag{1}$$

where $\lambda_1, \lambda_2 > 0$ are appropriate multipliers. We will modify this function in our CFA method in Section 4.1.

3.1.6. Transition

After the selection of action x_k, the decision process transitions to the next state s_{k+1} at update point τ_{k+1}. The positions of the vehicles, and the lists of carrying orders are updated. Briefly stated, if a vehicle v arrives at its destination factory before τ_{k+1}, that factory becomes its current factory, and the next factory to visit, if any, becomes the new destination factory. As a result, $\theta^x_{k,v}$ is transformed to the route plan $\theta_{k+1,v}$ in the new state s_{k+1}. $C_{k+1,v}$ is obtained from $C_{k,v}$ by removing the orders delivered at the current factory of vehicle v in state s_k, and adding to it the pickup orders, if any. If vehicle v departed from its current factory between τ_k and τ_{k+1}, then no current factory will be associated with the vehicle, and the factory to which the vehicle is heading will be the destination factory. Then $\Phi_{k+1,v} = (C_{k+1,v}, \phi^{\text{curr}}_{k+1,v}, \theta_{k+1,v})$ for each $v \in \mathcal{V}$. For details, we refer to Appendix C.

3.1.7. Objective function

The SDP eventually creates a feasible route θ_v for each vehicle v, and the routes are mutually compatible and serve all the requests. Moreover, we assume that for each $v \in \mathcal{V}$, f^1_v coincides with the initial factory of vehicle v in the initial state s_0. A formal definition of the feasibility of a solution is given in Appendix D. Then $x = (\theta_v : v \in \mathcal{V})$ is a solution of the problem. The solution is evaluated by the cost function

$$cost(x) = \lambda_1\mathbf{f}_1(s_0, x) + \lambda_2\mathbf{f}_2(s_0, x). \tag{2}$$

3.2. Example

In Fig. 4, we depict the route of a single vehicle v at different states. The gray nodes and edges represent the route of the vehicle before the respective update time points. The black thick edges outline the route to the destination factory, which cannot be changed, while the route indicated by dashed edges can be modified. To ease the calculations, we assume that the travel time between any two factories is 11 min, the dock approaching time is 2 min, and the loading/unloading time for an order is 1 min. The length of the epochs is $\Delta = 10$ min.

Before $\tau_4 = 40$, three orders have arrived: order o_1 from f_1 to f_2, order o_2 from f_1 to f_3, and order o_3 from f_4 to f_5. The vehicle departed from its initial factory f_0 at time 10 and traveled to factory f_1. The vehicle arrived at factory f_1 at time 21, picked up orders o_2 and o_1 and departed at time 25. The vehicle arrived at factory f_2 at time 36, delivered order o_1 and left the factory at time 39 to travel to factory f_3.

State s_k ($\tau_k = 40$). The left pane of Fig. 4 shows the state s_k at update point $\tau_k = 40$. The vehicle is currently on the way to factory f_3, where it will arrive at time 50. According to the tentative route plan, the vehicle after that will travel to factory f_4 to pickup and deliver order o_3 to factory f_5. Thus, the status of the vehicle is given by $\phi^{\text{curr}}_{k,v} = \emptyset$, $C_{k,v} = (o_2)$, and $\theta_{k,v} = ((f_3, 50, 53, (o_2), \emptyset), (f_4, 64, 67, \emptyset, (o_3)), (f_5, 78, 81, (o_3), \emptyset))$. Order o_4 from factory f_6 to f_5 is also revealed in the previous epoch (see the white circle node), thus $\omega_k = \{o_4\}$. Since order o_1 is already delivered, and order o_2 is already picked up, $\bar{\omega}_k = \{o_3, o_4\}$.

Action x_k. The decision maker decided to insert factory f_6 into the the tentative route plan of the vehicle, that is, $x_k = (\theta^x_{k,v})$, where $\theta^x_{k,v} = ((f_3, 50, 53, (o_2), \emptyset), (f_6, 64, 67, \emptyset, (o_4)), (f_4, 78, 81, \emptyset, (o_3)), (f_5, 92, 96, (o_3, o_4), \emptyset))$. In the center pane, we depict the updated route plan of the vehicle.

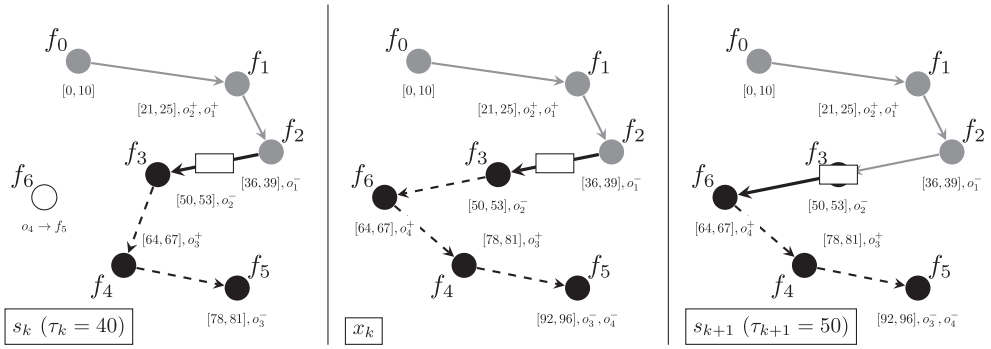


Fig. 4. An example for a route plan of a vehicle at different states and actions. Left pane shows the route at an intermediate state of the decision process. Center pane shows the updated route according to an action. Right pane shows the route at the next state of the decision process.

State s_{k+1} $(\tau_{k+1} = 50)$. The right pane of Fig. 4 shows the state s_{k+1} at update point $\tau_{k+1} = 50$. Since the vehicle just arrived at factory f_3, it became its current factory. According to action x_k, after this visit the vehicle will depart toward its new destination factory f_6 to pickup order o_4. Thus, the status is given by $\mathcal{O}_{k+1,v} = \emptyset$, $\phi^{\text{curr}}_{k+1,v} = (f_3, 53)$, and $\theta_{k+1,v} = \big((f_6, 64, 67, \emptyset, (o_4)), (f_4, 78, 81, \emptyset, (o_3)), (f_5, 92, 96, (o_3, o_4), \emptyset)\big)$. No new orders are requested in the previous epoch, thus $\omega_{k+1} = \emptyset$, however, the pickup of orders o_3 and o_4 can be still changed, thus $\tilde{\omega}_{k+1} = \{o_3, o_4\}$.

4. CFA approach for solving the routing problem in each epoch

In this section, we propose a cost function approximation based approach to solve the routing problem in each epoch. First, we add penalty terms to (1) in Section 4.1, then present our VNS procedure in Section 4.2 after describing a new representation of the vehicle routes.

4.1. Cost function approximation

In this section we modify the reward function (1) by adding two penalty terms to it. On the one hand, we will penalize waiting for service at the factories, and on the other hand, the idle vehicles.

Penalizing waiting for service. If the number of vehicles is much larger than the docking capacity of the factories, the vehicles may have to spend a considerable time with queuing. However, waiting times may create large delays in delivery, therefore, it is better to avoid them. Let $\eta^{x,j}_{k,v}$ be the waiting time (i.e., the time between the arrival and the start of service) at the jth factory visited in the route plan $\theta^x_{k,v}$ of vehicle v. Then the total waiting time is

$$\mathbf{f}_3(s_k, x_k) = \sum_{v \in \mathcal{V}} \sum_{j=1}^{\ell^x_{k,v}} \eta^{x,j}_{k,v}.$$

Idle vehicles. We noticed that in some cases the assignment of the first orders significantly affects the subsequent delivery times. That is, intuitively good solutions (orders with a common pickup factory were assigned to the same vehicle) in the first epochs caused often irreversible tardiness for future orders. In those cases, it proved better to spread the initial orders between several vehicles, in order to keep as many vehicles moving as possible. We call a vehicle *idle*, if it has no destination factory, and it will be available in the next epoch. Then, the total number of idle vehicles is

$$\mathbf{f}_4(s_k, x_k) = |\{v \in \mathcal{V} : \theta^x_{k,v} = \emptyset \text{ and } td^{\text{curr}}_{k,v} < \tau_{k+1}\}|.$$

Perturbed reward function We will use the following reward function

$$R(s_k, x_k) = \lambda_1 \mathbf{f}_1(s_k, x_k) + \lambda_2 \mathbf{f}_2(s_k, x_k) + \lambda_3 \mathbf{f}_3(s_k, x_k) + \lambda_4 \mathbf{f}_4(s_k, x_k), \tag{3}$$

where $\lambda_1, \lambda_2, \lambda_3, \lambda_4 \geq 0$ are appropriate multipliers. The effect of penalty terms on the efficiency of our method will be investigated in Section 5.3.

4.2. New representation of routes and the VNS based procedure

Before delving into the details of our solution procedure, first we introduce the internal representation of vehicle routes.

Recall that a vehicle's route plan outlines a sequence of factories to visit, along with sets of orders to pickup and deliver at each location, as well as timing information. In this section we use a different, but equivalent representation. A *route* is a sequence of nodes, where each inner *node* refers to a pickup or a delivery of an order. That is, a *pickup node* represents the pickup of an order, and a *delivery node* corresponds to the delivery of some order. The first node of a route is associated with the orders carried by the vehicle, i.e., those orders already picked up but not yet delivered, and the last node marks the end of the route. We refer to Fig. 5 for

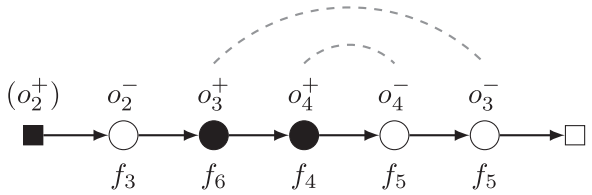


Fig. 5. Example for a route. The first and the last node is indicated with black and white rectangles, respectively. Each internal node represents a pickup or delivery, depicted with black or white circles, respectively. Above each pickup/delivery node the order, and below it the factory is indicated.

an illustration. It depicts the route of the vehicle corresponding to the center pane of Fig. 4. As we can see, the vehicle is on the way to factory f_3 to deliver the carried order o_2. After that, the vehicle picks up order o_3 at factory f_6 and order o_4 at factory f_4, and then delivers them in LIFO order to factory f_5.

A key gadget of our method is the insertion of an order into a route. Given a feasible route and some order o_i not in the route, the *insertion* of o_i into the route means that first the pickup node o_i^+ is inserted between two nodes of the route and then the delivery node o_i^- is inserted between two nodes of the updated route. An insertion is *feasible* if o_i^+ precedes o_i^- in the resulting route and the LIFO as well as the capacity constraints are satisfied, and the destination factory does not change (cf. Appendix B).

The *cost* of a solution is calculated by (3) throughout this section. It can computed by the procedure outlined in Appendix E.

After these preliminaries, our method consists of two main steps:

1) Construction of an initial set of routes for the vehicles (Section 4.2.1).
2) Improvement by variable neighborhood search (Section 4.2.2).

4.2.1. Construction of the initial set of routes

Since the solution obtained in the previous epoch could be a good starting point for the current epoch, first we reconstruct and update it. This involves removing delivery nodes associated with orders already fulfilled and pickups of orders that have already been collected.

Then, new orders are inserted into the updated solution one-by-one. First, the orders are divided into the sets of urgent and non-urgent orders, respectively. The classification is based on the *estimated delay* of an order $o_i \in \tilde{\omega}_k$:

$$ed_{k,i} = \left(t_i^d - \tau_k\right) - \left(h^{\text{docking}} + h_i^p + \text{travel}(f_i^p, f_i^d)\right).$$

The first term represents the remaining time to deliver order o_i without delay, while the second term expresses the minimum time needed to deliver order o_i. Let $\mathcal{U}_k = \{o_i \in \tilde{\omega}_k : ed_{k,i} \le U\}$ be the set of urgent orders, and $\overline{\mathcal{U}}_k = \tilde{\omega} \setminus \mathcal{U}$ the set of non-urgent ones, where U is a parameter of the algorithm.

First, the order in $\mathcal{U}_k$ are inserted into the routes of the vehicles, then those in $\overline{\mathcal{U}}_k$. When processing the orders in either category, orders are grouped by pickup factories, and if two or more orders have the same pickup factory, then priority is given to those order with a larger estimated delay. For each order $o_i \in \mathcal{U}_k \cup \overline{\mathcal{U}}_k$ the best feasible insertion is sought. That is, o_i is inserted in all feasible ways into the route of each vehicle in turn, and the insertion incurring the least cost increase is chosen.

4.2.2. Variable neighborhood search

In the following, we propose a method based on VNS to improve the initial set of routes. We commence by defining the neighborhood operators to be used within our VNS procedure.

The neighborhood operators rely on the notion of blocks and bridges that we define next. A *block* is a subsequence of consecutive nodes in a vehicle route such that the first and the last node refer to the pickup and the delivery of the same order, respectively. Refer to Fig. 6 for an illustration. Since orders o_1 and o_2 are already loaded on the vehicle, no pickup nodes, and thus no blocks correspond to these orders.

A *bridge* of a route is a subsequence of consecutive pickup nodes $(\ell_1, \dots, \ell_k)$, all belonging the the same factory, and a subsequence of consecutive delivery nodes $(r_k, \dots, r_1)$, all delivered to the same factory, where ℓ_j and r_j are the pickup and the delivery node of the same order, respectively, for $j = 1, \dots, k$. An example is depicted in Fig. 7. A bridge $\left((\ell_1, \dots, \ell_k), (r_k, \dots, r_1)\right)$ is *maximal*, if there is no bridge containing it properly. That is, neither $\left((\text{pred}(\ell_1), \ell_1, \dots, \ell_k), (r_k, \dots, r_1, \text{succ}(r_1)\right)$ nor $\left((\ell_1, \dots, \ell_k, \text{succ}(\ell_k), (\text{pred}(r_k), r_k, \dots, r_1)\right)$ constitutes a bridge, where $\text{pred}(n)$ and $\text{succ}(n)$ denote the immediate predecessor and the immediate successor of node n in the route, respectively.

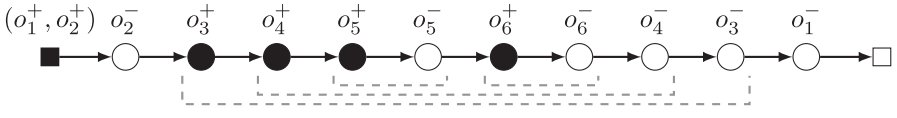


Fig. 6. A route with blocks indicated by dashed lines.

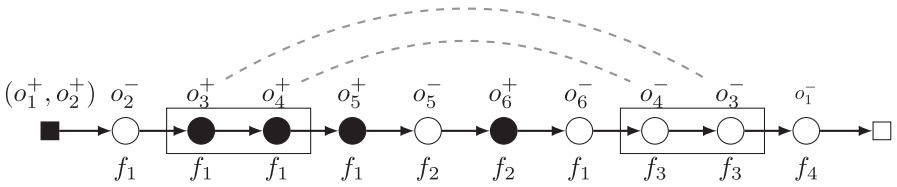
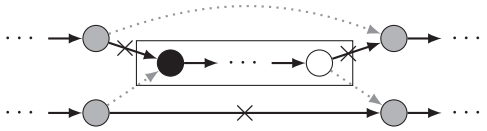
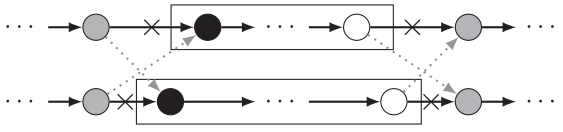


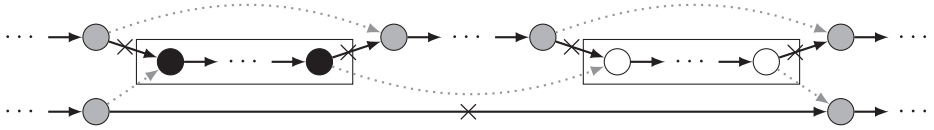
Fig. 7. Example for a (maximal) bridge. Left and right sequences of the bridge are framed by rectangles.



(a) Block relocation between routes.



(b) Block exchange between routes.



(c) Bridge relocation between routes. The two parts of the bridge will be adjacent after relocation.

Fig. 8. Example for schedule operations. Crossed/dotted arrows refer to old/new links.

A *neighborhood operator* perturbs a solution slightly and provides a modified solution. The following neighborhood operators will be applied to the solutions:

(i) The *relocate-block* operator selects a vehicle route and a block in the route, and relocates it to another position in the same or in another route (cf. Fig. 8a).
(ii) The *block-exchange* operator selects two disjoint blocks from the same or from distinct vehicle routes, and exchanges them as depicted in Fig. 8b.
(iii) The *relocate-bridge* operator selects a maximal bridge from a vehicle route and moves it into another position of the same route or to a different route as shown in Fig. 8c.

When applying an operator, the new position for a block or bridge is always chosen in such a manner that the resulting solution satisfies the LIFO constraint. However, if any other constraint is violated, the solution is dropped and the operator fails.

Algorithm 1 VNS procedure

1: **while** termination condition is not met **do**
2: Find the best neighbor S' of S using the *relocate-bridge* operator.
3: **if** S' has a lower cost than S **then**
4: $S := S'$ and proceed with Step 1.
5: Find the best neighbor S' of S using the *block-exchange* operator.
6: **if** S' has a lower cost than S **then**
7: $S := S'$ and proceed with Step 1.
8: Find the best neighbor S' of S using the *relocate-block* operator.
9: **if** S' has a lower cost than S **then**
10: $S := S'$ and proceed with Step 1.
11: **else**
12: Proceed with Step 13.
13: Output S.

Our VNS procedure, depicted in Algorithm 1, receives an initial feasible solution and tries to improve it repeatedly by applying the neighborhood operators. Throughout the algorithm, S denotes the current solution. The termination condition can be a time limit, or the number of iterations of the procedure. Solutions are compared based on their cost computed by the reward function (3). The neighborhood operators are applied exhaustively, i.e., they are applied in all possible ways to find the best neighboring solution. For instance, in case of the relocate-bridge operator, when seeking the best neighbor of S, in each route all maximal bridges are identified, and they are reinserted in the same or in a different route in all possible ways, and then the insertion producing the least cost feasible solution is chosen. If an operator cannot be applied, or fails to produce any feasible neighbor, then we assume that S' has a larger cost than S and proceed accordingly. Since only improvement of the current solution is allowed, cycling cannot occur, and the procedure terminates in a local minimum.

5. Computational experiments

In this section, we present the results our computational experiments. In Section 5.1, we describe our case study. In Section 5.2, we introduce the benchmark dataset. In Section 5.3, we tune the objective multipliers to get an efficient solution approach. We compare our best approach to other methods in Section 5.4. We present our sensitivity analysis in Section 5.5. Finally, we provide some key insights in Section 5.6.

Setup. All our experiments were performed on a workstation with an Intel Core i9-7960X 2.80 GHz CPU with 16 cores, under Debian 9 operating system using a single thread. Due to the 4 h time-window for the orders (see Section 5.2), we used parameter $U = 3600$ when urgent and non-urgent orders are determined for initial solution. Since each epoch is 10 min long, we applied a time limit of 9 min in our variable neighborhood search.

5.1. Case study: the dynamic pickup and delivery problem challenge

We consider *The Dynamic Pickup and Delivery Problem challenge* as a case study (Hao et al., 2022). This challenge was organized as a competition of the International Conference on Automated Planning and Scheduling (ICAPS) in 2021 with 152 participating teams. The best three teams (i.e., the teams whose algorithms achieved the smallest scores on a hidden dataset) presented their solution approaches in the conference and the challenge generated a series of papers in the scientific literature in the recent years.

The gold medal team, Zhu et al. (2021) proposed an RO method for the problem, where a variable neighborhood search was used to optimize the states with the currently known orders. In each epoch, the authors reconstructed the solution from the previous period, and dispatched new orders with a cheapest insertion heuristics. Four local search operators (couple-exchange, block-exchange, relocate-block, and multi-relocate) were used in a variable neighborhood search together with a disturbance operator to perturb solutions. Since then, the team published their results in (Cai et al., 2022a). The silver medal team, Ye and Liang (2021) developed an RO method for the problem, where a rule-based procedure was applied to dispatch orders. The bronze-medal team, Horváth et al. (2021) proposed a CFA method for the problem, where a local search-based approach was used to solve the problem with a perturbed objective function. In each epoch, the authors inserted non-dispatched orders into the solution with a cheapest insertion heuristic. Three local search operators (block-exchange, relocate-block, and a custom made) were used to improve solutions. An extra cost term was introduced to penalize if a factory is visited by too many vehicles, thereby reducing the chance of waiting for free docking ports.

Cai et al. (2022b) propose an RO method for the DPDP, where a reference point-based multi-objective evolutionary algorithm is used to solve the states. Four local search operators are used in a variable neighborhood search to improve solutions as in (Cai et al., 2022a). The authors compare their solution approach to the algorithms of the podium teams. For the comparison, the authors use the benchmark dataset with the exception of the largest instances. Recently, Cai et al. (2023) provide a review of the dynamic pickup and delivery problem literature covering the last two decades. As a case study, the authors provide the comparison of Cai et al. (2022b) on the full benchmark dataset.

Du et al. (2023) propose an RO method for the DPDP with a hierarchical optimization approach. The authors are the first to model the problem as an SDP. They apply several order dispatching strategies to assign orders to vehicles. A buffering pool is also used to postpone the assignment of some non-urgent requests. A local search operator is introduced to improve the solution. The authors compare their solution approach to a greedy baseline strategy, and to the silver-winning algorithm of Ye and Liang (2021) (which was falsely claimed to be the best algorithm of the competition). For the comparison, the authors use only a selection of the benchmark dataset, but also generated new problem instances.

5.2. Instances

We use the publicly available dataset of the ICAPS 2021 DPDP competition (Hao et al., 2022). This dataset consists of 64 instances based on 30 days of historical data of Huawei. These instances contain 50-4000 orders of a single day to be satisfied with 5-100 vehicles, see Table 1. In the following, we briefly describe the public instances, while for detailed description we refer to (Hao et al., 2022).

In case of all instances, there are 4 h to complete an order on time, that is, for each order $o_i \in \mathcal{O}$ we have $t_i^d - t_i^p = 14\,400$ s. Each order is given as a set of *order items*, where each item is either a *box*, a *small pallet*, or a *standard pallet*, with quantity of 0.25, 0.5, and 1 unit, respectively, while the uniform capacity limit of the vehicles is 15 units. Loading or unloading a box, a small pallet, and a standard pallet takes 15, 30, and 60 s, respectively. The underlying network is the same for all instances and consists of 153

Table 1
Basic properties of the benchmark dataset.

Group	Instances	Orders	Vehicles
1	1–8	50	5
2	9–16	100	5
3	17–24	300	20
4	25–32	500	20
5	33–40	1000	50
6	41–48	2000	50
7	49–56	3000	100
8	57–64	4000	100

factories, where each factory has 6 docking ports. Docking to a port takes half an hour (i.e., $h^{\text{docking}} = 1800$ s). Finally, $\lambda_1 = 1/n$ and $\lambda_2 = 10\,000/3600 \approx 2.78$ in the objective function (2), that is, a cost of 10000 monetary units must be paid for every hour of delay.

Note that the total size of the items of an order may exceed the uniform capacity of the vehicles. In this and only this case, the order items can be transported separately. The completion time of such an order is the latest completion time of its order items. For such big orders, we arranged items in a non-increasing size order, then we applied a first fit procedure to divide items into separate orders. By this, we got the same problem as proposed earlier, with the tiny difference that the tardiness has to be calculated differently for split orders.

Evaluation by simulation. The organizers of the ICAPS 2021 DPDP competition also provided a simulator, implemented in Python programming language, to support the dynamic evaluation of the vehicle routing algorithms. The simulator essentially follows the sequential decision procedure described in Section 3.1. It mimics the movement, loading and unloading of the vehicles with sufficient accuracy for decision making. The operating horizon is divided into 10-minutes long epochs. At the beginning of each epoch, the simulator recomputes the state of each vehicle and transportation request. Then, it invokes the vehicle routing algorithm. The algorithm receives the status information, and updates the routes of the vehicles. After termination, it has to return a set of vehicle routes to the simulator, which uses them in the next epoch for computing the vehicles' states. The final output of the simulator is the score of the dispatching algorithm on the given instance. For details, we refer to (Hao et al., 2022).

Real-time decision support. The simulator can be replaced by a decision support system for dispatching a fleet of vehicles in the real-world. The status information of the vehicles and that of the transportation requests can be easily maintained using modern IT technology. The periodic invocation of vehicle routing algorithms is a routine exercise, and combined with proper visualization, it can be a very effective decision support tool.

5.3. Evaluation of the cost function approximation method

In the following, we evaluate our method with different λ_3 and λ_4 parameters with the aim of finding the best settings.

5.3.1. Significance of penalizing the waiting times

In these experiments, we suppressed the fourth term in the objective function (3). That is, we ran our method with objective function coefficients $\lambda_1 = 1/n$, $\lambda_2 = 2.78$, $\lambda_4 = 0$, and λ_3 chosen from $\{0.0, 0.25 \times \lambda_2, 0.5 \times \lambda_2, 0.75 \times \lambda_2\}$. The average scores for each group, and for all instances are indicated in Table 2, while detailed results can be found in the supplementary data.

Clearly, penalizing waiting times has no impact on the first two groups, since the number of docking ports is greater than the number of vehicles, and thus no waiting occurs. On instances with 20 and 50 vehicles (Groups 3–6), the differences are negligible. However, in case of 100 vehicles (Groups 7-8), penalizing idle times significantly reduced the average score. On the largest group, the average improvement is 60–62%.

Now we analyze in more detail the impact of penalizing waiting times on problem `instance_61` which has 4000 orders and 100 vehicles. The instance is depicted in Fig. 9 using the spring layout. The nodes represent factories, and the edges pickup and

Table 2
Evaluation of penalizing waiting times.

	Multiplier for waiting times (λ_3)			
Group	$0 \times \lambda_2$	$0.25 \times \lambda_2$	$0.5 \times \lambda_2$	$0.75 \times \lambda_2$
1	1 307.5	1 307.5	1 307.5	1 307.5
2	31 707.6	31 707.6	31 707.6	31 707.6
3	690.0	690.0	690.0	690.0
4	6 750.9	6 750.5	6 750.5	6 750.5
5	11 597.1	11 225.6	11 225.6	11 225.6
6	49 011.9	53 183.2	54 091.5	49 841.4
7	1 076 526.1	590 379.6	738 200.9	769 756.1
8	11 605 754.6	4 696 315.2	4 618 058.6	4 412 835.5
average:	1 597 918.2	673 944.9	682 754.0	660 514.3

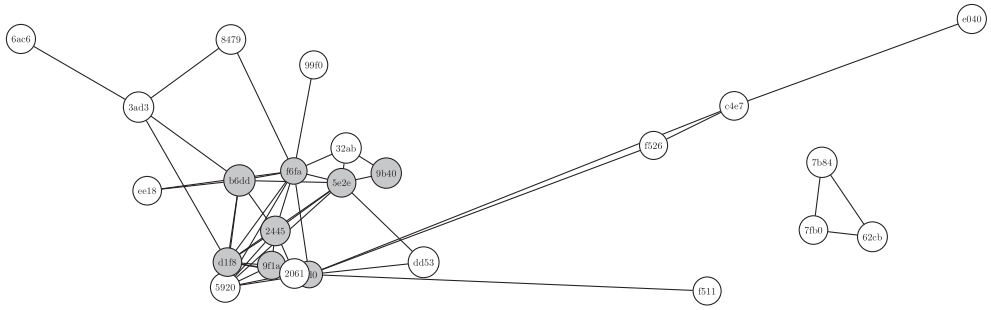


Fig. 9. `Instance_61` depicted in spring layout, where the edge weights are determined by the number of orders between the pairs of factories represented by the nodes.

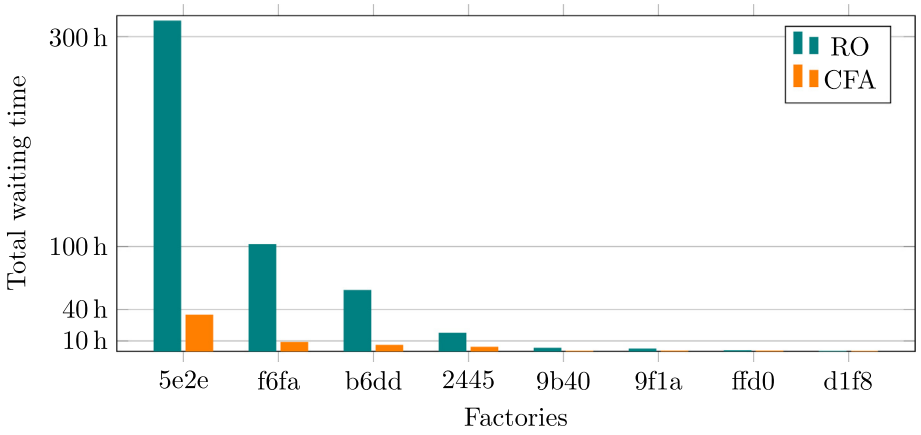


Fig. 10. Total waiting times at selected factories of `instance_61`.

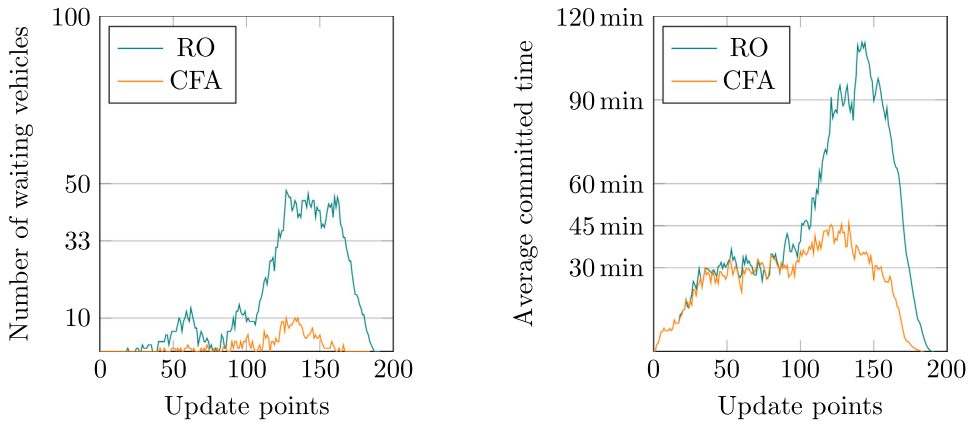
delivery requests between pairs of nodes. The edge lengths are inversely proportional to the number of pickup and delivery requests between their endpoints. This means that the more is the number of pickup and delivery requests between a pair of nodes, the shorter is the edge connecting them. As we can see, there is a dozen of factories with a lot of requests among them, and there are a number of "satellite" factories around connected to the center with much less transportation requests. We can also observe a triangle on the right not connected to any other factories.

We run the rolling horizon approach (RO), and the cost-function approximation approach (CFA) on this instance, where RO uses the original reward function (1), and CFA uses the perturbed reward function (3) with $\lambda_3 = 0.75 \times \lambda_2$ when solving the vehicle routing problem at each epoch. The largest waiting times occur at the factories colored gray in Fig. 9. The total waiting times at these factories are depicted in the bar chart in Fig. 10. As we can see, the CFA drastically decreases the waiting times.

A further way to compare the solutions of the RO and the CFA method is who the total number of waiting time and the committed time of the vehicles change over time. The *committed time* of a vehicle at an update time point is the length of the time interval that the next decision will no longer affect. For example, if vehicle v is at a location at update time τ_k, then the vehicle is committed until its earliest departure time $td^{\text{curr}}_{k,v}$, i.e., the committed time is $td^{\text{curr}}_{k,v} - \tau_k$. If the vehicle is on the way, then it is committed until the end of its service at the destination factory, that is, its committed time is $td^{1}_{k,v} - \tau_k$. In Fig., we depict for each update point the number of those vehicles that are currently waiting at a location for a free docking port, and the average committed time of the vehicles, respectively. Both indicators are suitable for measuring the flexibility of routes.

In Fig. 11a, we can see that without explicitly penalizing the waiting times, there is a long, namely a 45-epoch (7.5 h) period, when the number of waiting vehicles is between 34 and 48. That is, on one third of the planning horizon, 30 to 50% of the vehicles are waiting. When the waiting times are penalized in the reward function, the number of waiting vehicles decreases significantly, namely, never exceeds 10.

The inflexibility of routes in the myopic approach is even more noticeable in the other figure. According to Fig. 11b, in some cases the average committed time of the vehicles is more than one and a half hours. There are vehicles which have to wait more than 6 h for a free docking port. Consequently, if those vehicles came to that factory to pickup some orders, then those orders will be delivered at least 2 h after their due date, inherently. Explicitly penalizing waiting times also significantly decreases the average committed time, which never exceeds 46 min.



(a) Number of waiting vehicles.

(b) Average committed time.

Fig. 11. Number of waiting vehicles and average committed time obtained by the RO and by CFA method, respectively.

Table 3
Evaluation of penalizing idle vehicles.

	Multiplier for idle vehicles (λ_4)		
Group	0	5	10
1	1 307.5	979.5	1 076.0
2	31 707.6	28 532.8	20 265.7
3	690.0	887.5	887.5
4	6 750.9	6 658.2	6 658.2
5	11 597.1	2 969.8	2 969.8
6	49 011.9	21 919.9	21 919.9
7	1 076 526.1	4 856 878.2	4 909 406.0
8	11 605 754.6	15 046 689.2	15 046 689.2
average:	1 597 918.2	2 495 689.4	2 501 234.0

5.3.2. Penalizing the idle vehicles

We ran our method with objective function $1/n \times \mathbf{f}_1 + \lambda_2 \times \mathbf{f}_2 + \lambda_4 \times \mathbf{f}_4$ for all instances. The multiplier λ_4 for the number of idle vehicles was chosen from $\{0, 5, 10\}$. The average scores for each group, and for all instances are indicated in Table 3, while detailed results can be found in the supplementary data.

In contrast to the previous experiments, penalizing idle times has a bad impact on the largest instances (Groups 7-8) but can improve other ones. The average improvement on instances with 5 vehicles (Groups 1-2) is 10-36%, and on instances with 50 vehicles (Groups 5-6) is 55-74%.

5.3.3. Parameter tuning

As we can see, the introduced two penalty terms behave differently on the groups. We tested our algorithm for each pair (λ_3, λ_4) of parameters from the set $\{0, 0.25 \times \lambda_2, 0.5 \times \lambda_2, 0.75 \times \lambda_2\} \times \{0, 5, 10\}$. Summarized results are indicated in Table 4, while detailed results can be found in the supplementary data. We obtained the best results (in the sense of total average score) with parameters $\lambda_3 = 0.5 \times \lambda_2$ and $\lambda_4 = 10$.

Note that since the number of vehicles is part of the input, we could use different parameter settings for the different groups. However, we do not take advantage of this opportunity in the following, but use the same parameters for all groups.

Table 4
Averages scores for different (λ_3, λ_4) parameter pairs.

$\lambda_3 \backslash \lambda_4$	0	5	10
$0 \times \lambda_2$	1 597 918.2	2 495 689.4	2 501 234.0
$0.25 \times \lambda_2$	673 944.9	688 973.7	693 949.2
$0.5 \times \lambda_2$	682 754.0	644 685.2	643 520.0
$0.75 \times \lambda_2$	660 514.3	656 553.3	659 295.8

Table 5
Comparison of the solution approaches on the benchmark dataset. Best average scores are in bold.

Group	CFA-VNS	Gold	Silver	Bronze	VNSME	MOEA/D-ES
1	1 076.0	2 896.4	13 676.2	1 763.8	1 036.8	**1 024.2**
2	20 265.7	41 535.3	599 932.8	62 180.2	36 765.0	**14 182.4**
3	888.4	5 860.4	2 310.7	8 969.7	691.9	**686.0**
4	7 456.1	6 544.5	105 049.4	26 938.3	7 909.1	**5 489.9**
5	**3 159.2**	10 459.1	17 284.3	94 794.8	10 541.2	9 673.9
6	**16 178.2**	41 494.3	153 419.1	651 945.0	42 375.0	25 362.5
7	**632 913.0**	798 240.7	904 586.3	1 941 385.3	988 012.5	787 050.0
8	**4 466 223.7**	11 359 466.4	18 678 529.1	15 122 816.2	10 337 500.0	10 260 000.0
average	**643 520.0**	1 533 312.1	2 559 348.5	2 238 849.2	1 428 104.0	1 387 933.6

Gold, Silver, Bronze: Public algorithms of the ICAPS 2021 DPDP Competition VNSME: (Cai et al., 2022a)
MOEA/D-ES: (Cai et al., 2022b; 2023)

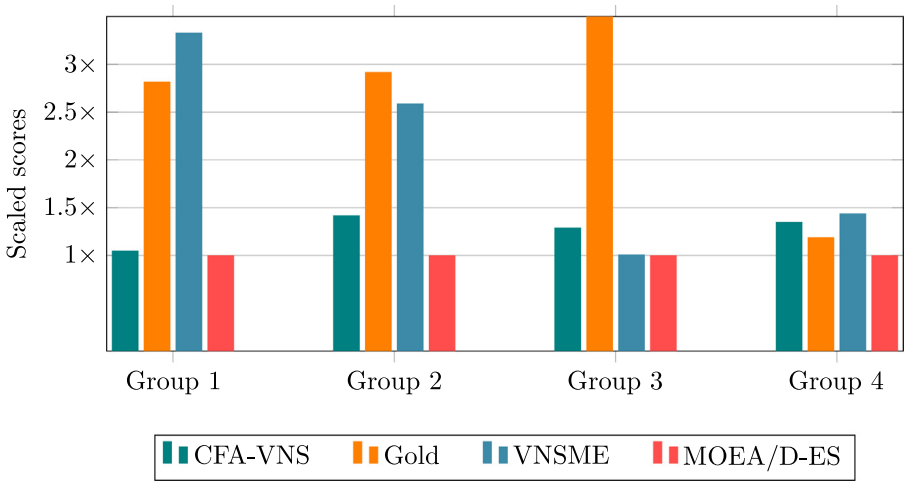


Fig. 12. Average scores on smaller instances are scaled to the solution approach MOEA/D-ES of Cai et al. (2022b, 2023).

5.4. Comparison of existing methods

In the following, we compare our method to the existing solution approaches. In Table 5, we indicate the average group scores of our best method (*CFA-VNS*), the algorithms of the podium teams of the DPDP competition (*Gold, Silver*, and *Bronze*), the variable neighborhood search of Cai et al. (2022a) (*VNSME*), and the multi-objective evolutionary method of Cai et al. (2022b, 2023) (*MOEA/D-ES*). For better understanding, we depict the scaled average scores of the best performing methods in Figs. 12 and 13. The first four algorithms (CFA-VNS, Gold, Silver, and Bronze) were tested in the same environment, since the algorithms of the podium teams are publicly available. Note that the available algorithm of Ye and Liang (2021) failed on seven instances of Group 2 due to a technical constraint of the simulator, thus the results for those instances are obtained by turning off that constraint. Note that methods Gold and VNSME are the same, but the values corresponding to VNSME are obtained from the results in (Cai et al., 2022a). Method VNSME was executed on a workstation with an Intel Core i5-9500 3.00 GHz CPU with 4 cores, under Ubuntu 18.04 LTS operating system. According to CPU benchmarks[2], the single thread rating is 2499 for our processor, and 2571 for the other one. The running environment of method MOEA/D-ES is unknown, the corresponding values are obtained from the results in (Cai et al., 2023).

Methods CFA-VNS and MOEA/D-ES outperform the other solution approaches on this benchmark dataset. On the smaller instances (Groups 1-4), MOEA/D-ES turned out to be the best performing approach as it obtained 5-30% better average scores on the small instances than CFA-VNS. See Fig. 12, where we depict the average scores scaled to MOEA/D-ES. In contrast, on the larger instances (Groups 5-8), our approach was the best as it achieved 20-67% better average scores on the larger instances. See Fig. 13, where the average scores are scaled to CFA-VNS. The average improvement of CFA-VNS on the full dataset over the second-best method is 54%.

Du et al. (2023) provide quantitative results for a selection of 9 instances from the ICAPS 2021 DPDP competition. Moreover, they compare their results to that of Ye and Liang (2021), the silver-winning team of the competition. In Table 6 we indicate the scores for these selected instances of our best method (*CFA-VNS*), and the hierarchical method of Du et al. (2023) (*Hierarchical*). For the sake of completeness, we also indicate the scores for the variable neighborhood search of Cai et al. (2022a) (*VNSME*), and the multi-objective evolutionary method of Cai et al. (2022b, 2023) (*MOEA/D-ES*). Note that the results in the latter papers are provided

[2] https://www.cpubenchmark.net.

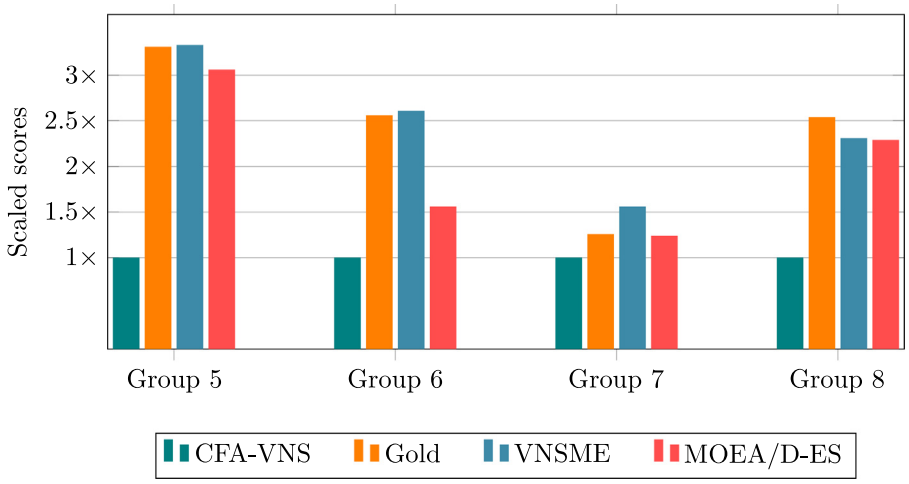


Fig. 13. Average scores on larger instances are scaled to our solution approach CFA-VNS.

Table 6
Comparison of the solution approaches on Du et al.'s selection.

Instance	CFA-VNS	VNSME[a]	MOEA/D-ES[a]	Hierarchical
38	**3 793.1**	15 000.0	14 200.0	28 121.0
39	**5 962.1**	14 900.0	17 400.0	24 013.0
40	**7 049.6**	10 200.0	13 100.0	23 338.0
41	**7 939.8**	29 300.0	28 300.0	126 292.0
44	**34 649.0**	77 800.0	36 200.0	185 288.0
45	**10 244.7**	36 300.0	22 800.0	185 909.0
49	788 852.0	1 580 000.0	**630 000.0**	1 082 139.0
54	**621 130.3**	2 070 000.0	1 630 000.0	1 619 514.0
55	530 422.7	**323 000.0**	1 000 000.0	678 125.0

[a] VNSME: (Cai et al., 2022a) MOEA/D-ES: (Cai et al., 2022b; 2023) Hierarchical: (Du et al., 2023) values were provided in 2-decimal exponential format (i.e., $E+n$)

in a 2-decimal exponential format (e.g., 1.42E+4), thus the values indicated in Table 6 for these methods are approximated values. We can see that CFA-VNS produced better scores on 8 out of 9 instances than VNSME and MOEA/D-ES, respectively. The difference compared to Hierarchical is even greater, as CFA-VNS gave 22-94% better results.

Also note that Du et al. (2023) generated 8 new instances with 50 orders and 5 vehicles, and 8 new instances with 2000 orders and 50 vehicles. The authors also make comparisons on this new dataset, however, they do not provide any numerical results. Only bar charts are given, from which it is difficult to read even approximate values. Since we could not get access to more detailed numerical results so far, we cannot provide comparison with our solution approach on this dataset.

5.5. Sensitivity analysis

We performed some tests to assess the sensitivity of our method to the run-time limit as well as to the perturbation of the data. As we mentioned, due to the length of the decision epochs (10 min), we apply a 9-minute run-time limit in our variable neighborhood search. Therefore, it may occur that during distinct runs the method terminates with different solutions in a decision epoch, which may influence the subsequent epochs and thus on the final solution. In consequence, our method is not deterministic in the sense that distinct runs may result distinct scores. Thus, we first investigated how deterministic our method is in Section 5.5.1.

In a different set of experiments, we perturbed the instances, and run our method on the modified inputs, the results are summarized in Section 5.5.2.

For these experiments, we chose 2-2 instances from the two largest groups, namely instances 49, 50, 61, and 62.

5.5.1. Impact of run-time limit

To check how deterministic our method is, we repeated the simulation 5 times for each instance. In all cases, we got the same result in each of the 5 runs for the same instance. Therefore, we can state that the method is deterministic on the chosen instances, and we expect the same behavior in most cases.

Table 7
Sensitivity analysis: perturbed instances.

	Instance 49	Instance 50	Instance 61	Instance 62
$p = 0$	1 038 609.4	828 236.4	3 499 727.4	3 717 233.1
$p = 0.1$	843 924.4	848 902.0	3 287 735.0	3 350 250.2
	975 795.1	895 422.6	3 353 655.2	3 361 829.9
	926 224.4	1 032 660.9	4 174 461.7	3 723 564.2
	828 752.2	1 013 947.7	3 678 377.0	3 217 902.2
	926 095.3	785 364.6	3 105 979.1	3 530 272.6
avg	900 158.3	915 259.6	3 520 041.6	3 436 763.8
std.dev	55 379.0	95 070.5	375 881.4	174 355.1
$p = 0.25$	1 037 437.1	745 218.8	3 188 694.0	3 579 119.2
	979 367.0	828 345.0	3 333 017.3	3 138 076.2
	909 484.3	892 606.9	3 818 538.0	3 625 110.4
	997 086.7	1 040 584.6	3 267 811.5	3 759 841.7
	891 435.3	783 811.6	3 514 348.2	3 668 758.5
avg	962 962.1	858 113.4	3 424 481.8	3 554 181.2
std.dev	54 692.0	103 573.8	224 438.6	216 434.9
$p = 0.5$	1 116 028.0	831 807.5	3 245 883.4	3 942 387.9
	1 006 283.4	994 466.5	3 451 303.7	3 843 628.3
	898 703.0	734 223.0	2 935 480.3	3 780 042.1
	932 487.2	837 958.4	3 684 915.5	3 387 376.5
	942 723.0	852 650.6	3 395 947.6	3 502 457.5
avg	979 244.9	850 221.2	3 342 706.1	3 691 178.5
std.dev	76 743.5	83 375.8	247 767.2	210 786.7

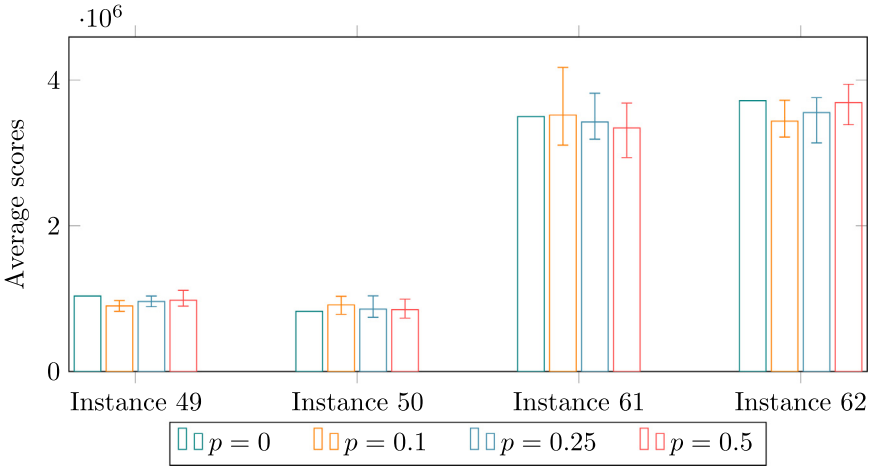


Fig. 14. Sensitivity analysis: perturbed instances. Average scores and minimum/maximum values.

We also examined how the run-time limit affects the quality of the final solutions. For each instance, we performed tests where we set reduced run-time limit for the variables neighborhood search. We made tests with run-time limits of 3 and 6 min as well. Surprisingly, we found that for each instance, we got again the same result in every run.

5.5.2. *Perturbed instances*

For each instance in the chosen dataset, we created 5 perturbed instances, where the release dates of the orders are shifted up to 20 min. That is, each order o_i is selected to be shifted with a given probability p, and the new release date of the order is chosen uniformly at random from the following set

$$\{\max\{t_i^p - 1200, 0\}, \dots, \min\{t_i^p + 1200, 86\,399\}\},$$

where 0 and 86 399 refer to the beginning and the end of the day in seconds, respectively. The due dates are adjusted accordingly. We performed experiments with $p = 0.1$, $p = 0.25$ and $p = 0.5$.

The results are provided in Table 7, where the final scores are indicated for the experiments, along with the average value (*avg*) and the *standard deviation* (*std.dev*) for each instance. The first row with $p = 0$ refers to the original, non-perturbed instances. We

also depict the result in Fig. 14, where the bar plots refer to the average values, and the error intervals refer to the minimum and maximum values.

The standard deviation compared to the average value is $5.1\% - 10.7\%$ for $p = 0.1$, $5.7\% - 12.1\%$ for $p = 0.25$, and $5.7\% - 9.8\%$ for $p = 0.5$. We think that these values are low, compared to how many orders can be shifted. For example, in case of $p = 0.5$, each order is shifted with a probability of 0.5, nevertheless, the standard deviation is lower than 10% for each instance.

5.6. Key insights

Our computational results indicate that solving a dynamic vehicle routing problem necessitates a thorough analysis of both the problem data and the resulting solutions. This process led us to introduce penalty terms into the objective function. We believe that formulating the appropriate objective function is at least as important as the specific details of the optimization algorithm used to solve these problems.

In the problem studied we introduced penalty terms for minimizing the waiting times of the vehicles at the facilities, and also for minimizing the number of idle vehicles. While the former one is only effective if the docking capacity of some of the locations is a bottleneck, the latter penalty is more effective if the number of vehicle is moderate (at most 50 in our benchmark instances), while it is rather counterproductive when the number of vehicles is large (100 in our dataset). However, using the two penalty terms simultaneously yielded better results than applying each one individually.

6. Conclusions

In this paper we investigated a dynamic pickup and delivery problem with docking constraints and the LIFO rule. We proposed a cost function approximation (CFA) method for the problem, where we perturbed the objective function to make solutions flexible for future changes. The main contribution of our method are the two penalty terms added to the cost function that penalize waiting for service at the locations and also if the vehicles are idle. We proposed a variable neighborhood search with three LIFO-specific local search operators to solve problems at the states with the perturbed objective function. As a case study, we evaluated our solution procedure on the instances of the ICAPS 2021 DPDP competition. The computational experiments show that our method significantly outperforms the other solution approaches on this dataset. The average improvement over the state-of-the-art on the full dataset is more than 50%. Our method is especially good on the largest (and hardest) instances.

While no exploitable probabilistic information may be available in a single problem instance, statistical data about the distribution of orders in time and space might be exploited even better than our current method does. This is an excellent subject for future research.

In an online setting with docking constraints, it seems difficult to handle hard time windows for the requests, which is a limitation of our method.

Our method is easy to extend with other constraints, such as FIFO constraints instead of LIFO constraints, different vehicle capacities, or travel times depending on the time of day. Such constraints frequently occur in practice, providing a wide application area for the method proposed.

An emerging trend is the combined operation of trucks and drones, we refer to Wang and Sheu (2019) and Wandelt et al. (2023). In this topic several static and dynamic problems appear, e.g., the vertiport location problem for urban air mobility (UAM) systems (Jin et al., 2024). The approach proposed in this paper may be further extended in this direction.

Declaration of competing interest

The authors declare that they had no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

CRediT authorship contribution statement

Markó Horváth: Writing – original draft, Software, Methodology, Formal analysis, Conceptualization. **Tamás Kis:** Writing – original draft, Supervision, Methodology, Formal analysis, Conceptualization. **Péter Györgyi:** Validation, Methodology, Formal analysis, Conceptualization.

Acknowledgments

This research has been supported by the TKP2021-NKTA-01 NRDIO grant on ”Research on cooperative production and logistics systems to support a competitive and sustainable economy”. Markó Horváth acknowledges the support of the János Bolyai Research Scholarship.

Appendix A. Serving vehicles at factories

In this section we describe the method of serving the vehicles at a factory in detail. Recall that each factory has a given number of docking ports for loading and unloading, and the vehicles are served in non-decreasing arrival time order, where ties are resolved randomly.

Given a factory with c ports for loading and unloading the vehicles, and suppose the current time is t. Let $\mathcal{L} = (v_1, v_2, \dots, v_k)$ be the *reservation list* consisting of those vehicles which are currently at the factory. Let td_i be the earliest departure time of vehicle v_i. $\mathcal{L}$ is ordered in non-decreasing departure time order, that is, $td_i \le td_{i+1}$ for $1 \le i \le k-1$. Suppose vehicle v_{k+1} arrives at the factory at time t. Finally, let st_{k+1} denote the service time of vehicle v_{k+1}, which is the sum of the dock approaching time, the unloading time and the loading time.

If $k < c$, the vehicle v_{k+1} immediately starts to approach a free port. The waiting time is zero, and the earliest departure time is $td_{k+1} = t + st_{k+1}$. Otherwise, if $k \ge c$, then all ports are occupied, and the vehicle must wait until vehicles $v_1, v_2, \dots, v_{k-c+1}$ finish, that is, until time td_{k-c+1}. Thus, the waiting time is $td_{k-c+1} - t$, and the earliest departure time is $td_{k+1} = td_{k-c+1} + st_{k+1}$. In both cases, the vehicle is inserted into the appropriate position of the reservation list. When a vehicle finishes, it is removed from the list.

Example. In Fig. A.15 we depict a situation where four vehicles arrive at a factory with two docking ports. Vehicle v_1 arrives at the factory at time t_1 and occupies a free docking port ($td_1 = t_6$, $\mathcal{L} = (v_1)$). Vehicle v_2 arrives at time t_2 and occupies the other free port ($td_2 = t_5$, $\mathcal{L} = (v_2, v_1)$). Vehicle v_3 arrives at time t_3, however, since both ports are in use (that is, currently vehicle v_1 is under unloading at the first port, and vehicle v_2 approaches the other one), it has to wait until a port is freed at time t_5 ($td_3 = t_7$, $\mathcal{L} = (v_2, v_1, v_3)$). Vehicle v_4 arrives at time t_4, however, both ports are in use, moreover, vehicle v_3 is already allocated to the port becoming free at time t_5, thus, it has to wait until time t_6, when loading is finished at the first port ($td_4 = t_8$, $\mathcal{L} = (v_2, v_1, v_3, v_4)$).

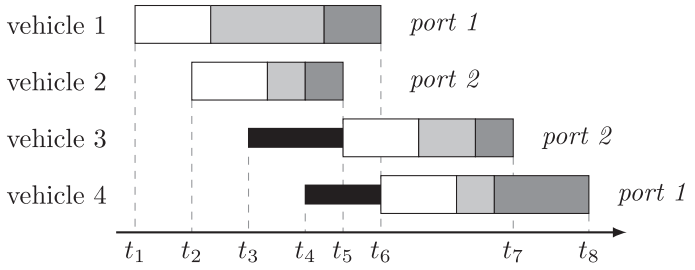


Fig. A.15. Example for vehicles arriving at a factory with two docking ports. White rectangles represent dock approaching, gray rectangles represent unloading and loading orders, black rectangles represent waiting for ports to become free.

Appendix B. Feasibility of route plans in states

In this section we characterize feasible route plans in some state s_k.

Orders. For a vehicle v, let $\mathcal{O}_{k,v}$ be the set of those orders that are carried by the vehicle or picked up in its route plan $\theta_{k,v}$. That is, $\mathcal{O}_{k,v} = C_{k,v} \cup \bigcup_{j=1}^{\ell_{k,v}} \mathcal{P}^j_{k,v}$. These, and only these orders must be delivered by the vehicle in the route plan, that is,

$$\bigcup_{j=1}^{\ell_{k,v}} \mathcal{D}^j_{k,v} = \mathcal{O}_{k,v}.$$

Moreover, the sets $\mathcal{P}^j_{k,v}$, $\mathcal{P}^{j'}_{k,v}$ must be disjoint for $j \ne j'$, and $C_{k,v}$ must be disjoint from $\bigcup_{j=1}^{\ell_{k,v}} \mathcal{P}^j_{k,v}$. Further on, $C_{k,v} \cup \bigcup_{j=1}^{\ell_{k,v}} \mathcal{P}^j_{k,v}$ must be disjoint from $C_{k,w} \cup \bigcup_{j=1}^{\ell_{k,w}} \mathcal{P}^j_{k,w}$ for distinct vehicles $v \ne w$. Clearly, orders must be picked up before their delivery, that is, if $o_i \in \mathcal{P}^{j_1}_{k,v}$ for some j_1, then $o_i \in \mathcal{D}^{j_2}_{k,v}$ for some $j_1 < j_2$. Finally, $\bigcup_{j=1}^{\ell_{k,v}} \mathcal{P}^j_{k,v}$ must be a subset of $\tilde{\omega}_k$ for each $v \in \mathcal{V}$.

LIFO constraint. Let $\mathcal{L}_{k,v}$ be the concatenation of lists $C_{k,v}$, $\mathcal{D}^1_{k,v}$, $\mathcal{P}^1_{k,v}$, …, $\mathcal{D}^{\ell_{k,v}}_{k,v}$, $\mathcal{P}^{\ell_{k,v}}_{k,v}$. Let $\mathrm{pos}(o_i^+)$ and $\mathrm{pos}(o_i^-)$ denote the position of the first and the second (i.e., last) occurrence of order $o_i \in \mathcal{O}_{k,v}$ in $\mathcal{L}_{k,v}$, respectively. Then, route plan $\theta_{k,v}$ satisfies the LIFO constraint, if

$$\mathrm{pos}(o_i^+) < \mathrm{pos}(o_j^+) \Rightarrow \mathrm{pos}(o_i^-) \le \mathrm{pos}(o_j^+) \vee \mathrm{pos}(o_j^-) \le \mathrm{pos}(o_i^-)$$

holds for all $o_i, o_j \in \mathcal{O}_{k,v}$.

Capacity constraint. The route plan $\theta_{k,v}$ satisfy the capacity constraint, if the total quantity of the loaded orders never exceeds the vehicle's capacity. That is,

$$\sum_{o_i \in C_{k,v}} q_i + \sum_{j=1}^{\ell} \left(\sum_{o_i \in \mathcal{P}^j_{k,v}} q_i - \sum_{o_i \in \mathcal{D}^j_{k,v}} q_i \right) \le Q$$

holds for all $\ell = 1, \dots, \ell_{k,v}$.

Fundamental routing constraints. The travel time between factories are fixed:

$$ta^j_v - td^{j-1}_v = \mathrm{travel}(f^{j-1}_v, f^j_v) \quad \text{for all } j = 1, \dots, \ell_v.$$

Let η_v^j be the waiting time of the vehicle at the jth visited factory.

$$ta_v^j + \eta_v^j + h^{\text{docking}} + \sum_{o_k \in \mathcal{D}_v^j} h_k^d + \sum_{o_k \in \mathcal{P}_v^j} h_k^p \leq td_v^j$$

Appendix C. Transition

In the following, we formally describe the transition from state s_k to state s_{k+1} according to action x_k, postponed from Section 3.1.6. The various cases are summarized in Table C.8, and explained in the following. Recall that $\theta_{k,v}^i$ refers to the ith visit of vehicle v in its route plan belongs to state s_k, and $\theta_{k,v}^{x,i}$ refers to the ith visit of the route plan belongs to action x_k.

Table C.8
Transition from state s_k into state s_{k+1} according to action x_k.

	state s_k		action x_k		state s_{k+1}		
Case	$\phi_{k,v}^{\text{curr}}$	$\theta_{k,v}^1$	$\theta_{k,v}^{x,1}$	$\theta_{k,v}^{x,2}$	τ_{k+1}	$\phi_{k+1,v}^{\text{curr}}$	$\theta_{k+1,v}^1$
1	yes	$\star$	$\star$	$\star$	$\tau_{k+1} < td_{k,v}^{\text{curr}}$	$\phi_{k,v}^{\text{curr}}$	$\theta_{k,v}^1$
2a	yes	$\star$	no	no	$td_{k,v}^{\text{curr}} \leq \tau_{k+1}$	$(f_{k,v}^{\text{curr}}, \tau_{k+1})$	no
2b	yes	$\star$	yes	$\star$	$td_{k,v}^{\text{curr}} \leq \tau_{k+1}$	no	$\theta_{k,v}^{x,1}$
3	no	yes	yes	$\star$	$\tau_{k+1} < ta_{k,v}^1$	no	$\theta_{k,v}^1$
4	no	yes	yes	$\star$	$ta_{k,v}^1 \leq \tau_{k+1}$	$(f_{k,v}^1, td_{k,v}^1)$	$\theta_{k,v}^{x,2}$

[$\star$] could be 'yes' or 'no' (i.e., 'given' or 'not given'). Recall that if $\theta_{k,v}^1$ is given, so is $\theta_{k,v}^{x,1}$ with $f_{k,v}^1 = f_{k,v}^{x,1}$ and $ta_{k,v}^1 = ta_{k,v}^{x,1}$.

Case 1. The vehicle is not finished at its current factory. If the vehicle had a current factory and the associated earliest departure time is later then the current update point, then the current factory remains the same as in the previous state, and the destination is the one specified in the previous action, if any.

Case 2. The vehicle is finished at its current factory. Assume that the vehicle had a current factory and the associated earliest departure time has passed. If the vehicle was not assigned a destination factory in the previous action (Case 2a), then the vehicle remains parked at this factory, but will be immediately available. Otherwise (Case 2b), the vehicle is already on the way to its destination specified in the last action.

Case 3. The vehicle is not reached its destination factory. If a vehicle was on the way to its destination factory, and did not reach it in the last epoch, then the destination remains the same (however, the list of orders to pickup may be changed in the action).

Case 4. The vehicle has reached its destination factory. Assume that the vehicle was on the way to its destination factory, and reached that in the last epoch. Then, that former destination becomes the current factory, and earliest departure time is also calculated. If the vehicle was not assigned further factories to visit in the previous action, then the vehicle is not assigned a destination factory in the current state. Otherwise, the first factory to visit becomes the next destination factory.

Carrying orders. In all cases $\mathcal{P}_{k+1,v}^0 = \mathcal{P}_{k,v}^0$, except in Case 4, when $\mathcal{P}_{k+1,v}^0 = (\mathcal{P}_{k,v}^0 \setminus \mathcal{D}_{k,v}^1) \cup \mathcal{P}_{k,v}^1$, since the vehicle has reached its destination, although loading and unloading may be in progress at time point τ_{k+1}.

Appendix D. Feasibility of solutions

In the following, we formally define the feasibility of the route plans in a solution.

Orders. First of all, all orders must be dispatched in a solution. Let $\mathcal{O}_v$ be the set of those orders which belong to (i.e., picked up and delivered by) vehicle v. Then, each order belongs to exactly one vehicle:

$$\bigcup_{v \in \mathcal{V}} \mathcal{O}_v = \mathcal{O},$$

with $\mathcal{O}_v \cap \mathcal{O}_w = \emptyset$ for all $v \neq w$. Also, each order is picked up and delivered only once:

$$\mathcal{O}_v = \bigcup_{j=1}^{\ell_v} \mathcal{D}_v^j = \bigcup_{j=1}^{\ell_v} \mathcal{P}_v^j,$$

with $\mathcal{D}_v^i \cap \mathcal{D}_v^j = \emptyset$ and $\mathcal{P}_v^i \cap \mathcal{P}_v^j = \emptyset$ for all $i \neq j$.

Each order must be picked up before its delivery, that is, if $o_i \in \mathcal{P}_v^{j_1}$ for some j_1, then $o_i \in \mathcal{D}_v^{j_2}$ for some $j_1 < j_2$.

LIFO constraint. Let $\mathcal{L}_v$ be the concatenation of lists $\mathcal{D}_v^1, \mathcal{P}_v^1, \ldots, \mathcal{D}_v^{\ell_v}, \mathcal{P}_v^{\ell_v}$. Let $\text{pos}(o_i^+)$ and $\text{pos}(o_i^-)$ denote the position of the first and the second (i.e., last) occurrence of order $o_i \in \mathcal{O}_v$ in $\mathcal{L}_v$, respectively. Then, route plan θ_v satisfies the LIFO constraint, if

$$\text{pos}(o_i^+) < \text{pos}(o_j^+) \Rightarrow \text{pos}(o_i^-) \leq \text{pos}(o_j^+) \vee \text{pos}(o_j^-) \leq \text{pos}(o_i^-)$$

holds for all $o_i, o_j \in \mathcal{O}_v$.

Capacity constraint. The route plan θ_v satisfy the capacity constraint, if the total quantity of the orders carried by the vehicle never exceeds its capacity. That is,

$$\sum_{j=1}^{\ell}\left(\sum_{o_i\in\mathcal{P}_v^j} q_i - \sum_{o_i\in\mathcal{D}_v^j} q_i\right) \le Q$$

holds for all $\ell = 1,\dots,\ell_v$.

Fundamental routing constraints. The travel time between factories are fixed:

$$ta_v^j - td_v^{j-1} = \text{travel}(f_v^{j-1}, f_v^j) \quad \text{for all } j = 1,\dots,\ell_v.$$

Let η_v^j be the waiting time of the vehicle at the jth visited factory.

$$ta_v^j + \eta_v^j + h^{\text{docking}} + \sum_{o_k\in\mathcal{D}_v^j} h_k^d + \sum_{o_k\in\mathcal{P}_v^j} h_k^p \le td_v^j$$

Appendix E. Evaluation procedure

The purpose of the evaluation procedure is to compute the value of the cost functions (2) and (3). To this end, the timing information for the routes defined in Section 4.2 has to be computed. We apply a basic event-based simulation procedure. The procedure maintains a priority queue (*event queue*) in order to process arrival and departure events. In addition, each factory is associated with a *reservation list* containing vehicles that are currently at the factory, see Appendix A.

Step 1 (Initialization). First, consider the vehicles with current factory. If the earliest departure time of a vehicle is in the future (i.e., the vehicle is not finished yet at this factory), we insert the vehicle into the sorted reservation list of the factory. We also create a *departure event* associated with the corresponding departure time. Then, for each vehicle without current factory, we create an *arrival event* associated with the corresponding arrival time.

Step 2 (Processing events). If the event queue is empty, we are done, and thus stop by returning the calculated objective function value. Otherwise, we take out from the queue an event with the earliest associated time. Denote the corresponding factory with f_e, and the associated time with t_e. If this event is a departure event, we remove the corresponding vehicle from the reservation list of the corresponding factory. Also, if the vehicle has a next factory to visit, say f_n, we create and store an arrival event associated with the arrival time $t_e + \text{travel}(f_e, f_n)$. Otherwise, if the event taken out is an arrival event, then we calculate and store its contribution to the cost function value, as defined in Section 3.1.7. We also calculate the earliest departure time of the vehicle (see Appendix A), and create a departure event associated with this time. We repeat this step.

Supplementary material

Supplementary material associated with this article can be found, in the online version, at 10.1016/j.multra.2025.100194

References

Bektaş, T., Repoussis, P.P., Tarantilis, C.D., 2014. Chapter 11: dynamic vehicle routing problems. In: Vehicle Routing: Problems, Methods, and Applications. SIAM, pp. 299–347.

Benavent, E., Landete, M., Mota, E., Tirado, G., 2015. The multiple vehicle pickup and delivery problem with LIFO constraints. Eur. J. Oper. Res. 243 (3), 752–762.

Bent, R.W., Van Hentenryck, P., 2004. Scenario-based planning for partially dynamic vehicle routing with stochastic customers. Oper. Res. 52 (6), 977–987.

Berbeglia, G., Cordeau, J.-F., Laporte, G., 2010. Dynamic pickup and delivery problems. Eur. J. Oper. Res. 202 (1), 8–15.

Cai, J., Zhu, Q., Lin, Q., 2022. Variable neighborhood search for a new practical dynamic pickup and delivery problem. Swarm Evol. Comput. 75, 101182.

Cai, J., Zhu, Q., Lin, Q., Li, J., Chen, J., Ming, Z., 2022. An efficient multi-objective evolutionary algorithm for a practical dynamic pickup and delivery problem. In: International Conference on Intelligent Computing. Springer, pp. 27–40.

Cai, J., Zhu, Q., Lin, Q., Ma, L., Li, J., Ming, Z., 2023. A survey of dynamic pickup and delivery problems. Neurocomputing, 126631.

Carrabs, F., Cordeau, J.-F., Laporte, G., 2007. Variable neighborhood search for the pickup and delivery traveling salesman problem with LIFO loading. INFORMS J. Comput. 19 (4), 618–632.

Chen, L., Wandelt, S., Dai, W., Sun, X., 2022. Scalable vertiport hub location selection for air taxi operations in a metropolitan region. INFORMS J. Comput. 34 (2), 834–856.

Cordeau, J.-F., Iori, M., Laporte, G., Salazar González, J.J., 2010. A branch-and-cut algorithm for the pickup and delivery traveling salesman problem with LIFO loading. Networks 55 (1), 46–59.

Dabia, S., Ropke, S., Van Woensel, T., 2019. Cover inequalities for a vehicle routing problem with time windows and shifts. Transp. Sci. 53 (5), 1354–1371.

Dayarian, I., Savelsbergh, M., 2020. Crowdshipping and same-day delivery: employing in-store customers to deliver online orders. Prod. Oper. Manag. 29 (9), 2153–2174.

Dong, H., Zhuang, W., Ding, H., Zhou, Q., Wang, Y., Xu, L., Yin, G., 2022. Event-driven energy-efficient driving control in urban traffic for connected electric vehicles. IEEE Trans. Transp.Electrif. 9 (1), 99–113.

Du, J., Zhang, Z., Wang, X., Lau, H.C., 2023. A hierarchical optimization approach for dynamic pickup and delivery problem with LIFO constraints. Transp. Res. Part E Logist.Transp. Rev. 175, 103131.

Fonseca-Galindo, J.C., de Castro Surita, G., Neto, J.M., de Castro, C.L., Lemos, A.P., 2022. A multi-agent system for solving the dynamic capacitated vehicle routing problem with stochastic customers using trajectory data mining. Expert Syst. Appl. 195, 116602.

Gendreau, M., Guertin, F., Potvin, J.-Y., Taillard, E., 1999. Parallel tabu search for real-time vehicle routing and dispatching. Transp. Sci. 33 (4), 381–390.

Ghiani, G., Manni, A., Manni, E., 2022. A scalable anticipatory policy for the dynamic pickup and delivery problem. Comput.Oper. Res. 147, 105943.

Gromicho, J., Van Hoorn, J.J., Schutten, J.M., et al., 2012. Vehicle Routing with Restricted Loading Capacities. Technical Report. Technische Universiteit Eindhoven.

Györgyi, P., Kis, T., 2019. A probabilistic approach to pickup and delivery problems with time window uncertainty. Eur. J. Oper. Res. 274 (3), 909–923.

Hansen, P., Mladenović, N., Brimberg, J., Pérez, J.A.M., 2019. Variable Neighborhood Search. Springer.

Hao, J., Lu, J., Li, X., Tong, X., Xiang, X., Yuan, M., Zhuo, H. H., 2022. Introduction to the dynamic pickup and delivery problem benchmark – ICAPS 2021 competition. 2202.01256.
Hempsch, C., Irnich, S., 2008. Vehicle routing problems with inter-tour resource constraints. In: The Vehicle Routing problem: Latest Advances and New Challenges. Springer, pp. 421–444.
Hildebrandt, F.D., Thomas, B.W., Ulmer, M.W., 2023. Opportunities for reinforcement learning in stochastic dynamic vehicle routing. Comput.Oper. Res. 150, 106071.
Horváth, M., Kis, T., Györgyi, P., 2021. Solution approach of the Quickest Route Team for solving the "ICAPS 2021: The Dynamic Pickup and Delivery Problem" challenge by Huawei. https://competition.huaweicloud.com/information/1000041411/Winning.
Hvattum, L.M., Løkketangen, A., Laporte, G., 2006. Solving a dynamic and stochastic vehicle routing problem with a sample scenario hedging heuristic. Transp. Sci. 40 (4), 421–438.
Ichoua, S., Gendreau, M., Potvin, J.-Y., 2000. Diversion issues in real-time vehicle dispatching. Transp. Sci. 34 (4), 426–438.
Jin, Z., Ng, K.K.H., Zhang, C., 2024. Robust optimisation for vertiport location problem considering travel mode choice behaviour in urban air mobility systems. J. Air Transp. Res. Soc. 2, 100006.
Lin, C., Choy, K.L., Ho, G.T.S., Lam, H.Y., Pang, G.K.H., Chin, K.-S., 2014. A decision support system for optimizing dynamic courier routing operations. Expert Syst. Appl. 41 (15), 6917–6933.
Liu, Y., Roberto, B., Zhou, J., Yu, Y., Zhang, Y., Sun, W., 2023. Efficient feasibility checks and an adaptive large neighborhood search algorithm for the time-dependent green vehicle routing problem with time windows. Eur. J. Oper. Res. 310 (1), 133–155.
Mitrović-Minić, S., Krishnamurti, R., Laporte, G., 2004. Double-horizon based heuristics for the dynamic pickup and delivery problem with time windows. Transp. Res. Part B Methodol. 38 (8), 669–685.
Mladenović, N., Hansen, P., 1997. Variable neighborhood search. Comput.Oper. Res. 24 (11), 1097–1100.
Pahwa, A., Jaller, M., 2024. Evaluating private and system-wide impacts of freight eco-routing. Transp. Res. Part D Transp.Environ. 130, 104170.
Pillac, V., Gendreau, M., Guéret, C., Medaglia, A.L., 2013. A review of dynamic vehicle routing problems. Eur. J. Oper. Res. 225 (1), 1–11.
Powell, W.B., 2011. Approximate Dynamic Programming: Solving the Curses of Dimensionality, 2nd ed. John Wiley & Sons.
Psaraftis, H.N., 1980. A dynamic programming solution to the single vehicle many-to-many immediate request dial-a-ride problem. Transp. Sci. 14 (2), 130–154.
Psaraftis, H.N., Wen, M., Kontovas, C.A., 2016. Dynamic vehicle routing problems: three decades and counting. Networks 67 (1), 3–31.
Riley, C., Legrain, A., Van Hentenryck, P., 2019. Column generation for real-time ride-sharing operations. In: Integration of Constraint Programming, Artificial Intelligence, and Operations Research: 16th International Conference, CPAIOR 2019, Thessaloniki, Greece, June 4–7, 2019, Proceedings 16. Springer, pp. 472–487.
Riley, C., Van Hentenryck, P., Yuan, E., 2020. Real-time dispatching of large-scale ride-sharing systems: integrating optimization, machine learning, and model predictive control. arXiv preprint arXiv:2003.10942.
Rios, B.H.O., Xavier, E.C., Miyazawa, F.K., Amorim, P., Curcio, E., Santos, M.J.a., 2021. Recent dynamic vehicle routing problems: a survey. Comput. Ind. Eng. 160, 107604.
Savelsbergh, M.W.P., Sol, M., 1995. The general pickup and delivery problem. Transp. Sci. 29 (1), 17–29.
Soeffker, N., Ulmer, M.W., Mattfeld, D.C., 2022. Stochastic dynamic vehicle routing in the light of prescriptive analytics: a review. Eur. J. Oper. Res. 298 (3), 801–820.
Srour, F.J., Agatz, N., Oppen, J., 2018. Strategies for handling temporal uncertainty in pickup and delivery problems with time windows. Transp. Sci. 52 (1), 3–19.
Ulmer, M., Nowak, M., Mattfeld, D., Kaminski, B., 2020. Binary driver-customer familiarity in service routing. Eur. J. Oper. Res. 286 (2), 477–493.
Ulmer, M.W., Soeffker, N., Mattfeld, D.C., 2018. Value function approximation for dynamic multi-period vehicle routing. Eur. J. Oper. Res. 269 (3), 883–899.
Ulmer, M.W., Streng, S., 2019. Same-day delivery with pickup stations and autonomous vehicles. Comput. Oper. Res. 108, 1–19.
Van der Zon, N.M., 2017. Vehicle Routing with Departure Smoothing. Erasmus University Rotterdam Master's thesis.
Van Heeswijk, W.J.A., Mes, M.R.K., Schutten, J.M.J., 2019. The delivery dispatching problem with time windows for urban consolidation centers. Transp. Sci. 53 (1), 203–221.
Van Hemert, J.I., La Poutré, J.A., 2004. Dynamic routing problems with fruitful regions: models and evolutionary computation. In: International Conference on Parallel Problem Solving from Nature. Springer, pp. 692–701.
Wandelt, S., Dai, W., Zhang, J., Sun, X., 2022. Toward a reference experimental benchmark for solving hub location problems. Transp. Sci. 56 (2), 543–564.
Wandelt, S., Wang, S., Zheng, C., Sun, X., 2023. Aerial: a meta review and discussion of challenges toward unmanned aerial vehicle operations in logistics, mobility, and monitoring. IEEE Trans. Intell. Transp. Syst..
Wang, Z., Sheu, J.-B., 2019. Vehicle routing problem with drones. Transp. Res. Part B Methodol. 122, 350–364.
Xu, X., Wei, Z., 2023. Dynamic pickup and delivery problem with transshipments and LIFO constraints. Comput. Ind. Eng. 175, 108835.
Ye, J., Liang, E., 2021. ICAPS2021: DPDP Challenge. https://competition.huaweicloud.com/information/1000041411/Winning.
Zhang, H., Ge, H., Yang, J., Tong, Y., 2022. Review of vehicle routing problems: models, classification and solving algorithms. Arch. Comput. Methods Eng. 29 (1), 195–221.
Zhang, J., Van Woensel, T., 2022. Dynamic vehicle routing with random requests: a literature review. Int. J. Prod. Econ., 108751.
Zhu, Q., Cai, J., Lin, Q., 2021. A variable neighborhood search method for Dynamic Pickup and Delivery Problem. https://competition.huaweicloud.com/information/1000041411/Winning.